

University of Southampton Research Repository ePrints Soton

Copyright © and Moral Rights for this thesis are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holders.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given e.g.

AUTHOR (year of submission) "Full thesis title", University of Southampton, name of the University School or Department, PhD Thesis, pagination

UNIVERSITY OF SOUTHAMPTON
FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

**Interconnection Networks Performance Modelling for Multi-core
Multi-Cluster Architecture**

By

Norhazlina Hamid

Thesis for the degree of Doctor of Philosophy in Computer Science

June 2016

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

**Interconnection Networks Performance Modelling for Multi-core
Multi-Cluster Architecture**

Norhazlina Hamid

In High Performance Computing (HPC) system design and deployment there is an increasing trend towards networked parallel systems such as cluster computing systems. Cluster computing is typically built from a group of workstations connected by high-speed networks to form a single high-availability system. One of the driving forces behind high-performance clusters is the advent of multi-core clusters. The aim of the research reported here is to design a new architecture for large-scale multi-core cluster computing systems and to investigate the interconnection network performance of the new architecture.

Since the overall performance of cluster computing systems always depends on the efficiency of its communication networks, performance analysis of the interconnection networks is vital. A general problem in the network may arise from the fact that multiple messages can be in transmission at the same time, using the same network links.

The contribution of this thesis is to develop a new architecture known as Multi-core Multi-cluster Architecture (MCMCA), composed of numbers of clusters where each cluster is a multi-core processor. Next, a simulation model is built to investigate the interconnection network performance of the new architecture, and the results are presented. The main performance metrics to be simulated are the latency and network throughput. The model is then used to evaluate the impact on scalability and cluster size of the interconnection network performance. Finally, analytical model including statistical analysis are used to validate the simulation results under various working conditions.

The analysis indicates that, from single-core to multi-core, there is a significant improvement in processor performance. To judge from the latency results, compared

to single-core cluster a multi-core cluster can improve the network performance. Another observation is that the architecture can achieve lower latency and higher throughput as the number of cores increases. The experiments also demonstrated that a multi-core cluster can scale better than a single-core cluster. The results comparison between the analytical model and those produced from the simulation experiments has shown that the derived simulation model provides a good basis for predicting the communication delay of the interconnection network performance of the Multi-Core Multi-Cluster Architecture (MCMCA).

Table of Contents

Table of Contents	iii
List of algorithms	ix
List of figures.....	xi
List of tables.....	xv
Declaration of Authorship	xvii
Acknowledgements	xix
Definitions and Abbreviations	xxi
Chapter 1..... Introduction	1
1.1 Research Motivation.....	1
1.2 Research Objectives	3
1.3 Published Papers	4
1.4 Thesis Structure	5
Chapter 2..... Literature Review.....	7
2.1 Introduction.....	7
2.2 Computers and Processors.....	7
2.3 Multi-core Processors	9
2.3.1 Moore's 'Law' Changes with Multi-core Processor.....	11
2.4 Cluster Computing	12
2.4.1 Cluster Interconnection Networks	14
2.5 Single-core Clusters	15
2.6 Multi-core Clusters	16
2.6.1 The Advantages of a Multi-core Cluster	18
2.6.2 Research Challenges.....	19

2.7 Multi-cluster Architectures.....	20
2.8 Message, Packet and Flits	21
2.9 Modelling and Simulation	21
2.9.1 Modelling and Simulation Techniques	22
2.9.2 Simulation Model Structure	23
2.10 Summary.....	27
Chapter 3 The New Architecture	29
3.1 Introduction.....	29
3.2 Multi-core Multi-cluster Architecture (MCMCA).....	29
3.3 Appropriate characteristics for cluster architecture design.....	31
3.3.1 Intra-Chip network (AC).....	31
3.3.2 Inter-Chip network (EC).....	32
3.3.3 Intra-Cluster Network (ACN).....	33
3.3.4 Inter-Cluster Network (ECN) and Multi-Cluster Network (MCN).....	33
3.4 The MCMCA Queueing Network Model	34
3.5 MCMCA Activity Diagrams	37
3.6 Simulation model to investigate interconnection network performance.....	39
3.6.1 OMNeT++ Network Simulation Tool	40
3.7 The MCMCA Simulation Model	42
3.7.1 Experiments with a Single-core Cluster.....	43
3.7.2 Experiments with Multi-core Clusters	46
3.8 Summary.....	48
Chapter 4 The Performance Model using Store-and-Forward Flow Control Mechanism	49

4.1	Introduction.....	49
4.2	Store-and-Forward Flow Control Mechanism	51
4.3	Assumptions and Notations	51
4.3.1	Assumptions.....	51
4.3.2	Notations	52
4.4	Evaluation Methodology	53
4.4.1	Simulation Structures	53
4.4.2	The Simulation Activity Diagram	57
4.4.3	Simulation Experimental Setup	57
4.5	The Analytical Model	62
4.5.1	Preliminaries.....	63
4.5.2	Average Message Latency of an Internal-Cluster Network.....	63
4.5.3	Average Message Latency of an External-Cluster Network.....	65
4.5.4	Average Message Latency of MCMCA.....	66
4.5.5	Implementation of the Analytical Model	66
4.6	Performance Evaluation (SQ4).....	70
4.6.1	Latency and Throughput Performance on MCMCA.....	71
4.6.2	The impact on cluster size.....	73
4.6.3	The impact on message length and scalability	75
4.7	Summary.....	77
Chapter 5.....The Performance Model using Wormhole Flow Control Mechanism.....		79
5.1	Introduction.....	79
5.2	Wormhole Flow Control Mechanism	80

5.3 Assumptions and Notations	80
5.4 The Simulation Model.....	80
5.4.1 Simulation Structure	80
5.4.2 The Experimental Setup.....	82
5.5 The Analytical Model	82
5.5.1 Preliminaries.....	82
5.5.2 Average Message Latency of an Internal-Cluster Network.....	83
5.5.3 Average Message Latency of an External-Cluster Network.....	85
5.5.4 Average Message Latency of MCMCA	87
5.5.5 Implementation of the Analytical Model	87
5.6 Performance Evaluation (SQ4)	93
5.6.1 Baseline experiment on MCMCA.....	93
5.6.2 Latency and throughput on MCMCA.....	94
5.6.3 The Impact on cluster size	97
5.6.4 The impact on message length and scalability.....	100
5.7 Summary.....	102
Chapter 6 Statistical Analysis.....	105
6.1 Introduction.....	105
6.2 Analysis of Experimental Results.....	105
6.2.1 Normality test	106
6.3 Mean Differences between Mechanisms	110
6.3.1 Discussion.....	111
6.4 Mean Differences between Processor Core Performance	112

6.4.1 Discussion.....	114
6.5 Summary.....	115
Chapter 7..... Conclusions, Contributions and Future Work.....	117
7.1 Conclusions	117
7.2 Research Contributions	120
7.3 Future Research Directions.....	121
7.3.1 Limitations of the Research.....	121
7.3.2 Specific Extensions.....	123
7.4 Concluding Remarks.....	124
Appendix 2-A	125
Appendix 3-A	127
Appendix 3-B.....	155
Appendix 4-A	161
References.....	164

List of algorithms

Algorithm 4-1: Internal cluster switch connection (ISwitch) for store-and-forward flow control mechanism	59
Algorithm 4-2: External cluster switch connection (ESwitch) for store-and-forward flow control mechanism	60
Algorithm 4-3: Message probability in the internal cluster and external cluster	62
Algorithm 4-4: Process flow in calculating the communication latency of interconnection networks in MCMCA	66
Algorithm 4-5: Pseudocode of process flow in calculating the communication latency of interconnection networks in MCMCA using MATLAB based on Store-and-Forward flow control	67
Algorithm 5-1: Process flow in calculating the communication latency of interconnection networks in MCMCA based on a wormhole flow-control mechanism	88
Algorithm 5-2: Pseudocode of process flow in calculating the communication latency of interconnection networks in MCMCA using MATLAB based on wormhole flow control	89

List of figures

Figure 2-1: Basic design of single-core processor	9
Figure 2-2: Basic design of multi-core processor.....	10
Figure 2-3: Cluster architecture (reproduced from (Baker et al., 2000)).....	13
Figure 2-4: Illustration of a Single-core cluster basic structure	16
Figure 2-5: Illustration of Multi-core Clusters basic structure	17
Figure 2-6: Communication level in Multi-core cluster.....	18
Figure 2-7: An illustration of the subdivision of a message into packets and of packets into flits (reproduced from (Dally & Towles, 2004))	21
Figure 2-8: An 8-port 2-tree constructed by proposed algorithm	24
Figure 2-9: The incorporate entities in MCMCA	28
Figure 3-1: Overview of the proposed Multi-Core Multi-Cluster Architecture (MCMCA)	30
Figure 3-2: Communication for message passing between two processor cores on the same chip	32
Figure 3-3: Communication for message passing across processors in different chips, but within a node.....	32
Figure 3-4: Communication routes for messages passing between processors on different nodes, but within the same cluster	33
Figure 3-5: Communication routes for transmitting messages between clusters	34
Figure 3-6: Queuing network of single-core cluster	35
Figure 3-7: Queuing network of multi-core cluster.....	36
Figure 3-8: Queuing network of Multi-core Multi-cluster Architecture (MCMCA)	36
Figure 3-9: Activity diagram of a packet traversing the cluster node of a single-core processor.....	37
Figure 3-10: Activity diagram of a packet traversing the cluster node of a multi-core processor.....	38
Figure 3-11: Model Structure in OMNeT++ (Varga & Hornig, 2008).....	42
Figure 3-12: Bahman's model for 8-cluster system with $M=32$	44
Figure 3-13: MCMCA model with $C=8$, $M=32$ based on Store-and-Forward Flow Control	44
Figure 3-14: Bahman's model for 8-cluster system with $M=64$	45
Figure 3-15: MCMCA model with $C=8$, $M=64$ based on Store-and-Forward Flow Control	45
Figure 3-16: MCMCA Simulation Results based on Store-and-Forward Flow Control.....	47
Figure 3-17: MCMCA Simulation Results based on Wormhole Flow Control	47
Figure 4-1: Flow diagram of the Store-and-Forward flow control mechanism (reproduced from (Dally & Towles, 2004))	51

Figure 4-2: Activity diagram of MCMCA simulation model	58
Figure 4-3: Message latency and throughput results based on store-and-forward flow control mechanism with M=8KB using Network Bandwidth I	71
Figure 4-4: Message latency and throughput results based on store-and-forward flow control mechanism with M=8KB using Network Bandwidth II	72
Figure 4-5: Simulation results of the impact on the message latency with various number of clusters based on Network Parameter I	73
Figure 4-6: Simulation results of the impact on the message latency with numbers of clusters based on Network Parameter II	74
Figure 4-7: Simulation results of the impact on the message latency with various message lengths based on Network Parameter I	75
Figure 4-8: Simulation results of the impact on the message latency with various message lengths based on Network Parameter II	76
Figure 5-1: MCMCA model with C=8, M=32 flits, F=256 bytes based on Wormhole Flow Control	94
Figure 5-2: Message latency and throughput simulation results based on Wormhole Flow Control Mechanism with M=32F, F=256 bytes	95
Figure 5-3: Message latency and throughput simulation results compared to analytical calculation results based on Wormhole Flow Control with M=32 flits, F=256 bytes	95
Figure 5-4: Message latency and throughput simulation results based on Wormhole Flow Control Mechanism with M=32 flits, F=512 bytes	96
Figure 5-5: Message latency and throughput simulation results compared to analytical calculation results based on Wormhole Flow Control with M=32 flits, F=512 bytes	96
Figure 5-6: Message latency predicted by the simulation model with M=32 flits, F=256 bytes and $\lambda g=0.001$	98
Figure 5-7: Message latency predicted by the simulation model with M=32 flits, F=256 bytes and $\lambda g=0.002$	98
Figure 5-8: Message latency predicted by the simulation model with M=32 flits, F=512 bytes and $\lambda g=0.001$	99
Figure 5-9: Comparison of message latency with M=32 flits, F= 256 and 512 bytes predicted by the simulation model using $\lambda g=0.001$	99
Figure 5-10: Simulation results of the impact on the message latency with various message lengths based on Number of cluster = 8	101
Figure 5-11: Simulation results of the impact on the message latency with various message lengths based on Number of cluster = 32	101

Figure 6-1: Histogram with distribution curve plotted for store-and-forward flow control mechanism with 1-core	109
Figure 6-2: Histogram with distribution curve plotted for wormhole flow control mechanism with 1-core	109
Figure 6-3: Histogram with distribution curve plotted for store-and-forward flow control mechanism with 2-core	109
Figure 6-4: Histogram with distribution curve plotted for wormhole flow control mechanism with 2-core	109
Figure 6-5: Histogram with distribution curve plotted for store-and-forward flow control mechanism with 4-core	110
Figure 6-6: Histogram with distribution curve plotted for wormhole flow control mechanism with 4-core	110
Figure 6-7: Error bar chart of average message latency between store-and-forward flow control and wormhole flow control	111

List of tables

Table 3-1: A comparison of OMNeT++ with other simulation tools	41
Table 3-2: Interconnection network parameters	43
Table 3-3: Model cases for single-core clusters	43
Table 3-4: Model cases for multi-core clusters	46
Table 4-1: Notations used in MCMCA	52
Table 4-2: Interconnection Networks Parameter I.....	70
Table 4-3: Simulation Input I	70
Table 4-4: Interconnection Networks Parameter II.....	73
Table 4-5: Simulation Input II	75
Table 4-6: Simulation Input III	75
Table 5-1: Notations used in MCMCA for Wormhole Flow Control.....	80
Table 5-2: Interconnection Networks Parameters	93
Table 5-3: Input Parameters	93
Table 5-4: Baseline results comparison between MCMCA with 1-core with multi-cluster model presented by Javadi et al. (2006)	94
Table 5-5: Simulation Input for cluster sizes	97
Table 5-6: Simulation Input for scalability.....	100
Table 6-1: Descriptive Results for Store-and-Forward Flow Control	106
Table 6-2: Descriptive Results for Wormhole Flow Control.....	106
Table 6-3: SPSS Output for S-W tests of normality using Store-and-Forward Flow Control Mechanism	107
Table 6-4: SPSS Output for S-W tests of normality using Wormhole Flow Control Mechanism	107
Table 6-5: Descriptive Statistics for Skewness with Store-and-forward flow control ...	108
Table 6-6: Descriptive Statistics for Skewness with Wormhole flow control.....	108
Table 6-7: The comparison of flow control mechanism performance using Mann- Whitney test	111
Table 6-8: Comparison between processor core performance with two flow control mechanism	113
Table 6-9: Pairwise comparison between processor core performance for two flow control mechanism	113
Table 6-10: Comparison between processor core performance with Store-and-Forward flow control mechanism.....	113
Table 6-11: Pairwise comparison between processor core performance with Store-and- Forward flow control mechanism	113

Table 6-12: Comparison between processor core performance with Wormhole flow control mechanism..... 114

Table 6-13: Pairwise comparison between processor core performance with Wormhole flow control mechanism 114

Declaration of Authorship

I, **Norhazlina Hamid**,

declare that this thesis and the work presented in it are my own and have been generated by me as the result of my own original research.

INTERCONNECTION NETWORKS PERFORMANCE MODELLING FOR MULTI-CORE MULTI-CLUSTER ARCHITECTURE

I confirm the following:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.
- Either none of this work has been published before submission, or parts of this work have been published as:
 1. Hamid, N., Walters, R. J. and Wills, G. B. (2014), Performance evaluation of multi-core multi-cluster architecture, presented at *Emerging Software as a Service and Analytics, Barcelona, ES, 03 - 05 Apr 2014*. Scitepress9pp, 46-54.
 2. Hamid, N., Walters, R. J. and Wills, G. B. (2014), Analytical Calculation of Multi-Core Multi-Cluster Architecture (MCMCA), presented at *International Conference of Postgraduate in Education, Melaka, Malaysia, 17 - 18 Dec 2014*.

3. Hamid, N., Walters, R. J. and Wills, G. B. (2015), An Architecture for Measuring Network Performance in Multi-Core Multi-Cluster Architecture (MCMCA). In, *International Journal of Computer Theory and Engineering* vol. 7, no. 1, pp. 57-61, February 2015.
4. Hamid, N., Walters, R. J. and Wills, G. B. (2015), Performance evaluation of multi-core multi-cluster architecture (MCMCA). In, Chang, Victor, Walters, Robert John and Wills, Gary (eds.) *Delivery and Adoption of Cloud Computing Services in Contemporary Organizations*. Hershey, US, IGI Global.
5. Hamid, N., Walters, R. J. and Wills, G. B. (2015), An analytical model of multi-core multi-cluster architecture (MCMCA). In *Open Journal of Cloud Computing (OJCC)*, 2, (1), 1-12.
6. Hamid, N., Walters, R. J. and Wills, G. B. (2015), Simulation and Mathematical Analysis of Multi-core Cluster Architecture, presented at *17th International Conference on Computer Modelling and Simulation (UKSim2015)*, Cambridge, United Kingdom, 27-29 March 2015.
7. Hamid, N., Walters, R. J. and Wills, G. B. (2015), Interconnection Network Performance of Multi-core Cluster Architecture, presented at *2nd International Conference on Computer, Communication and Control Technology (I4CT)*, Sarawak, Malaysia, 21-23 April 2015.
8. Hamid, N., Walters, R. J. and Wills, G. B. (2015), "Understanding the Impact of the Interconnection Network Performance of Multi-core Cluster Architectures," presented at *4th International Conference on Computer Technology and Science (ICCTS)*, Bandar Seri Begawan, Brunei, 01-02 June 2015.
9. Hamid, N., Walters, R. J. and Wills, G. B., "Understanding the Impact of the Interconnection Network Performance of Multi-core Cluster Architectures," *Journal of Computers*, vol. 11, no. 2, pp. 132-139, 2016.

Signed:

Date: 17 June 2016

Acknowledgements

All praise to Allah SWT, the most Gracious, the most Compassionate. Alhamdulillah, I am given this golden opportunity, will and strength to pursue my PhD. I am absolutely delighted that I finally made it, and this chapter in my life is finally over!

Certainly, this PhD journey may never see the light without the support and encouragement from many wonderful and brilliant people who have helped me in various ways towards the completion of my study.

My sincere and deepest appreciation goes to my superb supervisors, Dr. Robert J. Walters and Dr. Gary B. Wills. Indeed, I am so grateful to have them as my supervisors who have taught me so many priceless lessons which I will never forget. A very big thank you to them for the guidance, advice, patience, and continuous support throughout my study. I am truly indebted for their tremendous effort to supervise me, together with the friendship forged throughout this journey. The success of this research would not be possible without their excellent supervision.

I would also like to extend my appreciation to my internal examiner, Mr. Lester Gilbert, and my external examiner, Professor Vassil Alexandrov. Their invaluable advice and constructive comments were certainly helpful in improving this thesis.

To my beloved family, I would like to thank them for all their prayers, love and encouragement. To my beloved parents, Dato' Haji Abdul Hamid Hamat, and Datin Hajah Wan Khuzaimah Wan Mamat, I would like to thank them for their continuous prayers, support, and unconditional love throughout all my pursuits. A big thank you too to my mother-in-law, Hasmah Jusoh, for her prayers while we were thousand miles away from her. I believe that it was the prayers of my loved ones who made me sustain my PhD journey.

A special dedication goes to the most loving, supportive and encouraging husband, Saipulbahri Mustapha, and my three lovely children, Arif Fikri, Sarah Alya and Adam Haris, for their patience, love, understanding, faith and support during this challenging yet rewarding period. I am truly blessed with a great family and am certainly looking forward to the arrival of my baby Viva in July 2016!

Norhazlina Hamid

Thank you to my siblings, all my family members, my in-laws, my friends especially in PERJASA Group and DSG Group, and certainly the Malaysian community in Southampton, who have indirectly contributed to my success.

Thank you to all my friends at the ECS laboratory for their great help, and for motivating me to persevere and strive towards my goal. A special appreciation goes to Dr. Sei Ping Lau for his knowledge, time and continuous help in my simulation development which became an instrumental part of my thesis.

I would like to acknowledge the IRIDIS High Performance Computing Facility, and the associated support services at the University of Southampton, for the services and facilities offered to help with the completion of this work.

I would also like to acknowledge the Public Service Department of Malaysia for the scholarship awarded and the financial support to pursue my studies at this highly acclaimed University of Southampton, and fulfill my dream.

Last but not least, to all the many other amazing people I have met, who have contributed directly or indirectly towards this wonderful journey, from the bottom of my heart: Thank you, Thank you, and Thank You Very Much!

Definitions and Abbreviations

Analytical Model	A set of equations describing the performance of the cluster to support the simulation analysis.
Average message latency	The average amount of time elapsed from the generation of a message until the last packet reaches the destination node.
Bandwidth	The capacity of a network connection for supporting data transfers.
Blocking	A network is blocking if it cannot handle all switch requests that are a permutation of the inputs and outputs.
Buffered flow control	Store a packet in a buffer, preventing the waste of channel bandwidth caused by dropping or misrouting packets.
Bufferless flow control	Uses no buffering and simply allocates channel and bandwidth to competing packets.
Chip	A complex and tiny modules that store computer memory or provide logic circuitry for processors.
Clock cycle	The time measured between two adjacent pulses of the oscillator and sets the tempo of the computer processor.
Clock rates	An indicator of the processor's speed and typically refers to the frequency.
Cluster computing	A form of computing in which a group of computers are linked together to act like a single entity.
Channel bandwidth	Transports messages between nodes and buffers, such as registers and memories, which allow messages to be held temporarily at the nodes.
Communication	The exchange of data between a source and a destination receiver in interconnection networks architecture.
Computer cluster	A group of computers connected to each other by fast local area networks which work together to form a single computer.
Computer	<p>A piece of electronic equipment which, when given some data, will process that data in some pre-defined way to produce the required results (Willis & Kerridge, 1983).</p> <p>A complete system, composed of many interacting parts while computer systems are made of hardware and software (Sullivan, Lewis & Cook, 1988).</p>
Computer system network	Essential elements for the computer network system to function include multiple servers, terminals, printers, network links, software, users, and support systems, including maintenance and repair, training, and spare parts.
Contention	Occurs when two or more messages want to use the same shared resource in the network.
Control state	Tracks the resources allocated to the packet within the node and the state of the packet's traversal across the node.
Core	A complete computational engine (Burger, 2005).

Deadlock	Occurs when resources are waiting on another set of resources to complete a work cycle.
Deterministic routing	The path a packet takes is only a function of its source and destination.
Flit	Flow control digits are the basic units of bandwidth and storage allocation used by most flow control mechanisms.
Flow control	Determines how a network's resources – such as channel bandwidth, buffer capacity and control state – are allocated to messages as they progress along their route in the network.
Frequency	The number of waves that pass a fixed place in a given amount of time.
Hardware	Parts of a computer that we can see and touch and the most important piece of hardware is a tiny rectangular chip inside our computer called the central processing unit (CPU), or microprocessor (Karmakar, 2011).
High Performance Computing (HPC)	Any computational activity requiring more than a single computer to execute a task.
Homogeneous network	A computer network composed of computers using similar configurations.
Interconnection Network	A physical connection between the different components of a parallel system.
Microprocessor	A computer that has been made on a single chip of silicon about 4 to 6 millimetres square and ½ millimetre thick, and contains a minimum of a few thousand transistors (Stevens, 1986).
Model	A model is a representation of an actual system (J. Banks, 1998) or process (Carson, 2005).
Modelling	The process of identifying and abstracting relevant entities and relationships from a system under study.
Model validation	Substantiating that a computerized model, within its domain of applicability, behaves with satisfactory accuracy consistent with the intended application of the model.
Model verification	Ensuring that the computer program of the computerized model and its implementation are correct.
MCMCA Simulation Model	A descriptive model to investigate the performance of multi-core cluster system using simulation model.
Message	A logically contiguous group of bits that are delivered from a source node to a destination node.
Message Passing Interface (MPI)	A library specification and standard for message-passing between multiple computers running a parallel program across distributed memory.
Multi-cluster	A multiple cluster system that is connected via the cluster interconnection networks where each cluster system/node has multiple processors (Shahhoseini, Naderi & Buyya, 2000).

Multi-core processor	A single processor with two processing cores which means to place two or more processing cores on the same chip (Burger, 2005).
Multi-core cluster	A cluster where all the nodes in the cluster have multi-core processors.
Multi-core multi-cluster	An architecture built up of numbers of clusters where each cluster is composed of numbers of nodes consist a number of processors in a single chip, each with two or more cores.
Network	Characterised by the media it uses to carry messages, the way the network links devices and the expansiveness of the network.
Network latency	A delay that happens as data packets transmits from one point to another over a network.
Node	Consists a multiple processor chip.
Operating system (OS)	Software that manages the computer and the devices connected to it, for example Windows or Linux.
Packet	The basic unit of routing and sequencing.
Personal Computer (PC)	A computer that is designed to be used by one person.
Processor	The logic circuitry that responds to and processes the basic instructions that drive a computer; in the simplest terms, the computer's brain (Rouse, 2006).
Poisson distribution	A probability distribution which expresses the probability of a number of procedures occurring in a fixed phase of time.
Random number generator	A program written for and used in probability and statistics applications when large quantities of random digits are needed.
Routing algorithm	Determines the path to be used for data transmission.
Scalability	An ability to radically change the size of something in order to meet the additional requirements of a resource.
Simulation	An imitation of a system as it progresses through time (Robinson, 2004) and contains a set of entities and relationships to fulfill a certain purpose (Wehrle, Gunes & Gross, 2010).
Simulation model	The process of creating and analysing a digital prototype of a physical model to predict its performance in the real world.
Single-core processor	A processor with only one processing core.
Single-core cluster	A cluster consists of numbers of single-core processor.
Single-core multi-cluster	An architecture built up of numbers of clusters where each cluster is composed of numbers of nodes that consist a number of single-core processors.
Software	The instruction or programs that tell the hardware what to do.
Store-and-forward flow control	A packet switching mechanism whereby the message to be transmitted is partitioned into a sequence of packets.

Supercomputers	Large mainframes used primarily for the analysis of scientific and engineering problems (Sullivan et al., 1988).
System	A construct or collection of different elements such as people, machines, resources, that together produce results not obtainable by the elements alone.
Throughput	The rate at which traffic is delivered to the destination.
Topology	The interconnection structure used to connect different processors or processors and memory modules.
Transistor	A semiconducting device that switches and amplifies electronic signal.
Transmission time	The network cycle time taken by a single packet to travel from one node to another node in the simulator.
Up*/Down* routing	An assignment of direction (up or down) to network channels where a spanning tree whose node (also called 'vertex') corresponds to a switch in the network, based on building a 'breadth-first search' (BFS) spanning tree used in Autonet (M. D. Schroeder et al., 1991)
Workstations	Type of computer that requires a moderate amount of computing power and relatively high quality graphics capabilities.
Wormhole flow control	A packet switching mechanism that works by dividing packets into a sequence of fixed-size units called 'flits', with channel and buffers allocated to flits.

Chapter 1 Introduction

The emergence of High Performance Computing (HPC), including Cluster computing, has improved the availability of high performance computers and high speed network technologies. High performance in this context is defined as a computational activity requiring more than a single computer to execute a task (Qian, 2010). The main target of HPC is better performance in computing and one of the aims is to leverage cluster computing to solve advanced computation problems (Hope & Lam, n.d.). Cluster computing is playing a major role in solving large-scale computing application problems as they need faster and more reliable systems, especially as they are often built using commodity-off-the-shelf (COTS) hardware components and commonly-used software (Hamid, Walters & Wills, 2015c).

The exponential growth in computing performance quickly led to more sophisticated computing platforms. This rapid growth increased the demand for faster computing performance: every new enhancement in processors leads to greater performance demands (Jin et al., 2011). Moore's Law predicted that the number of transistors on a processor would double approximately every two years, providing regular leaps in computing power (Moore, 1965). Over more than four decades, this has driven the impressive growth in computer speed and accessibility. However, Moore's Law has begun to show signs of failing, being replaced by the emergence of multi-core processors, which involves placing two or more processing cores within the same processor (Burger, 2005). This allows the processor to perform more work within a given clock cycle.

With the emergence of high-speed networks, High Performance Computing (HPC) has adopted network-based computing clusters as cost-effective platforms to achieve high performance, which has led the trend towards cluster systems with multi-cores: the multi-core cluster (Wu & Taylor, 2013). The multi-core cluster becomes more powerful due to the combination of faster processors, faster memory and faster interconnection (Bethel & Howison, 2012).

1.1 Research Motivation

Processor performance is often associated with high processor clock frequencies and increasing power dissipation. Every new performance enhancement in processors leads to greater performance demands. The demand for increasing performance continues and as single-core processors reach their physical limits of possible complexity and

speed, the movement towards multi-core processors begins. A multi-core processor means one processor with two or more complete computational engines (cores) within a single processor to enhance performance, reduce power consumption and permit simultaneous processing of multiple tasks (Al-Babtain, Al-Kanderi, Al-Fahad & Ahmad, 2013). Multi-core processors represent a major trend over the past decade and allow faster execution of applications by taking advantage of parallelism.

The Top 500 Supercomputer List published in June 2014 (Admin, 2014) showed that multi-core processors have been widely deployed in clusters of parallel computing, and more than 95% of the systems are using dual-core or quad-core processors. Another trend is reflected in the advances in multi-core processor technology that makes multi-core processors an excellent choice to use in clustered nodes (Soryani, Analoui & Zarrinchian, 2013). Many studies (Fengguang, Moore & Dongarra, 2009; Ichikawa & Takagi, 2009; Lei, Hartono & Panda, 2006; Ranadive, Kesavan, Gavrilovska & Schwan, 2008) have been carried out to improve the performance of multi-core clusters but few clearly distinguish a key issue: that of the performance of the interconnection networks. The existing multi-core cluster architectures are therefore unable to capture the potential performance and the characteristics of the traffic of the interconnection networks within the implementation of a multi-core cluster architecture. The cluster interconnection network is nonetheless critical for delivering efficiency and scalability for the applications, as it needs to handle the networking requirements of each processor core (Dally & Towles, 2004; Shainer et al., 2013). Even so, existing multi-core cluster architectures do not address the potential performance issues of the interconnection networks within multi-core clusters.

In a multi-core cluster architecture, multiple computing nodes are connected via the cluster interconnection network. The implementation of the architecture typically imposes higher latency for communication between processors located on different nodes compared with the processors located on the same nodes.

Apart from addressing the above concern, this thesis will expand the multi-core cluster architecture to a more scalable approach by applying multi-cluster architecture. Existing studies (Abdelgadir, Pathan & Ahmed, 2011) have found that having a good network bandwidth and a faster network will produce a better performance in relation to the scalability of the clusters. The conventional approach to improving cluster throughput is to add more processors, but there is a limit to the scalability of this approach: the infrastructure cannot provide effective memory access to unlimited numbers of processors and the interconnection network(s) become saturated (Shahhoseini et al., 2000). This thesis will identify the potential scalability of the multi-core cluster and will expand the architecture by employing a multi-cluster

architecture. The combination of multi-core cluster and multi-cluster architecture presents a novel architecture known as Multi-core Multi-cluster Architecture (MCMCA). The research described here is motivated by the fact that it is considered to be the first investigation of interconnection network performance of multi-core multi-cluster architecture.

Multi-core clusters allow for the faster execution of applications by taking advantage of the ability to work on multiple cores simultaneously. Several performance models of cluster systems have been proposed in (Alzeidi, Ould-Khaoua & Khonsari, 2008; Geyong, Yulei, Ould-Khaoua, Hao & Keqiu, 2009; Bahman Javadi, Abawajy & Akbari, 2008b; Sarbazi-Azad, Ould-Khaoua & Mackenzie, 2001; Yulei, Geyong, Keqiu & Javadi, 2012), but the evaluations are confined to a single-core processor in a cluster. In order to take advantage of a multi-core processor in a cluster system, it is important to have an in-depth understanding of the characteristics of multi-core clusters and their impact on application performance and behaviour.

This research also develops novel simulation models for predicting and investigating the MCMCA interconnection network performance. The main performance metrics to be simulated are the latency, network throughput and bandwidth. Latency is the time required for a packet to travel from a source node to a destination node; network throughput is the rate at which the networks sends or receives data, and bandwidths refers to the maximum rate at which a packet can be transferred. The model is then used to evaluate the impact of performance metrics on scalability and cluster size.

1.2 Research Objectives

The thesis objectives are to investigate and evaluate the interconnection network performance of multi-core multi-cluster architecture (MCMCA). The main research question is:

RQ1: What is an appropriate architecture to investigate the communication latency of multi-core processors in multi-cluster?

Hence, this RQ1 addresses the following sub-research questions:

SQ1: What are the appropriate characteristics to be considered in designing cluster architecture?

SQ2: What is an appropriate simulation model to investigate interconnection network performance?

SQ3: How well does the MCMCA simulation model analyse cluster performance?

RQ2: What is an appropriate flow control mechanism for communication latency modelling of the Multi-core Multi-cluster Architecture (MCMCA)?

Hence, this RQ2 addresses the following sub-research questions:

SQ4: What is the impact of the flow control mechanism in improving communication latency?

In order to achieve this goal, three research hypotheses are tested:

Hypotheses H1: *Employing a multi-core processor in multi-cluster architecture will improve the performance of a cluster system.*

Hypotheses H2: *The proposed simulation model can be used to investigate the interconnection network performance in MCMCA.*

Hypotheses H3: *The common flow control mechanism can be employed to evaluate the impact of MCMCA on interconnection network performance*

1.3 Published Papers

The research undertaken in this thesis has contributed in part or full to the following publications:

1. Hamid, N., Walters, R. J. and Wills, G. B. (2014), Performance evaluation of multi-core multi-cluster architecture, presented at *Emerging Software as a Service and Analytics, Barcelona, ES, 03 - 05 Apr 2014*. Scitepress 9pp, 46-54.
2. Hamid, N., Walters, R. J. and Wills, G. B. (2014), Analytical Calculation of Multi-Core Multi-Cluster Architecture (MCMCA), presented at *International Conference of Postgraduate in Education, Melaka, Malaysia, 17 - 18 Dec 2014*.
3. Hamid, N., Walters, R. J. and Wills, G. B. (2015), An Architecture for Measuring Network Performance in Multi-Core Multi-Cluster Architecture (MCMCA). In, *International Journal of Computer Theory and Engineering* vol. 7, no. 1, pp. 57-61, February 2015.
4. Hamid, N., Walters, R. J. and Wills, G. B. (2015), Performance evaluation of multi-core multi-cluster architecture (MCMCA). In, Chang, Victor, Walters, Robert John and Wills, Gary (eds.) *Delivery and Adoption of Cloud Computing Services in Contemporary Organizations*. Hershey, US, IGI Global.

5. Hamid, N., Walters, R. J. and Wills, G. B. (2015), An analytical model of multi-core multi-cluster architecture (MCMCA). In *Open Journal of Cloud Computing (OJCC)*, 2, (1), 1-12.
6. Hamid, N., Walters, R. J. and Wills, G. B. (2015), Simulation and Mathematical Analysis of Multi-core Cluster Architecture, presented at *17th International Conference on Computer Modelling and Simulation (UKSim2015)*, Cambridge, United Kingdom, 27-29 March 2015.
7. Hamid, N., Walters, R. J. and Wills, G. B. (2015), Interconnection Network Performance of Multi-core Cluster Architecture, presented at *2nd International Conference on Computer, Communication and Control Technology (I4CT)*, Sarawak, Malaysia, 21-23 April 2015.
8. Hamid, N., Walters, R. J. and Wills, G. B. (2015), "Understanding the Impact of the Interconnection Network Performance of Multi-core Cluster Architectures," presented at *4th International Conference on Computer Technology and Science (ICCTS)*, Bandar Seri Begawan, Brunei, 01-02 June 2015.
9. Hamid, N., Walters, R. J. and Wills, G. B., "Understanding the Impact of the Interconnection Network Performance of Multi-core Cluster Architectures," *Journal of Computers*, vol. 11, no. 2, pp. 132-139, 2016.

1.4 Thesis Structure

This thesis is divided into seven chapters. This first chapter provides an overview of the research motivation, the research hypotheses and the research questions. The rest is organised as follows:

Chapter 2 presents a review of the literature on related research domains: multi core clusters. This chapter specifically highlights the importance of particular research topics and shows how they benefit the main contribution. This chapter also provides some background to modelling and simulation, including the simulation model structure.

Chapter 3 introduces the new Multi-core Multi-cluster Architecture (MCMCA) and its interconnection network. The research methodology involved with baseline experimental results is also covered in this chapter.

Chapter 4 presents the new simulation model with an analytical model as a validation for the simulation results based on a store-and-forward flow control mechanism.

Chapter 5 presents the new simulation model with an analytical model as a validation to the simulation results based on a wormhole flow control mechanism.

Chapter 6 discusses the statistical analysis and findings of the performance model.

Chapter 7 draws conclusions from the reported results, research limitations and also suggests a future direction for this research.

Chapter 2 Literature Review

2.1 Introduction

In the early days of personal computing, personal computers (PCs) were stand-alone devices with single-user operating systems. User interaction occurred via text-based interfaces and only one program would run at a time. Over time, however, the exponential growth in computing performance quickly led to a more sophisticated computing platform.

This chapter provides a background to the multi-core cluster that employs the characteristics of cluster computing and multi-core processors. Accordingly, this chapter starts in section 2.2 with an introduction to computers and processors, followed by an introduction to multi-core processing in section 2.3. This is followed in section 2.4 by an introduction to cluster computing. Since the multi-core cluster is a relatively new architecture (Wu & Taylor, 2013), it is important to have an in-depth understanding of the application behaviours and trends in order to obtain optimal performance. Thus section 2.5 provides reviews of the single-core cluster, to be compared with the multi-core cluster in section 2.6. Section 2.7 introduces the structure of multi-cluster architecture and section 2.8 introduces the differences between message, packet and flits. The background of the simulation model structure is presented in section 2.9 as an overview of the development of the new simulation model.

This chapter aims to clarify the apparent ambiguity by comparing multi-core clusters with traditional single-core clusters. Challenges and issues are identified, demonstrating why this study focuses on the issue of an interconnection network in a multi-core cluster.

2.2 Computers and Processors

A computer is a piece of electronic equipment which, when given some data, will process that data in some pre-defined way to produce the required results (Willis & Kerridge, 1983). Willis and Kerridge also state that a 'computer system' is a computer with software that makes the system more flexible in solving many different types of problem. A 'personal computer' (PC) is a computer system that is designed to be used by one person.

Every computer system has hardware components that perform four basic functions: input, output, processing and storage. In their book, Sullivan et al. (1988)

define a computer as a complete system composed of many interacting parts, while 'computer systems' are composed of hardware and software. 'Hardware' refers to the parts of a computer that we can see and touch, and the most important piece of hardware is the tiny rectangular chip inside the computer called the central processing unit (CPU), or microprocessor (Karmakar, 2011). Other hardware items are the monitor, keyboard, mouse, printer and devices such as a webcam. 'Software' refers to the instructions or programs that tell the hardware what to do (Kim & Bond, 2009). The operating system (OS) – for example Windows or Linux – is software that manages the computer and the devices connected to it. 'Workstations' refer to a type of computer that requires a moderate amount of computing power and relatively high-quality graphics capabilities, while 'supercomputers' are large 'mainframes', used primarily for the analysis of scientific and engineering problems (Sullivan et al., 1988). 'Mainframes' are very large computers that are built to perform complex and critical applications.

The 'processor' is the logic circuitry that responds to and processes the basic instructions that drive a computer; in the simplest terms, it is the computer's brain (Rouse, 2006). It is the part that translates instructions and performs calculations. Stevens (1986) mentioned that the term 'processor' has generally replaced the term 'central processing unit' (CPU), and the processor in a personal computer, or one embedded in a small device, is often called a 'microprocessor'. A microprocessor is a computer processor that has been made on a single chip of silicon about 4 to 6 millimetres square and $\frac{1}{2}$ millimetre thick, and contains a minimum of a few thousand transistors (Stevens, 1986). A transistor is a semiconducting device that switches and amplifies electronic signal, which is also an essential component in CPU (Intel, 1997). Stevens also points out that most of the chips called microprocessors are not complete computers, but rather the central processing units (CPUs) of computers. Personal computer systems are usually built from CPU-only chips and the term 'microprocessor' is therefore applied to any chip that contains a whole CPU. Each generation of processors has grown smaller and faster, dissipating more heat and consuming more power (Schauer, 2008).

Processors were originally developed with only one core, as in Figure 2-1. A 'core' is a complete computational engine or the processing element in a processor (Varghese, McKee & Alexandrov, 2010). 'Single-core' denotes only one processing element within a single processor, and a 'multi-core' processor combines two or more processing elements in a single processor, on a single chip or multiple chips (Roy, 2008). A single-core processor can perform one task at a time whereas a multi-core processor can divide the work between two or more 'execution cores', allowing more work to be done within a given clock cycle (Burger, 2005). A 'clock cycle' is the time measured between two adjacent pulses of the oscillator and sets the tempo of the

computer processor. Thus a 'dual-core' processor contains two cores (such as the Intel Core Duo), a 'quad-core' processor contains four cores (e.g. AMD Phenom II X4) and a 'hexa-core' processor contains six cores (e.g. Intel Core i7 Extreme Edition 980X).

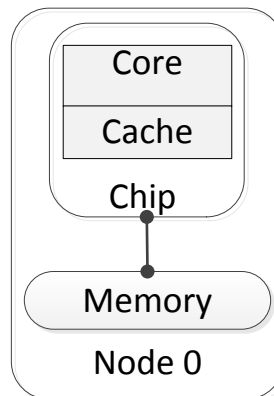


Figure 2-1: Basic design of single-core processor

2.3 Multi-core Processors

In 1965, Gordon Moore made his famous observation, the so-called 'Moore's Law', which predicted that the number of transistors per integrated circuit would double every year and that the speed would double every two years (Intel, 1997). Over more than four decades, this has driven the impressive growth in computer speed and accessibility.

In the past, the trend was to increase a processor's speed to get better performance. Transistor size had been reduced to increase the number of transistors that could be applied to processor functions and reduce the distance that signals must travel (Schauer, 2008). This allowed processor clock rates to soar. However, Lei, Qi and Panda (2007) have pointed out that nowadays it has become more difficult to speed up processors by increasing frequency. Frequency describes the number of waves that pass a fixed place in a given amount of time or the processor's speed (Stevens, 1986). As processor frequencies increase, the amount of heat produced by the processor increases (Pase & Eckl, 2005). The solution is to reduce the transistor size – because smaller transistors can operate at lower voltages, and this allows the processor to produce less heat. Unfortunately, David Geer (2005) demonstrated that, as a transistor gets smaller, it will be less able to block the flow of electrons. Also, smaller transistors keep using electricity even when they aren't switching, which wastes the power. However, transistors can't shrink forever, and chip manufacturers have struggled to cap power usage and heat generation which slow the processor performance (Chan, Ling & Aubanel, 2012). For these reasons, computer engineers are building a processor

with more processing cores, which means placing two or more processing cores on the same chip (Burger, 2005).

Multi-core processors are the solution to the deficiencies of single-core processors, as they increase bandwidth while decreasing power consumption (Burger, 2005). Multi-core processors have been developed to adhere to reasonable power consumption and heat dissipation. By dividing the workload among different cores, multi-core processors can speed up application performance by running at lower frequencies while minimising heat generation and the use of power (Lei et al., 2007). Multi-core processors do not necessarily run as fast as the highest-performing single-core models, but they improve overall performance by handling more work in parallel (Geer, 2005). Basic design of a multi-core processor is seen in Figure 2-2.

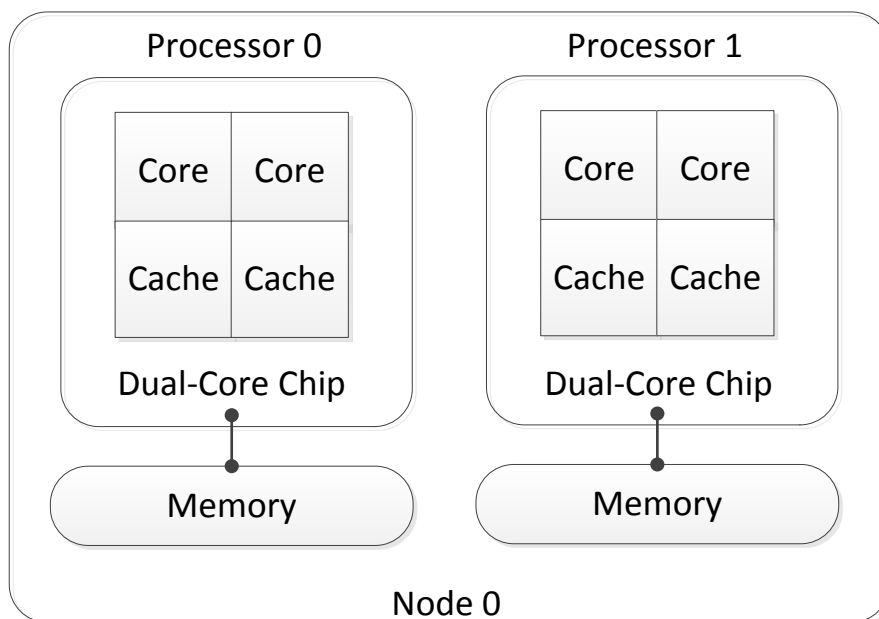


Figure 2-2: Basic design of multi-core processor

Despite such huge performance potential, many issues remain unsolved and need further attention – such as high-latency communication in interconnection networks (Rauber & Runger, 2010). Communication in this work refers to the exchange of data between a source and a destination receiver in interconnection networks architecture. Multiple cores on a single processor give rise to some problems and challenges, as follows (Hamid et al., 2015c; Karmakar, 2011; Lei et al., 2007; Schauer, 2008; Soryani et al., 2013):

- Interconnection networks: A faster network means a lower latency in network communication and memory transactions. Extra memory will be useless if the amount of time required for the memory request does not improve.

- Power and temperature: To reduce unnecessary power consumption and lessen the heat, the design model must run the multiple cores at a lower frequency. It must also ensure heat dissipation is distributed across the processor while being careful not to form any hot spots.
- Cache coherence: Since each core has its own cache, the copy of the data in that cache may not always be the most up-to-date version. This may produce invalid results.
- Multi-threading: Programmers have to write applications with subroutines able to be run on different cores. This is to ensure that the full advantage of multi-core capabilities can be exploited.
- Improved memory system: Larger caches sizes are needed for multi-threaded multi-core processors.
- Parallel programming: Programmers need to learn how to write parallel programs that can be split up and be run concurrently on multiple cores.
- Starvation: If a program is not developed correctly for use in a multi-core processor, one or more cores may be starved for data. The thread would simply run in one of the cores while the other cores sit idle.
- Homogeneous vs heterogeneous cores: Homogeneous cores have the same cache sizes, equivalent frequencies, functions, etc., while each core in a heterogeneous system may have a different function, frequency, memory model etc.

2.3.1 Moore's 'Law' Changes with Multi-core Processor

For years, Moore's 'Law' has been the one of the guiding principles of computer architecture. Instead of increasing clock speeds, which allows software to automatically run faster, chip manufacturers have been increasing the number of cores on a single chip (Bethel & Howison, 2012). Multicore or many cores, again embodying Moore's 'Law', has become one of the important technologies in the electronic chip industry. It is foreseeable that hundreds of cores on a single chip will appear in the future (Holt, 2016). With the continuously increasing number of cores, it is important to fully harness the abundant computing resources with programming models that are still easy-to-use (Bethel & Howison, 2012). It is more difficult to speed up processors nowadays by increasing frequency.

The prevalence of multi-core processors beyond quad-cores forms the building block of high performance computing architectures (Chan et al., 2012). Multi-core processors have started to put pressure on the application development and programming language development. Taking advantage of multi-core processors requires that software is able to split the work among cores. The software must be

designed to support parallelism to use multi-core processors and to avoid stagnant performance (Lin-Dong, De-Yu, Qiang & Jin-Xin, 2014). Exploiting parallelism as the number of cores grows is a challenge (Donald & Martonosi, 2006). Thus, switching to multi-core processors chips grows the technological frontier asymmetrically by benefiting parallelized software.

Even after 50 years, Moore's 'Law' still leaves an impact and benefits in many ways (Intel, 2015). For the technological impact, Moore's observation transformed computing into a pervasive and affordable necessity. The technologies also created two important keys to technology development, performance and cost. Processing power has been increased and energy efficiency improved at a lower cost by applying multi-core processor technology, which has had an economic impact in the chip industry. These drivers have set the pace for innovation and development, but more research is being conducted into computing performance to investigate how the impact of multi-core processor compares to Moore's 'Law' prediction.

2.4 Cluster Computing

Cluster computing is a form of computing in which a group of computers are linked together to act like a single entity (Baker & Buyya, 1999a). Cluster computing was first developed in the 1960s by IBM (Admin, 1999) as an alternative way of connecting large mainframes to provide a more cost effective form of commercial parallelism (Buyya, Hai & Cortes, 2002). However, cluster computing only gained momentum after the convergence in the 1980s of high performance microprocessors, high-speed networks and standard tools for high performance distributed computing. The recent advances in these technologies are making clusters an appealing solution for cost-effective parallel computing, and have emerged as mainstream parallel platforms for high-performance, high-throughput and high-availability computing.

A computer cluster can be defined as "*a collection of individual computers*" (Baldassari, Kopec, Leshay, Truszkowski & Finkel, 2005) "*connected to each other by fast local area networks*" (Baker & Buyya, 1999b) which "*work together to form a single computer*" (Baker, Apon, Buyya & Jin, 2000). A computer cluster works as one to execute intensive computation that would be not feasible on a single computer. Each individual computer in a cluster represents a computer node. Computer nodes are capable of full independent operation and are employed individually for stand-alone workloads and applications (T. Sterling, Apon & Baker, 2000). The nodes may incorporate a single processor or multiple processors with memories and operating systems (Hamid, Walters & Wills, 2014). The key components of a cluster include multiple stand-alone computers (PCs, workstations or SMPs), high-performance

interconnects, parallel programming environments and applications (Baker & Buyya, 1999b). The typical architecture of a cluster is shown in Figure 2-3.

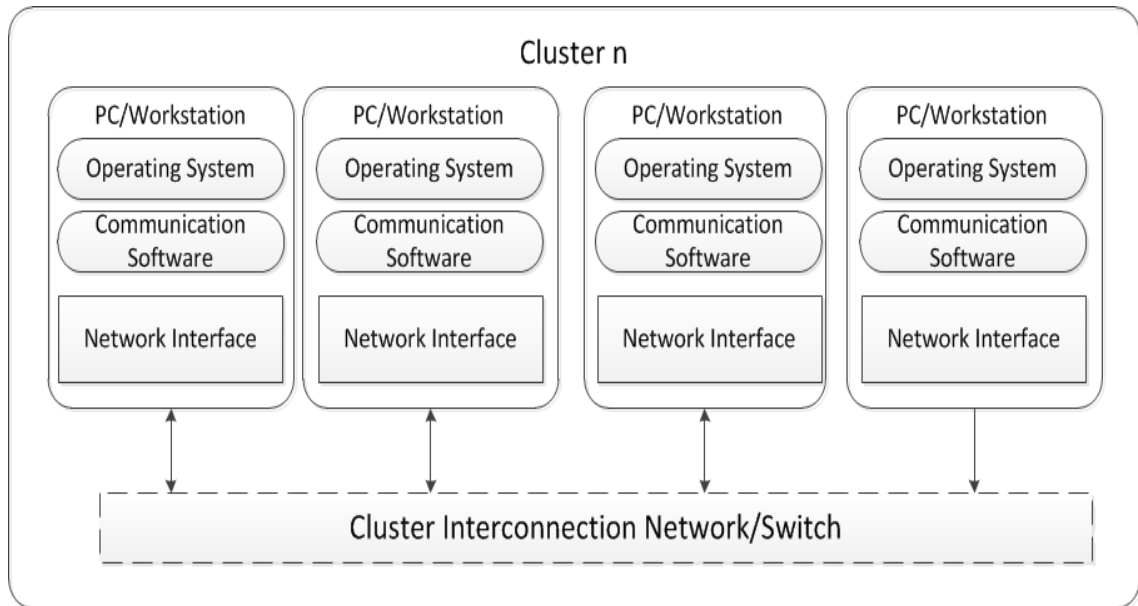


Figure 2-3: Cluster architecture (reproduced from (Baker et al., 2000))

There are many kinds of computer cluster, ranging from some which employ the world's largest computers to collections of personal computers. Clustering was among the first computer system architecture techniques to achieve significant improvements in overall performance, user access bandwidth and reliability (T. L. Sterling, 2002). In their 2002 paper, Buyya, Hai and Cortes described cluster computing as a fusion of the fields of parallel, high-performance, distributed, and high-availability computing. Despite the definition and stated key components, there are a number of important cluster features (Baker et al., 2000; Buyya et al., 2002; Goscinski, Hobbs & Silcock, 2001) that need to be noted, such as:

- Clusters are generally localized within a room or building.
- Clusters have a single administration.
- Clusters primarily focus only on compute-intensive problems and HPC.
- Clusters are typically homogeneous, based on a single processor and operating system.
- Clusters are static in nature, with fixed sets of processors and resources,

A cluster is deployed to increase performance and availability, and clusters are more cost-effective than a single computer. However, for all the benefits, there are challenges in cluster computing, as follows (Sadashiv & Kumar, 2011; Srinivas & Ramasubramaniam, 2011):

- Middleware: the need for a software environment that provides an illusion of a single system image, rather than a collection of independent computers.
- Program: applications that run on the clusters must be explicitly written so as to incorporate the division of tasks between nodes.
- Elasticity: providing a capability to adapt to changing potential requirements, for example the variance in real-time response time when the number of service requests changes dramatically.
- Scalability: an ability to scale up in order to meet the additional requirements of a resource. This can affect the performance of the system.

2.4.1 Cluster Interconnection Networks

An interconnection network is a physical connection between the different components of a parallel system, and it can be used with a multi-core cluster system. A network is characterised by the media it uses to carry messages, the way the network links devices, and the expansiveness size of the network (Sullivan et al., 1988). In multi-core cluster systems, the interconnection network is used to connect the nodes to each other (Peh, 2001) and is used to connect the processors to the memory modules. In a network, a node is a connection point, either a redistribution point or an end point for data transmissions (Rouse, 2006).

The main task of the interconnection network is to transfer messages from a specific processor to a specific destination, which can be another processor or a memory module (Gramsamer, 2003). The objective for the interconnection network is to perform the message transfer correctly as quickly as possible, even if several messages have to be transferred at the same time (Rauber & Runger, 2010). Interconnection networks are an attractive alternative to dedicated wiring because they allow limited wiring resources to be shared by several low-duty-factor signals (Dally & Towles, 2004).

Interconnection networks are critical in achieving high performance in clusters (Shainer et al., 2013). While ideal networks support both high bandwidth and low latency, there often exists a trade-off between these two parameters (Dally & Towles, 2004). In this work, bandwidth represents the capacity of a network connection to support data transfers while latency is a delay that happens when data packets are transmitted from one point to another over a network. High network bandwidth with low network latency often refers to better performance (Tanenbaum, 1996). For example, a network that offers low bandwidth tends to keep the network resources busy, often causing contention for the resources, which will increase the latency of the messages. Contention will occur when two or more messages want to use the same shared resource in the network. With a good combination of network topology, routing

technique and flow control mechanism, this problem can be minimised (Dally & Towles, 2004).

Clusters need to incorporate a high performance interconnection network to support low latency and high bandwidth communication between cluster nodes. Slow interconnection networks had always been a critical performance bottleneck for cluster computing (Baker et al., 2000). Cluster interconnection networks enable messages to be transferred through a combination of hardware and software support between logical elements distributed among a set of separate processor nodes within a cluster (T. Sterling, Apon & Baker, 2000). The nodes in a cluster communicate over high-speed networks using a standard networking protocol such as TCP/IP or a low-level protocol. Some interconnect technologies used in high-performance computers include Gigabit Ethernet (Koibuchi et al., 2011), Myrinet (Petrini, Frachtenberg, Hoisie & Coll, 2003) and QsNet (Qian, 2010). Each interconnect provides a different level of programmability, raw performance and integration with the operating system.

Interconnection networks exhibit a wide range of communication behaviours and impose diverse requirements on the underlying communication architecture. Some problems require the high bandwidth and low latency found only in earlier parallel processing systems, and may not be well suited to clusters (Buyya et al., 2002). As the network performance attributes improve, the range of problems that can be effectively handled also expands. With the emergence of the multi-core cluster, it is increasingly important to understand the capabilities and potential performance of interconnection networks for clusters.

2.5 Single-core Clusters

A traditional cluster with single processor nodes is shown in Figure 2-4. A single processor is a processor which contains only one core. The most important characteristic of such a cluster is its uniformity, where each node is identical to every other node. Each node has its own dedicated memory and cache as well as its own path to the interconnection network. A cache is a place to store active data temporarily in computing to shorten access times, reduce latency and improve application performance (Karmakar, 2011). Based on a common algorithm such as the Message Passing Interface (MPI), any cluster with single processor nodes is equally good for running any process independently (Yeo et al., 2006). MPI is a library specification and standard for message-passing between multiple computers running a parallel program across distributed memory.

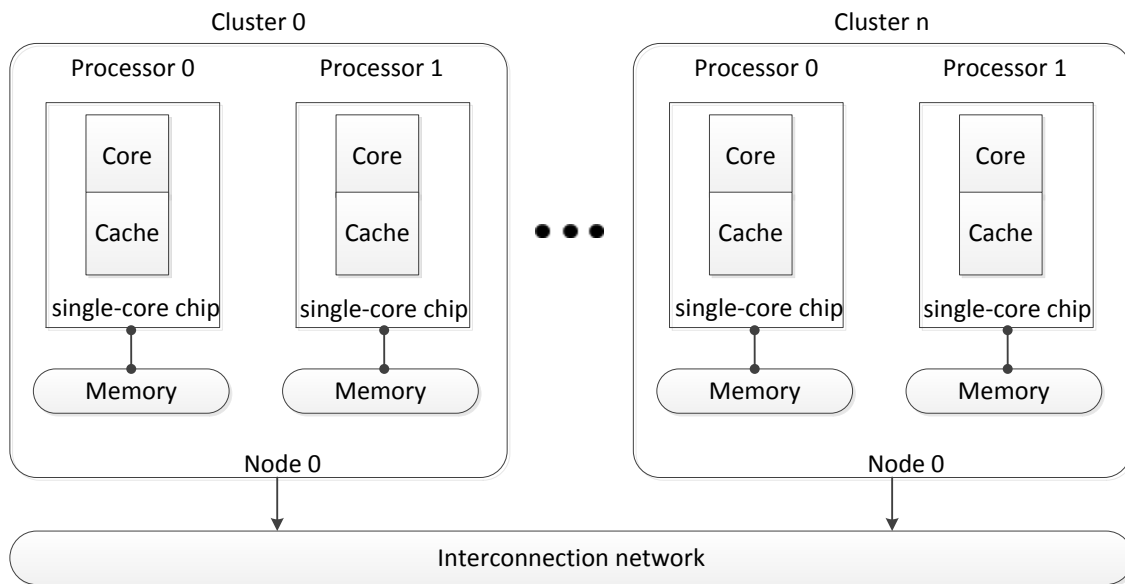


Figure 2-4: Illustration of a Single-core cluster basic structure

The performance of a single-core cluster depends on its processor frequency (Karmakar, 2011). Adding more single-core processors to the same chip would, in theory, result in twice the performance, although in practice the actual speed of each core is slower than the fastest single-core processor (Burger, 2005). This is due to latency in every communication level of the interconnection network in the cluster.

2.6 Multi-core Clusters

The convergence of high-speed networks in high performance computing has introduced network-based computing systems, i.e. clusters of multiprocessors. Earlier clusters were equipped with multiple single-core processors. The industry has therefore adopted the development of a chip with multiprocessors, or multi-cores, to overcome single-core cluster issues (Admin, 2014). The trade-off that must now be made is that each processor core is slower than a single-core processor (Hamid et al., 2015c). But two or more cores in a chip together may be able to provide greater throughput, even though the individual cores are slower (Geer, 2005). Each generation of multi-cores will thus likely increase the number of cores and decrease processing time.

The multi-core cluster is a cluster with multi-core processor nodes, as shown in Figure 2-5. Each cluster node has multiple processors, each of which contains multiple cores. With such cluster nodes, both the memory and the connection to the interconnection network are now shared.

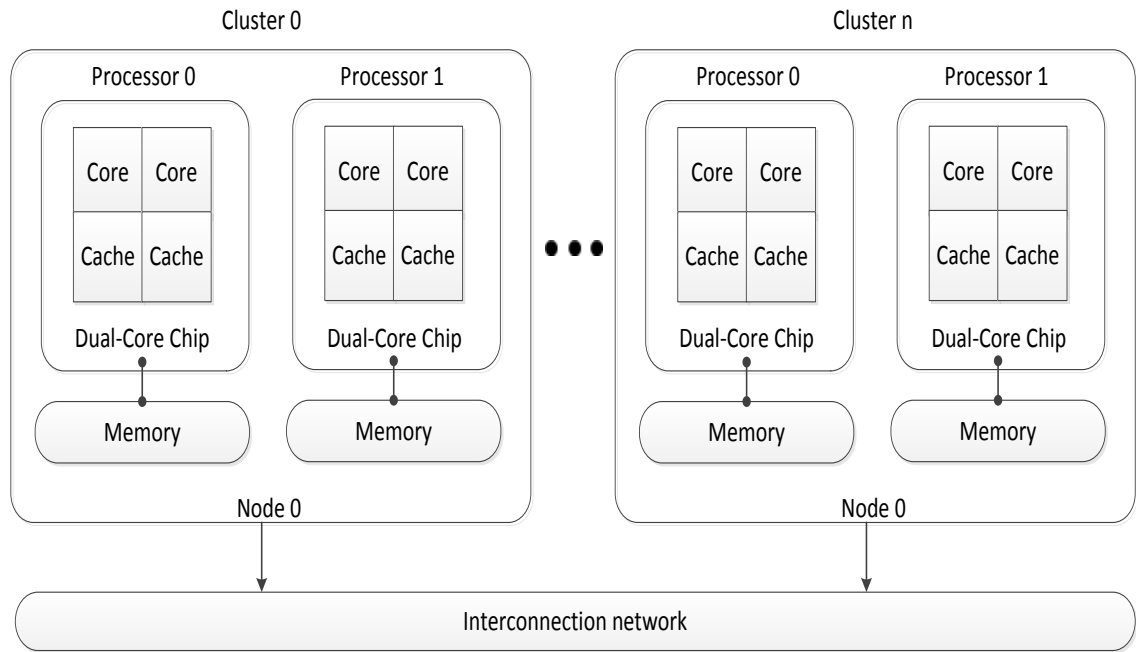


Figure 2-5: Illustration of Multi-core Clusters basic structure

Multi-core clusters typically have a hierarchical memory structure, where cores from the same processor share caches (Fengguang et al., 2009). On the other hand, cores belonging to distinct processors built from the same node share the main memory and cores belonging to different nodes do not share any memory. High performance can be achieved when executing parallel applications with tasks being allocated to the cores according to the application communication pattern and environment characteristics (Silva, Drummond & Boeres, 2010). Tasks that communicate more frequently should be allocated to the same node so as to avoid remote communication. However, depending on the amount of task computation and data to be processed, the allocation of multiple tasks to the same processor can constitute a bottleneck due to the resources being shared by the processor cores (Soryani et al., 2013).

In a multi-core cluster, there are three levels of communication in a multi-core processor (Lei et al., 2007), as shown in Figure 2-6. The communication between two cores on the same processors is referred to as 'intra-chip communication'. The communication across chip but within a node is referred to as 'inter-chip' and the communication between two processors on different nodes is referred to as 'inter-node' (Chan et al., 2012).

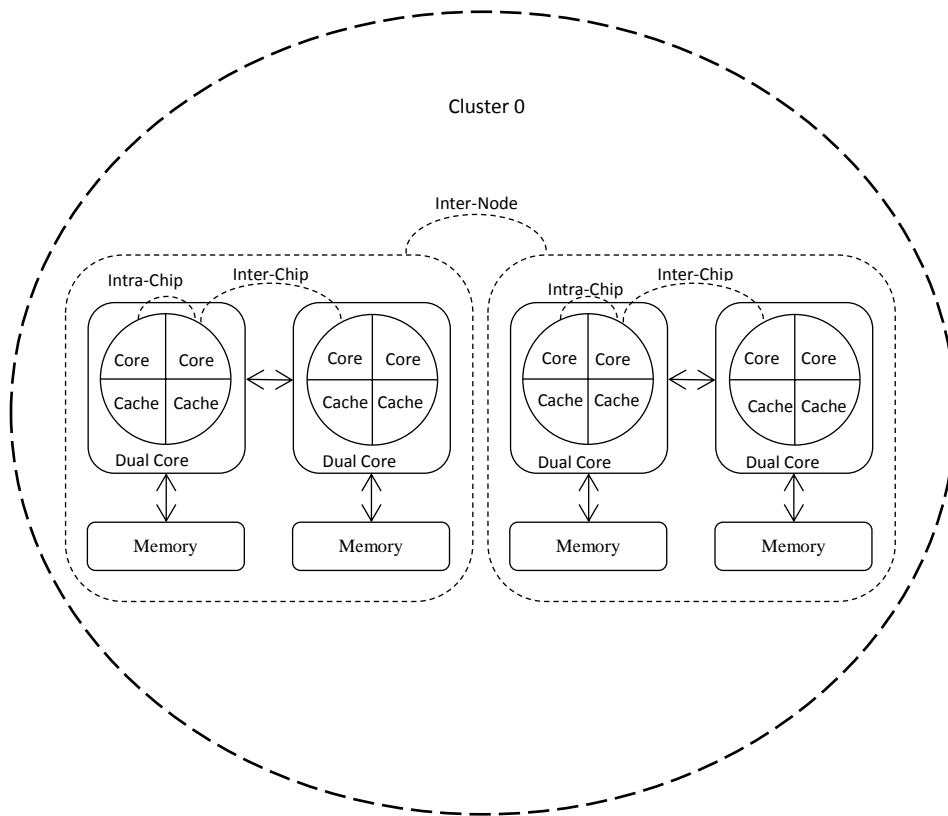


Figure 2-6: Communication level in Multi-core cluster

Message passing in multi-core clusters is more complicated due to the use of different interconnection networks for communication, depending on which cores are involved. This is because of the overhead cost in moving data between cores which involve multiple interconnection networks (Pourreza & Graham, 2007). Data movement between two cores in the same processor is faster than between those in different processors in the same nodes, which is significantly faster than moving data between cores in different cluster nodes (Hamid, Walters & Wills, 2015a). It demonstrates that a multi-core cluster with a good interconnection network will lead to better network performance than would a traditional cluster.

2.6.1 The Advantages of a Multi-core Cluster

A multi-core cluster has a lot of advantages, and nowadays more software is being designed to run with multiple threads. Burger (2005) also states multi-core technology allows systems to run tasks in parallel that previously would have required multiple processors, and multi-core clusters are more easily scalable and can put more processing power in a smaller package that uses less power and generates less heat for the computational power derived. As it gains in popularity, the multi-core cluster will provide greater advantages in speed, scalability and flexibility (Creel & Goffe, 2007; Geer, 2005).

A multi-core cluster can be used to run two programs side by side and when an intensive program is running, such as an audio visual scan, video conversion or CD ripping, another core can be utilised to run the browser to check e-mail, scanning for viruses or using another application (Burger, 2005). A multi-core cluster shows its capabilities when using a program that can utilise more than one core, called parallelisation, to improve the program's efficiency (Karmakar, 2011). Programs such as graphic software and games can run multiple instructions at the same time and deliver faster, smoother results (Creel & Goffe, 2007).

2.6.2 Research Challenges

a) Interconnection Network Performance

Earlier research around multi-core clusters has uncovered a wide variety of issues regarding the network architecture and the limitation of its performance. That focused on multi-core clusters has raised many issues about how to reduce execution time by adding more processors to minimise the communication between nodes. However, while such issues are important, the view of this researcher is that each study misses key issues raised by other studies. Although minimising the communication between nodes may reduce the execution time, it does not guarantee optimal execution time (Ichikawa & Takagi, 2009). B. Javadi, Akbari, Abawajy & Nahavandi (2006), for instance, state that it always depends on the effectiveness of its interconnection network to determine the overall performance of a cluster system.

Performance models in cluster architecture based on single-core clusters have been widely reported (Alzeidi, Khonsari, Ould-Khaoua & Mackenzie, 2007; Furhad, Haque, Kim & Kim, 2013; Geyong et al., 2009; Bahman Javadi, Abawajy & Akbari, 2008a; Khosravi, Khorsandi & Akbari, 2011; Sarbazi-Azad, Ould-Khaoua & Zomaya, 2005). Although various issues have been resolved, the evaluations cannot capture the communication capacities in individual processors. In order to take advantage of multi-core processors in a cluster system, it is important to have an in-depth understanding of the characteristics of multi-core clusters and their impact on application performance and behaviour.

Over time, architectures based on multi-core clusters have been proposed to predict and evaluate network communication performance (Abad, Puente & Gregorio, 2012; Jingjing, Ponomarev & Abu-Ghazaleh, 2012; Khanyile, Tapamo & Dube, 2012; Lei et al., 2007; Mei, Zheng, Gioachin & Kal, 2010; Shainer et al., 2013; Soryani et al., 2013). Previous work on modelling either concentrated on inter-node communication networks or focused on high performance multi-core architecture design without considering the effect of interconnection networks on the performance. Although various issues have been resolved, only a few distinguish the key issue of the

performance of interconnection networks. The existing models are therefore unable to evaluate the potential communication performance of the interconnection networks within the implementation of multi-core cluster architecture.

b) The Scalability of The Network

Scalability in this work refers to the ability of a cluster architecture and its interconnection network to handle an increasing amount of work and the ability to use additional resources with a predictable increase in performance (Sadashiv & Kumar, 2011). The ability of a cluster is not only to function well in the rescaled situation, but to take full advantage of it. Cluster architecture can scale to very large systems, with hundreds or even thousands of machines being networked to suit the application needs. In fact, the entire Internet can be viewed as one truly huge cluster (Leangsuksun et al., 2005). In this research, the scalability of the new architecture will be examined from small (8-cluster) to large (128-cluster), with up to 4-core processors in each cluster. It will also take advantage of the various message lengths to predict the potential performance in term of latency and throughput.

It is always important to examine scalability when evaluating clusters. Abdelgadir, Pathan and Ahmed (2011) found that having a good network bandwidth and a faster network will produce better performance in relation to the scalability of the clusters. The conventional approach to improving cluster throughput is to add more processors, but there is a limit to the scalability of this approach; the infrastructure cannot provide effective memory access to unlimited numbers of processors, and the interconnection networks become saturated (Shahhoseini et al., 2000). The interconnection networks in a cluster system need to be plan with the capacity to meet growing demand of services so that it can handle the expected workloads (Haddad, 2006). Therefore, it is important to address network scalability issue in a new architecture to maintain high network performance.

The new architecture will be examined for scalability potential from 1-core to 4-core processor, cluster size from 8 to 128 number of clusters and 128 bytes to 16 KB of message length. The experiments have been conducted and the results have been produced in Chapter 4 and Chapter 5.

2.7 Multi-cluster Architectures

A 'multi-cluster architecture' is a system of clusters connected with the cluster interconnection networks, where each cluster system/node has multiple processors. Multi-clusters were introduced to address the main concern of a basic cluster system of limited service capacity of its common resources, something that causes an increase in

the waiting time of the processor as the number of processors increases (Rechistov, Ivanov, Shishpor & Pentkovski, 2012). Using more powerful common resources is the conventional method for decreasing waiting time, but the capacity for servicing resources – such as effective memory access time and the interconnection network bandwidth – is saturated by the technology and the structures (Shahhoseini et al., 2000). To overcome the problems, advances in computational and communication technologies has made it economically possible to combine multiple clusters to develop a large-scale system known as a ‘multi-cluster system’ (Abawajy & Dandamudi, 2003).

2.8 Message, Packet and Flits

A message may be defined as a logically contiguous group of bits that are delivered from a source node to a destination node (Dally & Towles, 2004). Because messages may be arbitrarily long, resources are not directly allocated to messages, and messages may be divided into one or more packets. A packet is the basic unit of routing and sequencing, and a ‘control state’ is allocated to a packet. ‘Flits’, or flow control digits, are the basic units of bandwidth and storage allocation used by most flow control mechanisms (Bahman Javadi et al., 2008a). Figure 2-7 illustrates the partitioning of a message, packet and flit.

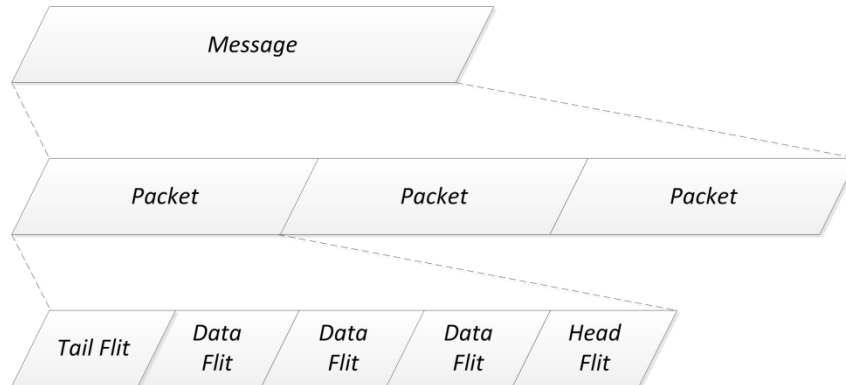


Figure 2-7: An illustration of the subdivision of a message into packets and of packets into flits (reproduced from (Dally & Towles, 2004))

2.9 Modelling and Simulation

Modelling is defined as the process of identifying and abstracting relevant entities and relationships from a system under study (Wehrle et al., 2010). It is a process of producing a close approximation to the real system which represents the construction and working method of a system of interest (Maria, 1997). A model is thus a representation of an actual system (J. Banks, 1998) or process (Carson, 2005).

Simulation is defined as an imitation of a system as it progresses through time (Robinson, 2004), and contains a set of entities and relationships to fulfil a certain purpose (Wehrle et al., 2010). Simulation modelling is the process of creating and analysing a digital prototype of a physical model to predict its performance in the real world. A 'system' is a construct or collection of different elements – such as people, machines and resources – that together produce results not obtainable by the elements alone (C. Banks, 2008).

Simulation is needed to predict the performance of systems that are subject to variability, for example in its interconnections and its complexity. With simulation it is possible to predict system performance, to compare alternative system designs and to determine the effect of alternative policies on system performance.

2.9.1 Modelling and Simulation Techniques

Whether the model and the simulation implementing it accurately represent the real system can be checked with two techniques, model verification and model validation. Model verification is defined as “ensuring that the computer program of the computerized model and its implementation are correct” (Sargent, 2013). Model validation is “substantiating that a computerized model, within its domain of applicability, behaves with satisfactory accuracy, consistent with the intended application of the model” (Schlesinger, 1979). Model validation deals with building the right model.

Development of modelling and simulation involves human actions and knowledge-intensive activities, in which errors, uncertainties, and inadequacies are inevitable, leading to quality deficiencies in models and simulation results (Sargent, 2011). Verification and validation focus on assessing the accuracy of an application with respect to its objectives (Shannon, 1977), and is intended to ensure that only correct and suitable models and simulation results are used in practice. Credibility of simulation results not only depends on model correctness, but is also significantly influenced by accurate formulation of the problem (Balci, 1994). Therefore, validation, verification and testing techniques must be employed throughout the life cycle of the study, which is the methodology used in this work.

This thesis will discuss an MCMCA simulation model verification by predicting new behaviours for a multi-core cluster architecture. An analytical model using mathematical calculations is used to validate the simulation results and the testing will be demonstrated in Chapters 4 and 5. Numerous sets of experimental conditions are designed to test the applicability of the simulation model. The outputs from various sets of experimental conditions are graphed to determine whether the simulation

model behaviour is comparable with the analytical model. This is intended to answer all the Research Questions and Sub-research Questions listed in section 1.2.

2.9.2 Simulation Model Structure

The main structure in the simulation model is its interconnection network. An interconnection network is a connection between two or more computer networks via network devices such as routers and switches, so as to exchange traffic back and forth and guide traffic across the complete network to its destination (Tanenbaum, 1996). The main task of the interconnection network is to transfer messages from a specific processor to a specific destination, which can be another processor or a memory (Gramsamer, 2003). Routers will determine the route for a packet based on a routing algorithm, and transmit it from the source to its destination on a node of another network. While routers connect the networks, switches create a network by filtering and forward packets between networks.

To achieve the performance specifications in multi-core cluster system, a comprehensive design of network architecture within technology constraints is essential by implementing the network topology, routing algorithm and flow control mechanism. These three components will be a basis in developing a simulation model.

a) Network Topology

Topology describes the interconnection structure used to connect different processors or processors and memory modules. It can be characterised as static or dynamic interconnection (Kumar, Grama, Gupta & Karypis, 1994). Static interconnection networks are also called 'direct' networks or 'point-to-point' networks, while dynamic interconnection networks are called 'indirect' networks. Direct networks connect nodes directly with each other by fixed physical links, while indirect networks connect nodes indirectly via switches and links. Some include a 'bus' network, a 'crossbar' network, a 'butterfly' network or 'fat-tree' network (Kumar et al., 1994; Tanenbaum, 1996).

The common topologies that are used for interconnection networks are shared bus and crossbar switches (Prisacari, Rodriguez, Minkenberg & Hoefler, 2013). However, shared bus and crossbar switches do not scale well with the growth of the processor numbers (Khosravi et al., 2011). Many network topologies have thus been proposed for clusters, of which fat-tree topologies are among the most popular (Furhad et al., 2013). 'Fat-tree' derives from a popular class multistage interconnection network, 'Butterfly-Fat-tree' (Xuan-Yi, Yeh-Ching & Tai-Yi, 2004). In Fat-tree, as in Figure 2-8, each node in the tree is represented by a set of coordinates (level, position), where 'level' denotes the level in the tree and 'position' denotes the location, using right to left ordering. The vertical levels are numbered from zero, starting at the leaves. The

'leaves' in the trees correspond to nodes, and the upper levels represent routers. Fat-tree topology can be scalable with more processing nodes, and the regularity of the processing node connection can also be exploited to develop more efficient parallel algorithms (Lin, 2003).

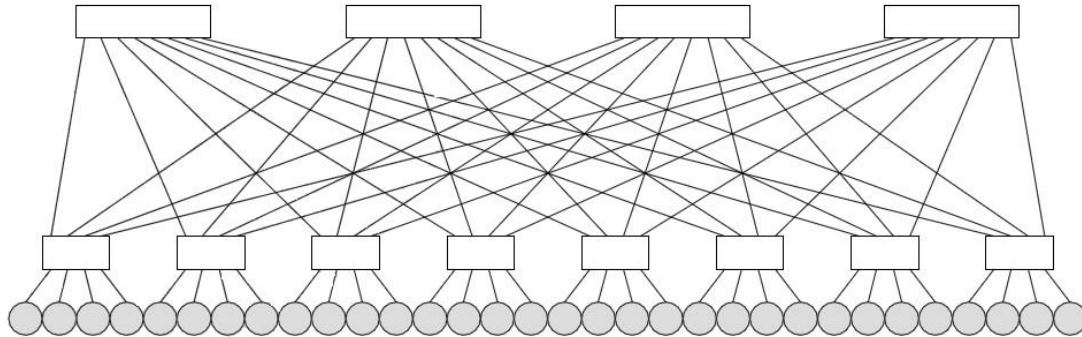


Figure 2-8: An 8-port 2-tree constructed by proposed algorithm

There can be many possible paths that a message could take through the network topology to reach its destination, and this will be determined by the routing algorithm, discussed in the next section.

b) Routing Algorithm

The routing algorithm determines the path to be used for data transmission, and a good routing algorithm balances the load uniformly regardless of the offered traffic pattern (Rauber & Runger, 2010). There are three main routing techniques: deterministic, oblivious and adaptive. Deterministic routing always sends the packet in the shortest direction; oblivious routing randomly picks a direction for each packet, while the adaptive routing technique sends the packet in the direction for which the local channel has the lowest load. While adaptive routing changes the path of packets dynamically, deterministic routing determines a path statically.

Deterministic routing has the following advantages (C. Gomez, Gilabert, Gomez, Lopez & Duato, 2007; Koibuchi, Watanabe, Kono, Akiya & Amano, 2003):

- (1) It guarantees the FIFO packet delivery, which is required with several message passing libraries.
- (2) It makes the detection and tracing of misrouted packets much easier than adaptive routing, since there is a pre-determined path between each pair of hosts.

Some routing algorithms for cluster networks have been proposed (Koibuchi, Jouraku & Amano, 2002; Koibuchi, Watanabe, et al., 2003; Sancho, Robles & Duato,

2004) but, in general, these routing algorithms are designed for irregular topologies. For regular topologies like fat-trees, most cluster networks adopt a deterministic routing algorithm (Bahman Javadi, Abawajy & Akbari, 2006).

The deterministic routing algorithm performs the balance traffic distribution and it will extinguish the switch contention problem (Bahman Javadi et al., 2008b). In deterministic routing, a message traverses a fixed path between source and destination, which simplifies the implementation, avoids a message deadlock and guarantees an in-order delivery (Yulei et al., 2012). Gomez et al., (2011) show that deterministic routing can achieve a similar, and in some scenarios an even higher, level of performance than adaptive routing in the fat-tree network topology.

Routing is carefully designed to avoid deadlocks. A deadlock occurs in an interconnection network when a group of packets is unable to make progress because they are waiting on one another to release a resource; usually this is a buffer or channel (Dally & Towles, 2004). The simplest deadlock-free deterministic routing used in cluster networks is Up*/Down* routing (Sancho et al., 2004; M. D. Schroeder et al., 1991).

'Up*/Down* routing' is based on an assignment of direction (up or down) to network channels where a spanning tree whose node (also called 'vertex') corresponds to a switch in the network, based on building a 'breadth-first search' (BFS) spanning tree used in Autonet (M. D. Schroeder et al., 1991). Based on this spanning tree, the 'up' end of each link is defined as:

- (1) The end whose switch is closer to the root in the spanning tree,
- (2) The end whose switch has the lower identifier, if both ends are at switches at the same tree level.

A deadlock occurs when flow control holding resources are waiting on another set of resources to complete a work cycle. To avoid deadlocks while still allowing all links to be used, this routing scheme uses the following Up*/Down* rule: a legal route must traverse zero or more links in the up direction followed by zero or more links in the down direction (Sancho et al., 2004). Thus the Up*/Down* rule prohibits any packet transfer from the down direction to the up direction. The Up*/Down* rule can never cause a deadlock because of the ordering imposed by the spanning tree, and no deadlock-producing loops are possible (M. D. Schroeder et al., 1991). It also guarantees deadlock-free routing since no cycles are formed among paths with the above rule while still allowing all hosts to be reached (Koibuchi, Akiya, Watanabe & Amano, 2003). Although Up*/Down* routing was originally an adaptive routing, it can be implemented as a deterministic routing by choosing a single path from several alternative paths (Koibuchi et al., 2002).

c) *Flow-Control Mechanisms*

Flow control determines how a network's resources – such as channel bandwidth, buffer capacity and control state – are allocated to messages as they progress along their route in the network (Gramsamer, 2003). Channel bandwidth transports messages between nodes and buffers – such as registers and memories, which allow messages to be held temporarily at the nodes – are storage-implemented within the nodes. The 'control state' tracks the resources allocated to the packet within the node and the state of the packet's traversal across the node (Dally & Towles, 2004).

The fact that multiple messages can be in transmission and attempt to use the same network link at the same time will cause a problem in the network. If this problem occurs, some of the message transmission must be blocked while other messages are allowed to proceed (Kumar et al., 1994).

A good flow control forwards packets with minimum delay and avoids deadlocks. Flow control can be divided into two methods: bufferless and buffered (Raubert & Runger, 2010). 'Bufferless' flow control uses no buffering and simply allocates channel and bandwidth to competing packets, while 'buffered' flow control can store a packet in a buffer, preventing the waste of 'channel bandwidth caused by dropping or misrouting packets.

d) *Poisson Distributions*

Poisson distribution expresses the probability of a number of occurrences in a fixed interval, and these happen independently of the time and with a known average speed (Sadeghi & Barati, 2012). The term can also be used for the number of events in other specified intervals such as distance, area, or volume. A Poisson distribution is a discrete distribution and focuses only on the discrete occurrences over some interval. (The Poisson distribution is characterized here by λ , the inter-arrival time in the interconnection network).

e) *Random Number Generators (RNGs)*

A random number generator (RNG) is a program written for and used in probability and statistics applications when a large volume of random digits are needed (Dally & Towles, 2004). RNGs are widely used in number of applications, particularly in simulation. These programs produce endless strings of single-digit numbers, usually in base 10, known as the decimal system. When large samples of random numbers are taken, each 10 digits in the set [0,1,2,3,4,5,6,7,8,9] occurs with equal frequency, even though they are not evenly distributed in the sequence. 'Mersenne Twister' is the random number generator employed by OMNeT++, used to distribute the message destinations in the simulation model in this thesis. The details of the tests of the

random number generator to establish its fitness for purpose is provided in Appendix 2-A.

2.10 Summary

This chapter described the motivation, as reflected in the literature, for using clusters as well as the technologies available for building a new cluster architecture. Much emphasis is placed on using commodity-based hardware and software components to achieve high performance and scalability, and at the same time to keep the ratio of price versus performance low. Cluster computing has emerged as a result of the convergence of several trends, including the availability of inexpensive high-performance microprocessors and high-speed networks, the development of standard software tools for high performance parallel and distributed computing, and the increasing need of computing power for computational science and commercial applications. Clusters have evolved through web servers and e-commerce, to support applications ranging from supercomputing and mission-critical software, to high-performance database applications.

It is clear that high-speed networks for cluster computing are important in order to support the needs of better performance. The rapid changes in interconnection network technology have provided a new opportunity to improve the performance of cluster computing. Although much progress has been made in the development of low-latency protocols and new standard architecture, it creates interesting new challenges. The capability of clusters to deliver high performance and availability on a single platform is empowering many existing and emerging applications and making clusters the platform of choice.

This chapter also presented the background to a simulation model structure which includes network topology, flow control mechanism and routing algorithms. These are important components of message-passing and communication in interconnection networks. Once a topology is ready, routing will pick a route that gets the messages to their destinations. Routing algorithms establish the path between the source and the destination of a message. Flow control determines how a network's resources are allocated to messages as they progress along their route in the network.

Several research issues are discussed in the context of multi-core clusters and interconnection networks which contribute to a new architecture. The various issues presented in this chapter show that disadvantages of a single-core cluster moved the present researchers to focus on multi-core clusters. This is the reason why this thesis will focus on designing a new architecture of interconnection network in a multi-core cluster.

This thesis will apply store-and-forward flow control mechanisms and wormhole flow control mechanisms to evaluate the performance of interconnection networks in a multi-core cluster architecture. The research will focused on buffered flow control, with store-and-forward flow control representing packet-buffer flow control and wormhole flow control, indicating flit-buffer flow control, in order to determine the allocation of the network's resources. Since a packet is a part of a message, in this thesis 'message' and 'packet' will be used interchangeably.

Multi-core clusters resolve the various concerns by dividing the workload between different cores to speed up performance. Multi-core clusters also provide a scalable performance computing solution, and use the aggregated power of computing nodes to form a high-performance solution for parallel applications. This thesis will tackle the challenge to incorporate multi-cluster and multi-core cluster into a novel architecture known as Multi-core Multi-cluster Architecture (MCMCA). The incorporate entities in MCMCA are shown in Figure 2-9.

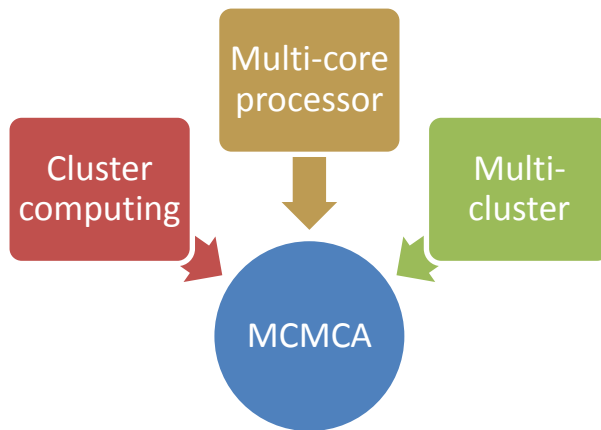


Figure 2-9: The incorporate entities in MCMCA

The next chapter presents the new architecture proposed to tackle the performance issue discussed in the literature.

Chapter 3 The New Architecture

3.1 Introduction

Chapter 2 introduced the concept and characteristics of cluster computing and multi-core processors to address cluster performance by networking. The need for a new architecture was highlighted.

This chapter presents the Multi-core Multi-cluster Architecture (MCMCA) as a new structure to improve cluster performance with low latency. This is the first published investigation of interconnection network performance of MCMCA by simulation. This chapter aims to answer the following research question:

RQ1: What is an appropriate architecture to investigate the communication latency of multi-core processors in multi-cluster?

Hence, the following sub-research questions will be addressed:

SQ1: What are the appropriate characteristics to be considered in designing a cluster architecture?

SQ2: What is an appropriate simulation model to investigate interconnection network performance?

SQ3: How well does the MCMCA simulation model analyse cluster performance?

Section 3.2 presents the new architecture, with the design of a modified queueing network. Section 3.3 provides architectural detail to be considered in cluster design, followed in section 3.4 by a queueing network model and, in section 3.5, a flow diagram, all focusing on a new structure of interconnection network to achieve low latency and high bandwidth. Finally, the methodologies which will be used for the performance analysis are discussed in section 3.6, while section 3.7 discusses the feasibility of an MCMCA simulation model by comparing the baseline results with previous research.

3.2 Multi-core Multi-cluster Architecture (MCMCA)

This new architecture is introduced in Figure 3-1. The structure of MCMCA is derived from a Multi-Stage Clustering System (MSCS) (Shahhoseini et al., 2000) which is based on a basic cluster using single-core nodes. The MCMCA for its part is built up of numbers of clusters where each cluster is composed of numbers of nodes, the numbers of which are determined at run time. Each node of a multi-core cluster has more than one processor. Cores on the same chip share local memory, but have their own cache. The interconnection network connects the cluster nodes.

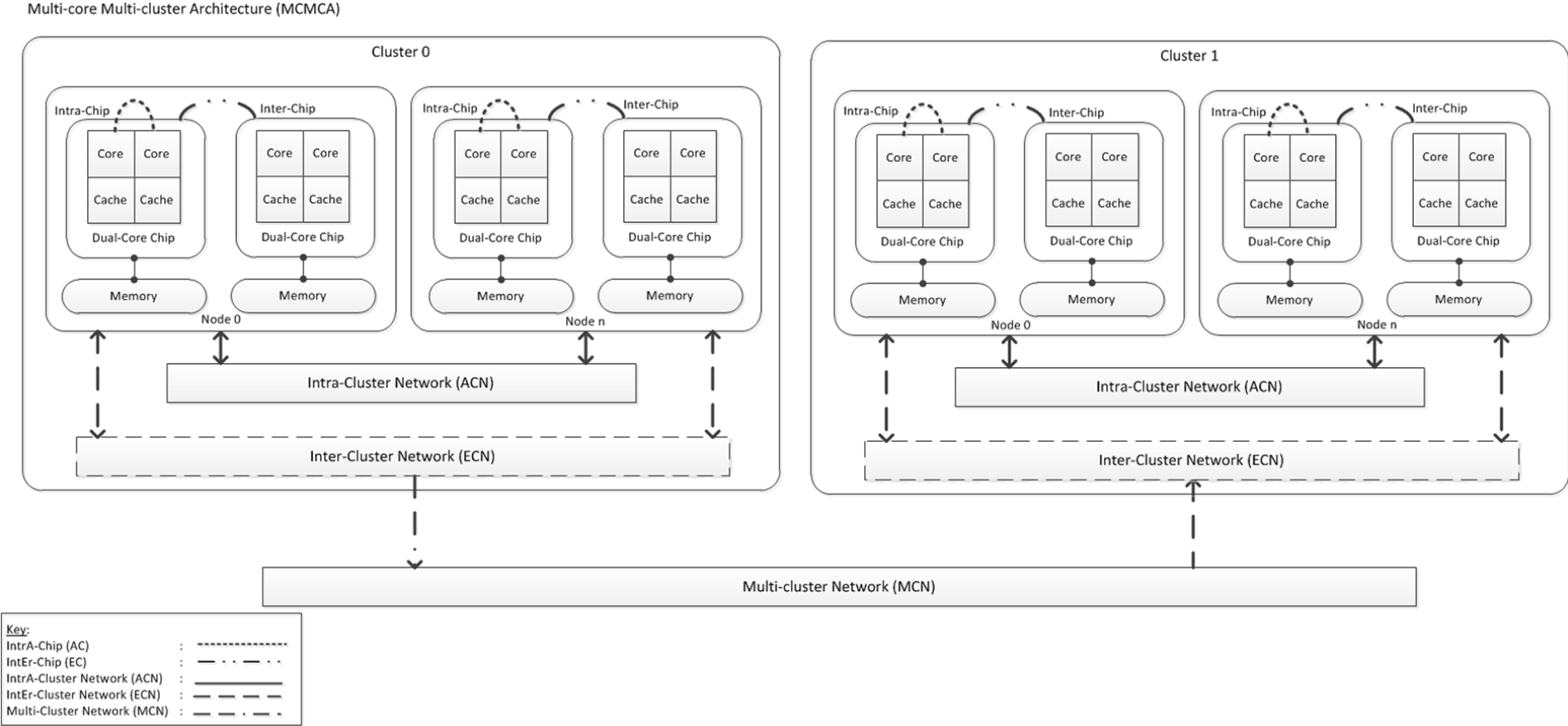


Figure 3-1: Overview of the proposed Multi-Core Multi-Cluster Architecture (MCMCA)

3.3 Appropriate characteristics for cluster architecture design

To answer research question RQ1 and sub-research question SQ1, a systematic review was conducted of existing literature. The characteristics for designing cluster architecture were investigated from 1999 onwards in the area of high performance computing.

With the emergence of multi-core clusters, each core will run at least one process with multiple interconnection networks to several other processes. This will put immense pressure on the interconnection network. Nonetheless, interconnection networks are critical in achieving high performance in clusters (Shainer et al., 2013). While ideal networks support both high bandwidth and low latency, there often exists a trade-off between these two parameters. For example, a network that supports high bandwidth tends to keep the network resources busy, often causing contention for the resources which will increase the latency of the messages. Contention occurs when two or more messages want to use the same shared resource in the network. When a packet has to travel from one interconnection network to another to get to its destination, many problems such as contention and blocking can arise, and this may contribute to communication latency of the interconnection network.

The performance of a cluster system depends on the communication latency of its interconnection network. The research conjecture is that, by employing a multi-core processor in a multi-cluster architecture, it is possible to achieve a lower latency and a faster transmission than is possible with a single-core processor. There are five interconnection networks in MCMCA.

- Two of them are commonly found in any multi-core cluster architecture; the intra-chip network (AC) and the inter-chip network (EC).
- The three new interconnection networks introduced in this research are the intra-cluster network (ACN), the inter-cluster network (ECN), and the multi-cluster network (MCN).

3.3.1 Intra-Chip network (AC)

The communication between two processor cores on the same chip is the intra-chip network (AC), as shown in

Figure 3-2. Messages are divided into a number of cores by the AC, as shown by arrow 1 and arrow 2, which act as connectors between two or more processor cores on the

same chip. In theory, dividing the messages into a number of cores results in more than twice the performance with lower communication delay (Furhad et al., 2013).

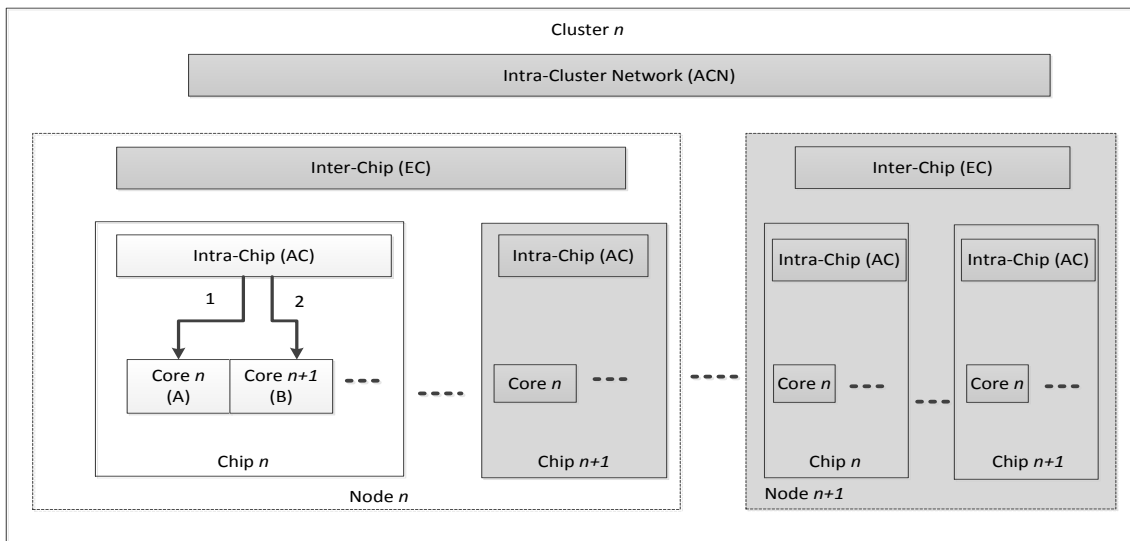


Figure 3-2: Communication for message passing between two processor cores on the same chip

3.3.2 Inter-Chip network (EC)

Figure 3-3 shows an inter-chip network (EC) for communicating across processors in different chips but still within the same node. Messages travelling to different chips in the same node communicate from source A by arrow 1 via the intra-chip (AC) and arrow 2 via inter-chip (EC) to reach their destination B by arrow 3 via EC and by arrow 4 via AC.

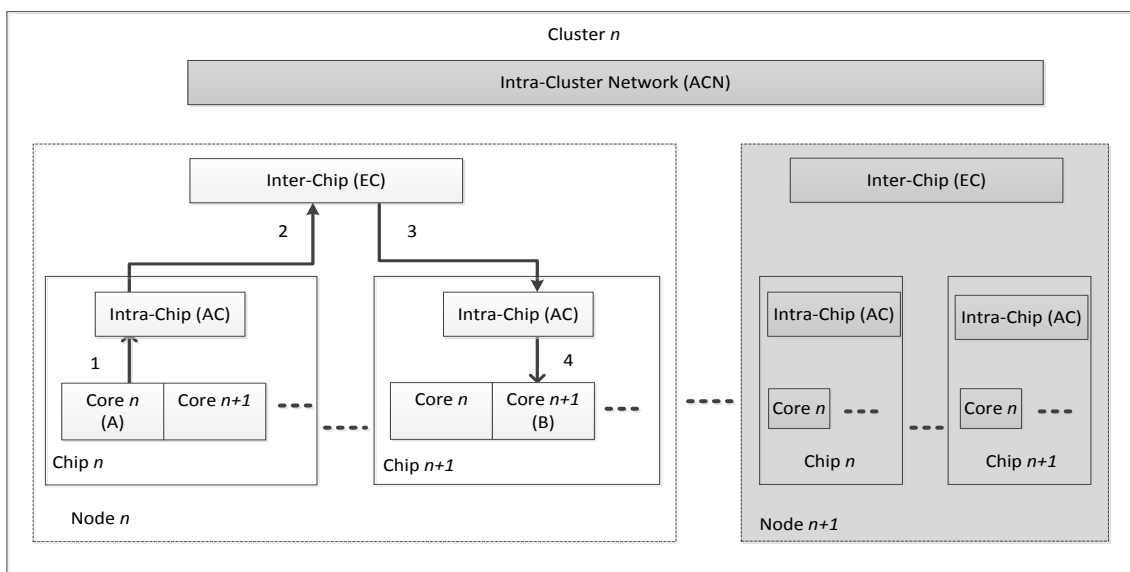


Figure 3-3: Communication for message passing across processors in different chips, but within a node

3.3.3 Intra-Cluster Network (ACN)

Intra-cluster network (ACN) is a network to connect nodes within a cluster. As depicted in Figure 3-4, messages that cross the nodes (arrow 3) to other nodes (arrow 4) in the same cluster are connected by ACN via intra-chip (AC) by arrow 1 and the inter-chip (EC) by arrow 2 to complete its journey by arrow 5 and 6.

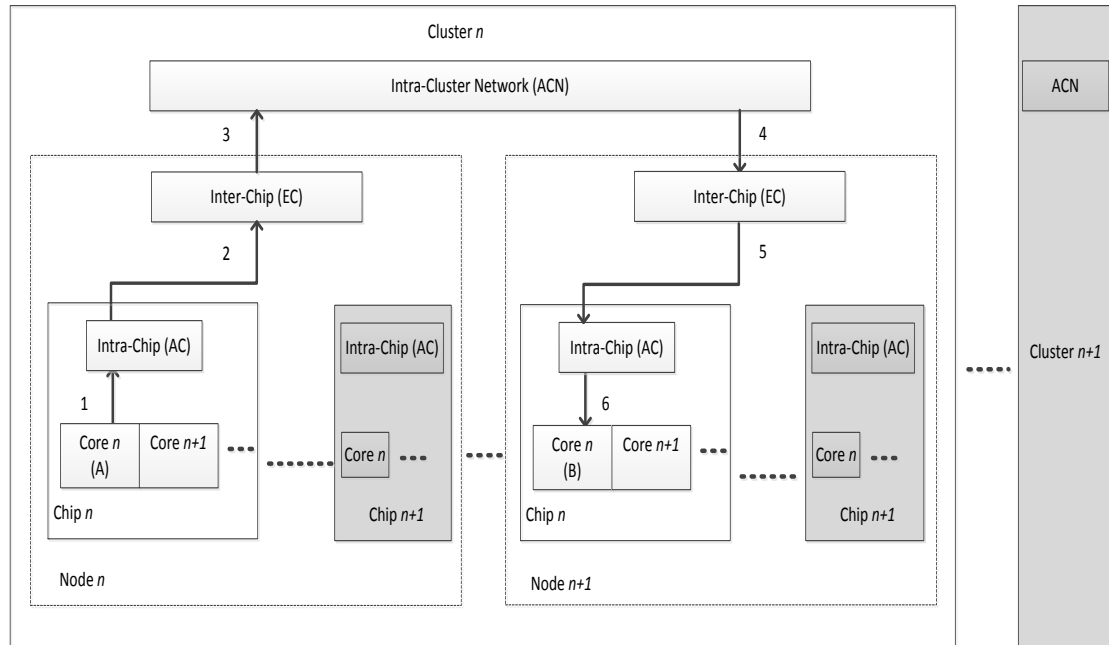


Figure 3-4: Communication routes for messages passing between processors on different nodes, but within the same cluster

3.3.4 Inter-Cluster Network (ECN) and Multi-Cluster Network (MCN)

The longest route for messages to travel involves ECN and MCN. As shown in Figure 3-5, messages travelling from their source to their destination between clusters communicate via two interconnection networks, ECN and MCN, to reach other clusters.

An inter-cluster network (ECN) is used to transmit messages between clusters. The clusters are connected to each other via the multi-cluster network (MCN). When the messages reach another cluster (arrow 5), they are connected by the ECN of the cluster before arriving at their destination. The same process will continue with the other clusters until all the packets exit the network.

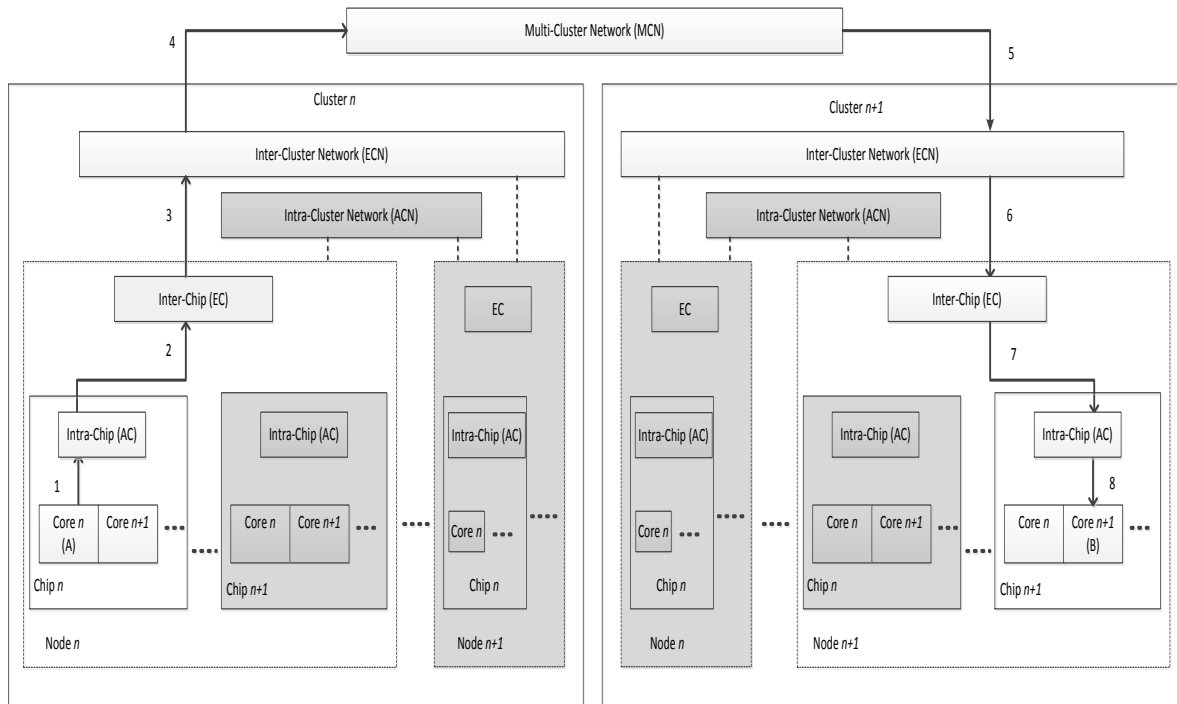


Figure 3-5: Communication routes for transmitting messages between clusters

3.4 The MCMCA Queueing Network Model

Message-passing in MCMCA is embedded within the queueing network model approach, as shown in Figure 3-8. In interconnection networks, packets spend a lot of time waiting in queues before they are transmitted by a processor core to their destination. A source will generate packets at a rate of $\frac{1}{\lambda}$ packets per second and they will be queued while waiting to be transmitted into the network. An interconnection network then removes the packets from the queue on a first-in-first-out (FIFO) basis and processes them with an average transmission time (Hamid, Walters & Wills, 2015b).

Approximations of packet latency are based on a queueing network model so as to predict the average amount of time that a packet spends waiting in each queue in the architecture (Deng & Purvis, 2011). A queueing network consists of service centres (i.e. interconnection networks) and customers (i.e. packets). A service centre has one or more queues to hold jobs waiting for service. After being serviced, a job either moves to another service centre or exits the network (Kaplan & Nelson, 1994). Illustration of the equation in internal-cluster and external-cluster with graphs of their individual behaviour is shown in Appendix 4-A.

This work will consider the distribution of the transmission time upon reaching a high traffic density due to a packet's arrival in an M/G/1 queueing network. Queueing networks with an M/G/1 Markovian queue model are used to analyse systems with a Poisson distribution transmission time (Sadeghi & Barati, 2012). The M/G/1 queueing network studies have been widely reported, which makes tractable the solution of modelling interconnection networks of MCMCA by simulation (Bahman Javadi et al., 2008b; Sarbazi-Azad et al., 2001; Sharifi, Akbari & Javadi, 2009). In general, an M/G/1 queueing network with arbitrary transmission time distribution has occupancy of –

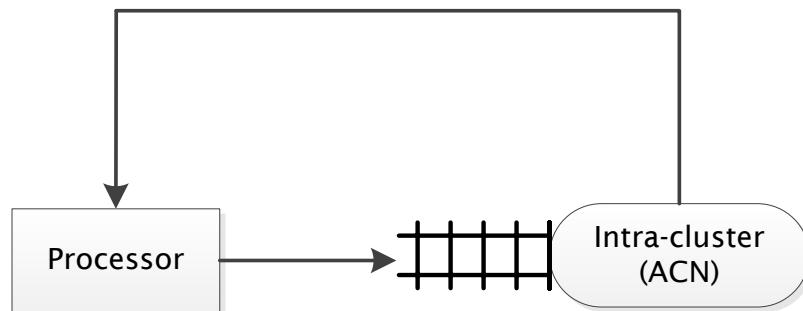
$$W = \frac{(\beta)^2 \lambda_i}{2(1 - \beta \lambda_i)}$$

Where

λ_i = arrival rate

β = average transmission time

As discussed in section 2.5 above, a traditional cluster contains single processor nodes with one interconnection network, as shown in Figure 3-6. The messages passing between processors in single clusters are going through an intra-cluster network (ACN) which involves queues for messages to enter the network. Queueing networks for multi-core clusters are shown in Figure 3-7. Multi-core clusters were also included in single cluster architecture, but with multiple cores in a processor. With multiple cores in a chip, the combination may be able to provide greater throughput by reducing the queues in each processor (Geer, 2007). Basically, this will decrease the latency and improve the interconnection network performance. Compared to traditional clusters, multi-core clusters involved three interconnection networks. Chip communication consists of inter-chip networks (AC) and inter-chip networks (EC), while communication between processors in the single cluster is via intra-cluster networks (ACN).



Key:

ACN – Message passing between processors in the same cluster

Figure 3-6: Queuing network of single-core cluster

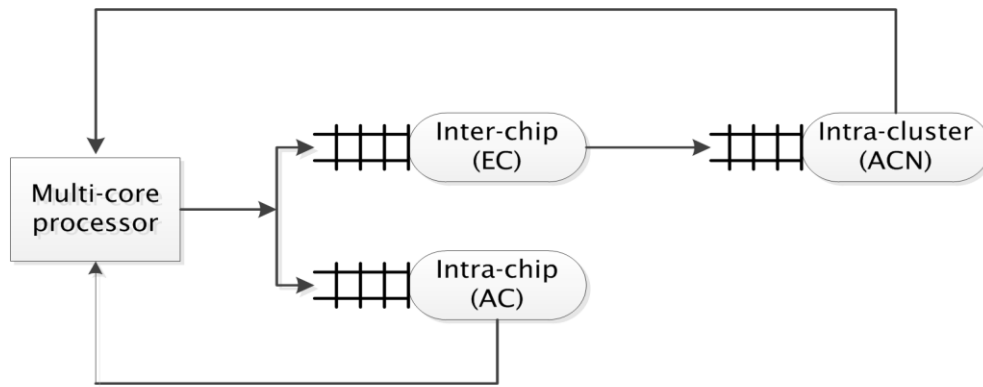


Figure 3-7: Queuing network of multi-core cluster

Referring to the new architecture proposed in section 3.2, queuing networks for MCMCA is shown in Figure 3-8. As discussed in section 3.3, MCMCA involves five interconnection networks in a multi-cluster architecture. Each processor generates packets independently, following a uniformly-distributed Poisson process. Intra-cluster networks (ACN) represent the process in the same cluster and inter-cluster networks (ECN) represent the communication between clusters through a multi-cluster network (MCN). The output of ECN in another cluster is a feedback to the same processor. This queuing network demonstrated that a long waiting queue can be reduced with multi-core processors, which will save transmission time. By incorporating multi-clusters, MCMCA has boundless service capacity of its common resources, which will contribute to reducing the waiting time of the processor.

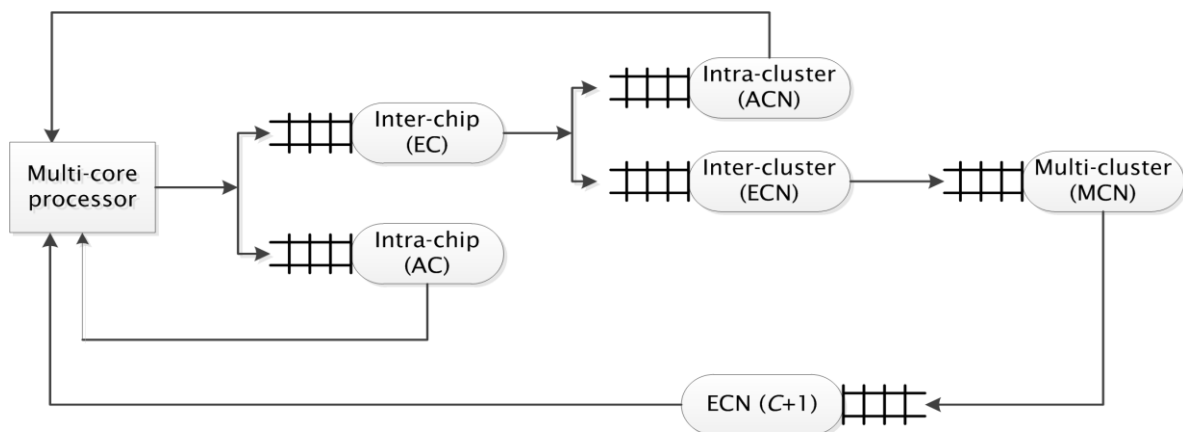


Figure 3-8: Queuing network of Multi-core Multi-cluster Architecture (MCMCA)

3.5 MCMCA Activity Diagrams

A flow diagram detailing the work flow in a cluster node with a single-core is depicted in Figure 3-9, and a flow diagram representing the work flow in a cluster node with a multi-core processor is shown in Figure 3-10. After *source* generates a packet, that packet will access the processor through a node, where a node can contain one or more processors. If the first processor in the first node is idle, that processor will divide the task between a number of cores on the processor. If the first processor is busy, it will first pass the packet to another processor in the same node before distributing it to the other processors in other nodes. The target node will communicate with nodes through the interconnection network.

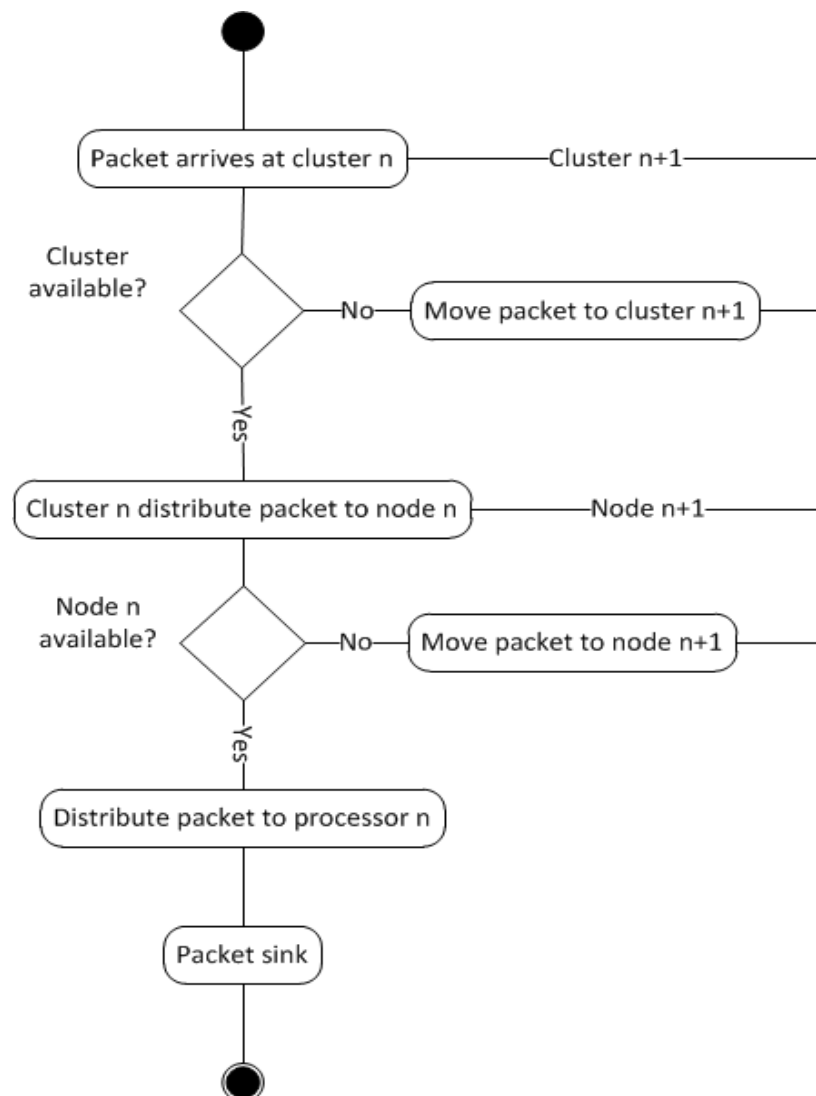


Figure 3-9: Activity diagram of a packet traversing the cluster node of a single-core processor

Since the packet distribution in single-core cluster only contains single cores in a cluster node, it has go through a longer work cycle to complete the network transmission (Ichikawa & Kawai, 2008). This will consume more time, and more latency will be created. The new architecture overcomes these issues by employing multi-core processors, by which the task will be divided between a number of cores in each processor. More network transmissions can thus take place at the same time, which will save time and decrease the latency.

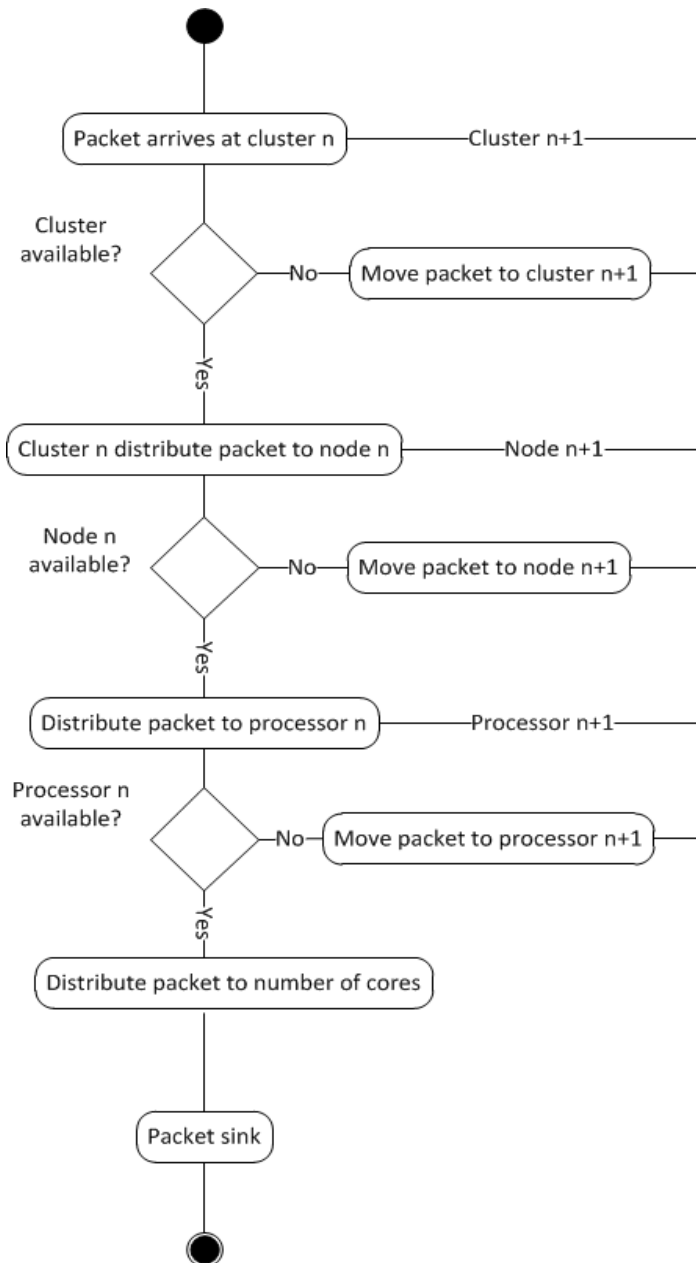


Figure 3-10: Activity diagram of a packet traversing the cluster node of a multi-core processor

3.6 Simulation model to investigate interconnection network performance

Simulation and analysis are the most popular tools to measure the performance of an interconnection network. Simulation provides accurate performance estimation but requires more time to generate results. For more accurate performance estimation, therefore, simulation is mostly performed to validate approximate results derived from analysis. This has been demonstrated by many analytical researchers in validating their model or results (Furhad et al., 2013; Khanyile et al., 2012; Shainer et al., 2013; Soryani et al., 2013).

Analysis provides approximate performance results with a minimum amount of effort and gives insight into how different factors affect performance. By deriving a set of equations, analysis can predict the performance of an entire network with different simultaneous configurations and parameters. However, analysis usually involves making a number of approximations that may affect the accuracy of results (Caliri, 2000).

Simulation is needed to predict the performance of systems that are subject to variability, for example in their interconnections and complexity. With simulation it is possible to predict system performance, to compare alternative system designs and to determine the effect of alternative policies on system performance (C. Banks, 2008). Maria (1997) stated that simulation is best used before an existing system is altered or a new system built, in order to reduce the chances of failure, to meet specifications, to eliminate unforeseen bottlenecks, to prevent under- or over-utilisation of resources, and to optimise system performance. To imitate or to produce a close approximation to the real system, a simulation model is thus the best choice.

In the area of communication networks, simulation is a useful methodology since the behaviour of a network can be modelled by calculating the interaction between the different network components such as routers, channels or packets using mathematical formulas (Brakmo & Peterson, 1996). The simulation is modelled by capturing and playing back experimental observations from real networks (Jia, Xiangzhan & Jun, 2009). The data from simulation experiments and the behaviour of the network can then be observed and analysed in offline test experiments, using mathematical and statistical analyses. All kinds of environmental attributes can be modified in a controlled manner to see how the network behaves under different parameters or different configuration conditions. Network simulation can be used, together with different applications and services, to observe performance in networks (Pan, 2008).

The next sections introduce the network simulation tool used in modelling the Multi-core Multi-cluster Architecture (MCMCA) in order to answer sub-research question SQ2:

‘What is an appropriate simulation model to investigate interconnection network performance?’

3.6.1 OMNeT++ Network Simulation Tool

Network simulation tools are computer-assisted technologies applied in the simulation of networking algorithms or systems (Pan, 2008). Most network simulation tools apply discrete-event simulation (Mehta, Sulatan & Kwak, 2010). Discrete-event simulation manages events in time: the simulator reads the queue and triggers new events as each event is processed. An event is an instantaneous occurrence that changes the model’s state (Carson, 2005); examples include the arrival of a packet for each node in a cluster, and the service completion of a packet for the same node. Simulation models for MCMCA used discrete-event simulation and were developed using the OMNeT++ network simulation tool.

OMNeT++ is a C++-based discrete-event simulator which uses the process-interaction approach and it has been publicly available since 1997 (A. Varga, 2001). It has been created with the simulation of distributed systems, parallel systems, communication networks and multiprocessors. OMNeT++ is an open-source discrete event simulation tool that can be used in the design and analysis of systems in which state changes are discrete (Jingjing, Ponomarev & Abu-Ghazaleh, 2012). It provides an open-source, modular, scalable, extendible and fully parameterizable framework for modeling multi-core multi-cluster architecture (MCMCA).

a) The Comparison of Network Simulation Tools

Network simulation tools have changed a lot in the past ten years (Varga & Hornig, 2008). Comparison studies of network simulators have been conducted (Mehta et al., 2010; Pan, 2008; Varga & Hornig, 2008; Weingartner, vom Lehn & Wehrle, 2009) which involved OMNeT++, MATLAB, ns-2, ns-3, OPNet and QualNet.

Table 3-1 examines the simulation packages most relevant for the analysis of telecommunication networks and compares them to OMNeT++. NS-2 is still the most widely used network simulator in academia, but OMNeT++ provides more infrastructures. OMNeT++ is also popular in academia and industry because of its extensibility, since it is also open-source. There is also plentiful online documentation and mailing lists for general discussion. Although NS-3 demonstrated the best overall performance, it still needs to improve its simulation credibility (Weingartner et al., 2009), and OMNeT++ can be considered as a viable alternative. Although it is a

commercial product, OPNet has similar foundations to OMNeT++ and contains an extensive model library; also it provides several additional programs and GUI tools. The other two commercial products are QualNet, which emphasizes wireless simulations, and MATLAB, which needs several prerequisite components for its files to function normally.

Table 3-1: A comparison of OMNeT++ with other simulation tools

No	Features\Tools	OMNeT++	MATLAB	ns-2	ns-3	OPNet	QualNet
1	Applicability for Network Simulation	Yes	Yes	Yes	Yes	Yes	Yes
2	Interface	C++	C++	C++	C++/ Phyton	C or C++	Parsec (C- Based)
3	Traditional Models eg.TCP/IP, Ethernet), Wireless Support	Yes	No	Yes	Yes	Yes	Yes
4	Parallelism	MPI/PVM	Yes	No	Yes	Yes	SMP/ Beowulf
5	Free License	Yes	No	Yes	Yes	No	No
6	Scalability	Large	Very Large	Small	Large	Medium	Very Large
7	Documentation and user support	Good	Excellent	Excellent	Excellent	Excellent	Good
8	Clustering	Yes	Yes	n/a	Yes	Yes	Yes
9	Graphic User Interface (GUI)	Yes	Yes	No	No	Yes	Yes

b) Model Structure of OMNeT++

An OMNeT++ model consists of modules that communicate with message-passing. The active modules are termed 'simple' modules; they are written in C++, using the simulation class library. Simple modules can be grouped into compound modules and so forth, and the number of hierarchy levels is unlimited. The whole model, called 'network' in OMNeT++, is itself a compound module. Messages can be sent either via connections that span modules or directly to other modules. This is shown in Figure 3-11: the boxes with heavy borders represent simple modules and boxes with thin borders represent compound modules, while the arrows connecting the small boxes represent connections and gates (Varga & Hornig, 2008).

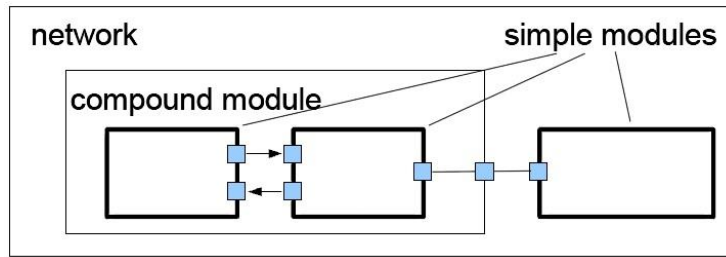


Figure 3-11: Model Structure in OMNeT++ (Varga & Hornig, 2008)

Modules communicate with messages, which may contain arbitrary data. Simple modules send messages via gates, which are the input and output interfaces of modules. An input and output gate can be linked with a connection that was created within a single level of module hierarchy; within a compound module, corresponding gates of two sub-modules or a gate of the compound module can be connected. Modules can have parameters that are used to pass configuration data to simple modules and to help define module topology. The model structure (i.e. the modules and the interconnection) is described in OMNeT++'s topology description language, Network Description (NED). NED lets the user declare simple modules, and to connect and assemble them into compound modules.

The model behaviour built into each NED file will be captured in C++ files as a code. After the program is started, it will first read all NED files, and then read a configuration file. The simulation can also perform with different inputs and all the values can be stored in a .INI file, usually called 'omnetpp.ini', containing settings that control how the simulation is executed.

The output of the simulation is written into result files: output vector files, output scalar files or histograms. Statistical methods are used to elicit the relevant data and reach a conclusion with the result by drawing a chart. The details of the simulation models, structure diagrams, the code of each modules and tests plan is provided in Appendix 2-A

As mentioned in 2.9.2e), a random number generator (RNG) is a program written for use in probability and statistics applications when large quantities of random digits are needed (Dally & Towles, 2004). Mersenne Twister is the random number generator employed by OMNeT++ (Varga, 2011), used to distribute the message destinations in the simulation model.

a) Mersenne Twister RNG

OMNeT++ primarily uses Mersenne Twister for random number generation. It uses the MT19937 RNG developed by Makoto Matsumoto and Takuji Nishimura in 1997 which has a cycle length of $2^{19937} - 1$ (Matloff, 2008). The Mersenne Twister has passed

numerous tests for randomness and is distributed uniformly in 623 dimensions, generating an output which is free of long-term correlations (Jagannatham, 2008). It is considered to be fast as it avoids multiplications and divisions by using the advantages of caches and pipelines.

A configurable number of random numbers are provided to the simulation. Global random number streams are mapped to OMNeT++'s module which allows the use of variance reduction techniques without the need to change the configuration in the simulation model (Varga & Hornig, 2008). While seeding is automatic, auto-assigned using the run number, it is also possible to use manually-selected seeds. The simulation requires as many seeds as the number of global RNG streams configured. Due to the practically infinite cycle length of Mersenne Twister, overlapping of RNG streams is not an issue.

b) Seeding the Mersenne Twister RNG

Seeding is the procedure of setting the initial states of the RNG, so that it will produce a stream of random numbers (Wehrle et al., 2010). The RNG class implements support for seeding. Seed sets can be specified in the initialization section or for each run of OMNeT++. Mersenne Twister has such a long cycle that there is no need for seed generation because chances are very small that any two seeds produce overlapping streams (Matloff, 2008).

c) Chi-square Goodness of Fit Test

This is a non-parametric test used to find out how the observed value is significantly different from the expected value. In Chi-Square test, the term *goodness of fit* is used to compare the observed sample distribution with the expected probability distribution. This test also determines how well theoretical distribution (such as normal or Poisson) fits the empirical distribution. In this test, sample data is divided into intervals. Then the numbers that fall into each interval are compared with the expected numbers in each interval. The formula for the statistic is:

$$\chi^2 = \sum \frac{(\text{observed value} - \text{expected value})^2}{\text{expected value}}$$

A high value of χ^2 implies a poor fit between the observed and expected value, so the upper tail of the distribution is used for most hypothesis testing for goodness of fit. To determine whether the traffic generation rates are random, the null and alternative hypotheses are as follows:

H_0 : Traffic generation rates are random

H_1 : Traffic generation rates are not random

Chi-Square	0.001
Degrees of Freedom	9
p-value	1.0

Table 1: Chi-Square goodness of fit test

Table 1 shows the test statistics and p-value. Since the p-value = 1.0 > 0.05, the null hypotheses was not rejected. At the $\alpha=0.05$ level of significance, there was not enough evidence to reject the null hypotheses, thus, the RNG fitted the theory that the traffic generation rates are random.

Appendix 3-A.

3.7 The MCMCA Simulation Model

The MCMCA simulation model is a descriptive model for investigating the performance of a multi-core cluster system, using a simulation model to experiment with, evaluate and compare various configurations and design parameters. The simulation model is an imitation of a physical network architecture to predict its performance in the real world. To get more insight on the feasibility of the MCMCA simulation model, different designs were developed based on two flow controls, store-and-forward and wormhole.

In order to illustrate the feasibility and the accuracy of the simulation model, a set of baseline experiments were conducted and compared with previous research, using several system configurations. The details of the actual running of the simulations including screen shots of parameter entries, simulation progress and results is demonstrated in Appendix 3-B. The next section presents a baseline experiment conducted to answer sub-research question SQ3:

'How well does the MCMCA simulation model analyse cluster performance?'

3.7.1 Experiments with a Single-core Cluster

This section presents the results of experiments based on Bahman's model (Javadi, Akbari & Abawajy, 2006) conducted on single-core clusters based on the interconnection network parameters listed in Table 3-2 and model cases in Table 3-3.

Table 3-2: Interconnection network parameters

Parameter	Internal-cluster	External-cluster
Network latency	0.01s	0.02s
Switch latency	0.01s	0.01s
Network Bandwidth	1000 b/s	500 b/s

Table 3-3: Model cases for single-core clusters

C,m,n	Message Length (M)	Flit length (F)
8,8,2	32 flits	256 bytes
8,8,2	32 flits	512 bytes
8,8,2	64 flits	256 bytes
8,8,2	64 flits	512 bytes

The results of the simulations with message length (M) = 32 flits are depicted in Figure 3-12 and Figure 3-13, in which average message latencies are plotted against the traffic rate with the store-and-forward flow control mechanism. The figures demonstrate the analytical results plotted against those provided by the simulation results predicted by Bahman's model (Javadi et al., 2006). The X-axis represents the traffic generation rate, while the Y-axis denotes the communication latency. The results in Figure 3-13 are derived from simulation of the new architecture using the same configuration and parameters as Bahman's model. These figures reveal that the latency results obtained from the simulation closely match to those obtained from Bahman's model.

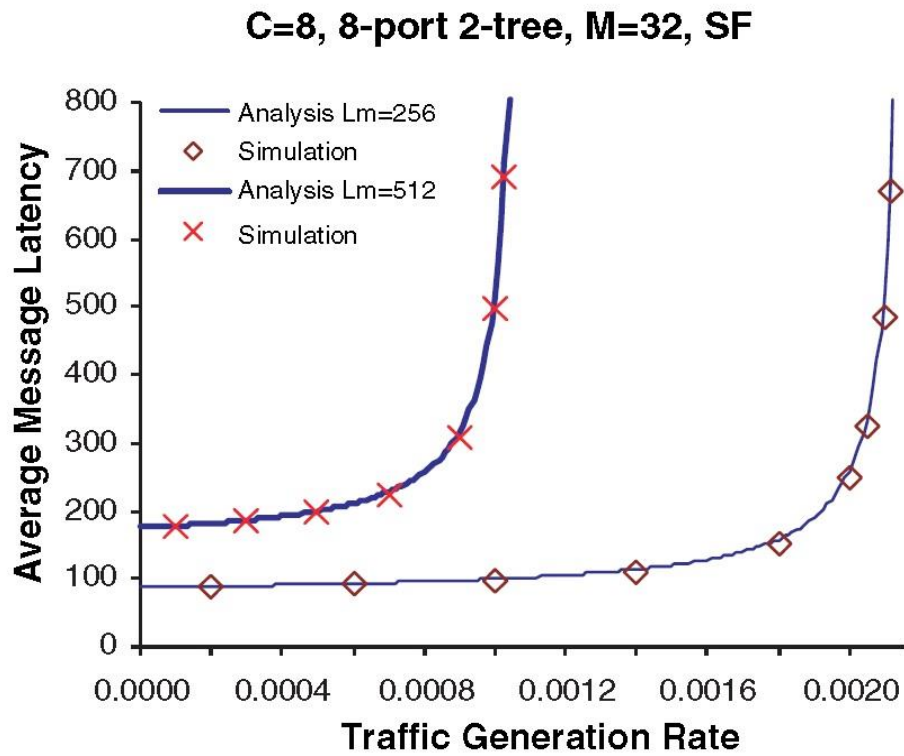


Figure 3-12: Bahman's model for 8-cluster system with M=32

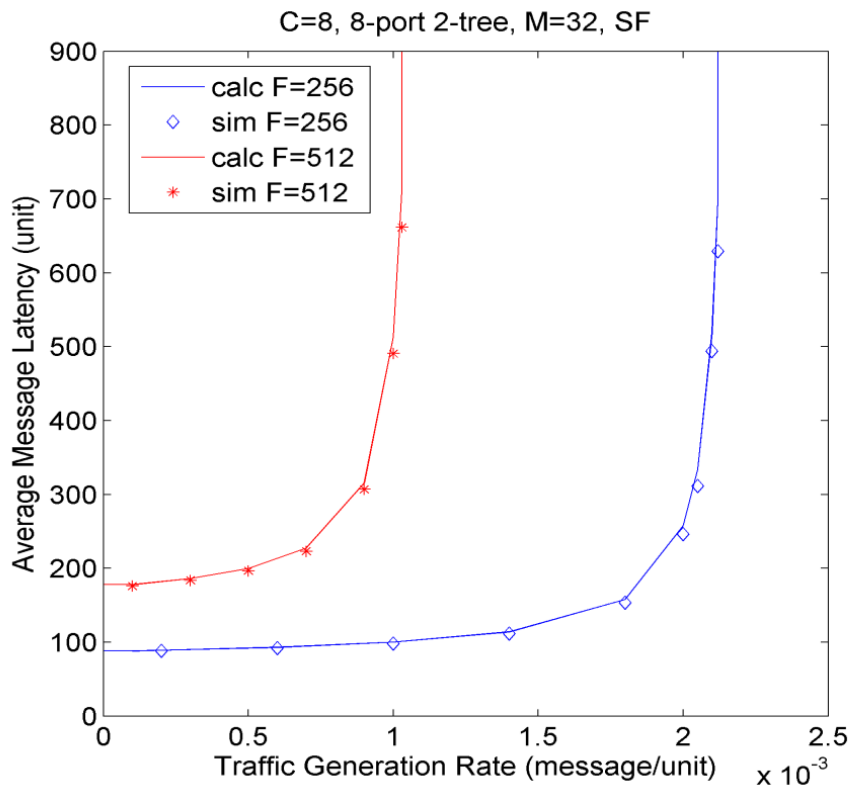


Figure 3-13: MCMCA model with C=8, M=32 based on Store-and-Forward Flow Control

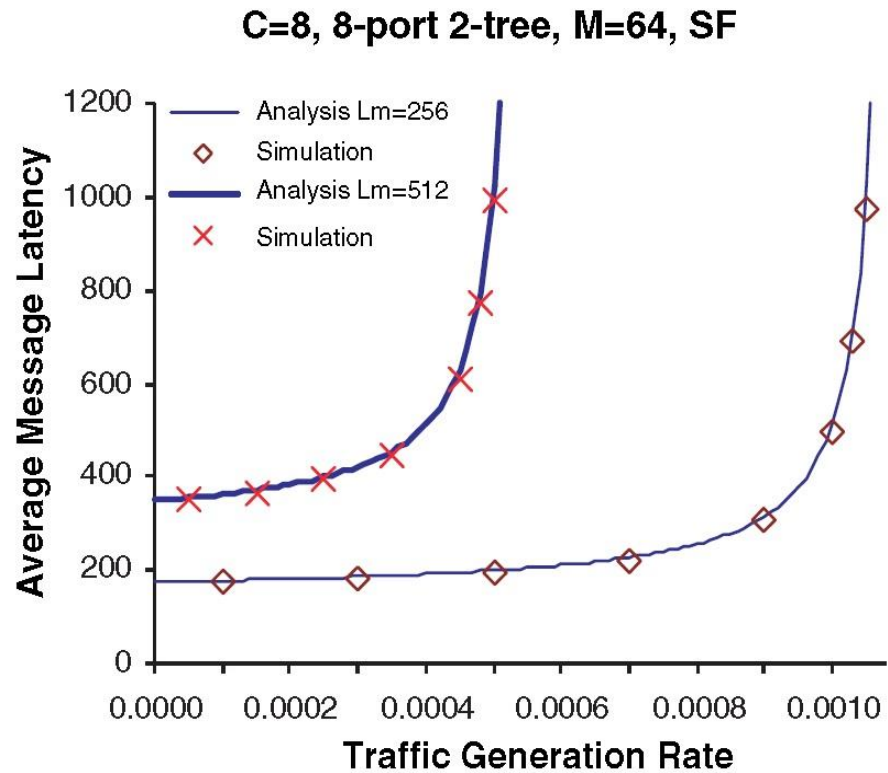


Figure 3-14: Bahman's model for 8-cluster system with M=64

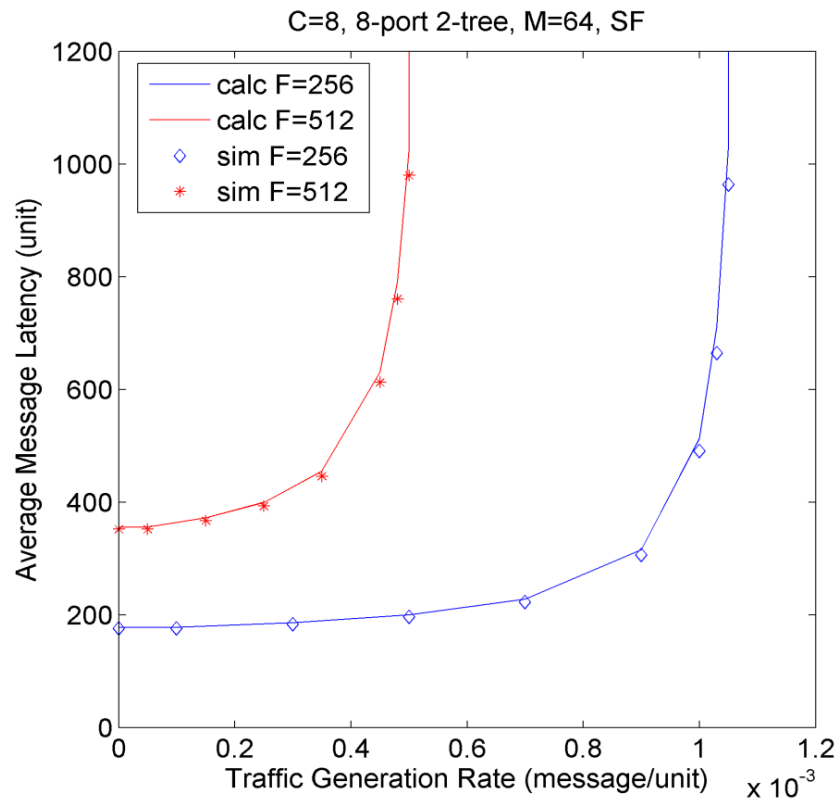


Figure 3-15: MCMCA model with C=8, M=64 based on Store-and-Forward Flow Control

Figure 3-15 shows the simulation results of the new architecture, MCMCA, for an 8-cluster system with $M=64$ using the same configuration as in Table 3-2 and Table 3-3 and as in Bahman's model in Figure 3-14. As traffic increased, increased contention causes latency to increase as messages must wait for buffers and channels, but at low traffic latency zero-load latency is approached. The zero-load latency assumption is that a packet never has to contend for network resources with other packets. It gives a lower bound to the average latency of a packet through the network. These figures reveal that the latency results obtained from the MCMCA with 1-core closely matches those obtained from Bahman's model.

These results show that there are inconsistencies when the analytical model and the simulation are under heavy traffic and start to saturate with less than 4-7% difference. However, under light traffic the analytical model differs from the simulation by less than 2%. This is due to the approximations to simplify the development of the model in Bahman's analytical model (Bahman Javadi et al., 2008a).

In conclusion, experiments with different message lengths (M) demonstrated that the MCMCA architecture predicts the average message latency with a good degree of accuracy and can be extended to large-scale cluster architecture.

3.7.2 Experiments with Multi-core Clusters

To test the validity of the simulation of the MCMCA, a simulation experiment was performed based on model cases in Table 3-4. Two different flow control mechanism, store-and-forward and wormhole, are used to verify the simulation model.

Table 3-4: Model cases for multi-core clusters

Items	Quantity
No. of cores (nc)	1, 2, 4
Message Length (M) and Flit Length (F)	32 flits, 256 bytes
No. of cluster, m -port, n -tree	8, 8, 2

Figure 3-16 depicts the average message latency vs. the traffic generation rate curve based on store-and-forward flow control, and Figure 3-17 shows the average message latency with the same traffic generation rate for the wormhole flow control with single-core and multi-core processors. As the traffic increases, the contention latency begins to dominate and the vertical asymptotes of the latency curves are determined by the saturation throughputs of the different flow control mechanisms. Both figures show that the traffic starts to saturate at about 50% of the traffic rate capacity with dual-core processors, but higher throughput is achieved with quad-core processors. These results confirmed that the simulation model of MCMCA architecture could predict the average message latency under various design issues.

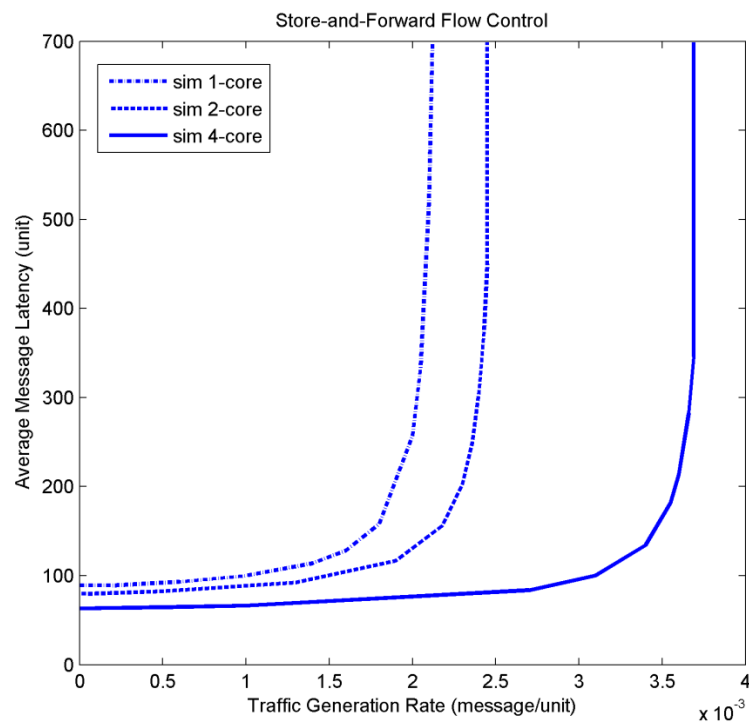


Figure 3-16: MCMCA Simulation Results based on Store-and-Forward Flow Control

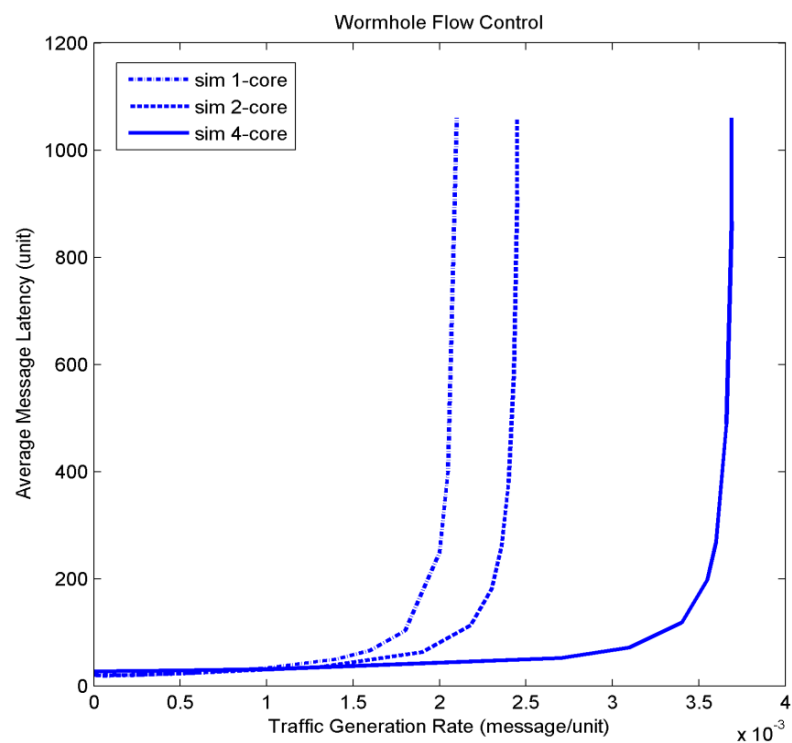


Figure 3-17: MCMCA Simulation Results based on Wormhole Flow Control

3.8 Summary

This chapter has presented a novel architecture as well as the simulation model for measuring the performance of interconnection networks in Multi-Core Multi Cluster Architecture (MCMCA), in order to answer research question RQ1 and sub-research question SQ1, SQ2 and SQ3. The architecture presented identifies the perceived core aspects of improving the network performance of multi-core clusters. This architecture will assist programmers in developing network applications with optimum network performance potential. By integrating multi-core processors and multi-clusters, the architecture's purpose is to provide an alternative for interconnection networks that improves their performance.

The new architecture is intended to improve this performance by allowing higher throughput and minimising network latency. The new architecture can thus be evaluated by two metrics: latency and throughput. This chapter started with introducing the new architecture designed to overcome the interconnection issues discussed in the literature. The new structure of interconnection networks was intended to provide greater throughput by reducing the waiting time.

Research methodology involved simulation development and experiment. The validation of the simulation model compared simulation experiments with various configurations and design parameters as conducted for Bahman's model, to match the work reported in earlier papers. The new model was examined with two message lengths (M) to evaluate the performance under various workload conditions. The results showed that, based on the assumptions that the messages have to wait for resources before traversing into a network, as traffic rate increases, the average latency will increase, while at low traffic density, latency will approach zero-load latency. Simulation experiments have revealed that the results obtained from the multi-cluster simulations compare with previous models and are closely matched. These results have shown that the simulation model can be extended to investigate the performance of a multi-core cluster system based on multi-cluster architecture. The simulation models are general and applicable to predicting the performance of a multi-core cluster.

The next chapter presents the methodology and performance evaluation of this research, leading to the implementation and evaluation of the proposed new architecture.

Chapter 4 The Performance Model using Store-and-Forward Flow Control Mechanism

4.1 Introduction

Chapter 3 reviewed the various structure of high performance computing focusing on cluster systems and multi-core processors. New approaches are proposed to enhance the outcome of cluster systems in term of improving the performance and reducing the network latency with efficient interconnection networks. This chapter aim to examine through simulation and experiment in order to answer the research question RQ2:

What is an appropriate flow control mechanism for communication latency modelling of the Multi-core Multi-cluster Architecture (MCMCA)?

Hence, this RQ2 addresses the following sub-research questions SQ4:

What is the impact of the flow control mechanism in improving communication latency?

The development of the MCMCA simulation model started with a small 8-cluster system (C) comprising a single-core processor with one interconnection network. To build the MCMCA simulation model using OMNeT++, modifications were made by changing this small cluster into progressively larger clusters, C=16, 32, 64, 128, with multi-core processors. Since the simulation development involved complex and large scale computation, it consumed more time by ran longer than expected. The basic design of a large cluster with a single-core processor was completed as planned so that it was tested for baseline results, as mentioned in section 3.7. The simulation results were compared with earlier similar research with single cores and demonstrated a good degree of accuracy.

The simulation results were validated with an analytical model using mathematical calculation, but inconsistencies became apparent when the analytical model and the simulation were under heavy traffic. Despite the small differences, the results demonstrated that the model can be extended to Multi-core Multi-cluster Architecture (MCMCA).

Another challenge in simulation development was the ability of the hardware to run the simulation execution. For each experiment, 10 traffic rates with 10 latency results were needed to form a single graph. Since the simulation involved larger

clusters with 10,000 messages for every traffic rate, using the normal high specification workstation, the simulation took about 1 hour to complete its cycle. This consumed around 10 to 12 hours to complete one full simulation experiment. The simulation was therefore ported to the 'Iridis Compute Cluster' (Admin, 2013), the University of Southampton's High Performance Computing (HPC) facility. By comparison, the simulation took around 10 minutes to complete one traffic rate, and total of around 2 hours to complete one experiment.

The main structure in the simulation model is its interconnection network. The simulation model was built at run time to form a topology based on a scalable fat-tree topology (Lin, 2003) that represents the geometric structure. As the simulation model setup is to send the packet in the shortest direction, this work will apply the 'Up*/Down*' deterministic routing (A. Gomez et al., 2011), a deadlock-free deterministic routing for interaction between modules. In order to determine the allocation of the network's resources, the research experiment focused on buffered flow control, with store-and-forward flow control representing packet-buffer flow control, and wormhole flow control indicating flit-buffer flow control. 'Mersenne Twister' is the random number generator employed by OMNeT++, used to distribute the message destinations in the simulation model. This thesis will also focus on Gigabit Ethernet, a high performance network technology, as it is synonymous with the cluster system.

The critical issue for network architecture is the impact of network latency on overall network performance. This chapter thus delves into an evaluation of the latency and throughput performance of networks constructed from the Multi-core-Multi-cluster Architecture (MCMCA) proposed in this thesis, with two flow control mechanisms: store-and-forward and wormhole.

However, to improve the performance of the interconnection networks, having a good flow control mechanism can minimise the delay and waiting time (Akhter & Roberts, 2006; Rauber & Runger, 2010). Evaluating the performance of interconnection networks using various flow control mechanism in Multi-core Multi-cluster Architecture (MCMCA) is therefore, important for predicting the performance potential. To this end, this chapter presents a new simulation model to investigate the performance of interconnection networks of a multi-core multi-cluster architecture based on a store-and-forward flow control mechanism. The performance of single-core cluster and multi-core cluster architectures are compared through simulation experiments and analytical model analysis is used to validate the simulation results. The main performance metrics to be simulated are latency and bandwidth. The model is then

used to evaluate the impact of interconnection network performance on scalability and cluster size.

The methodology guiding the simulations and evaluation is outlined in section 4.4, and the mathematical analysis to validate the simulation results is described in section 4.5. Various performance evaluations are performed in section 4.6.

4.2 Store-and-Forward Flow Control Mechanism

'Store-and-forward flow control' is a packet switching mechanism whereby the message to be transmitted is partitioned into a sequence of packets (Rauber & Runger, 2010). Each packet is sent separately to its destination according to routing information. Store-and-forward flow control allocates channel bandwidth and buffer resources to packets, and the resources are used by one packet at a time, as shown in Figure 4-1. Figure 4-1 contains 4 packets, a to d, which are completely transmitted across one channel before transmission across the next channel is started. With store-and-forward flow control, each node along a route waits until a packet has been completely received or stored and then forwards the packet to the next node.

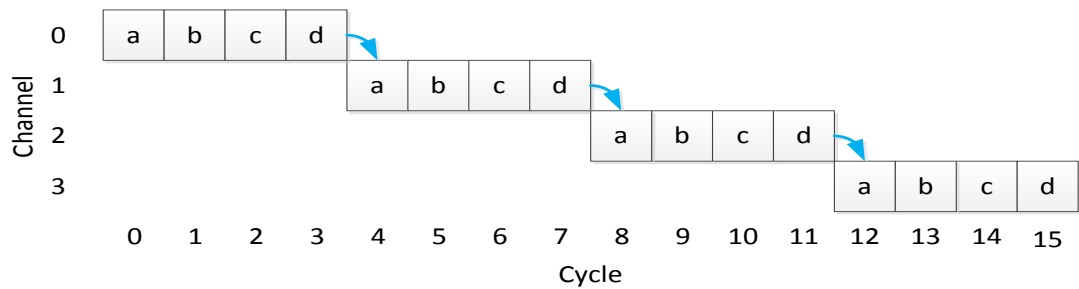


Figure 4-1: Flow diagram of the Store-and-Forward flow control mechanism
(reproduced from (Dally & Towles, 2004))

4.3 Assumptions and Notations

4.3.1 Assumptions

The simulation model and the analytical model were built on the basis of the following assumptions, which are used in similar studies (Bahman Javadi et al., 2008b; B. Javadi et al., 2006; Yulei et al., 2012):

1. Each processor generates packets independently, following a Poisson distribution with a mean rate of λ , and inter-arrival times are exponentially distributed.

2. The destination of each message is any node in the system with uniform distribution.
3. The numbers of processors and cores in all clusters are the same, and the cluster nodes are homogeneous.
4. The communication switches are input-buffered, and each channel is associated with a single packet buffer.
5. Message length is fixed.

4.3.2 Notations

The notations which are used in developing both models are presented in Table 4-1.

Table 4-1: Notations used in MCMCA

<i>Abbr.</i>	<i>Description</i>
<i>AC</i>	Intra-chip Network
<i>EC</i>	Inter-chip Network
<i>ACN</i>	Intra-cluster Network
<i>ECN</i>	Inter-cluster Network
<i>MCN</i>	Multi-cluster Network
<i>C</i>	Number of clusters in the architecture
<i>M</i>	Number of ports in m-port n-tree fat-tree topology
<i>n</i>	Number of trees in m-port n-tree fat-tree topology
<i>nc</i>	Number of cores in each processor
<i>M</i>	Message length (flit)
<i>F</i>	Flit length (bytes)
<i>N</i>	Number of nodes
ρ	Number of processors in each cluster
<i>nt</i>	Number of trees in the MCN
λ_i	Arrival rate in ICN (s)
λ_e	Arrival rate in ECN (s)
λ_{sw}	Arrival rate in MCN (s)
<i>SS</i>	Number of stages in the network
<i>S</i>	Stage number
β	Time to transfer packet/flit between two adjacent switches
α	Time to transfer packet/flit between a switch and a node
<i>P</i>	Probability for message exit from a cluster
$P(f, nc)$	Probability of a message crossing 2f-link
α_{net}	Network latency
α_{sw}	Switch latency
β_{net}	Network bandwidth

W_i	Average waiting time in internal-cluster
W_e	Average waiting time in external-cluster
W_{sw}	Average waiting time at transfer switches
T_i	Average transmission time in internal-cluster
T_e	Average transmission time in external-cluster
R_i	Average time for the last packet/flit to reach its destination in the internal-cluster
R_e	Average time for the last packet/flit to reach its destination in the external-cluster
$\bar{\Sigma}L_i$	Average message latency in the internal-cluster
$\bar{\Sigma}L_e$	Average message latency in the external-cluster
$\bar{\Sigma}L$	Average message latency of interconnection networks in MCMCA

4.4 Evaluation Methodology

The simulation model and experiments were developed and performed using OMNeT++ Network Simulation Tool as mention in 3.6.1. This section gives details on simulation modules and experiment designs.

4.4.1 Simulation Structures

The topology and the communication links between the modules are represented by the network description, NED. Six modules were built to describe the simulation model, as follows:

1. Network Topology module
2. Network Interface module
3. Communication Switch module
4. Routing module
5. Message-Generator module
6. Message-Sink module

a) The Network Topology Module

This module develops the building blocks of the fat-tree topology, including cores, nodes and clusters. The simulation structure was mainly based on interconnection network properties: network topology, routing and switching strategies, and the flow control mechanism.

Interconnection networks are composed of a set of shared router nodes and channels, and the topology of the network refers to the arrangement of these nodes and channels. The ‘fat tree’ is an efficient network topology to provide high performance and low latency structures in cluster systems (Prisacari et al., 2013; Yulei et al., 2012). The literature in fat tree has been discussed in section 2.9.2a).

A fat-tree topology is a complete ‘tree’ that gets thicker near the root. In this thesis, we employed an m -port n -tree to construct the MCMCA architecture. Moreover, the m -port n -tree topology is a full bisection bandwidth (Lin, 2003), which will eliminate contention and optimize a throughput. The network organized as a matrix of routers with n rows, and each network switch has m communication ports that are attached to other switches or processing nodes. The m -port n -tree consists of N processing nodes and S_w communication switches (Xuan-Yi et al., 2004) which are given in the equation

$$N = 2 \left(\frac{m}{2} \right)^2 \quad [4.1]$$

$$S_w = 2n - 1 \left(\frac{m}{2} \right)^{n-1} \quad [4.2]$$

The number of channels, N_{ch} , in this topology were given by

$$N_{ch} = 4n \left(\frac{m}{2} \right)^n \quad [4.3]$$

To achieve a low latency, the topology must maintain a small average distance between nodes. The distance between nodes is measured as, D , where messages need to traverse in number of channels and processing nodes on average to reach its destination. For $n > 1$, the equation for distances between nodes (Xuan-Yi et al., 2004) is as follows:

$$D = \frac{(nm - 2n - 1) \left(\frac{m}{2} \right)^n + 1}{\left(\left(\frac{m}{2} \right) - 1 \right) \left(\left(\frac{m}{2} \right)^n - \frac{1}{2} \right)} \quad [4.4]$$

All the equations above are preliminaries to the simulation input in the simulation experiment.

b) The Network Interface module

This module contains the interface of module types in fat-tree topology. Cores, nodes, clusters, switch, channel and the interconnection network are declared in this module and connections between them are established.

The number of clusters (C), number of processing nodes (N), m -port n -tree and message lengths (M) comprise the basic simulation input for each experiment. Number of processing nodes (N) is based on one processor in each node (Lin, 2003). However, for MCMCA, the number of processors in each cluster, ρ , is depends on number of core inputs (nc). To simulate the multi-core processors in MCMCA, the number of processors in a cluster will multiply with number of cores, where nc must be smaller or equal than number of clusters, C , following the Assumption 3; the numbers of processors and cores in all clusters are the same. The equation can be expressed as –

$$\rho = 2nc \left(\frac{m}{2}\right)^2, \quad nc \leq C \quad [4.5]$$

Thus, the packets will be distributed into a designated number of cores throughout the complete cycle, with the same number of cluster nodes based on message probability by given equation (Shahhoseini et al., 2000).

$$P_o = \frac{N - \rho}{N - 1} \quad [4.6]$$

$$P_i = 1 - P_o \quad [4.7]$$

c) The Message-Generator module

This will follow the generation of messages at each node with a message-generator module based on the assumptions that the network traffic follows a uniform Poisson distribution (Sadeghi & Barati, 2012). Messages are divided into packets or flits, the assumption being that the message destinations are uniformly distributed. The message destination is distributed by P_o , the probability for a message to exit from a cluster, and P_i , the probability of messages staying in a cluster, given by equations 4.6 and 4.7.

d) The Routing module

A routing module determines the route taken by a packet from source node to destination node. This module determines the path and schedules the routing algorithms for the packets in all communication networks based on FIFO

(First-In-First-Out), and it represents a single server queue that provides the same service rate for each packet.

Based on message probability expressed in equations 4.6 and 4.7, this module will determine total packet to be executed in internal-cluster and external-cluster. By theory, when number of cores in a cluster increase, more task can be done internally with less packets will be distributed to external-cluster. Therefore, lower latency and higher throughput can be achieved.

The attractive properties of deterministic routing motivate the use of the Up*/Down* routing algorithm (Schroeder et al., 1991). 'Up*/Down* routing' is deadlock-free routing which is required to avoid deadlock occurrence in the architecture. In this routing algorithm, a message will experience two phases: an ascending phase to get to the nearest common ancestor (NCA), followed by a descending phase until the message reaches its destination. This routing algorithm is able to perform balanced traffic distribution.

e) The Communication Switch module

This module acts as the connection for each switch and router in the model, and it will determine how a message is transmitted along a path that has been selected by the routing algorithm. This module also acts as a flow control mechanism to allocate resources such as channel and buffer to the packets. Since this chapter describes an evaluation based on store-and-forward flow control, the blocking probability is zero. This makes the transmission time of a packet to the next switch equal to the transmission time on a node to a switch connection, α_{ie} . This simulation adopts two types of connection: α is the time taken for a packet of the message to transmit on a node-to-switch (or vice versa) connection, while β is the time taken for a packet of the message to transmit on a switch-to-switch connection (B. Javadi, Akbari & Abawajy, 2005). M is the message length, α_{net} and α_{sw} are the network and switch latencies, and β_{net} is the transmission time of one byte.

$$\alpha_{ie} = 0.5 \alpha_{net} + M \frac{1}{\beta_{net}} \quad [4.8]$$

$$\beta_{ie} = \alpha_{sw} + M \frac{1}{\beta_{net}} \quad [4.9]$$

f) The Message-Sink module

This module will destroy the packets after each generation is completed and will gather event information for statistics. The sink module collects latency and

throughput at the packet and flit levels. The 'batch mean method' (Dally & Towles, 2004) is used to collect the statistics of the network performance for each simulation experiment. This method provides an average over the total number of messages, which are divided into many batches, and statistics are accumulated for these batches. The use of the batch mean method to gather the statistics reduces the standard deviation of the measurement, which leads to higher confidence in the estimation.

4.4.2 The Simulation Activity Diagram

Figure 4-2 depicts the simulation activity diagram, showing the connection between the simulation modules. After the generator module start distributes the packet, the routing module will determine the path by which the packets traverse the network. The flow control mechanism will allocate a channel to the packet, and since the store-and-forward flow control is a buffer flow control, a buffer will be allocated to the packet as well. When the buffer is available to hold the packets, it will send a request to the communication switch for the output port. Once the passage is granted, the packet leaves for the crossbar switch and goes on to the next step.

4.4.3 Simulation Experimental Setup

This thesis focuses on the measuring of the steady-state performance of a network: the performance of a network with a stationary traffic source after it has reached steadiness. A network has reached 'steadiness' when its average queue lengths have reached their steady-state values. To measure steady-state performance, the simulation experiments were conducted in three phases: warm-up, measurement and drain (Dally & Towles, 2004). To minimise systematic errors, the first 10% of the messages are discarded as part of the 'warm-up phase' and the last 10% in the 'drain phase' were not used in the measurement, so as to provide enough time for all packets to reach their destination. 'Warm-up' is when the influence of the simulation initialisation is minimal. The network has reached its steady state once the network is warmed up (Dally & Towles, 2004). This means that the statistics of the network are stationary and no longer change with time, which will determine an accurate estimation.

Each simulation experiment was run until the network reached its steady state, meaning that a further increase in the simulation time did not change the statistical results. The communication between processors relies on a message passing between the source and its destination. The message passes over a channel that directly connects two processor cores and might have to pass through several processor cores based on the designate flow control before it reaches its destination, where each

communication involves a lot of latency. To generalise the simulation details in this section, 'node' will represent processor and core as well as itself.

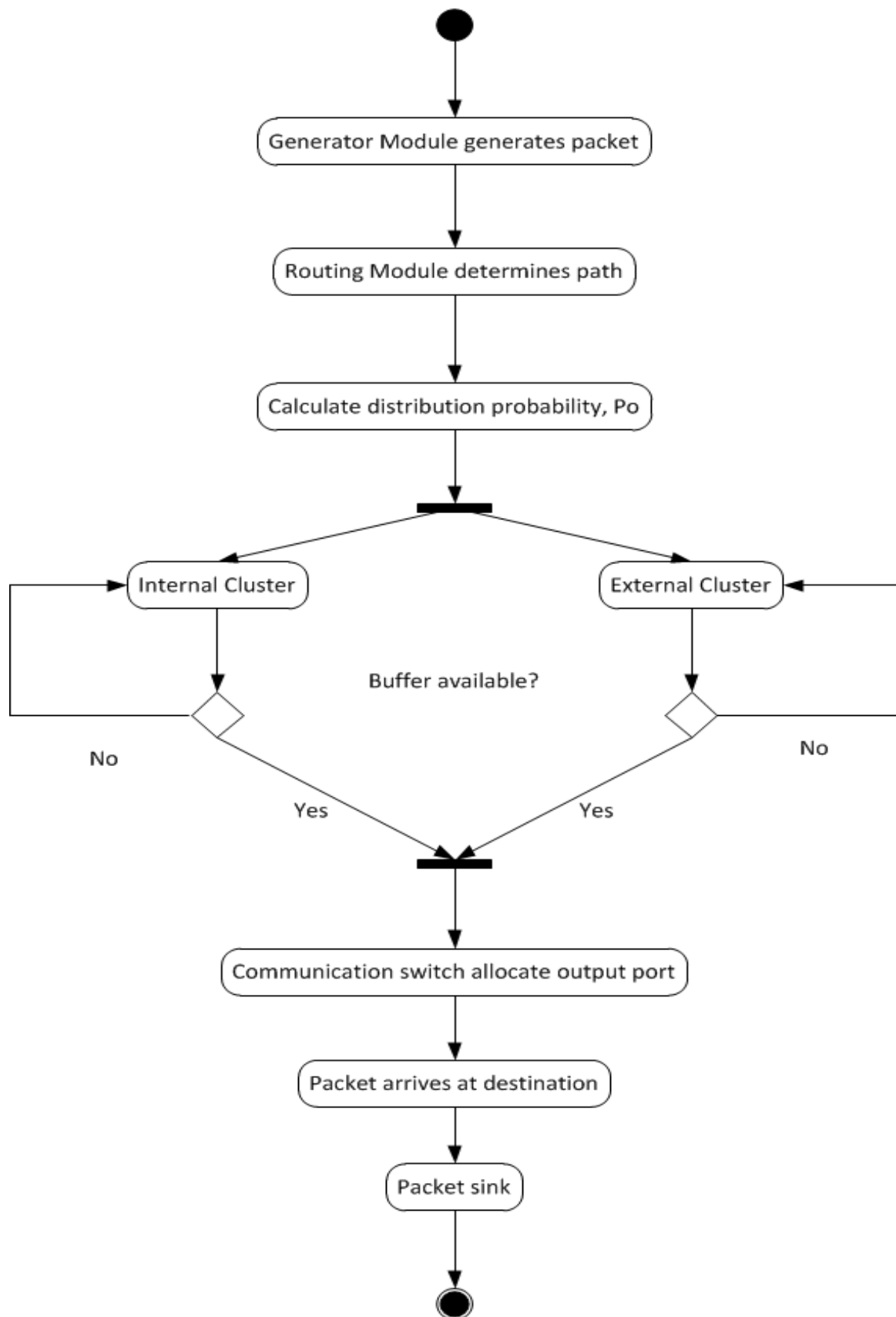


Figure 4-2: Activity diagram of MCMCA simulation model

In these simulation experiments, a total of 100,000 messages were used to gather statistics. The first 10,000 messages were discarded, as they were assumed to occupy the warm-up phase before the simulation reached its steady state; and the last 10,000 messages were discarded in the drain phase. In these experiments, 100,000 messages were divided into 10 batches, where the size of each batch was 10,000 messages.

As mentioned in section 4.4.1b), simulation inputs must be predetermined before each run. The simulation can behave with different inputs and specified parameters of the model, such as the number of cores (nc), number of clusters (C), number of messages to be generated (λg), message length (M) and inter-arrival time (λ). This will be followed by the generation of messages at each node by a message-generator module, based on the assumptions that the network traffic follows a uniform Poisson distribution (see section 2.9.2d) above). The addresses of the source node and destination node are randomly produced, based on the number of nodes and the number of switches.

Each simulation experiment started with the creation of the simulation structure based on fat-tree network topology by the network topology module. Each network switch has m communication ports that are connected with other switches or router or nodes with n level of tree. The switch connection is established by a communication switch module which will also transmit the messages based on the route selected by routing algorithm. The switch connection for an internal cluster is represented by 'ISwitch', and by 'ESwitch' for an external cluster, as reflected in Algorithm 4-1 and Algorithm 4-2 below.

Algorithm 4-1: Internal cluster switch connection (ISwitch) for store-and-forward flow control mechanism

```
// Internal cluster
void AbstractinFifo::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);

    simtime_t d = simTime()-msg->getTimestamp();

    if (gmsg!=NULL){
        if (gmsg->getRouted()==0)
        {
            endService(msg, 0);
            return;
        }
    }
}
```



```

    }
    gmsg->setRouted(0);

}
if (msg==endServiceMsg)
{
    endService( msgServiced ,1);
    if (queue.empty())
    {
        msgServiced = NULL;
    }
    else
    {
        msgServiced = (gensinkMsg *) queue.pop();
        simtime_t serviceTime = startService( msgServiced );
        scheduleAt( simTime()+serviceTime, endServiceMsg );
    }
}
else if (!msgServiced)
{
    arrival( msg );
    msgServiced = gmsg;
    simtime_t serviceTime = startService( msgServiced );
    scheduleAt( simTime()+serviceTime, endServiceMsg );
}
else
{
    arrival( msg );
    queue.insert( msg );
}
}

```

Algorithm 4-2: External cluster switch connection (ESwitch) for store-and-forward flow control mechanism

```

// External cluster

#include "efifo.h"
#include <math.h>
#include "gensinkMsg_m.h"

void AbstracteFifo::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);

    simtime_t d = simTime()-msg->getTimestamp();

    if (gmsg!=NULL){

```

```

        if (gmsg->getRouted()==0)
        {
            endService(msg, 0);
            return;
        }
        gmsg->setRouted(0);
    }
    if (msg==endServiceMsg)
    {
        if (queue.empty())
        {
            msgServiced = NULL;
        }
        else
        {
            msgServiced = (gensinkMsg *) queue.pop();
            simtime_t serviceTime = startService( msgServiced );
            scheduleAt( simTime()+serviceTime, endServiceMsg );
        }
    }
    else if (!msgServiced)
    {
        arrival( msg );
        msgServiced = gmsg;

        simtime_t serviceTime = startService( msgServiced );
        scheduleAt( simTime()+serviceTime, endServiceMsg );
    }
    else
    {
        arrival( msg );
        queue.insert( msg );
    }
}

```

The generated message has the probability, P_o , to be directed to the external-cluster network nodes, and the probability, $(1-P)$, of being evenly distributed to all internal-cluster network nodes. In the simulation, when the number of source and destination address is equal, the messages will be routed to an internal cluster by the internal routing module; otherwise the messages are exited to an external cluster by the external routing module, as presented in Algorithm 4-3. The average amount of time that a message spends waiting in each queue in the architecture estimates the message latency based on an M/G/1 queuing network model, as described in section 3.4 above. Lastly, each message is time-stamped after its generation, and the message

completion time is defined by a message-sink module on each node in order to compute the statistics.

Algorithm 4-3: Message probability in the internal cluster and external cluster

```

if ((int)(src/num_nodes) == (int)(dest/num_nodes)){
    // internal-cluster
    ev << "Send to internal Cluster" << endl;
    gmsg->setKind(0);

    if (gate("out",0)->getTransmissionChannel()->isBusy())
        sendDelayed(gmsg, gate("out",0)->getTransmissionChannel()-
        >getTransmissionFinishTime()-simTime(),"out",0);
    else
        send(gmsg,"out",0);
    }

    else
    {

        // external-cluster
        ev << "Send to external Cluster" << endl;
        gmsg->setKind(1);

        if(gate("out",1)->getTransmissionChannel()->isBusy())

        {
            sendDelayed(gmsg, gate("out",1)->getTransmissionChannel()-
            >getTransmissionFinishTime()-simTime(),"out",1);
        }
        else
        {
            send(gmsg,"out",1);
        }
    }
}

```

4.5 The Analytical Model

The analytical model comprises a set of equations describing the performance of the cluster to support the simulation analysis (Fengguang et al., 2009). Analytical models are constructs to gain an understanding of the current activity of the system, to measure performance and analyse the behaviour of the workloads in a multi-core cluster architecture (Caliri, 2000). The analytical model was created to validate the results of simulation experiments using the Matlab application (Attaway, 2013). Some modification was made to the equations to comply with the MCMCA design.

4.5.1 Preliminaries

This section explains in detail the implementation of the analytical model to compute the communication latency of the interconnection networks in MCMCA. The analytical model also needs to define simulation inputs as preliminaries to other equations. The first phase involves computing ρ in equation 4.5, where ρ is the number of processors in each cluster and nc is the number of cores on each processor, which is the new parameter in the calculation. Here nt is the number of ‘trees’ in the MCN, while C is the number of clusters and m is the number of ports (Geyong et al., 2009).

$$nt = \frac{(\log_2 C) - 1}{(\log_2 m) - 1} \quad [4.10]$$

The messages enter an internal cluster or an external cluster based on the probability P . The probability of an outgoing request P_o represents the messages generated by the source nodes that are sent to an external-cluster network, while messages injected from a source node with the probability P_i enter an internal-cluster network. Both can be computed with equations 4.6 and 4.7.

To traverse in the network, each message may use a different number of channel links to reach its destination. The transmission time for internal-cluster and external-cluster networks can therefore be considered as a $2f$ -channel with an f -channel in the source node and an f -channel in the destination node (Xuan-Yi et al., 2004). The probability that a message reaches its destination in MCMCA, $P(f, nc)$, can be computed by:

$$P(f, nc) = \begin{cases} \frac{\left(\frac{m}{2} - 1\right) \left(\frac{m}{2}\right)^{f-1}}{2 \left(\frac{m}{2}\right)^{nc} - 1} & 1 \leq f < nc \\ \frac{(m - 1) \left(\frac{m}{2}\right)^{f-1}}{2 \left(\frac{m}{2}\right)^{nc} - 1} & f = nc \end{cases} \quad [4.11]$$

$$P(f, nc) = P(h, nt) = P(j, n) \quad [4.12]$$

4.5.2 Average Message Latency of an Internal-Cluster Network

The communication latency in an inter-cluster network includes messages travelling in an intra-chip network (AC), an inter-chip network (EC), and an intra-cluster network (ACN). The calculation involves average waiting time at the source node in section 4.5.2a), average transmission time for a message to cross the networks in section

4.5.2b), and average time for the last packet of the message to reach its destination in section 4.5.2c).

a) Average Waiting Time at the Source Node

In an internal-cluster network, the total arrival rate is λ_i and each message travels an average distance to cross the network (B. Javadi, Khorsandi & Akbari, 2005). Thus, every channel in an internal-cluster will receive messages at a rate of W_i , where β is the time taken for a packet of the message to transmit on a switch-to-switch connection as given by equation 4.8.

$$W_i = \frac{(\beta_i)^2 \lambda_i}{2(1 - \beta_i \lambda_i)} \quad [4.13]$$

$$\lambda_i = (1 - P) \left(\frac{1}{\lambda} \right) \quad [4.14]$$

b) Average Transmission Time for a Message to Cross the Networks

Since this architecture applies store-and-forward flow control, blocking does not happen (Dally & Towles, 2004). This determined that the average network latency is equal to the transmission time of a packet to the next switch. Thus, the average transmission time in internal-cluster network is:

$$T_i = \alpha_i \quad [4.15]$$

c) Average Time for the Last Packet of the Message to Reach its Destination

Averaging over all possible cores that can be destinations for a packet in the internal-cluster (Sharifi et al., 2009), the average time for the last packet to reach its destination in the internal-cluster R_i is as follows:

$$R_i = \sum_{f=1}^{nc} \sum_{j=1}^n \left[P_{(f, nc)} P_{(j, n)} \left(\sum_{s=1}^{SS_i-1} \beta_i + \alpha_i \right) \right] \quad [4.16]$$

Finally, the equation for message latency in the internal-cluster communication networks can be expressed as:

$$\bar{\Sigma} L_i = W_i + T_i + R_i \quad [4.17]$$

4.5.3 Average Message Latency of an External-Cluster Network

An external-cluster network involves an inter-cluster network and a multi-cluster network. Latency in the external cluster will be determined by the same entity as in an internal-cluster network. An external message will need to cross more communication networks to reach its destination in another cluster.

a) Average Waiting Time at the Source Node

Waiting time on an external-cluster network can be determined by the network channel which follows the M/G/1 queueing network, as mentioned in section 3.4. Messages generated by the source nodes are sent to the external cluster with the probability of an outgoing request P_o , where λ_e is the message arrival rate. Based on equation 4.13, the waiting time in the external-cluster network (W_e) can be computed by:

$$W_e = \frac{(\beta_e)^2 \lambda_e}{2(1 - \beta_e \lambda_e)} \quad [4.18]$$

$$\lambda_e = 2 \left(\frac{1}{\lambda} \right) P \quad [4.19]$$

b) Average Transmission Time for a Message to Cross the Networks

The analysis in equation 4.15 also applies to external-cluster network. Since the probability of blocking is zero, the transmission time is:

$$T_e = \alpha_e \quad [4.20]$$

c) Average Time for the Last Packet of the Message to Reach its Destination

The average latency for the last packet to reach its destination in the external cluster (Sharifi et al., 2009) can be computed by:

$$R_e = \sum_{j=1}^n \sum_{h=1}^{nt} \left[P(j, n) P(h, nt) \left(\sum_{s=1}^{SS_e-1} \beta_e + \alpha_e \right) \right] \quad [4.21]$$

d) Average Waiting Time at Transfer Switches

External-cluster messages need to cross transfer switches during their journey across the network. The transfer switches act as simple buffers to combine traffic from/to one

cluster to/from other clusters (Dally & Towles, 2004). The waiting time at these buffers W_{sw} , with λ_{sw} as the message arrival rate, can be computed as:

$$W_{sw} = \frac{(\beta_e)^2 \lambda_{sw}}{2(1 - \beta_e \lambda_{sw})} \quad [4.22]$$

$$\lambda_{sw} = P \left(\frac{1}{\lambda} \right) \rho \quad [4.23]$$

Therefore, the equation for message latency in external-cluster communication networks can be expressed as:

$$\bar{\Sigma}L_e = W_e + T_e + R_e + 2W_{sw} \quad [4.24]$$

4.5.4 Average Message Latency of MCMCA

Using equations 4.17 and 4.24, the average message latency of communication networks in the multi-core multi-cluster architecture can be obtained by the sum of the message latency in internal-cluster and external-cluster situations as follows:

$$\bar{\Sigma}L = \bar{\Sigma}L_i (1 - P) + \bar{\Sigma}L_e (P) \quad [4.25]$$

4.5.5 Implementation of the Analytical Model

Algorithm 4-5 presents the implementation of the analytical model to compute the communication latency of interconnection networks in MCMCA.

Algorithm 4-4 and Algorithm 4-5 presents the implementation of the analytical model to compute the communication latency of interconnection networks in MCMCA.

Algorithm 4-4: Process flow in calculating the communication latency of interconnection networks in MCMCA

Input Parameters:	
Number of clusters C , parameter of m-port n-tree, message length M , number of cores nc , number of nodes N and lambda $\frac{1}{\lambda}$.	
1:	Calculate probability of entering/outgoing messages in the cluster P_o , number of processors in each cluster ρ , and number of trees in the MCN nt by using equations 4.6, 4.5 and 4.10.
2:	Calculate arrival rate in internal-cluster λ_i , external-cluster λ_e and transfer switch

	λ_{sw} by using equations 4.14, 4.19 and 4.23.
3:	Calculate the probability of the message crossing 2 channels, $P(j,n)$ and $P(f,nc)$, by using equations 4.11 and 4.12.
4:	Calculate the time taken for a packet of the message to transmit on a node to a switch or <i>vice versa</i> α_{ie} , and the time taken for a packet of the message to transmit on a switch to switch β_{ie} for the internal and external cluster using equations 4.8 and 4.9.
5:	Calculate average latency in the internal cluster: <ol style="list-style-type: none"> Calculate W_i, the waiting times at the source node based on equation 4.13. Calculate R_i, the time for the last packet of the message reach its destination using equation 4.16.
6:	Calculate average latency in the external cluster: <ol style="list-style-type: none"> Calculate W_e using equation 4.18 Calculate R_e using equation 4.21 Calculate W_{sw}, the waiting time at the transfer switch using equation 4.22.
7:	Calculate the message latency in the internal cluster and the external cluster using equations 4.17 and 4.24.
8:	Calculate $\bar{\Sigma}L$, the average message latency of interconnection networks in MCMCA, using equation 4.25.

Algorithm 4-5: Pseudocode of process flow in calculating the communication latency of interconnection networks in MCMCA using MATLAB based on Store-and-Forward flow control

```

clear all
clc

n_core=2;  % number of core(s)

C = 8;
m = 8;
n = 2;
M = 256;
Lm = 1;

```



```

lambda=5000;
lambda=1/lambda;

pp=(m/2)^n; %number of processor per node
NP=2*pp; %number of processors in each cluster
NP=NP*n_core;
N=(NP/n_core)*C;

PO=(N-NP)/(N-1);
PI=1-PO;

lambdaI1=PI*lambdaG;

anetI = 0.02;
aswI = 0.01;
BnetI = 500;

tcnI=(0.5*anetI)+(M*Lm*1/BnetI);
tcsI=aswI+(M*Lm*1/BnetI);

WI1=lambdaI1*tcsI*tcsI/(2*(1-lambdaI1*tcsI));

DI=tcnI;

for j=1:n-1;
    P(j,n) = (((m/2)-1)*((m/2)^(j-1)))/(NP-1)
End

for j=n;
    P(j,n) = ((m-1)*((m/2)^(j-1)))/(NP-1)
end

for f=1:n_core-1;
    Pnc(f,n_core) = (((m/2)-1)*((m/2)^(j-1)))/((2*((m/2)^n_core))-1)
End

for f=n_core;
    Pnc(f,n_core) = ((m-1)*((m/2)^(j-1)))/((2*((m/2)^n_core))-1)
end

davgICN=(((n*m)-(2*n)-1)*(pp)+1)/(((m/2)-1)*(pp-0.5));

RI=(davgICN-1)*tcsI+DI;

EC=0;

for f=1:n_core;

```

```

    EC=EC+Pnc(f,n_core)*RI;
End

%external cluster

nc=log(C/2)/log(m/2);
pc=(m/2)^nc;

lambdaE1=2*PO*lambdaG;

lambdaI2=((NP/n_core)*PO*lambdaG);

davgECN=davgICN;

davgICN2=((nc*m)-(2*nc)-1)*(pc)+1/((m/2)-1)*(pc-0.5));

anetE = 0.050;
aswE = 0.02;
BnetE = 300;

tcsE= aswE+(M*Lm*1/BnetE);

tcnE=(0.5*anetE)+(M*Lm*1/BnetE);

WE1=lambdaE1*tcsE*tcsE/(2*(1-lambdaE1*tcsE));

DE=tcnE;

for j=1:n-1;
    P(j,n) = ((m/2)-1)*((m/2)^(j-1))/((2*((m/2)^n))-1)
End

for j=n;
    P(j,n) = (m-1)*((m/2)^(j-1))/((2*((m/2)^n))-1)
end

RE=((davgECN+davgICN2)-1)*tcsE+DE;

WCD=lambdaI2*tcsE*tcsE/(2*(1-lambdaI2*tcsE));

TE1=WE1+RE+(2*WCD);

TI = PI*TI1;
TE = PO*TE1;

TMC=PI*TI1+PO*TE1;

```

4.6 Performance Evaluation (SQ4)

This section presents the results for the Multi-core Multi-cluster Architecture simulation model based on store-and-forward flow control, and discusses their implication. These results will show the impact of the store-and-forward flow control mechanism in improving communication latency by answering sub-research question SQ4.

The latency and bandwidth experiments were carried out on a single-core processor, a dual-core processor and a quad-core processor. Analysis of the results as presented in section 4.3 provides validation of the simulation results.

Both simulation and analytical models are divided into internal-cluster and external-cluster results. The communication latency in internal-cluster networks includes messages travelling in the intra-chip network (AC), inter-chip network (EC) and intra-cluster network (ACN), while messages travelling in external-cluster networks communicate via two interconnection networks: the inter-cluster network (ECN) and multi-cluster network (MCN). Both clusters were determined by four factors:

1. Average waiting time at the source node;
2. Average transmission delay for a message to cross the networks;
3. Average time for the last packet of the message to reach its destination;
4. Average waiting time at transfer switch (external cluster only).

The simulation experiments were performed with various combinations of parameters by using the interconnection network parameters given in Table 4-2 with simulation input I from Table 4-3.

Table 4-2: Interconnection Networks Parameter I

Parameter	Internal cluster	External cluster
Network latency	0.01s	0.02s
Switch latency	0.01s	0.01s
Network Bandwidth I	1000 b/s	500 b/s
Network Bandwidth II	800 b/s	600 b/s

Table 4-3: Simulation Input I

Items	Message length (<i>M</i>) 8K
No. of cores (<i>nc</i>)	1, 2, 4
No. of cluster (<i>C</i>)	8
No. of <i>m</i> -port <i>n</i> -tree	8, 2

4.6.1 Latency and Throughput Performance on MCMCA

The 'average message latency' is defined as the average amount of time elapsed from the generation of a message until the last packet reaches the destination node. The 'transmission time' is the network cycle time taken by a single packet to travel from one node to another node in the simulator. The 'throughput' is the rate at which traffic is delivered to the destination. When traffic is less than saturation level, the throughput is equal to the traffic generation rate.

To investigate the latency effect on MCMCA, the simulation experiments involved a single-core processor, a dual-core processor and a quad-core processor for comparison. To form the latency curve, a total of 10 different message generation rates, λ_g , were used for each core, and the accuracy of each result was validated by the analytical model. For the investigation of network throughput, the performance with 8 KB message length per traffic rate was graphed.

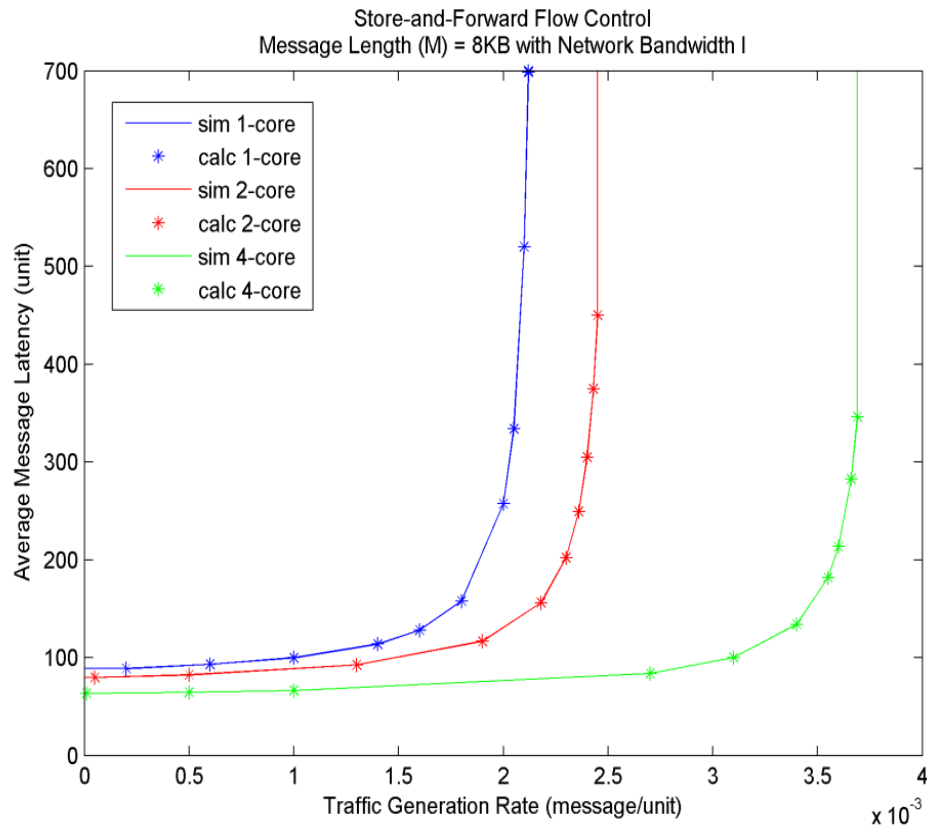


Figure 4-3: Message latency and throughput results based on store-and-forward flow control mechanism with M=8 KB using Network Bandwidth I

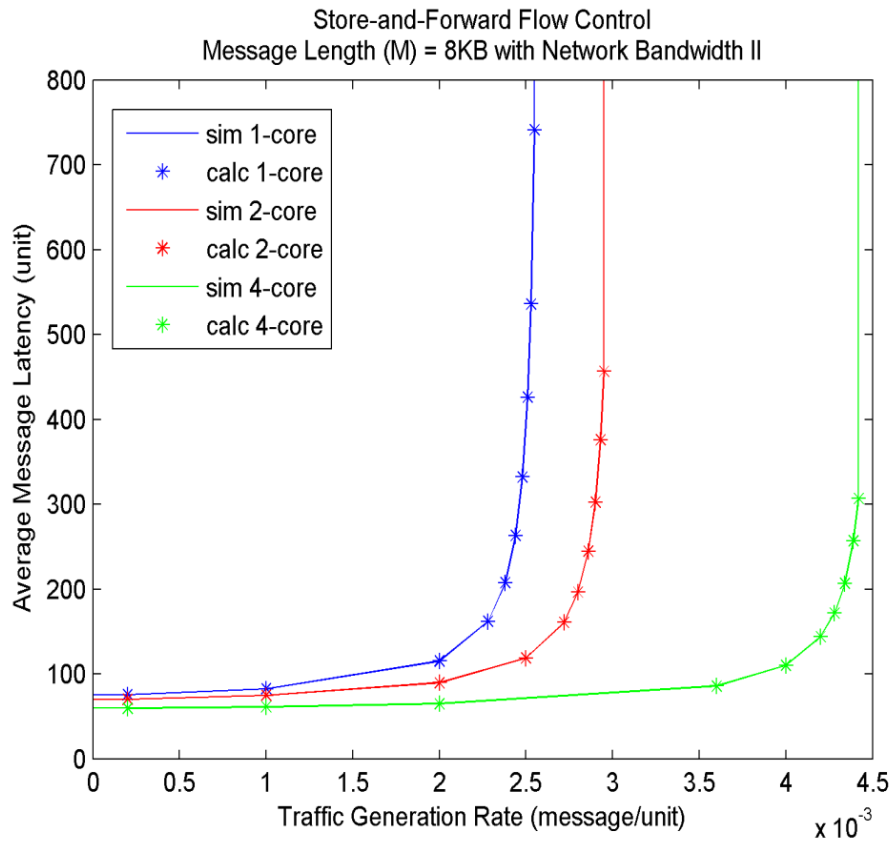


Figure 4-4: Message latency and throughput results based on store-and-forward flow control mechanism with M=8 KB using Network Bandwidth II

a) Discussion

Figure 4-3 and Figure 4-4 show the latency performance results of the interconnection network for single-core clusters and multi-core clusters. The results demonstrate that, with more than 1 core, the architecture can achieve lower communication latency. The probability of packet transmission in the internal cluster increased 51-76% with 2 and 4 cores in each processor compared with the single-core processor. This result shows that more packets can be transmitted at the same traffic rate, which will save waiting in a queue.

At the same time, a multi-core cluster is able to extend network throughput. The throughput of the network tends to increase as the number of cores is increased. With the dual-core processor, the experiments showed throughput extending over single-core by a significant 50%. With the quad-core processor, it extends the throughput by 75%. The improvement in throughput is due to more message transmission in light traffic. Even with a different network bandwidth, low latency and high throughput are achieved.

Both simulation experiments have been validated by analytical model. The consistency of the mathematical calculations with the simulation results shows that the simulation is a practical and cost-effective tool to measure the performance of interconnection networks in multi-core cluster architecture.

4.6.2 The impact on cluster size

These simulation experiments were designed to get more insight into the impact of the communication latency on the cluster size. The results reflected in Figure 4-5 based on the interconnection network parameters in Table 4-2, were compared with a higher network latency and a smaller bandwidth setup in the simulation experiments for Figure 4-6 and Table 4-4.

Table 4-4: Interconnection Networks Parameter II

Parameter	Internal-cluster	External-cluster
Network latency	0.02s	0.01s
Switch latency	0.01s	0.05s
Network Bandwidth	800 b/s	600 b/s

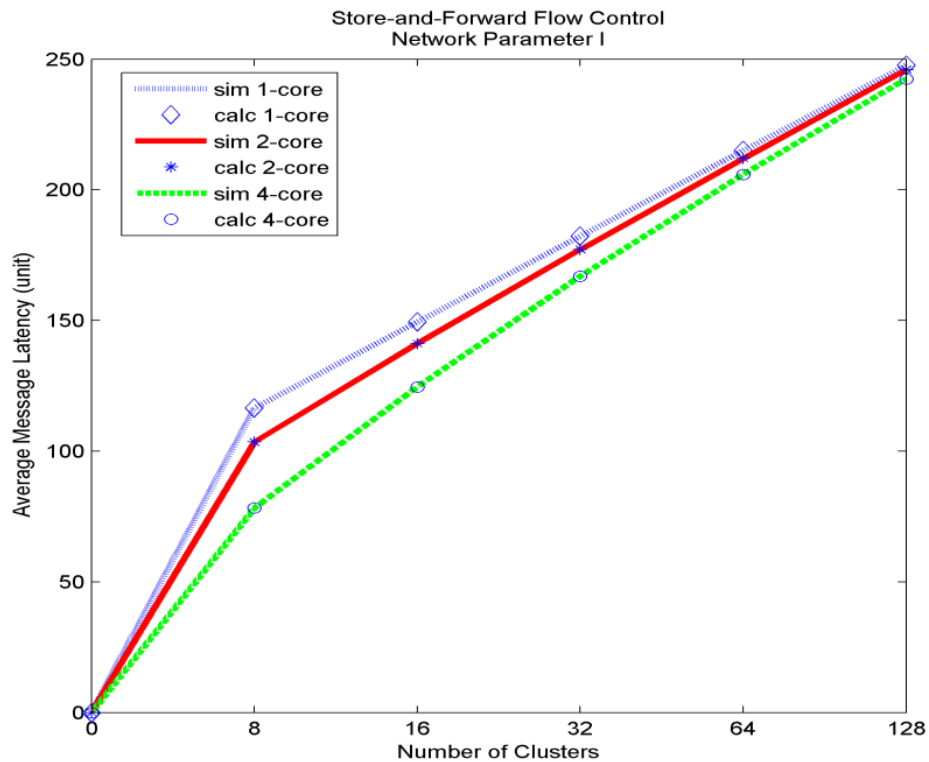


Figure 4-5: Simulation results of the impact on the message latency with various number of clusters based on Network Parameter I

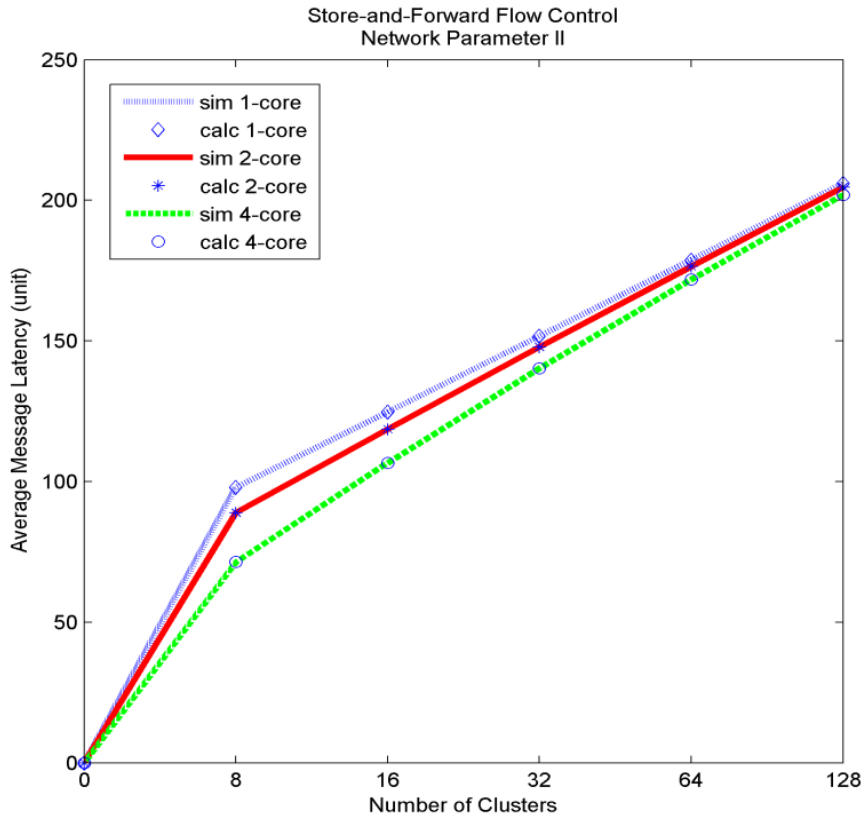


Figure 4-6: Simulation results of the impact on the message latency with numbers of clusters based on Network Parameter II

a) Discussion

Figure 4-5 and Figure 4-6 depict the simulation results and analytical analysis with 1, 2 and 4 cores in various cluster sizes. An experiment with a cluster size of five (8, 16, 32, 64, 128) with 3 cores was evaluated. The same message generation rate, 0.002 s, was used to maintain the uniformity of the results. The X-axis denotes the cluster size, and the Y axis the average message latency. The smallest size used in this experiment was an 8-cluster and the largest size was a 128-cluster.

The average message latency increased as the cluster size increased and all cluster sizes experienced almost the same latency rate. The saturation of the throughput also increased as number of clusters increased. The results also indicate that, even with a larger cluster size (16, 32, 64 and 128), MCMCA can save more transmission time and can finish the same tasks at a lower traffic rate.

These experiments are important, as they reveal that the MCMCA can be used with various cluster sizes, including the traditional single-core cluster. With MCMCA, the capacity of the resources increases, so that more packets can be transmitted while experiencing lower latency.

4.6.3 The impact on message length and scalability

In this experiment, to examine the potential scalability in the cluster architecture, different message lengths were run, as reflected in Table 4-5 and Table 4-6.

Table 4-5: Simulation Input II

Items	Quantity
No. of cluster (C)	8, 16, 32, 64, 128
No. of cores (nc)	1, 2, 4
Message generation rate (λg)	0.002s
Message Length (M)	8K
No. of m -port n -tree	4, 2

Table 4-6: Simulation Input III

Items	Quantity
No. of cores (nc)	1, 2, 4
Message generation rate (λg)	0.001s
Message Length (M /bytes)	128, 256, 512, 1K, 2K, 4K, 8K, 16K
No. of cluster, m -port n -tree	8, 8, 2

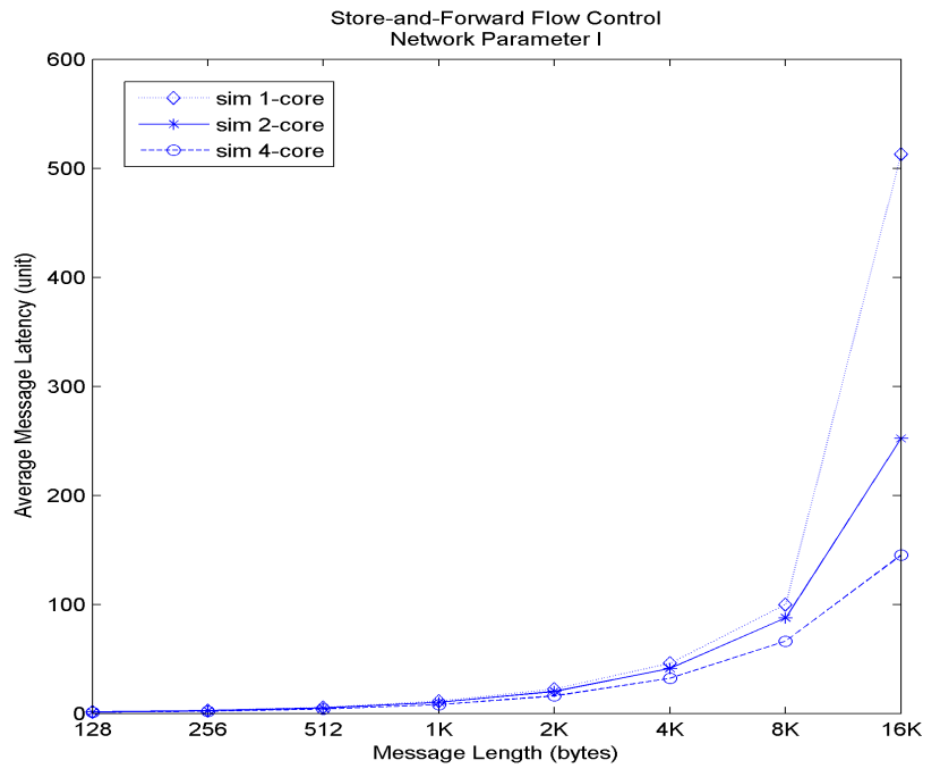


Figure 4-7: Simulation results of the impact on the message latency with various message lengths based on Network Parameter I

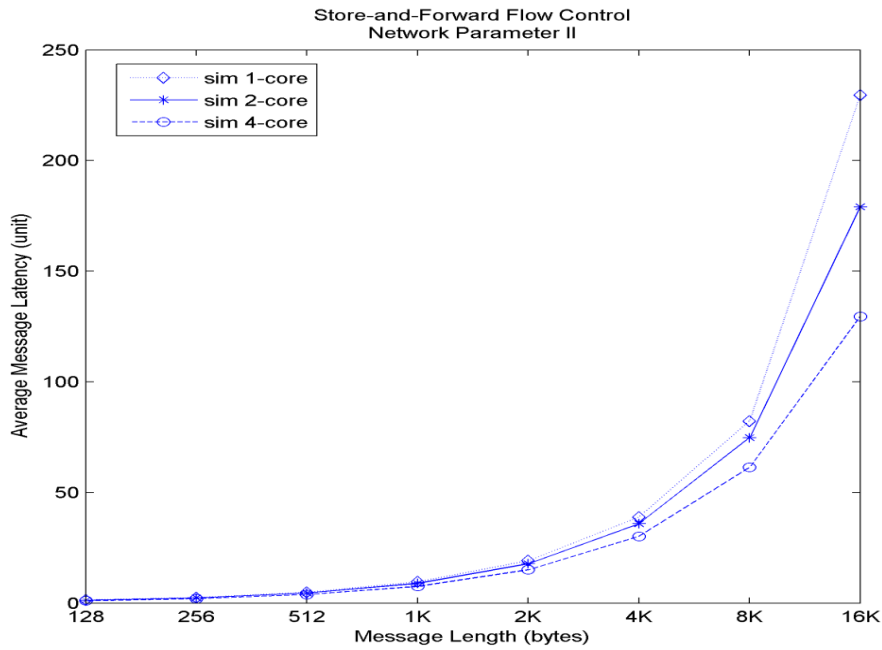


Figure 4-8: Simulation results of the impact on the message latency with various message lengths based on Network Parameter II

a) Discussion

Figure 4-7 and Figure 4-8 show the average message latency based on various message lengths. The results reflected in Figure 4-7 were for an experiment performed for an 8-cluster network parameter shown in Table 4-2, while the results in Figure 4-8 were obtained with the same cluster size but at a higher bandwidth value in network parameter II shown in Table 4-4. The X-axis represents the message length while the Y-axis denotes average message latency. The experimental message lengths ranged from 128 B to 16 KB.

With the same message generation rate, 0.001 s, both figures demonstrate that the traffic started to saturate the network at a message size of 8 KB. A larger throughput was obtained for 2-core and 4-core processors by 42-51% for network parameter I and 21-27% for network parameter II, compared to the single-core processor. Even when the message lengths were simulated using different bandwidths, the latency increased as the message lengths increased. With smaller message lengths (128 B, 256 B, 512 B) the message latency increments for all cores were very small (7-20%) and the latency rate was similar. The significant differences started to occur at a message length of 1 KB and became obvious at message lengths larger than 1 KB. This indicates that the architecture is scalable with different sizes of message. Based on this experiment, the figures also reveal that the architecture can scale well under various configurations.

4.7 Summary

A performance model is an essential tool to predict the communication latency behaviour of a cluster system. It is used to analyse the details of the cluster with various design optimization issues. This chapter has described a comprehensive performance evaluation and analysis of Multi-core Multi-cluster Architecture (MCMCA) using several configurations based on a store-and-forward flow control mechanism to answer research question RQ2. A novel simulation model of MCMCA was developed to investigate the interconnection network performance and various simulation results are compared. For the store-and-forward flow control mechanism, the simulation experiments with MCMCA affirm its ability to improve latency and throughput beyond that of a traditional cluster.

Several interesting observations from the experimental results give insights into both application and communication software developers by answering sub-research question SQ4. The latency results suggest that multi-core processors can improve network performance by 51%-76% compared to single-core processors. This indicates that optimizing all levels of interconnection network is important in this architecture. As the evaluation is based on store-and-forward flow control, the probability of blocking is zero, which contributes to higher saturation throughput.

Other simulation experiments were conducted with various sizes of cluster. The architecture can scale well with small to larger sizes of cluster while achieving lower latency and higher throughput. The impact of the architecture on message length was also tested. The results have reveals that small latency happens with smaller messages size but the latency increase with the larger message length. With the same message generation rate, 0.001s, both figures demonstrated that the traffic started to saturate the network at 8K size of messages, with 2-core and 4-core processors having a larger throughput compared to single-core processor. The experiments also demonstrated that MCMCA can scale well compared to traditional single-core cluster.

The comparison between the analytical model results and those produced by the simulation experiments has shown that the derived analytical model possesses a good basis for predicting the communication delay of interconnection network performance in the Multi-Core Multi Cluster Architecture (MCMCA).

The next chapter will focus on extending the architecture by applying the blocking mechanism which uses a wormhole flow control mechanism to solve the contention issue in the interconnection network.

Chapter 5 The Performance Model using Wormhole Flow Control Mechanism

5.1 Introduction

Performance models in cluster architecture based on single-cluster based on wormhole flow control mechanisms have been widely reported (Alzeidi et al., 2008; Geyong et al., 2009; Bahman Javadi et al., 2008b; Sarbazi-Azad et al., 2001; Yulei et al., 2012). Various issues have been described and solutions have been suggested but none of the research captures the interconnection network performance issue of multi-core multi-cluster architecture (MCMCA).

The contribution of this chapter is to investigate the interconnection network performance of MCMCA in harnessing the power of multi-core clusters to addresses research question RQ2 and sub-research question SQ4. The simulation experiment is based on wormhole flow control by involving different numbers of cores. The validity of the simulation model is demonstrated by comparing the results of the analytical model to those obtained by simulation experiments. With a large message and large cluster involved, the simulation experiments were performed by using the Iridis Compute Cluster (Admin, 2013), of the University of Southampton's High Performance Computing (HPC) facility.

The baseline simulation results demonstrated that our simulation model results, with analysis, show less than 1% difference in light traffic compared to the previous model, which showed a 3%-8% difference. The similarity of the results confirms that the MCMCA simulation model is a good basis for measuring the communication latency for a large cluster. Compared to store-and-forward flow control mechanism, the simulation experiments confirmed its ability to improve latency, but it is not able to match throughput performance due to blocking problems. The simulation experiments were extended to evaluate the impact of the interconnection network performance on scalability and cluster size.

This chapter investigates the modelling and simulation of interconnection network performance based on wormhole flow control mechanism. Section 5.4 presents the development of simulation models for a wormhole flow control mechanism by modifying the store-and-forward simulation model to allow the blocking mechanism. The blocking mechanism is represented by the Arbiter module. Section 5.5 outlines the analytical calculation to validate the simulation results and section 5.6 presents the simulation performance results.

5.2 Wormhole Flow Control Mechanism

‘Wormhole flow control’ has increased in popularity in cluster systems due to its low buffering (Alzeidi et al., 2008). Wormhole forwards a packet as soon as the header is received, and channel and buffers allocated to flits are acquired without waiting for the entire packet to be received. It works by dividing packets into a sequence of fixed-size units called ‘flits’, with channel and buffers allocated to flits. When a flit cannot acquire a buffer, blocking may occur. Wormhole flow control makes far more efficient use of buffer space, although it will increase some throughput (Dally & Towles, 2004).

5.3 Assumptions and Notations

Assumptions and notations used for the analysis are the same as shown in Chapter 4, in Table 4-1, with the additional notation reflected in Table 5-1 for wormhole flow control analysis.

Table 5-1: Notations used in MCMCA for Wormhole Flow Control

<i>Abbr.</i>	<i>Description</i>
φ_i	Channel arrival rate in ICN (s)
φ_e	Channel arrival rate in ECN (s)
φ_{ic}	Channel arrival rate in MCN (s)
ξ	Blocking probability
θ	Required coefficient to calculate the channel rate of the network
$W(s,j)$	Average time for a message to wait for a channel in internal cluster
$Ws(j,h)$	Average time for a message to wait for a channel in external cluster

5.4 The Simulation Model

5.4.1 Simulation Structure

Simulation models of MCMCA have been developed using OMNeT++, as described in section 3.6.1. The model is built at run time to form a topology that represents the geometric structure and the communication links between the modules, as presented in section 2.9.1. The simulation can behave with different inputs and parameters, such as the number of cores per node, number of clusters, number of messages to be generated, message length (M) and inter-arrival time.

The simulation modules represent hardware or a software entity that is capable of receiving messages from itself or from other modules. The modules are declared by specifying their attributes and ports by which a message to arrives and leaves. Generally, the MCMCA simulation model based on wormhole flow control mechanism was built from seven modules. These modules were programmed in C++ and assembled into larger components and models using a high-level language, the network description (NED). The same modules as described in section 4.4.1 were used to build the simulation model based on a wormhole flow control mechanism with Arbiter as a new module for this.

1. Network Topology module – this develops the building blocks of the fat-tree topology, including cores, nodes and clusters.
2. Network Interface module – this contains the interface of module types in fat-tree topology. Cores, nodes, clusters, switch, channel and the interconnection network are declared in this module, and connections between them are established.
3. Communication Switch module – this acts as the connection for each switch and router in the model, and it will determine how a message is transmitted along a path that has been selected by the routing algorithm.
4. Routing module – this determines the path, and schedules the routing algorithms for the messages/packets in all communication networks based on FIFO (First-In-First-Out); also it represents a single server queue that has the same service rate for each message/packet.
5. Generator module – messages/packets are generated by this module following the assumption that the message destinations are uniformly distributed.
6. Sink module – this will destroy the packets after each generation is completed, and will gather event information for statistics.
7. Arbiter module - this contains a ‘round robin’ loop for examining waiting messages. This module implements blocking functions in wormhole flow control.

As mentioned in an earlier section, wormhole forwards a packet as soon as the header is received, and the channel and buffers allocated to flits are acquired without waiting for the entire packet to be received. When a flit cannot acquire a buffer, blocking may occur. To simulate the blocking mechanism, the ‘Arbiter’ module was designed; whereby the flits were blocked from entering the network when the requested port was busy.

The traffic is configured by setting the destination and packet-arrival time parameters for each source. The destination is randomly distributed using the built-in OMNeT++ random number generation function, as mentioned in section 2.9.2e). The packet-arrival times are exponential, distributed according to a uniform traffic pattern.

The simulation also provides a set of statistical measurements collected by the sink module, generator module and routing module. The sink module collects latency and throughput at the packet and flit levels. The generator module collects from the source waiting time in order to identify the saturation point of the architecture. The routing module collects acquisition latencies from network transmission time, the time for a message to cross a network.

5.4.2 The Experimental Setup

The same simulation setup is used in this experiment as is presented in section 4.4.3. The experiment was designed to run based on the number of cores as the main input, with various combinations of message length, cluster size and traffic inputs. In each simulation experiment, the first 10,000 messages were discarded as belonging to the warm-up phase to make sure that the simulation reached a steady state. The total number of 100,000 messages were gathered to compute the message latency. These messages were split into 10 batches with the size of every batch being 10,000 messages. Message latency is measured from when the first flit of the packet is created to when its last flit is ejected at the destination node. The latency includes source queuing waiting time and network transmission time. The flit is destroyed after each generation is completed, to gather event information for statistics. Since each sample in the 'batch means' method is averaged over many of the original samples, the variance between batch means is greatly reduced. This decreases the standard deviation of the performance metrics and leads to greater confidence in the performance results.

5.5 The Analytical Model

5.5.1 Preliminaries

This section explains the implementation of the analytical model for computing the communication latency of the interconnection networks in MCMCA based on the wormhole flow control mechanism. The first phase is to compute ρ , nc and nt where ρ represents the number of processors in each cluster and nc the numbers of cores on each processor. Likewise, nt is the number of trees in the MCN while C is the number of clusters and m is the number of ports.

$$\rho = 2nc \left(\frac{m}{2} \right)^2 \quad [5.1]$$

$$nt = \frac{(\log_2 C) - 1}{(\log_2 m) - 1} \quad [5.2]$$

The messages enter the internal cluster and external cluster based on the probability P . The probability of outgoing requests P_o represents the messages generated by the source nodes that are sent to the external cluster while messages injected from a source node with the probability $(1 - P)$ enter an internal-cluster network.

$$P_o = \frac{N - \rho}{N - 1} \quad [5.3]$$

Here, α is the time taken by a packet of the message to transmit from a node to a switch connection (or *vice versa*), while β is the time taken by a packet of the message to transmit on a switch-to-switch connection; M is the message length, α_{net} and α_{sw} are the network and switch latencies, while β_{net} is the transmission time of one byte.

$$\alpha_{ie} = 0.5 \alpha_{net} + M \frac{1}{\beta_{net}} \quad [5.4]$$

$$\beta_{ie} = \alpha_{sw} + M \frac{1}{\beta_{net}} \quad [5.5]$$

5.5.2 Average Message Latency of an Internal-Cluster Network

The communication latency in an inter-cluster network includes messages travelling in an intra-chip network (AC), an inter-chip network (EC) and an intra-cluster network (ACN). In an inter-cluster network, the total arrival rate is λ_i (Shahhoseini et al., 2000) and each message travels an average distance to cross the network, with every channel receiving messages at a rate of:

$$\varphi_i = \theta \lambda_i \quad [5.6]$$

$$\lambda_i = \left(\frac{1}{\lambda}\right) (1 - P) \quad [5.7]$$

The required coefficient (Bahman Javadi et al., 2008a) to calculate the channel rate of the network is:

$$\theta = \frac{(nm - 2n - 1) \left(\frac{m}{2}\right)^n + 1}{4n \left(\frac{m}{2} - 1\right) \left(\left(\frac{m}{2}\right)^n - \frac{1}{2}\right)} \quad [5.8]$$

Each message may use a different number of channel links to reach its destination. Since this architecture applies to a multi-core processor, the total transmission time will be based on the numbers of cores on the processors. Therefore, the average transmission time in internal-cluster and external-cluster networks can be considered

as using a $2j$ -channel with a j -channel in the source node and a j -channel in the destination node, so that:

$$T_i = \sum_{j=1}^n (P_{(j,n)} T_{(j)} nc) \quad [5.9]$$

In an m -port n -tree, the probability of a message travelling $2j$ -channels to reach its destination is $P_{(j,n)}$. The probability of a message journey to reach its destination (Xuan-Yi et al., 2004) can be computed by:

$$P_{(j,n)} = \begin{cases} \frac{\left(\frac{m}{2} - 1\right) \left(\frac{m}{2}\right)^{j-1}}{2 \left(\frac{m}{2}\right)^n - 1} & 1 \leq j < n \\ \frac{(m-1) \left(\frac{m}{2}\right)^{j-1}}{2 \left(\frac{m}{2}\right)^n - 1} & j = n \end{cases} \quad [5.10]$$

a) Average Transmission Time for a Message to Cross the Networks

The received message rate in each channel can be defined by dividing the total channel rates by the number of channels in the internal cluster (Dally & Towles, 2004). The average amount of time for a message to wait for a channel with the blocking probability is $\xi = \varphi_i T_{s,j}$. Since each message travels on average channels to cross the network, every channel receives messages at a rate φ_i , as in equation 5.6. The value of blocking probability $T_{s,j}$ is derived from a Markov Chain (Sommereder, 2011).

$$\xi = \varphi_i T_{(s,j)} \quad [5.11]$$

The destination stage $K - 1$ is always able to receive a packet and the transmission time in the internal cluster may increase since the channel would be idle when the following stage is busy. The average amount of time that a packet waits to acquire a channel in the internal cluster with blocking probability, $W_{(s,j)_i}$ is given by:

$$W_{(s,j)_i} = \frac{1}{2} \varphi_i T_{s,j}^2, \quad 0 \leq s < K - 1 \quad [5.12]$$

Since the transmission time of a message at stage s is equal to the message transfer time and waiting time at subsequent stages in a channel (Bahman Javadi et al., 2008b), with $T_{(0,j)} = T_{(j)}$, it can be classified by:

$$T_{(s,j)} = \begin{cases} \sum_{l=s+1}^{K-1} W_{(l,j)} + M\beta_i & 0 \leq s \leq K-2 \\ M\alpha_i & s = K-1 \end{cases} \quad [5.13]$$

b) Average Waiting Time at the Source Node

Due to the probability of blocking happening in the network, the distribution of message latency becomes general. Therefore, a channel at source node is following M/G/1 queueing regime (Sommereder, 2011). The waiting time of a message W_i before entering the network with a λ_i message arrival rate can be calculated as:

$$W_i = \frac{\lambda_i(T_i)^2 \left(1 + \frac{T_i - (M\beta_i)^2}{(T_i)^2}\right)}{2(1 - \lambda_i T_i)} \quad [5.14]$$

c) Average Time for the Last Flit's Tail of the Message to Reach its Destination

The average time for the flit's tail to reach its destination in the internal-cluster R_i , can be found as (Sharifi et al., 2009):

$$R_i = \sum_{f=1}^{nc} \sum_{j=1}^n \left[P_{(f,nc)} P_{(j,n)} \left(\sum_{s=1}^{K_i-1} \beta_i + \alpha_i \right) \right] \quad [5.15]$$

Lastly, the equations for message latency in the internal-cluster communication networks can be expressed as:

$$\bar{\Sigma}L_i = W_i + T_i + R_i \quad [5.16]$$

5.5.3 Average Message Latency of an External-Cluster Network

Messages travelling in an external cluster communicate via two interconnection networks, the inter-cluster network (ECN) and the multi-cluster network (MCN) to get to their destinations in the other cluster.

a) Average Transmission Time for a Message to Cross the Networks

Similarly to internal-cluster network and based on equation 5.9, the average transmission time for an external cluster is:

$$T_e = \sum_{j=1}^n \sum_{h=1}^{nt} (P_{(j,n)} P_{(h,nt)} T_{(j,h)}) \quad [5.17]$$

$$P_{(j,n)} = P_{(h,nt)} = P_{(f,nc)} \quad [5.18]$$

The channel rate for external messages is:

$$\varphi_e = \theta \lambda_e \quad [5.19]$$

$$\varphi_{ic} = \theta \lambda_{ic} \quad [5.20]$$

Based on equation 5.12, the average time for an external message to wait for a channel at stage s with the channel message rate φ is be given by:

$$W_{s(j,h)} = \frac{1}{2} \varphi (T_{s(j,h)})^2 \quad [5.21]$$

$$\varphi = \begin{cases} \varphi_{ic} & j \leq s < j + 2h - 1 \\ \varphi_e & \text{other} \end{cases} \quad [5.22]$$

Similar to equation 5.13, the transmission time of a message at stage s in external-cluster, with $T_{0(j,h)} = T_{(j,h)}$, can be classified by:

$$T_{s(j,h)} = \begin{cases} \sum_{l=s+1}^{K-1} W_{l(j,h)} + M\beta_e & 0 \leq s \leq K-2 \\ M\alpha_e & s = K-1 \end{cases} \quad [5.23]$$

where $W_{s(j,h)}$ is the blocking time that a message has to wait for a channel at stage s in external-cluster networks and α_e is the time taken by a packet of the message to transmit from a node to a switch or switch-to-node as in equation 5.4.

b) Average Waiting Time at the Source Node

Messages generated by the source nodes are sent to the external cluster with the outgoing request probability P and λ_e as the message arrival rate following an M/G/1 queueing regime. Based on equation 5.14, the waiting time in the external-cluster network W_e can be computed by:

$$W_e = \frac{\lambda_e (T_e)^2 \left(1 + \frac{T_e - (M\beta_e)^2}{(T_e)^2} \right)}{2(1 - \lambda_e T_e)} \quad [5.24]$$

$$\lambda_e = 2 \left(\frac{1}{\lambda} \right) P \quad [5.25]$$

c) Average Waiting Time at Transfer Switches

External-cluster messages need to cross transfer switches during their journeys across the network. The transfer switches act as simple buffers to combine traffic from/to one cluster to/from other clusters. The waiting time at these buffers W_{sw} with λ_{ic} as the message arrival rate (Dally & Towles, 2004) can be computed as:

$$W_{sw} = \frac{\lambda_{ic}(M\beta_e)^2}{2(1 - \lambda_{ic}M\beta_e)} \quad [5.26]$$

$$\lambda_{ic} = NP \left(\frac{1}{\lambda} \right) P \quad [5.27]$$

d) Average Time for the Last Flit's Tail of the Message to Reach its Destination

The total average latency for the last flit's tail to reach its destination in the external cluster is based on equation 5.15 and can be computed by:

$$R_e = \sum_{j=1}^n \sum_{h=1}^{nt} \left[P_{(j,n)} P_{(h,nt)} \left(\sum_{s=1}^{SS_e-1} \beta_e + \alpha_e \right) \right] \quad [5.28]$$

The equation for message latency in the external-cluster communication networks can therefore be expressed as:

$$\bar{\Sigma}L_e = W_e + T_e + R_e + 2W_{sw} \quad [5.29]$$

5.5.4 Average Message Latency of MCMCA

From equations 5.16 and 5.29, the average message latency of communication networks in MCMCA can be obtained by the sum of the message latencies in internal-cluster and external-cluster networks as follows:

$$\bar{\Sigma}L = \bar{\Sigma}L_i (1 - P) + \bar{\Sigma}L_e (P) \quad [5.30]$$

5.5.5 Implementation of the Analytical Model

Algorithm 5-1 and Algorithm 5-2 presents the implementation of the analytical model to compute the communication latency of interconnection networks in MCMCA based on wormhole flow control.

Algorithm 5-1: Process flow in calculating the communication latency of interconnection networks in MCMCA based on a wormhole flow-control mechanism

Input Parameters:

Number of clusters C , parameter of m-port n-tree, message length M , number of cores nc , number of nodes N and λ .

1:	Calculate probability of entering/outgoing messages in the cluster P_o , number of processors in each cluster ρ , and number of trees in the MCN nt by using equations 5.3, 5.1 and 5.2.
2:	Calculate arrival rate in the internal cluster λ_i , external cluster λ_e and transfer switch λ_{ic} by using equations 5.7, 5.25 and 5.27.
3:	Calculate the arrival rate on a channel in internal cluster φ_i , external-cluster φ_e and transfer switch φ_{ic} by using equations 5.6, 5.19 and 5.20.
4:	Compute the required coefficient to calculate the channel rate, θ , with equation 5.8.
5:	Calculate the probability of message crossing 2 channel, $P_{(j,n)}$ and $P_{(f,nc)}$ by using equations 5.10 and 5.18.
6:	Calculate the time taken for a packet of the message to transmit on a node to a switch or <i>vice versa</i> α_{ie} , and the time taken for a packet of the message to transmit on a switch to switch β_{ie} for the internal and external cluster by equations 5.4 and 5.5.
7:	Calculate average latency in an internal cluster: <ol style="list-style-type: none"> Compute T_i, the transmission time in the network equation 5.9 Compute ξ, the blocking probability equation 5.11 Compute $W_{(s,j)_i}$, the amount of time for a message to wait for a channel h equation 5.12 Calculate W_i, the waiting times at the source node equation 5.14 Calculate R_i, the time for the last packet of the message reach its destination equation 5.15.
8:	Calculate average latency in the external cluster using the indicated equations: <ol style="list-style-type: none"> Compute T_e, equation 5.17 Compute ξ, equation 5.11 Compute $W_{s(j,h)}$, equation 5.21

	d. Calculate W_e , equation 5.24 e. Calculate W_{sw} , the waiting time at transfer switches equation 5.26 f. Calculate R_e , equation 5.28.
9:	Calculate the average message latency in internal-cluster networks and average message latency in external-cluster networks using equations 5.16 and 5.29.
10:	Calculate $\bar{\Sigma}L$, the average message latency of interconnection networks in MCMCA using equation 5.30.

Algorithm 5-2: Pseudocode of process flow in calculating the communication latency of interconnection networks in MCMCA using MATLAB based on wormhole flow control

```

clear all
clc

n_core=4; % number of core(s)
C = 8;
m = 8;
n = 2;
M = 32;
Lm = 256;

lambda=500;
lambdaG=1/lambda;
pp=(m/2)^n; %number of processor per node
NP=2*pp; %number of processors in each cluster
NP=NP*n_core;
N=NP/n_core*C;

PO=(N-NP)/(N-1);
PI=1-PO;

lambdaI1=PI*lambdaG;

davgICN=((n*m)-(2*n)-1)*(pp)+1/((m/2)-1)*(pp-0.5));
teta = ((n*m)-(2*n)-1)*pp+1/((4*n)*(m/2-1)*(pp-1/2));
eta = teta*lambdaI1;

anetI = 0.01;
aswI = 0.01;
BnetI = 1000;

tcnI=(0.5*anetI)+(Lm*1/BnetI);
tcsI=aswI+(Lm*1/BnetI);

```

```

for j=1:n-1;
    P(j,n) = ((m/2)-1)*((m/2)^(j-1))/((2*((m/2)^n))-1)
end
for j=n;
    P(j,n) = ((m-1)*((m/2)^(j-1)))/((2*((m/2)^n))-1)
end

for f=1:n_core
    if f==n_core
        Pnc(f,n_core) = ((m-1)*((m/2)^(f-1)))/((2*((m/2)^n_core))-1);
        break
    else
        Pnc(f,n_core) = (((m/2)-1)*((m/2)^(f-1)))/((2*((m/2)^n_core))-
1);
    end
end

Wsj=0;

for j=1:n
    SS=2*j-1;
    for s=SS:-1:1
        if s==SS
            D(s,j)=M*tcnI;
            W(s,j)=1/2*eta*(D(s,j).^2);
        else
            for l=SS:-1:s+1
                Wsj=Wsj+W(l,j);
            end
            D(s,j)=Wsj+(M*tcsI);
            if l-1>0
                W(l-1,j)=1/2*eta*(D(s,j)).^2;
            else
                break
            end
            Wsj=0;
        end
    end
end

DD(1)=M*tcnI;
DD(2:n)=D(1,2:n);
DI=0;

for j=1:n
    DI=DI+P(j,n)*DD(j)*n_core;
end

WI1=lambdaI1*(DI*DI)*(1+((DI-M*tcsI)^2)/(DI*DI))/(2*(1-
(lambdaI1*DI)));

```

```

f=1;

RI=((davgICN-2)*tcsI+tcnI);

TI1=WI1+DI+RI;

% ---- external cluster -----

dc=log(C/2)/log(m/2);
pc=(m/2)^dc;
h=1;

Nch=4*n*pp;
Nchnc=4*n*pc;

lambdaE1=2*PO*lambdaG;
lambdaI2=(NP/n_core*PO*lambdaG);

davgECN=davgICN;
davgICN2=((dc*m)-(2*dc)-1)*(pc)+1)/((m/2)-1)*(pc-0.5));

tetaI2 = ((dc*m-(2*dc)-1)*pc+1)/((4*dc)*(m/2-1)*(pc-1/2));

etaE1= teta*lambdaE1;
etaI2= tetaI2*lambdaI2;

anetE = 0.02;
aswE = 0.01;
BnetE = 500;

tcsE= aswE+(Lm*1/BnetE);
tcnE=(0.5*anetE)+(Lm*1/BnetE);

for j=1:n-1;
    P(j,n) = (((m/2)-1)*((m/2)^(j-1)))/((2*((m/2)^n))-1)
end
for j=n;
    P(j,n) = ((m-1)*((m/2)^(j-1)))/((2*((m/2)^n))-1)
end

for h=1:dc-1;
    P(h,dc) = (((m/2)-1)*((m/2)^(h-1)))/((2*((m/2)^dc))-1)
end
for h=dc;
    P(h,dc) = ((m-1)*((m/2)^(h-1)))/((2*((m/2)^dc))-1)
end

```



```

Wsjh=0;

for j=1:n
    for h=1:dc
        SE=2*(j+h)-1;
        SS=SE;
        for s=SE-1:-1:1
            if s==SE-1
                DDE(s,j,h)=M*tcnE;
            else
                for l=s+1:SS
                    if l<j+1
                        W(l)=1/2*etaI2*(DDE(l,j,h))^2;
                    else
                        W(l)=1/2*etaE1*(DDE(l,j,h))^2;
                    end
                    Wsjh=Wsjh+W(l);
                end
                DDE(s,j,h)=Wsjh+(M*tcSE);
            end
            SS=SS-1;
        end
    end
    if j==1
        WW1=1/2*etaI2*(DDE(s,j,h))^2;
    else
        WW1=1/2*etaE1*(DDE(s,j,h))^2;
    end
    DEE(j,h)=M*tcSE+WW1+Wsjh;
    Wsjh=0;
end

DE=0;
for j=1:n
    DE=DE+P(j,n)*DEE(j,h)*P(h,dc);
end

WE1=lambdaE1*(DE^2)*(1+((DE-M*tcnE)^2)/(DE*DE))/(2*(1-lambdaE1*DE));
RE=((davgECN+davgICN2)-2)*tcSE+tcnE;
WCD=lambdaI2*((M*tcSE)*(M*tcSE))/(2*(1-lambdaI2*M*tcSE));

TE1=WE1+DE+RE+(2*WCD);
TotalTI=PI*TI1;
TotalTE=PO*TE1;
TMC=(PI*TI1)+(PO*TE1);

```

5.6 Performance Evaluation (SQ4)

This section presents the results and discussion for the MCMCA simulation model based on worm-hole flow control. The simulation experiment was to test the sub-research question SQ4. The simulation experiments and analytical model calculations were performed, and the results for three numbers of cores were compared.

The simulation experiments were run using the interconnection network parameters as given in Table 5-2 with input parameters in

Table 5-3.

Table 5-2: Interconnection Networks Parameters

Parameter	Internal-cluster	External-cluster
Network latency	0.01s	0.02s
Switch latency	0.01s	0.01s
Network Bandwidth	1000 b/s	500 b/s

Table 5-3: Input Parameters

Items	Flit length (<i>F</i>) 256bytes	Flit length (<i>F</i>) 512bytes
No. Of cores (<i>nc</i>)	1, 2, 4	
No. Of cluster (<i>C</i>)	8	
No. Of <i>m</i> -port <i>n</i> -tree	8, 2	
Message Length (<i>M</i>)	32 flits	

5.6.1 Baseline experiment on MCMCA

The results of the baseline experiment are given for single-core cluster architecture, as depicted in Figure 5-1. Simulation and analysis revealed that, when compared, the results obtained from MCMCA with 1-core closely match the results from the model of multi-cluster architecture presented by Javadi et al. (2006) as given in Table 5-4.

The author claimed that his analytical model and simulation model in average differed by less than 3-8% under conditions of light traffic. The MCMCA simulation model outperformed the earlier model, in that the difference between the simulation model and analytical model was less than 1%. The similarity of the results confirms that the simulation model is a good basis for measuring the communication latency for a large-scale cluster, and can be extended to multi-core processor.

Table 5-4: Baseline results comparison between MCMCA with 1-core with multi-cluster model presented by Javadi et al. (2006)

Model	Simulation Model	Analytical Model	Different
Average Message Latency with F=256	127.33831	127.35218	0.01387
Average Message Latency with F=512	218.08425	218.11161	0.02736

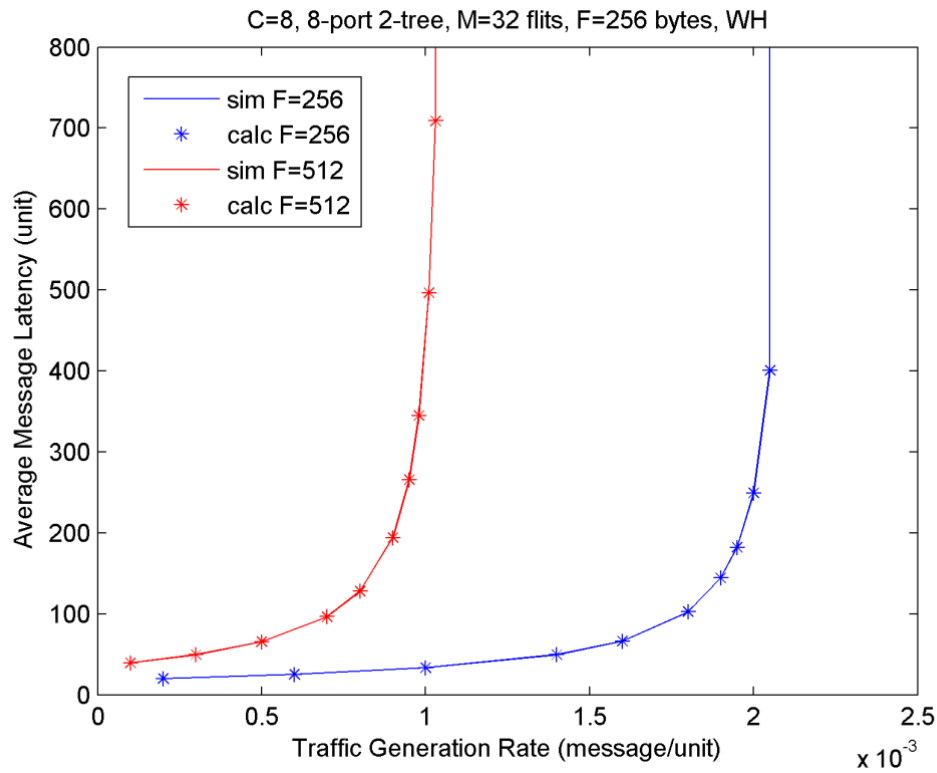


Figure 5-1: MCMCA model with C=8, M=32 flits, F=256 bytes based on Wormhole Flow Control

5.6.2 Latency and throughput on MCMCA

Three different numbers of cores in a processor were analysed. Figure 5-2 depicts the simulation and analytical model results with a flit size equal to 256 bytes, while Figure 5-4 depicts the results with a flit size equal to 512 bytes. The simulation parameters are based on Table 5-2 and

Table 5-3.

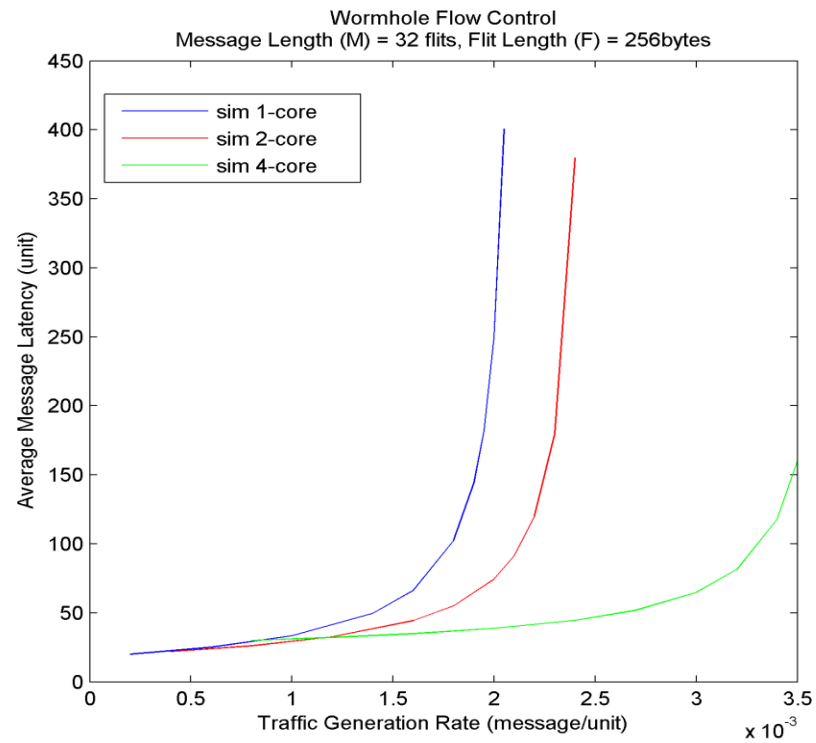


Figure 5-2: Message latency and throughput simulation results based on Wormhole Flow Control Mechanism with M=32F, F=256 bytes

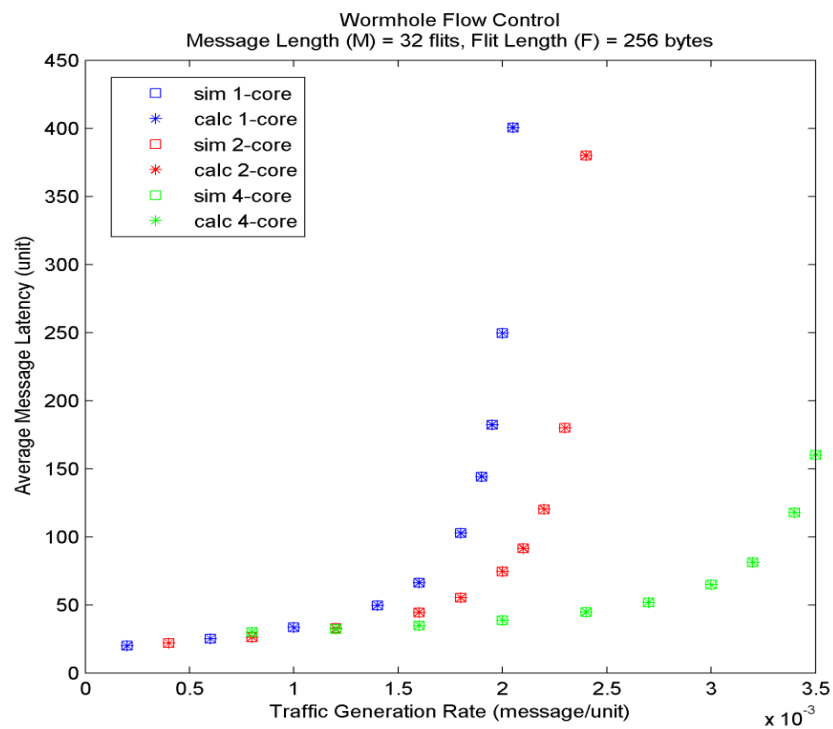


Figure 5-3: Message latency and throughput simulation results compared to analytical calculation results based on Wormhole Flow Control with M=32 flits, F=256 bytes

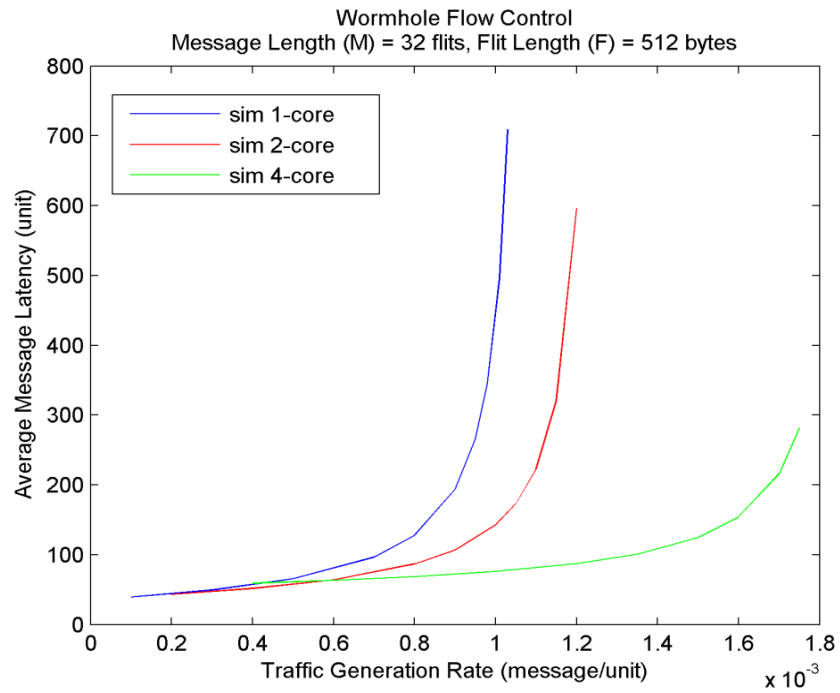


Figure 5-4: Message latency and throughput simulation results based on Wormhole Flow Control Mechanism with M=32 flits, F=512 bytes

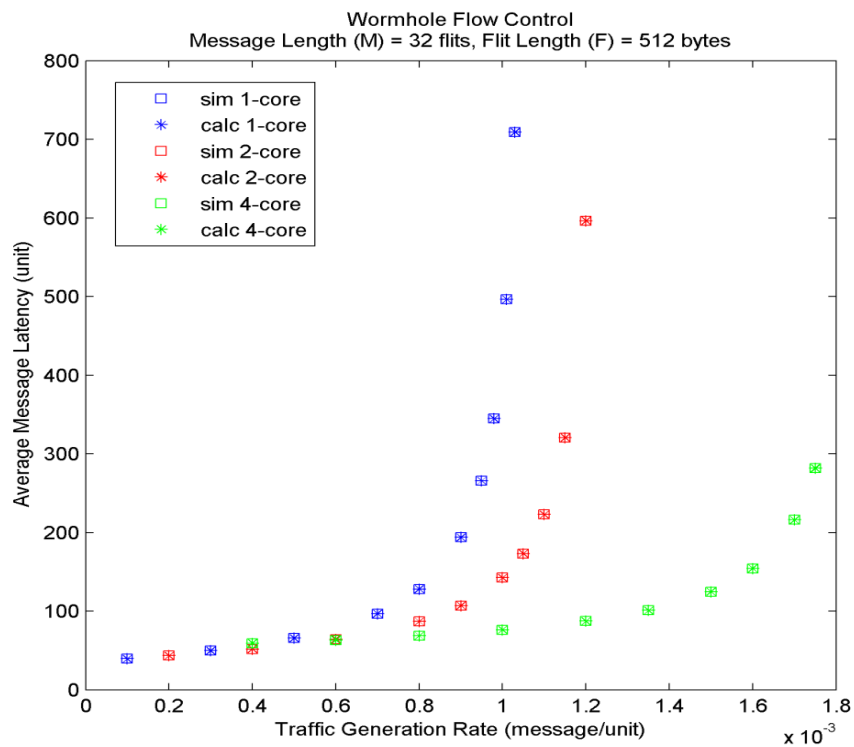


Figure 5-5: Message latency and throughput simulation results compared to analytical calculation results based on Wormhole Flow Control with M=32 flits, F=512 bytes

a) Discussion

The results showed that, as the traffic rate increases, the average communication latency increases as the messages have to wait for resources before travelling into a network. Results based on 4 cores show that there higher latency occurs at lower traffic volumes, since more messages need to be served in the internal cluster. The results also demonstrated that the architecture, with different numbers of cores, can achieve lower communication latency of the interconnection networks at the same traffic rate. However, as the traffic rate increased, higher latency occurred due to longer blocking times in the wormhole flow control mechanism. Packets are frequently blocked because the channel is held by another packet, even when there are buffers available.

The similarity between the analytical results and those produced from the simulation experiments as in Figure 5-3 for $M=8K$ and Figure 5-5 for $M=16K$, suggests that the derived simulation model possesses a good basis for predicting the communication delay of interconnection network performance of the Multi-Core Multi-Cluster Architecture (MCMCA).

5.6.3 The Impact on cluster size

The accuracy of the simulation model has been validated as being a cost-effective tool to investigate the interconnection network performance in MCMCA. It can thus be used to get an insight into the impact of cluster size on maximising network performance.

A set of simulation experiments were conducted using the simulation input in Table 5-5.

Table 5-5: Simulation Input for cluster sizes

Items	Quantity
No. of cluster (C)	8, 16, 32, 64, 128
No. of cores (nc)	1, 2, 4
Message generation rate (λg)	0.001s, 0.002s
Message Length (M)	18K, 16K
No. of m -port n -tree	4, 2

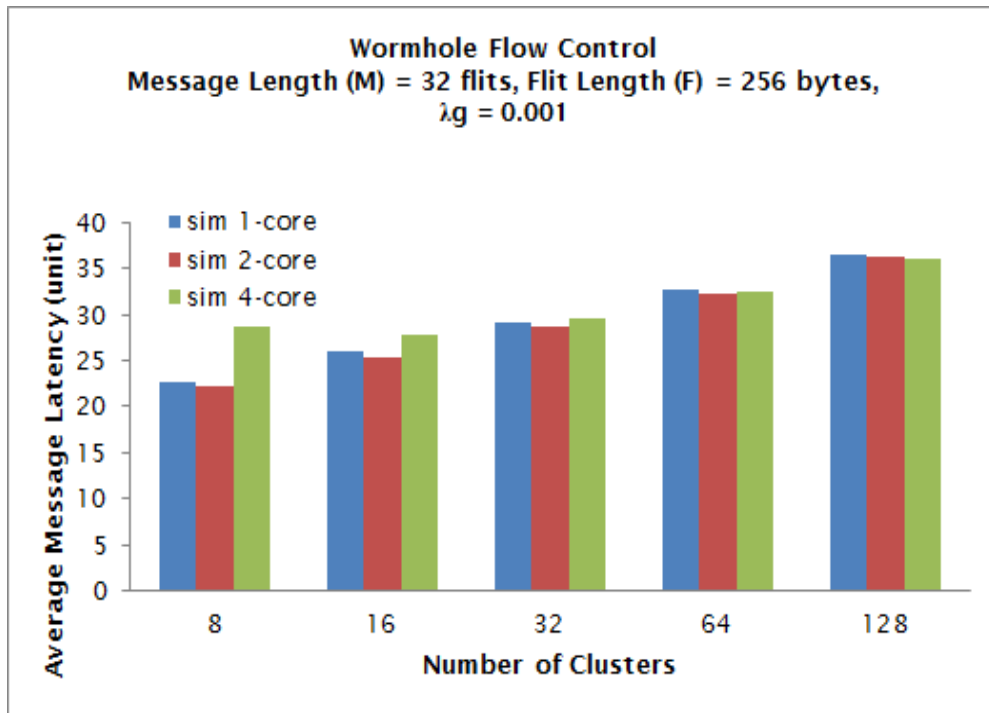


Figure 5-6: Message latency predicted by the simulation model with M=32 flits, F=256 bytes and $\lambda g=0.001$

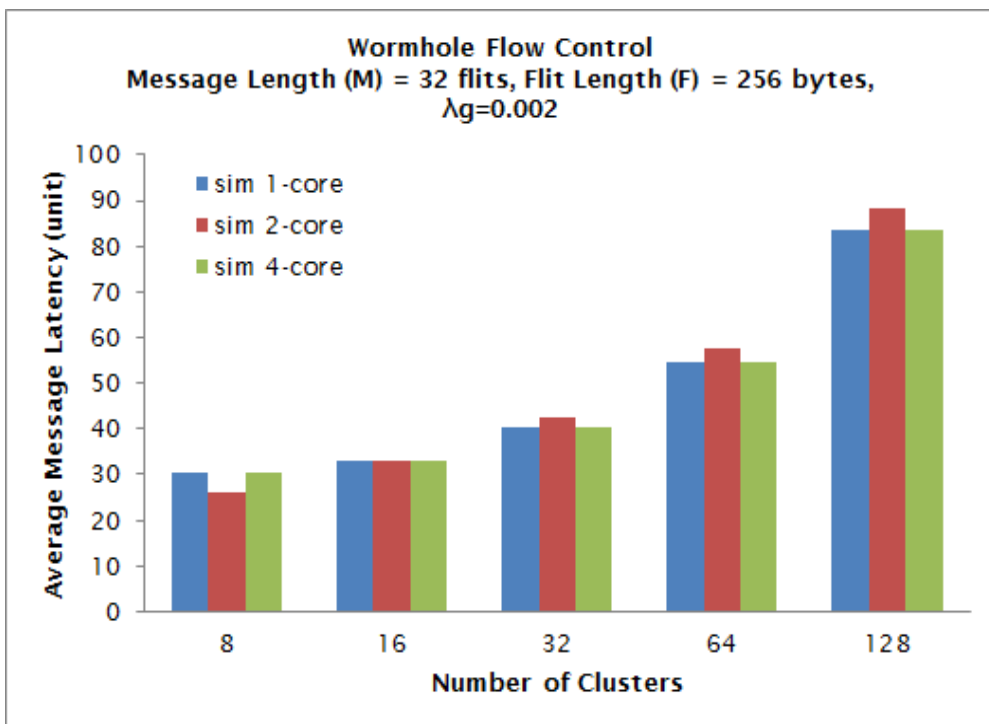


Figure 5-7: Message latency predicted by the simulation model with M=32 flits, F=256 bytes and $\lambda g=0.002$

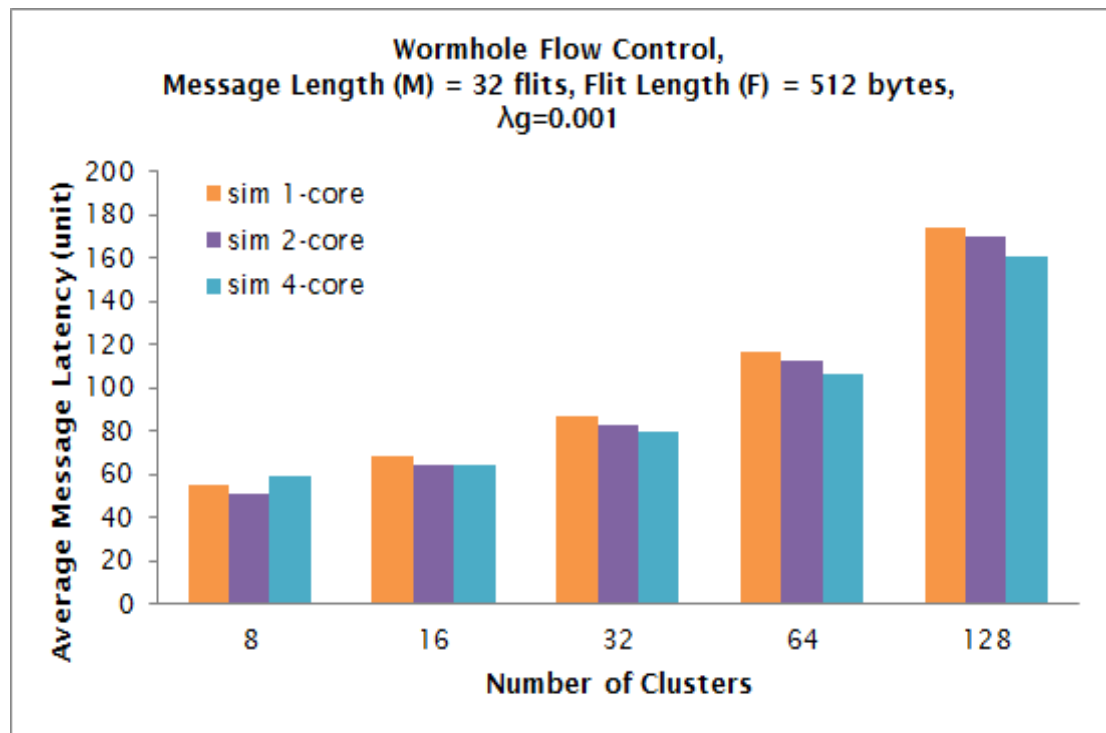


Figure 5-8: Message latency predicted by the simulation model with M=32 flits, F=512 bytes and $\lambda g=0.001$

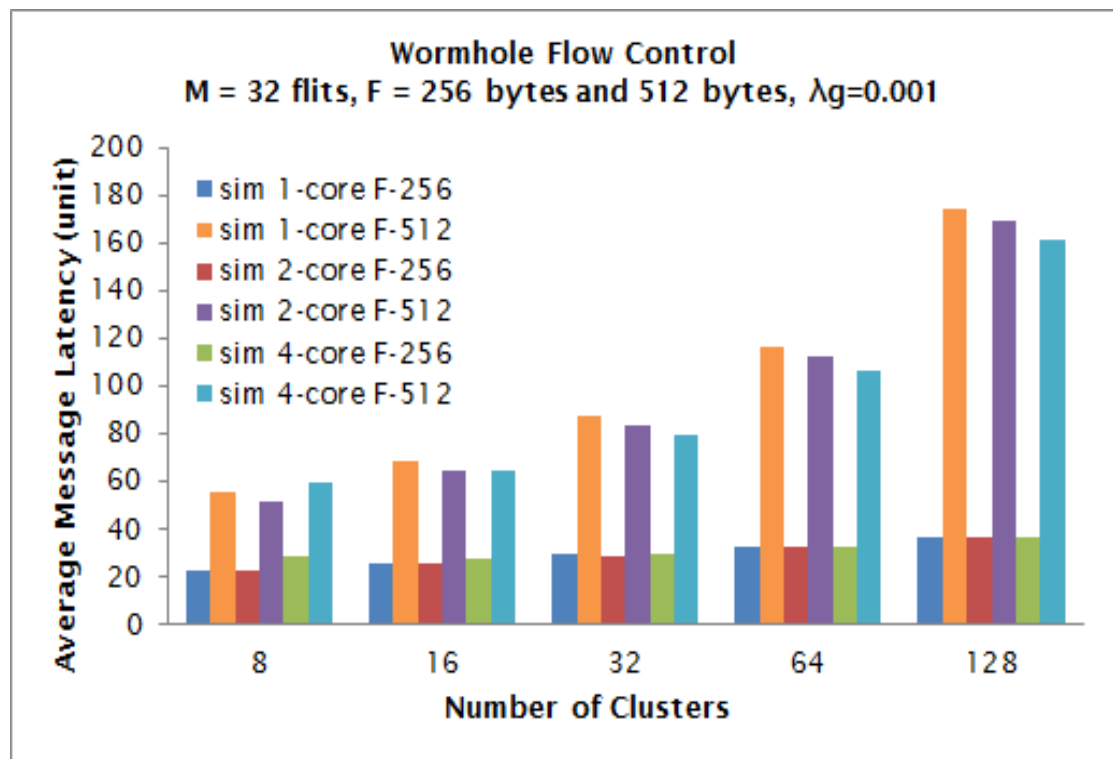


Figure 5-9: Comparison of message latency with M=32 flits, F= 256 and 512 bytes predicted by the simulation model using $\lambda g=0.001$

a) Discussion

An experiment was constructed with five sizes of cluster, $C = 8, 16, 32, 64, 128$, and three core sizes, $nc = 1, 2, 4$. The experiment was run with two message generation rates, $\lambda g = 0.001$ s and 0.002 s, and two message lengths, $M = 8$ KB and 16 KB. The probability (P) assumed was 0.89 for the internal cluster and 0.11 for the external cluster. From the resulting figures, we can deduce that, as the size of clusters increases, the network performance improves, since the message latency decreases, which also increases the maximum network throughput. However, it seems that the larger clusters, 64 and 128 , gave only small improvements in throughput. Even though the latency for an 8 -cluster with 4 -cores is a bit higher due to blocking, the latency is almost similar for larger clusters. Furthermore, the improvement caused by the trade-off between cluster size and the number of cores can relieve the negative effects of the blocking mechanism on network performance.

These results demonstrated that the architecture can be used to investigate the interconnection network performance of MCMCA for small to large cluster size. The performance results are also consistent with the results for the simulation model based on the store-and-forward flow control mechanism described in section 4.6.2. However, it is important to balance cluster size with the number of cores involved within the limited space to maximise network performance.

5.6.4 The impact on message length and scalability

Testing the various impacts on message length will give insights into scalability. For testing the model, seven message lengths (M/bytes) were involved in an eight multi-core cluster system, as in Table 5-6, with the numbers of cores being $1, 2$ and 4 . To make this experiment comparable with others, the same message length has also been tested in larger cluster sizes: 32 multi-core cluster systems with $4, 8$ and 16 cores. The same message generation rate (λg) was used to maintain the validity of the results.

Table 5-6: Simulation Input for scalability

Items	8-cluster	32-cluster
No. of cores (nc)	1, 2, 4	4, 8, 16
No. of m -port n -tree	8, 2	
Message generation rate (λg)	0.001s	
Message Length (M/bytes)	128, 256, 512, 1K, 2K, 4K, 8K	

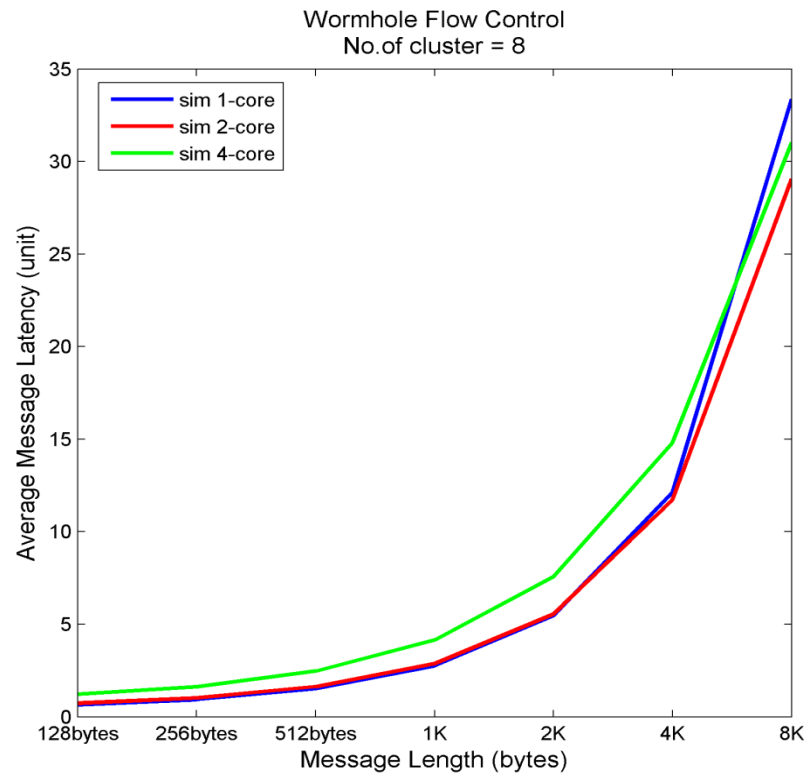


Figure 5-10: Simulation results of the impact on the message latency with various message lengths based on Number of cluster = 8

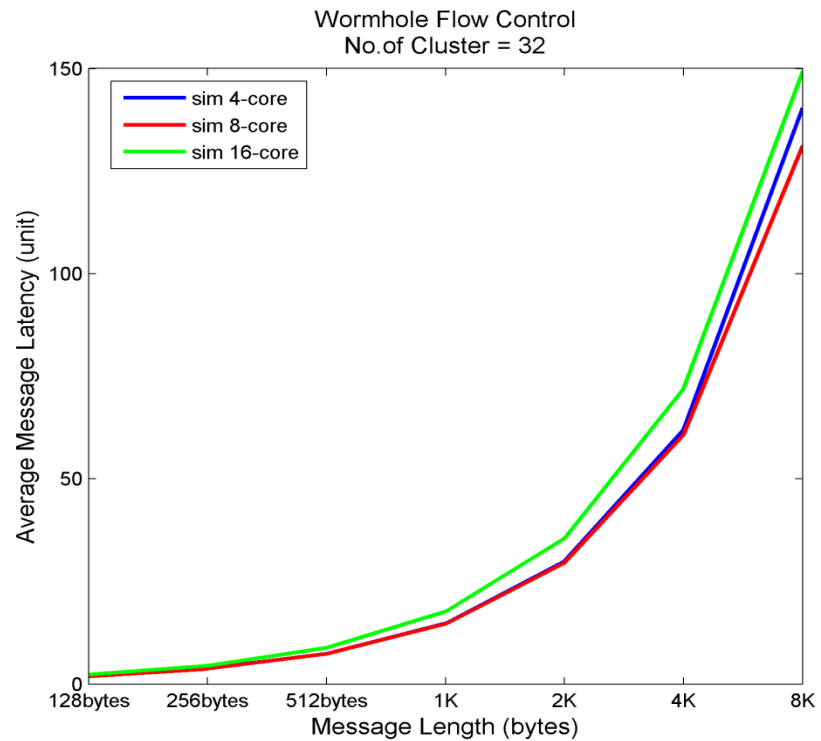


Figure 5-11: Simulation results of the impact on the message latency with various message lengths based on Number of cluster = 32

a) Discussion

For all message lengths, a simulation experiment with 2-core for an 8-cluster and 8-core for a 32-cluster outperforms the other core sizes especially for longer message lengths, as shown in Figure 5-10 and Figure 5-11. Higher latency occurs for larger numbers of cores in light traffic due to task allocation between cores and the blocking mechanism. Based on this analysis, the maximum network throughput for an 8-cluster with 1-core is higher due to the reduced message latency involved. Based on all message lengths, the larger number of cores, 2-core and 4-core, surpasses single-core performance by 4-10% under a uniform traffic pattern.

However, with longer message lengths, there is a small improvement in throughput. This is due to the fact that, when a packet is blocked, all the data flit continues to hold onto its buffer. Thus, the larger the message length, the more contention is found in channels and buffers.

This experiment suggested that longer message lengths have a significant impact on network performance, and the architecture is capable of scaling well with various message lengths.

5.7 Summary

Multi-core multi-cluster architectures (MCMCA) have a tremendous potential for the future of computing. However, there exist some issues that can affect the performance of the architecture and limit the gained advantages. This chapter presented a new simulation model to investigate the performance of interconnection networks in MCMCA, based on wormhole flow control mechanism (RQ2). Three sets of experiments were designed to gain an insight into the architecture's capability by answering sub-research question SQ4. The simulation results were validated by analytical model calculation which demonstrated a similar result.

The first simulation experiment was conducted to investigate MCMCA performance based on latency and network throughput. The results showed that a wormhole flow control mechanism is able to improve latency dramatically over a store-and-forward flow control mechanism. However, the store-and-forward flow control fares better in term of throughput, with a higher saturation point. While wormhole flow control performs better for light traffic, it does not achieve optimal throughput due to the probability of blocking while traversing the network. The simulation experiments also affirm the advantages of a wormhole flow control mechanism. Since packets are transferred using a smaller unit called 'flit', more flits can be transmitted in the same channel, thus minimising the latency.

Another simulation experiment was performed to get more insight into the architecture's impact on cluster size and message length, so as to reveal potential scalability. The use of a multi-core processor improves the latency even for larger cluster sizes and larger message lengths, with higher throughput performance. However, due to the blocking issue, the wormhole flow control mechanism was unable to achieve optimal throughput performance. Furthermore, the experiments indicated the importance of balancing the number of cores in each cluster with cluster size, so as to maximise the overall architecture performance.

The performance evaluation indicated that the architecture is able to measure the performance of multi-core clusters under various parameters and system configurations. The resulting performance evaluation can be an alternative to enable system designers to build cluster applications.

Chapter 6 Statistical Analysis

6.1 Introduction

The Statistical Package for the Social Sciences (SPSS) was used to analyse the experimental results. SPSS is a software for statistical analysis and it provides essential statistical analysis tools for every step of the analytical process (Field, 2013). The main aim was to determine whether multi-core multi-cluster simulation results were better than traditional cluster. The Shapiro-Wilk (S-W) test of normality was used to determine whether or not the experiment results follow a normal distribution. It is also suitable for a small to medium number of samples; in this case there were 10 samples for each experiment (Field, 2000).

The results of the simulation experiment focused on latency, as described in section 4.6.1 for store-and-forward flow control mechanism and in section 5.6.2 for a wormhole flow control mechanism. The S-W test depends on the normal distribution of experimental results. The first analysis was undertaken to compare the experimental results of the two flow control mechanisms. If the results were normally distributed, then the parametric t-test was used to compare the two flow control mechanisms, and if the results were not normally distributed, then the non-parametric Mann-Whitney test was applied to compare the results of the two flow control mechanisms.

The second analysis compared the experimental results obtained for the different number of cores. If the results followed a normal distribution, a one-way ANOVA test was conducted to determine whether there was a significant difference between the results. If the results did not follow a normal distribution, then the non-parametric Friedman's test was applied to compare the results for significant difference.

Section 6.2 describes the normality testing of the experimental results. The comparison of the two flow control mechanisms is presented in section 6.3, followed by the comparison of three core sizes in section 6.4. The results and their analysis are summarised in section 6.5.

6.2 Analysis of Experimental Results

The Shapiro-Wilk (S-W) tests were used to test the assumption that the experimental results are derived from a normally-distributed population. A typical use of the S-W tests is to check assumptions of normality required by other statistical tests to be used later for the experiment results. In other words, S-W tests the null hypothesis that "the results come from a normally-distributed population". The hypothesis is therefore that

the data come from a population that is not normally distributed. Consequently, if the results of the test are significant ($p < 0.05$), rejecting the null hypothesis means rejecting the assumption of normality for the population distribution. In this case, the population consisted of the results obtained in the simulation experiments. In this study, the data were derived from the results obtained from the simulation experiments based on Store-and-Forward flow control and Wormhole flow control mechanisms.

6.2.1 Normality test

a) Descriptive Statistic

Table 6-1 and Table 6-2 provide a summary of message latency results for store-and-forward flow control mechanism and wormhole flow control mechanism. The following tables show the summary of ten traffic rate including mean, median, mode, standard deviation, minimum and maximum latency per time unit.

Table 6-1: Descriptive Results for Store-and-Forward Flow Control

Store-and-Forward	1-Core (s)	2-Core (s)	4-Core (s)
N	10	10	10
Mean	249.168	109.864	69.699
Median	143.028	105.443	70.165
Mode	88.917	80.475	63.801
Std. Dev	209.816	23.920	3.747
Minimum	88.917	80.475	63.801
Maximum	698.637	143.300	73.895

Table 6-2: Descriptive Results for Wormhole Flow Control

Wormhole	1-Core (s)	2-Core (s)	4-Core (s)
N	10	10	10
Mean	127.338	102.639	65.652
Median	84.471	64.833	48.271
Mode	19.971	20.281	27.613
Std. Dev	122.047	109.018	42.932
Minimum	19.971	21.933	30.036
Maximum	400.628	379.939	160.106

In these simulation experiments, a total of 100,000 messages were used to gather statistics. In these experiments, 100,000 messages were divided into 10 batches, where the size of each batch was 10,000 messages. Results for each traffic rate were derived from simulation with N being the average of the simulation run where each N

represents 10,000 messages. The curves produced in each graph are from 10 traffic rates.

Normality test are sensitive to the size of the sample: with a large sample even small deviations from normality will be reported as significant.

Table 6-3: SPSS Output for S-W tests of normality using Store-and-Forward Flow Control Mechanism

	Shapiro-Wilk		
	Statistic	df	Sig.
core1	.789	10	.011
core2	.909	10	.271
core4	.910	10	.283

Table 6-4: SPSS Output for S-W tests of normality using Wormhole Flow Control Mechanism

	Shapiro-Wilk		
	Statistic	df	Sig.
core1	.848	10	.055
core2	.739	10	.003
core4	.815	10	.022

Table 6-3 and Table 6-4 show the SPSS output of the S-W tests for both flow control mechanism. For test output using Store-and-Forward flow control, the p -value for 1-core is smaller than 0.05, indicating a not-normally distributed data. Although the p -values for 2-core and 4-core are greater than 0.05 (i.e. normally distributed) parametric test (e.g. independent sample t test and one-way repeated measure ANOVA) cannot be used because all groups of comparison must be normally distributed. Therefore, the analysis will use non-parametric test (e.g. Mann Whitney and Friedman tests). SPSS output of the S-W test for Wormhole flow control indicate that 2-core and 4-core are not normally distributed. Therefore, the analysis based on Wormhole flow control mechanism will also use non-parametric test (e.g. Mann Whitney and Friedman tests).

b) Skewness and Histograms

Skewness is a measure of the asymmetry of a distribution. A perfectly normal distribution is symmetric and has a skewness value of 0. A distribution with a significant positive skewness has a long right tail and a significant negative skewness has a long left tail. As a guideline, skewness values with more than twice a standard error are taken to indicate a departure from symmetry.

This test was conducted on both the store-and-forward (SF) and wormhole (WH) flow control mechanisms.

Table 6-5: Descriptive Statistics for Skewness with Store-and-forward flow control

	N	Skewness	
Final test score	Statistic	Statistic	Std. Error
1-core	10	1.429	0.687
2-core	10	0.208	0.687
4-core	10	-0.397	0.687

Table 6-6: Descriptive Statistics for Skewness with Wormhole flow control

	N	Skewness	
Final test score	Statistic	Statistic	Std. Error
1-core	10	1.417	0.687
2-core	10	2.160	0.687
4-core	10	1.485	0.687

The SPSS analysis for both flow control mechanisms reported a statistic for standard error for the skewness. The result was greater than ± 1.96 when either score was divided by its standard error. This suggests that the results were not normally distributed with respect to that statistic. An SPSS output for skewness tests from experimental results is given in Table 6-5 and Table 6-6. Skewness for wormhole flow control mechanisms is positive, indicating that the results are slightly right-skewed and peaked compared to a normal distribution. But with store-and-forward flow control, 4-core gives negatives result, indicating that the results are slightly left-skewed. Applying the rule of thumb of dividing each value by its standard error, SF gives 1-core=2.062, 2-core=0.003 and 4-core=-0.578. While WH gives 1-core=2.080, 2-core=3.144 and 4-core=2.162 as a measure of skewness. Most of the results are greater than ± 1.96 limits, suggesting that the result is not normal. This is confirmed by visual inspection of the histogram of the same results for Store-and-Forward flow control mechanism shown in Figure 6-1 for 1-core, Figure 6-3 for 2-core and Figure 6-5 for 4-core. The histogram for Wormhole flow control mechanism is shown in Figure 6-2 for 1-core, Figure 6-4 for 2-core and Figure 6-6 for 4-core.

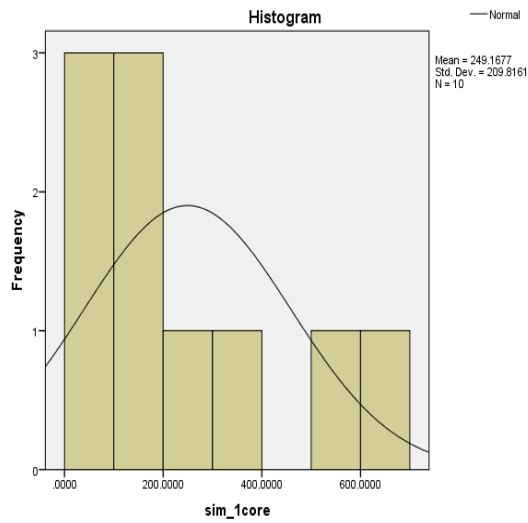


Figure 6-1: Histogram with distribution curve plotted for store-and-forward flow control mechanism with 1-core

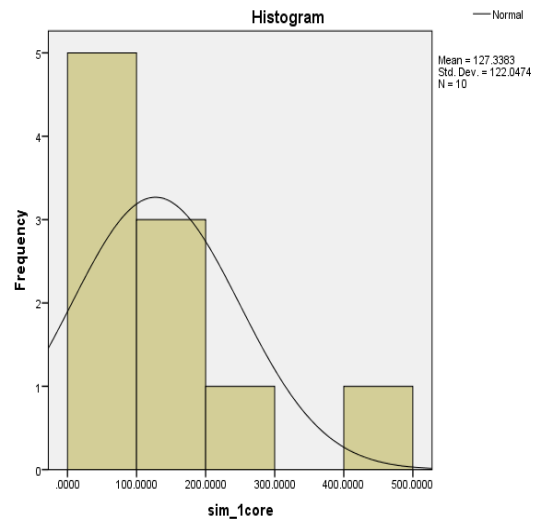


Figure 6-2: Histogram with distribution curve plotted for wormhole flow control mechanism with 1-core

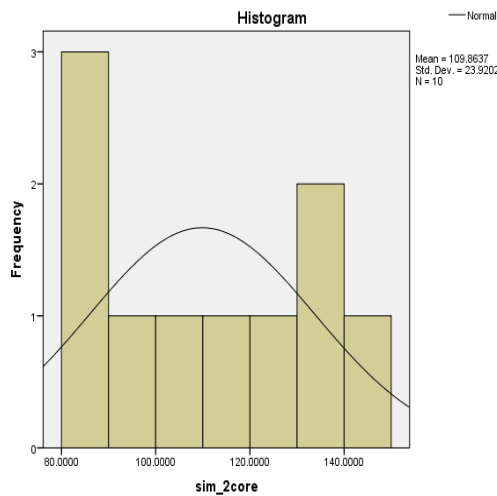


Figure 6-3: Histogram with distribution curve plotted for store-and-forward flow control mechanism with 2-core

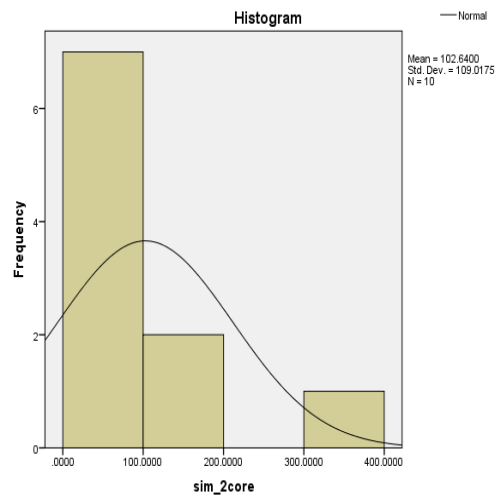


Figure 6-4: Histogram with distribution curve plotted for wormhole flow control mechanism with 2-core

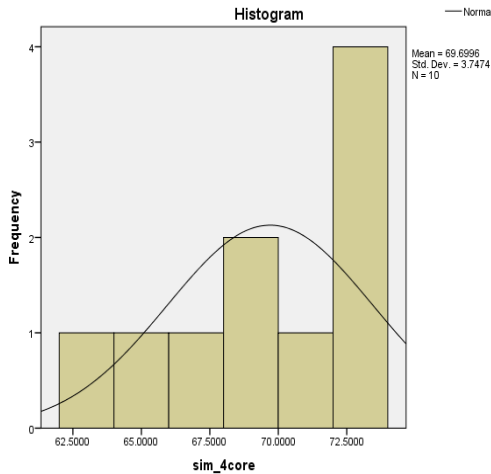


Figure 6-5: Histogram with distribution curve plotted for store-and-forward flow control mechanism with 4-core

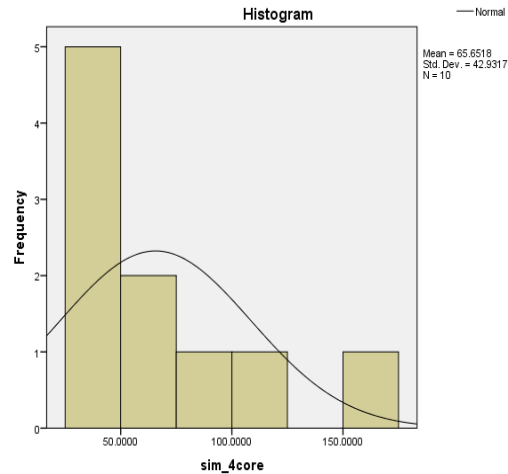


Figure 6-6: Histogram with distribution curve plotted for wormhole flow control mechanism with 4-core

6.3 Mean Differences between Mechanisms

Since there are two flow control mechanisms, store-and-forward (SF) and wormhole (WH), involved in the simulations, the Mann-Whitney test was applied to detect any statistically significant difference between the results. The Mann-Whitney test is used in analyses in which there is two conditions and different subjects have been used in each condition, but the assumptions of parametric testing are not tenable (Field, 2000).

The purpose of this test was to assess the interconnection performance by comparing message latency results between the two flow control mechanisms. It investigated which flow control mechanism yielded the best results for the latency evaluation metric.

Twenty traffic rates (N) were used in all: 10 traffic rates were based on SF and another 10 traffic rates were representing WH. Two flow controls are defined as two mechanism in the analysis (1=SF, and 2=WH).

Table 6-7: The comparison of flow control mechanism performance using Mann-Whitney test

Mechanism	N	Median	Interquartile range	U	Z	<i>p</i>
1-core SF	10	143.0282	98.2132 – 380.4279	28.000	-1.663	0.096
WH	10	84.4709	31.3061 – 198.9897			
Total	20					
2-core sf	10	105.4425	86.5303 – 134.1366	1.000	-3.704	<0.001
wh	10	49.8460	27.7618 – 69.8574			
Total	20					
4-core SF	10	70.1651	65.9904 – 73.2761	0.000	-3.780	<0.001
WH	10	35.7715	30.5565 – 38.3209			
Total	20					

Comparison of message latency between two flow control mechanisms using error bar charts

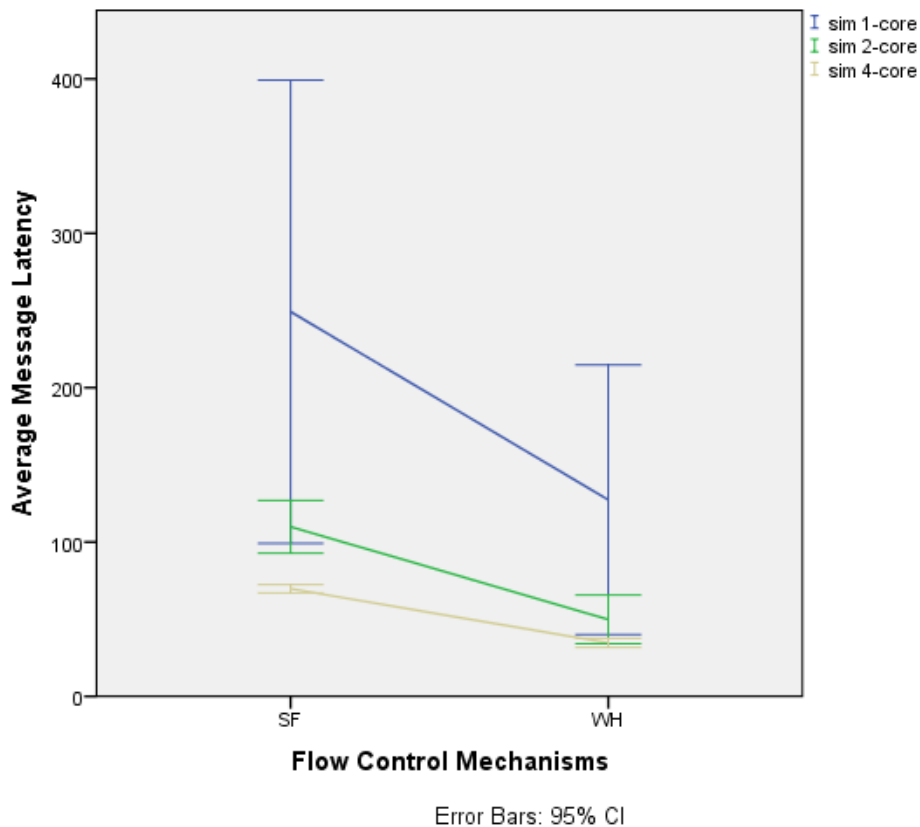


Figure 6-7: Error bar chart of average message latency between store-and-forward flow control and wormhole flow control

6.3.1 Discussion

The comparison of flow control mechanism performance using Mann-Whitney test are shown in Table 6-7. This test was conducted to examine whether the latency increase between the cores. A U of zero indicates the greatest possible difference between the two flow control mechanisms. Z -approximation of U is calculated and used to assess

statistical significance. For this experiment, a calculated z larger than ± 1.96 can be considered statistically significant. It can be concluded that, with p -value is below 0.05, there is a statistically significance difference between the message latency of SF flow control and WH flow control mechanisms.

A Mann-Whitney U is used when there are fewer than 20 cases in each group. A Mann-Whitney U test indicated that the message latency of Store-and forward flow control with 1-core were significantly higher than those of the Wormhole flow control ($N=10$, $U=28.000$, $z = -1.663$, $p=0.096$, two-tailed). However, this test also indicated that there was no significant difference between mechanisms in the message latency for 2-core and 4-core ($p<0.001$) where for 2-core with $N=10$, $U=1.000$, $z = -3.704$, $p<0.001$, two-tailed, and 4-core with $N=10$, $U=0.000$, $z = -3.780$, $p<0.001$, two-tailed.

The clustered bar graph in Figure 6-7 illustrates the differences between the message latency results for both flow control mechanism in this test. The Store-and-Forward flow control outnumbers the Wormhole flow control in all message latency results. This graph also shows that the two distributions are similarly shaped. The graph clearly illustrates how SF flow control tends to have much larger latency than WH flow control. However, the throughput saturation point of SF flow control is larger than WH flow control indicated that more packets can be transmitted into the network in the light traffic.

Thus, it can be concluded that there is a change in the latency as the number of cores increase when applying Store-and-Forward flow control. Whereas, there is almost no statistically significant changes in the latency when then number of cores increase when applying Wormhole flow control. However, Store-and-forward flow control is easier to implement since there is no blocking mechanism happen in the transmission. Although the latency increase, but with more cores, it will increase the performance.

The results indicated that both flow control mechanisms when applied in the MCMCA, the performance increased with two or more cores. The test demonstrated that the wormhole mechanism produces less latency than the store-and-forward flow control mechanism.

6.4 Mean Differences between Processor Core Performance

The following test was applied to discern any difference in the performance for the three sizes of processor core. The independent variable in this test is the traffic rate while core sizes constitute the dependent variable. The test for a difference in the true means of message latency used the Friedman test with a significance level of $\alpha=0.01$.

The Friedman test allows for the analysis of repeated-measures data for two or more conditions or to matched-subjects data which are matched in pairs, triplets or in some greater number (Field, 2013).

The test aimed to investigate in whether each core is produced as more latency than the other cores. For this test, analyses will evaluates whether message latency differ significantly between three number of cores. Twenty traffic rates were used in all: 10 traffic rates were based on Store-and-Forward flow control mechanism and another 10 traffic rates were representing Wormhole flow control mechanism.

Table 6-8: Comparison between processor core performance with two flow control mechanism

Processor cores		N	Median	Interquartile range	χ^2	p
1-core		20	121.044	71.949 - 255.311	25.900	0.000002
2-core		20	81.228	47.165 - 107.743		
4-core		20	51.571	35.335 - 70.525		

Table 6-9: Pairwise comparison between processor core performance for two flow control mechanism

	core2 - core1	core4 - core2	core4 - core1
Z	-3.883	-3.621	-3.733
Asymp. Sig. (2-tailed)	<0.01	<0.01	<0.01

Table 6-10: Comparison between processor core performance with Store-and-Forward flow control mechanism

Processor cores		N	Median	Interquartile range	χ^2	p
1-core	SF	10	143.0282	98.2132 - 380.4279	20.000	<0.01
2-core	SF	10	105.4425	86.5303 - 134.1366		
4-core	SF	10	70.1651	65.9904 - 73.2761		

Table 6-11: Pairwise comparison between processor core performance with Store-and-Forward flow control mechanism

	core2 - core1	core4 - core2	core4 - core1
Z	-2.803 ^c	-2.803 ^c	-2.803 ^c
Asymp. Sig. (2-tailed)	.005	.005	.005

Table 6-12: Comparison between processor core performance with Wormhole flow control mechanism

Processor cores		N	Median	Interquartile range	χ^2	<i>p</i>
1-core	WH	10	84.4709	31.3061 – 198.9897	7.800	0.020
2-core	WH	10	49.8460	27.7618 – 69.8574		
4-core	WH	10	35.7715	30.5565 – 38.3209		

Table 6-13: Pairwise comparison between processor core performance with Wormhole flow control mechanism

	core2 - core1	core4 - core2	core4 - core1
Z	-2.701 ^c	-1.988 ^c	-2.293 ^c
Asymp. Sig. (2-tailed)	.007	.047	.022

6.4.1 Discussion

Table 6-8 and Table 6-9 showed the comparison results for two flow control mechanism, Store-and-forward flow control (SF) and Wormhole flow control (WH). The table provides basic descriptive statistics for the three number of cores. *N* is the number of traffic rate in the research. Three post-hoc analyses with Wilcoxon pairwise comparison test were conducted to test if each mechanism was statistically significantly different from others (Allen & Bennett, 2010). If the result was significantly difference, the Bonferroni adjustment will be used to reduce the chances of obtaining false-positive results when multiple pair wise tests are performed on a single set of data (Field, 2013).

Friedman test were used to compare the performance of three results based on three different number of cores. Friedman test were carried out in two steps. The first step was to compare the processor performance regardless of the mechanisms. The second step was carried out to examine the difference of processor performance according to the flow control mechanisms.

For the first step, the results indicate that there are statistical differences of processor performance regardless of the mechanism involved ($N=20$, $\chi^2=25.900$, $p<0.000002$). The Bonferroni adjustment is needed to maintain an alpha rate of 0.05 over multiple comparisons (Allen & Bennett, 2010). It requires dividing the alpha level (0.05) by the number of comparisons being made (3 number of cores). Here, $0.05/3=0.017$ and a comparison can be considered statistically significant if *p* is less than 0.017.

Follow up pairwise comparisons with the Wilcoxon test and a Bonferroni adjusted α of 0.017 indicated that the 2-core was perceived as significantly produce more latency than 1-core ($z=-3.883$, $p<0.001$). There was also significant difference between 4-core and 1-core performance ($z=-3.733$, $p<0.001$), whereas the different between the 4-core and 2-core performance was clearly significant ($z=-3.621$, $p<0.001$).

Table 6-10 represents the results for the store-and-forward (SF) flow control mechanism while

Table 6-12 concerns the wormhole (WH) flow control mechanism. This test is to determine if message latency changed significantly with the number of cores based on two flow control mechanism.

For the second step, Friedman test were carried out in order to compare the processor core performance according to the flow control mechanisms involved. For the SF flow control mechanism, there was a statistical difference of processor core performance ($N=10$, $X^2=20.000$, $p<0.000045$).

Follow up pairwise comparisons in Table 6-11 with the Wilcoxon test and a Bonferroni adjusted α of 0.017 indicated that for SF flow control, there were difference between all processor cores performance ($z=-2.803$, $p<0.005$). Hence there exists enough evidence to conclude that there is a difference in the mean message latency for the three core sizes in interconnection performance based on SF flow control.

On the other hand, applied to the results for WH, with $N=10$, Friedman's chi-square has a value of 7.800 and a p -value <0.020 is less than 0.05. It was clear that there was a significant difference between message latency for the three cores. Hence, there is enough evidence that message latency results for the three core sizes are different. This is due to small latency occurs in the light traffic.

Follow up pairwise comparisons in Table 6-13 with the Wilcoxon test and a Bonferroni adjusted α of 0.017 for WH flow control, the difference between the rankings of the processor core with 2-core and 1-core is statistically significant ($p=0.007$); the difference between the rankings of 4-core and 1-core is approaching significance ($p=0.022$); and the difference between rankings of the 4-core and 2-core is clearly non-significant ($p=0.047$).

It can be concluded that the network performance improved as the number of cores increase. However, there is no relationship between the latency and the number of cores.

6.5 Summary

This chapter explained the approach used to analyse the experimental results provided in this thesis, which argues that simulation can be an effective method of investigating research problems and evaluating proposed solutions. The analysis was performed by using SPSS with two specific analyses. The normality test showed that both of the flow control simulation results indicated p -value for 1-core (SF), 2-core and 4-core (WH) is less than 0.05, indicating a not-normally distributed data. Although the p -values for other cores are greater than 0.05 (i.e. normally distributed) parametric test (e.g. independent sample t test and one-way repeated measure ANOVA) cannot be used because all groups of comparison must be normally distributed. Therefore, non-parametric test (e.g. Mann Whitney and Friedman tests) were applied.

The first analysis was to investigate the interconnection performance by comparing message latency results between the two flow control mechanisms by using the Mann-Whitney test. A Mann-Whitney U test indicated that the message latency of Store-and forward flow control with 1-core were significantly higher than those of the Wormhole flow control ($N=10$, $U=28.000$, $z = -1.663$, $p=0.096$, two-tailed). However, this test also indicated that there was no significant difference between mechanisms in the message latency for 2-core and 4-core ($p<0.001$) where for 2-core with $N=10$, $U=1.000$, $z = -3.704$, $p<0.001$, two-tailed, and 4-core with $N=10$, $U=0.000$, $z = -3.780$, $p<0.001$, two-tailed. This concluded that that there is a change in the latency as the number of cores increase when applying Store-and-Forward flow control. Whereas, there is almost no statistically significant changes in the latency when then number of cores increase when applying Wormhole flow control. Although the latency increase, but with more cores, it will increase the performance.

The second analysis was performed to test the significance of the differences between the processor cores performance. The Friedman test was conducted and the results indicate that there are statistical differences of processor performance regardless of the mechanism involved ($N=20$, $\chi^2=25.900$, $p<0.000002$). Follow up pairwise comparisons for both mechanism with 1-core, 2-core and 4-core, the Wilcoxon test were conducted and a Bonferroni adjusted α of 0.017 indicated that the 2-core was perceived as significantly produce more latency than 1-core ($z=-3.883$, $p<0.001$). There was also significant difference between 4-core and 1-core performance ($z=-3.733$, $p<0.001$), whereas the different between the 4-core and 2-core performance was clearly significant ($z=-3.621$, $p<0.001$).

The Friedman test confirmed that there was a significant improvement in processor performance from 1-core towards 4-core. It can be concluded that the network performance improved as the number of cores increase.

The next chapter presents the conclusion of the thesis and its contribution to the research. It will also discuss future research directions to arrive at alternative solutions to overcoming the limitations in the cluster system.

Chapter 7 Conclusions, Contributions and Future Work

This chapter begins by providing conclusions arising out of the research described, and outlining the contribution made. Possible directions for future work are then suggested. Concluding remarks are given at the end.

7.1 Conclusions

Clusters are now playing a major role in solving large-scale computing application problems, as they meet the need for faster and more reliable systems, especially as they are often built using commodity-off-the-shelf (COTS) hardware components and commonly-used software. The advances made in these technologies are making clusters an appealing solution for cost-effective parallel computing and have emerged as mainstream parallel platforms for high-performance, high-throughput and high-availability computing. However, various limitations to the available cluster system, which is further constrained by a single-core processor, are associated with enabling new architecture. This thesis addresses the limitations by providing a new multi-core multi-cluster architecture, based on collaborative architecture using multi-core clusters and multi clusters. The proposed architecture was developed for five interconnection networks, while the key contribution of this thesis has been the development of multi-core multi-cluster architecture. A brief summary of the aims and contributions of each chapter is outlined below, with additional unaddressed points.

Chapter 2 described the motivation for using clusters as well as the technologies available for building a new cluster architecture. Cluster computing has emerged as a result of the convergence of several trends, including the availability of inexpensive high-performance microprocessors and high-speed networks, the development of standard software tools for high performance parallel and distributed computing, and the increasing need of computing power for computational science and commercial applications. It is clear that high-speed networks for cluster computing are important to support the needs of better performance. The rapid change in interconnection network technology provides a new opportunity for a researcher to improve the

performance of cluster computing. Although much progress has been made in the development of low-latency protocols and new standard architectures, it creates interesting new challenges. The capability of clusters to deliver high performance and availability on a single platform is empowering many existing and emerging applications, and making clusters the platform of choice.

Several research issues are discussed in multi-core cluster and interconnection networks which contribute to a new architecture. The various issues presented in Chapter 2 show that the disadvantages of the single-core cluster moved researchers to focus on multi-core clusters. Multi-core clusters solve various concerns by dividing the workload between different cores, which speeds up performance. Chapter 2 also presented a background for the simulation model structure, which includes network topology, flow control mechanisms and routing algorithms. These are important components of message passing and communication in an interconnection network.

Chapter 3 presented the new architecture, known as Multi-core multi-cluster Architecture (MCMCA), built up of numbers of clusters where each cluster is composed of a number of nodes. Each node of a multi-core cluster has a number of processors, each with two or more cores with their own cache. Cores on the same chip share the local memory. The interconnection network connects the cluster nodes. By integrating multi-core processors and multiple clusters, its purpose is to provide an alternative architecture for the interconnection networks that improves the performance of the interconnection network by ensuring greater throughput and lower latency.

The research methodology described in Chapter 3 involved simulation development and experiment. The simulation models were built using the OMNeT++ network simulation tool, and baseline measures for the MCMCA were presented. A series of simulation experiments under various configurations and design parameters were performed. The results showed that the simulation model can be extended to investigate the performance of a multi-core cluster system based on multi-cluster architecture. Performance evaluation focused on communication latency and network throughput.

Chapter 4 demonstrated the performance evaluation of the interconnection network based on a store-and-forward flow control mechanism. The performance of single-core cluster and multi-core cluster architectures were compared by means of simulation experiments and an analytical model was used to validate the simulation results. Three simulation experiments were designed to demonstrate the MCMCA ability to predict interconnection network performance. The latency results suggested that, compared to single-core processor, a multi-core processor can improve the network performance by 51%-76%. This indicates that optimizing all levels of the

interconnection network is important in this architecture. Another observation was that the architecture can achieve lower latency and higher throughput as the number of cores increases.

The experiments also suggested that, compared to a single-core cluster, MCMCA can scale well from small (8-cluster) to large clusters (16-cluster to 128-cluster) while achieving low latency and high throughput. The impact of the architecture on message length reveals that small latency occurs with smaller message lengths (128 B to 512 B) but that latency increases with large message length (1 KB to 16 KB). The comparison of results between the analytical approach and those produced by the simulation experiments suggests that the analytical model provides a good basis for predicting the communication delay in interconnection network performance of the Multi-Core Multi Cluster Architecture (MCMCA).

Chapter 5 presented a simulation model based on the wormhole flow control mechanism. Three sets of experiments were designed to get an insight into the architectural capability. For the baseline experiment, the MCMCA simulation model outperformed the previous model where the difference between simulation model and the analytical model results was less than 1%. The first simulation experiment was conducted to investigate MCMCA performance based on latency and network throughput. The latency results suggested that as the traffic rate increases, the average communication latency increases, as the messages have to wait for resources before travelling into a network. In light traffic, higher latency occurs for larger numbers of cores due to task allocation between cores and a blocking mechanism. However, compared to the store-and-forward flow control mechanism, the wormhole flow control mechanism is able dramatically to improve latency, since packets are transferred using smaller units called flits: more flits can be transmitted in the same channel, thus minimising the latency.

Another simulation experiment was performed to gain more insight into the impact of the architecture on cluster size and message length, so to reveal potential scalability. Based on various message lengths, it was observed that a larger number of cores overtakes single-core performance by 4%-10% more under a uniform traffic pattern. However, due to the blocking issue, the wormhole flow control mechanism was unable to achieve optimal throughput performance. The resulting performance evaluation suggests an alternative for system designers building cluster applications.

In Chapter 6 presented the statistical analysis. Two statistical analysis were conducted to compare the experimental results based on store-and-forward and wormhole flow control mechanisms. The normality test showed that both of the flow control simulation results indicated p -value for 1-core (SF), 2-core and 4-core (WH) is

less than 0.05, indicating a not-normally distributed data. Although the p -values for other cores are greater than 0.05, parametric test cannot be used because all groups of comparison must be normally distributed. Therefore, non-parametric test (e.g. Mann Whitney and Friedman tests) were applied. The first analysis was to investigate the interconnection performance by comparing message latency results between the two flow control mechanisms by using the Mann-Whitney test. The finding indicated that the message latency of Store-and forward flow control with 1-core were significantly higher than those of the Wormhole flow control ($p=0.096$). However, this test also indicated that there was no significant difference between mechanisms in the message latency for 2-core and 4-core ($p<0.001$). This concluded that there is a change in the latency as the number of cores increase when applying Store-and-Forward flow control. Whereas, there is almost no statistically significant changes in the latency when then number of cores increase when applying Wormhole flow control.

The second analysis was performed to test the significance of the differences between the processor cores performance. The Friedman test was conducted and the results indicate that there are statistical differences of processor performance regardless of the mechanism involved ($p<0.000002$). The Friedman test confirmed that there was a significant improvement in processor performance from 1-core towards 4-core. It can be concluded that the network performance improved as the number of cores increase.

These analyses are important because they suggest that the simulation model can be an effective method to produced precise results in investigating research problems and evaluating proposed solutions in clusters system.

7.2 Research Contributions

The work described in this thesis was aimed at designing a high performance and scalable architecture for multi-core systems. The main contribution of this research is in developing a novel multi-core multi-cluster architecture (MCMCA) to improve interconnection network performance. In addition, this thesis develops a new simulation model based on two different flow control mechanisms and provides a method of testing message latency effects on system performance, the impact on cluster size and potential scalability.

Overall, this thesis contributes to innovative cluster architecture design and development to investigate the performance of interconnection networks in multi-core multi-cluster architecture. The major contributions of this thesis are thus:

1. **Multi-core Multi-cluster Architecture (MCMCA):** The design and development of a novel architecture to investigate the performance of interconnection

network in multi-core multi-cluster. The new architecture involved five communication networks compared to three in the existing multi-core cluster architecture. The performance measurements focused on overall communication latency within the simulation model, and the simulation results were analysed for comparison with the published results for existing cluster architectures. The research reported in this thesis incorporated multi-cluster architecture for a more scalable approach, and this is the first investigation into incorporating this.

2. **MCMCA Performance Model based on Store-and-Forward Flow Control Mechanism:** The development and evaluation of a new simulation model to investigate the performance of an interconnection network based on a store-and-forward flow control mechanism.
3. **MCMCA Performance Model based on Wormhole Flow Control Mechanism:** The development and evaluation of a new simulation model to investigate the performance of an interconnection network based on a wormhole flow control mechanism.
4. **Validation of the Models:** The validity and accuracy of the model were demonstrated by comparing the results obtained by simulation experiment results with those obtained by an analytical model.

7.3 Future Research Directions

The research reported in this thesis has successfully addressed the research questions outlined in Chapter 1, section 1.2. However, there is a comprehensive application area with complex underlying details which could not possibly be covered entirely in this thesis. Thus, the future work in this area was suggested by: (a) the limitations of the research; (b) specific extensions of this research.

7.3.1 Limitations of the Research

Simulation studies are not without their limitations. Improvements can be made in future studies in the following areas.

a) Energy Efficiency

Multi-core processors are designed to adhere to reasonable power consumption and heat dissipation. Burger (2005) states that multi-core processors allow systems to put more processing power in a smaller package that uses less power and generates less heat for the computational power derived. Multi-core processors can result in

significant power savings and performance improvements if the applications are mapped to cores judiciously (Geer, 2007). To reduce unnecessary power consumption and reduce the heat, the design model must run the multiple cores at a lower frequency to ensure heat dissipation is distributed across the processor (Shainer et al., 2013). Thus, there is a need to design the cores, the memory and the interconnection network to prevent energy inefficiency.

The main idea of this study was to design a comprehensive multi-core architecture which included energy efficient aspects. However, due to problems during the development of the simulation model, the MCMCA focused on the performance of the interconnection network and scalability issues as focus, leaving other aspects as interesting future directions.

b) Fault-tolerance and Resilience Architectures

Fault tolerance is the ability of the network to perform in the presence of one or more faults (Dally & Towles, 2004). With the development of the multi-core cluster system, more processors can be implemented on a single chip. Typically, a modern cluster system is deployed in a highly distributed manner, with communication between nodes becoming a critical part of the system architecture. This means that some form of fault tolerance of this communication network is required in order to achieve satisfactory system availability. An analysis conducted by Schroeder & Gibson (2010) showed that 22 different HPC systems exhibited failure rates ranging from 20 to more than 1000 per year on average. Their results also show that failure rate will continue to increase with system size. This demonstrates that it is important to consider the use of fault tolerance and resilience as a requirement for cluster systems. It is also important to understand the effectiveness of differing fault tolerances for MCMCA.

As chip cost is increasing, multi-core cluster design is a solution for enhancing system performance. Thus, cluster systems are in need of a high level of fault tolerance without substantial loss of overall performance. Several studies have investigated the fault tolerance by using analytical models (Requena, Requena, Rodriguez & Marin, 2009; Varghese et al., 2010). However, there are few studies on simulation modelling of fault tolerance and resilience in multi-core clusters. This issue requires further investigation and will be suggested for future work.

c) Cache Coherence

The shift towards multi-core clusters will rely on parallel software to achieve continuing exponential performance. Although processors logically access the shared memory, cache hierarchies are crucial to achieve fast performance. Since each core has its own cache, the copy of the data in that cache may not always be the most up-to-date

version and may produce invalid results. The design of the memory module with a cache will enable most of the shared variables to be accessed from a fast memory (cache) and only a small fraction of the shared variables from the slow main memory, thus leading to a small average memory access time (Rauber & Runger, 2010).

Cache size has become important for improving processor and network performance of a cluster system. Research in Das (2008) compared the network performance of different cache sizes with two widely used metrics, cache hit ratio and cache average access delay. As the cache size increases, cache hit ratio improves network performance and access delay decreases. However, the performance of large size caches may be severely constrained by the interconnection network (Muralimanohar & Balasubramonian, 2007). The main focus in this work is exploring various techniques to accelerate cache access that take advantage of a low bandwidth and low latency network.

For MCMCA, working within the cache become trickier since data is not only transferred between a core and memory, but also between cores. Building a cache coherence cluster system provided a flexible infrastructure to expand the systems in size and function (Liquan, Muralimanohar, Ramani, Balasubramonian & Carter, 2006). However, this flexibility comes at cost in performance such as substantial increase in memory latency when multiple cores share a cache. To gain more insight on the impact of cache size on network performance by the MCMCA needs a specific study, outside the scope of this work.

7.3.2 Specific Extensions

To take this research forward, numerous extensions are possible. This section highlights some of the areas for future research.

a) Extension to Non-Uniform Traffic

Future work should develop a simulation model for MCMCA based on a non-uniform traffic pattern. This should accommodate the traffic generated by real-world applications, which could provide communication network performance results so that comparisons may be made between different models of the MCMCA. The approach taken and the accuracy of the simulation outcome should provide a good basis for predicting the performance behaviour of MCMCA in both uniform and non-uniform traffic patterns.

b) Extension to Modern Interconnects

There are numbers of modern interconnections which offer high performance with rich features. Modern interconnects such as QsNetII and InfiniBand are very attractive for

large-scale system design. It will be interesting to exploit such modern features to extend this architecture in the future.

7.4 Concluding Remarks

The expected outcome of the research reported here was to produce a novel architecture and simulation to measure and improve interconnection network performance in the implementation of the multi-core multi-cluster architecture. Analytical methodology provided validation for the simulation results.

The approach taken and the accuracy of the simulation and analysis make it an attractive tool for predicting the performance behaviour of multi core multi cluster architecture. The research has already resulted in international conference and journal publications as mentioned in Chapter 1, section 1.3.

Ultimately this research will benefit not only cluster architecture, but also high performance computing architecture including cloud computing. More research into high performance computing, especially cloud computing, may now be based on cluster architecture to solve its limitations, especially in satisfying peak workload performance (Chang, Walters & Wills, 2013; Kosinska, Kosinski & Zielinski, 2010; Moreno-Vozmediano, Montero & Llorente, 2011). The research reported here should also provide an alternative platform for high performance computing that will yield several benefits, such as high availability, fault tolerance and infrastructure cost reduction.

Appendix 2-A

As mentioned in 2.9.2e), a random number generator (RNG) is a program written for use in probability and statistics applications when large quantities of random digits are needed (Dally & Towles, 2004). Mersenne Twister is the random number generator employed by OMNeT++ (Varga, 2011), used to distribute the message destinations in the simulation model.

d) Mersenne Twister RNG

OMNeT++ primarily uses Mersenne Twister for random number generation. It uses the MT19937 RNG developed by Makoto Matsumoto and Takuji Nishimura in 1997 which has a cycle length of $2^{19937} - 1$ (Matloff, 2008). The Mersenne Twister has passed numerous tests for randomness and is distributed uniformly in 623 dimensions, generating an output which is free of long-term correlations (Jagannatham, 2008). It is considered to be fast as it avoids multiplications and divisions by using the advantages of caches and pipelines.

A configurable number of random numbers are provided to the simulation. Global random number streams are mapped to OMNeT++'s module which allows the use of variance reduction techniques without the need to change the configuration in the simulation model (Varga & Hornig, 2008). While seeding is automatic, auto-assigned using the run number, it is also possible to use manually-selected seeds. The simulation requires as many seeds as the number of global RNG streams configured. Due to the practically infinite cycle length of Mersenne Twister, overlapping of RNG streams is not an issue.

e) Seeding the Mersenne Twister RNG

Seeding is the procedure of setting the initial states of the RNG, so that it will produce a stream of random numbers (Wehrle et al., 2010). The RNG class implements support for seeding. Seed sets can be specified in the initialization section or for each run of OMNeT++. Mersenne Twister has such a long cycle that there is no need for seed generation because chances are very small that any two seeds produce overlapping streams (Matloff, 2008).

f) Chi-square Goodness of Fit Test

This is a non-parametric test used to find out how the observed value is significantly different from the expected value. In Chi-Square test, the term *goodness of fit* is used to compare the observed sample distribution with the expected probability distribution. This test also determines how well theoretical distribution (such as normal

or Poisson) fits the empirical distribution. In this test, sample data is divided into intervals. Then the numbers that fall into each interval are compared with the expected numbers in each interval. The formula for the statistic is:

$$\chi^2 = \sum \frac{(\text{observed value} - \text{expected value})^2}{\text{expected value}}$$

A high value of χ^2 implies a poor fit between the observed and expected value, so the upper tail of the distribution is used for most hypothesis testing for goodness of fit. To determine whether the traffic generation rates are random, the null and alternative hypotheses are as follows:

H_0 : Traffic generation rates are random

H_1 : Traffic generation rates are not random

Chi-Square	0.001
Degrees of Freedom	9
p-value	1.0

Table 1: Chi-Square goodness of fit test

Table 1 shows the test statistics and p-value. Since the p-value = 1.0 > 0.05, the null hypotheses was not rejected. At the $\alpha=0.05$ level of significance, there was not enough evidence to reject the null hypotheses, thus, the RNG fitted the theory that the traffic generation rates are random.

Appendix 3-A

MCMCA Simulation Model

a) Module Designs and Structure Diagram

The model behaviour built into each Network Description (NED) file will be captured in C++ files as code and can be edited in the Integrated Development Environment (Varga, 2011). Each NED file has its own C++ module source. Unlike many formats of deterministic discrete event simulation, the model is built at run-time to form a topology that represents the geometric structure and the communication links between the modules.

The topology and the communication links between the modules are represented by the NED file. Six module files have been built to describe the simulation model. Figure 1 shows the structure diagram of the module designs in MCMCA simulation model.

1. Network Topology file – this file describes the building blocks of the fat-tree topology, including cores, nodes and clusters.
2. Network Interface file – this file contains the interface of module types in fat-tree topology. Cores, nodes, clusters, switches, channels and the communication network are declared in this file and connections between them are established.
3. Communication Switch – this file acts as the connection for each switch and router in the model and it will determine how a message is transmitted along a path that has been selected by the routing algorithm.
4. Routing file – this file determines the path and schedules the routing algorithms for the packets in all communication networks based on FIFO (First-In-First-Out). It represents a single server queue that has the same service rate for each packet.
5. Message-Generator file – packets are generated by this file following the assumption that the message destinations are uniformly distributed.
6. Message-Sink file – this file will destroy the packets after each generation is completed and will gather event information for statistics.

To illustrate the architecture, a model of the architecture is developed with the simulation program. The simulation model applies fat-tree topology that describes the geometric structure used for the arrangement of switches and communication links to connect the processors. Two levels of communication switch represent the intra-chip (AC) and inter-chip (EC), while each node in the tree represents the processor. The

dotted line represents the route taken by a packet from source to destination. Figure 2 illustrates the packet travelling path in the OMNeT++ simulation model.

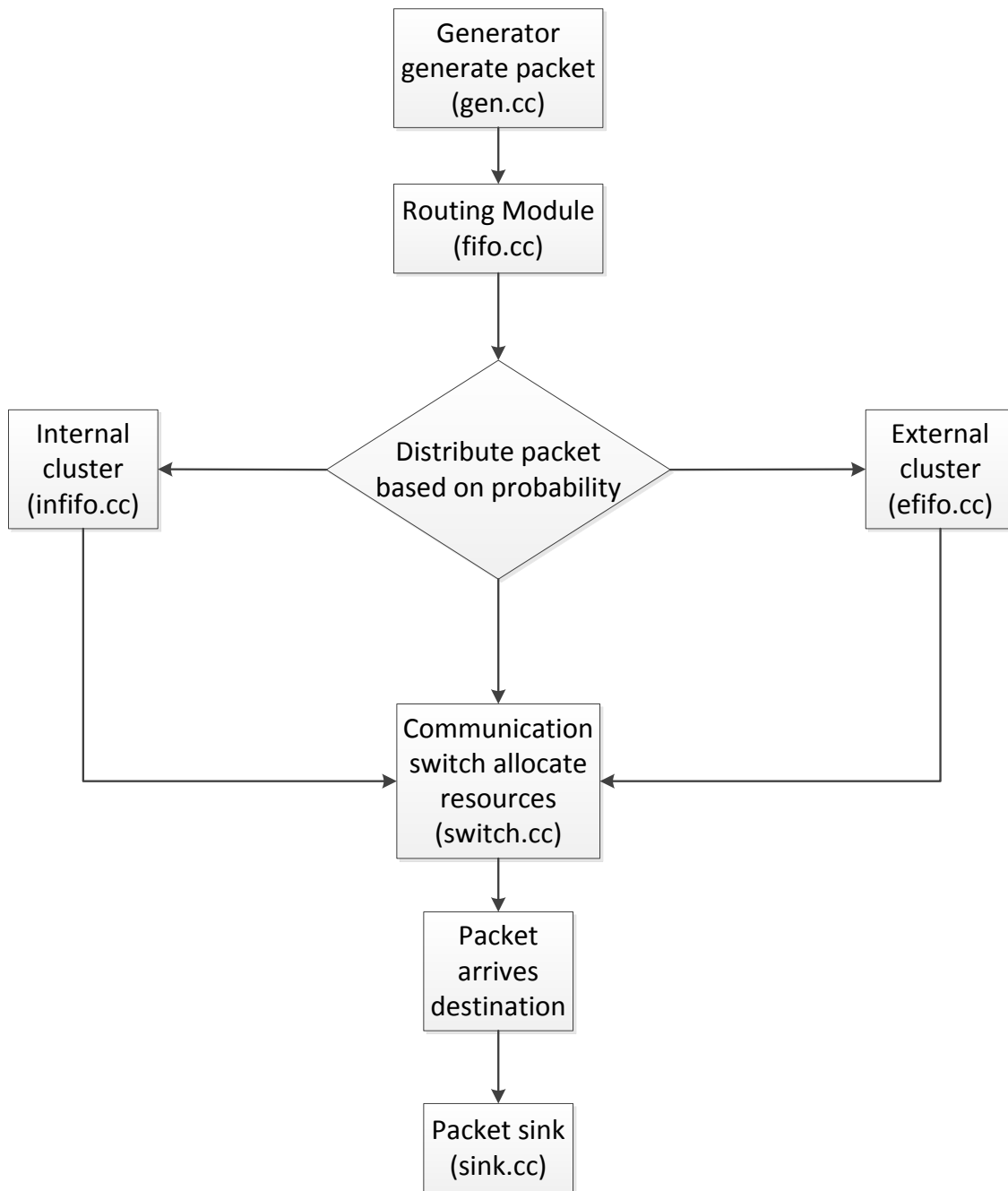


Figure 1: Structure diagram of simulation models

A packet travelling from source node to destination node will go up through internal switches of the tree until it finds the Nearest Common Ancestor (NCA) and then is transmitted down to the destination node. In this algorithm, each packet experiences two phases, an ascending phase to get a NCA, followed by a descending phase. For

example, a packet is to be sent from node 0:A to node 4:E and the switch connected to the source node is SW010. The packet will travel up in ascending phase to the NCA through switch SW001 and then go down in descending phase through switch SW012 until it reaches the destination node 4:E. More examples are shown in Table 1.

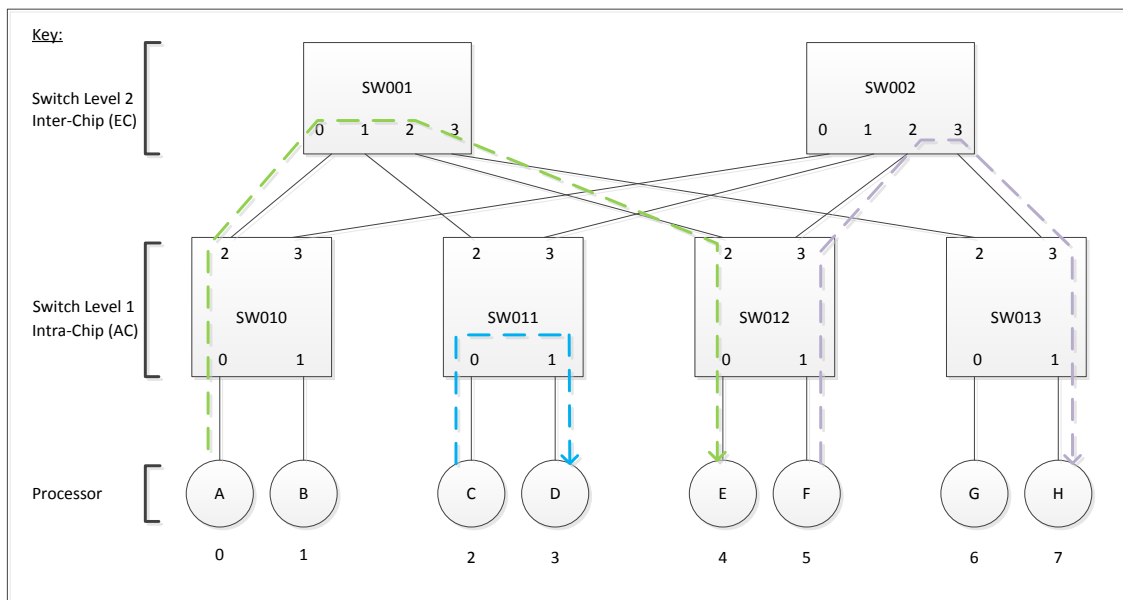


Figure 2: Illustration of packet travelling path in the simulation model

Table 1: Examples of routing algorithms

Source	Destination	Switch	In_port	Out_port	Communication Network
0:A	4:E	010	0	2	Intra-chip (AC)
		001	0	2	Inter-chip (EC)
		012	2	0	Intra-chip (AC)
2:C	3:D	011	0	1	Intra-chip (AC)
5:F	7:H	012	1	3	Intra-chip (AC)
		002	2	3	Inter-chip (EC)
		013	3	1	Intra-chip (AC)

b) Simulation Code

The topology and the communication links between the modules are represented by the network description, NED. The codes of the six modules were built to describe the simulation model, as follows.

a) Network Topology module

```

package mcsf;
//-----
// File: fattreenetwork.ned
//-----

module Node like IPNode
{
    parameters:
        int m;
        int n;
        int C;
        int address;
        @display("i=block/circle;is=vs");

    gates:
        input in[];
        output out[];

    submodules:
        gen: Generator {
            parameters:
                address = address;
                num_ports = m;
                num_trees = n;
                num_clusters = C;
                @display("p=100,50;i=gen");

            gates:
                out[2];
        }
        sink: Sink {
            parameters:
                address = address;
                @display("p=135,120;i=sink");

            gates:
                in[2];
        }
    connections:
        // to source queue to inject to the network
        gen.out[0] --> out[0];
        gen.out[1] --> out[1];
        in[0] --> sink.in[0];
        in[1] --> sink.in[1];
}

// NetworkSwitch --
//

module I_NetworkSwitch like ISwitch //ICN

```

```

{
    parameters:

        int m;
        int n;
        int C;
        int service_time;
        int cross_delay;
        int route_delay;
        int net_level;
        int address;

    gates:
        input in[];
        output out[];

    submodules:

        fifo[m]: inFifo {

            parameters:
                service_time = service_time;
            gates:
                in[2];
                out[2];
        }
        crsbar: CrossBar {

            parameters:
                cross_delay = cross_delay;
            gates:
                in[m];
                out[m];
        }
        rte: Router {

            parameters:
                num_ports = m;
                num_trees = n;
                num_clusters = C;
                route_delay = route_delay;
                net_level = net_level;
                address = address;
            gates:
                in[m];
                out[m];
        }
    connections:

        for i=0..m-1 {

            in[i] --> fifo[i].in[0];
            fifo[i].out[0] --> rte.in[i];
            rte.out[i] --> fifo[i].in[1];
            fifo[i].out[1] --> crsbar.in[i];
            crsbar.out[i] --> out[i];
        }
}

```



```

module E_NetworkSwitch like ISwitch
{
    parameters:
        int m;
        int n;
        int C;
        int service_time;
        int cross_delay;
        int route_delay;
        int net_level;
        int address;

    gates:
        input in[];
        output out[];

    submodules:
        fifo[m]: inFifo {
            parameters:
                service_time = service_time;
            gates:
                in[2];
                out[2];
        }
        efifo: eFifo {
            parameters:
                service_time = service_time;
            gates:
                in[2];
                out[2];
        }
        crsbar: CrossBar {
            parameters:
                cross_delay = cross_delay;
            gates:
                in[m+1];
                out[m+1];
        }
        rte: Router {
            parameters:
                num_ports = m;
                num_trees = n;
                num_clusters = C;
                route_delay = route_delay;
                net_level = net_level;
                address = address;
            gates:
                in[m+1];
                out[m+1];
        }
    connections:
        for i=0..m-1 {
            in[i] --> fifo[i].in[0];
            fifo[i].out[0] --> rte.in[i];
            rte.out[i] --> fifo[i].in[1];
            fifo[i].out[1] --> crsbar.in[i];
            crsbar.out[i] --> out[i];
        }
        in[m] --> efifo.in[0];

```

```

        efifo.out[0] --> rte.in[m];
        rte.out[m] --> efifo.in[1];
        efifo.out[1] --> crsbar.in[m];
        crsbar.out[m] --> out[m];
    }
module Merger like IMerg
{
    parameters:
        int m;
        int n;
        int C;
        int service_time;
        int address;

    gates:
        input in[];
        output out[];

    submodules:
        fifo_up: Fifo {

            parameters:
                service_time = service_time;
            gates:
                in[ (2*n-1) * (m/2) ^ (n-1) ];
                out[1];
        }
        fifo_down: Fifo {

            parameters:
                service_time = service_time;
            gates:
                in[1];
                out[ (2*n-1) * (m/2) ^ (n-1) ];
        }

    connections:

        for i=0..(2*n-1)*(m/2)^(n-1)-1 {
            in[i] --> fifo_up.in[i];
            fifo_down.out[i] --> out[i];
        }
        in[ (2*n-1) * (m/2) ^ (n-1) ] --> fifo_down.in[0];
        fifo_up.out[0] --> out[ (2*n-1) * (m/2) ^ (n-1) ];
    }
}
// Multi-Cluster System --
network FatTreeNetwork extends FatTree
{
    parameters:

        C = 8;
        m = 8;
        n = 2;
        nodetype = "Node";
        I_swtype = "I_NetworkSwitch";
        E_swtype = "E_NetworkSwitch";
        mergtype = "Merger";
}

```

b) Network Interface module

```
//-----
// File: fattreeinterface.ned
// The m-port n-tree topology
//-----

package mcsf;

moduleinterface IPNode
{
    parameters:
        int m;
        int n;
        int C;
        int address;
    gates:
        input in[];
        output out[];
}

moduleinterface ISwitch
{
    parameters:
        int m;
        int n;
        int C;
        int service_time;
        int cross_delay;
        int route_delay;
        int net_level;
        int address;
    gates:
        input in[];
        output out[];
}

moduleinterface IMerg
{
    parameters:
        int m;
        int n;
        int C;
        int service_time;
        int address;
    gates:
        input in[];
        output out[];
}

//declare simple
simple PNode like IPNode
{
    parameters:
```

```

        int m;
        int n;
        int C;

        int address;
    gates:
        input in[];
        output out[];
}
simple Switch like ISwitch
{
    parameters:
        int m;
        int n;
        int C;
        int service_time;
        int cross_delay;
        int route_delay;
        int net_level;
        int address;
    gates:
        input in[];
        output out[];
}
simple Merg like IMerg
{
    parameters:
        int m;
        int n;
        int C;
        int service_time;
        int address;
    gates:
        input in[];
        output out[];
}
// Definition of physical channel (NET1)
channel physical_n_net1 extends ned.DatarateChannel
{
    parameters:
        delay = 0.01s;
        datarate = 1000bps;
}
channel physical_net1 extends ned.DatarateChannel
{
    parameters:
        delay = 0.01s;
        datarate = 1000bps;
}
// Definition of physical channel (NET2)

```

```

channel phyNnet2 extends ned.DatarateChannel
{
    parameters:
        delay = 0.01s;
        datarate = 1000bps;
}
channel phynet2 extends ned.DatarateChannel
{
    parameters:
        delay = 0.01s;
        datarate = 1000bps;
}
// Definition of physical channel (NET3)
channel phyNnet3 extends ned.DatarateChannel
{
    parameters:
        delay = 0.02s;
        datarate = 500bps;
}
channel phynet3 extends ned.DatarateChannel
{
    parameters:
        delay = 0.01s;
        datarate = 500bps;
}
channel phyNnet4 extends ned.DatarateChannel
{
    parameters:
}

// M-port, N-tree

module FatTree
{
    parameters:
        int m;
        int n;
        int C;
        string nodetype;
        string I_swtype;
        string E_swtype;
        string mergtype;
        @display("bgb=1000,1000;bgp=10,20");

    submodules:
        node[(C*2)*(m/2)^n]: <nodetype> like IPNode {
            parameters:
                m = m;
                n = n;
                C = C;
        }
    }
}

```

```

        address = index;
        @display("p=100,300,row,50;i=comp_s");
    gates:
        out[2];
        in[2];
    }
    sw_ICN1[(C*(2*n-1))*(m/2)^(n-1)]: <I_swtype> like ISwitch {
        parameters:
            m = m;
            n = n;
            C = C;
            service_time = 0.0;
            cross_delay = 0;
            route_delay = 0;
            net_level = 0;
            address = index;
            @display("p=80,120,matrix,8,80,-80;i=queue");
        gates:
            out[m];
            in[m];
    }
    sw_ECNI[(C*(2*n-1))*(m/2)^(n-1)]: <E_swtype> like ISwitch {
        parameters:
            m = m;
            n = n;
            C = C;
            service_time = 0;
            cross_delay = 0;
            route_delay = 0;
            net_level = 1;
            address = index;
            @display("p=80,400,matrix,8,80,80;i=queue");
        gates:
            out[m+1];
            in[m+1];
    }
    sw_ICN2[(2*floor((log(C)-log(2))/(log(m)-log(2)))-
1)*(m/2)^(floor((log(C)-log(2))/(log(m)-log(2)))-1)]: <I_swtype> like
ISwitch {
        parameters:
            m = m;
            n = n;
            C = C;
            service_time = 0;
            cross_delay = 0;
            route_delay = 0;
            net_level = 2;
            address = index;
            @display("p=80,750,matrix,8,80,80;i=queue");
        gates:
            out[m];

```

```

        in[m];
    }
    merg[C]: <mergtype> like IMerg {
        parameters:
            m = m;
            n = n;
            C = C;
            service_time = 0;
            address = index;
            @display("p=80,600,matrix,8,80,80");
        gates:
            out[(2*n-1)*(m/2)^(n-1)+1];
            in[(2*n-1)*(m/2)^(n-1)+1];
    }
    connections:
        // ***** for ICN1 and ECN1
        *****
        // connect processing node with leaf switches
        for j=0..C-1, for i=0..2*(m/2)^n-1 {
            // ICN1
            node[j*(2*(m/2)^n)+i].in[0] <-- phyNnet2 <--
sw_ICN1[i/(m/2)+(j*(2*n-1))*(m/2)^(n-1)].out[i%(m/2)] if n>1;
            node[j*(2*(m/2)^n)+i].out[0] --> phyNnet2 -->
sw_ICN1[i/(m/2)+(j*(2*n-1))*(m/2)^(n-1)].in[i%(m/2)] if n>1;

            //ECN1
            node[j*(2*(m/2)^n)+i].in[1] <-- phyNnet3 <--
sw_ECN1[i/(m/2)+(j*(2*n-1))*(m/2)^(n-1)].out[i%(m/2)] if n>1;
            node[j*(2*(m/2)^n)+i].out[1] --> phyNnet3 -->
sw_ECN1[i/(m/2)+(j*(2*n-1))*(m/2)^(n-1)].in[i%(m/2)] if n>1;

            // if n==1 (M-port 1-Tree)
            //ICN1
            node[j*m+i].in[0] <-- phyNnet2 <-- sw_ICN1[j].out[i] if
n==1;
            node[j*m+i].out[0] --> phyNnet2 --> sw_ICN1[j].in[i] if
n==1;

            //ECN1
            node[j*m+i].in[1] <-- phyNnet3 <-- sw_ECN1[j].out[i] if
n==1;
            node[j*m+i].out[1] --> phyNnet3 --> sw_ECN1[j].in[i] if
n==1;
        }
        //connect internal (not root switch) switches with each other
        for h=0..C-1, for i=0..(n==1 ? -1 : (2*n-4)*(m/2)^(n-1)-1),
for j=0..(n==1 ? -1 : m/2-1), for k=i/(2*(m/2)^(n-
1))+1..i/(2*(m/2)^(n-1))+1, for w=((2*n-1)*(m/2)^(n-1))..((2*n-
1)*(m/2)^(n-1)), for q=((m/2)^(k-1))..((m/2)^(k-1)) {

            sw_ICN1[i+h*w].in[j+m/2] <-- phynet2 <-- sw_ICN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].out[(i%(m/2)^k)/q];

```

```

        sw_ICN1[i+h*w].out[j+m/2] --> phynet2 --> sw_ICN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].in[(i%(m/2)^k)/q];

        sw_ECN1[i+h*w].in[j+m/2] <-- phynet3 <-- sw_ECN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].out[(i%(m/2)^k)/q];
        sw_ECN1[i+h*w].out[j+m/2] --> phynet3 --> sw_ECN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].in[(i%(m/2)^k)/q];

    }
    //connect root switches with sub trees
    for h=0..C-1, for i=(n==1 ? 0 : (2*n-4)*(m/2)^(n-1))..(n==1 ?
-1 : (2*n-3)*(m/2)^(n-1)-1), for j=0..(n==1 ? -1 : m/2-1), for
k=i/(2*(m/2)^(n-1))+1..i/(2*(m/2)^(n-1))+1, for w=((2*n-1)*(m/2)^(n-
1))..((2*n-1)*(m/2)^(n-1)), for q=((m/2)^(k-1))..((m/2)^(k-1)) {

        //ICN1
        sw_ICN1[i+h*w].in[j+m/2] <-- phynet2 <-- sw_ICN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].out[(i%(m/2)^k)/q];
        sw_ICN1[i+h*w].out[j+m/2] --> phynet2 --> sw_ICN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].in[(i%(m/2)^k)/q];

        sw_ICN1[(i+(m/2)^(n-1))+h*w].in[j+m/2] <-- phynet2 <--
sw_ICN1[(((i-i%(m/2)^k)+2*(m/2)^(n-
1))+(m/2)*(i%q))+j)+h*w].out[(i%(m/2)^k)/q+m/2];
        sw_ICN1[(i+(m/2)^(n-1))+h*w].out[j+m/2] --> phynet2 -->
sw_ICN1[(((i-i%(m/2)^k)+2*(m/2)^(n-
1))+(m/2)*(i%q))+j)+h*w].in[(i%(m/2)^k)/q+m/2];

        //ECN1
        sw_ECN1[i+h*w].in[j+m/2] <-- phynet3 <-- sw_ECN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].out[(i%(m/2)^k)/q];
        sw_ECN1[i+h*w].out[j+m/2] --> phynet3 --> sw_ECN1[(((i-
i%(m/2)^k)+2*(m/2)^(n-1))+(m/2)*(i%q))+j)+h*w].in[(i%(m/2)^k)/q];

        sw_ECN1[(i+(m/2)^(n-1))+h*w].in[j+m/2] <-- phynet3 <--
sw_ECN1[(((i-i%(m/2)^k)+2*(m/2)^(n-
1))+(m/2)*(i%q))+j)+h*w].out[(i%(m/2)^k)/q+m/2];
        sw_ECN1[(i+(m/2)^(n-1))+h*w].out[j+m/2] --> phynet3 -->
sw_ECN1[(((i-i%(m/2)^k)+2*(m/2)^(n-
1))+(m/2)*(i%q))+j)+h*w].in[(i%(m/2)^k)/q+m/2];

    }
    //***** for Merger
    *****

    //ECN to merger
    for h=0..C-1, for i=0..(n==1 ? 0 : (2*n-1)*(m/2)^(n-1)-1) {
        sw_ECN1[i+h*((2*n-1)*(m/2)^(n-1))].out[m] --> phyNnet4 --
>merg[h].in[i];
        sw_ECN1[i+h*((2*n-1)*(m/2)^(n-1))].in[m] <-- phyNnet4 <--
merg[h].out[i];
    }

    // ***** for ICN2

```



```

*****
// connection to ICN2
for Cn=floor((log(C)-log(2))/(log(m)-log(2)))..floor((log(C)-
log(2))/(log(m)-log(2))), for i=0..2*(m/2)^Cn-1 {

    // ICN2
    merg[i].in[(2*n-1)*(m/2)^(n-1)] <-- phynet3 <--
sw_ICN2[i/(m/2)].out[i%(m/2)] if Cn>1;
    merg[i].out[(2*n-1)*(m/2)^(n-1)] --> phynet3 -->
sw_ICN2[i/(m/2)].in[i%(m/2)] if Cn>1;
    // if Cn==1 (M-port 1-Tree)

    //ICN2
    merg[i].in[(2*n-1)*(m/2)^(n-1)] <-- phynet3 <--
sw_ICN2[0].out[i] if Cn==1;
    merg[i].out[(2*n-1)*(m/2)^(n-1)] --> phynet3 -->
sw_ICN2[0].in[i] if Cn==1;
}
//connect internal (not root switch) switches with each other
for Cn=floor((log(C)-log(2))/(log(m)-log(2)))..floor((log(C)-
log(2))/(log(m)-log(2))), for i=0..(Cn==1 ? -1 : (2*Cn-4)*(m/2)^(Cn-
1)-1), for j=0..(Cn==1 ? -1 : m/2-1), for k=i/(2*(m/2)^(Cn-
1))+1..i/(2*(m/2)^(Cn-1))+1, for q=((m/2)^(k-1))..((m/2)^(k-1)) {

    sw_ICN2[i].in[j+m/2] <-- phynet3 <-- sw_ICN2[(((i-
i%(m/2)^k)+2*(m/2)^(Cn-1))+(m/2)*(i%q))+j].out[(i%(m/2)^k)/q];
    sw_ICN2[i].out[j+m/2] --> phynet3 --> sw_ICN2[(((i-
i%(m/2)^k)+2*(m/2)^(Cn-1))+(m/2)*(i%q))+j].in[(i%(m/2)^k)/q];
}

//connect root switches with sub trees
for Cn=floor((log(C)-log(2))/(log(m)-log(2)))..floor((log(C)-
log(2))/(log(m)-log(2))), for i=(Cn==1 ? 0 : (2*Cn-4)*(m/2)^(Cn-
1))..(Cn==1 ? -1 : (2*Cn-3)*(m/2)^(Cn-1)-1), for j=0..(Cn==1 ? -1 :
m/2-1), for k=i/(2*(m/2)^(Cn-1))+1..i/(2*(m/2)^(Cn-1))+1, for
q=((m/2)^(k-1))..((m/2)^(k-1)) {

    //ICN2
    sw_ICN2[i].in[j+m/2] <-- phynet3 <-- sw_ICN2[(((i-
i%(m/2)^k)+2*(m/2)^(Cn-1))+(m/2)*(i%q))+j].out[(i%(m/2)^k)/q];
    sw_ICN2[i].out[j+m/2] --> phynet3 --> sw_ICN2[(((i-
i%(m/2)^k)+2*(m/2)^(Cn-1))+(m/2)*(i%q))+j].in[(i%(m/2)^k)/q];

    sw_ICN2[i+(m/2)^(Cn-1)].in[j+m/2] <-- phynet3 <--
sw_ICN2[(((i-i%(m/2)^k)+2*(m/2)^(Cn-
1))+(m/2)*(i%q))+j].out[(i%(m/2)^k)/q+m/2];
    sw_ICN2[i+(m/2)^(Cn-1)].out[j+m/2] --> phynet3 -->
sw_ICN2[(((i-i%(m/2)^k)+2*(m/2)^(Cn-
1))+(m/2)*(i%q))+j].in[(i%(m/2)^k)/q+m/2];
}
}
}

```

c) Routing modules

a. Routing module Fifo

```
//-----
// file: fifo1.cc
//      (part of Fifo1 - an OMNeT++ demo simulation)
//-----

#include "fifo.h"
#include <math.h>
#include "gensinkMsg_m.h"

void FF1AbstractFifo::initialize()
{
    msgServiced = NULL;
    endServiceMsg = new cMessage("end-service");
    cModule *parent;
    parent = getParentModule();
    int i = parent->getIndex();
    ev << "index= " << i << endl;
}

void FF1AbstractFifo::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);
    gensinkMsg *msgServiced = dynamic_cast<gensinkMsg *>(msg);

    if(gmsg!=NULL){

        src_ix = gmsg->getSrc();
        des_ix = gmsg->getDest();

        if (gmsg->getKind()==1)
            NCA_NO[src_ix][des_ix] = 0;

        Else

            NCA_NO[src_ix][des_ix] = gmsg->getNCA();
            ev << "Merger: NCA of msg " << msg->getName() << " is : "
<< NCA_NO[src_ix][des_ix] << endl;
    }

    if (msg==endServiceMsg)
    {

        src_ix = msgServiced->getSrc();
        des_ix = msgServiced->getDest();
        out_ix = NCA_NO[src_ix][des_ix];

        endService( msgServiced ,out_ix);

        if (queue.empty())
```

```

        {
            msgServiced = NULL;
        }
        else
        {
            opp_warning("FFlAbstractFifo::handleMessage: Src3");
            msgServiced = (gensinkMsg *) queue.pop();
            simtime_t serviceTime = startService( msgServiced );
            scheduleAt( simTime()+serviceTime, endServiceMsg );
        }
    }
    else if (!msgServiced)
    {
        arrival( msg );
        msgServiced = gmsg;

        simtime_t serviceTime = startService( msgServiced )
        scheduleAt( simTime()+serviceTime, endServiceMsg );

    }
    else
    {
        arrival( msg );
        queue.insert( msg );
    }
}

void FFlAbstractFifo::finish()
{
    int j;
    cModule *parent = getParentModule();
    int i = parent->getIndex();
    int s = parent->size();
}

//-----

Define_Module( Fifo );

simtime_t Fifo::startService(cMessage *msg)
{
    ev << "Starting service of " << msg->getName() << endl;
    return par("service_time");
}

void Fifo::endService(cMessage *msg, int ix)
{
    send( msg, "out", ix);
}

```

b. Routing module infifo for Internal Cluster

```

//-----
// file: infifo.cc

#include "infifo.h"
#include <math.h>
#include "gensinkMsg_m.h"

void FF1AbstractinFifo::initialize()
{
    msgServiced = NULL;
    endServiceMsg = new cMessage("end-service");
    intCountEntry = 0;
}

void FF1AbstractinFifo::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);
}

simtime_t d = simTime()-msg->getTimestamp();

if (gmsg!=NULL){

    if (gmsg->getRouted()==0) {

        ev << "Unrouted header flit; send to router" << endl;
        endService(msg, 0);
        return;
    }
    gmsg->setRouted(0);
}

if (msg==endServiceMsg)
{
    endService( msgServiced ,1);
    if (queue.empty())
    {
        msgServiced = NULL;
    }
    else
    {
        msgServiced = (gensinkMsg *) queue.pop();
        simtime_t serviceTime = startService( msgServiced );

        scheduleAt( simTime()+serviceTime, endServiceMsg );
    }
}

else if (!msgServiced)
{
    arrival( msg );
    msgServiced = gmsg;
    simtime_t serviceTime = startService( msgServiced );
}

```

```

        scheduleAt( simTime()+serviceTime, endServiceMsg );
    }
    else
    {
        arrival( msg );
        queue.insert( msg );
    }
}

//-----

Define_Module( inFifo );

simtime_t inFifo::startService(cMessage *msg)
{
    ev << "Starting service of " << msg->getName() << endl;
    return par("service_time");
}

void inFifo::endService(cMessage *msg, int ix)
{
    send( msg, "out", ix);
}

```

c. Routing Module efifo for External Cluster

```

//-----
// file: efifo.cc

#include "efifo.h"
#include <math.h>
#include "gensinkMsg_m.h"

void FF1AbstracteFifo::initialize()
{
    msgServiced = NULL;
    endServiceMsg = new cMessage("end-service");
}

void FF1AbstracteFifo::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);
    simtime_t d = simTime()-msg->getTimestamp();

    if (gmsg!=NULL){
        if (gmsg->getRouted()==0){

            ev << "Unrouted header flit; send to router" << endl;
            endService(msg, 0);
        }
    }
}

```

```

        return;
    }
    gmsg->setRouted(0);
}

if (msg==endServiceMsg)
{
    endService( msgServiced ,1);
    if (queue.empty())
    {
        msgServiced = NULL;
    }
    else
    {
        msgServiced = (gensinkMsg *) queue.pop();
        simtime_t serviceTime = startService( msgServiced );
        scheduleAt( simTime()+serviceTime, endServiceMsg );
    }
}
else if (!msgServiced)
{
    arrival( msg );
    msgServiced = gmsg;
    simtime_t serviceTime = startService( msgServiced );
    scheduleAt( simTime()+serviceTime, endServiceMsg );
}
else
{
    arrival( msg );
    queue.insert( msg );
}
}

//-----

Define_Module( eFifo );

simtime_t eFifo::startService(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);

    ev << "Starting service of " << gmsg->getName() << endl;
    return par("service_time");
}

void eFifo::endService(cMessage *msg, int ix)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);
    send( gmsg, "out", ix);
}

```

d) Communication Switch module

```

//-----
// file: switch.cc
//-----
#include "switch.h"
#include <math.h>
#include "omnetpp.h"
#include "gensinkMsg_m.h"

void FF1AbstractCB::initialize()
{
    int cross_delay = par("cross_delay");
}

void FF1AbstractCB::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);

    if (gmsg)
        out_port = gmsg->getOut_port();
        ev << "CrossBar: Cross msg to port: " << out_port << endl;

        if(gate("out",out_port)->getTransmissionChannel()!=NULL)
        {
            if(gate("out",out_port)->getTransmissionChannel()->isBusy())
            {
                sendDelayed(gmsg,gate("out",out_port)-
>getTransmissionChannel()->getTransmissionFinishTime()-
simTime(),"out",out_port);
            }
            else
            {
                send(gmsg,"out", out_port);
            }
        }
    }
}

Define_Module( CrossBar );

simtime_t CrossBar::startService(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);

    ev << "Starting service of " << gmsg->getName() << endl;
    return par("service_time");
}

void CrossBar::endService(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);

    int dest, src ,n;
    dest = gmsg->par("dest");
    src = gmsg->par("src");
    dest = dest - (dest/n)*n;
    ev << "Completed service of " << gmsg->getName() << " in ICN1, send
to port: "<< dest << endl;
    send( gmsg, "out", dest);
}

```

e) Message-Generator module

```
// File: gen.cc
//
// Implementation of simple module types
//-----

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <string.h>
#include "omnetpp.h"
#include "gen.h"
#include "gensinkMsg_m.h"

// Turn on code that prints debug messages
#define TRACE_MSG

// Module registration:
Define_Module( Generator );

double Latency[ARY_SIZE][BATCH_NO];
int BATCH_SIZE;

void Generator::initialize()
{
    msg_cnt = 0;
    batch_no = 0;
    measurment = false;
    num_messages_measur = par("num_messages");
    msg_length = par("msg_length");
    fp_length = par("fp_length");
    my_address = par("address");
    ia_time = par("ia_time").doubleValue();

    m = par("num_ports");
    n = par("num_trees");
    C = par("num_clusters");

    num_nodes = 2*(int)pow(m/2, n);
    parent = getParentModule();
    ix = parent->getIndex();

    if (ix==0)
    ev << "Number of Nodes: " << C*num_nodes << endl;
    dix = ix /num_nodes;

    if (dix<C/2)
    fi[dix] = 1.0;
    else
    fi[dix] = 1.0;

    ev << "Pr index: " << ix << ",fi: " << 1/fi[dix] << endl;

    BATCH_SIZE = num_messages_measur/BATCH_NO;
    num_messages_warm = WARM_UP*BATCH_SIZE;
    num_messages_drain=DRAIN*BATCH_SIZE;
    sendMessageEvent = new cMessage("sendMessageEvent");
    scheduleAt(0.0, sendMessageEvent);
}
```



```

}
//
// Activities of the simple modules
//
void Generator::handleMessage(cMessage *msg)
{
    int dest = intrand(C*num_nodes);
    if (dest == my_address)
    {
        scheduleAt(simTime(), sendMessageEvent);
        return;
    }
    sprintf(msgname, "%d-->%d", my_address, dest);

    // generate packet

    gensinkMsg *gmsg = new gensinkMsg(msgname);
    gmsg->setSrc(my_address);
    gmsg->setDest(dest);
    gmsg->setNum_nodes(num_nodes);
    gmsg->setRouted(0);

    if (measurment)
    gmsg->setBatch_no(batch_no);
    gmsg->setLevel(n-1);
    gmsg->setOut_port(-1);
    gmsg->setBitLength(fp_length*msg_length);
    gmsg->setTimestamp();
    ev << "Generate data packet\n";

#ifdef TRACE_MSG
        ev.printf("gen[%d]: Generated new msg: '%s ' by node:
%d\n", my_address, gmsg->getName(), ix);
#endif

    ev << "Send packet to Source Queue (Channel)" << endl;

    if ((int)(my_address/num_nodes) == (int)(dest/num_nodes)){
    // internal

        ev << "Send to internal Cluster" << endl;
        gmsg->setKind(0);
        if (gate("out",0)->getTransmissionChannel()->isBusy())
            sendDelayed(gmsg, gate("out",0)-
>getTransmissionChannel()->getTransmissionFinishTime()-
simTime(), "out", 0);
        else
            send(gmsg, "out", 0);
    }
    else{
    // external
        ev << "Send to external Cluster" << endl;
        gmsg->setKind(1);
        if (gate("out",1)->getTransmissionChannel()->isBusy())
        {
            sendDelayed(gmsg, gate("out",1)->getTransmissionChannel()-
>getTransmissionFinishTime()-simTime(), "out", 1);
        }
        else

```

```

    {
        send(gmsg, "out", 1);
    }
}
msg_cnt++;
if (msg_cnt>=num_messages_warm)
    measurment = true;

if (msg_cnt>=num_messages_measur+num_messages_warm)
    measurment = false;

if (msg_cnt>=(batch_no+2)*BATCH_SIZE && measurment)
    batch_no++;

if (msg_cnt
num_messages_warm+num_messages_measur+num_messages_drain)

scheduleAt(simTime()+(double)exponential((double)ia_time*fi[dix]),
sendMessageEvent);
}
void Generator::finish()
{
    int m = par("num_ports");
    int n = par("num_trees");
    int C = par("num_clusters");
    int i;
    double iicnt, eecnt, in, en;
}
}

```

f) Message-Sink module

```

// File: sink.cc
//
// Implementation of simple module types
//-----

#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <string.h>
#include "omnetpp.h"
#include "sink.h"
#include "gensinkMsg_m.h"

#define TRACE_MSG

// Module registration:

Define_Module( Sink );

void Sink::initialize()
{
    parent = getParentModule();
    i = parent->getIndex();
    s = parent->size();
    j = 0;
    Latency[i][j] = 0;
    sinked_msg_cnt = 0;
}

```

```

}
void Sink::handleMessage(cMessage *msg)
{
    gensinkMsg *gmsg = dynamic_cast<gensinkMsg *>(msg);
    simtime_t d = simTime()-msg->getTimestamp();
    if (gmsg!=NULL){
        j = gmsg->getBatch_no();
        sinked_msg_cnt++;

        if (j<0 || j>BATCH_NO)
            error("error in batch_no: %simTime",gmsg->getBatch_no());
        qstats[j].collect( d );
    }

    if ((int)par("address")!= gmsg->getDest()){
        error ("error in destination, my addr:%d !=
dest:%d", (int)par("address"),gmsg->getDest());
    }
    // message no longer needed
    delete msg;
}

void Sink::finish()
{
    cModule *parent = getParentModule();
    double avg_latency[BATCH_NO];
    double latency1;
    int j,k;
    int ix = parent->getIndex();
    int s = parent->size();

    for(j=0; j<BATCH_NO; j++){
        Latency[ix][j]= qstats[j].getMean();
    }
    msg_cnt = 0;
    for(j=0; j<BATCH_NO; j++)
        msg_cnt+= qstats[j].getCount();

    if (ix==s-1){
        for(j=0; j<BATCH_NO; j++){
            avg_latency[j] = 0;

            for (k=0; k<s; k++)
                avg_latency[j]+= Latency[k][j];
            avg_latency[j] = avg_latency[j]/s;
            ev << "Avg Latency: " << avg_latency[j] << " Batch: " <<j<<
endl;
        }
    }
    // count avg latency
    latency1 = 0;
    for(j=0; j<BATCH_NO; j++)
        latency1+= avg_latency[j];

    latency1 = latency1/(BATCH_NO);
    ev << "Total Avg Latency (BATCH): " << latency1 << endl;
    recordScalar("TotalAvgLatency", latency1);
}

```

c) Simulation Test Plan

In order to illustrate the feasibility and the accuracy of the simulation model, four sets of experiments were conducted and compared with previous research, using several system configurations.

1. Baseline Experiments

Baseline experiment involved experiments with a single-core cluster and multi-core cluster. The experiments were conducted using the MCMCA simulation model, and the MCMCA analytical model for model validation compared to Javadi's model (Javadi et al., 2006) using the same interconnection network parameters from their paper.

a) Experiments with a Single-core Cluster

This section presents the results of experiments based on Javadi's model (Javadi et al., 2006) conducted on single-core clusters based on the interconnection network parameters listed in Table 2 and model cases in Table 3.

Table 2: Interconnection network parameters

Parameter	Internal-cluster	External-cluster
Network latency	0.01 s	0.02 s
Switch latency	0.01 s	0.01 s
Network Bandwidth	1000 bits/s	500 bits/s

Table 3: Model cases for single-core clusters

C,m,n	Message Length (M)	Flit length (F)
8,8,2	32 flits	256 bytes
8,8,2	32 flits	512 bytes
8,8,2	64 flits	256 bytes
8,8,2	64 flits	512 bytes

b) Experiments with Multi-core Clusters

To test the validity of the simulation of the MCMCA, a simulation experiment was performed based on model cases in Table 4. Two different flow control mechanisms, store-and-forward and wormhole, were used to verify the simulation model.

Table 4: Model cases for multi-core clusters

Items	Quantity
No. of cores nc	1, 2, 4
Message Length M and Flit Length F	32 flits, 256 bytes
No. of clusters, m -port, n -tree	8, 8, 2

2. Latency and Throughput Performance Experiments on MCMCA

a) Store-and-Forward Flow Control

To investigate the latency effect on MCMCA, the simulation experiments involved a single-core processor, a dual-core processor and a quad-core processor. To form the latency curve, a total of 10 different message generation rates λ_g were used for each core, and the accuracy of each result was validated by the analytical calculation. The simulation experiments were performed with various combinations of parameters by using the interconnection network parameter I given in Table 5 with simulation input I from Table 6.

Table 5: Interconnection Networks Parameter I

Parameter	Internal cluster	External cluster
Network latency	0.01 s	0.02 s
Switch latency	0.01 s	0.01 s
Network Bandwidth I	1000 bits/s	500 bits/s
Network Bandwidth II	800 bits/s	600 bits/s

Table 6: Simulation Input I

Items	Message length M 8KB
No. of cores nc	1, 2, 4
No. of cluster C	8
No. of m -port n -tree	8, 2

b) Wormhole Flow Control

Three different numbers of cores in a processor were analysed. The simulation parameters are based on Table 7 and Table 8.

Table 7: Interconnection Networks Parameters

Parameter	Internal-cluster	External-cluster
Network latency	0.01 s	0.02 s
Switch latency	0.01 s	0.01 s
Network Bandwidth	1000 bits/s	500 bits/s

Table 8: Input Parameters

Items	Flit length F 256 bytes	Flit length F 512 bytes
No. Of cores nc	1, 2, 4	
No. Of cluster C	8	
No. Of m -port n -tree	8, 2	
Message Length M	32 flits	

3. The Impact on Cluster Size Experiments

a) Store-and-Forward Flow Control

These simulation experiments were designed to get more insight into the impact of the communication latency on the cluster size. The results are based on the interconnection network parameters in Table 9, and were compared with a higher network latency and a smaller bandwidth setup in the simulation experiments in Table 10.

Table 9: Interconnection Networks Parameter I

Parameter	Internal cluster	External cluster
Network latency	0.01 s	0.02 s
Switch latency	0.01 s	0.01 s
Network Bandwidth I	1000 bits/s	500 bits/s

Table 10: Interconnection Networks Parameter II

Parameter	Internal-cluster	External-cluster
Network latency	0.02 s	0.01 s
Switch latency	0.01 s	0.05 s
Network Bandwidth	800 bits/s	600 bits/s

b) Wormhole Flow Control

The accuracy of the simulation model has been validated as being a cost-effective tool to investigate the interconnection network performance in MCMCA. It can thus be used to get an insight into the impact of cluster size on maximising network performance. A set of simulation experiments were conducted using the simulation input in Table 11.

Table 11: Simulation Input for cluster sizes

Items	Quantity
No. of cluster C	8, 16, 32, 64, 128
No. of cores nc	1, 2, 4
Message generation rate λg	0.001 s, 0.002 s
Message Length M	18 KB, 16 KB
No. of m -port n -tree	4, 2

4. The Impact on Message Length and Scalability Experiments

a) Store-and-Forward Flow Control

In this experiment, to examine the potential scalability in the cluster architecture, different message lengths were run, as reflected in Table 12 and Table 13.

Table 12: Simulation Input II

Items	Quantity
No. of cluster C	8, 16, 32, 64, 128
No. of cores nc	1, 2, 4
Message generation rate λg	0.002 s
Message Length M	8 KB
No. of m -port n -tree	4, 2

Table 13: Simulation Input III

Items	Quantity
No. of cores nc	1, 2, 4
Message generation rate λg	0.001 s
Message Length $M/bytes$	128 B, 256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB
No. of cluster, m -port n -tree	8, 8, 2

b) Wormhole Flow Control

Testing the various impacts on message length gave insights into scalability. Seven message lengths ($M/bytes$) were involved in an eight multi-core cluster system, as in Table 14, with the numbers of cores being 1, 2 and 4. To make this experiment comparable with others, the same message length was also been tested in larger cluster sizes: 32 multi-core cluster systems with 4, 8 and 16 cores. The same message generation rate λg was used to maintain the validity of the results.

Table 14: Simulation Input for scalability

Items	8-cluster	32-cluster
No. of cores nc	1, 2, 4	4, 8, 16
No. of m -port n -tree	8, 2	
Message generation rate λg	0.001 s	
Message Length $M/bytes$	128 B, 256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB	

Appendix 3-B

Figure 1 shows the OMNeT++ Integrated Development Environment (IDE) and workspace to run the simulations. The IDE is based on the Eclipse platform, and extends it with new editors, views, wizards, and additional functionality. OMNeT++ adds functionality for creating and configuring models (NED and ini files), performing batch executions, and analyzing simulation results, while Eclipse provides C++ editing and other optional features such as UML modelling, bugtracker integration and database access through various open-source and commercial plug-ins.

The NED Editor can edit NED files both graphically and in text mode, the user switching between the two modes at any time, using the tabs at the bottom of the editor window. The list of NED modules are previewed on the left side of the workspace.

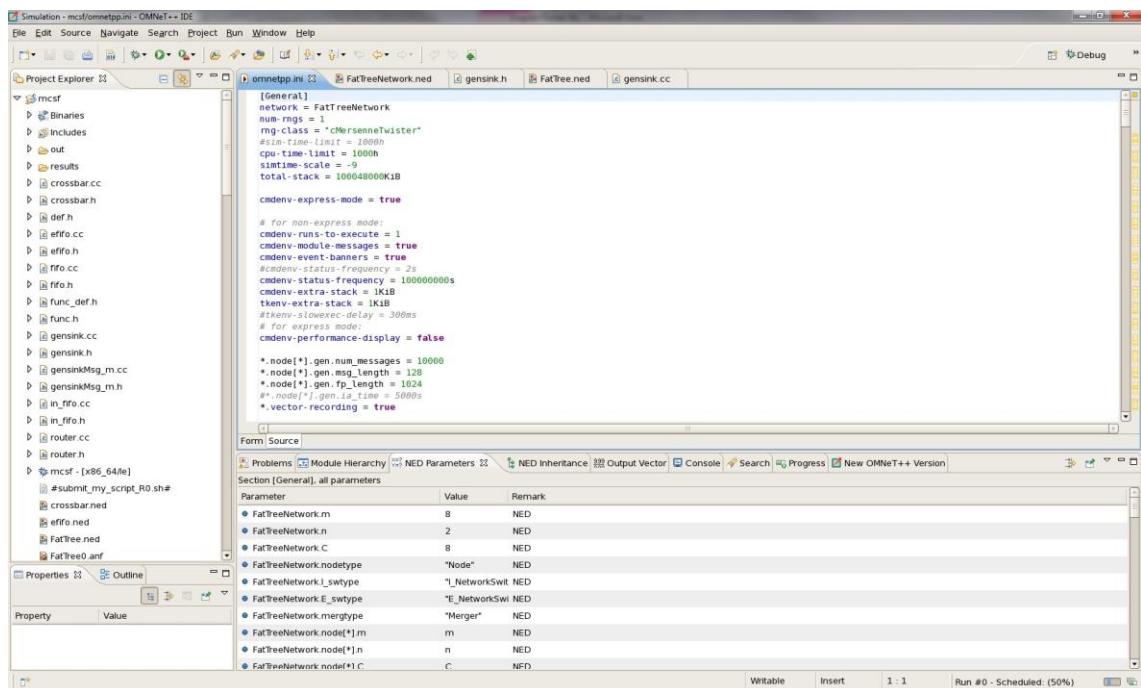
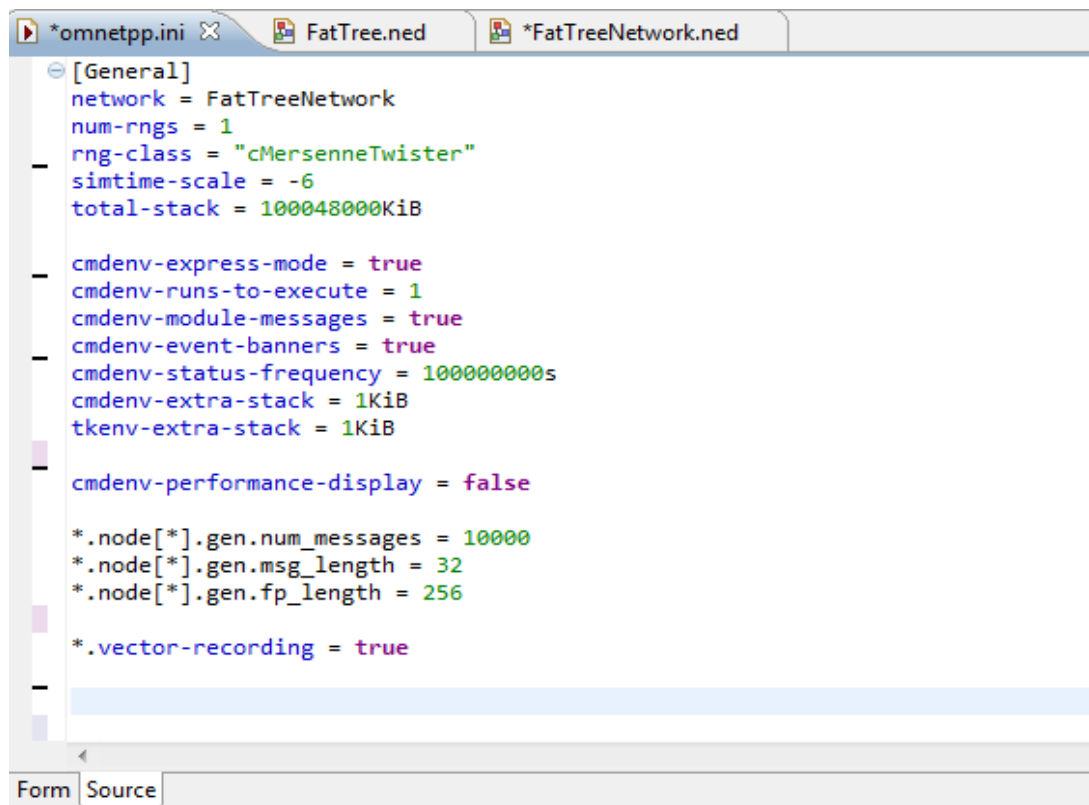


Figure 1: The OMNeT++ Integrated Development Environment and workspace

At the start of each execution, the simulator reads the initialisation file omnetpp.ini (see Figure 2) that tells the program which network file is to be simulated. Several networks can be listed in the same simulation program. omnetpp.ini will pass the parameters to the models, for example, it explicitly specifies seeds for the random number generators. The parameters are defined in the Network Topology Module (FattreeNetwork.ned) shown in Figure 3. In this initialisation file, the parameters of the model, such as Number of clusters C , parameter of m -port n -tree, message length M ,

number of cores nc , number of nodes N and λ , are specified. The simulation can also be run with different initialisation inputs and all the values can be stored in an initialisation file containing settings that control how the simulation is executed.



```

[General]
network = FatTreeNetwork
num-rngs = 1
rng-class = "cMersenneTwister"
simtime-scale = -6
total-stack = 100048000KiB

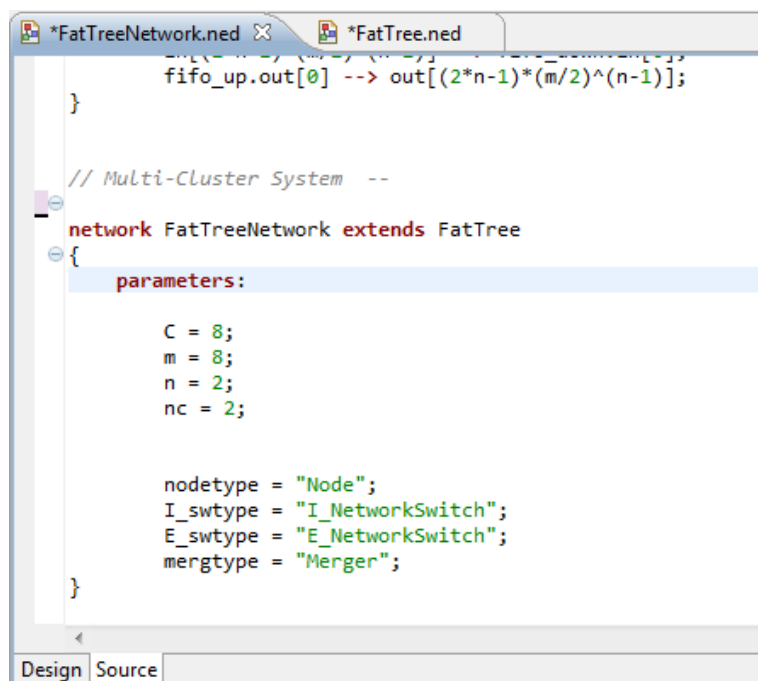
cmdenv-express-mode = true
cmdenv-runs-to-execute = 1
cmdenv-module-messages = true
cmdenv-event-banners = true
cmdenv-status-frequency = 100000000s
cmdenv-extra-stack = 1KiB
tkenv-extra-stack = 1KiB

cmdenv-performance-display = false

*.node[*].gen.num_messages = 10000
*.node[*].gen.msg_length = 32
*.node[*].gen.fp_length = 256

*.vector-recording = true
  
```

Figure 2: omnetpp.ini file source



```

}
fifo_up.out[0] --> out[(2*n-1)*(m/2)^(n-1)];

// Multi-Cluster System --
network FatTreeNetwork extends FatTree
{
    parameters:

        C = 8;
        m = 8;
        n = 2;
        nc = 2;

        nodetype = "Node";
        I_swtype = "I_NetworkSwitch";
        E_swtype = "E_NetworkSwitch";
        mergtype = "Merger";
}
  
```

Figure 3: Network Topology Module

To run the simulation from the IDE, *Run Configurations* is selected from the menu (Figure 4). OMNeT++ simulations can be run under different user interfaces. Currently the following user interfaces are supported:

- Tkenv: the traditional, Tcl/Tk-based graphical user interface
- Qtenv: the new, Qt-based graphical user interface
- Cmdenv: command-line user interface for batch execution

By default, Tkenv will be used if both runtime environments are present. The user interface may be selected by adding the `user-interface=Cmdenv` (or `=Tkenv`) option to the initialisation file, or by specifying `-u Cmdenv` or `-u Tkenv` to the command line. If both the configuration option and the command line option are present, the command line option takes precedence. The Run button is used to start the simulation, and the topology of m-port n-tree will be formed, as shown in Figure 5.

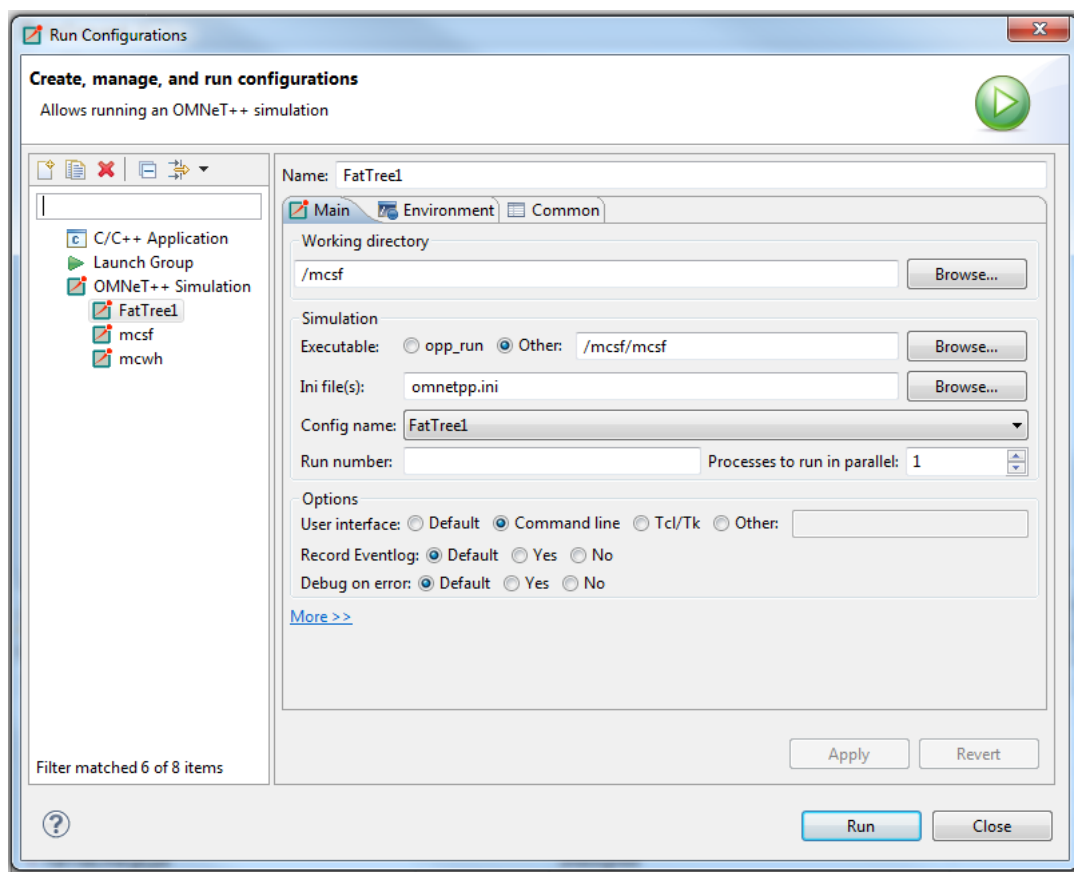


Figure 4: The *Run Configurations* Menu to start the simulation

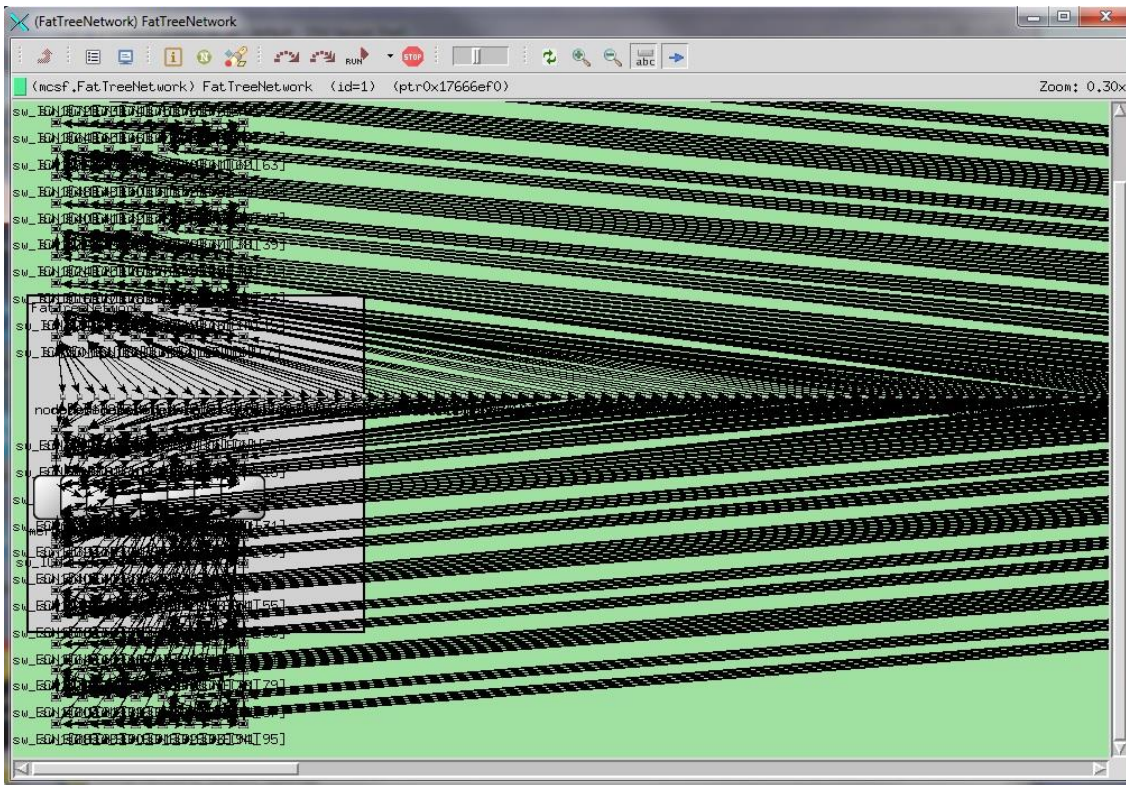


Figure 5: The network topology m -port n -tree

After initialising the input parameters, the display shows how the messages hop from module to module following the routing algorithm (see Figure 5). The Message-Generator file will partition each message into a sequence of packets first, before being generated at each tree-node, following the assumptions that the message destinations are uniformly distributed by using a uniform random number generator. In order to get the message to its destination, a packet will access the processor through a chip, and a chip can contain one or more processors. Processor n will divide the packets into the number of cores. If processor n is busy, it will pass the packets to another processor in the same chip within the same node first, before determining, via the communication network, if other processors in other chips can process the packets. Packets will access the processors, chips and nodes in its cluster first, before accessing other clusters via the communication network.

The main windows toolbar displays the simulated time (Red box in Figure 6). The simulated time is virtual time and is not based on the actual elapsed time that the program takes to execute. For this research, the simulation time is the propagation delay on the connections.

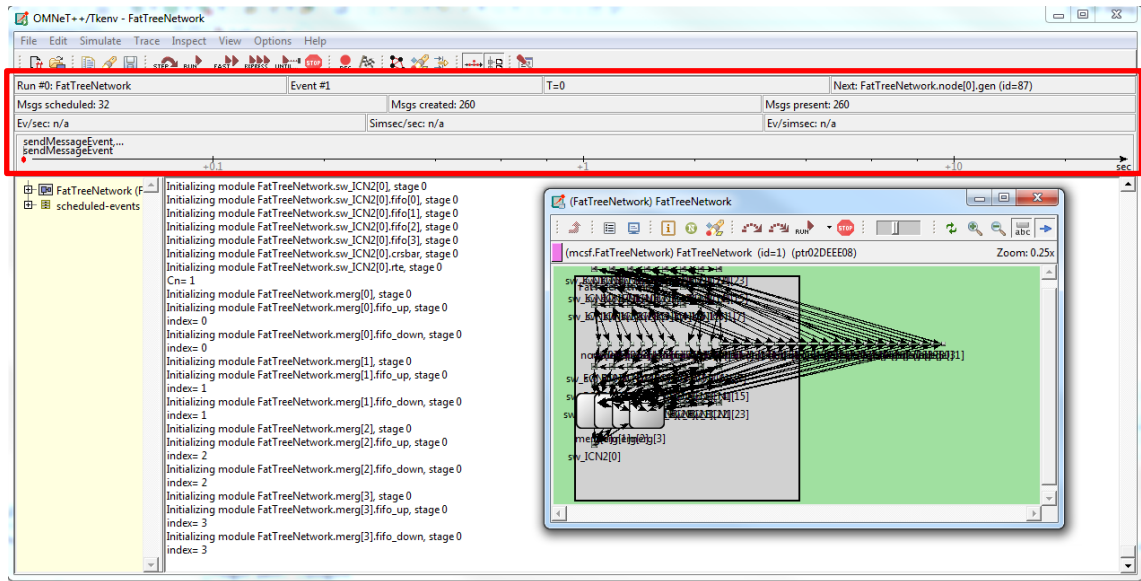


Figure 6: Simulation running environment

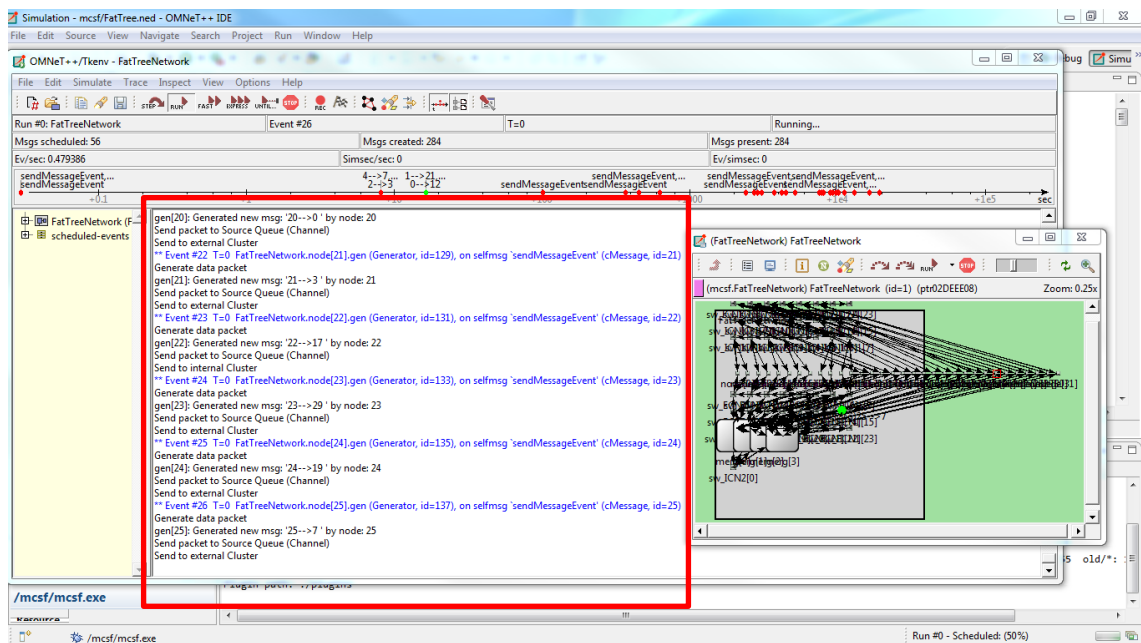


Figure 7: The simulation events status window

The routing file will determine the path the packets will follow in the network from the source to the destination in the simulation events status window (Red box in Figure 7). Based on the path given, the packets will hop on the communication switch to get through the communication network. If there is a situation where more than one packet needs to use the same route, the communication switch will determine which packet can go through first or whether the packet needs to be queued (buffered) until the route is available. Each packet is time-stamped after its generation and the message completion time is defined in the Message-Sink module on each tree-node to compute message latency. The Message-Sink file receives a message, tests the packet

type, prints a message and then deletes the packet before gathering the statistics for every event in the simulation for analysis of the results.

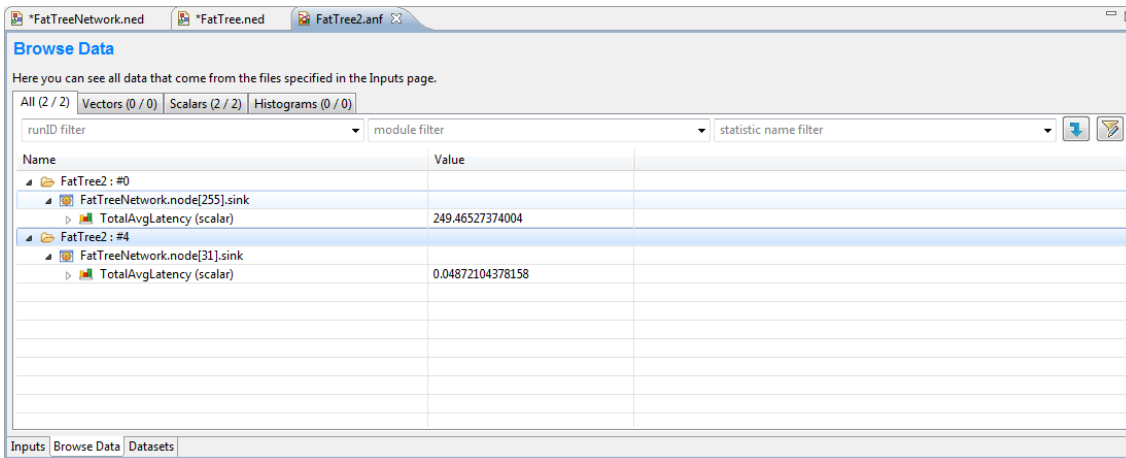


Figure 8: Inputs page of the simulation events

Messages about the events and the type of messages generated and received are displayed in another window. Simulation results are recorded into *output scalar files* that actually hold statistics results as well, and *output vector files*. The usual file extension for scalar files is .sca, and for vector files .vec, as shown in Figure 8. The eventlog file is created automatically during a simulation run upon explicit request, configurable in the ini file. The resulting file can be viewed in the IDE using the Sequence Chart and the Eventlog Table, or can be processed by the command line Eventlog Tool. The result produced by the simulation is the average message latency based on traffic generation rate, which has been defined earlier in the initialisation file.

Appendix 4-A

In order to illustrate the feasibility and the accuracy of the simulation model, a set of experiments were conducted using several system configurations, as listed in Table 1 and Table 2, to illustrate the individual behaviour of selected analytical model calculations. Two different flow control mechanisms were used to investigate the impact on interconnection network performance.

Table 1: Simulation input

Items	Quantity
No. of cores nc	1
Message Length M and Flit Length F	32 flits, 256 bytes
No. of cluster, m -port n -tree	8, 8, 2

Table 2: Interconnection network parameters

Parameter	Internal-cluster	External-cluster
Network latency	0.01 s	0.02 s
Switch latency	0.01 s	0.01 s
Network Bandwidth	1000 bits/s	500 bits/s

a) Average Waiting Time at the Source Node in Internal Cluster

Average waiting time at the source node in internal-cluster is the average time the packets will be in a queue while waiting to be transmitted into the network. It can be computed by:

$$W_i = \frac{(\beta_i)^2 \lambda_i}{2(1 - \beta_i \lambda_i)}$$

Where $\beta_i = 8.202$ and $\lambda_i = 0.000122$

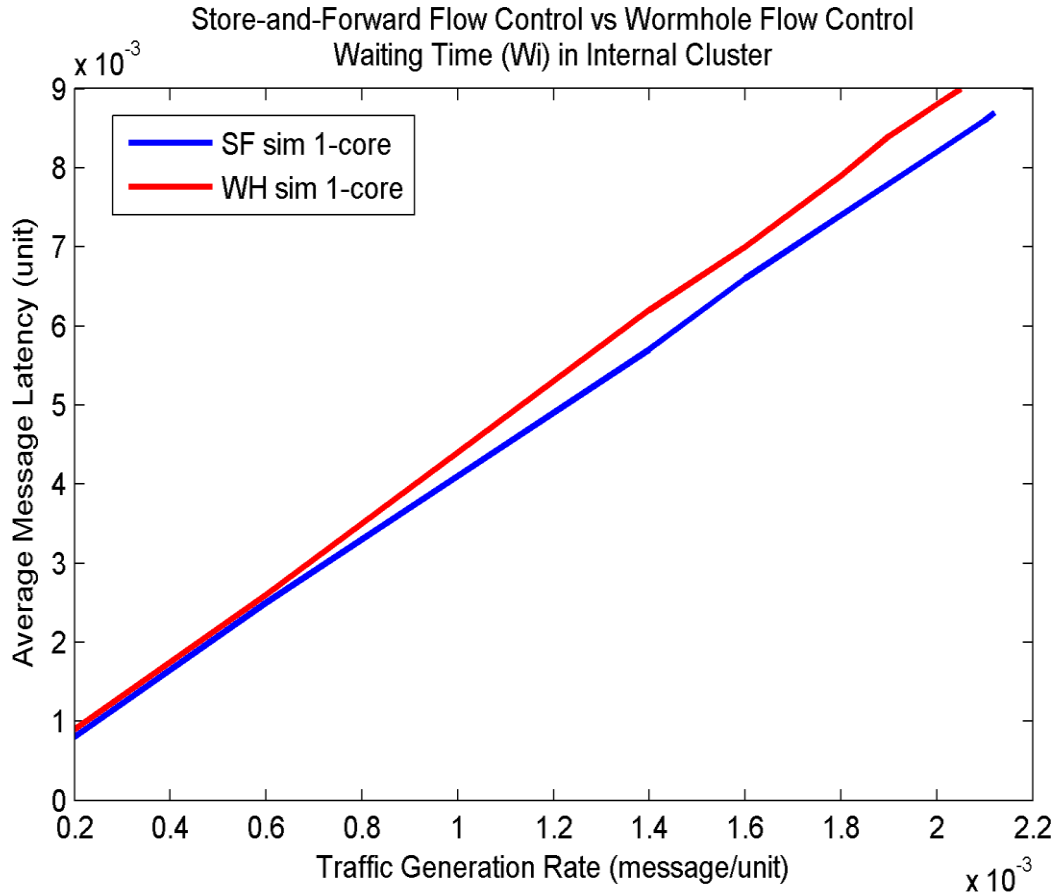


Figure 1: Average Waiting Time at the Source Node in Internal-Cluster

Figure 1 shows the average waiting time at the source node in an internal-cluster, based on two different flow controls: store-and-forward flow control and wormhole flow control. As the traffic generation rates increases, the load allocation time increases, which can affect the pipelining of the router. The zero-load latency tends to increase and may slightly decrease the saturation throughput in the internal-cluster because of the additional time required to allocate the task into a number of cores. The zero-load latency assumption is that a packet has never contended for network resources with other packets which applied to store-and-forward flow control. It gives a lower bound on the average latency of a packet through the network. Compared to wormhole flow control, the average message latency is higher because multiple messages can be in transmission and attempt to use the same network link at the same time. If this problem occurs, some of the messages must be blocked while other messages are allowed to proceed, which affects the performance.

b) Average Waiting Time at the Source Node in Internal Cluster

Approximations of packet latency, to predict the average amount of time that a packet spends waiting in each queue in the external-cluster, is given by:

$$W_e = \frac{(\beta_e)^2 \lambda_e}{2(1 - \beta_e \lambda_e)}$$

Where $\beta_e = 16.394$ and $\lambda_e = 0.0018$

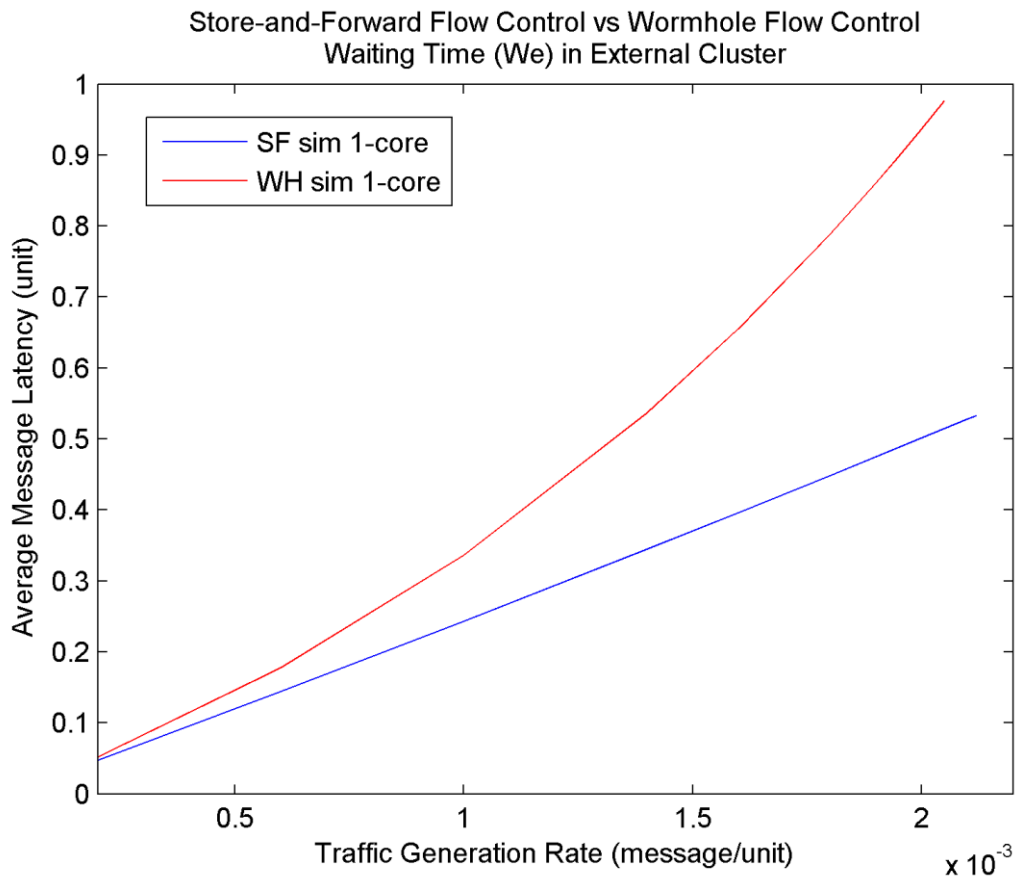


Figure 2: Average Waiting Time at the Source Node in External-Cluster

For the same traffic rate, both flow control mechanisms achieve lower latency in light traffic, but the latency increases massively with wormhole flow control due to blocking probability happening in transmission, as shown in Figure 2. The results show that both flow controls perform better in light traffic, but wormhole flow control does not achieve optimal performance throughput compared to store-and-forward flow control.

References

- Abad, P., Puente, V., & Gregorio, J. A. (2012). Balancing Performance and Cost in CMP Interconnection Networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(3), 452-459. doi: 10.1109/tpds.2011.173
- Abawajy, J. H., & Dandamudi, S. P. (2003, 1-4 Dec. 2003). *Parallel job scheduling on Multi-cluster Computing Systems*. Paper presented at the Proceedings of the IEEE International Conference on Cluster Computing, (CLUSTER'03), Hong Kong.
- Abdelgadir, A. T., Pathan, A.-S. K., & Ahmed, M. (2011). On the Performance of MPI-OpenMP on a 12 nodes Multi-core Cluster *In Algorithms and Architectures for Parallel Processing* (pp. 225-234): Springer Berlin Heidelberg.
- Admin. (1999). IBM Cluster Systems. *Drive performance while reducing energy and space with integrated cluster solutions*. Retrieved Mei 2012, from <http://www-03.ibm.com/systems/clusters/>
- Admin. (2013). The Iridis Compute Cluster. Retrieved Nov 2013, from <http://www.southampton.ac.uk/isolutions/computing/hpc/iridis/>
- Admin. (2014). Top500 Supercomputer Sites. Retrieved 04/11/2014, from <http://www.top500.org/lists/2014/06/>
- Akhter, S., & Roberts, J. (2006). *Multi-Core Programming* (Vol. 33). Hillsboro: Intel Press.
- Al-Babtain, B. M., Al-Kanderi, F. J., Al-Fahad, M. F., & Ahmad, I. (2013). A Survey on Amdahl's Law Extension in Multicore Architectures. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 3(3), 30-46.
- Allen, P., & Bennett, K. (2010). *PASW statistics by SPSS: A practical guide, version 18.0*. Melbourne: Cengage Learning Australia.
- Alzeidi, N., Khonsari, A., Ould-Khaoua, M., & Mackenzie, L. (2007). A new approach to model virtual channels in interconnection networks. *Journal of Computer and System Sciences*, 73(8), 1121-1130. doi: 10.1016/j.jcss.2007.02.002
- Alzeidi, N., Ould-Khaoua, M., & Khonsari, A. (2008). A new general method to compute virtual channels occupancy probabilities in wormhole networks. *Journal of Computer and System Sciences*, 74(6), 1033-1042. doi: 10.1016/j.jcss.2007.07.006
- Attaway, S. (2013). *MATLAB* (Butterworth-Heinemann Ed. 3rd ed.). United States: Elsevier Inc.
- Baker, M., Apon, A., Buyya, R., & Jin, H. (2000). *Cluster Computing and Applications*. 36.
- Baker, M., & Buyya, R. (1999a). *Cluster Computing at a Glance*. School of Computer Science and Software Engineering, Monash University, Melbourne, Australia: Prentice Hall.
- Baker, M., & Buyya, R. (1999b). Cluster computing: the commodity supercomputer. 29(6), 551-576.

- Balci, O. (1994, 11-14 Dec. 1994). *Validation, verification, and testing techniques throughout the life cycle of a simulation study*. Paper presented at the Proceedings of the Winter Simulation Conference
- Baldassari, J. D., Kopec, C. L., Leshay, E. S., Truszkowski, W., & Finkel, D. (2005, 4-7 April 2005). *Autonomic Cluster Management System (ACMS): A Demonstration of Autonomic Principles at Work*. Paper presented at the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, (ECBS '05).
- Banks, C. (2008). What Is Modeling and Simulation? *Principles of Modeling and Simulation* (pp. 3-24): John Wiley & Sons, Inc.
- Banks, J. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Applications and Practice*
- Bethel, E. W., & Howison, M. (2012). Multi-core and many-core shared-memory parallel raycasting volume rendering optimization and tuning. *International Journal of High Performance Computing Applications*, 26(4), 399-412. doi: 10.1177/1094342012440466
- Brakmo, L. S., & Peterson, L. L. (1996). *Experiences with network simulation*. Paper presented at the Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Philadelphia, Pennsylvania, United States.
- Burger, T. W. (2005). Intel Multi-Core Processors: Quick Reference Guide. from <https://software.intel.com/en-us/articles/intel-multi-core-processors-quick-reference-guide>
- Buyya, R., Hai, J., & Cortes, T. (2002). Cluster Computing. 18, 5-8.
- Caliri, G. V. (2000). *Introduction to Analytical Modeling*. Paper presented at the Proceedings of the 26th International Computer Measurement Group Conference, Orlando, FL, USA.
- Carson, J. S., II. (2005, 4-7 Dec. 2005). *Introduction to modeling and simulation*. Paper presented at the Proceedings of the 36th conference on Winter simulation Winter Simulation Conference.
- Chan, S., Ling, T., & Aubanel, E. (2012). The impact of heterogeneous multi-core clusters on graph partitioning: an empirical study. *Cluster Computing*, 15(3), 281-302. doi: 10.1007/s10586-012-0229-4
- Chang, V., Walters, R., & Wills, G. (2013). Cloud Storage and Bioinformatics in a Private Cloud Deployment: Lessons for Data Intensive Research *Cloud Computing and Services Science* (Vol. 367, pp. 245-264): Springer International Publishing.
- Creel, M., & Goffe, W. L. (2007). Multi-core CPUs, Clusters, and Grid Computing: a Tutorial. 36.
- Dally, W. J., & Towles, B. P. (2004). *Principles and practices of interconnection networks*: Elsevier.
- Das, A. (2008). *NETWORKING 2008: Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*: Springer.

- Deng, J. D., & Purvis, M. K. (2011). Multi-core application performance optimization using a constrained tandem queueing model. *Journal of Network and Computer Applications*, 34(6), 1990-1996. doi: <http://dx.doi.org/10.1016/j.jnca.2011.07.004>
- Donald, J., & Martonosi, M. (2006). An Efficient, Practical Parallelization Methodology for Multicore Architecture Simulation. *Computer Architecture Letters*, 5(2), 14-14. doi: 10.1109/l-ca.2006.14
- Fengguang, S., Moore, S., & Dongarra, J. (2009, Aug. 31 2009-Sept. 4 2009). *Analytical modeling and optimization for affinity based thread scheduling on multicore systems*. Paper presented at the IEEE International Conference on Cluster Computing and Workshops (CLUSTER '09)
- Field, A. P. (2000). *Discovering Statistics using SPSS for Windows: Advanced Techniques for the Beginner*. London: Sage.
- Field, A. P. (2013). *Discovering Statistics using IBM SPSS Statistics*. London: Sage.
- Furhad, H., Haque, M. A., Kim, C.-H., & Kim, J.-M. (2013). An Analysis of Reducing Communication Delay in Network-on-Chip Interconnect Architecture. *Wireless Personal Communications*, 1-17. doi: 10.1007/s11277-013-1257-y
- Geer, D. (2005). Industry Trends: Chip Makers Turn to Multicore Processors, 38, 11-13.
- Geer, D. (2007). For Programmers, Multicore Chips Mean Multiple Challenges. *Computer*, 40(9), 17-19. doi: 10.1109/mc.2007.311
- Geyong, M., Yulei, W., Ould-Khaoua, M., Hao, Y., & Keqiu, L. (2009, Nov. 30 2009-Dec. 4 2009). *Performance Modelling and Analysis of Interconnection Networks with Spatio-Temporal Bursty Traffic*. Paper presented at the Proceedings of IEEE Global Telecommunication Conference (GLOBECOM 2009), Honolulu.
- Gomez, A., Bermudez, A., & Casado, R. (2011). A Deadlock-Free Dynamic Reconfiguration Scheme for Source Routing Networks Using Close Up*/Down* Graphs. *IEEE Transactions on Parallel and Distributed Systems*, 22(10), 1641-1652. doi: 10.1109/tpds.2011.79
- Gomez, C., Gilabert, F., Gomez, M. E., Lopez, P., & Duato, J. (2007, 26-30 March 2007). *Deterministic versus Adaptive Routing in Fat-Trees*. Paper presented at the IEEE International on Parallel and Distributed Processing Symposium (IPDPS 2007).
- Goscinski, A., Hobbs, M., & Silcock, J. (2001). A Cluster Operating System Supporting Parallel Computing. *Cluster Computing*, 4(2), 145-156. doi: 10.1023/a:1011473016546
- Gramsamer, F. (2003). *Scalable Flow Control for Interconnection Networks*. (Doctor of Technical Sciences), Swiss Federal Institute of Technology Zurich.
- Haddad, I. (2006). *The HAS Architecture: A Highly Available and Scalable Cluster Architecture for Web Servers*. (PhD), Concordia University, Library and Archives Canada.
- Hamid, N., Walters, R., & Wills, G. (2015a). An architecture for measuring network performance in multi-core multi-cluster architecture (MCMCA). *International Journal of Computer Theory and Engineering*, 7(1), 57-61.

- Hamid, N., Walters, R., & Wills, G. (2015b, 21-23 April 2015). *Interconnection network performance of multi-core cluster architectures*. Paper presented at the International Conference on Computer, Communications, and Control Technology (I4CT 2015), Kuching, Sarawak.
- Hamid, N., Walters, R., & Wills, G. (2015c). *Performance evaluation of multi-core multi-cluster architecture (MCMCA)*. In V. Chang, R. J. Walters & G. Wills (Eds.), *Delivery and Adoption of Cloud Computing Services in Contemporary Organizations*. Hershey, US: IGI Global.
- Hamid, N., Walters, R. J., & Wills, G. B. (2014, 03 - 05 Apr 2014). *Performance evaluation of multi-core multi-cluster architecture (MCMCA)*. Paper presented at the Proceedings of the Emerging Software as a Service and Analytics, Barcelona, ES.
- Holt, W. M. (2016, Jan. 31 2016-Feb. 4 2016). *1.1 Moore's law: A path going forward*. Paper presented at the IEEE International Solid-State Circuits Conference (ISSCC 2016).
- Hope, L., & Lam, E. (n.d.). A Review of Applications of Cluster Computing. *World*, 1-10.
- Ichikawa, S., & Kawai, Y. (2008, 19-21 Nov. 2008). *Constructing execution-time estimation models from diverse processing elements of heterogeneous clusters*. Paper presented at the Proceedings of the IEEE Region 10 Conference (TENCON 2008).
- Ichikawa, S., & Takagi, S. (2009, 16-19 March 2009). *Estimating the Optimal Configuration of a Multi-Core Cluster: A Preliminary Study*. Paper presented at the Proceedings of the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '09)
- Intel. (1997). Moore's Law and Intel Innovation. <http://www.intel.com/about/companyinfo/museum/exhibits/moore.htm?wapkw=moore+laws>
- Intel. (2015). 50 Years of Moore's Law. <http://www.intel.in/content/www/in/en/silicon-innovations/moores-law-technology.html>
- Jagannatham, A. (2008). Mersenne Twister-A Pseudo Random Number Generator and its variants. *George Mason University, Department of Electrical and Computer Engineering*.
- Javadi, Akbari, M. K., & Abawajy, J. H. (2006). A performance model for analysis of heterogeneous multi-cluster systems. *Parallel Computing*, 32(11-12), 831-851. doi: 10.1016/j.parco.2006.09.006
- Javadi, B., Abawajy, J. H., & Akbari, M. K. (2006). *Modeling and analysis of heterogeneous loosely-coupled distributed systems Technical Report TR C06/1*. Australia, : School of Information Technology, Deakin University.
- Javadi, B., Abawajy, J. H., & Akbari, M. K. (2008a). A comprehensive analytical model of interconnection networks in large-scale cluster systems. *Concurrency and Computation: Practice and Experience*, 20(1), 75-97. doi: 10.1002/cpe.1222
- Javadi, B., Abawajy, J. H., & Akbari, M. K. (2008b). Performance modeling and analysis of heterogeneous meta-computing systems interconnection networks.

Computers & Electrical Engineering, 34(6), 488-502. doi:
<http://dx.doi.org/10.1016/j.compeleceng.2007.09.007>

- Javadi, B., Akbari, M. K., & Abawajy, J. H. (2005, 14-17 June 2005). *Performance analysis of heterogeneous multi-cluster systems*. Paper presented at the International Conference on Parallel Processing Workshops(ICPP 2005), Oslo, Norway.
- Javadi, B., Akbari, M. K., Abawajy, J. H., & Nahavandi, S. (2006, 20-23 Dec. 2006). *Multi-Cluster Computing Interconnection Network Performance Modeling and Analysis*. Paper presented at the International Conference on Advanced Computing and Communications (ADCOM 2006).
- Javadi, B., Khorsandi, S., & Akbari, M. K. (2005). Study of a Cluster-Based Parallel System Through Analytical Modeling and Simulation *Computational Science and Its Applications-ICCSA 2005* (pp. 1262-1271): Springer Berlin / Heidelberg.
- Jia, W., Xiangzhan, Y., & Jun, Y. (2009, 28-29 Dec. 2009). *Research on Network Simulation Abstract Technology Based on Simplicity Theory*. Paper presented at the International Conference on Wireless Networks and Information Systems (WNIS '09).
- Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., & Chapman, B. (2011). High performance computing using MPI and OpenMP on multi-core parallel systems. *Parallel Computing*, 37(9), 562-575. doi:
<http://dx.doi.org/10.1016/j.parco.2011.02.002>
- Jingjing, W., Ponomarev, D., & Abu-Ghazaleh, N. (2012, 15-19 July 2012). *Performance Analysis of a Multithreaded PDES Simulator on Multicore Clusters*. Paper presented at the ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation (PADS 2012)
- Kaplan, J. A., & Nelson, M. L. (1994). *A Comparison of Queueing, Cluster and Distributed Computing Systems*. NASA Langley Technical Report Server.
- Karmakar, N. (2011). Multi-core Architecture *The New Trend in Processor Making* (pp. 44): North Maharashtra University, India.
- Khanyile, N., Tapamo, J.-R., & Dube, E. (2012). An analytic model for predicting the performance of distributed applications on multicore clusters. *IAENG International Journal of Computer Science*, 39(3), 312-320.
- Khosravi, A., Khorsandi, S., & Akbari, M. K. (2011, 23-24 Feb. 2011). *Hyper node torus: A new interconnection network for high speed packet processors*. Paper presented at the International Symposium on Computer Networks and Distributed Systems (CNDS 2011).
- Kim, H., & Bond, R. (2009). Multicore Software Technologies. *Signal Processing Magazine, IEEE*, 26(6), 80-89.
- Koibuchi, M., Akiya, J., Watanabe, K., & Amano, H. (2003, 6-9 Oct. 2003). *Descending layers routing: a deadlock-free deterministic routing using virtual channels in system area networks with irregular topologies*. Paper presented at the Proceedings of the International Conference on Parallel Processing
- Koibuchi, M., Jouraku, A., & Amano, H. (2002). *The impact of path selection algorithm of adaptive routing for implementing deterministic routing* *Proceedings of the*

International Conference on Parallel and Distributed Processing Techniques and Applications (pp. 1431-1437).

Koibuchi, M., Watanabe, K., Kono, K., Akiya, J., & Amano, H. (2003, 1-4 Dec. 2003). *Performance evaluation of routing algorithms in RHiNET-2 cluster*. Paper presented at the Proceedings of the International Conference on Cluster Computing

Koibuchi, M., Watanabe, T., Minamihata, A., Nakao, M., Hiroyasu, T., Matsutani, H., & Amano, H. (2011, Nov. 30 2011-Dec. 2 2011). *Performance Evaluation of Power-Aware Multi-tree Ethernet for HPC Interconnects*. Paper presented at the Second International Conference on Networking and Computing (ICNC 2011).

Kosinska, J., Kosinski, J., & Zielinski, K. (2010, 20-25 Sept. 2010). *The Concept of Application Clustering in Cloud Computing Environments: The Need for Extending the Capabilities of Virtual Networks*. Paper presented at the Proceedings of the Fifth International Multi-Conference on Computing in the Global Information Technology (ICCGI).

Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to Parallel Computing*. Canada: The Benjamin/Cummings Publishing Company, Inc.

Leangsuksun, C., Liu, T., Liu, Y., Scott, S., Libby, R., & Haddad, I. (2005). *Highly Reliable Linux HPC Clusters: Self-Awareness Approach*

Parallel and Distributed Processing and Applications. In J. Cao, L. Yang, M. Guo & F. Lau (Eds.), (Vol. 3358, pp. 217-222): Springer Berlin / Heidelberg.

Lei, C., Hartono, A., & Panda, D. K. (2006, 25-28 Sept. 2006). *Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters*. Paper presented at the Proceedings of the IEEE International Conference on Cluster Computing.

Lei, C., Qi, G., & Panda, D. K. (2007, 14-17 May 2007). *Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System*. Paper presented at the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007).

Lin-Dong, L., De-Yu, Q., Qiang, C., & Jin-Xin, R. (2014, 28-30 Nov. 2014). *Efficient Scheduling Mechanism for Performance-Heterogeneous Multi-core Processor*. Paper presented at the 5th International Conference on Digital Home (ICDH 2014)

Lin, X. (2003). *An Efficient Communication Scheme for Fat-Tree Topology on Infiniband Networks*. (M.Sc), Feng Chia University Taiwan.

Liqun, C., Muralimanohar, N., Ramani, K., Balasubramonian, R., & Carter, J. B. (2006, 0-0 0). *Interconnect-Aware Coherence Protocols for Chip Multiprocessors*. Paper presented at the 33rd International Symposium on Computer Architecture (ISCA '06).

Maria, A. (1997, 7-10 Dec 1997). *Introduction To Modeling And Simulation*. Paper presented at the Proceedings of the 1997 Winter Simulation Conference.

Matloff, N. (2008). A Discrete-Event Simulation Course Based on the SimPy Language. <http://heather.cs.ucdavis.edu/~matloff/simcourse.html>

- Mehta, S., Sulatan, N., & Kwak, K. S. (2010). Network and System Simulation Tools for Next Generation Networks: a Case Study, Modelling, Simulation and Identification. In A. M. (Ed.) (Ed.), *Modelling, Simulation and Identification*. InTech.
- Mei, C., Zheng, G., Gioachin, F., & Kal, L. V. (2010). *Optimizing a parallel runtime system for multicore clusters: a case study*. Paper presented at the Proceedings of the 2010 TeraGrid Conference, Pittsburgh, Pennsylvania.
- Moore, G. E. (1965). Cramming more components onto integrated circuits. *Vol. 86*, pp. 114-117.
- Moreno-Vozmediano, R., Montero, R. S., & Llorente, I. M. (2011). Multicloud Deployment of Computing Clusters for Loosely Coupled MTC Applications. *IEEE Transactions on Parallel and Distributed Systems*, 22(6), 924-930. doi: 10.1109/TPDS.2010.186
- Muralimanohar, N., & Balasubramonian, R. (2007). Interconnect design considerations for large NUCA caches. *SIGARCH Computer Architecture News*, 35(2), 369-380. doi: 10.1145/1273440.1250708
- Pan, J. (2008). A Survey of Network Simulation Tools : Current Status and Future Developments. 1-13.
- Pase, D. M., & Eckl, M. A. (2005). *A Comparison of Single-Core and Dual-Core Opteron Processor Performance for HPC* I. Corporation (Ed.) (pp. 13).
- Peh, L. S. (2001). *Flow Control and Microarchitectural Mechanism for Extending the Performance of Interconnection Networks*. (PhD Thesis), Stanford University.
- Pourreza, H., & Graham, P. (2007, 13-16 May 2007). *On the Programming Impact of Multi-Core, Multi-Processor Nodes in MPI Clusters*. Paper presented at the 21st International Symposium on High Performance Computing Systems and Applications (HPCS 2007)
- Prisacari, B., Rodriguez, G., Minkenberg, C., & Hoeﬂer, T. (2013). *Bandwidth-optimal all-to-all exchanges in fat tree networks*. Paper presented at the Proceedings of the 27th international ACM conference on International conference on supercomputing, Eugene, Oregon, USA.
- Qian, Y. (2010). *Design and Evaluation of Effiecient Collective Communications on Modern Interconnects and Multi-core Clusters*.
- Ranadive, A., Kesavan, M., Gavrilovska, A., & Schwan, K. (2008). *Performance implications of virtualizing multicore cluster machines*. Paper presented at the Proceedings of the 2nd workshop on System-level virtualization for high performance computing, Glasgow, Scotland.
- Rauber, T., & Runger, G. (2010). *Parallel Programming for Multicore and Cluster Systems*: Springer.
- Rechistov, G., Ivanov, A., Shishpor, P., & Pentkovski, V. (2012). Simulation and Performance Study of Large Scale Computer Cluster Configuration: Combined Multi-level Approach. *Procedia Computer Science*, 9(0), 774-783. doi: <http://dx.doi.org/10.1016/j.procs.2012.04.083>

- Requena, C. G., Requena, M. E. G., Rodriguez, P. J. L., & Marin, J. F. D. (2009). FTEI: A Dynamic Fault-Tolerant Routing Methodology for Fat Trees with Exclusion Intervals. *IEEE Transactions on Parallel and Distributed Systems*, 20(6), 802-817. doi: 10.1109/TPDS.2008.130
- Robinson, S. (2004). *Simulation: The Practice of Model Development and Use*. England: John Wiley & Sons, Ltd.
- Rouse, M. (2006). Definition of a processor and a node. Retrieved 02 August, 2012
- Roy, P. V. (2008). The Challenges and Opportunities of Multiple Processors: Why Multi-Core Processors are Easy and Internet is Hard. *Position statement, in ICMC*, 2.
- Sadashiv, N., & Kumar, S. M. D. (2011). *Cluster, grid and cloud computing: A detailed comparison*. Paper presented at the Proceedings of the 6th International Conference on Computer Science & Education (ICCSE).
- Sadeghi, M., & Barati, M. (2012, 3-5 July 2012). *Performance analysis of Poisson and Exponential distribution queuing model in Local Area Network*. Paper presented at the Proceedings of the International Conference on Computer and Communication Engineering (ICCCE).
- Sancho, J. C., Robles, A., & Duato, J. (2004). An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 15(8), 740-754. doi: 10.1109/tpds.2004.28
- Sarbazi-Azad, H., Ould-Khaoua, M., & Mackenzie, L. M. (2001). An accurate analytical model of adaptive wormhole routing in k-ary n-cubes interconnection networks. *Performance Evaluation*, 43(2-3), 165-179. doi: 10.1016/s0166-5316(00)00049-3
- Sarbazi-Azad, H., Ould-Khaoua, M., & Zomaya, A. Y. (2005). Design and performance of networks for super-, cluster-, and grid-computing: Part i. *Parallel Distributed Computing*, 65(10), 1119-1122.
- Sargent, R. G. (2011, 11-14 Dec. 2011). *Verification and validation of simulation models*. Paper presented at the Proceedings of the 2011 Winter Simulation Conference (WSC).
- Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of simulation*, 7(1), 12-24.
- Schauer, B. (2008). Multicore Processors-A Necessity. *ProQuest*, 14.
- Schlesinger, S. (1979). Terminology for model credibility. *SIMULATION*, 32(3), 103-104. doi: 10.1177/003754977903200304
- Schroeder, B., & Gibson, G. (2010). A Large-Scale Study of Failures in High-Performance Computing Systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4), 337-350. doi: 10.1109/TDSC.2009.4
- Schroeder, M. D., Birrell, A. D., Burrows, M., Murray, H., Needham, R. M., Rodeheffer, T. L., . . . Thacker, C. P. (1991). Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8), 1318-1335. doi: 10.1109/49.105178

- Shahhoseini, H. S., Naderi, M., & Buyya, R. (2000, 14-17 May 2000). *Shared memory multistage clustering structure, an efficient structure for massively parallel processing systems*. Paper presented at the Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region.
- Shainer, G., Lui, P., Hilgeman, M., Layton, J., Stevens, C., Stemple, W., . . . Kresse, G. (2013). *Maximizing Application Performance in a Multi-core, NUMA-Aware Compute Cluster by Multi-level Tuning Supercomputing* (Vol. 7905, pp. 226-238): Springer Berlin Heidelberg.
- Shannon, R. E. (1977). Simulation modeling and methodology. *SIGSIM Simul. Dig.*, 8(3), 33-38. doi: 10.1145/1102766.1102770
- Sharifi, H., Akbari, M. K., & Javadi, B. (2009). *An Analytical Model of Communication Networks in Multi-cluster Systems in the Presence of Non-uniform Traffic*. Paper presented at the International Conference on Computational Science and Engineering (CSE '09).
- Silva, J. M. N., Drummond, L., & Boeres, C. (2010, 27-30 Oct. 2010). *On Modelling Multicore Clusters*. Paper presented at the Proceedings of the 22nd International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW).
- Sommereder, M. (2011). *Modelling of Queueing Systems with Markov Chains: An Introduction to Basic and Advanced Modelling Techniques: BoD-Books on Demand*.
- Soryani, M., Analoui, M., & Zarrinchian, G. (2013). Improving inter-node communications in multi-core clusters using a contention-free process mapping algorithm. *The Journal of Supercomputing*, 1-26. doi: 10.1007/s11227-013-0918-7
- Srinivas, V. V., & Ramasubramaniam, N. (2011). *Understanding the Performance of Multi-core Platforms*. In V. Das, J. Stephen & Y. Chaba (Eds.), *Computer Networks and Information Technologies* (Vol. 142, pp. 22-26): Springer Berlin Heidelberg.
- Sterling, T., Apon, A., & Baker, M. (2000). Cluster Computing White Paper. *Cluster Computing*, (December).
- Sterling, T. L. (2002). *Beowulf Cluster Computing with Windows Computers*, M. Press (Ed.) (pp. 445).
- Stevens, R. (1986). *Understanding Computers*. Oxford: Oxford University Press.
- Sullivan, D. R., Lewis, T. G., & Cook, C. R. (1988). *Computing Today: Microcomputer Concepts and Application*. USA: Houghton Mifflin Company.
- Tanenbaum, A. S. (1996). *Computer Networks*. USA: A Simon & Schuster Company.
- Varga. (2011). OMNeT ++ User Manual Version 4.2.2. <http://www.omnetpp.org/doc/omnetpp/Manual.pdf>
- Varga, & Hornig, R. (2008). *An overview of the OMNeT++ simulation environment*. Paper presented at the Proceedings of the 1st international conference on

Simulation tools and techniques for communications, networks and systems & workshops, Marseille, France.

Varga, A. (2001). OMNeT++. Retrieved October 2012

Varghese, B., McKee, G., & Alexandrov, V. (2010). Implementing intelligent cores using processor virtualization for fault tolerance. *Procedia Computer Science*, 1(1), 2197-2205. doi: <http://dx.doi.org/10.1016/j.procs.2010.04.246>

Wehrle, K., Gunes, M., & Gross, J. (2010). *Modeling and Tools for Network Simulation* K. Wehrle, M. Gunes & J. Gross (Eds.), Retrieved from <http://www.scribd.com/doc/88271950/89/Cluster-Computing-Support>

Weingartner, E., vom Lehn, H., & Wehrle, K. (2009). *A Performance Comparison of Recent Network Simulators*. Paper presented at the IEEE International Conference on Communications (ICC '09).

Willis, N., & Kerridge, J. (1983). *Introduction to Computer Architecture*. UK: Pitman Publishing.

Wu, X., & Taylor, V. (2013). Performance modeling of hybrid MPI/OpenMP scientific applications on large-scale multicore supercomputers. *Journal of Computer and System Sciences*(0). doi: <http://dx.doi.org/10.1016/j.jcss.2013.02.005>

Xuan-Yi, L., Yeh-Ching, C., & Tai-Yi, H. (2004, 26-30 April 2004). *A multiple LID routing scheme for fat-tree-based InfiniBand networks*. Paper presented at the Proceedings of the 18th International Parallel and Distributed Processing Symposium

Yeo, C., Buyya, R., Pourreza, H., Eskicioglu, R., Graham, P., & Sommers, F. (2006). *Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers*. In A. Zomaya (Ed.), *Handbook of Nature-Inspired and Innovative Computing* (pp. 521-551): Springer US.

Yulei, W., Geyong, M., Keqiu, L., & Javadi, B. (2012). Modeling and Analysis of Communication Networks in Multicluster Systems under Spatio-Temporal Bursty Traffic. *IEEE Transactions on Parallel and Distributed Systems*, 23(5), 902-912. doi: 10.1109/tpds.2011.198