

# **ENERGY-TIME PERFORMANCE OF HETEROGENEOUS COMPUTING SYSTEMS: MODELS AND ANALYSIS**

Lavanya Ramapantulu

*M.S. Microelectronics, Birla Institute of Technology and Science*

A THESIS SUBMITTED FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE  
NATIONAL UNIVERSITY OF SINGAPORE

2016

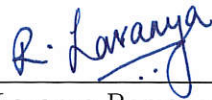


## DECLARATION

I hereby declare that the thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

No portion of the work referred to in this thesis has been submitted in support of an application for another degree of qualification at this or any other university or institution of learning.



---

Lavanya Ramapantulu

12 May 2016



# Abstract

While heterogeneity is increasingly becoming the norm in most computing platforms today, one of the key challenges is to determine the set of energy-time efficient system configurations among the large system configuration space to execute a parallel application. This large configuration space offers a new opportunity to improve the match between parallel application demands and system resources to achieve efficient energy-time performance. This thesis presents an approach to address this challenge using a *measurement-driven analytical model* that determines both time and energy efficient system configurations. Based on our taxonomy of heterogeneous computing systems, we first propose a core analytical model for a baseline heterogeneous system representing *inter-node heterogeneity* and consisting of brawny and wimpy nodes. The proposed core model is scalable for different types of heterogeneity and is formulated using parametric values obtained from baseline measurements of the application for better accuracy. The key novelties of our approach include modeling both inter and intra-node resource overlaps and resource contention.

Among heterogeneous systems, intra-node heterogeneous systems with Vector Processing Units (VPUs) are increasingly being adopted in the Top500 supercomputers as they offer accelerated performance gains. Secondly, the impact of heterogeneity in parallel programs leads to the wider adoption of hybrid programming models for scientific applications. Hybrid programming models are gaining traction as they exploit system resources and parallelism at both inter- and intra-node levels. The scalability of our proposed core model is shown by extending it to both *intra-node heterogeneous system* and *hybrid programs*. Key model extensions include (i) inter- and intra-core contentions for VPUs in a Many Integrated Core (MIC) architecture system, and (ii) inter- and intra-node communication for hybrid programs.

With the advent of heterogeneity at both program and system level, it is non-trivial for application developers or users to choose an energy and time optimal configuration from the large configuration space. The proposed core model and its extensions are applied to determine energy-time efficient system configurations

for inter-node heterogeneous system, intra-node heterogeneity with VPUs and hybrid programs. In determining these efficient system configurations, we exposed a number of insights. Firstly, there are *multiple Pareto-optimal “sweet-spot” configurations* that can be approximated using a distinct energy-deadline *Pareto-frontier*. These configurations facilitate energy-time trade-offs such as to minimize energy used for a given execution-time deadline and/or to minimize execution time for a given energy budget. Our analysis shows that for inter-node heterogeneous clusters and hybrid programs, energy savings of up to 75% can be achieved by selecting Pareto-optimal configurations as opposed to non-optimal configurations. Furthermore, among the Pareto-optimal configurations hybrid programs reduces the energy used by up to 65% at the expense of 18% increase in execution time.

With the explosion of the configuration space, we show that the Pareto-frontier can be analytically established using the node *performance-to-power ratios (PPR)*. We show that the Pareto-frontier can be further improved by replacing low PPR nodes with higher PPR nodes using our power substitution ratio. Additionally, our energy proportionality analysis reveals that inter-node heterogeneous clusters enable the scaling of the energy-proportionality wall by exposing sub-linear energy-proportional configurations. To further optimize the Pareto-frontier, we introduce a new metric called *useful computation ratio (UCR)* to quantify the degree of resource contentions and communication overheads in an execution. Lastly, we show how UCR and Pareto-optimal configurations can be used in conjunction by system designers to gain further insights into system resource imbalances, and how application developers can further fine-tune their hybrid programs.

*Science is bound, by the everlasting vow of honour, to face fearlessly every problem which can be fairly presented to it.*

– Lord Kelvin





## Acknowledgments

I would like to express my sincere gratitude to my thesis advisor Professor Teo Yong Meng. The door to Professor Teo's office was always open whenever I needed help or had a question about my research or writing. He has taught me the nuances of research by asking the right questions to help me think clearly and present my thoughts in a coherent manner. His guidance struck the right balance between giving me the freedom to follow my ideas and steering me in the right direction whenever he thought I needed it. His teaching style of asking the right questions inspires me to embark on an academic career. Thank you, Professor Teo for your advice and support during my PhD studies.

I would like to take this opportunity to thank my thesis advisory committee, Professor Ooi Wei-Tsang and Professor Wong Weng-Fai for their comments, feedback and support during the various milestones of my candidature. I would like to thank Professor Chin Wei Ngan, Professor Ooi Beng Chin, Professor Ananda and Professor Lakshminarayanan for being supportive and lending their systems for my research. My heartfelt gratitude to Madam Loo Line Fong for her continuous support both before joining and during my PhD candidature. I thank the staff in the graduate office for their support in administrative matters and enabling me to attend conferences to present my research. I recognize that this research would not have been possible without financial assistance and express my sincere gratitude to the Ministry of Education, Singapore.

A special acknowledgement to my mentors in the lab, Dr. Bogdan Marius Tudor and Dr. Cristina Carbutaru, from whom I have learnt the nitty-gritty details involved in writing technical research papers. A special thank you to my wonderful colleague and collaborator Dumitrel Loghin for discussing my research and reviewing my work. I would like to thank my lab mates over the years Khanh, Linh, Trang, Saeid, Thy, Oana, Sunimal, Suman, Irvan for making the work place enjoyable.

I thank all my friends who made my stay in Singapore such a wonderful experience. Marcel, Pooja, Sreetama, Subhasree, Sudipta, Shuang Liu for their advice during the different milestones of my PhD studies; Akshay, Asha, Parvathy, Neha, Sanat for making me feel at home; Prasanta, Anirudh, Abhra, Bhargava, Yang Yi for the discussions on life philosophies; Mahsa, Maryam, Huping, Chundong, Yamilet, Anh and the NUS Buddhist society for enhancing my cultural experience; Peichu, Sergey, Shin Hwei for company to play table-tennis; Kuldeep, Rajiv for

being wonderful cohort mates; and many more friends who made this journey an enjoyable one.

Last but not the least, I would like to thank my family and friends for being a source of strength, love and encouragement despite being thousands of miles away from me. Finally, I thank my friend, philosopher and life-guide Anshoo Tandon without whom I would not have embarked on this defining chapter of my life.

I am sure that there are many more to thank and sincerely apologize to those that are overlooked.

# List of Publications

1. L. Ramapantulu, B.M. Tudor, D. Loghin, T. Vu and Y.M. Teo, *Modeling the Energy Efficiency of Heterogeneous Clusters*, Proceedings of 43rd International Conference on Parallel Processing, pp 321-330, Minneapolis, USA, Sep 9-12, 2014. [Inter-node Heterogeneity]
2. L. Ramapantulu, D. Loghin and Y.M. Teo, *An Approach for Energy Efficient Execution of Hybrid Parallel Programs*, Proceedings of 29th IEEE International Parallel and Distributed Processing Symposium, pp 1000-1009, Hyderabad, INDIA, May 25-29, 2015. [Hybrid Programs]
3. D. Loghin, L. Ramapantulu, O. Barbu and Y.M. Teo, *A Time-Energy Performance Analysis of MapReduce on Heterogeneous Systems with GPUs*, Performance Evaluation - An International Journal, Vol 91, pp 255-269, Elsevier, 33rd International Symposium on Computer Performance, Modeling, Measurement and Evaluation (IFIP WG 7.3 Performance 2015), Sydney, AUSTRALIA, Oct 19-21, 2015. [Inter-chip and Intra-chip Heterogeneity]
4. L. Ramapantulu, D. Loghin and Y.M. Teo, *On Energy Proportionality and Time-Energy Performance of Heterogeneous Clusters*, Proceedings of 18th IEEE Cluster Conference, Taipei, Taiwan, Sep 12-16, 2016. (accepted) [Inter-node Heterogeneity]
5. L. Ramapantulu, T. Dao, D. Loghin, N. Thoai and Y.M. Teo, *Modeling the Energy-Time Performance of MIC Architecture System*, Proceedings of 24th IEEE Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, London, UK, Sep 19-21, 2016. (accepted) [Intra-node Heterogeneity]



# Table of Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>vii</b>
<b>List of Publications</b>	<b>ix</b>
<b>List of Figures</b>	<b>xv</b>
<b>List of Tables</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Challenges and Research Questions . . . . .	4
1.3 Objective and Contributions . . . . .	8
1.4 Thesis Organization . . . . .	11
<b>2 Related Work</b>	<b>15</b>
2.1 Heterogeneous Computing Systems . . . . .	15
2.1.1 Systems . . . . .	18
2.1.2 Programs . . . . .	25
2.2 Approaches . . . . .	28
2.2.1 Energy Efficiency . . . . .	29
2.2.2 Energy Proportionality . . . . .	31
2.2.3 Time Performance . . . . .	36
2.3 Heterogeneity and Energy-time Performance . . . . .	42
2.4 Summary . . . . .	43
<b>3 Approach and Core Model</b>	<b>45</b>
3.1 Overview . . . . .	45

3.2	Core Model . . . . .	48
3.2.1	Model Inputs . . . . .	48
3.2.2	Execution Time Model . . . . .	50
3.2.3	Matching Technique . . . . .	54
3.2.4	Energy Model . . . . .	56
3.3	Validation . . . . .	57
3.3.1	Workloads and Setup . . . . .	58
3.3.2	$WPI$ and $SPI_{core}$ . . . . .	60
3.3.3	$SPI_{mem}$ Regression over Core Frequency $f$ . . . . .	60
3.3.4	Execution Time and Energy . . . . .	62
3.3.5	Sources of Inaccuracy . . . . .	62
3.3.6	Accuracy of Related Work . . . . .	63
3.4	Summary . . . . .	63
<b>4</b>	<b>Analysis of Inter-node Heterogeneous Systems</b>	<b>65</b>
4.1	Pareto-optimal Configurations . . . . .	65
4.2	Impact of Performance-to-Power Ratio . . . . .	68
4.2.1	Performance-to-Power Ratio (PPR) . . . . .	68
4.2.2	Analytical Analysis of PPR on Sweet Region . . . . .	71
4.3	Impact of Power Substitution Ratio . . . . .	73
4.3.1	Power Substitution Ratio (PSR) . . . . .	73
4.3.2	Impact of Heterogeneous Mixes on the Sweet Region . . . . .	74
4.3.3	Analytical Analysis of PSR on Sweet Region . . . . .	77
4.4	Energy Proportionality Analysis . . . . .	78
4.4.1	Brawny versus Wimpy Node . . . . .	79
4.4.2	Cluster-wide Energy Proportionality . . . . .	85
4.4.3	Does Heterogeneity Scale the Energy Proportionality Wall? . . . . .	89
4.5	Summary . . . . .	92
<b>5</b>	<b>Extension to Intra-node Heterogeneity with VPU</b>	<b>95</b>
5.1	Overview . . . . .	98
5.1.1	MIC Architecture . . . . .	100
5.2	Time Performance Model . . . . .	102
5.2.1	Intra-core contention . . . . .	105
5.2.2	Inter-core contention . . . . .	106
5.3	Model Parameterization and Validation . . . . .	107

5.3.1	Workloads and Setup . . . . .	108
5.3.2	Model Parameterization . . . . .	109
5.3.3	Execution time Validation . . . . .	112
5.3.4	Power Characterization . . . . .	114
5.4	Analysis . . . . .	114
5.4.1	Pareto-optimal Configurations . . . . .	115
5.4.2	Performance-to-power Ratio . . . . .	117
5.5	Summary . . . . .	121
<b>6</b>	<b>Extension to Hybrid Programs with Communication</b>	<b>123</b>
6.1	Overview . . . . .	124
6.2	Communication Model . . . . .	128
6.2.1	Model Inputs . . . . .	128
6.2.2	Time Model . . . . .	130
6.2.3	Energy Model . . . . .	133
6.2.4	Validation . . . . .	134
6.3	Analysis . . . . .	138
6.3.1	Pareto-optimal Configurations . . . . .	139
6.3.2	Useful Computation Ratio . . . . .	142
6.4	Summary . . . . .	145
<b>7</b>	<b>Conclusion</b>	<b>149</b>
7.1	Thesis Summary . . . . .	149
7.1.1	Measurement-based Analytical Model . . . . .	150
7.1.2	Insights from Energy-Time Performance Analysis . . . . .	154
7.1.3	Limitations . . . . .	157
7.2	Future Directions . . . . .	157
7.2.1	Dynamic Adaptation of Configurations at Run-time . . . . .	157
7.2.2	Cost-Time Performance . . . . .	158
	<b>References</b>	<b>161</b>
<b>A</b>	<b>Experiment Setup</b>	<b>185</b>
A.1	Programs . . . . .	185
A.2	Systems . . . . .	187
A.3	Software Setup . . . . .	188
A.4	Validation Results . . . . .	190

<b>B</b>	<b>Model Parameters</b>	<b>192</b>
<b>C</b>	<b>Sensitivity Analysis</b>	<b>195</b>
C.1	What is a Good Mix of High-performance to Low-power Nodes?	195
C.2	Are Larger Mixes of Heterogeneous Nodes Better?	197
C.3	Does Static Workload Allocation Suffice?	199
C.4	Impact of Realistic Workloads on Heterogeneity	202
C.5	Impact of Jobs Queueing Delay	207



# List of Figures

2.1	A classification of heterogeneous computing systems . . . . .	17
2.2	Power-performance of processors . . . . .	19
2.3	Example system with inter-chip heterogeneity, Intel Xeon Phi . . .	21
2.4	Example system with intra-chip heterogeneity, ARM big.LITTLE .	24
2.5	Example system with intra-chip heterogeneity, Nvidia Jetson TK1 .	25
2.6	Hybrid program with MPI, OpenMP, Xeon Phi API and CUDA . .	28
2.7	Energy proportionality metric relationships . . . . .	32
2.8	Energy-performance of CPU, GPU and MIC [140] . . . . .	34
3.1	Methodology for core model . . . . .	47
3.2	Validation setup . . . . .	60
3.3	$WPI$ and $SPI_{core}$ across problem size . . . . .	61
3.4	Effect of frequency and number of cores on $SPI_{mem}$ . . . . .	61
4.1	Pareto frontier for EP with $d_{max} = 3$ . . . . .	67
4.2	Pareto frontier configurations for EP . . . . .	69
4.3	Pareto frontier configurations for RSA-2048 . . . . .	70
4.4	Configurations on the sweet-spot region . . . . .	72
4.5	Pareto frontier for EP with $d = 2$ . . . . .	75
4.6	Pareto frontier for EP with $d = 3$ . . . . .	75
4.7	Pareto frontier for RSA-2048 with $d = 2$ . . . . .	76
4.8	Pareto frontier for RSA-2048 with $d = 3$ . . . . .	77
4.9	Energy proportionality of EP . . . . .	80
4.10	Energy proportionality of x264 . . . . .	81
4.11	Energy proportionality of blackscholes . . . . .	81
4.12	PPR of EP . . . . .	83
4.13	PPR of x264 . . . . .	83
4.14	PPR of blackscholes . . . . .	84

4.15	Cluster-wide energy proportionality $d = 1$ and $d = 2$ . . . . .	87
4.16	Cluster-wide energy proportionality $d = 3$ . . . . .	88
4.17	Energy proportionality of Pareto-optimal configurations for EP . . . . .	90
4.18	Response time of sub-linear heterogeneous mixes for EP . . . . .	91
4.19	Energy proportionality of Pareto-optimal configurations for x264 . . . . .	91
4.20	Response time of sub-linear heterogeneous mixes for x264 . . . . .	92
5.1	Approach for intra-node heterogeneity . . . . .	99
5.2	MIC architecture . . . . .	101
5.3	Abstraction of a parallel program . . . . .	103
5.4	WPI validation . . . . .	110
5.5	Validation of instructions . . . . .	110
5.6	Validation of cache accesses . . . . .	111
5.7	Validation results for scatter thread affinity mode . . . . .	112
5.8	Validation results for compact thread affinity mode . . . . .	113
5.9	Pareto-optimal configurations for executing BT . . . . .	115
5.10	Pareto-optimal configurations for executing FT . . . . .	116
5.11	Pareto-optimal configurations for executing CG in scatter mode . . . . .	119
5.12	Pareto-optimal configurations for executing CG . . . . .	120
6.1	Abstraction of a hybrid program . . . . .	125
6.2	Model of hybrid program execution . . . . .	125
6.3	Approach for hybrid programs . . . . .	126
6.4	Network characterization . . . . .	129
6.5	Execution time validation . . . . .	136
6.6	Energy validation . . . . .	136
6.7	Scale-out program LU . . . . .	137
6.8	Xeon cluster executing SP program . . . . .	139
6.9	ARM cluster executing CP program . . . . .	140
6.10	UCR and time-energy performance on Xeon cluster . . . . .	143
6.11	UCR and time-energy performance on ARM cluster . . . . .	144
7.1	Measurement-based analytical model . . . . .	151
C.1	Heterogeneous mixes for memcached . . . . .	196
C.2	Heterogeneous mixes for EP . . . . .	196
C.3	Increasing cluster size for memcached . . . . .	198

C.4	Increasing cluster size for EP . . . . .	198
C.5	Memcached static workload allocation . . . . .	200
C.6	EP static workload allocation . . . . .	201
C.7	Pareto frontier with sequential fraction $\alpha = 0.1$ . . . . .	203
C.8	Pareto frontier with parallel overhead $k = 0.01$ . . . . .	206
C.9	Effect of job queueing delay on cluster utilization . . . . .	208



# List of Tables

2.1	Comparison of performance analysis approaches . . . . .	37
2.2	Heterogeneity and energy-time performance . . . . .	42
3.1	Core model parameters . . . . .	49
3.2	Types of heterogeneous nodes . . . . .	59
3.3	Single-node validation . . . . .	60
3.4	Cluster validation . . . . .	62
3.5	Accuracy of Related Work . . . . .	63
3.6	Summary of core model . . . . .	64
4.1	Performance-to-power ratio . . . . .	69
4.2	Cluster mixes for 1kW power budget . . . . .	74
4.3	Useful Operations . . . . .	80
4.4	Single-node energy proportionality . . . . .	80
4.5	Cluster-wide energy proportionality . . . . .	85
5.1	Model parameters . . . . .	100
5.2	Programs . . . . .	108
5.3	Intel node with MIC co-processor . . . . .	109
5.4	Validation results . . . . .	113
5.5	Performance-to-power ratio . . . . .	117
5.6	Summary of model extension for MIC architecture . . . . .	121
6.1	Communication model parameters . . . . .	128
6.2	Hybrid program system . . . . .	135
6.3	Hybrid program validation results . . . . .	137
6.4	Summary of model extension for hybrid programs . . . . .	147
7.1	Summary of all models . . . . .	152

A.1	Programs used in thesis . . . . .	187
A.2	Systems used in thesis . . . . .	188
A.3	Software versions used in thesis . . . . .	189
A.4	Execution time validation on a single ARM node . . . . .	190
A.5	Execution time validation on a single AMD node . . . . .	190
A.6	Energy validation on a single ARM node . . . . .	191
A.7	Energy validation on a single AMD node . . . . .	191
B.1	Notations used in thesis . . . . .	194
C.1	Energy and service time with $\alpha = 0.1$ . . . . .	204
C.2	Energy and service time with $k = 0.01$ . . . . .	206
C.3	Energy and service time for ideal case . . . . .	207

# Chapter 1

## Introduction

The last 50 years have seen a rapid growth in the performance of processors, with performance improvements by a factor of 10,000 over the last 20 years [61]. This growth in performance was sustainable due to the increase in the number of transistors with a reduction in their size, thus leading to reduced costs and manageable power with increase in performance. However, this sustained growth has come to a stand-still in the 21st century due to the fundamental limits in the power-efficiency of CMOS technology based processors. Therefore, along with improvements in execution-time performance, energy and power performance of processors is increasingly becoming very important.

This led to a paradigm shift from conventional sequential programming to parallel programming and parallel systems for sustained growth in computer performance. This shift presented opportunities to explore heterogeneity in both programs and systems, and develop novel approaches to meet both energy and time performance constraints of developers and users of parallel programs. With heterogeneity in both programs and systems driving the future of computing, it provides new opportunities to determine a better match between parallel programs and available system resources to execute applications efficiently.

Heterogeneous Computing Systems (HCS) is heterogeneity in all manifestations of computing, including processors, networks, programming, protocols and all combinations of these coming together to deliver a positive impact. While HCS is all encompassing, we consider three types of heterogeneity for the scope of this thesis, (i) inter-node heterogeneity - multi-core cluster of processors with diverse performance-to-power ratios among them, (ii) intra-node heterogeneity - compute node with accelerators/coprocessors such as Vector Processing Units (VPU), and (iii) hybrid programs - exploiting inter-node distributed-memory scalability and intra-node shared-memory performance in a cluster system.

While heterogeneity introduces new opportunities to gain a better balance between program demands and system resources, it brings forth many challenges. One of the key challenges due to heterogeneity is the large system configuration space made available. A heterogeneous mix of nodes offers a much larger system configuration space due to the different combinations of system parameters such as, types of nodes, number of nodes of each type, number of active cores per node and the operating core clock frequency. If  $d$  denotes the degree of heterogeneity ( $d = 1$  for a homogeneous system) then the space complexity of the system configuration space available for execution grows exponentially to the power of  $d$ . For example, a single type of processor having four cores per node and four operating clock frequencies can have a maximum of 16 configurations. When the types of processing nodes increase to two, then the configuration space becomes 256. Furthermore, by adding a third type of processor, the number of configurations grows to 4096. For an application developer or user choosing an energy and time optimal configuration from this large configuration space is non-trivial. In this thesis, we aim to address this challenge of determining energy-time efficient system configuration to execute a given parallel program.



## 1.1 Motivation

Energy efficiency of servers is a key concern for most datacenters in the world today. For example, the power consumed by datacenters in the United States (US) is estimated at nearly 91 billion kilowatt-hours in 2013 [127]. The power consumption costs of datacenters are increasingly becoming a larger fraction of the total cost of ownership [29]. From the environmental perspective, datacenter power consumption in the US alone is equivalent to several million tons of greenhouse gas emission per year [133]. Thus, improving the energy efficiency of servers not only benefits the total cost of ownership but also has a positive impact on the environment.

Many research findings advocate the usage of low-power processors (wimpy nodes) as an alternative to traditional servers (brawny nodes), for achieving higher energy efficiency [21, 84, 88, 102]. On the contrary, other researchers and practitioners indicate that clusters using only brawny high-performance nodes are more energy efficient [117, 156], but much remains to be explored to improve cluster-wide energy efficiency of scale-out workloads. While a system using only low-power nodes may not service the job fast enough to meet the desired QoS (e.g. deadline) [79], a system with only high-performance nodes may require an inordinate amount of energy due to over-provisioning [10]. Ideally, a system should allow a range of configurations where energy usage decreases progressively as the deadline is relaxed. This motivates the case for analyzing heterogeneous computing systems having different mixes of nodes with varied performance-to-power ratios (PPRs).

With the slowing down of “Moore’s law” and dark-silicon limiting the number of active cores in a multi-core processor, mixing CPUs and accelerators seems like a viable alternative to scale-up parallel computing performance. The recent

years have seen the wide adoption of accelerators by the HPC community. Among the Top500 systems in November 2015, there were about 104 systems with accelerators [8]. Traditionally, while GPUs have been dominating the accelerator arena, the launch of Intel’s Knight Corner in 2012, have seen another class of accelerators being adopted mainstream, namely the Many Integrated Core (MIC) architecture. The increasing adoption of this architecture is evident in the Top500 systems, where 32 among the 104 systems with accelerators use the Intel Xeon Phi co-processor which is based on the MIC architecture. Thus, it is important to address the challenges due to the adoptions of intra-node heterogeneous systems with VPUs.

While heterogeneous clusters address heterogeneity at the system level, programs are increasingly becoming hybrid to simultaneously exploit the computational capabilities of multi-core systems and to scale complex HPC applications [27, 105, 148]. To utilize the advantages offered by hybrid programs, there is a need to design the right balance between shared-memory computations and the associated message-passing overheads. Thus, improving the time and energy performance of heterogeneous computing systems involves not only determining time and energy-time efficient system configurations among heterogeneous multi-core clusters, but also determining the optimum trade-off between the number of threads and the number of logical processes for efficient execution of hybrid programs.

## 1.2 Challenges and Research Questions

With heterogeneity redefining the parallel computing landscape, it brings along a plethora of challenges to the research community. Some of the key challenges include improving efficiency, managing complexity, and improving dependability

[52]. With respect to improving efficiency, there are multiple performance metrics to choose from with the focus moving from performance per Dollar to performance per Watt per Dollar with ever increasing operational costs of energy and cooling. There is a need to define standards to inter-operate software and tools in to manage the complexity brought forth by heterogeneous systems. There is also a need to explore high computing systems having an optimum mix of heterogeneous processors, accelerators and interconnect technologies.

From a system perspective, a heterogeneous mix of nodes offers a large configuration space due to the different combinations of system parameters such as, types of nodes, number of nodes of each type, number of active cores per node and the operating core clock frequency. While heterogeneity is becoming the norm in most computing systems today, one of the key challenges is to determine the set of energy-time efficient system configurations among the large system configuration space<sup>1</sup>. Analyzing the energy-efficiency for such a large configuration space triggers a multitude of challenges including:

1. Given an execution-time deadline and energy budget, how can we determine the set of energy-time efficient configurations from the large system configuration space?
2. What is the impact of the performance-to-power ratio (PPR) on the set of energy-time efficient configurations?
3. For a given power budget, what is an energy-time efficient mix among the heterogeneous resources with diverse PPRs for a given power budget?

---

<sup>1</sup> For example, if we consider three types of nodes, with each node having four, six, and eight cores respectively and assuming the possible core clock frequencies for each type of node are  $f_1 \in [0.8, 1.4, 2.1]$  GHz,  $f_2 \in [0.2, 0.5, 0.8, 1.1, 1.4]$  GHz,  $f_3 \in [1.2, 1.5, 1.8]$  GHz. Then the total number of possible configurations =  $(3 \times 6 \times 3) \times (3 \times 4 \times 5) \times (3 \times 8 \times 3) + (3 \times 6 \times 3) \times (3 \times 4 \times 5) + (3 \times 4 \times 5) \times (3 \times 8 \times 3) + (3 \times 8 \times 3) \times (3 \times 6 \times 3) + (3 \times 6 \times 3) + (3 \times 4 \times 5) + (3 \times 8 \times 3) = \mathbf{244,914}$ .

#### 4. How does heterogeneity impact cluster-wide energy proportionality?

Though simple back of the envelope calculations using heuristics may provide the configuration, our aim is to obtain optimal time-energy configurations rather than a single configuration to aid in efficient trade-offs in both energy and time. In addition, we use a measurement-driven analytical model to obtain these configurations as the goal is to increase the accuracy of the predicted configuration.

With heterogeneity becoming inevitable, past research work has focused on mapping and scheduling multiple workloads on heterogeneous clusters [49, 64, 68, 126]. On the contrary, this thesis aims to address the aforementioned challenges and determine energy and time-efficient system configurations for executing a parallel application on heterogeneous computing systems.

While there is an increasing adoption of intra-node heterogeneous systems, such as MIC architecture system in Top500 supercomputers [8], reaching the theoretical peak performance of the Xeon Phi is challenging. It depends a lot on the scaling, how to bind threads to cores, the degree of vectorization and memory usage of the applications. Hence, for a HPC user, determining the optimal system configuration to execute the parallel application is non-trivial and poses a number of research challenges such as:

1. For a given program, what is the number of threads that achieves the best performance?
2. For a given program and the number of threads, what thread affinity mode achieves the best performance?

From a technology perspective, the advent of dark silicon era [55] is pushing computing systems towards heterogeneous parallelism. This trend is necessitating application developers to leverage the heterogeneity in systems and enable energy-time performance gains. Thus, HPC application developers are increasingly using

hybrid programming models to simultaneously exploit the computational capabilities of multi-core systems and use multiple nodes to scale complex applications. A hybrid parallel program is partitioned into a variable number of logical parallel processes and parallel threads. For a given hybrid program and a multi-node system with multi-core nodes operating at different core clock frequencies, there is a large system configuration space for executing these logical processes and threads. As the resource demands in a hybrid parallel program varies with its problem size, these resource demands have to be mapped onto different system configurations to minimize resource contention and runtime overheads. Hence, energy-time efficient execution of hybrid programs is non-trivial and poses a number of research challenges such as:

1. What is an energy efficient system configuration (number of nodes, number of cores and core clock frequency) to execute a hybrid program?
2. How much time does a hybrid program spend on computation (useful work) versus communication of data and other overheads, and what is a good Useful Computation Ratio (UCR)?
3. Does a higher UCR imply a more energy-time efficient configuration for executing a hybrid program?

Answers to these questions help both application developers to gain insights on program hot-spots, and system designers to identify capacity bottlenecks, and thus optimize software-hardware co-design to improve energy-efficiency. This thesis addresses these challenges and proposes a unified approach to not only determine energy and time-efficient system configurations for executing a parallel application on heterogeneous computing systems but also execute a hybrid (OpenMP+MPI) parallel program in an energy efficient manner.

## 1.3 Objective and Contributions

The objective of this thesis is to develop an approach to determine energy and time-efficient configurations for executing a parallel application on heterogeneous computing systems. With the advent of heterogeneity at both program and system level, this thesis presents a scalable measurement-based core analytical model to determine the time-energy performance of heterogeneous multi-core clusters. In contrast to pure analytical or mathematical models, the proposed approach exploits baseline measurements for better accuracy. Additionally, the proposed approach is scalable as we extend the core model to address different types of heterogeneity.

The key contributions of this thesis are:

### 1. Measurement-driven Analytical Model [136, 137]

#### (a) Core Model [137]

We propose a measurement-based analytical model to determine time and energy efficient system configurations for executing a parallel program on a baseline heterogeneous cluster consisting of multi-core brawny and wimpy nodes. Current approaches to analyze time-energy performance of parallel programs mainly include instrumentation or application profiling based measurement techniques or cycle-accurate simulations. However, the large configuration space due to heterogeneity prohibits their usage. The proposed core model is formulated using parametric values obtained from baseline executions of the application to measure workload and architectural artefacts. The key novelties of our approach are modeling both inter- and intra-node resource overlaps and resource contentions.

#### (b) MIC Architecture Model

We show the scalability of the core model by extending it to determine time and energy-efficient system configurations for parallel programs executing on co-processors with VPUs, e.g. Intel Xeon Phi. With at least 50 cores, Intel Xeon Phi coprocessor offers high parallelism and a theoretical peak of two TFLOPS for single precision, one TFLOPS for double precision and over 352 GB/s of memory bandwidth. The performance as well as the flexibility to use the Xeon Phi as a coprocessor or a standalone processor is promising for accelerating applications. Furthermore, Xeon Phi has a general-purpose programming environment and can be programmed with common programming languages, thus making it even more popular among HPC users. However, from a system perspective, coprocessors based on MIC architecture offer a large system configuration space to execute a parallel application due to the number of threads per core and the number of available cores. The core model is extended to address both inter- and intra-core contention for shared-memory in MIC architecture systems, thus providing a systematic method to users of MIC architecture systems to determine the thread affinity mode and the number of threads for efficient execution of a HPC application.

(c) **Communication Model [136]**

While extension of the core model to intra-node heterogeneous systems shows the scalability of the model from a system perspective, from a program perspective, the scalability of the core model is illustrated by modeling the communication for hybrid OpenMP and MPI programs. Hybrid programming model is becoming increasingly popular for HPC applications as it has the dual-advantage of exploiting inter-node scalability and intra-node shared-memory performance in a cluster system. As hybrid programs incur communication overheads, modeling these is key

to determine energy-time efficient system configurations. We extend our measurement-driven core model to incorporate both intra- and inter-node communication, and network contention across nodes, which enables us to determine energy-time efficient system configurations for executing a hybrid program.

## 2. Energy-Time Pareto-frontier [136, 137]

### (a) Formulation of the Energy-Time Pareto-frontier [136, 137]

While the proposed approach relies on a measurement-driven analytical model to determine energy-time efficient system configurations, the impact of an execution-time deadline and/or an energy budget on these system configurations is non-obvious. We show that a *distinct Pareto-frontier* consisting of optimal configurations exist for parallel applications executing on heterogeneous clusters, intra-node heterogeneous systems and a hybrid program running on homogeneous clusters. These Pareto-optimal configurations are energy-time efficient as they consume the minimum energy for a given execution time deadline or execute in the minimum possible time for a given energy budget. With the explosion of the configuration space due to heterogeneity, we show that the Pareto-frontier can be analytically formulated using the node performance-to-power ratios (PPR). Furthermore, we show the use-case of our approach to determine what applications will benefit from offloading of execution to co-processors with VPU such as Xeon Phi.

### (b) Pareto-frontier Optimization for Heterogeneous Systems [137]

Given a peak power budget and a service time deadline, we apply our model to analyze the Pareto-frontiers of different heterogeneous mixes. We define and use the power substitution ratios for replacing nodes among



these mixes. We show that the Pareto-frontier can be further improved by replacing low PPR nodes with higher PPR nodes using our power substitution ratio. Additionally, for a given power budget, we show that the Pareto-optimal configurations of heterogeneous clusters have sub-linear energy proportionality and thus scale the energy proportionality wall.

(c) **Pareto-frontier Optimization for Hybrid Programs [136]**

To optimize the Pareto-frontier of hybrid programs, we introduce a new metric, useful computation ratio (UCR) that quantifies the degree of resource contentions and communication overheads in an execution. In addition, we show how system architects and application developers can increase the UCR of Pareto-optimal configurations by balancing resource service demands with resource utilization, to further minimize system inefficiencies.

## 1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 discusses the current state-of-the-art in the execution time and energy performance of heterogeneous computing systems. We first discuss the landscape and taxonomy of heterogeneous computing systems. Next we discuss the different approaches in the state-of-the-art pertaining to energy-time performance models and analysis, and compare these approaches against our approach. Lastly, we discuss the impact of heterogeneity and the limitations of the current approaches for understanding the energy-time performance including simulations and analytical models.

The proposed approach and the core model to determine the energy-time performance of multi-core heterogeneous clusters is presented in Chapter 3. The proposed core model is scalable for different types of heterogeneity and is formulated

using parametric values obtained from baseline measurements of the application for better accuracy. We first present an overview of the core model followed by the input parameters to the model. Next, we present the derivation of the execution time and energy model. Lastly, we discuss the validation results against direct measurements of execution time and energy usage on a heterogeneous cluster with ARM Cortex-A9, AMD Opteron K10 and Intel Xeon E5 multi-core nodes, for a diverse range of parallel applications.

Chapter 4 shows the application of our core model to analyse the time-energy efficiency of different inter-node heterogeneous configurations under a given service time deadline and energy budget. We first evaluate the impact of multiple degrees of heterogeneity on energy efficiency. Next, we present the performance-to-power ratio (PPR) of the inter-node heterogeneous system and analytically determine the impact of PPR on the sweet region. Lastly, we show how to apply our approach to choose energy-time efficient cluster mixes constrained by a power budget followed by an analysis of whether heterogeneous clusters are more energy proportional.

Chapter 5 shows the scalability of the proposed core model by extending it to determine energy-time optimal system configurations for intra-node heterogeneous systems with VPUs. We present the extensions to the model and address the effects of thread level parallelism both within and across cores by considering inter and intra-core resource overlaps, memory contention among threads within a core and contention across multiple cores. Next, we show the validation results of the model using an Intel Xeon Phi coprocessor to represent intra-node heterogeneous system with VPU. Lastly, we discuss how the extended model can be applied to determine energy-time efficient configurations for systems with MIC architecture and how users of such systems can determine whether to offload program execution on coprocessors based on the PPR metric.

Additional scalability of the core model is shown by extending it to determine

energy-time optimal system configurations for hybrid programs in Chapter 6. We first present the model extensions to address the effects of using both distributed-memory and shared-memory communication by considering inter and intra-node resource overlaps, memory contention among cores within a node and network contention across multiple nodes. Next, we present the validation results for a range of hybrid (OpenMP+MPI) programs from different domains such as non-linear partial differential equation solvers, electronic structure calculations and computational simulation for fluid dynamics. Lastly, this chapter discusses the application of our extended model for hybrid programs to determine energy-time efficient system configurations the form a Pareto frontier and the application of the Useful Computation Ratio (UCR) metric to further optimize this Pareto frontier.

We summarize and present the details of future work in Chapter 7.



# Chapter 2

## Related Work

In this chapter, we discuss the heterogeneous computing landscape and the state-of-the-art with respect to energy-time performance analysis among these computing paradigms. Firstly, we categorize the different levels of heterogeneity in Section 2.1. There are several approaches to analyze the energy-time performance of heterogeneous systems and are discussed in Section 2.2. An important aspect of energy-time performance is improving the energy efficiency which has motivated the need for heterogeneous systems. Thus, we discuss the various approaches in the state-of-the-art that target improving energy efficiency in Section 2.2.1, followed by a discussion on energy proportionality in Section 2.2.2, and lastly we discuss the various performance analysis approaches for analyzing execution-time performance in Section 2.2.3. Next, we relate the research done on heterogeneous computing systems with the works on energy-time performance and contrast them with this thesis in Section 2.3 and then summarize in Section 2.4.

### 2.1 Heterogeneous Computing Systems

Heterogeneous computing is defined as [89]:

*...the well-orchestrated and coordinated effective use of a suite of diverse high-performance machines (including parallel machines) to provide superspeed processing for computationally demanding tasks with diverse computing needs.*

Heterogeneous Computing Systems (HCS) is heterogeneity in all manifestations of computing, including processors, networks, programming, protocols and all combinations of these coming together to deliver a positive impact. While HCS is all encompassing, we discuss the landscape of HCS with respect to computing systems and programming models pertaining to this thesis.

“Moore’s law” has enabled a continuous growth in the performance of computer systems over the last 50 years, by increasing the number of cores on a single chip. However the advantages of this multicore scaling phenomenon is limited to a large degree due to “dark silicon”, regardless of chip organization and topology [55]. To leverage the continued growth in performance and overcome the limitations of non-usability of components within a chip due to dark silicon, the most promising road ahead is to use a cluster system of heterogeneous nodes.

Recent trend in heterogeneous systems can be seen by the development of the Heterogeneous System Architecture (HSA) Standard in 2012, which is founded by an industrial consortium of companies including AMD, Texas Instruments, Qualcomm, ARM among many others [4]. Members of HSA are designing a heterogeneous computing ecosystem, to enable combined processing on both CPU and GPU nodes with higher memory bandwidth at lower power consumption. This consortium is not only focusing on the hardware platform but also the software development ecosystem, including the HSA Intermediate Language (HSAIL) to support a diverse set of high-level programming languages. Thus creating the foundation for the paradigm shift in general purpose computing.

Exploiting heterogeneity for energy-efficient executions has been studied to a small extent within the scientific community, with heterogeneity being defined at different levels. Traditionally, the advent of multi-core systems brought about the shared-memory programming model, but the dark silicon phenomenon advocates the use of distributed-memory model. However, increasingly programs are becoming hybrid to take the dual-advantage of exploiting inter-node distributed-memory scalability and intra-node shared-memory performance in a cluster system. Furthermore, at the system level, heterogeneity maybe further classified into (i) within a chip or (ii) within a single node or (iii) across nodes. This classification of the different levels of heterogeneity at both applications and systems is illustrated in Figure 2.1.

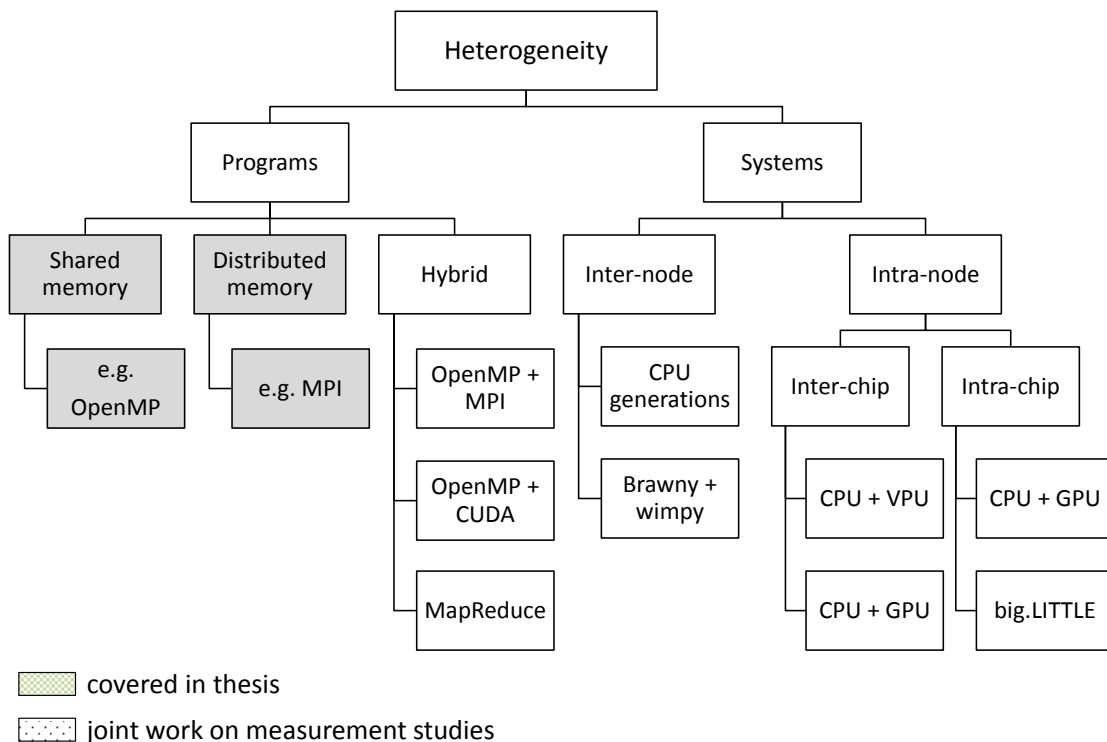


Figure 2.1: A classification of heterogeneous computing systems

### 2.1.1 Systems

#### Inter-node Heterogeneity

Inter-node heterogeneous systems are ubiquitous in datacenters today and is a direct consequence of (i) replacing failed systems with newer versions of CPU architecture, memory or higher operating clock frequency, (ii) scaling the capacity leads to procuring latest server processors that differ in architecture from the existing ones, and (iii) cost-efficiency leading to the purchase of low-end servers complementing existing high-performance high-cost servers. While these and many other reasons are paving the way for inter-node heterogeneity in datacenters, it leads to a number of challenges, namely, (i) necessitates novel work-scheduling strategies that are heterogeneity-aware, (ii) need for efficient cluster configuration to trade-off performance and save energy, and (iii) the need for a resource-aware middleware to efficiently utilize the underlying heterogeneous resources. While many studies explore the first challenge of workload provisioning and scheduling [35, 49, 109, 175, 176], this thesis addresses the second challenge and aims at determining energy-efficient configurations among the large configuration space due to inter-node heterogeneity.

The power-efficiency and form-factor of processors used in mobile devices have attracted the attention of several leading server vendors to use them to design their next-generation servers [17, 143, 152]. These processors are also referred to as “wimpy nodes” in contrast to the traditional high-performance high-power (brawny) nodes. This trade-off between the performance and power with respect to wimpy, brawny, GPU and VPU processors is illustrated in Figure 2.2<sup>1</sup>.

At a cluster level, Whare-Map [109] explores performance improvements by us-

---

<sup>1</sup>This diagram is indicative of the general trend. This trend is derived using a Cortex-A9 processor representing wimpy, AMD Opteron processor for brawny, Nvidia Tesla for GPU and Intel Xeon Phi processor for VPU.



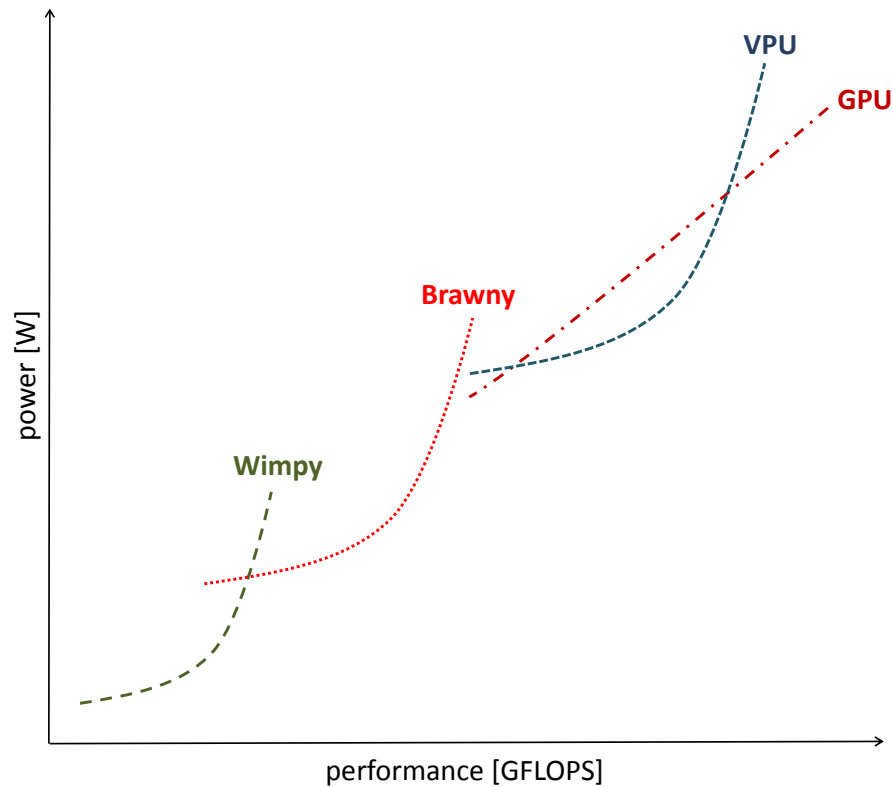


Figure 2.2: Power-performance of processors

ing existing heterogeneity in modern warehouse scale computers, while we model and analyze the impact of heterogeneity due to diverse performance-to-power ratios. Chun et al. [44] study the feasibility and potential of hybrid datacenters with Xeon and Atom platforms, but considering only one node of each type. In contrast, we explore heterogeneous mixes of several nodes. Nathuji et al. [126] exploit heterogeneity across platforms for power efficiency and their approach uses throughput as a measure of performance. At inter-node level, Heath et al. [74] proposed a modeling technique to optimize energy for a cluster of nodes with different CPUs and network capabilities. They model the request distribution among nodes to balance resource utilization. On the other hand, we consider nodes with diverse PPRs and model the execution time.

Inter-node heterogeneity not only refers to platform-level heterogeneity due to processor with different generations of the same CPU architecture but also refers

to clusters of processors with varied ISA, and performance-to-power ratio. This thesis addresses inter-node heterogeneity arising out of mixing wimpy and brawny processor nodes.

## **Intra-node Heterogeneity**

While inter-node heterogeneity explores different types of processors networked together in a cluster system, intra-node heterogeneity explores different processor systems within a node. In intra-node heterogeneous systems, either the processors have an exclusive memory and communicate with each other using interconnect such as a PCIe (inter-chip) or the processors are on the same chip and communicate using shared-memory (intra-chip).

## **Inter-chip Heterogeneity**

More recently, accelerators such as graphical processing units (GPUs) have seen a phenomenal increase in their performance due to the demand for high-resolution gaming and graphics processing. With the slowing down of “Moore’s law” and dark-silicon limiting the number of active cores in a multi-core processor, mixing CPUs and accelerators seems like a viable solution to scale-up parallel computing performance. One most commonly used approach is to use GPUs as accelerators with the CPU orchestrating the computations transferred to the GPU. Such systems are commonly available in most desktop computers today, where an eight-core CPU processor from Intel/AMD uses a GPU from Nvidia/ATI and communicate over a PCIe. Typically these systems have separate memory subsystem for the CPU and GPU but use a specialized DMA controlled by the CPU to enable high-speed data sharing between these processing units.

Another exciting approach is that of CPU processor communicating using a PCIe with an accelerator consisting of Many Integrated Core (MIC) architec-

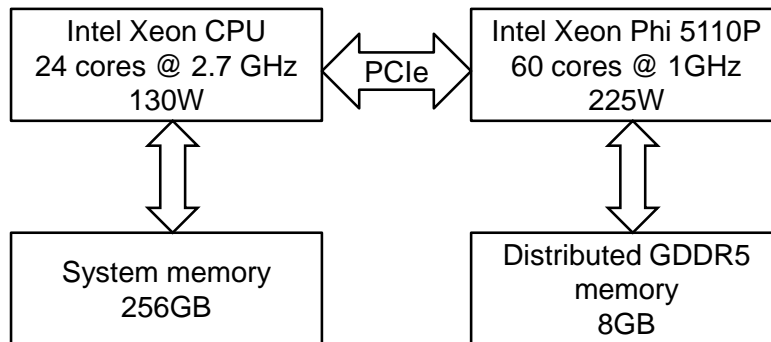


Figure 2.3: Example system with inter-chip heterogeneity, Intel Xeon Phi

ture, this accelerator is also referred to as a co-processor. An example system designed using CPU and MIC coprocessor is the Intel Xeon Phi [43] as shown in Figure 2.3. Multiple Intel Xeon Phi coprocessors can be installed in a single host system. Within a single system, the coprocessors can communicate with each other through the PCIe peer-to-peer interconnects without any intervention from the host. Similarly, the coprocessors can also communicate through a network card such as InfiniBand or Ethernet, without any intervention from the host.

The multiple compute cores on the Xeon Phi are based on the Pentium core architecture and are much smaller and simpler compared to the x86 CPU processing cores of today. However, these simpler cores have been strengthened by using vector processing units that can process vectors up to 512 bits wide. Typically a coprocessor has about 60 cores along with a vector processor per core and supports 64-bit instructions of the x86 ISA. The theoretical peak performance of an Xeon Phi coprocessor is about 1 TFLOP/s in double precision. However, this performance is achieved at the same power consumption as two Xeon CPU processors [73].

While both GPU and the Xeon Phi coprocessor have many simple cores to accelerate performance, it is easier to deploy code on the MIC architecture as the same application code can be easily cross-compiled for the Phi. However, to enable

applications to use the GPU, code needs to be rewritten in CUDA or OpenCL. While this programming difference favors Xeon Phi, it is non-obvious whether the MIC architecture is more attractive or the GPU architecture, as the application performance gains that are achieved on the Xeon Phi by simply recompiling and running code natively are lesser compared to the gains on the GPU.

**MIC performance studies** Fang et al. [59] presented an empirical study on Xeon Phi, stressing its performance limits and relevant performance factors using micro-benchmarks. The system architecture components that were studied in detail were the vector processing cores, the on-chip memory, the off-chip memory, the ring interconnects and the PCI express connection. They also attempted to provide a simplified machine view for performance tuning and application design. Closer to this work, Ramachandran et al. [134] examined the performance of OpenMP version of NPB on Xeon Phi, compared the performance with traditional Xeon CPUs and identified some issues that may degrade the performance. In contrast, Schmidl et al. [144] and Vladimirov et al. [165] evaluated the performance of OpenMP applications on Xeon Phi and compared the performance of the coprocessor with a Xeon-based compute node. While these works are purely measurement based studies, we use an analytical model to analyze MIC architecture performance.

Fang et al. [58] used benchmarking to evaluate the performance of various optimization techniques with a focus on guiding kernel design. By using a set of micro-benchmarks, they characterized the three major components of the Phi architecture - cores, memory, and interconnect. They also synthesized a set of four machine-centric optimization guidelines and a simplified machine model for facilitating kernel design and performance tuning on the Xeon Phi. Ramos et al. [138] proposed communication models for cache-coherent MIC architecture and

applied these models to optimize algorithms with complex data exchanges. While these works are related to optimizing the kernel or specific algorithms, we focus on modeling the performance of parallel programs on the MIC architecture.

Heinecke et al. [75] implemented the Linpack benchmark on single and multi-node systems based on Xeon Phi coprocessors in both native and hybrid configurations. Their implementation on Knights Corner employs novel dynamic scheduling and achieves close to 80% efficiency. Si et al. [148] developed a multithreaded MPI implementation on many-core environments such as Xeon Phi to coordinate the runtime engine of the threads and share idle threads with the application. In contrast, we focus on the model the impact of thread affinity on the performance to determine time-efficient system configurations to execute parallel programs.

### **Intra-chip Heterogeneity**

While inter-node heterogeneity exploits the existence of a brawny and wimpy node at a cluster level, intra-chip heterogeneity uses a multi-core architecture with shared-memory where the multiple cores differ in their performance and power usage [70]. Traditionally multi-core processors had symmetric cores, but with the increasing need for energy-proportional servers, has motivated analyzing multi-core architectures with asymmetric cores. Energy proportionality requires the power consumed by servers during idling to be as low as possible and expects the power to increase linearly with server utilization. To enable this within a server, KnightShift [170] tightly couples a single high-performance processor with a low-power processor to enable two energy-efficient operating modes. The KnightShift hardware consists of a low-power low-performance compute node, called the Knight, paired with a high-power high-performance server. Both the Knight and primary server can be independently powered on and off. Both the Knight and primary server share a common data disk and are able to communicate with one

another through traditional network interface.

Figure 2.4 illustrates the system architecture of ARM big.LITTLE, having four

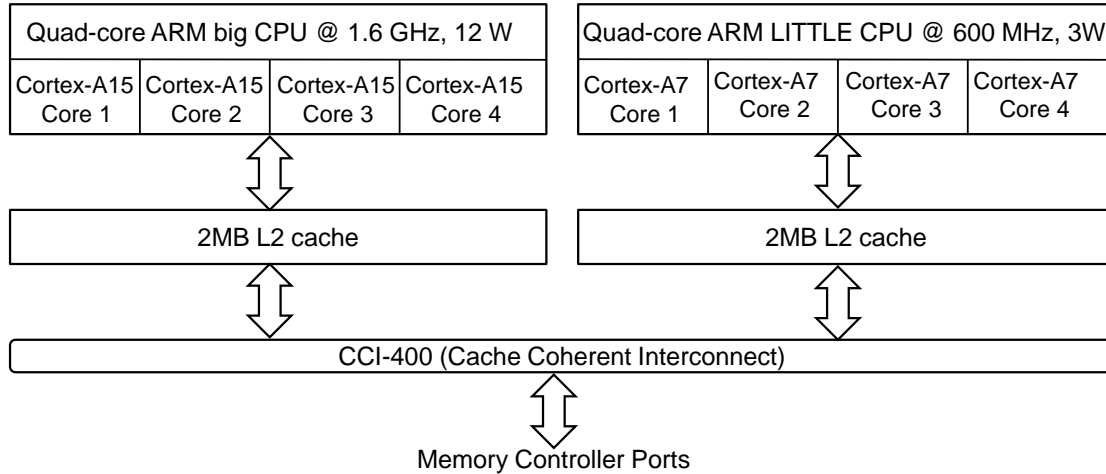


Figure 2.4: Example system with intra-chip heterogeneity, ARM big.LITTLE

big cores of Cortex-A15 and four LITTLE cores with Cortex-A7 processor. Other studies explore intra-chip heterogeneity [22, 91], much like the ARM big.LITTLE architecture and switch to low-power efficient cores during low-utilization periods. Van Craeynest and Eeckhout propose a study of single ISA multi-core heterogeneity to understand the extent to which both system-level throughput and per-program performance can be simultaneously satisfied [160]. Their study proposes the use of analytical models to explore heterogeneity across cores within a single chip, wherein the heterogeneity stems from cores with different memory subsystems and pipelines but having the same ISA. Dynamic core heterogeneity is investigated for programs that alternate between regions with high thread-level parallelism and instruction-level parallelism [131]. This is also a heterogeneous architecture study but within a single node. Here, the proposed reconfigurable multicore architecture, increases the achievable instruction-level parallelism by forming coalitions of two or more cores.

While all of these explore intra-chip heterogeneity using asymmetric cores of

the same ISA, another approach is to use CPU designs with different ISA cores or accelerator cores on the same chip. The Cell Broadband Engine (CellBE) processor from the Sony/Toshiba/IBM consortium, design is based on a main core from PowerPC and eight auxiliary cores also referred to as Synergistic Processing Elements. While the main core uses the traditional cache-coherent memory hierarchy to access system memory, the auxiliary cores use specialized high-bandwidth DMA to access the system memory [130]. Another intra-chip heterogeneous system is the recently introduced Jetson TK1 system [53] with CPU and GPU on the same chip as illustrated in Figure 2.5.

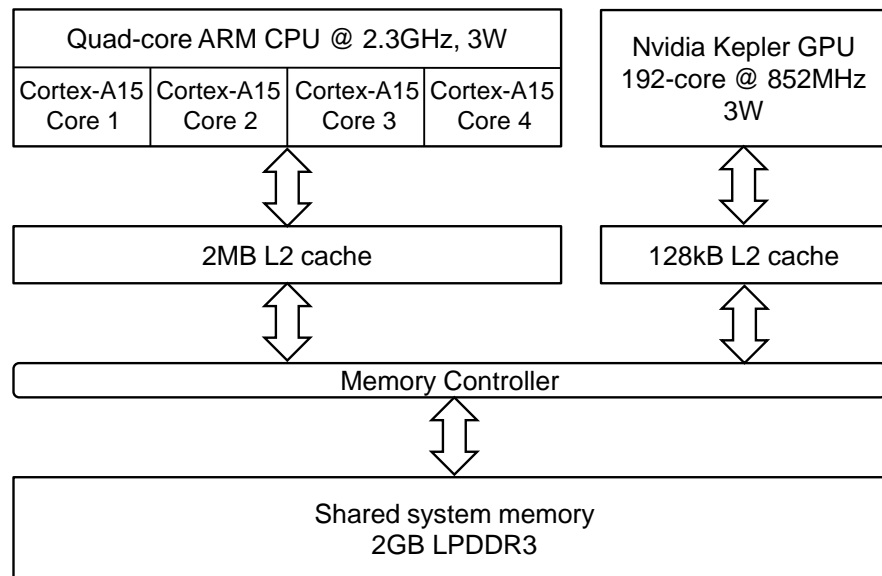


Figure 2.5: Example system with intra-chip heterogeneity, Nvidia Jetson TK1

### 2.1.2 Programs

The advent of heterogeneity at different levels of hardware hierarchy, has introduced a paradigm shift in the traditional parallel programming models being used. While instruction-level parallelism is feasible with the multi-stage out-of-order pipelines in processors today, thread-level parallelism is often exploited using the multi-core architectures of processors. However, the dark silicon era is limiting the

thermal and power budgets of a multi-core processing node [55] and paving the way for distributed systems. But, it is well known that High Performance Computing (HPC) applications are limited by data access bottlenecks. To simultaneously exploit the computational capabilities of multi-core systems and to scale complex HPC applications, application developers are increasingly using hybrid programming models. A hybrid programming model utilizes both distributed-memory across nodes for scalability and shared-memory within a node for improving performance [27, 105, 148].

To exploit thread-level parallelism in multi-core processors, many multi-threaded programming models have evolved over the years. While languages such as C, C++ and Fortran provide multi-threaded support with variegated implementations, many other programming models such as OpenMP, MPI and Cilk abstract threading details from the application developer. OpenMP is a popular programming language that provides compiler directives and subroutine calls, for ease of programming parallel programs. All OpenMP programs distribute the workload among all the spawned threads and use a fork-join execution model. While it is very easy to use, OpenMP has a large thread management overhead and is inefficient across multiple processor systems and is limited to thread-level parallelism within multi-cores.

For exploiting parallelism across processors using distributed memory architecture, MPI [60] is a popular standard used by HPC application developers worldwide. While MPI is a message passing interface standard, there are many implementations of this standard by different academic research institutes and industries, such as Open MPI [123], MVAPICH [125] and Intel MPI [100]. In the MPI programming model, users manage the communication between different processes by calling library routines to send and receive messages. Besides, users completely control the workload distribution and process synchronization, which



permits the optimization of data locality and workflow. Since it has the advantages of excellent scalability and portability, MPI is an appropriate programming model to accomplish the communication functionality between different compute nodes of large GPU clusters [104].

### Hybrid Programs

While MPI programming is useful for distributed cluster of nodes and OpenMP is popular for shared-memory multi-core system, recently hybrid programming is becoming popular among HPC application developers. A hybrid parallel program is partitioned into a variable number of logical parallel processes and parallel threads. For a given hybrid program and a multi-node system with multi-core nodes operating at different core clock frequencies, there is a large system configuration space for executing these logical processes and threads. As the resource demands in a hybrid parallel program varies with its problem size, these resource demands have to be mapped onto different system configurations to minimize resource contention and runtime overheads. There are many different programming models within the hybrid programming paradigm to cater to the different heterogeneous systems today.

Figure 2.6 illustrates the hybrid programming model for heterogeneous clusters with multi-core CPU, VPU and GPU. While the hybrid MPI+CUDA programming model uses a distributed cluster of GPUs, the hybrid program with OpenMP, MPI and CUDA utilize not only the GPUs but also exploit the multiple cores within the CPU attached to the GPU [104].

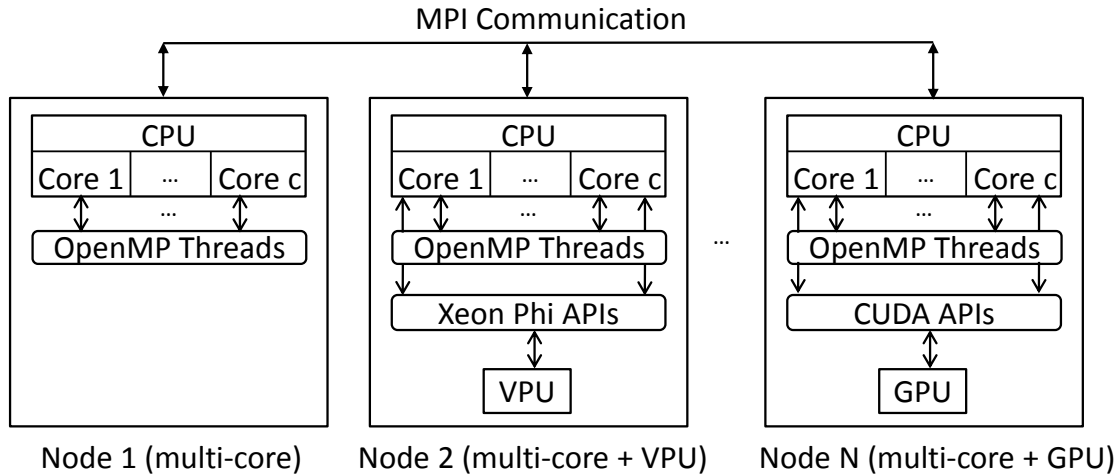


Figure 2.6: Hybrid program with MPI, OpenMP, Xeon Phi API and CUDA

## 2.2 Approaches

In this section we discuss the various approaches to achieve energy efficiency in distributed systems and also discuss the approaches used to analyse and infer execution time of programs in systems along with some methods to model communication overheads.

Many research studies explore micro-architectural models and dynamic algorithms for both power management [87, 99, 119], and energy efficiency of heterogeneous clusters [24, 177, 179]. While these approaches are dynamic and are applicable during run-time, the approach proposed in this thesis determines static energy-efficient configurations. For a given power budget, a proposed cluster-in-a-box production system using *only* low-power CPUs improves the performance per watt-hour [154]. However, we analyze the energy-time performance for a heterogeneous *mix* of nodes with diverse PPRs.

More recently, with the emergence of ARM big.LITTLE, a hierarchical power management approach to optimize the performance per watt within a thermal-design power budget is proposed by Muthukaruppan et al. [124]. Mixing CPU and

accelerators is another method used to improve the efficiency per watt at node-level [166]. While these works focus on *intra-chip* and *intra-node* heterogeneity, our proposed approach is applicable for *inter-node* heterogeneity and addresses multi-ISA heterogeneity. Nathuji et al. [126] propose an intelligent workload allocation method to exploit across-platform heterogeneity for power efficiency. However, we analyze the energy-time performance including energy proportionality of heterogeneous systems having nodes with diverse PPRs.

### 2.2.1 Energy Efficiency

There are many techniques to improve energy efficiency of systems. Dynamically changing the frequency of the microprocessor to reduce energy consumption for a given performance requirement is explored by Weiser et al. [168]. There has been a lot of research done in the area of energy aware dynamic scheduling algorithms, including dynamic speed scaling [169] which decreases the frequency at lower workloads, thus reducing energy. Dynamic Voltage and Frequency Scaling (DVFS) has been widely employed to achieve energy efficiency [76, 94]. Minimization of energy using a system level energy efficiency ratio for workloads with real-time constraints considering the characteristics of the core and the precedence relationship among logical processes has been proposed by [41, 178]. However, energy inefficiencies occur with all the above techniques as they rely on dynamic adaptations only to frequencies which do not alter the inherent device characteristics. These inherent device characteristics are implemented for nominal frequency and hence consume more energy [39]. These techniques are also not advantageous because of the utilization wall whereby the percentage of a chip that can be active is decreasing exponentially, a concept referred as dark silicon [55].

Energy-aware load balancing is another system architecture technique that can be used effectively to reduce the energy consumption [56] but this technique has

not been analyzed and explored for heterogeneous system architectures. Similarly [50, 150] analyze iso-energy efficiency for multiple cores of processors for complex scientific computations and propose analytical models for the same. Again, this analysis is not being done for heterogeneous processor systems.

Dynamic voltage scaling to mitigate pipeline imbalances within a core is proposed by [95]. PEPON [147] discusses power distribution among multiple-cores to maximize performance without exceeding a given power budget. We are also exploring ways to maximize performance for a given power budget but for heterogeneous systems. Algorithms for dynamic power management of clusters are discussed in [40, 87, 119]. These techniques complement our approach as we do not propose dynamic power management techniques within a core or node. A cluster-in-a-box production system using *only* low-power CPUs has been designed by [154], to improve the performance per watt-hour, for a given power budget. We also analyze the improvements in energy reductions for a given power budget but with a heterogeneous mix of *both* high-performance and low-power CPU nodes.

A modeling approach to determine a configuration of cores and core clock frequency that optimizes energy within a low-power ARM node is proposed in [158]. In contrast, our core model is applicable for systems having both low-power and high-performance nodes. Our proposal aims at lowering the energy by removing some of the causes of energy inefficiencies in the system. Energy inefficiencies in the system occur due to the imbalance of workload demands on system resources. For example, a memory bound application will not achieve best performance at neither the highest nor the lowest frequency but at an optimal frequency between the two and hence will consume the lowest energy at that optimal frequency [149].

Contention for resources not only slows down the execution of a application incurring performance losses but also increases inefficiencies in energy consumption because of the cores dissipating higher power without doing useful work [120].

Recent studies also show that the utilization of data center servers is only in the range of 10% to 50% which lead to energy inefficiencies because of under-utilization [11, 30]. Our proposal aims at improving the resource contention bottlenecks in the system and increasing the utilization of the cores to minimize energy inefficiencies. To this end we propose a measurement-based analytical model as our core model, which predicts heterogeneous configurations of nodes to be used for a given workload to achieve an optimum set of sweet spots for energy versus performance.

Next, we discuss the different metrics that are used to measure energy-performance of computing systems.

### 2.2.2 Energy Proportionality

In an ideal energy-proportional system, the system consumes no power when idle and its power consumption grows linearly with the amount of utilization of its resources. For example, the ideal energy-proportional system consumes 10% of its peak power at 10% utilization as shown in Figure 2.7. However, the actual proportionality of servers or clusters may be super-linear or sub-linear. Multiple metrics have been proposed by researchers to quantify the energy proportionality of individual server nodes. The first metric, dynamic power range (DPR), was proposed by Barroso et al. [30],

$$DPR = 100 - P_{idle}(\%)$$

where  $P_{idle}$  is the percentage of peak power consumed by the node while idling with zero utilization. Another similar metric that captures a server's power consumption during idle periods with respect to its peak power is the idle-to-peak

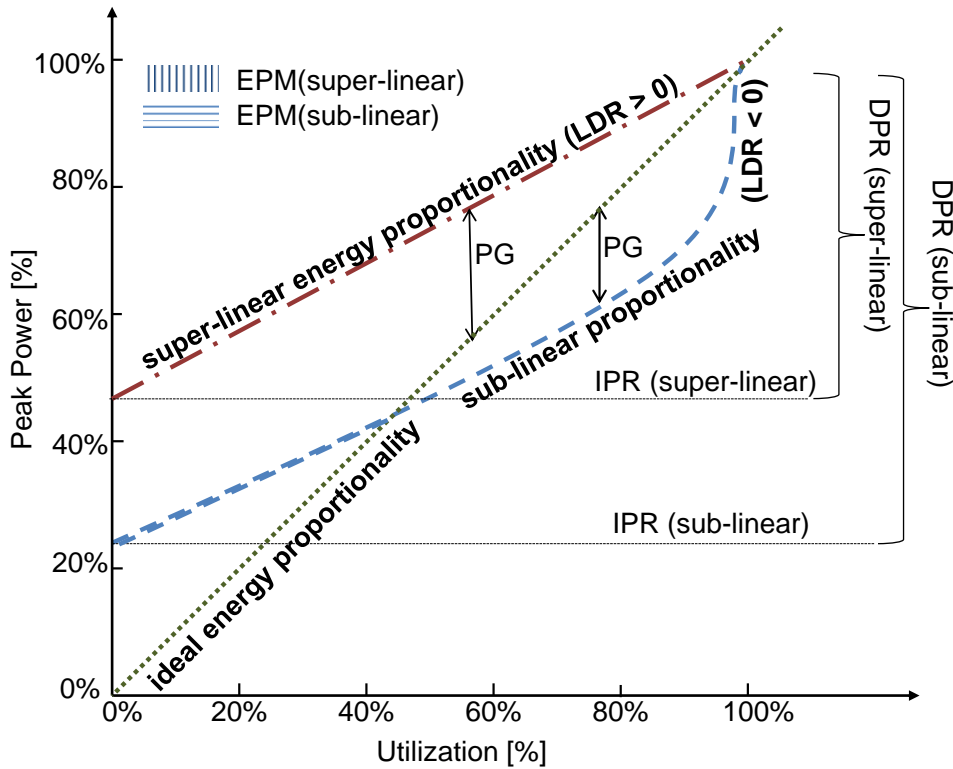


Figure 2.7: Energy proportionality metric relationships

power ratio (IPR) metric [162],

$$IPR = \frac{P_{idle}}{P_{peak}}$$

While both the DPR and IPR metrics capture server power at zero utilization, server's power consumption is known to increase non-linearly with utilization [156], which is not accounted for by these metrics. Since the majority of datacenters over-provision their servers to achieve reasonable QoS at peak utilization, most servers operate at 30% utilization on an average [29]. Hence, a more meaningful metric would be to capture the ratio between power consumption at 30% and 100% utilization. However, datacenter operators may find it convenient to shift workloads accordingly if they have prior knowledge of server proportionality at different utilization levels.

As shown in Figure 2.7, the energy proportionality metric (EPM) proposed by Ryckbosch et al. [141] measures server's power consumption at different utilization levels using the area between server's power consumption and the ideal power consumption curve,

$$EPM = 1 - \frac{\int_0^{100} P_{server} \cdot du - \int_0^{100} P_{ideal} \cdot du}{\int_0^{100} P_{ideal} \cdot du}$$

where  $P_{ideal}$  represents an ideal energy-proportional system. Varsamopoulos et al. [162] propose a metric called the Linear Deviation Ratio (LDR) to account for the linearity of server's energy proportionality curve across utilization and is defined as,

$$LDR = \max_u \frac{|P(u) - ((P_{peak} - P_{idle})u + P_{idle})|}{(P_{peak} - P_{idle})u + P_{idle}}$$

The  $\max_u$  is the maximum value computed using absolute value comparisons across utilization, such that the LDR value obtained retains the sign of the maximum deviation. While both EPM and LDR account for proportionality across utilization levels, the results are expressed as a single value. For the EPM metric, a value of one indicates that server consumes power proportional to its load, while a value of zero indicates that the server consumes a constant amount of power irrespective of its load [141]. For LDR, lower values indicate a more linear system, negative values represent *sub-linear energy proportionality* and positive values represent *super-linear proportionality*. As these metrics are aggregated as a single value, it limits the analysis of energy-efficient configurations across cluster utilization levels. In contrast, the Proportionality Gap (PG) metric proposed by Wong et al. [170], is defined at each utilization as,

$$PG(u) = \frac{P(u)_{server} - P(u)_{ideal}}{P(u)_{ideal}}$$

A lower value of PG is indicative of a more energy-proportional server.

Figure 2.7 summarizes the relationships among recent energy proportionality metrics.

A common metric to measure energy-efficiency is using the performance-to-power ratio (PPR) that factors the throughput of the workload per unit power across utilization levels and is defined as,

$$PPR(u) = \frac{\text{Throughput}[\text{operations}/s]}{\text{Power}[W]}$$

where throughput denotes the number of useful operations performed by the system per unit time. This metric is also used in SPEC benchmark [151]. Figure 2.8 shows the energy-time performance of CPU, GPU and the VPU (MIC architecture) with respect to the PPR metric.

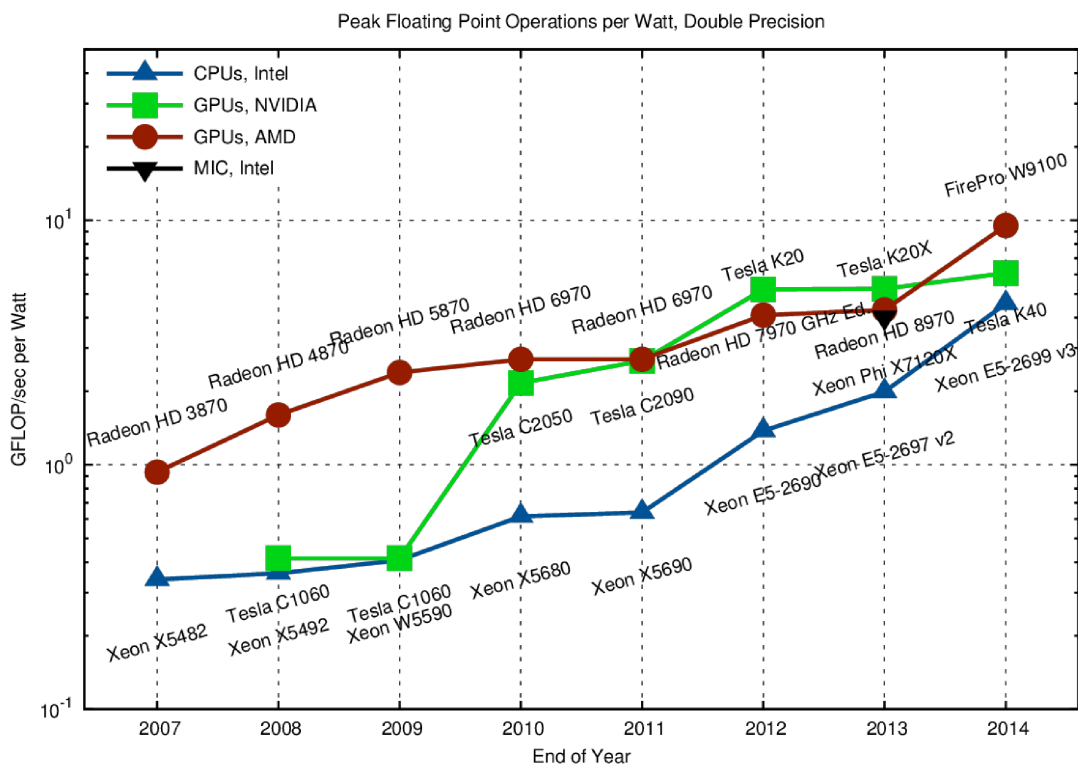


Figure 2.8: Energy-performance of CPU, GPU and MIC [140]



Energy proportionality studies of warehouse scale computers and strategies to improve non-peak power efficiency has been widely explored [57, 103, 153]. There are many research studies that employ dynamic strategies using software driven transitions to exploit multiple low-power server modes [62, 115, 116]. Such strategies and techniques complement the energy-time performance analysis in this thesis and can be applied to further reduce the inefficiencies of heterogeneous cluster mixes. While Hsu et al. [81] question the linearity of the energy proportionality curve and show that most modern servers follow a quadratic trend, we show that the energy proportionality metric alone does not suffice to study clusters consisting of nodes with diverse PPRs.

Barroso et al. [29] suggest that energy proportionality at the system level cannot be achieved through CPU optimizations alone, but instead requires improvements across all components, such as memory and network devices. Energy proportionality of server-level memories for datacenter workloads and datacenter network architectures have been proposed [107, 145], but these studies are tangential to our work, as we study the impact of inter-node heterogeneity on cluster-wide energy proportionality. With servers exhibiting lesser proportionality gaps at higher utilization levels, workload co-location strategies have been proposed to increase cluster or rack utilization [96, 110, 128, 174]. The implications of high energy-proportional servers such as KnightShift [170] on cluster wide energy proportionality was studied [172] and compared with cluster-level packing techniques such as auto-scale [63]. However, this thesis proposes a measurement-driven analytical model, and analyzes energy proportionality for inter-node heterogeneous clusters consisting of node mixes with diverse PPRs

### 2.2.3 Time Performance

Current approaches to analyze parallel programs mainly include instrumentation or application profiling based measurement techniques which trace the complete execution of the program on a particular hardware system to identify both application and hardware bottlenecks [12, 45, 108, 122]. However, they are generally intrusive and difficult to generalize across programming languages. Another approach to understand application hot-spots on prospective hardware is the use of cycle accurate micro-architecture level simulators [25, 32, 85]. However, analyzing application executions even with reasonable input sizes is very time-consuming [69]. This thesis presents an approach to determine time and energy efficient system configurations for executing a parallel program using a measurement-driven analytical model. The proposed analytical model is formulated using parametric values obtained from baseline executions of the application to measure workload and architectural artefacts.

Other alternative approaches to predict performance include statistical methods that rely on black-box regression to infer dependencies between hardware parameters and application performance [28, 93, 155]. Our approach is not black-boxed and thus enables analytical prediction of the impact of changing different system components such as memory/network bandwidth on program execution time performance. More recently, at an algorithm level, asymptotic analysis based modeling techniques are used to derive trade-offs between computation and communication [47, 48]. However, the approach presented in this thesis is at a lower level of abstraction, so that insights into both application and architecture bottlenecks can be inferred.

A summary of the existing approaches and the measurement-driven modeling approach proposed in this thesis is outlined in Table 2.1. Next, we discuss related

research work that models execution time of parallel programs.

	Profiling	Simulation	Statistical prediction	Analytical modeling	Measurement-based analytical model (this thesis)
Accuracy	Yes	Yes	Yes	No	Yes
Non-intrusiveness	No	Yes	No	Yes	Yes
Generic application	No	No	No	Yes	Yes
Related work	[45, 108, 122]	[25, 32, 85]	[28, 93, 163]	[42, 78, 173]	[136, 137, 158]

Table 2.1: Comparison of performance analysis approaches

## Execution Time Models

A popular model to estimate speed-up when executing a parallel program is Amdahl's law [18] which accounts for the speedup as a function of the sequential fraction of a program and the number of processors.

$$S(n, f) = \frac{1}{f + \frac{1-f}{n}}$$

where  $S$  is the speedup,  $f$  the sequential fraction and  $n$  the number of processors.

However, the above does not account for the dependencies of speed-up on problem size. Gustafson [71] shows that the scalability of a program increases with the size of the problem being evaluated. Now, the speedup  $S$  is also a factor of  $f(k)$  wherein, the sequential fraction  $f$ , depends inversely on the problem size  $k$ , and is small for large workloads.

$$S(n, f, k) = n + (n - 1)f(k)$$

But, both Amdahl's and Gustafson's model do not consider resource contention. Re-evaluating Amdahl's law for multi-core processors, Hill and Marty [78] have shown the impact of the different resource allocation schemes among cores on the speedup. They introduce the *perf(r)* and use it to quantify the sequential fraction

of a program, by using a supercore processor to execute the sequential fraction. The performance speedup of the sequential fraction using this supercore processor, by combining  $r$  cores is,  $\sqrt{r}$ . So the speed-up becomes,

$$S(f, n, r) = \frac{1}{\frac{f}{perf(r)} + \frac{1-f}{n}}$$

However, the limitation of Amdahl, Gustafson and Hill-Marty approach is that all these models are a function of the sequential fraction of a program, which is highly dependent on the execution platform and cannot be easily inferred.

There are other models based on the theory of parallelism, where no overhead is considered for parallelization, and the parallelism of a program depends on the number of work units completed in unit time. One such model is proposed by Eager et al. [54], gives the theoretical bounds on the speed-up of parallel programs. The upper bound for speedup is given as:

$$S_{upperbound} = \min(n, \pi)$$

where  $n$  is the number of parallel processors to execute the program and  $\pi$  is the inherent parallelism of the program. But, this model assumes that the processor does not stay idle when the parallel work unit cannot be executed because of resource sharing. While the inherent parallelism [83] is the number of parallel tasks in a program that can execute concurrently, explicit parallelism involves specific constructs in the programming language to specify the parallel tasks of a program. The inherent parallelism of a program is not completely exploited because of resource contentions (shared memory) and losses in parallelism due to communication and house-keeping overheads [34].

To quantify the execution time of parallel programs, it is important to identify

patterns of computation. Previous work has identified computation and communication patterns that are distinct among thirteen classes of parallel programs [23]. However, it is also important to quantify the impact of these patterns of computation into inherent parallelism and exploited parallelism.

A trace-driven analytical model to account for resource contention among multiple cores is proposed by Tudor et al. [157].

$$S(m, n) = \frac{A(m, n)}{1 + \omega(n)}$$

where  $S(m, n)$  is the speedup which depends on both  $A(m, n)$  denoting the active threads of the parallel program and  $\omega(n)$  denoting the exploited parallelism. This model not only accounts for parallelism loss due to memory contention but also parallelism loss due to data dependencies. From the perspective of the execution time of a program being defined by the bottleneck resource in a system, Tudor et al. classify program execution to be bound by either CPU core, memory or I/O resource [158].

$$T(n) = \max\left(\frac{C}{n}, I\right)$$

where  $T(n)$  is the execution of a parallel program on an n-core processor, and  $C(n)$  denotes the CPU response time and  $I$  denotes the I/O response time. The  $C(n)$  maybe computed using the useful work cycles of a program and the stall cycles of the program. Their analysis incorporates the memory response time of the system, by considering the stall cycles due to memory contention as a separate component, which can be modelled by measuring the number of last level cache misses and using an M/G/1 queueing system.

This type of a trace-driven modeling approach for predicting the execution time

of a parallel program in a shared-memory system may be extended for distributed systems. In our core model described in Chapter 3, we extend the above model and use the increase in memory contention stall cycles as a characteristic of memory response time instead of using the last-level cache misses. However, we assume there is no communication overhead among the distributed nodes. The main challenge in distributed system then becomes the modeling of the communication patterns among the nodes and the execution time overhead because of this.

### **Modeling Communication**

In this section, we present some of the widely used approaches to model the communication overhead among nodes. The previous section accounts for the execution time assuming an ideal case wherein the communication overhead among nodes completely overlaps with the time required for computation. An initial theoretical model, which conceptualizes the communication delay between a processor's computation and the data needed by the processor is the PRAM model. For the block PRAM model [14, 15], the data access time is modelled as

$$T_{data-access} = l + b$$

where  $l$  is the start-up time and  $b$  is the block access time. However, a limitation of the block PRAM model is it assumes uniform access time for all data which is not realistic. The Bulk Synchronously Parallel (BSP) model has been proposed, wherein the communication among the distributed processing nodes is bundled together in a step-oriented manner instead of point-to-point communication. The computing processes do local computations, communicate with each other when need of shared data, and then do a barrier synchronization to terminate the communication. This is termed as a super-step and the time to complete a super-step

is modelled as:

$$T_{super-step} = \max_{processors} w_i + h \cdot g + l$$

where  $w_i$  is the time required to do local computation,  $g$  is the time required for global communication for shared data, and  $l$  is the time for barrier synchronization. To overcome some of the limitations of the BSP model such as assuming there is support in hardware for barrier synchronization, another approach to model the communication among distributed nodes is the LogP model [46]. This model not only accounts for communication costs as a factor of latency but also for the overhead inside of a node which is the non-computational time of a CPU to initiate and terminate the communication. The parameters used in the LogP model are:

- L: This is the maximum latency of sending a short packet from one node to the other.
- o : This is the CPU overhead to process the packet
- g: The minimum gap between two consecutive packets.
- P: The total number of nodes in the distributed system

However, a limitation of this model is it does not account for heterogeneity among nodes that have different overheads and different bandwidths among interconnection links between nodes. An extension to the LogP model is the LogGP [16], which accounts for the transmission delay of the packets based on the bandwidth of the interconnection network among the nodes. It includes another parameter, G in addition to those of the LogP model, wherein G denotes the gap per byte of the packet. The reciprocal of G denotes the available network bandwidth for a node to transmit a large packet.

However, the challenge of modeling the execution time of a parallel program is computing the overlap fraction, wherein for some program parts, the communication overhead overlaps with the computation, while the communication overhead does not overlap for other parts of the program execution. The communication model proposed in Chapter 6 of this thesis addresses this challenge and models both intra and inter-node communication considering computational overlaps.

## 2.3 Heterogeneity and Energy-time Performance

The advent of heterogeneity in systems and programs introduces new research challenges for analyzing the energy-time performance of parallel programs. Section 2.1 discusses the heterogeneous computing landscape and Section 2.2 elucidates the various approaches for energy-time performance analysis. In this section, we compare the related work with respect to both heterogeneity and performance analyses approaches, and position this thesis with respect to the existing works.

Heterogeneous Computing		Related work	This Thesis
Systems	CPU generations	Workload Scheduling energy [126], time [109, 176]	1. energy-time performance 2. measurement-based analytical model 3. models inter and intra-node overlap, contention and communication
	Brawny + wimpy	Simulation-based [24, 38, 177, 179]	
		Analytical model energy [74], time [175]	
CPU + VPU	Application profiling [92, 97, 111, 135, 146]		
Programs	OpenMP + MPI	Statistical approach [28, 93, 98, 155]	4. determines Pareto-optimal system configurations 5. covers inter- and intra-node heterogeneous systems and hybrid parallel programs
		Dynamic voltage frequency scaling [65, 80, 86, 121]	
	MPI + CUDA	Application profiling [104, 129]	

Table 2.2: Heterogeneity and energy-time performance



## 2.4 Summary

The paradigm shift towards heterogeneous computing is a very recent trend and poses multiple challenges to the research community. One of the open challenges that we would like to address includes modeling the energy-time performance of a parallel program executing on heterogeneous computing systems, wherein the degree of overlap among the computation cycles and communication overhead needs to be computed. This would further complement and aid the design of novel workload allocation techniques and scheduling algorithms that improve the efficiency of such systems by increasing this degree of overlap.

We have shown that recent related work in the area of heterogeneity and energy-time performance is focused mainly on instrumentation or application-profiling based techniques or simulation-based approaches. However, this thesis proposes a measurement-based analytical modeling approach to model the energy-time performance. Specifically within inter-node heterogeneous systems, we propose a core model that determines both the execution time and energy performance of a parallel program running on multi-core heterogeneous clusters. The key novelties of our approach are modeling both inter and intra-node resource overlaps and resource contention. Furthermore, we show the scalability of the proposed core model by modeling both inter- and intra-node communication for energy-efficient execution of hybrid parallel programs.



# Chapter 3

## Approach and Core Model

This chapter describes our proposed approach for determining energy-efficient system configurations for multi-core clusters with multiple degrees of heterogeneity. First, we present an overview of our approach. Next, we show the derivation of our core model, the execution time model and energy model. Lastly, we validate the assumptions of our core model using measurements and then summarize.

### 3.1 Overview

This section presents an approach to determine time and energy efficient system configurations for executing a parallel program using a measurement-driven analytical model. Due to the wide spectrum of heterogeneity as shown in Figure 2.1, one possible approach is to develop a model for each type. However, we took a more scalable approach by identifying a core model that is applicable for a baseline system consisting of brawny and wimpy node clusters. While this core model does not include communication it is simple to apply and get quick analytical insights. To address the diversity in heterogeneous systems, we extend the core model for intra-node heterogeneous systems and hybrid programs. The proposed

core analytical model is formulated using parametric values obtained from baseline executions of the application to measure workload and architectural artefacts. The key novelties of our approach are modeling both inter and intra-node resource overlaps and resource contention.

Given a parallel program executing on multi-core heterogeneous clusters, the proposed approach determines energy-efficient Pareto-optimal configurations in terms of the number of nodes, number of cores per node and core clock frequency. These configurations either consume minimum energy for a given execution time deadline or execute in the minimum possible time for a given energy budget. Such a configuration is defined using a set of three-value tuples consisting of the types of nodes, number of nodes for each type, the active cores per node and the operating core clock frequency. Let the degree of heterogeneity,  $d_{max}$ , represent the maximum number of different types of multi-core nodes in a cluster. A homogeneous configuration is defined to have degree one. The total number of compute nodes available in the system configuration space is:

$$n_{max} = \sum_{i=1}^{d_{max}} n_{i,max}$$

While  $d_{max}$  is the maximum number of types of nodes available for execution, not all types of nodes are used in an energy-efficient configuration set, and we use  $d$  ( $\leq d_{max}$ ) to denote the degree of heterogeneity in the energy-efficient configurations. Among these varied types of nodes, an execution configuration is any combination of nodes with each node type having an efficient configuration of number of active cores,  $c_i$ , operating at a clock frequency,  $f_i$ . Hence, an efficient system configuration may be expressed as a set of  $(n_i, c_i, f_i)$  tuples:

$$\text{Energy-efficient configurations} \equiv \{(n_i, c_i, f_i) \mid i \in [1 \dots d]\}$$

As outlined in Fig. 3.1, the proposed approach determines the set of energy-efficient

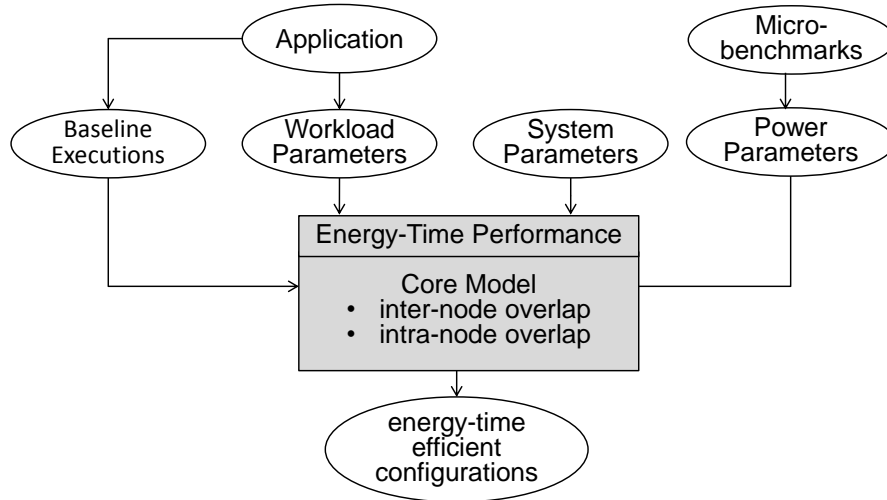


Figure 3.1: Methodology for core model

configurations for a given system. Using workload parameters obtained from baseline executions, we predict the execution time and energy used by the different configurations. We use the matching technique to determine the proportion of the workload distribution among the different types of nodes. Next, we remove sub-optimal configurations by deriving the Pareto frontier of the time-energy configuration space. Hence, given a service time deadline, our mix-and-match approach determines a set of energy-efficient configurations  $(\{(n_i, c_i, f_i) \mid i \in [1 \cdots d]\})$ . These configurations either meet a given deadline with minimum energy usage or execute in the minimum time for a given energy budget and hence form a time-energy Pareto frontier. The derivation of Pareto frontier is discussed in Chapter 4.

Let  $\mathcal{P}_s$  be a representative subset of a scale-out workload of program  $\mathcal{P}$  having repetitive phases of execution. For example, in memcached [5], the execution phases are the GET, SET, DELETE request types. The execution time for a heterogeneous mix of nodes is modeled based on measurements of architecture dependent parameters such as the number of instructions per GET command. For a given  $\mathcal{P}_s$ , we measure the number of instructions, work cycles and stall cycles to capture architecture specific parameters for each type of node. We determine the

execution time for  $\mathcal{P}$  based on  $\mathcal{P}_s$  by modeling the service time of CPU, memory and I/O. The execution time on each type of node is matched by distributing the workload in a ratio ( $r_i$ ) proportional to the computational rates of the node type. The energy consumed per node is determined as the product of the derived service time and the measured power consumption of each system component, such as cores, memory and I/O. Only network I/O operations are considered.

## 3.2 Core Model

We present the modeling of energy and execution time in the following sections using the notations<sup>1</sup> in Table 3.1.

### 3.2.1 Model Inputs

The measurement-driven inputs to our model are obtained from workload characterization using baseline executions and power characterization using micro-benchmarks. We first discuss the measurement of workload dependent input parameters such as work and stall cycles followed by the measurement of power parameters.

#### Workload Characterization

Typical scale-out workloads used in datacenters exhibit a lot of parallelism due to both user requests and data. The computations of such workloads can be divided into repetitive parallel execution phases within a request and also across a batch of requests [29]. The representative subset  $\mathcal{P}_s$  of the scale-out workload used in our model is this repeating parallel phase. For example, in memcached

---

<sup>1</sup>While the symbols for parameters described in the table are general, a subscript  $i$  denotes the parameter for a particular type of node.

Symbol	Description
<b>Workload Parameters</b>	
$\mathcal{P}$	program
$\mathcal{P}_s$	program $\mathcal{P}$ with smaller input size
$S$	number of iterations in $\mathcal{P}$
$S_s$	number of iterations in $\mathcal{P}_s$
$W$	total work units of $\mathcal{P}$
$r_i$	proportion of workload executed on nodes of type $i$ , where $i \in [1 \cdots d_{max}]$
$\lambda_{I/O}$	I/O requests inter-arrival rate
<b>System Parameters</b>	
$d_{max}$	maximum degree of inter-node heterogeneity of the system, $i \in [1 \cdots d_{max}]$
$n_{max}$	maximum number of nodes of type $i$
$c_{max}$	maximum number of cores for nodes of type $i$
$f_{max}$	maximum core clock frequency for nodes of type $i$
<b>Baseline Execution</b>	
$I_s$	number of instructions in $\mathcal{P}_s$
$w_s$	number of work cycles in $\mathcal{P}_s$
$b_s$	number of non-memory stall cycles in $\mathcal{P}_s$
$m_s$	number of memory-related stall cycles in $\mathcal{P}_s$
$U_s$	Average CPU utilization for $\mathcal{P}_s$
<b>Time Model</b>	
$I$	number of instructions in $\mathcal{P}$
$w$	number of work cycles for $\mathcal{P}$
$b$	number of non-memory stall cycles for $\mathcal{P}$
$m$	number of memory-related stall cycles for $\mathcal{P}$
$U$	CPU utilization for $\mathcal{P}$
$n$	number of nodes
$c$	number of active cores per node
$f$	operating core clock frequency
$T_{CPU}$	total CPU response time for $\mathcal{P}$
$T_{core}$	total core response time for $\mathcal{P}$
$T_{mem}$	total memory response time for $\mathcal{P}$
$T_{I/O}$	total I/O response time for $\mathcal{P}$
$T_{I/O_T}$	total I/O transfer time for $\mathcal{P}$
$T$	total execution time of program $\mathcal{P}$
<b>Power Parameters[W]</b>	
$P_{CPU,act}$	CPU power when executing work cycles
$P_{CPU,stall}$	CPU power when memory-related stalls
$P_{mem}$	power consumed by memory operations
$P_{net}$	power consumed by network card
$P_{sys,idle}$	power consumed by idle system
<b>Energy Model[J]</b>	
$E_{CPU,act}$	total energy consumed when CPU is active
$E_{CPU,stall}$	total energy consumed when CPU is stalling
$E_{mem}$	total energy consumed by memory sub-system
$E_{net}$	total energy consumed by network sub-system
$E_{idle}$	total energy consumed by idle system
$E$	total energy consumed by a program

Table 3.1: Core model parameters

program [5], each of the GET, SET and DELETE request types are a parallel phase of execution. We measure the number of instructions, work cycles and stall cycles for a single GET, SET and DELETE command to capture the architecture specific workload parameters for each type of node. The measurements used in this thesis are done using hardware event counters in the respective nodes. The measurements are done only once for each type of node being used.

### Power Characterization

During execution, a processor consumes varying amount of power depending on the number of active components. CPU active power,  $P_{CPU,act}$ , is measured across cores and frequencies for each type of node, using a micro-benchmark that maximizes the CPU utilization. Power incurred by CPU stall cycles,  $P_{CPU,stall}$ , is measured using a micro-benchmark that generates a stream of cache misses to maximize the number of stall cycles. Power used by active memory,  $P_{mem}$  is derived from specifications [2, 107]. Networking I/O power,  $P_{I/O}$ , is obtained through direct measurement when the NIC is used and the idle system power,  $P_{idle}$ , is measured without any workload. It suffices to do the measurements on a single node of each type because all the nodes of the same type exhibit similar power characteristics.

### 3.2.2 Execution Time Model

The workload is split between the  $d$  types of nodes:

$$W = \sum_{i=1}^d W_i \quad \text{and} \quad W_i = W \cdot r_i \quad (3.1)$$

where  $W_i$  denotes the proportion of the workload executed on  $n_i$  nodes of type  $i$  with each node using  $c_i$  active cores operating at clock frequency  $f_i$ . Workload



( $W_i$ ) is equally distributed among nodes of the same type. Hence, nodes of the same type complete their execution approximately at the same time:

$$T = \max_{i=1}^{d_{max}}(T_i) \quad (3.2)$$

The matching technique is used to distribute workload among nodes of different types in a ratio such that the execution rate among them is matched. Thus, the execution time of the program is:

$$T = T_i \quad (3.3)$$

The execution time on a node depends on workload service demands on node resources such as CPU and I/O response times. Since the I/O devices operate with minimal CPU intervention, the bottleneck device dominates the response time:

$$T_i = \max(T_{i,CPU}, T_{i,I/O}) \quad (3.4)$$

### **CPU Response Time**

The CPU response time includes both the execution time of the core while performing computations and the stall time of the core while waiting for memory requests completion. Let  $T_{core}$  denote the execution time of the core performing computations and non-memory stalls, and  $T_{mem}$  denote the response time of the memory requests. Most processors support out-of-order execution that may overlap waiting for memory requests with execution of work cycles. Hence, the CPU response time is determined by the bottleneck between core and memory:

$$T_{i,CPU} = \max(T_{i,core}, T_{i,mem}) \quad (3.5)$$

The number of instructions executed for the same amount of work is different on each type of node, because of the different ISAs and micro-architectures. Thus, we use the measured parameter  $I_{i,s}$ , the number of instructions incurred by  $\mathcal{P}_s$ , on each type of node to determine the total instructions per node type as:

$$I_i = W_i \cdot I_{i,s} \quad (3.6)$$

The instructions executed on each type of node,  $I_i$  are split among  $n_i$  nodes. Furthermore, within one node the instructions that can be executed in parallel are equally split among cores. The utilization of the CPU accounts for the data dependencies among the cores. Thus, the instructions executed per core are:

$$I_{i,core} = \frac{I_i}{n_i \cdot U_{i,CPU} \cdot c_i} \quad (3.7)$$

The number of useful work cycles and the non-memory stall cycles per instruction depend on the micro-architecture implementation of the particular type of node:

$$WPI_i = \frac{w_s}{I_{i,s}} \quad \text{and} \quad SPI_{i,core} = \frac{b_s}{I_{i,s}} \quad (3.8)$$

Next, we derive the execution time per core by modeling the number of cycles. For a compute-intensive workload, the number of cycles incurred by a core is equal to the work cycles and the non-memory related stall cycles:

$$cycles_{i,core} = I_{i,core} \cdot (WPI_i + SPI_{i,core}) \quad (3.9)$$

and thus

$$T_{i,core} = \frac{cycles_{i,core}}{f_i} \quad (3.10)$$

As the program scales from  $\mathcal{P}_s$  to  $\mathcal{P}$ , the number of instructions scale, but the ratio of work cycles to instructions remains constant. Similarly the ratio of stall cycles to instructions,  $SPI_{core}$ , also remains constant. This hypothesis of constant  $WPI$  and  $SPI_{core}$  is validated in Section 3.3.2. Hence, we measure  $w_s$  and  $b_s$  for  $\mathcal{P}_s$  and use them to determine the number of cycles for  $\mathcal{P}$ .

### Memory Response Time

The CPU response time,  $T_{CPU}$ , depends on the service time of the memory controller or the memory response time,  $T_{mem}$  (Eq. 3.5). Memory response time can be determined by the number of stall cycles incurred by the core due to memory requests,  $m$ . For a memory bound workload, the total number of cycles incurred during execution,  $cycles_{i,mem}$ , are the work cycles,  $w$ , and the stall cycles due to memory accesses that cannot be overlapped with useful work,  $m$ . Thus,

$$cycles_{i,mem} = w + m \quad (3.11)$$

By measuring the stall cycles incurred due to memory requests for executing  $\mathcal{P}_s$ , on each type of node, the memory response time for executing program  $\mathcal{P}$  can be directly inferred. Hence, we use the measured values of  $m$  for  $\mathcal{P}_s$  ( $m_s$ ) to compute  $SPI_{i,mem}$  as:

$$SPI_{i,mem} = \frac{m_s}{I_{i,s}} \quad (3.12)$$

Thus, the total cycles incurred by a program  $\mathcal{P}$  that is bound by memory service time can be determined from  $WPI$  and  $SPI_{i,mem}$  as:

$$cycles_{i,mem} = I_i \cdot (WPI_i + SPI_{i,mem}) \quad (3.13)$$

$$T_{i,mem} = \frac{cycles_{i,mem}}{f_i} \quad (3.14)$$

When the number of cores requesting memory accesses increases, memory response time also increases due to memory contention [157]. The increase in memory response time results in higher CPU stall cycles. In Section 3.3.3, we show that  $SPI_{mem}$  correlates to both changes in core clock frequency  $f$  and the active number of cores  $c$ . From Equations 5.4 and 3.12, memory response time can be determined using the measured values of  $w_s$  and  $m_s$  across all possible values of the number of active cores and core clock frequencies.

### I/O Response Time

Since CPU computation time overlaps with I/O request transfer time, and I/O transfer time in turn overlaps with inter-arrival waiting time of the next request, it suffices to consider the maximum of these two values. Hence, for an I/O bound program the response time is the maximum between the I/O transfer time as a function of I/O bandwidth and the I/O requests inter-arrival rate.

$$T_{i,I/O} = \frac{\max(T_{i,I/O_T}, \frac{1}{\lambda_{I/O}})}{n_i} \quad (3.15)$$

For an I/O bound program, with the increase in the number of nodes for a fixed workload, the I/O response time improves because the I/O bandwidth demands per node decreases.

### 3.2.3 Matching Technique

In this section, we describe the approach to determine the proportion of workload executed by each type of node,  $r_i$ , to match the execution rates among all the nodes. As seen from the time model equations, the execution time depends on the bottleneck resource, namely processing cores, memory or network and is

directly proportional to the amount of workload being executed. By abstracting the proportionality constant to be  $k_i$  for each type of node, the execution time on  $n_i$  nodes of the same type  $i$  can be expressed as:

$$T_i = k_i \cdot r_i \tag{3.16}$$

The proportionality constant  $k_i$  depends on the bottleneck resource for that type of node. While the bottleneck resource in a node shifts due to the problem size of the workload being executed (i.e.  $r_i$ ), such an abstraction using  $k_i$  leads to a closed-form solution for matching the execution rate among nodes. Matching the execution rate among different types of nodes makes the execution time due to each type of node approximately equal. Thus:

$$T_i = T_j \tag{3.17}$$

$$T_1 = T_2 \Rightarrow k_1 \cdot r_1 = k_2 \cdot r_2 \Rightarrow r_2 = \frac{k_1 \cdot r_1}{k_2} \tag{3.18}$$

Similarly, matching the execution time among  $d$  types of nodes, the proportion of workload executed on each type becomes a function of  $k_1$ ,  $r_1$  and  $k_i$  as:

$$r_i = \frac{k_1 \cdot r_1}{k_i} \tag{3.19}$$

Since the total work  $W$  of the program  $\mathcal{P}$  is split into the proportional amount,  $r_i$ , on each type of node:

$$r_1 + r_2 + \dots + r_d = 1 \tag{3.20}$$

Thus, from Equations 3.19 and 3.20, the percentage of workload  $r_1$  executed on  $n_1$  nodes of same type is determined as follows:

$$\begin{aligned}
 r_1 &= 1 - (r_2 + r_3 + \dots + r_d) \\
 r_1 &= 1 - \left( \frac{k_1 \cdot r_1}{k_2} + \frac{k_1 \cdot r_1}{k_3} + \dots + \frac{k_1 \cdot r_1}{k_d} \right) \\
 r_1 &= \frac{1}{1 + \frac{k_1}{k_2} + \frac{k_1}{k_3} + \dots + \frac{k_1}{k_d}} \tag{3.21}
 \end{aligned}$$

This derivation indicates that the proportion of workload on a type of node is inversely proportional to its execution time proportionality constant  $k$ , and hence a poor performing node (larger execution time) should execute smaller proportions of the workload.

### 3.2.4 Energy Model

For a given workload, the energy model determines the total energy consumed during program execution, by using system power parameters and the execution time of  $\mathcal{P}$ . Total energy for a given workload is the sum of energies consumed by core, memory, I/O devices and the idle energy within a node across all nodes in the system.

$$E = \sum_{i=1}^d E_i \tag{3.22}$$

$$E_i = (E_{i,CPU} + E_{i,mem} + E_{i,I/O} + E_{i,idle}) \times n_i \tag{3.23}$$

The measured power parameter  $P_{CPU,act}$  includes the number of active cores within a CPU. The power used by inactive cores is accounted by  $P_{idle}$  which in turn

determines  $E_{idle}$ . Hence the energy consumed by the CPU is:

$$E_{i,CPU} = (P_{i,CPU,act} \cdot T_{i,act}) + (P_{i,CPU,stall} \cdot T_{i,stall}) \quad (3.24)$$

where

$$T_{i,act} = \frac{I_{i,core} \cdot WPI_i}{f_i} \quad (3.25)$$

$$T_{i,stall} = \frac{I_{i,core} \cdot SPI_{i,core}}{f_i} \quad (3.26)$$

$I_{i,core}$ ,  $WPI_i$  and  $SPI_{i,core}$  are obtained as explained in Section 3.2.2. The energy consumed by memory is:

$$E_{i,mem} = P_{i,mem} \cdot T_{i,mem} \quad (3.27)$$

and the energy consumed by I/O is:

$$E_{i,I/O} = T_{i,I/O} \cdot P_{i,I/O} \quad (3.28)$$

The power parameter of each individual component excludes the system idle power. Thus, the idle energy is:

$$E_{i,idle} = T_i \cdot P_{i,idle} \quad (3.29)$$

### 3.3 Validation

This section presents the validation of our proposed model against measurements of execution time and energy usage for a diverse set of workloads. First, we present

the workloads and the system setup. Next, we show experimental evidence for our hypothesis of constant  $WPI$  and  $SPI_{core}$  across program input sizes, and the linearity of  $SPI_{mem}$  across cores and core clock frequencies. Lastly, we summarized the validation results.

### 3.3.1 Workloads and Setup

Many datacenter workloads must obey strict service time deadlines. To service requests within a deadline, processing is distributed over hundreds of server nodes. Jobs arrive at front-end nodes and are forwarded to a cluster of compute nodes that service job requests. Both response time and energy incurred by a job are dominated by compute nodes [116]. Thus, we focus on the energy efficiency of compute nodes only.

As we are targeting datacenter workloads, we select six programs representing different performance bottlenecks and with different deadline requirements. *EP*, from NPB benchmark [26], is an embarrassingly parallel distributed memory program that generates random numbers for Monte-Carlo numerical simulation. *Memcached* is widely used by Facebook, Amazon, Twitter, among others, as an in-memory key-value distributed storage. When a key request arrives, a front-end node dispatches the request to a set of nodes that are responsible for storing the key-values belonging to an application. All nodes in the pool perform a key look-up computation, but typically few nodes return the value. However, this operation may exert complex service demands on core, memory and I/O devices [101, 158]. We use memslap running on another system to generate requests to the memcached server over a 1Gbps network connection. Memslap generates requests with fixed key-value size and uniform popularity.

From the PARSEC benchmark suite [31], *x264* represents the widely used encoding algorithm for streaming video, and *blackscholes* represents a quantitative



model for determining option pricing. The open source speech recognition engine *Julius* [7] represents the increasing adoption of real-time speech processing workloads originating from smart devices. To analyze the energy efficiency of web security, we use the openssl *RSA-2048* speed benchmark because major web players are increasingly concerned with in-transit data security and are hardening the https encryptions [6].

While our proposed approach can analyze a generic mix of heterogeneous nodes, for validation, we consider a mix of **A9** (ARM), **K10** (AMD) and **E5** (Intel) nodes, as shown in Table 3.2. These representative nodes cover the broad spectrum of performance and power offered by computing platforms today. At one end of the spectrum is the low-power A9 node which consumes a peak power of only 5W. At the other end of the power spectrum, we select the recent and more energy-efficient high-performance E5 node with eight cores which consumes a peak power of about 80W. As an intermediary point, we select the older generation K10 node that consumes a peak power of about 60W and offers a peak performance around 50 GFLOPS that lies between the A9 and E5 nodes. We use *perf* to access hardware event counters and to measure execution time, and a Yokogawa WT210 power monitor to measure the power and energy, as shown in Fig. 3.2.

Node	ARM Cortex A9	AMD Opteron K10	Intel Xeon E5
ISA	ARMv7-A	x86_64	x86_64
Clock Freq	0.2–1.4 GHz	0.8–2.1 GHz	1.2–1.8 GHz
Cores/node	4	6	8
L1 data cache	32KB / core	64KB / core	32KB / core
L2 cache	1MB / node	512KB / core	2MB / node
L3 cache	NA	6MB / node	20MB / node
Memory	1GB LP-DDR2	8GB DDR3	8GB DDR3
I/O bandwidth	100Mbps	1Gbps	1Gbps
Peak power [W]	5	60	80

Table 3.2: Types of heterogeneous nodes

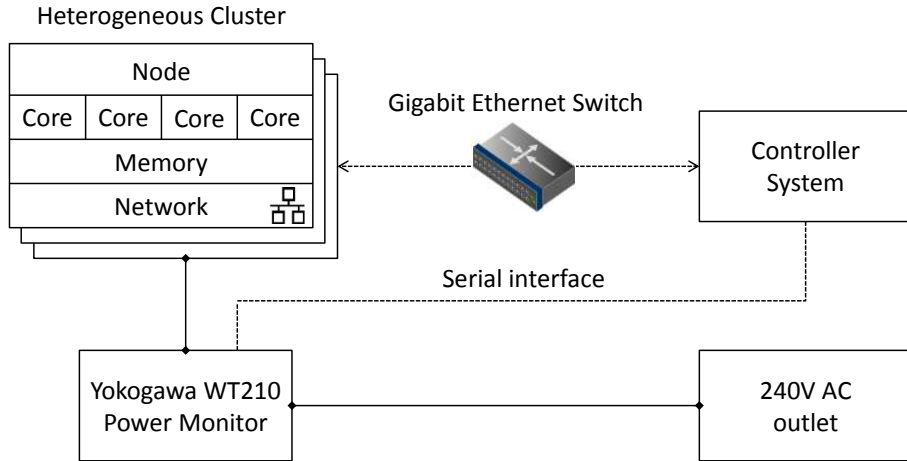


Figure 3.2: Validation setup

### 3.3.2 $WPI$ and $SPI_{core}$

This section validates our hypothesis of constant  $WPI$  and  $SPI_{core}$  as workload scales from  $\mathcal{P}_s$  to  $\mathcal{P}$ . Fig. 3.3 plots the  $WPI$  and  $SPI_{core}$  for the EP benchmark with increasing problem sizes from A to C on the three types of nodes. The problem size increases by four times from class A to class B and by sixteen times from class A to class C. The plot shows that our hypothesis holds for EP, as it does for all the other workloads.

Domain	Program	Problem Size	Execution time error [%]						Energy error [%]					
			A9 node		K10 node		E5 node		A9 node		K10 node		E5 node	
			Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
HPC	EP	2,147,483,648 random numbers	1	2	1	5	4	2	6	5	6	2	9	8
Web Server	memcached	600,000 GET/SET operations	4	8	7	5	3	2	5	4	6	5	10	8
Streaming video	x264	600 frames 704 × 576	1	3	6	4	11	4	5	8	9	2	11	4
Financial	blackscholes	500,000 stock options	1	1	1	1	15	13	5	2	7	3	14	12
Speech recognition	Julius	2,310,559 samples	4	9	10	2	13	4	2	5	7	6	9	6
Web security	RSA-2048	5000 keys verifications	10	1	1	1	1	1	2	7	7	2	6	5

Table 3.3: Single-node validation

### 3.3.3 $SPI_{mem}$ Regression over Core Frequency $f$

As our approach models the memory response time by measuring the stall cycles due to memory requests for the program  $\mathcal{P}_s$ , we present the validation of the

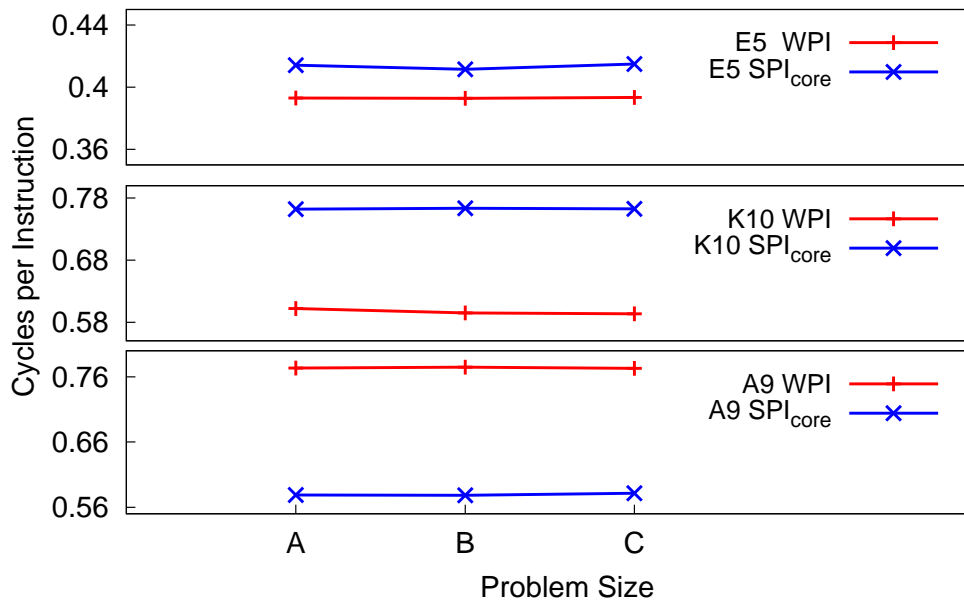


Figure 3.3:  $WPI$  and  $SPI_{core}$  across problem size

correlation between  $SPI_{mem}$  across different core clock frequencies and number of cores. Fig. 3.4 shows that as core frequency increases,  $SPI_{mem}$  grows linearly.

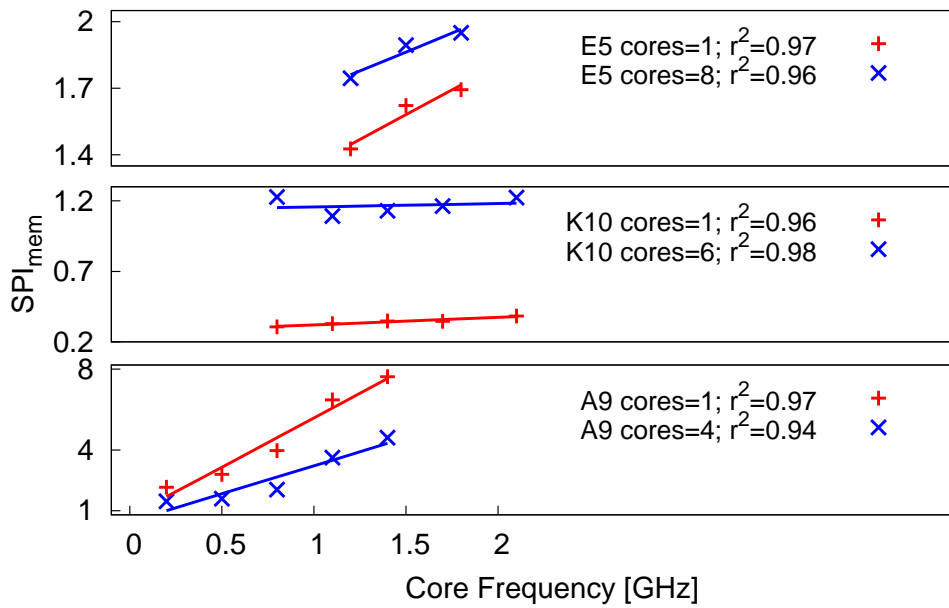


Figure 3.4: Effect of frequency and number of cores on  $SPI_{mem}$

The Pearson’s correlation coefficient between  $SPI_{mem}$  and  $f$  is  $r^2 \geq 0.94$ , showing a very strong correlation among them.

Program	Execution time error [%]	Energy error [%]
EP	3	10
memcached	10	8
x264	11	10
blackscholes	4	7
Julius	13	1
RSA-2048	2	8

Table 3.4: Cluster validation

### 3.3.4 Execution Time and Energy

The modeled execution time and energy are validated against measured values, for all workloads described above. We validated two aspects of our mix-and-match approach. First, we validated the predictions of execution time and energy for a single node across all three types of nodes and across all combinations of number of cores per node type and core clock frequencies. This validation tests the accuracy of selecting the most energy efficient configuration per node. Second, we validated multi-node energy efficient configurations. Together, these experiments validate our selection of the Pareto-optimal configurations. While Table 3.3 shows the average error and standard deviation on a single node, Table 3.4 summarizes the average error on heterogeneous clusters. The absolute values of the execution time and energy validated for a single node are shown in the appendix at Section A.4. In summary, the average error of the model is less than 15%.

### 3.3.5 Sources of Inaccuracy

We identify three factors that affect the accuracy of the model. Firstly, the most significant source of error is due to execution time irregularities during different executions of the same program because of operating system overheads. The

measured values of execution time and energy show irregularities of up to 10% for different runs of the same program. The second reason for model inaccuracy is that the workload is not perfectly divisible on multiple cores and nodes. Third, the model depends on the accuracy of the characterized power parameters. In particular, system power values for active cycles, stall cycles and idleness differ by up to 0.4W for the A9 node, 5W for the K10 node and 2W for the E5 node. This variability translates into a larger underestimation of the energy consumed especially for larger execution times.

### 3.3.6 Accuracy of Related Work

We present the average validation error of the state-of-the-art approaches relevant to our approach in Table 3.5. While the proposed approach in this thesis results in an average error up to 15%, the different related works have errors in the range from 16% to 25%.

Related Work	Average Error [%]
Blocking factor model [126]	20
Predictive metrics [37]	22
Sniper: interval simulation model [36]	25
GEM5 simulator on ARM Cortex-A15 [72]	16
Queuing architectural model [139]	22

Table 3.5: Accuracy of Related Work

## 3.4 Summary

This chapter proposes a measurement-based analytical modeling approach to determine the execution-time and energy performance of a parallel program executing on multi-core heterogeneous clusters. As heterogeneous clusters have different execution rate, we propose a *matching* technique, which splits the workload such that all the different types of nodes complete the parallel job at the same time.

By finishing at the same time, the energy incurred by idling in the cluster is minimized.

<b>Energy Performance</b>	
$E$	$\sum_{i=1}^d E_i$
$E_i$	$(E_{i,CPU} + E_{i,mem} + E_{i,I/O} + E_{i,idle}) \cdot n_i$
$E_{i,CPU}$	$(P_{i,CPU,act} \cdot T_{i,act}) + (P_{i,CPU,stall} \cdot T_{i,stall})$
$E_{i,mem}$	$P_{i,mem} \cdot T_{i,mem}$
$E_{i,I/O}$	$T_{i,I/O} \cdot P_{i,I/O}$
$E_{i,idle}$	$T_i \cdot P_{i,idle}$
<b>Time Performance</b>	
$T$	$\max_{i=1}^{d_{max}} (T_i)$
$T_i$	$max(T_{i,CPU}, T_{i,I/O})$
$T_{i,CPU}$	$max(T_{i,core}, T_{i,mem})$
$T_{i,core}$	$\frac{cycles_{i,core}}{f_i}$
$T_{i,mem}$	$\frac{cycles_{i,mem}}{f_i}$
$T_{i,I/O}$	$\frac{\max(T_{i,I/O_T}, \frac{1}{\lambda_{I/O}})}{n_i}$

Table 3.6: Summary of core model

To determine the proportion of workload that is assigned to each type of node, we propose and use the core analytical model that determines the energy required to execute a parallel program. The core model is applicable on heterogeneous clusters with wimpy and brawny nodes having different Instruction Set Architectures (ISAs). For each type of node, the core model predicts the execution time and energy usage of a job considering the overlap among the response times of service requests to the CPU, the memory and the network I/O devices. The core model is validated against direct measurements of execution time and energy usage on a heterogeneous cluster with ARM Cortex-A9, AMD Opteron K10 and Intel Xeon E5 multi-core nodes, for a diverse range of parallel applications. Table 3.6 summarizes the energy and time performance derived from our core model.

# Chapter 4

## Analysis of Inter-node Heterogeneous Systems

In this chapter, we apply our model to study the energy efficiency of different inter-node heterogeneous configurations under a given service time deadline. We first evaluate the impact of multiple degrees of heterogeneity on energy efficiency. Next, we present the performance-to-power ratio (PPR) of the inter-node heterogeneous system and analytically determine the impact of PPR on the sweet region. Lastly, we show how to apply our approach to choose energy-efficient cluster mixes constrained by a power budget followed by an analysis of whether heterogeneous clusters are more energy proportional.

### 4.1 Pareto-optimal Configurations

This section evaluates if heterogeneity reduces energy consumption while still meeting an execution time deadline. As the total energy depends on the number of active nodes, the number of cores per node and core clock frequency, finding the global optimum configuration is a complex task. For example, a system with

three A9, three K10 and three E5 nodes results in a total of a quarter-million possible configurations<sup>1</sup>. With the explosion of the configuration space, we show that the sweet region can be analytically established using the node performance-to-power ratios (PPR) in Section 4.2.2. In this section, we use EP workload to illustrate the observations, but similar observations hold for all workloads. For our energy efficiency analysis we choose an input of five million random numbers for EP as one job. However, we note that the input size does not impact the conclusion of the analysis because increasing the input size leads to an increase in both execution time and energy usage. The homogeneous configurations with up to eight A9 nodes have been validated against measurements and the data for other configurations are from the model.

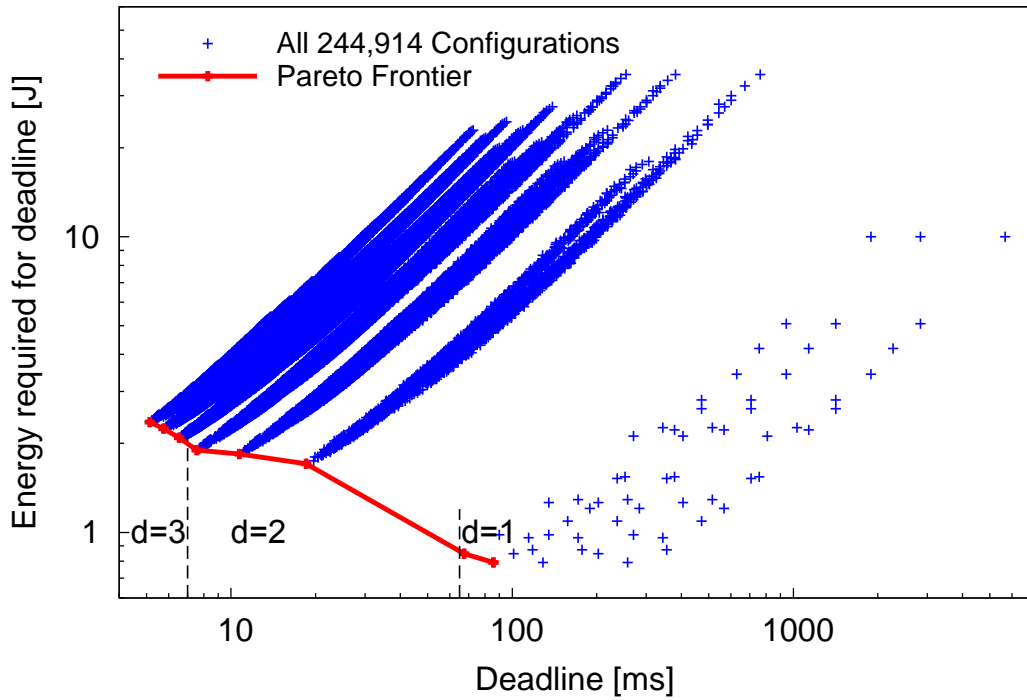
Fig. 4.1 plots the energy incurred to finish the EP job for all possible configurations. Each point in this plot represents a different configuration, determined by the number of A9, K10 and E5 nodes, number of cores per node and the core clock frequency. For each configuration point, the x-axis denotes the job service time and the y-axis represents the corresponding energy used. A configuration that meets the deadline with the minimum energy usage is *Pareto optimal*. The set of all Pareto optimal points across all possible deadlines forms the energy-deadline *Pareto frontier*, as shown in Fig. 4.1.

From the plot, it is observed that increasing the degree of heterogeneity results in more configurations on the Pareto frontier. The Pareto frontier in Fig. 4.1 can be divided into three parts. The leftmost part consists of mixes of all the three types of nodes, A9, K10 and E5, and hence has degree  $d = 3$ . The middle region has nodes of only two types, A9 and E5 having a degree  $d = 2$ . The rightmost

---

<sup>1</sup>  $d_{max} = 3$ ;  $n_1 = 3$ ,  $n_2 = 3$ ,  $n_3 = 3$ ;  $c_1 = 6$ ,  $c_2 = 4$ ,  $c_3 = 8$ ;  $f_1 \in [0.8, 1.4, 2.1]$  GHz,  $f_2 \in [0.2, 0.5, 0.8, 1.1, 1.4]$  GHz,  $f_3 \in [1.2, 1.5, 1.8]$  GHz. Total number of configurations =  $(3 \times 6 \times 3) \times (3 \times 4 \times 5) \times (3 \times 8 \times 3) + (3 \times 6 \times 3) \times (3 \times 4 \times 5) + (3 \times 4 \times 5) \times (3 \times 8 \times 3) + (3 \times 8 \times 3) \times (3 \times 6 \times 3) + (3 \times 6 \times 3) + (3 \times 4 \times 5) + (3 \times 8 \times 3) = 244,914$  configurations.



Figure 4.1: Pareto frontier for EP with  $d_{max} = 3$ 

part consisting of only A9 nodes has two homogeneous ( $d = 1$ ) configurations on the Pareto frontier. These configurations on the Pareto frontier represent a “sweet region”<sup>2</sup> where relaxing the deadline linearly reduces the energy used.

### Impact on Energy Savings

The existence of the Pareto-frontier and using Pareto-optimal configurations for executing a program imply two options for saving energy. Firstly, for a given execution time deadline, using a Pareto-optimal configuration instead of a non-optimal configuration results in energy savings. For example, for executing the EP program on the inter-node heterogeneous cluster, a Pareto-optimal configuration reduces energy by up to 75% as compared to a non-optimal configuration. Secondly, Pareto-optimal configurations reduces energy by 17% at the expense of

<sup>2</sup>A sweet region is a union of Pareto optimal heterogeneous sweet-spots.

increasing the execution time of the EP program on an inter-node heterogeneous cluster as shown in Figure 4.1.

In summary, we show that heterogeneity allows for the existence of a sweet region. With the explosion of the configuration space, we show how to analytically determine the configurations on the sweet region using the PPR of each type of node, in the following section.

## 4.2 Impact of Performance-to-Power Ratio

### 4.2.1 Performance-to-Power Ratio (PPR)

PPR is defined as the work done per unit of time, normalized by the average power consumption. This is equivalent to the work done per unit of energy. The PPRs computed for the most energy-efficient configuration per type of node are shown in Table 4.1. As observed from the table, A9 has a better PPR than K10 and E5, but with two notable exceptions. For web-security applications such as RSA-2048, E5 has better PPR due to its special instructions that accelerate cryptography processing. x264 encoding algorithm is memory-bound [31], and performs much better on the E5 node which has a higher memory bandwidth. For the other applications, A9 has a better PPR but lower overall performance. Hence, mixing the three nodes with diverse PPR optimizes both energy efficiency and performance, as shown in Section 4.1. We discuss the conditions under which a mix of heterogeneous nodes exhibits a sweet region. To assess the effect of PPR, we generate the mix-and-match sweet region exhibited by a mix of three A9, K10 and E5 nodes for two cases:(i) EP – where the A9 PPR is the best, and (ii) RSA-2048 – where the E5 PPR is the best.

Figs. 4.2 and 4.3 plot the sweet regions for both types of programs. For each

Program	Performance per Watt (PPR)	A9 node	K10 node	E5 node
EP	(random no./s)/W	6,048,057	1,414,922	2,982,616
memcached	(bytes/s)/W	5,224,004	2,68,067	1,851,587
x264	(frames/s)/W	0.7	1	1.6
blackscholes	(options/s)/W	11,413	2,902	694
Julius	(samples/s)/W	69,654	21,390	19,167
RSA-2048	(verify/s)/W	968	1091	1,483

Table 4.1: Performance-to-power ratio

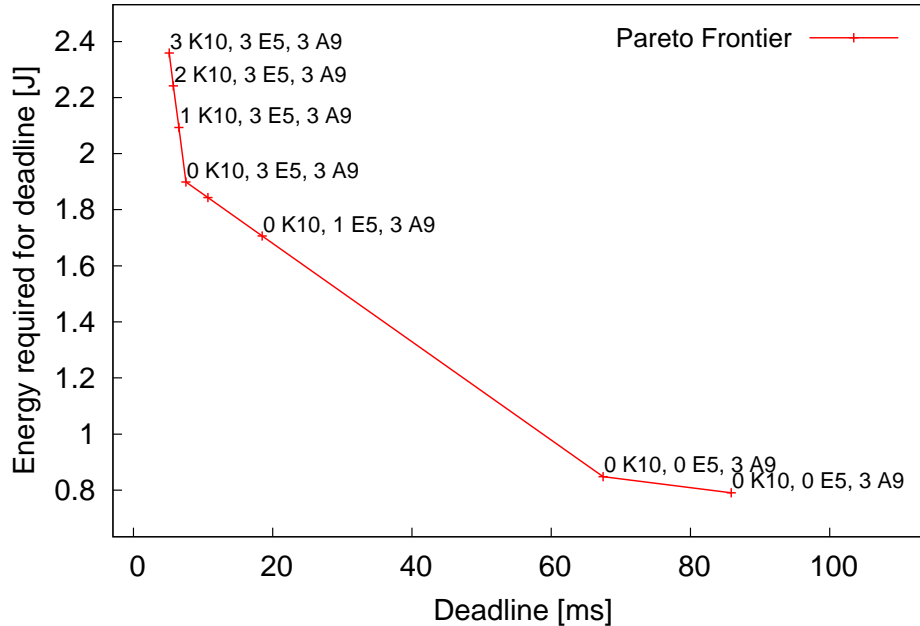


Figure 4.2: Pareto frontier configurations for EP



regions.

### 4.2.2 Analytical Analysis of PPR on Sweet Region

Given a server workload and an upper bound on the maximum number of nodes of each type, the upper and lower bounds of energy versus deadline for the configurations on the sweet region can be determined analytically. The Pareto-optimal configurations on the sweet region form a *total order* with respect to the relation of decreasing energy as deadline is relaxed. If  $\tau$  denotes a configuration on the sweet region, then the total order with respect to energy  $E(\tau)$ , and execution time deadline  $T(\tau)$  is represented as:

$$\forall i, j \in \mathbb{N} : i < j \Leftrightarrow E(\tau_i) > E(\tau_j) \wedge T(\tau_i) < T(\tau_j)$$

For a given workload ( $\mathcal{P}$ ), we sort each of the  $d$  types of nodes with respect to their PPRs in the ascending order, such that:

$$\forall i, j \in [1 \cdots d_{max}], i < j \Leftrightarrow PPR_i < PPR_j$$

Then the sweet region for executing  $\mathcal{P}$  on a heterogeneous mix of nodes with degree of heterogeneity  $d$  can be analytically determined based on the ascending order of the PPRs for each type of node. The sweet region configuration executing in the minimum possible execution time deadline,  $\tau_1$ , is the mix with the maximum number of nodes of each type:

$$\tau_1 = (n_{1,max}, n_{2,max}, \cdots, n_{d,max})$$

The next configuration  $\tau_2$  on the sweet region has an execution time slightly longer

and consumes lesser energy than  $\tau_1$  configuration.

$$\tau_2 = ((n_{1,max} - 1), n_{2,max}, \dots, n_{d,max})$$

The configurations on the sweet region thus have a decrease in the number of nodes for the node type having the lowest PPR such that the final configuration with the maximum execution time and minimum energy is a homogeneous one, as shown in Fig. 4.4. This homogeneous configuration consists of nodes with the

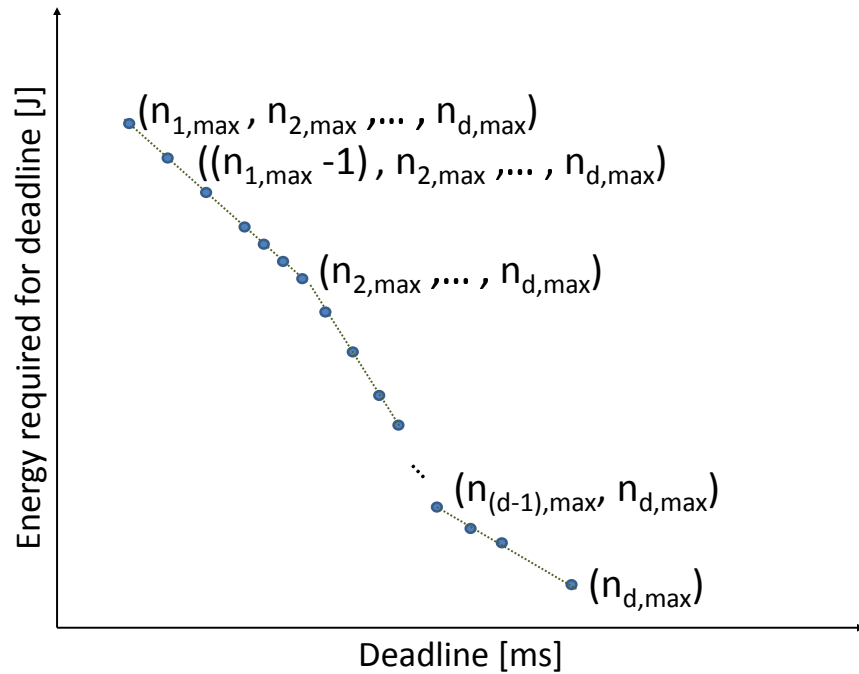


Figure 4.4: Configurations on the sweet-spot region

best PPR. Since the nodes have been ordered in ascending order of the PPRs, the configurations with  $n_{d,max}$  nodes has the best PPR among the  $d_{max}$  types of nodes in the system.

The characteristic of the sweet region is a decrease in energy with an increase in the execution time. This characteristic is dependent on the ratio of the PPRs among the  $d$  nodes and the bottleneck system resource for the particular workload being executed. For a given workload, changing the system configuration also

changes the proportion of workload being executed on a type of node. Thus, the bottleneck resource on a type of node varies among the configurations on the sweet region. Therefore, the sweet region exhibits multiple gradients, with each of these gradients being determined not only by the ratio of the PPR among the nodes but also by the type of resource bottlenecks (core, memory or I/O) within a type of node.

## 4.3 Impact of Power Substitution Ratio

### 4.3.1 Power Substitution Ratio (PSR)

Since datacenters often have an upper bound on their peak power consumption, we consider a fixed peak power budget drawn by our system that constrains the maximum number of nodes. Based on peak power proportion between A9, K10 and E5 nodes, we analyze the impact of replacing some K10 and E5 nodes with A9 nodes for workloads where A9 has a better PPR, such as the EP program. Similarly, we analyze the impact of replacing A9 nodes with E5 and K10 nodes, for workloads where E5 has a better PPR, such as the RSA-2048 program. These replacements are done such that the total peak power is within the allocated budget.

Assuming a peak power budget of one kilowatt, the number of A9, K10 and E5 nodes are chosen based on the power substitution ratios. Since each K10 node draws a peak power of 60W and each A9 node draws a peak power of 5W, one K10 node can be replaced by twelve A9 nodes. However, accounting for the twenty watts of peak power drawn by the switch [3] that connects the A9 nodes, gives us a power substitution ratio of 8:1 between A9 and K10 [137]. Similarly, a power substitution ratio between A9 and E5 nodes is 12:1 as the E5 node draws a peak

power of 80W. Therefore, to meet a budget of one kilowatt, there are multiple possibilities of mixing all three types or any two types of nodes, to constitute a heterogeneous cluster. Examples of some of these node mixes for a power budget of one kilowatt are shown in Table 4.3.1.

Next, we discuss the effect of these various heterogeneous mixes on the sweet region and show how the PPR discussed in Section 4.2.2 influences the energy efficiency among different mixes for a given power budget.

Degree of heterogeneity	A9 nodes	K10 nodes	E5 nodes
$d = 2$	96	4	0
	80	6	0
	0	4	8
	0	7	6
	32	0	8
	56	0	6
$d = 3$	72	4	2
	80	3	2
	32	3	6
	24	4	6
	24	1	8
	16	2	8

Table 4.2: Cluster mixes for 1kW power budget

### 4.3.2 Impact of Heterogeneous Mixes on the Sweet Region

Figs. 4.5 and 4.6 show the impact of changing the number of A9, K10 and E5 nodes, for a fixed peak power budget of one kilowatt, and degrees of heterogeneity being two and three. We use the power substitution ratios of replacing nodes among the mixes as shown in Section 4.3.1. As observed from Fig. 4.5, the heterogeneous mixes with only E5 and K10 nodes have the worst deadline-energy Pareto configurations compared with heterogeneous mixes with only A9 and K10 or only A9 and E5. The insight behind this observation is that A9 has the best PPR for the EP program compared to K10 and E5 nodes. Hence, node mixes with more A9 nodes outperform mixes with no A9 nodes or mixes with lower number of A9



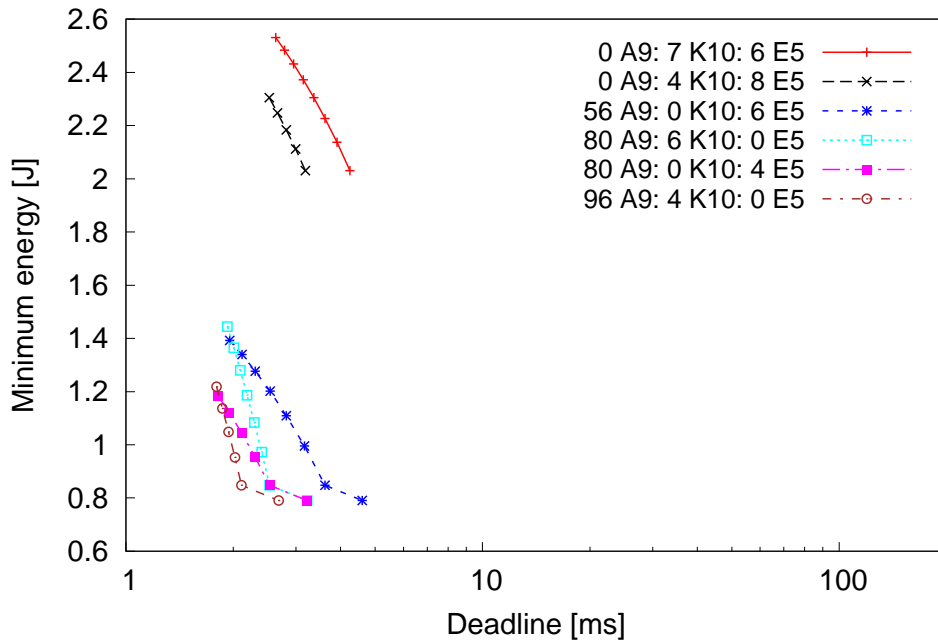


Figure 4.5: Pareto frontier for EP with  $d = 2$

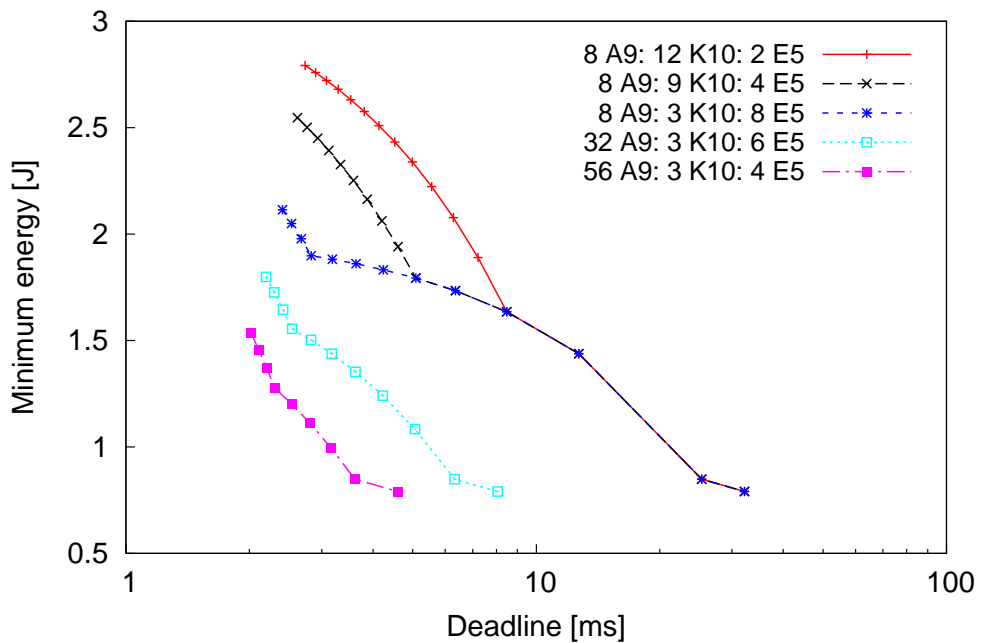


Figure 4.6: Pareto frontier for EP with  $d = 3$

nodes.

Fig. 4.6 illustrates that mixes with larger number of A9 nodes are more energy-efficient, because the configurations in such mixes either incur lower energy for a given execution time or execute faster for a given energy budget. Furthermore, mixes with larger number of E5 nodes compared to K10 nodes are more energy-efficient. Hence, for a given power budget, substituting K10 nodes with E5 and further replacing E5 nodes with A9 according to their respective power substitution ratios introduces a more energy-efficient sweet region.

For the RSA-2048 cryptographic encryption, the PPR of E5 is the best among the three types of nodes. As the PPR of K10 and A9 are very close (Table 4.1) and because the peak power consumed of K10 node is much higher than that of an A9 node, for a given power budget, the cluster can consist of many more A9 nodes rather than K10. Hence, heterogeneous mixes with only E5 and A9 nodes are more energy-efficient than other combinations for mixes with degree of heterogeneity two, as shown in Fig. 4.7. Similarly, among the varied combinations

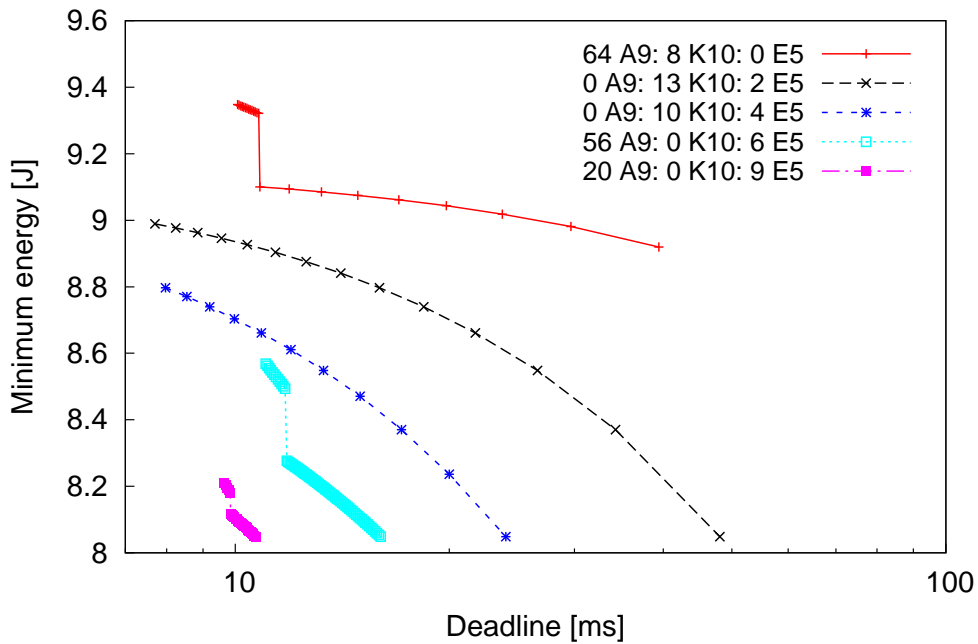


Figure 4.7: Pareto frontier for RSA-2048 with  $d = 2$

of heterogeneous mixes with degree three executing RSA-2048 encryption, mixes with more number of E5 nodes are more energy-efficient as E5 nodes have the best PPR for this program, as illustrated in Fig. 4.8. This observation is similar to the

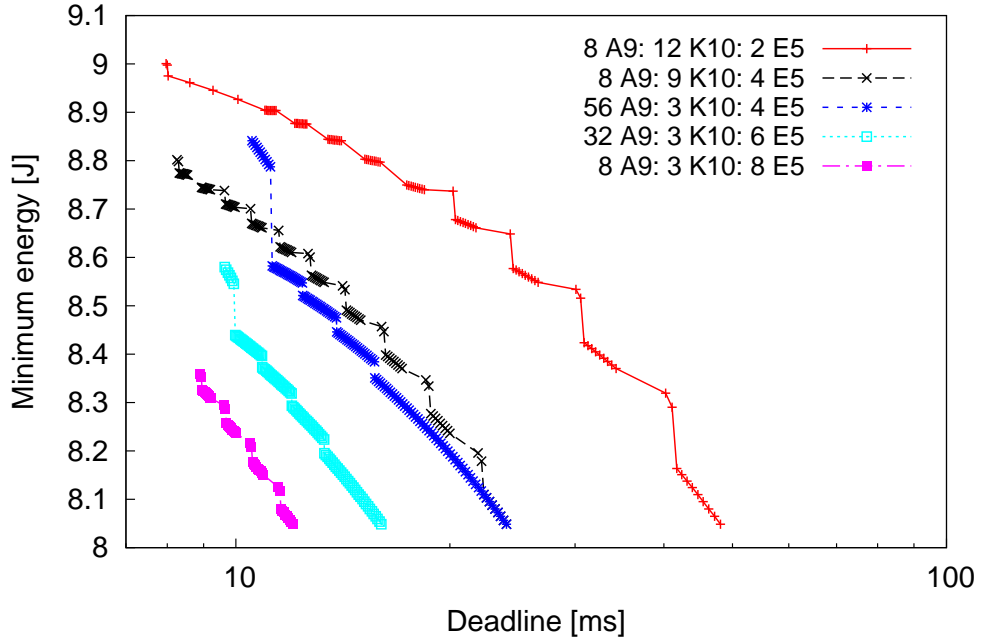


Figure 4.8: Pareto frontier for RSA-2048 with  $d = 3$

analysis of mixes executing EP program where increasing the number of A9 nodes improves the energy efficiency, as the A9 nodes have the best PPR for EP.

### 4.3.3 Analytical Analysis of PSR on Sweet Region

Given a power budget and the power substitution ratio for different types of nodes in the cluster for a given workload, energy efficiency among the different heterogeneous mixes can be analytically computed and compared. The PPRs of different types of nodes for a given workload form a total order which can be represented in the descending order such that:

$$\forall i, j \in [1 \cdots d_{max}] : i < j \Leftrightarrow PPR_i > PPR_j$$

Assuming that the power substitution ratios among the  $d_{max}$  different types of nodes is of the form  $\alpha_i : \alpha_j$ , where  $\alpha_i$  represents the number of type  $i$  nodes that can be substituted with  $\alpha_j$  number of type  $j$  nodes, such that the resulting cluster system does not exceed a given peak power budget. Given  $\pi$  different combinations of the various types of nodes to meet a given power budget  $\beta$ , the most energy-efficient mix should satisfy the constraint:

$$\sum_{i=1}^d n_i \cdot P_i \leq \beta$$

where  $P_i$  is the peak power of node  $i$  and the mix has degree  $d$  with  $n_i$  nodes of type  $i$ .

When the heterogeneous nodes are such that, if  $PPR_i > PPR_j$ , then  $\alpha_i \gg \alpha_j$ , then it implies that having more number of nodes of type  $i$  in the mix increases the overall energy efficiency. However, if the nodes are such that, when  $PPR_i > PPR_j$ , the power substitution ratio  $\alpha_i < \alpha_j$ , then a lower degree of heterogeneity is more energy-efficient than a higher degree of heterogeneity. This hypothesis is validated in Figs. 4.5 and 4.6, where mixes with only A9 and E5 nodes ( $d = 2$ ) are more energy-efficient than mixes with A9, E5 and K10 ( $d = 3$ ) for the EP program. This is because even though E5 nodes have a better PPR than K10 for the EP program, the power substitution ratio is 2:3 between E5 and K10 respectively. As only two nodes of E5 replace three nodes of K10, the advantages due to heterogeneity are not leveraged in this case.

## 4.4 Energy Proportionality Analysis

Energy proportionality was proposed as an important server design principle to address the mismatch between the energy consumed and the amount of useful work

performed [30]. However, energy proportionality quantified by the EP metric [141] has stalled at 80% for individual servers. This stall is referred to as the *energy proportionality wall*, and is attributed to the lack of improvements in the dynamic range of servers [171]. This has fuelled a lot of research to improve the energy efficiency at individual server level by introducing different low power operating modes.

Low power modes such as sleep or shutdown are not viable options as they induce idle periods in the order of minutes. Server shutdown is also impractical due to data availability concerns such as (i) longer response time during traffic spikes and (ii) the necessity to execute many background tasks in typical datacenters [20, 30, 118]. Therefore, research directions in the area of active low-power modes have been explored, where the server can perform some amount of useful work in a low-power state. Barely alive servers [19] and Somniloquy [13] perform only I/O operations in low-power, while KnighShift propose low-power computations using low-power at low utilization. However all of these research works explore energy efficiency at an individual server level. Complementing these techniques, we analyze whether heterogeneous clusters exhibit better cluster-wide energy proportionality.

#### 4.4.1 Brawny versus Wimpy Node

Table 4.4 shows the different energy proportionality metrics described in Section 2.2.2 for the A9 and K10 nodes across all the workloads considered in this chapter. Contrary to intuition, brawny K10 node has better energy proportionality compared to the wimpy low-power A9 node. As the metrics in the table do not illustrate the proportionality of nodes at individual utilization levels, we plot the percentage of peak power consumed by these nodes to depict the variation of the proportionality gap with utilization.

Domain	Program	Useful Operations	Bottleneck
HPC	EP	random numbers	CPU
Web Server	memcached	bytes	I/O
Streaming video	x264	video frames	memory
Financial	blackscholes	stock options	CPU
Speech recognition	Julius	audio samples	CPU
Web security	RSA-2048	key verifies	CPU

Table 4.3: Useful Operations

Program	DPR		IPR		EPM		LDR	
	A9	K10	A9	K10	A9	K10	A9	K10
EP	25.97	34.57	0.74	0.65	0.26	0.34	0.26	0.35
memcached	16.78	11.05	0.83	0.89	0.17	0.11	0.17	0.11
x264	35.54	38.41	0.64	0.62	0.36	0.38	0.36	0.39
blackscholes	32.11	37.30	0.68	0.63	0.32	0.37	0.32	0.37
Julius	30.48	38.10	0.70	0.62	0.30	0.38	0.31	0.38
RSA-2048	35.62	41.19	0.64	0.59	0.36	0.41	0.36	0.41

Table 4.4: Single-node energy proportionality

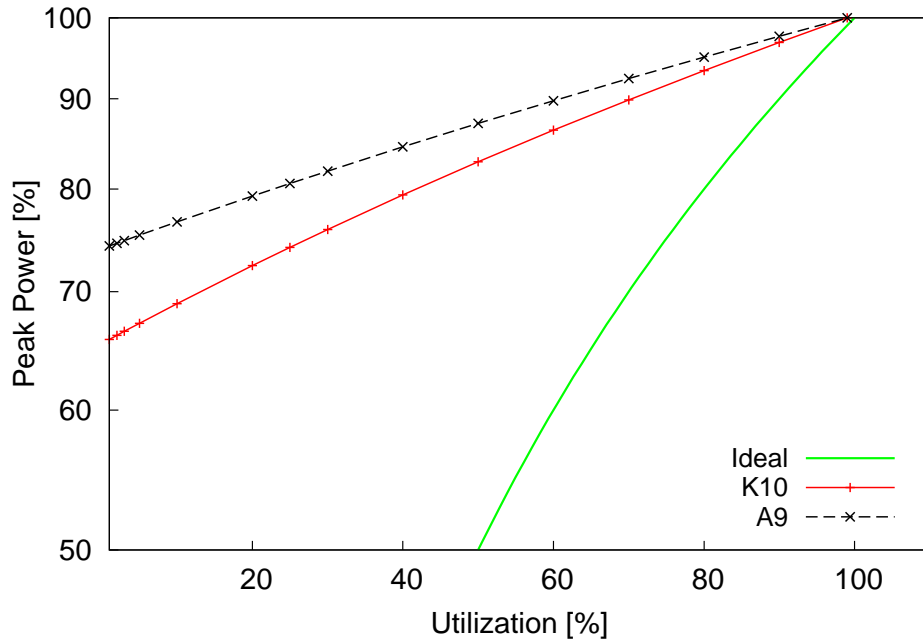


Figure 4.9: Energy proportionality of EP

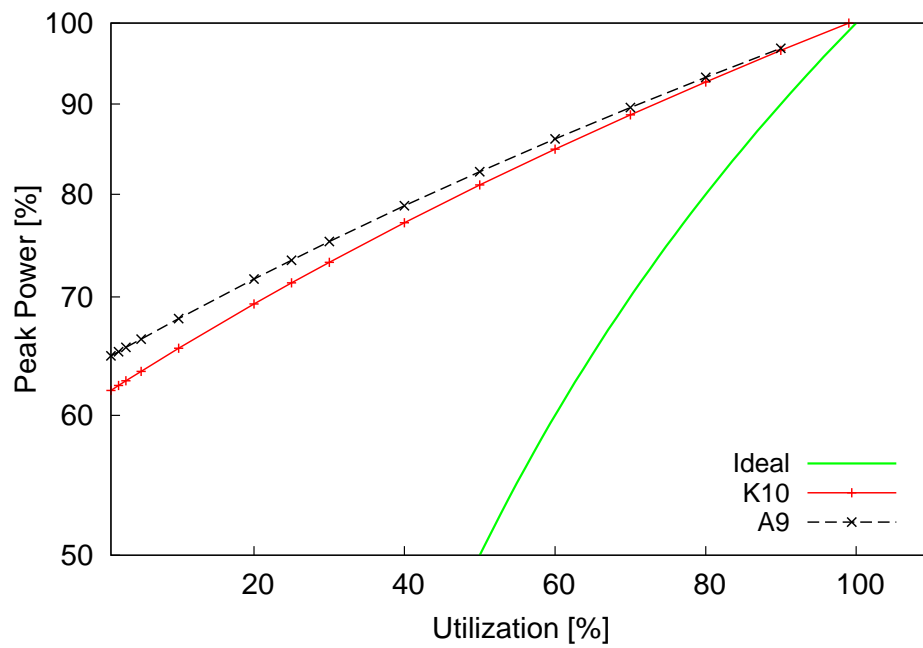


Figure 4.10: Energy proportionality of x264

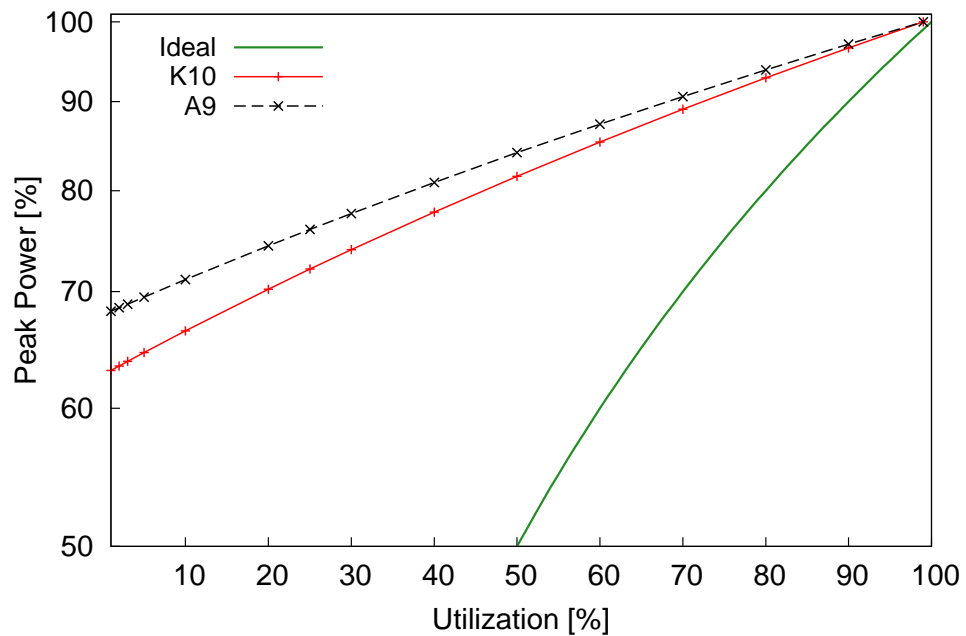


Figure 4.11: Energy proportionality of blackscholes

Figures 4.9, 4.10 and 4.11 plot the energy proportionality of a single AMD Opteron K10 node and ARM Cortex-A9 node for the EP, x264 and blackscholes programs respectively. From the plots, we conclude that usage of K10 nodes in system clusters is more energy-proportional than using the A9 node, for compute and memory intensive workloads. However, comparison between the absolute values of the idle power consumed by the two nodes shows that the idle power of A9 ( $\approx 1.8\text{W}$ ) is at least 25 times lower than that of K10 ( $\approx 45\text{W}$ ). This counter-intuitive result is because existing energy proportionality metrics discussed in Section 2.2.2, do not provide a complete picture as they only quantify the percentage of the power consumption with respect to the peak at different utilization levels. The metrics neither consider the absolute power values nor do they consider performance characteristics.

Next, we compare the two nodes using the performance-to-power ratio (PPR) that factors the throughput of the workload per unit power across utilization levels and is defined as,

$$PPR(u) = \frac{\textit{Throughput}[\textit{operations/s}]}{\textit{Power}[W]}$$

where throughput denotes the number of useful operations performed by the system per unit time. This metric is also used in SPEC benchmark [151]. The useful operations for the different workloads are tabulated in Table 4.3.

Figures 4.12, 4.13 and 4.14 plot the PPR across utilization levels for a single node of K10 and A9 executing EP, x264 and blackscholes workloads respectively. For certain workloads like x264, both the PPR and energy proportionality metrics concur (Figures 4.10 and 4.13), wherein K10 has both a better PPR and proportionality gap compared to A9. However, the comparison of the energy

---

<sup>3</sup>In the energy proportionality plots presented in this thesis, lower is better

<sup>4</sup>In the PPR plots presented in this thesis, higher is better



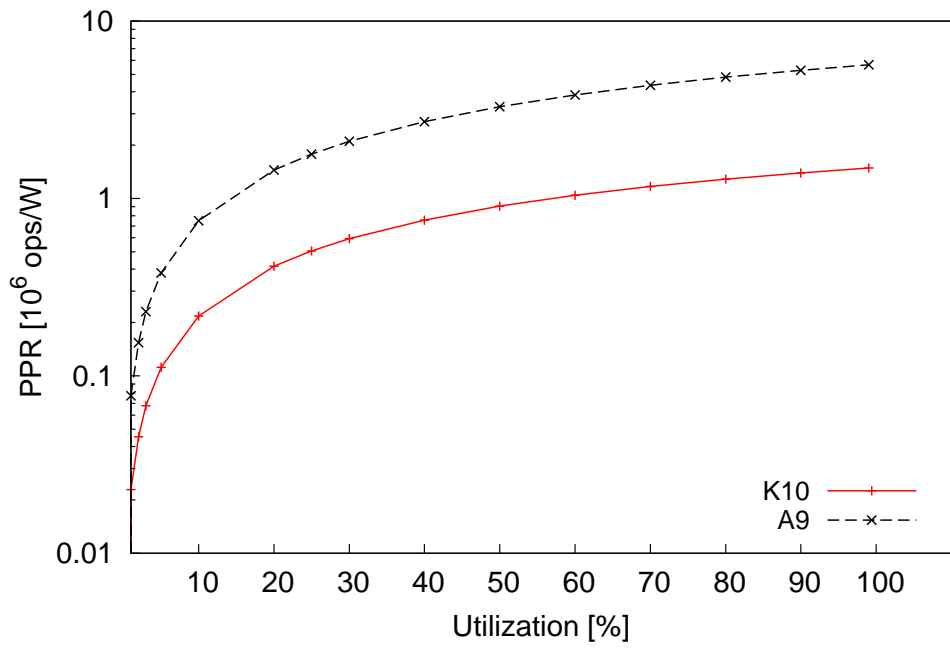


Figure 4.12: PPR of EP

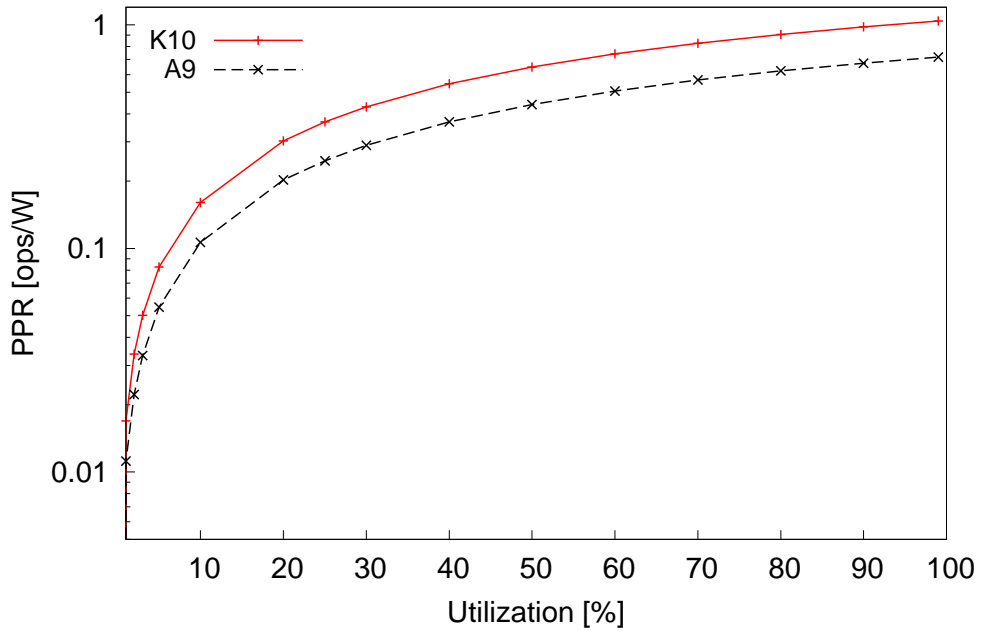


Figure 4.13: PPR of x264

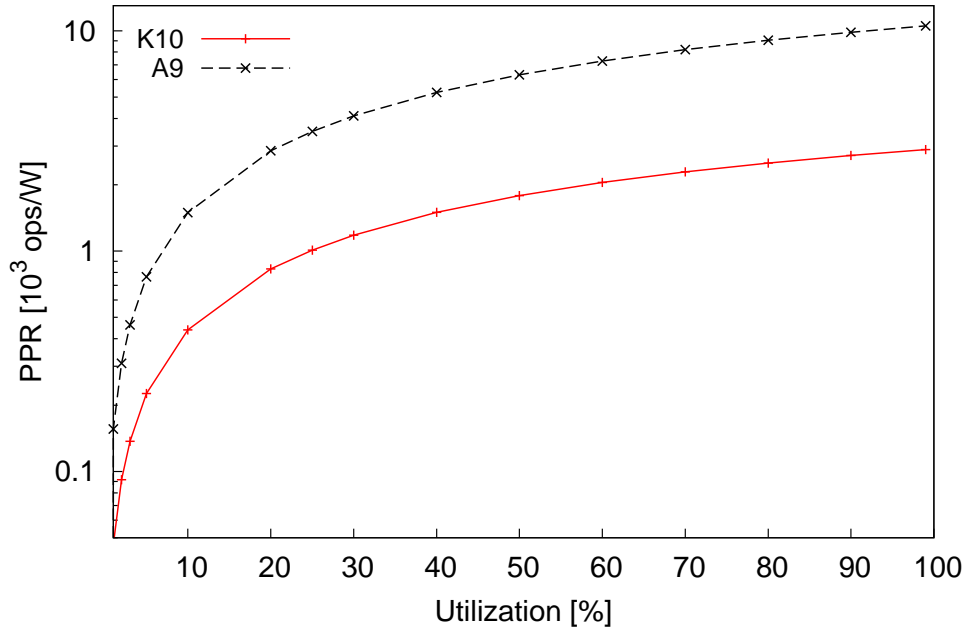


Figure 4.14: PPR of blackscholes

proportionality and the PPR values of the EP and blackscholes workloads show contradictory results in terms of determining the more efficient node.

While the PPR of the A9 wimpy node is better than that of brawny K10 for executing both EP and blackscholes workload, the proportionality gap of the A9 node is bigger than that of K10 for these workloads. This contradicting result stems from the fact that while energy proportionality determines how the server power consumption adapts to different utilization levels, it does not consider the throughput. Thus, using the energy proportionality metrics alone does not always suffice to make decisions regarding the nodes to be used for executing a workload. We further augment this conclusion by illustrating the insufficiency of current energy proportionality metrics using the cluster-level analysis results in the following section.

Program	DPR			IPR			EPM			LDR		
	128 A9 0 K10	64 A9 8 K10	0 A9 16 K10	128 A9 0 K10	64 A9 8 K10	0 A9 16 K10	128 A9 0 K10	64 A9 8 K10	0 A9 16 K10	128 A9 0 K10	64 A9 8 K10	0 A9 16 K10
EP	25.97	32.66	34.57	0.74	0.67	0.65	0.26	0.33	0.34	0.26	0.33	0.35
memcached	16.78	12.44	11.05	0.83	0.88	0.89	0.17	0.12	0.11	0.17	0.12	0.11
x264	35.54	37.73	38.41	0.64	0.62	0.62	0.36	0.38	0.38	0.36	0.38	0.38
blackscholes	32.11	36.10	37.30	0.68	0.64	0.63	0.32	0.36	0.37	0.32	0.36	0.37
Julius	30.48	36.39	38.09	0.70	0.64	0.62	0.30	0.36	0.38	0.30	0.37	0.38
RSA-2048	35.62	39.92	41.19	0.64	0.60	0.59	0.36	0.40	0.41	0.36	0.40	0.41

Table 4.5: Cluster-wide energy proportionality

#### 4.4.2 Cluster-wide Energy Proportionality

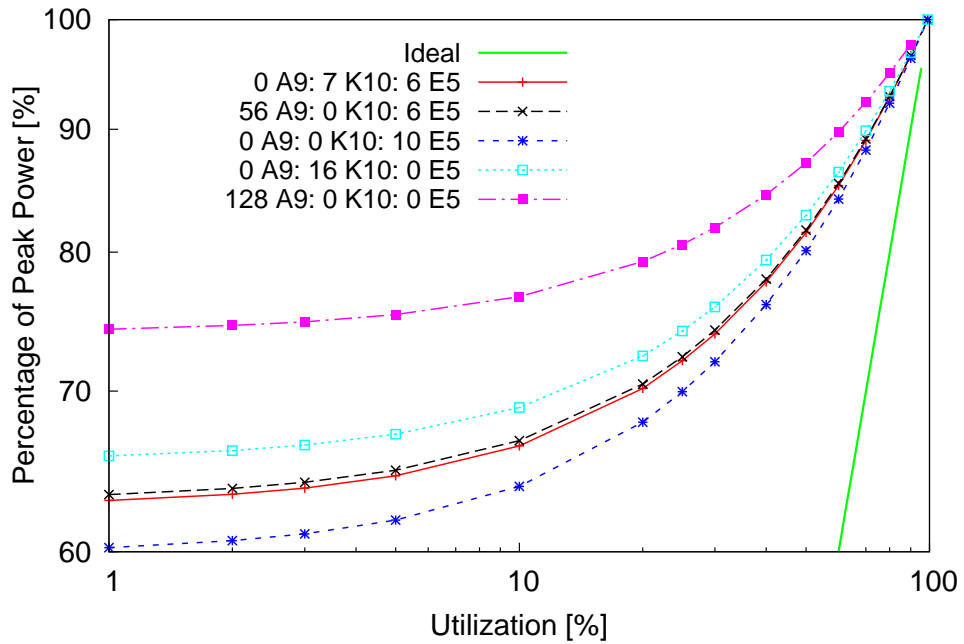
For analysing cluster-wide energy proportionality, we use the same metrics as for the single node case and the power consumed by the whole cluster under consideration. For cluster utilization, we consider a workload with batches of parallel jobs of the same type being executed by all the nodes in the cluster. While all nodes of the same type execute similar proportion of the workload, the amount of workload executed by nodes of different types is determined by matching the execution rates among the different types of nodes, such that all nodes finish executing at the same time. Thus, in our approach, the idling period of all nodes in a system configuration is approximately the same and depends only on the cluster utilization level.

Wong et al. [172] considered cluster-wide energy proportionality but their analysis pertained to increasing the number of nodes in a cluster with an increase in the utilization. In contrast, we assume a fixed system configuration across all utilization levels to ensure that the energy proportionality analysis is an unbiased comparison among different cluster mixes. Furthermore, to ensure a fair comparison among cluster mixes, we constrain the peak power of the cluster using a fixed power budget. This is motivated by the fact that datacenters often have an upper bound on their peak power consumption. Based on peak power consumed by the A9 and K10 node, we analyze both homogeneous clusters and cluster mixes such that the total peak power is within the allocated budget.

For this analysis we consider a peak power budget of 1kW. The combination of the different heterogeneous cluster mixes within a 1kW power budget can be determined using a power substitution ratio of 8:1 between the A9 and K10 nodes. This ratio is derived based on the peak powers of the A9 and K10 nodes. Since one A9 node draws a peak power of 5W and one K10 node draws a peak power of 60W, one K10 node can be replaced by 12 A9 nodes. Factoring about 20W peak power drawn by the switch, which connects the A9 nodes gives us a power substitution ratio of 8:1. For a peak power budget of 1kW, the homogeneous configurations consist either of 128 A9 nodes or 16 K10 nodes. As 16 K10 nodes draw a peak power of around 960W and since the switch connecting these nodes draws about 40W of peak power, the total power is 1kW, thus meeting the power budget constraint. For the A9 cluster using the power substitution ratio of 8 A9 nodes for every K10 node results in a 128 node cluster which is within the 1kW power budget.

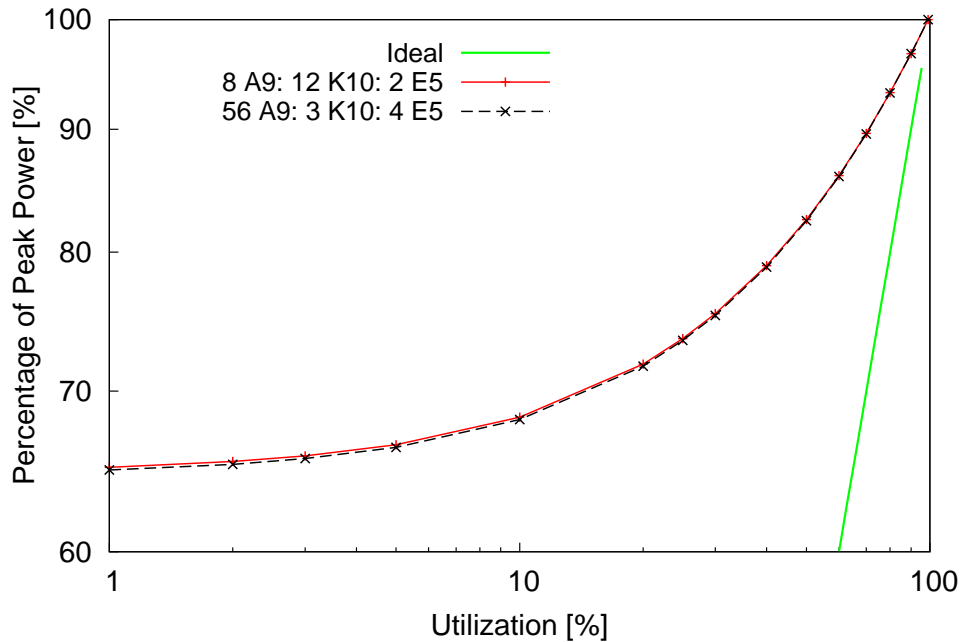
The values of the energy proportionality metrics for the homogeneous clusters and a heterogeneous cluster are shown in Table 4.5. As seen from the values in the table, high-power homogeneous clusters consisting of K10 nodes have better energy proportionality compared to the homogeneous cluster with A9 nodes. However, the K10 cluster consumes an idle power of around 720W which is about three times higher compared to the A9 cluster. Thus, this contradiction show that using only energy proportionality metrics may not always reveal the most efficient system configuration. This conclusion is further augmented by comparing the cluster-wide energy proportionality and PPR for different workloads.

Figs. 4.15 and 4.16 plot the cluster-wide energy proportionality of executing EP workload on clusters with degrees of heterogeneity one, two and three respectively. Cluster configurations used in these plots are constrained by a maximum peak power budget of one kilowatt. While all different mixes have the same peak power

Figure 4.15: Cluster-wide energy proportionality  $d = 1$  and  $d = 2$ 

budget, power usage at different system utilization levels vary and thus the mixes exhibit different proportionality gaps. The Proportionality Gap (PG) defined by Wong et al. [170] is a measure of deviation between the server's actual energy proportionality and the ideal energy proportionality at individual utilization levels. This definition is adapted to measure cluster-wide proportionality gap (CPG) and is defined as the deviation between the ideal energy proportionality and the actual usage of the cluster.

As observed from the plots in Figs. 4.15 and 4.16, clusters with degree one consisting of only E5 nodes have the best CPG compared to the different cluster mixes with degree two and three. Among the homogeneous clusters of degree one, systems with A9 nodes has the worst CPG at all utilization levels, followed by K10 and then E5. This order is in complete contrast to the peak idle power consumed by the A9, K10 and E5 nodes. This is because CPG considers the relative power consumed by the system as a percentage ratio with respect to the

Figure 4.16: Cluster-wide energy proportionality  $d = 3$ 

idle power and does not consider absolute power usage. In contrast to the energy proportionality plots, our previous analysis using the energy-deadline Pareto frontier illustrates that heterogeneous systems are more energy-efficient than homogeneous ones. This is because energy proportionality and the CPG metric in particular, do not consider execution time deadline, which is an important factor while choosing a system configuration for execution. The third counter-intuitive observation is that among heterogeneous mixes of degree three, mixes that are more energy efficient have the same CPG compared to mixes that are less efficient. For example, 56A9:3K10:4E5 mix is more efficient (shown in Fig. 4.6) but has the same CPG (shown in Fig. 4.16) compared to 8A9:12K10:2E5 mix. This contrast is because while cluster-wide energy proportionality is a useful metric to determine what cluster configuration to use based on the utilization levels, it alone does not suffice as it does not consider the energy usage with respect to an execution time deadline. Hence, to analyze the energy efficiency of programs

at different utilization levels, throughout this section we use the performance to power ratio (PPR) metric.

### 4.4.3 Does Heterogeneity Scale the Energy Proportionality Wall?

The energy proportionality and PPR values of homogeneous configurations versus heterogeneous mixes indicate that homogeneous configurations are better than heterogeneous ones. While heterogeneity enables a sweet spot region, wherein increasing the execution time deadline causes a reduction in the energy used [137], it is unclear whether the large system configuration space due to heterogeneity helps scaling the energy proportionality wall. Thus, in this section, we analyze the energy proportionality of the heterogeneous configurations with respect to an execution time deadline constraint. While Section 4.1 shows that, among the large set of configurations, there exists a Pareto-optimal set of heterogeneous configurations that form the energy-deadline Pareto frontier, the impact of these Pareto-optimal configurations on cluster-wide energy proportionality is non-obvious. Figure 4.17 plots the energy proportionality for the configurations on the Pareto-frontier using a maximum of 32 A9 and 12 K10 nodes, executing the EP workload. This maximum number of nodes constraint is chosen among the heterogeneous configurations within a 1kW power budget.

As observed, several Pareto-optimal configurations have sub-linear energy proportionality, as they fall below the ideal energy proportionality. These sub-linearly proportional configurations arise by reducing the number of nodes with lower PPRs. For example, given a maximum of 32 A9 and 12 K10 nodes executing the EP workload, a configuration with only 25 A9 and 7 K10 nodes exhibits sub-linear proportionality for cluster utilization above 50% as shown in Figure 4.17. These

configurations with smaller number of A9 and K10 nodes consume less energy than ideal and they trade-off execution time to save energy. This trade-off with respect to a response time deadline is not clear from the energy proportionality plots alone. Hence, the response time analysis for these configurations gives additional insights in choosing an energy-proportional configuration.

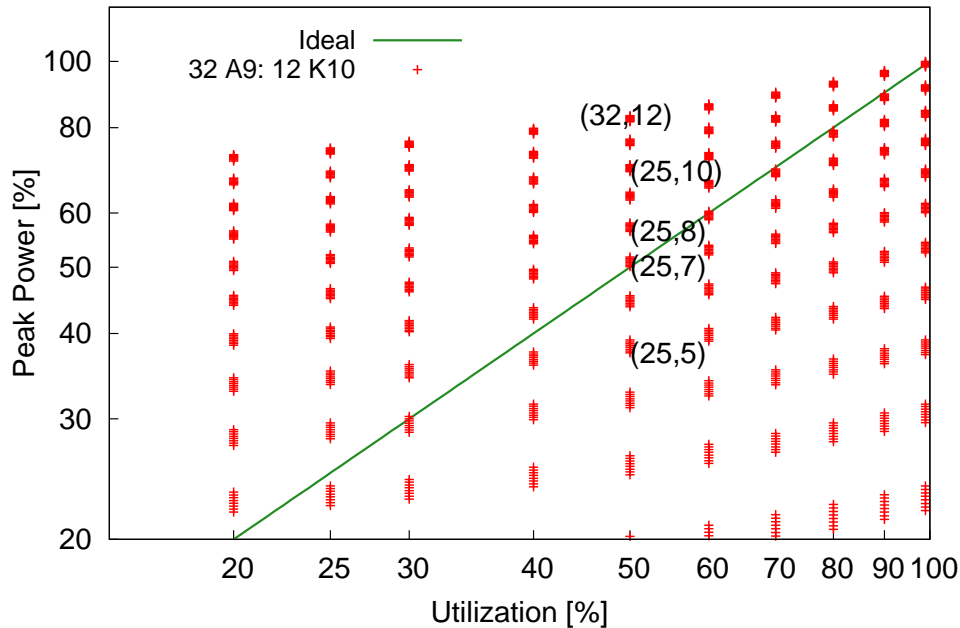


Figure 4.17: Energy proportionality of Pareto-optimal configurations for EP

Figure 4.18 plots the 95th percentile response times for the EP workload executed on different heterogeneous mixes that have sub-linear energy proportionality. This plot shows that the response time difference among the configurations is in the sub-millisecond range. *Thus, we show that heterogeneity introduces sub-linearly proportional configurations that do not impact response times.* However, this observation holds only when the PPR of wimpy nodes is better than the PPR of brawny nodes. For workloads such as x264, where the brawny clusters outperform wimpy clusters, heterogeneity introduces sub-linear energy-proportional configurations, but the execution time is degraded to the order of seconds.



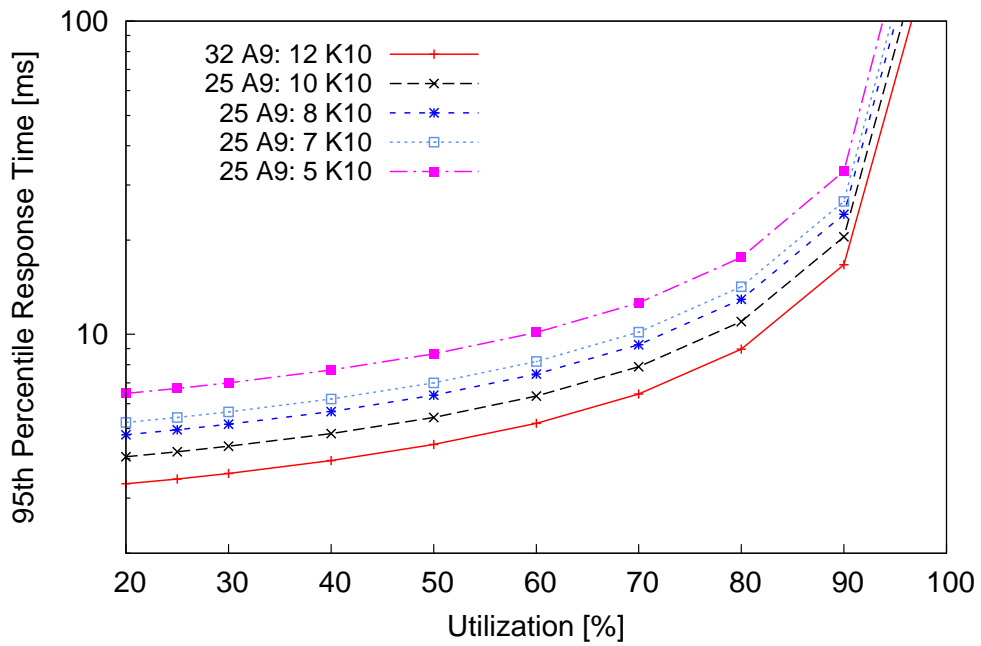


Figure 4.18: Response time of sub-linear heterogeneous mixes for EP

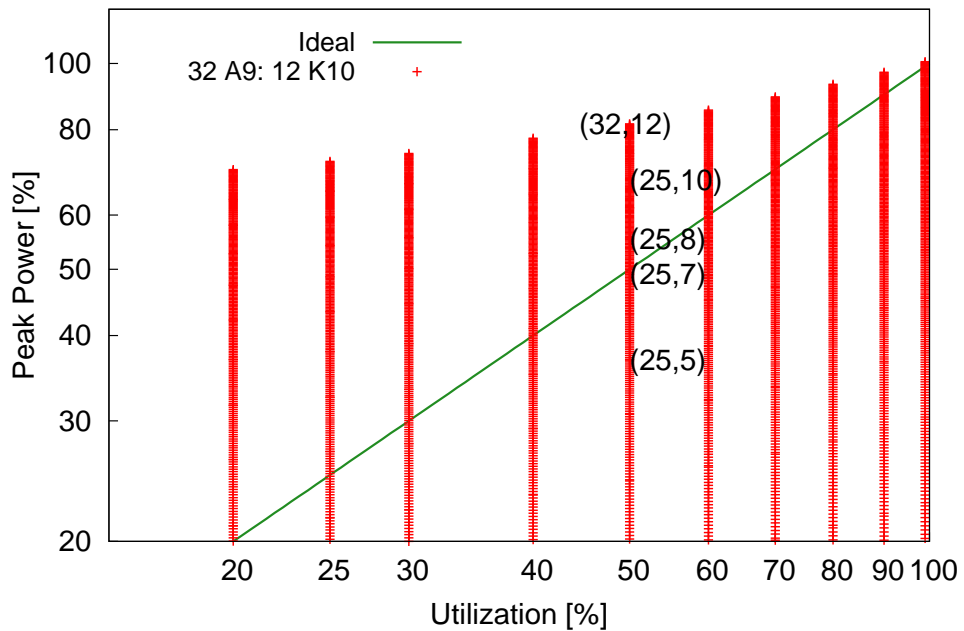


Figure 4.19: Energy proportionality of Pareto-optimal configurations for x264

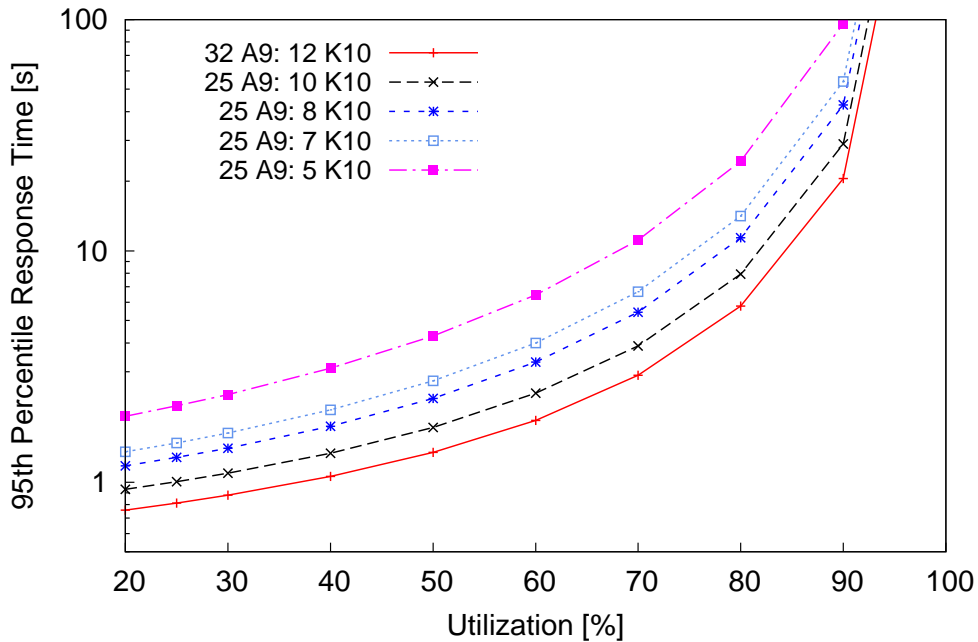


Figure 4.20: Response time of sub-linear heterogeneous mixes for x264

Figures 4.19 and 4.20 plot the energy proportionality and the 95th percentile response time for heterogeneous clusters with a maximum 32 A9 and 12 K10 nodes executing the x264 workload. While the number of sub-linear configurations for x264 is larger compared to the EP workload, these configurations suffer from response time degradation in the order of seconds as shown in Figure 4.20. Thus, heterogeneity does not scale the proportionality wall for workloads such as x264.

## 4.5 Summary

Heterogeneity is becoming the norm in systems today and it offers a larger system configuration space due to different combinations of system parameters such as types of nodes, number of nodes of each type, number of active cores per node and the operating core clock frequency. This chapter addresses some of the challenges due to this large configuration space and applies the core model to determine an energy-efficient *mix* of nodes that services a job while maintaining a service

time deadline. We obtain a Pareto-optimal set of configurations by *matching* the execution time of different nodes to minimize system idle time. This *mix and match* approach exposes a “sweet region” containing a set of sweet-spot configurations where the energy used by a job reduces as its service time deadline is relaxed.

Sweet-spot configurations either use minimum energy to meet a given execution time deadline or meet a given energy budget in a minimum possible execution time. Next, we observe that these sweet-spots form a distinct energy-deadline *Pareto frontier*. Despite the explosion of the configuration space, we show that the Pareto frontier can be analytically determined using the node performance-to-power ratios (PPR). Next, given a power budget, we show that replacing low PPR nodes with higher PPR nodes leads to a more energy-efficient configuration. Lastly, we study the impact of heterogeneity on energy proportionality. We show that the PPR metric is more apt compared to the cluster-wide proportionality gap metric for determining energy-efficient configurations.

For a given power budget, we show that inter-node heterogeneous clusters has advantages over homogeneous clusters by scaling the energy proportionality wall using sub-linear energy-proportional configurations. These configurations are enabled due to inter-node heterogeneity by replacing a few low PPR nodes with high PPR nodes while maintaining a given power budget. While these configurations save energy by trading execution time, we show that for workloads that have better PPR on wimpy systems, these configurations have minimal impact on the 95th percentile response time.



# Chapter 5

## Extension to Intra-node

## Heterogeneity with VPU

With the slowing down of “Moore’s law” and dark-silicon limiting the number of active cores in a multi-core processor, mixing CPUs and accelerators seems like a viable approach to scale-up parallel computing performance. The recent years have seen the wide adoption of accelerators by the HPC community. Among the Top500 systems in November 2015, there were about 104 systems with Vector Processing Units (VPUs) as accelerators [8].

Traditionally, while GPUs have been dominating the accelerator arena, the launch of Intel’s Knight Corner in 2012, have seen another class of accelerators being adopted mainstream, namely the Many Integrated Core (MIC) architecture. The increasing adoption of this architecture is evident in the Top500 systems, where 32 out of the 104 systems with accelerators use the Intel Xeon Phi coprocessor which is based on the MIC architecture.

With at least 50 cores, Intel Xeon Phi coprocessor adds high parallelism on a single node and has a theoretical peak of two TFLOPS for single precision, one TFLOPS for double precision and over 352 GB/s of memory bandwidth. This

performance gain as well the flexibility to be used both as a coprocessor or a standalone processor, offers a new intra-node heterogeneous platform for HPC applications. In contrast to accelerators using GPU, Xeon Phi has a general-purpose programming environment and can be programmed with common programming languages, thus making it even more popular among HPC users.

However, from a system perspective, coprocessors based on MIC architecture offer a large system configuration space to execute a parallel application. For a given parallel program, reaching the theoretical peak performance of the Xeon Phi is challenging. It depends a lot on the scaling, how to bind threads to cores, the degree of vectorization and memory usage of the applications. Hence, for a HPC user, determining the optimal system configuration to execute the parallel application is non-trivial and poses a number of research challenges such as:

1. For a given program, what is the number of threads that achieves the best performance?
2. For a given program and the number of threads, what thread affinity mode achieves the best performance?

Answers to these questions help both application developers to gain insights on program hot-spots, and system designers to identify capacity bottlenecks, and thus optimize software-hardware co-design to improve system performance. This chapter addresses these challenges and proposes an extension to the core modeling approach proposed in Chapter 3, to illustrate how that approach can easily be scaled and applied to a intra-node heterogeneous system architecture such as the Intel MIC architecture.

This chapter presents an approach to determine time efficient system configurations for executing a parallel program using a measurement-driven analytical model. The proposed analytical model is formulated using parametric values ob-

tained from baseline executions of the application to measure workload and architectural artefacts. The key novelties of our approach are modeling both inter and intra-core resource overlaps and resource contention.

Given a parallel program, the proposed approach determines the system configuration in terms of the tuple, number of cores and number of threads per core. Thus the approach provides a systematic method to users of MIC architecture systems to determine the thread affinity mode and the number of threads for efficient execution of a HPC application.

The proposed model is validated against direct measurements on an Intel Xeon Phi card having a 5110P coprocessor with 60 cores operating at 1.053 GHz using a range of NAS NPB benchmarks, including both kernel and applications [26]. Validation results for all possible configurations show that our model accuracy is within reasonable bounds of less than 15%. As an example, we apply our model to determine energy-time efficient configurations for HPC applications.

The key contributions of this chapter are:

1. A measurement-driven analytical model to determine time-efficient performance of parallel program. In contrast to current approaches, we model both inter and intra-core resource overlaps, memory contention within and across multiple cores in MIC architecture.
2. show that parallel programs executing on MIC architecture systems exhibit Pareto-optimal configurations, that execute in the minimum possible time for a given energy budget or consume the minimum energy for a given execution time deadline.
3. show the impact of performance-to-power ratio (PPR) metric to determine what thread affinity mode is more optimal when offloading on accelerators

with MIC architecture.

## 5.1 Overview

The objective of the approach is to determine a time-efficient configuration for executing a given parallel program on a coprocessor based on the MIC architecture. While the MIC architecture offers immense amount of thread-level parallelism (TLP), it is non-trivial for the user of such a system to determine the optimal system configuration for execution among the huge configuration space offered by such a system. A system configuration space is defined as a tuple consisting of the number of cores and the number of threads per core<sup>1</sup>. Due to the large TLP offered by the MIC architecture, users of such systems are given different options of utilizing the underlying resources using the different thread affinity modes. Thus, the configuration tuple determined using our approach translates to the number of threads and the thread affinity mode to be used by the user.

The proposed approach determines the system configuration that has the best execution time performance by determining the optimum number of threads ( $\tau$ ) per core and the number of cores ( $c$ ) to execute the program. Counter-to-intuition, scheduling the maximum number of threads per core does not necessarily translate to the best time performance because threads within a core compete with each other for shared resources such as memory. This is modeled in our approach as intra-core contention. Alternatively, keeping intra-core contention to a bare minimum and scheduling only a single thread per core may not be optimal as the threads executing across cores also contend for shared-memory, which we define as

---

<sup>1</sup>Considering a single Xeon Phi node can have 60 cores with 4 threads executing per core, and two different thread affinity modes, compact and scatter, result in a total configuration space of  $60 \times 4 \times 2 = 480 - 3$  (common configurations) = 477 configurations. These configurations grow linearly with the increase in the number of cores, or number of threads per core or the possible thread affinity modes.



inter-core contention. The performance impact of choosing either the policy with (i) maximum number of threads per core (compact) or (ii) single thread per core but using more cores (scatter) is non-trivial and thus we address this challenge by modeling both intra- and inter-core contention for shared memory.

To infer an application resource demands such as CPU and memory, we characterize the workload using baseline executions to derive program and architectural artefacts. These baseline executions are performed using a small program input size. Using these measurements, we derive useful work cycles and the intra- and inter-core memory contention. The effect of memory contention is observed using the measured parameter, the number of stall cycles due to cache misses. This is used to determine the total cycles for executing a program with larger input size. The notations used in the extended version of the core model for intra-node heterogeneity are described in Table 5.1, and the other notations are as described in Table 3.1 of Chapter 3. The proposed approach is outlined in Figure 5.1.

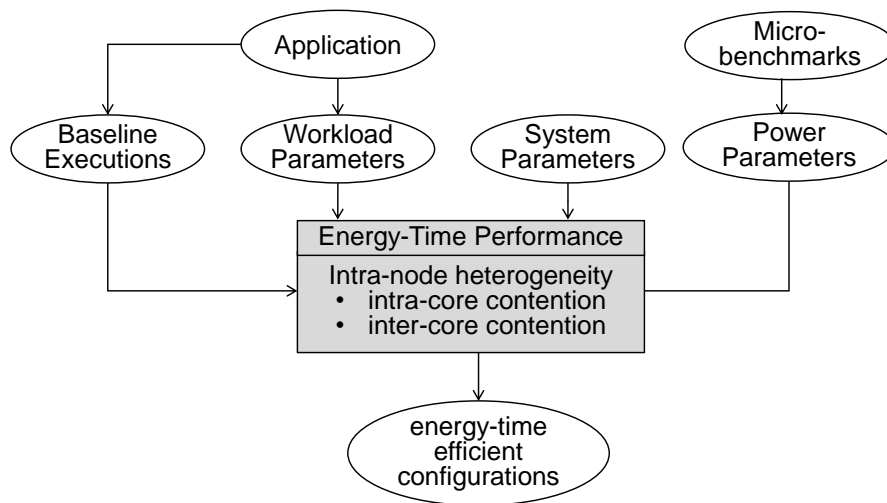


Figure 5.1: Approach for intra-node heterogeneity

Symbol	Description
<b>Baseline Execution</b>	
$m_{intra,s}$	no. of intra-core memory-related stall cycles in $\mathcal{P}_s$
$m_{inter,s}$	no. of inter-core memory-related stall cycles in $\mathcal{P}_s$
<b>System Parameters</b>	
$c_{max}$	maximum no. of cores in MIC architecture system
$\tau_{max}$	maximum no. of threads per core
$f$	core clock frequency
<b>Time Model</b>	
$m_{intra}$	intra-core memory-related stall cycles for $\mathcal{P}$
$m_{inter}$	inter-core memory-related stall cycles for $\mathcal{P}$
$c$	no. of cores per accelerator node executing program $\mathcal{P}$
$\tau$	no. of threads per core executing $\mathcal{P}$
$T_{intra}$	non-overlapped intra-core stalls
$T_{inter}$	non-overlapped inter-core stalls

Table 5.1: Model parameters

### 5.1.1 MIC Architecture

An overview of the Many Integrated Core (MIC) architecture is illustrated in Figure 5.2 using the Intel Xeon Phi co-processor as an example implementation. The Intel Xeon Phi coprocessor is primarily composed of processing cores, caches, memory controllers, PCI express client logic, and a very high bandwidth, bidirectional ring interconnect. Each core has a 32-KB L1 instruction cache, a 32-KB L1 data cache and a private 512-KB L2 cache that is kept fully coherent by a globally-distributed tag directory.

The memory controllers and the PCIe client logic provide a direct interface to the GDDR5 memory on the coprocessor and the PCI express bus, respectively. All these components are connected together by the ring interconnect. There are 64 tag directories (TD) connected to the ring. Address mapping to tag directory is based on the results of hash functions of the memory addresses, hence, the memory addresses will be distributed evenly over the ring, which helps better communication. The memory controller (GDDRMC) is distributed symmetrically around the circle. The addresses are distributed evenly across the controllers,

thereby eliminating “hot spots” and provide unified access model.

The interconnect is implemented as a bidirectional ring. Each direction consists of three separate rings [66]. The first ring, also the largest and most expensive ring of the three is the data block ring. This data block ring is 64 bytes wide to support high bandwidth. The address ring is smaller and is used to send read/write commands and memory addresses. Finally, the smallest ring and the least expensive ring is the acknowledgment ring, which sends flow control and coherence messages. When the cores access the L2 cache and cannot find the necessary data (cache miss occurs), one request will be sent to the tag directory. If data is located on the L2 cache of another core, a request will be sent to that core and the data will be sent via the data ring. If the requested data is not found

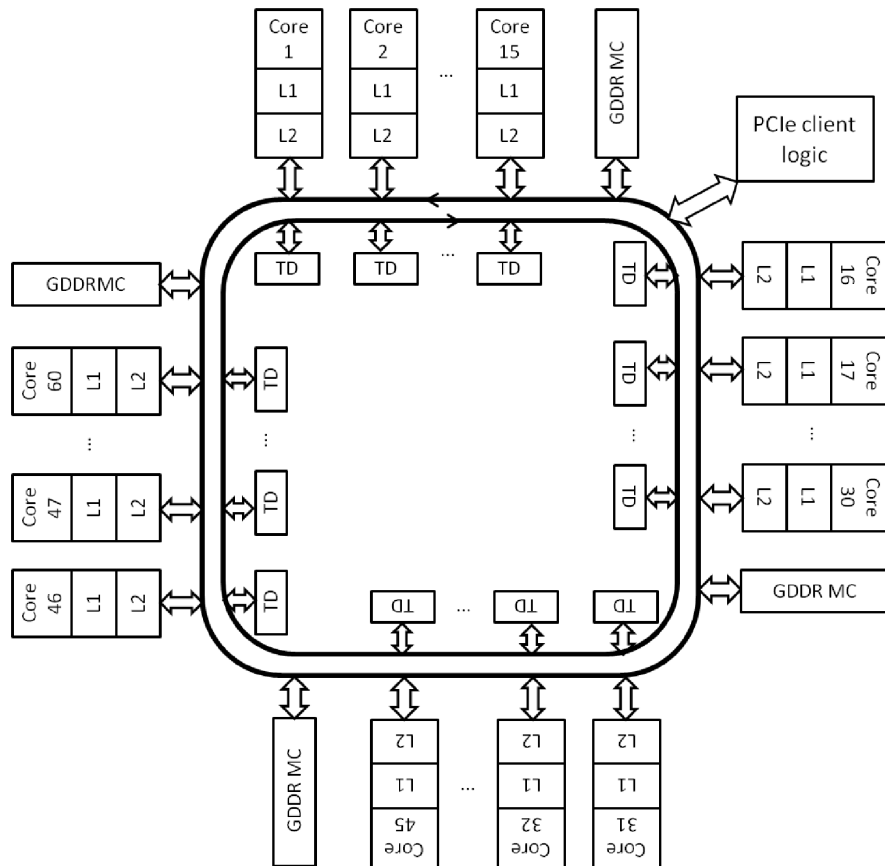


Figure 5.2: MIC architecture

in any caches of any cores, a memory address is sent from the tag directory to the memory controller.

## 5.2 Time Performance Model

In this section, we derive the execution time,  $T$ , for a shared-memory parallel program  $\mathcal{P}$  with input size  $S$ , executing on an accelerator based on the MIC architecture. An abstract view of a shared-memory parallel program is illustrated in Listing 6.1. While this shows only one loop, a typical program can consist of multiple loops. Adding more computational resources to a parallel program, splits the computations or useful work and thus achieves speedup but also incurs overheads due to thread communication and contention.

While intra-core communication using the shared L1 cache is faster, the overheads due to intra-core contention for the L1 cache causes speedup losses in parallelism. Similarly, when only a single thread is scheduled on a single core, it minimizes intra-core contention, but inter-core contention for the L2 memory causes speedup losses. We model both intra and inter-core contention for shared-memory by considering the overlap of these memory accesses with useful work or program computations.

The execution time of a parallel program executing  $\tau$  threads on  $c$  cores depends on the total number of cycles executed by the thread in the critical path of the execution. Thus,

$$T = \max_{\substack{1 \leq \tau \leq \tau_{max} \\ 1 \leq c \leq c_{max}}} T_{\tau,c} \quad (5.1)$$

The execution time of a thread executing on a core in MIC architecture systems depends not only on the computation time but also the contention time for shared resources both within and across cores. We model this execution time as service

```
for(iteration = 1..S)
{
  # pragma omp parallel //  $\tau$  threads on  $c$  cores
  {
    /* computations or useful work */
    ....
    ....
    ....
    /* intra and inter-core contention
    for shared-memory */
  }
}
```

Figure 5.3: Abstraction of a parallel program

time offered by two system resources in the MIC, namely, cores and memory. However, the response time of these resources overlap and thus the execution time of the thread cannot be simply determined by a simple summation of the service time and waiting time of both the servers, namely core and memory.

As the CPU cores have deep pipelines to simultaneously compute and access memory, we split the execution time of the thread into the useful work done by the core along with the overlapped memory accesses, and consider the non-overlapped time of core separately. Therefore, we derive this execution time by considering time overlapped with computation ( $T_{work}$ ) and non-overlapped time ( $T_{stall}$ ) waiting for a shared resource. Hence,

$$T_{\tau,c} = T_{work} + T_{stall} \quad (5.2)$$

The execution time on the core and the overlapped memory access time is determined by using the cycles spent in the execution stage of the pipeline, and the core clock frequency. Thus,

$$T_{work} = \frac{cycles_{work}}{f} \quad (5.3)$$

The number of overlapped cycles between computation and memory access depends on the instruction level parallelism (ILP) of the program and the underlying processing core. We measure this artefact by using a program subset  $\mathcal{P}_s$  and determine the useful work cycles executed by a core ( $w_s + b_s$ ). This in turn is determined by measurements using the PAPI [167] hardware counters for cycles and instructions for the program  $\mathcal{P}_s$ .

$$WPI = \frac{w_s + b_s}{I_s} \quad (5.4)$$

We validate our hypothesis in Section 5.3.2 that as programs scale from  $\mathcal{P}_s$  to  $\mathcal{P}$ , the WPI remains approximately the same, as both overlapped work cycles and instructions scale by the same amount for a given program and system, as these are effects of ILP. Thus, the overlapped work cycles for a program  $\mathcal{P}$  can be determined using the measured WPI for  $\mathcal{P}_s$  and using the scaling factor  $S$  for determining the instructions of program  $\mathcal{P}$ . This scaling of instructions with program input size is validated in Section 5.3.2.

$$cycles_{work} = WPI \times S \times I_{\mathcal{P}_s} \quad (5.5)$$

Next, we discuss the impact of TLP on the execution time, and derive how the model determines the non-overlapped cycles, due to both intra and inter-core contention. We use two separate measured parameters to determine both the contentions. While the first parameter  $m_{intra,s}$  measures the intra-core contention for the shared L1 cache using the compact thread affinity mode, the second parameter  $m_{inter,s}$  measures the inter-core contention for memory.

### 5.2.1 Intra-core contention

As the number of threads increase within a core, it is expected that the execution time improves due to TLP. But, there are also parallelism losses that happen and for an efficient software-hardware co-design it is imperative to gain insights into these losses. As the number of threads within a core increase, the number of instructions executed per thread decreases due to TLP, but the number of cycles does not decrease by the same ratio due to the waiting time of the memory requests per thread at the L1 cache. Thus, we model the parallelism loss due to contention for the shared L1 cache among the threads within a core by measuring the increase in the number of stall cycles.

We measure the increase in the non-overlapped stall cycles by increasing the number of threads per core for program  $\mathcal{P}_s$ , and use this to determine the non-overlapped stall cycles  $m_{intra,s}$  due to L1 cache contention for a given number of cache accesses. If  $\lambda_{intra,\tau}$  is the number of L1 data access requests when  $\tau$  threads are executing in a single core, the intra-core waiting time and non-overlapped service time ( $T_{intra}$ ) is

$$T_{intra,\tau} = \frac{m_{intra}}{f} = \frac{\lambda_{intra,\tau} \times \alpha_{\tau,s}}{f} \quad (5.6)$$

where  $\alpha_\tau$  is the number of stalls per L1 cache access due to  $\tau$  threads executing within a core for program  $\mathcal{P}_s$ . We use this time for determining the execution time for a program executing with compact affinity mode when the number of cores is one, and the number of threads vary from one to four. We use the measured values of  $\lambda_{intra,\tau,s}$  for the  $\mathcal{P}_s$  and derive the parameters for program  $\mathcal{P}$  using the scaling factor  $S$  as:

$$\lambda_{intra,\tau} = \lambda_{intra,\tau,s} \times S \quad (5.7)$$

The scaling of the number of data accesses is validated in Section 5.3.2.

As the program can be compute bound or memory bound in a given core depending on the number of threads being used and the amount of contention, the total execution time of a program  $\mathcal{P}$  using  $\tau$  threads on a single core may be determined as:

$$T_{\tau,1} = T_{work} + T_{intra} \quad (5.8)$$

While the above determines the execution time for TLP on a single core, it does not consider inter-core contention for shared-memory.

### 5.2.2 Inter-core contention

As the number of threads executing a program increase, based on the thread affinity mode, the number of cores increase and thus contend for shared resources such as memory. As there are local tag directories (TD) per core, a memory access to the GDDR5 only happens when the requested access misses the distributed L2 cache across all the cores. Thus, the service rate of the inter-core memory requests varies with the number of misses and the number of prefetches. The Xeon Phi uses a hardware prefetching mechanism that is triggered dynamically identifies cache miss patterns and generates prefetch requests [90]. Thus the service rate of the memory controller depends on the intra-core memory request arrival rate, which is a cumulative requests from both the prefetcher and misses during execution. As both of these requests queue at the memory controller (GDDRMCMC), and with  $\lambda_{inter}$  as the total number of memory requests, the inter-core non-overlapped memory response time is:

$$T_{inter} = \frac{m_{inter}}{f} = \frac{\lambda_{inter,c} \times \beta_{c,s}}{f} \quad (5.9)$$

where  $\beta_c$  is the total number of cache accesses due to  $c$  active cores in the MIC architecture system.



We use this time for determining the execution time for a program executing with scatter affinity mode when the number of threads per core is one, and the number of cores vary from one to 60. We use the measured values of  $\lambda_{inter,c,s}$  for the  $\mathcal{P}_s$  and derive the parameters for program  $\mathcal{P}$  using the scaling factor  $S$

$$\lambda_{inter,c} = \lambda_{inter,c,s} \times S \quad (5.10)$$

The total execution time of a program  $\mathcal{P}$  using a single thread on  $c$  cores is determined as:

$$T_{1,c} = T_{work} + T_{inter} \quad (5.11)$$

The execution time for a program using  $c$  active cores with  $\tau$  threads per core, considering overlap of both intra- and inter-core contention can be derived as the time due to the resource having the maximum response time. Thus, the time due to stalls is determined from both the contentions, and using Equation 5.2, the execution time is:

$$T_{\tau,c} = T_{work} + \max(T_{inter}, T_{intra}) \quad (5.12)$$

### 5.3 Model Parameterization and Validation

The measurement driven inputs to our analytical model are obtained by MIC architecture system characterization and program characterization. We first describe the programs and the systems used for validation of the proposed model. Next, we discuss an extensive validation of the model parameters for each of the hypothesis presented in Section 5.2. Next, we validate the model determined values of cycles against direct measurements of total cycles using the PAPI hardware counters. Lastly, we discuss the power characterization of the MIC architecture system.

### 5.3.1 Workloads and Setup

While our approach is applicable on generic shared-memory parallel programs, we selected a representative subset of six benchmark programs from NASA Parallel Benchmark (NPB) suite [161] for presentation in this section. This subset was chosen to represent different program demands on both CPU and memory resources. These programs also exert different inter and intra-core resource demands. A brief description of the programs and their problem input sizes for class A is shown in Table 5.2.

While the table lists the program input size for only class A, the problem sizes for class A,B and C used are in the ratio 1:4:16. We use the OpenMP version of the program and executed it in the native mode on the Xeon Phi coprocessor. The programs are compiled using the Intel compiler for C and Fortran, with full optimizations (-O3) and the MIC flag (-mmic).

The experiments were conducted on a system consisting of an Intel node with a Xeon Phi card based on the MIC architecture. The host system contains 128GB of main memory and a dual-socket Intel Xeon CPU E5-2680-core processor, each core operates at 2.7GHz. The Phi card used here are the 5110P coprocessor with 60 cores operating at 1.053 GHz. The card has 8GB of GDDR5 memory and is connected to the host through PCI express. Table 5.3 describes details of the systems used.

Program	Description	Problem Size (A)
BT	Dense linear algebra: use matrices/vectors to store data	$64^3$
EP	Embarrassingly parallel: low data dependency, low memory	$2^{28}$
FT	Spectral methods: fast Fourier transform	$256^2 \times 128$
IS	Parallel sorting: bucket sort on integers	$2^{23}$
CG	Sparse linear algebra: data with many 0 values	14000
LU	Lower-Upper Gauss-Seidel solver	$64^3$

Table 5.2: Programs

System	Host	MIC Co-processor
	Intel Xeon E5-2680	Intel XeonPhi 5110P
ISA	x86_64	x86_64
Cores	8	60
Threads/Core	4	4
Clock Frequency	2.7 GHz	1.053 GHz
L1 data cache	32kB / core	32kB / core
L2 cache	256kB	512kB
L3 cache	30MB	NA
Memory	128GB	8GB

Table 5.3: Intel node with MIC co-processor

### 5.3.2 Model Parameterization

#### Work Cycles per Instruction (WPI)

To validate our hypothesis of constant  $WPI$  as workload scales from  $\mathcal{P}_s$  to  $\mathcal{P}$ , we use hardware performance counters from PAPI to measure the cycles, instructions and determine WPI across problem input sizes. Figure 5.4 plots the  $WPI$  for problem sizes from A to C. As seen from the plot, as the input size for the programs scale from 1:4:16 times, the work cycles per instruction (WPI), a system and program characteristic remains approximately constant. Thus the proposed model measures WPI for class A and uses it to predict the work cycles for program input sizes B and C using Equation 5.4. Next, we show the validation results of determining instructions for a larger problem size using a scaling factor.

#### Instruction Scaling

We show the practical application of our approach to users of HPC programs, by using the measured parameters by executing programs with smaller input sizes and then determining the time-efficient configurations to execute the program with the scale-out problem size. Figure 5.5 plots the measured versus predicted number of instructions for class B for the CG and FT program using the measured values of class A. This validates the application of our approach and the usage of the

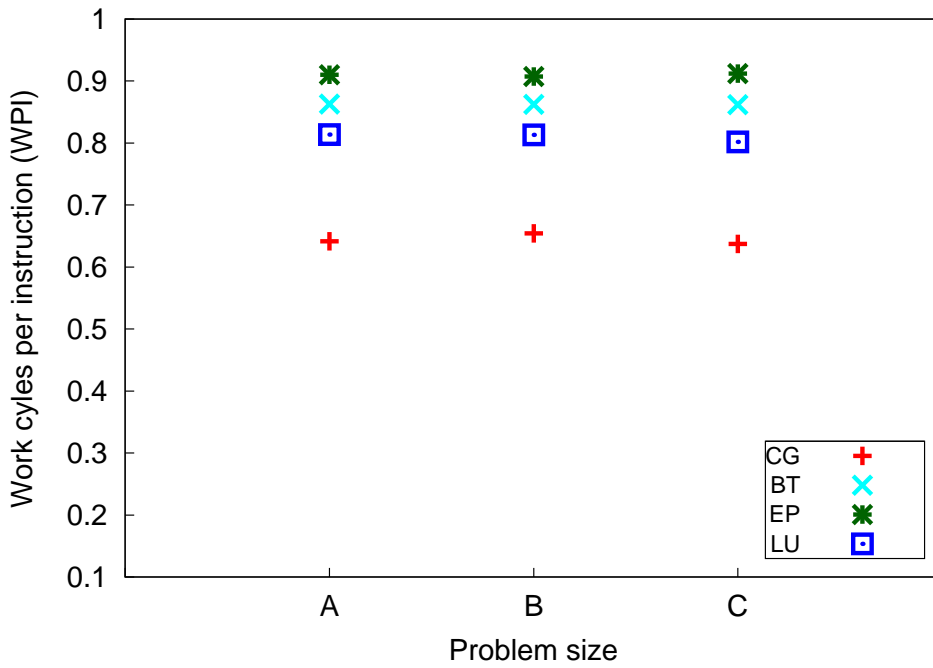


Figure 5.4: WPI validation

scaling factor  $S$  to determine the useful work cycles using Equation 5.5.

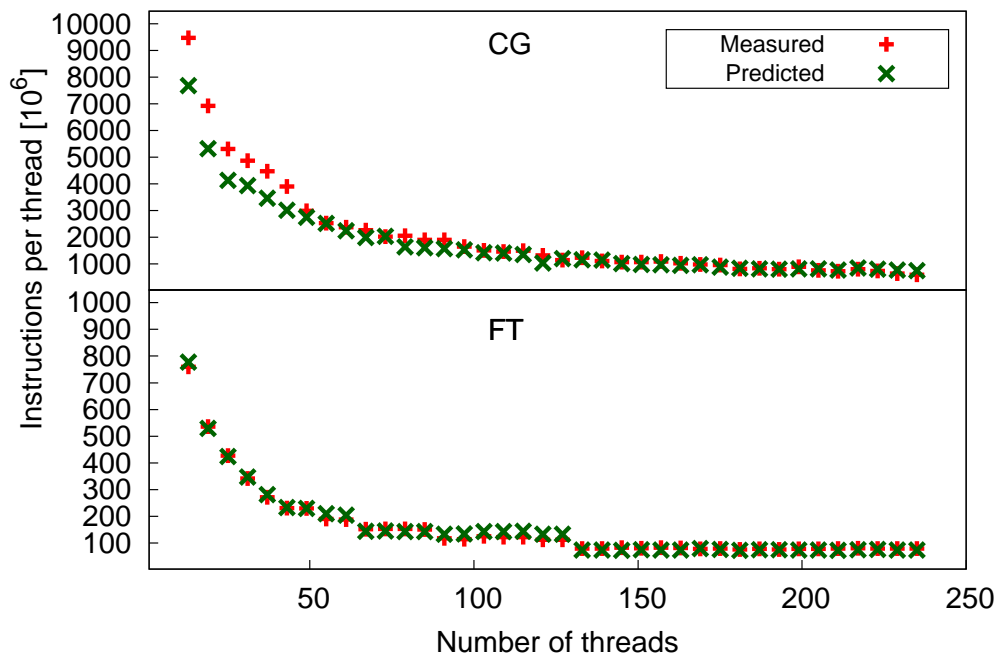


Figure 5.5: Validation of instructions

### Cache Accesses

The memory response time due intra- and inter-core contention depends on the number of requests to memory, model parameter,  $\lambda$ . Here, we show the validation of determining these model parameters for scale-out problem sizes using the baseline measurements for smaller program size,  $\mathcal{P}_s$ . Figure 5.6 plots the measured versus predicted number of cache accesses for the programs FT and IS with input size as class B. The model predicts the class B values using the measurements from class A, which is four times smaller than class B. This validates the application of our approach and the usage of the scaling factor  $S$  to determine the model parameter, number of cache accesses used in Equations 5.7 and 5.10. While all the programs have a scaling factor of four, program CG accesses much more data in B compared to A and while the scaling factor for instructions is four, a separate scaling factor of 25 is used for data accesses.

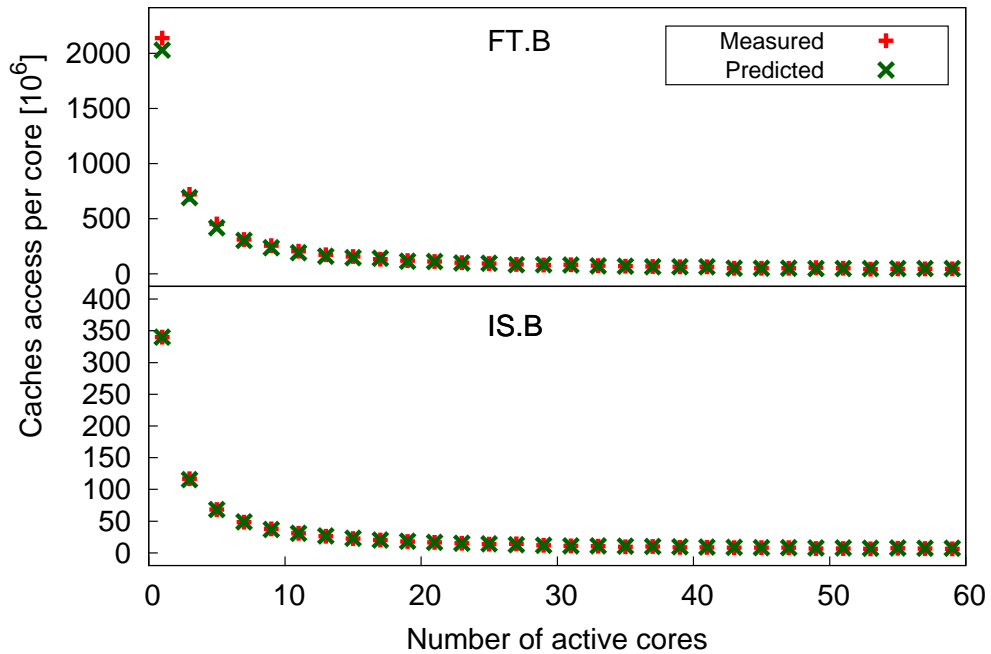


Figure 5.6: Validation of cache accesses

### 5.3.3 Execution time Validation

We use cycles as a handle for execution time. In this section we show the validation results for the cycles per thread for a single iteration of the program with problem size of class B. Extensive validation has been performed for all the programs in Table 5.2 across all possible configurations and the average error between the model and the measured values is shown in Table 5.4. Figure 5.7 and 5.8 show the validation for the total cycles executed by programs BT, LU in scatter mode and programs EP, FT in compact mode respectively.

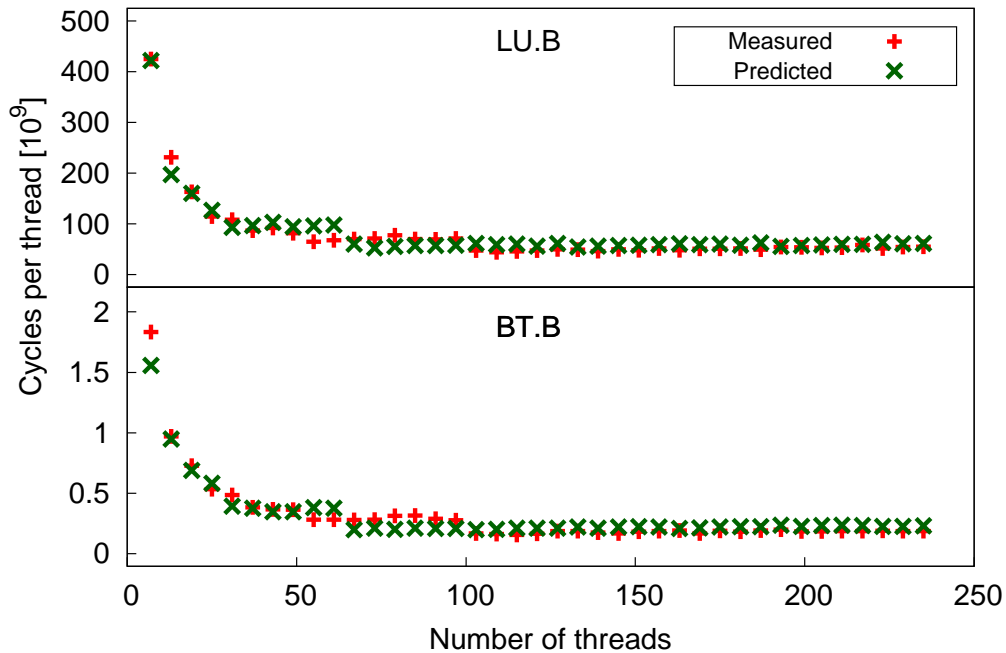


Figure 5.7: Validation results for scatter thread affinity mode

#### Sources of Inaccuracy

We identify three factors that affect the accuracy of the model. Firstly, the most significant source of error comes due to irregularities during different executions of the same program. The measured values of execution time and energy show

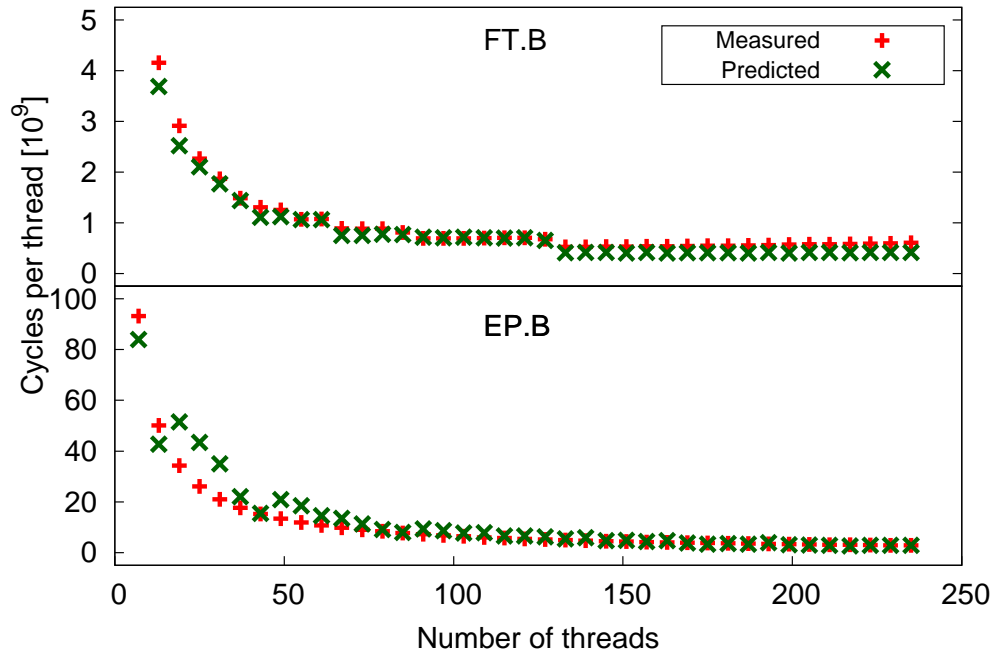


Figure 5.8: Validation results for compact thread affinity mode

Program	Average error [%]	
	Compact	Scatter
BT	7	8
EP	14	4
FT	15	8
IS	4	6
CG	1	6
LU	11	11

Table 5.4: Validation results

irregularities of up to 10% for different runs of the same program. Secondly, there are irregularities in both inter and intra-core contention for shared-memory due to the shared ring architecture as the shared data could be residing in either the neighbouring core or in the core that is half the distance of the ring from the current core. These irregularities vary the service time parameter of the memory and hence result in model errors. Thirdly, the model does not account for thread synchronizations and waiting time due to these barriers, which leads to loss of accuracy.

### 5.3.4 Power Characterization

To determine the power consumed by the cores in the MIC architecture, we use PAPI counters and measure the instantaneous power using micpower component. For each of the programs we measure the micpower:::tot0 [112], a native event of the micpower component and use this value ( $P_{\tau,c}$ ) to compute the energy of the programs executing on the MIC architecture system.

$$E_{\tau,c} = T_{\tau,c} \times P_{\tau,c}$$

## 5.4 Analysis

In this section, we apply our model to study the energy efficiency of different configurations in MIC architecture system under a given service time deadline. We first show how our model can be applied to determine Pareto-optimal system configurations to execute a HPC program on MIC architecture system. Next, we present the performance-to-power ratio (PPR) of the MIC architecture system and determine the impact of PPR on the Pareto-optimal configurations.



### 5.4.1 Pareto-optimal Configurations

Similar to the Pareto frontiers in heterogeneous systems as reported in our earlier work [137], time-energy efficient Pareto-optimal configurations are also exhibited by MIC architecture systems executing HPC parallel programs as shown in Figures 5.9 and 5.10. These Pareto-optimal configurations are energy efficient as they

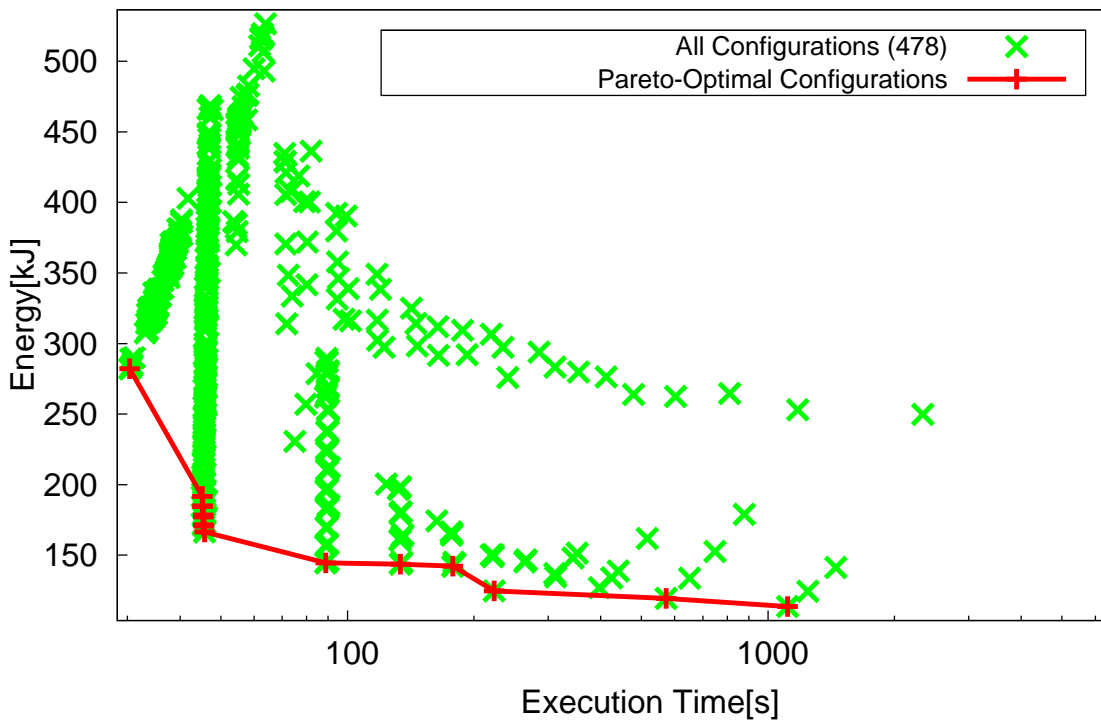


Figure 5.9: Pareto-optimal configurations for executing BT

consume the minimum energy for a given execution time deadline or execute in the minimum possible time for a given energy budget.

Figures 5.9 and 5.10 present two typical plots<sup>2</sup> showing the execution time and energy used to execute a parallel program for all possible configurations, program BT and FT (477 configurations<sup>3</sup>) on a MIC architecture system respectively. Each

<sup>2</sup>The execution time and energy values for all the configurations in these plots are derived using PAPI hardware counters.

<sup>3</sup> $c \in [1, 2, 3, \dots, 60]$ ,  $\tau \in [1..4]$ , mode  $\in$  [scatter, compact],  $60 \times 4 \times 2 = 480$ ;  $c = 1, \tau = 1$ ;  $c = 59$

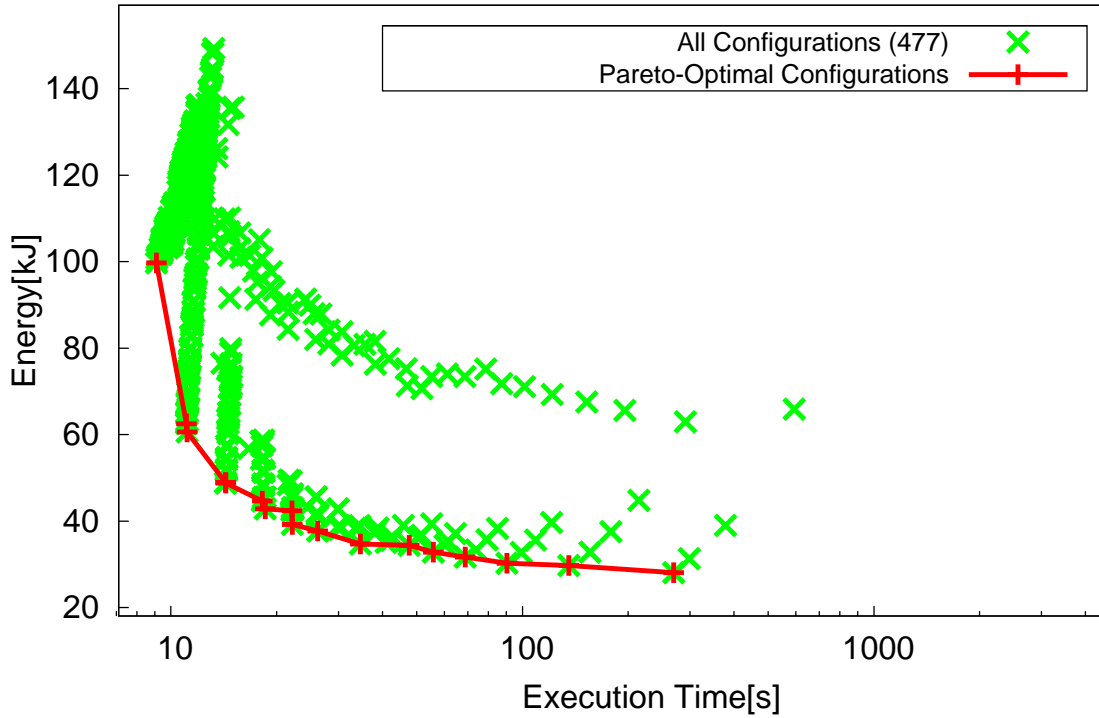


Figure 5.10: Pareto-optimal configurations for executing FT

configuration in these plots is a tuple consisting of the number of cores and the number of threads per core  $(c, \tau)$ , based on the thread affinity mode. For each configuration point, the x-axis denotes the program's execution time and the y-axis represents the corresponding energy used. Given an execution-time deadline, there exist a set of configurations that meet this deadline. The configuration that meets the deadline with the minimum energy usage is *Pareto optimal*. The set of all Pareto optimal points across all possible deadlines forms the time-energy *Pareto frontier*.

Counter-to-intuition as the execution-time deadline is relaxed, the configurations have lesser number of cores and threads but surprisingly use lesser energy. Decreasing the number of cores decreases the average power used but increases

---

with  $\tau = 4$ , &  $c = 1$  with  $\tau = 3$ ; and  $c = 60, \tau = 4$  are the same in both modes, resulting in  $480 - 3 = 477$  configurations.

execution time, and thus it is expected that the energy ( $power \times time$ ) will be constant. Although, decreasing the number of cores causes a linear decrease in the power used, the effect on the execution time is non-linear and is characterized by the inter-core contention for shared resources such as memory ( $T_{inter}$ ).

In the next section, we discuss the performance-to-power ratio metric and show how this metric is useful to determine the existence of Pareto-optimal configurations for a given program executing on accelerators with MIC architecture.

### 5.4.2 Performance-to-power Ratio

Program	Mega-operations (Mops)	PPR [Mops/J]							
		1 core		4 cores		16 cores		60 cores	
		$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 4$	$\tau = 1$	$\tau = 4$
BT	702,200	2.81	6.18	2.68	5.12	2.25	3.79	1.56	1.49
EP	2,147	0.06	0.14	0.06	0.13	0.06	0.11	0.06	0.07
FT	92,049	1.40	4.20	1.36	3.36	1.21	2.15	0.73	0.61
IS	335	0.10	0.20	0.09	0.20	0.08	0.18	0.07	0.08
CG	54,708	0.14	0.60	0.17	0.68	0.15	0.61	0.15	0.45
LU	498,816	1.83	4.20	1.67	3.36	1.39	2.26	1.06	0.83

Table 5.5: Performance-to-power ratio

PPR is defined as the work done per unit of time, normalized by the average power consumption. This is equivalent to the work done per unit of energy and defined as,

$$PPR = \frac{Throughput[operations/s]}{Power[W]}$$

where throughput denotes the number of useful operations performed by the system per unit time. This metric is also used in SPEC benchmark [151]. The floating point operations are used as the useful operations and the computed PPR for millions of floating point operations for all the programs across different number of active cores, for scatter and compact affinity modes with  $\tau = 1$  and  $\tau = 4$  respectively is shown in Table 5.5.

As observed from the table, increasing the number of cores, with  $\tau = 1$  (scat-

ter affinity) decreases the PPR, as the time decreases sub-linearly while power increases linearly thus resulting in an increase in energy, with the exception of the CG program. The execution time is much higher in the CG program compared to the power of the Xeon Phi, and thus increasing the number of cores decreases the execution time by a larger ratio compared to the increase in the power thus reducing the energy and resulting in a better PPR.

For a given number of active cores, increasing the number of threads from one to four, improves the PPR as seen from Table 5.5. This is intuitive because the increase in power consumed by the core increases marginally, while the execution time decreases by at least a factor of two [33, 78]. This improvement in PPR is for smaller number of active cores, because as the number of active cores increase, the sequential fraction becomes the bottleneck, resulting in an increase in energy and thus reduction in PPR, as seen from results in Table 5.5 for  $c = 60$ .

Next, we discuss the conditions under which a coprocessor with MIC architecture exhibits a Pareto-frontier. To assess the impact of PPR, we analyse the Pareto-optimal configurations for the CG program as this program exhibited an improvement in PPR with increase in the number of active cores. Figures 5.11 and 5.12 plot the execution time and energy used across all possible configurations to execute the program CG with only scatter mode and both modes respectively. As observed from the plots, while there are many Pareto-optimal configurations for the CG program, when we consider only the scatter thread affinity mode, there is a single Pareto-optimal configuration. This scatter mode configuration uses 238 total threads implying, 58 cores with four active threads, and two cores with three active threads.

The addition of the compact thread affinity mode exposes many more Pareto-optimal configurations as illustrated in Figure 5.12. These configurations are due to increasing the number of threads per core, with the number of active cores

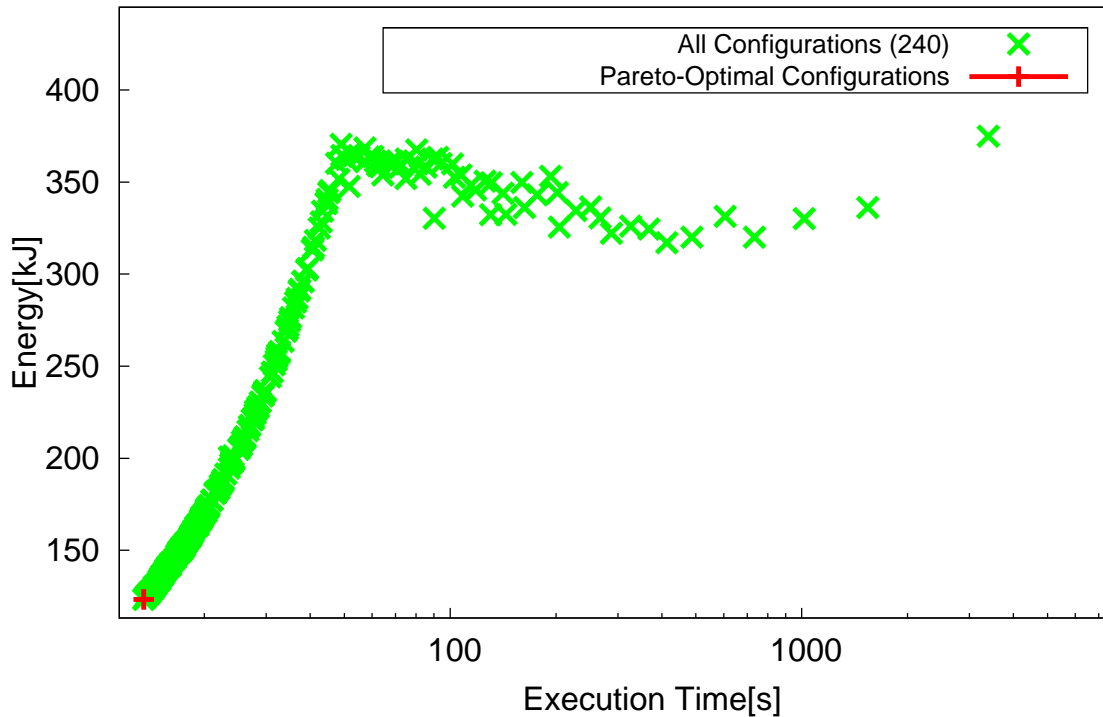


Figure 5.11: Pareto-optimal configurations for executing CG in scatter mode

between 60 cores and four cores, between the Pareto-configurations with the minimum and the maximum execution time. Thus, users of coprocessors with MIC architecture systems executing HPC applications, can apply our approach to determine the thread affinity mode to use, and the number of threads to execute their program with the best possible execution time and/or consuming the minimum possible energy.

**Impact on Energy Savings** The existence of the Pareto-frontier and using Pareto-optimal configurations for executing a program imply two options for saving energy. Firstly, for a given execution time deadline, a Pareto-optimal configuration consumes lesser energy compared to a non-optimal configuration thus resulting in energy savings. For example, to execute the CG program within 17

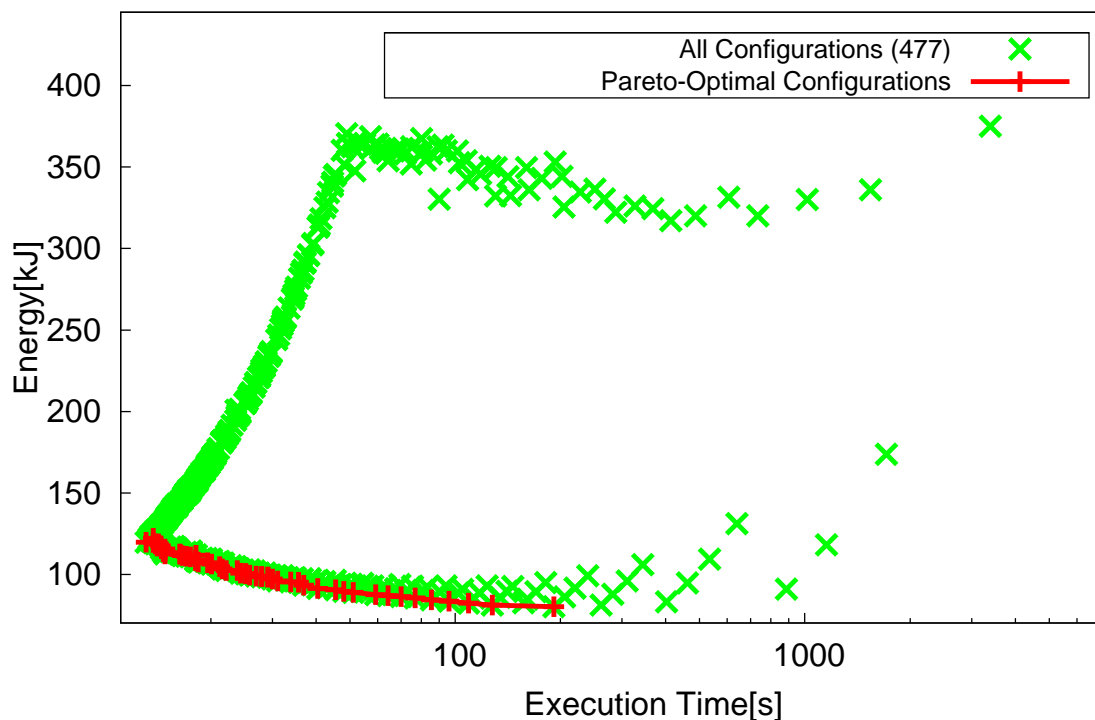


Figure 5.12: Pareto-optimal configurations for executing CG

seconds, a Pareto-optimal configuration with 187 threads<sup>4</sup> using compact mode consumes only 112kJ, while a non-optimal configuration with 192 threads<sup>5</sup> using scatter mode can consume up to 150kJ, thus resulting in energy savings of 25%.

Secondly, the existence of Pareto-frontier results in significant energy savings can be obtained by trading-off execution time by a negligible amount. For example, to execute the BT program within 45 seconds, for a 1% increase in execution time from 45.2 to 45.8 seconds, users can obtain around 14% of energy savings, by reducing energy usage from 192kJ to 166kJ as illustrated in Figure 5.9.

<sup>4</sup>This configuration uses 49 cores with four threads per core and one core with one thread.

<sup>5</sup>This configuration uses 12 cores with four threads per core and 48 cores core with three threads each.

## 5.5 Summary

While intra-node heterogeneous systems such as systems with MIC architecture coprocessors offer accelerated performance, users have to determine the time-energy optimal set of number of cores ( $c$ ) and threads per core ( $\tau$ ) for executing the program in an energy-efficient manner. This chapter presents an approach to determine time-energy Pareto-optimal system configurations ( $c, \tau$ ) for executing a parallel program on intra-node heterogeneous systems such as the MIC architecture using a measurement-driven analytical model.

The proposed extension of the core model addresses the effects of both TLP within and across cores by considering *inter and intra-core resource overlaps, memory contention among threads within a core and contention across multiple cores*. Table 5.6 summarizes the extensions to the core model to determine the time performance of intra-node heterogeneous systems with VPU. Validation of the proposed approach for a range of HPC programs against direct measurement on Intel Xeon Phi coprocessor show an average error of up to 15% between the predicted and measured execution time.

<b>Energy Performance</b>	
$E_{\tau,c}$	$T_{\tau,c} \times P_{\tau,c}$
$PPR$	$\frac{\text{Throughput}[\text{operations/s}]}{\text{Power}[W]}$
<b>Time Performance</b>	
$T_{\tau,c}$	$T_{work} + \max(T_{inter,c}, T_{intra,\tau})$
$T_{work}$	$\frac{\text{cycles}_{work}}{f}$
$T_{inter,c}$	$\frac{m_{inter}}{f} = \frac{\lambda_{inter,c} \times \beta_{c,s}}{f}$
$T_{intra,\tau}$	$\frac{m_{intra}}{f} = \frac{\lambda_{intra,\tau} \times \alpha_{\tau,s}}{f}$

Table 5.6: Summary of model extension for MIC architecture

We show that a Pareto frontier consisting of time-energy Pareto-optimal configurations exist for a parallel program executed on a MIC architecture system

representing intra-node heterogeneity. These configurations either consume minimum energy for a given execution time deadline, or execute in the minimum possible time for a given energy budget. Hence, HPC users can easily apply our approach for time-energy efficient execution. To further understand the Pareto frontier, we use the performance-to-power ratio metric (PPR), that quantifies the amount of useful computations performed per unit energy used in an execution. Furthermore, we show the use-case of our approach to determine energy-efficient system configurations to execute programs on accelerators such as Xeon Phi and save energy.



# Chapter 6

## Extension to Hybrid Programs with Communication

While inter-node heterogeneity is at the system level, programs are increasingly becoming heterogeneous to efficiently utilize system resources. The impact of heterogeneity in programs is observed by the wide-adoption of hybrid (OpenMP+MPI) programs by HPC application developers [27, 105, 148]. This chapter presents an approach to determine time and energy efficient system configurations for executing a hybrid program using a measurement-driven analytical model. The scalability of the core analytical model proposed in Chapter 3 is shown by applying it to hybrid programs.

A hybrid parallel program is partitioned into a variable number of logical parallel processes and parallel threads. For a given hybrid program and a multi-node system with multi-core nodes operating at different core clock frequencies, there is a large system configuration space for executing these logical processes and threads. As the resource demands in a hybrid parallel program varies with its problem size, these resource demands have to be mapped onto different system configurations to minimize resource contention and runtime overheads. Thus, we

extend the core model with communication parameters that infer both inter and intra-node communication overheads.

Given a hybrid program, the proposed approach determines energy-efficient Pareto-optimal configurations in terms of the number of nodes, number of cores per node and core clock frequency. These configurations<sup>1</sup> either consume minimum energy for a given execution time deadline<sup>2</sup>, or execute in the minimum possible time for a given energy budget. Thus, the approach provides a systematic method to set the number of logical processes and threads for efficient execution of a hybrid program. Secondly, to quantify the degrees of resource contention and communication overhead in an execution, we introduce the Useful Computation Ratio (UCR) metric. We also discuss how UCR and Pareto-optimal configurations can be used in conjunction by system designers to gain further insights into resource imbalances and how application developers can fine-tune their hybrid program.

## 6.1 Overview

An abstract view of a typical hybrid parallel program can be represented as iterations of alternating computation and communication phases. While the computation phase is further split into parallel threads performing computations using shared-memory data, the communication phase consists of logical processes on different nodes using MPI over the network. We assume that these phases have negligible resource demands from storage devices such as disks. The code in Figure 6.1 shows an abstraction of a hybrid parallel program [132] with annotations

---

<sup>1</sup>For example, if we consider ten nodes ( $n$ ), with each node having eight cores ( $c$ ) and assuming the possible core clock frequencies for each node is  $f \in [0.8, 1.4, 2.1]$  GHz, this results in  $10 \times 8 \times 3 = 240$  configurations. This configuration space grows linearly with  $n, c$  or the number of possible operating core clock frequencies.

<sup>2</sup>While most HPC applications do not have strict deadlines, their execution times are constrained due to sharing of cluster resources. Also, with the advent of pay-per-use models, an execution time deadline translates to a cost budget.

illustrating the contention for shared-memory and network as inferred by our approach.

```

for(iteration = 1..S)
{
  # pragma omp parallel //  $\tau$  threads on  $c$  cores
  {
    /* computations or useful work */
    ....
    ....
    ....
    /* intra-node (shared-memory contention) */
  }
  /* inter-node (network contention) */
  MPI_Send //  $\ell$  logical processes on  $n$  nodes
  MPI_Recv
}

```

Figure 6.1: Abstraction of a hybrid program

A typical hybrid parallel program is split into  $\ell$  logical processes with  $\tau$  parallel threads per process as shown in Figure 6.2. Systems executing hybrid programs have a peak power budget that limits the number of nodes to a maximum of  $n_{max}$ . Each node can be configured to use a maximum number of  $c_{max}$  cores, with each core operating at a clock frequency  $f \in [f_{min}, f_{max}]$ . Hence, the different combinations of  $n$ ,  $c$ , and  $f$  values within these bounds form the total number of system configurations for executing the program.

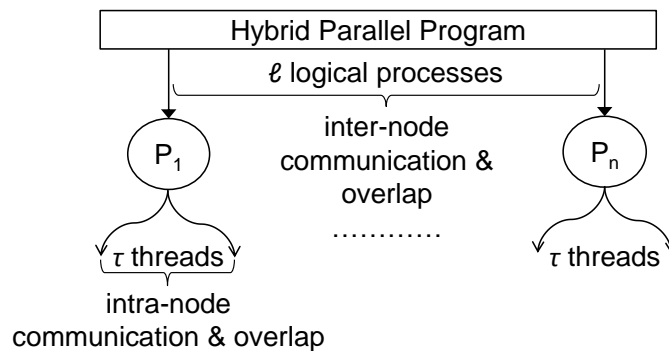


Figure 6.2: Model of hybrid program execution

Users of hybrid programs often face the challenge of determining the optimal number of  $\ell$  and  $\tau$  for energy efficient execution. Using values of  $\ell < n$  incurs energy wastage due to idling resources. On the contrary, it may be beneficial to have  $\tau < c_{max}$  because of the energy saved by reducing the waiting time due to shared-memory contention among the  $\tau$  threads. Hence, choosing an energy efficient number of  $\ell (= n)$  and  $\tau (= c)$  to execute a hybrid program is not obvious and requires an approach that models the execution time considering inter and intra-node (i) overlap, (ii) communication, and (iii) contention for shared resources. Therefore, as outlined in Figure 6.3, given a hybrid parallel program, our approach uses a measurement-driven analytical model to determine Pareto-optimal system configurations, i.e.  $(n, c, f)$ , such that these configurations consume minimum energy for a given execution time deadline and/or execute in the minimum time for a given energy budget.

To infer the program's demands on system resources such as CPU and memory, we characterize the workload using baseline executions to derive program and architectural artefacts. These baseline executions are performed on a single node

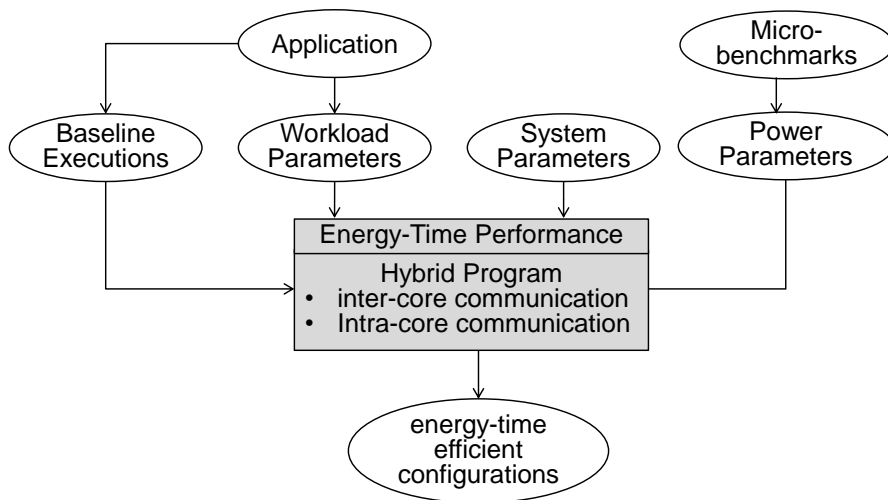


Figure 6.3: Approach for hybrid programs

across all possible  $c$  and  $f$  values using a small program input size. To derive the energy consumed by the program, we use micro-benchmarks to measure the power characteristics of the processor system.

While Chapter 3 models the overlap between inter-node computation phases, it considers data center workloads that have minimal inter-node communication. However, hybrid programs not only exhibit considerable amount of inter-node communication among  $\ell$  processes but also intra-node communication among  $\tau$  threads. Hence modeling the overlaps, communication and contention for shared resources for a hybrid program is non-trivial and challenging.

To determine the inter-node communication, we use the volume of communication per node ( $\nu$ ) and characterize the CPU overheads in communication using the utilization of the cores ( $U_s$ ). This models the overlap between computation and inter-node communication via the network. Next, we use queueing theory to model contention among the nodes for network and compute the waiting time ( $T_{w,net}$ ) at each node with the network switch as a server servicing communication requests. To model the overlap between computation and intra-node communication via shared-memory, we infer the waiting time of the cores due to contention for memory using stall cycles ( $m$ ). The notations used in the communication model are described in Table 6.1 and the notations not listed in this table are from the core model in Chapter 3, and are described in Table 3.1.

To show that our approach is independent of a programming language, we have chosen benchmark applications in both C++ and Fortran. The five hybrid programs are, Block Tri-diagonal solver (BT), Lower-Upper Gauss-Seidel solver (LU), Scalar Penta-diagonal solver (SP) [161], Car-Parrinello Molecular Dynamics (CP) [67] and a lattice Boltzmann method (LB) [77]. To predict the energy usage for a configuration, it is assumed that the hybrid program is the only application being executed, apart from background operating system tasks.

To demonstrate the application of our approach on different system architectures, we validated it on two systems with diverse time-energy performance, ARM Cortex-A9 processor based low-power cluster and Intel Xeon *x86\_64* architecture based processor cluster. The systems used for validation have nodes with a single NIC and cores access shared-memory via Uniform Memory Architecture (UMA). Nodes communicate via an Ethernet-based switch.

## 6.2 Communication Model

Symbol	Description
<b>Workload Parameters</b>	
$\eta$	no. of messages sent/received by $\mathcal{P}$
$\nu$	volume (in bytes) per message
<b>System Parameters</b>	
$B$	communication throughput
<b>Time Model</b>	
$T_{w,net}$	waiting time due to network contention
$T_{s,net}$	non-overlapped network service time

Table 6.1: Communication model parameters

### 6.2.1 Model Inputs

The measurement driven inputs to our analytical model are obtained from workload, network and power characterization.

#### Workload Characterization

To derive workload dependent architectural artefacts, we use baseline executions of the program on a single node. To determine the overlap among useful computation cycles, data-accesses from shared-memory and network, we measure the translation of a given hybrid program into useful work cycles ( $w_s$ ). To model the

non-overlapped intra-node contention, we measure the stall cycles due to memory accesses ( $m_s$ ). These measurements are recorded for a single node across the possible values of  $c$  and  $f$  using hardware performance counters. Hence, these measurements are non-intrusive with respect to the execution of the application. Program dependent communication characteristics, such as number of communication calls ( $\eta$ ) and communication volume per message call ( $\nu$ ) are measured using the lightweight profiling tool mpiP [163]. It suffices to perform baseline executions only on a single node, as workload characteristics from these measured values can be inferred from  $\ell$  and  $\tau$ .

### Network characterization

To measure communication overheads of MPI over TCP for a given link bandwidth, we use NetPIPE [159]. Figure 6.4 shows that the maximum achievable

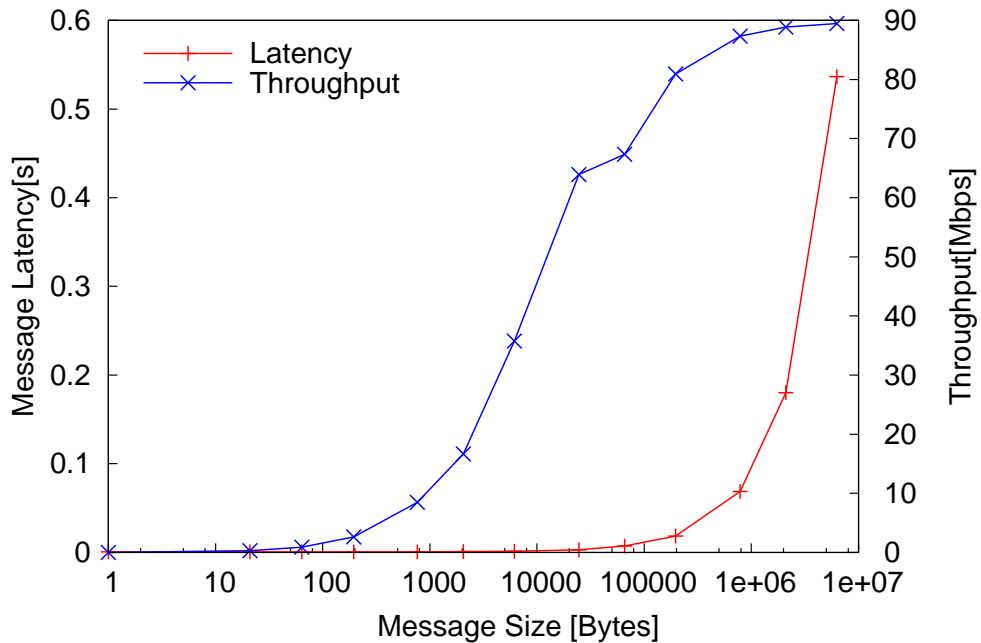


Figure 6.4: Network characterization

throughput on a 100 Mbps Ethernet link is only 90 Mbps due to MPI overheads

and the operating system. This characterization of the network link latency and bandwidth is used to compute network service time.

### Power Characterization

With energy proportionality becoming increasingly important, processors exhibit a wide dynamic energy range [171]. Hence, during program execution, cores have a wide-ranging power consumption depending on the amount of computations being executed. We classify the power states of a core into active power and stall power corresponding to computation cycles and stall cycles respectively. The idle power of the core is accounted by the total system idle power ( $P_{sys, idle}$ ). We developed benchmarks that stress the processor pipeline to measure active and stall CPU power. These measurements are done for the complete range of cores ( $c$ ) and frequencies ( $f$ ) supported by the system to characterize the processor across its dynamic power range. We derive  $P_{mem}$  from JEDEC memory specifications and directly measure  $P_{net}$ .

### 6.2.2 Time Model

In this section, we derive the execution time,  $T$ , for a hybrid program  $\mathcal{P}$  with  $S$  iterations, executing on  $n$  homogeneous nodes, each having  $c$  cores. Inter and intra-node overlap between the computation and communication phases in the program is accounted as useful work cycles ( $T_{CPU}$ ). Non-overlapped execution time including both inter and intra-node data dependencies is modeled as waiting and service time for communication over the network ( $T_{w, net}, T_{s, net}$ ) and within shared-memory ( $T_{w, mem}, T_{s, mem}$ ) respectively. Hence, the total execution time is summed up as the overlapped time for useful work cycles and the non-overlapped



queueing delays:

$$T = T_{CPU} + T_{w,net} + T_{s,net} + T_{w,mem} + T_{s,mem} \quad (6.1)$$

To derive  $T_{CPU}$ , we use the total work cycles incurred by the hybrid program. The total work cycles for a given program is split equally among number of processing cores across all nodes, operating at a clock frequency  $f$  as:

$$T_{CPU} = \frac{cycles_{core}}{n \cdot c \cdot f} \quad (6.2)$$

Overlap of computations and shared-memory data accesses is accounted by using the work cycles spent for computing ( $w$ ) and stall cycles that are not due to memory contention ( $b$ ). These non-memory stalls are due to the complex out-of-order pipeline architectures that are prevalent in most processors today [113]. Hence, the total useful cycles considering overlaps are:

$$cycles_{core} = w + b \quad (6.3)$$

As non-memory stalls vary based on the application, they are measured using the baseline execution of a program, but they scale well with program input size for a given processor architecture [137]. Hence, they are derived easily for scale-out programs, by using the measured values of the subset program  $\mathcal{P}_s$ :

$$w(c, f) = \frac{w_s \cdot S}{S_s} \quad b(c, f) = \frac{b_s \cdot S}{S_s} \quad (6.4)$$

While parallelism increases speedup by enabling overlap, overlap causes increased contention for shared resources. For hybrid parallel programs, the logical processes across nodes contend for access to the network and the parallel threads within a

logical process contend for shared-memory. Network contention causes messages to wait in the operating system's network socket buffer before being serviced by the network. As a result, CPU idles while waiting for data from the network. This is modeled using a M/G/1 queue with a mean waiting time [114, 142]:

$$T_{w,net} = \frac{\lambda \cdot \hat{y}^2}{1 - \rho} \quad (6.5)$$

where  $\hat{y}$  and  $\rho$  are the service time and utilization of the network respectively, and  $\lambda$  is the inter-arrival rate of messages to the buffer.

Communication characteristics of the program affect inter-node communication ( $T_{s,net}$ ). For hybrid parallel programs, the communication characteristics are determined using the number of messages transmitted ( $\eta$ ) and the volume of communication per message ( $\nu$ ). In most modern processing systems, the CPU time incurred for processing, overlaps with transfer time of the messages over the network. Thus, the service time of inter-node communication is:

$$T_{s,net} = \max \left( ((1 - U) \cdot T_{CPU}), \left( \frac{\eta \cdot \nu}{B} \right) \right) \quad (6.6)$$

In hybrid programs, the parallel threads within a logical process contend for shared-memory. Shared-memory contention is derived from the waiting time and service time of memory requests queueing up for service at the memory controller. Queueing delay due to contention for memory causes stall cycles in the processor. Therefore, these stall cycles due to non-overlapped memory accesses ( $m$ ) are used to model shared-memory contention overheads:

$$T_{w,mem} + T_{s,mem} = \frac{m}{f} \quad m(c, f) = \frac{m_s \cdot S}{S_s} \quad (6.7)$$

### 6.2.3 Energy Model

Total energy for a given hybrid program on a cluster of  $n$  nodes is the sum of the energies consumed per node. Energy consumed by a node is divided among three active components, processing unit (CPU), memory resources and network card. Energy consumed by other system components such as power regulators, storage, video, etc. are considered under the  $E_{idle}$ . Hence, the total energy consumed by the system during execution is:

$$E = (E_{CPU} + E_{mem} + E_{net} + E_{idle}) \times n \quad (6.8)$$

Energy consumed by the active cores in a node is:

$$E_{CPU} = ((P_{core,act} \cdot T_{CPU}) + (P_{core,stall} \cdot (T_{w,mem} + T_{s,mem}))) \cdot c \quad (6.9)$$

Energy consumed by the memory and network for each node is:

$$E_{mem} = P_{mem} \cdot (T_{w,mem} + T_{s,mem}) \quad (6.10)$$

$$E_{net} = P_{net} \cdot (T_{w,net} + T_{s,net}) \quad (6.11)$$

When the system is completely idle, the power consumption includes the idle power of the cores, memory and I/O devices, as well as the fixed power consumption for the rest of the components. Thus,

$$E_{idle} = P_{sys,idle} \cdot T \quad (6.12)$$

## 6.2.4 Validation

Here, we first describe the hybrid programs, systems and setup used for validation followed by validation results. The proposed approach is validated against direct measurements for both execution time and energy.

### Workloads and Setup

While our approach is applicable on generic hybrid parallel programs, we selected a representative subset of five benchmark programs for presentation in this chapter. This subset was chosen to represent a wide range of HPC domain applications that exert different inter and intra-node communication resource demands and use different programming languages. We use three hybrid programs from NASA Parallel Benchmark (NPB) suite [161]. These solve discretized version of Navier-Stokes equations in three dimensions, and are (i) Lower-Upper Symmetric Gauss-Seidel (LU), (ii) Scalar Penta-diagonal (SP), and (iii) Block Tri-diagonal (BT). The fourth program uses the Car-Parinello (CP) method to simulate  $H_2O$  molecules from the Quantum Espresso suite [67]. While the above programs are in Fortran, we chose the fifth program in C++, to illustrate that our approach is independent of the programming language. This is an open source Lattice Boltzmann (LB) code [1], that simulates fluid flows in a three-dimensional lid-driven cavity.

To illustrate the generalization of our approach, we validate on two diverse processor system clusters as detailed in Table 6.2. The increase in the computing capabilities of mobile-based smart devices, have caught the attention of leading server providers to design their next-generation systems based on such mobile-based processors [17, 143, 152]. Hence, other than the traditional server system based on two Intel Xeon CPUs, we also choose a low-power ARM Cortex-A9-based system for validating the proposed approach. The Xeon and ARM systems not

System	Intel Xeon E5-2603	ARM Cortex-A9
ISA	x86_64	ARMv7-A
Nodes	8	8
Cores/node	8	4
Clock Frequency	1.2–1.8 GHz	0.2–1.4 GHz
L1 data cache	32kB / core	32kB / core
L2 cache	2MB / node	1MB / node
L3 cache	20MB / node	NA
Memory	8GB DDR3	1GB LP-DDR2
I/O bandwidth	1Gbps	100Mbps

Table 6.2: Hybrid program system

only have a diverse performance-to-power ratio but also have different ISAs and differ in orders of magnitude in their cache, memory and network bandwidths. These large differences among the resource capabilities of the selected systems illustrate that our approach can be applied to a generic processor system and the approach independent of any specific ISA. We validate our model against direct measurements of both execution time and energy usage of each cluster with the setup shown in Figure 3.2. The system time command is used to measure execution time and a WattsUp meter [9] measures both power and energy.

To increase the credibility of our approach, we performed extensive validation for each of the five benchmarks on a large number of Xeon and ARM system configurations. These configurations arise from varying the (i) number of nodes,  $n_{xeon/arm} \in [1, 2, 4, 8]$ , (ii) number of active cores per node,  $c_{xeon} \in [1 \cdots 8]$  and  $c_{arm} \in [1 \cdots 4]$ , and (iii) operating core clock frequency,  $f_{xeon} \in [1.2, 1.5, 1.8]$  GHz and  $f_{arm} \in [0.2, 0.5, 0.8, 1.1, 1.4]$  GHz. Thus, the number of configurations used for validation was 96 and 80 for Xeon and ARM clusters respectively. Table 6.3 summarizes the average error and the standard deviation from the measured values for all of these configurations. The predicted values of time and energy using our approach follow the trends of the measured values across hardware configurations as shown in Figures 6.5 and 6.6 respectively. Due to paucity of space, for each

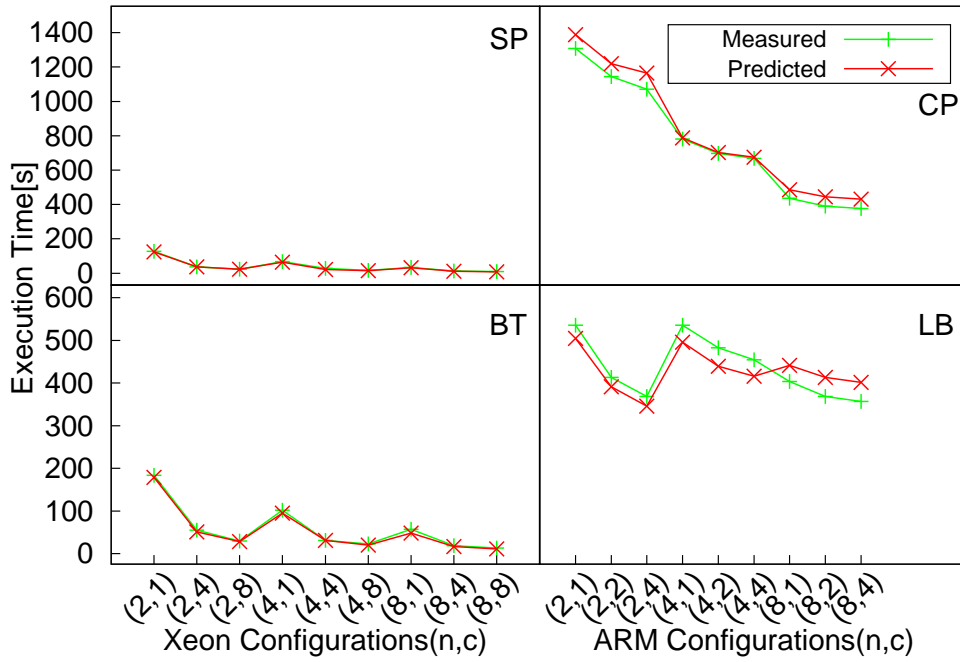


Figure 6.5: Execution time validation

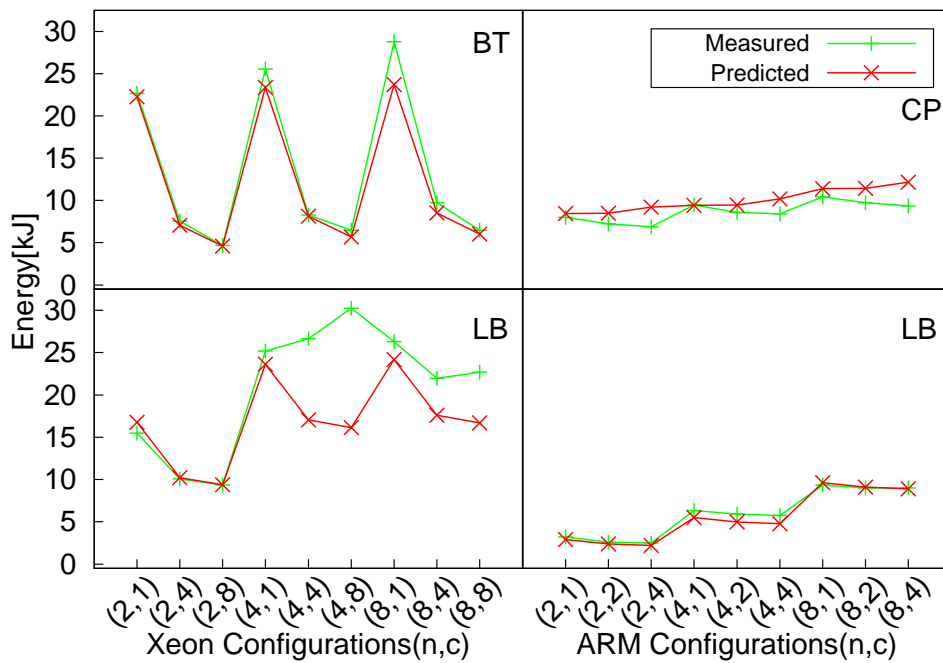


Figure 6.6: Energy validation

cluster we plot the execution time and energy for programs with the worst-case error.

Domain	Benchmark Suite	Program	Execution Time error [%]				Energy error [%]			
			Xeon		Cortex-A9		Xeon		Cortex-A9	
			Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
3D Navier-Stokes Equation Solver	NAS Multi-zone Parallel Benchmark (NPB3.3-MZ)	LU	4	5	3	2	5	8	6	6
		SP	6	9	4	3	2	10	4	5
		BT	8	7	4	6	8	7	5	6
Electronic-structure Calculations	Quantum Espresso (v5.1)	CP	1	10	5	12	1	14	7	12
Computational Fluid Dynamics	OpenLB (olb-0.8r0)	LB	6	8	4	8	15	12	7	9

Table 6.3: Hybrid program validation results

Many research studies show a correlation between communication patterns exhibited by a program with scale-out input sizes [106, 164]. We show the application of our approach for scale-out HPC programs by plotting the validation results for LU program with input size of class C (four times larger than the baseline measurement program size) in Figure 6.7. Thus, we show the application of our model

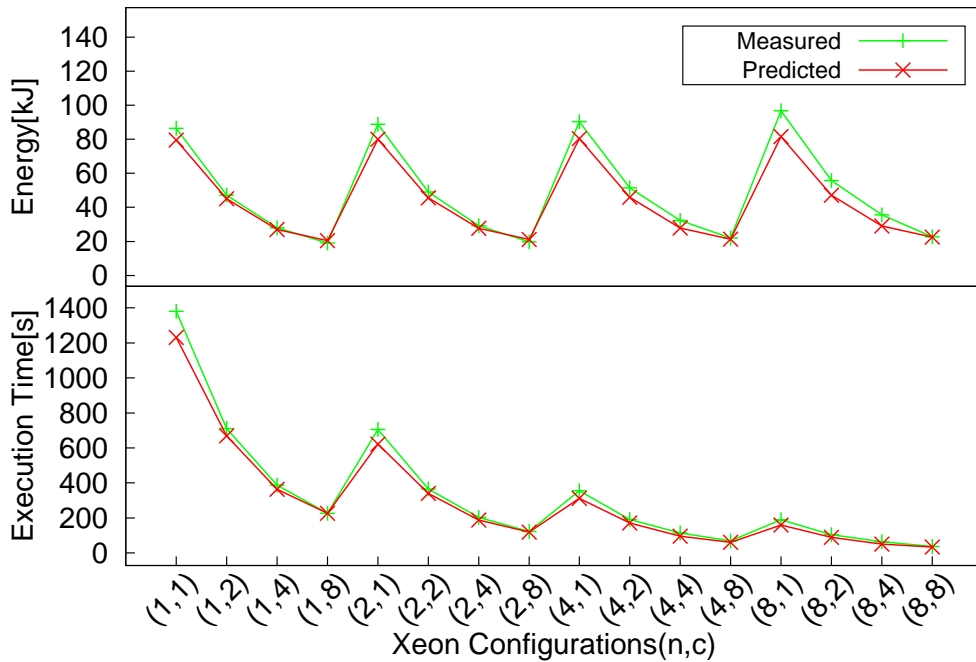


Figure 6.7: Scale-out program LU

to programs whose communication characteristics scale linearly with respect to program input size.

### Sources of Inaccuracy

We identify three factors that affect the accuracy of the model. Firstly, the most significant source of error comes due to irregularities during different executions of the same program from the operating system overheads. The measured values of execution time and energy show irregularities of up to 10% for different runs of the same program. Secondly, there are irregularities in the communication overheads due to explicit synchronizations in the program among logical processes and threads. For example, LB program incurs more instructions on higher number of nodes at higher number of cores, due to the synchronization among the logical processes and threads. This significantly increases the energy used, but does not reduce the execution time. This increase causes our model to underestimate the energy used by Xeon configurations (4,4) and (4,8) as shown in Figure 6.6. The third reason for model inaccuracy is the accuracy of the characterized power parameters. In particular, the system power values for active cycles, stall cycles and idleness differ by up to  $0.4W$  for the ARM node and  $2W$  for the Xeon node. This variability translates into a larger underestimation of the energy consumed especially for larger execution times.

## 6.3 Analysis

This section discusses the application of our approach to determine time-energy Pareto-optimal configurations for efficient execution of hybrid parallel programs. Next we discuss the application of the Useful Computation Ratio (UCR) metric to further optimize the Pareto frontier.



### 6.3.1 Pareto-optimal Configurations

Similar to the Pareto frontiers in heterogeneous systems as discussed in Chapter 4, time-energy efficient Pareto-optimal configurations are also present in homogeneous systems executing hybrid parallel programs as shown in Figures 6.8 and 6.9. These Pareto-optimal configurations are energy efficient as they consume the minimum energy for a given execution time deadline or execute in the minimum possible time for a given energy budget. Figures 6.8 and 6.9 present two

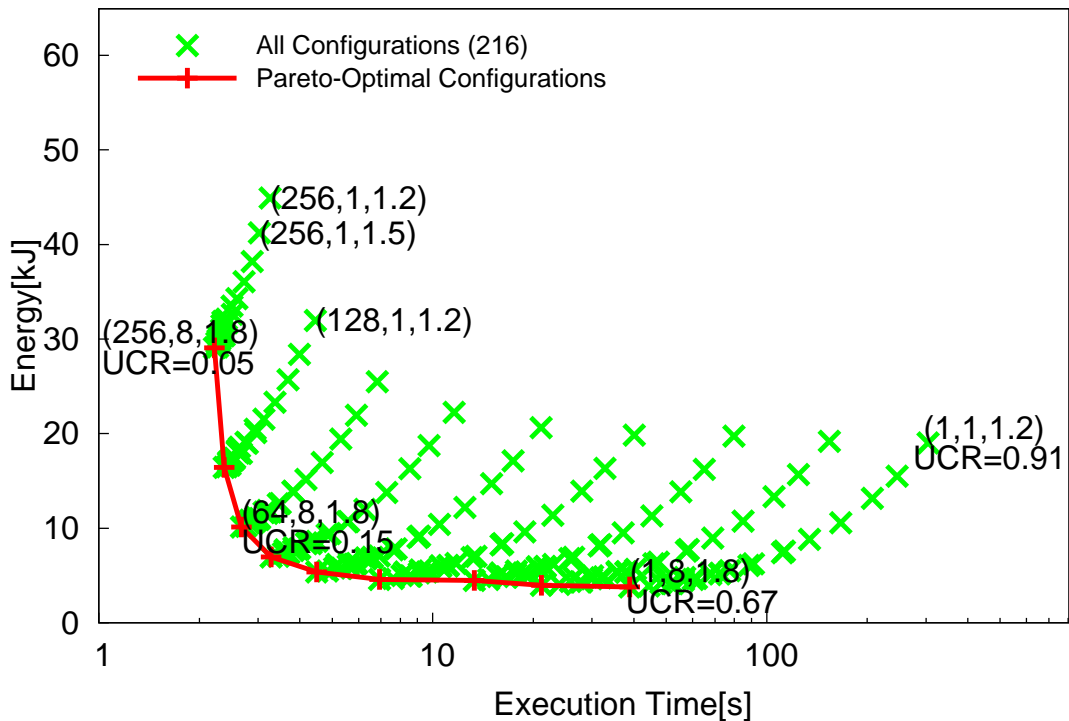


Figure 6.8: Xeon cluster executing SP program

typical plots<sup>3</sup> showing the execution time and energy used to execute a hybrid program for all possible configurations, program SP (216 configurations<sup>4</sup>) on a Xeon cluster and program CP (400 configurations<sup>5</sup>) on an ARM cluster respectively.

<sup>3</sup>The configurations in these plots for  $n \in [1..8]$  for both A9 and Xeon clusters have been validated as shown in Section 6.2.4.

<sup>4</sup> $n \in [1, 2, 4, 8, \dots, 256], c \in [1..8]$  and  $f \in [1.2, 1.5, 1.8]GHz$

<sup>5</sup> $n \in [1, 2, 3, \dots, 20], c \in [1..4]$  and  $f \in [0.2, 0.5, 0.8, 1.1, 1.4]GHz$

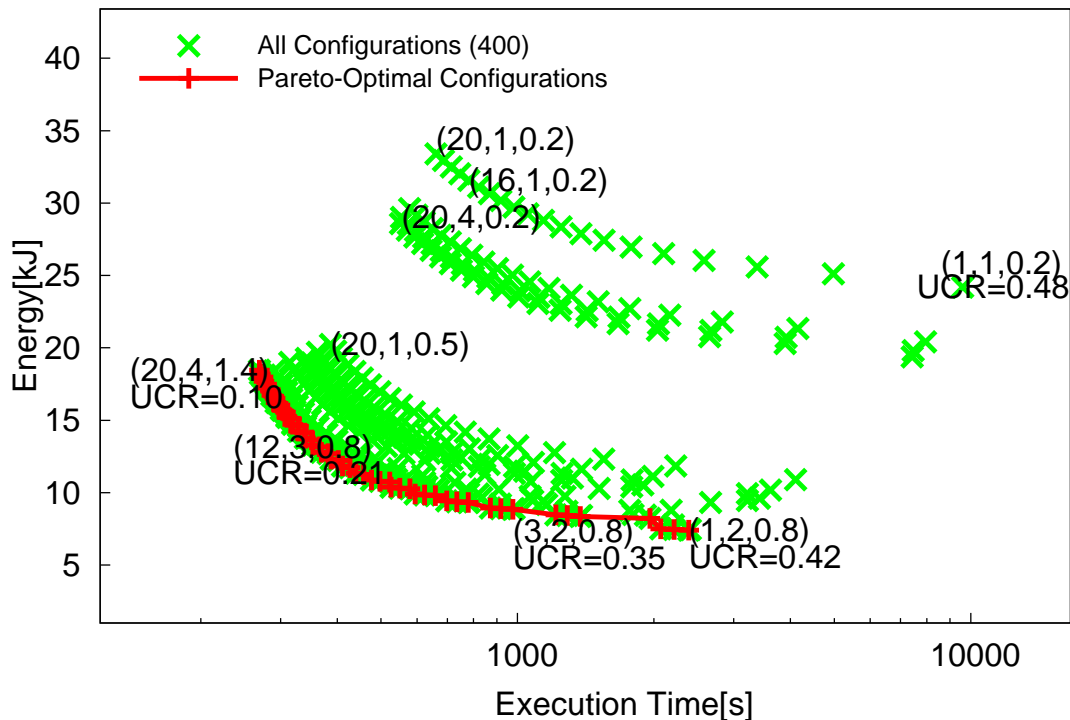


Figure 6.9: ARM cluster executing CP program

Each configuration in these plots is a tuple consisting of the number of nodes, number of cores and the core clock frequency  $(n, c, f)$ . For each configuration point, the x-axis denotes the program's execution time and the y-axis represents the corresponding energy used. Given an execution-time deadline, there exist a set of configurations that meet this deadline. The configuration that meets the deadline with the minimum energy usage is *Pareto optimal*. The set of all Pareto optimal points across all possible deadlines forms the time-energy *Pareto frontier*.

These time-energy plots illustrate three counter-intuitive insights. Firstly, as the execution-time deadline is relaxed, the configurations have lesser number of nodes but surprisingly use lesser energy. Decreasing the number of nodes decreases power used but increases execution time, and thus it is expected that the energy ( $power \times time$ ) will be constant. Although, decreasing the number of nodes causes a linear decrease in the power used, the effect on the execution time is non-linear

and is characterized by the queuing delays due to network contention ( $T_{w,net}$ ,  $T_{s,net}$ ). Secondly, as the energy budget is reduced, counter-to-intuition, the number of cores and core clock frequency increases. Increasing the number of cores or core clock frequency reduces execution time but increases power. Although the increase in the power is a factor of processor design, the decrease in execution time is not linear and is characterized by shared-memory contention ( $T_{w,mem}$ ,  $T_{s,mem}$ ). Thirdly, Pareto-optimal configurations do not necessarily use all available cores operating at the maximum frequency, e.g. ARM system configuration (3,2,0.8) is on the Pareto frontier of the CP program.

### **Impact on Energy Savings**

The existence of the Pareto-frontier and using Pareto-optimal configurations for executing a program imply two options for saving energy. Firstly, for a given execution time deadline, a Pareto-optimal configuration consumes lesser energy compared to a non-optimal configuration thus resulting in energy savings. For example, as shown in Figure 6.8, for executing the SP program on the Xeon cluster within three seconds, a non-optimal configuration may consume energy up to 41kJ while a Pareto-optimal configuration consumes only 10kJ, resulting in energy savings of four times or 75% reduction in energy.

Secondly, the existence of Pareto-frontier results in significant energy savings can be obtained by trading-off execution time by a negligible amount. For example, with an execution time deadline of three seconds, increasing the execution time of the SP program on a Xeon cluster from 2.2 to 2.7 seconds results in energy savings of 65% as the energy consumed reduces from 29kJ to 10kJ as shown in Figure 6.8.

### 6.3.2 Useful Computation Ratio

While Computation-to-Communication Ratio (CCR) is a widely used metric to quantify the communication costs (both inter and intra-node) of a parallel program and a higher CCR implies better efficiency, this metric is not normalized and hence is less useful for making comparisons across configurations. As HPC applications become increasingly data-centric and with the widening gap between floating-point speed and memory bandwidth [51], it is very important to characterize the performance of a program with respect to an upper bound to compare and evaluate its execution across a large system configuration space. To address this, we propose a new metric called the Useful Computation Ratio (UCR) of a hybrid program as:

$$UCR = \frac{T_{useful}(= T_{CPU})}{T} \quad (6.13)$$

Total execution time,  $T$ , for a hybrid program is defined as:

$$T = T_{CPU} + T_{data\_dep} + T_{mem\_contention} + T_{net\_contention} \quad (6.14)$$

Since  $T_{CPU}$  is defined as the time spent by the program in the system for useful computations including overlapped data accesses (Equation 6.2), the maximum value for normalized UCR is one.  $T_{data\_dep}$  is a program characteristic, and does not change for a given program with a fixed input size executing on a specific system architecture. As computations need data to be fetched,  $T_{mem\_contention}$  represents the communication cost of fetching data from shared-memory within a node and  $T_{net\_contention}$  accounts for the inter-node communication cost. Thus, UCR is a useful measure for comparing the execution efficiencies of a hybrid program across different system configurations. Figures 6.10 and 6.11 plot UCR and the time-energy performance of five hybrid programs for different configurations.

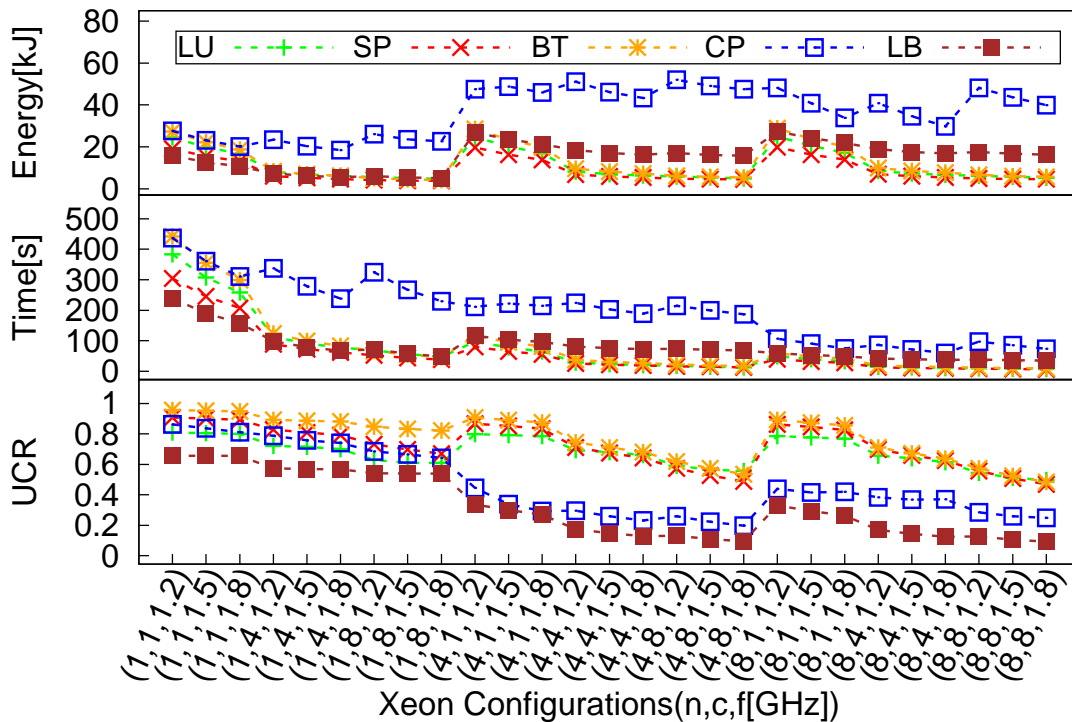


Figure 6.10: UCR and time-energy performance on Xeon cluster

For a given program and an input size, the upper bound of UCR is obtained for an execution configuration with a single node, single core and the lowest operating frequency,  $(1, 1, f_{min})$ , as this configuration incurs negligible communication overheads. The differences between the CISC and RISC ISA of the processors causes UCR for Xeon to be much higher (0.96 for BT program) than UCR for ARM (0.54 for BT program).

UCR not only exhibits resource mismatches between computational processing and communication resources of a system but also expresses mismatches in the program implementation due to an imbalance in the parallelism among logical processes versus parallel threads. The CP and LB programs illustrate this imbalance, as seen from the steep drop in the UCR values (Figures 6.10 and 6.11) with increasing number of logical processes and threads. Increasing the number of nodes, or cores or core clock frequencies, increases contention for shared resource

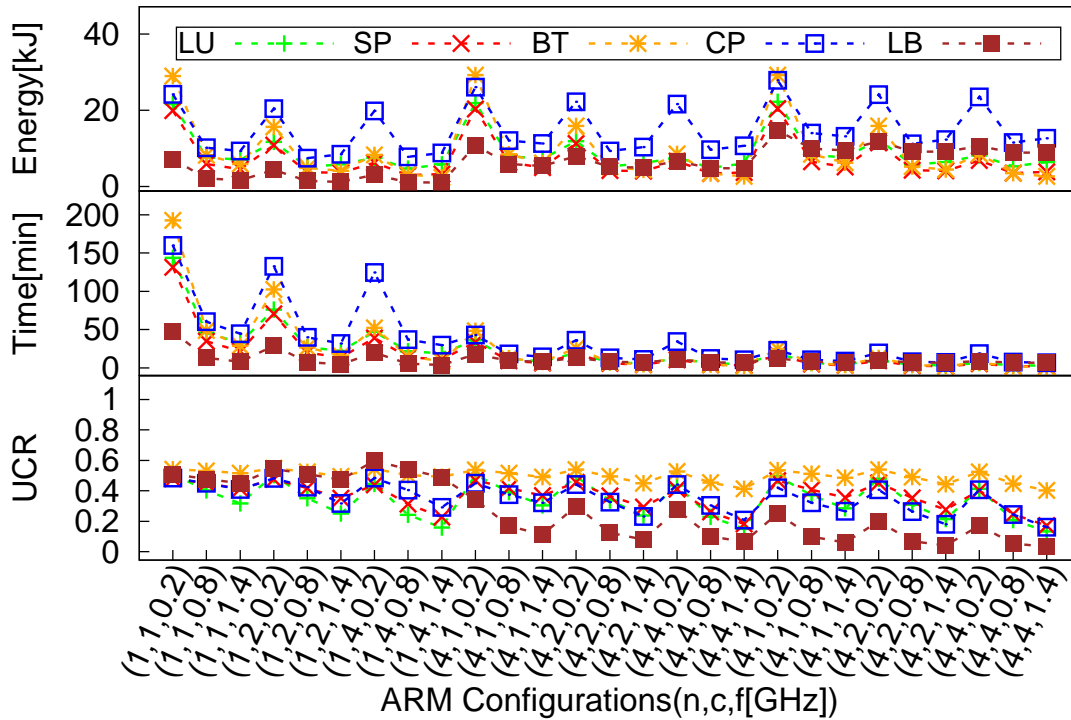


Figure 6.11: UCR and time-energy performance on ARM cluster

and thus decreases UCR. However, increasing the number of nodes, or cores or core clock frequencies, reduces both the execution time and energy used for certain configurations. Hence, while the UCR metric gives useful information regarding the balance between the computation and communication resources in a system for a program, it cannot be used to determine efficient execution configurations, as configurations with high UCR are not necessarily energy-efficient.

Figures 6.8 and 6.9 also show the UCR values for some Pareto-optimal configurations. An increase in the execution time results in lesser number of nodes in the Pareto-optimal configurations. This decrease in the number of nodes, reduces contention thus increasing the UCR as observed. While the Pareto-optimal configurations are energy efficient, they do not necessarily imply a high UCR. As is observed from Figures 6.8 and 6.9, the UCR values of the Pareto-optimal configurations (0.05 to 0.67 for Xeon and 0.10 to 0.42 for ARM) is quite small compared

to the best possible UCR (0.91 for Xeon and 0.48 for ARM). Hence there is room for further optimizing the Pareto configurations for better balance between the computation and communication phases of the hybrid parallel program executing on a given system architecture.

### **Optimizing UCR for Pareto-optimal configurations**

UCR represents the balance between the execution rates and the communication rates of resources in a system and hence can be improved by either changing the program or system design to achieve a better matching between these rates. For example, doubling the memory bandwidth reduces the number of stall cycles due to shared-memory contention by two times, and thus improves the UCR of SP program executed on Xeon configuration (1,8,1.8) from 0.67 to 0.81. This increase in UCR also reduces the execution time by 7 seconds and energy used by 590 Joules, thus further optimizing the Pareto-frontier configuration. Hence, the proposed approach can be easily applied by system architects to gain insights into resource imbalances, and further optimize the Pareto-frontier using UCR.

Secondly, for a given system configuration, application developers can fine-tune their implementations by re-structuring the iterations during the computational and communication phases of the program for different  $l (= n)$  and  $\tau (= c)$  to further improve the UCR of Pareto-optimal configurations. The proposed approach thus offers a holistic hardware-software co-design by gaining useful insights from the predicted execution time, energy and UCR of hybrid parallel programs.

## **6.4 Summary**

While hybrid parallel programs offer the dual-advantage of scalability via distributed-memory and better performance using shared-memory, users of these programs

face an uphill task in determining the time-energy optimal set of logical processes ( $\ell$ ) and threads ( $\tau$ ) for executing the program. From a system's perspective, this challenge translates to determining energy efficient execution configurations in terms of  $(n, c, f)$ , where  $n$  is the number of nodes,  $c$  the number of cores and  $f$  is the operating core clock frequency. This chapter presents an approach to determine time-energy Pareto-optimal system configurations  $(n, c, f)$  for executing a hybrid program using a measurement-driven analytical model.

The proposed model addresses the effects of using both distributed-memory and shared-memory communication by considering inter and intra-node resource overlaps, memory contention among cores within a node and network contention across multiple nodes. Table 6.4 summarizes the extensions to the core model to determine the energy-time performance of hybrid programs. We validate the proposed approach for a range of HPC programs from different domains such as non-linear partial differential equation solvers, electronic structure calculations and computational simulation for fluid dynamics. These representative HPC applications are validated against direct measurement on Intel Xeon and ARM Cortex-A9 clusters as they have a diverse time-energy performance. Validation results show a mean error of less than 15% between the predicted and measured execution time and energy.

We show that a Pareto frontier consisting of time-energy Pareto-optimal configurations exist for a hybrid program executed on a homogeneous cluster. These configurations either consume minimum energy for a given execution time deadline, or execute in the minimum possible time for a given energy budget. Hence, users of hybrid programs can easily apply our approach for time-energy efficient execution. To further optimize the Pareto frontier, we introduce a new metric, useful computation ratio (UCR) that quantifies the degree of resource contentions and communication overheads in an execution. We also show how system archi-



<b>Energy Performance</b>	
$E$	$(E_{CPU} + E_{mem} + E_{net} + E_{idle}) \times n$
$E_{CPU}$	$((P_{core,act} \cdot T_{CPU}) + (P_{core,stall} \cdot (T_{w,mem} + T_{s,mem}))) \cdot c$
$E_{mem}$	$P_{mem} \cdot (T_{w,mem} + T_{s,mem})$
$E_{net}$	$P_{net} \cdot (T_{w,net} + T_{s,net})$
$E_{idle}$	$P_{sys,idle} \cdot T$
<b>Time Performance</b>	
$T$	$T_{CPU} + T_{w,net} + T_{s,net} + T_{w,mem} + T_{s,mem}$
$T_{CPU}$	$\frac{cyclescore}{n \cdot c \cdot f}$
$T_{w,net}$	$\frac{\lambda \cdot \hat{y}^2}{1 - \rho}$
$T_{s,net}$	$\max\left(\left((1 - U) \cdot T_{CPU}\right), \left(\frac{\eta \cdot \nu}{B}\right)\right)$
$T_{w,mem} + T_{s,mem}$	$\frac{m}{f}$

Table 6.4: Summary of model extension for hybrid programs

tects and application developers can increase the UCR of Pareto-optimal configurations by balancing resource service demands with resource utilization, to further minimize system inefficiencies.



# Chapter 7

## Conclusion

### 7.1 Thesis Summary

With heterogeneity becoming ubiquitous due to the paradigm shift from high-performance to low-power designs in server systems, maturity of multi-core clusters, and wide-adoption of compute node accelerators among others, new opportunities arise for an energy-time efficient matching of workload service demands and resource capabilities. However, this opportunity introduces many challenges with respect to energy-time efficient execution of parallel programs on the large configuration space made available due to heterogeneity in systems and programs. In addition to inter-node heterogeneity, we address intra-node heterogeneous systems with VPU accelerators and hybrid programs. A heterogeneous mix of nodes offers a large system configuration space due to the different combinations of system parameters such as, types of nodes, number of nodes of each type, number of active cores per node and the operating core clock frequency. While the configuration space due to inter-node heterogeneity grows exponentially with respect to the different types of nodes, the configuration space due to intra-node heterogeneity and hybrid programs grows linearly with the number of cores, nodes and the thread

affinity modes.

In this thesis, we address some of the challenges due to this large configuration space and propose a measurement-driven analytical modeling approach to determine energy-time efficient configurations. The proposed approach addresses the effects of using both distributed-memory and shared-memory communication by considering inter and intra-node resource overlaps, memory contention among cores within a node and network contention across multiple nodes.

The two major contributions of this thesis are: (i) an approach to determine energy-time efficient configurations addressing the large configuration space and (ii) novel insights from the energy-time performance of inter-node heterogeneous systems, intra-node heterogeneous systems with VPU and hybrid programs. These contributions are detailed below.

### **7.1.1 Measurement-based Analytical Model**

We propose a measurement-based analytical model to determine time and energy efficient system configurations for executing a parallel program on a baseline heterogeneous cluster consisting of multi-core brawny and wimpy nodes. We show the scalability of the core model by extending it to determine time and energy-time efficient system configurations for parallel programs executing on co-processors with VPUs, e.g. Intel Xeon Phi. While extension of the core model to intra-node heterogeneous systems shows the scalability of the model from a system perspective, from a program perspective, the scalability of the core model is illustrated by modeling the communication for hybrid OpenMP + MPI programs. As outlined in Fig. 7.1, the proposed approach determines the set of energy-time efficient configurations for a given system. Table 7.1 summarizes the energy and time performance derived from the proposed analytical model. The details of the experiment setup used for validating the model and the description of the model parameters are in

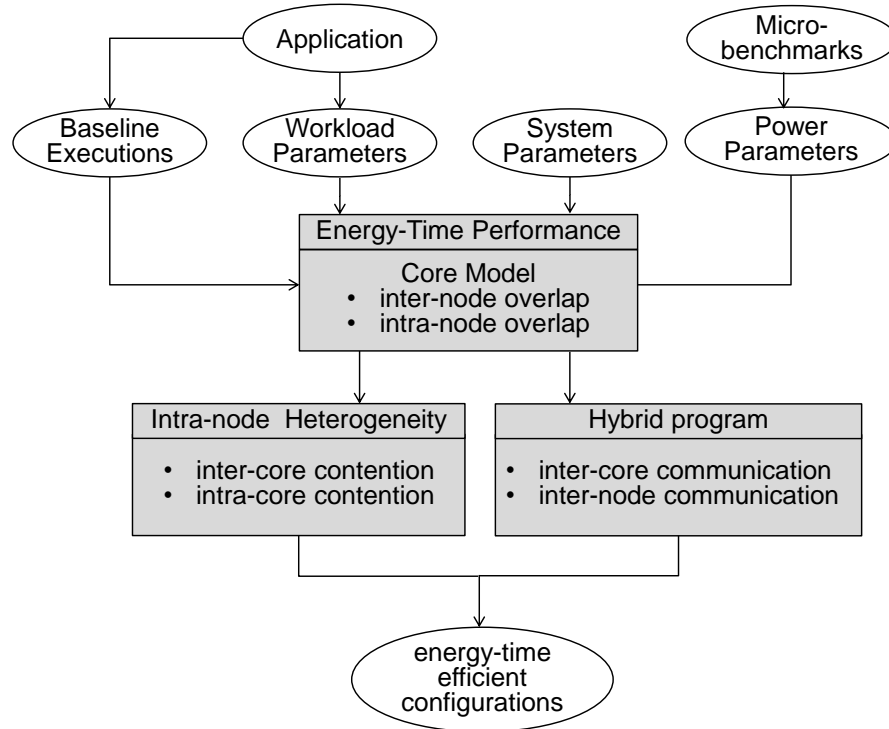


Figure 7.1: Measurement-based analytical model

Appendix A and B respectively.

### Modeling Overlap among Heterogeneous Nodes [137]

A heterogeneous mix of nodes offers a large system configuration space due to the different combinations of system parameters such as, types of nodes, number of nodes of each type, number of active cores per node and the operating core clock frequency. While this large configuration space offers an opportunity to obtain a better match between the application performance and the system, the key challenge is to determine the set of energy-time efficient configurations. Thus, this thesis presents an approach to address this challenge using a measurement-driven analytical model that determines both time and energy performance of heterogeneous computing systems. In contrast to pure analytical or mathematical models, the proposed approach relies on baseline measurements to gain accuracy

<b>Core Model (Inter-node Heterogeneity)</b>	
<b>Energy</b>	$E = \sum_{i=1}^d E_i$ $E_i = (E_{i,CPU} + E_{i,mem} + E_{i,I/O} + E_{i,idle}) \cdot n_i$ $E_{i,CPU} = (P_{i,CPU,act} \cdot T_{i,act}) + (P_{i,CPU,stall} \cdot T_{i,stall})$ $E_{i,mem} = P_{i,mem} \cdot T_{i,mem}$ $E_{i,I/O} = T_{i,I/O} \cdot P_{i,I/O}$ $E_{i,idle} = T_i \cdot P_{i,idle}$
<b>Time</b>	$T = \max_{i=1}^{d_{max}} (T_i)$ $T_i = \max(T_{i,CPU}, T_{i,I/O})$ $T_{i,CPU} = \max(T_{i,core}, T_{i,mem})$ $T_{i,core} = \frac{cycles_{i,core}}{f_i}$ $T_{i,mem} = \frac{cycles_{i,mem}}{f_i}$ $T_{i,I/O} = \frac{\max(T_{i,I/O_T}, \frac{1}{\lambda_{I/O}})}{n_i}$
<b>Intra-node Heterogeneity</b>	
<b>Energy</b>	$E_{\tau,c} = T_{\tau,c} \times P_{\tau,c}$ $PPR = \frac{Throughput[operations/s]}{Power[W]}$
<b>Time</b>	$T_{\tau,c} = T_{work} + \max(T_{inter,c}, T_{intra,\tau})$ $T_{work} = \frac{cycles_{work}}{f}$ $T_{inter,c} = \frac{m_{inter}}{f} = \frac{\lambda_{inter,c} \times \beta_{c,s}}{f}$ $T_{intra,\tau} = \frac{m_{intra}}{f} = \frac{\lambda_{intra,\tau} \times \alpha_{\tau,s}}{f}$
<b>Hybrid Programs</b>	
<b>Energy</b>	$E = (E_{CPU} + E_{mem} + E_{net} + E_{idle}) \times n$ $E_{CPU} = ((P_{core,act} \cdot T_{CPU}) + (P_{core,stall} \cdot (T_{w,mem} + T_{s,mem}))) \cdot c$ $E_{mem} = P_{mem} \cdot (T_{w,mem} + T_{s,mem})$ $E_{net} = P_{net} \cdot (T_{w,net} + T_{s,net})$ $E_{idle} = P_{sys,idle} \cdot T$
<b>Time</b>	$T_{hybrid} = T_{CPU} + T_{w,net} + T_{s,net} + T_{w,mem} + T_{s,mem}$ $T_{CPU} = \frac{cycles_{core}}{n \cdot c \cdot f}$ $T_{w,net} = \frac{\lambda \cdot \hat{y}^2}{1-\rho}$ $T_{s,net} = \max(((1-U) \cdot T_{CPU}), (\frac{n \cdot \nu}{B}))$ $T_{w,mem} + T_{s,mem} = \frac{m}{f}$

and uses a core analytical model that determines the energy required to execute a parallel program. The core model is applicable on heterogeneous clusters with wimpy and brawny nodes having different Instruction Set Architectures (ISAs). For each type of node, the core model predicts the execution time and energy usage of a parallel task considering the overlap among the response times of service requests to the CPU, the memory and the network I/O devices. As heterogeneous clusters have different execution rate, we propose a *matching* technique, that splits the workload such that all the different nodes complete the parallel job at the same time. By finishing at the same time, the energy incurred by idling in the cluster is minimized.

### **Modeling Overlap and Contentions in VPU**

The Many Integrated Core (MIC) architecture enhances traditional host CPU performance by providing multiple cores that can accelerate vector processing and are also termed Vector Processing Units (VPUs).

While the core model proposed in Chapter 3 addresses inter-node heterogeneity, to address intra-node heterogeneity, we extend the core model to determine energy-time efficient configurations for executing parallel programs on the Intel Xeon Phi coprocessor. Apart from determining the optimal number of nodes, cores and operating core clock frequency, Xeon Phi additionally offers the challenge of determining the optimal thread affinity mode. Thread affinity modes restrict the execution of OpenMP threads to a subset of the available physical processing cores and thus have a significant impact on both time and energy performance of the program execution on the co-processor.

The Xeon Phi offers two main thread affinity modes, namely scatter and compact. The scatter mode allocates a single thread per physical core till all the cores are used up before allocating another thread on the same core. In contrast, the

compact mode allocates the threads to the same physical core, till the core executes the maximum possible number of threads before allocating another core to a thread. These differences in the thread allocation policies offer some challenges in modeling the intra-core thread contention for shared resources like L1 cache and inter-core contention for shared memory. We extend the core model with these additional resource contentions and determine time and energy-time efficient system configurations to execute a parallel application on a intra-node heterogeneous system such as Xeon Phi.

### **Modeling Communication of Hybrid Programs [136]**

While the proposed core model considers both intra-node overlap between useful work and memory accesses, it does not consider communication overheads among nodes. A hybrid programming model has a dual-communication impact, from communication among logical processes (inter-node), and communication among threads (intra-node). We extend the core model by considering both distributed-memory and shared-memory communication and modeling their effects on execution time and energy consumed by a hybrid parallel program. Given a hybrid program, the extended core model determines time and energy-time efficient system configurations in terms of the number of nodes, number of cores per node and core clock frequency. Thus, the proposed approach provides a systematic method to set the number of logical processes and threads for efficient execution of a hybrid program.



## 7.1.2 Insights from Energy-Time Performance Analysis

### Inter-node Heterogeneous Systems [137]

While the proposed approach relies on a measurement-driven analytical model to determine energy-time efficient system configurations, the impact of the performance-to-power ratio (PPR) of the individual nodes on these system configurations is non-obvious. The key insights from energy-time performance analysis of inter-node heterogeneous systems may be summarized as:

1. sweet-spot configurations: Heterogeneity introduces “sweet-spots”, representing energy-time efficient system configurations, and set of sweet-spots form the energy-deadline Pareto frontier.
2. energy-deadline Pareto-frontiers: The Pareto frontier can be analytically determined using the node PPRs, thus addressing the challenge of the large configuration space due to heterogeneity. The Pareto-frontier is further optimized by replacing low PPR nodes with higher PPR nodes using our power substitution ratio.
3. energy proportionality wall: Inter-node heterogeneous clusters provide an avenue to scale the energy proportionality wall by exposing sub-linear energy-proportional configurations.

The existence of Pareto-optimal configurations for executing a program implies energy savings. For example, to execute the EP program within a given time deadline, Pareto-optimal configuration result in energy savings up to 75% compared to non-optimal configurations.

### **Intra-node Heterogeneous System with VPU**

The key insights from energy-time performance analysis of intra-node heterogeneous system with VPU may be summarized as:

1. show that parallel programs executing on coprocessors with VPU exhibit Pareto-optimal configurations, that execute in the minimum possible time for a given energy budget or consume the minimum energy for a given execution time deadline.
2. show how the performance-to-power ratio (PPR) metric can be used to determine what programs benefit from offloading on VPUs.

Similar to the energy savings obtained due to Pareto-optimal configurations of inter-node heterogeneous systems, intra-node heterogeneity with VPU also exhibit energy savings. Firstly, for a given execution time deadline a Pareto-optimal configuration reduces energy by up to 25% as compared to a non-optimal configuration. Secondly, for configurations on the Pareto-frontier, energy savings of 15% can be obtained at the expense of a 1% increase in execution time.

### **Hybrid Programs [136]**

While the proposed model addresses the effects of using both distributed-memory and shared-memory communication, it does not address the challenge of determining program hot-spots or system resource bottlenecks. To aid application developers to gain insights on program hot-spots, and system designers to identify capacity bottlenecks, we further analyze the energy-time performance of hybrid programs and thus optimize software-hardware co-design to improve energy-efficiency. Similar to the Pareto frontiers of inter-node and intra-node heterogeneous systems, we show that time-energy efficient Pareto-optimal configurations are also present in homogeneous systems executing hybrid parallel programs.

As HPC applications become increasingly data-centric and with the widening gap between floating-point speed and memory bandwidth, it is very important to characterize the performance of a hybrid program with respect to an upper bound to compare and evaluate its execution across a large system configuration space. To quantify the degrees of resource contention and communication overhead in an execution, we introduce a novel metric, namely Useful Computation Ratio (UCR). In addition, we illustrate how UCR and Pareto-optimal configurations can be used in conjunction by system designers to gain further insights into resource imbalances and how application developers can fine-tune their hybrid program. Similar to energy savings due to inter and intra-node heterogeneity, hybrid programs also save energy using Pareto-optimal configurations. Firstly, for a given execution time deadline a Pareto-optimal configuration reduces energy by up to 75% as compared to a non-optimal configuration. Secondly, Pareto-optimal configurations reduce energy by 65% at the expense of 18% increase in execution time<sup>1</sup>.

### 7.1.3 Limitations

While the analysis of inter-node heterogeneous systems gives many useful insights on energy savings obtained by varying the degree of heterogeneity, it assumes that the workload can be divided among different node types and assumes that a program can be divided into finer chunks which may not be possible for the sequential fraction parts of the parallel workload. Appendix C discusses the impact of factoring such parameters into the model.

While the programs used in the modeling and analysis scale linearly with an increase in problem size, some parallel applications might have high overheads due to synchronization (e.g., barrier, locks) at both node and cluster levels which are not considered in the model.

---

<sup>1</sup>These values are derived from the plots in Section 6.3.1

## 7.2 Future Directions

### 7.2.1 Dynamic Adaptation of Configurations at Run-time

For a given parallel application, the approach proposed in this thesis uses measurement based parameters derived from baseline executions and workload characterization. These measured parameters are used as inputs to analytically model the execution time and energy usage of a parallel application executing on heterogeneous computing systems. Such an approach has the dual advantage of improving the accuracy of the time-energy prediction and is non-intrusive to the actual application's performance, but is a static mapping of the application to a system configuration. While such a static approach does achieve desired energy-time performance gains and minimizes inefficiencies, at run-time available system resources may fluctuate widely. Thus, combining this approach with a dynamic configuration selection during the execution of a program may result in more energy savings.

### 7.2.2 Cost-Time Performance

Cloud computing is becoming ubiquitous and a compelling business model for all, from start-ups to large established corporations. The reason for the widespread adoption of cloud computing can be attributed to two key characteristics offered by this computing model, namely elastic resources and pay-per-use pricing. While elastic resources offer dynamic on-demand scaling of computational resources, the key challenge lies in matching applications elastic resource demands across elastic resources to achieve optimal cost-performance. These characteristics also introduce manifold challenges to cloud consumers to optimize the execution of their applications for a given cost budget.

The energy-time performance models and analysis presented in this thesis can

be adapted to explore two main directions: (i) cost-time performance models for heterogeneous compute resources in the cloud (system) and (ii) cost-time performance models for elastic applications using elastic resources such as cloud (program).

### **System**

A research direction by adapting this thesis is modeling the time and cost-efficient performance of heterogeneous computing systems in the cloud. Cloud resources are inherently heterogeneous as they need to cater to a widespread array of consumers with different computational needs. Additionally, cloud service offerings have multiple pricing models bringing about another dimension to the resource configuration space. These variegated pricing models is a key challenge in mapping an application's resource demands to the available elastic cloud resources and needs to be addressed further. The approaches proposed in this thesis could be used as an inception to explore cost and time efficient performance analysis of heterogeneous resources in the cloud.

### **Program**

Traditionally, the cloud computing model has attracted enterprise software applications that need on-demand scalability, but currently even scientific applications and machine-learning applications are moving from self-hosted systems to the cloud to meet their computational demands. Conventionally algorithms are designed to solve a given problem with a “definite” and “precise” output. However, many classes of applications such as scientific computations, image processing, data mining, and pattern recognition do not need precise answers, and loss of accuracy is acceptable. While users of such applications can accept imprecise results or results of different quality, elastic algorithms are necessary to exploit the

full potential of elastic cloud resources. The models and analysis presented in this thesis can be adapted to investigate the hypothesis: an elastic algorithm optimizes the cost and quality-of-result (QoR) on elastic cloud resources.

# References

- [1] OpenLB: <http://www.openlb.net>, [online].
- [2] DDR3 Specification, <http://www.webcitation.org/6JN7G4r3x>, 2010.
- [3] Cisco Catalyst 2960-S and 2960 Series Switches, <http://www.webcitation.org/6JPSUNMj0>, 2011.
- [4] Heterogeneous System Architecture Standard, <http://hsafoundation.com/hello-hsa-foundation>, 2012.
- [5] Memcached 1.4.15, <http://www.webcitation.org/6JLdcKxsc>, 2012.
- [6] Google finishes 2048-bit RSA migration, Yahoo to encrypt all data early next year, <http://www.webcitation.org/6LaPMkEj0>, 2013.
- [7] Julius, <http://julius.sourceforge.jp/>, 2013.
- [8] Top 500 supercomputer sites, online, 2015.
- [9] WattsUpMeters: <https://www.wattsupmeters.com/secure/index.php>, [online], May. 2014.
- [10] Z. Abbasi, G. Varsamopoulos and S. K. Gupta, TACOMA: Server and Workload Management in Internet Data Centers considering Cooling-computing Power Trade-off and Energy Proportionality, *Transactions on Architecture and Code Optimization (TACO)*, 9(2):11, 2012.

- 
- [11] D. Abts, M. R. Marty, P. M. Wells, P. Klausler and H. Liu, Energy Proportional Datacenter Networks, *Proc. of 37th Annual International Symposium on Computer Architecture*, pages 338–347, 2010.
- [12] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin et al, HPC-Toolkit: Tools for Performance Analysis of Optimized Parallel Programs, *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.
- [13] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl et al, Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage, *Proc. of NSDI*, pages 365–380, 2009.
- [14] A. Aggarwal and A. Chandra, Communication Complexity of PRAMs, T. Lepist and A. Salomaa, editors, *Automata, Languages and Programming*, volume 317, pages 1–17. 1988.
- [15] A. Aggarwal, A. K. Chandra and M. Snir, On Communication Latency in PRAM Computations, *Proc. of 1st annual ACM symposium on Parallel algorithms and architectures*, pages 11–21, 1989.
- [16] A. Alexandrov, M. F. Ionescu, K. E. Schauser and C. Scheiman, LogGP: Incorporating Long Messages into the LogP Model - One Step Closer Towards a Realistic Model for Parallel Computation, *Proc. of the 7th Symposium on Parallel Algorithms and Architectures*, pages 95–105, New York, NY, USA, 1995.
- [17] AMD, AMD to Accelerate the ARM Server Ecosystem with the First ARM-based CPU and development Platform from a Server Processor Vendor: <http://www.amd.com/en-us/press-releases/Pages/amd-to-accelerate-2014jan28.aspx>, [online], Jan. 2014.



- [18] G. M. Amdahl, Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, *Proc. of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485, 1967.
- [19] V. Anagnostopoulou, S. Biswas, H. Saadeldeen, A. Savage, R. Bianchini et al, Barely alive memory servers: Keeping data active in a low-power state, *J. Emerg. Technol. Comput. Syst.*, 8(4):31:1–31:20, Nov. 2012.
- [20] V. Anagnostopoulou, S. Biswas, A. Savage, R. Bianchini, T. Yang et al, Energy Conservation in Datacenters through Cluster Memory Management and Barely-alive Memory Servers, *Proc. of Workshop on Energy Efficient Design*, 2009.
- [21] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan et al, FAWN: A Fast Array of Wimpy Nodes, *Proc. of 22nd SOSP*, pages 1–14, 2009.
- [22] M. Annavaram, E. Grochowski and J. Shen, Mitigating Amdahl’s Law Through EPI Throttling, *Proceedings of the 32Nd Annual International Symposium on Computer Architecture*, ISCA ’05, pages 298–309, Washington, DC, USA, 2005. IEEE Computer Society.
- [23] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands et al, The Landscape of Parallel Computing Research: A View from Berkeley, Technical report, EECS Department, University of California, Berkeley, 2006.
- [24] N. Auluck, S. Betha and B. Mangipudi, Contention Aware Energy Efficient Scheduling on Heterogeneous Multiprocessors, *IEEE Transactions on Parallel and Distributed Systems*, 2014.

- 
- [25] T. Austin, E. Larson and D. Ernst, SimpleScalar: an Infrastructure for Computer System Modeling, *Computer*, 35(2):59–67, Feb 2002.
- [26] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo et al, The NAS Parallel Benchmarks 2.0, Technical Report NAS-95-020, NASA Ames Research Center, 1995.
- [27] P. Balaji, D. Buntinas, D. Goodell, W. Gropp and R. Thakur, Fine-grained Multithreading Support for Hybrid Threaded MPI Programming, *International Journal of High Performance Computing Applications*, 24(1):49–57, 2010.
- [28] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski et al, A Regression-based Approach to Scalability Prediction, *Proc. of 22nd ICS*, pages 368–377, 2008.
- [29] L. A. Barroso, J. Clidaras and U. Hlzlé, The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition, *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 2013.
- [30] L. A. Barroso and U. Hölzle, The Case for Energy-Proportional Computing, *IEEE Computer*, 40, 2007.
- [31] C. Bienia, S. Kumar, J. P. Singh and K. Li, The PARSEC Benchmark Suite: Characterization and Architectural Implications, *Proc. of PACT*, pages 72–81, 2008.
- [32] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi et al, The M5 Simulator: Modeling Networked Systems, *Micro, IEEE*, 26(4):52–60, July 2006.
- [33] S. Borkar, Thousand core chips: a technology perspective, *Proceedings of the 44th annual Design Automation Conference*, pages 746–749, 2007.

- [34] J. M. Bull, A Hierarchical Classification of Overheads in Parallel Programs, *Proc. of 1st International Workshop on Software Engineering for Parallel and Distributed Systems*, pages 208–219, 1996.
- [35] J. Burge, P. Ranganathan and J. Wiener, Cost-aware Scheduling for Heterogeneous Enterprise Machines (CASH’EM), *Proc. of Cluster Computing*, pages 481–487, 2007.
- [36] T. E. Carlson, W. Heirman and L. Eeckhout, Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-core Simulation, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 52, 2011.
- [37] L. C. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell Jr and L. P. Davis, How Well can Simple Metrics Represent the Performance of HPC Applications?, *Proc. of Supercomputing*, pages 48–48, 2005.
- [38] C. Castillo, G. N. Rouskas and K. Harfoush, Efficient QoS Resource Management for Heterogeneous Grids, *22nd. IEEE International Parallel and Distributed Processing Symposium (IPDPS08)*. Citeseer, 2008.
- [39] K. Chakraborty and S. Roy, Topologically Homogeneous Power-performance Heterogeneous Multicore Systems, *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, 2011.
- [40] J.-J. Chen, K. Huang and L. Thiele, Power Management Schemes for Heterogeneous Clusters under Quality of Service Requirements, *Proc. of 26th ACM Symposium on Applied Computing*, pages 546–553, 2011.
- [41] Y.-S. Chen and M.-Y. Chen, On-line Energy-efficient Real-time Task

- 
- Scheduling for a Heterogeneous Dual-core System-on-a-chip, *Journal of Systems Architecture*, 59(45):234 – 244, 2013.
- [42] S. Cho and R. G. Melhem, On the Interplay of Parallelization, Program Performance, and Energy Consumption, *Trans. on Parallel and Distributed Systems*, 21(3):342–353, 2010.
- [43] G. Chrysos and Intel, Intel Xeon Phi Coprocessor - the Architecture, <https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner>, November 2012.
- [44] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee et al, An Energy Case for Hybrid Datacenters, *SIGOPS Operating Systems Review*, 44(1):76–80, Mar. 2010.
- [45] I.-H. Chung, S. R. Seelam, B. Mohr and J. Labarta, Tools for Scalable Performance Analysis on Petascale Systems, *Proc. of IPDPS*, pages 1–3, May 2009.
- [46] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser et al, LogP: towards a realistic model of parallel computation, *Proc. of the 4th Symposium on Principles and Practice of Parallel Programming*, pages 1–12, New York, NY, USA, 1993.
- [47] K. Czechowski, C. Battaglini, C. McClanahan, K. Iyer, P.-K. Yeung et al, On the Communication Complexity of 3D FFTs and its Implications for Exascale, *Proc. of 26th ICS*, pages 205–214, 2012.
- [48] K. Czechowski and R. Vuduc, A Theoretical Framework for Algorithm-architecture Co-design, *Proc. of 27th IPDPS*, pages 791–802, 2013.

- [49] C. Delimitrou and C. Kozyrakis, Paragon: QoS-aware Scheduling for Heterogeneous Datacenters, *Transactions on Computer Systems (TOCS)*, 41(1):77–88, 2013.
- [50] Y. Ding, K. Malkowski, P. Raghavan and M. Kandemir, Towards Energy Efficient Scaling of Scientific Codes, *International Symposium on Parallel and Distributed Processing*, pages 1–8, 2008.
- [51] J. Dongarra, High Performance Computing - Future Directions, *Keynote of 43rd ICPP*, 2014.
- [52] M. Duranton, D. Black-Schaffer, S. Yehia and K. De Bosschere, Computing Systems: Research Challenges Ahead: The HiPEAC Vision 2011/2012, 2011.
- [53] F. Dustin and Nvidia, Low-Power Sensing and Autonomy With NVIDIA Jetson TK1, <http://devblogs.nvidia.com/parallelforall/low-power-sensing-autonomy-nvidia-jetson-tk1>, June 2014.
- [54] D. Eager, J. Zahorjan and E. Lazowska, Speedup Versus Efficiency in Parallel Systems, *IEEE Trans. on Computers*, 38(3):408–423, 1989.
- [55] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam and D. Burger, Dark Silicon and the End of Multicore Scaling, *Proc. of 38th ISCA*, pages 365–376, 2011.
- [56] M. Etinski, J. Corbalan, J. Labarta, M. Valero and A. Veidenbaum, Power-aware Load Balancing of Large Scale MPI Applications, *International Symposium on Parallel Distributed Processing*, pages 1–8, 2009.
- [57] X. Fan, W.-D. Weber and L. A. Barroso, Power Provisioning for a Warehouse-sized Computer, *Proc. of 34th ISCA*, pages 13–23, 2007.

- 
- [58] J. Fang, A. L. Varbanescu, H. Sips, L. Zhang, Y. Che et al, Benchmarking Intel Xeon Phi to Guide Kernel Design.
- [59] J. Fang, A. L. Varbanescu, H. Sips, L. Zhang, Y. Che et al, An Empirical Study of Intel Xeon Phi, *arXiv preprint arXiv:1310.5842*, 2013.
- [60] M. P. I. Forum, <http://www.mpi-forum.org/>.
- [61] S. H. Fuller and L. I. Millett, Computing performance: Game over or next level?, *Computer*, 44(1):31–38, 2011.
- [62] A. Gandhi, M. Harchol-Balter and M. Kozuch, Are Sleep States Effective in Data Centers?, *Proc. of IGCC*, pages 1–10, 2012.
- [63] A. Gandhi, M. Harchol-Balter, R. Raghunathan and M. A. Kozuch, AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers, *ACM Transactions on Computer Systems*, 30(4):14:1–14:26, Nov. 2012.
- [64] S. Garg, S. Sundaram and H. D. Patel, Robust Heterogeneous Data Center Design: a Principled Approach, *SIGMETRICS Performance Evaluation Review*, 39(3):28–30, 2011.
- [65] R. Ge, X. Feng and K. W. Cameron, Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters, *Proc. of SC*, page 34, 2005.
- [66] C. George and I. , Intel Xeon Phi X100 Family Coprocessor - the Architecture, <https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner>, November 2012.

- [67] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car et al, QUANTUM ESPRESSO: a Modular and Open-source Software Project for Quantum Simulations of Materials, *Journal of Physics: Condensed Matter*, 21(39):395502 (19pp), 2009.
- [68] M. Guevara, B. Lubin and B. C. Lee, Market Mechanisms for Managing Datacenters with Heterogeneous Microarchitectures, *Transactions on Computer Systems (TOCS)*, 32(1):3, 2014.
- [69] J. Guo, J. Meng, Q. Yi, V. Morozov and K. Kumaran, Analytically Modeling Application Execution for Software-Hardware Co-design, *Proc of 28th IPDPS*, pages 468–477, 2014.
- [70] V. Gupta and K. Schwan, Brawny vs. Wimpy: Evaluation and Analysis of Modern Workloads on Heterogeneous Processors, *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 74–83. IEEE, 2013.
- [71] J. L. Gustafson, Reevaluating Amdahl’s law, *Commun. ACM*, 31(5):532–533, May 1988.
- [72] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi et al, Sources of Error in Full-system Simulation, *Proc. of Internation Symposium on Performance Analysis of Systems and Software*, pages 13–22, 2014.
- [73] V. Halyo, P. LeGresley, P. Lujan, V. Karpusenko and A. Vladimirov, First Evaluation of the CPU, GPGPU and MIC Architectures for Real Time Particle Tracking based on Hough Transform at the LHC, *arXiv preprint arXiv:1310.7556*, 2013.
- [74] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr. and R. Bianchini, Energy

- Conservation in Heterogeneous Server Clusters, *Proc. of 10th PPOPP*, pages 186–195, 2005.
- [75] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov et al, Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems based on Intel® Xeon Phi Coprocessor, *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 126–137, 2013.
- [76] S. Herbert and D. Marculescu, Analysis of Dynamic Voltage/frequency Scaling in Chip-multiprocessors, *International Symposium on Low Power Electronics and Design*, pages 38–43, 2007.
- [77] V. Heuveline, M. J. Krause and J. Latt, Towards a Hybrid Parallelization of Lattice Boltzmann Methods, *Computers Mathematics with Applications*, 58(5):1071 – 1080, 2009, Mesoscopic Methods in Engineering and Science.
- [78] M. Hill and M. Marty, Amdahl’s Law in the Multicore Era, *Computer*, 41(7):33–38, 2008.
- [79] U. Hölzle, Brawny Cores Still Beat Wimpy Cores, Most of the Time, *IEEE Micro*, 2010.
- [80] C. H. Hsu and W. C. Feng, A Power-aware Run-time System for High-performance Computing, *Proc. of SC*, 2005.
- [81] C.-H. Hsu and S. Poole, Revisiting Server Energy Proportionality, *Proc. of 42nd ICPP*, pages 834–840, 2013.
- [82] Intel, Intel Xeon Phi Coprocessor x100 Product Family Datasheet, <http://www.intel.sg/content/dam/www/public/us/en/documents/datasheets/xeon-phi-coprocessor-datasheet.pdf>, April 2015.



- [83] E. Ipek, B. Supinski, M. Schulz and S. McKee, An Approach to Performance Prediction for Parallel Applications, J. Cunha and P. Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 196–205. 2005.
- [84] V. Janapa Reddi, B. C. Lee, T. Chilimbi and K. Vaid, Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency, *Proc. of 37th ISCA*, pages 314–325, 2010.
- [85] C. L. Janssen, H. Adalsteinsson and J. P. Kenny, Using Simulation to Design Extremescale Applications and Architectures: Programming Model Exploration, *SIGMETRICS Perform. Eval. Rev.*, 38(4):4–8, Mar. 2011.
- [86] N. Kappiah, V. W. Freeh and D. K. Lowenthal, Just in Time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI programs, *Proc. of SC*, 2005.
- [87] U. R. Karpuzcu, A. Sinkar, N. S. Kim and J. Torrellas, EnergySmart: Toward energy-efficient manycores for Near-Threshold Computing, *Proc. of 19th HPCA*, pages 542–553, 2013.
- [88] L. Keys, S. Rivoire and J. D. Davis, The Search for Energy-efficient Building Blocks for the Data Center, *Proc. of 37th ISCA*, pages 172–182, 2010.
- [89] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban and C.-L. Wang, Heterogeneous Computing: Challenges and Opportunities, *Computer*, (6):18–27, 1993.
- [90] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin et al, Compiler-based data prefetching and streaming non-temporal store generation for the intel (r) xeon phi (tm) coprocessor, *Proc. of 27th International*

- 
- Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1575–1586, 2013.
- [91] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan and D. M. Tullsen, Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.
- [92] G. Lawson, M. Sosonkina and Y. Shen, Energy Evaluation for Applications with Different Thread Affinities on the Intel Xeon Phi, *International Symposium on Computer Architecture and High Performance Computing Workshop (SBAC-PADW)*, pages 54–59, 2014.
- [93] B. C. Lee and D. M. Brooks, Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction, *In Proc. of 12th ASPLOS*, volume 41, pages 185–194, 2006.
- [94] J. Lee and N. S. Kim, Optimizing Throughput of Power and Thermal-constrained Multicore Processors using DVFS and Per-core Power-gating, *Proc. of 49th Design Automation Conference*, pages 47–50, 2009.
- [95] S. Lee, S. Das, T. Pham, T. Austin, D. Blaauw et al, Reducing Pipeline Energy Demands with Local DVS and Dynamic Retiming, *Proc. of ISLPED*, pages 319–324, 2004.
- [96] J. Leverich and C. Kozyrakis, Reconciling High Server Utilization and Sub-millisecond Quality-of-service, *Proc. of 9th EuroSys*, pages 4:1–4:14, 2014.
- [97] B. Li, H.-C. Chang, S. Song, C.-Y. Su, T. Meyer et al, The Power-Performance Tradeoffs of the Intel Xeon Phi on HPC Applications, *In-*

- ternational Parallel & Distributed Processing Symposium Workshops*, pages 1448–1456, 2014.
- [98] D. Li, B. de Supinski, M. Schulz, D. Nikolopoulos and K. Cameron, Strategies for Energy-Efficient Resource Management of Hybrid Programming Models, *Trans. on Parallel and Distributed Systems*, 24(1):144–157, Jan 2013.
- [99] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen et al, McPAT: An Integrated Power, Area, and Timing modeling Framework for Multicore and Manycore Architectures, *Proc. of 42nd MICRO*, pages 469–480, Dec 2009.
- [100] I. M. Library, <https://software.intel.com/en-us/intel-mpi-library>.
- [101] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan and T. F. Wenisch, Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached, *Proc. of 40th ISCA*, pages 36–47, 2013.
- [102] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge et al, Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments, *Proc. of 35th ISCA*, pages 315–326, 2008.
- [103] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso and C. Kozyrakis, Towards Energy Proportionality for Large-scale Latency-critical Workloads, *Proc. of 41st ISCA*, pages 301–312, 2014.
- [104] F. Lu, J. Song, F. Yin and X. Zhu, Performance Evaluation of Hybrid Programming Patterns for Large CPU/GPU Heterogeneous Clusters, *Computer physics communications*, 183(6):1172–1181, 2012.
- [105] M. Luo, X. Lu, K. Hamidouche, K. Kandalla and D. K. Panda, Initial Study

- 
- of Multi-endpoint Runtime for MPI+OpenMP Hybrid Programming Model on Multi-core Systems, *Proc. of 19th PPOPP*, pages 395–396, 2014.
- [106] C. Ma, Y. M. Teo, V. March, N. Xiong, I. R. Pop et al, An approach for matching communication patterns in parallel applications, *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.
- [107] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi et al, Towards Energy-proportional Datacenter Memory with Mobile DRAM, *Proc. of 39th ISCA*, pages 37–48, 2012.
- [108] G. Mao, D. Böhme, M.-A. Hermanns, M. Geimer, D. Lorenz et al, Catching Idlers with Ease: A Lightweight Wait-State Profiler for MPI Programs, *Proc. of the 21th European MPI Users' Group Meeting*, Sept. 2014.
- [109] J. Mars and L. Tang, Whare-map: Heterogeneity in "Homogeneous" Warehouse-scale Computers, *Proc. of ISCA*, pages 619–630, 2013.
- [110] J. Mars, L. Tang, R. Hundt, K. Skadron and M. L. Soffa, Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations, *Proc. of 44th MICRO*, pages 248–259, 2011.
- [111] H. McCraw, J. Ralph, A. Danalis and J. Dongarra, Power Monitoring with PAPI for Extreme Scale Architectures and Dataflow-based Programming Models, *Proc. of International Conference on Cluster Computing*, pages 385–391, Sept 2014.
- [112] H. McCraw, J. Ralph, A. Danalis and J. Dongarra, Power monitoring with papi for extreme scale architectures and dataflow-based programming mod-

## REFERENCES

---

- els, *Proc. of International Conference on Cluster Computing*, pages 385–391, 2014.
- [113] D. S. McFarlin, C. Tucker and C. Zilles, Discerning the Dominant Out-of-order Performance Advantage: Is It Speculation or Dynamism?, *Proc. of 18th ASPLOS*, pages 241–252, 2013.
- [114] M. K. Mehmet-Ali, J. F. Hayes and A. Elhakeem, Traffic Analysis of a Local area Network with a Star Topology, *Trans. on Communications*, 36(6):703–712, 1988.
- [115] D. Meisner, B. T. Gold and T. F. Wenisch, PowerNap: Eliminating Server Idle Power, *Proc. of 14th ASPLOS*, pages 205–216, 2009.
- [116] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber and T. F. Wenisch, Power Management of Online Data-intensive Services, *Proceedings of the 38th ISCA*, pages 319–330, 2011.
- [117] D. Meisner and T. F. Wenisch, Does Low-power Design Imply Energy Efficiency for Data Centers?, *Proc. of 17th ISLPED*, pages 109–114, 2011.
- [118] D. Meisner and T. F. Wenisch, Dreamweaver: Architectural support for deep sleep, *Proc. of ASPLOS*, pages 313–324, 2012.
- [119] K. Meng, R. Joseph, R. P. Dick and L. Shang, Multi-optimization Power Management for Chip Multiprocessors, *Proc. of 17th PACT*, pages 177–186, 2008.
- [120] A. Merkel, J. Stoess and F. Bellosa, Resource-conscious Scheduling for Energy Efficiency on Multicore Processors, *Proc. of 5th European conference on Computer systems*, EuroSys '10, pages 153–166, 2010.

- 
- [121] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony and R. Rajkumar, Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling, *Proc. of 16th ICS*, pages 35–44, 2002.
- [122] A. Morris, A. Malony, S. Shende and K. Huck, Design and Implementation of a Hybrid Parallel Performance Measurement System, *Proc. of 39th ICPP*, pages 492–501, Sept 2010.
- [123] O. MPI, <http://www.open-mpi.org/>.
- [124] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra and S. Vishin, Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era, *Proc. of 50th DAC*, pages 174:1–174:9, 2013.
- [125] MVAPICH, <http://mvapich.cse.ohio-state.edu/>.
- [126] R. Nathuji, C. Isci and E. Gorbatoov, Exploiting Platform Heterogeneity for Power Efficient Data Centers, *Proc. of 4th ICAC*, pages 5–5, 2007.
- [127] NRDC and Anthesis, Scaling Up Energy Efficiency Across the Data Center Industry: Evaluating Key Drivers and Barriers, <http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf>, 2014.
- [128] V. Patil and V. Chaudhary, Rack Aware Scheduling in HPC Data Centers: An Energy Conservation Strategy, *Proc. of IPDPSW*, pages 814–821, May 2011.
- [129] S. J. Pennycook, S. D. Hammond, S. A. Jarvis and G. R. Mudalige, Performance Analysis of a Hybrid MPI/CUDA Implementation of the NASLU Benchmark, *ACM SIGMETRICS Performance Evaluation Review*, 38(4):23–29, 2011.

- [130] J. Polo, D. Carrera, Y. Becerra, V. Beltran, J. Torres et al, Performance Management of Accelerated Mapreduce Workloads in Heterogeneous Clusters, *Proc. of 29th International Conference on Parallel Processing*, pages 653–662, 2010.
- [131] M. Pricopi and T. Mitra, Bahurupi: A Polymorphic Heterogeneous Multi-core Architecture, *ACM TACO*, 8(4):22:1–22:21, 2012.
- [132] R. Rabenseifner, G. Hager and G. Jost, Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-core SMP Nodes, *Proc. of 17th PDP*, pages 427–436, 2009.
- [133] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang and X. Zhu, No Power Struggles: Coordinated Multi-level Power Management for the Data Center, *ACM SIGARCH Computer Architecture News*, volume 36, pages 48–59, 2008.
- [134] A. Ramachandran, J. Vienne, R. Van Der Wijngaart, L. Koesterke and I. Sharapov, Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi, *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 736–743, 2013.
- [135] A. Ramachandran, J. Vienne, R. Van Der Wijngaart, L. Koesterke and I. Sharapov, Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi, *42nd International Conference on Parallel Processing (ICPP)*, pages 736–743, Oct 2013.
- [136] L. Ramapantulu, D. Loghin and Y. M. Teo, An Approach for Energy Efficient Execution of Hybrid Parallel Programs, *Proc. of International Parallel and Distributed Processing Symposium*, pages 1000–1009, 2015.

- 
- [137] L. Ramapantulu, B. M. Tudor, D. Loghin, T. Vu and Y. M. Teo, Modeling the Energy Efficiency of Heterogeneous Clusters, *Proc. of 43rd ICPP*, pages 321–330, 2014.
- [138] S. Ramos and T. Hoefler, Modeling Communication in Cache-coherent SMP systems: A Case-study with Xeon Phi, *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 97–108, 2013.
- [139] D. Roca Marí, High Level Queuing Architecture Model for High-end Processors, 2014.
- [140] K. Rupp, CPU, GPU and MIC Hardware Characteristics over Time, <http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/>, June 2013.
- [141] F. Ryckbosch, S. Polfliet and L. Eeckhout, Trends in Server Energy Proportionality, *Computer*, 44(9):69–72, 2011.
- [142] M. Sadiku and S. Musa, Local Area Networks, *Performance Analysis of Computer Networks*, pages 167–195. Springer International Publishing, 2013.
- [143] V. Sarah, Dell offers 64-bit ARM Microserver Proof-of-concept for Hyperscale on the Heels of Open Compute Summit Momentum: <http://en.community.dell.com/dell-blogs/dell4enterprise/b/dell4enterprise/archive/2014/02/04/dell-offers-64-bit-arm-microserver-proof-of-concept-for-hyperscale-on-the-heels-of-open-compute-summit-momentum>, [online], Feb. 2014.
- [144] D. Schmidl, T. Cramer, S. Wienke, C. Terboven and M. S. Müller, Assessing



## REFERENCES

---

- the Performance of OpenMP Programs on the Intel Xeon Phi, *Euro-Par 2013 Parallel Processing*, pages 547–558. Springer, 2013.
- [145] Y. Shang, D. Li and M. Xu, A Comparison Study of Energy Proportionality of Data Center Network Architectures, *Proc. of 32nd ICDCSW*, pages 1–7, 2012.
- [146] Y. S. Shao and D. Brooks, Energy Characterization and Instruction-level Energy Model of Intel’s Xeon Phi Processor, *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 389–394. IEEE Press, 2013.
- [147] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir and C. R. Das, PE-PON: Performance-aware Hierarchical Power Budgeting for NoC based Multicores, *Proc. of 21st PACT*, pages 65–74, 2012.
- [148] M. Si, A. J. Peña, P. Balaji, M. Takagi and Y. Ishikawa, MT-MPI: Multithreaded MPI for Many-core Environments, *Proc. of the 28th ICS*, pages 125–134, 2014.
- [149] D. C. Snowdon, E. Le Sueur, S. M. Petters and G. Heiser, Koala: a Platform for OS-level Power Management, *Proc. of 4th ACM European conference on Computer systems*, EuroSys ’09, pages 289–302, 2009.
- [150] S. Song, C.-Y. Su, R. Ge, A. Vishnu and K. Cameron, Iso-Energy-Efficiency: An Approach to Power-Constrained Parallel Computation, *International Symposium on Parallel Distributed Processing*, pages 128–139, 2011.
- [151] SPEC, SPEC Power and Performance, Benchmark Methodology V2.1, [https://www.spec.org/power/docs/SPEC-Power\\_and\\_Performance\\_Methodology.pdf](https://www.spec.org/power/docs/SPEC-Power_and_Performance_Methodology.pdf), 2011.

- 
- [152] J. V.-N. Steven, Canonical claim the First ARM 64-bit Server production Software Deployment, Linux and Open Source: <http://www.linuxtoday.com/upload/applied-micro-canonical-claim-the-first-arm-64-bit-server-production-software-deployment-140529135505.html>, [online], May. 2014.
- [153] B. Subramaniam and W.-c. Feng, Towards Energy-proportional Computing for Enterprise-class Server Workloads, *Proc. of 4th ACM/SPEC International Conference on Performance Engineering*, pages 15–26, 2013.
- [154] K. Sudan, S. Balakrishnan, S. Lie, M. Xu, D. Mallick et al, A Novel System Architecture for Web Scale Applications using Lightweight CPUs and Virtualized I/O, *Proc. of 19th HPCA*, pages 167–178, 2013.
- [155] V. Taylor, X. Wu and R. Stevens, Prophecy: an Infrastructure for Performance Analysis and Modeling of Parallel and Grid applications, *SIGMETRICS Perf. Eval. Review*, 30(4):13–18, 2003.
- [156] D. Tsirogiannis, S. Harizopoulos and M. A. Shah, Analyzing the Energy Efficiency of a Database Server, *Proc. of SIGMOD*, pages 231–242, 2010.
- [157] B. M. Tudor and Y. M. Teo, A Practical Approach for Performance Analysis of Shared-memory Programs, *Proc. of 25th IPDPS*, pages 652–663, 2011.
- [158] B. M. Tudor and Y. M. Teo, On Understanding the Energy Consumption of ARM-based Multicore Servers, *Proc. of SIGMETRICS*, pages 267–278, 2013.
- [159] D. Turner, A. Oline, X. Chen and T. Benjegerdes, Integrating new capabilities into netpipe, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 37–44. Springer, 2003.

- [160] K. Van Craeynest and L. Eeckhout, Understanding Fundamental Design Choices in Single-ISA Heterogeneous Multicore Architectures, *ACM TACO*, 9(4):32:1–32:23, 2013.
- [161] R. F. Van der Wijngaart and H. Jin, NAS Parallel Benchmarks, Multi-zone Versions, 2003.
- [162] G. Varsamopoulos and S. Gupta, Energy Proportionality and the Future: Metrics and Directions, *Proc. of 39th ICPPW*, pages 461–467, 2010.
- [163] J. S. Vetter and M. O. McCracken, Statistical Scalability Analysis of Communication Operations in Distributed Applications, *Proc. of 8th PPOPP*, pages 123–132, 2001.
- [164] J. S. Vetter and F. Mueller, Communication characteristics of large-scale scientific applications for contemporary cluster architectures, *Journal of Parallel and Distributed Computing*, 63(9):853–865, 2003.
- [165] A. Vladimirov and V. Karpusenko, Heterogeneous Clustering with Homogeneous Code: Accelerate MPI Applications without Code Surgery using Intel Xeon Phi coprocessors, *Colfax White Paper*, 2013.
- [166] J. Wang, N. Rubin, H. Wu and S. Yalamanchili, Accelerating Simulation of Agent-Based Models on Heterogeneous Architectures, *Proc. of 6th Workshop on GPGPU*, 2013.
- [167] V. M. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula et al, Papi 5: Measuring power, energy, and the cloud, *Proc. of International Symposium on Performance Analysis of Systems and Software*, pages 124–125, 2013.

- 
- [168] M. Weiser, B. Welch, A. Demers and S. Shenker, Scheduling for Reduced CPU Energy, *Proc. of 1st USENIX conference on Operating Systems Design and Implementation*, 1994.
- [169] A. Wierman, L. L. H. Andrew and A. Tang, Power-aware Speed Scaling in Processor Sharing systems: Optimality and Robustness, *Perform. Eval.*, 69(12):601–622, Dec. 2012.
- [170] D. Wong and M. Annavaram, Knightshift: Scaling the Energy Proportionality Wall through Server-level Heterogeneity, *Proc. of 45th International Symposium on Microarchitecture*, pages 119–130, 2012.
- [171] D. Wong and M. Annavaram, Scaling the Energy Proportionality Wall with KnightShift, *Micro, IEEE*, 33(3):28–37, 2013.
- [172] D. Wong and M. Annavaram, Implications of High Energy Proportional Servers on Cluster-wide Energy Proportionality, *Proc. of 20th HPCA*, pages 142–153, 2014.
- [173] D. H. Woo and H.-H. S. Lee, Extending Amdahl’s Law for Energy-Efficient Computing in the Many-Core Era., *IEEE computer*, 41(12):24–31, 2008.
- [174] H. Yang, A. Breslow, J. Mars and L. Tang, Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers, *Proc. of 40th ISCA*, pages 607–618, 2013.
- [175] S. Yeo and H.-H. S. Lee, Using Mathematical Modeling in Provisioning a Heterogeneous Cloud Computing Environment, *Computer*, 44(8):55–62, 2011.
- [176] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz and I. Stoica, Improving MapReduce Performance in Heterogeneous Environments, *Proc. of the*

## REFERENCES

---

- 8th USENIX Conference on Operating Systems Design and Implementation*, pages 29–42, Berkeley, CA, USA, 2008.
- [177] X. Zhu, C. He, K. Li and X. Qin, Adaptive Energy-efficient Scheduling for Real-time Tasks on DVS-enabled Heterogeneous Clusters, *Journal of Parallel and Distributed Computing*, 72(6):751–763, 2012.
- [178] S. Zikos and H. D. Karatza, Performance and Energy Aware Cluster-level Scheduling of Compute-intensive Jobs with Unknown Service Times , *Simulation Modelling Practice and Theory*, 19(1):239 – 250, 2011.
- [179] Z. Zong, X. Qin, X. Ruan, K. Bellam, M. Nijim et al, Energy-efficient Scheduling for Parallel Applications Running on Heterogeneous Clusters, *Proc. of ICPP*, 2007.



# Appendix A

## Experiment Setup

This thesis presents a measurement-based modeling approach for time-energy performance analysis of heterogeneous systems and hybrid programs. This section describes the setup of the experiments conducted for this thesis. We first describe the complete set of programs used, followed by the software setup needed, the hardware used for the experiments and lastly the validation results.

### A.1 Programs

As the core model presented in this thesis is applicable for datacenter workloads, there are six programs representing different performance bottlenecks and with different deadline requirements. *EP*, from NAS Parallel Benchmarks (NPB), is an embarrassingly parallel distributed-memory program that generates random numbers for Monte-Carlo numerical simulation. *Memcached* is widely used by Facebook, Amazon, Twitter, among others, as an in-memory key-value distributed storage. When a key request arrives, a front-end node dispatches the request to a set of nodes that are responsible for storing the key-values belonging to an application. All nodes in the pool perform a key look-up computation, but

typically few nodes return the value. However, this operation may exert complex service demands on core, memory and I/O devices. We use memslap running on another system to trigger requests to the memcached server over a 1 Gbps network connection. Note that memslap generates requests with fixed key-value size and uniform popularity.

From the PARSEC benchmark suite, *x264* represents the widely used encoding algorithm for streaming video, and *blackscholes* represents a quantitative model for determining option pricing. The open source speech recognition engine *Julius* represents the increasing adoption of real-time speech processing workloads originating from smart devices. To analyze the energy efficiency of web security, we use the openssl *RSA-2048* speed benchmark because major web players are increasingly concerned with the in-transit data security and are hardening the https encryptions.

While the core model is applicable on heterogeneous systems, to illustrate the scalability of the core mode, this thesis presents a communication model applicable on generic hybrid parallel programs. To illustrate the accuracy of the communication model, we selected a representative subset of five hybrid openMP+MPI benchmark programs for validation. This subset was chosen to represent a wide range of HPC domain applications that exert different inter and intra-node communication resource demands and use different programming languages. We use three hybrid programs from NASA Parallel Benchmark (NPB) suite. These solve discretized version of Navier-Stokes equations in three dimensions, and are (i) Lower-Upper Symmetric Gauss-Seidel (LU), (ii) Scalar Penta-diagonal (SP), and (iii) Block Tri-diagonal (BT). The fourth program uses the Car-Parinello (CP) method to simulate  $H_2O$  molecules from the Quantum Espresso suite. While the above programs are in Fortran, we chose the fifth program in C++, to illustrate that our approach is independent of the programming language. This



is an open source Lattice Boltzmann (LB) code, that simulates fluid flows in a three-dimensional lid-driven cavity.

Table A.1 summarizes the complete set of programs used in the thesis.

Domain	Benchmark Suite	Program
Monte-Carlo Simulation	NAS NPB (v3.3)	EP
Webserver Requests	memcached.org (v1.4.25)	memcached
Streaming Video	PARSEC (v3.0)	x264
Financial	PARSEC (v3.0)	blacksc-holes
Speech Recognition	<a href="http://julius.osdn.jp">http://julius.osdn.jp</a> (v4.2.3)	Julius
Web Security	openssl speed	RSA-2048
3D Navier-Stokes Equation Solver	NAS Multi-zone Parallel Benchmark (NPB3.3-MZ)	LU SP BT
Electronic-structure Calculations	Quantum Espresso (v5.1)	CP
Computational Fluid Dynamics	OpenLB (olb-0.8r0)	LB

Table A.1: Programs used in thesis

## A.2 Systems

Many datacenter workloads must obey strict service time deadlines. To service requests within a deadline, processing is distributed over hundreds of server nodes. Jobs arrive at front-end nodes and are forwarded to a cluster of compute nodes that service job requests. Both response time and the energy incurred by a job are dominated by compute nodes. Thus, this thesis focuses on the energy efficiency of compute nodes only. Table A.2 presents the full set of compute systems used in the thesis. The nodes are tabulated in the ascending order of number of cores and system idle power.

The ARM server node analyzed throughout this thesis is the Odroid-X development board with Samsung Exynos 4412 System on a Chip (SoC). Specific to the Exynos 4412 SoC is a quad-core ARM Cortex-A9 processor. The operating core clock frequencies supported are between 200 MHz and 1400 MHz, in incre-

Node	ARM Cortex A9	AMD Opteron K10	Intel Xeon E5	Intel Xeon Phi (5110P)
Year procured	2012	2010	2013	2012
ISA	ARMv7-A	x86_64	x86_64	x8664
Clock Freq	0.2–1.4 GHz	0.8–2.1 GHz	1.2–1.8 GHz	0.6–1.0GHz
Cores/node	4	6	8	60
L1 data cache	32KB / core	64KB / core	32KB / core	32KB / core
L2 cache	1MB / node	512KB / core	2MB / node	512KB /core
L3 cache	NA	6MB / node	20MB / node	NA
Memory	1GB LP-DDR2	8GB DDR3	8GB DDR3	8GB DDR3
I/O bandwidth	100Mbps	1Gbps	1Gbps	NA
Peak power [W]	5	60	80	225
Power efficiency [Gflops/W]	0.16	0.59	1.37	4.5

Table A.2: Systems used in thesis

ments of 100 MHz, the available bandwidth between cores and main memory is 800 MB/s, and the network device is a 100 Mbps Ethernet card. The AMD server node consists of the K10 processor architecture with six cores, each operating at a maximum frequency of 2.10GHz. These six cores share a 6MB L3 cache among them and are equipped with a 8 GB dual-channel DDR3 RAM. The Intel node is a Supermicro 813M 1U server system based on two Intel Xeon E5-2603 CPUs with four cores each. This system has 8 GB DDR3 memory, 1 TB hard disk and 1 Gbit Ethernet network card. The data for the Intel Xeon Phi coprocessor, 5110P is from the data sheet specification of Intel [82].

### A.3 Software Setup

To derive workload dependent architectural artefacts, we use baseline executions of the program on a single node. To determine the overlap among useful computation cycles, data-accesses from shared-memory and network, we measure the translation of a given parallel program into useful work cycles ( $w_s$ ). To model the non-overlapped intra-node contention, we measure the stall cycles due to memory accesses ( $m_s$ ). These measurements are recorded for a single node across the possible values of  $c$  and  $f$  using hardware performance counters. Hence, these

measurements are non-intrusive with respect to the execution of the application. We use the linux *perf* tool to measure backend stall cycles and derive the work cycles from the total cycles and the backend stall cycles.

Program dependent communication characteristics, such as number of communication calls ( $\eta$ ) and communication volume per message call ( $\nu$ ) are measured using the lightweight profiling tool mpiP. It suffices to perform baseline executions only on a single node, as workload characteristics from these measured values can be inferred from  $\ell$  and  $\tau$ . To measure communication overheads of MPI over TCP for a given link bandwidth, we use NetPIPE. This is used to measure the throughput loss due to the additional software layers between the hybrid program and the hardware drivers. This characterization of the network link latency and bandwidth is used to compute network service time of the communication model.

Table A.3 presents the versions of the software used for conducting the experiments.

Software	Version
Host OS, Linux	<b>A9:</b> 3.6.11 32-bit <b>K10:</b> 2.6.35 64-bit <b>E5:</b> 3.8.0 64-bit
perf	<b>A9:</b> 3.6.0 <b>K10:</b> 3.13.11-ckt20 <b>E5:</b> 3.8.13.13
Open MPI	1.6.5
NetPIPE	3.7.2
mpiP	3.4.1
Xeon Phi uOS	Linux 2.6.38.8
Xeon Phi software stack	MPSS 3.4.2
PAPI	5.3.2

Table A.3: Software versions used in thesis

## A.4 Validation Results

In this section, we present the absolute validation numbers for both execution time and energy. Table A.4 and A.5 present the measured and model determined values of execution time across all possible active core values for the single ARM and AMD node respectively.

Configuration		Execution Time [s]											
Freq (GHz)	Cores	EP		memcached		x264		blackscholes		Julius		RSA-2048	
		Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted
0.2	1	2390.66	2433.86	245.72	209.66	4393.39	4603.83	282.10	280.40	78.53	90.81	25.67	27.87
0.5	1	973.43	973.54	112.86	104.04	1756.45	1841.53	112.47	112.16	31.81	36.32	10.08	11.13
0.8	1	593.77	608.46	88.03	89.35	1102.61	1150.96	70.36	70.16	20.32	22.70	6.29	6.95
1.1	1	445.65	442.52	95.67	101.14	813.05	837.06	51.26	51.12	14.93	16.51	4.57	5.05
1.4	1	343.60	347.69	87.32	94.64	642.17	657.69	40.33	40.26	11.93	12.97	3.59	3.97
0.2	2	1207.35	1216.93	175.60	194.25	2292.48	2301.92	158.60	157.97	79.65	90.81	12.72	13.96
0.5	2	493.14	486.77	74.06	69.14	920.23	920.77	63.37	63.19	32.81	36.32	5.05	5.56
0.8	2	306.45	304.23	63.73	59.92	578.91	575.48	39.63	39.49	21.19	22.70	3.15	3.47
1.1	2	217.96	221.26	70.05	65.54	426.62	422.57	28.87	28.79	15.95	16.58	2.29	2.53
1.4	2	177.74	173.85	66.96	62.92	338.43	335.53	22.83	22.70	12.91	13.42	1.80	1.98
0.2	3	826.62	811.29	139.95	152.43	1538.79	1534.61	118.35	117.67	81.07	90.81	8.48	9.31
0.5	3	332.32	324.51	69.48	64.86	619.22	613.84	47.09	46.95	33.96	36.32	3.36	3.71
0.8	3	208.49	202.82	60.13	56.17	391.46	386.52	29.40	29.31	22.23	23.09	2.10	2.32
1.1	3	151.39	147.51	66.21	62.29	289.32	286.09	21.42	21.36	17.01	17.65	1.53	1.68
1.4	3	119.51	115.90	63.58	59.92	230.38	228.02	16.88	16.82	14.00	14.52	1.20	1.32
0.2	4	617.81	608.46	135.17	125.83	1173.19	1150.96	97.63	97.19	92.79	90.81	6.34	6.98
0.5	4	244.99	243.39	65.98	61.68	474.74	461.05	38.88	38.77	41.54	36.50	2.52	2.78
0.8	4	150.97	152.12	58.56	54.71	303.56	292.55	24.29	24.21	26.36	24.22	1.58	1.74
1.1	4	112.23	110.63	63.60	59.92	224.92	217.42	17.69	17.64	21.42	19.12	1.15	1.26
1.4	4	87.33	86.92	62.03	58.26	181.30	173.53	13.93	13.89	17.70	15.90	0.90	0.99

Table A.4: Execution time validation on a single ARM node

Configuration		Execution Time [s]											
Freq (GHz)	Cores	EP		memcached		x264		blackscholes		Julius		RSA-2048	
		Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted
0.8	1	329.05	328.72	56.47	51.65	119.23	119.82	37.87	37.78	7.67	8.36	0.95	0.94
1.4	1	186.84	187.84	56.45	52.08	69.17	68.89	21.58	21.78	4.32	4.72	0.54	0.54
2.1	1	124.73	125.10	56.31	51.23	47.05	46.33	14.44	14.46	2.96	3.15	0.36	0.36
0.8	2	165.79	165.44	56.84	52.08	62.81	59.86	21.23	21.15	7.61	8.32	0.48	0.47
1.4	2	94.77	94.92	56.21	51.23	36.91	34.43	12.07	12.19	4.34	4.75	0.27	0.27
2.1	2	63.72	63.60	56.65	52.08	25.44	23.21	8.04	8.10	2.96	3.18	0.18	0.18
0.8	3	110.66	110.61	56.13	51.65	42.17	39.90	15.60	15.73	7.57	8.38	0.32	0.31
1.4	3	63.65	63.64	56.78	52.00	24.76	23.04	8.94	9.00	4.34	4.78	0.18	0.18
2.1	3	42.76	42.76	55.77	52.43	17.27	15.60	6.02	5.97	2.95	3.22	0.12	0.12
0.8	4	84.93	83.22	56.76	62.05	31.75	29.93	12.77	12.83	7.53	8.42	0.24	0.24
1.4	4	47.90	48.02	55.79	52.00	18.69	17.18	7.27	7.32	4.37	4.83	0.14	0.13
2.1	4	32.34	32.35	56.41	52.00	13.11	11.65	5.07	5.03	2.94	3.25	0.09	0.09
0.8	5	67.10	66.84	55.85	56.09	25.59	23.95	11.16	11.14	7.53	8.42	0.19	0.19
1.4	5	38.61	38.65	56.38	52.00	15.09	13.85	6.34	6.34	4.34	4.81	0.11	0.11
2.1	5	26.13	26.11	56.23	51.15	10.65	9.35	4.24	4.27	2.94	3.24	0.07	0.07
0.8	6	56.18	55.93	56.66	52.00	21.46	20.03	10.03	10.02	7.55	8.45	0.16	0.16
1.4	6	32.45	32.40	56.49	52.00	12.71	11.57	5.81	5.80	4.30	4.84	0.09	0.09
2.1	6	22.00	21.96	56.39	52.00	8.94	7.82	3.83	3.82	2.99	3.25	0.06	0.06

Table A.5: Execution time validation on a single AMD node

Table A.6 and A.7 present the measured and model determined values of energy across all possible active core values for the single ARM and AMD node respectively.

## Chapter A. Experiment Setup

Configuration		Energy [J]											
		EP		memcached		x264		blackscholes		Julius		RSA-2048	
Freq (GHz)	Cores	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted
0.2	1	4849.20	4537.49	550.80	471.09	8917.20	9287.42	546.12	563.61	158.40	180.15	158.40	162.04
0.5	1	1893.60	1901.16	219.60	244.45	3484.80	3912.11	221.04	234.50	72.00	75.05	158.40	170.76
0.8	1	1245.60	1269.16	190.80	218.85	2372.40	2602.40	148.32	155.75	46.80	49.95	172.80	181.25
1.1	1	1047.60	1019.13	237.60	256.81	2023.20	2085.59	120.24	124.30	36.00	39.85	187.20	199.11
1.4	1	1036.80	948.77	273.60	262.76	2185.20	1935.48	111.96	114.68	32.40	36.76	244.80	234.20
0.2	2	2466.00	2305.64	442.80	446.98	4676.40	4738.93	298.44	321.34	162.00	182.51	162.00	165.48
0.5	2	1036.80	1006.21	147.60	168.25	1958.40	2068.44	128.88	138.92	72.00	79.17	169.20	179.97
0.8	2	709.20	689.20	140.40	149.22	1389.60	1413.07	86.76	94.37	50.40	53.95	187.20	195.88
1.1	2	604.80	580.25	180.00	173.12	1278.00	1208.06	73.80	78.61	43.20	45.08	223.20	228.58
1.4	2	658.80	563.31	212.40	180.08	1342.80	1182.42	74.16	75.55	46.80	44.04	309.60	282.15
0.2	3	1663.20	1558.39	313.20	361.24	3088.80	3215.55	227.88	241.94	162.00	184.69	162.00	168.48
0.5	3	730.80	698.75	140.40	154.28	1404.00	1438.34	98.64	106.91	75.60	82.28	172.80	187.41
0.8	3	525.60	493.90	129.60	135.86	1051.20	1026.42	68.40	74.57	54.00	58.37	201.60	212.22
1.1	3	478.80	431.36	169.20	159.46	997.20	910.77	59.76	64.32	50.40	51.96	252.00	255.43
1.4	3	511.20	437.67	205.20	166.69	1105.20	944.44	60.12	64.30	54.00	52.90	370.80	335.63
0.2	4	1260.00	1191.69	262.80	300.74	2376.00	2458.31	189.00	203.05	187.20	188.04	165.60	171.67
0.5	4	565.20	546.73	133.20	145.23	1137.60	1128.96	81.72	91.58	93.60	85.92	180.00	195.79
0.8	4	410.40	395.37	126.00	130.98	874.80	830.32	56.52	65.21	68.40	64.02	216.00	227.51
1.1	4	388.80	355.08	162.00	151.20	860.40	760.54	51.12	57.73	68.40	59.36	280.80	282.82
1.4	4	442.80	377.62	187.20	160.18	928.80	829.68	56.52	60.36	61.20	62.78	432.00	392.04

Table A.6: Energy validation on a single ARM node

Configuration		Energy [J]											
		EP		memcached		x264		blackscholes		Julius		RSA-2048	
Freq (GHz)	Cores	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted	Measured	Predicted
0.8	1	17054.13	15263.11	2371.73	2552.49	5652.73	5851.94	1933.31	1847.54	478.08	429.74	3800.00	3911.71
1.4	1	10343.40	8886.97	2378.75	2606.23	3452.50	3433.04	1254.16	1084.51	228.98	242.59	4340.00	4000.57
2.1	1	6360.71	6132.54	2782.40	2592.70	2438.68	2402.94	823.57	745.62	151.77	162.03	4700.00	4199.27
0.8	2	9007.80	7918.46	2436.03	2657.60	3033.20	3003.36	1048.37	1063.96	497.07	462.85	3944.00	4010.23
1.4	2	5444.89	4710.13	2812.54	2648.80	1859.55	1796.33	669.27	634.87	242.06	264.36	4592.00	4193.68
2.1	2	3811.08	3373.51	3036.10	2710.11	1323.72	1304.81	528.52	450.64	171.09	176.82	5168.00	4592.40
0.8	3	5603.24	5452.45	2480.05	2718.14	1924.36	2055.87	769.89	813.06	546.11	503.30	4088.00	4110.40
1.4	3	3474.64	3304.75	2633.52	2756.49	1369.45	1255.40	497.64	489.07	313.50	287.06	4880.00	4387.41
2.1	3	2450.56	2439.77	2593.00	2697.04	1104.18	943.19	416.66	356.93	192.88	193.32	5528.00	4976.46
0.8	4	4373.74	4221.62	2927.58	3340.47	1404.03	1582.02	659.95	681.13	520.05	540.92	4556.00	4208.65
1.4	4	2913.66	2604.78	2944.78	2728.73	1031.56	977.13	417.45	414.94	270.47	310.21	5132.00	4579.36
2.1	4	2288.33	1975.66	2750.45	2654.90	713.81	755.56	352.71	320.42	231.96	208.80	5600.00	5365.82
0.8	5	3777.65	3486.42	2927.70	3060.19	-91.76	1298.12	564.17	607.51	664.83	577.85	4736.00	4308.27
1.4	5	2379.96	2185.60	2761.25	2713.05	779.32	819.00	386.01	374.42	315.67	330.33	5024.00	4768.11
2.1	5	1816.09	1699.70	2989.12	2599.30	733.91	646.24	338.47	289.50	249.21	222.30	5960.00	5768.73
0.8	6	3061.58	2997.11	2673.41	2854.37	1199.00	1111.71	516.21	560.33	588.10	616.42	4484.00	4412.33
1.4	6	1892.15	1906.97	2724.88	2701.57	810.97	710.34	367.10	355.80	327.69	353.03	5276.00	4967.60
2.1	6	1381.08	1517.75	2756.56	2633.49	650.17	573.47	324.47	274.84	268.44	236.99	6680.00	6158.39

Table A.7: Energy validation on a single AMD node

# Appendix B

## Model Parameters

The measurement-driven inputs to our model are obtained from workload characterization using baseline executions and power characterization using micro-benchmarks. Typical scale-out workloads used in datacenters exhibit a lot of parallelism due to both user requests and data. The computations of such workloads can be divided into repetitive parallel execution phases within a request and also across a batch of requests.

The representative subset  $\mathcal{P}_s$  of the scale-out workload used in our model is this repeating parallel phase. For example, in memcached program, each of the GET, SET and DELETE request types are a parallel phase of execution. We measure the number of instructions, work cycles and stall cycles for a small subset of GET, SET and DELETE commands to capture the architecture specific workload parameters for each type of node. The measurements used in this thesis are done using hardware event counters in the respective nodes. The measurements are done only once for each type of node being used. It suffices to do the measurements on a single node of each type, because all the nodes of the same type exhibit very similar power characteristics, which we have validated. Table B.1 presents the

complete list of notations<sup>1</sup> used in this thesis.

---

<sup>1</sup>Notations with \* denotes measured parameters of the model.

Symbol	Description
<b>Workload Parameters</b>	
$\mathcal{P}$	program
$\mathcal{P}_s$	program $\mathcal{P}$ with smaller input size
$S$	number of iterations in $\mathcal{P}$
$S_s$	number of iterations in $\mathcal{P}_s$
$W$	total work units of $\mathcal{P}$
$r_i$	proportion of workload executed on nodes of type $i$ , where $i \in [1 \cdots d_{max}]$
$\lambda_{I/O}^*$	I/O requests inter-arrival rate
$\eta^*$	number of messages sent/received by $\mathcal{P}$
$\nu^*$	volume (in bytes) per message
<b>System Parameters</b>	
$d_{max}$	maximum degree of inter-node heterogeneity of the system, $i \in [1 \cdots d_{max}]$
$n_{max}$	maximum number of nodes of type $i$
$c_{max}$	maximum number of cores for nodes of type $i$
$f_{max}$	maximum core clock frequency for nodes of type $i$
$B$	communication throughput
<b>Baseline Execution</b>	
$I_s^*$	number of instructions in $\mathcal{P}_s$
$w_s^*$	number of work cycles in $\mathcal{P}_s$
$b_s^*$	number of non-memory stall cycles in $\mathcal{P}_s$
$m_s^*$	number of memory-related stall cycles in $\mathcal{P}_s$
$U_s^*$	Average CPU utilization for $\mathcal{P}_s$
<b>Time Model</b>	
$I$	number of instructions in $\mathcal{P}$
$w$	number of work cycles for $\mathcal{P}$
$b$	number of non-memory stall cycles for $\mathcal{P}$
$m$	number of memory-related stall cycles for $\mathcal{P}$
$U$	CPU utilization for $\mathcal{P}$
$n$	number of nodes
$c$	number of active cores per node
$f$	operating core clock frequency
$T_{CPU}$	total CPU response time for $\mathcal{P}$
$T_{core}$	total core response time for $\mathcal{P}$
$T_{mem}$	total memory response time for $\mathcal{P}$
$T_{I/O}$	total I/O response time for $\mathcal{P}$
$T_{I/O_T}$	total I/O transfer time for $\mathcal{P}$
$T_{w,net}$	waiting time due to network contention
$T_{s,net}$	non-overlapped network service time
$T$	total execution time of program $\mathcal{P}$
<b>Power Parameters[W]</b>	
$P_{CPU,act}^*$	CPU power when executing work cycles
$P_{CPU,stall}^*$	CPU power when memory-related stalls
$P_{mem}^*$	power consumed by memory operations
$P_{net}^*$	power consumed by network card
$P_{sys,idle}^*$	power consumed by idle system
<b>Energy Model[J]</b>	
$E_{CPU,act}$	total energy consumed when CPU is active
$E_{CPU,stall}$	total energy consumed when CPU is stalling
$E_{mem}$	total energy consumed by memory sub-system
$E_{net}$	total energy consumed by network sub-system
$E_{idle}$	total energy consumed by idle system
$E$	total energy consumed by a program

Table B.1: Notations used in thesis



# Appendix C

## Sensitivity Analysis

### C.1 What is a Good Mix of High-performance to Low-power Nodes?

Since datacenters often have an upper bound on their peak power consumption, we consider a fixed peak power budget drawn by our system that constrains the maximum number of nodes. Based on peak power proportion between ARM and AMD nodes, we analyze the impact of replacing some high-performance AMD nodes by low-power ARM nodes such that the total peak power is within the budget.

Figures C.1 and C.2 show<sup>1</sup> the impact of changing the number of ARM and AMD nodes, for a given budget of 1kW. We use an ARM to AMD power substitution ratio<sup>2</sup> of 8:1. The graphs clearly show that heterogeneous mixes with a larger number of ARM nodes incur lower energy for a given execution time.

---

<sup>1</sup>Henceforth each figure plots Pareto frontiers with x-axis in log-scale.

<sup>2</sup>Since each AMD node draws a peak power of 60W and each ARM node draws a peak power of 5W, one AMD node can be replaced by 12 ARM nodes. Factoring the 20W peak power drawn by the switch [3] that connects the ARM nodes, gives us a power substitution ratio of 8:1.

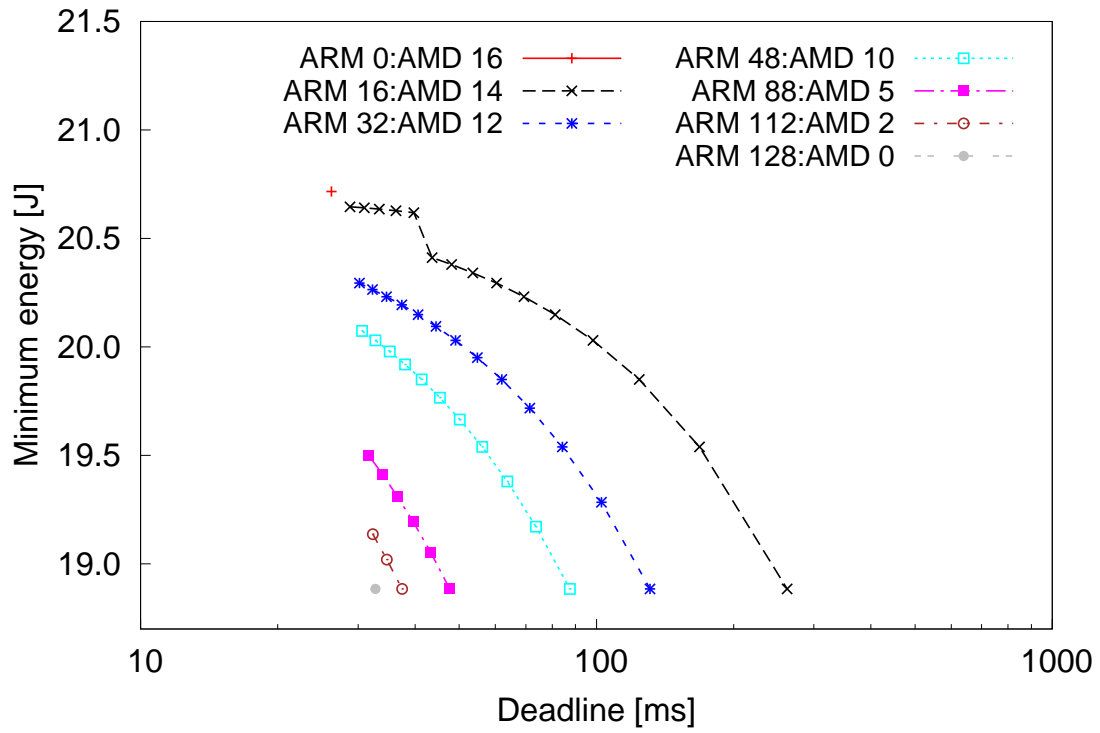


Figure C.1: Heterogeneous mixes for memcached

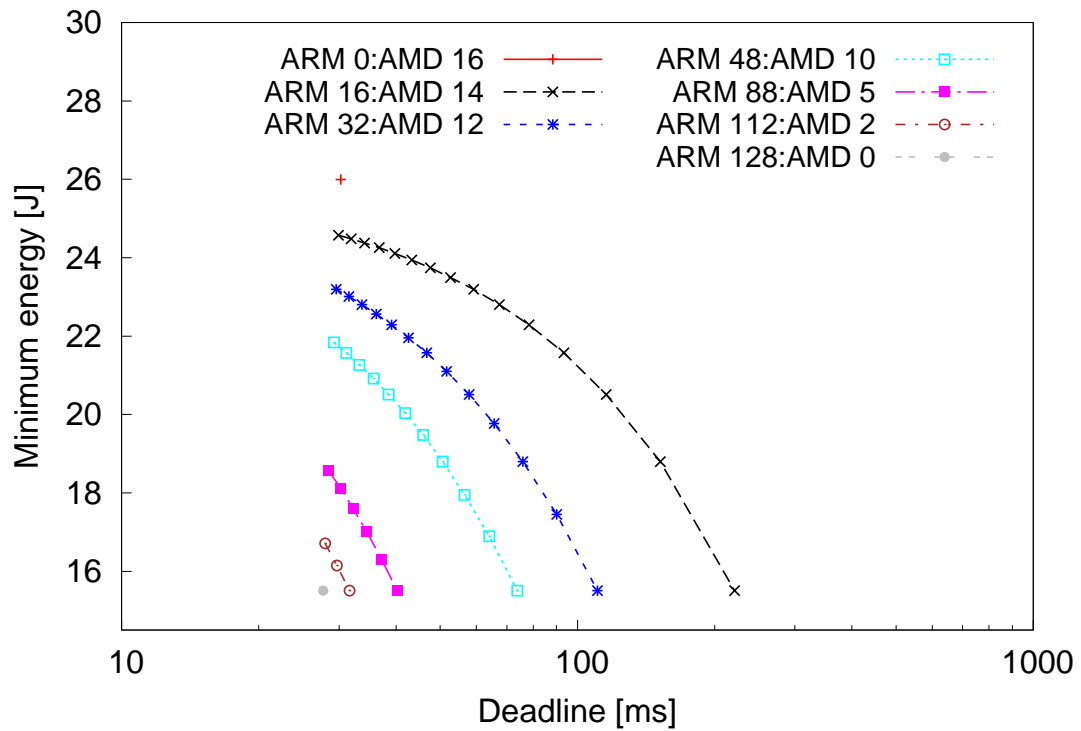


Figure C.2: Heterogeneous mixes for EP

**Observation 1:** *Replacing even a few high-performance nodes based on the power substitution ratio, introduces a sweet region.*

However, using only low-power nodes may not meet the service time deadline. For example, Figure C.1 shows that low-power ARM only configurations do not meet deadlines smaller than 30ms.

For an application that is compute-bound, such as EP, replacing even a few AMD nodes triggers a sweet region. However, the most energy-efficient configuration is achieved by replacing all AMD nodes with ARM nodes. This is possible because, while eight ARM nodes are power-equivalent to one AMD nodes, the execution rate of eight ARM nodes is higher than one AMD node. The implications of the performance difference between high-performance and low-power nodes are further discussed in section C.3.

## C.2 Are Larger Mixes of Heterogeneous Nodes Better?

Using the same power substitution ratio, Figures C.3 and C.4 show that increasing the number of heterogeneous nodes does not change the energy bounds of a sweet region. Secondly, it increases the number of configurations on a sweet region. Thirdly, as expected, increasing the number of nodes results in faster execution time, causing the sweet regions to shift to the left.

**Observation 2:** *Increasing the number of nodes in a heterogeneous mix, while maintaining the same power substitution ratio increases the number of configurations on a sweet region without changing its energy bounds.*

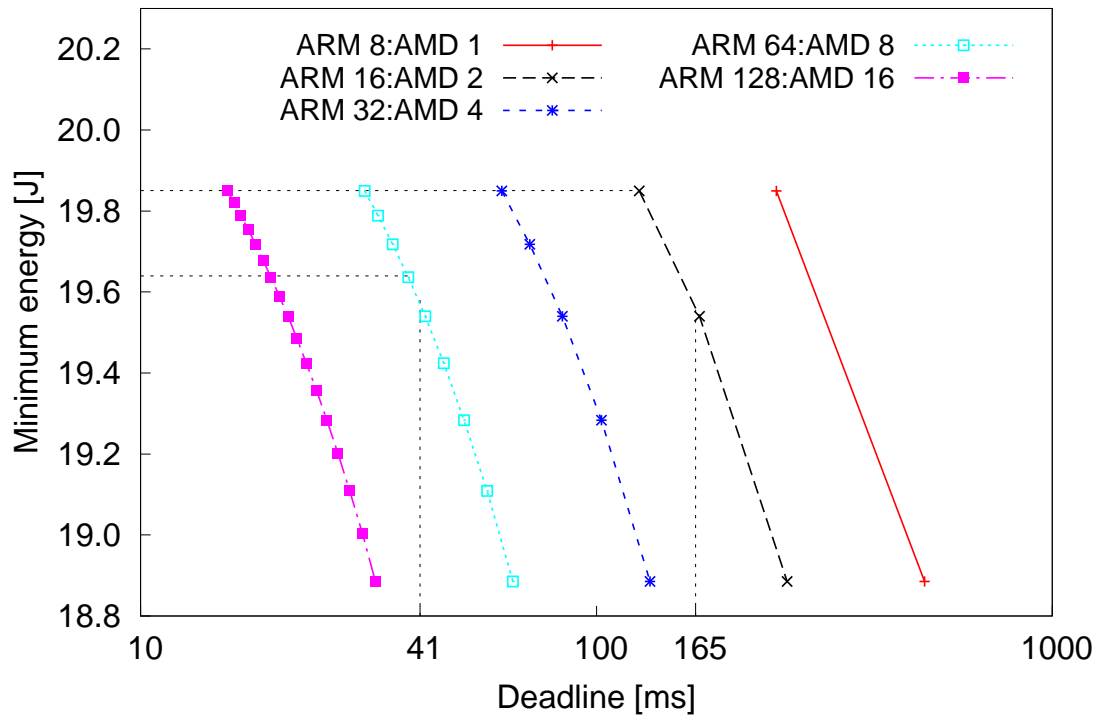


Figure C.3: Increasing cluster size for memcached

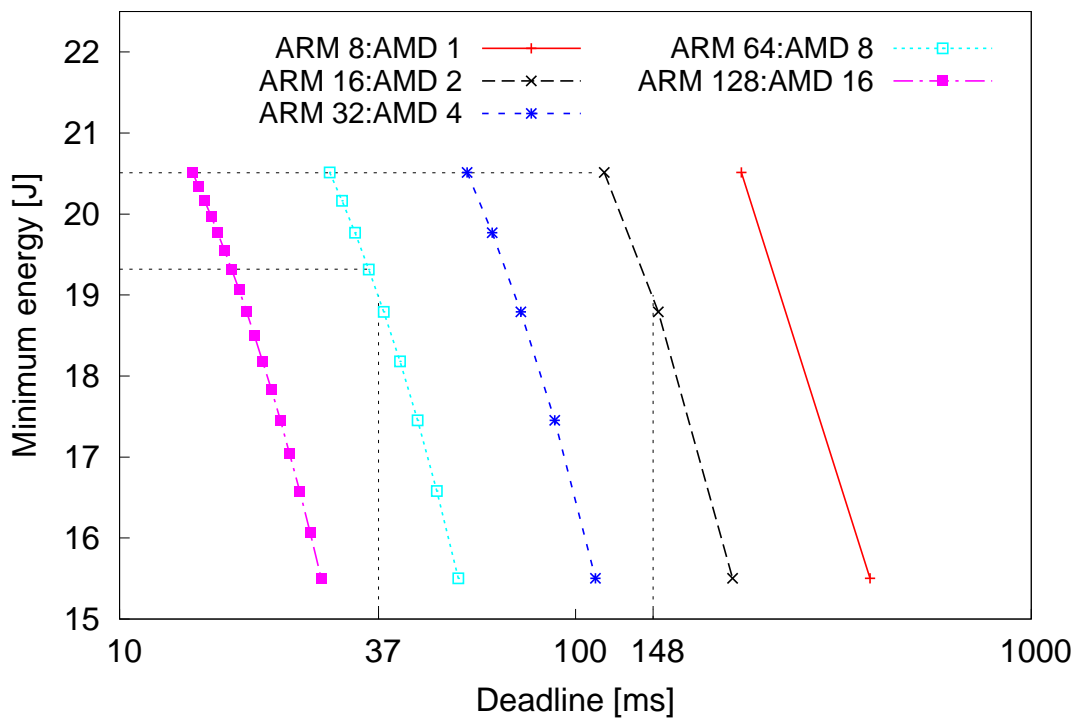


Figure C.4: Increasing cluster size for EP

This observation has an interesting implication. Given  $n$  jobs, it is better to schedule both on the same cluster than assigning each job to  $n$  clusters each with  $\frac{1}{n}$  of the total capacity. For example, consider four memcached jobs with a deadline of 165 milliseconds each, and 64 ARM and 8 AMD nodes, and two possible setups: (i) we can create four clusters, each with 16 ARM and 2 AMD nodes, or (ii) one large cluster with all nodes. In the first setup, Figure C.3 indicates that the configuration that meets the deadline incurs 19.8 Joules per job. In contrast, the configuration that meets a deadline of four times smaller (41 ms), incurs 19.6 Joules per job.

### C.3 Does Static Workload Allocation Suffice?

Our matching approach ensures that the two types of nodes in a heterogeneous mix finish servicing the job at the same time. This minimizes waiting time among the nodes, and thus eliminates idling during job service time. Because the two types of nodes have different execution rates, execution time of different nodes can be bounded by different components. For example, low-power nodes can be CPU-bounded, while the high-performance nodes can be I/O-bounded for the same application. A perfect matching analyzes all possible combinations of boundedness among nodes and chooses a workload distribution ratio based on the execution rates on that configuration. Furthermore, this is applied to all configurations on the sweet region. Thus, different points on a sweet region may require different workload allocation ratios.

However, in practice, few workloads change from one type of boundedness to another when the configuration is changed. For example, memcached is I/O bounded on almost all our configurations. Similarly, EP is CPU-bounded. Thus, it may be possible that the nodes operate at the same execution rate on all con-

figurations on the sweet region. In this case, a static workload allocation based on the differences among execution rates will result in a matching of the execution times.

In a static workload allocation, we change the distribution ratio until it is as close as possible to our matching distribution (i.e. mean squared error among the Pareto frontiers is minimized). Figures C.5 and C.6 plot the Pareto frontiers for

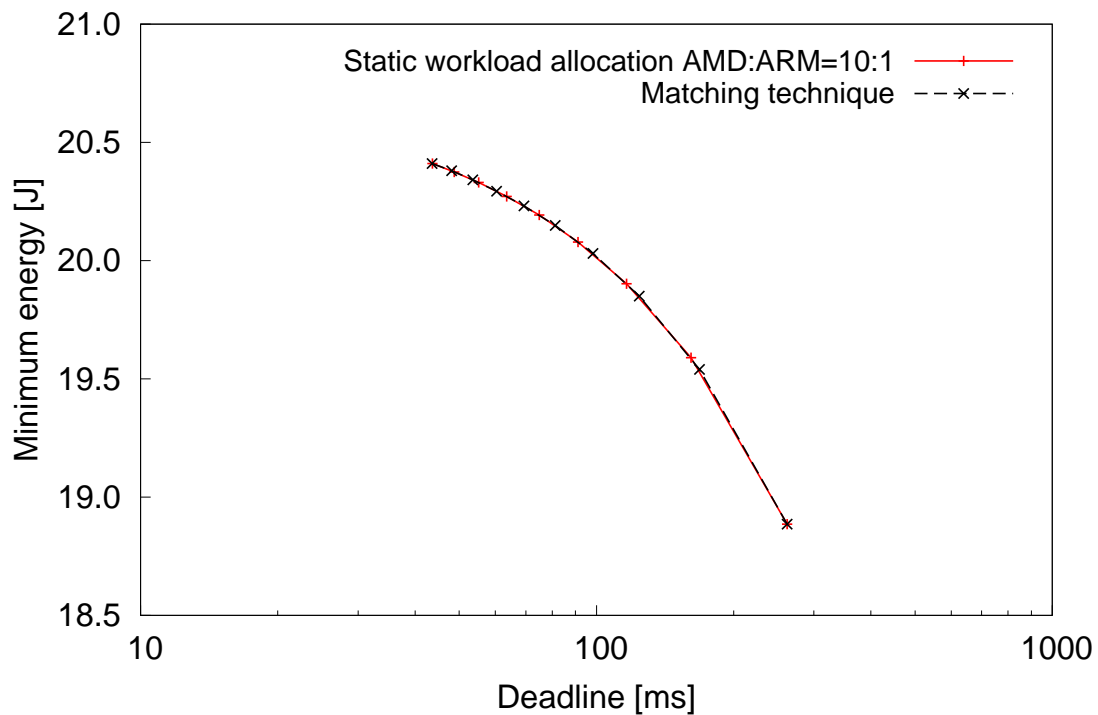


Figure C.5: Memcached static workload allocation

the static workload allocation that are closest to the distributions obtained using the matching technique, for 16 ARM and 14 AMD nodes. The number of ARM and AMD nodes are chosen from the power substitution ratios to meet a budget of 1kW. It is observed that Pareto frontier for a workload distribution of 10:1 among AMD and ARM is close to the Pareto frontier of the matched workload distribution. This is because memcached operates in an I/O bound region and the AMD node has 10 times more I/O bandwidth compared to the ARM node. For

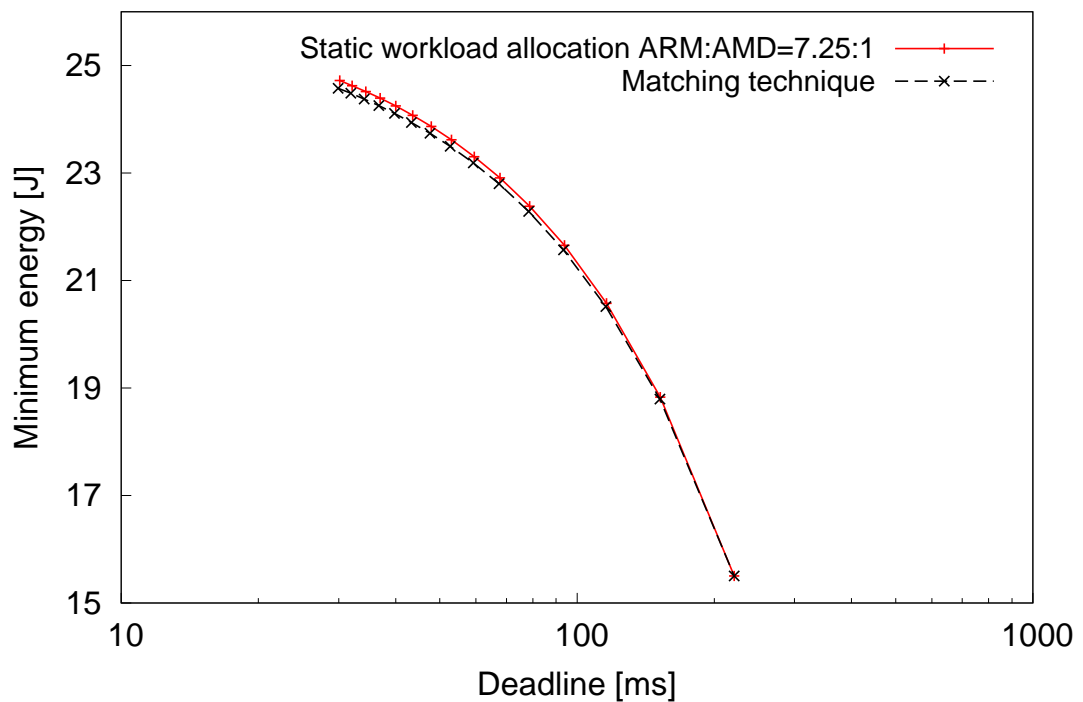


Figure C.6: EP static workload allocation

the EP program, a static workload distribution of 7.25:1 among AMD and ARM suffices to achieve minimum energies within 1% of the matching case.

**Observation 3:** *A static workload allocation that approximates the performance ratio of the two types of nodes results in a Pareto frontier similar to the matching technique.*

## C.4 Impact of Realistic Workloads on Heterogeneity

In this section we extend the analysis beyond online data-intensive applications, in a datacenter setup with variable cluster utilization. First we analyze the benefits the heterogeneity on a class of programs with imperfect parallelization. Second, we analyze the impact of cluster utilization on the response time, and on the energy-deadline Pareto profile of a job in a heterogeneous cluster.

### Sequential Fraction and Parallel Overhead

Although many datacenter applications show high level of scalability across nodes, there are still workloads without good parallel scalability. For such programs, the number of nodes that can be efficiently used depends on the fraction of work that is parallelizable. To capture this effect, we use Amdahl's sequential fraction, denoted by  $\alpha$ , to represent the fraction of work that cannot be parallelized among nodes. Furthermore, scaling across nodes may introduce both housekeeping overhead and contention for shared resources. These effects are captured using a coefficient for parallelization overhead  $k$ , that models the increase in service time when using multiple server nodes, relative to a single node. So far, our analysis assumed the ideal case where  $\alpha$  and  $k$  are equal to zero.

### Sequential Fraction

Previous work suggests that datacenter computing favors high-performance nodes, because the sequential fraction in Amdahl's law limits the usage of low-power nodes [79]. We analyze the Pareto frontier for a program that includes a portion of work that can only be executed on a single node. To model this



effect, we use the baseline memcached, but employ a work assignment strategy that assigns  $\alpha$  work units to a single high-performance node, and  $1 - \alpha$  work units to a heterogeneous mix. Furthermore, we assume that there always exists a high-performance node that executes the sequential fraction (i.e. even for a configuration 128 ARMs and zero AMDs, there is still one AMD node reserved for the sequential part).

We analyzed memcached-like workloads with sequential fraction  $\alpha$  ranging from 0.0 (fully parallel, identical to Figure C.1) to 0.5 (half of work is sequential, half is parallel). For each value of  $\alpha$ , we vary the mix of high-performance and low-power nodes for a power budget of 1kW. Figure C.7 shows an example

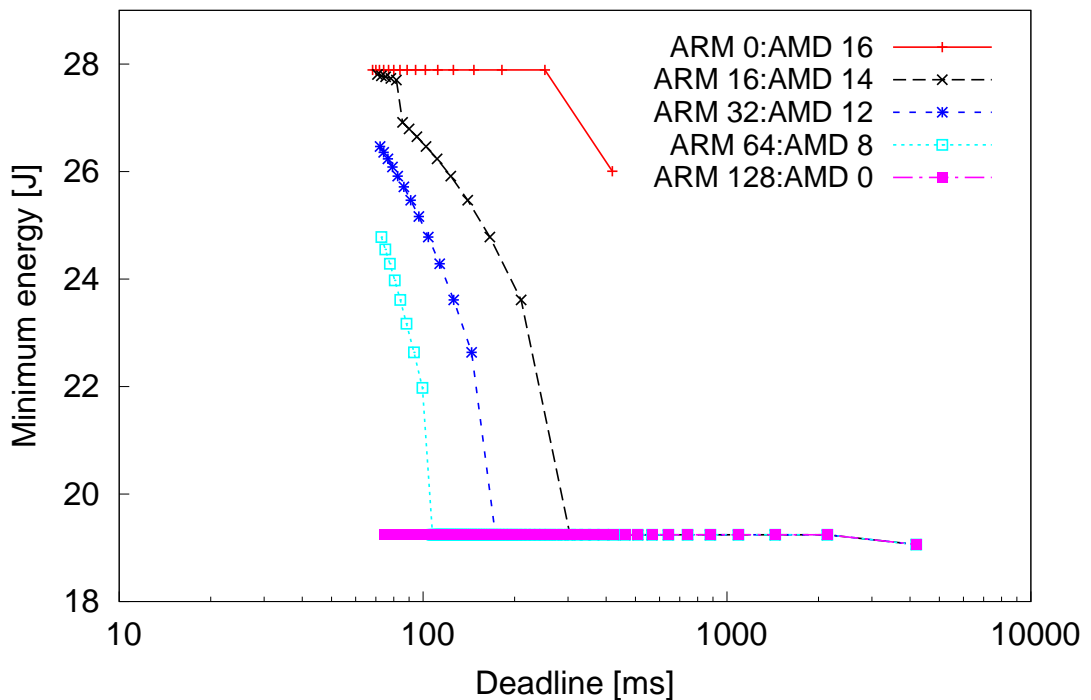


Figure C.7: Pareto frontier with sequential fraction  $\alpha = 0.1$

of this analysis for  $\alpha = 0.1$ . Counter to intuition, executing programs with large sequential fraction does not lead to an advantage for the high-performance nodes. Executing on any mix of high-performance and low-power nodes does not lead to

a significant execution time reduction. However, the energy decreases when more low-power nodes are added. Due to space constraints, graphs for other values of  $\alpha$  are omitted, but this effect is consistent across all values of  $\alpha$ . The only effect of increasing  $\alpha$  is an increase in the energy required for meeting the deadline, as larger sequential fractions imply more idleness in the system. The explanation behind this effect is that, given enough nodes, the execution time of the program is dominated by the sequential fraction. Thus, the only avenue for optimizing energy is to execute the parallel part as efficiently as possible. As long as there are enough low-power nodes to execute the parallel part faster than the sequential part, an execution using low-power nodes achieves the same execution time, but less energy than using only high-performance nodes. As Figure C.7 does not show

ARM \ AMD	0	1	2	4	6	8	10	12	14
0	-	419	251	146	111	94	83	76	71
1	<b>4194</b>	447	248	145	111	94	83	76	71
2	<b>2139</b>	426	244	145	111	93	83	76	71
4	<b>1090</b>	367	234	143	110	93	83	76	71
6	<b>741</b>	323	218	141	109	93	83	76	71
8	<b>566</b>	290	204	138	108	92	82	76	71
10	<b>461</b>	263	192	133	108	92	82	75	71
12	<b>391</b>	242	182	130	106	91	82	75	71
14	<b>341</b>	225	173	126	104	91	81	75	70
16	<b>304</b>	<b>210</b>	<b>166</b>	<b>123</b>	<b>102</b>	<b>90</b>	<b>81</b>	<b>75</b>	<b>70</b>

Energy Legend	19 J	21 J	23 J	25 J	27 J
---------------	------	------	------	------	------

Table C.1: Energy and service time with  $\alpha = 0.1$ 

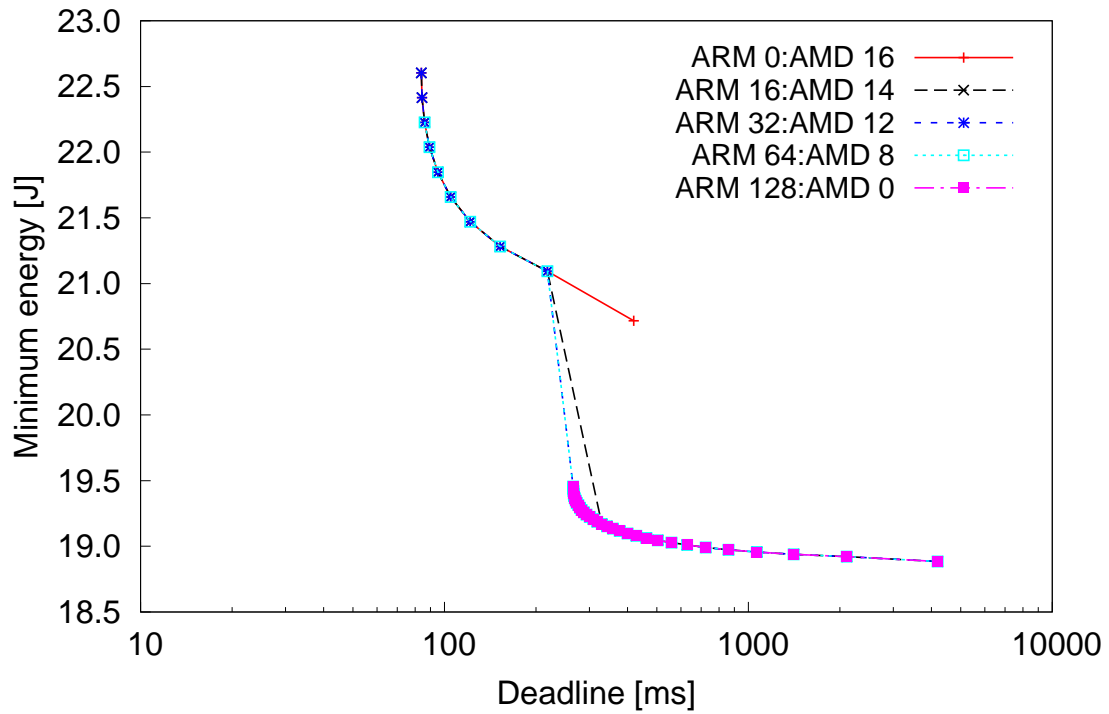
the configurations on the Pareto frontier, Table C.1 depicts possible service times achieved for combinations of ARM and AMD for a sequential fraction  $\alpha = 0.1$ . Each cell in the table indicates the execution time for that configuration, and the color represents the energy required to achieve that execution time. Configurations on the Pareto frontier are further highlighted in bold.

**Observation 4:** *Programs with a sequential fraction still exhibit a sweet region and save energy. However, for larger sequential fractions, the energy use increases because of more idleness in the system.*

Another interesting aspect is that the energy consumed for the single AMD node is lower than that on multiple AMD nodes. When the program is executed on more than one AMD node, there will be an idleness in the system, and this increases the energy spent to finish the job.

### Parallel Overhead

We study the effects of parallel overhead on the energy-deadline Pareto frontier. We consider a hypothetical program similar to memcached with respect to service requests, but with parallel overhead that increases the service time of each node proportional to the number of nodes utilized and relative to the service time on a single node. If  $T(n)$  is the execution time incurred on  $n$  nodes without parallel overhead then total service time with parallel overhead is  $T'(n) = T(n) + n \cdot k \cdot T(n)$ . We consider  $k \in [0.01 : 0.1]$ . Figure C.8 shows the Pareto frontier for an execution with parallel overhead  $k = 0.01$ . In contrast to the small effects of the Amdahl's fraction, even a small parallel overhead of  $k = 0.01$  significantly undermines the efficiency of using low-power nodes. Another effect of the parallel overhead is that the Pareto optimal configurations are different compared to executions without parallel overhead. Table C.2 shows the execution time achieved for each tuple  $(n_{ARM}, n_{AMD})$  with parallel overhead  $k = 0.01$ , for a mix of 16 ARM and 14 AMD nodes. The table shows that when parallelization overhead is considered, the Pareto frontier does not have configurations with more than ten AMD nodes. The parallel overhead compromises execution time when using large number of nodes,

Figure C.8: Pareto frontier with parallel overhead  $k = 0.01$ 

ARM \ AMD	0	1	2	4	6	8	10	12	14
0	-	419	<b>218</b>	<b>121</b>	<b>95</b>	<b>85</b>	<b>83</b>	85	88
1	<b>4194</b>	413	218	124	98	89	87	89	92
2	<b>2105</b>	397	219	128	102	93	91	93	96
4	<b>1065</b>	346	217	134	110	101	100	101	105
6	<b>724</b>	310	209	141	117	109	108	109	113
8	<b>557</b>	285	204	146	125	117	116	118	121
10	<b>461</b>	268	201	150	133	125	124	126	129
12	<b>399</b>	255	199	155	139	133	132	134	138
14	<b>358</b>	246	198	160	146	141	140	142	146
16	<b>329</b>	239	199	165	152	148	148	150	154

Energy Legend	19 J	21 J	23 J	25 J	27 J
---------------	------	------	------	------	------

Table C.2: Energy and service time with  $k = 0.01$ 

and favors optimal configurations with few high-performance nodes. Heterogeneous configurations including ARM nodes execute more than two times slower than those with only high-performance nodes.

ARM \ AMD	0	1	2	4	6	8	10	12	14
0	-	419	209	104	69	52	41	34	29
1	4194	405	206	103	69	52	41	34	29
2	2097	384	202	103	69	51	41	34	29
4	1048	325	192	101	68	51	41	34	29
6	699	281	176	99	67	51	41	34	29
8	524	248	162	96	66	50	40	34	29
10	419	221	150	91	66	50	40	33	29
12	349	200	140	88	64	49	40	33	29
14	299	183	131	84	62	49	40	33	28
16	<b>262</b>	<b>168</b>	<b>124</b>	<b>81</b>	<b>60</b>	<b>48</b>	<b>39</b>	<b>33</b>	<b>28</b>

Energy Legend	19 J	21 J	23 J	25 J	27 J
---------------	------	------	------	------	------

Table C.3: Energy and service time for ideal case

**Observation 5:** *Parallel overheads of a program significantly reduces the benefits of using low-power nodes.*

Comparing the Pareto optimal configurations of Tables C.1, C.2 and C.3 confirms that parallelism overhead is one of the main reasons why high-performance nodes are necessary for achieving good performance in data centers. However, a mix of high-performance and low-power nodes can significantly reduce the energy required to achieve the same execution time, even for programs with sequential fraction.

## C.5 Impact of Jobs Queueing Delay

So far we assumed that each job does not wait for other jobs inside a datacenter. Next, we extend the Pareto frontier to model job arrivals with waiting time.

We model the arrivals and departures of jobs to a datacenter using an M/D/1 queueing model. Jobs are assumed to arrive with inter-arrival time exponentially

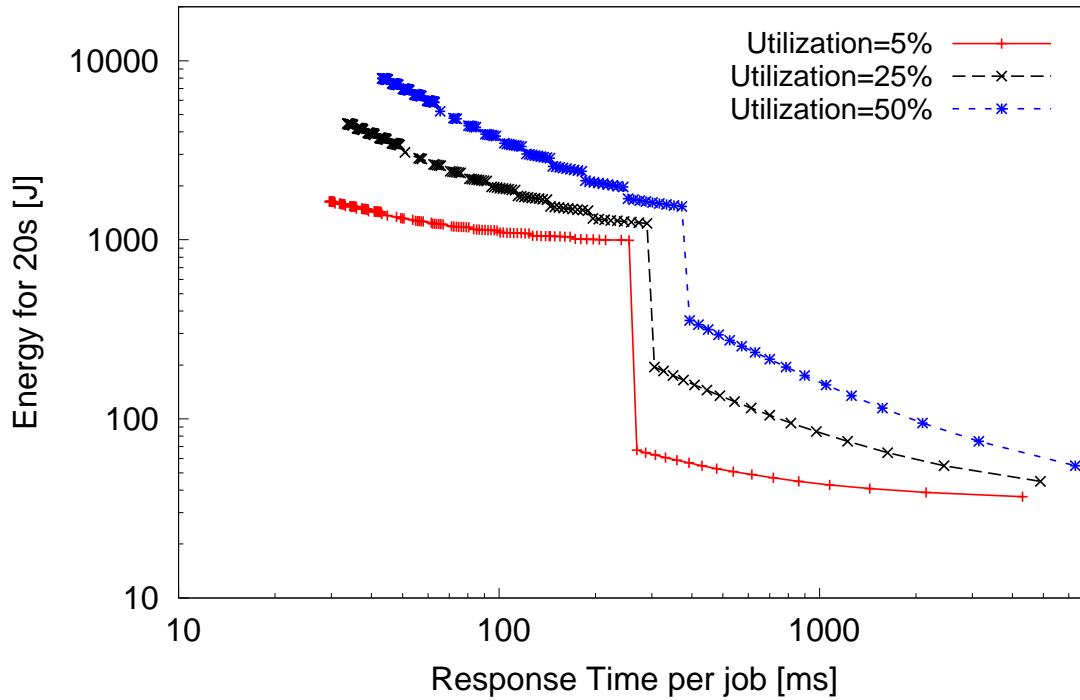


Figure C.9: Effect of job queuing delay on cluster utilization

distributed with parameter  $\lambda_{job}$ , and are queued in a dispatcher node until all the previous jobs have been serviced. The service time for a job is considered fixed, and modeled by our *matching* scheduling policy. According to the M/D/1 queuing model, the utilization of the cluster is  $U = T\lambda_{job}$ , where T is the service time of a job.

We analyze the effect of changing arrival rate by varying the arrival rate such that the utilization varies between 0 and 1. Figure C.9 plots in log-log scale the total energy consumed by a cluster of 16 ARM and 14 AMD nodes servicing multiple memcached jobs each with 50,000 requests, for an observation period of 20 seconds. We plot three profiles of utilization, corresponding to a tenfold increase in arrival rate. For a configuration point that does not use all 16 ARM and 14 AMD nodes, we consider the unused nodes as turned off. As arrival rate increases, the average waiting time in the dispatcher queue also increases. To meet

the same response time deadline, jobs need to be serviced faster, which requires a configuration with more high-performance nodes. Thus, as the utilization increases from 5% to 50%, the energy required to meet the same deadline increases almost by an order of magnitude.

However, Figure C.9 shows that the sweet region is still present, for all values of utilization. Unlike our previous analysis where we considered only energy incurred by job service time, the sweet region has a more complex shape and can be divided into two linear regions delimited by a sharp drop in the energy used. In the leftmost part of the sweet region, the configurations always include high-performance AMD nodes. Because AMD idle power is 45 watts, the idle energy use is considerable. In contrast, the rightmost part of the sweet region consists of configurations with only ARM nodes, which idle at less than 2 watts, thus incurring much lower idle energy.

When considering the idle energy and job queueing delay of a system, the energy reductions achievable by heterogeneous systems are much larger, spanning almost two orders of magnitude. As cluster utilization increases due to faster job arrivals, the energy savings are further amplified, but the minimal response time achievable is reduced.

**Observation 6:** *Energy savings achieved by the mix-and-match approach are amplified when cluster utilization increases.*

