Application Level Energy Measurements and Models for Hybrid Platforms with Accelerators

Kenneth O'Brien

UCD student number: 05538874

The thesis is submitted to University College Dublin
in fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science

School of Computer Science

Head of School: Professor Pádraig Cunningham

Research Supervisor: Alexey Lastovetsky

May 2018

Acknowledgements

I would first like to thank Dr. Alexey Lastovetsky for his supervision during this project, for accepting me into the Heterogeneous Computing Laboratory and for facilitating the beginnings of my career as a research scientist. I also extend my thanks to Prof. Adrian Ottewill for accepting me into the first offering of the Simulation Science Structured PhD Programme which gave me the fundamentals to transition into this field. I would like to thank Dr. Emmanuel Jeannot, Inria for endorsing my access to the GRID5000 infrastructure which was critical to this work.

I offer special thanks to Ravi Manumachu for his guidance and encouragement which got me over the line.

I am forever grateful to Michaela Blott, Kees Vissers, and Ivo Bolsens of Xilinx Inc for their support and direction.

I thank Adam Molloy for his encouragement over many lunches in UCD. I thank John Curran, my triathlon training partner throughout the PhD., bringing that essential work-life balance. I thank Muireann Lynch for her support and belief that I could successfully transition to a more mathematical field.

I thank my colleagues for fruitful collaborations: Christian Lalanne, Servesh Muralidharan, Michael Lysaght from ICHEC, Massimiliano "Max" Meneghin from IBM, Rizos Sakellariou and Ilia Pietri from the University of Manchester. Lorenzo Di Tucci and Marco Santambrogio from Politecnico di Milano.

This thesis would not be possible without the support and encouragement of my Xilinx colleagues: Baris Ozgul, Ronan Keryell, David Clarke, Lisa Liu, Giulio Gambardella, Peter McColgan, Cathal McCabe, Lisa Halder, Alexandre Isoard, Brian Colgan, Gianluca Durelli, Johannes De Fine Licht, Nicholas Fraser, and Yaman Umuroğlu.

I'd like to thank my colleagues in UCD's Heterogeneous Computing Laboratory: Kiril Dichev, Ashley DeFlumere, Zhong Ziming, Khalid Hasanov, Tania

Malik, Oleg Girko, Amani Al Onazi, Jean-Noël Quintin, Jun Zhu, Brett Becker, Semen Khokhriakov, Arsalan Shahid, Muhammad Fahad, Emin Nuriyev, and Hamidreza Khaleghzadeh.

I thank Edele Grey for her love, support and encouragement.

Finally, I thank my parents and brother Graham for their support.

For my parents Vincent and Mary, for providing encouragement, computers, and for helping me with my maths homework as a child.

Abstract

High Performance Computing is essential to continued advancement in many scientific and engineering fields. In recent years, due to the increasing scale of the platforms and the breakdown of laws which had long since supported rapid expansion, energy efficiency has emerged as a new design constraint on HPC platforms and applications. This constraint has increased the heterogeneity found in HPC nodes, seen in the form of higher CPU core counts and the adoption of specialised hardware accelerators. In an effort to address this new metric, models have been developed to capture and predict power and energy consumption, of application and systems.

In this thesis, we survey the state of the art in modelling and perform an evaluation of the existing methods. We then address the problem of accurate application energy and power measurement at multiple levels of granularity, including node, components and cluster level. We contribute multiple measurement tools targeted at researchers, and include case studies demonstrating their application, to facilitate standard methods in our field. Finally, we introduce a methodology for selecting the ideal accelerator device with respect to performance and energy efficiency to execute a given algorithm [1].

[1]This research is supported by the Structured PhD in Simulation Science which is funded by the Programme for Research in Third Level Institutions (PRTLI) Cycle 5 and co-funded by the European Regional Development Fund

Contents

List of Figures

List of Tables

List of Acronyms

BRAM Block Random Access Memory. 92, 93

CMOS Complementary Metal-oxide-semiconductor. 8

CPU Central Processing Unit. 9, 81

DDR3 Double Data Rate 3. 80

DDR4 Double Data Rate 4. 80

DSP Digital Signal Processor. 92, 93

EIST Enhanced Intel Speed Step. 8

FF Flip-flop. 92, 93

FLOPS Floating Point Operations Per Second. 2

FPGA Field Programmable Gate Array. 2, 75, 92

GDDR5 Graphics Double Data Rate 5. 25, 90, 91, 99

GPU Graphics Processing Unit. 2

HBM High Bandwidth Memory. 99

HDL Hardware Description Language. 92

HLS High Level Synthesis. 92

HPC High Performance Computing. 1, 63, 90

IC Integrated Circuit. 6, 7

ISA Instruction Set Architecture. 76, 91

kWh Kilo Watt Hour. 53

LUT Lookup Table. 92, 93

MW Mega-Watt. 7

OPS Operations per Second. 85

PMC Performance Monitoring Counter. 4, 13, 19, 21–25, 34, 36, 39, 44, 45

PSU Performance Supply Unit. 8

RAM Random Access Memory. 80

RTL Register Transfer Logic. 92

SIMD Single Instruction Multiple Data. 91

SMX Streaming Multiprocessor Architecture. 25, 90

SPIR Standard Portable Intermediate Representation. 83, 87

TDP Thermal Design Power. 74, 77, 99

Statement of Original Authorship

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the Title Page, and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

List of Publications

- A Survey of Power and Energy Predictive Models in HPC Systems and Applications, K O'Brien, I Pietri, R Reddy, A Lastovetsky, R Sakellariou: ACM Computing Surveys (CSUR) [3]
- Towards Application Energy Measurement and Modelling Tool Support, K O'Brien, A Lastovetsky, I Pietri, R Sakellariou: International Conference on Parallel Computing Technologies [4]
- A Semi-Automated Tool Flow for Roofline Analysis of OpenCL Kernels on Accelerators, S Muralidharan, K O'Brien, C Lalanne: Proceedings of the 1st International Workshop on Heterogeneous High-performance Reconfigurable Computing, H2RC [5]

Chapter 1

Introduction

1.1 Motivation

High Performance Computing (HPC) is the use of extremely powerful computing infrastructure, known as supercomputers to process large amounts of data or compute complex calculations in a reasonable timeframe. The primary applications of the field are weather prediction [6], molecular modelling [7], bioinformatics [8], physical simulations [9], and petrochemical exploration [10]. These applications have been driving the development of supercomputers possessing ever greater computational capability, however progress has hit new challenges in the form of physical constraints on scaling processor frequency for increased performance and power consumption of both computational hardware and cooling infrastructure.

A modern supercomputer is typically a distributed system, in which many commodity server computers known as *nodes* cooperate through a high speed communication network. These nodes, like all modern computers, are subject to Dennard scaling [11], which states that as the component transistors of the processing cores are reduced in size, power consumption scales down for a unit area of processor chip area. This scaling has allowed manufacturers to increase clock frequency and maintain a constantly stable power consumption, low enough to be dissipated by traditional cooling. The relation of frequency (f), voltage (V), capacitance (C), and switching activity (α) to power dissipa-

tion (P) can be seen in equation 1.1.

$$P = \alpha C V^2 f \tag{1.1}$$

As of 2006, this law has failed for new technology nodes and the resulting increases in power consumption and heat production has driven chip manufacturers to transition to multi-core CPUs, with minimal clock frequency increases. The move to multi-core processors has increased the heterogeneity of these systems and with it, an increase in programming difficultly. If the current trend continues, increasing the number of cores per CPU will increase the set of permutations of processor configurations, expanding the potential for optimisation. To explore this space, we will need advanced tools and models similar to those which exist for performance optimisation [12]. We require the means to measure application energy consumption at multiple levels of granularity, namely, the component, node, and distributed cluster level. As we will discuss later, these tools are not yet available.

In parallel to this multi-core revolution, computational scientists are using Graphics Processing Units (GPUs) which have demonstrated significant performance increases relative to optimised CPU implementations [13, 14, 15]. Multiple accelerators are now commonly found in modern heterogeneous nodes, including Intel's Xeon Phi, Nvidia and AMD's GPU and the emerging Field Programmable Gate Array (FPGA) based accelerators from Xilinx and Intel (formerly Altera). Each accelerator has its own advantages and disadvantages [16, 17], with each having differing compute capabilities, memory bandwidth, power consumption and cost. Recent efforts have focused on modelling and instrumenting these devices in isolation.

In HPC, the top performing systems are ranked by their performance achieved in running the LINPACK benchmark [18] as measured in Floating Point Operations Per Second (FLOPS). In a similar vein, as the electrical consumption of compute infrastructure has continuously risen [19], to rival the cost of that infrastructure [20], it was recognised in 2007 [21], that a ranking of supercomputer infrastructure by energy efficiency would be valuable. As such, energy efficiency is recognised by the Green500 [22] list, ranking the top sys-

tems in the world by energy efficiency rating as measured by performance per Watt.

1.2 Approach

Modern high performance computing infrastructure is opaque to researchers studying energy efficiency. In this thesis, we consider the problem of accurately measuring energy related data at multiple levels of granularity. By providing tools, we hope to establish repeatable experiments and methods, and by targeting commodity hardware and a large European scientific platform (GRID5000), we hope to remove the need for bespoke solutions commonly found in the literature. The combination should facilitate foundational research in this area.

Throughout this work we demonstrate our solutions by case study using relevant applications or benchmark suites.

For the purposes of our investigation we consider the extreme case of heterogeneity in our platform. The trend in supercomputing infrastructure design is to increase the number of processor cores and to augment the node with several accelerator devices. An exemplar of this construction is our own HCL node as seen in Fig 1.1 which we use in several experiments.

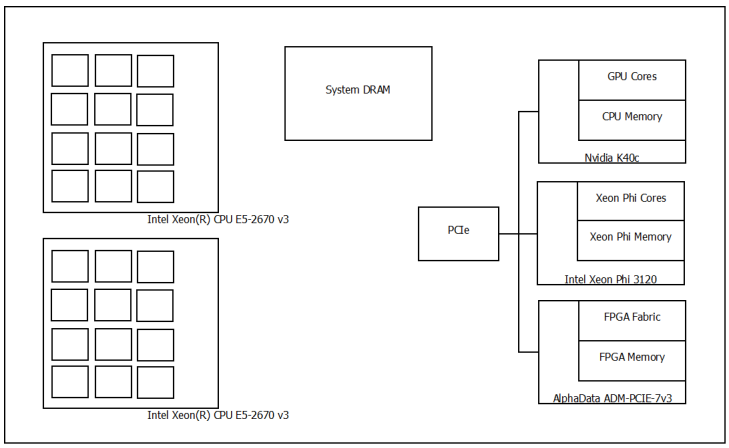


Figure 1.1: UCD HCL Heterogeneous Node

1.3 Contributions

The major contributions of this thesis are:

- A comprehensive survey of energy and power models and measurement methods for HPC platforms and applications, including evaluation and criticism of existing Performance Monitoring Counter (PMC) based models on multicore devices.
- Design of an energy measurement methodology at multiple levels of hardware granularity, including node level, cluster level and accelerator level.
- Implementations of this measurement methodology in the form of several measurement and modelling tools for applications at these granularities, providing meaningful statistical analysis not found in other tools.
- A Methodology for predicting which accelerator device is optimal in terms of performance or energy efficiency for executing an application. The method extends the Berkeley Roofline model and we provide a semi-automated toolflow implementation. This work includes a mapping of OpenCL programming model metrics to the Berkeley roofline model for multiple devices, including a novel FPGA accelerator.
- An evaluation of this methodology using multiple accelerators, including architecturally driven analysis for the performance of each accelerator.

1.4 Thesis Structure

The structure of this thesis is as follows: In Chapter 2, we describe the background and related work, detailing the state of the art in modern heterogeneous computing infrastructure, current tools and methods, giving context for our contributions. In Chapter 3 we study instrumentation, specifically we consider the problem of studying application energy efficiency on a modern HPC node, and we describe our solution and demonstrate its use in a case study

of its use on a wide variety of applications, under multiple configurations. In Chapter 4 we describe the use of accelerators in HPC, the role they play in energy efficiency and our solution for studying the energy efficiency on modern accelerators. In Chapter 5 we describe the problem of identifying how to select the appropriate accelerator device for an application and our modelling solution. Our solution is demonstrated by use of a case study of a data center application across multiple accelerators, including a novel FPGA based accelerator. Finally, we conclude with a discussion, claims and future work.

1.5 Relationship to Other Work

The work of Chapter 2 is the product of a collaboration with Dr. Ilia Pietri of the University of Manchester, UK. The merging of our background research in this field was the basis of the literature survey. The work was advised by our respective supervisors Dr. Alexey Lastovetsky and Dr. Rizos Sakellariou. The experimental component was designed and carried out as a collaboration between myself and Dr. Ravi Reddy, the postdoctoral researcher of the Hetergeneous Computing Lab at UCD.

With respect to Chapter 3, on node level measurements, my contributions were the design and development of the software *HCLEnergyAPI*, the work of instrumenting the NAS NPB software with *HCLEnergyAPI*, and the design and execution of the experiments. The project idea came from the previously mentioned collaboration with Dr. Ilia Pietri, who with her supervisor Dr. Rizos Sakellariou contributed improvements to the text of final paper. The work was advised by Dr. Alexey Lastovetsky.

Concerning the work of Chapter 4, all contributions were my own. I designed and developed the tool *HCLpower*. I also carried out the design and execution of the experiments on the GRID5000 infrastructure. The work was advised by Dr. Alexey Lastovetsky.

Finally, the work of Chapter 5, was carried out during my six month long industry internship project, at Xilinx Inc., Ireland. My contributions were the development of the optimised hardware and software components of the FPGA

system using experimental tools. I collaborated with Dr. Servesh Muralidharan and Mr. Christian Lalanne, both performance engineers from the Irish Center of High End Computing (ICHEC) on the implementation of the GPU and Xeon Phi accelerator software components. The experimental design and execution, was a collaboration between myself, Dr. Muralidharan, and Mr. Lalanne. The modelling component was a collaboration between myself and Dr. Muralidharan. The work was supervised by Michaela Blott, my internship supervisor at Xilinx, with prior project approval of Dr. Lastovetsky.

Chapter 2

Background and Related Works

In this chapter the background to our research is surveyed [3], giving all necessary foundations for the rest of this thesis.

2.1 Definitions

Throughout this thesis there is reference made to both power and energy. For clarity the definitions of these terms and their units are stated. Power is the rate of consumption of electricity per second as measured in units of Watt (W). Energy is the amount of work carried out as measured in units of Joules(J). As power is a rate over time measured in seconds (s), all three terms can be related by equation 2.1.

$$E_{(J)} = P_{(W)} * t_{(s)} \tag{2.1}$$

There are several related laws which must be considered with respect to performance and energy efficiency. They are:

1. **Moore's Law** - the number of transistors in Integrated Circuits (ICs) doubles every two years [23].

2. **Dennard scaling** - as transistor are made smaller, the power density of a processor made of these transistors will remain constant, causing

power use to remain proportional to the area of the IC as voltage and current scale down with length of the transitor [11].

3. **Koomey's Law** - The number of computations per Joule doubles every 1.57 years [24].

2.2 Measurement

In this section the various methods used to obtain power measurements are discussed.

2.2.1 Supercomputer Scale

Supercomputers draw enormous amounts of power, with the top 10 fastest supercomputers drawing between 1.3 and 17.8 Mega-Watts (MWs) [25]. At this scale power may be measured through the datacenter power distribution system as a single value, but as supercomputer infrastructure is often composed of *N* identical nodes, the total power $\bar{P}(R_{max})$ consumed by a supercomputer is typically determined by direct measurement of a single node's power $\bar{P}_{node}(R_{max})$ while running the linpack benchmark [18], as is advised for submissions to the GREEN500 list [22]. The total power consumed is then dictated by equation 2.2.

$$\bar{P}(R_{max}) = N * \bar{P}_{node}(R_{max})[26] \tag{2.2}$$

Though large scale measurement at the node level [27] and even to a multi-site grid [28] have begun to emerge, there is no standard way to measure or access total power across a supercomputer. In this thesis, we focus on smaller granularity of multiple nodes, single node and single accelerators.

2.2.2 Node Scale - Direct Measurement

The primary means of direct measurement is to use a clamp meter, which though affordable offer moderate accuracy, or inline power meters (Watts Up

Pro for example) which may be expensive but accurate, to measure the mains power entering the Performance Supply Unit (PSU) of the node. Another method supported by some commodity servers is the provision of power consumption data by the server's internal maintenance interface, through a network protocol such as IPMI.

2.2.3 Component Scale - Direct Measurement

To measure individual components of a node, such as an accelerator, a shunt resistor may be used in conjunction with a PCIe riser card to intercept the power rails of the device [29]. Though intrusive and not scalable, this method offers high accuracy as it is transparent to the device, and so the state is not altered by measurement.

2.3 Developments in Energy Efficient Hardware

2.3.1 Processor level

Several advancements have been made in processor design to aid in energy efficiency. The being dynamic frequency scaling, which enables a processor to reduce the power it consumes by lowering its clock frequency. As frequency is a component of the power consumption equation of Complementary Metal-oxide-semiconductor (CMOS) (Equation 1.1) systems, processor vendors added the ability to reduce frequency in discrete steps. When frequency is reduced, voltage is also typically lowered, leading to further power savings. Each of the possible frequency/voltage combinations are given labels known as P-States. P_0 is the maximum frequency of the processor and subsequent lower states $P_1 ... P_N$ are given by stepping back the frequency by a vendor defined step.

Driven by extending operating time on battery powered laptops, AMD introduced its implementation "PowerNow!" in 2000, followed by Intel's implementation called Enhanced Intel Speed Step (EIST) in 2005. These technologies were later advanced and ported to desktop and server platforms. The fre-

quency of the processor is controlled by the operating system the processor is executing. There are multiple strategies for selecting which frequency should be set at a given time for a given core. These strategies are implemented as various user selectable "governors".

In addition to scaling frequency and voltage, processor designers have added functionality to processors to control which of the multiple components of the processor is allowed to receive power and thus be in an active state. As with P-States, the possible configurations are discrete states, known as C-States. The default state, in which the processor is functioning with all components active is known as C_0. Lower states, such as C_1 means the processor is not currently executing instructions, but could return to C_0 immediately. Lower states begin to disable parts of the processor such as deactivating the caches or stopping the clock signal entirely (a technique known as clock gating) to drastically reduce the power consumed.

Multicore

Approaches to processor design at the core level have also been developed with benefits to energy efficiency. Driven by the collapse of Dennard scaling, Central Processing Units (CPUs) with multiple cores were mass produced from 2005.

Several novel CPU designs have been produced to increase performance and power efficiency. One such example is the heterogeneous ARM "big.LITTLE" architecture [30], in which small, slower clocked cores are coupled with larger, faster more power hungry cores. The rationale behind this approach is that most of the time, the small cores will be sufficient to deliver reasonable performance whilst drawing a small amount of power, but when more complex operations or heavier workload is presented, the small cores power off and the large cores power on, with the overall effect of reducing power consumption but without compromising performance when it is needed. The origin of ARM processors as embedded processors with power consumption an architectural concern, has led the company to produce server class processors [31] as this property is now valuable in datacenters, showing en-

ergy efficiency benefits, using only 33% of the energy compared to similar x86 servers [32].

2.4 Models

There have been many efforts to model power consumption of nodes based on summation of the power drawn by individual components. Some efforts have focused on CPU with single or multiple cores [33], [34], [35], [36], [37], [38], [39], [40], whilst others have focused on accelerator devices such as GPU[41], [42], [43], [44], [45], [46].

These models are primarily linear regression based, and attempt to correlate the power consumption of components with some metric of activity or utilisation such as device performance counters. Others try to capture non-linearity through methods such as artificial neural networks [45], random forests [42] and fuzzy wavelet neural networks [46].

Compute devices are the primary focus of these efforts as they are the most power hungry compared to storage and networking components.

In this section the existing models are surveyed and classified, extracting the core common features with the aim to highlight any shortcomings.

2.4.1 Classification of Models

In table 2.1 the key features of the models are outlined in this survey. Bibliographical reference are used as the name of the model. The attribute "Level of abstraction" characterises how the model does or does not capture the hierarchical nature of the device its modelling.

1. *Linear Independence* - All the components considered are assumed to operate independently. The model's output is the summation of all contributions.

2. *Linear Dependence* - The models are constructed taking into account the shared physical resources, such as caches. The model's output is the summation of all contributions.

Table 2.1: Key characteristics of the power and energy models.

Model Characteristic		**Description**
Model		Name of the model and year of publication
Parameters		Parameters of the model
Level of abstraction		How the model captures the hierarchical nature of modern processor architectures?
Type of power		Is the predicted power instantaneous or average?
Is energy predicted?		Is Energy predicted? If yes, how?
Decomposition		Does the model provide per-component power and energy breakdown?
Accuracy of power prediction	**Accuracy of dynamic power prediction**	The maximum percent error in the dynamic power prediction calculated from the total power and static power consumptions reported by the authors
	Accuracy of total power prediction	The maximum percent error in the total power prediction reported by the authors
Accuracy of energy prediction		The maximum percent error in the energy prediction reported by the authors
Implementation Complexity (effort-week/effort-month/effort-year (EW/EM/EY))		The implementation effort required to build the model
Portability		Is the model portable to next-gen processors in the same architecture space?

3. *Nonlinear Independence* - All the components considered are assumed to operate independently. The model's output is a non linear combination of its parts.

4. *Nonlinear Dependence* - The models are constructed taking into account the shared physical resources, such as caches. The model's output is a non linear combination of its parts.

$$P_{instantaneous} = \lim_{\Delta T \to 0} P_{avg} = \lim_{\Delta T \to 0} \Delta E / \Delta T = dE/dt \tag{2.3}$$

The attribute "Type of power" specifies if the power predicted by the model is average power for an execution, or instantaneous power. The average power is defined as $\Delta E/\Delta T$ where ΔE is the total energy consumed and ΔT is the execution time of the application. Instantaneous power is defined in equation 2.3. For models in this survey, instantaneous power is representative of a one second interval as that is the resolution of the measurement devices.

Power can be further broken down into two components, static and dynamic. Static power is power drawn when the device is not performing any work. Dynamic power is power drawn by the switching activity of the components due to activity. Static power is also referred to as base power, leakage power or idle power. Some authors [47] differentiate base power from idle power, where base power is minimum power drawn when all processor cores are in C-State C_0, i.e. active, and idle power is the minimum possible power drawn when the processor cores are in low C-State of C_3 to C_6.

From an application point of view, we define dynamic and static power as the power drawn when the application we are studying is or is not executing respectively. From a component point of view, accelerators for example, to determine the static power draw, the total system power drawn by a node is measured, then the accelerator is added and the measurement is taken again. The difference in power consumption is the static power. To determine dynamic power consumption, an application is measured and the static power value is subtracted. In this work, when reference is made to power, it is defined as the sum of dynamic and static power, unless otherwise stated.

The attribute "Is energy predicted?" specifies whether or not energy is predicted by the model and if so, how? There are two possibilities. First, the model could predict performance and power separately, then compute energy by their product. Secondly, energy could be predicted explicitly.

The error of the prediction is given by equation 2.4. *Actual value* is recorded by a power meter, and *Predicted* value is given by the model.

$$\text{Percent error of prediction} = \frac{|\text{Actual value - Predicted value}|}{\text{Actual value}} \times 100 \quad (2.4)$$

The attribute "Accuracy of energy prediction" is the maximum error of the model's prediction as reported by the authors, in Joules.

The attribute "Implementation complexity" evaluates the effort required by a researcher in our field to construct the model. It is an attempt to gauge the difficulty of build some of the models given, the more difficult models being artificial neural networks.

The attribute "Portability" conveys the applicability of a model to make accurate predictions for devices of subsequent architectures. For instance, for Nvidia GPU, a model for Kepler architecture devices, should be usable for subsequent Maxwell and Pascal devices. It is not expected that models for CPU would be applicable to GPU devices, so we do not consider this. Portability in this work, rather measures how resistant a model is to large architectural changes such as architectures deprecating performance counters and those based on specific performance counters.

2.4.2 Power and Energy Models for CPU

The first models for power consumption focused on CPU components as they are from an era before hardware accelerators were commonplace. Subsequent models included other system components such as memory. These models are presented in table 2.2. One of the first models was developed by Bellosa [33] and is based on PMCs. Performance events such as integer and floating point operations, as well as memory requests were used to build

a linear model of power consumption of the application. To build the model, multiple synthetic benchmarks are executed and counters are correlated to fit the model.

Isci [34] proposed a model which captures the power contributions of individual components of the CPU, and determines 22 performance counters which are representative of these components. Benchmarks were executed and the total power of the system was measured with a multimeter.

Table 2.2: Power and Energy Models. '-' indicates not reported.

Model	Level of Abstraction	Type of Power	Is Energy Predicted?	Accuracy of Power Prediction		Accuracy of Energy Prediction	Implementation Complexity	Portability
				Dynamic	Total			
CPU								
[33]	Linear Independence	Average	Explicit	-	-	-	1 EM	Yes
[34]	Linear Independence	Instantaneous, Average	No	-	-	-	3 EM	No
[35]	Linear Independence	Instantaneous	No	-	2.7%	-	1 EW	No
[36]	Linear Independence	Instantaneous	No	-	15%	-	1 EW	No
[48]	Linear Independence	Average	Power $\times$ Timing	-	24.5%	-	1 EM	No
[37]	Linear Independence	Instantaneous	No	-	-	-	1 EW	No
[37]	Non-linear Independence	Instantaneous	No	-	-	-	1 EW	No
[49]	Linear Independence	Instantaneous	Yes	-	-	4%	1 EW	Yes

[38]	Non-linear Independence	Instantaneous	No	-	-	-	1 EW	No
[39]	Linear Independence	Instantaneous	No	-	9%	-	1 EM	Yes
[50]	Linear Dependence	Average	No	-	4%	-	1 EM	Yes
[51]	Linear Independence	Average	No	14%	10.15%	-	3 EM	Yes
[40]	Non-linear Independence	Instantaneous, Average	No	-	14.1%	-	3 EM	No
GPU								
[41]	Linear Independence	Average	Power $\times$ Timing	18%	8.94%	-	3 EM	No
[44]	Linear Independence	Average	No	-	23%	-	3 EM	Yes
[42]	Non-linear Independence	Average	No	12.95%	7.77%	-	3 EM	Yes
[52]	Non-linear Independence	Average	No	-	4.34%	-	3 EM	Yes
[43]	Linear Independence	Average	No	24%	12%	-	1 EM	Yes

[45]	Non-linear Independence	Average	Power × Timing	-	2.1%	11.02%	3 EM	Yes
[53]	Non-linear Independence	Average	Yes	15.14%	12.8%	-	3 EM	Yes
[46]	Non-linear Independence	Average	Power × Timing	-	6%	-	3 EM	Yes
Intel Xeon Phi								
[54]	Linear Independence	Average	Explicit	-	-	5%	3 EM	No
HPC Applications								
[55]	Linear Independence	Average	No	-	-	-	3 EM	Yes
[56]	Linear Independence	Average	Power × Timing	-	5.6%	-	1 EM	No
[57]	Non-linear Independence	Average	Explicit	-	5.5%	7%	3 EM	Yes
[58]	Linear Independence	Average	No	-	4.94%	-	2 EW	Yes
[59]	Linear Independence	Average	Power × Timing	-	-	1.41%	1 EM	Yes

[60]	Linear Independence	Average	Power × Timing	-	-	-6.82%	2 EM	Yes
[61]	Linear Independence	Average	No	-	10%	-	3 EM	Yes
[62]	Linear Independence	-	Explicit	-	-	15%	2 EW	Yes

$$TotalPower = \sum_{i=1}^{22} Power(C_i) + BasePower \tag{2.5}$$

$$\begin{aligned} Power(C_i) = {} & AccessRate(C_i) \times ArchitecturalScaling(C_i) \\ & \times MaxPower(C_i) + NonGatedClockPower(C_i) \end{aligned} \tag{2.6}$$

The parameter, $BasePower$, is the base power consumption of the Pentium 4 platform used in their experiments. The parameter, ($ArchitecturalScaling(C_i)$), is a conditional clock power factor used to model the non-linear behaviour of some issue logic units. The values of the parameters, $MaxPower(C_i)$ and $NonGatedClockPower(C_i)$, are obtained for each of the units using physical areas on the die and training benchmarks.

Lee [48] used PMCs to build a regression model of power in two steps. First they derived a baseline power estimation, then they refined it with further sampling. They reported a median error rate of 4.3% and a maximum error of 24.5%.

Node power was modelled linearly by [35] and its parameters are utilization of CPU, disk, and network as seen below,

$$P = C_{base} + C_1 \times U_{CPU} + C_2 \times U_{Disk} + C_3 \times U_{Net} \tag{2.7}$$

where C_{base} is the base power consumption of a node and the coefficients C_1, C_2, and C_3 for power consumptions of CPU, disk, and network, respectively are determined using several benchmarks. Total power measurements and utilization data for CPU, disk, and network were collected using a multimeter to compute the least-squares fit and to determine the coefficients of the model. Reported average and maximum prediction errors of the models were 1.3% and 2.7% respectively.

A similar but more complex power model (Mantis) was proposed in [36], adding memory access to the model. It is formulated as:

$$P = C_{base} + C_1 \times U_{CPU} + C_2 \times U_{Mem} + C_3 \times U_{Disk} + C_4 \times U_{Net} \tag{2.8}$$

The models were calibrated for two server systems using idle runs and

different configurations of Gamut [63] to emulate applications with varying resource needs. Several benchmarks (SPECcpu2000, SPECjbb2000, and SPECweb2005 suites) [64] and STREAM benchmark [65]) were used in the evaluation.

Fan [37] proposed a linear model of power based solely of single core CPU utilisation.

$$P_{CPU} = P_{base} + (P_{max} - P_{base}) \times (U/100) \tag{2.9}$$

Here, P_{max} represents the power consumption at maximum utilization. The parameter, U, signifies the utilization of the processor. They refine these models further as follows:

$$P = C_{base} + C_1 \times (2 \times U_{CPU} - U_{CPU}^r) \tag{2.10}$$

where an empirical term is added to increase the accuracy of the model. The tuning parameter r is obtained during calibration using a model similar to Mantis [36].

Lewis [49] proposed a linear regression energy model as seen below:

$$E = a \times (E_{CPU} + E_{DRAM}) + b \times E_{em} + c \times E_{Support\_Chipsets} + d \times E_{HDD} \tag{2.11a}$$

$$E_{HDD} = P_{spin-up} \times t_{su} + P_{read} \times \sum N_r \times t_r + P_{write} \times \sum N_w \times t_w + P_{base} \times t_{base} \tag{2.11b}$$

$$E_{em} = \sum P_{fan} \times t_{ipmi-slice} + \sum P_{optical} \times t_{optical} \tag{2.11c}$$

E_{CPU} is the energy consumption of the dual-core AMD Opteron processor. It is predicted from four key contributors. These are traffic on the HyperTransport bus [66], L2 cache misses, CPU core temperatures, disk read and write bandwidths, and ambient temperatures. E_{DRAM} is the energy consumed by the DRAM banks. E_{em} is the energy consumption of the cooling fans and optical drives. E_{HDD} is the energy consumption of the disk predicted from four components: 1). $P_{spin-up}$, the power consumption to spin the disk to full rotation and t_{su} is the time taken for the spin-up, 2). P_{read} is the power consumption

to read kilobyte of data from the disk, 3). P_{write} is the power consumption to write kilobyte of data to the disk, and 4). P_{base} is the base power consumption of the disk [49]. They reported a worse-case prediction error of 4% for common processor benchmarks.

Energy profiling for applications using CPU and disk activity is the subject in [67]. The profiling tool consists of a workload manager that triggers the measurement process and event tracing related to OS operations, such as CPU and disk I/O usage, the event logger for event logging using Windows Xperf [68], and the energy profiler that correlates resource usage with the event traces and profiles application energy usage across the various resources.

Rivoire [69], [70] studied and compared five full-system real-time power models using a variety of machines and benchmarks. Four of these models are utilization-based whereas the fifth includes CPU PMCs in the model parameter set along with the utilizations of CPU and disk. They reported that PMC-based model is the best overall in terms of accuracy since it is able to account for majority of the contributors to system's dynamic power (especially the memory activity). They also questioned the generality of their PMC-based model since the PMCs used in their model parameter set may not have the same essence across different architectures (Intel, AMD).

Wang[38] proposed a linear model to predict power consumption of a server based on CPU utilisation based on the work in [67]:

$$P = C_{base} + C_1 \times U_{CPU} + C_2 \times U_{I/O} \tag{2.12}$$

Here, in addition to already described parameters, $U_{I/O}$ the I/O bandwidth in MB/sec, and C_1, C_2 the coefficients of the model. A WattsUp power meter is used to measure overall power consumption of the servers and to calibrate the model.

Basmadjian [39] constructed a power model of a server as a summation of power models of its components, the processor (CPU), memory (RAM), fans, and disk (HDD). Based on the model evaluations on tower and blade servers, the authors report maximum prediction error rates of 8% and 9% for tower servers and blade servers respectively. This model is presented in detail in

Appendix A.2.1.

Bertran [71] presented a power model that provides per-component power breakdown of a multicore CPU. It can be described as follows:

$$P_{total} = \sum_{j=1}^{j=cores} (\sum_{i=1}^{i=ncomponents} AR_{ij} \times P_i) + P_{base} \quad (2.13)$$

The parameter AR_{ij} is the activity ratio of component i in core j. The dynamic power consumption of a component i in core j is given by the product, $AR_{ij} \times P_i$. The parameter, $ncomponents$, represents the number of components accommodated in their model. They reported their model can give a per-component power breakdown. SPECcpu2006 benchmarks [64] were used to validate the model, with average prediction errors between [1.89-6]% and a maximum error of 10.15%.

Bircher [40] proposed an iterative modelling procedure to predict power using PMCs. They used PMCs that represent activity of various system components such as GPU, IO controllers and memory subsystems. The highest error reported is 14.1% for the memory controller and the average error reported is less than 9% per subsystem.

Basmadjian [50] reported that summation of power consumptions of all active cores to derive the total power consumption is inaccurate and demonstrated a model which takes into account resource sharing in their power prediction model for multicore processors. They reported a maximum prediction error of 5%.

Wang [72] presented an energy consumption model using three parameters, which are the concurrency level of the workload, the average power dissipation of a thread, and the total time taken to execute the workload. Liu [73] presented a method to predict the energy consumption of a task executing in a multicore with several other tasks running simultaneously. Their method used the resource utilization and occupancy of a task to predict its energy consumption. On a 32-core multicore, they report an average and maximum errors of 4% and 9% respectively.

Several other research efforts used a PMC approach to model power con-

sumption. Gurumurthi [74] used analytical power models for CPU, memory, and disk in their power simulator called SoftWatt to predict power consumptions of applications and OS services. Li [75] proposed power models for the operating system (OS) based on their observations of strong correlation between instructions per cycle (IPC) and OS routine power. Singh [76] developed per-core power models based on multiple linear regression using PMCs. Powell [77] used a linear regression model to estimate activity factors and power for a large number of micro-architectural structures using a small number of PMCs. Goel [78] derived per-core power models using PMC values and temperature readings. Roy [79] proposed an energy model for an algorithm, which expresses the energy for an algorithm as a weighted linear combination of the time complexity of the algorithm and the number of "parallel" accesses to the memory. Spiliopoulos [80] proposed linear regression models that predict power consumption for any DVFS voltage and frequency combination supported by a computing platform.

McCullough [81] evaluated the competence of predictive power models for modern node architectures and show that linear regression models show prediction errors as high as 150%. They suggested that direct physical measurement of power consumption should be the preferred approach to tackle the inherent complexities posed by modern node architectures. As discussed in 2.2, direct measurement requires instrusive equipment, with additional support from a datacenter operator. Dargie [82] used the statistics of CPU utilization (instead of PMCs) to model the relationship between the power consumption of multicore processor and workload quantitatively. They demonstrated that the relationship is quadratic for single-core processor and linear for multicore processors.

Finally, hardware vendors now provide software interfaces to measure and control power and energy consumption of their processors, originally for system adminstration. However, due to the disparate capabilities and the non-uniformity of these interfaces, there is a real need for standardization to facilitate development of supporting infrastructures and tools. PowerAPI [83] was the first large-scale effort in this direction. PowerAPI is an interface for standardizing power and energy measurement and control for wide range of

systems spanning desktops and datacenters to large-scale HPC systems.

Some key observations follow from the analysis of these models:

- The models by Econoumou [36] and Basmadjian [39] have high prediction error than Heath [35] even though these models are more comprehensive by accounting for the power consumption from *RAM*.
- There is a noticeable difference between the power prediction accuracies of the models Economou [36] and Basmadjian [39], even though they both take into account all the major resource utilizations (*CPU*, *RAM*, *Disk*, *NIC*).
- The model by Basmadjian [39] has a higher prediction error than the model [40], which employs a complex PMC-based approach to model power consumption of all the architectural units of a CPU.

2.4.3 Power and Energy Models for GPUs

Driven by their immense computational capability, GPU are commonly found in supercomputing platforms [25]. In this section we survey the prominent GPU models, summarised in table (2.2) shows the key features of the GPU models.

Rofouei [84] used a linear model to calculate the energy consumption of a GPU in their server from real-time energy measurements. The linear relationship is presented below:

$$E_{gpu} = t_{gpu} \times (P_{avg-gpu} + P_{base-cpu}) + E_{transfer} \tag{2.14}$$

where $t_{gpu}, P_{avg-gpu}, P_{base-cpu}$, and $E_{transfer}$ denote the time of execution of the application on the GPU, average power consumption of GPU, base power consumption of CPU, and the energy consumption of data transfer between CPU and GPU.

One of the first comprehensive models developed for a GPU architecture was by Hong [41]. The GPU power consumption in their prediction model is modelled similar to the PMC-based unit power prediction approach of Icsi et al. [34]. Their model is presented in detail in Appendix A.2.2. In their

model, the power consumption is calculated as sum of power consumptions of all the components composing the Streaming Multiprocessor Architectures (SMXs) and Graphics Double Data Rate 5 (GDDR5) memory. They reported that the prediction error for the total power consumption is 8.94% and the average energy consumption savings are 10.99% with a NVIDIA GTX280. The main factor hindering the portability of this model is that it requires detailed architectural information and contains a large set of parameters.

Nagasaka [44] presented a GPU PMC based approach for the Nvidia GTX 285. They reported an average error of 4.7% and a maximum error of 23% in prediction of total power consumption.

Chen [42] used linear regression tree and random forest methods in their models to predict GPU power consumption. The random forest method is used to select the predictors that are the dominant contributors to the power consumption. They also used Nvidia GTX 280 GPU. They report an average percentage error (PE) of 7.77% for the total power consumption.

Kiran [43] proposed an analytical model to estimate activity factors and power for micro-architectural structures on GPUs. The key difference from the model [41] is that only two parameters, execution time and load rate, are used to estimate unit-level dynamic power of a GPU. Their model can be described as follows:

$$Total\_power\_consumption = DynamicPower + BasePower \quad (2.15a)$$

$$DynamicPower = \sum_{i=1}^{n}(N_{SM,i} \times P_i \times U_i) + B_i \times U_i \quad (2.15b)$$

where n is the number of architectural components, $N_{SM,i}$ is the number of streaming processors utilizing an architectural component, P_i, B_i, and U_i are the dynamic power consumption, base power consumption, and utilization of the architectural component i respectively. A unit is defined as an architectural component such as a floating-point unit (FP), shared memory (Shared), or global memory (GlobalMem). A micro-benchmark is designed for each architectural unit stressing the unit to obtain its P_i and B_i values. The utilization rates U_i of the units are obtained from the application execution. The Nvidia

Fermi C2075 GPU and MAGMA kernels are used for model evaluation in their experiments. The maximum error in predictions for the total power consumption is reported to be 12%.

Song [45] proposed power and energy prediction models that employ a configurable, back-propagation, artificial neural network (BP-ANN). The parameters of the BP-ANN model are ten carefully selected PMCs of a GPU. The values of these PMCs are obtained using the CUDA Profiling Tools Interface (CUPTI) [85] during the application execution. Their energy model for a GPU-based cluster can be described as follows [45]:

$$\forall l, m, l \in \Psi, m \in \Gamma, E = \sum_{l=1}^{\Psi}[(\sum_{m=1}^{\Gamma} \bar{P_{l,m}} \times t_{gpu}) + \bar{P}_{base} \times t_{gpu} + E_{parallel-overhead}]t_{gpu} = t_{pci} + t_{kernel} \tag{2.16}$$

Ψ represents the set of hybrid identical nodes where each node has multicore processors and multi-GPUs. The set of GPUs in a node is denoted by Γ. The parameter, t_{gpu}, represents the execution time of the application on the GPU cluster. $\bar{P_{lm}}$ denotes the average dynamic power consumption of the *m*th GPU on node *l*. This is predicted using an artificial neural network using the PMCs. $\bar{P}_{base}$ is the average base power consumption of the whole system on node *l* without the GPU. The total execution time of the application, t_{gpu}, is the sum of PCIe data transfer time, t_{pci}, and the kernel execution time, t_{kernel}. The parameter, t_{pci}, is calculated as the size of the data transferred via the PCIe link divided by the bandwidth of the PCIe link. The parameter t_{kernel} is predicted using a performance model. $E_{parallel-overhead}$ represents the energy consumption from parallel overhead due to network communications in the cluster. A NVIDIA Tesla C2075 is used for model validation and analysis. The authors [45] reported an average prediction error rate of 2.1% for their power model and maximum prediction error percentage of 11% for their energy model.

Marowka [86] presented analytical models for studying the energy consumption of various architectural design choices for hybrid CPU-GPU chips.

Their model for performance per watt for an asymmetric processor follows:

$$\frac{Perf}{W_a} = \frac{1}{P_s + P_c + P_g} \tag{2.17a}$$

$$P_s = (1-f)(1 + (c-1) \times k_c + g \times w_g \times k_g) \tag{2.17b}$$

$$P_c = \frac{\alpha \times f}{c} \times (c + g \times w_g \times k_g) \tag{2.17c}$$

$$P_g = \frac{(1-\alpha) \times f}{g \times \beta} \times (g \times w_g + c \times k_c) \tag{2.17d}$$

c is equal to the total number of CPU cores and g is equal to the total number of GPU cores. α represents fraction of program's execution time spent in parallel execution on the CPU cores. β is the GPU core's performance normalized to that of the CPU core. f represents is the execution time of the fraction of program that can be parallelized. P_s represents the power consumption during the sequential computation phase. It is equal to power of 1 consumed by one active CPU core plus remaining $c-1$ CPU idle cores consuming a fraction of power, k_c, plus g idle GPU cores consuming fraction of power, $k_g \times w_g$. P_c represents parallel computation on CPU cores only. P_g denotes parallel computation on GPU-cores only [86].

Lim[53] enhanced McPAT [87] to write a power model for GPUs. In their power model, the total average power is calculated as a sum of the power consumptions of all the components. It can be described as follows:

$$TotalPower = \sum_{i=0}^{n} P\_Component_i = a \times P_{SP\_fpu\_dyn} + b \times P_{SP\_fpu\_lkg} + c \times P_{SP\_alu\_dyn} + d \times P_{SP\_alu\_lkg} + e \times P_{ConstMem} + P_{Others} \tag{2.18}$$

where a, b, c, d, and e are scaling parameters. NVIDIA's Fermi architecture is used for building the power model. They report average prediction errors of 7.7% and 12.8% for the micro-benchmarks (used to build the model) and Merge benchmarks [88] respectively.

Wang [46] used the technique of program slicing to model GPU power consumption. The source code of an application is decomposed into slices

and these slices are used as basic units to train a power model based on fuzzy wavelet artificial neural networks (FWNN). So, unlike earlier research efforts which use PMCs, slicing features are extracted from the programs and used in their model. They used three GPUs for evaluation of their model but prediction error rates are not reported.

Ma [89] used support vector regression (SVR) to predict the power consumption of a GPU based on workload signals. They choose five major workload signals representing the runtime utilizations of major pipeline stages in a NVIDIA GeForce 8800gt GPU. Li [90] presented power and performance prediction models to identify an energy-efficient consolidation of workloads. Xie [91] built a power model based on native instructions to analyse and estimate the power consumption at an architecture level. With proper profilers and tools, they identify major power contributors in GPU architecture. These contributors are divided into two groups, i.e. computing unit and memory access on which different native instruction were run and measured separately as a foundation for their energy prediction model.

AMD GPUs are also used in many HPC systems. For example: three systems in Top500 list ([25] and two in Green500 [22] use AMD FirePro GPUs (the AMD Firepro Server S9150 also topped the November 2014 Green500 list). However, power and energy models for these GPUs are abysmally lacking.

Zhang [52] employed a rigorous statistical model to predict power consumption of GPGPU applications executing on an ATI Radeon HD5870 GPU. They also used a Random Forest method to correlate the execution characteristics and the power consumption of the GPU. They reported a median absolute error of 4.34% for total power consumption. Zhao [92] examine the power breakdown using McPAT [87] of different components of NVIDIA and AMD GPUs such as cores and caches, memory controllers, and off-chip memory. Based on experiments on AMD Radeon HD7970, they report that the off-chip DRAM accesses consume a significant portion (32 %) of the total system power. To reduce the memory power consumption without compromising the memory bandwidth, they scale down the supply voltage and frequency of memory interface.

Table 2.3: Power/Energy measurement infrastructures for the HPC applications on FPGAs

Model	Power/Energy Measurement Infrastructure
[93]	Data sheet TDPs ([94]) for FPGA, GPU, and CPU
[95]	Xilinx XPower Estimator for FPGA [96]. For CPU and GPU, power measured indirectly between mains and Host PC power supply
[97]	No details of how power is measured
[98]	Wattsup? Pro power meter
[99]	Olson remote power monitoring meter for the whole system
[100]	Data sheet TDP for GPU. No details of how power is measured for FPGA and CPU
[101]	Xilinx XPower Estimator for FPGA [96]. No details of how power is measured for GPU and CPU
[102]	Fluke Norma 4000 Power Analyzer for the whole system
[103]	Xilinx XPower Estimator 13.2 for FPGA [96], Data sheet TDPs for GPU and CPU
[104]	Xilinx Power Estimater for FPGA [96], data sheet TDP for GPU.
[105]	PC diagnostics software utility EVEREST for CPU. For FPGAs and GPUs, a pincer galvanometer (equipment type HIOKI3290)
[106]	Power meter for the whole system
[107]	No details of how power is measured
[108]	Monitoring a wattmeter

One observation from the analysis of these models is that even though both Song [45] and Wang [46] use sophisticated artificial neural network (ANN) methods, there is a remarkable difference in their reported power prediction errors, indicating that sufficient training set has not yet been identified.

2.4.4 Power and Energy Models for Xeon Phi and FPGA

In this section, we cover the other accelerators that are used in high performance computing systems.

Xeon Phi [109] is the competing offering from Intel in the accelerator space based on Intel's Many Integrated Core Architecture or Intel MIC. The Top 500 (November 2017) list contains 26 supercomputers using Xeon Phi. This list

includes the No.2 supercomputer Tianhe-2, which uses Xeon Phi 31S1P containing 57 cores running at 1.1GHz. Table (2.2) shows the key features of the models for Xeon Phi under the heading “Intel Xeon Phi”. We found just one energy prediction model for this accelerator even though it appropriates an appreciable share in the Top500 supercomputers. Shao [54] constructs an instruction-level energy model of a Xeon Phi processor and report an accuracy between 1% and 5% for real world applications. We believe this may be due to the ability of the Xeon Phi to report its own power consumption directly from an internal meter.

FPGAs (Field Programmable Gate Arrays) are another acceleration technology that holds immense promise for high performance computing platforms. They have created the Reconfigurable Computing (RC) market segment, which exploited the inherent parallelism and reconfigurability provided by them to “hardware accelerate” software algorithms. High-Performance Reconfigurable Computing (HPRC) [110] brought Reconfigurable Computing into the high-performance computing sphere by combining FPGAs with multicore CPUs. In a node, the FPGA is used as a coprocessor in the same manner as a GPU or Xeon Phi, and is connected to a CPU through a PCI-Express (PCIe) bus.

Mitta [111] reported that for several applications, FPGAs have demonstrated better performance and energy efficiency than CPUs and GPUs. Table (2.3) summarizes the power/energy measurement infrastructures used in the studies that have compared the energy efficiency of FPGAs to CPUs and GPUs [111]. All of these works use direct physical measurements and none use models; some use power estimators for FPGAs provided by the vendors which are based on a model of the switching factor of the components in the design based on the CMOS power equation 1.1.

There are no linear regression models using PMCs because PMCs are not yet offered by FPGAs. Ou [112] constructed a linear energy prediction model based on instruction level energy profiling. Poon [113] presented a hardware-level model, which uses a placement and routing CAD tool and in-depth knowledge of FPGA architecture. Wang [114] proposed a linear component-based model to predict energy consumption of a reconfigurable Multiprocessors-on-

a-Programmable-Chip (MPoPCs) implemented on Xilinx FPGAs. Khatib [115] proposed a linear instruction-level model to predict dynamic energy consumption for soft processors in FPGA. The model considers both inter-instruction effects and the operand values of the instructions.

2.4.5 Analytical Energy Models

Demmel [116] proved that a region of strong scaling in energy exists for matrix multiplication and N-body problem. That is, they showed that for a given problem size n, the energy consumption remains constant as the number of processors p increases and the runtime decreases proportionally to p.

Choi [117] presented an energy roofline model based on the time-based roofline model [118]. Their models for time and energy can be described as follows:

$$\begin{aligned} T &= max(W \times \tau_{flop}, Q \times \tau_{mem} \\ &= W \times \tau_{flop} \times max(1, \frac{B_\tau}{I}) \end{aligned} \tag{2.19}$$

T denotes the running time of the algorithm (under the assumption of perfect overlap between computations and memory operations). W and Q are the number of arithmetic and memory operations respectively. I is the algorithm's computational intensity. τ_{flop} and τ_{mem} respectively are the time per flop and time per mop. B_τ is called the time-balance point. Their energy model is presented below [117]:

$$\begin{aligned} E &= W \times \epsilon_{flop} + Q \times \epsilon_{mem}) + \pi_0 \times T \\ &= W \times \hat{\epsilon}_{flop} \times (1 + \frac{\hat{B}_\epsilon(I)}{I}) \end{aligned} \tag{2.20}$$

ϵ_{flop} and ϵ_{mem} respectively are the energy per flop and energy per mop. $\hat{B}_\epsilon(I)$ is called the effective energy balance. They conclude that the balance gap ($\frac{B_\epsilon}{B_\tau}$) represents the difficulty in achieving energy efficiency compared to time efficiency. They validated their model on a Intel multicore CPU and a NVIDIA Fermi GPU. They used PowerMon2 apparatus [119] for power and energy

measurements.

Choi [120] extended the roofline model by adding parameters such as power caps, memory hierarchy access costs, and measurement of random memory access patterns. They conduct a microbenchmarking study of time, energy, and power of computation and memory access for over dozen diverse platforms, which include x86, ARM, GPU, and hybrid (AMD APU, SoC) processors. They conclude that constant power (or base power) is a critical limiting factor accounting for about 50% of total power in 7 out of 12 platforms evaluated. Based on these conclusions, they recommend further tighter integration of non-processor and non-memory components.

Marszałkowski [121] analysed the impact of memory hierarchies on time-energy trade-off in parallel computations. They formalised non-linear dependence of execution time and energy on problem size. They use this formalisation in their multi-objective optimisation problem of minimising time and energy in parallel processing of divisible loads. The total energy consumed is computed as sum of energy consumed of all components.

2.4.6 Power and Energy Models in High Performance Computing Applications

In this section, studies for saving power and energy specifically in HPC applications are presented. We examine

- How power and energy consumptions were measured (either via direct measurements or models) for a node and for the whole cluster.
- How a power/energy model for a cluster is composed from the power/energy models of nodes.
- The prediction error of application-specific models to application-agnostic (i.e., full-system or the component-specific) models.

Table 2.2 summarizes these studies under the heading “HPC Applications”. Studies not shown in the table use direct physical measurements.

Kamil [122] is a pioneering effort that studied power consumption of applications executing on large-scale HPC systems. They conclude that power consumption of HPL benchmark is a good representative predictor for the power consumption of a HPC workload and that power consumption of a large-scale system can be accurately predicted from power measurements of smaller subsets of the system. Their study was conducted on stand-alone systems and a large-scale NERSC Cray XT4 system. Using results of a single cabinet in the system, they model the system's power usage. The power consumption of a single rack is extrapolated linearly to the 102 racks that compose the HPC system. Their model is described as follows:

$$\begin{aligned} DCWatts_{system} &= 102 \times DCWatts_{rack} + 50KW \\ &= 102 \times DCAmps_{rack} \times Volts_{rack} + 50KW \end{aligned} \tag{2.21}$$

where 50KW is the power consumed by the disk subsystem.

Bui [55] proposed a power model using PMCs to predict power consumption of parallel scientific applications running on modern multicore/multiprocessor systems. They modify the power model of [34] to model power consumption for a modern Intel Itanium2 processor. For the architecture scaling factor (a parameter modelled in [34] using component area ratios), transistor counts are used as initial values. The power for each component is modelled as a linear function of access rates of its architectural units. The total power consumed by a processor is calculated as the sum of power consumptions of its components and its base power. The total power consumption of a multicore system is then calculated as the sum of power consumptions of all the cores/processors that are contained in it.

One of the most popular frameworks used for fine-grained power and energy profiling on parallel and distributed systems is the PowerPack framework [123]. This toolkit contains both hardware and software components. An AC power meter (Watt's Up Pro) is used to measure the power draw from the wall and DC power data acquisition devices (Analog Input Module NI 9205NI and cDAQ chassis NI cDAQ9172) are used to obtain component-level power consumptions inside a node through precise instrumentation of all its power rails.

The software components provide facilities for synchronized collection of the measurement data from various data streams and analysis of the data. The toolkit is used to measure the power and energy consumption of one node at a time. To obtain total power consumption of a whole cluster, a node remapping approach [123] is used.

Subramaniam [56] used multiple linear regression to model the power consumption of the High Performance Linpack (HPL) benchmark [124]. Out of the 18 HPL parameters, they select four parameters (problem size, block size, number of process rows, number of process columns) for the regression modelling. To evaluate their model, they performed experiments using 64 nodes from a cluster called SystemG [125]. The power and energy values were obtained using "Watts UP? Pro E" power meter. The power consumption of the cluster is predicted by extrapolating the prediction of power consumption of a single node.

Dongarra [126] studied energy consumptions of high performance dense linear algebra libraries LAPACK [127] and PLASMA [128] using PowerPack [123] and Intel RAPL API [129]. They conclude that, for the applications using these libraries, RAPL API is a good alternative to power meters based on near identical power measurements observed between PowerPack and RAPL.

Tiwari [57] studied CPU and DIMM power and energy models using artificial neural networks. They studied three important HPC kernels, matrix multiplication, stencil computation, and LU factorization. To derive component-level power measurements, they used the PowerMon2 apparatus [119]. They reported an absolute error rate of 5.5% for the total power consumption and energy usage predictions for the three kernels.

Lively [58] and [130] proposed application-centric predictive models for power consumption with multivariate linear regression models for system power, CPU power, and memory power constructed using PAPI performance events (PMC) [131] as predictors. To train and validate their model, they conducted experiments in a power-aware cluster called SystemG [125]. They validated the models using four hybrid scientific applications and reported a maximum prediction error percentage of 4.93%.

Kestor [61] proposed a per-core power model based on a regression anal-

ysis of core activity. Ltaief [132] and Bosilca [133] also compared the power consumptions of two high performance dense linear algebra libraries, LAPACK and PLASMA. Their results indicate that PLASMA outperforms LAPACK both in performance as well as energy efficiency.

Witkowski [134] and Jarus [135] proposed system-wide power prediction models for HPC servers based on performance counters and used decision trees to select an appropriate model for the current system load. They report average prediction error of 4% based on experiments on four servers, one AMD and three Intel.

Gschwandtner [62] presented linear regression models based on hardware counters for prediction of energy consumption of HPC applications executing on IBM POWER7 processor. They use NAS parallel benchmarks [136] to train the models. They picked a small subset from 500 different hardware counters offered by the POWER7 processor. They report a maximum prediction error of 15%.

There are exhaustive studies primarily focusing on the performance analysis of Xeon Phi but very few on power/energy models. [137] present a detailed study of the performance-energy trade-offs of the Xeon Phi architecture. They propose extensions to PowerPack infrastructure [123] to measure component-wise power and energy consumptions of systems hosting accelerators. The energy efficiency of Xeon Phi 5110P is compared to a Tesla Fermi c2050 GPGPU and the results are found to be mixed.

Reddy [138] presented an application-level energy model where the dynamic energy consumption of a processor is represented by a function of problem size. Unlike PMC-based models that contain hardware-related PMCs and do not consider problem size as a parameter, this model takes into account highly non-linear and non-convex nature of the relationship between energy consumption and problem size for solving optimization problems of data-parallel applications on homogeneous multicore clusters for energy.

2.4.7 Summary

The goal of this section was to compare prediction accuracies of power and energy consumptions of HPC applications and the prediction accuracies of application-agnostic power and energy models. We expected application-specific predictive power and energy models to have higher prediction accuracy but our survey shows results to the contrary.

2.5 Case Study of Performance Monitoring Counter Based Models

Table 2.4: Specification of the Intel Haswell workstation used to build the PMC-based power and energy models.

Technical Specifications	**Intel Haswell i5-4590**
Processor	Intel(R) Core(TM) i5-4590 3.3 GHz
Microarchitecture	Haswell
Memory	8 GB
Socket(s)	1
Core(s) per socket	4
L1d cache	32 KB
L11 cache	32 KB
L2 cache	256 KB
L3 cache	6144 KB
TDP	84 W
Base Power	22.3 W
Max Turbo Frequency	3.7 GHz

From this background reading it can be found that the most dominant approach employs linear regression using PMCs as parameters to model the power/energy consumption of a node. In this section, the accuracy of this approach for a modern node is assessed. Power and energy models using this approach on a Intel Haswell platform with the specification shown in Table 2.4

Table 2.5: Applications used for the training set

Benchmark Suite	**Applications**
NPB SER [136]	BT.SER.α, CG.SER.α, DC.SER.α, EP.SER.α, FT.SER.α, IS.SER.α, LU.SER.α, MG.SER.α, SP.SER.α, UA.SER.α where α = S, W, A
NPB OpenMP [136]	BT.OMP.α, CG.OMP.α, DC.OMP.α, EP.OMP.α, FT.OMP.α, IS.OMP.α, LU.OMP.α, MG.OMP.α, SP.OMP.α, UA.OMP.α where α = S, W, A
Rodinia OpenMP [139]	bfs, heartwall, kmeans, leukocyte, nn, particlefilter, srad, backprop, cfd, hotspot, lavaMD, lud, nw, pathfinder, streamcluster
Stream [65]	Stream C
BLAS [140]	daxpy, dgemv, dgemm

are built. The Likwid API [2] is used to get the PMC values of an application execution. 35 PMCs from 13 performance groups are selected from the total of 390 supported PMCs (Table A.5 in Appendix A.3). Some groups share some of the selected PMCs.

To train the model, applications from Table 2.5 from NAS serial and OpenMP benchmarks [136], Rodinia OpenMP benchmarks [139], STREAM benchmark [65], and BLAS double-precision benchmarks [140] were selected. Benchmark classes (S, W, A) were used in the NAS benchmark suite. For the parallel benchmarks, NAS OpenMP and Rodinia OpenMP, the number of OMP threads executed is 4, which is equal to the number of physical cores in our platform. For all the other applications, the number of threads in the application is set to 1.

Since PMC values differ for different runs of the same application, the methodology described below was followed to make sure the experimental results are reliable:

- The Intel Haswell platform is fully reserved and dedicated to these experiments during their execution. It was made certain that there are no drastic fluctuations in the load due to abnormal events in the platform by monitoring its load continuously for a week using the tool *sar* to establish

a baseline. Insignificant variation in the load was observed during this monitoring period suggesting normal and clean behavior of the platform.

- When an application is executed, its execution was bound to physical cores using the *numactl* tool.

- The application was repeatedly executed until the sample mean lies in the 95% confidence interval and within a threshold precision of 0.025 (2.5%). For this purpose, Student's t-test was used assuming that the individual observations are independent and their population follows the normal distribution. The validity of these assumptions were verified by plotting the distributions of observations. The application was repeatedly run until one of the following three conditions is satisfied:

  1. The input maximum number of repetitions have been exceeded.
  2. The sample mean falls in the input confidence interval of 95% (or the precision of 0.025 has been achieved).
  3. The elapsed time of the application runs has exceeded the maximum time allowed (3600 seconds).

  Each application was always run for a minimum number of repetitions. The input minimum and maximum number of repetitions differ based on the application or the problem size solved. For small problem sizes (NAS benchmarks class S, W), these values are set to 10000, and 100000 respectively. For medium problem sizes (NAS benchmarks class A), these values were set to 100 and 1000. For large problem sizes (NAS benchmarks class B, C, D), these values were set to 5 and 50. For each data point, the sample mean was selected as the output only when the input precision of 0.025 (2.5%) had been reached. If the precision of measurement was not achieved before the maximum number of repeats have been completed, the number of repeitions were increased, as well as the maximum elapsed time allowed. However, it was observed that condition (2) was always satisfied before the other two in our experiments.

The PMC values, power and energy consumptions, and execution times were obtained separately using three separate programs. This was to isolate the overhead (constant but low) in collecting PMC values using likwid API. The energy consumptions at the socket and DRAM level were obtained using the RAPL PMC values (PWR_PKG_ENERGY:PWR0, PWR_DRAM_ENERGY:PWR3) of the performance group, ENERGY. The average dynamic power consumption during the execution of an application was obtained by dividing the total energy consumption (given by the PMC values) by the total execution time of the application (using *gettimeofday* timer). The power and energy models were built using the GSL multiple linear regression API [141]. The main steps of an application execution in the training and validation set were initializing Likwid runtime, starting of monitoring PMC, executing the benchmark code, stopping the monitoring of PMC, querying for values of the PMCs, and finalizing the Likwid runtime.

The model took two weeks (*implementation complexity* of 2 EW) to implement. The implementation tasks involved studying Likwid API [2], short-listing the PMC to use in the modelling which includes using rigorous statistical techniques to prune the PMC set, studying GNU scientific library linear regression API [141], short-listing the applications (or benchmark test suites) to use in the training and validation sets, writing scripts to automate the model building and validation process, and cleaning the validation data, for example, removing outliers since GSL multiple linear regression (MLR) is quite sensitive to them. The models took 6 hours of experimental time to build. Figure A.1 (Appendix A.3.2) shows the execution times of applications in the training set that are used to calculate their average dynamic power consumptions. It also shows that we have used applications with a wide range of execution times to build the models. This would allow for a determination of the true capability of a model to predict accurately the power and energy consumptions for applications with a wide range of execution times.

One interesting observation from the results is that some PMCs have negative coefficients (Table A.6 in Appendix A.3.1). For example, consider the PMC, L2_RQSTS_MISS. It represents how often it was necessary to get cachelines from memory and therefore it should only increase the average dynamic power

Table 2.6: Prediction error percentages for NAS Serial and OpenMP Applications (Benchmark Class W)

Benchmark	**Avg. Dynamic Power Prediction Error % (std. dev.)**	**Energy Prediction Error % (std. dev.)**
BT.SER.W	98.716442 (0.10)	235.52 (1.50)
CG.SER.W	96.66 (0.41)	171.02 (5.90)
DC.SER.W	99.76 (0.01)	97.30 (0.20)
EP.SER.W	99.92 (0.01)	99.39 (0.15)
FT.SER.W	99.24 (0.34)	100.34 (4.91)
IS.SER.W	130.57 (18.66)	40.94 (6.79)
LU.SER.W	50.19 (4.5)	542.05 (64.37)
MG.SER.W	98.21 (0.07)	6.70 (1.06)
SP.SER.W	99.43 (0.03)	72.08 (0.50)
UA.SER.W	79.74 (0.94)	12.86 (13.51)
BT.OMP.W	98.68 (0.17)	10.67 (2.52)
CG.OMP.W	100.00 (0.00)	99.99 (0.00)
DC.OMP.W	92.09 (2.33)	85.55 (13.39)
EP.OMP.W	92.96 (0.87)	10.52 (2.53)
FT.OMP.W	99.41 (0.23)	112.76 (3.33)
IS.OMP.W	64.66 (8.48)	31.83 (1.22)
LU.OMP.W	81.96 (3.22)	626.71 (46.07)
MG.OMP.W	99.12 (0.07)	48.53 (1.06)
SP.OMP.W	99.25 (0.64)	561.84 (9.28)
UA.OMP.W	88.29 (1.18)	15.81 (16.90)
daxpy	99.27 (0.19)	45.41 (2.84)
dgemv	89.00 (1.29)	39.87 (18.46)
dgemm	83.53 (1.83)	77.14 (26.16)

consumption. However, this coefficient is negative for the power model but positive for the energy model. Same applies for PMC, ICACHE_MISSES. Now, at this point, this set of PMC can be pruned further to remove the parameters with negative coefficients or collinearity between parameters using rigorous statistical techniques ([48]) or detailed iterative methodologies ([40], [51]).

To validate the models, they were used to predict power and energy consumptions of NAS serial and OpenMP benchmarks (class W) and BLAS applications. The (minimum, average, maximum) prediction error percentages of average dynamic power and total energy consumptions are (50, 93, 130) and

(6, 136, 626) respectively (Table 2.6 with standard deviations). The prediction error percentage for a model is calculated using the actual value provided by Likwid API and the fitted value provided by the GSL MLR function using the model.

During an application execution in the training and validation set, DVFS was disabled by pre-setting the frequencies of CPUs to base frequency using "userspace" [142] governor. However, in a real-life situation, dynamic power management (DPM) is not disabled and the default governor is "ondemand" [142]. Due to DPM (and runtime DVFS), there can be fluctuations in power consumption. Therefore, the rationale behind use of PMCs is questionable, which are accumulated for a whole application execution, to model instantaneous dynamic power. Models must be made aware of *C-State* and *P-State* values. Even if PMCs are used to model power for small but distinct phases of an application, the significant overhead of acquiring these PMC values and building phase-level models prohibits their use for this case. For example, CUPTI [85], due to its design limitation, allows querying for only one PMC (or event) of a NVIDIA GPU platform in a single CUDA kernel invocation [143]. A CUDA kernel must be executed many times to get the values of all the desired PMCs for a given GPU.

2.6 Discussion

In this section, a summary of all the power/energy models that we have surveyed is presented. The prominent observations for each of the model characteristics are as follow:

1. *Level of Abstraction*:

    - There exists no model today that truly and comprehensively captures the highly heterogeneous and hierarchical architecture of a node illustrated in Figure 1.1. That is, no model exists that satisfies the property of *Non-linear Independence*.
    - Many models for a node have the property of *Linear Independence*.

Such models are constructed by summation of models of the components.

2. *Type of power*:

   - There are very few models that predict instantaneous power of an application.
   - We expect the models that predict instantaneous power accurately to maintain their accuracy for a wide range of problem sizes and execution times.
   - However, we are sceptical of models that are used for predicting average total power consumption for applications running for long durations (hours to days) to accurately predict power for a wide range of problem sizes and execution times.

3. *Decomposition*:

   - Many models focus exclusively on modelling power consumption of either the CPU or an accelerator in a node. They can be further classified into:
     (a) Models that adopt linear regression methods using performance-monitoring counters (PMCs) as the parameters.
     (b) Models that adopt non-linear methods such as artificial neural networks, etc. using PMCs as the parameters.
   - Models using PMC employ iterative methodologies or methods such as Principal Component Analysis (PCA), Random Forest etc. to prune the set of PMCs.
   - Very few models take into account contention for shared resources between components (for ex: cores sharing last-level cache).
   - Very few models study power consumption of communication links in a node such as
     (a) On-chip interconnect between CPUs.

(b) PCIe bus connecting the multicore CPUs and accelerators.

4. *Accuracy of power prediction*:

   - Almost all the models report prediction error for only the total average power consumption but not the dynamic power consumption. We would expect the prediction errors for dynamic power to be higher.
   - There are many models that do not mention clearly the proportion of dynamic to static power consumptions in the total power consumption as can be seen from the empty entries for *dynamic power* in the tables for the models.

5. *Implementation Complexity*: The implementation complexity for the models employing linear regression or neural networks using the *PMC* as parameters is quite high. This is due to several reasons:

   - The first step in these models is to design and write a micro-benchmark test suite for a component. This test suite contains benchmarks that stress the various architectural units of the component to create a model of the power consumption of the architectural unit as a function of its access rate (activity factor) or its PMC set. To the best of our knowledge, there are no standard micro-benchmark suites. So, if one were to implement one of these models, one has to write a micro-benchmark test suite. We believe that writing micro-benchmarks is a very complex, tedious, and error-prone task. One key design requirement of the test suite is to stress a single architectural unit avoiding completely or minimizing interaction with other architectural units. This is required so that its activity can be decoupled to derive its contribution to the total power consumption. If there is interaction, the micro-benchmark stressing an architectural unit must ensure that the utilization is constant for the other architectural units during its execution. So, this interaction must somehow be removed or accounted in the model for the architectural unit to avoid collinearity problems when multiple linear

regression is applied later. This is difficult to accomplish as micro-benchmark test suites are usually written in assembly language. So, the model implementer must now become an expert in assembly language to write a reliable micro-benchmark test suite.

- Wu [144], Bertran [71], Molka [47], Kestor [145], Pandiyan [146] describe in great detail this complex process of writing a micro-benchmark test suite. Bertran [71] have designed 97 micro-benchmarks for their power model.
- So, after a micro-benchmark test suite is written, each of the micro-benchmarks must be run for a sufficiently long time to ensure that the power consumption reading (using a power meter) is stable. However, there is no straightforward method to separate the static and dynamic components of this power consumption. The access rates or the PMC values for the architectural unit are also obtained separately.
- Then, the power consumption of an architectural unit is modelled as a function of its PMC. Today, each architectural unit comes with a large set of PMC. For example: Haswell architecture [2] has 390 events. So a question arises as to which events to select from this set of PMCs. Models include some form of Principal Component Analysis (PCA) to prune this set.
- Some models use complex methods such as Random Forests (RF) and Neural Networks etc. to incorporate non-linearity or dependence between the PMCs. Therefore, implementation of these models necessitates the model writer to possess or acquire knowledge of these methods.

6. *Portability*:

   - Many models ensure their portability to next-generation processors in the same architecture space by using the PMC approach.
   - However, the PMC approach hinders their generality or applicability to all architectures since PMCs are architecture-specific.

7. *Scalability*

   - The standard approach to determine the total power/energy consumption of a large-scale HPC system is based on the following steps:
     (a) Measure the power/energy consumption of a node (either using direct measurements or power/energy models). Multiply this power/energy consumption by the total number of nodes in the system.
     (b) Or use a node remapping technique. In this technique, the number of measurements is equal to the number of nodes and each time the node that is measured is mapped to a different physical node. The total power/energy consumption is equal to the sum of all the measurements taken.

PMC-based approaches are frequently used to model unit-level power consumptions of a nodal component to predict the total power consumption of it. However, we question its continuing use to predict power consumption of a nodal component or a node due to two reasons. Firstly, it has high implementation complexity. In most cases (For example: Likwid, CUPTI), the values of all the performance events can not be obtained during a single execution of an application. Secondly, while values of some PMCs can be determined from static analysis of the source code of an application, values of most PMCs are gathered during the application execution. Therefore, an application must be executed to get the values of PMCs, which are then input to a PMC-based model to get the predicted power/energy consumption. However, there are software libraries available today on all mainstream commodity processors providing interfaces to determine power consumption at component level during an application execution via in-built power meters or models. The PAPI library ([131], [147]) provides API for energy consumption. For Intel's CPU processors, the library uses the "Running Average Power Limit" (RAPL) component [148], which uses a software model to predict energy. Intel PCM (Performance Counter Monitor) [149] is a tool to monitor performance hardware counters on Intel processors, similar to PAPI. The difference between

PCM and PAPI is that PCM supports only Intel hardware, but it can monitor also uncore metrics, like memory controllers and QuickPath Interconnect links. The Intel PCM utility *pcm-power* displays energy usage and thermal headroom for CPU and DRAM sockets. For Intel's Xeon PHI processor, the MPSS *MicMgmt* library [150] is used to get the instantaneous power consumption from the on-chip power meters. NVIDIA Management Library (NVML) API [151] can be used to determine the power consumption of the NVIDIA GPUs from the on-chip power meter. Although there is some overhead introduced by these library calls, it can be minimized by choosing appropriate sampling frequency. Therefore, when one can get the power/energy consumption from vendor-specific software libraries (that provide access to readings from all on-chip power meters at low sampling frequency), we question the necessity of using performance events for modelling power/energy consumption. There is however a need for tooling to better expose these new power measurement metrics to researchers in a unified and useful manner. This issue is tackled in this thesis.

2.7 Conclusion

This background survey presented a classification of predictive power and energy models for the major components at the node architecture level in modern HPC computing platforms.

One overarching conclusion can be made from the survey. During the era of single-core processors, models were able to accurately predict the dynamic power of the full-system by having parameters that accurately but separately modelled dynamic power consumption of all components in the system. But now such an approach will be erroneous. Unless the inherent complexities (contention for shared resources, dynamic power management, etc.) of the modern node architectures are methodically taken into account, models aspiring to predict power/energy consumptions for these architectures will be inaccurate.

Power models have been the main research focus for HPC applications

and outnumber energy models. There are very few studies that directly model energy. Studies that do not directly model energy predict it using a power model and a timing model (or by expliciting measuring execution time). However, the prediction error in accuracy of energy consumption in these studies is compounded by the errors in accuracies of its constituent models (power and timing). Therefore, we would like to ask why not model energy consumption directly for HPC applications instead of constructing its constituent models. Power models and power management algorithms are necessary for system designers to ensure that an application execution does not exceed the power/thermal constraints of a system. For example, Intel RAPL [148] allows the power consumption of an application to exceed the TDP for short periods of time but monitors the power consumption closely to keep it close to an average limit by controlling frequency. Many research efforts propose power models and use them to find inefficiencies in the system and thereby provide suggestions to designers to improve the system architecture. However, if the goal of energy efficiency of HPC applications is to minimize the total energy consumption without sacrificing performance, one can strive to accomplish it by directly modelling energy consumption.

In the case of accelerators, we found that models for NVIDIA GPUs are predominant. Even though Intel Xeon Phi holds a notable portion (6%) of Top500 list [25], very few studies have modelled its power/energy consumptions. Considering that FPGAs are now emerging in the HPC space, there are no studies dedicated to their power/energy models for HPC applications. It is expected that due to their high degree of configurability and relatively high power efficiency, they will be a future topic of study.

Chapter 3

Node Level Power Measurement

3.1 Introduction

The goal of this work is to provide an API for developers, initially through the C programming language [152], for taking accurate measurements of critical code sections within their applications. The critical sections, may be nested so that finer grain measurements (such as a compute kernel) can be taken without sacrificing coarser grained measurements (such as the entire application). It is assumed that the user has exclusive access to the node when measurements are taken, which is standard practice in HPC environments. This is necessary to avoid other application's activity being erroneously reported as part of the user's target application's power consumption. Furthermore, as our focus is on application measurement, it is required that only one application may be measured per experiment, though this application can be multi-threaded or multi-process if using the MPI functionality.

In this chapter instrumentation at node level granularity is considered. At node level only the power that is delivered to the server via the mains power is considered. The node is treated as a *black box* that consumes power. The practical ways in which this level of measurement may be implemented is discussed below, with their advantages and disadvantages, followed by the solution this work implemented for experimental deployment on the GRID5000 [153] infrastructure. The work described here was published in [4].

3.2 Background and Related Works

There are several works related to this project, which are described in this section.

Kwapi [154], provides a web based interface to GRID5000's [153] energy measurement infrastructure. While it exposes the raw data of the API upon which we build *HCLEnergyAPI*, it is intended for post execution analysis and lacks an API for measurement of applications or kernels within an application. We see this as a necessary feature to facilitate dynamic load balancing. It does however produce useful graphs and visualisations.

Barrachina [155] presents pmlib, a software package for measuring energy states on CPUs and provides whole node level measurements accurately through use of external power meters. It has the advantageous ability to interface with high frequency external measurement devices. This approach requires a daemon to execute on all instrumented nodes. It is not clear how duplicate contributions from multiple processes on the same node are handled.

Cabrera provides EML [156] which are similar contributions to our *HCLEnergyAPI*, but lacks the ability to report statistical confidence and transparently calculate per node power in MPI applications.

In addition to these software methods, there are such as PowerPack [157] and PowerMon2 [158] that intercept power rails of components to give component level. These methods are difficult to deploy in real systems and are disadvantaged by the complexity of measuring devices with multiple power rails [159].

This work is limited to a node level granularity, but should the reader be interested in finer grained measurement at the device level, the following tools which are performance counter based may be useful.

Intel provides the RAPL interface [160] which is a software power model and similarly AMD provides APM [161]. Though easily accessible, both have disadvantages. Intel RAPL fails to provide power measurements, only providing energy with no timestamp data, hindering indepth analysis [162] and AMD APM has been shown to be inaccurate due to assumptions during sleep

```
https://api.grid5000.fr/stable/sites/lyon/metrics/pdu/
    timeseries?resolution=1&from=1502367972&to=1502367997&
    only=nova-19.lyon.grid5000.fr
```

Figure 3.1: GRID5000 request

modes [162].

Likwid [163], a lightweight performance tool offers RAPL measurements from Intel SandyBridge and IvyBridge x86 processors.

Nvidia, through their management library (NVML [164]) provides access to milliwatt power consumption metrics, accurate to 5%, as well as current Pstate of each graphics card in a system. NVML also provides related metrics such a fan speed and temperature.

3.3 Method

3.3.1 GRID5000

GRID5000 [153] is an experimental testbed infrastructure spanning multiple cities in France, for large scale research on parallel and distributed applications. GRID5000 provides modern compute nodes, including heterogeneous nodes containing Nvidia GPU and Intel Xeon Phi accelerator devices. The nodes are connected through high speed 10Gb/s interconnects.

GRID5000 was used as our experimental platform as it provides a HTTP REST based metrology API. The API may be called by sending a HTTP request with query parameters specifying the information desired and the user's node.

The GRID5000 API provides detailed data ranging from available memory, processor and network utilisation, as well as whether or not the node hosts an accelerator. Crucially, for the work of this thesis, the API provides power measurement data in the form of a timeseries.

In Figure 3.1, the GRID5000 API is queried for power meter data for the *nova-19* node, at a resolution of 1 second. The response is shown in Figure

```
{
   "items":[
      {
         "resolution":360,
         "from":1502367840,
         "metric_uid":"pdu",
         "to":1502368200,
         "uid":"nova-19",
         "type":"timeseries",
         "values":[
            134.122222222222
         ],
         "links":[
            {
               "href":"/3.0/sites/lyon/metrics/pdu/timeseries/nova
                  -19",
               "rel":"self",
               "type":"application/vnd.fr.grid5000.api.Timeseries+
                  json;level=1"
            },
            {
               "href":"/3.0/sites/lyon/metrics/pdu",
               "rel":"parent",
               "type":"application/vnd.fr.grid5000.api.Metric+json;
                  level=1"
            }
         ],
         "hostname":"nova-19.lyon.grid5000.fr"
      }
   ],
   "total":1,
   "links":[
      {
         "href":"/3.0/sites/lyon/metrics/pdu/timeseries",
         "rel":"self",
         "type":"application/vnd.fr.grid5000.api.Collection+json;
            level=1"
      },
      {
         "href":"/3.0/sites/lyon/metrics/pdu",
         "rel":"parent",
         "type":"application/vnd.fr.grid5000.api.Metric+json;level
            =1"
      }
   ],
   "offset":0
}
```


Figure 3.2: Raw GRID5000 data

3.2 in JSON [165] format, where the true resolution of the response that the API could provide is given. The resolution degrades over time due to storage constraints of the GRID5000 database. Also included in the response is the *to* and *from* UNIX timestamps for the data points returned. Finally, the *values* section is an array containing power data in units of Watts, the first occurring at *from*, and subsequent values occurring at *resolution* intervals.

This raw data may be useful to a scientist asking for power data after the fact from a script, however it requires processing to extract meaningful results. The API contributed in this thsis, attempts to minimally process the data to a useful form during the application lifetime, so that decisions based on the data can be made at runtime. Possible uses of this API would be evaluating the energy cost of executing applications on a particular node in comparison to other nodes, allowing for automated dynamic energy aware load balancing of parallel workers, minimizing for energy use.

3.3.2 HCLEnergyAPI

The API provides the developer with a minimal set of calls with which to instrument their application. They can be grouped into three broad categories. First, the control functions from Table 3.1 are responsible for the initialisation of the API and the start/stop calls with which to wrap sections of code the developer is interested, known as an *event*. Secondly, there are the data functions in Table 3.2, which allow a developer to query an *event* for information such as how much energy was consumed, the idle power of the node, the cost of energy used given a currency/Watt Hour, or a time series of the power consumed throughout the event.

In Figures 3.3 and 3.4 the easy integration of the energy and power measurement functions into an application respectively are shown. There is no limitation on the number of events that an application can record using HCLEnergyAPI, and they may be nested such that an entire application can be instrumented, as well as constituent components, facilitating in-depth characterisation of the application.

In Figure 3.5, the infrastructure at GRID5000 and how it relates to our API

Table 3.1: HCLEnergy API - Control functions

hclenergy_t *hcl_init();
void hcl_start(hclenergy_t *event);
void hcl_stop(hclenergy_t *event);

Table 3.2: HCLEnergy API - Data functions

double energy_consumed(hclenergy_t *event);
struct host_power_series *getPowerSeries(hclenergy_t *event);
double idlePower();
double cost(hclenergy_t *event, double price);

is illustrated. Each compute node is directly connected (*a.*) to a power meter at the mains which measures and records the power consumed by the node. The meter streams its measurement data (*c.*) to a central metrology server which can be queried by any of the compute nodes via HTTP (*b.*).

As the GRID5000 infrastructure provides a timeseries of power data, our API calculates energy by integrating over this timeseries from the start to end timestamps of the event in question. Our *idlePower* function is included to provide insight to the static power consumed by the node. It calls the *sleep* system call, causing the application to wait for 1 minute until told to proceed by the operating system's kernel. Our API then requests the power consumed during the time the application was not performing any operations. A *cost* function is also provided which when called with a monetary cost per Kilo Watt Hour (kWh), the function will provide the cost in the given currency of the energy used by this application. This is useful for reporting purposes, and also for determining the ideal node to execute an application. If dealing with a multi-site infrastructure, the cost of electricity may differ between the sites and load balancing based on monetary value of this energy would be desirable.

Accuracy of measurement is essential when executing experiments. As the applications this work instruments do not have exclusive use of the platform on which they are running, as with performance measurements we cannot assume that each execution is deterministic and representative of subsequent

```
int main() {
        hclenergy_t *event = hcl_init();

        hcl_start(event);
        // Execute code for instrumentation
        hcl_stop(event);

        double energy = energy_consumed(event);
}
```

Figure 3.3: HCLEnergyAPI Example of Energy Measurement

```
int main() {
        hclenergy_t *event = hcl_init();

        hcl_start(event);
        // Execute code for instrumentation
        hcl_stop(event);

        struct host_power_series *power = getPowerSeries(event);
}
```

Figure 3.4: HCLEnergyAPI Example of Power Measurement

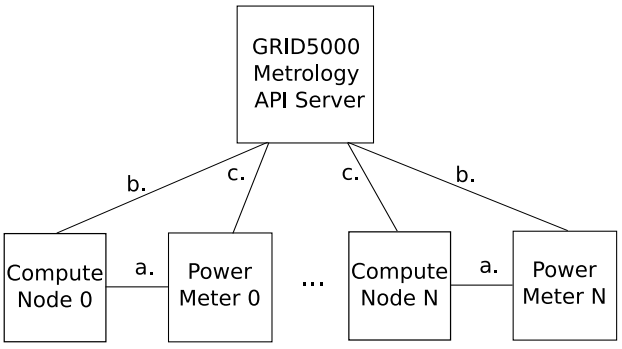


Figure 3.5: Power Measurement Infrastructure

```
./hclenergyrun --confidence=0.95 --threshold=5 <app>
```

Figure 3.6: hclenergyrun execution example

runs [1]. For this reason, statistical confidence features were built into our API. First, a researcher chooses a confidence interval and tolerance. Our utility called *hclenergyrun* executes the application of interest, and logs the data to a file on disk representing each run and continues in the same fashion for subsequent runs. When there are more than one run, the tool calculates the average of each event present in the application. The tool uses these values and the confidence and threshold values to calculate the confidence interval of each event. When all measurements fall within tolerance, the tool exits. The raw data remains on disk for further analysis.

By conducting experiments in this fashion, we can assert that the true energy value lies within the threshold amount above or below the value we report with the confidence selected by the researcher. For example, with threshold=5 and confidence=0.95, we can say that the true value for the energy values is found X$\pm$ threshold, 95% of the time, where X is the mean of the values we record across our experiments.

This test is valid if the distribution of measurements is gaussian. The tool checks for this condition and will not return a measurement if confidence cannot be calculated.

As these statistical methods are valid only when the number of measurement points exceeds 20 and the experimental data fits a normal distribution, checks are provided to ensure that this is the case.

3.4 Case Study: NAS NPB

In this section, the use of *HCLEnergyAPI* is demonstrated on a set of scientific codes. The C implementation of the NASA NPB suite [166] (originally Fortran) known as SNU NPB [167] was instrumented with our API. This code was

[1] That is to say, whilst there are no other users using the node, there are many background applications running as part of the operating system of the node.

chosen for this study as it contains multiple applications with various computational complexities and memory access patterns. The applications reflect several compute patterns that are found in computational fluid dynamics and are often used to evaluate new supercomputer infrastructure. They are listed in Table 3.3.

Table 3.3: Applications within NPB [1]

Label	**Description**
BT	Block Tri-diagonal solver
CG	Conjugate Gradient, irregular memory access and communication
DC	Data Cube
EP	Embarrassingly Parallel
FT	Discrete 3D fast Fourier Transform, all-to-all communication
IS	Integer Sort, random memory access
LU	Lower-Upper Gauss-Seidel solver
MG	Multi-Grid on a sequence of meshes, long and short distance communication, memory intensive
SP	Scalar Penta-diagonal solver
UA	Unstructured Adaptive mesh, dynamic and irregular memory access

For each application, there are multiple problem sizes labelled $\{S, W, A, B, C, D, E, F\}$. The *S* configuration, is small and only for testing purposes. The *W* configuration is targeted at workstations of the 1990s. Configurations $\{A, B, C\}$ are standard problem sizes, each increasing by 4x from the last. Configurations $\{D, E, F\}$ are larger again, with a 16x increase from the previous size.

In our experiment we target machines described in appendix A.1. We run all appropriate sizes of problem class for all applications. The number threads were varied from 1 to the maximum number of threads supported by the given processor. The default governor (ondemand) was used, allowing the processor to alter its *C* states and *P* states as it saw fit. For the target machine, several of the smaller benchmarks were too small to execute and some of the larger benchmarks do not have $\{D, E, F\}$ configurations.

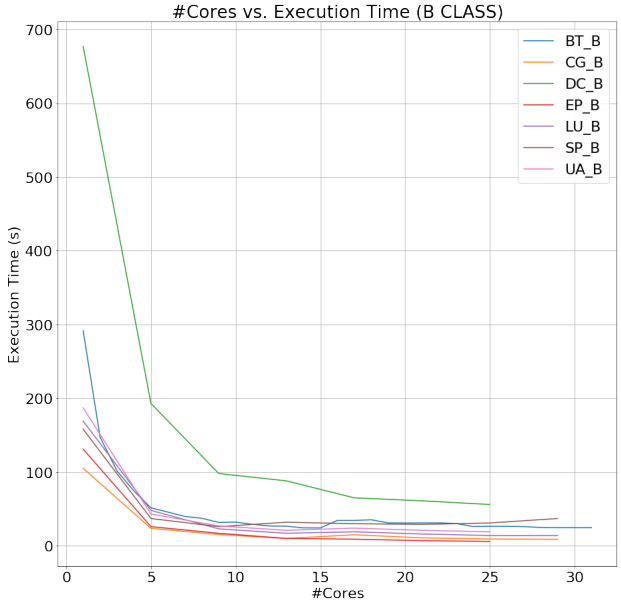


Figure 3.7: B Class Peformance

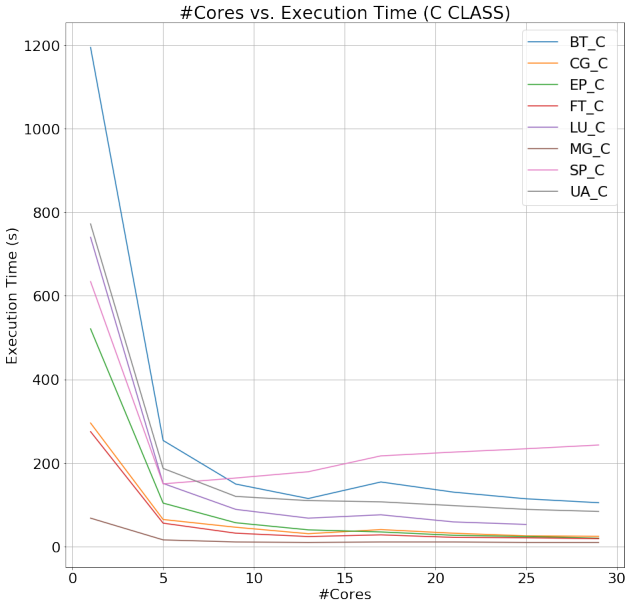


Figure 3.8: C Class Peformance

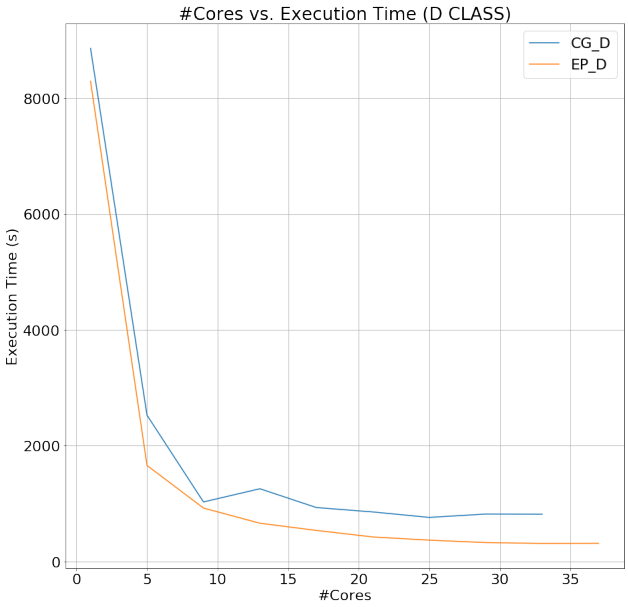


Figure 3.9: D Class Peformance

3.4.1 Results

First, the study looked at the effect of using multiple threads on performance. As these applications are highly tuned it was observed that all applications in Figures 3.7, 3.8, 3.9 benefit from additional threads, however, due to Amdahl's law and thread synchronisation overhead there is no improvement in performance after 10 cores for most applications regardless of their problem size class. The performance is also scaling linearly between the problem sizes classes when using low numbers of cores.

Secondly, the study progressed to the same experiments from an energy expenditure perspective as presented in Figures 3.10, 3.11, 3.12. Similarly, here it was found there are initial benefits of executing with more threads, but little improvement beyond 10 cores. It may seem counter-intuitive to see energy expenditure decrease when adding additional compute resources. This behaviour is a consequence of how power and time relate to energy as previously shown in equation 2.1. The time component dominates the equation, meaning increasing speed, decreases overall energy even if the power consumes is raised. This indicates a heuristic known as "race to idle" is beneficial,

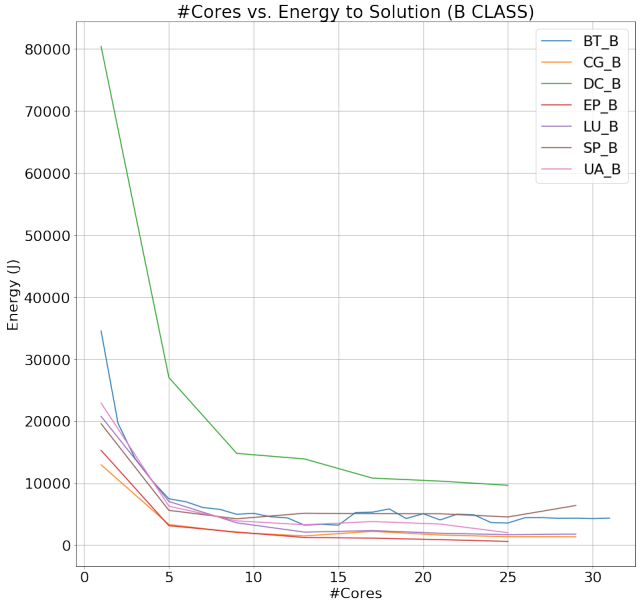


Figure 3.10: B Class Energy

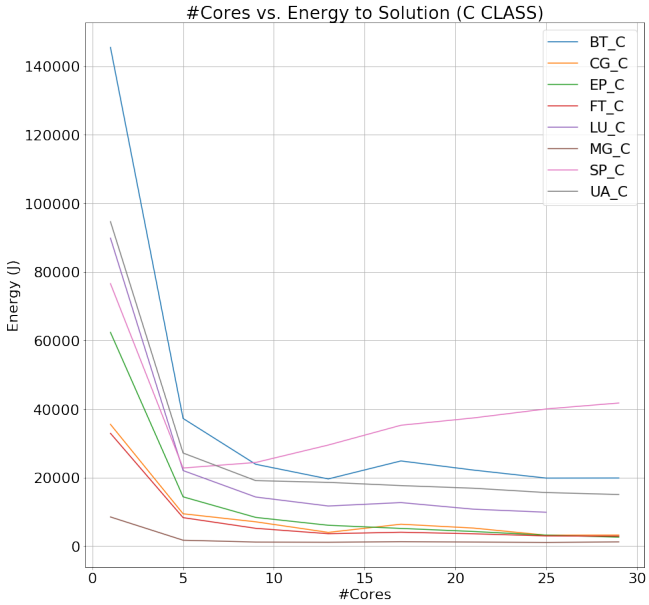


Figure 3.11: C Class Energy

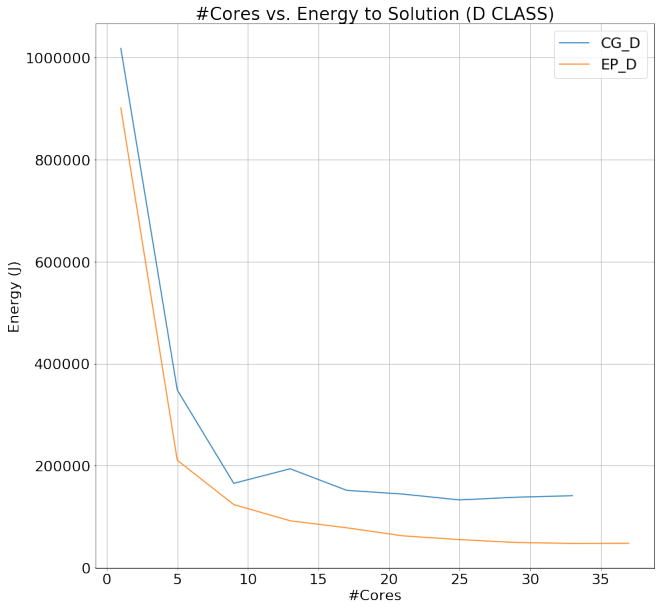


Figure 3.12: D Class Energy

Table 3.4: HCLEnergy API - MPI functions

struct host_power_series *gatherHostSeries(struct host_power_series *local);
struct host_power_series *getRawPowerSeries(hclenergy_t *event)
double *energy_per_host(hclenergy_t *event);

where speed may be maximised in order to decrease energy. However, this is not the case in all applications as is demonstrated in [168, 169]. It is our hope that HCLEnergyAPI will be helpful in furthering this and other areas of research.

Lastly, the power consumed during these experiments was considered. As can be seen in Figures 3.13, 3.14 and 3.15, power consumption is trending upwards as more threads are added. This is consistent with the operating system scheduling these threads in parallel in multiple cores, causing fewer of the cores to remain in lower power states.

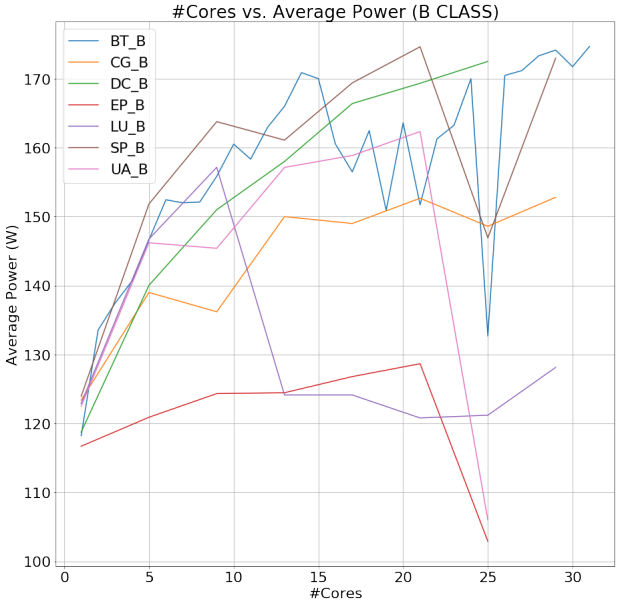


Figure 3.13: B Class Power

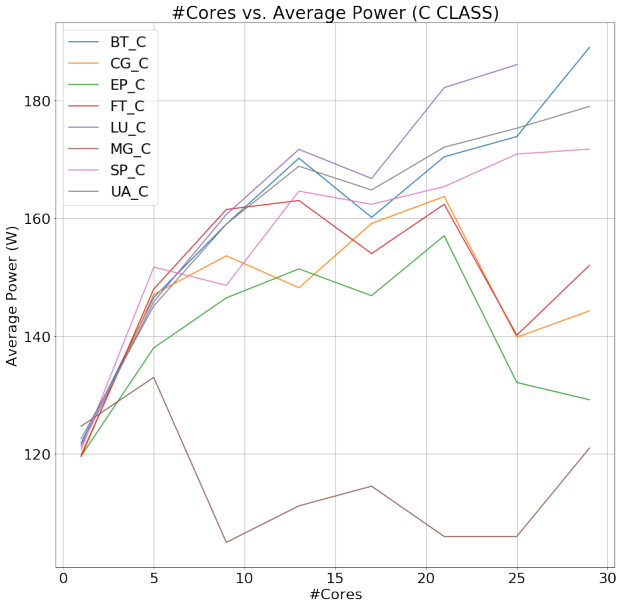


Figure 3.14: C Class Power

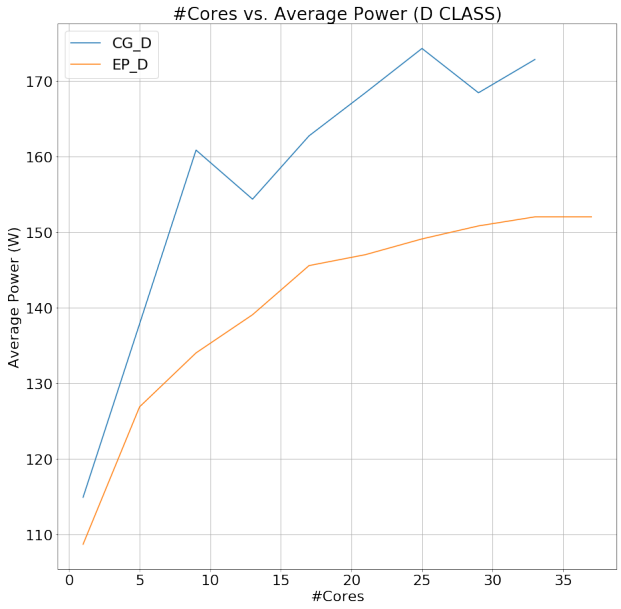


Figure 3.15: D Class Power

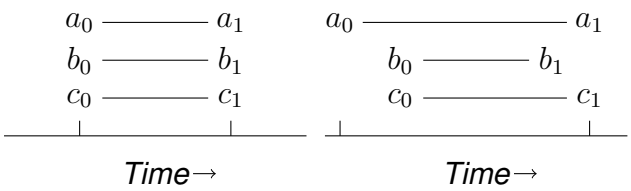


Figure 3.16: Multiple processes on single node, both cases

3.4.2 Measurement of Distributed Applications

Applications in the field of HPC are commonly distributed in nature. MPI is the prevalent technology for constructing distributed applications, so for that reason HCLEnergyAPI supports instrumentation of MPI based applications. The calls are identical to those described earlier, but in addition several collective operations are added. If the API is compiled with MPI enabled, the MPI functions in Table 3.4 are available. These functions make available a time series of power consumption for all nodes participating in the *event*, the energy consumed by each node by hostname and a collection of "raw" time series, which contain the power consumption of all nodes participating even when that particular node is not executing useful instructions at that point in time. This last feature is useful for observing the total power consumed for a distributed application when some of the nodes are left idling but still consuming power and is illustrated in Figure 3.16. In the left part of the diagram, we illustrate for nodes *a, b, c* the first of two cases. Here, the event completes in all nodes at the same time. The energy consumed is the sum of energy from all processes across all nodes. On the right, the event does not complete in the same time on all nodes due to some heterogeneity. In this case, only the power consumed by the nodes when they are active are measured , as this is the true energy of the application. However, if experiments are running in a HPC environment, there may be reserved nodes we are using, which aren't currently executing the application, therefore it may be relevant to know the total energy consumed for all nodes, from when the first node begins executing, until the final node has finished its work. This measurement, given by *getRawPowerSeries* (and integrated to calculate energy), provides this measurement and includes nodes which are drawing power, but potentially idle as they wait for other nodes to finish.

HCLEnergyAPI performs useful filtering operations on its data. As many processes may be executing on the same host, we deduplicate the data which overlaps. Duplication of readings occur, when more than one process is executing on a node and HCLEnergyAPI is called. Data from the processes running on the same node must be merged to avoid over-reporting the con-

sumption.

3.5 Conclusions and Future Works

In this chapter *HCLEnergyAPI* was presented. This project was developed to measure power and energy readings at a node level on large scale distributed systems. This prototype specifies the API for performing these measurements in a system agnostic fashion. As such, it should be possible to port this project to a wide variety of supercomputing infrastructure.

Chapter 4

Component Level Power Measurement

4.1 Introduction

In this chapter, measurement at a finer granularity than the node is investigated. Modern compute nodes contain one or more accelerator devices, ranging across GPU, manycore devices such as Intel's Xeon Phi accelerator, and increasingly FPGA [170]. As the majority of the top 10 of the TOP500 and GREEN500 lists [25, 22] include such an accelerator, there is motivation to examine the energy consumption of these devices.

Within the target research platform, there is an Intel Xeon Phi 3120P manycore accelerator card and an Nvidia Tesla K40c. Their specifications are described in Tables 4.1 and 4.2.

4.2 Background and Related Works

PAPI [171] exposes Intel RAPL counters and can be used to access Nvidia GPU power data. There are two approaches that can be taken using PAPI. First, the PAPI can be used to execute an application and capture power data for the whole application execution. Secondly, the PAPI API can be used to sample critical sections of code. The limitation of both approaches, is that

Table 4.1: Intel Xeon Phi 3120P Specfications

Feature	Value
Processors	57
Base Frequency	1.10Ghz
Main Memory Size	6GB GDDR5
L2 Cache Size	28.5MB
Memory Bandwidth	240GB/s
Memory Clock	3.0GHz
TDP	300W
Idle Power	91W
Technology Node	22nm

Table 4.2: Nvidia K40c Specifications

Feature	Value
Processors	2880 [1]
Base Frequency	0.745Ghz
Main Memory Size	12GB GDDR5
L2 Cache Size	1536KB
Memory Bandwidth	288GB/s
Memory Clock	3.0GHz
TDP	235W
Idle Power	68W
Technology Node	28nm

there is no statistical meaning to the measurements, leaving the user to devise their own approach to this problem.

Suda [172] reports the collection of power data from Nvidia GPU by intercepting the power rails of PCIe subsystem. There is no tool provided to replicate the process and the experimental hardware platform is not practical for widespread deployment.

Kindratenko [173] presents power data collected from AC inline power meters, which is then correlated against benchmark performance values. There is no tool provided.

Collange [174] measured the DC power of GPUs using an oscilloscope and clamp meter.

Huang [175] used a "Watts Up Pro" power meter to measure energy consumed while executing scientific kernels on GPU. This again, is an intrusive yet accurate approach and requires the use of a secondary monitoring computer.

Shao [176] instruments a Xeon Phi coprocessor directly to build a model of energy as a function of performance counters.

From these works, is it observed that there is no existing tool that facilitates the instrumentation of an accelerator for energy measurement, at a resolution of programmer specified critical sections, with statistical confidence.

4.3 HCLpower

Accelerator devices such as Intel's Xeon Phi and Nvidia GPU contain power measurement features, generally for the purposes of systems administration. Both vendors supply utility libraries that offer access to many interesting values, such as fan speed, temperature, power state and most interestingly to this work, instantaneous power consumption. In the case of Intel, the library is the "Manycore Platform Software Stack" or MPSS [177]. Nvidia distribute a similar library for their hardware called the "NVIDIA Management Library" or NVML [164]. We also have preliminary support for AMD GPU, but we do not have physical hardware with which to test. The software architecture is shown in Table 4.3.

```
#include <chrono>
#include <thread>

#include <HCLEvent.hpp>

using namespace std;
using namespace std::chrono;

int main(int argc, char *argv[]) {
  HCLEvent event;
  event.recordStart("SLEEP-10000");
  this_thread::sleep_for(std::chrono::milliseconds(10000));
  event.recordEnd();

  event.toFile(Device::GPU);
  event.toFile(Device::PHI);

  return 0;
}
```

Figure 4.1: HCLPower code example

Driven by the uptake of these accelerators [25], we wish to expose this energy data to a developer in a useful fashion. To this end, a major contribution of this thesis, a tool called *HCLPower* was developed.

HCLpower is an energy consumption profiler for accelerator compute devices. The tool is executed on the target application in order to collect power consumption timeseries as well as energy consumption. A key feature of the tool is that it provides the user the ability to profile specific sections of their code using *HCLEvent*. A user first creates a *HCLEvent* and then surrounds the code of interest with calls to a start and end method of this event.

When the tool is invoked, it forks the target application and begins polling the device for its instantaneous power consumption value at a configurable interval which is every 200ms by default. The tool collects these values as a timeseries, with each entry of the form of (*timestamp*, *watt*). After execution ends, the tool extracts the relevant data for each *HCLEvent* and calculates their power and energy values. Each event is created with a name string

Table 4.3: Software Architecture

HCLEnergyAPI		
NVML	MPSS	Future Others(AMD)

which allows it to be distinguished from the other events.

In Figure 4.1, the full source code of an application designed to measure the idle power of both supported devices is presented. The application's only thread sleeps whilst the *HCLpower* process continues to monitor the power activity. The output of executing this application is shown in Figure 4.2. This application has one event which consumes 619J $\pm$ 13.05. The application runs for 10 seconds and does not perform any computation on the device and therefore it draws a mean consumption of 61.9W at idle, in this case on this GPU.

When taking measurements it is desirable to report a statistical confidence in our results. A compute node has many processes other than our application, which we assume to all be running in the background. In order to filter out the effect of these processes on our measurements we execute our application multiple times until the average energy consumption falls within a user defined confidence interval. Energy is calculated by numerically integrating the measured power values over the event's execution.

More specifically, when a user executes their application with *HCLpower*, they must provide a confidence level (typically 90%, 95%, 99%). *hclpower* will execute the application multiple times, noting the energy consumed for each execution. After each execution, the average of all these energy measurements are taken, and a confidence interval is calculated on this sample mean. The executions are stopped only when the confidence interval is below a user supplied threshold (within a set number of joules), or we reach a user specified maximum number of executions. There is a requirement of a minimum of 20 samples for the measurement to be valid. The confidence interval is only calculable when the distribution of the measurements are gaussian. The tool will report an error if this condition is not met.

As the devices exhibit different levels of energy consumption, it is desir-

```
[ken@hclserver01 hclpower]$ ./test.sh
Sun Jan 29 18:04:09 GMT 2017
Running until Confidence of .99 for 250 J or
until 1000 iterations.
Iteration: 1
Iteration: 2
        624.1565 ± 0.676645035607  Joules
Iteration: 3
        623.911666667 ± 0.684568155375  Joules
Iteration: 4
        624.1485 ± 0.736695342816  Joules
Iteration: 5
        624.051 ± 0.63071325232  Joules
Iteration: 6
        606.100333333 ± 42.2124642885  Joules
Iteration: 7
        608.692 ± 36.7061805311  Joules
Iteration: 8
        610.88025 ± 32.5478026068  Joules
Iteration: 9
        612.403444444 ± 29.1669007762  Joules
Iteration: 10
        613.8003 ± 26.4712108421  Joules
Iteration: 11
        614.843181818 ± 24.2006547704  Joules
Iteration: 12
        615.63725 ± 22.2702013992  Joules
Iteration: 13
        616.377461538 ± 20.6385666803  Joules
Iteration: 14
        617.005857143 ± 19.2277518546  Joules
Iteration: 15
        617.581933333 ± 18.0030685902  Joules
Iteration: 16
        618.0963125 ± 16.9265621698  Joules
Iteration: 17
        618.605352941 ± 15.9815868789  Joules
Iteration: 18
        619.005222222 ± 15.1268755286  Joules
Iteration: 19
        619.338789474 ± 14.3551052383  Joules
Iteration: 20
        619.71465 ± 13.6699585875  Joules
Met confidence after  20  iterations
```

Figure 4.2: HCLpower output

Figure 4.3: GEMM Execution Time

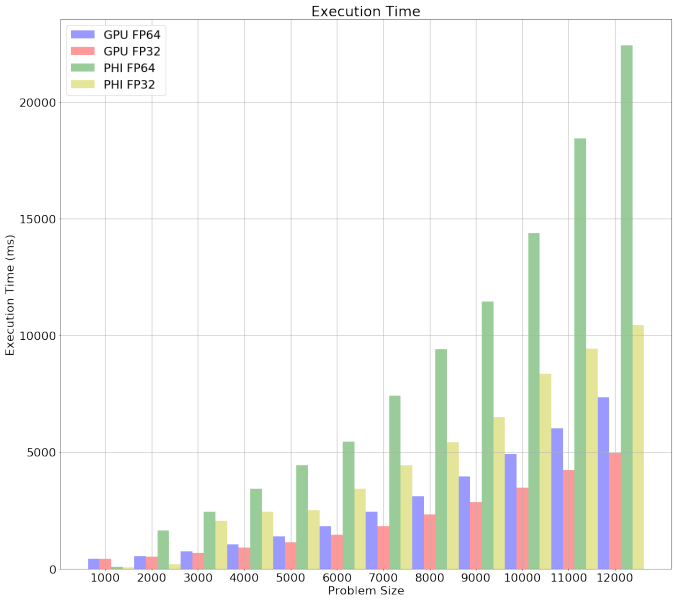


Figure 4.4: Energy consumed computing GEMM kernel

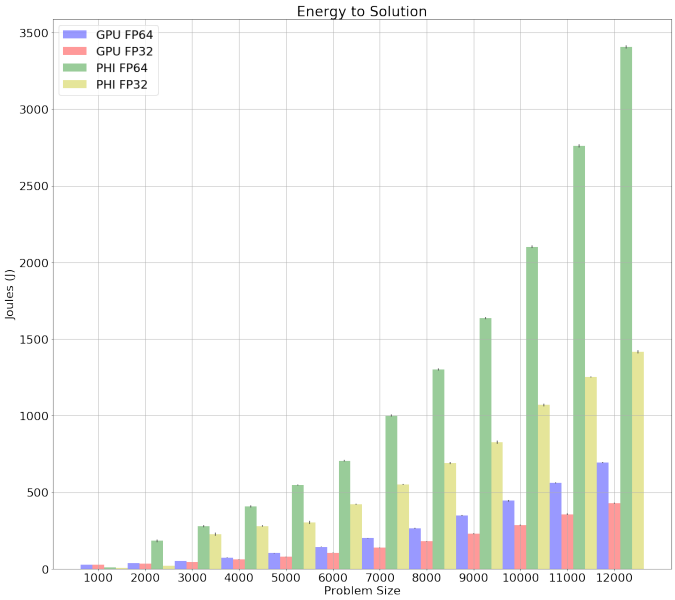
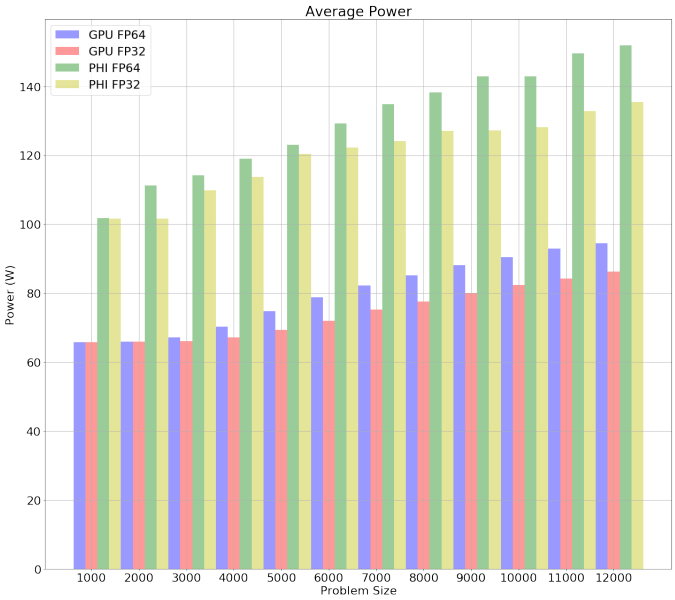


Figure 4.5: GEMM Average Power Consumption



able to know the idle power consumption of the devices. To achieve this, an application we distribute with *hclpower* called "*idlesleep*" can be profiled. This application simply calls the UNIX system call *sleep* for 1 minute while *hclpower* polls the accelerator for its power consumption values. For the Xeon Phi a value of 93.9W is measured and for the Nvidia GPU, 61.9W. Note however, that the act of polling the device keeps it in an active state. This observation was made when comparing this method to a direct measurement of the PCIe power rails using the ICHEC SEMA framework [29].

4.4 Case Study: Matrix-Matrix Multiplication

Matrix-Matrix multiplication is a fundamental computational kernel in scientific computing. It is commonly referred to as a GEMM (General Matrix Multiplication) within BLAS [178], the *defacto* linear algebra API. All major vendors of computational hardware provide an implementation of BLAS.

Given two matrices $\mathcal{A}$ of dimension $\mathcal{M} * \mathcal{K}$ and $\mathcal{B}$ of dimension $\mathcal{K} * \mathcal{N}$, GEMM produces the product matrix $\mathcal{C}$ of dimension $\mathcal{M} * \mathcal{N}$.

In these experiments the GEMM kernel is executed on varying problem size, by varying $\mathcal{M}, \mathcal{N}, \mathcal{K}$ on square matrices, so $\mathcal{M} = \mathcal{N} = \mathcal{K}$. Both devices are evaluated computing *float* (FP32) and *double* (FP64) precision formats of real numbers. The performance, energy and power consumed by GEMM on Xeon Phi and Nvidia Tesla GPU hardware is captured. The vendor optimised implementations of BLAS is called for each device. The HCLPower tool is used to measure energy and power.

In the case of the Xeon Phi, the gemm kernel is offloaded entirely from the host, such that none of the CPU are used in the computation. This is necessary to isolate the power consumed by the kernel to that which can be measured by our tool. [2]

As the execution speed is quite fast for matrix multiplication on these devices, all setup and teardown of the operation is measured. That is to say, memory allocation and initialisation time and energy is accounted for within the measurements.

4.4.1 Results

In Figure 4.3, the execution time of each device for both numerical formats is presented. It is observed that as problem size increases, so does execution time in all cases. Also, it is observed that the GPU outperforms the PHI for both numerical formats and that FP64 is always slower than FP32 in both devices. The performance difference between the two number formats depends on the architecture of the device. For the PHI, vector registers of a fixed width of 512b are used to compute both FP32 and FP64 precisions, calculating 16 or 8 operations per cycle respectively. This leads to the 2x performance difference once the matrices are large enough to saturate the hardware. In the case of the GPU, the Nvidia Kepler architecture has 3x as many FP32 units as FP64, leading to a greater acceleration of FP32 computation as shown in the figure.

The energy consumption of this experiment is shown in Figure 4.4, and greatly reflects the execution speed data from Figure 4.3. This is caused, as

[2] This is achieved by setting the environmental variables "MKL_MIC_ENABLE=1" and "MKL_MIC_WORKDIVISION=1.0".

discussed in previous section, due to the dominance of the time component of equation 2.1. The confidence interval is shown on each bar for a confidence of 99% with threshold within 0.5 J.

Figure 4.5 shows the average power consumption of this experiment. It is observed that within a device, the power consumption for either device are similar for a given problem size. As measured earlier in 4.3, the idle power of the PHI is 93.9W and the GPU idles at 61W. We consider these values static power which will always be consumed, regardless of device activity. The power values consumed in this experiment shows the dynamic component on top of this static component caused by the computation we have executed. The power increases due to wider utilisation of the compute resources in the devices and as the problem sizes increase, the number of memory banks performing work increases and with it their contribution to power consumption.

From these experiments it is found that for matrix multiplication of these sizes, for both numerical formats, the GPU is both the faster and more energy efficient solution. It is also found that power consumed by either device was significantly lower than the TDP of the device, demonstrating that calculating the energy efficiency of a device using Thermal Design Power (TDP) is not a valid approach.

Though the tool allows for easy measurement of critical sections of code, it was found that polling the device for power measurements keeps the device in a higher power state than if power is directly measured using intrusive and expensive infrastructure at the PCIe power rails. It is believed that this effect may alter measurements critical sections consuming low power and less likely to effect measurements in the case of the device being in higher power state than is brought about by the polling action. Future work would calibrate this tool to compensate for the polling action against a direct measurement infrastructure.

4.5 Conclusions and Future Works

In this chapter *HCLPower* was presented, a tool for collecting power and energy data from accelerators. It supports Nvidia GPU and Intel Xeon Phi accelerators. Data is collected for programmer specified critical sections of their application, such as key compute kernels. The use of this tool to instrument the GEMM kernel on both accelerators for various problem sizes was presented.

HCLPower targets accelerator devices, though it is easily extensible to support any device which supplies component level power values, for example the Intel RAPL performance counters which provides data for CPU and DRAM components [160, 171].

In future it is expected that FPGA devices will have the capability to expose power data. At that time, support for those devices will be extended in the tool.

As stated previously this approach could be enhanced by a calibration against a direct measurement approach to calculate an offset to compensate for perturbing the device.

Chapter 5

Accelerator Performance and Power Modelling

5.1 Introduction

In previous chapters an investigation into node level and accelerator level power measurement was conducted. In this chapter a methodology for choosing the optimal accelerator device with respect to performance and energy efficiency is presented.

As development of accelerated kernel code is a time consuming exercise, a semi-automated tool flow for determining the expected maximum performance of an algorithm on a particular accelerator was developed as a contribution. The results of this tool flow frees a developer to concentrate on developing and optimising a kernel for the most appropriate device, instead of exhaustively developing for all accelerators or choosing an accelerator based on coarse data sheet performance values, which may not accurately reflect which accelerator would yield the best achievable performance for their chosen algorithm.

Comparing multiple accelerators for performance is a challenging problem. Different devices implement different Instruction Set Architectures (ISAs), memory hierarchies and models, and there are no common profiling tools available.

This analysis leverages the Berkeley Roofline model [179] to calculate the

peak performance of the hardware device and it allows for the prediction of what proportion of this peak, can be achieved given an algorithm.

A number of metrics based on the OpenCL computation model are derived for the first time, for our comparisons. OpenCL was chosen for this study as it allowed for a common C++ application to run on a standard CPU in our compute node and an OpenCL kernel executing on any of the three accelerator devices in our platform. Furthermore, OpenCL standardises the metrics we from which parameters for the roofline model can be derived.

The collection of these metrics has been semi-automated by use of an OpenCL device emulator called OCLgrind [180]. In this chapter the use and benefit of this flow is demonstrated by applying it to an example application on three distinct accelerators.

The novel contributions are:

- A mapping from OpenCL terminology to the roofline model for three accelerator devices, including a novel FPGA based accelerator.
- A proposed semi-automated toolflow for measuring and comparing performance and power.
- An initial evaluation of this flow using a target application on three accelerator devices.

5.2 Background and Related Works

The state of the art approach for selecting an accelerator is often to choose a device that has the highest data sheet performance value for floating point. This may not reflect the achievable performance on that device as the application may be predominantly integer based, or memory bandwidth may constrain the achievable performance.

Additionally, it is common to choose accelerators with lower TDP if energy efficiency is a concern, though this choice does not guarantee an energy efficient solution, if the device performs poorly for each watt it consumes.

This work leveraged multiple benchmarks to obtain multiple metrics for the target accelerator. The SHOC L0 benchmark [181], STREAM [182] and LINPACK [18] benchmarks were used to obtain measured values on our accelerators.

There are many other benchmarks available, such as NASA NAS [1], Rodinia[183] and OpenDwarfs [184]. They were considered for obtaining performance and power metrics, but they only target a subset of the available target devices. Although when conducting the experiments, some of these benchmarks could target the Xeon Phi and FPGA through OpenCL, the kernels themselves were optimized for GPU architectures.

This work builds on the roofline [179] model. There have been notable extensions to this work, notably a cache aware extension that integrates the effect of cached data on the expected performance [185]. As there is no data reuse in the application we studied, we leave the integration of this extension as a future work.

The Spiral project [186], which heavily inspired this work extended the roofline model by demonstrating the use of measured performance values for performance bounds within the model instead of theoretical datasheet values. This extension increases the practical use of the model by more accurately accounting for the bounds in which the user is working. Without this extension, the user could find themselves trying to surpass a performance which is theoretically achievable, but due to unknown architectural implementation techniques, impossible to attain.

From an energy modelling point of view, Choi [187], extended the roofline to examine the relationship between time-efficiency and energy-efficiency and developed a mechanism for determining whether a race to halt strategy is the most efficient approach for a given application.

5.3 Preliminaries

In this section the preliminaries of our flow is described, starting with the OpenCL model [188]. OpenCL is an open compute standard specified by

the Khronos group, which is the foundation on which this work is built. Within OpenCL, there is a concept of a host, which is a computer containing a CPU and some memory, and secondly the concept of an accelerator, which is usually a PCIe attached device where accelerated code is executed. There are two core relevant models in OpenCL for our flow. They are the compute model, which describes scheduling and work allocation and the memory model, which describes how data is accessed and shared between parallel workers.

5.3.1 Compute

OpenCL is specified as an offload model parallel framework. Primarily, this means that a specific kernel of computation within a CPU based application, is identified as possibly benefiting from acceleration. The compute kernel is extracted from the application and rewritten as an OpenCL kernel, in a C99 like syntax. The application then transfers data to accelerator device and executes this kernel on it, and then returns the data to the host program.

A key feature of this language is explicit thread parallelism. The kernel is written in such a way as to specify what a single thread, given a unique thread ID will execute. A thread executing one of these kernel instances is defined as a work item. Typically, for performance reasons, work items are launched together in parallel and scheduled by the OpenCL runtime onto parallel hardware. This batch of work items is known as a work group. As will be seen in the memory section, work items within a work group have certain benefits in terms of data sharing.

Kernels can be expressed as having multi dimensional thread indices. For example, a two dimensional index, representing the column and row of a matrix, would be useful in expressing a matrix multiplication kernel. Work items would be issued to compute all values of the resulting matrix, by multiplying row x by column y. This is illustrated in Figure 5.1

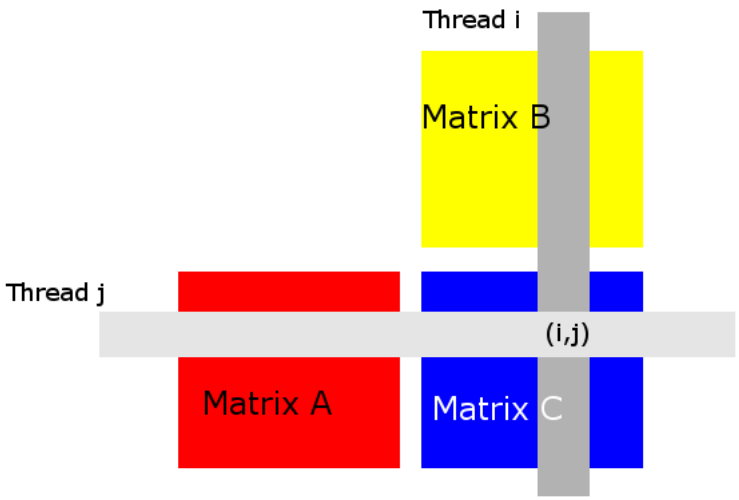


Figure 5.1: OpenCL Matrix Multiplication

5.3.2 Memory

The OpenCL specification [188], defines four distinct memory types from the perspective of a computational kernel. The first, is *global* memory, which is a large Random Access Memory (RAM), accessible by all work items in any kernel execution. Accessing *global* memory is typically expensive in terms of memory bandwidth, as it is most often constructed from memory technology favouring size over speed, such as Double Data Rate 3 (DDR3) and Double Data Rate 4 (DDR4). According to the specification, all data transfer between the host and the accelerator must use *global* memory as a buffer.

The second memory type is *local* memory. This is a typically a faster, yet significantly smaller memory. Typically *local* memory is implemented as a cache memory. When a kernel allocates local memory, only work items within a workgroup can access data within that *local* memory. *Local* memory is used as an explicit cache for global memory when reuse is known to occur or when the computational kernel requires data from neighbouring work items as an input to its execution.

The third memory type is *constant* memory which is simply *global* memory but with the promise on the developer's behalf, that the data will not be modified, allowing the compiler to make data access optimisations.

The final memory type is *private* memory which is the default for all memories which are not annotated. *Private* memory variables are not shared between work items and are essentially stack variables within a work item.

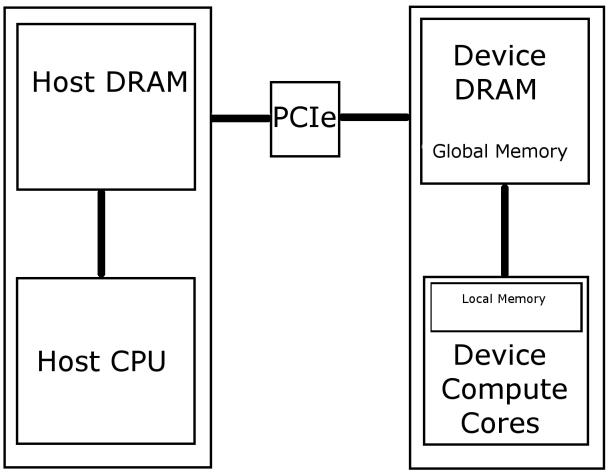


Figure 5.2: OpenCL Memory Model

These memories are illustrated in relation to the host, that is, the CPU based system on the left side of Figure 5.2.

5.4 Roofline Model

First proposed by researchers at Berkeley for multi-core processors, the roofline model[179] models the peak achievable performance attainable for any computation device with respect to its compute ability bound by its memory bandwidth. The maximum performance a device can achieve is dependent on a number of factors. Taking a CPU as an example, the number of operations a CPU can execute is calculated by equation 5.1. As seen in this equation, the number of operations a CPU can execute per cycle depends on the operation. Most operations can be executed at a rate of 1 per cycle, however CPU vector technology, such as Intel AVX2 [189], Power Altivec [190] and ARM Neon [191], can execute up to 8 32bit additions per cycle. Support for vector instructions varies by processor and instruction.

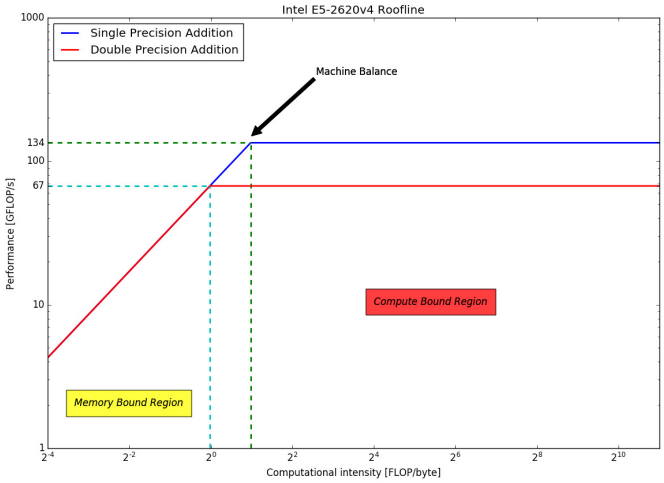


Figure 5.3: Roofline of Single and Double Precision Additions on Intel E5-2620v4

The second term, is simply the clock frequency. It should be assumed that the processor is operating at its C_0 and P_0 states, i.e., fully active and at maximum frequency, as this is the best case for achievable performance. The third and fourth terms are the number of cores per socket and the number of sockets in the system.

For a given operation and processor the peak theoretical compute performance can be calculated. Taking an Intel E5-2620v4 [192], which has 8 cores, and runs at 2.10Ghz, using its AVX2 instructions, it could calculate 8 32bit additions per cycle, resulting in a peak theoretical performance of 134.4GFLOPS.

This theoretical performance does not however take into account whether or not the processor can access the input data sufficiently quickly. The maximum memory bandwidth attainable by this processor is 68.3 GB/s, which depending on the application, may limit the attainable peak performance. The resulting roofline is seen in Figure 5.3

$$OPS = \frac{ops}{cycle} * \frac{cycles}{second} * \frac{\#cores}{socket} * \#sockets \tag{5.1}$$

5.4.1 Fundamental Metrics

In this section the metrics used in this analysis are described. Fundamental metrics are those that can be measured, for example using a benchmark or profiling tool. Subsequently *"Derived metrics"*, those that are formulated from the roofline model are described.

Device Memory Bandwidth ($\mathcal{B}$): is the memory bandwidth from global to the compute cores of an OpenCL device as specified by the manufacturer's data sheet. **Device Peak Memory Bandwidth ($\hat{\mathcal{B}}$)** is the memory bandwidth measured by a benchmark kernel where the data types being moved are identically sized to the kernel under investigation. For example, if a primarily integer based kernel is considered, a memory benchmark that moves integers is required. The unit of both these metrics is 'Bytes/Second'.

OpenCL Kernel Operations ($\mathcal{W}$): is defined as the number of operations performed by an OpenCL kernel. As many devices implement specialised vector based instructions, this methodologies relies on the count of the number of mathematical operations performed and not the number of instructions executed. This facilitates fairer comparisons. The operations counted are those that provide meaningful computation, such as arithmetic or comparisons, but not branching or those related to data movement, such as read/write operations.

Some modern compute devices support hardware for specialised instructions such as the number of bits set in a data word (population count). In order to count these fairly, they are counted as 'base' operations as if they were computed using basic arithmetic operations in software. There are two methods to obtain $\mathcal{W}$. The first is a static code analysis, where the developer examines a pseudo code of their algorithm and counts the appropriate operations. The second is, if an OpenCL implementation of the kernel is available, it can be based to the OpenCL device simulator OCLgrind [180] which can extract a SPIR representation of the kernel and report what SPIR level operations would be executed. Standard Portable Intermediate Representation (SPIR) is an intermediate presentation used by compilers which we can inspect to account for the mathematical instructions expressed to the code generation phase of

the compiler. This SPIR representation should be common to all OpenCL devices and representative of what any OpenCL device would execute. The unit of $\mathcal{W}$ is operations.

OpenCL Kernel Global Memory Traffic Size ($\mathcal{Q}$): is the number of bytes which travel between the off-chip DRAM memory and on-chip memory such as OpenCL local memory or registers, during a kernel execution. The value of $\mathcal{Q}$ depends on both the kernel and the target device. Given a kernel of identical problem size to several devices, one device may be able to keep resident all data in local memory for the duration of the kernel's lifetime, whereas another may not. The second device would have a higher value of $\mathcal{Q}$, as it would need to fetch spilled data back into the compute cores more than one time.

In this tool flow, $\mathcal{Q}$ is obtained through a combination of OCLgrind memory access counters and by values obtained by the accelerator vendor's profiler. In these tests, these profilers include Intel VTune [193], Nvidia's nvprof [194] and Xilinx's SDAccel [195]. The value of (Q) is measured in Bytes.

OpenCL Kernel Execution Time ($\mathcal{T}$): is the execution time for a given kernel execution. This value does not include the setup of the OpenCL environment, nor the transfer of data for the computation from the host to accelerator DRAM. It measures, the time in 'Seconds' from OpenCL's 'CL_PROFILING_COMMAND_START' event until the 'CL_PROFILING_COMMAND_END' event, representing the time the kernel is executing only. This value does however include the time reading from global memory and writing any results back into the device's global memory.

OpenCL Kernel Power Consumption ($\mathcal{P}$): As seen in chapter 4, modern accelerators expose power monitoring functionality, through there vendor libraries [164, 177, 196]. In this model, only the **peak power** is measured as it is considered the peak value to be worst case power consumption for the measured kernel. $\hat{\mathcal{P}}$, the **accelerator** peak power, is also reported, measured by taking peak power while executing the same memory benchmark as used to calculate $\hat{\mathcal{B}}$.

5.4.2 Derived Metrics

From these fundamental metrics the following metrics are derived which facilitates the construction of a combined analysis of a device and application.

Operational Intensity ($\mathcal{I}$): is a measure of how much computational operations are performed on average on each byte of data during a kernel execution. It is the ratio of OpenCL Kernel Operations ($\mathcal{W}$) to the OpenCL Kernel Global Memory Traffic Size ($\mathcal{Q}$). The units of $\mathcal{I}$ is 'OPS/Bytes'.

$$\mathcal{I} = \mathcal{W}/\mathcal{Q} \tag{5.2}$$

Energy ($\mathcal{E}$): represents the energy consumed by the kernel during per execution. It is the product of execution time ($\mathcal{T}$) and peak power($\mathcal{P}$).

$$\mathcal{E} = \mathcal{T} * \mathcal{P} \tag{5.3}$$

Performance ($\mathcal{F}$): is the performance of the kernel. It is computed as the ratio of operations performed ($\mathcal{W}$) to the execution time of that kernel ($\mathcal{T}$). It is measured in units of Operations per Second (OPS).

$$\mathcal{F} = \mathcal{W}/\mathcal{T} \tag{5.4}$$

Performance Per Watt ($\mathcal{R}$): is the energy efficiency if the kernel on the given device. It is computed as the ratio of the performance of the kernel ($\mathcal{F}$) to the peak power consumed by the kernel ($\mathcal{P}$).

Energy efficiency is a useful metric as it conveys how much work is done by the energy put into the system. Devices consuming large amounts of power but achieving a high performance can still be more energy efficient than low power devices contributing less performance.

$$\mathcal{R} = \mathcal{F}/\mathcal{P} \tag{5.5}$$

5.4.3 Data Type Specialisation

For each metric presented, subscripts are used to denote the data type with which it is representing. For integer data '$_i$' is used and for single precision floating point, '$_f$' is used. This approach can be extended to arbitrary data types, once the key characteristics of data type, such as width in memory and device computational throughput is captured.

There must be a differentiation between data types as accelerator devices often have unbalanced throughput for the same mathematical operation for differing data types. This difference is a result of micro-architectural design decisions of the device vendor. Some devices, such as GPUs for example are targeted at high performance floating point arithmetic.

Currently this model only supports kernels of a single data types. In future works we will generalise the model so that kernels containing multiple data types may be more accurately modelled.

$$A \in \{\mathcal{W}, \mathcal{I}, \mathcal{F}, \mathcal{R}\} \tag{5.6}$$

A_i : refers to integer operations
A_f : refers to single precision floating point operations

$$M \in \{\mathcal{B}, \mathcal{P}, \mathcal{E}, \mathcal{F}, \mathcal{R}\} \tag{5.7}$$

$\hat{M}$: refers to values obtained from benchmarking
$\bar{M}$: refers to values obtained from datasheets

5.5 Methodology

In this section the details of a methodology for choosing the most appropriate accelerator for an application is presented as a contribution, beginning with the combination of the Berkeley Roofline model with the metrics described previously in section 5.4.

5.5.1 Tool Flow

In this section the two step semi-automated toolflow is described.

Stage 1: Device Analysis

The first of the two stages is the *"Device Analysis"*. In this stage the benchmarks are executed on the target device to obtain the roofline model device parameters $\hat{\mathcal{B}}, \hat{\mathcal{F}}$ and $\hat{\mathcal{P}}$. More specifically, the steps are listed below:

- The toolflow measures the peak performance and memory bandwidth of the device using benchmarks to acquire $\hat{\mathcal{F}}_f$, $\hat{\mathcal{F}}_i$, $\hat{\mathcal{B}}$ and $\hat{\mathcal{P}}$ for a given device.
- Given a range of operational intensities $(\mathcal{I}_{min}, \mathcal{I}_{max})$ a performance roofline is constructed using equation5.8
- Using the same approach, a performance per watt roofline is constructed using the equation 5.9, which graphs performance per watt.

$$min(x \times \hat{B}, \hat{F}) \forall x \in (I_{min}, I_{max}) \tag{5.8}$$

$$min((x \times \hat{B})/\hat{P}, \hat{F}/\hat{P}) \forall x \in (I_{min}, I_{max}) \tag{5.9}$$

Stage 2: OpenCL Kernel Analysis

Having obtained the device metrics, the next step is to consider the kernel.

The steps proposed are:

- The OpenCL kernel is compiled to the SPIR language and is executed using the OCLgrind [180] tool. OCLgrind counts the number of SPIR instructions that are executed, yielding the $\mathcal{W}$. The memory access data is also recorded.
- In the next step, the device specific profiling tools are used to measure the $\mathcal{Q}$, that is the memory traffic from global memory of a kernel.

- The toolflow then executes the kernel and measures the time of execution giving $\mathcal{T}$.
- The toolflow samples power consumption at regular intervals using measurements acquired from the *hclpower* tool presented in chapter 4, giving us $\mathcal{P}$
- Using equation 5.2, $\mathcal{I}$ is obtained, facilitating the graphing of the predicted max performance on the target device.
- Using equations 5.4 and 5.5, the measured performance and energy efficiency of the kernel, which are $\mathcal{F}$ and $\mathcal{R}$ in our toolflow are calculated.

5.6 Case Study

In this section a case study of applying this toolflow is described. The algorithm and the target devices are discussed, followed by the construction of the rooflines resulting from this toolflow. Finally the results are presented.

5.6.1 Algorithm

The algorithm of interest in this study is the *lookup3* [197] hashing function by Bob Jenkins. It is the core hashing function within the memcached [198] key-value store. This algorithm was chosen as it is called frequently within the larger memcached application and is sufficiently complex to evaluate the 3 devices on which the experiments were carried out. The algorithm has many possible execution paths, which can be taken depending on the input. Given the maturity of the tooling at the time of the experiments, a more complex algorithm would have resulted in longer development time.

Given a key string and a value object, a key-value store will hash the key to determine the address of where to store the object. This is known as a *SET* operation. The second core operation, a *GET* operation, is the retrieval of an object. In this case, the key-value store is given a key; the hash is computed and the object at this hash's address is returned from the system. The primary

use of these systems are to act as caches for data returned from databases, to lessen the amount of unnecessary database computation.

The algorithm operates by taking three input values; the key, the length of that key and a random seed known as *init*. No matter the length of the key, the resulting hash is always a 32 bit integer. The algorithm is composed of two main stages. First, there is a loop which consumes three 32 bit words, or twelve characters of the key, and performs a *Mix* operation composed of multiple operations to shift, xor, add and subtract the *state* variables $\{a, b, c\}$. This loop continues until there is less than twelve characters of the key remaining. Then, the second stage processes the remaining characters with a length dependent mixing of the characters. Finally, the resulting hash value is returned as state variable c.

Algorithm 1: Bob Jenkins `lookup3` hash function

```
key ← Input string to hash ;
length ← Length of input string ;
init ← Initialization value of the hash ;
hash → Returns the hash value ;

begin
    a, b, c ← Initialize based on length and init ;
    index ← Index of the key ;
    while length > 12 do                                   // mixing
        a += key[ index + 0 ] ;
        b += key[ index + 1 ] ;
        c += key[ index + 2 ] ;
        Mix ( a, b, c ) ;
        length −= 12 ;
        index −= 3 ;
    end
    ; /* Mix the remainder in a, b, c                          */
    /* and return the hash                                     */
    return MixRemainder ( a, b, c, key, length ) ;
end
```

The $\mathcal{W}$ and $\mathcal{Q}$ values are obtained for this application by using OCLGrind [180] on a uniformly random test dataset as 1224 MOPs and 367MB

respectively. Having both values, the arithmetic intensity is obtained ($\mathcal{I}$) as 3.33.

5.6.2 Target Devices

Inspired by the proliferation of accelerator devices across the Top500 [25] supercomputers, to assess this toolflow three diverse accelerators are targeted. They are two traditional accelerators, such as an Nvidia K20c GPU and an Intel Xeon Phi 5110P, and a novel accelerator, a Xilinx Virtex 7 based ADM-PCIE-7V3 FPGA board. In this section the capabilities of the devices and their mapping to the Roofline Model are described.

Nvidia Tesla K20c

The Nvidia K20c is a Kepler architecture [199] based GPU for HPC in a PCIe form factor. It has 13 SMX cores, totalling 2496 CUDA cores. Each SMX has the capability to execute arithmetic operations in parallel. The number of operations per cycle depends on the complexity of the operation. This device can compute 32 integer multiply-accumulate operations per clock cycle. In the case of Nvidia GPU the intruction throughputs depend on the microarchitectural generation of the device [200].

The device executes at a base clock frequency of 0.706Ghz. This value is used for the calculations in this work, though when certain conditions dependent on thermal environment and compute workload are fulfilled, the device can operate at a boosted clock frequency.

In OpenCL terms, the device's global memory is composed of 5GB of GDDR5 memory. The peak memory bandwidth is given from the datasheet at 208GB/s. This is the value for $\bar{\mathcal{B}}$ for this device. It is obtained by multiplying the memory interface of data width of 320 bits by the memory clock frequency of 5.2Ghz and converting to GB units.

The device includes 64KB of L1 cache (48KB of which can be configured as a shared memory) per SMX totalling 624KB. There is a 1536KB shared L2 cache between all of the SMX. The combination of L1 shared memory and L2

cache memory gives the a total of 2160KB local memory.

$$\mathcal{F}_f = 0.706Ghz * 13 * 192 * 2 \tag{5.10}$$

$$\mathcal{F}_i = 0.706Ghz * 13 * 32 * 2 \tag{5.11}$$

For this device, $\mathcal{F}_i$ and $\mathcal{F}_f$ are calculated in units of giga-operations by the Equations 5.11 and 5.10. These are calculated as the product of the clock frequency, the number of SMX cores within the GPU, the number of instructions per SMX per cycle and the number of mathematical operations per instruction. Typically this last component is 2, as the operation used for calculating maximum performance is the fused multiply add instruction, as de facto standard.

Intel Xeon Phi 5110P

The Intel Xeon Phi 5110P is 60 core PCIe based accelerator implementing a subset of the x86 ISA running at a clock frequency of 1.053 Ghz. Mapping to the OpenCL model, this device has 8GB of GDDR5 memory, constituting the global memory. The local memory is comprised of 32KB of L1 and 512 KB L2 cache each per core. The L2 caches are connected via a ring interconnect. Cache misses on one core's L2 result in a lookup of the other core's L2 caches, before accessing global memory.

The Intel Xeon Phi 5110P cores have 512 bit wide vector registers facilitating the execution of the same instruction on multiple data (Single Instruction Multiple Data (SIMD)).

$$\mathcal{F}_{f,i} = 1.053Ghz * 59 * 16 * 2 \tag{5.12}$$

Due to it's relatively low clock frequency (compared to CPU), this device relies heavily on vectorising computation within its vector registers to achieve high performance. The throughput performance values $\mathcal{F}_i$ and $\mathcal{F}_f$ are computed similarly to the Nvidia K20c device. Note, though the Phi has 60 physical cores, one is reserved for running the embedded operating system on the board. For this device, $\mathcal{F}_i$ and $\mathcal{F}_f$ are equal. The performance is calculated

as the product of the clock frequency, the number of cores, the operations per instruction, per core, per cycle and finally a factor of 2, again because of the fused multiply add operation. The result is presented in Equation 5.12.

The memory subsystem of this device is comprised of 8 channels of 64 bit interfaces, executing at 5.5Ghz, giving a total maximum theoretical global memory bandwidth $\bar{\mathcal{B}}$ of 352GB/s.

Xilinx ADM-PCIE-7V3

The Xilinx ADM-PCIE-7V3 is a half length PCIe based accelerator manufactured by AlphaData, containing a Xilinx Virtex 7 690T FPGA. Unlike the previous two accelerators, the FPGA has no fixed architecture. It cannot be described in terms of cores or SIMD units. The FPGA is a fabric of building blocks, composed of Lookup Tables (LUTs), Flip-flops (FFs), Digital Signal Processors (DSPs) and Block Random Access Memorys (BRAMs).

Typically these resources are mapped to specific roles. LUTs for example are used to implement arithmetic operations on integer based data types. They are implemented as a configurable truth table. Instead of building an operation from multiple logic gates, LUTs can be used to define any boolean operation, from 6 inputs in this generation of FPGA. Flip-flops are register memories. Each flip flop can hold a single bit in memory. They are the mechanism by which state is held within the FPGA between clock cycles. DSPs are specialized logic for performing signal processing operations at high speed. They are often employed to perform floating point arithmetic within the FPGA. BRAMs are blocks of dedicated memories usually of 36KB in size for Xilinx based devices. These BRAMs are used to construct a custom memory hierarchy within the user's design. A developer can use these BRAMs to construct on-chip caches when deemed appropriate.

The FPGA is programmed by expressing a design in either Register Transfer Logic (RTL), which are low level Hardware Description Languages (HDLs) in which the developer describes their design at the level of wires and signals which propogate on clock inputs, High Level Synthesis (HLS) tool such as Xilinx Vivado HLS C++ [201] in which the design is described through C++ with

Device	$\bar{F}_f$	$\bar{F}_i$	$\bar{B}$	$\bar{P}$	$\hat{F}_f$	$\hat{F}_i$	$\hat{B}$	$\hat{P}$
Tesla K20	3524	587	208	225	2903	585	143	225
Phi 5510P	1988	1988	320	245	1189	946	119	245
ADM 7v3	738	8880	21	25	200	3032	8.5 [2]	25

Table 5.1: Tool flow values of lookup3 on these test devices

pragma to direct the tool to generate the desired hardware or as in our case OpenCL. This code does not execute on the device, rather, it describes the hardware architecture that the FPGA will instantiate. FPGA tools such as Xilinx Vivado, transform the code into a binary form, which is used to reconfigure the hardware in the FPGA to operate on data as described by the code.

When mapped to the OpenCL model, this FPGA device has 16GB of DDR3 memory, which constitutes the global memory. The local memory is composed of BRAMs.

Xilinx SDAccel 2015.1 [195] was used which constrained our resource availability to 70%, as platform infrastructure such as PCIe communication hardware and memory controllers for the onboard DDR memories occupied the remaining 30%. Within this version of the SDAccel tool, the target clock frequency for this device is set to 200Mhz.

The 690T FPGA contains 433200 LUTs, 866400 FFs, 3600 DSPs and 52MB of BRAMs.

5.6.3 Rooflines

For each device the rooflines are plotted to demonstrate the maximum achievable performance and the performance/watt for integer operations. Based on the previous subsection, the required metrics are summarised in Table 5.1[1].

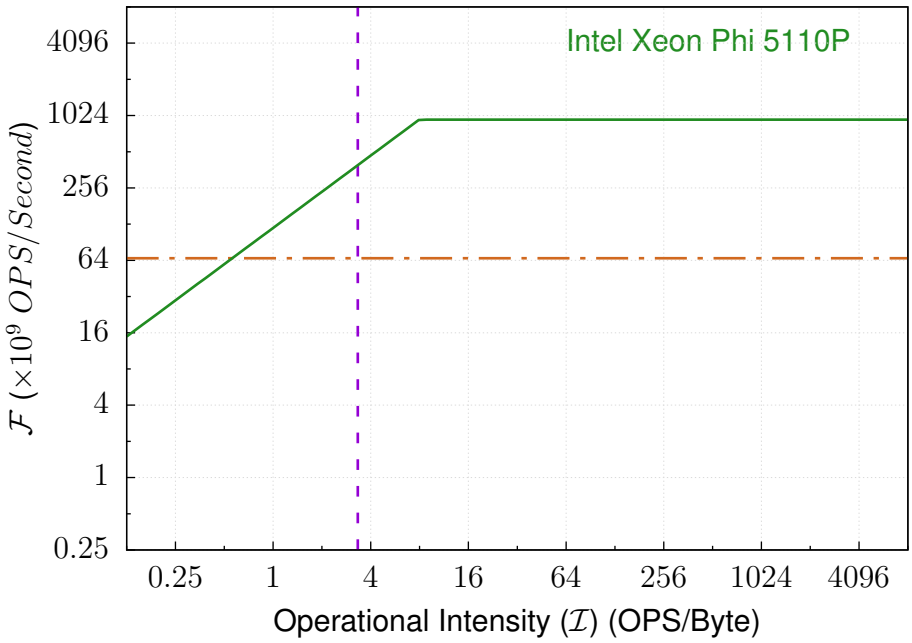


Figure 5.4: Intel Xeon Phi Performance

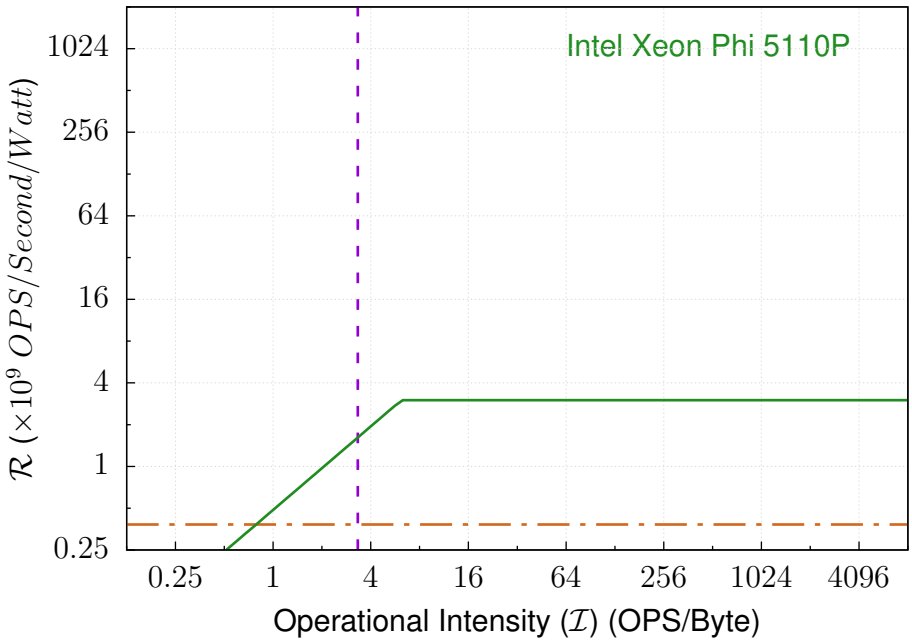


Figure 5.5: Intel Xeon Phi Performance/Power

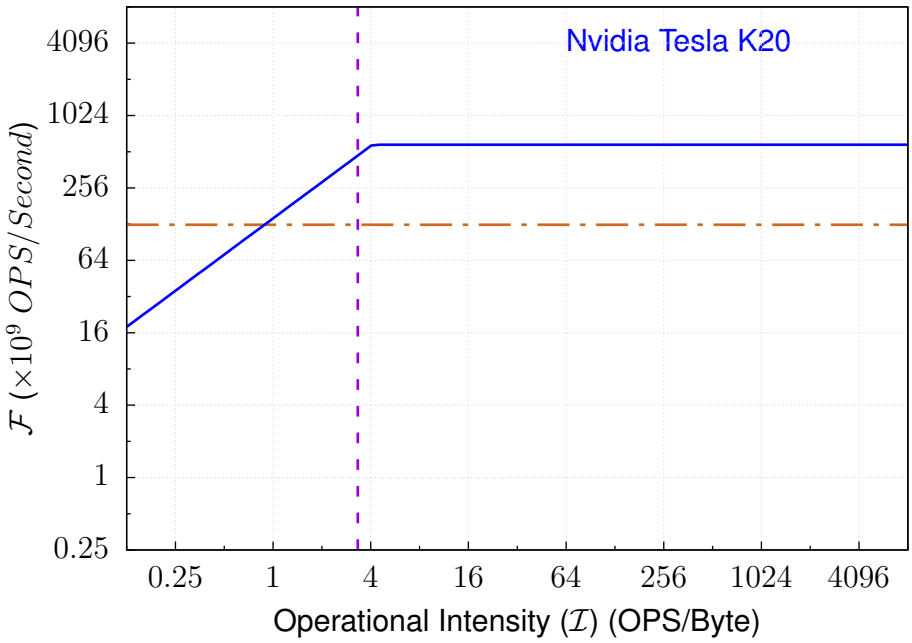


Figure 5.6: Nvidia K20 Performance

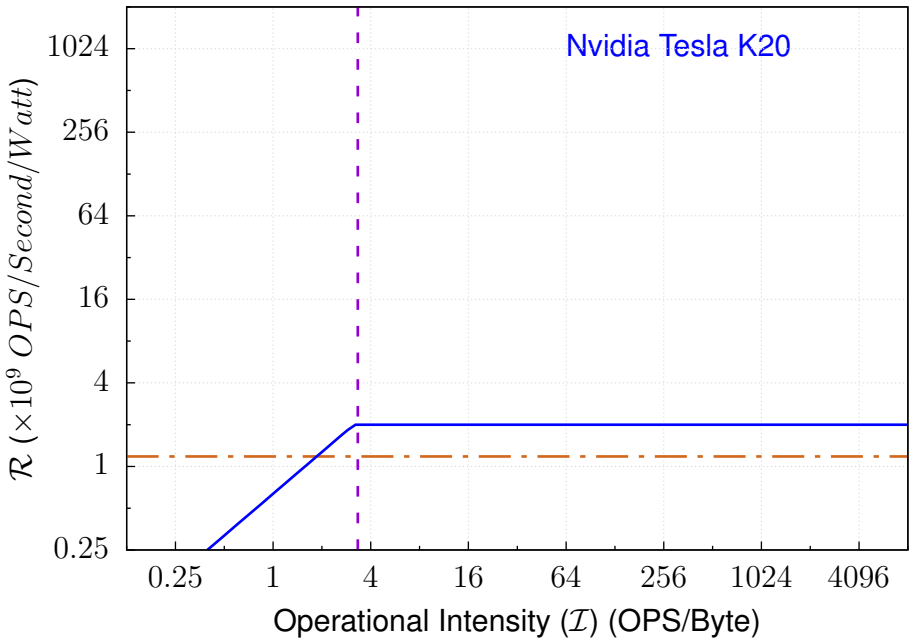


Figure 5.7: Nvidia K20 Performance/Power

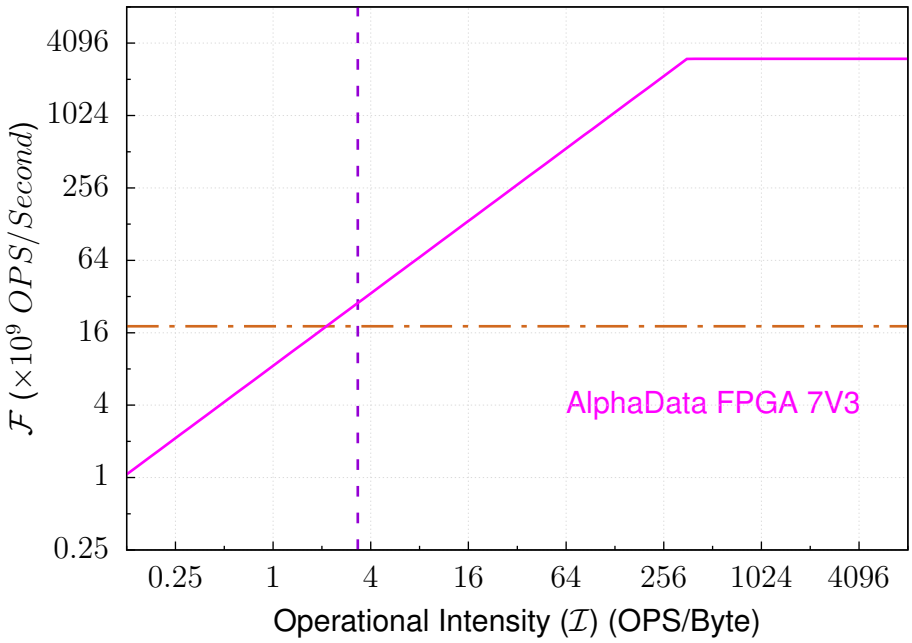


Figure 5.8: Xilinx Virtex 7 Performance

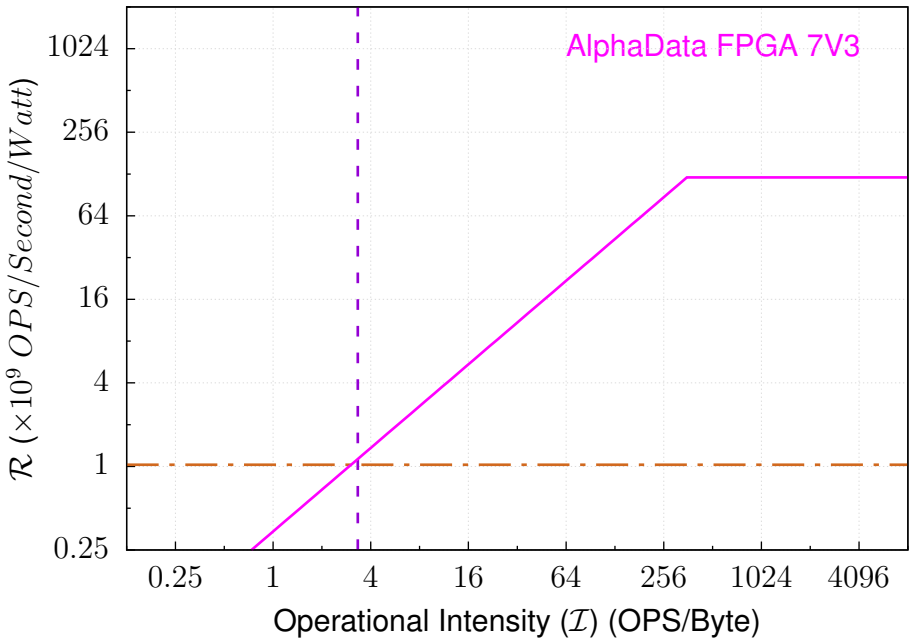


Figure 5.9: Virtex 7 Performance/Power

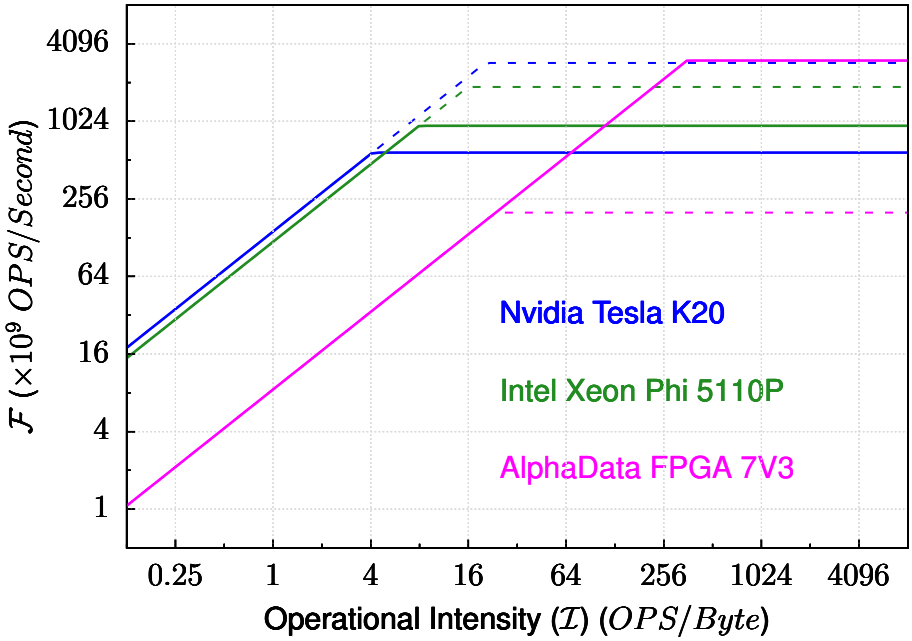


Figure 5.10: All devices, Performance

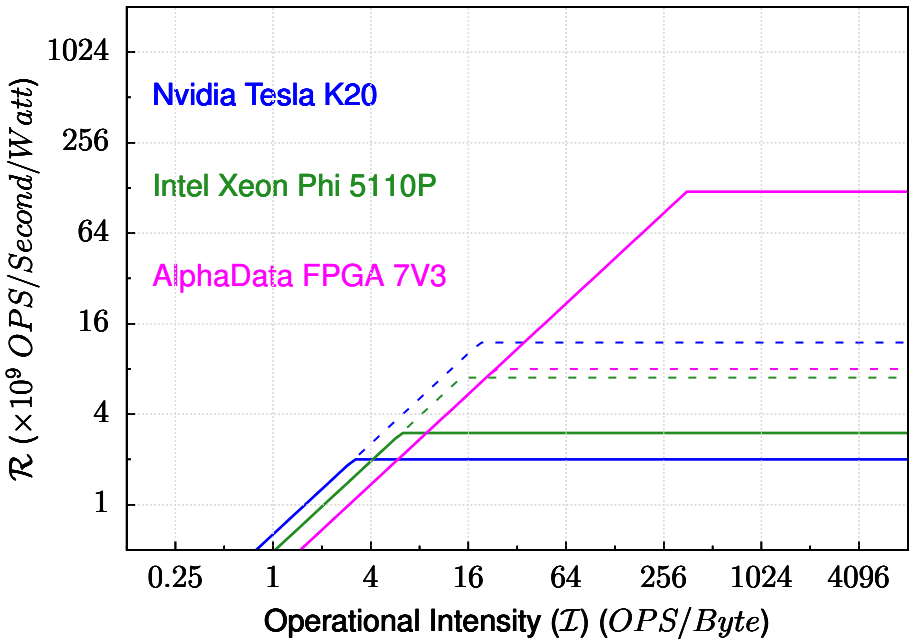


Figure 5.11: All devices, Performance/Power

5.6.4 Results

A first order analysis of Table 5.1 would suggest that the best accelerator device for this application will be the FPGA, followed by the Phi, and then the GPU. This is the case whether considering the theoretical or measured benchmarked performance of the device. The rooflines 5.4 5.6 5.8 that were constructed show that for this application, in theory the rank order of the accelerators is GPU, Phi, then FPGA, due to the role of memory bandwidth in the performance of this application. Given this information the developer should choose to construct a kernel for the GPU device.

In these experiments a kernel was built and measured for each device. All kernels were optimised to the best of our abilities. For the GPU and Phi devices, this meant maximising the number of parallel worker threads to utilise all of the Nvidia SMX and Intel vector lanes respectively with each thread processing an individual key. For the FPGA, the strategy different. By default, the FPGA will read data from global memory in bursts equal to the size of the data type it is requesting. In this case this would lead to requesting 32bit per cycle. By studying the algorithm it was found that each individual key holds 16 32bit words, totalling 512 bits. In this design, the memory interface was optimised, such that 512 bits were read from global memory on every cycle leading to a 16x improvement in bandwidth efficiency. Secondly, the tools were directed to construct a hardware pipeline of our design, meaning that each operation in the algorithm would be computed by a separate piece of hardware, leading to the ability to read, compute and write in parallel. This contrasts the GPU and Phi approach to parallelism of scheduling instructions over the same hardware. It should be noted that an additional advantage to the FPGA for this heavily branching algorithm is that conditional branches simply lead data through differing paths of logic in the FPGA and so have no performance overhead at runtime. Contrasting with the GPU or Phi, a conditional branch may cause the hardware cause a performance penalty if their branch prediction logic fails to predict the outcome of the branch.

[1]ADM 7v3 $\hat{F_f}$ was measured using an in-house Xilinx benchmark. ADM 7v3 $\hat{F_i}$ is estimated using 70% utilisation of LUTS, 20 LUTS per 32bit integer operation, at 200Mhz which is 0.7*(433200/20)*200 = 3032.4 and 1/2 memory channels available, gives $\hat{B}$ of 10.6GB/s.

The results are presented in Table 5.2. It can be seen that the best performance is obtained by the GPU, followed by the Xeon Phi and FPGA. Though the roofline wasn't met in all cases, due to insufficient development time, the qualitative results are as we would predict from the roofline analysis. Further optimisations are possible for each of the target device, but the outcome will be unchanged.

From a power consumption point of view, it is seen from Table 5.1, that the FPGA consumes the least power, followed by the GPU, and the Phi consumes the most. This is TDP, i.e. the worst case power consumption for that device. In reality power consumption varies as a function of utilisation.

Considering energy efficiency, the rooflines suggest that for the chosen application, the most efficient device is the GPU, followed by the Phi, followed by the FPGA. This is a result of the theoretical maximum number of operations that each device can compute in unit time, consuming power at the TDP rate. From the measured results, in Table 5.2, it is seen that the order of energy efficiency is GPU, FPGA, followed by Xeon Phi. These results are due to the GPU doing a large number of operations, making good use of the high rate of power being consumed, the FPGA doing a relatively low number of operations but in a small power envelope, and the Phi doing a middling amount of work, but drawing a large amount of power.

It is expected if subsequent performance gains were implemented to the stage that our application reached peak performance as shown in the roofline, the devices would be ranked as GPU, Phi then FPGA. As seen in the roofline Figure 5.10, the FPGA device has the highest theoretical integer performance, but lacks the necessary bandwidth to utilise this capability. It is our recommendation to FPGA vendors to increase memory bandwidth by adding more advanced memory subsystems such as GDDR5 or High Bandwidth Memory (HBM).

Accelerator	GOPS	GOPS/W
Nvidia K20	126.42	1.18
Xeon Phi 5110P	66.70	0.38
Xilinx 690T	18.11	1.02

Table 5.2: Bob Jenkins Lookup3 measured performance and energy efficiency results

5.7 Conclusions

Choosing the most appropriate accelerator device for a given kernel is non trivial. As seen in this chapter, the device with the highest datasheet performance for integer based computation failed to provide the highest performance. Moreover, it was the device with the lowest integer based performance that provided the highest performance.

The roofline models illustrated that memory bandwidth and arithmetic intensity of the application are essential to consider when choosing an accelerator. As shown in subsection 5.6.2, an understanding of micro-architecture of each device is important to factor into building the roofline models and we have provided the mechanism by which we calculate them.

We have provided a semi-automated toolflow for evaluating subsequent applications for both performance and power efficiency, in order to lessen development effort and aid selecting the appropriate accelerator.

This work was published as "A Semi-Automated Tool Flow for Roofline Analysis of OpenCL Kernels on Accelerators" [5]. From our tool-flow, we have found that integer based applications with a high arithmetic intensity are especially suitable for FPGA. Using the insights developed in this work, we applied ourselves to local sequence alignment problem of the genomics domain as presented in [202]. Furthermore, for the same reason, FPGAs have since demonstrated state of the art performance in the field of machine learning, specifically on inference of deep neural networks using reduced precision integer types [203, 204].

Additionally the applicability of this approach has been seen in subsequent works such as found in [205] which displays rooflines with multiple neural net-

work applications evaluated on several state of the art FPGA devices.

Chapter 6

Conclusions

6.1 Discussion

In this thesis the state of the art of modelling and instrumentation of modern hybrid platforms with respect to energy efficiency was presented and evaluated It was found through case study that models built in the era of single core computing fail to accurately model modern platforms, as they fail to capture the hierarchical and shared nature of the architecture. We have seen the emergence of power models for accelerator devices, but not on the scale of total node. We attribute this to the difficulty of electrically isolating accelerator devices to build such models. Now, with the advent of on-board measurement capabilities for accelerators, and accelerator components, we question the need for predictive models of energy as we have seen in the past.

It was observed that tools for instrumenting applications in this field are lacking. To tackle this problem, multiple novel research tools were developed. For node level power measurement, *HCLEnergyAPI* was developed allowing researchers to instrument their application for energy and power data by wrapping their critical sections of code with our API calls. It supports instrumentation of distributed MPI enabled applications. The prototype implementation of the tool supports the French scientific computing platform GRID5000.

For accelerator level measurement, *hclpower* was created, which samples power data from Nvidia GPU and Intel Xeon Phi accelerators from integrated

power meters. The researcher accesses this data by wrapping their critical sections of code with our API calls, much like *HCLEnergyAPI*.

This thesis studied how to choose an appropriate accelerator for performance or energy efficiency by extending and applying the Berkeley roofline model. Our approach combines an architectural evaluation of the candidate accelerators and an analysis of the target algorithm.

6.2 Claims

This thesis provides a wide survey and evaluation of the literature in this field. Having found that predictive models from the single core era are no longer applicable to modeling applications on modern systems, and with the emergence of physical measurement capabilities, two novel projects *HCLEnergyAPI* and *HCLPower* were developed. These projects provide accurate measurements of power and energy at the granularity of the node and the accelerator device, providing statistical confidence values for the measurements provided. This is a novel capability. The work further provides the capability to measure distributed applications using the MPI extension to our API, a novel feature in our field.

The mapping of the OpenCL standard to the roofline model and further use of it to formulate a performance model for FPGA devices is a novel approach. The modelling of an application on multiple accelerator devices through the OpenCL standard to choose the optimal accelerator has not previously been shown.

6.3 Future Works

Partitioning of data in parallel applications is a well studied area, complete with advanced models and tools for performance optimization [12, 206]. With the tools and methods developed in Chapter 3 of this thesis, accurate energy and power measurements of applications can be taken. Future work of this thesis would be to extend functional performance models to include en-

ergy measurements as a constraint to workload partitioning of data parallel applications. This would facilitate the partitioning of applications with respect to energy consumption and performance across a network of heterogeneous nodes. The interface specified is system agnostic and as such, it should be easily ported to a wide range of supercomputing infrastructure.

Using the tools developed in Chapter 4, functional models could be extended to partition data across accelerator devices with respect to energy consumption, in addition to the existing performance optimiser [207]. This tool could be further validated by calibration against an external power monitoring device to ensure the validity of the vendor provided power readings.

The work of Chapter 5 can be further developed by unifying the work presented with the cache aware roofline model extension [185]. There is also scope to resolve a limitation of the roofline model for devices with heterogeneous throughput. Currently the roofline can only be used to plot maximum expected performance for a specific datatype, typically that of the instruction with maximum throughput of that device. The roofline should be reformulated to capture the heterogeneity of the instruction throughput by weighting the instructions by their relative throughput.

Bibliography

[1] NASA NAS, "NAS Parallel Benchmarks." `https://www.nas.nasa.gov/publications/npb.html`, 2016.

[2] Haswell, "Performance Monitoring Counters for Haswell Architecture." `https://code.google.com/p/likwid/wiki/Haswell`, 2013.

[3] K. O'Brien, I. Pietri, R. Reddy, A. Lastovetsky, and R. Sakellariou, "A Survey of Power and Energy Predictive Models in HPC Systems and Applications," *ACM Comput. Surv.*, vol. 50, pp. 37:1–37:38, June 2017.

[4] K. O'Brien, A. Lastovetsky, I. Pietri, and R. Sakellariou, "Towards Application Energy Measurement and Modelling Tool Support," in *Parallel Computing Technologies* (V. Malyshkin, ed.), vol. 9251 of *Lecture Notes in Computer Science*, pp. 91–101, Springer International Publishing, 2015.

[5] S. Muralidharan, K. O'Brien, and C. Lalanne, "A Semi-Automated Tool Flow for Roofline Analysis of OpenCL Kernels on Accelerators," in *Proc. of the 1st Int. Workshop on Heterogeneous High-performance Reconfigurable Computing, H2RC*, vol. 15, 2015.

[6] L. F. Richardson, *Weather Prediction by Numerical Process*. Cambridge University Press, 2007.

[7] T. Blundell, B. Sibanda, M. Sternberg, and J. Thornton, "Knowledge-based Prediction of Protein Structures and the Design of Novel Molecules," *Nature*, vol. 326, no. 6111, pp. 347–352, 1987.

[8] A. Stamatakis, "RAxML-VI-HPC: Maximum Likelihood-based Phylogenetic Analyses with Thousands of Taxa and Mixed Models," *Bioinformatics*, vol. 22, no. 21, pp. 2688–2690, 2006.

[9] M. A. Clark, R. Babich, K. Barros, R. C. Brower, and C. Rebbi, "Solving Lattice QCD Systems of Equations using Mixed Precision Solvers on GPUs," *Computer Physics Communications*, vol. 181, no. 9, pp. 1517–1528, 2010.

[10] O. Kolditz, S. Bauer, L. Bilke, N. Böttcher, J.-O. Delfs, T. Fischer, U. J. Görke, T. Kalbacher, G. Kosakowski, C. McDermott, *et al.*, "OpenGeoSys: an Open-Source Initiative for Numerical Simulation of Thermo-Hydro-Mechanical/Chemical (THM/C) Processes in Porous Media," *Environmental Earth Sciences*, vol. 67, no. 2, pp. 589–599, 2012.

[11] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of Ion-implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[12] D. Clarke, Z. Zhong, V. Rychkov, and A. Lastovetsky, "Fupermod: A Framework for Optimal Data Partitioning for Parallel Scientific Applications on Dedicated Heterogeneous HPC Platforms," in *International Conference on Parallel Computing Technologies*, pp. 182–196, Springer, 2013.

[13] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten, "GPU-Accelerated Molecular Modeling Coming of Age," *Journal of Molecular Graphics and Modelling*, vol. 29, no. 2, pp. 116–125, 2010.

[14] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[15] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra, "From CUDA to OpenCL: Towards a Performance-Portable Solution for

Multi-Platform GPU Programming," *Parallel Computing*, vol. 38, no. 8, pp. 391–407, 2012.

[16] M. Vestias and H. Neto, "Trends of CPU, GPU and FPGA for High-Performance Computing," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pp. 1–6, IEEE, 2014.

[17] H. Giefers, P. Staar, C. Bekas, and C. Hagleitner, "Analyzing the Energy-Efficiency of Sparse Matrix Multiplication on Heterogeneous Systems: A Comparative Study of GPU, Xeon Phi and FPGA," in *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*, pp. 46–56, IEEE, 2016.

[18] J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: Past, Present and Future," *Concurrency and Computation: practice and experience*, vol. 15, no. 9, pp. 803–820, 2003.

[19] J. Koomey, "Growth in Data Center Electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, vol. 9, 2011.

[20] M. Poess and R. O. Nambiar, "Energy cost, the Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1229–1240, 2008.

[21] S. Sharma, C.-H. Hsu, and W. chun Feng, "Making a Case for a Green500 List," in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pp. 8 pp.–, April 2006.

[22] W.-c. Feng and K. Cameron, "The green500 list: Encouraging Sustainable Supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007.

[23] R. R. Schaller, "Moore's Law: Past, Present and Future," *IEEE Spectrum*, vol. 34, pp. 52–59, June 1997.

[24] J. Koomey and S. Berard and M. Sanchez and H. Wong, "Implications of Historical Trends in the Electrical Efficiency of Computing," *IEEE Annals of the History of Computing*, vol. 33, pp. 46–54, March 2011.

[25] J. J. Dongarra, H. W. Meuer, and E. Strohmaier, "Top500 supercomputer sites," 1994.

[26] R. Ge, X. Feng, H. Pyla, K. Cameron, and W. Feng, "Power Measurement Tutorial for the Green500 list," *The Green500 List: Environmentally Responsible Supercomputing*, 2007.

[27] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M. E. Papka, "Measuring Power Consumption on IBM Blue Gene/Q," in *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pp. 853–859, May 2013.

[28] M. D. de Assunção, J.-P. Gelas, L. Lefèvre, and A.-C. Orgerie, "The Green Grid'5000: Instrumenting and Using a Grid with Energy Sensors," in *Remote Instrumentation for eScience and Related Aspects* (F. Davoli, M. Lawenda, N. Meyer, R. Pugliese, J. Węglarz, and S. Zappatore, eds.), (New York, NY), pp. 25–42, Springer New York, 2012.

[29] S. Leibson, "You have three wishes said the HPC Leprechaun at SC15: low power, fast execution, and." `https://forums.xilinx.com/t5/Xcell-Daily-Blog-Archived/You-have-three-wishes-said-the-HPC-Leprechaun-at-SC15-low-power/ba-p/666055`, 2015.

[30] P. Greenhalgh, "Big.little Processing with Arm Cortex-A15 & Cortex-7," *ARM White paper*, vol. 17, 2011.

[31] ARM Holdings., "ARM Architecture Reference Manual." `https://static.docs.arm.com/ddi0487/ca/DDI0487C_a_armv8_arm.pdf`, 2017.

[32] J. Kalyanasundaram and Y. Simmhan, "ARM Wrestling with Big Data: A Study of Commodity ARM64 Server for Big Data Workloads," *arXiv preprint arXiv:1701.05996*, 2017.

[33] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, ACM, 2000.

[34] C. Isci and M. Martonosi, "Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data," in *36th annual IEEE/ACM International Symposium on Microarchitecture*, p. 93, IEEE Computer Society, 2003.

[35] T. Heath, B. Diniz, B. Horizonte, E. V. Carrera, and R. Bianchini, "Energy Conservation in Heterogeneous Server Clusters," pp. 186–195, 2005.

[36] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system Power Analysis and Modeling for Server Environments," in *In Proceedings of Workshop on Modeling, Benchmarking, and Simulation*, pp. 70–77, 2006.

[37] X. Fan, W.-D. Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 13–23, 2007.

[38] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed Systems Meet Economics: Pricing in the Cloud," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, USENIX Association, 2010.

[39] R. Basmadjian, N. Ali, F. Niedermeier, H. de Meer, and G. Giuliani, "A methodology to predict the power consumption of servers in data centres," in *2nd International Conference on Energy-Efficient Computing and Networking*, ACM, 2011.

[40] W. L. Bircher and L. K. John, “Complete System Power Estimation Using Processor Performance Events,” *IEEE Transactions on Computers*, vol. 61, pp. 563–577, Apr. 2012.

[41] H. Hong, Sunpyand Kim, “An Integrated GPU Power and Performance Model,” *SIGARCH Comput. Archit. News*, vol. 38, pp. 280–289, June 2010.

[42] J. Chen, B. Li, Y. Zhang, L. Peng, and J.-K. Peir, “Statistical GPU Power Analysis Using Tree-based Methods,” in *International Green Computing Conference and Workshops (IGCC)*, IEEE, 2011.

[43] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, “Power Aware Computing on GPUs,” in *Symposium on Application Accelerators in High Performance Computing (SAAHPC)*, IEEE Computer Society, 2012.

[44] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, “Statistical Power Modeling of GPU Kernels Using Performance Counters,” in *International Green Computing Conference and Workshops (IGCC)*, IEEE, 2010.

[45] S. Song, C. Su, B. Rountree, and K. W. Cameron, “A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures,” in *27th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pp. 673–686, IEEE Computer Society, 2013.

[46] H. Wang and Y. Cao, “Predicting Power Consumption of GPUs with Fuzzy Wavelet Neural Networks,” *Parallel Computing*, vol. 44, pp. 18–36, May 2015.

[47] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, “Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86- 64 Processors,” in *Green Computing Conference, 2010 International*, pp. 123–133, IEEE, 2010.

[48] B. C. Lee and D. M. Brooks, "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction," *SIGARCH Comput. Archit. News*, vol. 34, pp. 185–194, Oct. 2006.

[49] A. Lewis, S. Ghosh, and N.-F. Tzeng, "Run-time Energy Consumption Estimation Based on Workload in Server Systems," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, USENIX Association, 2008.

[50] R. Basmadjian and H. de Meer, "Evaluating and Modeling Power Consumption of Multi-Core Processors," in *Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on*, pp. 1–10, May 2012.

[51] R. Bertran, M. Gonzalez Tallada, X. Martorell, N. Navarro, and E. Ayguade, "A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs," *IEEE Trans. Comput.*, vol. 62, pp. 1289–1302, July 2013.

[52] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and Power Analysis of ATI GPU: A Statistical Approach," in *Proceedings of the 2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*, NAS '11, pp. 149–158, IEEE Computer Society, 2011.

[53] J. Lim, N. B. Lakshminarayana, H. Kim, W. Song, S. Yalamanchili, and W. Sung, "Power Modeling for GPU Architectures Using McPAT," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, pp. 26:1–26:24, June 2014.

[54] Y. S. Shao and D. Brooks, "Energy Characterization and Instruction-level Energy Model of Intel's Xeon Phi Processor," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, pp. 389–394, IEEE Press, 2013.

[55] V. Bui, B. Norris, K. Huck, L. C. McInnes, L. Li, O. Hernandez, and B. Chapman, "A Component Infrastructure for Performance and Power Modeling of Parallel Scientific Applications," in *Proceedings of the 2008*

compFrame/HPC-GECO Workshop on Component Based High Performance, CBHPC '08, pp. 6:1–6:11, ACM, 2008.

[56] B. Subramaniam and W.-c. Feng, "Statistical Power and Performance Modeling for Optimizing the Energy Efficiency of Scientific Computing," in *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pp. 139–146, IEEE Computer Society, 2010.

[57] A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely, "Modeling Power and Energy Usage of HPC Kernels," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pp. 990–998, IEEE, 2012.

[58] C. Lively, X. Wu, V. Taylor, S. Moore, H.-C. Chang, C.-Y. Su, and K. Cameron, "Power-aware Predictive Models of Hybrid (MPI/OpenMP) Scientific Applications On Multicore Systems," *Computer Science-Research and Development*, vol. 27, no. 4, pp. 245–253, 2012.

[59] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick, S. Pakin, V. Pascucci, and S. Klasky, "Exploring Power Behaviors and Trade-offs of In-situ Data Analytics, booktitle = Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis," SC '13, ACM, 2013.

[60] M. E. M. Diouri, O. Glück, J.-C. Mignot, and L. Lefèvre, "Energy Estimation for MPI Broadcasting Algorithms in Large Scale HPC Systems," in *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, ACM, 2013.

[61] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Enabling Accurate Power Profiling of HPC Applications on Exascale Systems," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '13, ACM, 2013.

[62] P. Gschwandtner, M. Knobloch, B. Mohr, D. Pleiter, and T. Fahringer, "Modeling CPU Energy Consumption of HPC Applications on the IBM POWER7," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 536–543, IEEE, 2014.

[63] J. Moore, "Gamut: Generic Application EMULaTOR," 2004.

[64] SPEC, "SPEC's Benchmarks." `https://www.spec.org/benchmarks.html`, 2015.

[65] Stream, "STREAM: Sustainable Memory Bandwidth in High Performance Computers." `https://www.cs.virginia.edu/stream/`, 2015.

[66] AMDHT, "HyperTransport." `https://en.wikipedia.org/wiki/HyperTransport`, 2001.

[67] A. Kansal and F. Zhao, "Fine-grained Energy Profiling for Power-Aware Application Design," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, p. 26, Aug. 2008.

[68] WINXPERF, "Windows Performance Toolkit Technical Reference." `https://msdn.microsoft.com/en-us/library/windows/hardware/hh162945.aspx`, 2015.

[69] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A Comparison of High-level Full-system Power Models," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, USENIX Association, 2008.

[70] S. Rivoire, "Models and Metrics for Energy-Efficient Computer Systems. PhD Thesis.," Stanford University, Stanford, California, 2008.

[71] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and Responsive Power Models for Multicore Processors Using Performance Counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pp. 147–158, ACM, 2010.

[72] S. Wang and W. Shi, "CPT: An Energy Efficiency Model for Multi-Core Computer Systems," *Wayne State University, Tech. Rep. MIST-TR-2012-008*, 2012.

[73] Q. Liu, M. Moreto, V. Jimenez, J. Abella, F. J. Cazorla, and M. Valero, "Hardware Support for Accurate Per-task Energy Metering in Multicore Systems," *ACM Trans. Archit. Code Optim.*, vol. 10, Dec. 2013.

[74] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, HPCA '02, IEEE Computer Society, 2002.

[75] T. Li and L. K. John, "Run-time Modeling and Estimation of Operating System Power Consumption," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, pp. 160–171, June 2003.

[76] K. Singh, M. Bhadauria, and S. A. McKee, "Real Time Power Estimation and Thread Scheduling via Performance Counters," *SIGARCH Comput. Archit. News*, vol. 37, pp. 46–55, July 2009.

[77] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. Yardi, "CAMP: A Technique to Estimate Per-Structure Power at Run-Time Using a Few Simple Parameters," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 289–300, Feb 2009.

[78] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable, Scalable, per-Core Power Estimation for Intelligent Resource Management," Green Computing Conference, 2010 International, 2010-08-16 2010.

[79] S. Roy, A. Rudra, and A. Verma, "An Energy Complexity Model for Algorithms," in *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pp. 283–304, ACM, 2013.

[80] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green Governors: A Framework for Continuously Adaptive DVFS," in *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–8, IEEE, 2011.

[81] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the Effectiveness of Model-based Power Characterization," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, USENIX Association, 2011.

[82] W. Dargie, "A Stochastic Model for Estimating the Power Consumption of a Processor," *IEEE Transactions on Computers*, vol. 64, no. 5, 2015.

[83] PowerAPI, "Sandia National Laboratories: High Performance Computing Power Application Programming Interface (API) Specification." =http://powerapi.sandia.gov/, 2016.

[84] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-aware High Performance Computing with Graphic Processing Units," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, USENIX Association, 2008.

[85] CUPTI, "CUDA Profiling Tools Interface." `https://developer.nvidia.com/cuda-profiling-tools-interface`, 2015.

[86] A. Marowka, "Analytical Modeling of Energy Efficiency in Heterogeneous Processors," *Comput. Electr. Eng.*, vol. 39, Nov. 2013.

[87] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing," *ACM Trans. Archit. Code Optim.*, vol. 10, Apr. 2013.

[88] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: A Programming Model for Heterogeneous Multi-core Systems, booktitle =

Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems," ASPLOS XIII, ACM, 2008.

[89] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical Power Consumption Analysis and Modeling for GPU-based Computing," in *Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[90] D. Li, S. Byna, and S. Chakradhar, "Energy-Aware Workload Consolidation on GPU," in *Proceedings of the 2011 40th International Conference on Parallel Processing Workshops*, ICPPW '11, pp. 389–398, IEEE Computer Society, 2011.

[91] Q. Xie, T. Huang, Z. Zou, L. Xia, Y. Zhu, and J. Jiang, "An Accurate Power Model for GPU Processors," in *Computing and Convergence Technology (ICCCT), 2012 7th International Conference on*, pp. 1141–1146, IEEE, 2012.

[92] J. Zhao, G. Sun, G. H. Loh, and Y. Xie, "Optimizing GPU energy efficiency with 3D die-stacking graphics memory and reconfigurable memory interface," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, p. 24, 2013.

[93] D. B. Thomas, L. Howes, and W. Luk, "A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '09, pp. 63–72, ACM, 2009.

[94] TDP, "Thermal design power." `https://en.wikipedia.org/wiki/Thermal_design_power`, 2015.

[95] H. Lange, F. Stock, A. Koch, and D. Hildenbrand, "Acceleration and Energy Efficiency of a Geometric Algebra Computation Using Reconfigurable Computers and GPUs," in *Proceedings of the 2009 17th IEEE*

Symposium on Field Programmable Custom Computing Machines, FCCM '09, pp. 255–258, IEEE Computer Society, 2009.

[96] XPE, "Xilinx Power Estimator (XPE)." `http://www.xilinx.com/products/technology/power/xpe.html`, 2015.

[97] T. Hamada, K. Benkrid, K. Nitadori, and M. Taiji, "A Comparative Study on ASIC, FPGAs, GPUs and General Purpose Processors in the Gravitational N-body Simulation," in *Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems*, AHS '09, pp. 447–452, IEEE Computer Society, 2009.

[98] S. Kestur, J. D. Davis, and O. Williams, "BLAS Comparison on FPGA, CPU and GPU," in *Proceedings of the 2010 IEEE Annual Symposium on VLSI*, ISVLSI '10, pp. 288–293, IEEE Computer Society, 2010.

[99] B. Betkaoui, D. B. Thomas, and W. Luk, "Comparing Performance and Energy Efficiency of FPGAs and GPUs for High Productivity Computing," in *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 94–101, IEEE, 2010.

[100] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, "Highly Parameterized K-Means Clustering on FPGAs: Comparative Results with GPPs and GPUs," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pp. 475–480, IEEE, 2011.

[101] C. d. Schryver, I. Shcherbakov, F. Kienle, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "An Energy Efficient FPGA Accelerator for Monte Carlo Option Pricing with the Heston Model," in *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs*, RECONFIG '11, pp. 468–474, IEEE Computer Society, 2011.

[102] B. Duan, W. Wang, X. Li, C. Zhang, P. Zhang, and N. Sun, "Floating-Point Mixed-Radix FFT Core Generation for FPGA and Comparison with GPU and CPU," in *Field-Programmable Technology (FPT), 2011 International Conference on*, Dec 2011.

[103] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?, booktitle = Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines," FCCM '12, pp. 232–239, IEEE Computer Society, 2012.

[104] M. Birk, M. Balzer, N. Ruiter, and J. Becker, "Comparison of Processing Performance and Architectural Efficiency Metrics for FPGAs and GPUs in 3D Ultrasound Computer Tomography," in *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on*, pp. 1–7, IEEE, 2012.

[105] D. Zou, Y. Dou, and F. Xia, "Optimization Schemes and Performance Evaluation of Smith-Waterman Algorithm on CPU, GPU and FPGA," *Concurr. Comput. : Pract. Exper.*, vol. 24, pp. 1625–1644, Sept. 2012.

[106] K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian, "High Performance Biological Pairwise Sequence Alignment: FPGA Versus GPU Versus Cell BE Versus GPP," *Int. J. Reconfig. Comput.*, vol. 2012, Jan. 2012.

[107] K. Pauwels, M. Tomasi, J. Diaz, E. Ros, and M. M. V. Hulle, "A Comparison of FPGA and GPU for Real-Time Phase-Based Optical Flow, Stereo, and Local Image Features," *IEEE Transactions on Computers*, vol. 61, no. 7, pp. 999–1012, 2012.

[108] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, "A Performance and Energy Comparison of Convolution on GPUs, FPGAs, and Multicore Processors," *ACM Trans. Archit. Code Optim.*, vol. 9, Jan. 2013.

[109] XEONPHI, "Intel Many Integrated Core Architecture." `https://en.wikipedia.org/wiki/Xeon_Phi`, 2015.

[110] HPRC, "High-Performance Reconfigurable Computing." `http://www.chrec.org/`, 2015.

[111] S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques," *ACM Computing Surveys (CSUR)*, vol. 47, July 2015.

[112] J. Ou and V. K. Prasanna, "Rapid Energy Estimation of Computations on FPGA Based Soft Processors," in *SOC Conference, 2004. Proceedings. IEEE International*, Sept 2004.

[113] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A Detailed Power Model for Field-programmable Gate Arrays," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, pp. 279–302, Apr. 2005.

[114] X. Wang, S. G. Ziavras, and J. Hu, "System-Level Energy Modeling for Heterogeneous Reconfigurable Chip Multiprocessors," in *2006 International Conference on Computer Design*, Oct 2006.

[115] Z. Al-Khatib and S. Abdi, "Operand-Value-Based Modeling of Dynamic Energy Consumption of Soft Processors in FPGA," in *International Symposium on Applied Reconfigurable Computing*, pp. 65–76, Springer, 2015.

[116] J. Demmel, A. Gearhart, B. Lipshitz, and O. Schwartz, "Perfect Strong Scaling Using No Additional Energy," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013.

[117] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A Roofline Model of Energy," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 661–672, IEEE, 2013.

[118] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, vol. 52, Apr. 2009.

[119] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon: Fine-Grained and Integrated Power Monitoring for Commodity Computer Systems," in *Proceedings Southeastcon 2010*, IEEE, 2010.

[120] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, “Algorithmic time, energy, and power on candidate HPC compute building blocks,” in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 447–457, IEEE, 2014.

[121] J. M. Marszalkowski, M. Drozdowski, and J. Marszalkowski, “Time and Energy Performance of Parallel Systems with Hierarchical Memory", journal="Journal of Grid Computing,” vol. 14, no. 1, pp. 153–170, 2016.

[122] S. Kamil, J. Shalf, and E. Strohmaier, “Power efficiency in high performance computing,” in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, IEEE, 2008.

[123] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, “Powerpack: Energy profiling and analysis of high-performance systems and applications,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658–671, 2010.

[124] HPL, “HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers.” `http://www.netlib.org/benchmark/hpl/`, 2008.

[125] SYSTEMG, “System G cluster.” `https://www.cs.vt.edu/facilities/systemg`, 2015.

[126] J. Dongarra, H. Ltaief, P. Luszczek, and V. Weaver, “Energy Footprint of Advanced Dense Numerical Linear Algebra using Tile Algorithms on Multicore Architecture,” in *The 2nd International Conference on Cloud and Green Computing*, November 2012.

[127] LAPACK, “Linear Algebra Package.” `http://http://www.netlib.org/lapack/`, 2013.

[128] PLASMA, “Parallel Linear Algebra for Scalable Multi-core Architectures.” `http://http://icl.cs.utk.edu/plasma/`, 2015.

[129] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, pp. 20–27, Mar. 2012.

[130] C. Lively, V. Taylor, X. Wu, H.-C. Chang, C.-Y. Su, K. Cameron, S. Moore, and D. Terpstra, "E-AMOM: an energy-aware modeling and optimization methodology for scientific applications," *Computer Science-Research and Development*, vol. 29, no. 3-4, pp. 197–210, 2014.

[131] PAPI, "Performance Application Programming Interface 5.4.1." `http://icl.cs.utk.edu/papi/`, 2015.

[132] H. Ltaief, P. Luszczek, and J. Dongarra, "Profiling High Performance Dense Linear Algebra Algorithms on Multicore Architectures for Power and Energy Efficiency," *Comput. Sci.*, vol. 27, pp. 277–287, Nov. 2012.

[133] G. Bosilca, H. Ltaief, and J. Dongarra, "Power Profiling of Cholesky and QR Factorizations on Distributed Memory Systems," *Computer Science-Research and Development*, vol. 29, no. 2, pp. 139–147, 2014.

[134] M. Witkowski, A. Oleksiak, T. Piontek, and J. Weglarz, "Practical Power Consumption Estimation for Real Life HPC Applications," *Future Gener. Comput. Syst.*, vol. 29, Jan. 2013.

[135] M. Jarus, A. Oleksiak, T. Piontek, and J. Węglarz, "Runtime Power Usage Estimation of HPC servers for Various Classes of Real-Life Applications," *Future Generation Computer Systems*, vol. 36, 2014.

[136] NAS, "NAS Parallel Benchmarks." `https://www.nas.nasa.gov/publications/npb.html`, 2015.

[137] B. Li, H.-C. Chang, S. Song, C.-Y. Su, T. Meyer, J. Mooring, and K. W. Cameron, "The Power-Performance Tradeoffs of the Intel Xeon Phi on HPC Applications," in *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pp. 1448–1456, IEEE, 2014.

[138] A. Lastovetsky and R. R. Manumachu, "New Model-Based Methods and Algorithms for Performance and Energy Optimization of Data Parallel Applications on Homogeneous Multicore Clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, 2017.

[139] Rodinia, "The Rodinia Benchmark Suite, version 3.0." `https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating_Compute-Intensive_Applications_with_Accelerators`, 2015.

[140] BLAS, "BLAS (Basic Linear Algebra Subprograms)." `http://www.netlib.org/blas/`, 2015.

[141] GSLMLR, "Multi-Parameter Fitting." `https://www.gnu.org/software/gsl/manual/html_node/Multi_002dparameter-fitting.html`, 2015.

[142] CPUFreq, "CPU Frequency Scaling - On-demand Governor." `https://wiki.archlinux.org/index.php/CPU_frequency_scaling`, 2015.

[143] "Compute Unified Device Architecture." `http://www.nvidia.com/object/cuda_home_new.html`, 2015.

[144] W. Wu, L. Jin, J. Yang, P. Liu, and S. X.-D. Tan, "A Systematic Method for Functional Unit Power Estimation in Microprocessors," in *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, pp. 554–557, ACM, 2006.

[145] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Quantifying the Energy Cost of Data Movement in Scientific Applications," in *2013 IEEE international symposium on workload characterization (IISWC)*, 2013.

[146] D. Pandiyan and C. J. Wu, "Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms," in *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pp. 171–180, Oct 2014.

[147] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with PAPI," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pp. 262–268, IEEE, 2012.

[148] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pp. 189–194, IEEE, 2010.

[149] IntelPCM, "Intel Performance Counter Monitor - A Better Way to Measure CPU Utilization.." `https://software.intel.com/en-us/articles/intel-performance-counter-monitor`, 2012.

[150] IntelMPSS, "Intel Manycore Platform Software Stack (Intel MPSS)." `https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss`, 2014.

[151] NVML, "NVIDIA Management Library (NVML)." `https://developer.nvidia.com/nvidia-management-library-nvml`, 2011.

[152] B. W. Kernighan, *The C Programming Language*. Prentice Hall Professional Technical Reference, 2nd ed., 1988.

[153] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *Cloud Computing and Services Science* (I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, eds.), vol. 367 of *Communications in Computer and Information Science*, pp. 3–20, Springer International Publishing, 2013.

[154] F. Clouet, S. Delamare, J.-P. Gelas, L. Lefèvre, L. Nussbaum, C. Parisot, L. Pouilloux, and F. Rossigneux, "Kwapi: A Unified Monitoring Framework for Energy Consumption and Network Traffic," in *4th GENI/FIRE Collaboration Workshop*, 2015.

[155] S. Barrachina, M. Barreda, S. Catalán, M. F. Dolz, G. Fabregat, R. Mayo, and E. S. Quintana-Ortí, "An Integrated Framework for Power-Performance Analysis of Parallel Scientific Workloads," in *ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pp. 114–119, 2013.

[156] A. Cabrera, F. Almeida, J. Arteaga, and V. Blanco, "Measuring Energy Consumption using EML (Energy Measurement Library)," *Computer Science - Research and Development*, pp. 1–9, 2014.

[157] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, pp. 658–671, May 2010.

[158] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon: Fine-grained and Integrated Power Monitoring for Commodity Computer Systems," in *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pp. 479–484, March 2010.

[159] C.-H. Hsu and S. Poole, "Power Measurement for High Performance Computing: State of the art," in *Green Computing Conference and Workshops (IGCC), 2011 International*, pp. 1–6, July 2011.

[160] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pp. 189–194, Aug 2010.

[161] AMD, "BIOS and Kernel Developer's Guide(BKDG) for AMD Family 15h Models 00h-0Fh Processors," 2013.

[162] D. Hackenberg, T. Ilsche, R. Schone, D. Molka, M. Schmidt, and W. Nagel, "Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pp. 194–204, April 2013.

[163] J. Treibig, G. Hager, and G. Wellein, "Likwid: A lightweight performance-oriented tool suite for x86 multicore environments," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pp. 207–216, IEEE, 2010.

[164] Nvidia Corp, "Nvidia Management Library." `https://developer.nvidia.com/nvidia-management-library-nvml`, 2017.

[165] T. Bray, "The javascript object notation (json) data interchange format." `https://tools.ietf.org/html/rfc7159`, 2014.

[166] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, *et al.*, "The NAS parallel benchmarks," *The International Journal of Supercomputing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[167] "Performance Characterization of the NAS Parallel Benchmarks in OpenCL, author=Seo, Sangmin and Jo, Gangwon and Lee, Jaejin," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pp. 137–148, IEEE, 2011.

[168] C. Fu, V. Chau, M. Li, and C. J. Xue, "Race to Idle or Not: Balancing the Memory Sleep Time with DVS for Energy Minimization," *Journal of Combinatorial Optimization*, vol. 35, pp. 860–894, Apr 2018.

[169] M. A. Awan and S. M. Petters, "Enhanced Race-To-Halt: A Leakage-Aware Energy Management Approach for Dynamic Priority Systems," in *2011 23rd Euromicro Conference on Real-Time Systems*, pp. 92–101, July 2011.

[170] Amazon Inc., "Amazon F1 Instances." `https://aws.amazon.com/ec2/instance-types/f1/`, 2017.

[171] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring Energy and Power with PAPI," in *2012 41st International Conference on Parallel Processing Workshops*, pp. 262–268, Sept 2012.

[172] R. Suda and D. Q. Ren, "Accurate Measurements and Precise Modeling of Power Dissipation of CUDA Kernels toward Power Optimized High Performance CPU-GPU Computing," in *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 432–438, Dec 2009.

[173] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W. m. Hwu, "GPU Clusters for High-Performance Computing," in *2009 IEEE International Conference on Cluster Computing and Workshops*, pp. 1–8, Aug 2009.

[174] S. Collange, D. Defour, and A. Tisserand, *Power Consumption of GPUs from a Software Perspective*, pp. 914–923. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[175] S. Huang, S. Xiao, and W. Feng, "On the Energy Efficiency of Graphics Processing Units for Scientific Computing," in *2009 IEEE International Symposium on Parallel Distributed Processing*, pp. 1–8, May 2009.

[176] Y. S. Shao and D. Brooks, "Energy Characterization and Instruction-level Energy Model of Intel's Xeon Phi Processor," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ISLPED '13, (Piscataway, NJ, USA), pp. 389–394, IEEE Press, 2013.

[177] Intel Corp, "Intel MPSS." `https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss`, 2017.

[178] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Softw.*, vol. 16, pp. 1–17, Mar. 1990.

[179] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[180] J. Price and S. McIntosh-Smith, "OCLgrind." `https://github.com/jrprice/Oclgrind`, 2017.

[181] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) Benchmark Suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU-3, (New York, NY, USA), pp. 63–74, ACM, 2010.

[182] J. D. McCalpin, "A Survey of Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE TCCA Newsletter*, vol. 19, p. 25, 1995.

[183] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pp. 44–54, Ieee, 2009.

[184] K. Krommydas, W.-c. Feng, C. D. Antonopoulos, and N. Bellas, "Opendwarfs: Characterization of Dwarf-Based Benchmarks on Fixed and Reconfigurable Architectures," *Journal of Signal Processing Systems*, vol. 85, no. 3, pp. 373–392, 2016.

[185] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware Roofline Model: Upgrading the Loft," *IEEE Computer Architecture Letters*, vol. 13, no. 1, pp. 21–24, 2014.

[186] G. Ofenbeck, R. Steinmann, V. C. Cabezas, D. G. Spampinato, and M. Püschel, "Applying the Roofline Model," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 76 – 85, 2014.

[187] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A Roofline Model of Energy," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 661–672, IEEE, 2013.

[188] A. Munshi, "The OpenCL Specification," in *Hot Chips 21 Symposium (HCS), 2009 IEEE*, pp. 1–314, IEEE, 2009.

[189] Intel Corp., "Introduction to Intel® Advanced Vector Extensions." https://software.intel.com/sites/default/files/m/d/4/1/d/8/Intro_to_Intel_AVX.pdf, 2017.

[190] NXP Corp, "AltiVec Technology Programming Environments Manual." http://www.nxp.com/docs/en/reference-manual/ALTIVECPEM.pdf, 2017.

[191] ARM Ltd., "Neon Technology." https://developer.arm.com/technologies/neon, 2017.

[192] Intel Corp, "Intel E5-2620v4." https://ark.intel.com/products/92986/Intel-Xeon-Processor-E5-2620-v4-20M-Cache-2_10-GHz, 2017.

[193] Intel Corp, "Intel VTune." https://software.intel.com/en-us/intel-vtune-amplifier-xe, 2017.

[194] Nvidia Corp., "Nvidia SDK." https://developer.nvidia.com/cuda-downloads, 2017.

[195] Xilinx Inc., "Xilinx SDAccel." https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html, 2017.

[196] Texas Instruments, "Texas Instruments Fusion Digital Power Studio." http://www.ti.com/tool/FUSION-DIGITAL-POWER-STUDIO, 2017.

[197] B. Jenkins, "Hash functions," *Dr Dobbs Journal*, vol. 22, no. 9, pp. 107–+, 1997.

[198] B. Fitzpatrick, "Distributed Caching with Memcached," *Linux J.*, vol. 2004, pp. 5–, Aug. 2004.

[199] Nvidia Corp., "NVIDIA's Next Generation CUDA Compute Architecture: Kepler TM GK110/210." http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf, 2014.

[200] "Nvidia Throughput Table." `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#maximize-instruction-throughput`, 2018.

[201] Xilinx Inc., "Xilinx Vivado HLS." `https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html`, 2018.

[202] L. Di Tucci, K. O'Brien, M. Blott, and M. D. Santambrogio, "Architectural Optimizations for High performance and Energy Efficient Smith-Waterman Implementation on FPGAs using OpenCL," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 716–721, IEEE, 2017.

[203] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74, ACM, 2017.

[204] M. Blott, T. B. Preußer, N. Fraser, G. Gambardella, K. O'Brien, Y. Umuroglu, and M. Leeser, "Scaling Neural Network Performance through Customized Hardware Architectures on Reconfigurable Logic," in *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 419–422, Nov 2017.

[205] M. Blott, T. B. Preusser, N. J. Fraser, G. Gambardella, K. O'Brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, pp. 16:1–16:23, Dec. 2018.

[206] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *Journal of Parallel and Distributed Computing*, vol. 61, no. 4, pp. 520 – 535, 2001.

[207] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on heterogeneous multicore and multi-gpu systems using functional performance models of data-parallel applications," in *2012 IEEE International Conference on Cluster Computing*, pp. 191–199, IEEE, 2012.

[208] O. Mämmelä, M. Majanen, R. Basmadjian, H. De Meer, A. Giesler, and W. Homberg, "Energy-aware job scheduler for high-performance computing," *Computer Science - Research and Development*, vol. 27, no. 4, 2012.

[209] N. Farooqui, A. Kerr, G. Diamos, S. Yalamanchili, and K. Schwan, "A Framework for Dynamically Instrumenting GPU Compute Applications Within GPU Ocelot," in *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-4, ACM, 2011.

[210] R. Thakur and W. D. Gropp, "Improving the Performance of Collective Operations in MPICH," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, Springer, 2003.

[211] MPICH, "MPICH - High Performance Portable MPI." `http://www.mpich.org/`, 2016.

[212] OpenMPI, "OpenMPI - Open Source High Performance Computing." `https://www.open-mpi.org/`, 2016.

Appendix A

Background - Supporting Experimental Data

A.1 HCLEnergy Experiments Platform Definition

The experiments in section 3 were carried out on nova-19.lyon.grid5000.fr, a compute node at the Lyon site of GRID5000. It is a Dell PowerEdge R430 with **2x** Intel(R) Xeon(R) CPU E5-2620 v4 running at 2.10GHz. The node contains 32GB of RAM, a 10Gb ethernet card and a 2x 300GB SAS hard drives. It was added to the GRID5000 infrastructure in March 2017.

A.2 Power and Energy Models: Details

A.2.1 Power and Energy Models for CPUs

[39] construct a power model of a server as a summation of power models of its components, the processor (CPU), memory (RAM), fans, and disk (HDD). The power consumptions for different types of servers are given by the following equations :

$$P_{Blade} = \sum_{i=1}^{l} P_{Mainboard} \tag{A.1}$$

$$P_{Tower\_or\_Rackable} = \sum_{i=1}^{l} P_{Mainboard} + \sum_{j=1}^{m} P_{Fan} + \sum_{k=1}^{n} P_{PSU} \tag{A.2}$$

where *l*, *m*, and *n* denote respectively the total number of mainboards, fans, and power supply units. For purposes of brevity, only their power model for *blade* servers is presented here.

The power consumption of the mainboard is given by the following equation [39]:

$$P_{Mainboard} = \sum_{i=1}^{l} P_{CPU} + P_{RAM} + \sum_{j=1}^{m} P_{NIC} + \sum_{k=1}^{n} P_{HDD} + c \tag{A.3}$$

where *l*, *m*, and *n* denote the total number of processors (CPU), the total number of network interface cards (NIC), and the total number of attached hard disk drives (HDD) respectively and c is a constant (55 W for blade servers).

The power consumption of a multicore processor, P_{CPU}, is calculated as the sum of power consumptions of individual cores. The power consumption of each individual core is based on the linear single-core model of [37]. [39] modelled the power consumption of a multicore processor containing n cores as follows:

$$P_{C_i} = P_{max} \times (U_{C_i}/100) \tag{A.4a}$$

$$P_{CPU} = P_{base} + \sum_{i=1}^{n} P_{C_i} \tag{A.4b}$$

$$P_{max} = V_{max}^2 \times f_{max} \times C_{eff} \tag{A.4c}$$

where P_{C_i} indicates the power consumption of a core, p_{base} represents the base power consumption of a processor, V_{max} and f_{max} signify the voltage and frequency at maximum utilization respectively, and C_{eff}, the effective capacitance.

For memory module, the power consumption is given by [39]:

$$P_{RAM} = P_{RAM\_base} + \gamma \times \alpha \times \beta \tag{A.5}$$

where α = 1, 2.3, 1.3, and 1.9 for unbuffered DDR_2 SDRAM, buffered DDR_2 SDRAM, unbuffered DDR_3 SDRAM, and buffered DDR_3 SDRAM respectively. The value of β used is 7.347. A probabilistic approach is used to model γ for a processor not in idle state, otherwise the value of γ used is 0. [39] and [208] model the idle power consumption of a unbuffered SDRAM of type DDR_2 or DDR_3 as follows:

$$P_{RAM\_base} = \sum_{i=1}^{r} s_i \times p \tag{A.6}$$

where *r* represents the total number of installed memory modules and s_i indicates the size of memory module i. [39] and [208] model the power consumption of the hard disk as follows:

$$P_{HDD} = a \times 1.4 \times P_{base} + b \times P_{HDD\_base} + c \times 3.7 \times P_{base} \tag{A.7a}$$

$$P_{HDD\_base} = P_{base} \times (d + 0.2 \times e) \tag{A.7b}$$

where a, b, c, d, and e denote probabilities of different states of the disk and P_{base} is the base power consumption.

Based on the model evaluations on tower and blade servers, the authors report maximum prediction error rates of 8% and 9% for tower servers and blade servers respectively.

A.2.2 Power and Energy Models for GPUs

[41] present a GPU power consumption prediction model that is modelled similar to the PMC-based unit power prediction approach of [34]. Their model is

described below.

$$GPU\_power = DynamicPower + BasePower$$

$$DynamicPower = \sum_{i=0}^{n} DP\_Component\_i = DP\_SMs + DP\_Memory$$

$$\sum_{i=0}^{n} SM\_Component_i =$$

$$DP\_Int + DP\_Fp + DP\_Sfu + DP\_Alu + DP\_Texture\_Cache +$$

$$DP\_Const\_Cache + DP\_Shared + DP\_Reg + DP\_FDS + DP\_Const\_SM$$

$$DP\_SMs = Num\_SMs \times \sum_{i=0}^{n} SM\_Component_i$$

where $BasePower$ is the idle power consumption of a GPU and Num_SMs is the total number of Streaming Multiprocessors (SM) in a GPU. The dynamic power for each component is calculated as follows [41]:

$$\begin{aligned} DP\_Component\_i &= MaxPower_{component} \times AccessRate_{component} \\ AccessRate_{component} &= \frac{DAC\_per\_th_{component} \times Warps\_per\_SM}{Predicted\_Exec\_cycles/4} \\ DAC\_per\_th_{component} &= \sum_{i=0}^{n} Number\_Inst\_per\_warps_i(comp) \\ Warps\_per\_SM &= (\frac{\#Threads\_per\_block}{\#Threads\_per\_warp} \times \frac{\#Blocks}{\#Active\_SMs}) \end{aligned}$$

The parameter, $DAC\_per\_th_{component}$), is calculated as the total number of instructions that access an architectural component. The parameter, $Warps\_per\_SM$, indicate the number of warps that are executed in one SM. The parameter, ($Predicted\_Exec\_cycles$), is calculated using an analytical timing model, which we don't present here since our main focus in this paper is power and energy models. The dynamic power of GDDR memory (DP_Memory) is modelled based on the dynamic powers of the global mem-

ory and local memory that share it [41]:

$$DP\_Memory = \sum_{i=0}^{n} Memory\_component_i = DP\_GlobalMem + DP\_LocalMem$$

The other memories (shared, constant, texture) are modelled separately as SM components. The parameter, $MaxPower$, is empirically determined by stressing different architectural units in a GPU using synthetic micro-benchmarks. A special piecewise linear approach is used for eight power components due to their non-linear behaviour similar to how it was done in [34]. Finally, dynamic power is modelled as follows [41]:

$$DP\_SMs = Max\_SM \times log_{10}(\alpha \times Active\_SMs + \beta)$$
$$Max\_SM = Num\_SMs \times \sum_{i=0}^{n} SM\_Component_i$$
$$DynamicPower = (Max\_SM + DP\_Memory) \times log_{10}(\alpha \times Active\_SMs + \beta)$$
$$\alpha = (10 - \beta)/Num\_SMs, \beta = 1.1$$

where $Active\_SMs$ is the number of active SMs in the GPU.

To demonstrate the accuracy of their model, NVIDIA GTX280 GPU is used. The number of dynamic instructions and instruction types (which are used to calculate the access rates) are determined using a GPU PTX emulator, Ocelot ([209]). The authors [41] report that the IPP prediction error for the total power consumption is 8.94% and the average energy consumption savings are 10.99%. The main factor hindering the portability of this model is that it requires detailed architectural information and contains a large set of parameters.

A.2.3 High Performance Computing Applications

[60] present energy prediction model for MPI broadcast algorithms in large scale HPC systems. They present models for four broadcast algorithms, Scatter and AllGather algorithm (MPI/SAG) [210] used in MPICH2 [211], Pipelining algorithm (MPI/Pipeline) provided in OpenMPI 1.4.4, hybrid broadcasting algo-

rithm which combines MPI/SAG and OpenMP, and a hybrid algorithm, which combines MPI/Pipeline and OpenMP [212]. The energy consumption of MPI broadcast operation (MPI/SAG) is modelled as follows [60]:

$$\begin{aligned} E_{MPI/SAG} &= \sum_{i=1}^{N} e_{MPI/SAG}^{Node_i} + \sum_{j=1}^{M} e_{MPI/SAG}^{Switch_j} \\ &= t_{Scatter}(p,N) \times (\sum_{i=1}^{N} \rho_{Scatter}^{Node_i}(p) + \sum_{j=1}^{M} \rho_{Scatter}^{Switch_j}) \\ &+ t_{AllGather}(p,N) \times (\sum_{i=1}^{N} \rho_{AllGather}^{Node_i}(p) + \sum_{j=1}^{M} \rho_{AllGather}^{Switch_j}) \end{aligned} \tag{A.8}$$

$t_{op}(p,N)$ is the time required to perform op (Scatter, AllGather, Pipeline, or CopyPrivate) over the $N \times p$ processes, where N is the number of nodes. Within each node i, p processes are involved in the execution of op. $\rho_{op}^{Node_i}(p)$ is the power consumption of the node i during t_{op}. $\rho_{op}^{Switch_j}(p)$ is the power consumption of the switch j during t_{op}. They validate their energy prediction model on Grid5000 [153] and they report a worst prediction error of -6.82%.

[59] explore energy and performance trade-offs of data movement and I/O at extreme scales. Their energy consumption model follows:

$$E = E_{system} + E_{comm} \tag{A.9a}$$

$$E_{system} = T \times (P_{static} + P_{dynamic}) \tag{A.9b}$$

$$P_{static} = P_{cpu}^{base} + P_{mem}^{base} + P_{nic}^{base} + P_{misc}^{base} \tag{A.9c}$$

$$P_{dynamic} = a \times P_{cpu}^{active} + b \times P_{mem}^{active} \tag{A.9d}$$

$$P_{cpu}^{active} = C \times V^2 \times \alpha \times f \tag{A.9e}$$

where P_{cpu}^{base}, P_{mem}^{base}, P_{nic}^{base}, and P_{misc}^{base} are the base power consumptions of CPU, DRAM, NIC, and other miscellanous components. Dynamic power is predicted by using the capacitance (C), switching activity (α), operational voltage (V), and frequency (f). The parameters a and b are determined using the number of arithmetic operations per second and the number of memory accesses per second. The energy consumption model of a MPI communication operation is

constructed as follows [59]:

$$E_{comm} = \sum_{i=0}^{M} \frac{data_i}{BW_{net}} \times P_{transfer}, \quad if \quad smp(src_i) \neq smp(dest_i) \tag{A.10a}$$

$$E_{comm} = \sum_{i=0}^{M} \frac{data_i}{BW_{mem}} \times (P_{cpu}^{active} + P_{mem}^{active}), \quad if \quad smp(src_i) = smp(dest_i) \tag{A.10b}$$

where $smp(i) = smp(j)$ indicates that MPI ranks i and j are mapped to cores that share memory, BW_{net} and BW_{mem} are the bandwidth values of network and memory respectively, and $P_{transfer}$ depends on the characteristics of the underlying network (for example, Infiniband, Gemini).

A.2.4 Power and Energy Models: Parameters

Table A.1: Parameters and Decomposition characteristics of the CPU Power and Energy Models Surveyed.

Model	Parameters	Decomposition
[33]	microoperation, floating-point operations, second-level address strobes, memory transactions **[33]**	single-core CPU

[34]	128bit_MMX_uop, 64bit_MMX_uop, BPU Fetch Req, Branch Retired, BSQ Cache Ref, Bus Ratio, Front End Event, FSB Data Activity, IOQ Allocation, ITLB Reference, Ld Port Replay, Machine Clear, MOB Load Replay, packed_DP_uop, packed_SP_uop, scalar_DP_uop, scalar_SP_uop, St Port Replay, TC Deliver Mode, Uop Queue Writes, Uops retired, uop_type, x87_FP_uop, x87_SIMD_moves_uop **[34]**	1st Level BPU, 2nd Level BPU, Allocation, Bus Control, Data TLB, FP Exec, FP Regfile, Inst Dec, Inst Queue1, Inst Queue2, INT Exec, INT Regfile, ITLB & Fetch, L1 cache, L2 Cache, MEM control, MOB, Rename, RETIRE, Schedule, Trace Cache, Ucode ROM **[34]**
[35]	$C_{base}, U_{CPU}, U_{Disk}, U_{Net}$	CPU, disk, network
[36]	C_{base}, U_{CPU}, U_{Mem}, U_{Disk}, U_{Net}	CPU, memory, disk, network

[48]	ALU Latency, Branch, Branch Latency, Depth, D-L1 Cache Size, Fixed-Point/Memory, Floating-Point, Floating-Point (FP), FP-Divide Latency, FPU Latency, Functional Units, FX-Divide Latency, FX-Multiply Latency, General Purpose (GP), I-L1 Cache Size, L2 Cache Latency, L2 Cache Size, Load/Store Latency, L/S Reorder Queue, Main Memory Latency, Special Purpose (SP), Store Queue, Width **[48]**	single-core CPU
[37]	P_{base}, P_{max}, U	single-core CPU
[37]	C_{base}, U_{CPU}, r	single-core CPU
[49]	Ambient_Temp0, Ambient_Temp1, CPU0 Die Temp, CPU1 Die Temp, HT1 Bus X-Actions, HT2 Bus X-Actions, L1/L2 Cache Miss (Core 0 to Core 3), Disk Bytes Read, Disk Bytes Written, $P_{spin-up}$, t_{su}, P_{read}, N_r, t_r, P_{write}, N_w, t_w, P_{base}, t_{base}, RPM_{Fan}, RPM_{base}, P_{fan}, $t_{ipmi-slice}$, $P_{optical}$, $t_{optical}$, $V_{pow-line}$, $I_{pow-line}$, $t_{timeslice}$	Cor0 to Core 3, DRAM, HDD, Fan, Support Chipsets
[38]	C_{base}, U_{CPU}, $U_{I/O}$	CPU, I/O

[39]	P_{base}, V_{max}, f_{max}, C_{eff}, U_C, n, s_i, p, γ, β, a, b, c, d, e **[39]**	CPU, RAM, NIC, HDD, Fan
[51]	$AR_{(comp,core)}$, P_{comp}, P_{static} where $core$ = 0,1, $comp$ = BPU, FE, FP, FSB, INT, L1, L2, SIMD **[51]**	BPU and branch execution, DECODE, FETCH_UNIT, Floating point arithmetic units, FSB, Integer arithmetic units, L1, L1 DTLB, L1_ICACHE, L1_ITLB, L2, L2 DTLB, LD/ST execution, LSD, memory, MOB, PREDECODE, RAT, RETIRE, ROB, SIMD arithmetic units, SPT, uCODE ROM, uOP BUFFER **[51]**

[40]	*For server systems*, parameters are: Cycles, DMA misses, Fetched μops, Halted cycles, Interrupts, L3 Cache misses, Processor memory bus transactions, TLB misses, Uncacheable accesses **[40]** *For desktop systems*, parameters are: CPU clock frequency, CPU to I/O transactions, CPU voltage, DC accesses, DCTPage-Conflicts, DCTPageHits, DCTPageMisses, DRAM active percent, Fetched μops, FP μops retired, GPU nongated clocks, %Halted/%Not-Halted, Link active percent, Spindle active percent, Temperature **[40]**	Chipset, CPU, Disk, I/O, Memory, Memory Controller **[40]**

Table A.2: Parameters and Decomposition characteristics of the GPU Power and Energy Models Surveyed.

Model	**Parameters**	**Decomposition**

[41]	α, β, Num_SMs, $Active\_SMs$, $Predicted\_Exec\_cycles$, $\#Threads\_per\_block$, $\#Threads\_per\_warp$, $\#Blocks$ $DAC\_per\_th_{component}$, where component = Alu, Const_cache, Const_SM, FDS, Fp, GlobalMem, Int, LocalMem, Reg, Sfu, Shared, Texture_cache **[41]**	ALU, Constant cache, FDS (Fetch/Dec/Sch), Floating Point Unit, Global memory, Int. arithmetic unit, Local memory, Register File, SFU, Shared memory, Texture cache **[41]**
[44]	branch, divergent_branch, gld_128b, gld_32b, gld_64b, gst_128b, gst_32b, gst_64b, instructions, local_load, local_store, tlb_miss, warp_serialize **[44]**	GPU
[42]	ALU, Atomic, BankConf, Barrier, Branch, CMInst.hit, CMInst.miss, D.Branch, D.FP, GMInst, INT, LMInst, M.Barrier, Occupancy, PMInst, Register, S.FP, SFU, ShMInst, TMInst.hit, TMInst.miss, UncoalesMem **[42]**	GPU
[43]	$N_{SM,i}$, P_i, U_i, B_i **[43]**	Floating Point Unit, Global Memory, Shared Memory **[43]**

[45]	branch, divergent branch, gld request + l1 global load hit + l1 global load miss, global store transaction, inst executed, local load, local store, Shared load + l1 shared bank conflict, tex0 cache sector misses, tex0 cache sector queries, t_{pci}, t_{kernel}, $\bar{P}_{base}$, $E_{parallel-overhead}$ **[45]**	Floating Point Unit, Global Memory, Shared Memory, Local Memory, Texture Cache **[45]**

[53]	access time, access time, address bus width, associativity, associativity, #banks, #banks, buffer line size, chip coverage, cycle time, cycle time, data bus width, decoded opcode width, duty cycle, #entries, #entries, flit bits, input line width, input line width, I/O buffer entries, link latency, link throughput, #memory channels, output line width, output line width, peak transfer rate, percentage of pipelining, pipeline stages, #ports (in, out), #ports(R, W, RW), #ranks, request window entries, router or bus, selection input size, selection output size, tag width, tag width, #virtual channels, width **[53]**	Block/Warp States, Cache Buffers, Constant Cache, Data TLB, Fetch Queue, Instruction Cache, Instruction Decoder, Instruction Issue Selection Logic, Instruction Queue, Instruction TLB, L1 Cache, L2 Cache, LD/ST units, Memory Controller, NoC, PipelineLatches, Register File, Scoreboard, SFU, Shared Memory, SP, Texture Cache **[53]**
[46]	A, S_a, N_{inst}, T_{arc} **[46]**	GPU

[52]	ALUBusy, ALUFetchRatio, ALUInsts, ALUPacking, FastPath, FCStacks, FetchInsts, FetchSize, FetchUnitBusy, GPR, LDSBankConfict, LDSFetchInsts, LDSWriteInsts, LocalMemSize, ScratchRegs, Wavefronts, WriteInsts, WriteUnitStalled **[52]**	GPU

Table A.3: Parameters and Decomposition characteristics of the Intel Xeon Phi Power and Energy Models Surveyed.

Model	**Parameters**	**Decomposition**
[54]	$EPI_{accessmode,optype}$ where $accessmode$ = $Register$, $L1$, $L2$, $Mem\_with\_prefetch$, $Mem\_without\_prefetch$, $Write\_to\_mem$, and $optype$ = $ScalarOp$, $VectorOp$, $vprefetch0\_to\_L1$, $vprefetch1\_to\_L2$ **[54]**	Compute, Hardware PF, MEM, Private Cache, Redundant SW-PF, Remote Cache, Software PF **[54]**

Table A.4: Parameters and Decomposition characteristics of the Power and Energy Models used in the HPC Applications surveyed.

Model	**Parameters**	**Decomposition**
[55]	$\Delta Cycles$, L1 Total Cache Access, L2 Total Cache Access, L3 Total Cache Access, Total Instructions Retired **[55]**	L1 cache, Core Logic, L3 Cache, L2 Cache **[55]**

[56]	N, NB, P, Q	Node
[57]	ti, tj, tk (i, j, k tiles), ti_1, tj_1, tk_1 (trsm tiles), tj (loop j tile), ti_2, tj_2, tk_2 (MM tiles), CPU freqs ($freq$), matrix sizes ($msize$), ui, uj (i, j unrolls), ui_1, uj_1 (trsm unrolls), ui_2, uj_2 (MM unrolls) **[57]**	CPU, DIMM
[58]	Cache_FLD_per_instruction, LD_ST_stall_ per_cycle, PAPI_BR_INS, PAPI_L1_DCM, PAPI_L1_ICA, PAPI_L1_TCA, PAPI_L1_TCM, PAPI_L2_ICM, PAPI_L2_TCA, PAPI_L2_TCH, PAPI_RES_STL, PAPI_TLB_DM, PAPI_TOT_INS **[58]**	CPU, Memory
[61]	FP operations, last level cache misses, Retired instructions, stalled cycles **[61]**	Integer, Floating point, L1 cache, L2 Cache, L3 Cache, Memory **[61]**

[62]	_DISP, _DOUBLE_ISSUED, PAPI_FP_INS, PAPI_INT_INS, PAPI_L1_DCM, PAPI_L2_DCM, PAPI_L3_DCM, PAPI_L3_DCR, PAPI_TOT_CYC, PAPI_TOT_INS, PM_CMPLU_STALL, PM_CMPLU_STALL_THRD, PM_L1_ICACHE_MISS, PM_L2_INST_MISS, PM_L3_MISS, PM_L3_PREF_MISS, PM_LSU_DC_PREF, PM_LSU_FX_FIN, PM_LSU_LDF, PM_LSU_LDX, PM_MEM0_PREFETCH, PM_VSU_FMA_DOUBLE, PM_VSU_SIMPLE_ISSUED, PM_VSU_VECTOR, PM_VSU_VECTOR, _SINGLE_ISSUED, _STREAM_CONFIRM **[62]**	CPU

A.3 Case Study: Performance Monitoring Counters

Table A.5 shows the Likwid [2] performance groups and performance counters (PMCs) that are used as parameters in the regression model in the experiments.

Table A.5: Likwid [2] performance groups and performance counters (PMCs)

Performance Group	**Performance Monitoring Counters**
BRANCH (Branch prediction miss rate/ratio)	BR_MISP_RETIRED_ALL_BRANCHES, BR_INST_RETIRED_ALL_BRANCHES **[2]**
DATA (Load to store ratio)	UOPS_RETIRED_ALL, MEM_UOPS_RETIRED_STORES, MEM_UOPS_RETIRED_LOADS **[2]**
ICACHE (Instruction cache miss rate/ratio)	ICACHE_ACCESSES, ICACHE_MISSES, ICACHE_IFETCH_STALL, ILD_STALL_IQ_FULL **[2]**
L2CACHE (L2 cache miss rate/ratio)	L2_RQSTS_MISS, L2_TRANS_ALL_REQUESTS **[2]**
L2 (L2 cache bandwidth in MBytes/s)	L1D_REPLACEMENT, L2_TRANS_L1D_WB **[2]**
L3CACHE (L3 cache miss rate/ratio)	UOPS_RETIRED_ALL, MEM_LOAD_UOPS_RETIRED_L3_MISS, MEM_LOAD_UOPS_RETIRED_L3_ALL **[2]**

L3 (L3 cache bandwidth in MBytes/s)	L2_TRANS_L2_WB, L2_LINES_IN_ALL **[2]**
TLB_DATA (L1 Data TLB miss rate/ratio)	DTLB_STORE_MISSES_WALK_DURATION, DTLB_STORE_MISSES_CAUSES_A_WALK, DTLB_LOAD_MISSES_WALK_DURATION, DTLB_LOAD_MISSES_CAUSES_A_WALK **[2]**
TLB_INSTR (L1 Instruction TLB miss rate/ratio)	ITLB_MISSES_CAUSES_A_WALK, ITLB_MISSES_WALK_DURATION **[2]**
UOPS_EXEC (UOPs execution)	UOPS_EXECUTED_USED_CYCLES, UOPS_EXECUTED_STALL_CYCLES, CPU_CLOCK_UNHALTED_TOTAL_CYCLES, UOPS_EXECUTED_STALL_CYCLES **[2]**
UOPS_ISSUE (UOPs issueing)	UOPS_ISSUED_USED_CYCLES, UOPS_ISSUED_STALL_CYCLES **[2]**
UOPS_RETIRE (UOPs retirement)	UOPS_RETIRED_USED_CYCLES, UOPS_RETIRED_STALL_CYCLES **[2]**
UOPS (UOPs execution info)	UOPS_ISSUED_ANY, UOPS_EXECUTED_THREAD, UOPS_RETIRED_ALL, UOPS_ISSUED_FLAGS_MERGE **[2]**
Fixed function performance counter registers	CPU_CLK_UNHALTED_CORE, INSTR_RETIRED_ANY, CPU_CLK_UNHALTED_REF **[2]**

A.3.1 Case Study: Regression Model Coefficients

Table A.6 shows the multiple linear regression coefficients obtained for power and energy models.

Table A.6: Multiple linear regression coefficients for power and energy Models

Predictors (Likwid PMCs) [2]	**Model Coefficients for Average Dynamic Power**	**Model Coefficients for Energy**
INSTR_RETIRED_ANY:FIXC0	-4.489847e-09	9.837091e-08
CPU_CLK_UNHALTED_CORE:FIXC1	-8.192357e-09	-4.365642e-08
CPU_CLK_UNHALTED_REF:FIXC2	7.601429e-09	6.070421e-08
BR_INST_RETIRED_ALL_BRANCHES:PMC0	5.102516e-10	-8.391657e-09
BR_MISP_RETIRED_ALL_BRANCHES:PMC1	-1.049940e-11	1.071510e-09
MEM_UOPS_RETIRED_LOADS:PMC0	2.383515e-11	-3.902474e-09
MEM_UOPS_RETIRED_STORES:PMC1	5.102516e-10	-8.391657e-09
ICACHE_ACCESSES:PMC0	-1.049940e-11	1.071510e-09
ICACHE_MISSES:PMC1	5.102516e-10	-8.391657e-09
ICACHE_IFETCH_STALL:PMC2	-1.049940e-11	1.071510e-09
ILD_STALL_IQ_FULL:PMC3	2.383515e-11	-3.902474e-09
L2_TRANS_ALL_REQUESTS:PMC0	-6.054451e-10	3.830272e-07
L2_RQSTS_MISS:PMC1	5.102516e-10	-8.391657e-09
MEM_LOAD_UOPS_RETIRED_L3_ALL:PMC0	-1.049940e-11	1.071510e-09
MEM_LOAD_UOPS_RETIRED_L3_MISS:PMC1	5.102516e-10	-8.391657e-09
DTLB_LOAD_MISSES_CAUSES_A_WALK:PMC0	-1.049940e-11	1.071510e-09

DTLB_STORE_MISSES_CAUSES_A_WALK:PMC1	5.102516e-10	-8.391657e-09
DTLB_LOAD_MISSES_WALK_DURATION:PMC2	-1.049940e-11	1.071510e-09
DTLB_STORE_MISSES_WALK_DURATION:PMC3	2.383515e-11	-3.902474e-09
ITLB_MISSES_CAUSES_A_WALK:PMC0	-6.054451e-10	3.830272e-07
ITLB_MISSES_WALK_DURATION:PMC1	5.102516e-10	-8.391657e-09
UOPS_EXECUTED_USED_CYCLES:PMC0	-1.049940e-11	1.071510e-09
UOPS_EXECUTED_STALL_CYCLES:PMC1	5.102516e-10	-8.391657e-09
CPU_CLOCK_UNHALTED_TOTAL_CYCLES:PMC2	-1.049940e-11	1.071510e-09
UOPS_EXECUTED_STALL_CYCLES:PMC3: EDGEDETECT	2.383515e-11	-3.902474e-09
UOPS_ISSUED_USED_CYCLES:PMC0	-6.054451e-10	3.830272e-07
UOPS_ISSUED_STALL_CYCLES:PMC1	5.102516e-10	-8.391657e-09
UOPS_ISSUED_STALL_CYCLES:PMC3: EDGEDETECT	-1.049940e-11	1.071510e-09
UOPS_RETIRED_USED_CYCLES:PMC0	-6.054451e-10	3.830272e-07
UOPS_RETIRED_STALL_CYCLES:PMC1	5.102516e-10	-8.391657e-09
UOPS_RETIRED_STALL_CYCLES:PMC3: EDGEDETECT	-1.049940e-11	1.071510e-09
UOPS_ISSUED_ANY:PMC0	-6.054451e-10	3.830272e-07
UOPS_EXECUTED_THREAD:PMC1	5.102516e-10	-8.391657e-09

UOPS_RETIRED_ALL:PMC2	-1.049940e-11	1.071510e-09
UOPS_ISSUED_FLAGS_MERGE:PMC3	-6.054451e-10	3.830272e-07

A.3.2 Case Study: Regression Model Training Times

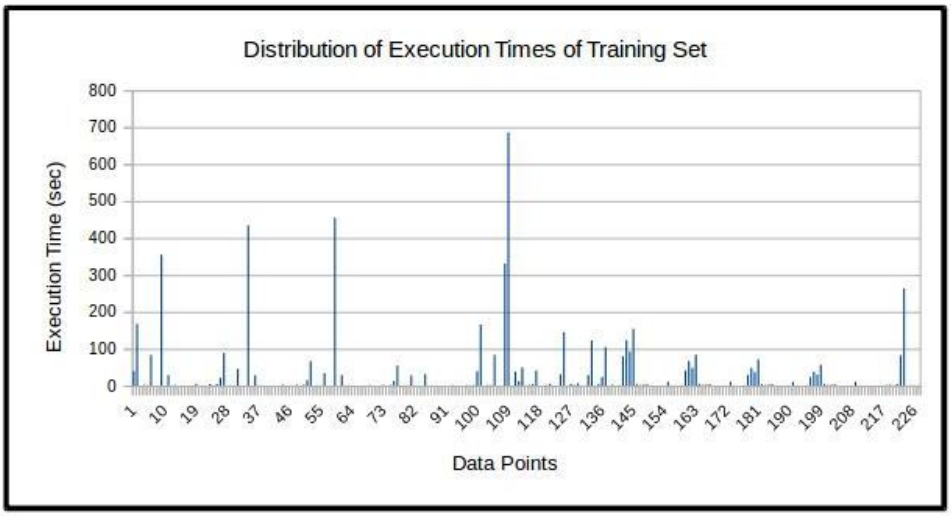


Figure A.1: Execution times of applications in the training set.

Figure A.1 shows the distribution of execution times of applications in the training set.