Effective visualisation of callgraphs for optimisation of parallel programs: a design study

by

Mabule Samuel Mabakane

Submitted in fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

at the

Faculty of Science

Department of Computer Science

University of Cape Town

Supervised by:

Associate Professor M. Kuttel

April 2019

ABSTRACT

Parallel programs are increasingly used to perform scientific calculations on supercomputers. Optimising parallel applications to scale well, and ensuring maximum parallelisation, is a challenging task. The performance of parallel programs is affected by a range of factors, such as limited network bandwidth, parallel algorithms, memory latency and the speed of the processors. The term "performance bottlenecks" refers to obstacles that cause slow execution of the parallel programs. Visualisation tools are used to identify performance bottlenecks of parallel applications in an attempt to optimize the execution of the programs and fully utilise the available computational resources. TAU (Tuning and Analysis Utilities) callgraph visualisation is one such tool commonly used to analyse the performance of parallel programs. The callgraph visualisation shows the relationship between different parts (for example, routines, subroutines, modules and functions) of the parallel program executed during the run. TAU's callgraph tool has limitations: it does not have the ability to effectively display large performance data (metrics) generated during the execution of the parallel program, and the relationship between different parts of the program executed during the run can be hard to see.

The aim of this work is to design an effective callgraph visualisation that enables users to efficiently identify performance bottlenecks incurred during the execution of a parallel program. This design study employs a user-centred iterative methodology to develop a new callgraph visualisation, involving expert users in the three developmental stages of the system: these design stages develop prototypes of increasing fidelity, from a paper prototype to high fidelity interactive prototypes in the final design. The paper-based prototype of a new callgraph visualisation was evaluated by a single expert from the University of Oregon's Performance Research Lab, which developed the original callgraph visualisation tool. This expert is a computer scientist who holds doctoral degree in computer and information science from University of Oregon and is the head of the University of Oregon's Performance Research Lab. The interactive prototype (first high fidelity design) was evaluated against the original TAU callgraph system by a team of expert users, comprising doctoral graduates and undergraduate computer scientists from the University of Tennessee, United States of America (USA). The final complete prototype (second high fidelity design) of the callgraph visualisation was developed with the D3.js JavaScript library and evaluated by users (doctoral graduates and

undergraduate computer science students) from the University of Tennessee, USA. Most of these users have between 3 and 20 years of experience in High Performance Computing (HPC). On the other hand, an expert has more than 20 years of experience in development of visualisation tools used to analyse the performance of parallel programs. The expert and users were chosen to test new callgraphs against original callgraphs because they have experience in analysing, debugging, parallelising, optimising and developing parallel programs. After evaluations, the final visualisation design of the callgraphs was found to be effective, interactive, informative and easy-to-use. It is anticipated that the final design of the callgraph visualisation will help parallel computing users to effectively identify performance bottlenecks within parallel programs, and enable full utilisation of computational resources within a supercomputer.

DECLARATION

I, the undersigned, Mabule Samuel Mabakane hereby declare that the thesis submitted for the degree of Doctor of Philosophy in Computer Science, at the University of Cape Town, is my own original work and has not been previously submitted to any other institution of higher education. I know the meaning of plagiarism and declare that all of the work in the document, save for that which is properly acknowledged, is my own work.

Signature: Signed by candidate

Mabule Samuel Mabakane

Date: 04 April 2019

ACKNOWLEDGEMENTS

The author would like to acknowledge the following:

Prof. Michelle Kuttel (supervisor) for her useful guidance, advice and continual review of the manuscript throughout the research study.

The National Integrated Cyber Infrastructure System (NICIS), Centre for High Performance Computing (CHPC), South Africa, under the auspices of the Council for Scientific and Industrial Research, for providing financial support and computational resources to evaluate the performance of parallel programs.

Prof. Jack Dongarra (University of Tennessee and Oak Ridge National Laboratory, USA) for his continual support of the study and his help in testing the callgraph visualisations.

The University of Tennessee, Innovative Computing Laboratory doctoral graduates and under-graduate students who tested both original and new callgraph visualisations.

Dr. Sameer Shende (University of Oregon, USA) for providing performance data (metrics) to test the new callgraph visualisation system.

Mr. Alan Morris (University of Utah, USA) for providing essential information about the original design of the callgraph visualisation system.

Dr. Abiodun Babatunde (University of Cape Town, South Africa) for helping to compile and visualize the performance of the Weather Research Forecast model using an original callgraph visualisation system.

Ms. Heike Jagode (University of Tennessee) for useful discussions about the use of visualisations for analyses of parallel programs on the supercomputers.

The University of Cape Town computer science department, which shared useful experiences about the execution of parallel programs.

My wife, Elizabeth Nontombi Mabakane, and daughter, Reneiloe Mabakane, for their continual motivational support throughout the research study.

My parents, Makgomo Linah Mabakane and Malesela Phillipus Mabakane, who encouraged me to work hard for this study.

TABLE OF CONTENTS

ABSTRACT ..... **i**

DECLARATION ..... **iii**

ACKNOWLEDGEMENTS ..... **iv**

TABLE OF CONTENTS ..... **vi**

LIST OF FIGURES ..... **x**

LIST OF TABLES ..... **xii**

GLOSSARY ..... **xiii**

CHAPTER 1: INTRODUCTION **1**

1.1 Objectives 5

1.2 Impact 5

1.3 Approach and methodology 6

1.4 Contributions 8

1.5 Synopsis 9

CHAPTER 2: BACKGROUND **10**

2.1 Parallel programming 10

2.1.1 Parallel programming with MPI and the distributed memory model 11

2.1.2 Parallel programs running on the CHPC supercomputer 12

2.2 Visualisation 12

2.2.1 Software for analysis and visualisation of parallel performance 13

2.2.1.1 TAU visualisation system 13

2.2.1.2 Scalasca analysis tool 20

2.2.1.3 Vampir visualisation tool 22

2.2.1.4 Paraver performance analysis system 23

2.2.1.5 Periscope analysis tool 24

2.2.2 Comparison of current tools 26

2.3 Visualisation design 27
2.3.1 Network Visualisations 29
2.4 User-centred design 33
2.4.1 Prototyping 35
2.4.2 Case studies 37
2.5 Summary 38
CHAPTER 3: TAU Paraprof Callgraph Visualisation 39
3.1 Overview of the callgraph visualisation 39
3.1.1 Colour 46
3.1.2 Size and space 47
3.1.3 Network links and texture 48
3.1.4 Interaction 49
3.2 Discussion 49
3.3 Conclusions 50
CHAPTER 4: DESIGN METHODOLOGY 51
4.1 Design principles 52
4.2 Design approach 53
4.2.1 Establishing requirements 54
4.2.2 Low-fidelity prototype 54
4.2.3 First high-fidelity prototype 55
4.2.3.1 Evaluation 57
4.2.4 Second high-fidelity prototype 58
4.2.4.1 Evaluation 60
4.2.5 Final design 60
4.3 Conclusion 60
CHAPTER 5: LOW-FIDELITY PROTOTYPE 62
5.1 Design goals 62

5.2 Design description 62
5.2.1 Star design 64
5.2.2 Tree design 67
5.3 Evaluation 69
5.4 Conclusion 70
CHAPTER 6: HIGH-FIDELITY PROTOTYPE 71
6.1 Interactive design 71
6.2 Star interactive design 72
6.2.1 Expandable star design 72
6.2.2 Collapsible star design 79
6.3 Tree interactive design 86
6.3.1 Expandable tree design 86
6.3.2 Non-expandable tree design 94
6.4 Evaluation 102
6.4.1 Testing methods 102
6.4.2 Testing environment 103
6.4.3 Users 103
6.4.4 Testing procedure 105
6.4.5 Analysis 106
6.4.6 Results and discussion 106
6.4.6.1 Users visual queries 106
6.4.6.2 Users' questionnaire and interviews 109
6.5 Experts evaluation 118
6.6 Analysis of the evaluation 120
6.7 Conclusions 137
CHAPTER 7: FINAL DESIGN 138
7.1 Design goals 138

7.2 Design description 139
7.3 Filtering and search design on small performance data 139
7.4 Filtering and search design on large performance metrics 145
7.5 Conclusions 150
CHAPTER 8: CONCLUSIONS 151
8.1 Conclusions and discussions 151
8.2 Recommendations for future work 160
REFERENCES 161

LIST OF FIGURES

Figure 1.1: TAU's ParaProf callgragh visualisation for TFS (Morris, Malony and Shende, 2007).....4
Figure 2.1: TAU's callgraph report used to analyse TFS parallelised using ParaWise (Morris, Malony and Shende, 2007).....16
Figure 2.2: Callgraph performance analysis of Micromegas parallel simulation (none parallelised interaction forces) (from Ciorba, Groh and Horstemeyer, 2010) 17
Figure 2.3: Callgraph performance analysis of Micromegas parallel simulation (parallelised interaction forces) (Ciorba, Groh and Horstemeyer, 2010) 19
Figure 2.4: TAU's expandable tree-based report (Morris, Malony and Shende, 2007) 19
Figure 2.5: Scalasca performance report of the Zeus/MP2 program on 512 processors (Böhme, Wolf and Geimer, 2012).....21
Figure 2.6: Vampir used to visualise the performance of pF3C parallel program (Isaacs *et al.*, 2014).....22
Figure 2.7: Paraver utilised to visualize the performance of OmpSs parallel model (Filgueras, 2014).....24
Figure 2.8: Human disease network visualisation (Lima, 2010).....30
Figure 2.9: Analysis of online social networks (Heer and Boyd, 2005) 32
Figure 2.10: Visualisation of obese and non-obese social network users (Christakis and Fowler, 2007).....32
Figure 3.1: Performance analysis of DL_POLY_2.18 on the CHPC supercomputer. 40
Figure 3.2: Visualisation of DL_POLY_2.20 on CHPC's Sun cluster 42
Figure 3.3: Visualisation of DL_POLY_2.20 on forty-eight processors 43
Figure 3.4: Left-side *(A)* and right side *(B)* visualisation results of WRF-3.5 on Sun cluster 45
Figure 4.1: Interaction design model (Rogers, Sharp and Preece, 2011).....51
Figure 5.1: Star (A), Tree (B) and Time-level (C) prototype design 66
Figure 6.1: Expandable star design on two processors 73
Figure 6.2: Expandable star design with active nodes on processors 74
Figure 6.3: Expandable star design on four processors.....75
Figure 6.4: Expandable star design on eight processors 76
Figure 6.5: Expandable star design on sixteen processors 78
Figure 6.6: Collapse star design on two processors 80

Figure 6.7: Collapsible star design with active nodes on two processors 81
Figure 6.8: Collapsible star design (normal view) on four processors.....82
Figure 6.9: Collapsible star design (increased view) on four processors.....83
Figure 6.10: Collapsible star design on eight processors 84
Figure 6.11: Collapsible star design on sixteen processors.....85
Figure 6.12: Expandable tree design (first view) on two processors 87
Figure 6.13: Expandable tree design (second view) on two processors.....89
Figure 6.14: Expandable tree design on four processors 90
Figure 6.15: Expandable tree design on eight processors 92
Figure 6.16: Expandable tree design on sixteen processors.....93
Figure 6.17: None expandable tree design on two processors 95
Figure 6.18: None expandable tree design on four processors 96
Figure 6.19: None expandable tree design on eight processors 97
Figure 6.20: None expandable tree design on sixteen processors.....98
Figure 6.21: Original callgraph visualisation designed by the University of Oregon Performance Research Lab, used to visualise the Weather Research and Forecast (WRF) model on the CHPC's Sun cluster.....101
Figure 6.22: First high-fidelity prototype interactive visualisation designs: (top-left) collapsible star, (top-right) expandable tree, (bottom-left) non-expandable tree, and (bottom-right) expandable star visualisations of the NAS Parallel Benchmark program.....101
Figure 6.23: Users experience (A) and duties (B) in computational science 104
Figure 6.24: Visual queries performed on the original (A) and new (B) designs 107
Figure 6.25: Easy of use (A) and helpful (B) design 111
Figure 6.26: Users wishes (A) and important queries (B) on the new design.....113
Figure 6.27: Users working on the original design 117
Figure 6.28: Users working on the original and the first high-fidelity design 118
Figure 7.1: Filtering design of the callgraph visualisation system.....140
Figure 7.2: Search design of the callgraph visualisation system.....141
Figure 7.3: Filtering design visualising performance data 144
Figure 7.4: Search design visualising performance data.....145
Figure 7.5: Filtering design used to analyse the WRF 146
Figure 7.6: Search design used to analyse the WRF program 149

LIST OF TABLES

Table 2.1: Performance analysis of NAS parallel benchmarks using Periscope 26
Table 2.2: Factors used to measure usability of the visualisation design 29
Table 3.1: Visual queries performed by users.....44
Table 4.1: Basic principles of designing an interactive visualisation (Abras, Maloney-Krichmar and Preece, 2004; Zuk *et al.,* 2006).....52
Table 4.2: Visual queries performed by users on the first high fidelity prototype 55
Table 4.3: Answers to the visual queries performed by users.....56
Table 4.4: Users questionnaires about the original and first high fidelity prototype design.....57
Table 4.5: Visual queries performed by users on the second high fidelity prototype.....59
Table 4.6: Users questionnaires about the original and second high fidelity prototype designs 59

GLOSSARY

HPC - High Performance Computing

TAU - Tuning and Analysis Utilities

MPI - Message-Passing Interface

WRF - Weather Research and Forecast

CHPC - Centre for High Performance Computing

UT - University of Tennessee

UO - University of Oregon

NAS - NASA Advanced Supercomputing

TFS - Thermal and Fluid Sciences

CHAPTER 1: INTRODUCTION

Different scientific disciplines, such as material science and climatology, use parallel programming to perform scientific calculations on supercomputers; parallel computers that perform many calculations in a short period of time (Dongarra and Van der Steen, 2012). Parallel programs are computer applications that can perform different calculations simultaneously, and share resources within the parallel system. Examples are the DL_POLY (Smith and Todorov, 2006), Weather and Research Forecast (WRF) (Rolph, Stein and Stunder, 2017) and NAS (NASA Advanced Supercomputing) (Coutinho *et al.,* 2015) parallel programs. DL_POLY is a molecular dynamics model used to simulate molecular interactions (Dimitroulis, Raptis and Raptis, 2015; Chokbunpiam *et al.,* 2016).

WRF is a weather prediction model used to perform atmospheric and weather forecasting (Nehrkorn *et al.,* 2010; Coen *et al.,* 2013) and the NAS parallel benchmark is a program used to evaluate the performance (processing speed) of supercomputer (Shun *et al.,* 2012; Patki *et al.,* 2015). The performance of supercomputers can be measured by the number of floating point operations performed per second for example, megaflops/s, gigaflops/s, teraflops/s and petaflops/s. Supercomputing systems can be classified into different categories, namely, shared memory systems (for example, symmetric multi-processors), distributed memory systems (for example, clusters and hybrid systems), massive parallel processors and vector computers (Strohmaier *et al.,* 2005; Kindratenko and Trancoso, 2011), where each category is determined by the architectural design of the system (Strohmaier and Meuer, 2004).

Here we focus on the optimisation of parallel programs for cluster architectures programed with the message-passing interface (MPI) (Komatitsch *et al.,* 2010; Pereira *et al.,* 2013), as this is a common architecture used in High Performance Computing (HPC) (Sumanth, Swanson and Jiang, 2003; Freeman *et al,* 2014; Verma *et al.,* 2015). A cluster is a group of computers (nodes) interconnected via a network and working as an integrated single computational resource (Werstein, Situ and Huang, 2006; Freeman *et al.,* 2014). MPI is a parallel programming model used to exchange data by sending/receiving messages between nodes in a cluster (Gropp, 2012; Losada *et al.,* 2017). In South Africa, the Centre for High Performance Computing (CHPC) currently hosts the Lengau cluster, which performs up to 1.029 petaflops/s

(Mabakane, Moeketsi and Lopis, 2017). The Lengau cluster consists of different computational nodes: Dell nodes (Intel Xeon E5-2690 V3 processors) and "fat" nodes (Intel Xeon E7-4850) processors. Each Dell node consists of twenty-four cores connected to 64 gigabytes of memory. Moreover, Dell servers are equipped with processors that can perform up to 2.6 GHz per single core. Each fat node contains fifty-six cores, of which, each core can produce a maximum processing speed of 2.2 GHz connected to a total memory of 1 terabyte. All the nodes are connected to each other via Infiniband network, which scales up to 56 gigabytes/s. The CHPC's Lengau cluster has a total of 1368 compute nodes, which results in 32832 cores and 148.5 terabytes of memory connected to the storage area network (SAN) of 4 petabytes running a Lustre file system.

Various scientists and technologists utilise parallel programs to perform calculations and solve scientific problems using the CHPC parallel system. Many common parallel programs do not perform well and do not fully utilise computational resources (such as memory and processors) within the parallel system. The *performance* of parallel programs equates to the amount of time taken to execute the target application; *efficiency* is when the target application fully utilises the computational resources allocated to it. The performance of parallel programs can be negatively affected by many factors, such as limited network bandwidth, uneven distribution of message-passing, slow read/write operations in the storage, improper logic of the parallel code, high memory latency and processor utilisation in the execution nodes of the parallel system (Adve and Vernon, 2004; Ebnenasir and Beik, 2009).

Parallel programs written using general parallel programming technologies must usually be optimised to perform well on specific parallel architectures. The reason parallel programs tend to show poor performance is caused primarily by developers' limited parallel programming experience (Stone, Gohara and Shi, 2010), which leads to suboptimal performance of the parallel application. Continuous changes in hardware architectures also contribute to a lack of standard approaches for developing efficient applications for parallel systems.

Optimisation seeks to understand and solve these performance problems, and is the process of re-writing program instructions with the purpose of increasing the performance and efficiency of the code on a particular computing system. It is hard for application users (people who can find performance bottlenecks and tackle them) to identify an exact area of the code that causes

a performance bottleneck in the system, particularly for large complex programs with many routines. Optimisation of parallel applications can be very challenging, one needs to understand and analyse various factors, such message-passing activities, I/O (Input/Output) performance, network communication and the level of parallelism within the code of the application. Visualisation tools therefore are often used to identify bottlenecks in parallel programs (Geimer *et al.,* 2010; Subotic *et al.,* 2010). Moreover, visualisation is useful for identifying relationship on complex multidimensional data. Two popular visualisation tools are TAU (Shende and Malony, 2006; Knüpfer *et al.,* 2012) and Scalasca (Knüpfer *et al.,* 2012; Huck *et al.,* 2015).

The TAU visualisation utilises an easy-to-use graphical user interface tool, ParaProf (Delistavrou and Margaritis, 2011) to analyse the performance of the program. In particular, TAU ParaProf parallel profiling system generates callpath data (performance metrics): records of when each object (routine, subroutine, module or function) was executed during the execution of the parallel application. This callpath data is then displayed as callgraph: tree-based report a graphical representation of the relationship and dependencies between the many different computational events. The callgraph visualisation display relationship between objects using square boxes connected to each other via black network lines, as shown in Figure 1.1.

It is also able to demonstrate the kinship between the boxes using blue network lines. The callgraph system use different colours (light blue, dark blue, green, light green, yellow, orange and red) to showcase the status of each object within the parallel program. The status of the object refers to the execution time taken to compute an object during the run. Callgraph visualisations are useful for mapping the execution time back to specific routines in the source code (Morris, Malony and Shende, 2007).

Users typically aim to identify parts of the application that consume excessive amounts of time, and the relationship between different parts of the application. The currently TAU's callgraph visualisation tool does not scale well for large callpath data sets - that require very large callgraphs. In large callgraphs, it is difficult for users to identify modules and their relationships. For example, Figure 1.1 shows a complicated callgraph for the execution of the TFS (Thermal and Fluid Sciences) computational fluid dynamics program. The small boxes (dark blue, yellow and green) represent modules, routines and/or subroutines and black lines depict callpaths - the relationship between the events. This large callgraph is complex to

analyse and understand. Firstly, Figure 1.1 does not clearly show the relationship between the events because callpaths (black lines) obscure events (dark blue, yellow and green boxes).

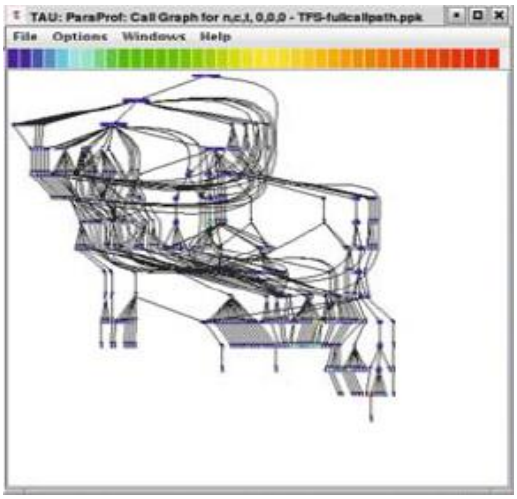

Figure 1.1: TAU's ParaProf callgragh visualisation for TFS (Morris, Malony and Shende, 2007).

Secondly, it is difficult for the user to identify the name of an event and its colour (which is a classification) due to their small size. Thirdly, performance bottlenecks are hard to identify because of the overlapping callpaths (black lines). Lastly, it is not clear how the colour key at the top relates to the events. Some users have explored the use of the Fisheye view to allow for zooming in to specific areas in a large callgraph, but Fisheye distorts the shape of the callgraph and so does not completely solve the problem (Karrer *et al.*, 2011; LaToza and Myers, 2011).

The TAU callgraph visualisation is currently not effective for large callgraphs of complex parallel programs, whereas an effective visualisation is essential for users to rapidly identify obstacles causing poor performance and slow execution of parallel programs. An effective callgraph visualisation is a system that has ability to easily help users identify performance bottlenecks within the parallel program.

1.1 Objectives

The aim of this work is to develop an effective callgraph network visualisation to enable users to efficiently and rapidly identify performance bottlenecks and thus optimise and improve the performance and efficiency of the parallel programs. In particular, we aim to develop an effective callgraph visualisation that will help users to analyse the performance of parallel programs running on the supercomputers. The visualisation should be informative, intuitive, interactive and easy to use. Informative means providing useful information about performance of parallel programs, of which, intuitive is easy to understand the information presented. The term "interactive" relates to exchanges of information between the user and system, where both parties send, receive and respond to queries (Rogers, Sharp and Preece, 2011; Liem and Sanders, 2011; Van der Ryn, 2013). The easy to use is when users find it simple to use the information about the performance of the parallel program.

The focus of this research is to perform a design study - project whereby a visualisation researcher analyse real-world problem facing a specific domain, consequently, design a visualisation system to support solving this problem, validate the design and outline lessons learned in applying the principles of visualisation design (Sedlmair, Meyer and Munzner, 2012). We intend to develop an effective callgraph visualisation by conducting a design study because it is a problem-driven research, where the objective is to work with the actual users to solve their real problems (Rampersad *et al.*, 2017).

1.2 Impact

An effective callgraph visualisation analysis helps users to obtain meaningful visualisation graphs that enable easy optimisation of parallel programs for supercomputing systems. TAU's callgraph visualisation tool accelerates the performance and efficiency (utilisation of computational resources) of the programs used in the parallel systems. In particular, the design study produced a visualisation tool that will enable users to obtain essential visualisation information to assist with optimisation and increased performance/efficiency of parallel programs.

1.3 Approach and methodology

TAU's callgraph visualisation was used to conduct computational experiments of the study. TAU's performance analysis tool is an open-source package and we have collaborated with the developers of this analysis tool. All computational experiments were performed in the CHPC's Lengau and Sun cluster (Mabakane, Moeketsi, Lopis, 2017) due to its availability to researchers around South Africa. In this research study, the intention is to design an effective callgraph visualisation for identifying performance bottlenecks in parallel programs. Design is a practical and creative exercise that aims to develop a product which will help users achieve their needs, desires and goals (Rogers, Sharp and Preece, 2011; Munzner, 2014). A good callgraph visualisation design enables parallel program users to easily optimise applications for scalable performance and efficiency on supercomputers.

The study plan is to design paper prototypes, interactive prototypes and final visualisation, where each design will be tested against the original callgraph visualisation system. Each new design was iteratively tested by the users to achieve the final design of the callgraph visualisation. This research study follows the user-centred design, which is the process of iteratively testing with users throughout the development cycle of the system (Schulze, 2001; Garrett, 2010). Figure 4.1 discusses different development stages followed to design the new callgraph visualisation. The original callgraphs are produced using JGraph (Falcone and Sharif, 2013) - a JavaScript graphics library, which is now a commercial product. We used the freely available D3.js (Murray, 2017) library for our new visualisation of large callgraphs. The D3.js library is open-source and it has useful features for manipulating data.

Different users and one expert from a range of scientific disciplines (computer science and applied mathematics) were selected to participate in the design process. Seven users tested both original and first high fidelity prototypes. Users who tested original design against first high fidelity prototype have between 3 and 20 years of experience in High Performance Computing. For security purposes, the users who evaluated the designs will be named as follows: User 1 to User 7. The names of the evaluators will not be revealed in this document as it will be a breach of the voluntary consent memorandum signed by the users and experts during the evaluation of the visualisation designs. After analysing the data, it was found that the users had varying experience, in terms of years spent working in optimising, parallelising, debugging, analysing

and developing parallel programs: User 1 - three years; User 2 - five years; User 3 - eight years; User 4 - nine years; User 5 - 13 years; User 6 - twenty years; User 7 - twenty years, as shown in Figure 6.23 *(A)*. Four users tested the original design against the second high fidelity prototype. These users who tested second interactive prototype have variety of experience in performance analyses and development of parallel programs. In particular, User 1 - five years; User 2 - fifteen years; User 3 - twenty years; User 4 - in excess of twenty years. We rely on the feedback of seven users who evaluated first high fidelity prototype with more than 70 years of experience altogether in HPC. Users who tested the second high fidelity also have a total of more than 60 years of experience in HPC.

Users were selected from the University of Tennessee (UT), Innovative Computing Laboratory, of which, an expert was chosen from the University of Oregon's Performance Research Lab because it is the designer of the original callgraph visualisation. Moreover, this expert holds Doctoral degree in Computer and Information Science from University of Oregon and has over 20 years of experience in development of visualisation tools used to analyse the performance of parallel programs. The users were chosen to participate in the design process because they were willing to evaluate functionalities and capabilities of the original/new callgraph visualisation. Moreover, UT users were selected to test the interactive prototypes and final design because of their experience in analysing, debugging, parallelising, optimising and developing parallel programs.

The University of Tennessee's Innovative Computing Laboratory (UT)[1] computer science doctoral graduates and undergraduates tested the visualisation (both the original and new) as these users understand the process of optimising applications used on supercomputers. These computer scientists from the University of Tennessee's (UT) Innovative Computing Laboratory tested the first high-fidelity, second high-fidelity and final interactive design of the callgraph visualisation system. An expert from the University of Oregon's (OU) Performance Research Lab was selected to test the low-fidelity prototype.

The WRF model (Khain and Lynn, 2009; Barker *et al.*, 2012; Kim *et al.*, 2013), the NAS Parallel Benchmark program with MPI (Shun *et al.*, 2012), and the DL_POLY_2 and 3 (Hein *et*

[1] For more information about UT, please visit: https://icl.cs.utk.edu/index.html.

al., 2005; Smith, 2006; Tang, 2008) were used to evaluate both original and new callgraph visualisations. Moreover, WRF and DL_POLY_2 and 3 parallel programs were used to analyse and scrutinise how the original callgraph visualisations work. These parallel programs were selected because most of the CHPC supercomputing users simulate scientific calculations utilising these application. The NAS Parallel Benchmark was recommended for evaluating the usability (how easy is to use) (Nielsen and Loranger, 2006; Rubin and Chisnell, 2008) of the new callgraphs by the University of Oregon's (OU) Performance Research Lab[2], which developed original callgraph visualisations. Different problem sizes (small and large) of the selected parallel programs were executed and analysed using both original and newly revamped callgraph visualisations.

1.4 Contributions

In this study, we demonstrated how visual properties (e.g. colour, texture, size) of the nodes and network links were used to design effective callgraph network visualisations to analyse the performance of parallel programs. Our new designs of the callgraphs also revealed how visualisation designers can use techniques such as filtering and search of information to efficiently visualise large data within a network visualisation. We have further discussed efficient ways of involving users to be co-designers of the visualisation design by following an interactive design model (Figure 4.1).

An interactive design model enabled the users to iteratively evaluate the visualisation design - which helped us to practically identify errors at an early stage of the research. Most notably, this study has developed a novel technique of expanding, disbanding and collapsing nodes to visualize performance of parallel programs using new callgraph visualisations. Users should be able to efficiently optimize parallel programs using new callgraph designs and thus increase the performance/efficiency of the applications on the supercomputers (Mabakane, Moeketsi and Lopis, 2017).

[2] For more information about OU, visit: http://nic.uoregon.edu/prl/home.php.

1.5 Synopsis

Chapter 2 provides an overview of different visualisation tools used to optimise the performance of parallel programs. Chapter 3 demonstrates the capabilities and limitations of the original callgraph. Chapter 4 focuses on the methodological approach used to design callgraph visualisations, while Chapter 5 discusses new paper prototype designs. Subsequently, Chapter 6 debates interactive prototype designs and how experts tested these designs of the callgraph visualisations. Chapter 7 discusses the abilities of the final visualisation design. Lastly, Chapter 8 provides the conclusions about this study.

CHAPTER 2: BACKGROUND

This chapter presents a summary of MPI parallel programming methods used to design parallel programs for various scientific disciplines, such as astrophysics, chemistry and material science. MPI parallel programing is discussed to help one understand the design of the parallel programs. It further describes different types of parallel programs used to perform calculations and generate scientific data. Visualisation tools help to analyse the execution of different parallel programs for optimisation purposes. Descriptions of various visualisation tools used to analyse the performance of parallel programs is also summarised in this chapter. Different types of visualisations, such as information, network, and scientific visualisation, are discussed. The different types of visualisations are discussed in this chapter to provide a background theory on different ways of presenting visualised information. The challenges and importance of optimising parallel programs are also presented. The user-centred method for designing visualisation tools is also discussed to give a broader perspective on how users help to test the design of the visualisation system.

2.1 Parallel programming

The term “parallel programming” means set of program instructions used to carry out many calculations simultaneously, whereby a large problem can be split into smaller pieces for faster execution in a parallel system (Yang, Huang and Lin, 2011). Parallel programming is often used to execute many calculations by splitting a large task into smaller pieces within the computational system. MPI is usually used for parallel programming in the distributed memory systems such as clusters because it offers point-to-point message passing operations between the nodes (Yang, Huang and Lin, 2011).

The distributed memory programming enables processing of multiple tasks on different number of physical computer and exchange data by sending/receiving messages between the computers. It is a standard message passing tool that provides bindings for different programming languages such as C, C++ and Fortran. The OpenMP is typically used for parallel programming in shared memory systems. In particular, shared memory programs are designed to share, read and write tasks into a single memory connected to different number of processors

(Nickolls *et al.,* 2008). The shared memory programming can be able to easily handle data communication between the tasks, however, it is expensive to keep data local to the processors due to memory access, cache refresh and traffic that occurs when multiple processors access data (Wong, 2009).

2.1.1 Parallel programming with MPI and the distributed memory model

MPI enables parallel programs to distribute and process multiple tasks which are executed at the same time in parallel on a number of different physical computers. Message-passing tools, such as MPICH, MVAPICH and Open MPI all support MPI, which is a standard interface to communicate messages in a distributed memory systems (Gropp, 2012). The differences between these message-passing tools relate to how well the software can perform within the parallel program. The continuous development of these message-passing technologies makes it difficult for programmers to develop a standard approach to write parallel codes for parallel systems.

In fact, various parallel programs parallelised using MPI face numerous performance challenges, such as memory latency, fault tolerance and network communication errors (Gabriel *et al.,* 2004; Gropp, 2012). Optimising parallel programs is seen as a way in which to overcome these performance issues, and obtain scalable performance/efficiency within the parallel system. The process of optimising parallel programs often includes re-defining the parallelisation of the application, and enabling adequate communication between the MPI processes used to simulate the run in a parallel system.

MPI is the most popular way of parallelising parallel applications on distributed memory systems (Yang, Huang and Lin, 2011) because it enables processing of multiple tasks on a number of different physical computers and exchanges data by sending/receiving messages between the computers (Arabe *et al.,* 1995; Nickolls *et al.,* 2008; Wong, 2009). MPI exchange data by sending/receiving messages between the computers, which result in a parallel execution of different tasks within a distributed memory system. It is further the most popular way of parallelizing scientific programs because it is portable on distributed memory architectures.

2.1.2 Parallel programs running on the CHPC supercomputer

Parallel programs are usually parallelised using MPI, and typically run for a long time on the supercomputers. For example, parallel programs process complex problems may run for days, weeks or months on a supercomputer. The author often simulates scientific calculations for weeks on the CHPC's supercomputer, of which, this should not always be the case. The parallelisation strategy of the code plays a significant role in determining the performance (processing speed) and efficiency (utilisation of computational resources) of the parallel program.

Various scientific disciplines utilise parallel programs to simulate different calculations in an attempt to improve and address different aspects of life in the real world. Scientific programs such as DL_POLY, NAMD (Wang *et al.*, 2011), Gaussian (McNicholas *et al.*, 2010), Vienna Ab-initio Simulation Package (VASP) (Davis *et al.*, 2015), and Material Studio (Wang *et al.*, 2015) are molecular dynamics simulations that are used mainly in material science and chemistry to simulate structures of atoms in different supercomputers across the world. Furthermore, WRF and Community Atmospheric Model (CAM) (Wehner *et al.*, 2014) are popular climatology programs that are used by scientists to simulate climate and weather around the globe.

2.2 Visualisation

Visualisation has become a critical component of simulating parallel programs (Rohrer, 2000; Ware, 2013). Parallel programs can generate two different types of data, namely, performance data (metrics) and model data. Performance data is visualised to analyse how the parallel program executed in a computational system, of which, model data is visualised to examine scientific calculations performed during the run. For example, it will be hard for the parallel program user to understand the behaviour of the model without visualising data generated during the run. Visualisation further enables the users to understand the reaction of different systems being modelled within a parallel program. It also helps scientists to share difficult ideas and communicate more information in a shorter period of time. The large data that would have taken years to manually analyse can now be visualised and presented in seconds. It is

inevitable that visualisation generate graphics that helps people to identify patterns of complex data generated using the computer.

2.2.1 Software for analysis and visualisation of parallel performance

Different visualisation tools utilise various techniques to analyse and trace the execution of the application in the parallel system. Some analysis tools trace and analyse the performance of a parallel model during its execution (Geimer *et al.,* 2010), while others visualise the statistics of the application after the simulation has completed (Becker, Frings and Wolf, 2008). Performance analysis tools also support different parallel programming technologies like MPI and OpenMP. The performance of parallel programs can be analysed using visualisation tools such as TAU, Scalasca, Vampir, Paraver and Periscope. These visualisation tools help optimise and analyse the performance of parallel programs by identifying bottlenecks like load imbalance, and inefficient network communication between message passing interface processes.

2.2.1.1 TAU visualisation system

TAU is a visualisation, profiling and analysis tool that is used to evaluate performance of parallel programs (Shende and Malony, 2006; Spear *et al.,* 2006). The tool was developed by the University of Oregon (Performance Research Lab), the Los Alamos National Laboratory (Advanced Computing Laboratory)[3] and the Jülich Research Centre[4] and supports message-passing parallelisation (for example, MPI). TAU enables users to generate visualisation reports, which help to identify performance bottlenecks for optimisation of parallel programs. TAU consists of three major components: instrumentation, measurement and analysis (Morris, Malony and Shende, 2007; Shende *et al.,* 2008; Gehrke, Ra and Connors, 2011).

[3] More information can be found on: http://www.lanl.gov/orgs/adtsc/ACSPOindex.shtml [19 January 2019].

[4] For more information, visit: http://www.fz-juelich.de/portal/EN/Home/home_node.html [19 January 2019].

The tool has several instrumentation methods, namely: source-level, pre-processor, compiler, wrapper library, binary, interpreter, component, virtual machine, multi-level, selective and compiler instrumentation (Shende and Malony, 2006). Source-level instrumentation relates to inserting instructions or probes into the program's code in order to invoke the visualisation tool to measure events and collect performance data (Spear *et al.,* 2006). In this case, the execution of the program is regarded as a sequence of events. When an event is executed, the analysis system activates the probe to collect the measurement data. This involves passing the source code to a pre-processor before it can be compiled.

Compiler instrumentation adds instrumentation calls, which are used to transform, instrument and optimise the code. The wrapper library instrumentation enables the performance analyser to substitute the standard library routines of the program with the instrumented ones, which are used to call the original routines. The interpreter instrumentation makes it possible to handle multiple instrumentations of different programming languages, for example, one can simultaneously instrument Python and Fortran programs at the same time. Virtual instrumentation represents the process of instrumenting the program in a system that consists of virtual machines. The multi-level instrumentation enables the profiling user to simultaneously instrument programs written in different programming languages, wherein selective instrumentation has the capability of excluding unnecessary events (Shende and Malony, 2006).

Moreover, TAU visualisations have two measurement methods - tracing and profiling. Tracing measurement provides information about the relationship between events, but generates many trace files. The tracing can observe the amount of time taken to execute each event on different processing threads. The profiling mode measures the different phases or regions of the program's code by collecting performance metrics (callpath data) of modules, routines, subroutines and loops (Shende *et al.,* 2006; Shende *et al.,* 2008).

In particular, the callgraph profiling system provides information about relationships between events (for example, calls to modules and routines) and the distance between the events from the root node. It utilises an easy-to-use graphical user interface tool, ParaProf, to analyse the performance of the program. TAU's parallel profiling tool uses callpath data to provide performance statistics such as callgraph and expandable tree-based reports. TAU can also

display 2D and 3D reports about MPI, threads and hardware performance counters, which are useful for identification of communication patterns between the parallel application and the computational system (Hammond *et al.,* 2011; Spear *et al.,* 2009). TAU can identify various performance bottlenecks such as excessive communication or inefficient distribution of computational events (for example, calls to modules and routines), messages sent/received, slow read/write performance, memory leaks and processor utilisation.

This study focuses on the callgraph profiling system, which is used to visualise the relationship between events such as calls to modules, routines and sub-routines. Figure 2.1 shows the analysis of the callgraph profiling system over the thermal and fluid sciences (TFS) parallel program, which is used to calculate heat transfer (Morris, Malony and Shende, 2007). TAU's ParaProf portable profiling system was used to visualise the performance of a computational fluid dynamics application (TFS) on a parallel system. The computational experiment of this fluid dynamics program was computed using many different number of compute nodes, however, Figure 2.1 shows computation of eight processes.

The performance of each process was visualised and analysed. In particular, the callgraph visualisations show eight boxes, which represent eight processes used to execute the program. Each box contains smaller boxes (red, yellow, green and blue ones) that represent computational events like calls to modules, routines and sub-routines. Critical events that consume excessive amounts of execution time are indicated with red boxes, while yellow, green and blue boxes show events that take up successively less time during the execution of the application.

Processing threads 2, 3 and 4, simulated expensive modules and routines (red small boxes) that consume undesirable amounts of execution time and may need to be optimised in order to achieve the scalable performance in a computational system. Processing threads 0, 1, 5 and 7 shows that some of the modules and routines (yellow and green boxes) may have slightly consumed excessive amount of execution time. The Micromegas parallel applications is commonly used in material science to study the plastic deformation that occurs in various crystalline materials under certain levels of stress (Ciorba, Groh and Horstemeyer, 2010; Bélanger *et al.,* 2007).

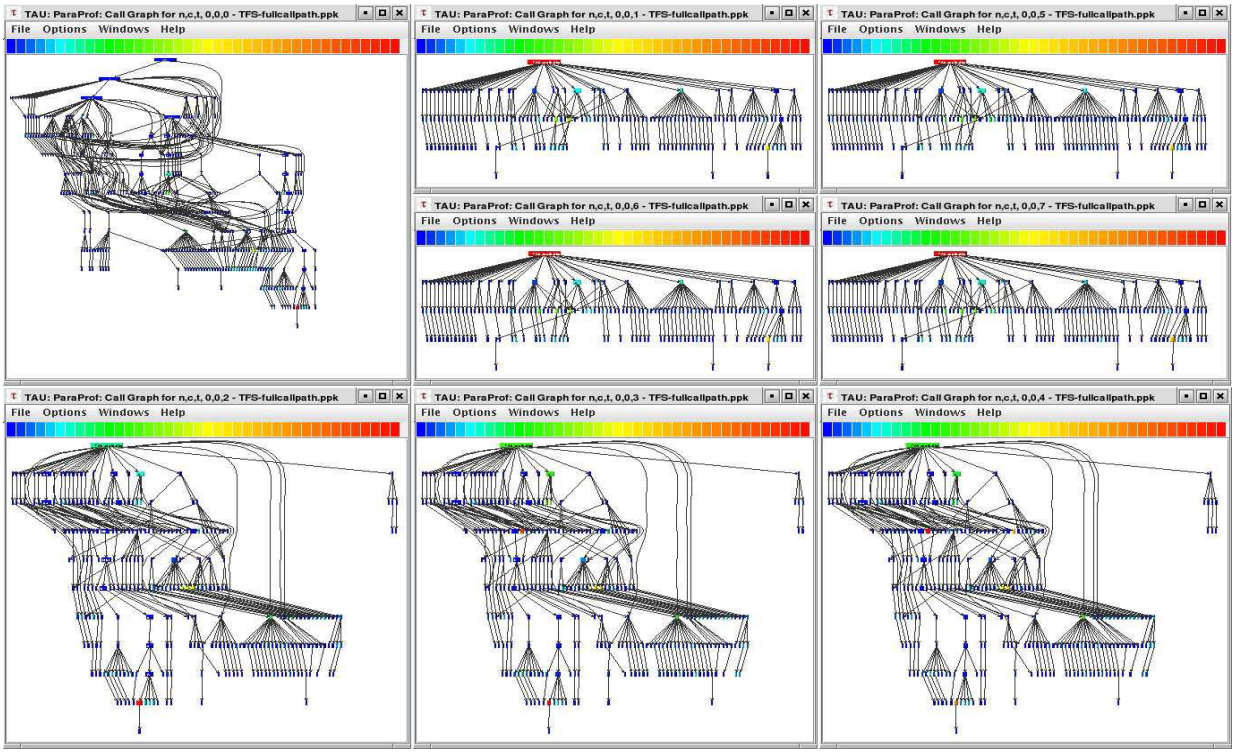

Figure 2.1: TAU's callgraph report used to analyse TFS parallelised using ParaWise (Morris, Malony and Shende, 2007)

Figure 2.2 shows the scaling of a parallel simulation of Micromegas program on four cores (Intel Xeon W3570 processors). The performance of Micromegas code was visualized using a callgraph profiling analysis tool in order to identify execution bottlenecks of the program. A TAU callgraph report was generated to show the relationship between these various computational tasks (calls to modules, routines and sub-routines) and execution time of each task processed on four cores, as shown in Figure 2.2. In this callgraph visualisation report, the red boxes represent the most time-consuming part of the application; green boxes represent the second most time-consuming part; blue shows the least time-consuming part of the program. It is easy to see that the module ELASTI, and sub-routine SIGMA_INT_CP consumed excessive amounts of the execution time. Ciorba, Groh and Horstemeyer (2010) reported that ELASTI:SIGMA_INT_CP used 77% of the overall total simulation run time. In this case, one needs to consider the relationship between the FORCE and the SIGMA_INIT_CP sub-routine in order to enable optimum performance. The other problematic event is module, CONTACT, and routine, UPDATE, which accounts for the second most time-consuming part of the program.

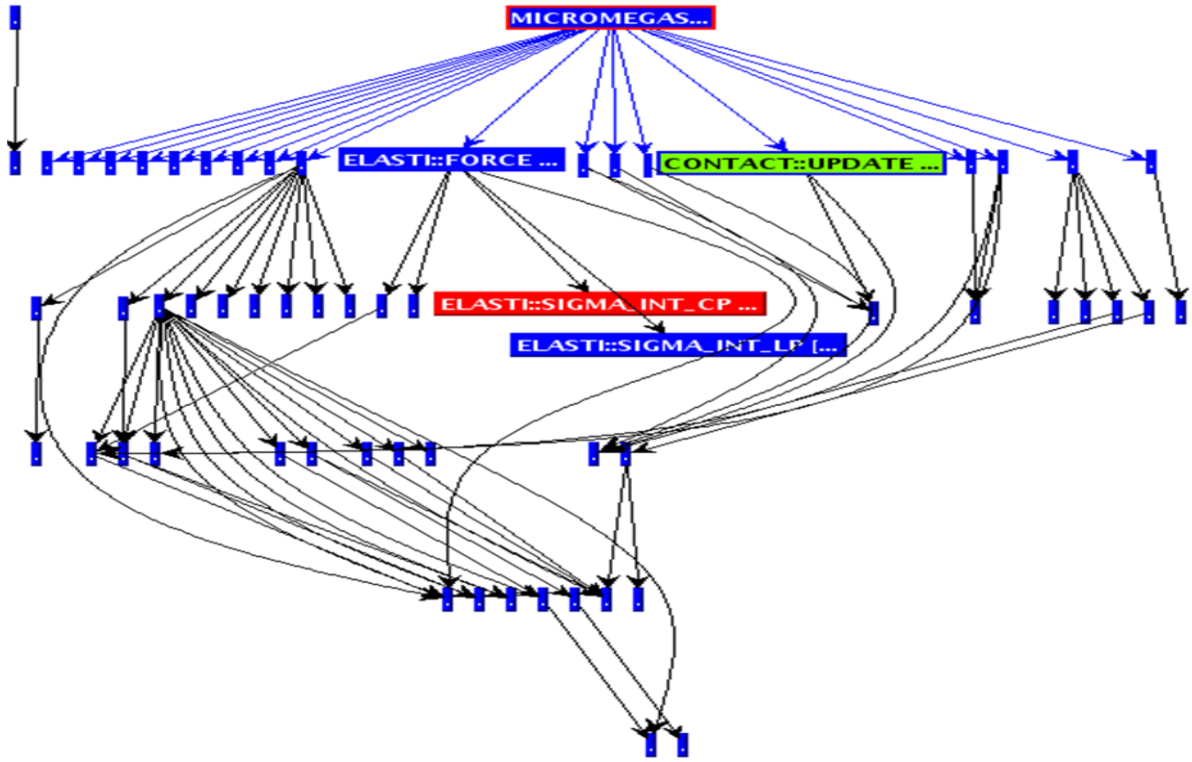

Figure 2.2: Callgraph performance analysis of Micromegas parallel simulation (none parallelised interaction forces) (from Ciorba, Groh and Horstemeyer, 2010)

After thorough consideration the SIGMA_INT_CP, sub-routine (used to calculate short range interactions between the dislocation segments), was parallelised and enabled the application to achieve a better performance. The relationship between parallel tasks of a Micromegas application were analysed and visualised. Figure 2.3 demonstrates the performance of the Micromegas parallel program executed using four Intel Xeon W3570 cores, and visualised using a callgraph profiling system. In Figure 2.2, the simulation is executed without a parallelised module, namely, FORCE used to calculate short and long range interaction between the forces. Figure 2.3 shows a visualisation report of a parallelised interaction between the forces within Micromegas parallel program. In these parallel simulations (Figure 2.2 and 2.3), the sequence in which modules and subroutines were executed is different from each other. For example, Figure 2.3 shows parallel simulation executing CONTACT:UPDATE, which calls on CONNEC:VOISINAGE, of which, Figure 2.2 demonstrates only the execution of CONTACT:UPDATE within that particular parallel region. The callgraph statistics (Figure

2.3) show that Module CONTACT and Routine UPDATE consume the greatest execution time (red colour), whereas the rest of the modules/routines are well balanced (blue). TAU does not have the ability to display all events due to limited length and width of the callgraph report. Figure 2.3 illustrates only the most important modules and routines that may impact negatively on the overall performance of the application.

Morris, Malony and Shende (2007) studied the relationship between events and child events in an effort to understand the performance of the TFS parallel program, as shown in Figure 2.4. The expandable tree report shows the relationship between routines and loops that were simulated on different number of processors. Most importantly, the report shows the duration spent on each routine and loop, with the associate number of calls and child events. This performance statistic is very useful when the computer scientist needs to understand how many times a particular event or child event is occurred during execution of the program.

It is therefore essential for the scientist to have both callgraph and tree-based reports in order to understand the relationship between the events and the number of times the events/child events were called. In the tree-based report dark blue boxes represent a normal event, while light blue boxes represent an event that consumed more time than expected during the execution of the program. Other colours, green, yellow and red (not shown in this report) represent different execution statuses of the events within the parallel program. Figure 2.4 further shows that the function ALGO function started the parallel execution of the program but this function consumed less execution time (0.007 seconds) compared to do loop (light blue box) within the Function AUSM which spent 18.106 seconds and was called 360 times.

The first do loop of AUSM needs careful consideration as one can see that it consumed almost sixteen times more execution time than other loops within the AUSM function. TAU also provides tree-based reports, which can identify the number of times a computational event is executed in the system. TAU's ParaProf system is able to generate useful tree-based reports, however, it does not have the ability to effectively visualise callgraph events.

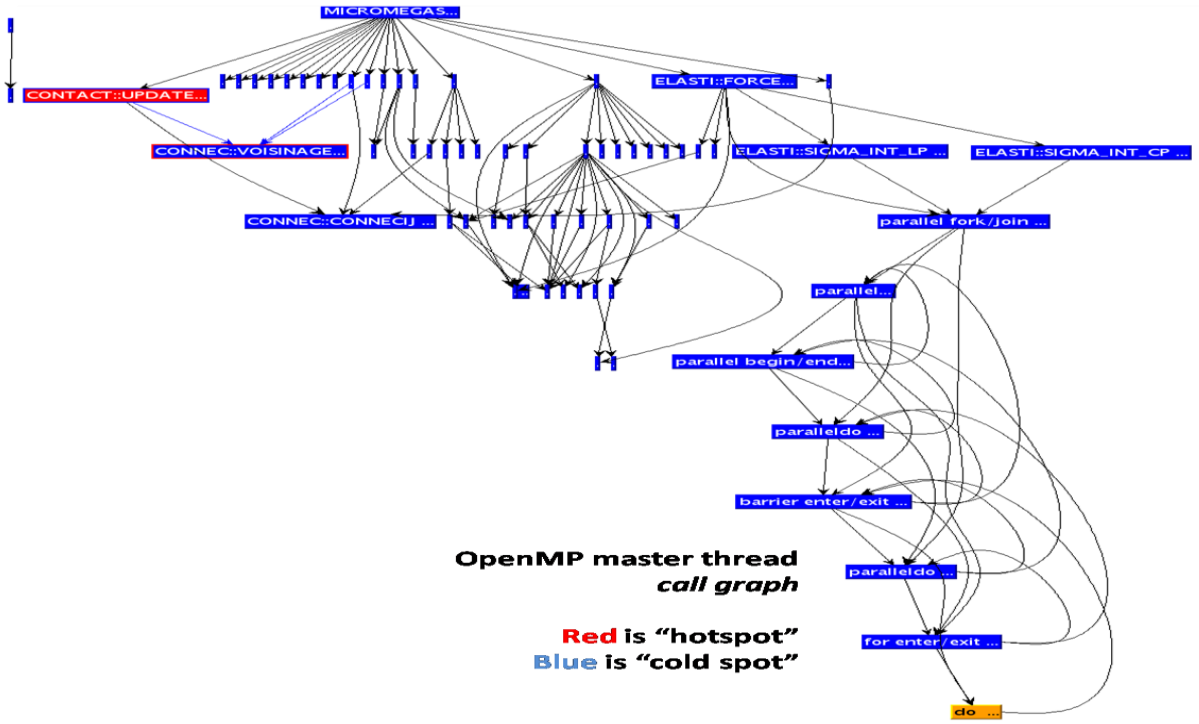

Figure 2.3: Callgraph performance analysis of Micromegas parallel simulation (parallelised interaction forces) (Ciorba, Groh and Horstemeyer, 2010)

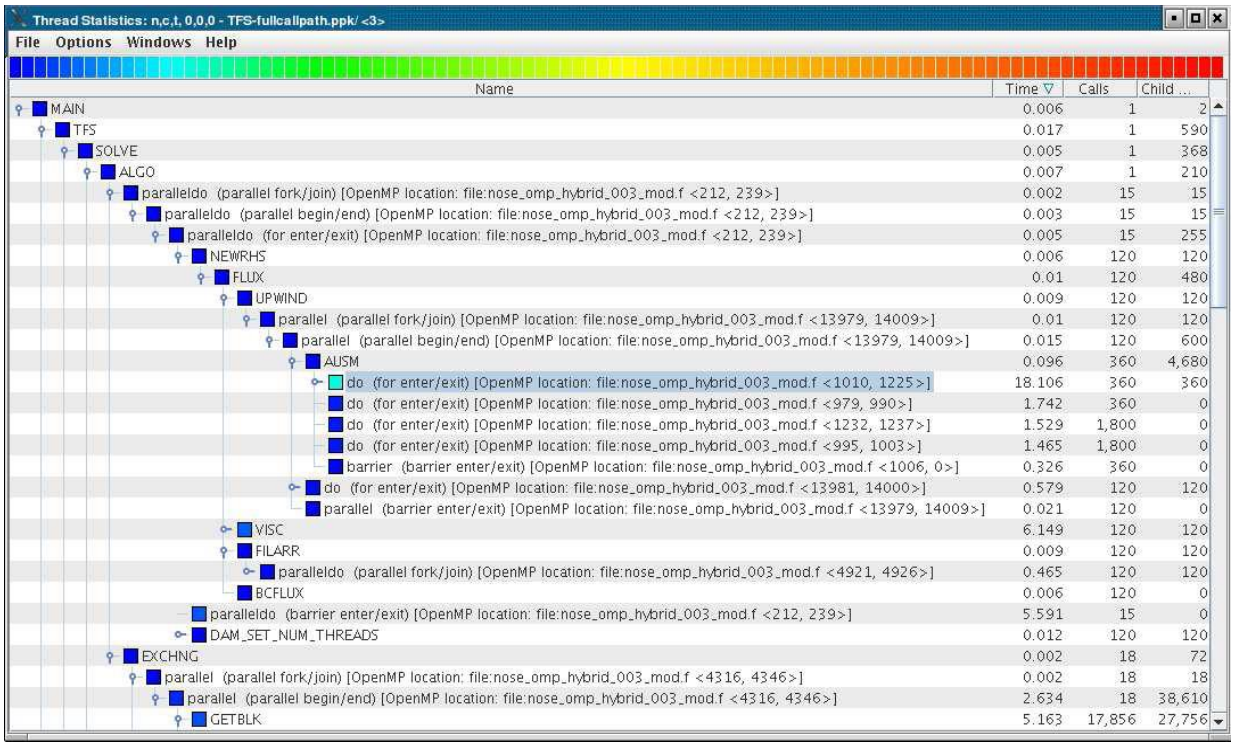

Figure 2.4: TAU's expandable tree-based report (Morris, Malony and Shende, 2007)

TAU callgraph visualisation system is not able to effectively visualise the performance of parallel programs. For example, events are presented using small boxes; network lines (black lines - sometimes blue) do not show clear connections between the events and the colour inside the callgraph report does not clearly indicate the status of an event. In some cases, program developers do not use performance analysis tools due to lack of sufficient knowledge regarding how the analysis tools work (Spear *et al.,* 2009).

Visualisation tools needs to be intuitive, interactive and informative for users to understand how the systems work. To compare different visualisations, Scalasca performance analysis tool is discussed. Scalasca performance analysis tool visualises message-passing communications such as message sent/received and message delayed/computed; this, as compared to TAU, which identifies performance bottlenecks within a parallel program.

2.2.1.2 Scalasca analysis tool

Scalable performance analysis of large-scale applications (Scalasca) is an open-source performance analysis tool widely used to identify message passing, threading communication and synchronisation within a parallel program (Geimer *et al.,* 2010; Böhme *et al.,* 2010). Salasca was developed by the Jülich Supercomputing Centre and the German Research School for Simulation Science Laboratory[5]. It provides both runtime summaries and event traces. The runtime summary is a performance report of the program during execution, while event tracing provides a detailed performance analysis report after run completion. During a run, Scalasca reports on the total execution time of the program; hardware (for example, memory and processor) utilisation, number of bytes transferred per task, message send versus receive requests, and write versus read activities of the application. The event traces provide reports on parts of the program that cause wait states (idle process waiting to synchronise with another process that is not yet ready to act); resources wasted on message-passing delays; and time spent on different processes, events and computation of the program (Böhme *et al.,* 2010). Figure 2.5 shows the performance of an astrophysics application, Zeus/MP2, that was simulated on 512 processors and analysed using Scalasca. Figure 2.5 (a) refers to the actual computing of the application, while (b) shows the duration spent on wait states. Figure 2.5 (c)

[5] For more information visit: http://www.grs-sim.de [18 January 2019].

represents the root cause of wait states, such as delayed synchronisation of messages within the system. Yellow reflects low values and red reflects high values of computation, wait states or delayed communication messages of the application. Figure 2.5 (a) shows that the parallel application did not efficiently utilise all processors to compute calculations - only few processors were used for computation. The Zeus/MP2 generates many idle processes that wait for an extended period of time (red) before synchronising the communication of messages, as shown in Figure 2.5 (b). In this case, one may need to focus on re-structuring the distribution of message passing and level of parallelism within the code of the application. The Zeus/MPs program showed less computational delays in the overall execution performed in the parallel system. Scalasca is used predominantly to trace performance bottlenecks, such as imbalanced message-passing communication, causes of wait states, and message delays associated with wasted resources (Böhme *et al.*, 2010; Böhme, Wolf and Geimer, 2012; Mey *et al.*, 2010). The main advantage of Scalasca is its ability to map wait states with the wasted computational resources, identify late message senders versus receivers, and display load and communication imbalance that occurred between the system and application. One of the challenges of this analysis tool is that it creates a file for each process/event and this may increase the need for more storage capacity in the system. Scalasca further depends on the local time of the execution nodes to accurately record the movement of the process and associate it with time, which can be disadvantageous for a system that has execution nodes with different time zones.

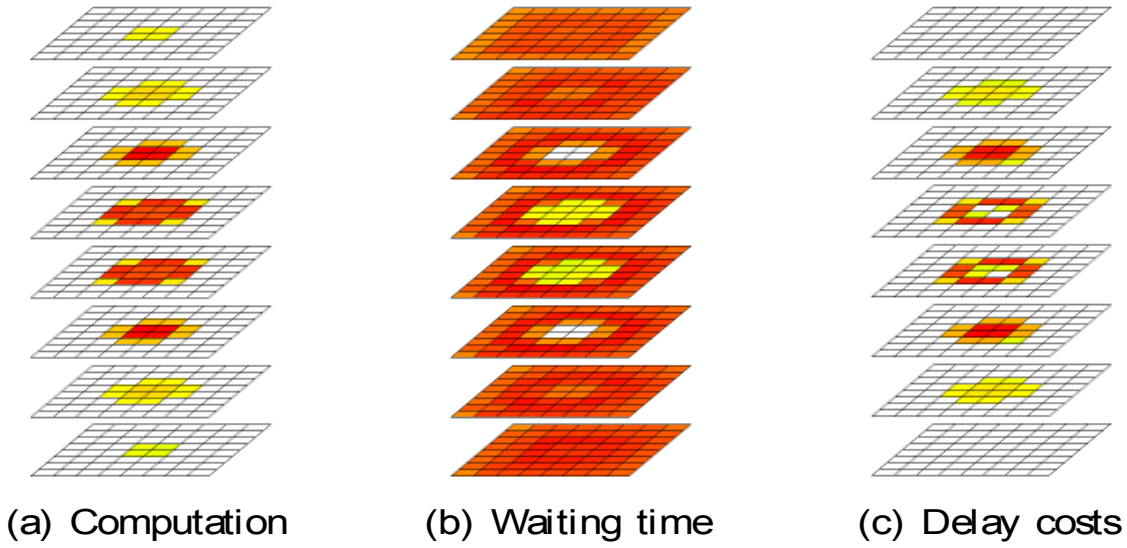

Figure 2.5: Scalasca performance report of the Zeus/MP2 program on 512 processors (Böhme, Wolf and Geimer, 2012)

2.2.1.3 Vampir visualisation tool

Visualisation and Analysis of MPI Resources (Vampir) is visualisation software that can be used to analyse the performance of parallel applications. Vampir is used to identify performance bottlenecks of parallel programs such as Implicit Radiation Solver (Knüpfer, Brunst and Nagel, 2005) and Semtex code (Kluge, Knüpfer and Nagel, 2010) using visualisation techniques that "zoom" inside the events and display detailed information about various computational actions (Becker, Frings and Wolf, 2008; Sunderland and Porter, 2007). This zoom analysis tool (Vampir) is available as an open-source and commercial package and was developed at both the John von Neumann Institute for Computing (Centre for Applied Mathematics of Research Centre) (Attig, 2006) and the Centre for Information Services and High Performance Computing [6]. Vampir can display graphical views, such as wait state changes, execution time of routines, communication volumes, and transmission rates between the processes. Figure 2.6 demonstrate a visualisation report of computational processes used to send and receive message during the execution of pF3C on a computation system. The pF3C is a parallel program used to study laser-plasma interactions in experiments (Isaacs *et al.,* 2014).

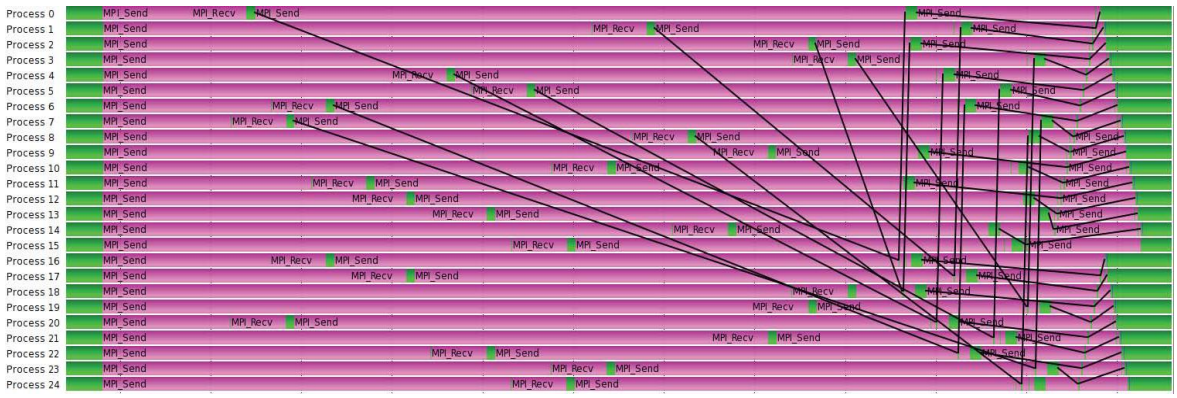

Figure 2.6: Vampir used to visualise the performance of pF3C parallel program (Isaacs *et al.,* 2014)

Vampir analysed the MPI communication used to parallelise pF3C parallel application on Blue Gene/G supercomputer, as shown in Figure 2.6. In particular, it was used to visualise messages sent/received during the execution of pF3C parallel program on 1024 processors, however,

[6]More info can be found on: https://tp.dresden-concept.de/en/partners/view/id/1132.

Figure 2.6 shows processes from 0 to 24 but the user have ability to expand the report for more processes to be displayed within the visualisation system. Different sizes of green square boxes represent *MPI_Send* and *MPI_Recv* functions used to parallelise computational tasks within the pF3C code. The size of the green box determines the amount of execution time taken to compute each function during the run. For example, large green boxes represent functions that consumed excessive execution time while smaller boxes illustrate functions that consumed lesser time during computation of the parallel program.

Black network lines are used to show the relationship between message passing processes, of which, pink layer is used as a background to display the connection between the message passing processes. In summary, Vampir's graphical user interface provides useful performance information such as time taken to: simulate the application's code, perform message-passing activities, and trace overheads, as well as displaying the overall total execution time of the program. The tool can also read and visualize data that is generated by other visualisation tools such as TAU; it visualizes various types of performance data irrespective of the visualisation tool used to instrument the parallel program. Vampir supports the Vampir Trace Format (VTF) and Open Trace Format (OTF) file formats to display graphical performance statistics of the program.

2.2.1.4 Paraver performance analysis system

The Barcelona Supercomputing Center[7] developed the visualisation tool Paraver to handle large amounts of visualisation data generated by parallel programs that are computationally intensive. Paraver has capabilities to "zoom" inside events (message sent/received) and provide performance statistics of modules and routines in the parallel application (Subotic *et al.,* 2010). Paraver can identify major performance bottlenecks, such as wait states, poor network communication, and hardware performance in MPI, OpenMP, mixed MPI+OpenMPI, MPICH and HPF applications (Dorta, Leon and Rodriguez, 2006; Truong *et al.,* 2001). The Paraver visualisation provides statistics on the application's parallelism level, including a comprehensive report about hardware counters and I/O activities. In the timeline display, the system generates graphs of the performance of the application over time; the statistics display

[7] For more information about the center visit: https://www.bsc.es [18 January 2019].

performs numerical analysis of data on a user-selected region to help identify the region of the code which needs to be optimised. Figure 2.7 illustrate the performance of OmpSs parallel applications used to compute on a heterogeneous system that comprise of SMP machines and traditional compute nodes equipped with processors. OmpSs is a parallel programming model that has ability to run on a heterogeneous system (Filgueras, 2014).

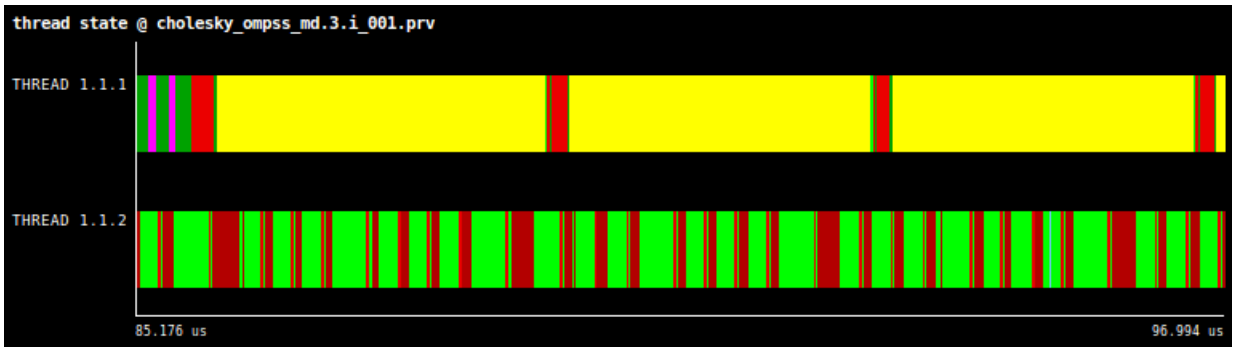

Figure 2.7: Paraver utilised to visualize the performance of OmpSs parallel model (Filgueras, 2014)

The above visualisation results (Figure 2.7) indicate 2 processing threads, namely, *THREAD 1.1.1* and *THREAD 1.1.2* used to compute OmpSs parallel program on a heterogeneous system. Each colour (e.g. yellow, pink light green and dark green) represents a task executed on a thread, however, the size of the each block with different colours illustrate the amount of time taken to execute the computational task. One can also be able to double-click the task to obtain more information about it.

2.2.1.5 Periscope analysis tool

Periscope (Knüpfer *et al.*, 2012) allows the users to start, stop and resume performance analysis at any time during execution of the program. This analysis tool is under development at the Technische Universität München. Periscope is able to detect and map problems in the specific area of the code, indexing and highlighting syntax of the code within the program (Benedict, Petkov and Gerndt, 2010). This process of mapping the problem to a specific area of the code helps to quickly identify parameters and optimise areas of the code that negatively impact on the overall performance of the parallel program. Moreover, Periscope has the ability to analyse

programs parallelised using technologies such as MPI, OpenMP, MPI+OpenMPI, MPICH and HPF; it simply attaches the target executable of the program without compiling the actual code of the program (Gerndt and Ott, 2009; Benedict *et al.*, 2010a; Petkov and Gerndt, 2010). Some visualisation tools, such as TAU, Scalasca and Paraver, need one to compile the application with necessary visualisation libraries before the user can attempt to analyse the execution of the program. Periscope follows the method of automatic instrumentation where information about the source code (for example, main routine, subroutines, loops, and data structure) is transformed and stored in a file for later use.

The data can then be used at any time to automatically compile the application without reading the source code, however this may be disadvantageous for users who frequently make changes within the parameters of the code (Petkov and Gerndt, 2010). Periscope utilise summary of runtime information to automatically analyse and discover problems within the program. Periscope contains a central management system, including analysis agents in the execution nodes that are used to collect measurement data and automatically refine the code during the execution of the application (Petkov and Gerndt, 2010). During the collection of data the tool builds and identifies sets of inefficiencies that will be evaluated instantaneously and discarded/approved based on whether the event has a high impact on the overall performance of the application. Should such inefficiency be approved, Periscope will automatically refine the discovered problem in a particular region of the code.

One of the important features of a visualisation system is ability to hover nodes, use different shapes to depict status of the nodes, filter and search information within a large dataset. Table 2.1 depict the performance analyses for an OpenMP version of the NAS Parallel Benchmark computed on a computational facility. In particular, left column contains objects (e.g. functions, loops) that caused performance bottlenecks during the execution of the NAS Parallel benchmark program. The right column contains the severity of the objects within the NAS parallel program. Different numbers in the right column are used to showcase the different negative impact that each object cause on the overall execution of the target application.

Property	BT	CG	EP	FT	IS	LU	MG	SP
ImbalanceAtBarrier					1	3		
ImbalanceInParallel-Sections								
ImbalanceInParallelLoop	12	13	1	8	2	9	12	16
ImbalanceInParallel-Region	6	9	1		2	8	2	5
UnparallelizedInSingle-Region						3		
UnparallelizedInMaster-Region	4					13	2	5
ImbalanceDueToNotEnough-Sections								
ImbalanceDueToUneven-SectionDistribution								
CriticalSectionContention								1
LockContention								

Table 2.1: Performance analysis of NAS parallel benchmarks using Periscope

2.2.2 Comparison of current tools

A good visualisation should be interactive, informative and easy-to-use (Rohrer, 2000), which defines an effective visualisation system. The research on visualisation of parallel programs has been an area of interest in the supercomputing community, with a number of tools developed for visualising application execution in order to obtain scalable performance on a parallel system. Paraver was developed to understand the communication patterns and load balancing of programs that are computationally intensive in the parallel systems. Subotic *et al.* (2010) demonstrated the ability to see the communication and execution time of events (for example, calls to routines) using Paraver, however, it does not provide details of the relationship between the events, like TAU does.

It is essential for visualisation tools users to understand the relationship between the executing application and the computational environment. The Vampir visualisation tool provides an extensive report on the performance of processors versus executing events that take place during execution (Knüpfer, Brunst and Nagel, 2005). The tool also has the capability to "zoom" inside the events of computational tasks in order to identify the root cause of performance bottlenecks incurred during execution of the program. Scalasca uses a different approach to

map the performance of the application against the executing computational system. Scalasca separates these computational factors into three aspects - computation, and messages waiting and delayed (Böhme, Wolf and Geimer, 2012). As for Periscope, it does not have features to present data graphically; instead it provides the application statistics using tables with rows and columns, which may be disadvantageous when there are many events recorded in the simulated system.

It is thus critical to have a system such as TAU's callgraph visualisation system, which can clearly display the relationships between the computational events of the programs simulated in the computational systems. Such visualisation information is important to optimise parallel programs for better performance in the computational resources. The design of the callgraph visualisation does not enable users to easily obtain performance information of parallel programs. The current design of the callgraphs is disadvantageous to the users and these include the use of many different colours to depict status of the nodes, network lines move over each other, incomplete names of the nodes and random different sizes of the nodes within the visualisation system.

2.3 Visualisation design

A visualisation is a conceptual or graphical representation of data, used to reveal useful information about a particular situation (Ware, 2013; Lima, 2010). Data can be visualised and presented using different methods, namely, scientific visualisation, information visualisation, artistic visualisation and network visualisation (Lima, 2010). The scientific visualisation is the process of exploring spatial data that are typically time-varying and multi-variate, including scalar and vector quantities (Brodile, 1992; Wang and Tao, 2017). Information visualisation is widely used as a method of understanding data patterns, connections and structure (Chen, 2006; Ware, 2013; Elkind *et al.,* 2014).

An artistic visualisation is more of concern rather than presenting data in a computer screen. However, this study focuses on network visualisation - is a graphical presentation of data displayed using collection of points (nodes) joined together using pairs of lines (Dunne and Shneiderman, 2013; Newman, 2018). Visualisations provide graphical representations of data to help people carry out tasks more effectively. There are a variety of different visualisation

idioms that can be used to present data on parallel execution, for example the tree and star visualisations shown in Figure 2.8 and 2.9. In this work, we seek to optimize the callgraph visualisation to effectively analyse the performance of parallel programs. It is essential, but difficult, to validate how effective a visualisation is, due to the sheer number of questions that may come up. One of the fundamental principles of analysing a visualisation tool is to look at the following three factors: computational resources, human capacity, and display capacity (Munzner, 2015). In particular, a visualisation should not require excessive computational resources (memory and processor speed) when processing visual queries.

The word "visual query" defines a request send by the user to the visualisation system, which responds with a visual representation of information (Catarci *et al.*, 1997). The visualisation should not expect a user to carry out many tasks when performing visual queries on the system because it may be difficult for the user to hold all these tasks in a memory. The focus of this study is to analyse human capacity when using callgraph visualisation system. Moreover, the designer should consider display capacity and ensure that the information displayed on the screen is neither overwhelming nor limited.

Interactive visualisation helps to control information displayed on the screen and supports multiple levels of displaying information - from high-level overviews, to summaries and detailed viewing of the information. An interactive visualisation is a visual representation of information that is able to interact with the user of the system (Bade, Schlechtweg and Miksch, 2004). In principle, an effective visualisation should support the user's tasks.

A good method to follow when designing an effective visualisation tool is to analyse existing tools. Munzner (2015) recommended that a designer consider the following, what do data users see?, why do users intend to use visualisation? and how is the visual aspect represented? In general, visualisations should present data using different methods in order to enable a quick understanding of the information however this is not the case with existing (original) callgraph visualisations. Scientific visualisation is regarded as the process of exploring data in order to extensively understand it (Brodile, 1992).

2.3.1 Network Visualisations

This study focuses on network visualisation, which is the process of analysing data to identify and locate relationships between different objects. Network visualisation further shows the relationship between objects (nodes) connected to each other via network links (lines). In most cases, one needs to utilise appropriate visual properties (for example, colour, size and shape) during the design of a network visualisation system. Some colours may be distractive rather than providing useful information to the user. It is therefore essential to choose appropriate colours when one designs a visualisation system. In the network visualisation colour, size and shape play an important role in conveying relevant messages about the data.

It can be challenging to visualize the relationship between many objects that contain different values, especially when the developer does not choose appropriate visual properties during the design of the system. The most important part of analysing data is being able to identify questions that require answering through that particular data. Answers will help to design well-structured visualisations that generate quality and meaningful graphics. Most notably, the design of the visualisation needs to be usable. Rogers, Sharp and Preece (2011) recommend that visualisation designers measure usability of the design with the following factors that are described in the below Table 2.2:

Effectiveness	How many users complete critical tasks?
Efficiency	How long do users take to complete critical tasks?
Memorability	How long does it take to remember previously executed tasks?
Learnability	How long does it take to learn the design?
Utilities	Does the design have good features that attract the users?

Table 2.2: Factors used to measure usability of the visualisation design

The network visualisation framework (Figure 2.8) depicts the relationship of different types of human diseases such as cardiovascular, dermatological, metabolic, muscular and nutritional. This human network visualisation demonstrates an excellent manner in which to use visual properties (such as colour and shape) to classify and locate the relationship between many

different objects. The human disease exercise was performed on 903 human genes, and diseases are classified into twenty-two categories. In Figure 2.8, different circles (objects) represent different categories of diseases and genes. The system uses different colours to represent different categories of disease, while white demonstrates the genes.

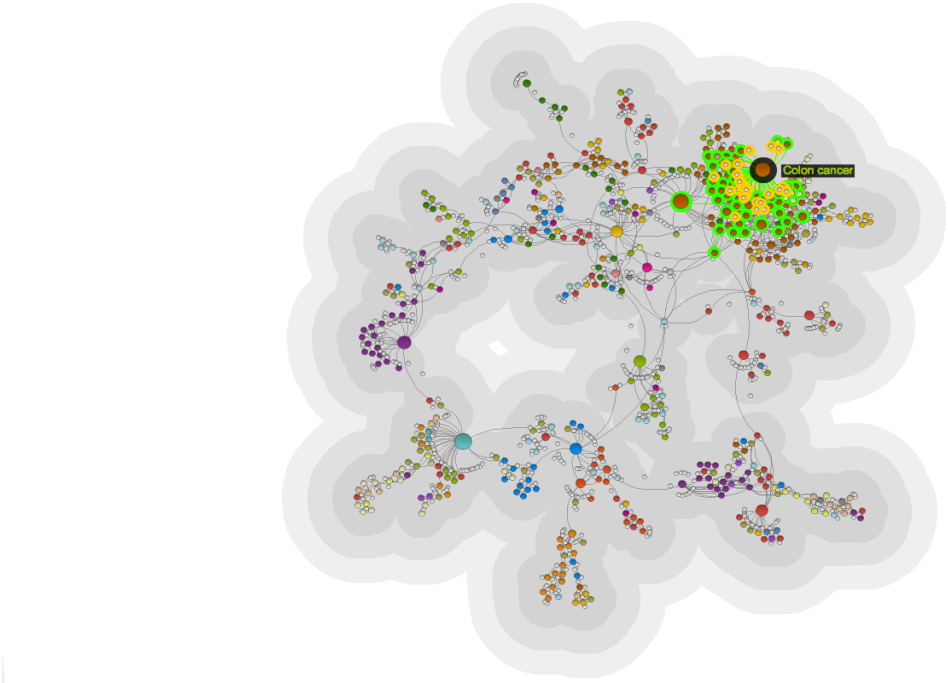

Figure 2.8: Human disease network visualisation (Lima, 2010)

The white nodes are not clearly visible within the grey background of the visualisation. Most interestingly, a bright green colour is used to select and demonstrate the relationship between objects of a particular group of diseases. At the same time, this green colour is used to highlight the relationship between the diseases and genes; the colour is different from all the other colours that represent diseases and genes. Furthermore, large black shading shows an active and selected circle. This is an interactive visualisation.

The name of the selected object appears clearly when one clicks the object, but the system does not show the names of other associated objects. As for the background, the grey colour adds more visibility to all the objects (diseases), except the white circles (genes). Grey connection links (lines) show the relationship between objects. In this human disease system, grey lines are

not ideal due to the fact that the connection links use the same colour as the background and these results in invisible connection lines between some of the objects. It can be challenging to analyse data that contains objects related to many other objects. Heer and Boyd (2005) analysed the relationship between social network users and group users, according to their relations (Figure 2.9). The community analysis visualisation tool (Vister) displays the relationship between users of the social network systems, such as Friendster, Tribe.net and Orkut (Heer and Boyd, 2005; Lima, 2010). Figure 2.9 represents the relationship between these social network users, grouped according to their relations in the "slices". Most interestingly, slices use different colours to make it easier for the user to identify certain groups closely related to each other.

Grouping of objects in the slices provides an opportunity to show clear connection links between the objects. The size of the connection lines is too small and could be expanded to provide more clarity on this system. The grouping of objects enables sufficient space to display the names of the objects in the system. In this social network visualisation, it is also alluring to see that each object contains the name and picture that can be used to identify it. Figure 2.9 further shows objects displayed within pieces of circles, which resulted in a more presentable view of the visualisation system. Figure 2.10 denotes the analyses of 2,200 social network users whose health status is obese (body mass index >= 30), and non-obese (Christakis and Fowler, 2007; Lima, 2010).

Each circle (node) represents one person. In this visualisation design, colour is used - in many cases to locate and provide relationship between the nodes. The yellow nodes represent social network users who are obese, while blue indicates non-obese status. At the same time, nodes with red borders depict women while those with blue borders show men selected from the dataset. Two different colours - purple and orange - are used to describe the relationship status between the nodes. In this regard, purple connection links (lines) show the relationship between the users who are friends or are married - orange lines show users who have familial ties. The size of each node is used as a key point to indicate the person's body mass index. It is evident that colour is one of the visual properties that can be used to showcase the relationship between nodes (objects).

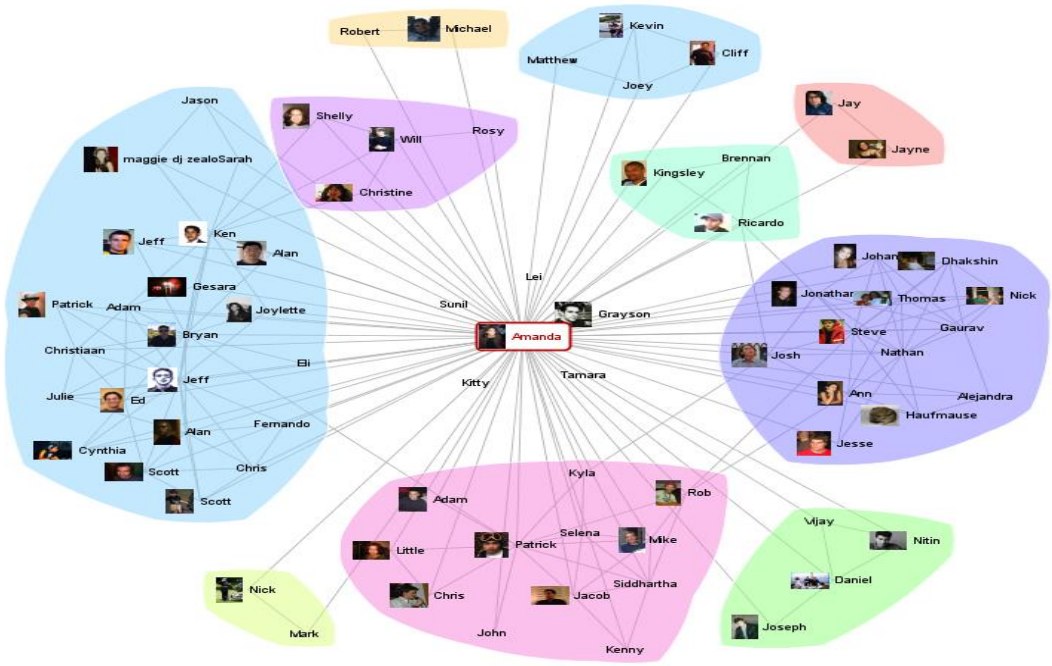

Figure 2.9: Analysis of online social networks (Heer and Boyd, 2005)

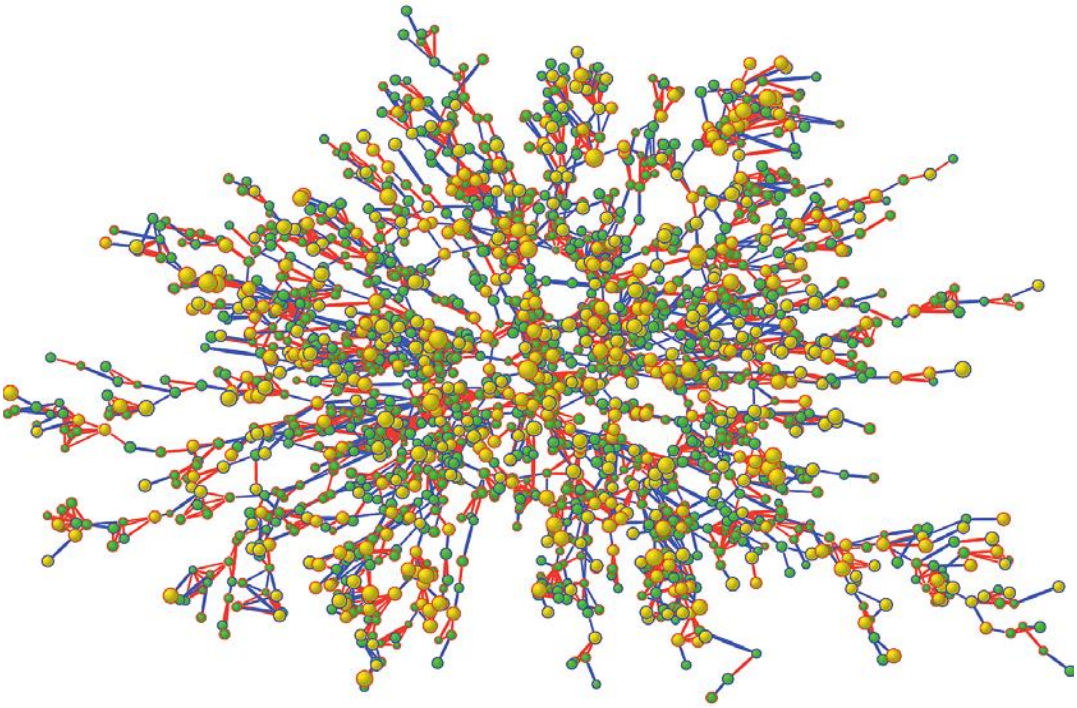

Figure 2.10: Visualisation of obese and non-obese social network users (Christakis and Fowler, 2007)

2.4 User-centred design

The process of involving users as co-designers of the system is called the "user-centred design" (Norman, 1999; Abras, Maloney-Krichmar and Preece, 2004; Gulliksen *et al.*, 2010; Garrett, 2010). The term "user-centred design" was originally introduced in the 1980s (Beyer and Holtzblatt, 1999; Abras, Maloney-Krichmar and Preece, 2004) and became widely used in various design fields such as graphical design, software design and architectural design (Rogers, Sharp and Preece, 2011). User-centred design involves different phases, namely, requirements gathering, prototyping and evaluation. Requirement gathering is used to establish users' activities in order to develop a system that supports their goals (Rogers, Sharp and Preece, 2011). Furthermore, it is used to identify users' requirements that are not likely to change during the development of the system.

Involves data gathering, analysis and interpretation in order to understand the manner in which users perform tasks, as well as their goals in the current system (Rogers, Sharp and Preece, 2011). The data-gathering can be achieved by means of questionnaires, interviews, observation or studying documentation. The questionnaires are drafted to get initial responses and can be used to identify users who are more likely to participate in the interviews or overall design process of the system. Interviews are often used to establish issues regarding the current operation of the system; it also helps to conduct face-to-face interviews with users in order to build a good relationship, making users feel involved in the design of the system. One could also introduce focus groups, where users can publicly discuss issues about the current system. Focus group discussions help in exchanging ideas about the current system.

Most notably, direct observation and record taking by the designer, of the users when performing tasks, is vital for understanding how they execute tasks and in what manner the tasks are performed. It is also worthwhile reading documentation regarding the current system in order to understand further, how users perform their daily tasks. Researching similar products will help the designer to make comparisons, which may help establish requirements. The designer may decide to follow more than one data-gathering technique in order to get different views about the current (original) design of the system. The process of requirements activities iterates repeatedly until stable user requirements are identified, and this involves continually analysing gathered data to achieve deeper understanding of the requirements.

Cooper, Reimann and Cronin (2007) describe interaction as a means to improve understanding, communication and efficiency among the users of the system (Rogers, Sharp and Preece, 2011). In an interaction design model (Rogers, Sharp and Preece, 2011), the designer is expected first to observe the gathered data and attempt to make patterns of it, for example, to associate visualisation users' experiences with qualification. From there, data can be structured in a form of quantitative and qualitative analysis, where quantitative analysis refers to assessing data that is numerical (or can be translated into numbers) and qualitative analysis is a method of examining data that is non- numerical (Rogers, Sharp and Preece, 2011).

Both quantitative and qualitative analysis can be used to evaluate data generated through interviews, observation and questionnaires (Rogers, Sharp and Preece, 2011; Munzner, 2015). In most cases, interviews are recorded using audio/video equipment and interviewers' notes. The interviewer (that is, the designer of the system) should listen to the recordings immediately after interviews in order to clarify notes taken during the meeting. It can be challenging to elucidate some of the respondents' answers if the interviewer listens to recordings after a prolonged period. For interviews, closed questions, such as age and years of experience, should be analysed quantitatively, while open questions (such as job title, qualification and programming language) can be assessed in a qualitative manner.

It can be difficult to qualitatively analyse and identify the relationship between general responses generated through interviews, for example identifying system administrators who believe programming is complicated in a computer science degree. Some statistical techniques like observation may be required to help analyse such general responses. Observation data such as notes, photographs, video, audio recordings and data logs can be quantitatively and qualitatively analysed accordingly.

In this case, qualitative analysis can be used to evaluate the notes generated when users perform tasks on the system. The notes can be expanded upon when video and audio recordings are analysed, in order to get a deep understanding of the users' reactions to the tasks. Nevertheless, data logs are measured using quantitative analysis techniques to provide a clear understanding of the tasks performed by users during the observation. The questionnaires aid in quickly perceiving the goal of the users when executing tasks on the system. The questionnaires can be recorded using an online survey or a printed document. It is recommended that a questionnaire

analyser first “clean” any errors occurring (incorrect users’ responses), in order to get an accurate picture of the data (Rogers, Sharp and Preece, 2011). It will then be easy to filter data by extracting numbers (quantitative analysis), which can also be associated with statements (qualitative analysis) generated using online surveys or printed documents. The goals, tasks and needs of the users can be further analysed by studying documentation and conducting research about the system to be designed. Analysed data can then be interpreted into percentages and averages (Rogers, Sharp and Preece, 2011).

For example, 50% of the users managed to identify normal objects used to simulate programs. These figures (percentages and average) will therefore produce graphical representation of the data to makes it easier for the designer to understand users’ goals and challenges when using the system. Consequently, the designer will be able to set the initial user requirements based on the interpretation and presentation of the findings. Establishing users’ requirements will also aid in fixing errors at an early stage of system development.

In this research study, the author will follow data-gathering techniques, namely: questionnaires, interviews, observation, study documentation, and research about the system. After establishing requirements, the designer will be able to develop prototype designs, which will be evaluated by the users of the system. There are two different types of design - conceptual and physical. The former deals with what the system will do and how it will do it, while the latter is concern with the layout of the design, such as screens, buttons, icons, textboxes and menus (Rogers, Sharp and Preece, 2011).

2.4.1 Prototyping

The prototype is a skeleton system that allows users to both interact with it, and explore different ideas through it (Rogers, Sharp and Preece, 2011). Moreover, there are different types of prototypes - low-fidelity and high-fidelity prototype systems. The low fidelity prototype is a design that does not mimic the final product, for example sketches, storyboards and index cards. High-fidelity prototypes refer to a design that mimics the final design and is capable of interacting with the users. The low-fidelity prototypes are useful for exploring ideas because it is easy to generate and modify these prototypes. In fact, low-fidelity prototypes are essential for assessing multiple designs without using excessive effort to generate designs. It does not

provide detailed information to the users. The high-fidelity prototype interacts with the users but demands a great deal of time to design it. PowerPoint can combine both low- and high-fidelity prototypes because it provides paper-based prototypes with a polished software design (Rogers, Sharp and Preece, 2011). The evaluation phase is a stage used to assess the physical and conceptual designs of the system. Rogers, Sharp and Preece (2011) suggested that evaluation usually involves observing users to measure usability of the system, however, in some cases one can also model users' behaviours to measure their performance on the system. Various evaluation methods - controlled setting, natural setting, and any setting - are used to determine the environment (for example, laboratory, home and office) in which users will be evaluated (Rogers, Sharp and Preece, 2011).

The *controlled setting* is an evaluation method that assesses users in a controlled environment (for example, a laboratory) where users will know what to do, how to do it and at what time to perform the tasks. In addition, the controlled evaluation method is useful for recording users' tasks and behaviour during testing of the design (Rogers, Sharp and Preece, 2011; Munzner, 2015). As a result, the designer is able to easily identify usability problems and user experience. The *natural setting* assesses the design in an environment such as online public communities. In natural settings, the designer does not have control on how users either perform tasks or behave when executing tasks.

Any setting refers to an evaluation method where the designer consults with experts, researchers and consultants to identify the most obvious usability problems of the system to be designed. Any setting is similar to heuristics evaluation. Heuristic evaluation is an inspection method used to identify usability problems, where experts examine the system according to heuristics, or guidelines (Zuk *et al.*, 2006). The designer needs to know the usability problems and users' experience (Rubin and Chisnell, 2008; Tullis and Albert, 2013) after developing sketches, storyboards, indexes and prototypes. As a result, usability problems and users' experience help the designer to create an effective interactive system. The method of gathering data using observation, interviews and questionnaires can be utilised to test the usability and users' experience (Abras, Maloney-Krichmar and Preece, 2004).

2.4.2 Case studies

A user-centred design technique was utilised to re-design a crime analysis system (GeoVista CrimeViz) to better understand the crime activities taking place in different areas of Washington, USA (Roth *et al.,* 2010). The software companies also adopt similar strategy to develop their products. For example, users (police) were involved in different development stages of re-designing the crime visualisation tool. Consequently, GeoVista CrimeViz became an easy-to-use tool that helps users to easily identify crime activities and dedicate resources (police officers) to relevant locations. Autodesk, Inc has adopted a user-centred design method called Agile software development (Sy, 2007).

Autodesk previously used waterfall technique to develop various products for customers, however, the company later decided to test user-centred technique in the development process of the products. The company's user experience team together with developers adopted user-centred design method to test usability of Agile software. Consequently, different prototypes of the software were developed and tested by the users. Interviews were conducted with the user in the laboratory and field. Various usability problems were identified and immediately resolved throughout the entire development process, which follows a user-centred design approach. As a result, a user-friendly Agile software was developed, which enable the users to easily help customers develop products (Sy, 2007).

A design study was conducted to develop a dashboard network visualisation used to analyse the 2000 medical records of patients diagnosed with prostate cancer (Bernard *et al.,* 2018). In this exercise, five non-experts, five visualisation experts and four medical experts evaluated different prototypes including final design of the dashboard visualisations. These experts tested the visualisations designs over nine iterations, which resulted with an effective dashboard network visualisation that provide useful guidance on how to treat different patients based on their medical history.

The user-centred design method was also followed to re-design the user-interface of the battlespace visualisation game, namely, Dragon - which was developed by Naval Research Laboratory's Virtual Reality Laboratory (Gabbard, Hix and Swan, 1999). To develop new interface, Dragon developers interviewed users (US Navy personnel) who used the game to

simulate a battle of war in the field. US commanders and technicians were further asked to perform battle-field visualisation tasks using Dragon in order to observe how users execute tasks on the visualisation system. The user interaction design experts evaluated the interaction between the system and users. Experts found various usability problems such as inability to zoom an object, inflexible flight stick, inadequate graphical and textual feedback to the user. Different prototypes were developed and evaluated by the users for a period of nine months, which resulted with an informative and interactive visualisation of the Dragon visualisation game.

2.5 Summary

This chapter outlined MPI parallel programming that can be used to implement parallel programs for different supercomputing architectures. Different types of parallel applications used in various scientific disciplines were also discussed. Most importantly, the background theory regarding different visualisation tools TAU, Scalasca, Vampir, Paraver and Periscope were described in detail to provide useful information that can be used to optimise the performance of parallel programs. A thorough comparison of these visualisation software packages was also outlined in order to show their differences. The description of several ways of presenting visualisation data was also highlighted in this chapter. Most notably, different methods of evaluating visualisations with the users/experts were discussed. Three evaluation methods, namely, controlled setting, natural setting, and any setting were discussed to showcase various ways used to examine usability of visualisation designs. The controlled setting and any setting will be used to evaluate the new designs against original callgraph visualisations. For more information about evaluation of the callgraphs, please refer to Chapter 4.

CHAPTER 3: TAU Paraprof Callgraph Visualisation

This chapter provides an overview of the TAU's ParaProf callgraph visualisation system used track the performance of parallel programs such as WRF-3.5 and DL_POLY, when executing on the CHPC's Sun cluster. Furthermore, visual queries that are typically performed by users are discussed and summarised in this chapter. We further focus on the design of the callgraph network visualisation tool used to present the relationship between various objects (such as modules, routines, subroutines) during execution of the program. The quality of a visualisation can be affected by visual properties such as colour, shape, size, orientation, texture, position, and contrast of the visual elements (Lima, 2010). Therefore, the strengths and weaknesses of the original (current) callgraph visualisation are discussed and summarised under the following categories: colour, size, space and texture. This chapter aim to demonstrate how different visual properties affect the design of the original callgraph visualisation system.

3.1 Overview of the callgraph visualisation

For our analysis, TAU's ParaProf callgraph visualisation tool was used to analyse and identify performance bottlenecks for molecular dynamics package, DL_POLY, executed on the CHPC Sun Microsystems supercomputer. The DL_POLY molecular dynamics package was selected due to its popularity on the CHPC's Sun cluster. Most of the CHPC supercomputing users utilise this molecular dynamics simulation to perform scientific work in various fields such as chemistry and material science. We further analysed a weather model, namely, Weather Research and Forecast (WRF), in order to understand how the callgraph system handles applications using many modules, routines and subroutines to perform calculations in the computational system.

The WRF is the parallel application that has been selected for the purpose of this study due to its popularity for use in the CHPC Sun cluster. We conducted a design study by analysing the problem facing users (Sedlmair, Meyer and Munzner, 2012) who run parallel programs such as DL_POLY and WRF used to perform scientific calculations on the supercomputers. This WRF weather model was simulated on forty-eight processors and visualised using TAU's callgraph system running in the CHPC Sun cluster. TAU's callgraph visualisation is used to identify the

relationship between different parts (objects) of the program and to establish time taken to execute each object during the run. The callgraph is the network visualisation that shows the relationship between different parts of the program, such as message-passing functions, modules, routines and subroutines, during execution of the program. TAU outputs profile files, where each file represents the computational events executed in a single processing thread: for example, profile.0.0.0 file will have events that were simulated on thread 0. TAU's callgraph has two different modes for generating these files - exclusive and inclusive. The exclusive mode only counts the time spent in the event, without calculating the time consumed on subroutines.

For example, the exclusive time spent on the main routine may be very low because the main routine usually just calls other subroutines. The inclusive mode includes time spent on subroutines. In this case, the inclusive time spent on the main routine will be the total execution time of the application hence every routine called is the subroutine of the main. In the callgraph visualisation, black network lines indicate inclusive mode, as shown in Figure 3.1. Blue network links indicate exclusive mode, as presented in Figure 3.2.

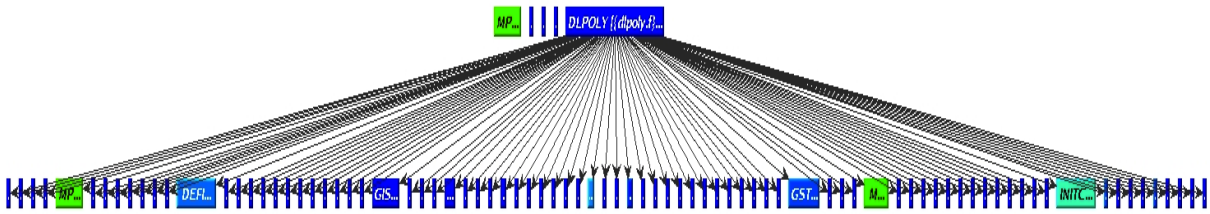

Figure 3.1: Performance analysis of DL_POLY_2.18 on the CHPC supercomputer.

Figure 3.1 shows the first view of an inclusive-mode callgraph for the execution of DL_POLY on forty-eight processors using CHPC's Sun cluster. The callgraph shows objects (small coloured boxes) with white text to display the name of each object. In the callgraph visualisation, different objects (modules, routines, subroutines and functions) are presented in different colours depending on the amount of time each object took to be executed in the system. The colour map for node execution time ranges from dark blue (least time) to red (most time). Dark blue nodes represent objects that consumed normal execution time, while light blue

boxes illustrate objects that spent an amount of time close to the normal execution time. The light green nodes represent objects that consumed the amount of time slightly closer to the normal execution time, whereas the green containers demonstrate objects that have consumed the maximum expected normal execution time. Furthermore, yellow nodes (not displayed) represent parts of the program that consumed between normal and moderate execution time. The orange and red colours are not shown because none of the objects took either average or excessive time to execute on the system. Orange nodes depict objects that have consumed a moderate execution time; red containers demonstrate objects that consumed excessive (meaning immoderate) execution time during execution of the program.

The callgraph users may click on the node in order to obtain the inclusive and exclusive time value taken during the execution of that particular object. The colour map will be discussed further below. Figure 3.2 shows the same callgraph system in exclusive mode. The DL_POLY_2.20 used the same configuration (69120 atoms and 1000 degrees celsius) as DL_POLY_2.18 however performance results are not the same. Figure 3.1 demonstrates that DL_POLY_2.18 was performing well using forty-eight processors because all objects are shaded with blue.

Figure 3.2 shows an orange object - *MPI_Init()* - while other objects are coloured with dark and light blue. Orange nodes do not indicate critical parts of the program that reduce performance of the application; instead, they demonstrate parts of the application that may have little impact on the overall performance of DL_POLY. It is further expected that if DL_POLY_2.20 can simulate a long run (using more than 69120 atoms) then this molecular dynamics model will perform adequately, hence the *MPI_Init()* function only runs at the beginning of the simulation.

Moreover, the above callgraph analysis tool uses red borders to highlight active object, which is an object that the user has selected. The above visualisation results (Figure 3.2) show that DLPOLY module is an active and selected node. The study's author manually dragged and increased the size of the objects (small boxes) in order to establish the names of the objects, as well as the relationship between them - in particular, the relationship between the selected main module, *DLPOLY*, and other objects. The callgraph users can also display the objects' full name by clicking *Options*, *Box width by*, followed by *Name length*. Adjustment of the callgraphs screen settings does not eradicate the problem of displaying the full names of the

objects because some network lines move over the boxes, which consequently prevents objects' names being displayed accordingly, as shown in Figure 3.3.

Figure 3.2: Visualisation of DL_POLY_2.20 on CHPC's Sun cluster

The visualisation results of WRF could not fit on the screen. Consequently, the results are divided into two views - Figure 3.4 *(A)* and *(B)*. In this case, Figure 3.4 *(A)* shows results from the left side, whereas Figure 3.4 *(B)* displays the right side of the callgraph system. Figure 3.4 *(A)* shows the relationship between different modules, routines and subroutines used to simulate WRF in the Sun system. Figure 3.4 *(A)* shows dark blue, light blue, light green and green nodes, which indicate parts of the program that consumed normal execution time during the simulation of the WRF parallel program. Figure 3.4 *(A)* shows visualisation results of WRF where white dots represent full or incomplete names of the objects within the nodes of the callgraph system. Figure 3.4 *(B)* illustrates some complicated visualisation results of WRF simulated on forty-eight processors running on the CHPC's Sun Microsystems cluster. Figure 3.4 *(B)* shows many black network lines move over different nodes. In some cases, different network lines form one large black pole, making it difficult to understand the connection between the nodes and network links.

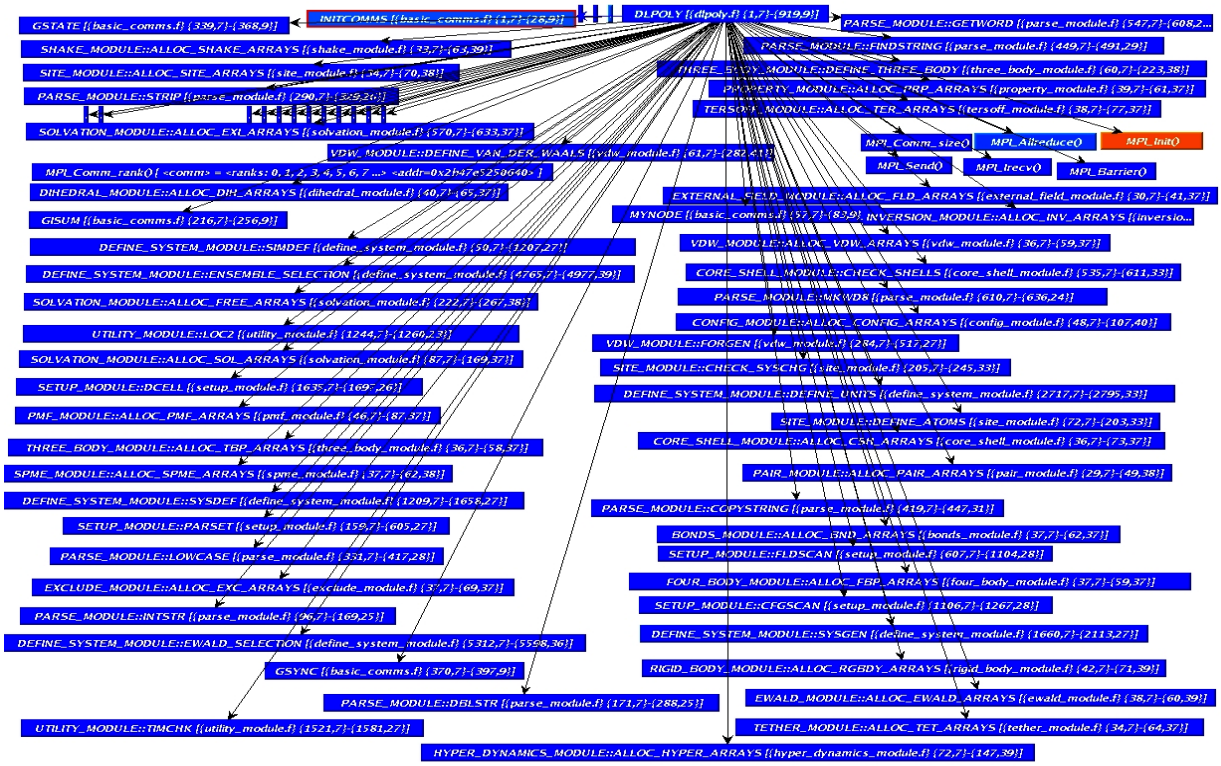

Figure 3.3: Visualisation of DL_POLY_2.20 on forty-eight processors

The relationship between objects cannot be easily identified and located in the right-view of the callgraph system. Some of the nodes are hidden underneath the network lines, which is problematic since the details of the objects are useful in order to understand the relationship and execution status of the WRF parallel program. In Figure 3.4 *(B)*, orange nodes depict objects that consumed moderate program execution time. The red nodes illustrate objects that consumed immoderate execution time while light blue and dark blue nodes represent the normal execution time, as described in Figure 3.4 *(A)*.

Various boxes contain white dots and incomplete names, which represent the names of each object computed in the computational system. The Figure 3.4 (*B*) visualisation results show that the original callgraph system is unable to effectively visualise applications that have many objects (modules and routines, calling subroutines). The users are likely to perform the following visual queries when using callgraph visualisation to analyse the performance of parallel programs:

How long does an object take to execute on a single processing thread?
What is the relationship between objects that have kinship with each other?
How can one identify groups of objects that are associated with each other?
What is the cause of the performance bottleneck in the application?
Which objects consumed normal, moderate or excessive execution time?
What is the name of each object?

Table 3.1: Visual queries performed by users

Table 3.1 depicts visual queries that are likely to be performed by users on the callgraph visualisation. The first query request the user to outline an amount of time taken to execute an object on a single processing thread while the second query enquire about the relationship between objects that relate to each other. As for the third query, it defines how the user can identify group of objects that associate with each other. Furthermore, fourth query ask the user about the cause of the performance bottleneck within the application. The fifth query enquires to the user about different status (normal, moderate and excessive) of the objects within the program. As shown in Table 3.1, sixth query request the user to name each object computed during the run.

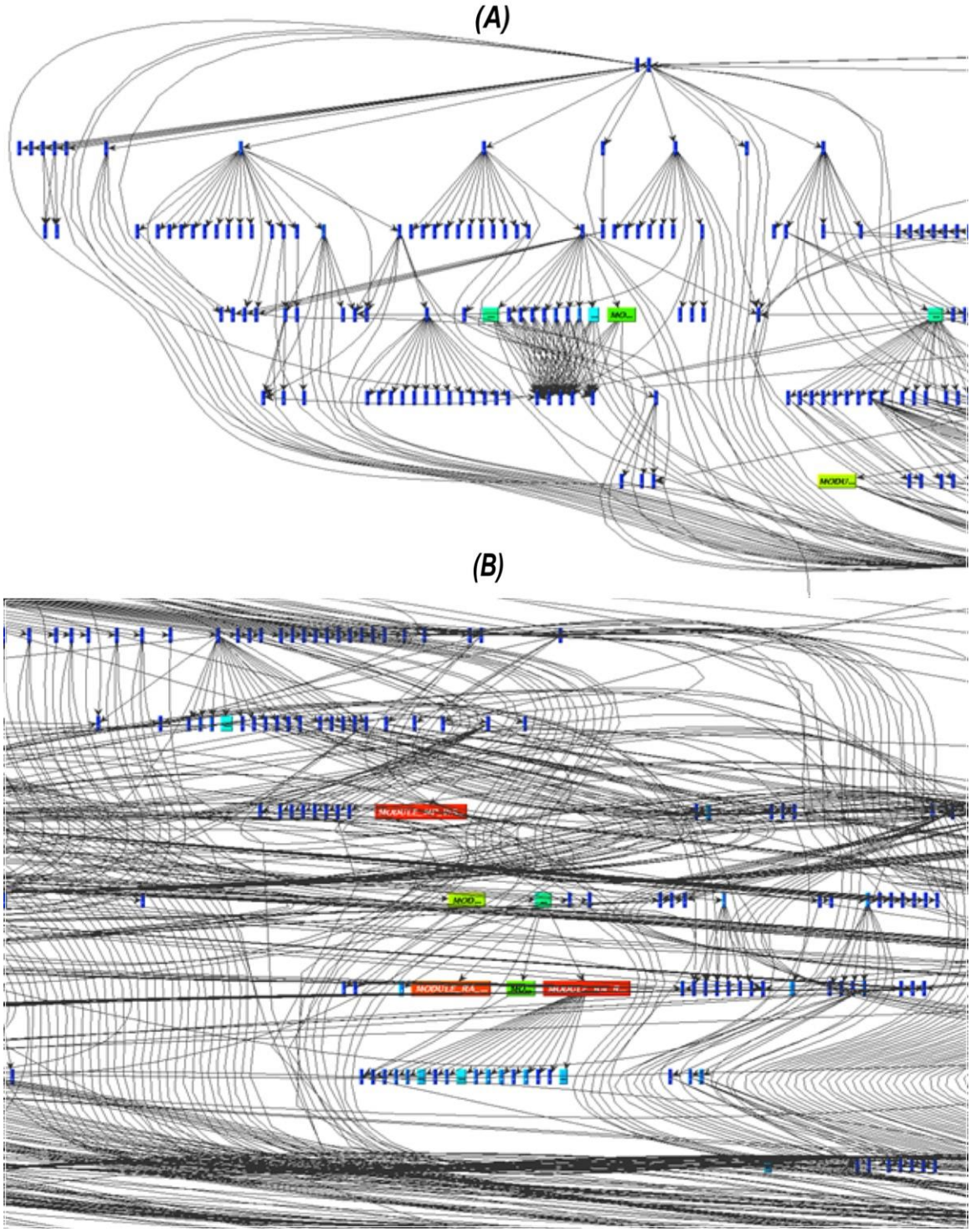

Figure 3.4: Left-side *(A)* and right side *(B)* visualisation results of WRF-3.5 on Sun cluster

3.1.1 Colour

Colour is one of the most essential visual properties that contribute to an effective, informative and attractive network visualisation system. In the additive colour model, it can be represented using the following formula:

$$C \equiv rR + gG + bB \qquad \text{(Ware, 2013)}$$

C represents a colour to be matched; R, G and B are the primary sources to be used to create a match; r, g and b indicate an amount of light from each primary source; and ≡ is a symbol used to create a match. For example, black ≡ 0R + 0G + 0B. The science writer, Ware (2013), explicitly emphasised that colour is extremely useful in visualisation and is excellent for categorising, but not for shaping an object. The original callgraph systems clearly do not use colour effectively to help users analyse the performance of parallel programs.

TAU's callgraph system utilises a saturated blue colour to demonstrate the state of an object that consumes normal execution time, as shown in Figure 3.1. In fact, TAU uses dark blue to represent objects that consumed normal execution time, while light blue demonstrates objects that consumed close-to-normal execution time. It is complicated for visualisation users to establish which object consumed normal execution time because both dark and light blue represents almost the same situation. As demonstrated in Figure 3.1, TAU's system further utilise light green to depict objects that have spent an amount of time slightly closer to the normal execution time.

The green colour represents objects that consumed the maximum amount of expected normal execution time of the program. In this case, one is unable to precisely pin-point the objects that consumed normal execution time because many different colours - dark blue, light blue and light green and green - represent practically the same execution status of the nodes. As previously discussed in Section 3.1, yellow nodes are used to represent objects that consumed between normal and average execution time, but they do not provide a clear status regarding the detail of the node. Orange is used to represent objects that consumed moderate execution time. As previously discussed in Figure 3.4 *(B)*, callgraph views use red to show objects that consumed excessive execution time. It is advantageous for callgragh systems to use red as a representation of objects that cause critical performance problems because the colour demands

more attention. TAU's callgraph system utilises too many colours to describe situations that are effectively the same, which causes misunderstanding over the performance of the program. Keller and Keller (1993) state that red and yellow colours demand more attention while blue is difficult to focus on - the callgraph view in Figure 3.2 shows dominant light and dark blue colours. TAU's callgraph system struggles to identify the relationship between objects (such as modules, routines, subroutines and functions) utilising a saturated blue colour. It would be beneficial for the callgraph tool to use different colours to represent the various objects, and identify relationship between each other.

3.1.2 Size and space

The first view of the callgraph visualisation tool shows small boxes with white dots, as shown in Figure 3.1. The original callgraph system does not use symbols effectively. Humans are very good at recognising different symbols but TAU callgraphs only use rectangles to represent the nodes. The small boxes (nodes) remain an obstacle in the callgraph network visualisation tool and make it harder for users to identify the root cause of the performance bottleneck, including relationships between the objects. The TAU callgraph system present objects in a linear manner, making it difficult to show the names of all the objects due to limited space within the system. One of the complicated aspects of designing a network visualisation is to identify, locate and clearly display nodes (objects) from the selected dataset. Various colours can be applied to the connection links to differentiate the nodes in the visualisation design.

Figure 3.4 *(A)* and *(B)* further demonstrated some of the boxes showing incomplete details (names) of the objects. These incomplete details make it difficult for callgraph users to both learn the full names of the objects or, consequently, the relationship between them. Some of the objects are not the same size within the system, but this may not be a problem to the callgraph users. The first view of the callgraph system further demonstrated small boxes with white dots. TAU's callgraph visualisation users need to drag these small boxes and manually expand their size in order to access the object's details. In Figure 3.2, the sizes of the small boxes (objects) were manually increased. The main purpose for increasing the size of the objects was to identify the names of the modules and routines associated with the main module, *DLPOLY*. In this exercise, it was found that callgraph visualisation users required excessive amounts of time and effort to drag/increase the size of the objects. The time it takes to drag and increase the size

of these small boxes is not inconsiderable but the detail given in these boxes remains important in order to understand the relationship between the object and its associates. In Figure 3.2, the module *DLPOLY* is selected and highlighted within red borders. The relationship between the *DLPOLY* module and other objects is shown using blue lines. The size of the red borders is too small to identify it in the callgraph system; it will be challenging to identify an active and selected object, especially when the visualised application has too many objects simulated during the run.

3.1.3 Network links and texture

In Figure 3.4 *(A)* and *(B)*, the relationship between the objects is displayed using black lines (network links). It is not easy to identify the relationship between these objects due to many network links using the same colour (black lines) to point-out different objects. The colour of the network links plays a crucial role in identifying which line is connected to which object however, if all the links are black and displayed as many then it becomes difficult to identify the connection between the objects. It is also noticeable that, in some cases, dark blue is used to represent both nodes (objects) and network links when the callgraph visualisation is operating in an exclusive mode, as shown in Figure 3.2. It takes a great deal of effort to identify the names and relationships between the objects when the colour of the network links is the same as that of the nodes.

In this performance analysis exercise (Figure 3.2), the callgraph analysis system shows network links (lines) moving over the textboxes (objects) when callgraph analysers increase the size of the objects. The names of the objects become invisible when the network lines move over the actual nodes - this is the primary reason callgraph users are not able to easily identify the names of the objects and the relationship between them. Moreover, the original callgraph system does not have functionalities to display certain groups of objects that are related to each other. The text within the objects is presented in a small italic font style, making it hard to see the details. In some cases, the callgraph presents the names (text) of the objects using white on a blue background. The structure of the callgraph report becomes even more complicated to analyse when one manually expands the size and position of the nodes in a suitable place. Various visualisation properties such as inadequate colour, shape, size, texture and overall position of the textboxes affects the functionality of the callgraph network visualisation system.

3.1.4 Interaction

The original callgraph visualisation has limited interaction with users of the system. The interaction takes place when the user clicks a node, which then results with an inclusive and exclusive time of an object simulated during the execution of the program. It do not show the name of the clicked node when displaying an inclusive and exclusive time, which makes it hard for the user to associate the target object with the execution time. When the user click a node it further highlight the target node with thin red border, which is not easy to notice when the callgraph system displayed many nodes at the same time.

The design of the original callgraph visualisation system do not have any other interactive features such as zoom, filtering data, search information, expanding and disbanding nodes. It has limited interaction, which do not help users to manoeuvre information and easily analyse the performance of parallel programs. However, too much interaction is also not desirable. The user should be able to see as much as a glance as possible (Ware, 2013) without having to resort to mouse clicks. TAU requires users to do too much clicking in order to find the name of the nodes and relationship between them.

3.2 Discussion

TAU's callgraph system display nodes that are very crowded, which makes it hard to read the details of the objects computed in the system. In addition, the callgraph system does not use sophisticated mechanisms such as zoom and/or filter to enable users to easily retrieve information about the execution of the program. Moreover, the system does not use different shapes (for example, square, diamond and round) to differentiate the execution status (normal, moderate and excessive time spent) of the objects used to perform calculations in the computational arena.

In a callgraph visualisation tool, different symbols will enable users to easily identify and locate objects that consumed normal, average and immoderate execution time. It would also be appropriate for the callgraph visualisation to apply different sizing to the nodes and network links in order to easily convey the root cause of the performance bottlenecks occurring during the execution of the program. The use of different nodes sizes and network lines will further

enable the callgraph viewer to identify group or objects that are related to each other. For example, large nodes and network lines could be associated with an excessive execution time of different parts of the program, while small ones could represent parts of the program that performed well. Furthermore, the original callgraph system does not accommodate colour-blind people due to the fact that it does not use dual channels (colour + size) to represent the relationship between objects simulated in the computational system. For instance, routine 1 and 2 can be grouped using the same colour and size to show familial ties between the objects.

Another problem with the callgraph visualisation tool is the use of similar colours to represent similar situations. As discussed in Section 3.2.1, the TAU callgraph profiling system uses dark and light blue to represent almost the same execution time of an object. This is an impractical way of displaying nodes in a network visualisation system. It would also be useful to use few colours as a means of improving the visibility of the nodes (objects) within the visualisation system, however, callgraphs utilise many colours for the similar execution status of the objects simulated during the run.

3.3 Conclusions

This chapter presented an overview of the callgraph visualisation used to analyse the performance of parallel programs. The performance of the DL_POLY molecular dynamics package and the WRF model was analysed using the callgraph system, and scaling results were adequately summarised in this chapter. Significant information about the characteristics, advantages and disadvantages of the callgraph network visualisation were also provided. Some of the colours (e.g. dark blue) and small sizes of the nodes are distractive within the original callgraph design. The original callgraphs utilise too many colours of the nodes to represents objects computed during the run. The use of dark blue colour on the nodes and network lines is also not appealing within the original callgraph visualisations.

CHAPTER 4: DESIGN METHODOLOGY

We aim to design an informative, intuitive, interactive and effective callgraph visualisation that enables users to easily obtain information on the performance of a parallel program. The design of the visualisation should enable users to easily interact with the system. An interactive design methodology was followed with prototypes that were improved from low-fidelity (paper-based) prototypes to high-fidelity (interactive) prototypes. Here we describe our user-centred design methodology. It comprises of five design stages: establishing requirements, designing alternatives, prototyping, evaluating, and final design, as shown in Figure 4.1. Visualisation design is normally an iterative refinement process, where better understanding of a development stage helps to refine other phases of the design (Munzner, 2015). The design alternatives and evaluation phases are discussed in detail to explain the process of involving users in the development of both the prototypes and the final design of the callgraph visualisation system.

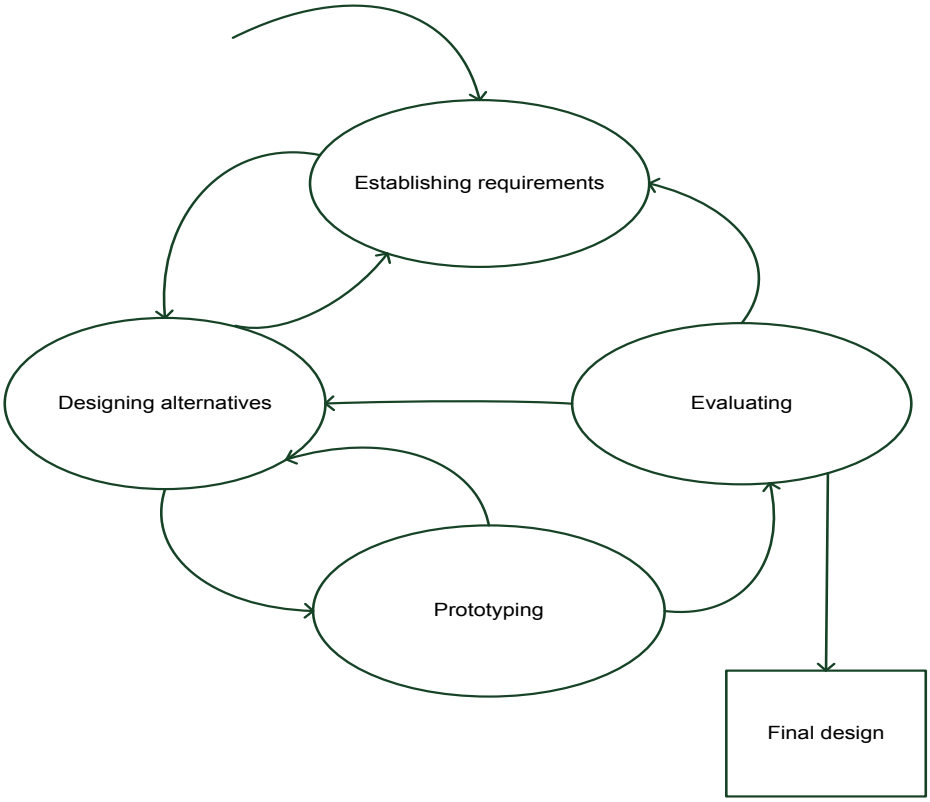

Figure 4.1: Interaction design model (Rogers, Sharp and Preece, 2011)

4.1 Design principles

Basic principles, or heuristics, of designing an interactive visualisation tool include factors that are shown in Table 4.1.

Information overload	Do not overload users with too much information.
Aid users	Help the users to be in control of the tasks.
Design for errors	Plan for errors that may occur when users performs tasks.
Visibility controls	Enable important controls (for example, buttons) to be visible in the system.
Feedback	The system must provide feedback about users' actions.
Constraints	Restrict some users' actions by enabling and disabling certain controls.
Mapping	Manage the relationship between controls and effects of the system.
Consistency	Operation of the tasks should be the same, irrespective of the controls.
Affordance	Enable mechanisms that guide users on how to perform certain tasks.
Ensure visual variables have sufficient length.	
Enable multiple levels of information.	
Consider users who are colour blind.	
Display the greatest amount of data in the least amount of space	

Table 4.1: Basic principles of designing an interactive visualisation (Abras, Maloney-Krichmar and Preece, 2004; Zuk *et al.*, 2006)

These design principles, or heuristics, help the designer to provide an interactive, usable and effective system (Preece, Rogers and Sharp, 2002; Sharp, Rogers, and Preece, 2009; Rogers, Sharp and Preece, 2011; Shneiderman *et al.*, 2016). Heuristics can also be used in the evaluation of a visualisation system. For more information on different evaluation methods,

please refer to Section 2.6. The heuristics are most suitable for evaluating low-fidelity paper-based prototypes of the visualisation tool. The design principles ensure that the designer follows basic standards for designing a good system. Adherence to the principles is not the only factor in achieving a successful interactive system. Users can help to evaluate and test the design (for example, the capabilities and functionalities) of the system.

4.2 Design approach

A user-centred method is followed here because it focuses on the needs and desires of the users by involving them on every development stage of the design (Endsley and Jones, 2004). It is important for the designer to understand the users' needs and wishes in order to develop an effective system. Moreover, user-centred design enables the designer to easily exchange information with the users during the design of the system. The exchange of information enables the designer to perceive users' goals and tasks. This user-centred technique further allows iteration of some design phases to ensure thorough testing and evaluation of the system.

We followed this development stage to design a new callgraph visualisation system, namely, requirements gathering, low-fidelity prototype, first high-fidelity prototype and evaluation, second high-fidelity prototype and evaluation and final design. The low-fidelity prototype, high-fidelity prototype and final visualisation design were developed to support solving the problem of parallel programs not performing well and fully utilise computational resources. As earlier discussed in this Section 4.2, we followed a user-centred method to design new callgraph visualisations intended to solve the problem of performance and efficiency within the parallel programs. Each of these design stages is discussed in more detail below. The outcome of the first paper prototypes helped to create high-fidelity interactive prototypes which were implemented using D3.js - JavaScript library used to develop visualisation systems. The users further evaluated the design of these interactive prototypes to assess the conceptual and physical view. The conceptual design deals with what the system does and how it do it, while the physical part is concerned with the layout of the design such as screens, buttons, icons, textboxes and menus of the design.

4.2.1 Establishing requirements

The first phase of an interaction design process (Figure 4.1) is to establish users' requirements for the system. A review of the TAU manuals, tutorials, helpdesk queries and online community discussions [8] was used to establish users' requirements and identify tasks performed by users on a regular basis. An online survey identified users who were willing to participate in the interviews or focus groups. The selected people had experience in optimising, parallelising, debugging, analysis, and/or development of parallel programs. The selection process was performed in consultation with the leaders (Directors) of the University of Tennessee, Innovative Computing Laboratory and University of Oregon's Performance Research Lab in order to get necessary clearance to perform research within the institutions. A key part of the requirements gather was to identify common visual queries, which are described in Table 3.1.

4.2.2 Low-fidelity prototype

The first paper prototypes were developed using Microsoft PowerPoint. Three different paper prototype designs were developed, as shown in Figures 5.1 *(A)*, *(B)* and *(C)*. The objective of developing these paper prototypes was to look at different ways of displaying the physical view of the callgraph visualisation. Moreover, principles of designing an interactive visualisation (as shown in Table 4.1) were followed to develop the first paper prototypes. Any settings evaluation was followed and applied to assessing the low-fidelity prototype design. As discussed in Section 2.4, Any settings is an evaluation method where the designer consults with experts, researchers and consultants to identify the most obvious usability problems of the system to be designed.

The expert evaluated the paper prototype design by performing various tasks on the blueprint of the callgraph visualisation system. During this exercise, we did not record expert's answers using video and audio recordings, however, the responses were hand written on the paper. He performed visual queries stated on Table 3.1. After evaluation, feedback was provided and considered, which consequently brought about the formation of the first high-fidelity prototype

[8] The TAU mailing list can be accessed via: http://nic.uoregon.edu/mailman/listinfo/tau-users.

(interactive design).

4.2.3 First high-fidelity prototype

The first high-fidelity prototype was designed to analyse the performance of parallel programs, as articulated in Chapter 6 about the High-fidelity prototype. This first interactive prototype design was developed to enable users to interact with the system by performing visual queries on it. In this exercise, the aim was to understand both how users interact with the systems and how the system interacts with the users. Director of the University of Tennessee's Innovative Computing Laboratory requested various computer scientists to help us evaluate the new callgraph visualisation, consequently, those chosen scientists performed practical evaluation on the first interactive prototypes of the callgraphs. Seven users evaluated the usability of the first high fidelity prototype design. The same visual queries performed by an OU expert on the low-fidelity prototypes were refined and executed on the first interactive prototype system by UT users. Table 4.2 depict the visual queries performed by the users on the first high fidelity prototype and original design of the callgraph visualisation:

Indicate objects that consumed normal, moderate and excessive execution time
Establish the status of a particular node
Identify details of the node:
Which routine name is represented by a node?
Which procedure called an object?
What amount of time is consumed by different nodes?
What is the relationship between an object and its associates?
Demonstrate how the system works:
Which nodes have children nodes?
How to expand or collapse nodes?
How to filter and display a node with its associates?
How to return back to main screen after expanding or collapsing nodes?
Which objects cause performance bottlenecks?

Table 4.2: Visual queries performed by users on the first high fidelity prototype

During the evaluation process, the same visual queries performed on the first high fidelity prototype were also executed on the original design of callgraph visualisation. Users performed these queries in a controlled laboratory (office) with no access of the public members. The visual queries (Table 4.2) helped us to identify usability problems on both first interactive prototype and original design of the callgraph visualisation. It was anticipated that the queries would provide answers to the following:

Can users perform the important visual queries quickly and effectively?
Does the design provide various ways in which to obtain performance information about the program?
Does the callgraph design meet the users' needs and expectations?
Is the design informative to the users?
Does the system design offer a distractive user interface?
Does the final design expect users to carry out multiple tasks to acquire information?

Table 4.3: Answers to the visual queries performed by users.

After performing visual queries, users were interviewed and completed questionnaire about the first high fidelity prototype and original design of the callgraphs. The questionnaire focused on how easy or hard it was to use both the original and first high-fidelity prototype design of the callgraph visualisation system. In the questionnaires (survey), users indicated job titles and number of years (experience) in the computational science field.

During completion of the questionnaires, each user was interviewed to understand more about the physical and conceptual design of both the original and first interactive prototype design of the callgraph visualisation system. Users performed tasks on both the original and first high-fidelity prototype design of the callgraph system for comparisons. Afterwards, interviews were conducted with the users to record feedback about the systems. During the interview, the users were asked the following questions (Table 4.4):

How helpful is the visualisation designs in terms of optimising programs?
Which design(s) is more helpful?
Do you like the physical layout of the screens?
What will you like to see in the new callgraph design?
How easy or difficult was it to learn the system?
Which visual queries were easier to do than others?
Which questions were hard or easy to answer with the design?
What are the most important queries that users perform on callgraphs?

Table 4.4: Users questionnaires about the original and first high fidelity prototype design

We conducted the interview by asking questions to the users. For interview purpose, all the questions were given to each user before the discussion can start. The user's answer to each question was written on the question paper and responses were also recorded for future analysis of the interview. Every user was interviewed on a one-to-one basis immediately after performing visual queries on the callgraph visualisations.

4.2.3.1 Evaluation

The method of observing users when performing tasks was precisely applied to evaluate both the original and the first interactive prototype system. The controlled settings evaluation was practised to evaluate the first high-fidelity (interactive) design and original callgraphs in a laboratory where users were tasked to perform visual queries on the systems. Users were recorded using video footage when performing tasks on both the original and first high-fidelity prototype design. Consequently, audio was recorded, which was analysed after the evaluation in order to understand the interaction between the users and callgraph visualisations. Furthermore, we took note of the users' reactions when using the original and the first high-fidelity prototype system. As discussed in Section 4.2.3, users were interviewed to get feedback about the usability of the original and the first high-fidelity prototype. The following data - visual queries, interviews, notes, video and audio recordings were assessed using quantitative and qualitative analysis techniques to ensure that the system met users' requirements. All the

interviews, notes, videos, audio recordings and visual queries performed by the users on the first and second high fidelity prototypes can be found our website [9] by clicking on the "Research" tab followed by "Visualization of parallel programs". The results were also interpreted using the method of converting data into percentages and averages, as discussed in Chapter 6. Graphical presentation of the data collected was also analysed and interpreted accordingly, as discussed in Chapter 6. Nonetheless, analysis, interpretation and presentation of the observation/interviews data helped to determine the usability of the first high-fidelity prototype design. The process of re-designing and evaluating the prototypes was continuously repeated when the system did not meet users' requirements and needs, until the final visualisation managed to fulfil both.

4.2.4 Second high-fidelity prototype

The users' feedback about the original and the first interactive design helped us to develop the second high-fidelity prototype of the callgraph visualisation system. During the evaluation of the first interactive design, users recommended improvements and extra features to be added on the new callgraph visualisations. Moreover, users recommended the second high-fidelity prototype design to be created by following the first interactive design (tree design) and tested using large datasets (performance metrics) generated during the simulation of the parallel program. The second interactive prototype design was developed by following the design of the first high-fidelity design, and extra visualisation features were added as recommended by the users.

The UT users evaluated both the original and second high-fidelity prototype designs in a controlled laboratory. Four users evaluated both original and second high fidelity prototype design, which resulted with the final design of the callgraph visualisation. Users performed visual queries by means of the original and second high-fidelity prototypes used to simulate large performance data (performance metrics) produced during the simulation of the parallel program. The same visual queries performed on the first interactive prototype design were slightly refined to be clearer and performed on both the original and second high-fidelity

[9] Users data can be found on the following website: https://people.cs.uct.ac.za/~mmabakane [19 January 2019].

prototype design of the callgraph visualisation system. Table 4.5 shows queries performed by users on the second high fidelity prototype design:

Which nodes use excessive time?
What is the execution status of MODULE_WRF_TOP::WRF_DFI?
What is the full name of MODULE_WRF_TOP::WRF_RUN?
Which node (object) calls MODULE_WRF_TOP::WRF_RUN?
What is the amount of time consumed by MODULE_WRF_TOP_INIT?
What is the relationship between MODULE_WRF_TOP::INIT & its associates?
Which node, between MODULE_INTEGRATE::INTEGRATE and WRF_DEBUG, has children nodes?
If any of the below-mentioned nodes (MODULE_INTEGRATE::INTEGRATE and WRF_DEBUG) have children nodes, how many children nodes are there?
How to expand or collapse the nodes?
How to search and display a node with its associates?
How to return back to the main screen after expanding or collapsing the nodes?
What is the name of the object (node) that causes performance bottlenecks?

Table 4.5: Visual queries performed by users on the second high fidelity prototype

How helpful is the visualisation design in terms of optimising the programs?
Which visual queries were easy to perform?
Which visual queries were difficult to perform?
Which design is more helpful, and why?
Can you suggest any improvements to the design?

Table 4.6: Users questionnaires about the original and second high fidelity prototype designs

After performing the visual queries, interviews were conducted with the users about the experience of using both original and second high-fidelity prototype designs. Users were asked to answer the questionnaires shown in Table 4.6.

4.2.4.1 Evaluation

As discussed in Section 4.2.4, users evaluated both the original and second interactive designs (second high-fidelity prototype design). Video was used to capture interaction between the users and original/second interactive design of the callgraph visualisation system. Consequently, video and audio recordings of the users were captured. In addition, notes were taken to analyse how users employ both original and second interactive prototype designs of the callgraph profiling system. Data such as interviews, notes, audios, videos and visual queries performed by users were analysed using the quantitative and qualitative method of evaluating systems. After analysing data, the final design was developed to satisfy users' needs and requirements.

4.2.5 Final design

The final visualisation design was developed using D3.js JavaScript library. The visualisation methods applied on the second high-fidelity prototype design were applied on the final visualisation design, which displayed information using two different visualisation methods - filtering and search. The final design (filtering and search mode) of the callgraph visualisation was developed for users to identify performance bottlenecks incurred during the execution of the parallel program. In particular, filtering enabled the user to filter a certain amount of information, while the search design allowed the user to search a particular query within the visualisation, as indicated in Chapter 7. These designs were used to analyse both small and large performance data produced by parallel applications.

4.3 Conclusion

This chapter discussed the interaction design models used to design and evaluate the final design of the callgraph visualisation system. It demonstrated how users evaluate original

prototypes and final designs of the callgraph visualisation system. Different development stages - establishing requirements, designing alternatives, prototyping, evaluation, and final design were followed together with the users to develop an advance callgraph profiling system. The chapter further highlighted the process of how users became the co-designers during the development of the new callgraph visualisation. Most notably, users evaluated the usability of the original, prototypes and final callgraph visualisation designs.

CHAPTER 5: LOW-FIDELITY PROTOTYPE

This chapter discusses our goals we initially presented: design stages of the low fidelity prototype (paper prototype) of the callgraph visualisation. It further describes different types of low-fidelity prototype (paper-based) visualisations, namely star and tree designs, used to depict the performance of parallel programs on a computational system. In addition, different features of the designs are presented, using visual properties such as colour, size and shape of nodes or network links within the paper-based prototype designs. The advantages and drawbacks of these non-interactive (low-fidelity) prototypes designs are also examined in this chapter.

5.1 Design goals

The aim of the low fidelity prototype was to learn different ways of displaying the physical view (e.g. nodes and network lines) of the callgraph visualisation. It was also important to find a suitable visualisation method of displaying the relationship between the nodes within the callgraphs. Most importantly, we designed a low fidelity prototype to assess various mechanisms of presenting different types of nodes. The low fidelity prototype was also designed to analyse the technique of presenting the relationship between the nodes without overcrowding information on the screen as it is the case with the original callgraph visualisation system.

5.2 Design description

The performance analysis of the following parallel programs: WRF, DL_POLY_2.18, and 2.20 was performed using callgraphs, as discussed in Chapter 3. The primary purpose of this experiment was to analyse the capabilities of the callgraphs in order to identify the most common visual queries performed by users. It was found that in most cases the callgraph user needed to perform the visual queries (as shown in Table 3.1) to understand the performance of the parallel program. It is challenging to perform these queries using the current (original) design of the callgraph visualisation system, as demonstrated in Sections 3.1. To this end, two paper-based prototypes with polished callgraph network system software designs (star and tree)

were designed, as shown in Figure 5.1. The star design is a visualisation method that shows the relationship between different objects, which spread to different directions of the system (Lima, 2010; Heer and Boyd, 2005; Christakis and Fowler, 2007). In this star design, different nodes can connect to each other and be grouped according to their associates. The tree design is also one of the visualisation methods used to analyse and display the relationship between different nodes in a particular scenario. In a computational arena, tree design can be used to demonstrate the affinity between different objects (such as modules, routines and functions), which are utilised to perform computational calculations.

Most interestingly, the tree design shows the relationship of objects spread like a tree across the entire system (Lima, 2010; Newman, 2018). Various designs mechanisms, such as the method of grouping objects and using colour to identify relationship between the objects, were adopted in the design of the prototype systems. As shown in Figure 2.8, the grouping of objects provides a clear indication of the relationship between the nodes. Figure 2.8 further demonstrated an appealing and user-friendly system that used colour to emphasise the relationship between objects of the same group.

Both star and tree prototype designs adopted colour to identify the relationship between objects, while star design also has objects grouped together. Colour, size, space and texture are the most influencing factors when selecting this method of analysing data. In particular, the star design was created primarily due to its ability to easily identify groups of objects that are closely related to one another and is able to display objects within large circles of different colours. The tree design was developed because of its ability to spread objects across the system and clearly display many objects within the limited space of the system.

The tree design was also selected due to the fact that it can quickly identify many groups of objects that are related to one another, using different colours over the borders and network links of the nodes. Shneiderman *et al* (2016) principles of providing overview of the information and then followed by details-on-demand was followed to design low fidelity prototypes, namely, star (Figure 5.1 *A*) and tree design (Figure 5.1 *B* and *C*). The star design display overview of the information and tree design provides more details about different nodes within the low fidelity prototype.

5.2.1 Star design

Figure 5.1 *(A)* illustrates the star design of the callgraph prototype system using small circle, square, and diamond shapes connected to the large square box in the centre of the system. The first view of the callgraph profiling system shows only objects, and their associates, that consumed moderate and excessive execution time. The reason for displaying only those objects that consumed moderate and excessive execution time is due to the limited space caused by the large circles. In particular, the star design adopted the style of grouping objects in different boxes (Heer and Boyd, 2005; Adar, 2005) and used colour to emphasise the relationship between objects (Christakis and Fowler, 2007). Figure 5.1 *(A)* indicates the connection between nodes (square, round and small diamond shaped boxes) encapsulated within six large circles; these circles have different border colours (black, dark purple, yellow, sky blue, light pink and grey). It enables users to easily identify groups of objects that are closely related to each other. Moreover, these six circles use the same colour as the borders in order to connect to the main computational task (MAIN MODULE).

The white background enables objects (small circles), network links, and large circles to be more visible within the system. Because of this one can quickly identify the names of the objects and the relationship between each another. The size, shape and colour play a further very important role in differentiating the execution status and relationship between the objects visualised using the callgraph system. The white nodes represent aspects of the program that consumed normal execution time; orange indicates objects that consumed average execution time, and red nodes demonstrate objects that consumed excessive time during execution of the program. The use of dual channels (colour, shape and size) was further applied on the nodes to ensure that all the callgraph users (including colour-blind people) could differentiate the objects within the system.

In particular, white objects are shaped using round circles while orange nodes have a diamond shape and red nodes are reflected using square boxes. The size of the white, orange and red nodes is also different. The red nodes are larger than orange nodes, which are bigger than white nodes. The red and orange nodes were intentionally made larger because of their critical negative impact on the overall performance of the program. All curved connection links are used to lead the callgraph visualisation user to the red nodes. Elbow connectors position orange

nodes, while straight network lines connect to the white objects within the visualisation. Different colours (black, dark purple, yellow, sky blue, light pink and grey) are used on borders of the objects (small boxes) to showcase the relationship between an object and its associates. The objects that have the same border colour are connected to each other within the large circles.

Figure 5.1 *(A)* shows that objects that have black borders connect to each other via black connection lines, while objects surrounded by dark purple connect to each other via dark purple network links. At the same time, yellow nodes (objects) are linked to each other using yellow connection lines to showcase their kinship to one another. The mechanism of using different border colours enables the callgraph user to rapidly identify the relationship between groups of objects that are associated with each other. It is evident that different border colours demonstrate an ability to differentiate, locate and identify the relationship between many objects within a network visualisation.

The star prototype design (Figure 5.1 *A*) also enables the user to easily identify an active object (one that has been clicked) using a large brown border to highlight an operating object. In Figure 5.1 *(A)*, routine 4's border is brown, which reflects an active object. When a callgraph user clicks an object, the system magnifies the object into the centre of the screen in order to display the full name of that particular object. Once clicked, the object displays a brown border. The purpose of magnifying an object is to ensure that the user can easily obtain the full name of each object without dragging and increasing its size.

It is difficult to display all long object names within a single screen. Therefore, white objects can only show up to thirteen characters (including spaces) if not clicked or active. Orange nodes can display up to a maximum of fifteen characters while red nodes show up to nineteen characters within the system. The associates of an active object also change their border colour and connection links to a bright green colour. The TAU callgraph prototype design (Figure 5.1 *A*) utilises an ordinary black text to display the name of each object within the network system. The black text within the small circles (objects) is very clear. It is evident that the star prototype design clearly demonstrates the execution status and relationship between the objects using size, shape and colour.

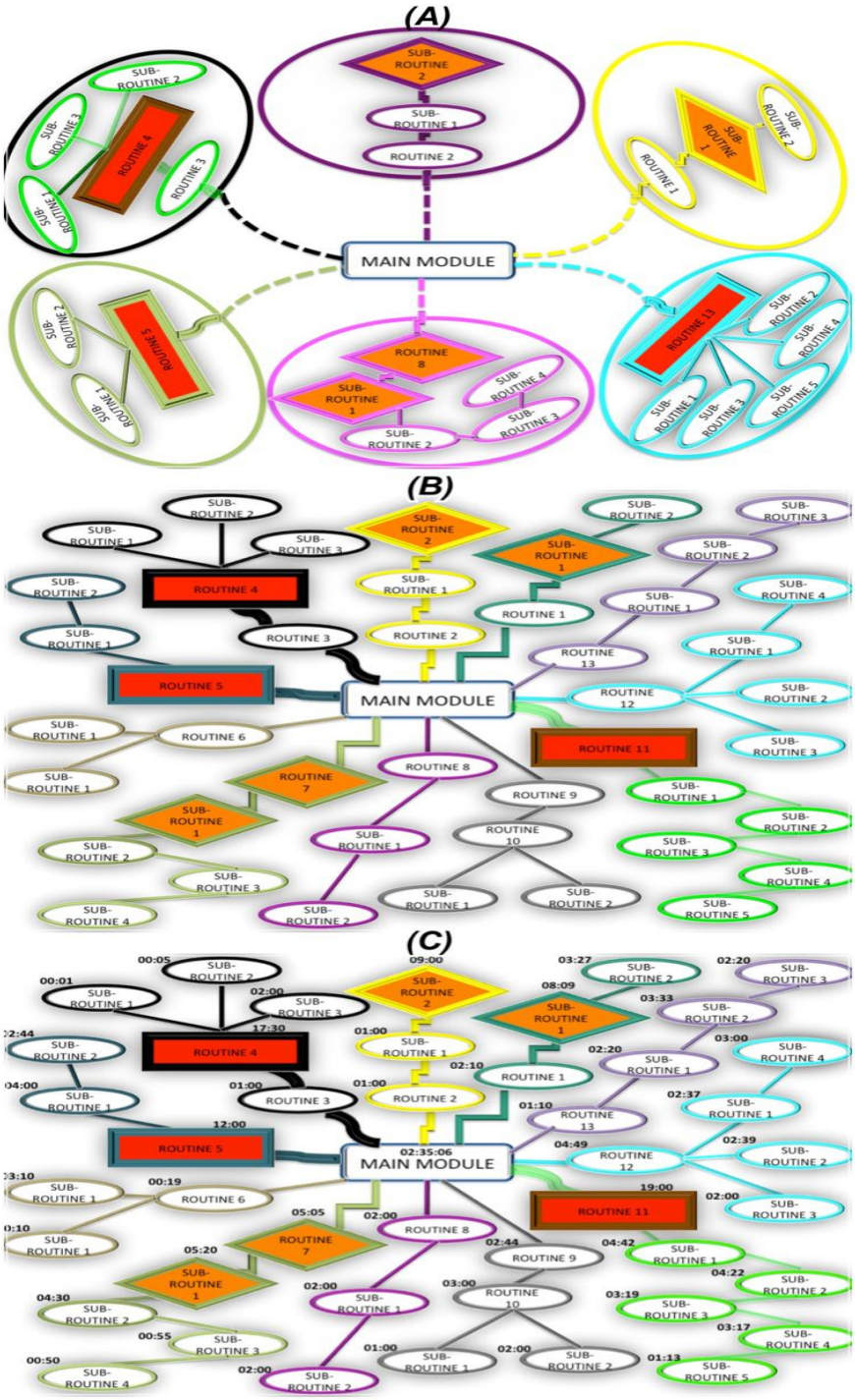

Figure 5.1: Star (A), Tree (B) and Time-level (C) prototype design

Most importantly, red and orange nodes enable the callgraph user to easily identify objects that have an undesirable impact on the overall performance of the program. It is advisable for callgraph users to focus on red and orange objects in order to successfully increase the scaling

of the program. The advantages of this star design are as follows: objects are grouped according to their associates; it is easy to see the relationship between objects; and the visualisation user can easily identify objects that have either a negative or positive impact on the performance of the program. Moreover, the system has appealing colours that clearly show differentiation of the objects, ease of identifying the names of objects using auto-zoom capabilities, and an ability to clearly display acceptable numbers of objects simultaneously. The disadvantage of the star design is that it displays a limited number of objects (only moderate and excessive ones) in the first view of the callgraph visualisation tool. It would be difficult to see the details of each object if the program simulates thousands of objects.

5.2.2 Tree design

Different types of tree designs such as: ramification, circular ties, centralised ring, centralised burst, organic rhizome, radial implosion, radial convergence, segmented radial convergence, and elliptical implosion design (Lima, 2010; Smith, 2009; Newman, 2018; De Nooy, Mrvar, Batagelj, 2018) were studied to construct a meaningful callgraph network visualisation system. The ramification design was selected and applied on the tree prototype design of the callgraph system, as shown in Figure 5.1 *(B)*. The reason for selecting this design was due to its capability to spread out many objects from the centre, top-left, top-right, down-left and down-right of the visualisation design. In this case, the visualisation was able to handle many objects.

Figure 5.1 *(B)* demonstrates the tree design of the callgraph prototype used to analyse the performance of different objects, such as modules, routines and subroutines. Yellow, black, light pink and grey nodes were used to indicate the relationship between different objects related to each other. For example, yellow networks links are used to show the relationship between routine 2, sub-routine 1 and sub-routine 2. The borders of these three objects are yellow in order to showcase the kinship between the objects. In this case, one is able to quickly and easily understand the relationship between the groups of objects associated with one another. Each object is surrounded by a 3D shadow, which makes nodes more visible within the tree prototype design. It is evident that the tree design can display many objects and their details, as shown in Figure 5.1 *(B)*. In this tree prototype design, when the user clicks an object it will magnify it to provide the details of that particular node. Thereafter, the clicked object will automatically return to its position and change border colour to brown, as depicted in

Figure 5.1 *(A)* - routine 11. The network lines and borders of the objects that are related to an active one will also change to a bright green colour. Three colours have been used - white, orange and red - to establish the execution time consumed by each object. In particular, objects that contain white backgrounds consumed an acceptable execution time while orange ones illustrate objects that consumed a moderate amount of the execution time. Red objects show aspects of the program that consumed excessive execution time during execution of the program. Different node shapes have been used to establish the execution status of the objects. For instance, circular nodes represent objects that consumed normal execution time while diamond boxes illustrate aspects of the program that consumed moderate execution time during program simulation; square nodes are used to demonstrate objects that consumed excessive execution time.

Moreover, the tree prototype design (Figure 5.1 *B*) uses size to depict the execution status of objects. In particular, the square nodes that consumed excessive execution time are bigger than diamond objects, which consumed moderate execution time. At the same time, diamond nodes are larger in size than the circular objects. Black text is used to display the name of each object. The main advantage of this tree design (Figure 5.1 *B*) is that it uses different colours to show the relationship between nodes. Different colours enable the user to easily identify groups of objects that are associated with one another. The tree design also provides an opportunity to display many objects simultaneously, in a limited space.

Each relationship between the objects is clearly shown using network connection links and border colours. Different node colours, sizes and shapes allow callgraph users to easily locate and identify objects that have consumed normal, average and excessive execution time. The names of the objects are clearly displayed using black text. Tree visualisation designs can only display a maximum of thirteen characters within a single circular node, while each diamond node handles up to fifteen characters if not clicked; red objects have the capability to present up to nineteen characters if not active. Nevertheless, the system automatically provides the full detail of each object when the user clicks on it. To this end, another version of tree design - the time-level prototype system - was designed to display the execution time of each object, as shown in Figure 5.1 *(C)*. The time-level design represents the second view of the callgraph network visualisation system. Basically, the structure of the time-level design is the same as the tree design (Figure 5.1 *B*) except that the time-level prototype precisely displays the execution

time of the node within the visualisation. Figure 5.1 *(C)* demonstrates routine 11 in an active mode (clicked), which consumed nineteen minutes of the overall execution time taken to simulate the program in the system. The central node, MAIN MODULE, contains the total program execution time while other, surrounding nodes also show their execution time. The main advantage of this time-level design is to provide the execution time of all the nodes, which enables the callgraph user to easily identify the execution time of each object.

5.3 Evaluation

The paper prototype designs (star, tree and time-level) were evaluated by an expert (developer) in parallel computing from UO's Department of Computer and Information Science as discussed in Section 1.3. An expert performed different visual queries using low fidelity prototypes (paper prototype designs), as discussed in Section 4.2.2. Visual queries performed by an expert are listed in Table 3.1. Consequently, the expert found the use of different shapes (circle, square, and diamond), sizes (small, average and large) and colours (white, orange and red) of the nodes to be useful for establishing the execution status of the objects within the parallel program.

The assessor also found the use of different border sizes to be helpful on differentiating the execution status of the nodes. An expert was adjudicated that both star and tree visualisation method are good to present the relationship between the nodes. It was however found that the use of different colours on the nodes and network lines caused distraction within the paper-prototype designs of the callgraph visualisation system. An expert showed dissatisfaction about the short names of the nodes, however, black texture was found to be clearer within the nodes.

Different types of network links, namely, straight, elbow and curved lines were also found to be none beneficial within the callgraph visualisation. Based on this analysis, different shapes, sizes, colours and borders of the nodes will be adopted on the high fidelity prototype design (interactive prototype design). Both star and tree visualisation design will also be applied and tested on the high fidelity prototype. The technique of adequately displaying names of the nodes irrespective of their length will be investigated and applied accordingly on an interactive design. The low fidelity prototype (Figure 5.1) does not have capacity to display long names because it has none-interactive features - which cannot hide/show information depending on the

user's action on the system. It has provided a foundation to solve the problems of identifying performance bottlenecks using callgraph visualisations. In fact, low fidelity prototype is effective when one explore different ideas because the design does not need excessive effort and time to be created. It also allows the designer to change the design on the fly without using excessive effort to generate the prototype. This enables one to explore as many ideas as possible. The challenge is to evaluate low fidelity prototype with experts and/or users. In our experience, it was found that most of the experts do not show more interest in evaluating low fidelity prototype because it does not immediately solve their problems within the system. Experts and users' needs immediate solution to the problems of the systems. One expert evaluated the low fidelity prototype but this will not be sufficient for subsequent designs (high fidelity prototype). Nonetheless, evaluation of the low fidelity prototype was successful and provided a platform for high fidelity prototype - which will be discussed in the next Chapter 6.

5.4 Conclusion

The chapter discussed paper-based prototype designs namely, star, tree and time-level prototypes designed using PowerPoint, which present paper-based prototypes with polished software design. Most importantly, the prototypes demonstrated different ways of using colour, size, shape and texture to design an effective visualisation system. In fact, these different visualisation properties were applied on the prototype designs in order to develop an effective callgraph network visualisation system. The method of evaluating the system with experts was successfully used to assess the physical view of the low fidelity prototype designs, which was evaluated by an expert.

CHAPTER 6: HIGH-FIDELITY PROTOTYPE

The chapter presents the first and second high-fidelity (interactive) prototype designs of the callgraph visualisation developed using the D3.js JavaScript library. Firstly, the chapter demonstrates the capabilities of the first interactive prototype designs (expandable star, collapse star, expandable tree and non-expandable tree) used to identify performance bottlenecks within a parallel program. The chapter further discusses how users evaluated the first high-fidelity prototypes and original designs of the callgraph visualisation system; it also describes the functionalities of the second interactive prototype design and how users evaluated the usability of this design versus the original callgraph design. Most notably, advantages and disadvantages of each visualisation design are discussed and summarised.

6.1 Interactive design

As previously stated in Section 1.3, the aim of this study is to design an effective callgraph visualisation with which users can easily interact. – Using paper-based low fidelity prototypes (star and tree visualisation designs), we demonstrated different ways of using visual properties such as colour, text, size and shape to depict the skeleton picture of the callgraph visualisation system. For more information about low-fidelity paper prototypes, please refer to Sections 5.1 (*A*), (*B*) and (*C*). Fundamentally, paper-based prototypes were used to explore various mechanisms of designing the physical layout of the nodes and network links, however, they do not demonstrate what the system does or how it does it.

D3.js is a set of functions used to develop the first high-fidelity prototype designs (star and tree interactive) as discussed in Section 4.2. The star design further consists of two different visualisations - expandable and collapsible callgraph visualisation designs. The tree design also has two different interactive visualisations: i) expandable and ii) non-expandable callgraph designs. The reason for developing different star and tree interactive designs is to compare which design is the most suitable one for visualising callgraphs. Both star and tree interactive visualisation designs were used to visualise the performance of NAS Parallel Benchmark executed on two, four, eight and sixteen processors. More information about the designs will be discussed in the next two sections.

6.2 Star interactive design

The star interactive visualisation design has two main prototypes, expandable and collapsible callgraph visualisations, as explained in Section 6.1. The expandable star visualisation design is discussed below.

6.2.1 Expandable star design

The expandable star design is an interactive prototype visualisation, which spreads hidden children nodes when a user clicks on a particular node that has associates. Figure 6.1 represents an expandable interactive prototype system used to visualise performance data simulated on two computer processors. It shows a relationship between objects such as functions, routines and subroutines displayed on the first view of a callgraph prototype visualisation. In particular, expandable star visualisation designs present a relationship between objects using different types of nodes such as circle, oval and square nodes connected to each other. In all interactive designs, white circles represent objects that consumed normal execution time; orange oval nodes shows parts of the program that utilised moderate computing time.

Lastly, red square nodes represent objects that have consumed excessive execution time during execution of the program. In the first interactive prototype design, the method used to calculate the status of the nodes is as follows: normal execution time is when an object consumed less than, or equal to five minutes; moderate execution time is when an object consumed between six and eleven minutes; excessive execution time is when part of the program consumed more than twelve minutes of the overall simulation time. Most notably, square nodes are bigger than oval nodes, while oval nodes are larger than circles. All the nodes have a steel blue border, which complements the different colours (white, orange and red) of the nodes. A node that has children nodes will have a large border, which makes the node look like a button. When the user points the mouse at the node, the system will display a textbox containing the full name and execution time of the object. Figure 6.1 shows different types of nodes such as circles, an oval and a square box. This expandable star visualisation enables the user to obtain the full details of an object by pointing the mouse at the node.

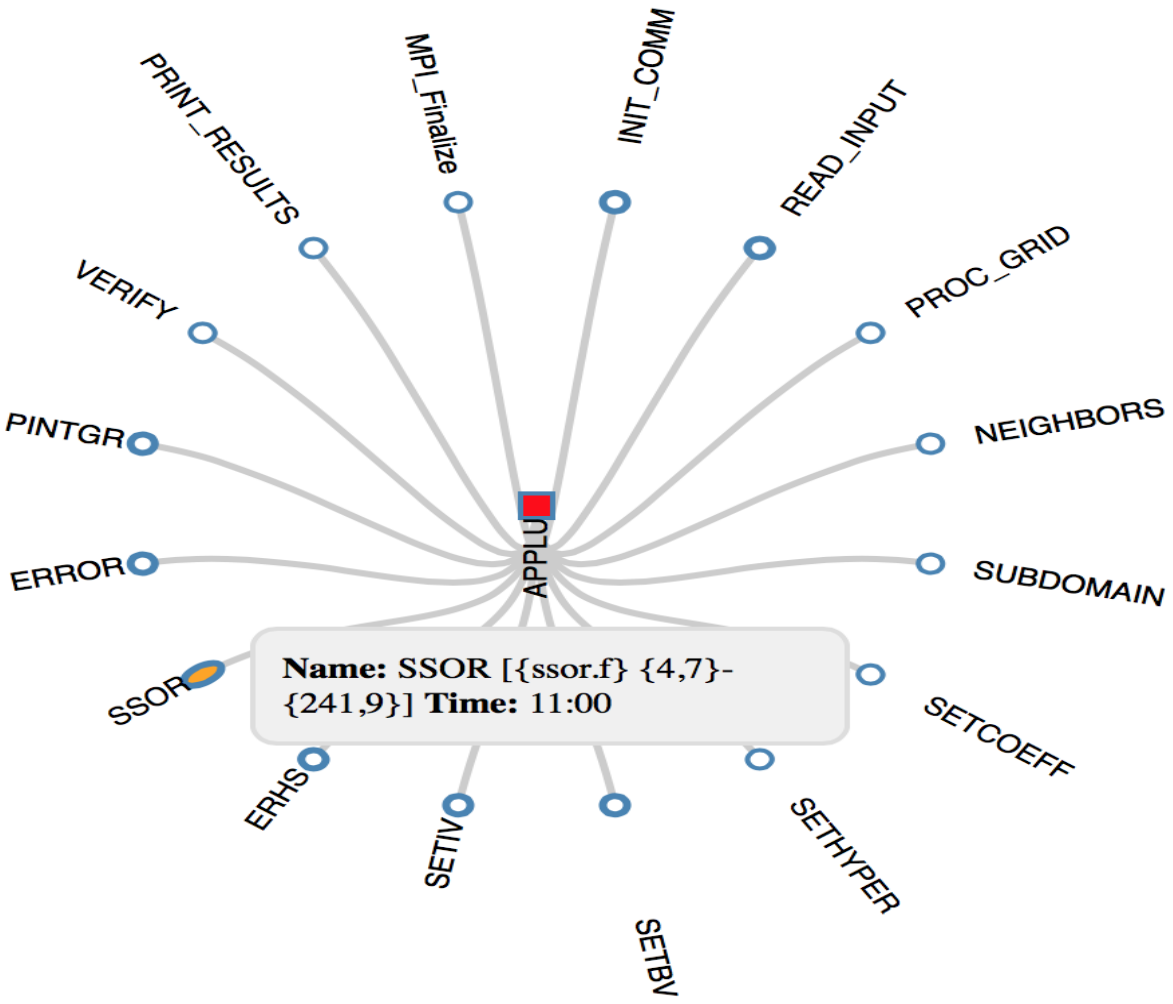

Figure 6.1: Expandable star design on two processors

In Figure 6.1, the user has pointed at the SSOR node and so the system displayed a textbox with full name and execution time of the SSOR object, as indicated in Figure 6.1. The textbox disappears when the user moves the mouse off the node. When the user clicks on a node that has a large border, the system will automatically expand to display the clicked/active node with its associates, as shown in Figure 6.2. It illustrates an expandable visualisation design after the user clicked a node to expand children nodes. The user clicked an orange oval node, SSOR, which then expanded and displayed its children nodes - RHS, L2NORM, MPI_Barrier, TIMER_CLEAR, TIMER_START, JACLD, BLTS, JACU, BUTS, TIMER_STOP, TIMER_READ, and MPI_allreduce.

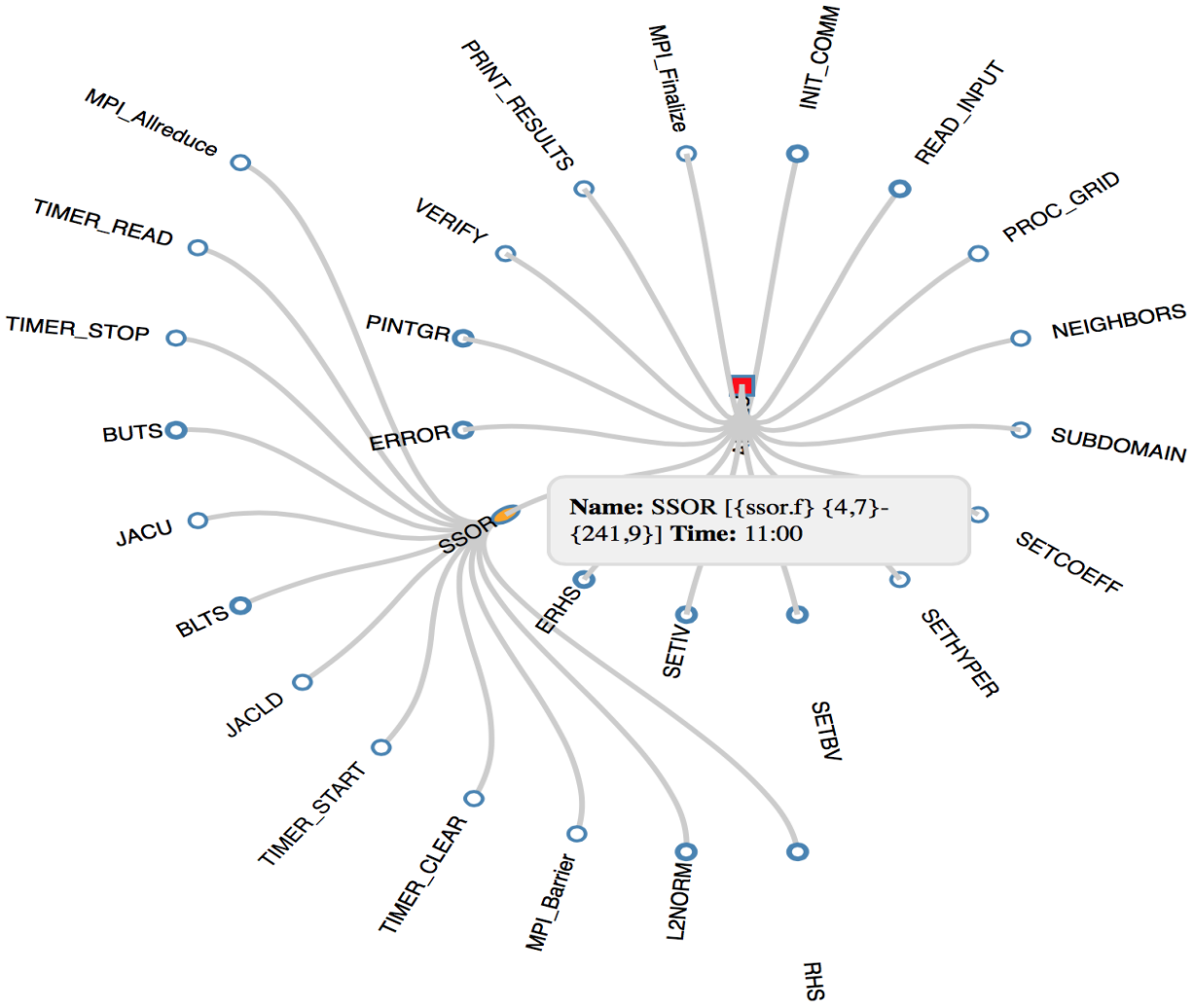

Figure 6.2: Expandable star design with active nodes on processors

In this visualisation design (Figure 6.2), when the user clicks a node with a large border, such as SSOR, the system expands slightly and spreads the children nodes like a star. The children nodes are then connected to an active node using grey network links. The colour (grey) of the network lines correlates very well with the steel blue borders of the nodes. In fact, the grey network links accentuate both steel blue borders and text colour of the nodes. The black text colour is used to display the names of the nodes within the expandable interactive star visualisation. Actually, the name and execution time of the node within the textbox is also presented using black. Both name and time are highlighted in a bold font style to ensure that the user does not confuse the details of the node, as shown in Figure 6.2. This expandable star visualisation design may be overwhelming, even if the user wishes to display excessive detail of the nodes simultaneously. Figure 6.3 shows an expandable star visualisation that contains

many nodes. The expandable star design utilises the D3.js intelligible visualisation to manage and handle nodes displayed on the screen. The system visualises parts of the NAS Parallel Benchmark program on four processers. The expandable star visualisation further shows more than forty-five different types of nodes lucidly displayed at the same time, as illustrated in Figure 6.3.

Figure 6.3: Expandable star design on four processors

D3.js is used to normalise the nodes when the user expands children nodes. When the user expands more nodes, the system minimises the size of all the nodes including names displayed on the screen. The expandable star visualisation designs (Figure 6.1, 6.2 and 6.3) have different sized nodes and names because the systems carry a different number of nodes. For example, the

visualisation design that displays seven nodes will have a larger node size than a visualisation design that has fourteen nodes. The expandable star system further spins the network lines to position the nodes accordingly. When the user clicks on a node to expand children nodes, the system spins the network links of all the nodes in order to accommodate the newly-expanded set of children nodes. Figures 6.2 and 6.3 show textboxes that contain the full name and execution time of the SSOR nodes on both pictures. It also illustrates that the status of the SSOR nodes on Figures 6.2 and 6.3 is different. In Figure 6.2, the SSOR routine is on a moderate status (orange, oval-shaped node), while in Figure 6.3 it is on an exorbitant status (red, square node). The status of the nodes can change when one runs an application on a different number of processors, as it is evident in both Figures 6.2 and 6.3.

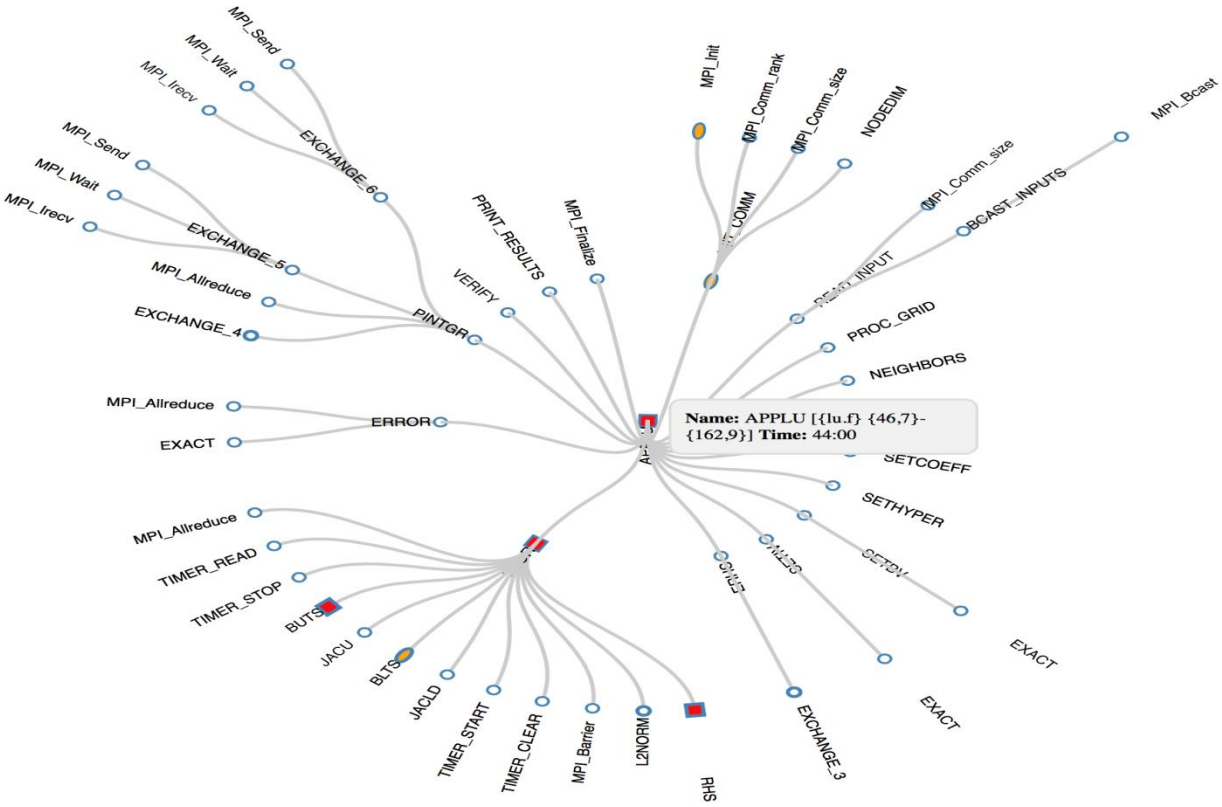

Figure 6.4: Expandable star design on eight processors

Figure 6.4 shows the expandable star visualisation design with the nodes of different statuses. It is evident that the expandable star callgraph visualisation design is able to identify the performance bottlenecks of different parts of the program incurred during the simulation of the parallel program. The expandable interactive prototype system (Figure 6.4) demonstrates the use of colour, shape and size to establish the status (normal, average and excessive) of the node. As previously articulated, white circle nodes represent normal execution, while orange

oval nodes demonstrate moderate execution of the object (module, routine, subroutine and function) simulated during the run. Most importantly, red square nodes show parts of the program that consumed excessive execution time. The border size of the node is also used to emphasise the status, such as non-expandable or expandable nodes that have children nodes, as shown in Figure 6.4. In particular, nodes with small borders show non-expandable nodes, while nodes with large borders demonstrate expandable nodes. For example, the APPLU module is non-expandable because its node has a small border, while BUTS and RHS routines are expandable because the nodes have large borders.

This applies to moderate nodes such as the BLTS and MPI_Init function of the program. The BLTS is an expandable node due to its large border colour, which makes the node appear like a button; the MPI_Init node is not considered an expandable, hence its border size is small. The expandable star design is able to clearly show red square nodes that consume excessive simulation time, as shown in the below Figure 6.5. The user needs to pay more attention to red square nodes that represent objects that are vulnerable to the performance of the program simulated on a computational system. It is also noted that square nodes are the most essential parts of the program that needs to be optimised in order to increase the performance of the parallel programs on a computational system. Figure 6.5 clearly shows eight red square nodes, representing different parts of the program that consumed excessive execution time. The expandable star interactive design (Figure 6.5) visualised the performance of the NAS Parallel Benchmark program simulated on sixteen processors and identified performance bottlenecks incurred during the run. It established the following objects: EXCHANGE_1, BUTS, BLTS, RHS, SSOR, APPLU, INIT_COMM, and the MPI_Init function as the objects that decelerate the performance of the program during the run. This expandable star prototype visualisation also enables the user to obtain the exact execution time of each object by moving the mouse over the node. The textbox in the design (Figure 6.5) contains the following information regarding an object: ‘Name’ represents the full name of the object, and ‘Time’ indicates the execution time spent processing an object. It is also noticeable that the expandable star design displayed white circles, which represent objects that have no negative impact on the overall performance of the NAS Parallel Benchmark program executed on sixteen processors of the computational system. The main advantage of the expandable star design is that it displays many different types of nodes at the same time even though network lines sometimes slightly obscure the names of the objects.

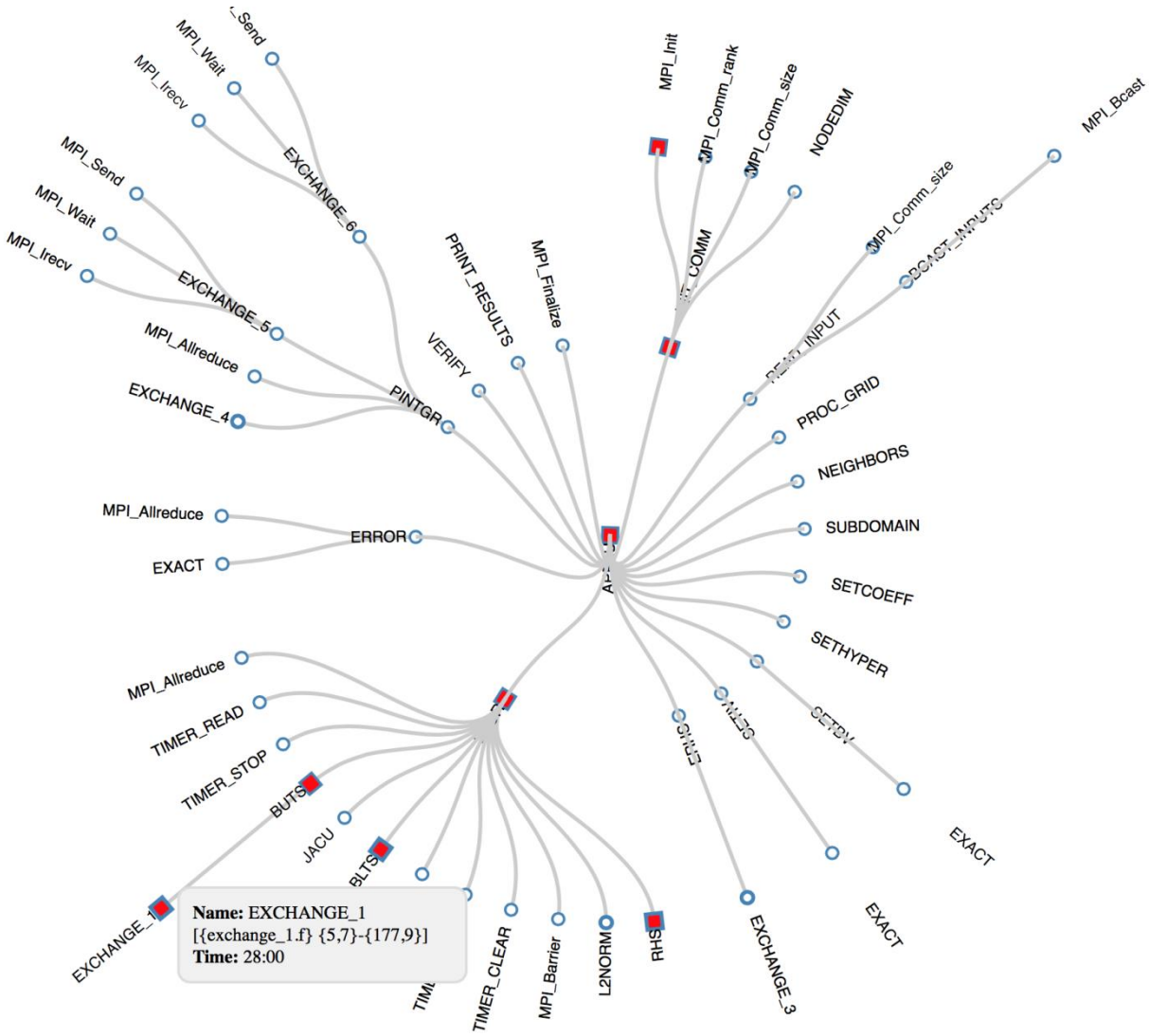

Figure 6.5: Expandable star design on sixteen processors

The full name of each object can be clearly displayed when one hovers the mouse over the node. It is also notable that the design uses different colours, sizes and shapes to establish the status (normal, moderate and excessive execution time) of the nodes. The grey network links showcase the relationship between the nodes. Moreover, spinning the network links also ensures that the nodes are positioned correctly and accordingly. This expandable star design can only display a limited number of nodes within the diameter. It is also not able to identify a specific group of nodes that are associated with each other. In this case, the collapsible design is explored and discussed in the next section.

6.2.2 Collapsible star design

The collapsible star design is an interactive prototype design that filters a specific group of nodes when the user clicks on a particular node. As stated in Section 6.1, it has been used to visualise the performance of the NAS Parallel Benchmark program executed on a different number of processing capabilities starting from two, four, eight and sixteen processors. Figure 6.6 presents the first view of the collapsible star prototype design on two processors. It uses different type of nodes: white circles, an orange oval and a red square node connected to each other via grey network links.

The collapsible star design shows different execution statuses (normal, moderate and excessive) of the node. Different colours, sizes and shapes are used to determine the status of the nodes as discussed in Section 6.1.1. The first view (Figure 6.6) displays all the nodes associated with the names of the object used to run the NAS parallel program. In this case, the size of all the node borders remains small because there are no hidden children nodes. Black text is used to display the names of the nodes. In addition, the textbox contains the full name and execution time of an object executed during the run.

Figure 6.6 (collapsible star design) shows two nodes - SSOR and APPLU - as objects that have a negative impact on the overall performance of the program. The SSOR consumed an average execution time of eleven minutes, while the APPLU module consumed fourteen minutes of the simulation. The grey network links shows the relationship between the objects even though some names of the nodes move over each other, which consequently causes some of these to be invisible. Nevertheless, one can point the mouse at a node in order to obtain the full name of the object executed during the run. The collapsible star prototype design (Figure 6.6) displays the nodes within a diameter, which is the same method used by the expandable star design to present the relationship between objects, as shown in Figure 6.5. The main difference between the expandable star and collapsible star design is the manner in which the system shows the relationship between the nodes. It is noticeable that the expandable star design uses a method of expanding the nodes to showcase the relationship between a particular node and its associates.

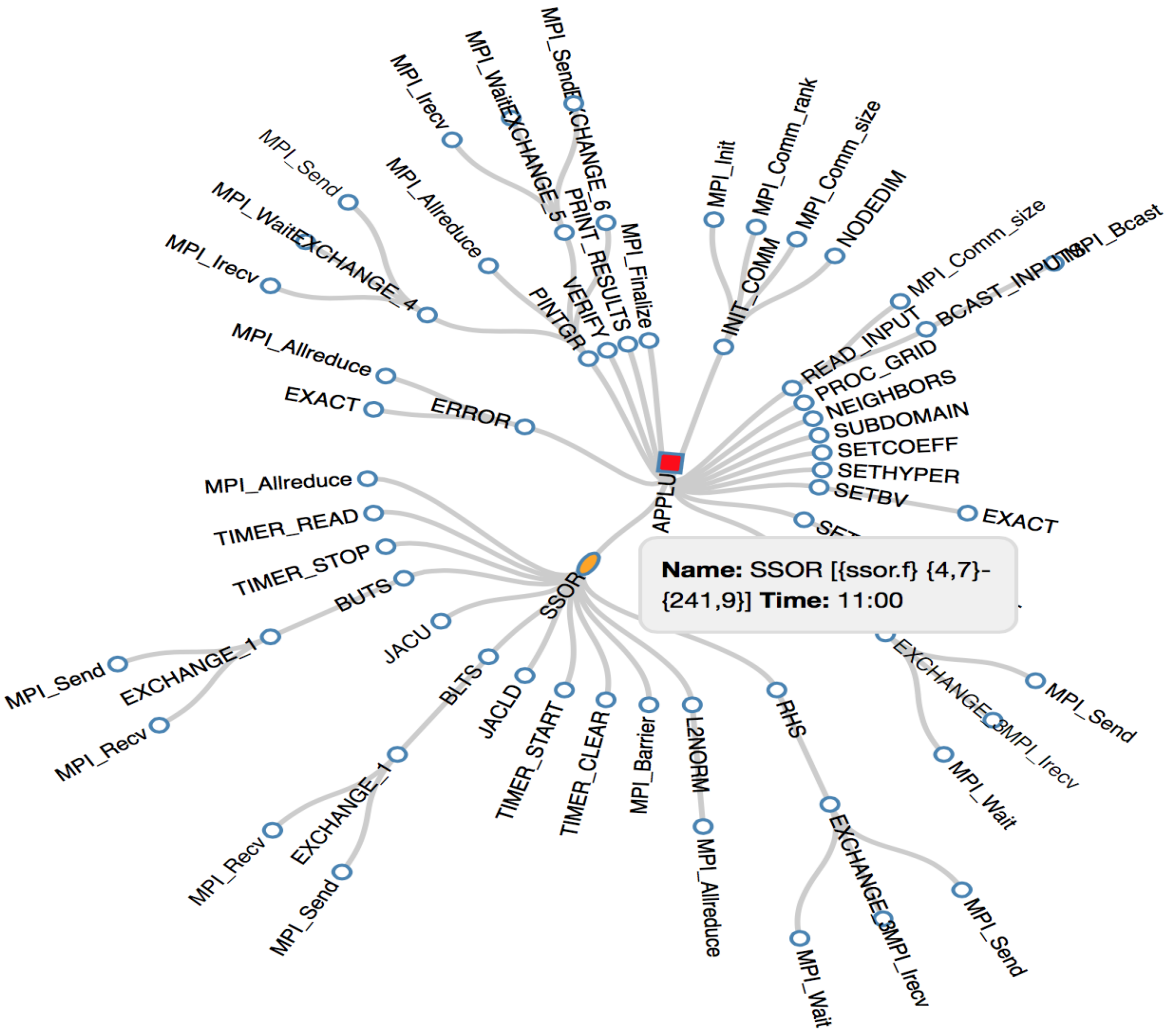

Figure 6.6: Collapse star design on two processors

As for the collapsible star design, it has the ability to filter and collapse the nodes. When the user clicks a node, the system clears the area and only shows the active (clicked) node with its associates. The method of filtering the node is helpful in avoiding an overload of information to the user. In Figure 6.6, the user has hovered over node name, SSOR, in order to obtain the details (full name and execution time) of the object. Most interestingly, the user clicked the SSOR node and therefore the following visualisation (Figure 6.7) was presented with a relationship of different nodes. In fact, the collapsible star design cleared the screen area and collapsed only the SSOR node with its associates. The user can then able to clearly see the relationship between group of nodes that associate with the SSOR object utilised by the NAS Benchmark program on two processors.

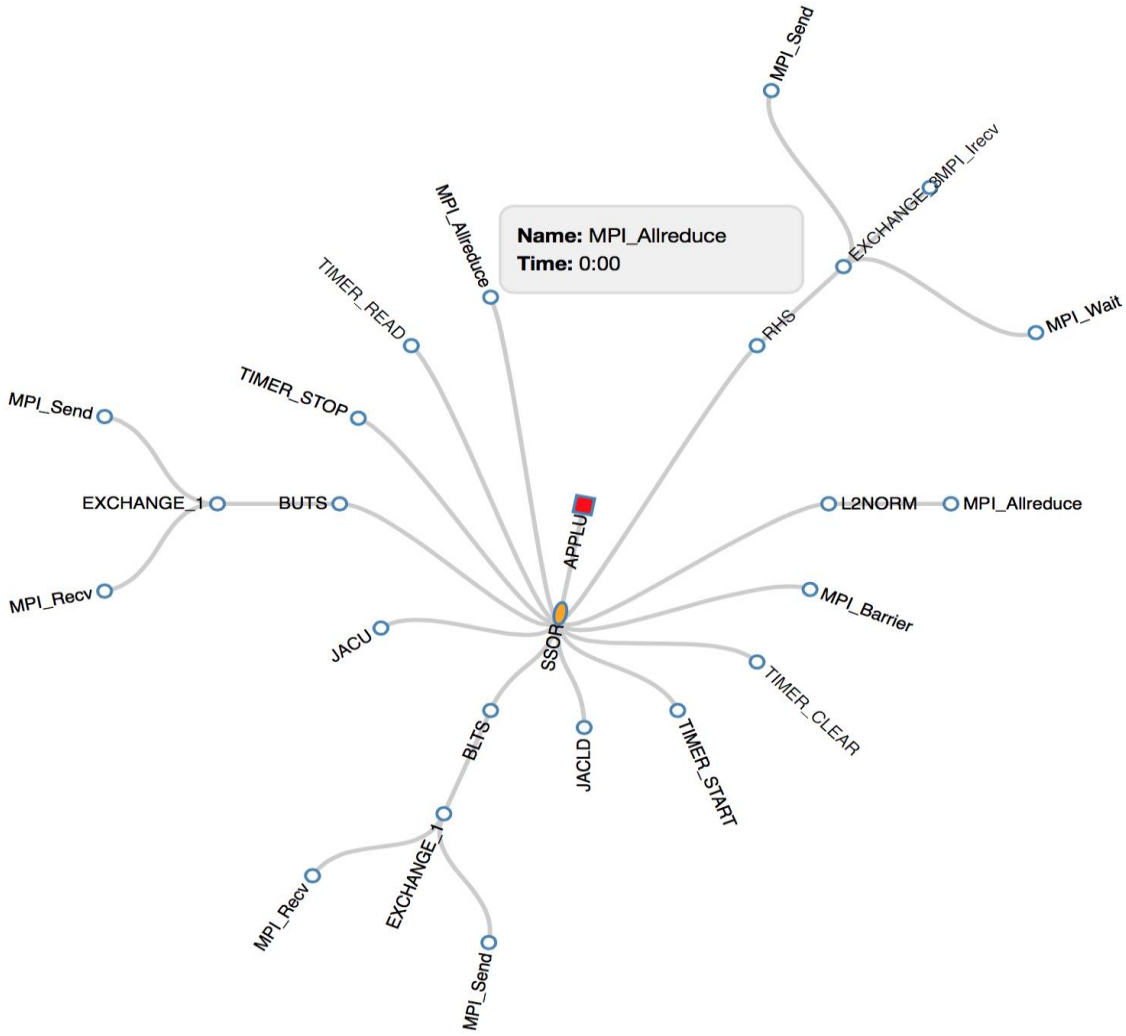

Figure 6.7: Collapsible star design with active nodes on two processors

The collapsible star prototype design also enables one to obtain the full details of the nodes after collapsing them. Figure 6.7 shows the full name and execution time of the MPI_Allreduce function, which is displayed after the user hovers over the node. The user can also click anywhere in the white area in order to return to the main screen of the design (Figure 6.6). The collapsible star design also enables the user to increase and decrease the size of the nodes (including names) when required. Figure 6.8 shows a collapsible star visualisation design with a filtered group of nodes; it demonstrates the second view of the collapsible star design used to visualise the performance of the NAS Parallel Benchmark on four processors of the computer.

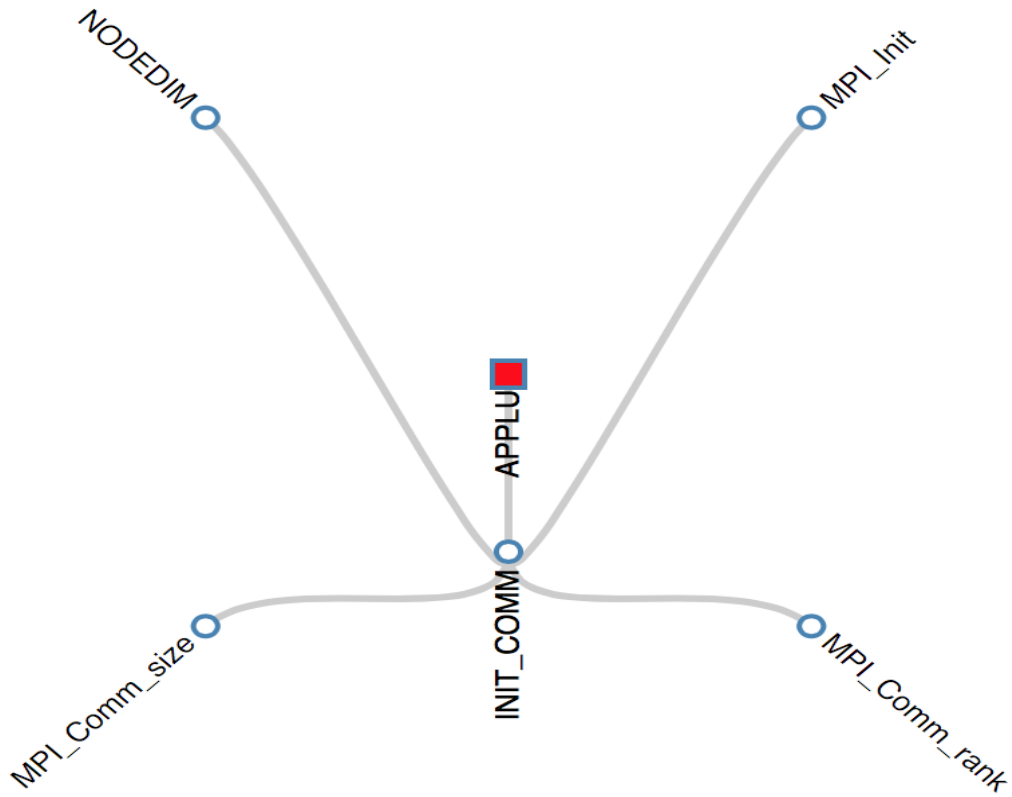

Figure 6.8: Collapsible star design (normal view) on four processors

It clearly displays the relationship between the INIT_COMM function and its associates. In all the interactive designs, the names of the objects are displayed using the following standard settings: font name - sans-serif, and size - 12, the system can re-adjust the settings depending on the number of nodes displayed on the screen. Now, the user can increase the size of the text in order to display a larger node size. When the user double-clicks on any node, the system then increases the size of the nodes (including names), as presented in Figure 6.9. It shows the second view of the collapsible star design after the user double-clicked the MPI_Comm_size node within the system. This collapsible star design indicates large white circles and a red square node associated with the names of the objects used to simulate the run. When the user continually double-clicks a node the system will also repeatedly increase the size of the nodes and names of the objects executed during the simulation.

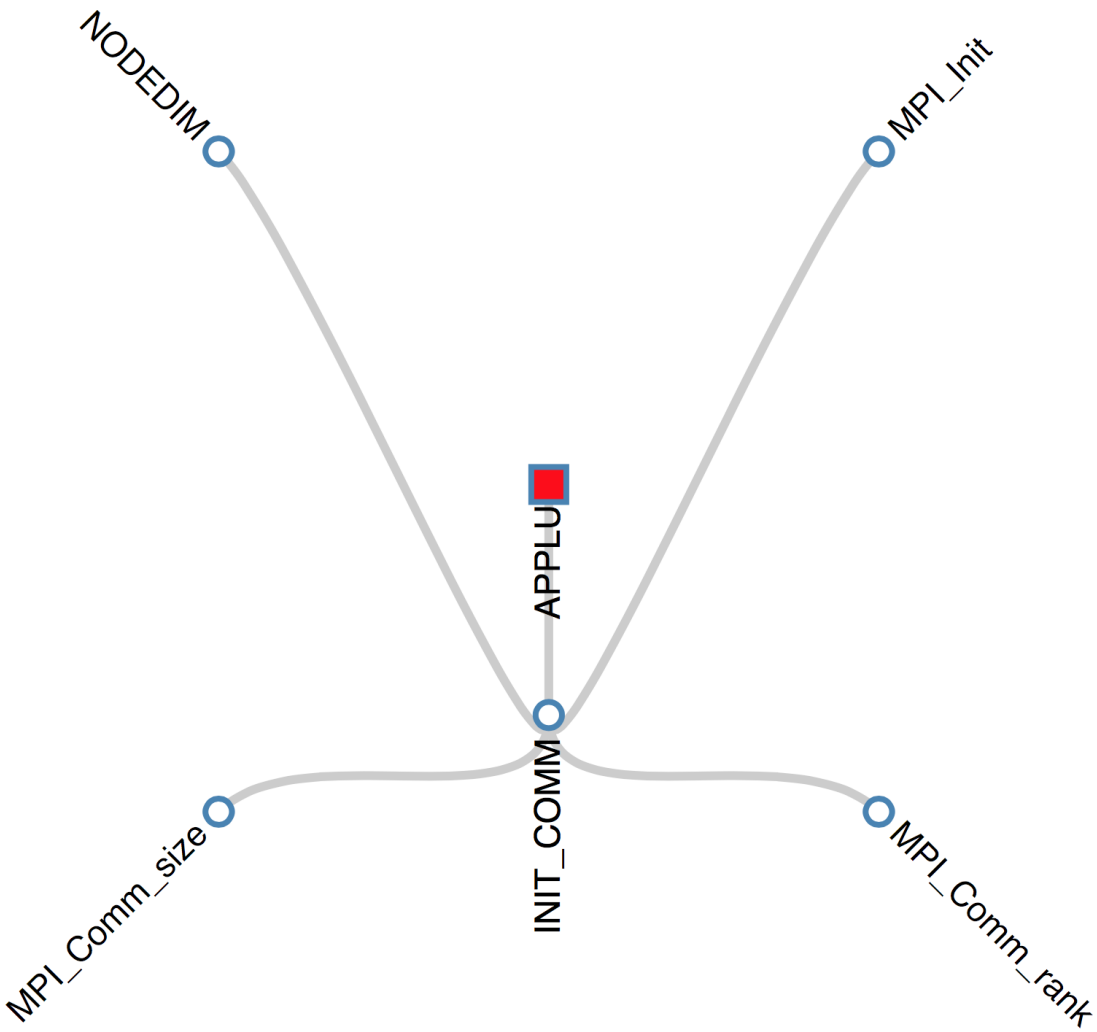

Figure 6.9: Collapsible star design (increased view) on four processors

The user can return to the main screen by clicking anywhere in a clear white area. Both Figures 6.8 and 6.9 illustrate the same design with different sized nodes and names of objects, such MPI functions, routines and the modules used by the NAS Parallel Benchmark program on four processors. The collapsible star design further utilises different visualisation methods to visualise performance data generated through the simulation of a program. In particular, the design shows a relationship of nodes (objects) connected to each other like a star, as shown in Figures 6.8 and 6.9. The design also uses the method of presenting the affinity of nodes like the structure of a human womb, as illustrated on Figure 6.10.

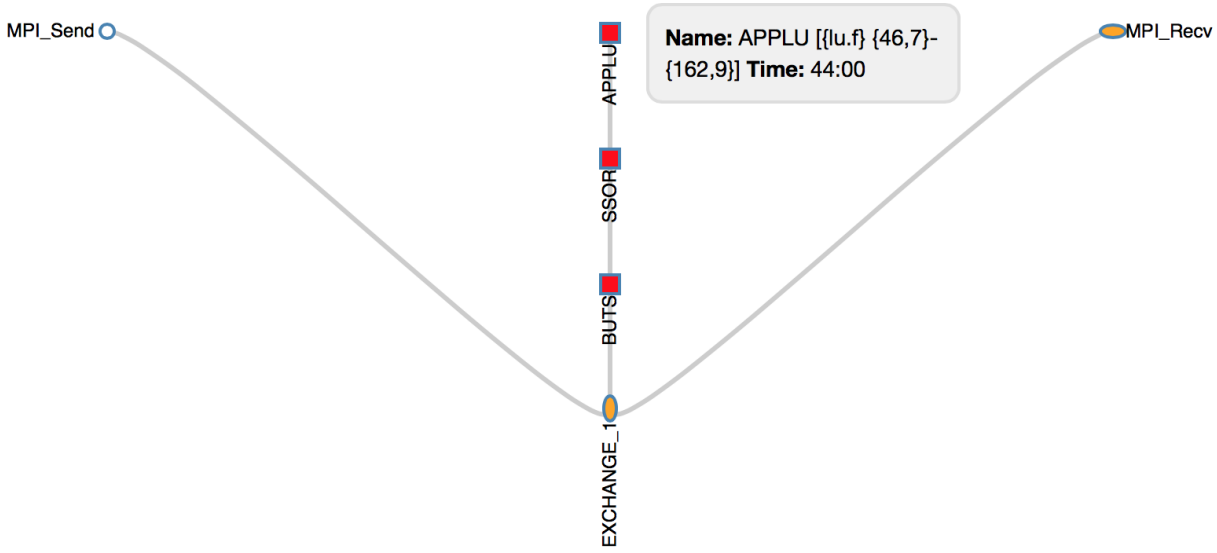

Figure 6.10: Collapsible star design on eight processors

The above picture (Figure 6.10) demonstrates the second view of the collapsible prototype design used to visualise the performance of the NAS Benchmark parallel program simulated on eight processors of the computer. D3.js uses intelligible methods of visualising data to present the relationship between nodes in a linear manner, while other children nodes act as flanks, as shown in Figure 6.10. In this collapsible star design (Figure 6.10), various types of nodes such as white circles, an orange oval and red square nodes were used to demonstrate different execution statuses (normal, moderate and excessive) of the objects used to compute the NAS program. Based on this analysis result, APPLU, BUTS and SSOR objects need to be optimised in order to obtain peak performance of the program. Figure 6.10 further shows that the APPLU module consumed an execution time of forty-four minutes during the run. The execution time of the main module, such as APPLU, represents the overall time taken to simulate a program. In some cases, it may be unnecessary to focus on the main module because it always carries overall execution time of the simulation, which can be identified as a performance bottleneck within the visualisation. Nonetheless, the EXCHANGE_1 and the MPI_Recv function were also identified as the objects that need attention in order to scale the performance of the program. The collapsible star visualisation design further shows the MPI_Send function being represented as a white circle, which means it performed well. The parallel programs can perform differently when using a different number of processors. On this front, the collapsible

star design was used to visualise the performance of the NAS Parallel Benchmark on sixteen processors, as shown in Figure 6.11.

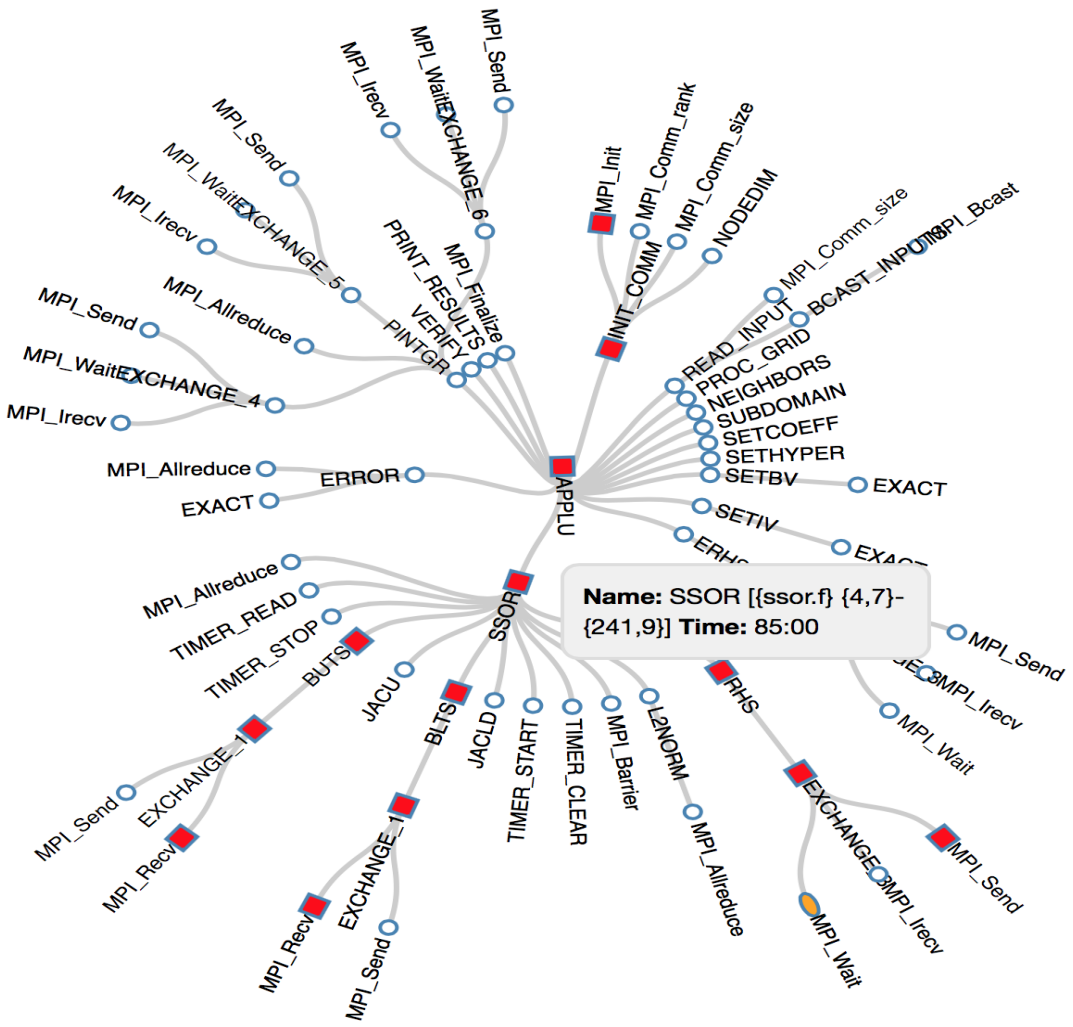

Figure 6.11: Collapsible star design on sixteen processors

It is noticeable that the above collapsible star visualisation design (Figure 6.11) shows the relationship of objects that have different execution statuses (normal, moderate and excessive) incurred during execution of the program. The visualisation result shows many nodes that have consumed excessive execution time during the run. For example, the SSOR node consumed an overall execution time of eight-five minutes. The collapsible interactive star design managed to present the relationship between objects despite the fact that it is a challenge to display many nodes on a screen simultaneously, as illustrated in Figure 6.11. It is important for the design to filter and collapse group of nodes, which enables the user to focus only on a particular group of

objects when optimising the program. The use of the mouse to increase and decrease the size of the nodes is also helpful in terms of accommodating users who may struggle with vision. The collapsible star design offers remarkable features when presenting program performance, using different visualisation mechanisms such as displaying the relationship between nodes in a star pattern.

It is also advantageous for the collapsible prototype design to both automatically enlarge the size and names of the nodes when the system displays a limited number of nodes, and decrease the font size of the names when there are too many nodes within the system. The main disadvantage of the collapsible star design is when some of the node's names move slightly over one another, but the user can hover over the node to obtain full details (name and execution time) of the object simulated during the execution of the program. It can only display a limited number of nodes within the diameter.

6.3 Tree interactive design

The tree interactive design consists of two prototypes visualisations – the expandable and non-expandable callgraph designs – as indicated in Section 6.1. Different tree interactive visualisation designs of the callgraph are discussed in the next Sections, 6.3.1 and 6.3.2.

6.3.1 Expandable tree design

It is expected that an interactive design should not overload the user with excessive information; instead, it should help the user remain in control of the tasks (Preece, Rogers and Sharp, 2002; Sharp, Rogers and Preece, 2009; Rogers, Sharp and Preece, 2011). The expandable tree callgraph visualisation is specifically designed so as not to present excessive amounts of information simultaneously. In fact, the system expands and disbands nodes to release/hide performance information of the program simulated on a computational system. In addition, the visualisation method of displaying nodes like the structure of a tree is used to represent the relationship between objects executed during the run. Figure 6.12 illustrates the first view of an expandable tree callgraph design used to visualise the performance of the NAS Parallel Benchmark program executed on two processors.

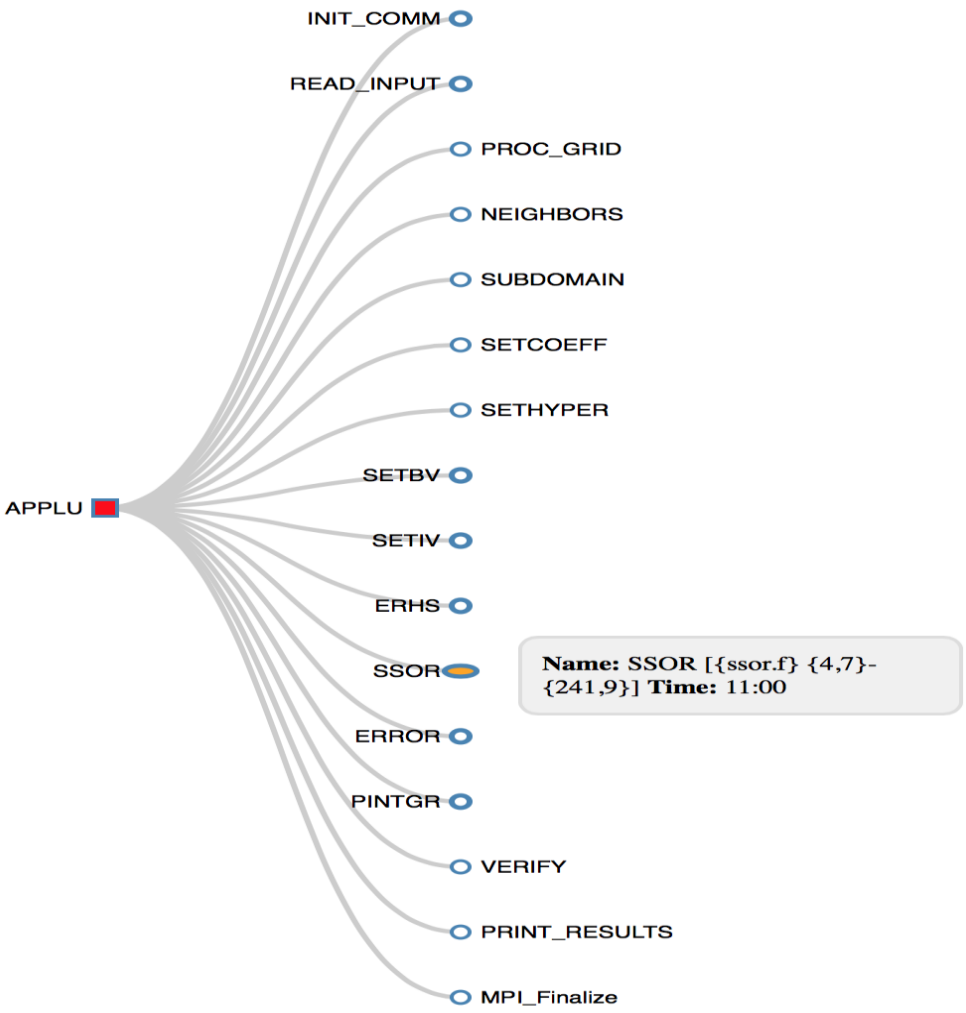

Figure 6.12: Expandable tree design (first view) on two processors

The above picture (Figure 6.12) illustrates a relationship between objects used to simulate the NAS Parallel Benchmark program. In particular, it demonstrates different types of nodes (white circles, an orange oval and a red square) that represent the performance of objects. In this case, it means that the expandable tree visualisation (Figure 6.12) identified objects that consumed different execution times (normal, average and immoderate) executed on two processors. Most importantly, the first view of the expandable tree design shows only the relationship between main module and objects that are directly associated with it. In Figure 6.12, the expandable tree prototype design shows the main module (APPLU) connected to different nodes, which represent routines and functions of the program. It does not display children nodes that are associated with the nodes.

The children nodes are hidden under the nodes with large borders that look like buttons. For example, the following nodes: PINTGR, ERROR, SSOR, ERHS, SETIV, SETBV, READ_INPUT, and INIT_COMM look like buttons, which means that the nodes have children nodes. The expandable tree design system (Figure 6.12) enables the user to click a button-like node in order to release and spread the children nodes across the system. On the other hand, one can also re-click the expanded node in order to disband, or hide, children nodes again. The border colour of the nodes is steel blue, which complements the white, orange and red colours inside the nodes, as shown in Figure 6.12.

The visualisation further shows the names of the objects represented using black, which makes the text clear on a white background. The expandable callgraph tree design also enables the user to point the mouse at a node in order to obtain the full details (name and execution time) of the object simulated during the execution of the program. Figure 6.12 shows full details of the node being displayed using black text within a textbox – the title of the name and the time of the object is displayed in a bold size to separate information.

The size of the textbox is designed to automatically increase and fit the name of the object when it is too long. Moreover, grey network links are used to showcase the kinship between the node and its associates, as demonstrated in Figure 6.12. The colour (grey) of the network lines play a very important role in ensuring that the user can clearly see the relationship between nodes that are connected to each other. In fact, grey network links work very well with the steel blue borders of the nodes, making the connection between the nodes clearer. Most interestingly, the expandable tree prototype system is able to manage and clearly display many nodes. Figure 6.13 shows the second view of an expandable tree callgraph visualisation design used to analyse the performance of a parallel program. The second view (Figure 6.13) of the callgraph tree design shows nodes and expanded children nodes connected to each other. It should also be noted that the network links expand when the system increases and adds more nodes in the design of the callgraph visualisation system. Most notably, the expandable tree visualisation design decreases the sizes of the nodes (including names) when the user expands or adds more children nodes in the system.

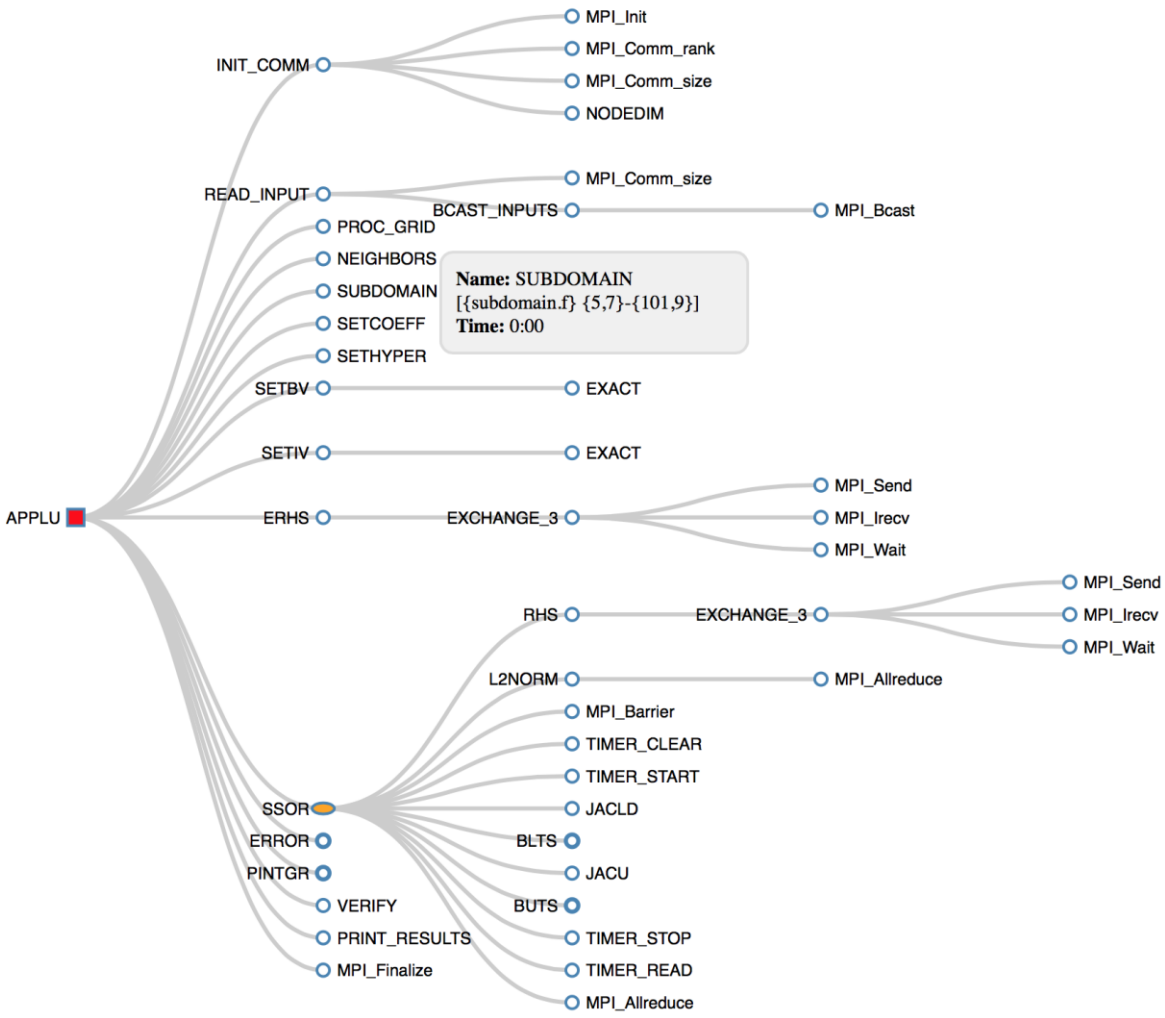

Figure 6.13: Expandable tree design (second view) on two processors

The system further displays all the names of the nodes using black text. In addition, the names of the nodes are clearly displayed in a vertical direction and do not move over each other, as shown in Figure 6.13. The expandable star visualisation is able to explicitly display many nodes without mixing the details of the nodes. A textbox displayed when the user hovers over the node also helps to provide full details (name and execution time) of an object executed during execution of the program. In the expandable tree design (Figure 6.13), the user has hovered over a SUBDOMAIN node and so the system displays a textbox that contains the full details of the node. The details of the node are also clear due to being black on a textbox with a grey background. The expandable star interactive design further enables the user to display some nodes, and to leave others. Figure 6.14 illustrates the expandable star design with expanded and non-expanded nodes.

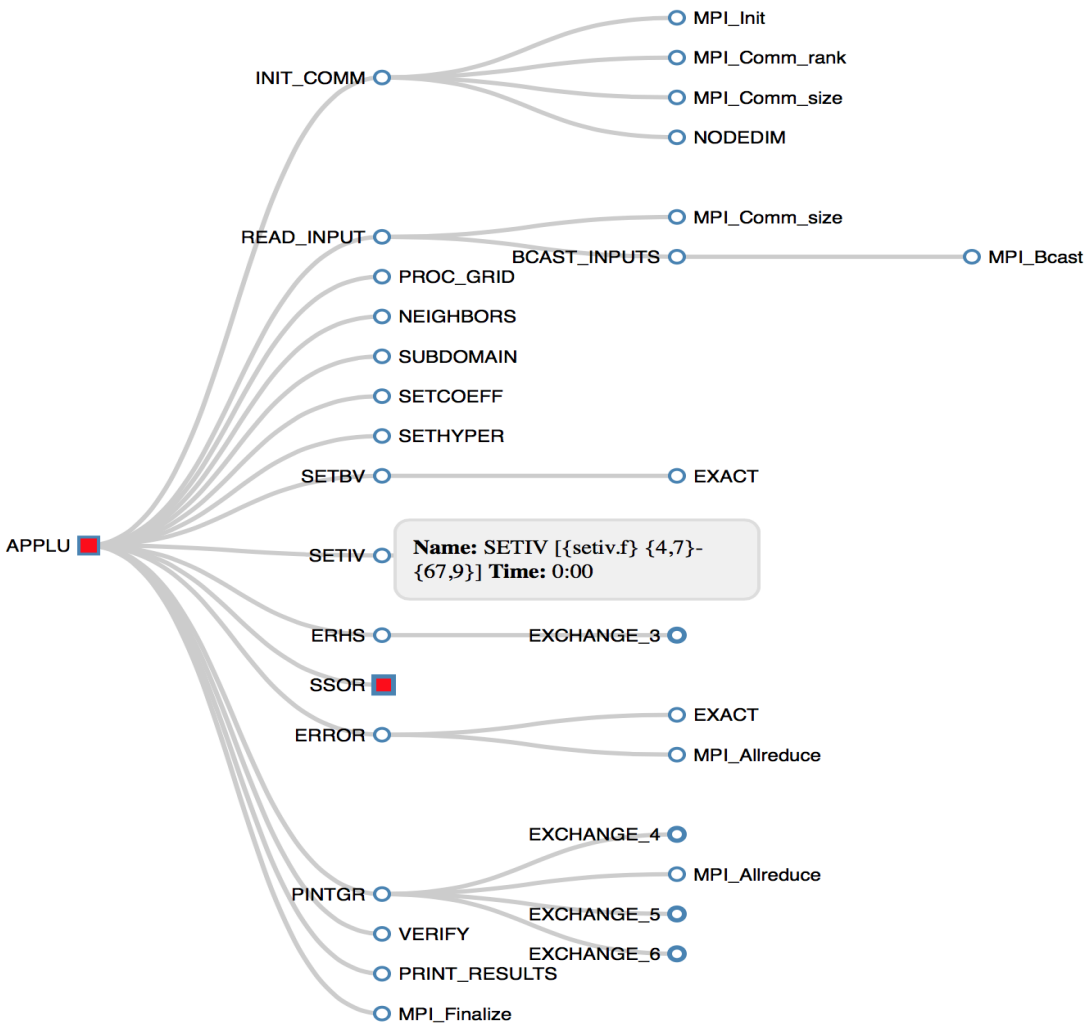

Figure 6.14: Expandable tree design on four processors

The above picture demonstrates an affinity between different nodes connected to each other via sophisticated network lines of the callgraph visualisation design; it further shows the relationship of different parts of the program simulated during the run. An expandable tree visualisation (Figure 6.14) illustrates expanded nodes with a small blue border and non-expanded nodes surrounded by a large border. In this tree interactive design (expandable), most of the nodes have been expanded to obtain the details of the children nodes, of which only few nodes – SSOR, EXCHANGE_3, 4, 5 AND 6 – are not expanded. The user is able to obtain the details of the non-expanded objects by either moving the mouse over the node, or clicking the node. When the user hovers over the non-expanded node, the system will provide full details of the node, as shown in Figure 6.14. When the user clicks the non-expanded node (for example, SSOR in Figure 6.14) the visualisation will expand and spread children nodes, like a tree. The

status of non-expanded nodes does not reflect the status of the children nodes within the tree interactive prototype visualisation. It is essential for the user to open/click the non-expanded nodes in order to obtain the status and detail of the nodes, which may be useful for optimisation of the parallel program simulated on a computational system. The below Figure 6.15, demonstrates a relationship between different nodes, especially SSOR and its associates. It further shows an expandable tree design used to showcase the relationship between different objects simulated during the execution of the NAS Parallel Benchmark program, executed on eight processors of a parallel system.

Various nodes are associated with different execution statuses such as normal, moderate and excessive and which represents objects. Both Figure 6.14 and 6.15 show red square nodes (SSOR) that have consumed excessive execution time during execution of the program. It does not mean that children nodes of the SSOR have also consumed immoderate execution time. Figure 6.15 indicates that the children nodes of the SSOR have obtained different execution statuses (normal, moderate and excessive) when simulating on the computer. The expandable star prototype design (Figure 6.15) illustrates that the following white circle children nodes - L2NORM, MPI_Barrier, TIMER_CLEAR, TIMER_START, JACLD, JACU, TIMER_STOP, TIMER_READ, and MPI_Allreduce function performed well.

The BLTS subroutine represented by an orange oval node shows that it consumed moderate simulation time, while the RHS and BUTS subroutines consumed too much execution time during the run. The expandable tree interactive design can visualise many different types of nodes. It can also clearly show a relationship between groups of nodes associated with each other. The use of the mouse to expand and disband the nodes helps the user control the group of nodes to be displayed. Figure 6.16 demonstrates the relationship between objects visualised using an expandable callgraph visualisation. Each node represent an object simulated on sixteen processors. In this expandable tree design (Figure 6.16), one is clearly able to identify a group of nodes that are associated with each other.

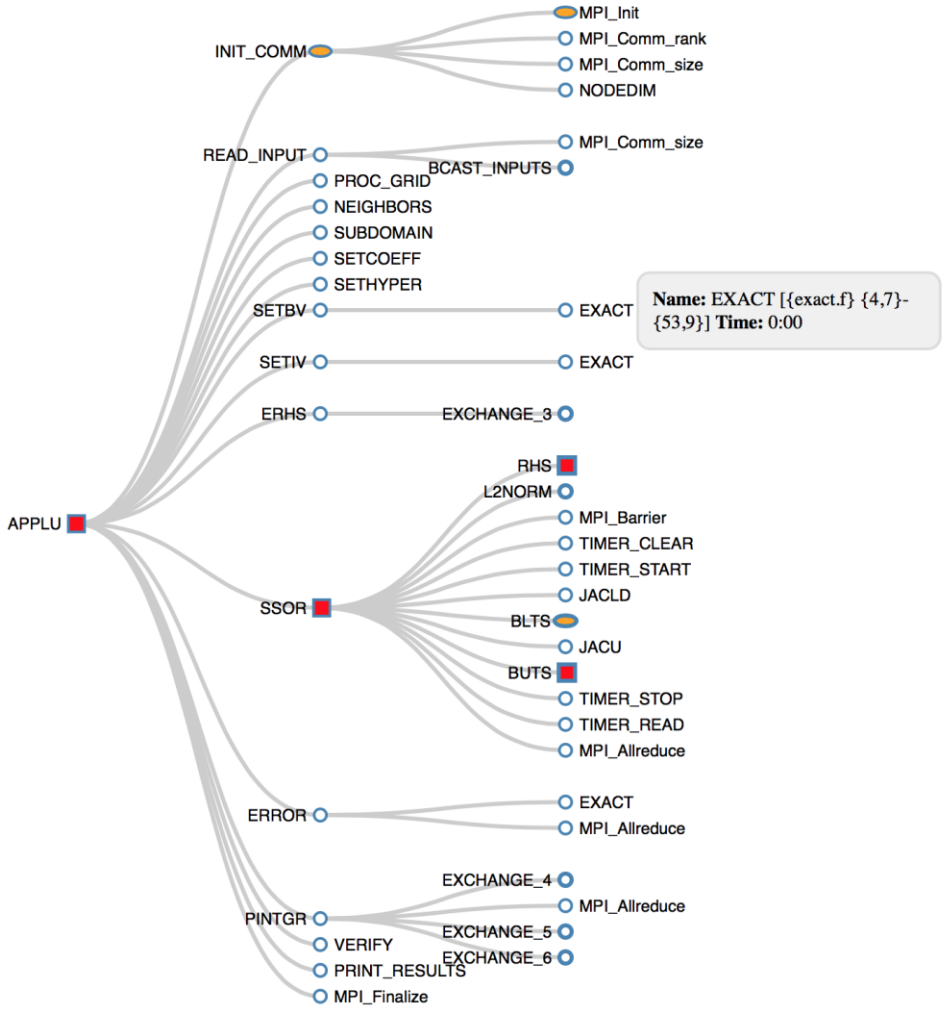

Figure 6.15: Expandable tree design on eight processors

The above expandable tree design (Figure 6.16) illustrates an SSOR routine calling RHS subroutine, while RHS calls EXCHANGE_3 subroutine, followed by MPI functions such as MPI_Send, MPI_Irecv and MPI_Wait. To this end, the user can also disband or remove all the children nodes in the screen by re-clicking a node.

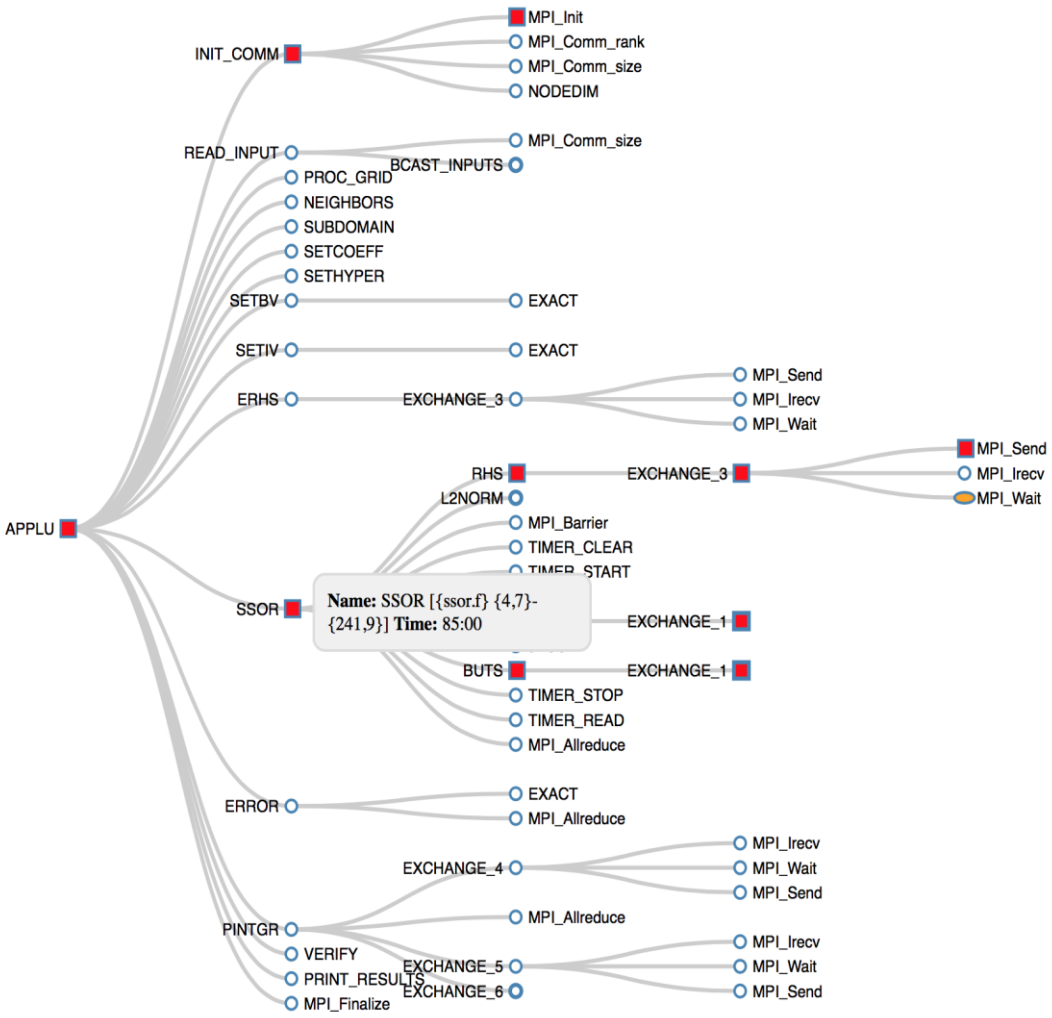

Figure 6.16: Expandable tree design on sixteen processors

When a user clicks an expanded node the system will automatically clear and remove all the children nodes of that particular clicked/active node. It was found that an expandable tree design was able to explicitly visualise many nodes without any conflict of names because the network links spread without hiding the details of the nodes. It is a further benefit of this expandable tree callgraph visualisation that it expands and disbands the nodes in order to avoid overloading the user with excessive performance information regarding the program.

All the names of the nodes are displayed in one single direction (vertical), which helps the user to easily identify the names of the objects simulated during the execution of the parallel program. The use of the mouse to display textboxes which contain details of objects is valuable in a sense that the user requires little effort in order to obtain full details (name and execution

time) of the objects. In this expandable tree design, it is also useful to have button-like nodes, which indicate nodes that have children nodes. The different node colours (white, orange, red) clearly identify execution statuses such as normal, moderate and excessive time consumed by an object during the simulation of the run. Various node shapes (circle, oval and square) of the nodes complement the use of colour in order to identify the execution status of the nodes. Different sizes such as small white circles, medium orange ovals and large red nodes help the user to differentiate the execution status of each object executed within the computational system.

The expandable tree design is able to display the relationship between many nodes because it expands the network links and decreases the size of the nodes when the user adds or spreads more children nodes. The relationship between groups of nodes is effectively displayed using grey network lines, which work very well with the steel blue border of the nodes, as shown in Figure 6.16.

When the user hovers over a node, the expandable tree visualisation will automatically increase the size of the textbox to fit in all details of the node. The black text used to display the names of the nodes is very clear on the white background of the expandable tree design. In fact, when the user hovers over a node, the system displays full details of the nodes using black texture surrounded by a grey background colour within a textbox. The text within the textbox is very clear. There are no identifiable disadvantages to the expandable tree design currently. Another type of tree prototype visualisation - non-expandable tree design - is looked at next.

6.3.2 Non-expandable tree design

The non-expandable tree design is a visualisation similar to the expandable tree design except that it does not hide and expand the children nodes; it displays all the nodes like a tree, simultaneously. In particular, Figure 6.17 demonstrates a non-expandable callgraph tree design used to visualise the performance of the NAS Parallel Benchmark program simulated on two processors of a computational system.

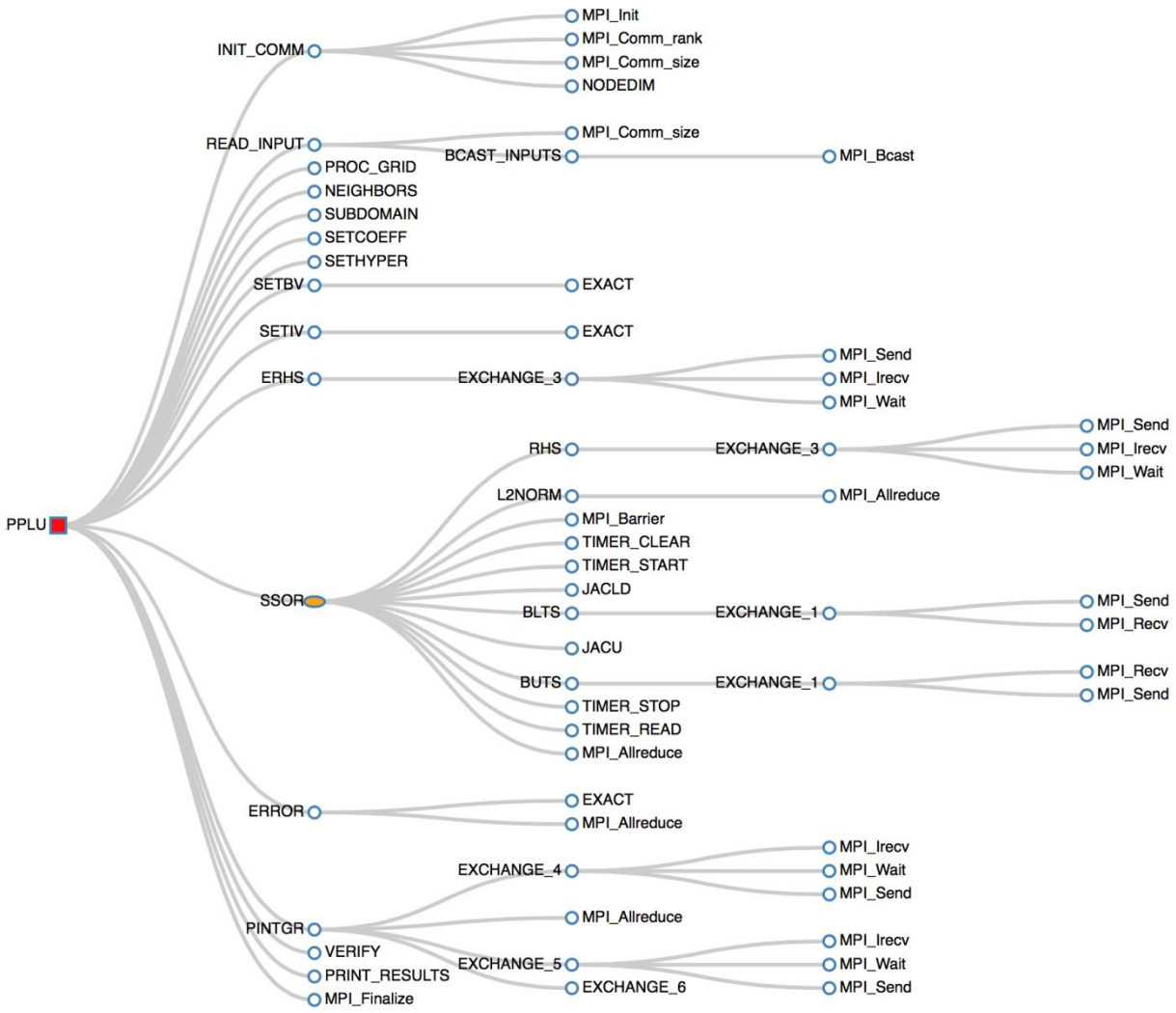

Figure 6.17: None expandable tree design on two processors

The diagram further shows the relationship between objects using different type of nodes such as white circles, an orange oval and a red square node, which represent different execution statuses (normal, moderate and excessive) incurred by an object during the run. Different node sizes are also used to identify parts of the program that consumed normal, moderate and immoderate execution time of the simulation. In this non-expandable tree design, white circles are smaller in size, while orange nodes are moderate and red square nodes are larger than all the nodes within the system, as shown in Figure 6.17. The grey network lines are used to showcase the relationship between the nodes, which has a steel blue border. Black text is used to display the names of the objects executed during the simulation of the parallel program. The

performance of parallel programs can fluctuate when one increases the number of processor to perform computational calculations within the system.

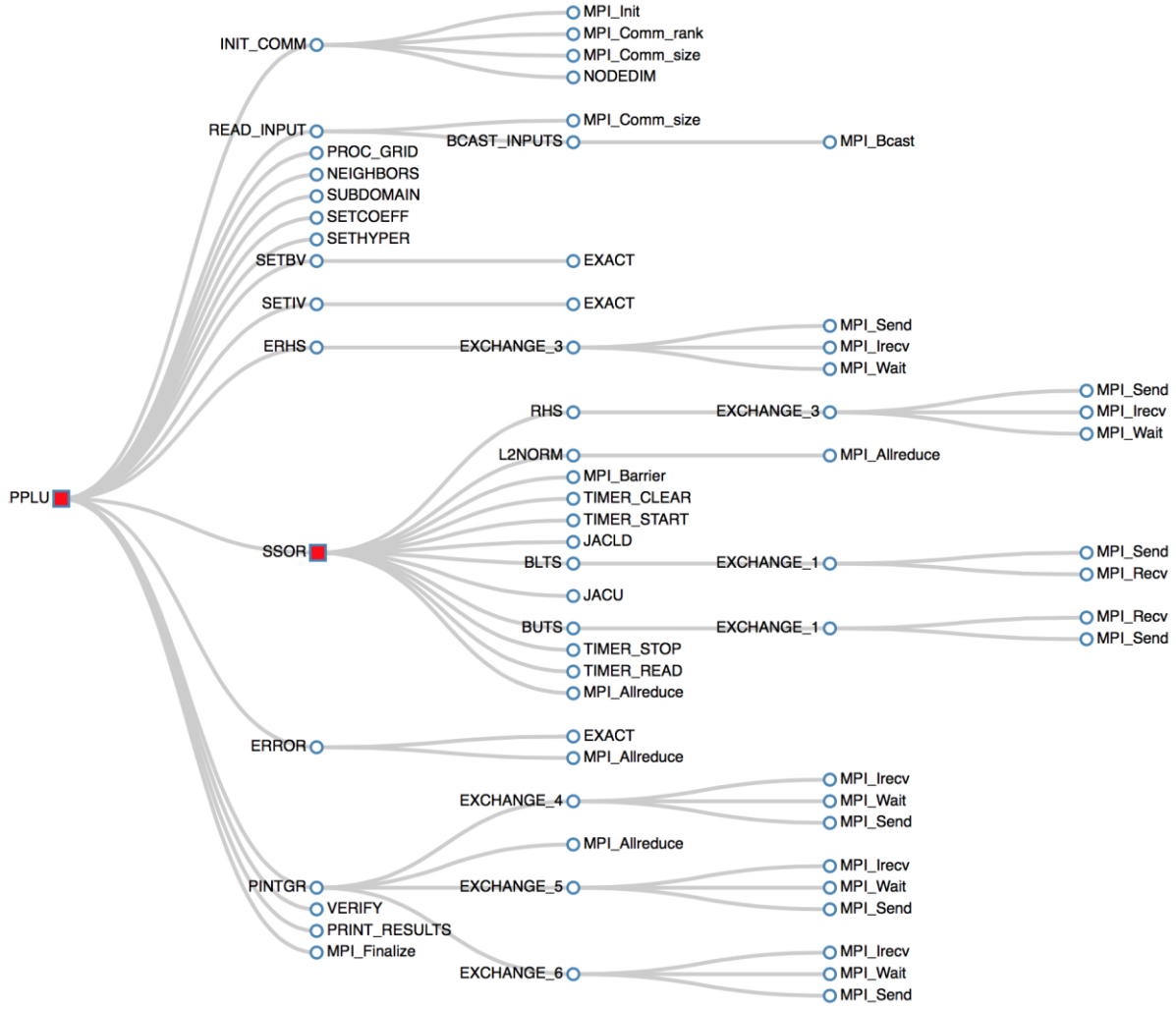

Figure 6.18: None expandable tree design on four processors

It is evident that the NAS Parallel Benchmark program performed differently when simulated on the following, different number of processors - two and four, respectively. Figure 6.17 shows that the SSOR object consumed an average execution time; Figure 6.18 indicates the SSOR node in an immoderate execution status. The non-expandable tree visualisation is able to identify different execution statuses (normal, moderate and excessive) of the nodes, which represents objects computed during execution of the program. It further use grey network lines to emphasise the connection between different nodes within the design, as depicted on Figure 6.17 and 6.18. The non-expandable callgraph tree design displays the names of the objects

using black, but it does not have the capability for showing the full details (full name and execution time) of an object simulated during the analysis of the program.

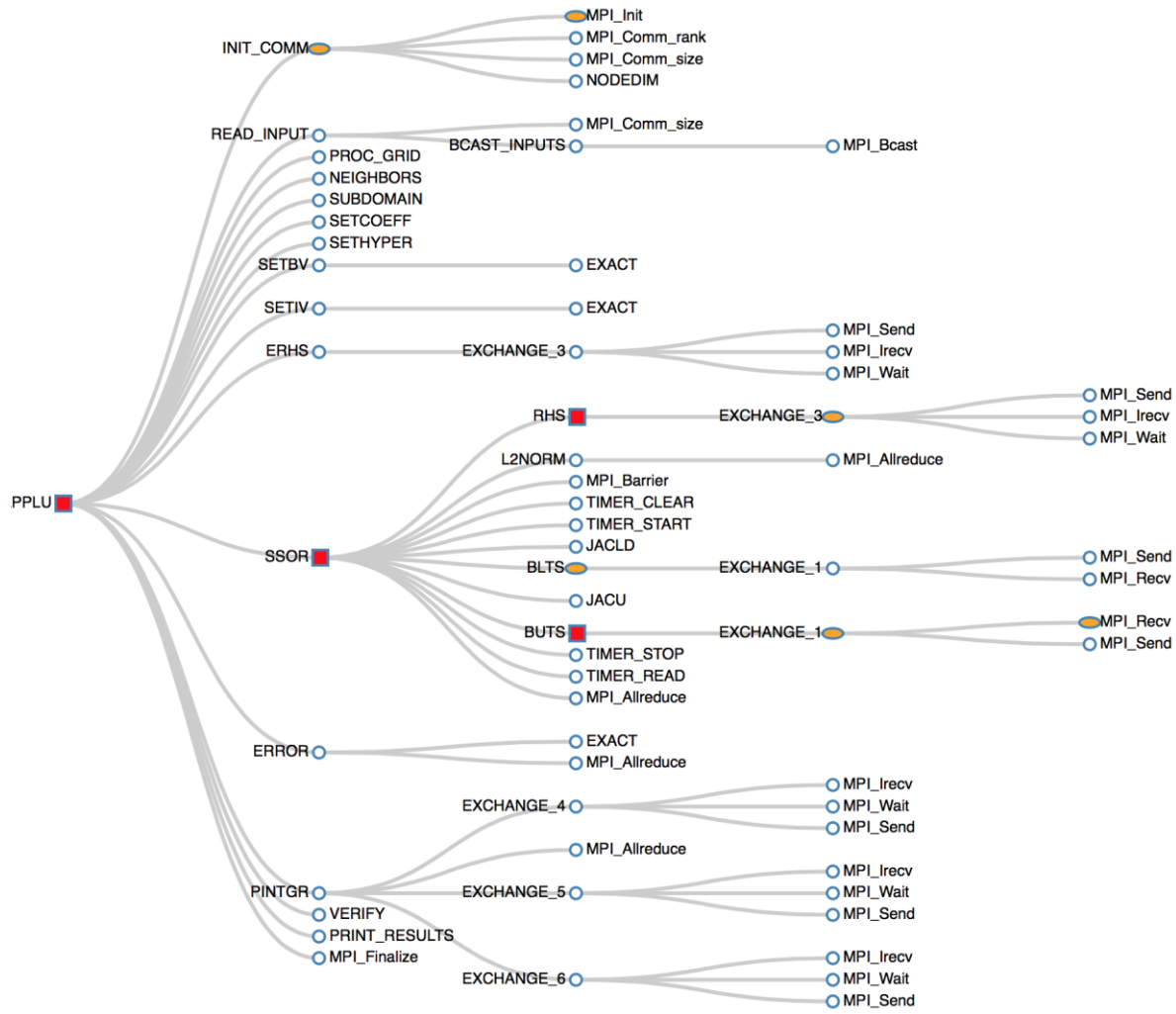

Figure 6.19: None expandable tree design on eight processors

It can still handle different types of nodes even though it may become problematic to identify the relationship between the objects when there are too many nodes displayed on the screen at the same time. The above (Figure 6.19) illustrates the connection between different types of nodes (white circles, orange ovals and red squares), which represent different parts of the program executed on eight processors. In this non-expandable tree design (Figure 6.19), many different nodes consumed moderate and excessive execution time compared to the visualisation (Figure 6.18) of objects on two processors. The following objects, INIT_COMM, MPI_Init,

MPI_Recv, BLTS, EXCHANGE_1 and 3 consumed moderate execution time; APPLU, SSOR, RHS and BUTS consumed excessive execution time during the run, as shown in Figure 6.19.

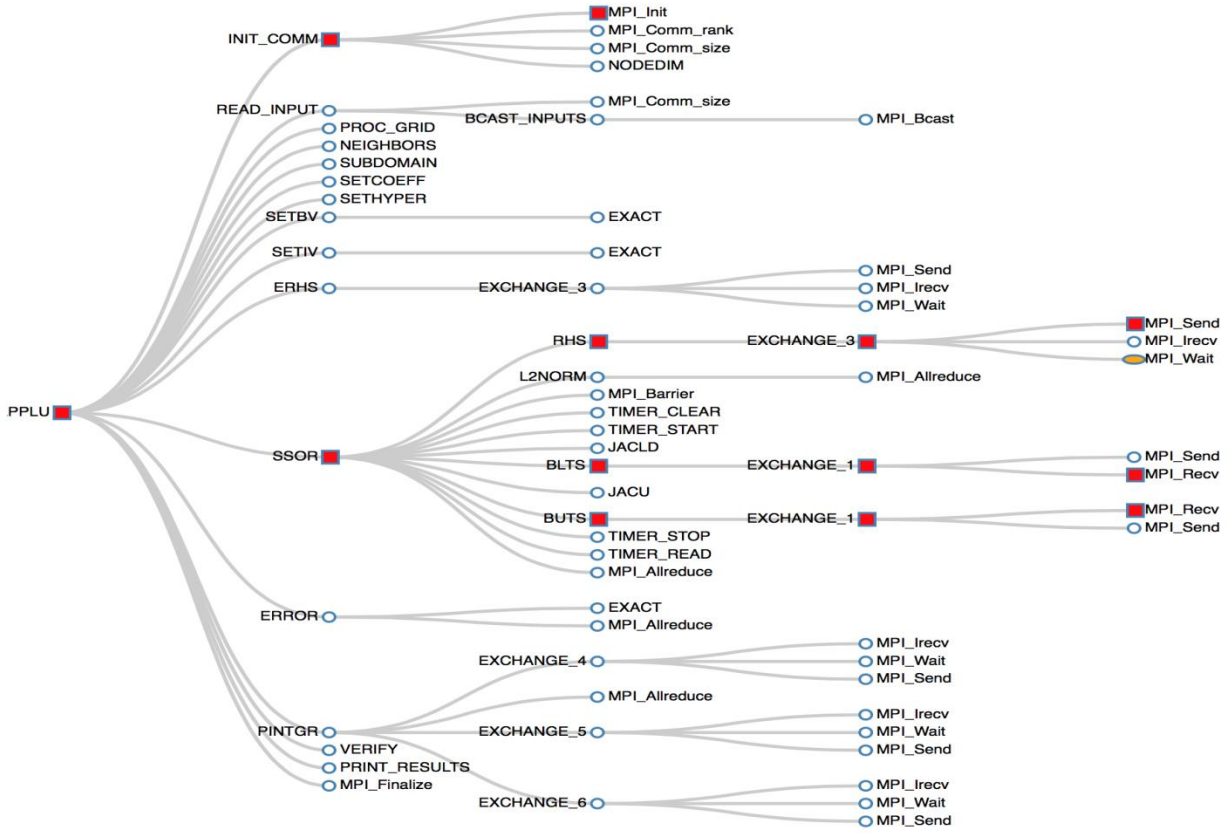

Figure 6.20: None expandable tree design on sixteen processors

The non-expandable tree visualisation design is able to establish the execution status of nodes, however, it cannot provide the exact execution time of the node. The full details of the node remain critical to the user because the details provide a clear indication of where the object was executed; it also ensures that the user is able to understand the procedure used to execute the object. Figure 6.20 demonstrates different types of nodes, which represent objects with a different execution status. The figure further shows two nodes that have the same name - EXCHANGE_1 - but the objects are called by different routines, BLTS and BUTS. In this case, the user needs the full details of the node in order to identify the procedure that calls an object. The non-expandable tree design has the ability to clearly show the execution status of the nodes, however, it will be challenging to showcase the relationship between nodes when the

program simulates too many objects during the run. These visualisation results further presents excessive performance information of parallel programs simultaneously, which may consequently lead the user not to obtain useful information that could help optimise the application. The non-expandable tree design visualises performance of parallel programs using almost the same visualisation method that is applied on the original callgraphs. Figures 3.4 and 3.5 demonstrate the performance analysis of the parallel program using the original callgraphs in which all the nodes were displayed on the screen at the same time. Figures 6.19 and 6.20 also show a non-expandable callgraph visualisation, which presents all the nodes simultaneously.

Both the non-expandable tree design and the original callgraph present the relationship between the nodes in virtually the same way, except that the tree design uses different sizes, shapes and colours to establish the execution status of an object computed during performance analysis of the program. The grey network lines of the non-expandable tree visualisation design also help to emphasise the affinity between the nodes. It is also appealing for the non-expandable tree design to use black text, which works very well on a white background. Both interactive star visualisations - expandable and collapsible designs - have good capabilities for visualising the performance of parallel programs.

Most notably, the expandable tree interactive visualisation design has proven to be the most suitable visualisation for analysing the performance of parallel programs due to its ability to handle many objects as well as clearly show the relationship between, and the execution status of, the nodes. The users tested all the interactive designs in order to determine which prototype is easy-to-use, informative and effective for the callgraph visualisation system. The aim is to develop an effective callgraph visualisation that will help users easily identify performance bottlenecks in parallel programs, as stated in Section 1.1.

Users are involved in the development stages of the system, as discussed in Section 4.2, design approach. Questionnaires and interviews are used to assess the conceptual and physical design of the callgraph visualisation system. In this chapter, the usability evaluation and comparison of the original (Figure 6.21) and first high fidelity prototype (Figure 6.22) callgraph visualisation designs are discussed. The objective of the evaluation exercise is to compare the original and first interactive design (expandable star, collapsible star, expandable tree and non-expandable

tree) callgraph design to better understand which one is more effective in terms of analysing the performance of parallel programs. In particular, users assessed the original, and first interactive prototype design, to identify the visualisation that is easy-to-use, informative and interactive when one analyses the performance of parallel programs. Users test the physical and conceptual designs of the callgraph visualisations in order to determine the strength and weakness of the designs. It is also anticipated that the users' tests will help identify the callgraph visualisation design that effectively identifies performance bottlenecks of the parallel programs. Most notably, feedback from the users will also act as a guideline to select the most appropriate visualisation design that has the ability to help users optimise parallel programs for optimum performance on the computational systems.

The process of evaluating the original and first interactive designs will further enable us to understand the physical interaction between the users and designs, which will be useful in developing an effective callgraph visualisation system. Different users tested the original and first interactive design of the callgraph visualisation system. More information about the tests will be discussed in Section 6.4. Below is the original callgraph visualisation design (Figure 6.21), which was used to visualise the Weather Research and Forecast (WRF) model running on the CHPC's Sun cluster.

TAU's callgraph visualisation (the original design) was used to analyse the execution of different objects like routines, subroutines and functions used to simulate the WRF on CHPC's Sun cluster, as shown in Figure 6.21. The execution status of different objects is represented by different colours such as dark blue, green and light green, as previously discussed in Section 3.1 - the overview of the callgraph visualisation tool. Users performed practical tests using the original callgraph visualisations (Figure 6.21) to analyse the performance of the WRF parallel program.

Users also performed practical exercises using first high-fidelity callgraph visualisations (Figure 6.22) - expandable star, collapsible star, expandable tree and non-expandable tree designs - to analyse the execution of the NAS Parallel Benchmark program. For more information about these designs, please refer to Section 6.2 and 6.3. The below picture (Figure 6.22) depicts different visualisation designs of the first high-fidelity prototype design, tested by the users.

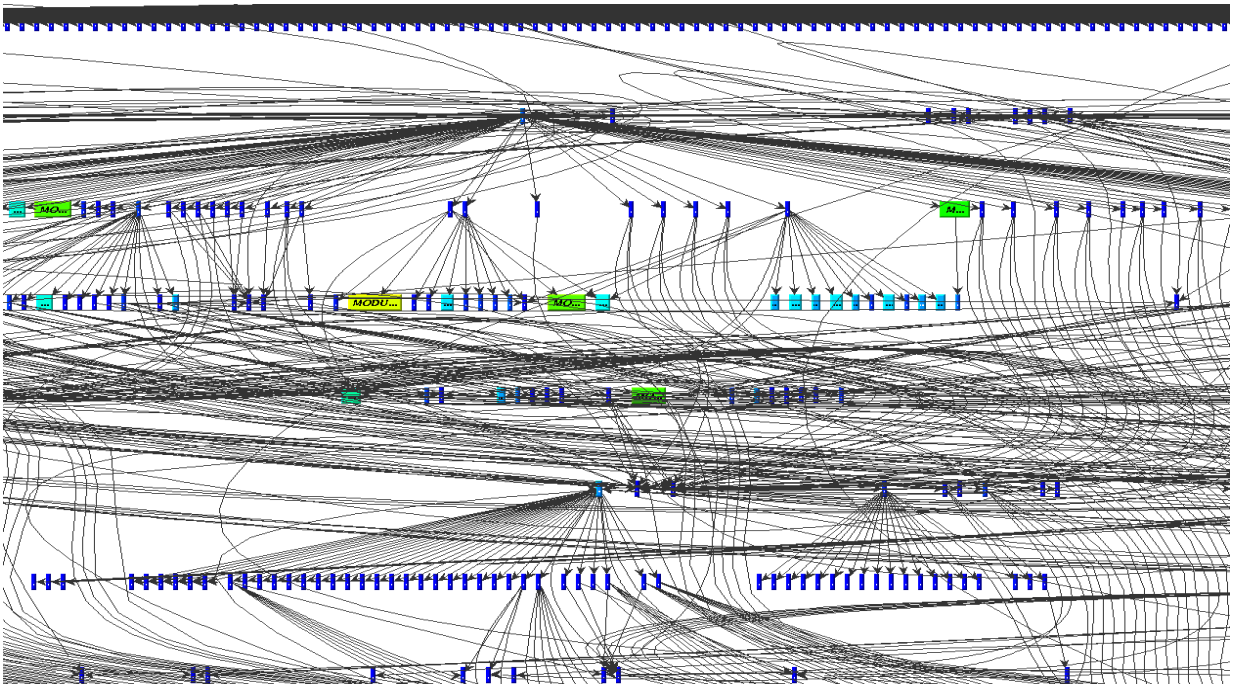

Figure 6.21: Original callgraph visualisation designed by the University of Oregon Performance Research Lab, used to visualise the Weather Research and Forecast (WRF) model on the CHPC's Sun cluster

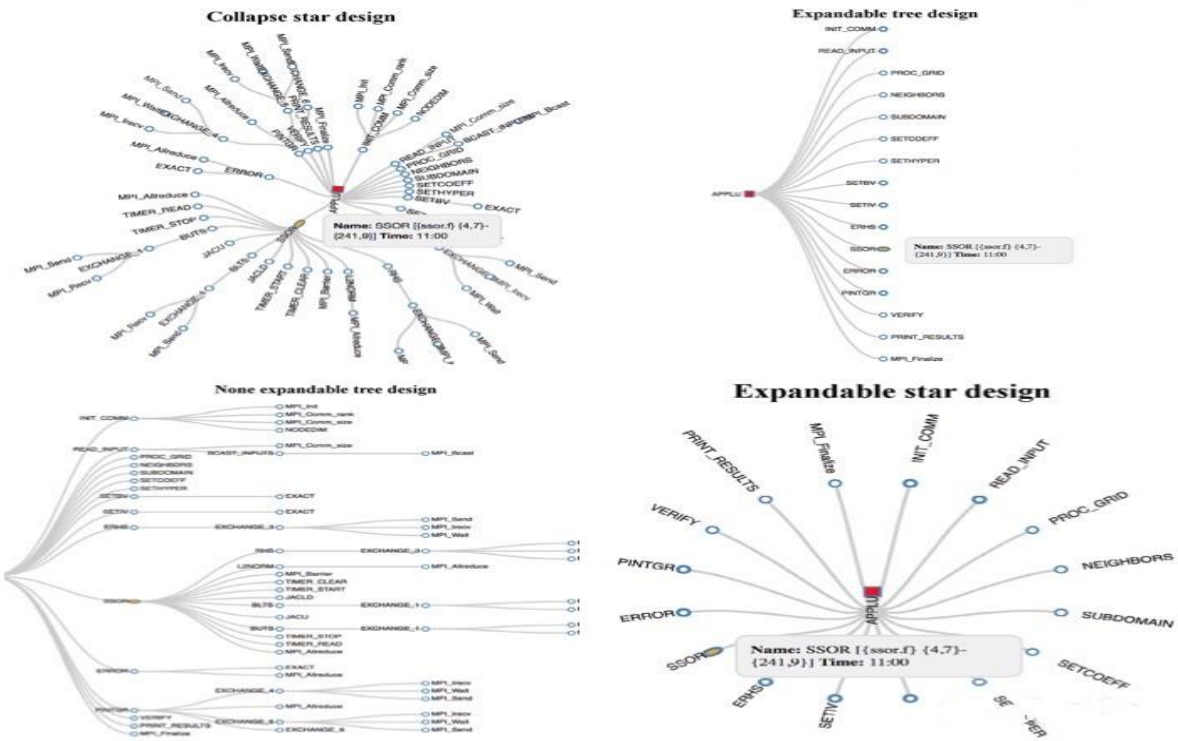

Figure 6.22: First high-fidelity prototype interactive visualisation designs: (top-left) collapsible star, (top-right) expandable tree, (bottom-left) non-expandable tree, and (bottom-right) expandable star visualisations of the NAS Parallel Benchmark program

6.4 Evaluation

6.4.1 Testing methods

The controlled settings mechanism (Section 2.6) was used to evaluate the callgraph visualisation designs. The controlled settings method was chosen to evaluate the callgraph design because the technique enables one to physically monitor and observe the tasks performed by users in a controlled area, such as laboratory. As part of conducting a design study (Sedlmair, Meyer and Munzner, 2012), users validated the new designs by performing visual queries using high fidelity prototypes and final design of the callgraph visualisations. As discussed in Section 1.3, seven users evaluated the usability of the first high fidelity prototypes against the original design of the callgraph visualisation system.

Moreover, four users inspected the usability of the second high fidelity prototypes against the original callgraph design. We selected users who were willing to voluntarily participate on the evaluation of the new and original callgraph visualisations. Each user signed the University of Cape Town (UCT) voluntary form to participate in the research. The University of Tennessee (UT) and UCT gave us human ethics approval to conduct a research study with the users (staff and students) of the university. To this end, users performed different tasks (visual queries) on the callgraph visualisations, filled in questionnaires and completed interviews. Users performed visual queries stated in Table 4.2 and completed questionnaires as shown in Table 4.4.

The visual queries, questionnaires and interviews were drafted by assessing documentation utilised to guide users of the original callgraph visualisation system. During the tests, the interaction between users and the visualisations (the original and first interactive designs) was captured using video, as well as audio recording of the interviews. In addition, notes were taken to summarise how users use the original and first interactive design (first high fidelity) of the callgraph visualisation system.

6.4.2 Testing environment

Testing was performed at the University of Tennessee (UT) Innovative Computing Laboratory environment, where users were seated in a closed, controlled office. During tests, all users had computers that ran Java 8 on a UNIX operating system (Narten and Burgess, 2003) - required to execute the callgraph visualisation - as well as internet access to the CHPC's Sun cluster. Moreover, Open Java Development Kit 8 software (Li and Tesfatsion, 2009) was installed on users' computers in order to execute the first interactive designs developed using the D3.js JavaScript visualisation library.

6.4.3 Users

Users were recruited from the UT's Innovative Computing Laboratory group and comprised PhD graduates (staff) and undergraduate computer science students. In particular, five staff members and two undergraduate students participated in the process of testing the designs (original and first interactive prototype design) in order to develop effective callgraph visualisations. UT staff and students (users) were selected to participate in the evaluation of the callgraph visualisations because they have computer science skills in optimising, parallelising, debugging, analysis and development of parallel programs. Users' job titles (duties) and experience (number of years) in High Performance Computing were recorded because user's parallel programming experience influences their interaction with callgraphs.

For example, programmers with one year of experience should be able to run, analyse and optimise parallel applications. The quantitative evaluation method was used to assess the experience (number of years) and duties of the users who tested the designs, as shown in Figure 6.23. In fact, Figure 6.23 (*A*) indicates experiences (number of years) of the parallel program users who performed usability tests on the original and first interactive design of the callgraph visualisations.

In particular, the X axis shows different users who tested the designs while the Y axis indicates the user's experience, particularly, the number of years in the programming field. For security reasons, users' personal details (such as names) are not listed in Figure 6.23 (*A*); instead, the name of each is replaced by the word "User", followed by a number - such as User 1, up to

User 7. Seven users tested the original and first interactive callgraph visualisation. Parallel program users have a range of experience from novice to experienced (Figure 6.23 *A*).

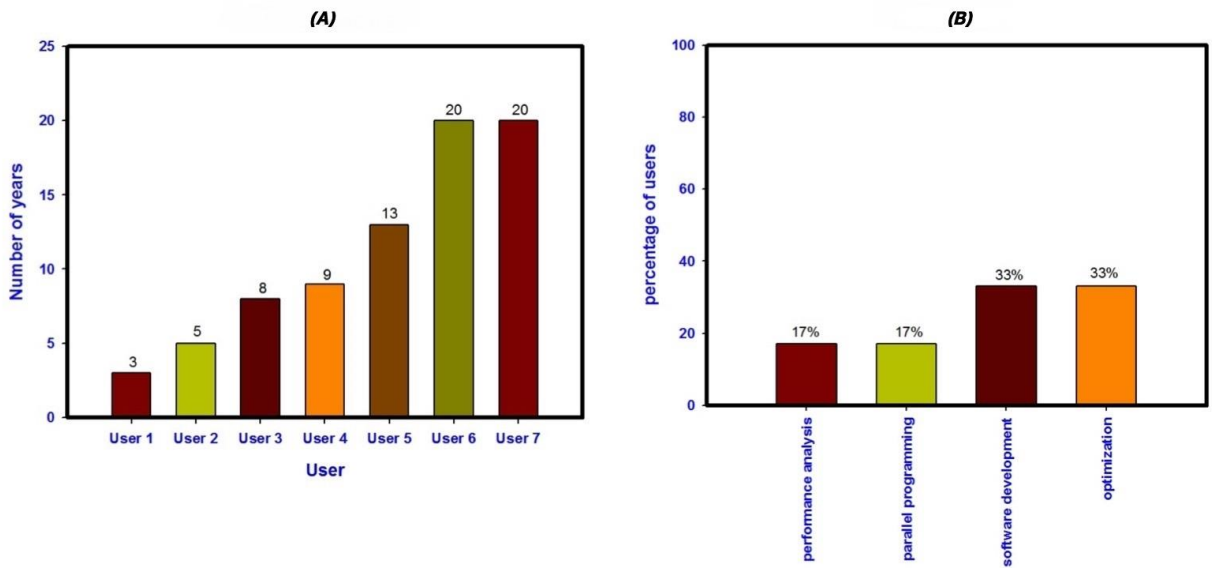

Figure 6.23: Users experience (A) and duties (B) in computational science

Various users with different levels of knowledge and skills on parallel programming were selected to test the original and first interactive prototype designs of the callgraph visualisation tool. Users with different levels of experience are likely not to have the same requirements from the visualisation system. For example, a person whose duty is to develop parallel programs will have a different need from the users whose job is to analyse the performance of the parallel programs. Figure 6.23 (*B*) demonstrates different duties performed by the users on a regular basis. The regular duties of these users are as follows: software development, optimisation, performance analysis, and parallel programming, as indicated in Figure 6.23 (*B*). The figure also indicates that 33% of the users' duties is to develop parallel software while another 33% of the users' obligations is to optimise parallel programs.

Furthermore, 17% of the users perform analysis of parallel applications, while another 17% of the users parallelise the computational tasks. This means that most of the users who tested the visualisations had experience in optimising and developing parallel programs, while others analysed the performance and execution of the programs. A performance analysis tool such as

callgraphs, should help parallel program users to identify inefficiencies and easily establish performance bottlenecks when analysing, optimising and developing parallel programs.

6.4.4 Testing procedure

Two parallel programs, the WRF and the NAS Parallel Benchmark program, were selected to perform tests using both the original and first interactive prototype design of the callgraph visualisation system. As discussed in Section 1.3, the WRF was chosen because most of the CHPC users utilise it to simulate scientific calculations on the CHPC cluster. Furthermore, the NAS Parallel Benchmark was selected to perform visual tests because the designer of the original callgraphs, the University of Oregon (UO) Performance Research Laboratory, recommended that the test be performed using the program. For testing purposes, the performance of the NAS parallel program was visualised using the first interactive visualisation design with performance metrics collected from two, four, eight and sixteen computational processors. The performance data was provided by the UO Performance Research Laboratory, which developed the original callgraph visualisation system.

The original callgraph visualisation was used to visualise the performance data of the weather simulation (WRF-3.5) executed on forty-eight processors of the CHPC's Sun cluster. Forty-eight processors were chosen because most of the CHPC supercomputing users use this number of processors to perform scientific calculations using WRF-3.5. To familiarise users with the callgraphs, they were given a live demonstration showing both how the original callgraph works, as well as how the first high-fidelity (first interactive prototype) callgraph visualisation operates. During this exercise, parallel program users (UT staff and students) were provided with information such as the features and capabilities of both the original and first interactive prototype callgraph visualisations. Subsequently, users were given an opportunity to clarify information about the operation of the visualisation designs. Afterwards, users started to perform tests and evaluation of the designs. During the tests, each user was expected to independently perform the visual queries (as shown in Table 4.2) using both original and first interactive design of the callgraph visualisations. These visual queries were chosen after reading the documentation and analysing how the callgraphs worked. After performing each query; the users wrote their response for that particular query on an answer sheet. Firstly, users performed different visual queries using the original callgraphs by analysing the performance

of a model - the Weather Research and Forecast (WRF-3.5), running on the CHPC's Sun cluster. In this exercise, users performed the visual queries that were discussed earlier in Section 6.4.4. All the users performed the visual queries and wrote answers on the question papers without any assistance. After performing visual queries on the original designs, users then performed the same visual queries on the first interactive prototype designs of the callgraph visualisation system. To perform visual queries on the new designs, users visualised the performance of the parallel application - the NAS Parallel Benchmark - simulated on a different number of processors: two, four, eight and sixteen. After performing these practical visual query exercises, users completed the questionnaires (as stated on Table 4.4) regarding the usability of the original and first interactive design of the callgraphs.

6.4.5 Analysis

Various data such as visual queries, questionnaires, interviews, notes, videos and audio recordings [10], collected during the evaluation of the callgraphs was analysed using the quantitative and qualitative method of assessing different types of data, for example numerical and non-numerical. The quantitative method is a mechanism used to analyse numerical data, or data that can be converted into numbers, while qualitative method is used to assess non-numerical data. For more information about quantitative and qualitative assessments methods, please refer to Section 2.6.

6.4.6 Results and discussion

6.4.6.1 Users visual queries

The users need an effective callgraph visualisation that will easily identify performance bottlenecks incurred during the execution of the run. It was found that users find it hard to perform visual queries on the original design, which is meant to effectively provide performance information on parallel programs to the users. In this research study, visual queries performed by the users were analysed and interpreted accordingly. The quantitative

[10] Data can be found on this website: https://people.cs.uct.ac.za/~mmabakane.

analysis method of converting non-numerical data to numbers was applied to understand how users performed visual queries on the original and first interactive design of the callgraph visualisations, as shown in Figure 6.24. The visual queries performed by users on the first interactive prototypes against original callgraph design were analysed to compare usability and interactivity of the designs. In essence, the answer for each visual query was analysed to identify if the user either did, or did not, manage to positively answer the query. For example, did a user manage to establish the amount of execution time consumed by the MPI_Init() function? Should the answer be "yes" then one point is added to that particular query, as shown in the below Figure 6.24.

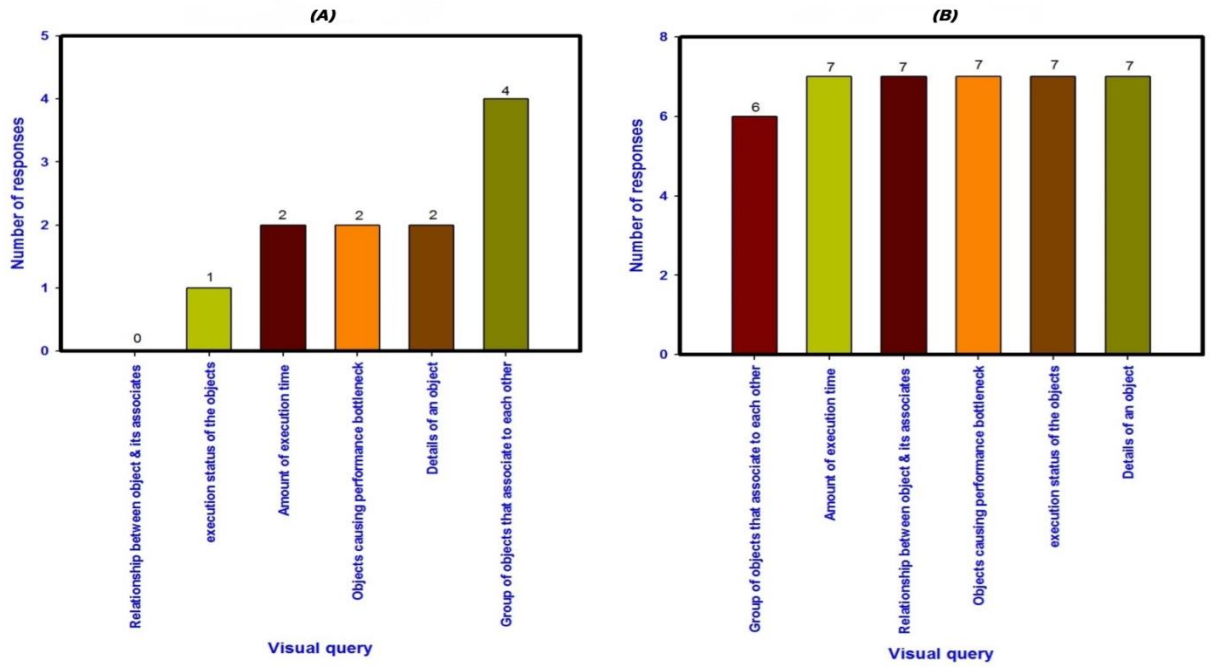

Figure 6.24: Visual queries performed on the original (A) and new (B) designs

In this exercise, the data collected was analysed to identify how many users managed to positively answer the questions relating to the following six visual queries: the relationship between object and their associates; execution status of the objects; amount of execution time; objects causing performance bottlenecks; obtaining details of an object (for example name); and groups of objects that are associated with each other. First and foremost, Figure 6.24 (*A*) presents the results regarding how many users correctly answered the answers relating to these visual queries using the original callgraph visualisation.

Figure 6.24 (*A*) shows that none of the parallel program users managed to establish a relationship between a particular object and its associates using the original design. It is evident that the original callgraph visualisation cannot clearly show the relationship between many objects simulated during the execution of the parallel program. The reason that parallel users (UT staff and students) appeared not to be able to identify the relationship between objects was due to the numerous network links moving over the nodes, which consequently hides the details of the object simulated during the execution of the parallel model. In addition, the WRF (Weather Research and Forecast) model simulates too many objects (modules, routines, subroutines and functions) at the same time, and the original design of the callgraphs does not have sufficient capacity to visualise, present and scale well large graphs.

Figure 6.24 (*A*) reasserts the early findings of this research study, which indicated that the original callgraphs do not have the ability to visualise large performance data and display an effective graph to show the relationship between objects, as discussed in Chapter 1. During tests, User 2 Figure 6.23 (*A*) wrote the following statement in a question paper during the performance of visual queries on the original design: "It's not easy to obtain the details of the node, which lead the user to the relationship". This statement was responding to the visual query regarding the relationship between an object and its associates. User 1 wrote the following statement regarding the same visual query: "Too much information. Search will be a good tool". User 4 wrote: "It can be hard to follow the amount of information through the visual brainfall in many cases".

With these three statements, it is clear that users find it hard to establish the relationship between a particular object and its associates when using the original visualisation design. Nevertheless, at least two users managed to establish objects causing performance bottlenecks, amount of execution time, and details (names) of the objects - as shown in Figure 6.24 (*A*). One of the main purposes of using the callgraph visualisation is to identify objects that cause performance bottlenecks, in order to optimise negatively affected parts of the parallel programs for optimum performance in parallel systems. In this case, all users who tested the original design were expected to identify objects that caused performance bottlenecks; this was not the case. In fact, Figure 6.24 (*A*) depicts that only one user managed to establish the execution status of the objects. It is a clear indication that the original callgraph did not have a useful

visualisation method to help users analyse and optimise parallel programs. Four users were able to identify groups of objects that were associated with each other, as shown in Figure 6.24 (*A*). In summary, most of the users found it hard to perform visual queries on the original callgraph visualisation. On this front, parallel program users performed visual queries on the new designs (first interactive prototype designs) in order to test the conceptual and physical design of the visualisations in terms of how useful, informative and interactive the designs were. Figure 6.24 (*B*) demonstrates how parallel program users performed visual queries on the first interactive prototype designs of the callgraph visualisation. The figure shows that all users successfully managed to perform the following six visual queries on the new designs: execution status of the objects; amount of execution time; objects causing performance bottlenecks; obtaining details of an object (for example, name); amount of execution time consumed by different objects; and relationship between the object and its associates.

Moreover, six parallel program users were also able to establish groups of objects that were associated with each other. In fact, only one user did not manage to successfully perform these visual query of identifying groups of objects that relate to one another; the same user managed to successfully perform the other five visual queries on the new design. It is believed that this user (User 6) misunderstood the visual query regarding how to identify groups of objects that relate to one another.

Figure 6.24 (*B*) clearly shows that users managed to easily perform visual queries on the first interactive prototype design, which suggests that the new visualisation design (first interactive prototype) is effective, useful and informative in terms of optimising parallel programs. It is evident that the first interactive design helps users analyse the performance of parallel programs, which is not the case with the original callgraph visualisations. The interaction between the users and system will be discussed in the next Section, 6.4.6.2.

6.4.6.2 Users' questionnaire and interviews

The users (UT staff and students) completed the questionnaires about the usability of the original and first interactive prototype designs, as discussed in Section 6.4.4. The data generated through questionnaires was analysed and interpreted using quantitative assessment method. Addressed first was how users responded to the question regarding whether or not the

visualisation design was easy to understand. The users were asked to indicate which design (the original or first high fidelity prototype) was easy to understand, as shown in Table 4.4. Figure 6.25 (*A*) demonstrates that 78% of the parallel program users found the new designs (first interactive prototype designs) to be easily understood while 22% found both the new and the original design easy. All users found the first interactive prototype designs to be easy; some felt that both the original and the first interactive design were easy to understand. Moreover, all the users liked the physical layout (for example, nodes and network links) and how the first interactive visualisations worked, while none of the users explicitly showed interest in only the original design. The parallel program users who tested the visualisation designs felt very positive about the first interactive prototype design and some of the users stated the following about the physical layout of the new designs: "Yes, they look nice", "Yes, for the star and tree", and "Yes, the physical layout is good".

It is clear that the first interactive prototype design has good visual properties (for example, colours, sizes and shapes) of the nodes and network lines that attract the users when visualising the simulation of the parallel programs executed on the computer. Figure 6.25 (*A*) further shows that the users can easily understand the first interactive visualisation design used to analyse the performance of the parallel program, the NAS Parallel Benchmark, executed on the computational systems.

Most notably, none of the users found the original design to be easier than the first interactive design, as shown in Figure 6.25 (*A*). Clearly users were unable to easily understand the original design and this could be attributed to the poor structure of the original design, many network lines moving over the nodes; small size of the nodes; unclear text (italic font style) within the nodes; and many colours representing almost the same execution statuses of the objects processed during the run. For more information about factors that attributed to the poor visualisation design of the original callgraph system, see Sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4.

The users also found the first interactive prototype design to be easily understandable in terms of analysing and optimising parallel programs, as shown in Figure 6.25 (*A*). The understanding is made easy due to many factors, such as clear details (name and execution time) of the nodes; different shapes and sizes representing the execution statuses of the objects; limiting colours of the nodes to three, which clearly emphasises the execution statuses of the objects; grey network

links, which clearly show the relationship between the nodes; the use of the mouse to quickly obtain details of the node and light steel blue borders of the nodes, which compliments the grey network lines used to showcase the relationship between objects simulated during the execution of the parallel program.

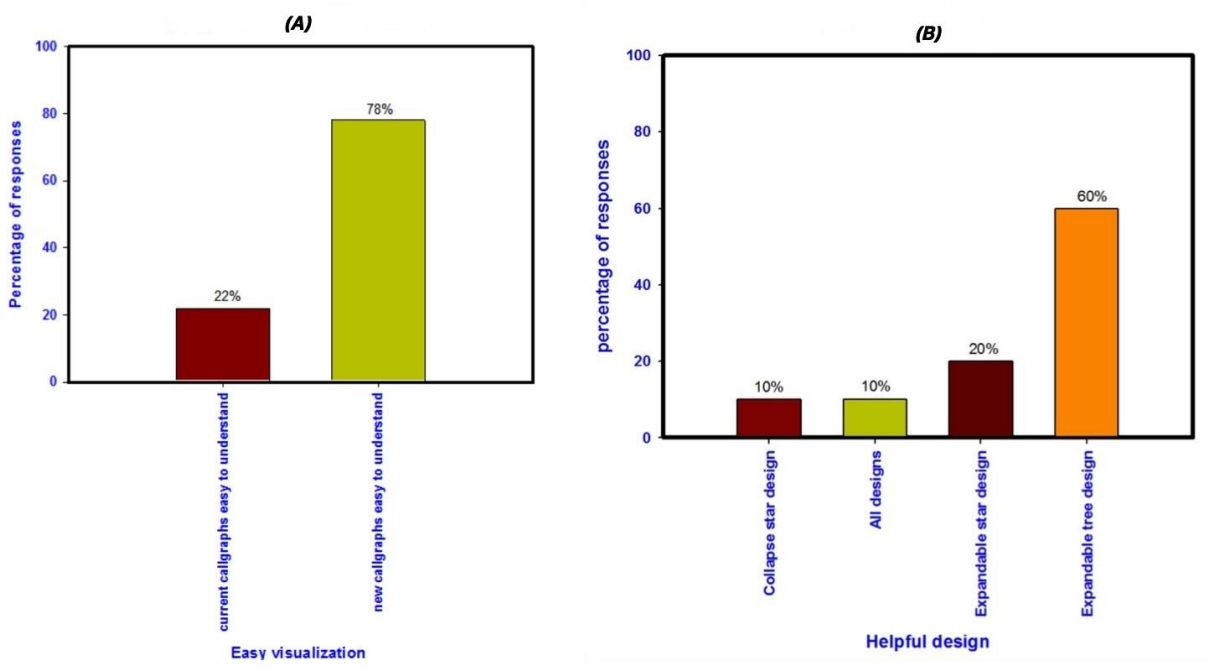

Figure 6.25: Easy of use (A) and helpful (B) design

Most notably, users also found all the first interactive prototype designs to be helpful in terms of analysing and optimising parallel programs. Figure 6.25 (*B*) reflect users' feedback regarding which design (the original or first interactive design) was helpful when analysing and optimising parallel programs. It shows that 60% of the users' responses found the new expandable tree design to be more helpful than all the designs, while 20% of the feedback shows that the expandable star design is also useful. At least 10% of the users' feedback indicates that the new collapsible star design is meaningful; another 10% indicate that all the designs (new and original) are helpful. This means that 90% of the users' responses indicate that the first interactive designs (expandable tree, expandable star and collapsible star visualisation) are helpful while only 10% of the users' views show that both first interactive designs and original are thought to be useful. In fact, all the users indicated that the new expandable tree design was helpful even though other designs are also beneficial for optimising parallel programs. As previously discussed in Section 6.3.1, the expandable tree visualisation

design visualises the performance data of parallel programs using its intelligent features, such as expanding and disbanding the nodes, which users found to be more helpful in terms of filtering and searching for specific sets of nodes when analysing large data sets. The expandable tree design also has the unique feature that automatically minimises or maximises the size of the nodes depending on the amount of information displayed on the screen. All the network links clearly point to the nodes without moving over the details of the object, which is one of the reasons the users find this expandable tree design more beneficial than others.

Below are the users' requirements regarding the first interactive callgraph design, as illustrated in Figure 6.26 (*A*). During the questionnaires, users were tasked with making suggestions and recommendations about the new visualisation designs. The below analysis (Figure 6.26 *A*) indicate that 45% of the users responded that they required the final design of the callgraph visualisations to display a percentage of execution time for each object, while 33% required vulnerable nodes (objects that consumed excessive execution time) to be scaled, based on the execution time.

The users further recommended a search tool to highlight specific objects and another feature to display total execution time on a legend, in order to explicitly state the total simulation time of the program. All users' recommendations are relevant and will be used by us to improve the expandable tree design, which will be followed in designing the second high fidelity prototype design of the new callgraph visualisation. The expandable tree design will be used to develop the second high fidelity prototype design of the callgraph system because users found it easy and helpful in terms of analysing parallel programs, as shown in Figure 6.25.

The basic principles (for example, information overload, and design for errors) of designing a useful, informative and interactive tool were applied to the first high fidelity prototype visualisation design. Most of these design principles can be found in Section 4.1 and will be used to further develop the second high fidelity prototype design. The purpose of this study is developing an effective visualisation that enables users to easily perform visual queries when analysing and optimising parallel programs. As discussed in Section 6.4.4, users performed different visual queries and thus identified queries that are important when analysing, optimising, developing and parallelising applications running on the supercomputers.

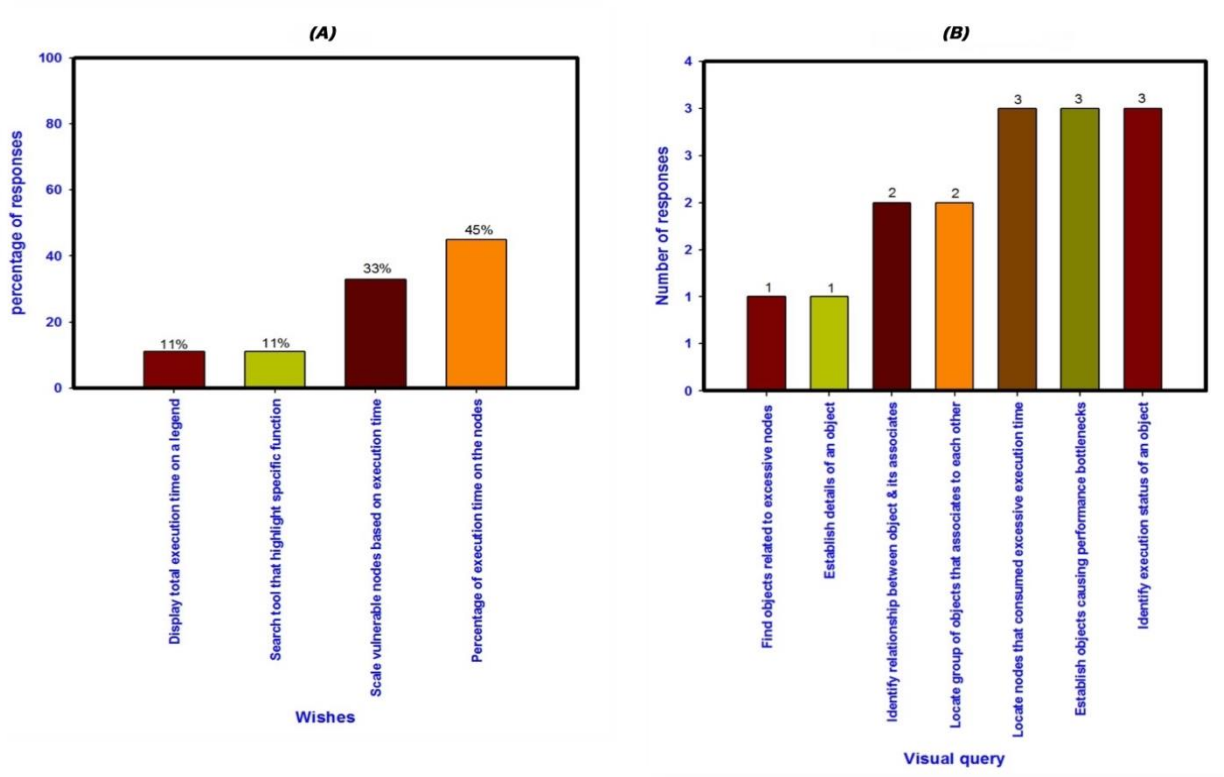

Figure 6.26: Users wishes (A) and important queries (B) on the new design

This process of enabling users to identify the most important visual queries was designed as a means of reviewing and reassessing the visual queries that were earlier identified by us (Section 5.1). During early investigation, we presumed that users performed visual queries (as shown in Table 3.1) when using the callgraph visualisation. The data gathered during the questionnaires was analysed, and it reaffirmed the visual queries (as indicated in Table 3.1) as users' requirements. In particular, users confirmed in the questionnaires that all of the visual queries (as shown on Table 3.1) are the most important queries performed by callgraph users when analysing, parallelising and optimising parallel programs, as illustrated in Figure 6.26 (*B*).

The important visual queries (Figure 6.26 (*B)*) were determined and established by the users during the usability tests of the original and first interactive prototype designs. Firstly, users demonstrated that it is important to establish details of an object, as indicated in Figure 6.26

(*B*). This means needing to identify the name and execution time of the object executed on a particular thread - a requirement that is similar to the visual queries previously identified: what is the name of an object, and how long does an object take to execute on a single processing thread? Figure 6.26 (*B)* also illustrate that users need to identify the relationship between an object and its associates, which again matches our query: what is the relationship between objects that have kinship to each other? Users further recognised another vital visual query - that of locating groups of objects that are associated with each other. This query is the same as our visual query, which is as follows: how can one identify groups of objects that are associated with each other? It is also noticeable that users need to locate nodes that consumed excessive execution time. The query of identifying excessive nodes is the same as our question: which objects consumed normal, moderate or excessive execution time?

Additionally, users need to establish objects causing a performance bottleneck - it matches our visual query earlier described: what is the cause of the performance bottleneck in the application? It was also found that users would like to identify an object's status, echoing our visual query: which objects consumed normal, moderate or excessive execution time? In summary, the visual queries identified by the users are similar to the ones established by us. Lastly, it was discovered that users also required the ability to find objects related to excessive execution time, which is the only visual query that was not exactly identified by us.

This users' query is similar to the visual query of identifying objects causing performance bottlenecks. For example, when the users needs to know which objects cause performance bottlenecks, it is obvious that the person will have to identify which nodes are related to the excessive nodes in order to understand what caused setbacks during the execution of the parallel program. In other words, the visual query - locating nodes related to excessive nodes - is one of the processes taken to establish objects that cause performance bottlenecks.

The results Figure 6.26 (*B*) confirm that the visual queries originally identified by us remain the most critical queries performed by users when analysing, optimising, parallelising and developing parallel programs. It is a given that users will have different requirements, depending on what they wish to achieve on the system. Figure 6.26 (*B*) further illustrates that most of users need to perform the following three visual queries: identify execution status (normal, moderate and excessive) of an object; establish objects causing performance

bottlenecks; and identify nodes that consumed excessive execution time. The three visual queries ranked at 20% importance when considering the users' visual queries. Users who analyse, optimise and develop parallel programs need to perform these three important visual queries - evidenced by a 60% response-rate on the questionnaires. Nonetheless, 13% of the users responded that the visual query of identifying groups of objects that are associated with each other remains important, while another 13% demonstrates that users need to identify the relationship between an object and its associates. The visual query of establishing the details of an object attained a 7% response rate; the same applied to finding objects related to excessive nodes.

All the visual queries (Figure 6.26 *B*) identified by the users are important, irrespective of their percentage value. Most importantly, users performed visual queries similar to these (Figure 6.26 *B*) when analysing the performance of parallel programs using both the original and first interactive prototype design of the callgraphs. It is essential to understand how users interact with the original and first interactive prototype visualisations in their attempts to analyse the performance of parallel programs.

In this case, the qualitative method of analysing non-numerical data is applied to assess video recordings and photographs, used to capture the interaction between users and the visualisation designs of the callgraph system. . It was observed during video a capture that some of the Users 1 and 2 found the original design demotivating and tedious when performing analysis of parallel programs. In fact, User 2 reported that four colours (dark blue, light blue, light green and green) of the original design are distracting, which consequently forced the user to have to read the screen up close in an attempt to obtain details (names and execution time) of the nodes.

User 2 also found it hard to obtain the details of the nodes even though he was leaning close in to the screen which was showing the visualisation results of the original callgraphs. User 1 displayed little interest in using the original visualisation design. Instead, he seemed unimpressed with the interface of the original callgraphs. User 1 later gave up performing visual queries on the original design, and simply observed his colleague (User 2) instead, as shown in the photograph (Figure 6.27) depicts two users attempting to analyse the performance of parallel programs using the original callgraphs displayed on the computers. The user on the left (User 1) is clearly not motivated to perform the visual queries, while the user on the right

(User 2) found it hard to obtain details of the nodes visualised using the original design of the callgraphs. Most of the users found the original system to have little interaction especially when visualising large data. For example, User 7 found it complicated to establish the relationship between objects - instead, he suggested that the original system should have a magnifying feature in order to quickly obtain the details of the node. In addition, User 3 also found it difficult to perform visual queries on the original design when it presented too many nodes. Consequently, this user was forced to lean in close to the screen because it was not easy to locate the names of the nodes.

The interaction between User 3 and the original design of the callgraph system was recorded on video. All the users found the first interactive prototype visualisations more interactive, effective and informative because it was easy to perform all visual queries when using it. All users managed to perform the rest of the visual queries barring one when utilising the first interactive visualisation designs. This is a clear indication that users were able to easily perform visual queries utilising the first interactive designs of the callgraph visualisation. During usability tests, video captured a moment where User 4 requested that User 6 provide the name of the main node displayed on the first interactive visualisation design.

As a result, User 6 hovered the mouse over the main node and the system displayed the name "APPLU", which then conveyed to User 4. It was a moment that demonstrated that the features of the first interactive design helped users to quickly perform visual queries in order to analyse and optimise parallel programs. During usability tests most of the users spent 10-15 minutes performing visual queries on the first interactive design, while it took more than an hour to perform visual queries on the original design. The original design does not have necessary features (for example a magnifier or filter) to help users quickly obtain performance information about the object executed during the simulation of the parallel program.

This is one of the reasons users spend an undesirable amount of time when performing visual queries on the original visualisation design. Figure 6.28 illustrates users performing visual queries on both the original and first interactive visualisation designs. The image shows three users on the left side of the picture and two users on the right; the picture was taken when users on the left side performed visual queries on the first interactive design, while users on the right executed queries on the original design.

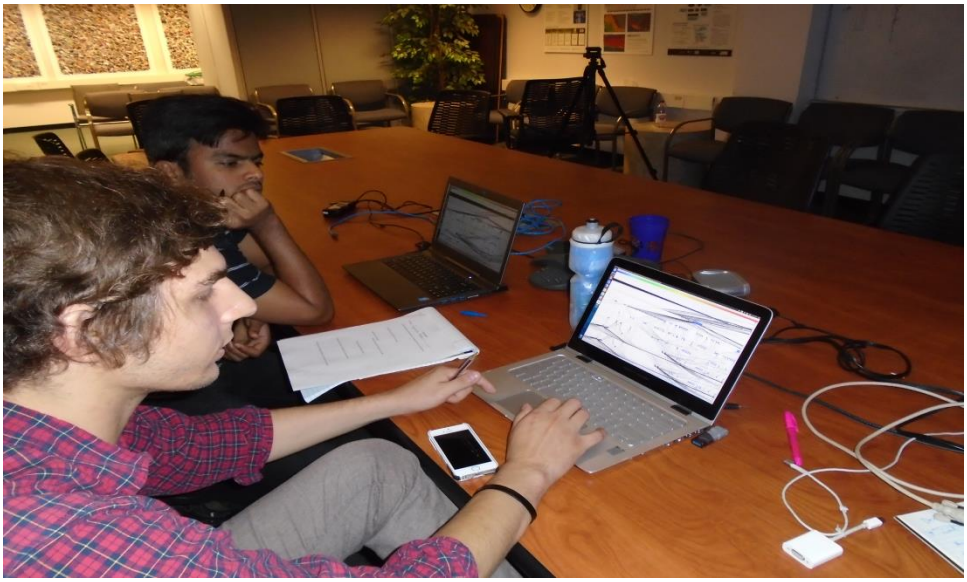

Figure 6.27: Users working on the original design

It was not an arranged set up, but it so happened that some users finished performing visual queries before others. It is noticeable on the picture (Figure 6.28) that users on the left easily completed answers to the visual queries performed on the first interactive prototype designs while those on the right seemed to struggle to obtain answers to the visual queries using the original design. It is evident that the first interactive prototype designs have the capacity to easily and quickly interact with the users when they are analysing the performance of parallel programs being executed on the supercomputers. Voice recordings and notes were assessed to further analyse how users interacted with designs. Users' desires and recommendations raised during the evaluation of the original and the first high-fidelity prototype design were applied to the second high-fidelity prototype design of the callgraph visualisations. It was recommended by the users that the new callgraph visualisations should scale vulnerable nodes based on execution time; display a percentage of execution time on the nodes; show total execution time on a legend, and provide a search tool that highlights specific nodes. Moreover, users found the expandable tree design to be more effective than other designs (non-expandable tree, expandable star and tree) in terms of analysing the performance of the parallel programs. Consequently, the expandable tree visualisation design was followed to develop the second high-fidelity prototypes. Some of the appealing features from other designs (non-expandable tree and collapsible star) were also added into the second high-fidelity prototype design. Most importantly, all recommended features were developed and added into the second interactive

design of the callgraph performance analysis system. This second interactive prototype design consists of two different modes - filtering, and search.

Figure 6.28: Users working on the original and the first high-fidelity design

The filtering mode is a combination of expandable tree and non-expandable tree design, whereby nodes expand/disband or display all at the same time. The search mode is a combination of expandable tree and collapsible star design, whereby nodes are presented in a tree visualisation method but collapse when one clicks an active node. The second interactive prototype design (filtering and search mode) is similar to the final design except that the search tool has been improved in the final design of the callgraph profiling system. For more information about the final visualisation design (filtering and search mode), please see Chapter 7.

6.5 Experts evaluation

The University of Tennessee computer science experts (doctoral graduates and undergraduates) evaluated the efficacy of both the original and the second high-fidelity prototype design of the callgraph visualisation. Both the original and the second interactive callgraph design were tested by four computer science experts who have experience in optimising, parallelising, debugging, analysing and developing parallel programs. In particular, experts evaluated the

conceptual and physical designs of the original/second interactive design by visualising large performance data (metrics) generated using the WRF on the CHPC's Lengau cluster, as discussed in Section 7.4. Both the original and the second interactive designs were used to visualise the performance of WRF-3.5 because it contains more than 2000 objects (modules, routines and subroutines) compared to the NAS parallel program which has sixty-nine objects used to simulate the program.

The evaluation of the designs (original and second interactive design) was performed in a controlled laboratory environment, where only assessors were allowed to enter the room. Each expert was equipped with a computer running Java 8 on a Linux or Mac operating system. To perform tests on the visualisation designs, we have provided a description of how the original and the second interactive callgraph design worked. Consequently, experts were split into two groups. Group A performed visual queries on the original design, while the Group B analysed the performance of the WRF using the second interactive visualisation design.

Afterwards, the groups swapped so that Group A performed visual queries utilising the second interactive design while Group B executed visual queries on the original design. The experts performed the visual queries (as shown in Table 4.5) on both the original and the second high-fidelity prototype design of the callgraph profiling system. During the tests, experts' reactions on both the original and the second interactive design were recorded using video, audio and photographs. The amount of time each user (expert) took to perform all visual queries on the designs was also recorded in order to measure human performance on the tasks. Notes were also taken on how the experts used the callgraphs.

After performing these practical exercises on both the original and the second interactive design, experts were interviewed and asked the questions about the designs. Users were interviewed and asked questions that are described in Table 4.6. The interviews were also recorded using audio. All the experts signed the University of Cape Town (UCT) voluntary form to confirm that they willingly participated in the evaluation of the original and the second interactive callgraph visualisations.

6.6 Analysis of the evaluation

The following data - visual queries, questionnaires, voice recordings, videos, photographs, and notes - gathered during the evaluation of the original and second interactive visualisation design, will be analysed using qualitative analysis method - a technique used to analyse non-numerical data, as stated in Section 2.6. Firstly, experts performed various visual queries using the original design and wrote responses on the answer sheet provided during the evaluation of the visualisation designs. It was found that three evaluators (experts) did not manage to correctly answer all visual queries using the original design, barring one expert who was able to positively provide answers to some queries when utilising the original callgraph visualisations.

The queries performed by experts were as follows, identify the nodes that use excessive time; establish the execution status of MODULE_WRF_TOP::WRF_DFI; name the full details of MODULE_WRF_TOP::WRF_RUN; identify a node that calls MODULE_WRF_TOP::WRF_RUN; indicate time consumed by MODULE_WRF_TOP_INIT; establish the relationship between MODULE_WRF_TOP::INIT and its children nodes; identify that MODULE_INTEGRATE::INTEGRATE has children nodes; indicate the number of children nodes associated with MODULE_INTEGRATE::INTEGRATE; illustrate how to collapse or expand nodes; indicate how to search a node and its associates; show how to return back to main screen; list objects that cause a performance bottleneck during the simulation of the WRF model executed on the CHPC's Lengau cluster. It is clearly hard for the experts to perform visual queries using the original design when it visualises large performance data, as discussed in Chapter 1. The results of the expert evaluations (Section 6.6) indicate that the original callgraph visualisation does not have the ability to efficiently analyse the performance of parallel programs - the purpose of the callgraph visualisations.

During evaluation, most of the experts found it difficult to obtain details (name, execution time and status) of the nodes which were hidden underneath a pile of network links moving over the objects executed during the simulation. For instance, when Expert 1 was asked the following visual query: which nodes use excessive time, he answered : "hard to locate because of too many nodes". Expert 3 also found it difficult to find nodes that consumed excessive time. In this case, Expert 3's answer was as follows: "could not tell". Furthermore, Expert 1 and 3 also could not perform visual queries that would provide answers regarding the execution status of

the nodes. During evaluation, Expert 1 was asked the following query: what is the execution status of MODULE_WRF_TOP::WRF_DFI? Expert 1's answer to this was: "not easy to get the name because of too many nodes". Expert 3's answer to this query was "unknown". The evaluation results of the original design clearly indicate that the experts found it complicated to obtain details (for example, name) of the nodes because of too many nodes displayed using many different colours and sizes. The visualisation technique of filtering the nodes should help to limit the amount of information displayed on the screen, however, original callgraphs displays all the nodes at the same time

The original callgraph further utilises many different colours and sizes of nodes which does not help the user obtain useful performance information about the execution of the program. Most notably, experts could not identify the relationship between the nodes using the original callgraph visualisation. The evaluators were asked the following query: which node (object) calls MODULE_WRF_TOP_INIT? One of the experts gave the answer: "difficult to track"; another said "unknown". A further query that was asked was: what is the relationship between MODULE_WRF_TOP:INIT and its associates? Experts gave answers such as: "can't figure out" and "unknown". This is a sign that the original callgraph visualisation does not enable users to easily establish the relationship between the nodes. The connection between nodes is vital in order to understand how objects relate to one another.

One of the key elements of the callgraph visualisation is to showcase the relationship between the nodes in order to emphasise the relationship between the objects that relate to one another. For example, which module calls routines? Which routine calls subroutines? Which function calls sub-functions? It is difficult for callgraph users to optimise programs using the original design which does not efficiently indicate the relationship between objects simulated during the run.

During the evaluation of the original design, experts were asked to provide answers to the following queries: which node, between MODULE_INTEGRATE::INTEGRATE and WRF_DEBUG, has children nodes? Expert 1's answer was "very closely spaced so hard to make out", while Expert 3 said "unknown". Other experts also showed a lack of interest in finding information using the original visualisation design because the system is not user-friendly and informative about execution of the program in a supercomputer. It is evident that

the original design of the callgraph does not have the capacity to visualise large performance data (metrics) generated during execution of the programs. In fact, visual properties such as colour, size and texture of the nodes are distracting, as articulated in Section 3.2. Consequently, the original callgraphs do not help the user obtain useful performance information for optimisation of parallel programs.

The most notable problem is also the network links, which moves over the nodes and, as a result, hide the details of the object simulated during the execution of the program. The experts (evaluators) were unable to efficiently identify performance bottlenecks incurred during the simulation of the WRF parallel program using the original callgraph visualisation due to inappropriate colours, sizes and textures of the nodes. Many network links that conceal node details (name, execution time and status) also played an undesirable role in inefficient use of the original callgraph visualisation by the evaluators.

Nevertheless, Expert 1 managed to positively answer the following queries: how does one expand or collapse the nodes; how does one search and display a node with its associates. The former query was answered with the word: “click”. This is the correct answer because the original design allows the user to click and manually expand the node in order to obtain the details of an object executed during the run. Moreover, when Expert 1 was asked how to search and display a node with its associates, his answer was: “not possible here”. Again, this is a correct answer because the original callgraph visualisations do not have a feature for searching a particular node and displaying its associates.

In the original design, all the nodes are displayed at the same time within the callgraph visualisation. Surprisingly, Expert 4 gave the impression that the original callgraph visualisations have a search tool, which is not the case. Expert 4’s answer to the query about the search feature on the original design was as follows: “Enter your search query in the search box above the graph”. This response is erroneous because the original design does not have a search box within the callgraph visualisation. The search textbox feature is only available in the second interactive visualisation design of the callgraphs, as shown in Figures 7.2, 7.4 and 7.6. Expert 4 evaluated the second interactive prototype visualisation design before assessing the original design. It was found that Expert 4 used the information obtained using the second interactive design to answer some queries in the original visualisation design. He was the only

expert who performed this error. The exercise of analysing data (visual queries, interviews, videos, photographs and notes) was vigorously performed to ensure that errors such as repeating answers could be easily identified, and addressed accordingly. Visual queries performed by the expert (evaluators) on the second interactive (high-fidelity) prototype design were summarised and examined using qualitative analysis method.

It was found that three experts managed to successfully perform all visual queries using the second interactive design (filtering and search mode) of the callgraph visualisation. Only one expert did not manage to positively answer all visual queries using the second interactive visualisation design. The same visual queries performed on the original design were also executed using the second interactive prototype design. In fact, three experts successfully performed the following visual queries: identify nodes that use excessive time; establish the execution status of MODULE_WRF_TOP::WRF_DFI; name full details of MODULE_WRF_TOP::WRF_RUN; identify a node that calls MODULE_WRF_TOP::WRF_RUN; indicate time consumed by MODULE_WRF_TOP_INIT; establish the relationship between MODULE_WRF_TOP::INIT and its children nodes; identify that MODULE_INTEGRATE::INTEGRATE has children nodes; indicate number of children nodes associated with MODULE_INTEGRATE::INTEGRATE; illustrate how to collapse or expand nodes; indicate how to search a node and its associates; indicate how to return back to main screen; and list objects that cause performance bottlenecks. Most interestingly, all experts managed to correctly name the full details of MODULE_WRF_TOP::WRF_RUN using the second interactive visualisation design. In response to this visual query, Expert 2 wrote the following answer: [{module_wrf_top.f90} {116,4}-{120,25}] Time: 79 Percentage: 89.7%. In the second interactive design, one need to simply point the mouse on the node in order to obtain full details of the object executed during the run.

In this case, it is clear that Expert 2 hovered over the node to obtain full details of the module computed during the execution of the WRF on the CHPC's Lengau Petaflop cluster. It is essential for callgraph visualisation users to obtain full details (name, execution time and percentage) of the node in order to successfully analyse and optimise parallel programs. For more information about percentage of the node, please refer to Section 7.2. It is also important for users to understand the relationship between the nodes. All experts managed to indicate the node that calls MODULE_WRF_TOP::WRF_RUN using the second interactive visualisation

design. For example, evaluators (Expert 2 and 4) wrote: “wrf” when asked which node (object) calls MODULE_WRF_TOP::WRF_RUN. The answer is correct because the object “wrf” is the main module, which calls MODULE_WRF_TOP::WRF_RUN within the WRF parallel program. In the second interactive design, the relationship between the nodes is clearly shown using grey network lines connected to the nodes which have steel blue borders. Different types (white circles, orange ovals and red squares) of nodes also help the user distinguish between objects simulated during the execution of the parallel program.

At the same time, one needs to know the exact time consumed by each object during execution of the program. This notion of obtaining the execution time of the node was tested by asking experts the following query: what is the amount of time consumed by MODULE_WRF_TOP_INIT? All experts managed to correctly state the execution time of this MODULE_WRF_TOP_INIT using the second interactive prototype visualisation design. For instance, Experts 1 and 2 wrote: “9 min”, which is correct and was established by simply pointing the mouse over the node. Expert 2 also wrote the percentage of the total execution time consumed by MODULE_WRF_TOP_INIT. The feedback from Expert 2 is an indication that callgraph users notice the importance of percentages when analysing the performance of parallel programs utilising the second interactive visualisation design.

The percentage of the total execution time helps the visualisation user to compare the amount of time consumed by each object during the simulation of the parallel program. It is also noticeable that the second visualisation design is able to adequately provide details of the objects executed during the run. Furthermore, callgraph visualisation users are able to establish the relationship between nodes. During evaluation of the second high-fidelity design, all four experts successfully established the relationship between MODULE_WRF_TOP_INIT and its associates. The experts were asked the following visual query: what is the relationship between MODULE_WRF_TOP_INIT and its associates?

All experts provided the same answer: “siblings”, which is correct because MODULE_WRF_TOP_INIT objects have a relationship with siblings, or closest children nodes. In the second interactive design, the relationship between a node and its children nodes is explicitly displayed using the grey network links with a white background colour. The nodes that have large borders indicate a node that is clickable, and which has children nodes. The

above indicates that all experts were able to notice the difference between non-clickable nodes with small borders, and nodes (large borders) that have siblings. The size of the nodes also plays a very important role in helping callgraph users easily identify the relationship between objects simulated during the run. In the second interactive prototype, red square nodes are bigger than orange oval nodes; which, in turn, are bigger than white circle nodes. The size of these nodes (red squares, orange ovals and white circles) helps callgraph visualisation users to establish nodes that are critical and that need attention when optimising the performance of the parallel programs. In the second interactive design, nodes with small border do not have children nodes, while nodes with large borders contain sibling nodes.

The different sizes of the nodes' borders help identify the relationship between objects that are associated with each other. To this end, experts were asked the following visual query: which node, between MODULE_INTEGRATE::INTEGRATE WRF_DEBUG, has children nodes? All experts confirmed that MODULE_INTEGRATE::INTEGRATE was an object with children nodes - the correct answer because the MODULE_INTEGRATE::INTEGRATE node has a large border, while the WRF_DEBUG node is surrounded by a small border. It is remarkable that all evaluators (experts) were able to notice the difference between the nodes that have siblings and ones that do not have children nodes. It is less than ideal to display all the children nodes at the same time using the original design of the callgraph visualisation, as discussed in Section 3.2.

During the evaluation of the second interactive design, experts were asked to count and outline the number of children nodes that were associated with MODULE_INTEGRATE::INTEGRATE; all experts managed to successfully report the number of children nodes when using second interactive prototype design of the callgraph profiling system. In particular, experts were asked the following visual query: if any of the below-mentioned nodes (MODULE_INTEGRATE::INTEGRATE and WRF_DEBUG) have children nodes, how many children nodes are there? All the evaluators indicated that MODULE_INTEGRATE::INTEGRATE nodes have twenty children nodes - a correct answer. Most notably, Expert 4 further stated that the WRF_DEBUG has 0 children nodes - also a correct answer because this node does not have a large border. It is essential to obtain the details of the children nodes when using the second interactive prototype design to analyse performance of the parallel programs. Experts (assessors) were next asked the following visual

query: how to expand or collapse the nodes? All experts managed to positively answer this query. In summary, Expert 1 and 2 said “click”; Expert 3 said “click to expand”; Expert 4 said “by clicking on it”. It is evident that experts were able to expand or collapse the children nodes by clicking the target node within the second interactive design of the callgraph visualisation. Most interestingly, experts (evaluators) were able to differentiate between non-clickable nodes without children nodes and clickable nodes with siblings. It is the size of the nodes that differentiate between a clickable node and a non-clickable node within the second interactive visualisation design.

Best principles were followed (see Section 4.1) with regard to designing an effective and interactive visualisation when developing the second interactive callgraph visualisation design. The most notable principle of the design is to ensure that important controls (for example, nodes) are visible within the system. In this case, grey network lines enable the steel blue borders of the nodes and children nodes to be more visible within the second interactive design of the callgraph profiling system. Nonetheless, experts (evaluators) were asked to indicate how to search a node and its associates. All experts managed to provide a positive answer regarding the process of searching a node and its associates. In fact, the experts were asked the following visual query: how to search and display a node with its associates?

Expert 1’s answer to this query was: “enter node name tab” – a correct response because one needs to enter a node name in the textbox and click the “search” button in order to search a node and its associates. Expert 2’s answer was: “enter the full name and click search” - also a correct answer. Expert 3’s answer was recorded as follows: “use search box”, which is an accurate answer to this query. Lastly, Expert 4’s response was articulated as follows: “type in the search box on top” - another correct response because one needs to type the full name of the node in a textbox on top of the visualisation graph. In summary, all experts positively answered the visual query on how to search and display a node with its associates.

Most interestingly, three out of four experts used the second interactive design to list objects that caused performance bottlenecks during the simulation of the WRF parallel program executed on the CHPC’s Lengau Petaflop supercomputer. These three experts also utilised the second interactive visualisation design to positively identify nodes that use excessive execution time during the execution of the WRF parallel program. It is useful for the second interactive

design to have visualisation techniques that are able to indicate parts of the program that both caused performance bottlenecks and consumed immoderate execution time during execution of the program. One evaluator (Expert 3) was not able to correctly answer the following query: what is the name of the object (node) that causes performance bottlenecks and which nodes use excessive time? Expert 3's answer was as follows: "MODULE_RA_RRTM::RRTM" - incorrect because this module consumed normal execution time during the simulation of the WRF model on the CHPC Petaflop cluster. The objects that caused performance bottlenecks are as follows: MODULE_WRF_TOP::WRF_RUN, MODULE_INTEGRATE::INTEGRATE, SOLVE_INTERFACE and SOLVE_EM.

The second interactive design clearly displays the objects that consumed excessive execution time, using red square nodes within the visualisation. It is believed that Expert 3 misunderstood the question. Nevertheless, the majority of the evaluators used the second prototype visualisation design to establish both objects that caused performance bottlenecks and nodes that utilised exorbitant execution time during execution of the program. The second interactive visualisation design is able to efficiently identify performance bottlenecks incurred during the simulation of the parallel program, which is the main objective of this study - as discussed in Section 1.3 (objectives).

It is also evident that the second interactive visualisation design is able to effectively visualise large performance data generated during the execution of the parallel program. As discussed in Section 6.5, experts were interviewed and asked questions about both the original and the second interactive design of the callgraph profiling system. The first question to the experts was as follows: how helpful is the visualisation design in terms of optimising programs? Expert 1's response regarding the original design was as follows: "current design - very cumbersome". Expert 1's feedback about the second interactive design was: "visual tracking of the program flow is easy". Please note that most of the experts used the phrase "new design" when referring to the second interactive design; "current design" refers to the original design.

Expert 1 found the second visualisation design to be useful in terms of optimising parallel programs. This same expert found the original design difficult to use. Expert 2's response to the first question was as follows: "old one is more complicated to get the info" and "new design - help analysing algorithms". It is clear that Expert 2 also found the original design to be

complicated in obtaining information when using the callgraph visualisation, but described the second high-fidelity prototype design as one that helps in the analysis of algorithms when optimising the performance of parallel programs. Expert 3's response to the first question was the following: "old design not for finding hotspot" and "new design - easier to find hotspots". Expert 3 found the second interactive design to be helpful in terms of finding hotspots. The second interactive visualisation design contains red square nodes, which helps the user find hotspots that cause performance bottlenecks within the parallel program. It is important for callgraph visualisations to clearly show parts of the program that cause performance bottlenecks in order to help the user optimise the parallel program.

Expert 4's answer to the first question was as follows: "the new design is helpful because it shows where the bottlenecks are and where the time is spent" and "it was hard to read the graph and node names using original design". On this front, Expert 4 found the second interactive design to be helpful because it led the user to where the bottlenecks are located within the parallel program. This expert found it hard to read the names of the nodes when using the original design of the callgraph visualisation. During evaluation of the original design, most of the experts found it hard to locate the details (name, execution time and status) of the objects due to many network links moving over the nodes within the visualisation.

It is expected that the callgraph visualisation should effectively demonstrate the relationship between the nodes; however, this is not the case because the details of the nodes are hidden underneath the network lines within the original design. During the interview, the experts were asked the following (second) question: which visual queries were easy to perform? During the interview, all four experts provided feedback regarding this question. Expert 1's response was as follows: "new design - search and track" and "nothing in old design was easy". In this situation, Expert 1 found the second interactive visualisation (search mode) to be easy when searching for the node and establishing its relationship with other associates. The second interactive prototype design (search mode) enables the user to enter the name of the node in the textbox and obtain the information about the relationship between the searched node and its associated objects, executed during the simulation of the parallel program. In fact, it was noted that all experts found the search tool within the second interactive visualisation design to be easier when analysing the performance of the parallel program running on the supercomputer. Expert 1 found that no visual query was easy to perform utilising the original design of the

callgraph visualisation. This negative feedback about the original design is not surprising because the system does not have the capacity to visualise large performance data and display effective callgraph reports, as indicated in Figure 3.4 *(A)* and *(B)*. During the interview, Expert 2 provided the following answer regarding the second question: “new - most of them” and “old - hard to deal with it very unresponsive”. In this case, Expert 2 was able to perform most of the visual queries using the second interactive visualisation design. The second interactive prototype design presents information in different modes, whereby it enables the user to collapse or expand the nodes in order to quickly establish the relationship between the objects (parts) of the program simulated on a computational system. Most of the experts found this visualisation technique of collapsing and expanding the nodes to be very useful in terms of enabling visual queries within the visualisation. It is anticipated that most of the experts may have found it easy to perform visual queries using the second interactive design because it allows the user to manipulate information displayed on the screen.

In particular, the second interactive design enables the visualisation user to filter or search particular sets of nodes while also having the capability to expand and disband the tree of the relationship between the nodes. All these sophisticated features of the second interactive design enable the user to easily perform visual queries within the visualisations, which is not the case with the original design. Expert 2 clearly indicates that the original (old) design is hard to use because it is unresponsive. The original design displays all the nodes at the same time, making it slow to respond when manoeuvring different parts of the visualisation. For example, experts took approximately one minute when scrolling from top to bottom of the original visualisation, whereas scroll bars on the second interactive design immediately move from one point to another (for example, top to bottom or left to right) when performing a manoeuvre within the visualisation. The original design responds slowly when it visualises large performance data (metrics), as articulated by Expert 2.

It should also be noted that the original design’s inability to limit information results in all the nodes being displayed on the screen - consequently causing the slow response of the system. Expert 4 found the search tool and tracking of hotspots to be easy when performing visual queries using the second interactive design. In response to the second question, Expert 4 provided the following answer: “new design - search if you know the thing you want and track down to hotspot was easier” and “original design - could not evaluate old design”. In the

second interactive design, the search tool is easier to use because the user only needs to know the full name of the node in order to search for an object and its partners. This is one of the reasons some of the evaluators (Expert 1, 2 and 3) found the search tool to be easier than expanding or disbanding a tree of nodes within the second interactive visualisation. Expanding and disbanding the nodes is also useful when one needs to analyse the entire relationship between the objects simulated during the run. Interestingly, Expert 3 stated that it was easy to track down a hotspot - the parts (objects) that caused performance bottlenecks during the simulation of the WRF parallel program on the CHPC's Lengau cluster. It is of great importance for the experts to easily identify performance bottlenecks within the parallel program as it is the main objective of this research study. Expert 3 found it complicated to evaluate the original design because it was too slow when displaying all the nodes at the same. Expert 4 concurred with all other experts that the search tool is easy-to-use when analysing the performance of parallel programs.

In particular, Expert 4's response to the second question was as follows: "new design - the search feature was very easy" and "original design - it was hard to read the graph and node names". It is not surprising that experts found the search tool of the second interactive design easy-to-use because users need a system that can provide a mechanism to quickly obtain information within the visualisation. Expert 4 reiterated that it was hard to read the graph and obtain the names of nodes when utilising the original callgraph design. During evaluation of the original design, experts could not obtain useful information such as names of the nodes, due to the poor structural design - inadequate size and colours of the nodes, including network links within the visualisation.

The experts were also asked to state which visual queries were difficult to perform when using the original and the second interactive design. In response to this third question, Expert 1 provided the following answer: "original design - finding the execution time, file and relationship", and: "original design - difficult to track the names of the nodes". At the same time, Expert 1 found no difficulty with visual queries when using the second interactive design. It is evident that Expert 1 found it difficult to use the original design in obtaining performance information, such as execution time and the object file executed during the run. It is also clear that Expert 1 could not establish the relationship between the nodes using the original design, an important feature of the callgraphs. The original design failed to efficiently display the

names of the nodes when Expert 1 assessed the usability of the visualisation. It is unfortunately impossible for the user to analyse and optimise the performance of parallel programs without knowledge of the objects within the application. It is of significance for callgraph users to be able to obtain the names of the nodes when optimising the parallel applications running on the supercomputing systems. The original design does not efficiently provide details (name, execution time and status) of the nodes within the callgraph visualisation. During the interview about the original and the second interactive design, Expert 2's response to the third question was as follows: "new design - going back" and "original design - most of the item". Expert 2 found it complicated to go back to the first view of the graph after searching or clicking the nodes using the second interactive design (search mode). Most interestingly, Expert 4 found that going back into the graph was difficult when using the second interactive prototype design (search mode); his response to the third question was as follows: "new design - going back in the graph" and "original design - going through all displayed nodes and finding relationships".

It is also indicative that expert4 struggled to go back to graph when using second interactive design - search mode. The second interactive prototype design is not a full implementation of the callgraph visualisation - hence it will not have a menu that contains backward and forward buttons. The second interactive design (search mode) is designed primarily to search/click a specific node and display its associates, which is the reason it does not disband backwards or expand forwards like the second interactive design (filtering mode) of the visualisation. As for the second interactive design (filtering mode), it enables the user to go back and forth to any view by disbanding/expanding the nodes within the tree visualisations.

The tree's forward/backward feature is available within the second interactive prototype design (filtering) of the callgraph visualisation because it is designed to display and handle many nodes at the same time. A manual will be developed to help guide users on how to operate the final design (search and filtering mode) of the callgraph visualisations. It is of concern that Expert 2 found it difficult to perform most of the visual queries when utilising the original design and the reason for this could be that the original design was not able to positively respond on time, causing the user to lose interest in using the system. One of the factors that impacted on Expert 2's ability to perform most of the visual queries using the original design could be attributed to poor visualisation design features such as distracting node colours, italicised font of the names, and many network lines hiding the details of the node within the

callgraph visualisation. Expert 4 found it difficult to establish the relationship between all the nodes displayed at the same time using the original design. In the original design, one needs to manually expand the node in order to get details of the object, but with many network lines moving over the nodes this makes it more complicated to establish the correlation between different parts of the program. Expert 3 answered the third question with: "new - full name for searches were awkward". Expert 3 found it awkward that one needs to enter the full name of the node to be searched. The D3.js library used to develop the new design only allows the user to search a full string within the performance data (metrics); however, this feature can be improved upon in future as it is not part of the actual design. Moreover, it was found that the other three experts (Expert 1, 2 and 4) found the search tool to be easy to use when analysing large performance data generated during execution of the program.

At the same time, Expert 3 did not comment about visual queries that were difficult to perform on the original design. It can only be assumed that this expert did not want to provide an opinion regarding which visual queries were difficult to perform when using the original design of the callgraph profiling system. Expert 3 managed to provide answers regarding all other visual queries performed on both the original and the second interactive design. Experts were asked the following (fourth) question: which design is more helpful and why? The purpose of the question was to establish which visualisation - the second interactive prototype or the original design - is more helpful in terms of optimising parallel programs.

All four experts found the second interactive visualisation design to be helpful when analysing and optimising parallel programs simulating on the computer. In fact, none of the evaluators (experts) found the original design to be beneficial when optimising the performance of parallel programs on the supercomputer. Expert 1's answer to the fourth question was as follows: "new one, especially search". It means the expert found the second interactive (search mode) prototype design to be helpful when optimising parallel programs. Furthermore, expert1 was fascinated by the second interactive design (search mode) used to visualise large performance data generated during the simulation of the parallel program. As previously stated, three out of four experts found the second interactive visualisation (search design) easy to use, which could be the reason for Expert 1 finding the second interactive design more helpful than the original design. The experts also found the second interactive design (search) to be easier because it enables one to enter the node name and quickly obtain information about certain parts of the

program. Expert 2's answer to the fourth query was the following: "new because it is less complicated and easy to follow". It is noticeable that Expert 2 also found the second interactive visualisation design to be helpful because it is less complicated and easy to follow the tree of the nodes within the visualisation. The second interactive design (filtering) has an intelligent feature which allows the user to expand or disband the nodes at any time. As a result of this feature, users are able to easily follow the tree of nodes by either adding or reducing information when required. Expert 2 found this feature to be more informative and easier to follow the tree of information displayed on the screen using the second interactive design of the callgraph performance analysis system. Expert 4 also believed that the second interactive design was more informative than the original design because it limits the information displayed on the screen. Expert 4's answer to the fourth question was as follows: "the new design is more helpful because it limits information displayed on screen at any given time".

It is indisputable that some of the experts found the second high-fidelity design (filtering mode) to be helpful when analysing the performance for optimisation of parallel programs. The second interactive visualisation design (filtering and search) also has a special feature that automatically reduces or increases the size of the nodes, depending on the amount of information displayed on the screen. This could be the reason the experts found the second high-fidelity prototype design more informative than the original design. It is also clear that the second visualisation design led users to the hotspots where performance bottlenecks were incurred during execution of the program.

During the interview, Expert 3's response to the fourth question was as follows: "new is better for hotspots". In the second interactive prototype design, red square nodes clearly indicate objects that cause performance bottlenecks within the parallel program. Moreover, red square nodes scale differently depending on the amount of execution time, as shown in Figure 7.3. The feature of scaling nodes enables users to easily establish hotspot areas that cause performance bottlenecks during the run.

In the second interactive design, the red colour of the square nodes calls the user to pay more attention to these nodes (objects), which cause performance issues within the parallel application. White circles and orange nodes are not distracting to the user. Experts were asked to respond to the following (fifth) question: "can you suggest any improvements to the design?"

The intention with this query was to establish whether or not any improvements could be made to the second interactive design of the callgraph visualisation. It was found that all experts were satisfied with the second interactive design but would like to see improvements on the search tool used to locate specific sets of nodes. In particular, Expert 1's response regarding improvement of the second interactive design was as follows: "search and zoom". This expert wanted the second interactive design to search and magnify the nodes. The search tool is intended to search and display an active node with its associates – not to magnify the node. In fact, when one searches a node in the second interactive design of the callgraph profiling system, the visualisation patently displays the target node with its associates.

All the details of the nodes are clearly displayed when utilising the search tool to locate specific sets of nodes within the visualisation. In the second interactive prototype design, users do not need to zoom in, in order to display an active node with its partners because the search tool explicitly displays all details of the nodes accordingly. Expert 2's answer to the fifth question was as follows: "search by short name". Expert 2 found that the search tool would be more useful when it allowed the user to type just a few characters of the target node to be searched within the visualisation. It is a relevant suggestion, however D3.js was used to develop the second interactive design and requires that the user enter the full name (string) of the node in order to search within the performance data (metrics).

As earlier indicated, the search tool is not a design but compliments how the second interactive design works. The search tool was developed to allow users to easily locate specific sets of nodes. Expert 3 gave the same response as Expert 2 in that it would be useful to have a search tool that enables the user to enter a short name, rather than the full name, of the target node to be searched within the visualisation. During the interview, Expert 3's feedback regarding the fifth question was as follows: "full names for searches were awkward".

It is recommended that future work should look into simplifying the search tool within the second interactive prototype design of the callgraph visualisation. Lastly, Expert 4's answer to the fifth query was: "include a link to go back to the beginning". This means that Expert 4 suggests that the second interactive design should have buttons to go back and forth through the visualisations. As earlier indicated, the second interactive prototype design is able to move back and forth by expanding or disbanding the nodes within the visualisation. The buttons to

move back and forth should be included in the menu of the full implementation of the callgraph visualisation system, which is not a design and therefore not within the scope of this research work. The experts' voice recordings were analysed and interpreted accordingly and they found the new design to be helpful when analysing the performance of the parallel program - not the case with the original design. During interviews, Expert 1 said the following: "Nothing in old design". This expert found nothing to be easy when using the original design to analyse the performance of parallel programs. Moreover, Expert 1 said that, "it was difficult to track the names of the nodes".

In this case, Expert 1 found it difficult to obtain the names of the nodes when using the original design to evaluate the execution of the parallel program, however, he discovered that the second interactive design was easy to search and track the names of the nodes. During the interview, Expert 1 said: "Search and track"; he found the search tool and tracking of names of the nodes to be easier when using the second interactive prototype design to optimise parallel programs. Expert 2 also found the second interactive prototype design to be helpful when analysing algorithms within the parallel program. During evaluation of the designs, Expert 2 said the following: "It helps analysing the algorithm". On this note, it was evident that the second interactive design helped users analyse different objects simulated during the execution of the parallel programs.

Most importantly, the second interactive prototype design was easy-to-use and informative, however, this was not the case with the original design. Expert 2 said the following statement about the original design: "The old one is more complicated to get info". It is evident that Expert 2 found the original design to be complicated and not useful in terms of providing performance information about the simulation of the parallel program. In fact, another evaluator (Expert 3) found the original design to be unsuitable for identifying performance bottlenecks incurred during the run. During the interview, Expert 3 said the following: "The old design is not appropriate for finding hotspot".

These findings indicated that Expert 3 found the original design to be unfit for finding objects that cause performance bottlenecks within the parallel programs. Some of the evaluators found that the second interactive design helped to locate hotspot objects that consumed excessive execution time during the run. For example, Expert 4 said the following statement about the

second interactive design: “it helps to find bottlenecks and functions that consume lot of time.” The above is an indication that the second interactive design is able to demonstrate objects that consumed immoderate execution time during the simulation of the parallel program. Furthermore, Expert 4 said the following statement about the second interactive prototype design: “it is easier to navigate”, meaning that he found the second interactive design to be easier when navigating from one view to another. The structural designs, such as node sizes and colours within the second interactive visualisation enables easy identification of performance bottlenecks in the parallel programs. The second interactive design also interacts very well with the users when analysing the performance of parallel programs.

During the usability test of the visualisation designs, it was noted that the second interactive prototype design automatically provided the details (name, execution time and percentage) of the object when the users (experts) pointed the mouse over the target node within the visualisation. It was found that the second interactive design of the callgraph visualisation was easy to use, informative and interactive, whereas the original design was complicated to use when analysing the performance of parallel programs. The design of the visualisation plays a significant role in enabling the user to easily utilise the system. For example, different node sizes should help the user identify various execution statuses (normal, moderate and excessive) of objects simulated during the run.

The visualisation users also found the system to be easy as it provided relevant information within a short period of time. It is also essential for the visualisation designer to design using colours that will help the user obtain useful information about the system. For instance, red attracts more attention and can be used to emphasise hotspots or bottlenecks within the parallel program, whereas it would not be wise to use white for this purpose. In fact, other colours can be distracting, which may lead the user to lose interest in utilising the system.

In the original design, it was found that the dark blue colour of the nodes was difficult to focus on, especially when there were too many nodes in the system, as discussed in Section 3.2.1. Nevertheless, the second high-fidelity design of the callgraph visualisation used different colours (white, orange and red) for the nodes, in order to demonstrate the execution statuses for different objects within the parallel application. Most importantly, the red square nodes of the

second high-fidelity design clearly identify different objects that consumed excessive execution time during the simulation of the parallel program.

6.7 Conclusions

The chapter discussed two prototypes visualisation - The first and the second high-fidelity visualisation designs used to identify performance bottlenecks of parallel programs, the NAS Parallel Benchmark and the WRF model, were executed on the computational system. It further demonstrated different visualisation methods (for example, star and tree) used to present and visualise data. Most importantly, different visual properties such as colour, shape and size were used to design easy-to-use, informative, and effective interactive prototype designs. The chapter explicitly revealed an efficient means of visualising the relationship between the nodes using the first and the second interactive prototype designs. It also discussed the methods and procedures used to evaluate the conceptual and physical designs of the first/second callgraph visualisations. Most notably, parallel program users, including experts, evaluated the usability of the original and first/second interactive designs in terms of optimising parallel programs to achieve optimum performance on the supercomputer. It was found that the original callgraph design does not effectively visualise performance metrics (data) of the simulation, whereas the second interactive design is able to effectively and interactively provide useful performance information about the execution of the parallel model. The experts also found the second interactive design to be easy to use and informative when analysing the performance of parallel programs that generate large amounts of performance data. It was noted that the search design should allow users to enter a short name of an object when searching the relationship between the nodes. The improvements on the search design will be discussed in Chapter 7, where the final design of the callgraph visualisation is described.

CHAPTER 7: FINAL DESIGN

This chapter presents the final design of the callgraph visualisation used to identify performance bottlenecks in parallel programs. In particular, it describes the two different visualisation modes - the filtering and search designs of the final callgraph visualisations used to analyse the execution of parallel applications on supercomputers. Most interestingly, the chapter demonstrates how both final designs (filtering and search) visualise different sizes (small and large) performance datasets (metrics) generated during execution of the program.

7.1 Design goals

One of the design goals for the final design of the callgraph visualisations is to enable users to identify an object and its associates. During the evaluation of the first high-fidelity prototype design (Figure 6.22), users recommended the final design of the callgraph visualisation to have the following visualisation features, namely, show the total execution time of the application on a legend, display the percentage of execution time on the nodes, create a search tool that highlights specific nodes and scale vulnerable nodes based on execution time. During evaluation of the second high-fidelity prototype design, users further recommended that the search tool within the final design should use the short name of an object when searching the relationship between the objects computed during the run. Sedlmair, Meyer and Munzner (2012) state that a visualisation reseacher needs to outline lessons learned in applying principles of visualisation design. To this end, we describe the lessons experienced during the design of the new callgraph visualisations. During evaluation of the designs (low-fidelity and high-fidelity prototype), we have experienced the following design problems:

- Our design softwares needed excessive time to add different visualisation features.
- Manipulation and visualisation of large data was complicated.
- Some of the users do not strictly follow instructions when evaluating the designs.

Nonetheless, all this design obstacles were addresssed, which resulted with an effective callgraph visualisation for analysing the performance of parallel programs. The final design of the callgraph visualisations will be discussed in the next Section 7.2.

7.2 Design description

As discussed in Chapter 6, users performed practical evaluations of the first and second high-fidelity prototype designs used to analyse the performance of parallel programs. All the users (experts) who assessed the prototypes found the second interactive prototype (filtering and search design) to be easy to use, informative, interactive and effective when analysing application performance for optimisation of parallel programs. The second high-fidelity prototype was adopted in the final design of the callgraph visualisation, however, search tool was improved in the final design. The final design of the callgraph visualisations consists of two different modes - filtering (Figure 7.1 and 7.5) and search design (Figure 7.2 and 7.6), used to visualise performance metrics (data) of the NAS and WRF parallel program executed on the supercomputer.

In the final design, we re-programmed the search tool to process information faster and display the relationship between the objects simulated during the run. However, search tool does not allow a user to enter a short name of the object when searching relationship between the objects simulated during the run. D3.js visualisation library currently does not have the ability to read a few characters of an object and convert it to the full name of an object stored in the performance data. As a result, this makes it difficult to allow users to enter a short name of an object when searching the relationship between the nodes.

However, users are able to enter the full name (not the short name) of an object and retrieve the relationship between the searched object and its associates. The search tool within the final callgraph visualisation is not a design feature, which means it falls outside of the scope of this research study. Nonetheless, all users' recommendation were applied in the final design except one of searching an object using a short name. The next section discuss the features of the final design (filtering and search design) used to analyse the performance of NAS parallel program executed on a computational system.

7.3 Filtering and search design on small performance data

The filtering visualisation design presents an expandable tree design that has the ability to expand or disband the nodes within the system, as shown in Figure 7.1. As for the search

design, it demonstrates an expandable tree design that has the ability to display only an active (searched or clicked) node and its associates, as highlighted in Figure 7.2. One of the user's requirements is to identify an object and its associates, as indicated in Figure 6.26 (*B*). The search visualisation design enables users to select and identify an active node with its related objects. In the search design (Figure 7.2), when a user clicks the node, the system clears the area and displays only the clicked node with its associates. The filtering design enables the user to present the relationship between many different nodes at the same time, as shown in the below, Figure 7.1. In the final design (filtering and search), white circle nodes represent objects that consumed normal execution time, orange oval nodes show parts of the program that consumed moderate execution time, and red square nodes illustrate objects that consumed excessive execution time during execution of the program, as indicated in Figures 7.1, 7.2. 7.3, 7.4, 7.5 and 7.6.

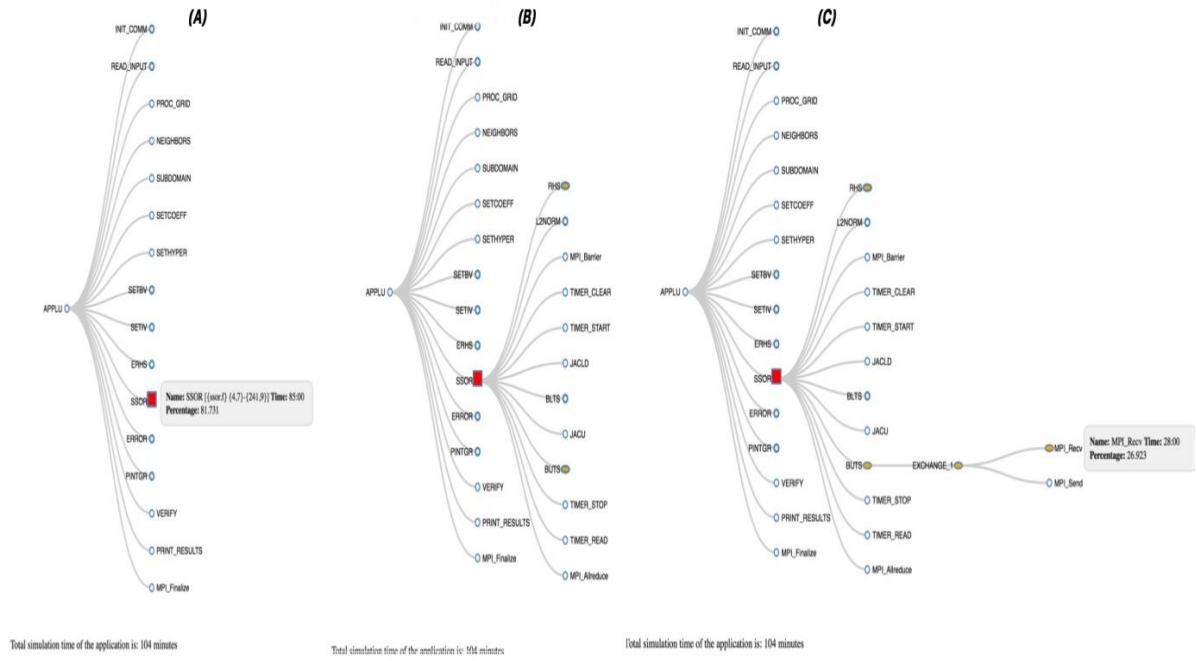

Figure 7.1: Filtering design of the callgraph visualisation system

In both filtering (Figure 7.1) and search (Figure 7.2) visualisations, the relationship between the nodes is presented using grey network lines. All the nodes are surrounded by steel blue borders, which work very well with the grey network lines. In this final design, nodes surrounded by a big steel blue border represent clickable nodes that have children nodes.

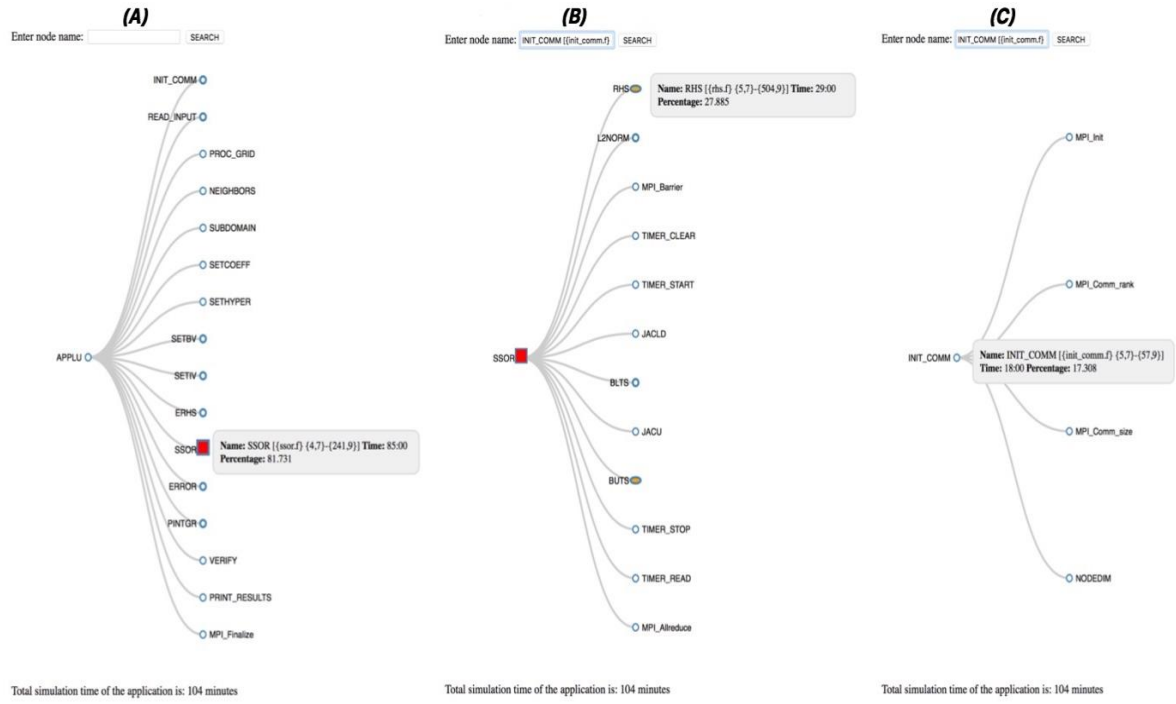

Figure 7.2: Search design of the callgraph visualisation system

The final visualisation design (filtering and search) displays the total execution time of the application at the bottom of the visualisation system, as demonstrated in Figure 7.1 and 7.2. Both search and filtering visualisations further display the percentage of the total execution time of each object when one hovers over a node within the callgraph profiling system. In the final design (Figure 7.1 and 7.2), objects that consumed between 1.00% and 20.00% of the total execution time are displayed as white circle nodes, while objects that consumed between 21% and 40% of the total simulation time are presented as orange oval nodes within the visualisation.

Moreover, parts of the program that consumed between 41.00% and 99.99% of the total execution time are shown as red square nodes. An object that consumed 100.00% of the total simulation time is displayed as white circle node because it is the main module that normally calls all other objects. Figure 7.1 (*A*) shows that an object namely, the *SSOR* node, consumed 81.73% of the total simulation time, which is regarded as excessive execution time of an object. Figure 7.1 (*C*) illustrates that MPI_Recv object consumed 26.92% of the total execution time, which is an average execution time of the node. In fact, users require that both the details and

execution status is shown when analysing the performance of parallel programs, as shown in Figure 6.26 (*B*). In the final design (filtering and search) of the callgraphs, when a user hovers over, or points the mouse at a node the system automatically displays the details (name, execution time and percentage) of that particular node. The details of the node further enable the user to understand how long an object took to execute on a single processing thread.

The final callgraph visualisations clearly indicate the execution status (normal, moderate or excessive) of the node using three different colours and shapes, whereby white circles show normal execution time, while orange ovals indicate average execution time. The red squares represent objects that have consumed immoderate execution time. As earlier discussed in Section 7.2, users recommended that the final design have a search tool that can quickly identify a group of nodes that are associated with each other. To this end, a search design was developed (Figure 7.2), which enabled the user to enter the full name of an object and quickly obtain a performance report of that particular object with its associates.

The first view of the search visualisation design presented the main module and its first children nodes, as depicted in Figure 7.2 (*A*). When the user enters the full name of the node in the textbox, the visualisation will clear the area and display the entered node name with its associates. In Figure 7.2 (*B*), the user entered the full name "*INIT_COMM [{init_comm.f} {5,7}-{57,9}]*" and clicked the "*SEARCH*" button. Consequently, the search visualisation design clears the area and displays only the *INT_COMM* node with its associates, as demonstrated in Figure 7.2 (*C*). The third view (FIGURE 7.2 *C*) of the search design further shows the full name of the searched object, *INIT_COMM*, associated with an amount of execution time and percentage of time consumed during the run.

It is evident that users are able to establish a relationship between a group of nodes that are associated with each other using the search visualisation design, which is one of the users' requirements, discussed in Figure 6.26 (*B*). Most importantly, both filtering and search designs are able to indicate parts of the program that cause performance bottlenecks during execution of the program. Figure 7.1 (filtering) and 7.2 (search) visualisation design shows red nodes (*SSOR* object) that cause performance bottleneck during the simulation of the NAS parallel program on a computational system. The final design (Figure 7.1) further led the user to the exact area of the problem by identifying *Exchange_1* and *MPI_Recv* as the objects that need attention

when optimising the performance of the NAS parallel program, as shown in Figure 7.1 (*C*). It is evident that users are able to establish the cause of performance bottlenecks incurred during execution of the program, which fulfils one of the user's needs when analysing the performance of parallel programs, as shown in Figure 6.26 (*B*). As earlier indicated, the user recommended the final visualisation design to scale vulnerable nodes based on their execution time. In the final design (filtering and search), red square nodes are scaled based on the total execution time of the parallel program. In particular, red nodes that consumed between 81% and 99.99% of the total simulation time are bigger than the red square nodes that have consumed between 60% and 80% of the total execution time, of which, red squares that consumed between 41% and 60% are smaller than all other red nodes, as shown in Figures 7.3 and 7.4.

The visualisation results (Figure 7.3) demonstrate the first view of the filtering callgraph design used to visualise performance data of the NAS parallel program executed on four and sixteen processors. In this filtering design, Figure 7.3 (*A*) illustrates the performance of the NAS parallel program running on four processors. Figure 7.3 (*B*) indicates the execution of the program on sixteen computational processors. Most notably, the user pointed the mouse to the SSOR node, which is displayed as a red square node within the filtering visualisation (Figures 7.3 *A* and *B*).

The red square node (SSOR object) visualised on the filtering design (Figure 7.3 *A*) is smaller than the square node displayed on Figure 7.3 (*B*). The SSOR object scales different sizes of the nodes on four and sixteen processors because the node consumed different execution times during execution of the program. The filtering visualisation design (Figure 7.3 *A*) indicates that the SSOR node consumed 75% of the total execution on four processors. Figure 7.3 (*B)* shows that the SSOR node consumed 81.73% of the total simulation time on sixteen processors runing on the computational system.

Furthermore, the search visualisation design also scales excessive nodes based on the total execution time of the program executed on a computational facility. Figure 7.4 shows the first view of the search design used to simulate the NAS parallel program on four and sixteen processors. In particular, Figure 7.4 (*A*) demonstrates the search design visualising performance metrics on four processors.

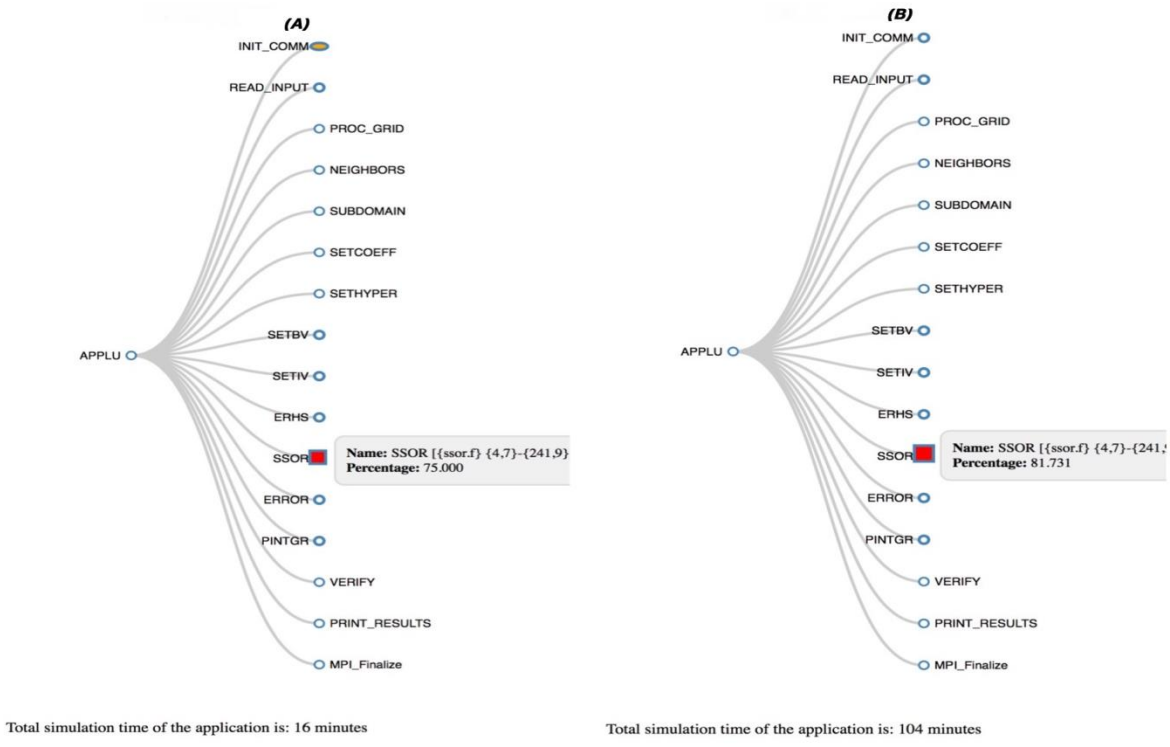

Figure 7.3: Filtering design visualising performance data

Figure 7.4 (*B*) indicates search visualisation on sixteen processors. The search design (Figure 7.4) visualises the same performance data analysed using the filtering visualisation design (Figure 7.3). Most importantly, it is noticeable that the search design (Figure 7.4 *A*) shows red square nodes (SSOR), which are smaller than the SSOR red node presented in Figure 7.4 (*B*). The SSOR object is presented in a small size on four processors because it consumed 75% of the total simulation; slightly bigger on sixteen processors due to 81.73% consumed during execution of the program. The scaling of nodes allows the users to understand which objects need more attention than others. In this study, an interactive design model was followed, which involved users at each stage of the visualisation design, as discussed in Section 4.2. This interactive design model helped us to identify and understand users' needs on the callgraph visualisations used to analyse the performance of parallel programs simulating on the supercomputers. It further helped to develop an effective callgraph visualisation that would efficiently enable users to identify performance bottlenecks of parallel programs. Most importantly, the users' evaluation of the visualisation designs enables us to identify errors and take corrective measures at an early stage of the research.

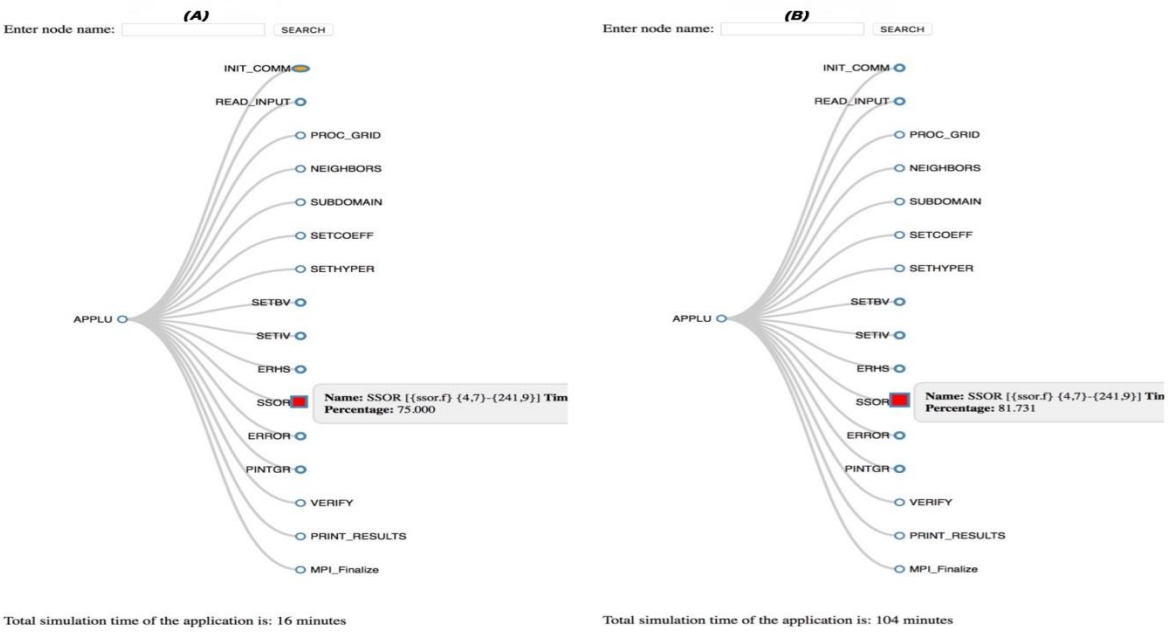

Figure 7.4: Search design visualising performance data

7.4 Filtering and search design on large performance metrics

The final visualisation design visualised large performance metrics (data) of more than 2000 objects (modules, routines, subroutines and functions) generated during the simulation of the WRF model on the CHPC's Lengau cluster. The final visualisation design consists of two modes - search and filtering design - as discussed in Section 7.2.

The below picture (Figure 7.5) shows the filtering visualisation design used to analyse and visualise large performance data generated during the execution of the WRF parallel program. Figure 7.5 (*A*) demonstrates the first view of the final callgraph design (filtering) used to analyse the performance of different objects executed during the simulation of the WRF parallel application. It further shows different types of nodes, namely red squares and white circles, which represent different execution statuses of each object processed during execution of the program.

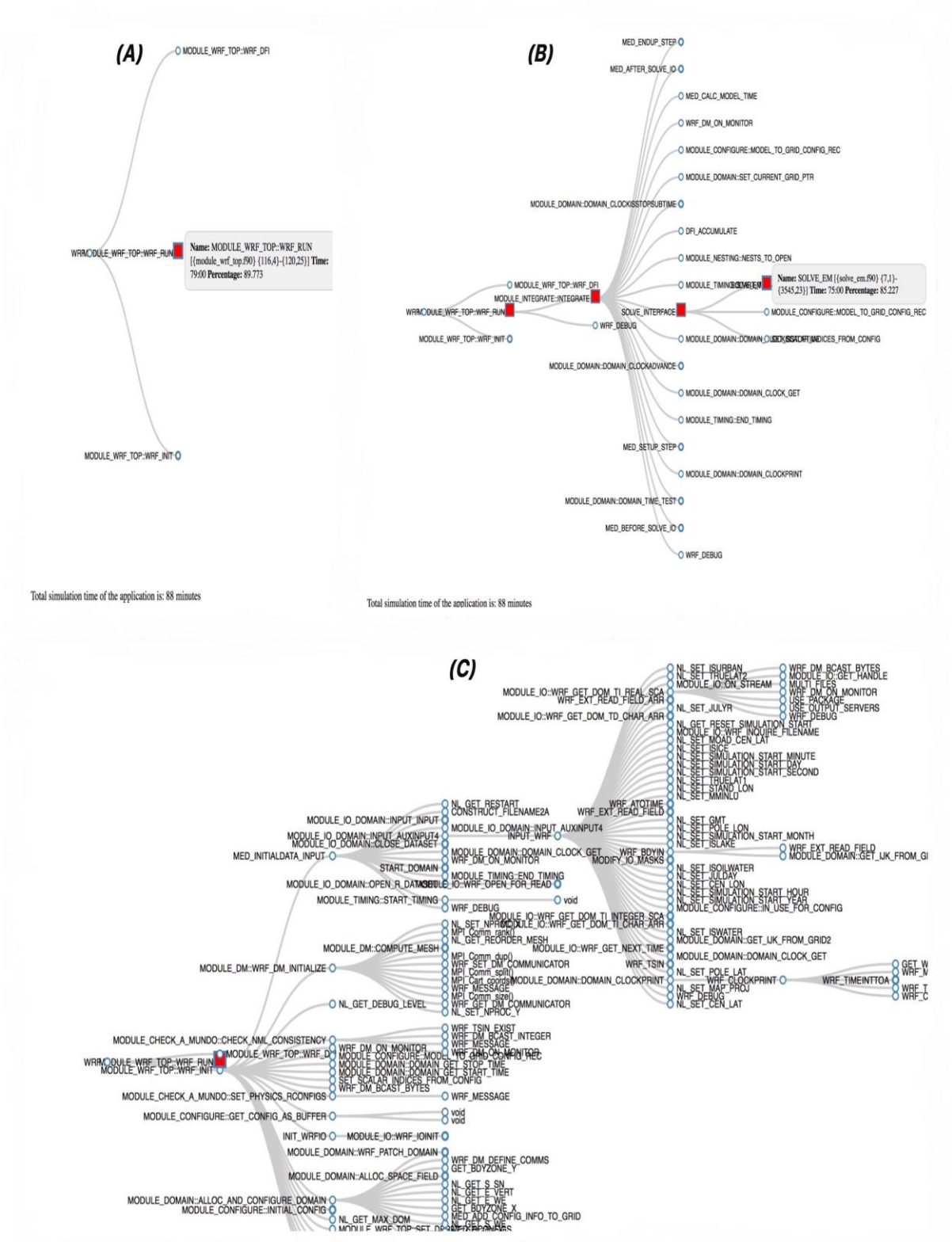

Figure 7.5: Filtering design used to analyse the WRF

In the final filtering design, white circle nodes represent objects that consumed normal execution time; orange oval nodes indicate objects that consumed average execution time, while red square nodes show objects that consumed excessive execution time, as discussed in Section 7.2. Furthermore, Figure 7.5 (*A*) indicates a situation where the visualisation user has hovered over the node (MODULE_WRF_TOP::WRF_RUN), which consequently displays a textbox with the following details: full name, execution time and percentage of the target object executed during the run. It further shows a white circle and red square node with big borders, which means that these two nodes contain children nodes. Figure 7.5 (*B*) shows different types of nodes (white circles and a red square) that consumed different execution time during the simulation of the WRF on the CHPC's Lengau cluster.

In particular, the filtering visualisation (Figure 7.5 *B*) illustrates that four objects: MODULE_WRF_TOP::WRF_RUN; MODULE_INTEGRATE::INTEGRATE; SOLVE_INTERFACE, and SOLVE_EM, consumed immoderate execution time while other nodes consumed normal execution time during the simulation of the WRF parallel program. The filtering design (Figure 7.5 *B*) further shows the relationship between red square nodes that caused performance bottlenecks during the run. In fact, the figure clearly demonstrates the connection path from the root cause of the bottleneck to the last object that decreases the overall performance of the WRF on a CHPC supercomputer.

The filtering design shows the full details of an object, namely, SOLVE_EM when the mouse hovers over the node, as illustrated in Figure 7.5 (*B*). In this filtering design (Figure 7.5 *C*) the node, MODULE_WRF_TOP::WRF_INIT, has children nodes; is clicked and expanded to display its related associates. Consequently, the filtering visualisation design displayed 130 children nodes which belong to the target node, MODULE_WRF_TOP::WRF_INIT. Most interestingly, the filtering design clearly displayed all 130 children nodes related to the target node at the same time, as shown in Figure 7.5 (*C*).

This filtering visualisation design also enables one to expand and disband the nodes in order to filter the information displayed on the screen. The filtering design works differently from the search design, the latter of which was also tested by the experts when analysing the execution of large performance metrics (datasets) generated through the simulation of the WRF on the CHPC's Lengau cluster. The search visualisation design does not expand or disband the nodes -

in fact, it collapses the nodes when the target node with children nodes is clicked, as discussed in Section 7.1. The first view of the search design looks similar to the first view of the filtering design except that the designs present information differently, as indicated in Figure 7.5 and 7.6, respectively. As per the below picture (Figure 7.6 *A*), the search design presents different types of nodes together with the textbox used to search a particular node and its associates. Moreover, the total execution time of the visualised program is displayed at the bottom of both the filtering and search visualisations, as indicated in Figure 7.5 (*A*) and 7.6 (*A*).

In both the filtering and search designs, one can obtain details (full name, execution time and percentage) of the object by hovering the mouse over the target node within the visualisation system. The search design only displays the searched/clicked node and its associates, rather than showcasing a tree of nodes, as indicated in Figure 7.6 (*B*). The search visualisation design (Figure 7.6 *B*) indicates that an object - MODULE_INTEGRATE::INTEGRATE - was searched; consequently, the visualisation displayed only the searched node and its kindred children nodes.

In this search design, the mouse was pointed to a red square node - SOLVE_INTERFACE - which resulted in a textbox containing the full details of the target node, as demonstrated in Figure 7.6 (*B*). As part of the demonstration, the target red square node - SOLVE_INTERFACE - was then clicked, which resulted in only the active node and its associated children nodes, as articulated in Figure 7.6 (*C).* Finally, the search visualisation design was used to search an object - OUTPUT_WRF - which then resulted in the active (searched) node and all its eighty-seven children nodes, as shown in Figure 7.6 (*D*). Most notably, the search design clearly displayed all these children nodes at the same time.

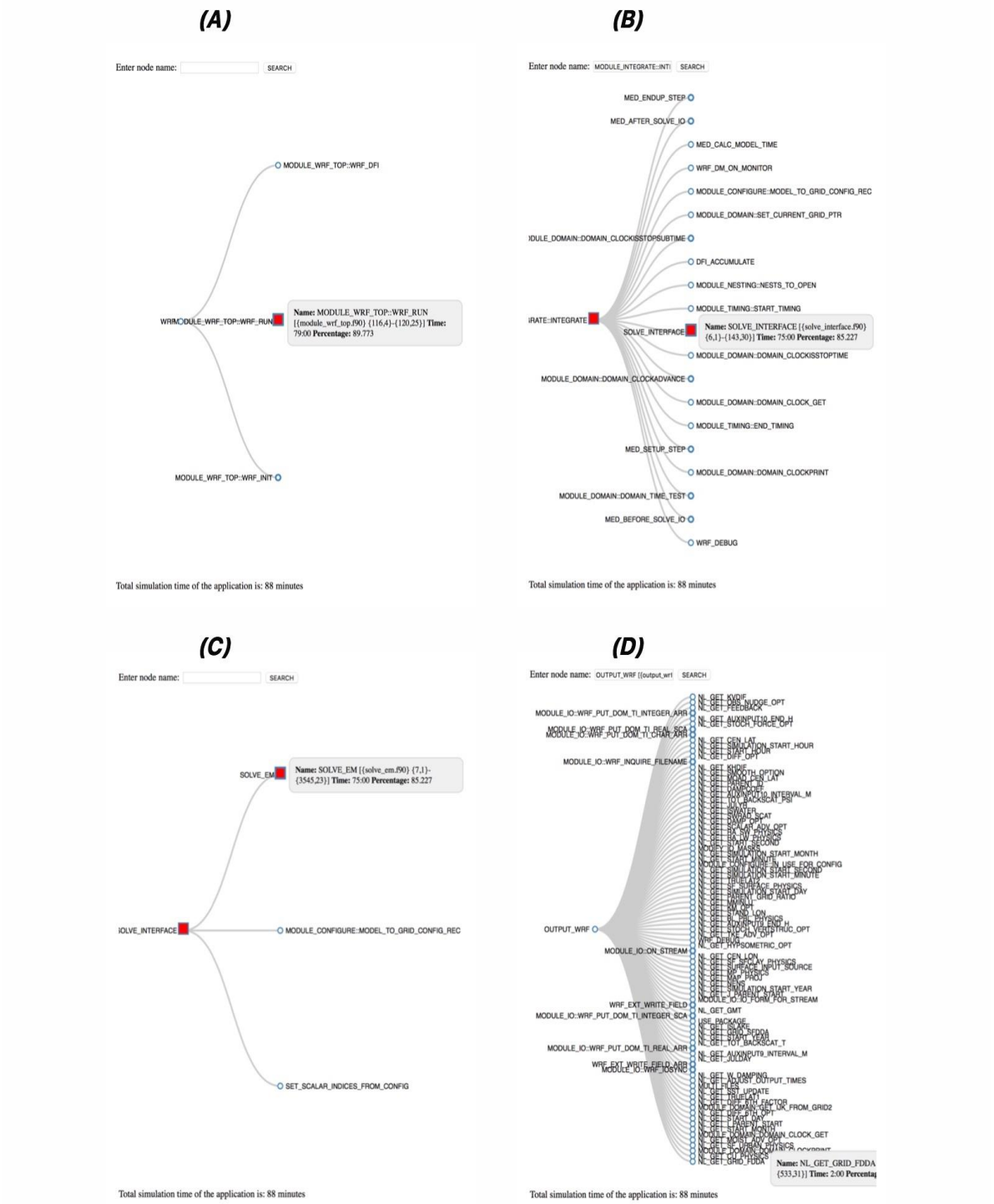

Figure 7.6: Search design used to analyse the WRF program

7.5 Conclusions

The chapter presented a comprehensive analysis of the final visualisation design used to present the relationship between different parts (functions, subroutines and routines) of the program simulated during the execution of the run. The chapter discussed how the final design visualised the small and large performance data (metrics) generated during the execution of the NAS parallel program and the WRF model computed in a computational system. Moreover, it precisely demonstrated how the final design (filtering and search mode) of the callgraph visualisation worked in terms of analysing the performance of parallel programs. In particular, two different modes - the filtering and search design - of the final callgraph visualisations were demonstrated. The final callgraph visualisation (filtering and search) enables one to effectively identify performance bottlenecks incurred during the simulation of the parallel program. In fact, this chapter showed that the newly developed search and filtering visualisations are effective, interactive, easy to use and informative when analysing parallel execution for optimising the performance of parallel programs.

CHAPTER 8: CONCLUSIONS

8.1 Conclusions and discussions

The study found the final design of the callgraph visualisation to be more effective in terms of analysing the performance of parallel programs and thus enabling better optimisation of parallel applications running on the supercomputers. Most of the parallel programs do not perform well due to various factors such as logic of the code, read/write activities within the storage, high memory latency, and processor utilisation within the execution node of the supercomputers. It is envisaged that the final design of the callgraph visualisations will enable users to efficiently optimise applications and increase performance/efficiency of the parallel programs. The original callgraph network visualisation does not have the ability to visualise large performance data (metrics) and display effective reports about the performance of the parallel applications. To this end, the design of the callgraph visualisation system was investigated, which was found to be inefficient in terms of analysing the execution of parallel programs. For example, original design of the callgraph visualisation use too many colours to represent the statuses (normal, moderate and excessive) of the nodes and many network links move over the nodes, which consequently hide details (e.g. name and execution time) of the objects simulated during the run.

In the original design, users need to manually expand the nodes to obtain their details - not an easy exercise, especially when the executed applications generate large performance data. The aim of this study was to develop an effective callgraph visualisation that would enable users to efficiently identify performance bottlenecks within the parallel programs. To this end, a new callgraph visualisation (final design) was developed over three iterations, namely, paper prototype, first high-fidelity prototype and second high-fidelity prototype. The final visualisation design is more effective than the existing (original) callgraph visualisation for analysing the performance of parallel programs. An interaction design (user-centred) model was followed to design an effective callgraph visualisation for analysing the execution of parallel programs running on the supercomputers. This interaction design model (Figure 4.1) enabled users to become co-designer of the system during the development of the new callgraph visualisations. It further allowed us to develop new callgraph visualisations, which

will be easily used by end users. Most notably, interactive design model assisted us to continual review and assess the users' requirement throughout the development of the final visualisation design. The process of involving users to evaluate the prototypes helped to identify and rectify errors within final design of the callgraph visualisation system. During evaluation of the prototypes, users brought new ideas and suggestions which helped to develop a successful visualisation design. Users' evaluations helped to identify strength/weaknesses of the original and prototype (low-fidelity and high-fidelity) designs used to analyse the performance of parallel programs. The principles (Table 4.1) of designing an interactive visualisation also helped us to develop an effective callgraph visualisation for analysing the performance of callgraphs. These principles enabled us to eradicate overloading of information and ensure the that the system provide feedback to the users and showcase visibility of the important controls such as nodes that consumed excessive execution time within the prototypes.

The paper prototype design (low-fidelity prototype design) - star (Figure 5.1 *A*) and tree (Figure 5.1 *B* and *C*) - was drafted. The paper prototypes design (low-fidelity prototype) was developed using Microsoft PowerPoint, which enables one to design paper prototypes with a polished software design. The paper prototype design (star and tree) enabled us to explore different ideas without using an excessive effort to modify the blueprint of the visualisation design. Most importantly, it was found that paper (low-fidelity) prototypes enable one to quickly modify and change the structure of the design without a need to develop a code, which requires sufficient time to program the visualisation. In the paper-based prototype design (star and tree), white circle nodes represent objects that consumed normal execution time while orange diamond nodes indicate objects that consumed moderate execution time of the simulation.

Furthermore, red square nodes depict objects that consumed excessive execution time during execution of the program. The three colours (white, orange and red) of the nodes were used in the low-fidelity prototypes in order to eliminate the use of too many colours for the nodes as it is the case with the original design. Too many colours of the nodes cause confusion to the users of the visualisation design. It is advisable to use limited number of colours when designing a visualisation system. Any visual property (colour, size or shape) should represent information to the user of the visualisation. For example, the low-fidelity paper prototype design utilises three different shapes (circle, diamond and square) to represent different execution statuses of the nodes within the design. At the same time, circle nodes are smaller than diamond nodes,

while diamond nodes are slightly smaller than the square nodes within the low-fidelity prototype with polished software design. One should make it easier for the visualisation users to utilise different colours, shapes and sizes to locate useful information (for example, execution status) of the node within the system. Moreover, it was found that the dark blue nodes connected to each other via blue network lines of the original design were not attractive. As a result, different colours of the network lines were used to establish the relationship between the nodes within low fidelity prototype design (star and tree). In fact, the relationship between the nodes that were associated with each other was displayed using network links with different colours within the paper prototype design (low-fidelity design). The borders of the nodes also used the same colours as the network lines within the design. An interactive design model was followed in this study, which means that visualisation users/experts evaluated each design of the callgraph visualisations.

In this case, an expert evaluated the paper-based prototype designs and found the prototypes to be useful in terms of analysing different parts (objects) of the parallel program simulated on a computational system. The evaluator (expert) opposed too many colours for the network lines and borders of the nodes within the prototype designs (tree and star). The expert found the colours of the network lines and nodes to be distracting because the design contains too many colours that do not provide useful information about the performance of the parallel program. During evaluation of the low-fidelity (paper) prototypes, experts found no problem with the names and execution time of the nodes within the design. In the star and tree paper prototypes, a shadow was placed next to each node, which was also found to be worthless within the paper designs.

Both the star and tree paper based design have a white background, which were found to be adequate within the design. The white background was chosen because it correlates very well with the black text (names) of the nodes in both the star and tree paper prototypes. It was beneficial to follow an interactive design model in this research study because it enabled usto identify errors at the early development stages of the visualisation system. The first high-fidelity prototype design was developed, which used one colour for network lines and few colours for the nodes as compared to the low-fidelity design. Nonetheless, the star and tree paper prototype designs enabled the user to click a target node, which consequently magnified the names of the active node within the visualisation. At the prototyping stage, it was

imperative to isolate the target node with a zoom feature in the design. The zoom tool was incorporated in both the star and tree paper prototype design in an attempt to resolve the difficulty of efficiently displaying the names of the nodes within the low fidelity prototypes. Most notably, the names of the nodes were displayed using regular black text within the paper prototype design. In this case, the names of the nodes were clearly displayed in the paper-based prototypes. The difference between the star and tree paper-based design is the way information is presented within the visualisation. In the star prototype design (paper), relationships between the nodes are displayed within large round circles. As for the tree paper prototype design, it spreads the nodes across different parts of the visualisation. The tree paper-based prototype was designed to accommodate many nodes at the same time, while the star paper design was drawn to test how the first view of the design will demonstrate the relationship between objects executed during the simulation of the parallel program.

The paper prototypes helped us to identify appropriate visual properties (for example, colour, size and shape) of the nodes and network links of the callgraph visualisation system. The paper prototypes focused on the physical design, but not on the conceptual design (how the system works). Consequently, the first high-fidelity prototype designs - the expandable star, collapsible star, expandable tree and non-expandable tree - were developed which focused on both the physical and conceptual design of the callgraph visualisations. In these first interactive prototypes, attractive colours (e.g. white, orange, red) for nodes and network links were applied within the visualisation. Our contribution in this study is to propose a novel technique of expanding, disbanding and collapsing the nodes within the new callgraph visualisations.

We have also introduced a new approach of filtering and searching performance bottlenecks incurred during the execution of the parallel programs. The interactive prototype designs, expandable star, collapsible star, and expandable tree design have the ability to expand, disband, and collapse the nodes. The non-expandable tree design displays all the nodes at the same time within the visualisation. The first interactive prototype designs consist of different types of nodes, which represent different execution statuses of the objects processed on a computational system. In particular, white circle nodes represent objects that consumed normal execution time, while orange oval nodes indicate parts of the program that consumed moderate execution time. The red square nodes illustrate objects that consumed excessive execution time during execution of the program. Furthermore, grey network lines are used to demonstrate the

relationship between the nodes in the first high-fidelity interactive prototype visualisations. The following interactive prototype designs - expandable star, collapsible star, and expandable tree utilise textboxes to display the details of the node within the visualisation. In these first interactive prototype designs, visualisations display a textbox which contains the details (full name and execution time) of the executed object when the user hovers over a node. The background of the interactive prototype visualisations is white, which works very well with the grey textbox used to provide details of the node within the callgraph visualisations.

It is also noticeable that black, regular texture is used to display details of the node within the textbox of the interactive prototype visualisations. The italic texture of the node in the original design was not applied in the interactive prototype design because it makes it hard for one to easily see the details of the node within the callgraph visualisation system. In the first high-fidelity visualisation design, different sizes of the nodes are used to emphasise the execution status of an object processed during the simulation of the parallel program. The white circle nodes are smaller than the orange oval nodes, while the orange nodes are smaller than the red square nodes within the visualisations.

Different sizes of the nodes helped to establish objects that are vulnerable and need more attention when optimising the performance of the parallel program. For example, large red squares demand more attention than the moderate orange oval nodes within the interactive prototype designs. At the same time, orange oval nodes also demand more attention than small white circles, which are used to depict the normal execution status of an object simulated during the execution of the parallel program. The first interactive prototype designs used different visualisation methods to present performance information of the parallel programs computed on the supercomputers. In particular, the expandable star visualisation design displays the *MAIN MODULE* in the centre and spreads other nodes in different directions on the screen.

In fact, this visualisation design looks like a star but it expands the nodes, which is the reason it is named the ‘expandable star’ design. All expandable nodes (children nodes) are surrounded by large borders while non-expandable nodes have smaller borders within the expandable star visualisation. These different border sizes helped to identify between the nodes that have children nodes, and those that do not have children objects, within the parallel program. In the

first interactive prototype designs, all the nodes are surrounded by steel blue borders. The steel blue colour of the border makes the node extremely visible on the white background of the interactive prototype visualisations. Nevertheless, the expandable star visualisation design is able to display the relationship between the objects by expanding and disbanding the nodes within the visualisation. The visualisation technique of expanding and disbanding the nodes helps to limit the information displayed on the screen. This visualisation technique is useful when the system visualises and presents large amounts of data. The expanding and disbanding feature was also applied on the expandable tree design of the callgraph visualisation system.

The expandable tree design visualises and presents information differently to the expandable star design. It expands/disbands the node from left to right, while the expandable star design expands/disbands the nodes to various directions of the callgraph visualisation system. The expandable tree design has the ability to display many nodes at the same time because the information is presented in a linear manner. As for the expandable star design, it can only display a limited number of nodes within the diameter of the visualisation design. The expandable tree design further increases or decreases the size of the nodes depending on the information presented and displayed on the screen.

For example, the expandable tree system will automatically reduce the size of the nodes when the visualisation user expands the children nodes within the visualisation. It will, however, increase the size of the nodes when one disbands the children nodes within the callgraph visualisation system. This feature of expanding and disbanding the nodes enables the visualisation to display many numbers of nodes simultaneously. In the case of the expandable tree design, the disbanding feature helps to make the node and its details (full name, execution time and status) more visible when the system presents less information on the screen. The grey network lines showcase the relationship between the nodes and its associates within the expandable tree design. The non-expandable tree design also presents the relationship between nodes differently. As earlier indicated, it visualises and presents all the nodes at the same time.

In particular, the relationship between the objects is demonstrated by displaying the connection between the nodes from the left to the right side of the visualisation. It can become complicated when displaying too many nodes at the same time, especially when the visualisation visualises large datasets. Nonetheless, the collapsible star design was developed, which also displays all

the nodes and their names at the same time. It is noted that the collapsible star design presents the information in a diameter where the nodes are spread across the entire system. The collapsible star displays details (full name and execution time) of the object when one hovers over the node in the visualisation design of the callgraph profiling system. When the user clicks the node, the system collapses a tree and displays only the relationship between the target (clicked) node and its associates. The collapsible star visualisation design also enabled the user to double-click a node in order to magnify the details (name) of that particular object simulated during the execution of the target application. The collapsible star design does not expand and disband the nodes, however, it has the capability of going back to the previous views of the visualisation design.

In the collapsible star design, the visualisation automatically goes back to the previous view when the user clicks anywhere on the white background of the callgraph visualisation system. This feature helps the user to move from one view to another within the visualisation. The click feature is not easy, unlike the expand/disband feature, which enables the user to move different sets of nodes back and forth at any time. As earlier indicated, the expandable star and tree designs are equipped with this feature to help the user easily manoeuvre different sets of nodes within the callgraph visualisation system. Nevertheless, the first interactive prototype designs - expandable star, expandable tree, collapsible star and non-expandable tree - demonstrated the ability to competently visualise the performance of parallel programs.

This first high-fidelity prototype design presented the relationship between the nodes in a more sophisticated manner than the original design of the callgraph visualisation does. In the first interactive prototype design, different sizes, shapes and colours of the nodes are used to indicate the execution status of the objects while grey network lines clearly emphasise the connection between the nodes and its associates. It is useful for the first interactive prototype visualisation design to surround nodes with small and large borders when establishing children and non-children nodes; this is not the case with the original design. In fact, the original design does not use different sizes, shapes and colours to help users identify the execution status of the node within the callgraph visualisation system. In the original design, different sizes of the nodes are randomly displayed within the callgraph visualisation system, which is not the case with the first interactive prototype designs. The users performed a usability test (to establish how easy it is to learn) on both the first high-fidelity interactive prototypes and the original

design of the callgraph performance analysis system. The users who tested these designs (first interactive prototypes and original) have experience in optimising, parallelising, debugging, analysis and development of parallel programs, as discussed in Section 6.4.3. In fact, these users have a variety of experience (between 5 - 20 years) in parallel programming. During the tests, users (evaluators) analysed the performance of the WRF model using the original design. Afterwards, these parallel program users visualised and analysed the performance of the NAS parallel program using the following first interactive prototype designs: expandable star, expandable tree, collapsible and non-expandable tree designs of the callgraph visualisation system. The users performed the same visual queries to analyse the performance of both the NAS and WRF parallel programs.

It was found that the first interactive prototype (new) design is easy and helpful when analysing and optimising the performance of parallel programs on the supercomputer. Some users believe that both the original and the first interactive prototype designs are easy or helpful in terms of analysing the execution of the parallel programs. It is noted that none of the visualisation users found the original design solely to be easy and helpful and many of the users found the first interactive prototype design solely to be easy to use and effective when visualising the performance data generated during the execution of the parallel programs. At the same time, 60% of the users discovered that an expandable tree design is helpful while 20% maintain that the expandable star design is meaningful.

The other 10% of users found the collapsible star design to be useful while 10% found both the original designs to be beneficial when analysing the performance of parallel programs. Basically, 90% of the users found the first interactive prototype design to be easier to use and more helpful than the original design of the callgraph visualisations. The first interactive (high-fidelity) prototype visualisation was also found to be interactive and informative, of which the expandable tree design appeared to be the easiest to use, the most informative, and the most interactive visualisation to analyse the performance of parallel program running on the supercomputers. During evaluations the following improvements were recommended by the users to be added to the new visualisation design of the callgraph visualisations: displaying total execution time on a legend; showing percentage of execution time on the node; scaling vulnerable nodes based on excessive execution time; and a search tool to highlight specific objects. To this end, the second high-fidelity prototype design was developed by following an

expandable tree visualisation design, which was found to be easy and useful to the users of the callgraph visualisation system. All recommendations (improvements) made by the users were adopted and applied on the second high-fidelity prototype visualisation design. The author continued to follow an interactive design model, whereby experts evaluated both the original and the second high-fidelity prototype design by analysing the performance of large datasets (metrics) generated during the execution of the WRF parallel model simulated on the CHPC's Petaflop Lengau cluster. Once again, experts performed the same visual queries using the original and the second high-fidelity design to analyse the execution of the WRF model. The purpose of this exercise was to establish whether the designs (original and second high-fidelity design) are able to effectively visualise large performance data of the parallel programs.

The results indicated that experts (evaluators) found the original design to be cumbersome and complicated when analysing the performance of parallel programs. All experts who evaluated the visualisation designs detected that the original design does not have the ability to visualise and present the large performance metrics generated during the simulation of the parallel program. In fact, some of the experts found it hard to use the original design due to the distracting colours of the nodes - the many colours representing similar execution statuses of the nodes and many network links moving over the boxes consequently hide the most important details about the object simulated during the execution of the parallel program. Experts found the second interactive visualisation design to be easy to use, informative and interactive when analysing the performance of the parallel program.

It was found that the second high-fidelity visualisation design efficiently visualised and presented large performance data generated during the simulation of the parallel applications. The experts further noticed that the second high-fidelity prototype design of the callgraph visualisation is effective when identifying performance bottlenecks (hotspot) within the parallel programs simulated on a supercomputing system. It was also perceived that the second high-fidelity prototype design is able to easily track the flow of the objects and establish the relationship between different parts (objects) of the parallel application executed on a computational facility. As a result, the second high-fidelity prototype was adopted in the development of the final design of the callgraph visualisation system. The final visualisation design, which consists of two different modes - filtering and search - was used to visualise the performance of different sizes (small and large) performance data, as discussed in Chapter 7. In

this study, we have demonstrated that visualisation researcher can use different visual properties such as colour, size and shape of the nodes to effectively visualise large datasets. This research study further outlined efficient ways of involving users as the co-designers of the visualisation system. An interactive design model (Figure 4.1) helped us to work closely with the expert users and identify errors at an early development stage of the new callgraph visualisation design. The method of conducting a design study enabled us to develop an effective callgraph visualisation for users to solve the problem of performance bottlenecks within parallel programs. This user-centred method (Figure 4.1) also allowed users to validate the design on each development stage of the new callgraph visualisations. The user-centred method enables visualisation researcher to develop a system that support solving real problem facing a specific domain and outline lessons learned in applying the principles of designing a visualisation system (Sedlmair, Meyer and Munzner, 2012).

8.2 Recommendations for future work

Both filtering and search design of the final visualisations can effectively visualise the performance of parallel programs and enable better optimisation of parallel programs. For future work, we recommend the following:

- Integrate new callgraphs design with TAU (Tuning and Analysis Utilities) visualisation system.
- After integration, add back and forth buttons in the menu of the TAU visualisations.
- Enable final visualisation design (search mode) to accept the short name of the object when searching for the relationship between the nodes that are associated with each other.

REFERENCES

Abras, C., Maloney-Krichmar, D., and Preece, J. (2004). User-centered design. Thousand Oaks: Sage Publications, edited by: Encyclopedia of human-computer interaction.

Adar, E. (2005). The graph exploration system. Available: http://graphexploration.cond.org/screenshots/Picture2.png [14 January 2019].

Adve, V.S., and Vernon, M.K. (2004). Parallel program performance prediction using deterministic task graph analysis. ACM Transactions on Computer Systems. 22(1): 94-136.

Arabe, J.N.C., Beguelin, A., Lowekamp, B., and Seligman, E. (1995). Dome: parallel programming in a heterogeneous multi-user environment. U.S Defense Advanced Research Projects Agency (technical report).

Attig, N. (2006). The John von Neumann Institute for Computing (NIC): A survey of its supercomputer facilities and its Europe-wide computational science activities. Nuclear Physics B - Proceedings Supplements. 153(1): 3-8.

Bade, R., Schlechtweg, S., and Miksch, S. (2004). Connecting time-oriented data and information to a coherent interactive visualization. Proceedings of the SIGCHI conference on human factors in computing systems. New York: ACM Digital Library, pp. 105-112.

Barker, D., Huang, X., Liu, Z., Auligné, T., Zhang, X., Rugg, S., Ajjaji, R., Bourgeois, A., Bray, J., Chen, Y., Demirtas, M., Guo, Y., Henderson, T., Huang, W., Lin, H., Michalakes, J., Rizvi, S., and Zhang, X. (2012). The weather research and forecasting model's community variational/ensemble data assimilation system: WRFDA. American Meteorological Society. 93 (6): 831-843.

Becker, D., Frings, W., and Wolf, F. (2008). Performance evaluation and optimization of parallel grid computing applications. 16th Euromicro conference on parallel, distributed and network-based processing. Washington, DC: IEEE computer society, pp. 193-199.

Benedict, S., Petkov, V., and Gerndt, M. (2010). Periscope: An online-based distributed performance analysis tool. Proceedings of the 3rd international workshop on parallel tools for high performance computing, edited by: M.S. Müller, M.M. Resch, A. Schulz and W.E. Nagel. Berlin, Heidelberg: Springer, pp. 1-16.

Bernard, J., Sessler, D., Kohlhammer, J., and Ruddle, R.A. (2018). Using dashboard networks to visualize multiple patient histories: a design study on post-operative prostate cancer. IEEE Transactions on Visualization and Computer Graphics. 1(1): 1077-2626.

Beyer, H., and Holtzblatt, K. (1999). Contextual design: defining customer-centered systems. San Francisco: Morgan Kaufmann Publishers.

Bélanger, G., Boudjema, F., Pukhov, A., and Semenov, A. (2007). Micromegas 2.0: A program to calculate the relic density of dark matter in a generic model. Computer Physics Communications. 176(5): 367-382.

Böhme, D., Geimer, M., Wolf, F., and Arnold, L. (2010). Identifying the root causes of wait states in large-scale parallel applications. 39th international conference on parallel processing. San Diego: IEEE, pp. 90-100.

Böhme, D., Wolf, F., and Geimer, M. (2012). Characterizing load and communication imbalance in large-scale parallel applications. IEEE 26th international parallel and distributed processing symposium workshops and PhD forum. Shangai: IEEE Computer Society, pp. 2538-2541.

Brodile, K.W. (1992). Scientific visualization: techniques and applications. New York: Springer-Verlag.

Catarci, T., Costabile, M.F., Levialdi, S., and Batini, C. 1997. Visual query systems for database: a survey. Journal of Visual Languages and Computing. 8(2): 215-260.

Chen, C. (2006). Information visualization: beyond the horizon (second edition). London: Springer-Verlag.

Chokbunpiam, T., Fritzsche, S., Chmelik, C., Caro, J., Janke, W., and Hannongbua, S. (2016). Gate opening effect for carbon dioxide in ZIF-8 by molecular dynamics – confirmed but at high CO2 pressure. Chemical Physics Letters. 648: 178-181.

Christakis, N.A., and Fowler, J.H. (2007). The spread of obesity in a large social network over 32 years. The New England Journal of Medicine. 357: 370-379.

Ciorba, F.M., Groh, S., and Horstemeyer, M. (2010). Early experiences and results on parallelizing discrete dislocation dynamics simulations on multi-core architectures. Mississippi State University: Centre for Advanced Vehicular Systems Report.

Coen, J.L., Cameron, M., Michalakes, J., Patton, E.D., Riggan, P.J., and Yedinak, K.M. (2013). WRF-fire: coupled weather-windland fire modeling with the weather research and forecasting model. Journal of Applied Meteorology and Climatology. 52: 16-38.

Cooper, A., Reimann, R., and Cronin, D. (2007). About face 3: the essentials of interaction design. Indianapolis: Wiley Publishing, Inc.

Coutinho, E.F., De Carvalho, F.R., Rego, P.A.L., Gomes, D.G., and De Souza, J.N. (2015). Elasticity in cloud computing: a survey. Annals of Telecommunications. 70(7-8): 289-309.

Davis, J.B.A., Shayeghi, A., Horswell, S.L., and Johnston, R.L. (2015). The Birmingham parallel genetic algorithm and its application to the direct DFT global optimisation of Ir^N (N = 10-20) clusters. Royal Society of Chemistry. 7: 14032-14038.

Delistavrou, C.T., and Margaritis, K.G. (2011). Towards an integrated teaching environment for parallel programming. 15th Panhellenic conference on informatics, edited by: P. Angelidis and A. Michalas. Los Alamitos, CA: IEEE Computer Society, pp. 508-514.

De Nooy, W., Mrvar, A., Batagelj, V. (2018). Exploratory social network analysis with Pajek: revised and expanded edition for updated software (third edition). New York: Cambridge University Press.

Dimitroulis, C., Raptis, T., and Raptis, V. (2015). POLYANA - a tool for the calculation of molecular radial distribution functions based on molecular dynamics trajectories. Computer Physics Communications. 197: 220-226.

Dongarra, J., and Van der Steen, A.J. (2012). High-performance computing systems: status and outlook. Acta Numerica. 21: 379-474.

Dorta, I., Leon, C., and Rodriguez, C. (2006). Performance analysis of branch-and-bound skeletons. 14th Euromicro international conference on parallel, distributed and network-based processing, edited by: J.D. Cantarella. New York: IEEE Computer Society, pp. 8.

Dunne, C., and Shneiderman, B. (2013). Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. Proceedings of the SIGHI Conference on Human Factors in Computing Systems. New York: ACM Digital Library, pp. 3247-3256.

Ebnenasir, A., and Beik, R. (2009). Developing parallel programs: a design-oriented perspective. Proceedings of the 2009 IEEE 31st international conference on software engineering. Vancouver: IEEE Computer Society, pp. 1-8.

Elkind, J.I., Card, S.K., Hochberg, J., and Huey, B.M. (2014). Human performance models for computer-aided engineering. Washington, D.C: Academy Press.

Endsley, M.R., and Jones, D.G. (2004). Designing for situation awareness: an approach to user-centered design (2nd edition). New York: CRC Press.

Falcone, M., and Sharif, B. (2013). UnionUML: an eclipse plug-in for visualizing UML class diagrams in onion graph notation. IEEE 21st international conference on program comprehension. Piscataway: IEEE Xplore Digital Library, pp. 233-235.

Filgueras, A., Gil, E., Jimenez-Gonzalez, D., Alvarez, C., Martorell, X., Langer, J., Noguera, J., and Vissers, K. (2014). OmpSs@Zynq all-programmable SoC ecosystem. Proceedings of the 2014 ACM/SIGDA international symposium on field-programmable gate arrays. New York: ACM Digital Library, pp. 137-146.

Freeman, J., Vladimirov, N., Kawashima, T., Mu, Y., Sofroniew, N.J., Bennett, D.V., Rosen, J., Yang, C., Looger, L.L., and Ahrens, M.B. (2014). Mapping brain activity at scale with cluster computing. Nature America, Inc. 11: 941-950.

Gabbard, J.L., Hix, D., and Swan, J.E. (1999). User-centered design and evaluation of virtual environments. IEEE Computer Graphics and Applications. 19(6): 51-59.

Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., and Graham, R.L. (2004). Open MPI: goals, concept, and design of a next generation MPI implementation. Recent Advances in Parallel Virtual Machine and Message Passing Interface. Heidelberg: Springer-Verlag, pp. 97-104.

Garrett, J.J. (2010). The elements of user experience: user-centered design for the web and beyond (2nd edition). Berkeley: New Riders Publishing.

Gehrke, A.S., Ra, I. and Connors, D.A. (2011). A framework for automated performance tuning and code verification on GPU computing platforms. IEEE international symposium on parallel and Phd forum, edited by: B. Werner. New York: IEEE Computer Society, pp. 2113-2116.

Geimer, M., Wolf, F., Wylie, B.J.N., Abraham, E., Becker, D., and Mohr, B. (2010). The scalable performance toolset architecture. Concurrency and Computation: Practice and Experience. 22(6): 702-719.

Gropp, W. (2012). MPI 3 and beyond: why MPI is successful and what challenges it faces. Recent Advances in the Message Passing Interface. Heidelberg: Springer-Verlag, pp. 1-9.

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., and Cajander, A. (2010). Key principles for user-centred systems design. Behaviour and Information Technology. 22(6): 397-409.

Hammond, J.R., Krishnamoorthy, S., Shende, S., Romero, N.A., and Malony, A.D. (2011). Performance characterization of global address space applications: a case study with NWChem. Concurrency and Computation: Practice and Experience. 24(2): 135-154.

Heer, J., and Boyd, D. (2005). Vizster: visualizing online social networks. Proceedings of the 2005 IEEE symposium on information visualization, edited by: J. Stasko and M. Ward. New York: IEEE Computer Society, pp. 32-39.

Hein, J., Reid, F., Smith, L., Guest, M., and Sherwood, P. (2005). On the performance of molecular dynamics applications on current high-end systems. Philosophical Transactions of the Royal Society. 363(1833): 1987-1998.

Huck, K.A., Porterfield, A., Chaimov, N., Kaiser, H., Malony, A.D., Sterling, T., and Fowler, R. (2015). An autonomic performance environment for exascale. Supercomputing Frontiers and Innovations. 2(3): 49-66.

Isaacs, K.E., Bremer, P., Jusufi, I., Gamblin, T., Bhatele, A., Schulz, M., and Hamann, B. (2014). Combining the communication hairball: visualizing parallel execution traces using logical time. IEEE Transactions on Visualization and Computer Graphics. 20(12): 2349-2358.

Karrer, T., Krämer J., Diehl, J., Hartmann, B., and Borchers, J. (2011). Stacksplorer: call graph navigation helps increasing code maintenance efficiency. Proceedings of the 24th annual symposium on user interface software and technology. New York: ACM Digital Library, pp. 217-224.

Keller, P.R., and Keller, M.M. (1993). Visual cues: practical data visualization. New York: IEEE Computer Society Press.

Kerren, A., Ebert, A., and Jörg, M. (2007). Human-centered visualization environments. Berlin, Heidelberg: Springer-Verlag.

Khain, A., and Lynn, B. (2009). Simulation of a supercell storm in clean and dirty atmosphere using weather research and forecast model with spectral bin microphysics. Journal of Geophysical Research: Atmospheres. 114(D19): 2156-2202.

Kim, Y., Sartelet, K., Raut, J., and Chazette, P. (2013). Evaluation of the weather research and forecast/urban model over greater Paris. Boundary-Layer Meteorology. 149(1): 105-132.
Kindratenko, V., and Trancoso, P. (2011). Trends in High-Performance Computing. Computing in Science and Engineering. 139(3): 92-95.

Kluge, M., Knüpfer, A., and Nagel, W.E. (2010). Efficient pattern based I/O analysis of parallel programs. Proceedings of the 2010 39th international conference on parallel processing workshops. Washington, DC: IEEE computer society, pp. 144-153.

Knüpfer, A., Brunst, H., and Nagel, W.E. (2005). High performance event trace visualization. Proceedings of the 13th euromicro conference on parallel, distributed and network-based processing. Washington, DC: IEEE computer society, pp. 258-263.

Knüpfer, A., Rössel, C., Mey, D., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W.E., Oleynik, Y., Phillippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., and Wolf, F. (2012). Score-P: a joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU and Vampir. Berlin, Heidelberg: Springer-Verlag.

Komatitsch, D., Erlebacher, G., Göddeke, D., and Michéa, D. (2010). High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. Journal of Computational Physics. 229(20): 7692-7714.

LaToza, T.D., and Myers, B.A. (2011). Visualizing call graphs. IEEE symposium on visual langauages and human-centric computing. Pittsburgh: IEEE Xplore Digital Library, pp. 117-124.

Li, H., and Tesfatsion, L. (2009). Development of open source software for power market research: the AMES test bed. The Journal of Energy Market. 2(2): 111-128.

Liem, A., and Sanders, E.B.-N. (2011). The impact of human-centered design workshops in strategic design projects. Proceedings of the second international conference on human centered design, edited by: M. Kurosu. Berlin, Heidelberg: Springer-Verlag, pp. 110-119.

Lima, M. (2010). Visual complexity mapping patterns of information. New York: Princeton Architectural Press.

Losada, N., Cores, I., Martin, M.J., and González, P. (2017). Resilient MPI applications using an application-level checkpointing framework and ULFM. The Journal of Supercomputing. 73(1): 100-113.

Mabakane, M.S., Moeketsi, D.M., and Lopis, A. (2017). Scalability of DL_POLY on high performance computing platform. South African Computer Journal. 29(3): 81-94.

Malony, A.D., Biersdorff, S., Spear, W., and Mayanglambam, S. (2010). An experimental approach to performance measurement of heterogeneous parallel applications using CUDA. Proceedings of the 24th ACM international conference of supercomputing, edited by: T. Boku, H. Nakashima and A. Mendelson. New York: ACM Digital Library, pp. 127-136.

McNicholas, P.D., Murphy, T.B., McDaid, A.F., and Frost, D. (2010). Serial and parallel implementations of model-based clustering via parsimonious Gaussian mixture models. Computational Statistics & Data Analysis. 54(3): 711-723.

Mey, D., Biersdorf, C., Diethelm, K., Eschweiler, D., Gerndt, M., Knüpfer, A., Lorenz, D., Malony, A., Nagel, W.E., Oleynik, Y., Rössel, C., Saviankou, P., Schmidl, D., Shende, S.,

Murray, S. (2017). Interactive data visualization for the web: an introduction to designing with D3 (second edition). Sebastopol: O'Reilly Media, Inc.

Munzner, T. (2014). Visualization analysis and design. New York: CRC Press.

Wagner, M., Wesarg, B., and Wolf, F. (2010). Score-p: A unified performance measurement system for petascale applications. Proceedings of an International Conference on Competence

in High Performance Computing, edited by: C. Bischof, H. Hegering, W.E. Nagel and G. Wittum. Berlin, Heidelberg: Springer, pp. 85-97.

Morris, A., Malony, A.D., and Shende, S. (2007). Supporting nested OpenMP parallelism in the TAU performance system. International Journal of Parallel Computing. 35(4): 417-436.

Narten, T., and Burgess, M. (2003). Unix operating system. Chichester: John Wiley and Sons Ltd.

Nehrkorn, T., Eluszkiewicz, J., Wofsy, S.C., Lin, J.C., Gerbig, C., Longo, M., and Freitas, S. (2010). Coupled weather research and forecasting-stochastic time-inverted lagrangian transport (WRF-STILT) model. Meteorology and Atmospheric Physics. 107(1): 51-64.

Newman, M. (2018). Networks (second edition). New York: Oxford University Press.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with CUDA. GPU Computing. 6(2): 40-53.

Nielsen, J., and Loranger, H. (2006). Prioritizing web usability. California: New Riders.

Norman, D.A. (1999). The invisible computer: why good products can fail, the personal computer is so complex, and information appliances are the solution. Cambridge: The MIT Press.

Patki, T., Lowenthal, D.K., Sasidharan, A., Maiterth, M., Rountree, B.L., and Schulz, M. (2015). Practical resource management in power-constrained, high performance computing. Proceedings of the 24th international symposium on high-performance parallel and distributed computing, edited by: T. Kielmann, D. Hildebrandand M. Taufer. New York: ACM Digital Library, pp. 121-132.

Pereira, C.M.N.A., Mól, A.C.A., Heimlich, A., Moraes, S.R.S., and Resende, P. (2013). Development and performance analysis of a parallel Monte Carlo neutron transport simulation program for GPU-Cluster using MPI and CUDA technologies. Progress in Nuclear Energy. 65:

88-94.

Petkov, V., and Gerndt, M. (2010). Integrating parallel application development with performance analysis in periscope. IEEE international symposium on parallel & distributed processing, workshops and Phd forum. Atlanta: IEEE Computer Society, pp. 1-8.

Preece, J., Rogers, Y., and Sharp. (2002). Interaction design: beyond human-computer interaction (1st edition). New York: John Wiley & Sons Ltd.

Rampersad, L., Blyth, S., Elson, E., and Kuttel, M.M. (2017). Improving the usability of the scientific software with participatory design: a new interface design for radio astronomy visualisation software. Proceedings of the South African Institute of Computer Scientists and Technologists' 17. New York: ACM Digital Library, pp. 9.

Rogers, Y., Sharp, H., and Preece, J. (2011). Interaction design: beyond human-computer interaction (3rd edition). Chichester: John Wiley & Sons Ltd.

Rohrer, M.W. (2000). Seeing is believing: the importance of visualization in manufacturing simulation. Proceedings of the 2000 Winter Simulation Conference, edited by: J.A. Jones, R.R. Barton, K. Kang and P.A. Fishwick. Piscataway: IEEE Xplore Digital Library, pp. 1211-1216.

Rolph, G., Stein, A., and Stunder, B. (2017). Real-time environmental applications and display system: ready. Environmental Modelling & Software. 95: 210-228.

Roosta, S.H. (2000). Data parallel programming. New York: Springer. pp. 477-499.

Roth, R.E., Ross, K.S., Finch, B.G., Luo, W., and MacEachren, A.M. (2010). A user-centered approach for designing and developing spatiotemporal crime analysis tools. Proceedings of GIScience, edited by: S.I. Fabrikant, T. Reichenbacher, M. Van Kreveld and C. Schlieder. Zurich: Springer, pp. 325.

Rubin, J., and Chisnell, D. (2008). Handbook of usability testing: how to plan, design, and conduct effective tests (second edition). New York: John Wiley & Sond Ltd.

Schulze, A.N. (2001). User-centered design for information professionals. Journal of Education for Library and Information Science. 42(2): 116-122.

Sedlmair, M., Meyer, M., and Munzner, T. (2012). Design study methodology: reflections from the trenches and the stacks. IEEE Transactions on Visualization and Computer Graphics. 18(12): 2431-2440.

Sharp, H., Rogers, Y., and Preece, J. (2009). Interaction design: beyond human-computer interaction (2nd edition). Chichester: John Wiley & Sons Ltd.

Shende, S., and Malony, A. (2006). The Tau parallel performance system. International Journal of High Performance Computing Applications. 20(2): 287-311.

Shende, S., Malony, A., Morris, A., and Beckman, P. (2006). Performance and memory evaluation using TAU. Cray user's group conference, edited by: S.R. Alam, R.F. Barrett, M.R.

Shneiderman, B., Plaisant, C., Cohen, M.S., Jacobs, S., Elmqvist, N., and Diakopoulos, N. (2016). Designing the user interface: strategies for effective human-computer interaction (6th edition). Hoboken: Pearson.

Stone, J.E., Gohara, D., and Shi, G. (2010). OpenCL: A parallel programming standard for heterogenous computing systems. Computing in Science and Engineering. 12(3): 66-73.

Sumanth, J.V., Swanson, D.R., and Jiang, H. (2003). Performance and cost effectiveness of a cluster of workstations and MD-GRAPE 2 for MD simulations. Proceedings of the second international symposium on parallel and distributed computing. Ljubljana: IEEE Computer Society, pp. 244-249.

Shende, S., Malony, A., Morris, A., and Cronk, D. (2008). Observing parallel phase and I/O performance using TAU. DoD HPCMP users group conference, edited by: S. Ceballos. New York: IEEE Computer Society, pp. 431-436.

Shun, J., Blelloch G.E., Fineman, J.T., Gibbons, P.B., Kyrola, A., Vardhan Simhadri, H, and Tangwongsan, K. (2012). Brief announcement: the problem based benchmark suite. Proceedings of the twenty-fourth annual ACM symposium on parallelism in algorithms and architectures. New York: ACM Digital Library, pp. 68-70.

Smith, M.A., Schneiderman, B., Milic-Frayling, N., Rodrigues, E.M., Barash, V., Dunne, C., Capone, T., Perer, A., and Gleave, E. (2009). Analyzing (social media) networks with NodeXL. Proceedings of the fourth international conference on communities and technologies, edited by: J.M. Caroll. New York: ACM Digital Library, pp. 255-264.

Smith, W. (2006). Guest editorial: DL_POLY-applications to molecular simulation II. Journal of Molecular Simulation. 32(12-13): 933-933.

Smith, W., and Todorov, I.T. (2006). A short description of DL_POLY. Journal of Molecular Simulation. 32(12-13): 935-943.

Spear, W., Malony, A., Morris, A., and Shende, S. (2006). Integrating TAU with Eclipse: A performance analysis system in an integrated development environment. High Performance Computing and Communications. 4208: 230-239.

Spear, W., Shende, S., Malony, A., Portillo, R., Teller, P.J., Cronk, D., Moore, S., and Terpstra, D. (2009). Making performance analysis and tuning part of the software development cycle. Proceedings of the DoD high performance computing modernization program users group conference, edited by: P. Kellenberger. Washington, DC: IEEE Computer Society, pp. 430-437.

Strohmaier, E., Dongarra, J., and Meuer, H.W., and Simon, H.D. (2005). Recent trends in the marketplace of high performance computing. Parallel Computing. 31(3+4): 261-273.

Subotic, V., Sancho, J.C., Labarta, J., and Valero, M. (2010). A simulation framework to automatically analyse the communication-computation overlap in scientific applications. IEEE international conference on cluster computing, edited by: P. Kellenberger. New York: IEEE Computer Society, pp. 275-283.

Sunderland, A., and Porter, A. (2007). Profiling parallel performance using Vampir and Paraver. Available: www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0704.pdf [14 January 2019].

Sy, D. (2007). Adapting usability investigations for agile user-centered design. Journal of Usability Studies. 2(3): 112-132.

Tang, E. (2008). DL_POLY 3.0: Performance study of a Si02/Water system. Edinburgh: The University of Edinburgh, pp. 1-96.

Truong, H., Fahringer, T., Madsen, G., Malony, A.D., Moritsch, H., and Shende, S. (2001). On using SCALEA for performance analysis of distributed and parallel programs. Proceedings of the ACM/IEEE SC2001 conference. Denver, Colourado: IEEE Xplor Digital Library, pp. 37.

Tufte, E.R. (2001). The visual display of quantitative information. Connecticut: Graphics Press.

Tullis, T., and Albert, B. (2013). Measuring the user experience: collecting, analyzing, and presenting usability metrics (second edition).New York: Elsevier Inc.

Van der Ryn, S. (2013). Design for an empathic world: reconnecting people, nature and self. Washington: Island Press.

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wikes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the tenth European conference on computer systems. New York: ACM Digital Library, pp. 18.

Wang, C., and Tao, J. (2017). Graphs in scientific visualization: a survey. Computer Graphics Forum. 36(1): 263-287.

Wang, K., Lv, X., Feng, D., Li, J., Chen, S., Sun, J., Song, L., Xie, Y., Li, J., and Zhou, H. (2015). Pyrazolate-based porphyrinic metal-organic framework with extraordinary base-resistance. Journal of the American Chemical Society. 138: 914-919.

Wang, Y., Harrison, C.B., Schulten, K., and McCammon, J.A. (2011). Implementation of accelerated molecular dynamics in NAMD. Computational Science and Discovery. 4(1): 105-2586.

Ware, C. (2013). Information visualization: perception for design. San Francisco: Morgan Kaufmann Publishers.

Wehner, M.F., Reed, K.A., Li, P., Bacmeister, C.C., Paciorek, C., Gleckler, P.J., Sperber, K.R., Collins, W.D., Gettelman, A., and Jablonowski, C. (2014). The effect of horizontal resolution on simulation quality in the Community Atmospheric Model, CAM5.1. Journal of Advances in Modelling Earth Systems. 6(4): 980-997.

Werstein, P., Situ, H., and Huang, Z. (2006). Load balancing in a cluster computer. Proceedings of the seventh international conference on parallel and distributed computing, applications and technologies, edited by: L. O'Corner. Los Alamitos: IEEE Computer Society, pp. 569-577.

Wong, H.J. (2009). Integrating software distributed shared memory and message passing programming. IEEE international conference on cluster computing and workshops. New York: IEEE Computer Society, pp. 1-10.

Yang, C., Huang, C., and Lin, C. (2011). Hybrid CUDA, OpenMP, and MPI parallel programming on muilticore GPU clusters. Computer Physics Communications. 182(1): 266-269.

Zuk, T., Schlesier, L., Neumann, P., Hancock, M.S., and Carpendale, S. (2006). Heuristics for information visualization evaluation. Proceeding of the 2006 AVI workshop on beyond time and errors: novel evaluations methods for information visualization. New York: ACM Digital Library, pp. 1-6.