Exploiting Simple Analytical Models for Modeling Hardware Accelerators

by

Muhammad Shoaib Bin Altaf

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Electrical & Computer Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2016

Date of final oral examination: 12/08/2016

The dissertation is approved by the following members of the Final Oral Committee:

Mark Hill, Professor, Computer Science
Mikko Lipasti, Professor, Electrical & Computer Engineering
Karthikeyan Sankaralingam, Associate Professor, Computer Science
Michael Swift, Associate Professor, Computer Science
David Wood, Professor, Computer Science

To my parents Tehseen Kausar and Sheikh Altaf Hussain, and my wife Iram Majeed for their love and support.

ACKNOWLEDGMENTS

I consider myself fortunate enough to work under the guidance of my advisor, David Wood. I would not have completed my thesis without his support. Working with David, can be a challenge in the beginning and you take time in getting settled with his unique style of mentoring. He gave me the freedom to choose a problem of my own choice but made sure that I stayed on the right path. He has a knack for communicating ideas succinctly, and expects (and forces) his students to develop the same. Thanks to David, I consider myself a better writer and researcher. Thanks David.

I am also thankful to my committee members for providing useful feedback and comments on my work. Mark Hill encouraged and showed excitement about the modeling framework right form the beginning. His advice on making slides has helped me become a better presenter. Mikko Lipasti suggested the retrospective study of the various interfaces. Micheal Swift suggested the name for “Accelerometer", and his system background improved my work in general and this thesis in particular. Karu Sankaralingam challenged several of my assumptions and made me rethink some parts of the modeling infrastructure.

I would also like to thank my architecture colleagues here at University of Wisconsin-Madison, for the technical and non-technical discussions, including Raghuraman Balasubramanian, Arka Basu, Emily Blem, Yasuko Eckert, Chris Feilbach, Jayneel Gandhi, Dan Gibson, Dibarkar Gope, Venkatraman Govindaraju, Gagan Gupta, Swapnil Haria, Joel Hestness, Derek Hower, Marc de Kruijf, Jason Lowe-Power, Jaikrishnan Menon, Tony Nowatzki, Marc Orr, Sankaralingam Panneerselvam, Vignyan Reddy, Somayeh Sardashti, Rathijit Sen, Srinath Sridharan, Nilay Vaish, Aditya Venkataraman, Amit Yadav, and Hongil Yoon. I would also like to thank the computer architecture affiliates for their feedback.

I am also thankful to Newsha Ardalani—my office-mate for last five years—for introduc-

ing me to the Persian cuisine, lengthy political discussions and sharing mutual frustrations of the graduate school.

I am also thankful to Waheed Bajwa, Muhamamd Zeeshan Nazir and Atif Hashmi for helping me during my early days in Madison. I would also like to thank the Pakistani community and my friends at the Pakistani Students Association for making Madison a home away from home, including Farha Ahmed, Rafay Ahmed, Ali Ahsan, Yasmin Flodin Ali, Bilal Idress Allalwala, Ammad Amin, Umar Anjum, Farhat Hashim, Faisal Khan, Nayab Khan, Samina Khan, Farhat Khan, Lamin Manneh, Rehan Rauf, Rehman Rauf, Nabeel Sharif, Sara Siddiqui, Saad Siddiqui, Anoushka Syed, Deeba Syed, Hashim Waris and Hiba Zakai. Special thanks to Neelofer Khilji for the food.

I thank my close friends in Madison: Syed Zohaib Masood Gilani—my apartment-mate for four years—for willingly consuming the meals I made, Arslan Zulfiqar for the late night "chai" sessions, Rehan Ahmed for the on-demand coffee, Zehra Imam for the cooking sessions, Anisa Ali for the skating lessons, and Junaid Khalid for constantly reminding me about graduating.

I would also like to thank my friends and colleagues at the Daily Cardinal including Aarushi Agni, Alison Bauter, Abby Becker, Riley Begin, Sam Cusick, Stephanie Daher, Scott Girard, Kayla Johnsonova, Mark Kauzlarech, Garce Liu, Dylan Moriarty, Grey Satterfield. They provided the opportunity to not only hone my photography skills but also watch the Badger games for free.

I am also thankful to organizations and individuals which supported me during my graduate studies: University of Engineering & Technology, Lahore Pakistan for funding me in the early semesters at UW-Madison; Mark Allie for providing me teaching assistantship for two years; Kevin Moore for the internship opportunity at Oracle Labs; Wisconsin

Alumni Research Foundation (WARF) for the student ambassadorship opportunity; and Brian Wilson at the DoIT and Eric Sedlar at Oracle Labs for providing access to SPARC T3 and T4 servers, respectively.

Completing my graduate studies would have not been possible without the love and support of my family. I am thankful to my parents, Tehseen Kausar and Sheikh Altaf Hussain, for their love, support and lifelong sacrifices. They have been waiting for this day for a long time. I am also thankful to my siblings for providing inspirations right from my childhood. They brought the best out of me by always raising the bar and supporting my decisions. My elder brother, Muhammad Umair Bin Altaf, encouraged me to apply at Wisconsin and made graduate school life experience easier. My sister, Abeera Batool, lent her ear whenever I wanted to vent out my frustrations. She has a talent for convincing people to look at the brighter perspective. I always felt better after talking to her. My younger brother, Muhammad Awais bin Altaf, acted like an elder brother by regularly providing comforting messages and making sure that I am having a good time. He used to share his personal PhD experiences to help me feel better. Thank you for everything.

I am also thankful to my in-laws for keeping me motivated by regularly asking about my graduation plans. Thank you Zulfiqar Majeed, Musarrat Zulfiqar, Sobia Majeed, Affifa Majeed, Anam Majeed and Asim Majeed.

Last but not the least, I'm thankful to my wife, Iram Majeed and son, Shaffan Altaf, for providing the daily reminders that there is more to life than the graduate school. There were times during thesis writing when I did not want to be with myself and Iram still had to bear with me. I would have not succeeded without her sacrifices and support. Love you and thanks for being a part of my life.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the end of Dennard scaling, architects have increasingly turned to special-purpose hardware accelerators to improve the performance and energy efficiency for some applications. Unfortunately, accelerators do not always live up to their expectations and may under-perform in some situations. Understanding the factors which effect the performance of an accelerator is crucial for both architects and programmers early in the design stage. Detailed models can be highly accurate, but often require low-level details which are not available until late in the design cycle. In contrast, simple analytical models can provide useful insights by abstracting away low-level system details.

In this dissertation, we aim to explore the potential of such simple models for hardware accelerators. To this end, we propose two complementary proposals. In our first proposal, we develop a simple analytical model, LogCA, to argue whether an accelerator is helpful for a given task or not. Once the usefulness of an accelerator is established, our second proposal, Accelerometer, helps in identifying bounds and bottlenecks associated with an accelerator design.

We validate our modeling framework across kernels of varying complexity on both on-chip and off-chip accelerators. We also describe the utility of our models using two retrospective case studies. First, we discuss the evolution of interface design in SUN/Oracle's encryption accelerators. Second, we discuss the evolution of memory interface design in three different GPU architectures. In both cases, we show that the adopted design optimizations for these machines are similar to the suggested optimizations. We argue that architects and programmers can use insights from these retrospective studies for improving future designs.

1 INTRODUCTION

> *In recent years, with increasing complexity of the computational systems, empirical methods are now needed to discover system limits and predict future behavior.*
>
> — Clayton Morrison and Richard Snodgrass, *Computer Science can use More Science [4]*

Over the years, research and progress in the compute industry has been driven by the doubling of transistors every two years [5]—a phenomenon known as *Moore's law*. The trend of increasing transistor density and shrinking transistor sizes has been facilitated by *Dennard Scaling* [6], which has kept a chip's power consumption within acceptable limits. This growth has resulted in over billion of transistors on a chip [7] and this trend was expected to continue with *Moore's law*.

Unfortunately, with the breakdown of Dennard scaling, we have hit a *utilization wall* [8], resulting in an exponential decrease in the chip area operating at maximum frequency, with each new generation. This leads to the so-called problem of *Dark Silicon* [9, 10, 11, 12, 13] where some or most of the transistors on a chip have to be in-active, i.e., in a Dark state, to meet the power budget.

This failure of Dennard scaling over the last decade has inspired architects to introduce specialized functional units such as accelerators [14, 15]. These accelerators have shown considerable performance and energy improvement over general-purpose cores for some applications [8, 16, 17, 18, 19, 20, 21, 22]. In the server space, commercial processors already incorporate a variety of accelerators, ranging from encryption to compression, from video streaming to pattern matching, and from database query engines to graphics processing [23,

24, 25]. In the mobile space, we are also witnessing a consistent increase in the area of accelerators across generations. For example, more than 60% of the die area of Apple A8's SoC is dedicated to these accelerators [26]. Moreover, for a seamless integration of these accelerators, there are recent industrial efforts like IBM's Coherent Accelerator Processor Interface (CAPI) [27, 28] and Cache Coherent interconnect for Accelerators (CCIX) [29]. These efforts seek to develop coherent interfaces for achieving higher bandwidth and lower latencies while providing a uniform interface.

Unfortunately, despite these developments, accelerators do not always perform as expected [30, 31]. The gains from an accelerator, among others, depend on: 1) coupling with the host, i.e., whether an accelerator is part of pipeline, attached to a cache, memory or I/O bus, 2) granularity of the computation, i.e., instruction-level, kernel-level or application-level, and 3) overheads involved in transferring control and data from the host to an accelerator. Occasionally, these factors may offset the potential benefits of an accelerator, resulting in lower or no performance benefits.

With this uncertainty in the performance and expected ubiquitousness of accelerators in the future architectures [26], programmers and architects face these obvious questions: When not to use an accelerator? If using an accelerator, how much data to offload? What are the performance bounds associated with an accelerator design? Where to connect an accelerator in the system and how does it affect the gains? Does a performance-efficient solution guaranty energy efficiency and vice versa?

To answer these questions, programmers and architects can employ a number of existing performance analysis techniques. These techniques range from simple to complex analytical models and from functional to full-system simulation models [32, 33].

Complex modeling techniques, including full-system simulators [34, 35, 36] can provide

highly accurate performance estimates. However, this accuracy comes at the cost of long simulation times and low-level system details which are not available till late in the design cycle.

In contrast, analytical models—simpler ones in particular—abstract away these low-level system details using mathematical equations [37]. These equations help in understanding and analyzing complex systems by exposing the relationship among various design parameters and the sensitivity of the performance on these parameters without long simulation times. Consequently, providing key insights early in the design cycle that are useful for experts and non-experts alike.

Over the years, a number of simpler analytical models have been extremely useful in analyzing complex systems: Little's law [38] delineates the relationship between average wait time and number of waiting items in a system; Amdhal's law [39] provides theoretical limits for parallel computations; Hill and Marty [40] extends Amdhal's law for multi-core architectures; PRAM [41] and LogP [42] for parallel computations; Roofline explicates a bound and bottleneck analysis for the multi-core architectures [1], and recently Cache Calculus [43] for modeling cache performance.

The aforementioned models provide useful insights despite their simplicity. Therefore, in this dissertation we aim to explore the effectiveness of simple analytical models for hardware accelerators.

1.1 Contributions

In this section, we briefly describe our main contributions.

1.1.1 LogCA: A High-Level Unified Performance and Energy Model for Hardware Accelerators

Despite the increasing trend towards heterogeneous architectures, the architecture community lacks a simple model to reason about accelerators. Such a model should help programmers and architects—early in the design cycle—in deciding whether an accelerator is helpful for a given task or not.

To this end, we propose LogCA—a unified performance and energy model for hardware accelerators—in Chapter 4 and 5. LogCA strikes a balance between too simple and too complex models. It provides useful insights for both programmers and architects by abstracting away the underlying architecture. With LogCA, we also formalize performance metrics for predicting the "right" amount of offloaded data and demonstrate that sub-linear and linear complexity algorithms are more sensitive to overheads and communication latencies than higher complexity algorithms. While the general trend may not be surprising, we argue that LogCA is accurate enough to answer important what-if questions very early in the design cycle.

While LogCA does not tell the architect how to design hardware to achieve those values, it can help him/her decide whether a given design will satisfy the high-level performance goals. For architects, we believe an important benefit of LogCA is the ability to formalize appropriate design goals for a new accelerator. History is replete with accelerators whose overheads were so high that they were rarely used, e.g., SuperSPARC I copy engine, Niagara

1 cryptographic accelerator, etc.

An earlier version of this work, encompassing the performance model with linear complexity kernels, appears in Computer Architecture Letters in June 2014 [44]. In this dissertation, we extend the performance model with a complementary energy model and kernels ranging from sub-linear to super-linear complexity. We also discuss the effect of pipelining and parallel execution on the performance.

1.1.2 Accelerometer: A Sensitivity Analysis of the LogCA Parameters

Oftentimes LogCA justifies the use of an accelerator, however the gains from an accelerator are less than expected. In such scenarios, programmers and architects need to know the cause of bottlenecks and whether these bottlenecks need attention of a programmer or an architect.

To this end, we extend LogCA through a sensitivity analysis of its parameters and develop Accelerometer in Chapter 6. The sensitivity analysis helps in visually identifying performance bounds and bottlenecks. Furthermore, the identification of these bottlenecks provides an opportunity for optimization and Accelerometer demarcates these optimization opportunities through various optimization regions. With these optimization regions, programmers and architects can visually identify the potential gains associated with each optimization.

We demonstrate the utility of our model using two retrospective case studies. In the first case study, we consider the evolution of the interface in cryptographic accelerators on Sun/Oracle's SPARC T-series processors. For the second case, we consider the memory interface design in three different GPU architectures: a discrete, an integrated and a heterogeneous system architecture (HSA) supported GPU. In both cases, we show that

the adopted design optimizations for these machines are similar to the model's suggested optimizations. We argue that architects and programmers can use insights from these retrospective studies for improving future designs.

Parts of this work have been submitted for publication. In this dissertation, we include a thorough sensitivity analysis of the parameters for different complexity kernels.

1.2 Thesis Organization

In this dissertation, we provide a brief overview of the taxonomy of accelerators and discuss related work in Chapter 2. We describe our evaluation methodology and workloads in Chapter 3. We present our first proposal, LogCA, in Chapter 4. In Chapter 5, we complement the performance model with an energy model. Chapter 6 presents Accelerometer for identifying bounds and bottlenecks associated with an accelerator design. Finally, we conclude this dissertation in Chapter 7 with pointers to future work.

2 BACKGROUND AND RELATED WORK

This chapter compares and contrasts our work with the prior approaches for modeling accelerator architectures. With the availability of a broad range of accelerators, we find it pertinent to first define an accelerator.

2.1 Accelerator Architectures

Patel and W. Hwu [14] define an accelerator as:

An accelerator is a separate architectural substructure (on the same chip, or on a different die) that is architected using a different set of objectives than the base processor, where these objectives are derived from the needs of a special class of applications. Through this manner of design, the accelerator is tuned to provide higher performance at lower cost, or at lower power, or with less development effort than with the general-purpose base hardware.

Based on this definition, the current breed of accelerators can be classified into various categories. Cascaval et al. [15] provide a taxonomy of these accelerators from an architecture and programming perspective. In this dissertation, we are mostly concerned with the architecture specific classification. Hence, we classify them in the following three categories.

Fixed-Function Accelerators provide the best performance per watt ratio at the cost of flexibility. They are implemented in ASICs or other custom logic and are favorable for frequently used application domains. Examples of these accelerators include: floating point units, cryptographic accelerators, compression accelerators, and database accelerators [25, 24].

Reconfigurable Accelerators provide maximum functional flexibility at the cost of performance. These accelerators are most suitable for those application domains which

are still in the prototyping phase. Accelerators designed using Field Programmable Gate Arrays (FPGAs) are the main examples of this class of accelerators. These accelerators are mostly connected through either PCIe [45] or QPI interface [46, 47]. Recently, there has been a growing interest in using these accelerators in the data centers [48, 49].

Programmable Accelerators provide better flexibility as compared to the fixed-function accelerators. They are more apt for application domains which are not mature or are going through continuous developments. Common examples include the graphics processing units (GPUs), general purpose GPUs (GPGPUs) and the programmable network interface controllers.

In this dissertation, our focus is on the effect of the interface in the design of accelerators. Since reconfigurable accelerators do not provide many options for the interface, we have primarily focused on the fixed-function and programmable accelerators. Although we have not validated our modeling framework for the reconfigurable accelerators, we believe that with the generality and abstraction our model provides, our modeling framework is equally applicable for the reconfigurable accelerators.

2.2 Analytical Models

There is a rich body of work exploring analytical models for prediction of an accelerator's performance [50]. For some models, the motivation is to determine the future trend in heterogeneous architectures [51, 52], and exploring the potential benefits associated with an accelerator [39, 53, 54, 55], whereas for others, the motivation is to determine the right amount of data to offload [56].

2.2.1 Simple Modeling Frameworks

Using simplistic models to predict the performance of a machine has a long history [39, 40, 57, 58, 59, 60, 61, 62, 54, 42]. In the early days of parallel computing, *Amdahl's Law* [39] provided a simple model to predict the performance gains for parallel machines. The model argued that despite the number of parallel resources, the speedup is limited by the serial fraction of an algorithm. Hence, the speedup is given by

$$Speedup = \frac{1}{s + \frac{p}{N}} \tag{2.1}$$

where s is the time spent on the serial part of the program, p is the time spent on the part of the program that can be parallelized and N is the number of processors.

Over the years, Amdahl's law has been extended for various domains, ranging from multi-core architectures [40, 59, 62] to data-centers [60] and from energy-constrained environments [57, 61, 58] to heterogeneous architectures [54, 63].

To this end, Daga et al. [54] revisit Amdahl's law for accelerator-based architectures while analyzing the effectiveness of integrated GPUs over discrete GPUs. They argue that for discrete GPUs, data-copying overheads are high and can not be ignored. They factor in the overheads in the speedup computation, and the speedup is given by

$$Speedup = \frac{1}{s + p^{'} + o} \tag{2.2}$$

where s is the serial fraction of the program, p' is the accelerated parallel fraction and o is the parallel overhead.

We present a similar simple model—LogCA—for accelerators. We argue that despite the peak performance of an accelerator, the achievable speedup is limited by the setup

overheads and communication latencies. Our proposed model, LogCA, is inspired by LogP [42] which is a model for the design and analysis of algorithms for parallel machines.

With LogP, the goal was to develop a simple yet accurate model for parallel computation at a time when the available models were either too simplistic, i.e., zero communication latencies and infinite bandwidth, or too specific, i.e., applicable for a particular configuration of a machine. LogP was developed in an era when a diverse set of parallel machines were emerging and it was difficult to model each of them. To overcome this difficulty and provide a simple interface, LogP abstracted away the underlying details with four parameters. These four parameters represent communication bandwidth (g), communication overhead (o), communication delay (L), and the number of processors (P). We use a similar strategy of using system parameters for abstracting away the low-level details. LogCA retains the *L* and *o* parameters from LogP but as discussed in Chapter 4 its remaining three parameters focus on properties of an accelerator and the offloaded computation.

2.2.2 Early-Stage Modeling Frameworks

Chung et al. [51] in a detailed study predict the future landscape of heterogeneous computing. They extend the work of Hill and Marty [40] by incorporating power, bandwidth and scalability of the unconventional cores (U-cores)—a term they coined to encompass GPUs, FPGA and ASICs. They also include the ITRS roadmap in deriving parameters for their model. Conceptually, their U-core's efficiency parameter is similar to our Acceleration (A) parameter. Similar to our modeling framework, the main goal of this work is not to provide exact numbers but provide high-level ideas of the trade-offs involved in various design decisions.

In another work in early-stage modeling, Hempstead et al. [52] propose Navigo. Navigo

provides a compromise between back-of-the-envelope calculations and detailed simulators. The model takes technology node, input voltage, frequency, number of cores and market selection as inputs. A designer can then sweep across these inputs and observe the expected system output as throughput and power. Overall, Navigo provides an estimate about the amount of specialization required to maintain performance in power-constrained future architectures. In contrast, our early-stage framework helps architects reason about accelerators for a given task.

2.2.3 Modeling Communication Cost

Oftentimes, performance models exclude communication cost while estimating performance on accelerators. However, similar to our model, various studies [64, 54, 53] argue that communication cost can not be ignored. To this end, Nilakantan et al. [64] incorporate communication cost for early-stage modeling of accelerator-rich architectures. Their analytical model also predicts the minimum accelerator area required to reach particular performance goal. Similarly, Daga et al. [54] show the overheads for communicating over the PCIe bus as a major bottleneck in exploiting the full potential of GPUs. They observe that integrated GPUs fail to deliver comparable performance as their discrete counterpart for memory-bound kernels because of the limited bandwidth.

Meswani et al. [53] explore such analytical models for high performance computing (HPC) applications. They argue that HPC applications' code size can be huge, resulting in years of development time. For these applications, it make sense to predict the performance gains before porting an application code to an accelerator. They divide an application into idioms—a particular computation or memory access pattern. Using micro-benchmarks, they evaluate the effect of data size on various idioms, e.g., streaming, gather scatter, stencil,

etc. They later predict the performance of an application by identifying the frequency of occurrence of an idiom in that application. This study is performed on high performance computing applications with GPUs and FPGAs as the target devices.

2.2.4 Predicting Offloaded Data

The idea of offloading computationally intensive workloads to remote servers has been extensively studied for mobile networks. Kumar et al. [56] provide a detailed survey on models which tried to answer the basic question: when (and where) to offload the workloads. We define performance and energy metrics to answer a similar question. Unlike our modeling framework, these models for mobile networks do not provide insights for programmers and architects in designing either the interface or the remote servers.

2.2.5 Performance Prediction on GPUs

A number of studies [65, 55, 66, 67, 68, 69, 70] have been performed for predicting performance on GPU architectures. These studies differ by the complexity of the model and the type of input code. We briefly discuss the most relevant below.

Hong et al. [65] develop an analytical performance model for predicting execution time on GPUs. The model requires the PTX code and uses the number of running threads and parallel memory requests. The authors later extend their model and develop an integrated power and performance model for GPUs [55]. They use the execution time from the earlier timing model to predict the power consumption. The power model also helps in predicting the number of cores to achieve optimal performance per watt.

Sim et al. [70] extend Hong et al.'s [65] work and develop an analytical model to help programmers predict the performance benefits from GPUs. Their model also helps in

identifying the potential bottlenecks and suggest optimizations to alleviate these bottlenecks. The model employs a number of architectural parameters to compute the execution time. From an optimization perspective, apart from thread-level parallelism, the model tries to exploit optimizations in instruction and memory-level parallelism. However, as a down-side it also requires the assembly code of the CUDA kernel.

Lai and Seznec [71, 68] develop TEG, Timing Estimation tool for GPUs, which is an analytical tool for estimating performance on GPUs. The model also helps in quantifying the bottlenecks. Similar to earlier approaches, TEG requires CUDA kernel assembly code and instruction trace for estimating the execution time.

Unlike earlier approaches, which require device specific code, GROPHECY [69] tries to predict the performance of a kernel on a GPU using the CPU code. The model defines a CPU code skeleton—an abstraction of the portion of the CPU code to be ported to the GPU—and use an analytical model to predict the execution time on GPUs.

In terms of goals, our work is closely related to the work of Zhang and Owens [67]. The authors have developed a micro-benchmark-based model for GPUs. The model identifies performance bottlenecks and helps programmers and architects predict performance on GPUs. The model is developed at an instruction and architectural level and determines the bottlenecks by breaking down the total execution time in the instruction pipeline, shared memory, and global memory. Unlike our approach, the model does not predict the potential benefits from these optimizations.

In contrast to studies that require understanding of the underlying architecture, Song et al. [66] use a simple counter-based approach augmented with machine learning to predict power and performance efficiency for GPUs. The model also helps in identifying bottlenecks and suggests optimizations to overcome them.

In general, our work is different from these studies because of the complexity. These models use a large number of parameters to accurately predict the power and/or performance, whereas we limit the number of parameters to reduce the complexity of our model. They also require deep understanding of the underlying architecture. Most of these models also require access to GPU specific assembly or PTX codes. Unlike these approaches, we use CPU code to provide bounds on the performance.

2.3 Roofline Models

In terms of simplicity and motivation, our work in Chapter 6—Accelerometer—closely follows the Roofline model [1]—a visual performance model for multi-core architectures. For the Roofline model, the goal is not to predict the exact performance but provide insights about the relationship among various system parameters.

Roofline employs a “bound and bottleneck" analysis to provide useful insights about the expected performance of a system. It exposes inherent hardware limitations for a particular kernel and suggests several optimizations which programmers can use to fine tune that kernel on a given system.

To establish these bounds, Roofline limits the performance with the peak floating point performance and peak memory bandwidth. For a given system, the operating point of a kernel remains within these bounds (hence the name ’roofline’). The model defines the intersection of these two bounds as the *ridge point*. This point helps programmers and compiler writers understands how close they are to in achieving peak performance. The location of the operating point also helps in determining whether a kernel is compute bound or memory bound. For example, Kernel 1 in Figure 2.1 is memory bound while Kernel 2 is compute bound. Also bottlenecks in a design may limit a computation’s performance way

below the peak performance. Roofline exposes these bottlenecks with performance *ceilings* and suggests various optimizations to reach these *ceilings* and ultimately the roof of the model.

Because of the effectiveness in providing useful insights, Roofline has been extended in the past for various architectures [72, 73, 74]. Below, we discuss some of them briefly.

Choi et al. [72] propose an energy-based counterpart of the Roofline. The model exposes the interplay of time, energy and power using algorithmic and architecture specific parameters. Among these parameters, the number of operations, concurrency, and memory traffic characterizes an algorithm, whereas per operation cost (time and energy) characterizes a particular machine. By considering time and energy simultaneously, this model provides a much broader picture to an algorithm designer.

In contrast to earlier approaches, the Boat Hull model [73] predicts the execution time without using the accelerator specific source code. The authors use their previous work on algorithm classification [75] to partition a given kernel into different classes. After

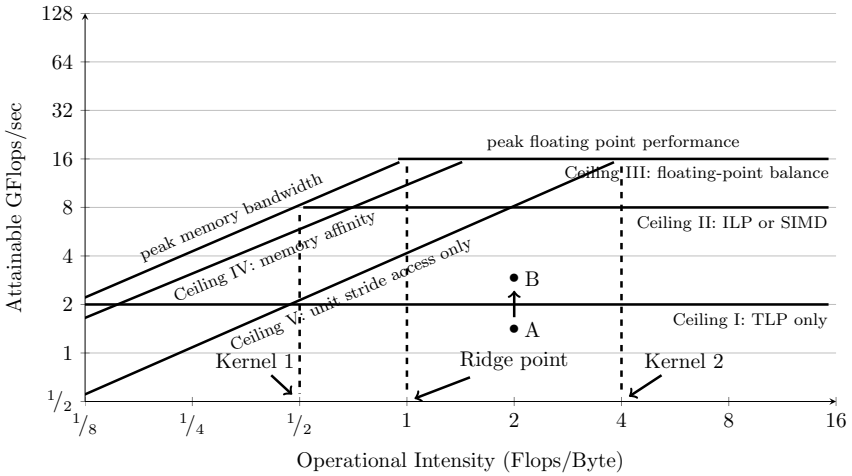


Figure 2.1: Description of the Roofline model [1].

classifying the kernel, the model generates a Roofline for each class. The authors validate the model for both GPUs and traditional multi-core architectures. Unlike Boat Hull, our classification is based on algorithmic complexity. We model complexity of the algorithms using the power-law.

Jia et al. [74] propose GPURoofline, a Roofline model for GPUs. Similar to the original Roofline model, GPURoofline targets programmers who are not expert GPU architects, but need help tuning kernels for a particular GPU architecture. They validate the model for both AMD and NVIDIA GPUs.

In a similar spirit to Roofline, Lai and Seznec [76] propose a model to provide an upper bound on the kernel performance on GPUs. The model identifies the architectural bottlenecks in the systems and scope of optimization for achieving peak performance. Similar to other modeling methodologies, the model requires GPU assembly code as an input.

Despite the similarities, unfortunately Roofline and its extensions can not be used for exposing design bottlenecks in an accelerator's interface for several reasons: First, the primary goal of these models has been to help programmers and compiler-writers write better code for a given system while providing no insights for architects. Second, performance gains for accelerators are measured in speedup while Roofline measures performance in terms of operations per bytes. And third, Roofline does not account for the communication cost of offloading data from the host to an accelerator. Our extension for *LogCA*, *Accelerometer*, is inspired by Roofline and is intended to address these shortcomings.

3 EVALUATION METHODOLOGY

This chapter describes the experimental setup and benchmarks for validating our models on real machines. We also discuss our methodology for measuring model parameters and performance metrics.

3.1 Experimental Setup

Our experimental setup comprises of on-chip and off-chip cryptographic accelerators (Table 3.2) and various GPUs (Table 3.3). The on-chip cryptographic accelerators include cryptographic units on Sun/Oracle UltraSPARC T2 [77], SPARC T3 [78], SPARC T4 [79] and AES-NI (AES New Instruction) [80] on Sandy Bridge, whereas the off-chip accelerator is a Hifn 7955 chip connected through the PCIe bus [81]. The GPUs include a discrete NVIDIA GPU, an integrated AMD GPU (APU), and a heterogeneous system architecture (HSA) supported integrated GPU [82].

For the on-chip cryptographic accelerators, each core in UltraSPARC T2 and SPARC T3 has a physically addressed cryptographic unit. These cryptographic units require privileged DMA calls for operation, which may incur overheads of thousands of cycles. However, the cryptographic unit on SPARC T4 is integrated within the pipeline and does not require privileged DMA calls. SPARC T4 also provides non-privileged cryptographic instructions to access the cryptographic unit, thus reducing the overheads. Similar to SPARC T4, Sandy bridge provides a non-privileged cryptographic instruction – AESNI – for encryption.

The discrete GPU is connected through the PCIe bus, whereas for the APU, the GPU is co-located with the host processor on the same die. For the APU, the system memory is

Kernel	**Algorithmic Complexity**
Advanced Encryption Standard (AES)	$\mathcal{O}(n)$
Secure Hashing Algorithm (SHA)	$\mathcal{O}(n)$
Matrix Multiplication (GEMM)	$\mathcal{O}(n^3)$
Fast Fourier Transform (FFT)	$\mathcal{O}(n \log n)$
Radix Sort	$\mathcal{O}(kn)$
Binary Search	$\mathcal{O}(\log n)$
Discrete Cosine Transform (DCT)	$\mathcal{O}(n^2)$

Table 3.1: Algorithmic complexity of the kernels used in the evaluation

partitioned between host and GPU memory. This eliminates the PCIe bottleneck of data copying but still requires copying data between memories. In both cases, depending on the size of the data, a number of cycles are spent in copying the data from the host memory to device memory. Unlike discrete GPU and APU, HSA supported GPU provides a unified and coherent view of the system memory. With the host and GPU sharing the same virtual address space, explicit copying of data between memories is not required.

3.2 Workloads

In this section, we describe the workloads we have used in our evaluation. We have selected our kernels to cover a diverse set of workloads. These kernels range from sub-linear to super-linear complexity algorithms. Table 3.1 list the algorithmic complexities of these kernels.

Our workloads consist of encryption, hashing and GPU kernels. For encryption and hashing kernels, we have used OpenSSL [83] – an open source cryptography library, and for GPU kernels, we have used AMD OpenCL SDK [84]. We also want to observe how speedup varies with the size of the offloaded data. To do that, we have modified our kernels to enable input size from 16B to 32MB.

3.3 Experimental Methodology

We have used Linux utilities to calculate the execution time on the cryptographic accelerators, whereas for the GPUs we have used NVIDIA and AMD OpenCL profilers to compute the setup, kernel and data transfer times. For our performance numbers, we report the average of one hundred executions. We have modified OpenSSL to enable the support for underlying hardware accelerators [85]. For verifying the usage of cryptographic accelerators, we have used built-in counters in UltraSPARC T2 and T3 [86]. SPARC T4, however, no longer supports these counters, so we have used Linux utilities to trace the execution of the cryptographic instructions [87].

We have measured the average power using “Watts up? PRO" power meter [88]. We compute energy by multiplying measured power by the execution time.

Crypto Accelerator	**PCIe Crypto**	**UltraSPARC T2**	**SPARC T3**	**SPARC T4**	**Sandy Bridge**
Processor	AMD A8-3850	S2	S2	S3	Intel Core i7-2600
Frequency GHz	2.9	1.16	1.65	3	3.4
Caches	L1: 24K L2: 4M	L1: 8K L2: 4M	L1: 8K L2: 6M	L1: 16K L2: 128K	L1: 64K L2: 256K
OpenSSL version	0.98o	0.98o	0.98o	1.02, 1.0.1k	0.98o
Kernel	Ubuntu 3.13.0-55	Oracle Solaris 11	Oracle Solaris 11	Oracle Solaris 11.2	Linux 2.6.32-504

Table 3.2: Description of the Cryptographic accelerators

Platform	**NVIDIA Discrete GPU**	**AMD APU**	**AMD HSA**
Name	Tesla C2070	Radeon HD 6550	Radeon R7
Architecture	Fermi	Beaver Creek	Kaveri
Stream Processors	16	5	8
Compute Units	448	400	512
Device Memory	6G	1G	4G
Peak Core Clock Freq.	1.5 GHz	600 MHz	720 MHz
Peak FLOPS	1 T	480 G	856 G
Host:			
Processor	Intel Xeon E5520	AMD A8-3850	AMD A10-7850K
Frequency GHz	2.27	2.9	1.7
Cache	L1: 32K L2: 256K L3: 8M	L1: 24K L2: 4M	L1: 16K L2: 2M
Kernel	Linux 2.6.32-504	Ubuntu 3.13.0-55	Ubuntu 4.0.0-040050

Table 3.3: Description of the GPUs

4 LogCA: A High-level Performance Model for Hardware Accelerators

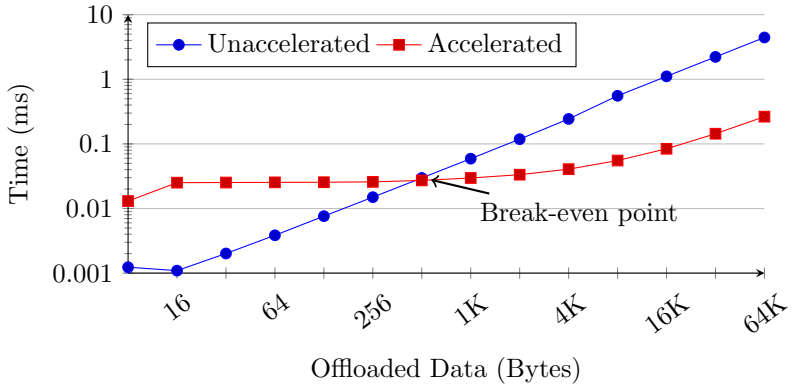
The most that can be expected from any model is that it can supply a useful approximation to reality: All models are wrong; some models are useful.

— George Box

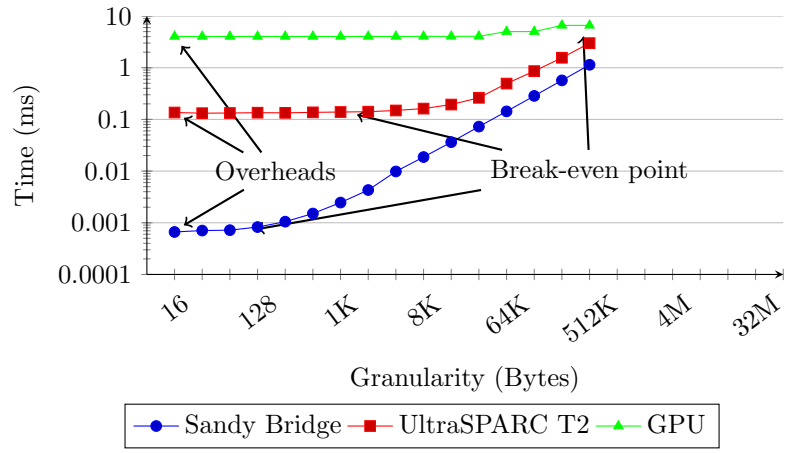
4.1 Introduction

Earlier, we defined an accelerator as a specialized component for providing performance and energy efficiency. Unfortunately, accelerators do not always live up to their name or potential. Offloading a kernel to an accelerator incurs latency and overhead that depends on the amount of offloaded data, location of the accelerator, and its interface with the system. In some cases, these factors may outweigh the potential benefits, resulting in lower than expected or — in the worst case — no performance gains (speedup). Figure 4.1 illustrates such an outcome for the cryptographic accelerator in UltraSPARC T2 running the Advanced Encryption Standard (AES) kernel [2].

Figure 4.1 provides two key observations: First, accelerators can under-perform as compared to a general-purpose core, e.g., the accelerated version in UltraSPARC T2 outperforms the unaccelerated one only after crossing a threshold block size, i.e., the break-even point (Figure 4.1-a); second, different accelerators – while executing the same kernel – have different break-even points and overheads. For example, Sandy Bridge breaks even for smaller offloaded data while UltraSPARC T2 and GPU break even for large offloaded data



(a) Execution time on UltraSPARC T2.



(b) Variation in overheads and break-even points for different cryptographic accelerators.

Figure 4.1: Executing Advanced Encryption Standard (AES) [2] on different cryptographic accelerators. Break-even point (speedup of 1) represents the data at which the accelerated version outperforms the unaccelerated one.

(Figure 4.1-b).

We also observe that a light-weight accelerator, like Sandy Bridge, breaks-even for a smaller offloaded data. It incurs low overheads but provides lesser gains. On the other hand, a heavy-weight accelerator, like a GPU, provides substantial gains at the cost of high overheads and large offloaded data. While offloading large data to get gains may seem a possible solution, applications may have a restriction on the amount of data that can be

offloaded. This is one of the challenges an architect may face in the design and placement of an accelerator.

Understanding the factors which dictate the performance of an accelerator are crucial for both architects and programmers. Programmers need to be able to predict when offloading a kernel will be performance-efficient. Similarly, architects need to understand how the accelerator's interface — and the resulting latency and overheads to offload a kernel — will affect the achievable accelerator performance.

To address these issues, we propose *LogCA*, a performance model for hardware accelerators. LogCA derives its name from the five parameters listed in Table 4.1. These parameters characterize the communication latency (L) and overheads (o) of the accelerator interface, the granularity or size (g) of the off-loaded data, the complexity (C) of the computation, and the accelerator's peak performance improvement (A) as compared to a general purpose core.

LogCA is inspired by LogP [42], the well-known parallel computation model. LogP sought to find the right balance between overly simple models (e.g., PRAM) and the detailed reality of modern parallel systems. LogCA seeks to strike the same balance for hardware accelerators, providing sufficient simplicity such that programmers and architects can easily reason with it. Just as LogP was not the first model of parallel computation, LogCA is not the first model for hardware accelerators [50]. With LogCA, our goal is to develop a simple model that supports the important implications (§4.2) of our analysis using as few parameters as possible while providing sufficient accuracy. In Einstein's words, we want our model *to be as simple as possible and no simpler*.

The rest of the chapter is organized as follows. We develop our model and discuss the resulting implications in Section 4.2. We describe our methodology for calculating model

parameters in Section 4.3, and discuss the results in Section 4.4. Section 4.6 delineates how programmers and architects use LogCA, and we conclude with a summary in Section 4.7.

4.2 The LogCA Model

LogCA assumes an abstract system with three components (Figure 4.2 (a)): *Host* is a general-purpose processor; *Accelerator* is a hardware device designed for the efficient implementation of an algorithm; and *Interface* connects the host and accelerator abstracting away system details including the memory hierarchy.

Our model uses the interface abstraction to provide intuition for the overhead and latency of dispatching work to an accelerator. This abstraction enables modeling of different paradigms for attaching accelerators – directly connected, system bus or PCIe. This also gives the flexibility to use our model for both on-chip and off-chip accelerators. This abstraction can also be trivially mapped to shared memory systems or other memory hierarchies in heterogeneous architectures. The model further abstracts the underlying architecture using the five parameters defined in Table 4.1.

Figure 4.2 (b) illustrates the overhead and latency model for an un-pipelined accelerator where computation 'i' is returned before requesting computation '$i + 1$'. Figure 4.2 (b) also shows the breakdown of time for an algorithm on the host and accelerator. We assume that the algorithm's execution time is a function of granularity, i.e., the size of the offloaded data. With this assumption, the unaccelerated time T_0 (time with zero accelerators) to process data of *granularity* g, will be

$$T_0(g) = C_0(g) \tag{4.1}$$

Parameter	Symbol	Description	Units
Latency	L	Cycles to move data from the host to the accelerator across an un-pipelined interface, including the cycles data spends in the caches or memory	Cycles
Overhead	o	Cycles the host spends in setting up the algorithm	Cycles
Granularity	g	Size of the offloaded data	Bytes
Computational Index	C	Cycles the host spends per byte of data	Cycles/Byte
Acceleration	A	The peak speedup of an accelerator	N/A

Table 4.1: Description of the LogCA model.

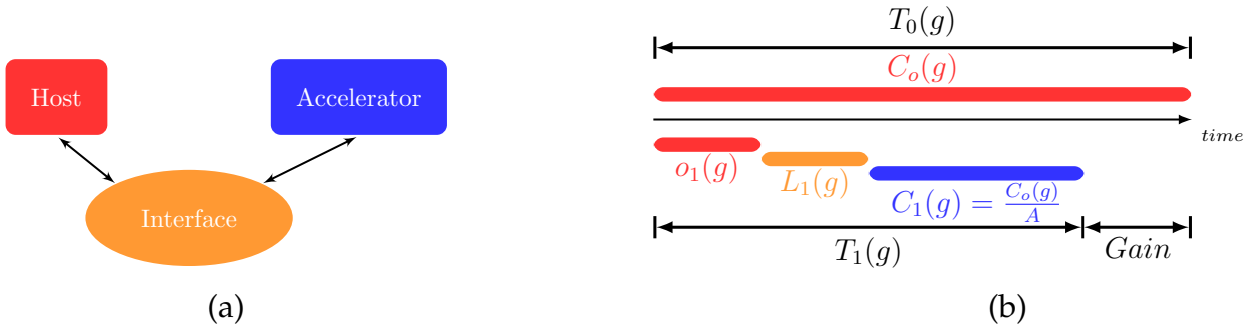


Figure 4.2: Top level description of the LogCA model (a) Shows the various components (b) Time-line for the computation performed on the host (above) and on an accelerator (below)

where $C_0(g)$ is the computation time on the host.

When the data is offloaded to an accelerator, the new execution time T_1 (time with one accelerator) is

$$T_1(g) = O_1(g) + L_1(g) + C_1(g) \tag{4.2}$$

where $O_1(g)$ is the host overhead time in offloading 'g' bytes of data to the accelerator, $L_1(g)$ is the interface latency and $C_1(g)$ is the computation time in the accelerator to process data of granularity g.

To make our model more concrete, we make several assumptions. We assume that an accelerator with acceleration 'A' can decrease, in the absence of overheads, the algorithm's computation time on the host by a factor of 'A', i.e., the accelerator and host use algorithms with the same complexity. Thus, the computation time on the accelerator will be $C_1(g) = \frac{C_0(g)}{A}$. This reduction in the computation time results in performance gains, and we quantify these gains with speedup, the ratio of the un-accelerated and accelerated time:

$$Speedup(g) = \frac{T_0(g)}{T_1(g)} = \frac{C_0(g)}{O_1(g) + L_1(g) + C_1(g)} \tag{4.3}$$

We assume that the computation time is a function of the computational index 'C' and granularity, i.e., $C_0(g) = C * f(g)$, where $f(g)$ signifies the complexity of the algorithm. We also assume that $f(g)$ is power function of 'g', i.e., $\mathcal{O}(g^{\beta})$. This assumption results in a simple closed-form model and bounds the performance for a majority of the prevalent algorithms in the high-performance computing community [89].

The above assumption works well for modeling a variety of kernels, ranging from sub-linear ($\beta < 1$) to super-linear ($\beta > 1$) complexities. However, this assumption may not work well for logarithmic complexity algorithms, i.e., $\mathcal{O}(\log(g)), \mathcal{O}(g\log(g))$. This is because, asymptotically, there is no function which grows slower than a logarithmic function. Despite this limitation, we observe that—in the granularity range of our interest—LogCA can also bound the performance for logarithmic functions.

For many algorithms and accelerators, the overhead is independent of the granularity, i.e., $O_1(g) = o$. Latency, on the other hand, will often be granularity dependent, i.e., $L_1(g) = L*g$. Latency may be granularity independent if the accelerator can begin operating when the first byte (or block) arrives at the accelerator, i.e., $L_1(g) = L$. Thus, LogCA can also model pipelined interfaces using granularity dependent latency assumption.

We define *computational intensity*[1] as the ratio of computational index to latency, i.e., $\frac{C}{L}$ and it signifies the amount of work done on a host per byte of offloaded data. Similarly, we define *accelerator's computational intensity* as the ratio of computational intensity to acceleration, i.e., $\frac{C/A}{L}$ and it signifies the amount of work done on an accelerator per byte of offloaded data.

For simplicity, we begin with the assumption of granularity independent latency. We revisit granularity dependent latencies later (§ 4.2.3). With these assumptions,

$$Speedup(g) = \frac{C * f(g)}{o + L + \frac{C*f(g)}{A}} = \frac{C * g^{\beta}}{o + L + \frac{C*g^{\beta}}{A}} \tag{4.4}$$

The above equation shows that the speedup is dependent on LogCA parameters and these parameters can be changed by architects and programmers through algorithmic and design choices. An architect can reduce the *latency* by integrating an accelerator more closely with the host. For example, placing it on the processor die rather than on an I/O bus. An architect can also reduce the *overheads* by designing a simpler interface, i.e., limited OS intervention and address translations, lower initialization time and reduced data copying between buffers (memories), etc. A programmer can increase the *computational index* by increasing the amount of work per byte offloaded to an accelerator. For example, kernel fusion [90, 91]—where multiple computational kernels are fused into one—tends to increase the *computational index*. Finally, an architect can typically increase the *acceleration* by investing more chip resources or power to an accelerator.

[1] not to be confused with operational intensity [1], which signifies operations performed per byte of DRAM traffic.

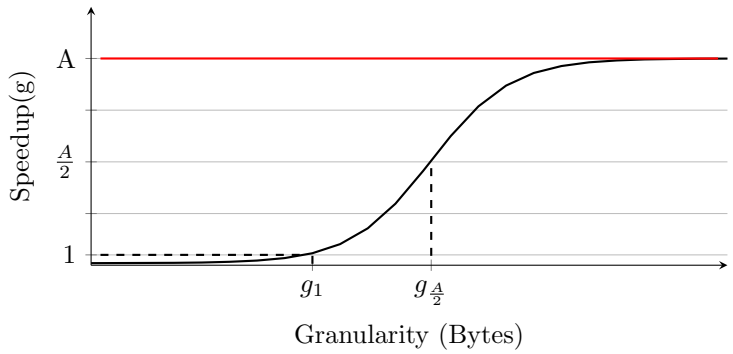


Figure 4.3: A graphical description of the performance metrics

4.2.1 Effect of Granularity

A key aspect of LogCA is that it captures the effect of granularity on the accelerator's speedup. Figure 4.3 shows this behavior, i.e., speedup increases with granularity and is bounded by the acceleration 'A'. At one extreme, for large granularities, equation (4.4) becomes,

$$\lim_{g\to\infty} Speedup(g) = A \tag{4.5}$$

While for small granularities, equation (4.4) reduces to:

$$\lim_{g\to 0} Speedup(g) \simeq \frac{C}{o + L + \frac{C}{A}} < \frac{C}{o + L} \tag{4.6}$$

Equation (4.6) is simply *Amdahl's Law* [39] for accelerators, demonstrating the dominating effect of overheads at small granularities.

4.2.2 Performance Metrics

To help programmers decide *when* and *how much* computation to offload, we define two performance metrics. These metrics are inspired from the vector machine metrics N_v and

$N_{1/2}$[92], where N_v is the vector length to make vector mode faster than scalar mode and $N_{1/2}$ is the vector length to achieve half of the peak performance. Since vector length is an important parameter in determining performance gains for vector machines, these metrics characterize the behavior and efficiency of vector machines with reference to scalar machines. Our metrics tend to serve the same purpose in the accelerator domain.

g_1 : The granularity to achieve a speedup of 1 (Figure 4.3). It is the break-even point where the accelerator's performance becomes equal to the host. Thus, it is the minimum granularity at which an accelerator starts providing benefits. Solving equation (4.4) for g_1 gives:

$$g_1 = \left[\left(\frac{A}{A-1}\right) * \left(\frac{o+L}{C}\right)\right]^{\frac{1}{\beta}} \tag{4.7}$$

Implication 1. *g_1 is essentially independent of acceleration for large values of 'A'.*

For reducing g_1 , the above implication guides an architect for investing resources in improving the interface.

Implication 2. *Doubling computational index reduces g_1 by $2^{-\frac{1}{\beta}}$.*

The above implication demonstrates the effect of algorithmic complexity on g_1 and shows that varying computational index has a profound effect on g_1 for sub-linear algorithms. For example, for a sub-linear algorithm with $\beta = 0.5$, doubling the computational index decreases g_1 by a factor of four. However, for linear ($\beta = 1$) and quadratic ($\beta = 2$) algorithms, g_1 decreases by factors of two and $\sqrt{2}$, respectively.

$g_{\frac{A}{2}}$: The granularity to achieve a speedup of half of the acceleration. This metric provides information about a system's behavior after the break-even point and shows how

quickly the speedup can ramp towards acceleration. Solving equation (4.4) for $g_{\frac{A}{2}}$ gives:

$$g_{\frac{A}{2}} = \left[A * \left(\frac{o+L}{C}\right)\right]^{\frac{1}{\beta}} \tag{4.8}$$

Using equation (4.7) and (4.8), g_1 and $g_{\frac{A}{2}}$ are related as:

$$g_{\frac{A}{2}} = (A-1)^{\frac{1}{\beta}} * g_1 \tag{4.9}$$

Implication 3. *Doubling acceleration 'A', increases the granularity to attain* $\frac{A}{2}$ *by* $2^{\frac{1}{\beta}}$.

The above implication demonstrates the effect of acceleration on $g_{\frac{A}{2}}$ and shows that varying acceleration has a prominent effect on $g_{\frac{A}{2}}$ for sub-linear algorithms. For example, for a sub-linear algorithm with $\beta = 0.5$, doubling acceleration increases $g_{\frac{A}{2}}$ by a factor of four. However, for linear and quadratic algorithms, $g_{\frac{A}{2}}$ increases by factors of two and $\sqrt{2}$, respectively.

For architects, equation (4.9) also exposes an interesting design trade-off between acceleration and performance metrics. Typically, an architect may prefer higher acceleration and lower g_1 , $g_{\frac{A}{2}}$. However, equation (4.9) shows that increasing acceleration also increases $g_{\frac{A}{2}}$. This presents a dilemma for an architect to favor either higher acceleration or reduced granularity, especially for the sub-linear algorithms. LogCA helps by exposing these trade-offs at an early design stage.

In our model, we also use g_1 to determine the complexity of the system's interface. A lower g_1 (on the left side of plot in Figure 4.3) is desirable, as it implies a system with lower overheads and thus a simpler interface. Likewise, g_1 increases with the complexity of the interface or when an accelerator moves further away from the host.

4.2.3 Granularity dependent latency

The previous section assumed latency is granularity independent but we have observed granularity dependent latencies in GPUs. In this section, we discuss the effect of granularity on speedup and derive performance metrics assuming granularity dependent-latency.

Assuming granularity dependent latency, equation (4.3) reduces to:

$$Speedup(g) = \frac{C * g^{\beta}}{o + L * g + \frac{C * g^{\beta}}{A}} \tag{4.10}$$

For large granularities, equation (4.10) reduces to:

$$\lim_{g \to \infty} Speedup(g) = \left(\frac{A}{\frac{A}{C * g^{\beta}} * (L * g) + 1} \right) < \frac{C}{L} * g^{\beta - 1} \tag{4.11}$$

Unlike equation (4.5), speedup in the above equation approaches $\frac{C}{L} * g^{\beta-1}$ at large granularities. Thus, for linear algorithms with granularity dependent latency, instead of acceleration, speedup is limited by $\frac{C}{L}$. However, for super-linear algorithms this limit increases by a factor of $g^{\beta-1}$, whereas for sub-linear algorithms this limit decreases by a factor of $g^{\beta-1}$.

Implication 4. *With granularity dependent latency, the speedup for sub-linear algorithms asymptotically decreases with the increase in granularity.*

The above implication suggests that for sub-linear algorithms, on systems with granularity dependent latency, speedup may decrease for some large granularities. This happens because for large granularities, the communication latency (a linear function of granularity) may be higher than the computation time (a sub-linear function of granularity) on the accelerator, resulting in a net de-acceleration. This implication is surprising as earlier we observed that—for systems with granularity independent latency—speedup

for all algorithms increase with granularity and approaches acceleration for very large granularities.

For very small granularities, equation (4.10) reduces to:

$$\lim_{g \to 0} Speedup(g) \simeq A * \frac{C}{A * (o + L) + C} \tag{4.12}$$

Similar to equation (4.6), the above equation exposes the increasing effects of overheads at small granularities. Solving equation (4.10) for g_1 using Newton's method [93]:

$$g_1 = \frac{C * (\beta - 1) * (A - 1) + A * o}{C * \beta * (A - 1) - A * L} \tag{4.13}$$

For a positive value of g_1, equation (4.13) must satisfy $\frac{C}{L} > \frac{1}{\beta}$. Thus, for achieving any speedup for linear algorithms, $\frac{C}{L}$ should be at least 1. However, for super-linear algorithms a speedup of 1 can achieved at values of $\frac{C}{L}$ smaller than 1, whereas for sub-linear algorithms algorithms, $\frac{C}{L}$ must be greater than 1.

Implication 5. *With granularity dependent latency, computational intensity for sub-linear algorithms should be greater than 1 to achieve any gains.*

Thus, for sub-linear algorithms, computational index has to be greater than latency to justify offloading the work. However, for higher-complexity algorithms, computational index can be quite small and still be potentially useful to offload.

Similarly, solving equation (4.10), using Newton's method, for $g_{\frac{A}{2}}$ gives:

$$g_{\frac{A}{2}} = \frac{C * (\beta - 1) + A * o}{C * \beta - A * L} \tag{4.14}$$

For a positive value of $g_{\frac{A}{2}}$, equation (4.14) must satisfy $\frac{C/A}{L} > \frac{1}{\beta}$. Thus, for achieving

a speedup of $A/2$, $\frac{C}{L}$ should be at least 'A' for linear algorithms. However, for super-linear algorithms a speedup of $\frac{A}{2}$ can achieved at values of $\frac{C}{L}$ smaller than 'A', whereas for sub-linear algorithms, $\frac{C}{L}$ must be greater than 'A'.

Implication 6. *With granularity dependent latency, accelerator's computational intensity for sub-linear algorithms should be greater than 1 to achieve speedup of half of the acceleration.*

The above implication suggests that for achieving half of the acceleration with sub-linear algorithms, the computation time on the accelerator must be greater than latency. However for super-linear algorithms, that speedup can be achieved even if the computation time on accelerator is lower than latency. Programmers can use the above implications to determine—early in the design cycle—whether to put time and effort in porting a code to an accelerator. For example, consider a system with a minimum desirable speedup of one half of the acceleration but has a computational intensity of less than the acceleration. With the above implication, architects and programmers can infer early in the design stage that the desired speedup can not be achieved for sub-linear and linear algorithms. However, the desired speedup can be achieved with super-linear algorithms.

We are also interested in quantifying the limits on achievable speedup due to overheads and latencies. To do this, we assume a hypothetical accelerator with infinite acceleration, and calculate the granularity (g_A) to achieve the peak speedup of 'A'. With this assumption, the desired speedup of 'A' is only limited by the overheads and latencies. Solving equation 4.4 for g_A gives:

$$g_A = \frac{C * (\beta - 1) + A * o}{C * \beta - A * L} \tag{4.15}$$

Surprisingly, we find that the above equation is similar to equation(4.14), i.e., g_A equals $g_{\frac{A}{2}}$. This observation shows that with a hypothetical accelerator, the peak speedup can now be

achieved at the same granularity as $g_{\frac{A}{2}}$. This observation also demonstrates that if $g_{\frac{A}{2}}$ is not achievable on a system, i.e., $\frac{C/A}{L} < \frac{1}{\beta}$ as per equation (4.14), then despite increasing the acceleration, g_A will not be achievable, and the speedup will still be bounded by the computational intensity.

Implication 7. *If a speedup of $\frac{A}{2}$ is not achievable on an accelerator with acceleration 'A', despite increasing acceleration to $\tilde{A}$ (where $\tilde{A} > A$), the speedup is bounded by the computational intensity.*

The above implication helps architects in allocating more resources for an efficient interface instead of increasing acceleration.

4.3 Experimental Methodology

This section describes our methodology for determining LogCA parameters. We measure these parameters using the execution time on host (equation 4.1) and accelerator (equation 4.2) and accelerator specifications. We determine these parameters once and can be later used for different kernels on the same system. In Figure 4.4, we sketch a work flow for determining these parameters.

- **C :** We calculate computational index and β by profiling the CPU code on the host. We profile by varying the granularity from 16B to 32MB. At each granularity, we measure the execution time and use regression analysis to determine C and β.
- **o :** For overheads, we use the observation from equation (4.2) that for very small granularities the execution time for a kernel on an accelerator is dominated by the overheads, i.e.,

$$\lim_{g \to 0} T_1(g) \simeq o$$

- **A :** We use different methods for calculating acceleration on the on-chip accelerators and GPUs. For on-chip accelerators, we calculate acceleration using equation (4.5) and the observation that the speedup curve flattens out and approaches acceleration for very large granularities. We use this observation and track the speedup variation with granularity until it becomes constant. On the other hand, for the GPUs, we use the ratio of peak GFLOPs on CPU and GPU, i.e.,

$$A = \frac{Peak\,GFLOP_{GPU}}{Peak\,GFLOP_{CPU}}$$

  For GPUs, we do not use equation (4.5) as it requires computing acceleration for each kernel, as each application has a different access pattern which affects the speedup. So, we bound the maximum performance using the peak flops from the device specifications.

- **L :** Similar to acceleration, we employ two techniques for calculating latency. For the on-chip accelerators, we run micro-benchmarks and use execution time on host and accelerators i.e., equations (4.1) and (4.2). On the other hand, for the GPUs, we compute latency using peak memory bandwidth of the GPU. Similar to Meswani et al. [53], we use the following equation for measuring data copying time for the GPUs:

$$L = \frac{1}{BW_{peak}}$$

We have developed our model using assumptions of granularity independent and dependent latencies. In our setup, we observe that the on-chip cryptographic accelerators represent accelerators with granularity independent latency while the off-chip cryptographic accelerator and discrete GPU/APU represent the granularity dependent accelerators. For each accelerator, we calculate the speedup and performance metrics using the respective

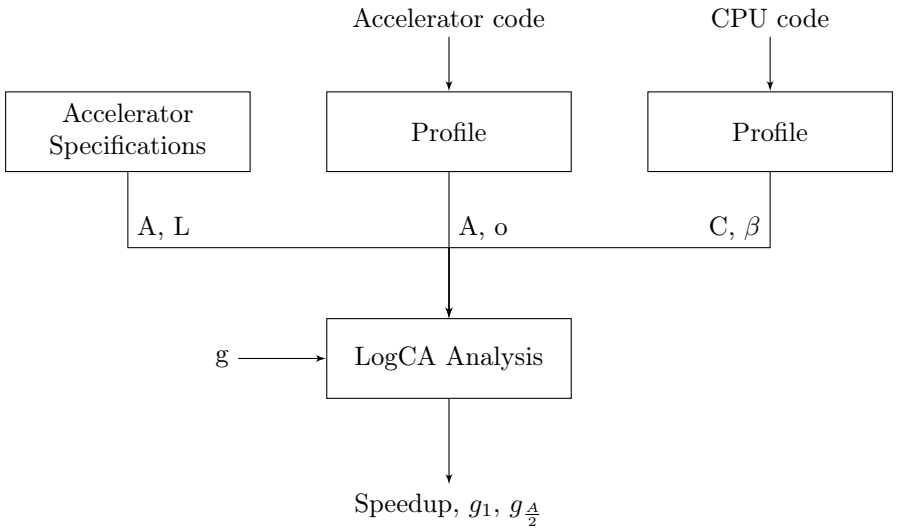


Figure 4.4: Flowchart for calculating LogCA parameters.

Kernel	Algorithmic Complexity	
Advanced Encryption Standard (AES)	$\mathcal{O}(n)$	$\mathcal{O}(g^{1.01})$
Secure Hashing Algorithm (SHA)	$\mathcal{O}(n)$	$\mathcal{O}(g^{0.97})$
Matrix Multiplication (GEMM)	$\mathcal{O}(n^3)$	$\mathcal{O}(g^{1.7})$
Fast Fourier Transform (FFT)	$\mathcal{O}(n \log n)$	$\mathcal{O}(g^{1.2})$
Radix Sort	$\mathcal{O}(kn)$	$\mathcal{O}(g^{0.94})$
Binary Search	$\mathcal{O}(\log n)$	$\mathcal{O}(g^{0.14})$
Discrete Cosine Transform (DCT)	$\mathcal{O}(n^2)$	$\mathcal{O}(g^{0.99})$

Table 4.2: Algorithmic complexity of various kernels with number of elements and granularity. The power of g represents β for each kernel.

equations earlier defined in (§4.2).

In Table 4.2, we list the complexities of each kernel, both in terms of number of elements n and granularity g. We expect these complexities to remain same in both cases, but we observe that they differ for Matrix multiplication and Discrete Cosine Transform. For example, for a square matrix of size n, Matrix Multiplication has complexity of $\mathcal{O}(n^3)$, whereas the complexity in terms of granularity is $\mathcal{O}(g^{1.7})$. This happens because for Matrix Multiplication (and DCT)—unlike others—computations are performed on matrices and

		LogCA Parameters			
Device	**Benchmark**	**L** (cycles)	**o** (cycles)	**C** (cycles/bytes)	**A**
Discrete GPU	AES	3×10^3	2×10^8	174	30
	Radix Sort			290	
	Matrix Multiplication			2	
	FFT			290	
	Binary Search			116	
APU	AES	15	4×10^8	174	7
	Radix Sort			290	
	Matrix Multiplication			2	
	FFT			290	
	Binary Search			116	
UltraSPARC T2	AES	1,500	2.9×10^4	90	19
	SHA		10.5×10^3	72	12
SPARC T3	AES	1,500	2.7×10^4	90	12
	SHA		10.5×10^3	72	10
SPARC T4	AES	500	435	32	12
	SHA		16×10^3	32	10
SPARC T4 instructions	AES	4	111	32	12
	SHA		1,638	32	10
Sandy Bridge	AES	3	10	35	6

Table 4.3: Calculated values of LogCA Parameters.

not vectors. So offloading a square matrix of size n corresponds to offloading n^2 elements, which results in the apparent discrepancy in the complexities.

4.4 Evaluation

In this section, we present our results and show that LogCA closely captures the behavior for different complexity algorithms on both off and on-chip accelerators. We also list the calculated LogCA parameters in Table 4.3.

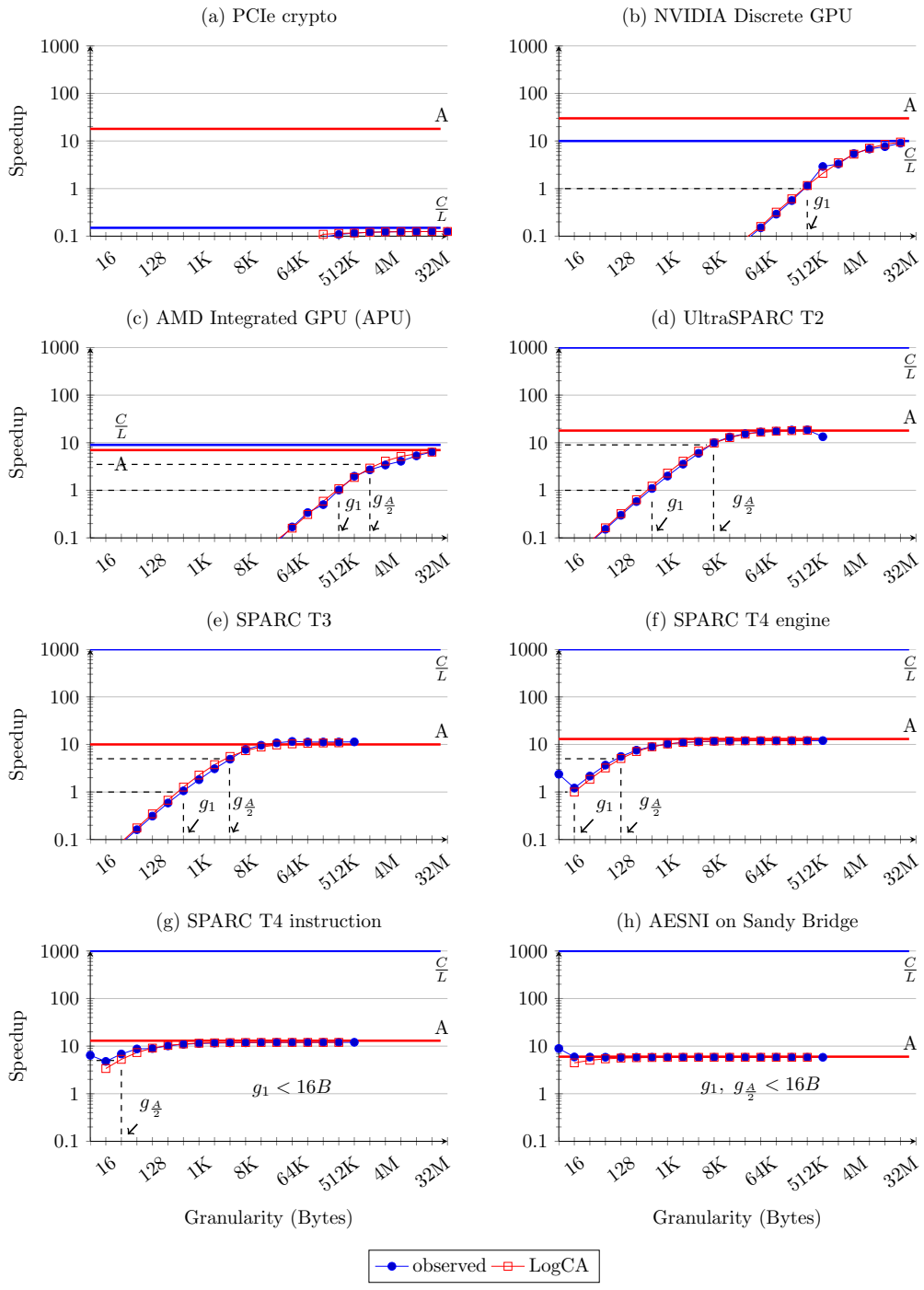


Figure 4.5: Speedup curve fittings plots comparing LogCA with the observed values of AES [2] over a range of granularities. LogCA starts following observed values after 16B.

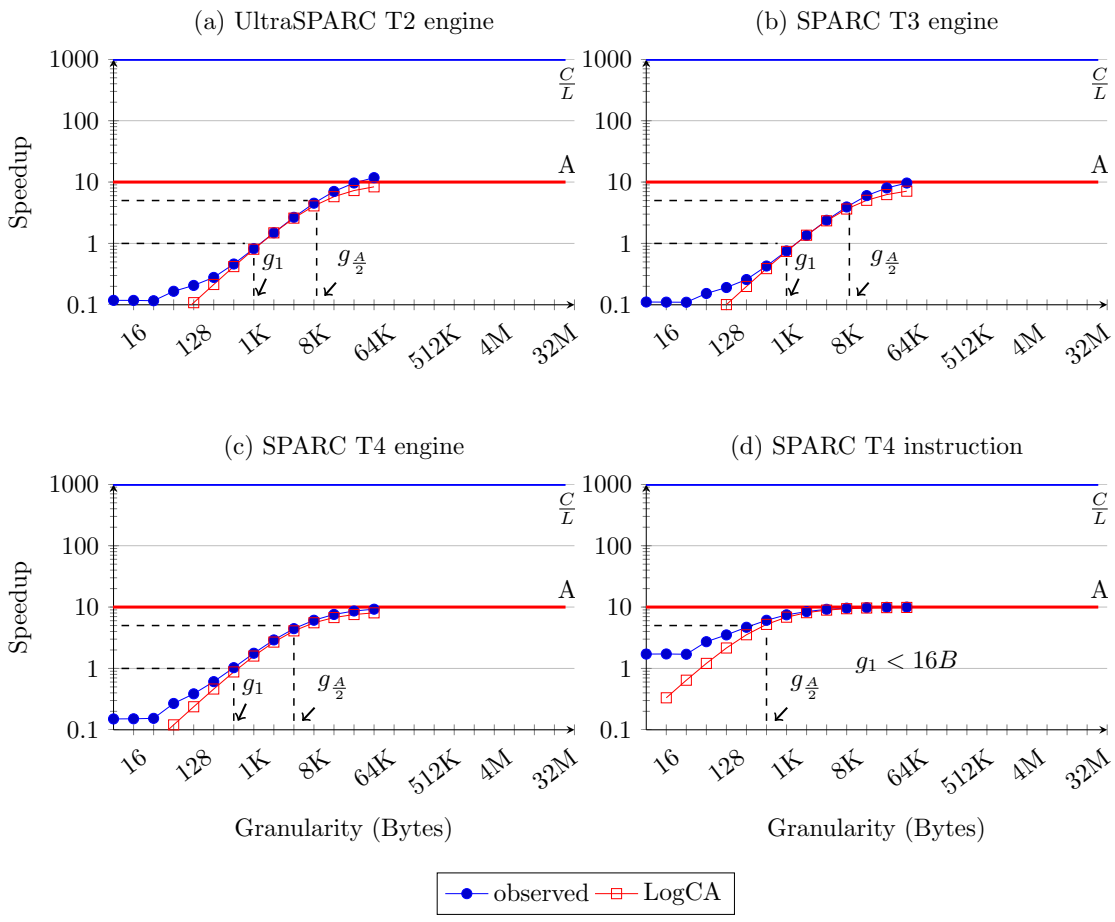


Figure 4.6: Speedup curve fittings plots comparing LogCA with the observed values of SHA256 [3] over a range of granularities. LogCA starts following observed values after 64B.

4.4.1 Linear-Complexity Kernels ($\beta = 1$)

Figure 4.5 shows the curve-fitting of LogCA for AES. We consider both off-chip and on-chip accelerators, connected through different interfaces, ranging from PCIe bus to special instructions. We observe that the off-chip accelerators and APU, unlike on-chip accelerators, provide reasonable speedup only at very large granularities. We also observe that the achievable speedup is limited by computational intensity for off-chip accelerators and

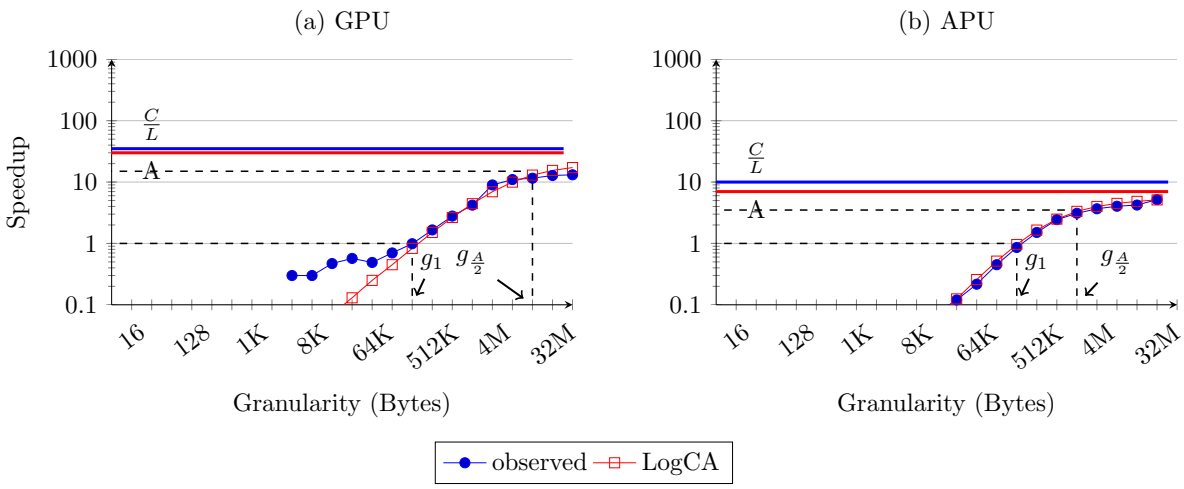


Figure 4.7: Speedup curve fittings plots comparing LogCA with the observed values of Radix Sort over a range of granularities.

acceleration for on-chip accelerators. This observation supports earlier implication on the limits of speedup for granularity independent and dependent latencies in equation (4.5) and (4.11), respectively.

Figure 4.5 also shows that UltraSPARC T2 provides higher speedups compared to Sandy Bridge, but it breaks-even at a larger granularity. Sandy Bridge, on the other hand, breaks-even at very small granularity but provides limited speedup. The discrete GPU with powerful processing cores has the highest acceleration among others. But its observed speedup is less than others due to high overheads and latencies involved in communicating through the PCIe bus.

We have also marked g_1 and $g_{\frac{A}{2}}$ for each accelerator in Figure 4.5 which help programmers and architects identify the complexity of the interface. For example, g_1 for cryptographic instructions, i.e., SPARC T4 and Sandy Bridge, lies on the extreme left while for the off-chip accelerators, g_1 lies on the far right. It is worth mentioning that we have marked $g_{\frac{a}{2}}$ for on-chip accelerators but not for the off-chip accelerators. For off-chip accelerators,

computational intensity is less than acceleration and as we have noted in equation (4.14) that $g_{\frac{A}{2}}$ for these designs does not exist.

We also observe that g_1 for the cryptographic-card connected through the PCIe bus does not exist, showing that this accelerator does not break-even even for large granularities. Figure 4.5 also shows that g_1 for GPU and APU is comparable. This observation shows that despite being an integrated GPU and not connected to the PCIe bus, APU spends considerable time in copying data from the host to device memory.

Figure 4.6 shows the curve fitting for SHA on various on-chip cryptographic accelerators. We observe that g_1 and $g_{\frac{A}{2}}$ do exist, as all of these are on-chip accelerators. We also observe that the LogCA curve mostly follows the observed value. However, it deviates from the observed value before 64B. This happens because SHA requires block size of 64B for hash computation. If the block size is less than 64B, it pads extra bits to make the block size 64B. Since LogCA does not capture this effect, it does not follow the observed speedup for granularity smaller than 64B.

Figure 4.7 shows the speedup curve fitting plots for radix sort. We observe that LogCA does not follow the observed values for smaller granularities on GPU. This happens because LogCA does not model caches, and with a single latency parameter, it can not capture the memory access pattern for both small and large granularities. Despite this inaccuracy, LogCA still accurately predicts g_1 and $g_{\frac{A}{2}}$. We also observe that $g_{\frac{A}{2}}$ for GPU is higher than APU.

4.4.2 Super-Linear Complexity Kernels ($\beta > 1$)

Figures 4.8 and 4.9 show the speedup curve fitting plots for super-complexity kernels on discrete GPU and APU. These Figures provide some key observations: First, we observe

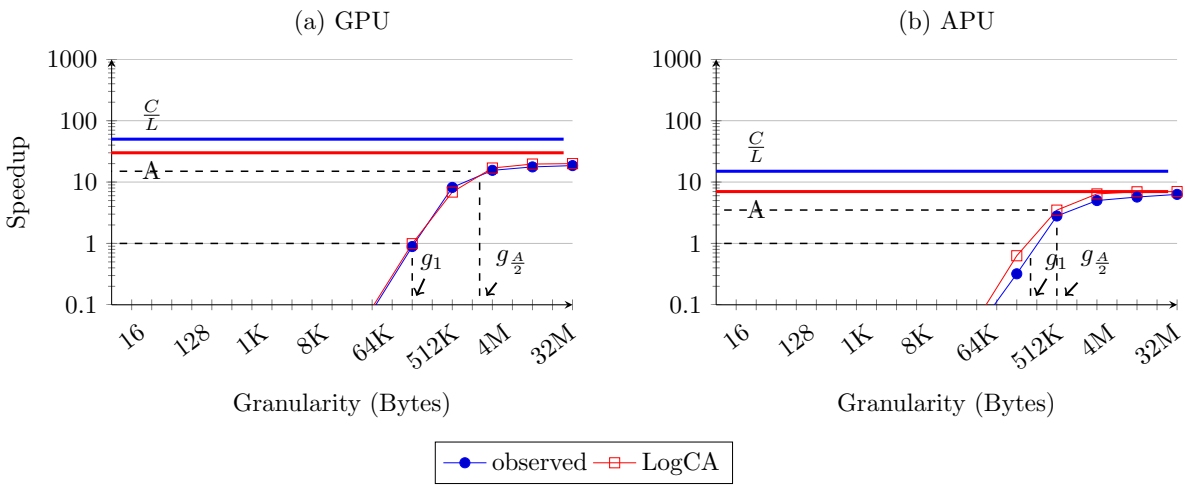


Figure 4.8: Speedup curve fittings plots comparing LogCA with the observed values of Matrix Multiplication over a range of granularities.

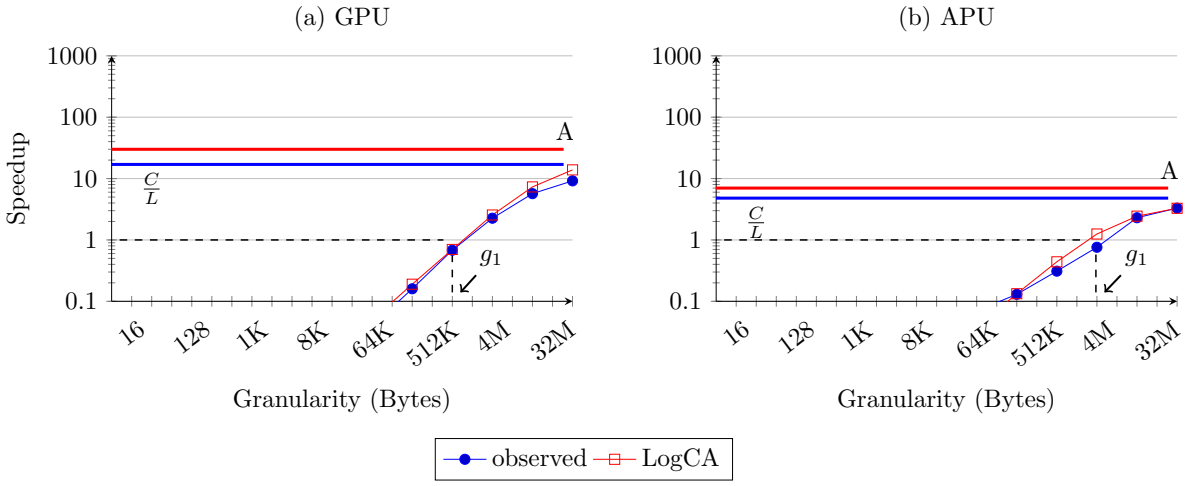


Figure 4.9: Speedup curve fittings plots comparing LogCA with the observed values of FFT over a range of granularities.

that matrix multiplication (Figure 4.8) with higher complexity ($\mathcal{O}(g^{1.7})$) achieves higher speedup as compared to sort (Figure 4.7) and FFT (Figure 4.9) with lower complexities of $\mathcal{O}(g)$ and $\mathcal{O}(g^{1.2})$, respectively. This observation corroborates results from equation (4.11) that achievable speedup of higher-complexity algorithms is higher than lower-complexity

algorithms. We also observe that $g_{\frac{A}{2}}$ does not exist for FFT. This happens because as we note in equation (4.14) that for $g_{\frac{A}{2}}$ to exist for FFT, $\frac{C}{L}$ should be greater than $\frac{A}{1.2}$. However, Figure 4.9 shows that $\frac{C}{L}$ is smaller than $\frac{A}{1.2}$ for both GPU and APU.

4.4.3 Sub-Linear Complexity Kernels ($\beta < 1$)

Figure 4.10 shows the curve fitting for binary search which is a sub-linear algorithm ($\beta = 0.14$). We observe that g_1 does not exist even for very large granularities and $\frac{C}{L} < 1$. This observation supports implication (5) that for a sub-linear algorithm of $\beta = 0.14$, $\frac{C}{L}$ should be greater than 7 to provide any speedup. We also observe that for large granularities, speedup starts decreasing with an increase in granularity. This observation supports our earlier claim in implication (4) that for systems with granularity dependent latencies, speedup for sub-linear algorithms asymptotically decrease. We also observe that LogCA deviates from the observed value at large granularities. This deviation occurs because LogCA does not model caches. As mentioned earlier, LogCA abstracts the caches and memories with a single parameter of latency which does not capture the memory-access pattern accurately.

Even though LogCA does not accurately captures binary search behavior, it still provides an upper bound on the achievable performance. This information can be still helpful for architects and programmers early in the design stage.

Consider a scenario where multiple kernels—with different computational indices—are available for an algorithm. LogCA can helps programmers in identifying the kernel which meets the computational index requirement at a certain g_1 . For example, Figure 4.11 shows the computational indices of various encryption and hashing kernels on UltraSPARC T2. We can make two key observations: First, encryption kernels have high computational index as

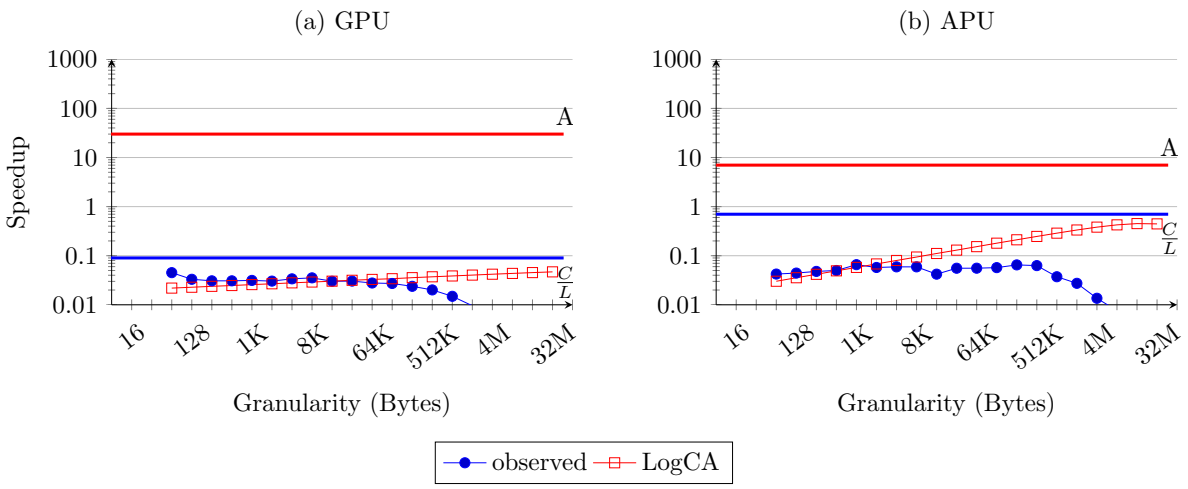


Figure 4.10: Speedup curve fittings plots comparing LogCA with the observed values of Binary Search over a range of granularities.

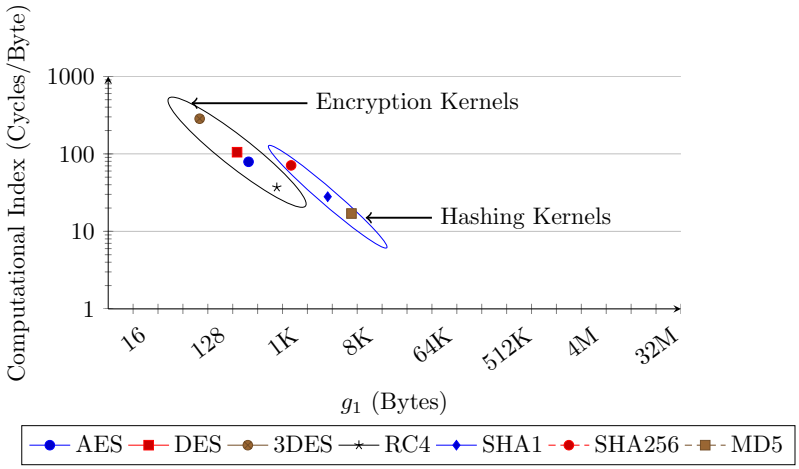


Figure 4.11: g_1 for various encryption and hashing kernels on UltraSPARC T2.

compared to hashing kernel. And second, g_1 decreases with the increase in computational index. This observation also supports earlier implication (2) that doubling computational index reduces g_1 by a factor of 2.

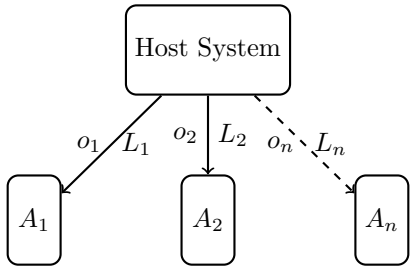


Figure 4.12: System configuration for multiple accelerators running in parallel.

4.5 Case for Multiple Accelerators

Earlier we observed in Section 6.2.1 that for latency bound kernels, speedup is bounded by computational intensity. Therefore, to increase this bound we can either increase computational index or decrease latency. We can decrease the effective latency by overlapping computation with communication. To do this, the host—instead of sitting idle after offloading—can offload the sub-computations to multiple accelerators operating in parallel or pipeline these sub-computations on a single accelerator. In this section, we explore the potential impact of parallelizing and pipelining computations.

4.5.1 Parallel Execution

We consider a system with 'n' accelerators, operating in parallel, each with an acceleration of $A_1, A_2, ..., A_n$. These accelerators are connected through different interfaces, resulting in different overheads, i.e., $o_1, o_2, ..., o_n$ and latencies, i.e., $L_1, L_2, ..., L_n$. Figure 4.12 illustrates such a configuration.

For a computation with computational index 'C', the execution time on the host is:

$$T_0(g) = C_0(g) = C * f(g) \tag{4.16}$$

We assume that a granularity of 'g' can be partitioned into 'n' parts, i.e,

$$\sum_{i=1}^{n} g_i = g$$

For parallel computation, the host offloads a fraction of the total granularity, g_i to each accelerator, and we assume that each fraction has the same computational index 'C'. We also assume that the host sets up the i-th computation after offloading (i-1)st computation. Therefore, the overhead for the first accelerator is o_1, $o_1 + o_2$ for the second, and $\sum_{i=1}^{n} o_i$ for the n-th accelerator and we define the overheads for the n-th accelerator as the lumped overheads, o_n:

$$o_n = \sum_{i=1}^{n} o_i$$

With these assumptions, the execution time on the first accelerator is given by:

$$T_1(g_1) = o_1 + L_1 * g_1 + \frac{C * f(g_1)}{A_1}$$

Similarly, the execution time for the second accelerator is:

$$T_2(g_2) = o_1 + o_2 + L_2 * g_2 + \frac{C * f(g_2)}{A_2}$$

And for the n-th accelerator:

$$T_n(g_n) = o_n + L_n * g_n + \frac{C * f(g_n)}{A_n}$$

Therefore, the total execution time is limited by the accelerator which completes the execu-

tion at the end, i.e.,

$$T_{parallel}(g) = max\left(T_1(g_1), T_2(g_2), ..., T_n(g_n)\right) \tag{4.17}$$

where, $T_1(g_1)$, $T_2(g_2)$, and $T_n(g_n)$ are the execution times on the first, second and n-th accelerator, respectively. The speedup in this case is given by:

$$Speedup_{parallel} = \frac{C * f(g)}{max\left(T_1(g_1), T_2(g_2), ..., T_n(g_n)\right)} \tag{4.18}$$

The above equation exposes some interesting trade-offs in partitioning the granularity among the accelerators. Ideally, we would like to partition the granularity such that each accelerator finishes the work at the same time. To achieve that, an optimal distribution strategy would be such that the accelerator which incurs low overheads and/or latencies, gets a larger portion of granularity.

For a concrete analysis, we make some simplifying assumptions here. We assume that the computation has linear complexity and the granularity 'g' can be partitioned into 'n' equal parts, i.e., $g_i = \frac{g}{n}$. We also assume that each accelerator has the same acceleration, i.e., '$A_i = A$', and is connected through the same interface, i.e., '$L_i = L$'. We also assume that the overheads are constant, i.e., $o_i = o$. With these assumptions, the n-th accelerator finishes at the end—equal distribution of work is not an optimal strategy with these assumptions. Therefore, the total execution time from equation (4.17) reduces to:

$$T_{parallel}(g) = n * o + L * \frac{g}{n} + \frac{C * g}{nA}$$

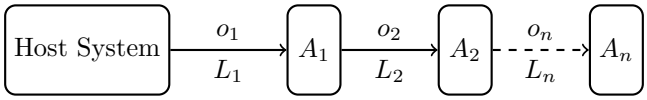


Figure 4.13: System configuration for multiple accelerators running in serial.

And the speedup from equation (4.18) is given by:

$$Speedup_{parallel} = \frac{C * f(g)}{n * o + L * \frac{g}{n} + \frac{C * g}{nA}} \tag{4.19}$$

We observe that the above equation is similar to the speedup equation (4.4) for a single accelerator in Chapter 4. This observation shows that multiple accelerators can be treated as a single accelerator with lumped parameters. For very small overheads, the above equation reduces to:

$$Speedup_{parallel} \approx nA \tag{4.20}$$

However, for large overheads, equation (4.19) reduces to:

$$Speedup_{parallel} \ll nA \tag{4.21}$$

The above equations show that offloading sub-computations to multiple accelerators is beneficial with smaller overheads. As the overheads increase, the potential benefits start decreasing.

4.5.2 Serial Execution

Now we consider a system with 'n' accelerators, but unlike previous subsection, operating serially. Figure 4.13 illustrates such a configuration. Each accelerator has an acceleration of $A_1, A_2, ..., A_n$, and is connected through a different interface, resulting in different over-

heads, i.e., $o_1, o_2, ..., o_n$ and latencies, i.e., $L_1, L_2, ..., L_n$. We also assume that algorithms running on these accelerators have different computational indices, i.e., $C_1, C_2, ..., C_n$. Assuming each accelerator process a fraction of granularity 'g', i.e., 'g_i', the execution time on the host is:

$$T_0(g) = C_0(g) = C_1 * f(g_1) + C_2 * f(g_2) +, ... + C_n * f(g_n) = \sum_{i=1}^{n} C_i * f(g_i) \tag{4.22}$$

Assuming no overlapping between computation and communication, the serial execution time for 'n' accelerators is:

$$T_{serial}(g) = o_1 + L_1 * g_1 + \frac{C * f(g_1)}{A_1} + o_2 + L_2 * g_2 + \frac{C * f(g_2)}{A_2} + ... + o_n + L_n * g_n + \frac{C * f(g_n)}{A_n}$$

$$\implies T_{serial}(g) = \sum_{i=1}^{n} o_i + \sum_{i=1}^{n} L_i * g_i + \sum_{i=1}^{n} \frac{C_i}{A_i} * f(g_i) \tag{4.23}$$

The speedup for such a system will be given by:

$$Speedup_{serial} = \frac{\sum_{i=1}^{n} C_i * f(g_i)}{\sum_{i=1}^{n} o_i + \sum_{i=1}^{n} L_i * g_i + \sum_{i=1}^{n} \frac{C_i}{A_i} * f(g_i)} \tag{4.24}$$

4.5.2.1 Pipelined Execution

Equation (4.24) shows the speedup for 'n' accelerators operating in serial. For a carefully pipelined system where the overheads and link latencies are completely hidden by the accelerator's computation time, equation (4.24) will be given by:

$$Speedup_{pipelined} = \frac{\sum_{i=1}^{n} C_i * f(g_i)}{max_{i \in n}(o_i) + max_{i \in n}(L_i * g_i) + \sum_{i=1}^{n} \frac{C_i * f(g_i)}{A_i}} \tag{4.25}$$

Therefore, in this section we explore the potential for pipelined execution, where these

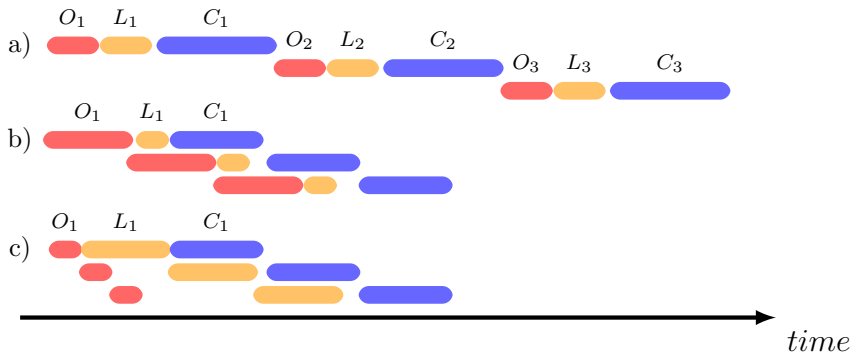


Figure 4.14: Overlapping scenarios for pipelined execution of three computations on an accelerator. (a) The base case of an un-pipelined execution. Pipelined execution when (b) overhead is higher than latency (c) overhead is smaller than latency.

'n' sub-computations are now run on a single accelerator. We assume that the host can start processing i-th component before the completion of the (i+1)-st component of the computation.

Equation (4.25) highlights some additional potential bottlenecks in the system, despite overcoming the overheads and latencies. For example, while operating computations with different computational indices, the one with higher computational index and/or β can become a new bottleneck. Consequently, not allowing other accelerators to proceed.

Similar to the parallel execution, we make some simplifying assumptions for a concrete analysis. We assume that each accelerator process a granularity of 'g', i.e., $g_i = g$. We also assume that each accelerator has the same acceleration, i.e., '$A_i = A$', and is connected through the same interface, i.e., '$L_i = L$'. We also assume that the overheads are constant, i.e., $o_i = o$.

Figure 4.14 shows different scenarios for pipelining three computations. Figure 4.14-a shows the base case, i.e., the un-pipelined execution. Figure 4.14-b shows the scenario

when overhead is larger than latency. In this case, speedup is given by:

$$Speedup_{pipelined} = \frac{n * C * f(g)}{L * g + max[n * o + \frac{C*f(g)}{A}, o + n * \frac{C*f(g)}{A}]} \tag{4.26}$$

Increasing the number of sub-computations, the above equation reduces to:

$$\lim_{n\to\infty} Speedup_{pipelined} = \frac{C * f(g)}{max[o, \frac{C*f(g)}{A}]} \tag{4.27}$$

Similarly, Figure 4.14-c shows the scenario when latency is larger than overheads. In this case, speedup is given by:

$$Speedup_{pipelined} = \frac{n * C * f(g)}{o + max[n * L * g + \frac{C*f(g)}{A}, L * g + n * \frac{C*f(g)}{A}]} \tag{4.28}$$

Increasing the number of sub-computations, the above equation reduces to:

$$\lim_{n\to\infty} Speedup_{pipelined} = \frac{C * f(g)}{max[L * g, \frac{C*f(g)}{A}]} \tag{4.29}$$

Moreover, when overhead and latency are smaller than computation time, speedup is given by:

$$Speedup_{pipelined} = \frac{n * C * f(g)}{L * g + o + n * \frac{C*f(g)}{A}} \tag{4.30}$$

Similarly, increasing the number of sub-computation for the above equation results in:

$$\lim_{n\to\infty} Speedup_{pipelined} \approx A \tag{4.31}$$

The above equations highlight the significance of overheads while pipelining the computations.

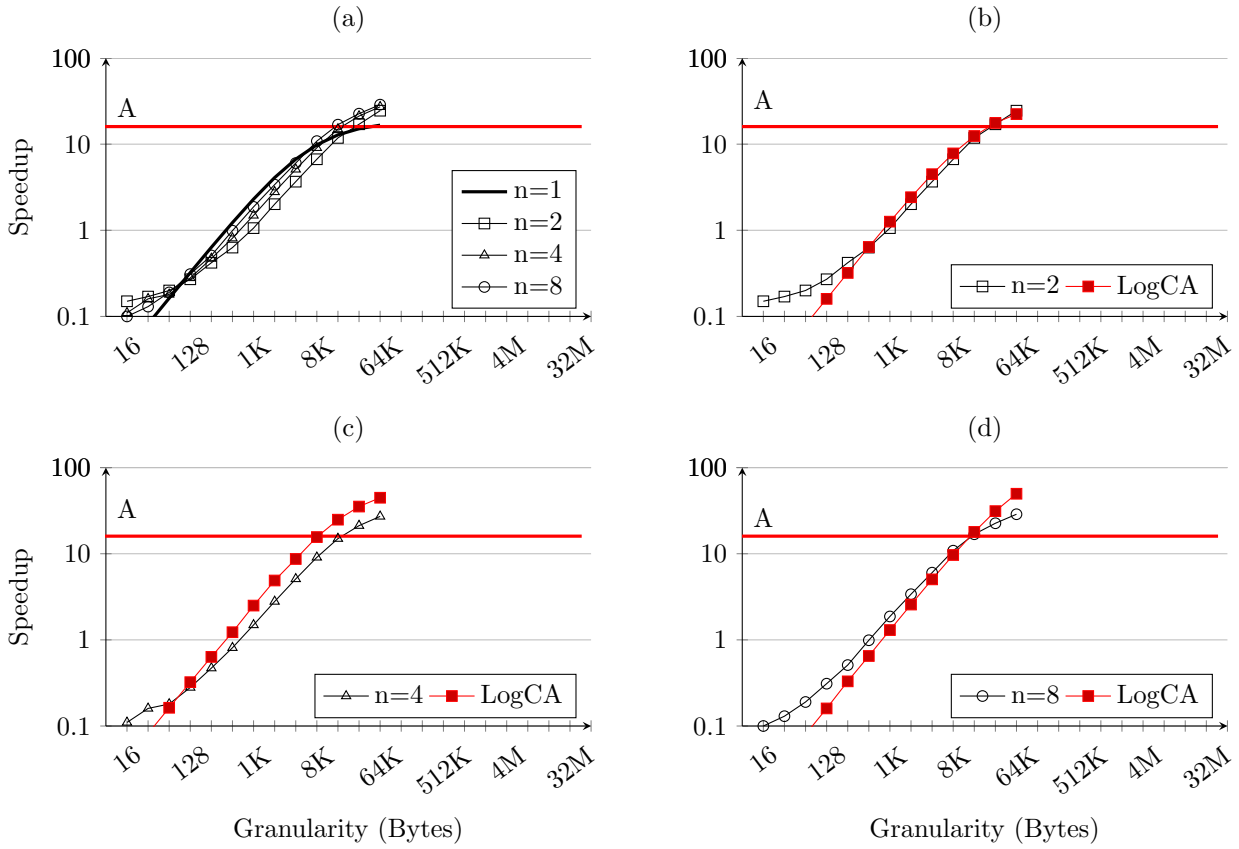


Figure 4.15: Parallel execution of AES. (a) Comparing execution on multiple accelerators with a single accelerator. Curve fitting of LogCA with the observed values while executing on (b) two (c) four and (d) eight accelerators.

4.5.3 Results

In this section, we discuss our results for parallel and pipelined computations.

Figure 4.15 illustrates the effect of running AES on eight cryptographic accelerators on UltraSPARC T2 in parallel. In Figure 4.15-a, we compare the parallel computations with the base case—on a single accelerator, whereas for (b), (c) and (d), we plot the curve-fittings for running on two, four and eight cryptographic accelerators, respectively. We also list the speedup for the base case and multiple accelerators in Table 4.4.

Granularity	**Speedup**			
(Bytes)	**n=1**	**n=2**	**n=4**	**n=8**
16	0.04	0.15	0.11	0.1
32	0.08	0.17	0.16	0.13
64	0.16	0.2	0.18	0.19
128	0.33	0.27	0.28	0.31
256	0.63	0.42	0.47	0.51
512	1.22	0.63	0.81	0.99
1K	2.30	1.06	1.49	1.87
2K	4.10	2.01	2.79	3.39
4K	6.70	3.67	5.10	6.03
8K	9.8	6.68	9.08	10.8
16K	12.8	11.8	15.0	16.8
32K	15.1	17.0	21.2	22.6
64K	16.6	24.4	27.1	28.8

Table 4.4: Parallel execution of AES for various granularities. 'n' represents the number of accelerators running in parallel.

Granularity	**Speedup**		
(Bytes)	**n=1**	**n=2**	**n=4**
16K	0.01	0.01	0.01
64K	0.08	0.08	0.08
256K	.89	0.9	.93
1M	8.18	8.30	8.60
4M	15.5	18.5	20.2
16M	17.6	21.3	22.6
64M	18.5	23.5	23.6

Table 4.5: Pipelined execution of matrix multiplication on a discrete GPU. 'n' represents the number of pipeline stages.

Figure 4.15-a shows that the gains from parallelization vary with granularity. For very small granularities, the speedup decreases with more parallelization. This happens because as we noted in equation (4.21) that for smaller granularities, overheads are higher than the computations time. However, for higher granularities, the speedup increases with increasing parallelization.

Granularity	**Speedup**		
(Bytes)	**n=1**	**n=2**	**n=4**
16K	0.01	0.01	0.01
64K	0.04	0.04	0.04
256K	0.16	0.16	0.16
1M	0.68	0.69	0.72
4M	2.26	2.50	2.60
16M	5.68	7.20	8.40
64M	9.18	15.6	17.4

Table 4.6: Pipelined execution of FFT on discrete GPU. 'n' represents the number of pipeline stages.

Granularity	**Speedup**		
(Bytes)	**n=1**	**n=2**	**n=4**
4K	0.30	0.28	0.25
8K	0.30	0.28	0.27
16K	0.47	0.45	0.35
32K	0.57	0.46	0.41
64K	0.49	0.47	0.43
128K	0.70	0.65	0.60
256K	0.99	0.95	0.85
512K	1.66	1.70	1.90
1M	2.79	3.00	4.20
2M	4.21	5.20	7.80
4M	8.97	9.80	10.1
8M	11.0	12.0	13.3
16M	11.6	13.1	14.2
32M	12.8	14.4	15.4
64M	13.1	15.6	16.8

Table 4.7: Pipelined execution of Radix Sort on a discrete GPU. 'n' represents the number of pipeline stages.

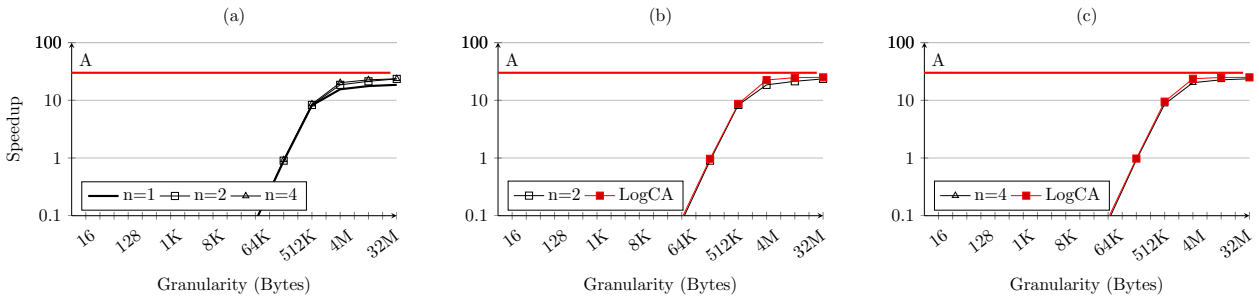


Figure 4.16: Pipelined execution of matrix multiplication on a discrete GPU. (a) Comparing pipelined and un-pipelined execution. Curve fitting of LogCA with the observed values for pipelining (a) two and (b) four computations.

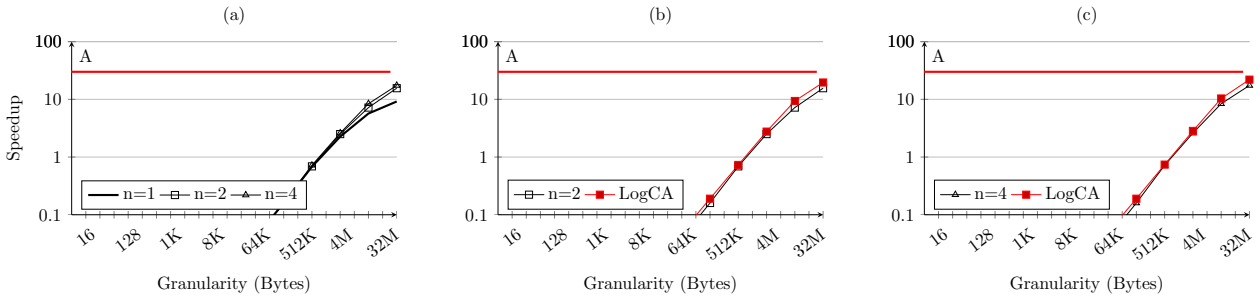


Figure 4.17: Pipelined execution of FFT on a discrete GPU. (a) Comparing pipelined and un-pipelined execution. Curve fitting of LogCA with the observed values for pipelining (a) two and (b) four computations.

Figure 4.16- 4.19 illustrate the effect of pipelining on different kernels running on a discrete GPU. Each of these figures has three different plots. In each figure, (a) plots the observed speedups for un-pipelined and pipelined computations. Subsequently, (b) and (c) plot the curve fitting plots for the pipelined execution for two and four computations, respectively. For a better understanding, we also list the actual speedup numbers for each algorithm in Tables 4.5- 4.7.

These figures provide two key observations. First, LogCA closely follows the observed values for kernels with regular memory access pattern, i.e., matrix multiplication and FFT, whereas it over-predicts for kernels with irregular memory access patterns, i.e., radix

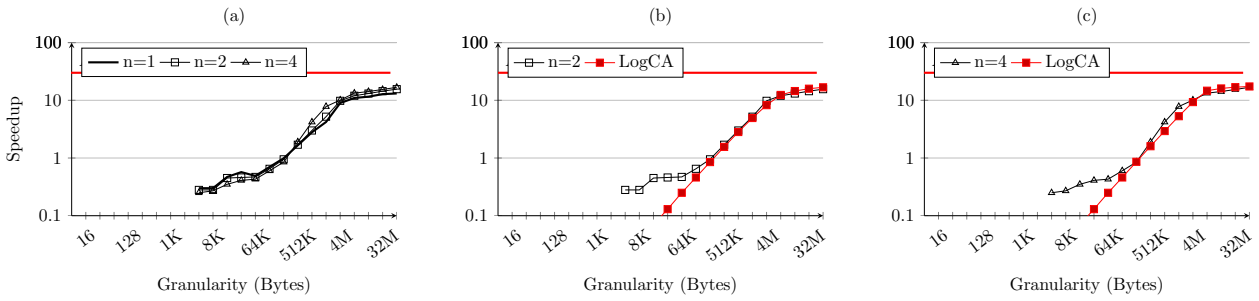


Figure 4.18: Pipelined execution of radix sort on a discrete GPU. (a) Comparing pipelined and un-pipelined execution. Curve fitting of LogCA with the observed values for pipelining (a) two and (b) four computations.

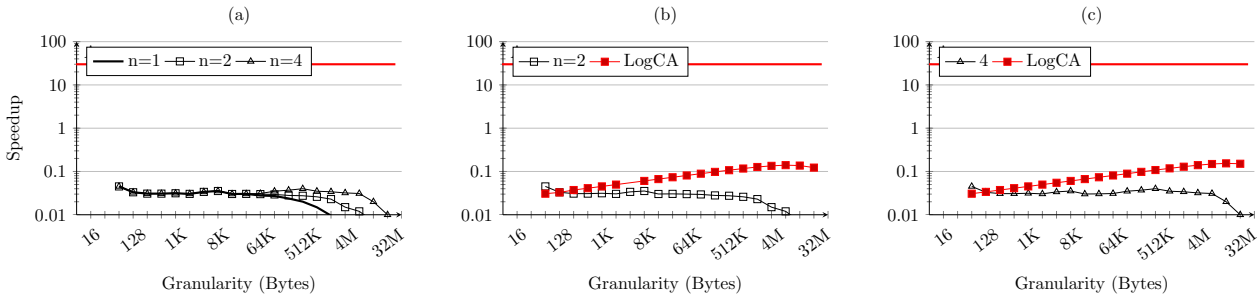


Figure 4.19: Pipelined execution of binary search on a discrete GPU. (a) Comparing pipelined and un-pipelined execution. Curve fitting of LogCA with the observed values for pipelining (a) two and (b) four computations.

sort and binary search. And second, pipelining barely improves speedup by overlapping computation with communication. It does not provide any benefits for lower granularities when overheads are high. With the increase in granularity, when the overheads are amortized, speedup start improving. This observation corroborates equation (4.27) that speedup is limited by overheads at smaller granularities. We also observe that pipelining does not reduce g_1 . In contrast to this observation, Lustig and Martonosi [97] showed that pipelining can reduce the break-even point by fine-grained synchronization between the host and GPU.

4.6 How to Use LogCA?

In this section we describe how programmers and architects can use LogCA, and make decisions early in the design stage. We delineate the process with flowcharts for programmers and architects in Figure 4.20 and 4.21, respectively.

4.6.1 For Programmers

As mentioned earlier, LogCA helps programmers in deciding whether a kernel is worth offloading or not. To this end, Figure 4.20 describes the various steps in the decision process. In the first step, we profile the CPU code and determine the complexity of the algorithm by measuring its β. We have chosen a threshold of 0.9 for the β of the kernel, this threshold roughly signifies a linear complexity kernel.

After profiling, determine whether the latency parameter of the system is granularity independent or dependent. For granularity-dependent latencies, do not offload kernels with sub-linear complexities, whereas for kernels with at-least linear complexity, compute the LogCA parameters. In contrast, for granularity independent latencies, compute the LogCA parameters irrespective of the complexity of the kernel.

After computing the LogCA parameters, determine the speedup and performance metrics, g_1 and $g_{\frac{A}{2}}$. If g_1 turns out to be negative, do not offload the kernel as a negative g_1 shows that the break-even point can not achieved with the current system. However, if g_1 is positive, proceed to the next step and calculate speedup.

If the calculated speedup is greater than the desired speedup, we are done. However, if it is less than the desired speedup, use insights from equation (4.5) and (4.11) that speedup can be increased by offloading more data. Therefore, unless there are design or algorithmic constraints, offload more data to increase the speedup. If increasing granularity achieves

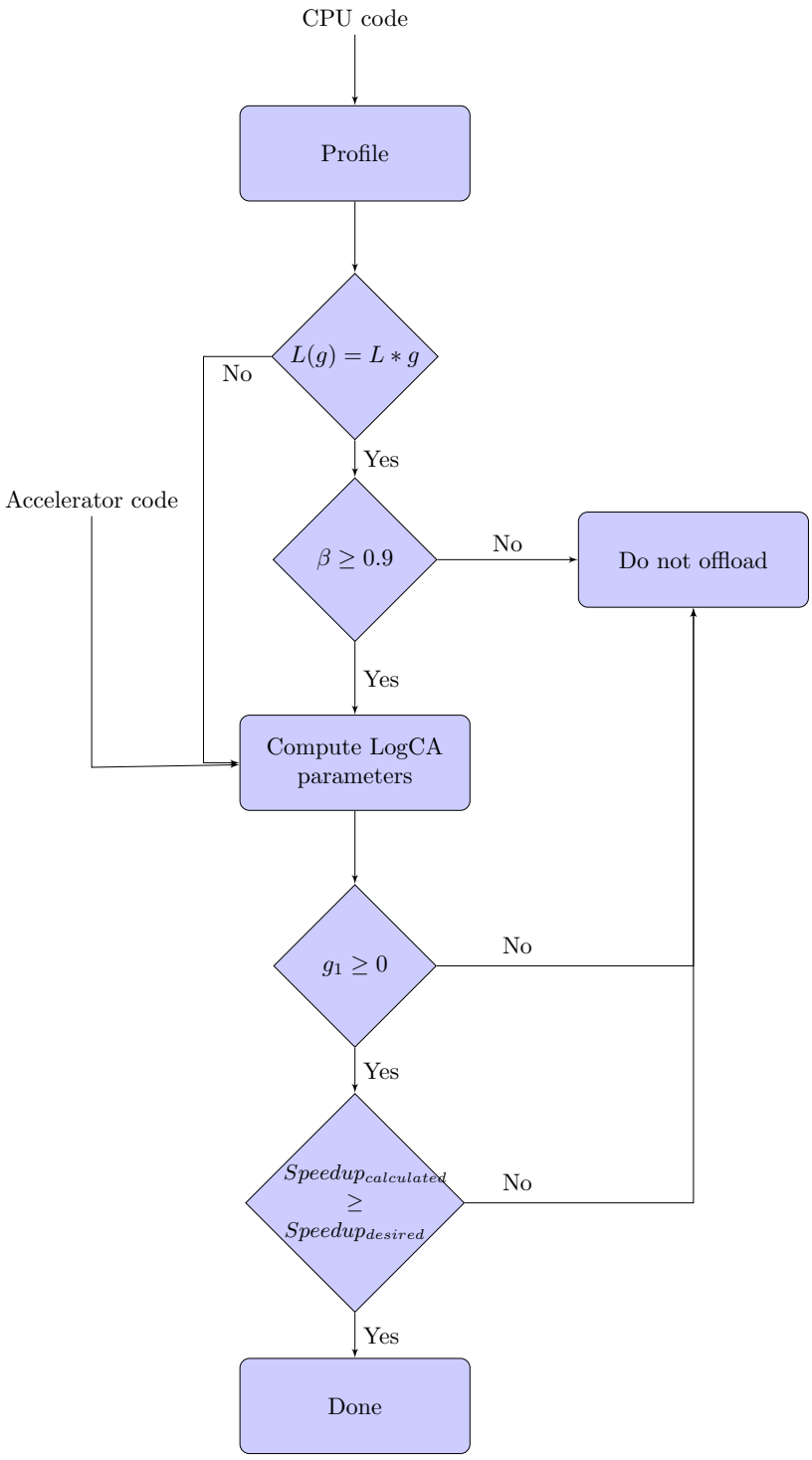


Figure 4.20: A flowchart for programmers for using LogCA early in the design stage.

the desired speedup, we are done. Otherwise, do not offload.

4.6.2 For Architects

For architects, LogCA provides insights about allocating resources in the right direction. The flow chart in Figure 4.21 illustrates the various steps which help architects make a decision.

In the first step, determine the LogCA parameters and g_1 by profiling the CPU and accelerator's code. If g_1 is negative, it shows that the interface needs improvement. As we have noted earlier in implication (1) that allocating resources to accelerator will not affect g_1 . However, if g_1 is positive, proceed to the next step and calculate the speedup. If the calculated speedup is greater than the desired speedup, we are done.

However, if the calculated speedup is less than the desired speedup, calculate $g_{\frac{A}{2}}$. If $g_{\frac{A}{2}}$ is negative, the interface needs improvement. On the other hand, if $g_{\frac{A}{2}}$ is positive, resources should be spent on the accelerator.

4.7 Summary

With the recent trend towards heterogeneous computing, we feel that the architecture community lacks a model to reason about the need for accelerators. To this end, we have proposed LogCA—an insightful performance model for hardware accelerators. LogCA provides insights, early in the design stage, to both architects and programmers by abstracting away low-level details using five system parameters. These parameters signify the latency (L), setup overheads (o), granularity (g) of the data, computational complexity (C) of the kernel and the peak speedup of an accelerator (A).

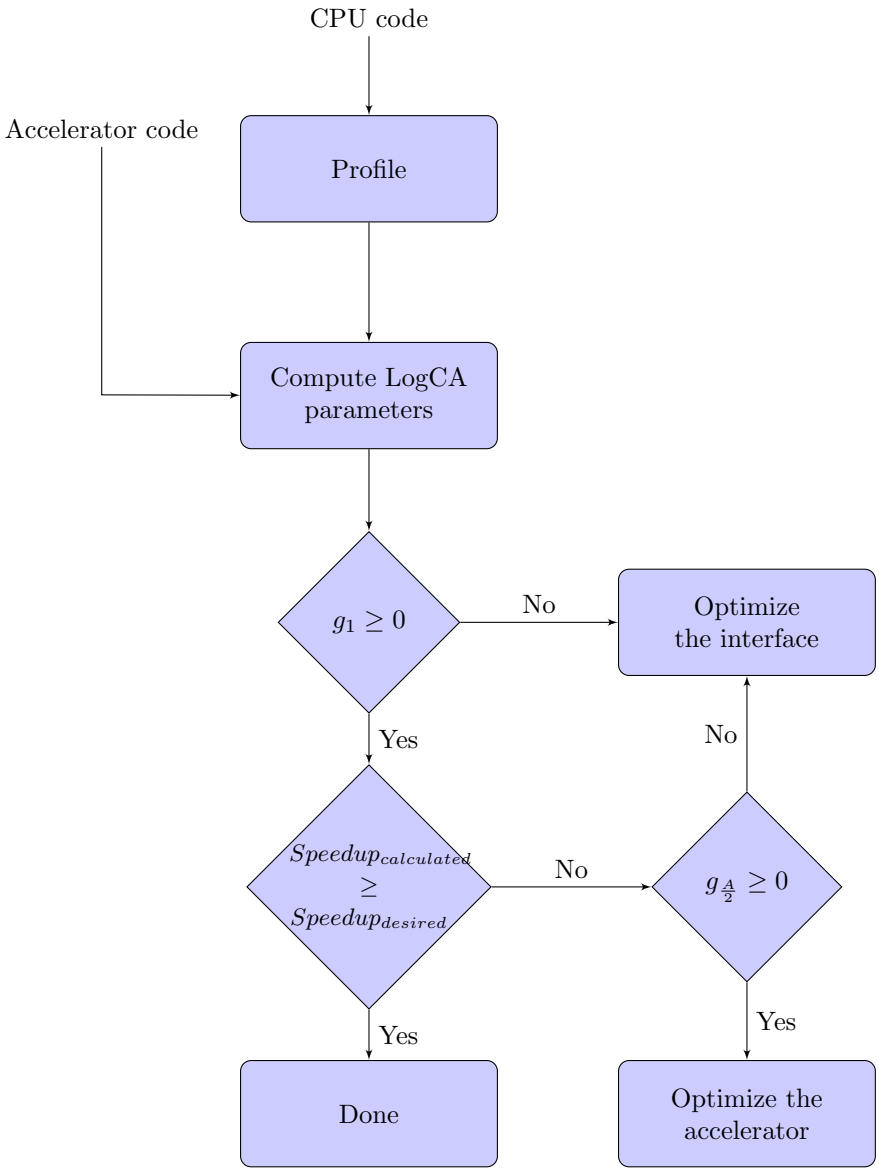


Figure 4.21: A flowchart for architects for using LogCA early in the design stage.

LogCA leverages power-law for modeling the complexity of algorithms. This helps in modeling a diverse range of kernels, ranging from sub-linear to super-linear complexities. We also observe that the power-law helps in modeling logarithmic function within the granularity range of our interest. LogCA exposes the relationship between speedup and the offloaded data, and defines performance metrics which help programmers in deciding about the right amount of data to offload. For architects, LogCA provides insight whether resources should be spent on either the interface or the accelerator.

We have validated LogCA across a diverse range of kernels, on both on-chip and off-chip accelerators. We have shown that LogCA accurately predicts the performance for kernels with regular memory access pattern. However, for irregular access patterns, it mispredicts for either lower or higher granularities.

We have also explored the potential impact of pipelining and parallelizing computations. We have observed that exploiting maximum benefits out of these techniques require careful partitioning of the work among the accelerators. We have also discussed the bottlenecks resulting from an equal distribution of work in parallel computations, and from pipelining computations with different algorithmic complexities.

5 LOGCA^{E}: THE ENERGY COUNTERPART FOR THE LOGCA MODEL

Power is a design constraint not only for portable computers and mobile communication devices but also for high-end systems, and the design process should not subordinate it to performance.

— Trevor Mudge

5.1 Introduction

With the recent trend in energy-efficient systems, performance is no longer the primary design criterion rather energy has become a major design concern [94]. Therefore, while designing accelerators, architects need to make sure that the performance gains provided by an accelerator should not be accompanied by excessive energy consumption. Likewise, programmers need to ascertain, while offloading a kernel, the amount of data to offload which can provide both performance and energy benefits.

To illustrate our point, Figure 5.1 plots the execution time and energy consumption for the AES kernel on a discrete GPU. We observe that the unaccelerated execution time is smaller than the accelerated one for smaller granularities, whereas it is higher than accelerated execution time for large granularities. We also observe that the energy follows a similar trend as execution time, i.e., the energy consumption for the accelerated kernel is higher than the unaccelerated one for lower granularities and smaller than the unaccelerated one for higher granularities. However, the break-even points are different for these two

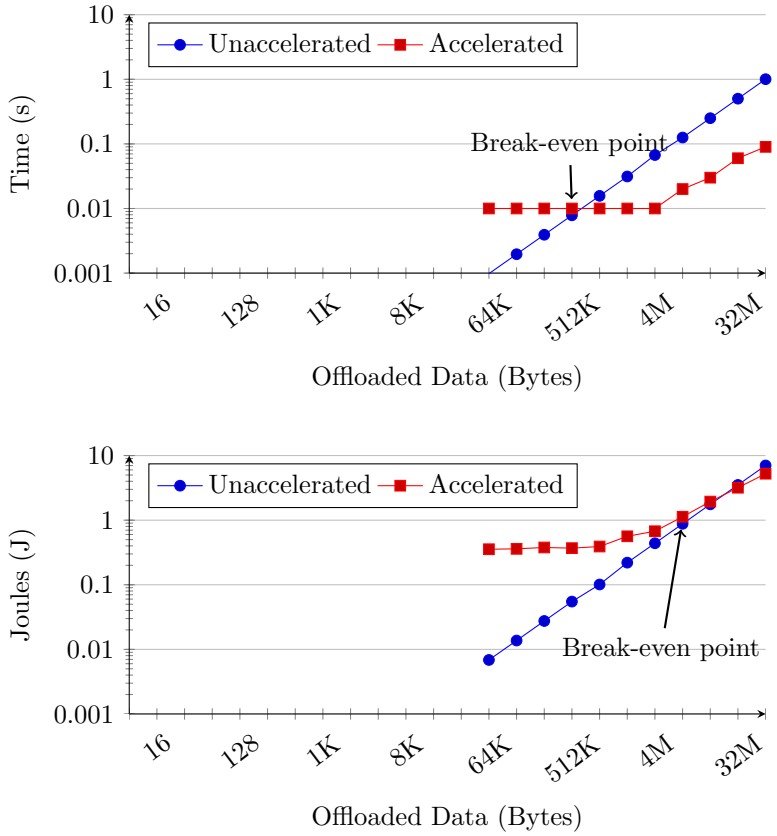


Figure 5.1: Execution time (above) and energy (below) for running Advanced Encryption Standard (AES) kernel on a discrete GPU.

cases, i.e., the break-even point for energy consumption is roughly 32 times higher than for execution time.

This observation highlights the possibility of an offloaded kernel to be performance but not energy efficient. To help programmers and architects determine these scenarios early in the design cycle, we have developed an energy model similar to the performance model described earlier in Chapter 4.

The rest of the chapter is organized as follows. We develop the energy model in Section 5.2, and define speedup-efficiency product in Section 5.3. We discuss the results in

Parameter	Symbol	Description	Units
Link Energy	L^e	Energy to move data from the host to the accelerator across an un-pipelined interface, including the cycles data spends in the caches or memory	Joules
Overhead Energy	o^e	Energy the host spends in setting up the algorithm	Joules
Granularity	g	Size of the offloaded data	Bytes
Computational Energy	C^e	Energy the host spends on computation per byte of data	Joules/Byte
Energy Acceleration	A^e	The peak energy efficiency of an accelerator	N/A

Table 5.1: Description of the energy counterpart of LogCA model.

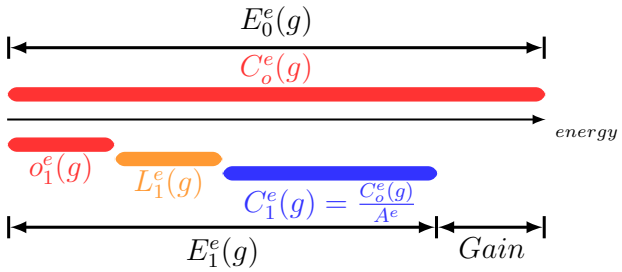


Figure 5.2: Energy consumption for the computation performed on the host (above) and on an accelerator (below)

Section 5.4 and finally, conclude with Section 5.5.

5.2 Energy Model

In this section, we develop the LogCA energy model and compare and contrast it with the LogCA performance model. We note that the energy model is similar to the performance model, and they differ only in the assumption of latency[1]. Therefore, for consistency with

[1]We describe the latency assumption at the end of this section

the performance model, we use the same five symbols—each with an 'e' superscript, except for 'g'—to denote the corresponding energy parameters. Table 5.1 provides a description of each parameter.

Figure 5.2 shows the breakdown of energy consumed by an algorithm on the host and accelerator. We assume that the host is idle after offloading the algorithm and we do not factor in the energy consumed during that time. We also assume that the algorithm's execution energy is a function of granularity, i.e., the size of the offloaded data. With this assumption, the unaccelerated energy E_0 (energy with zero accelerators) to process data of *granularity* g, will be

$$E_0(g) = C_0^e(g) \tag{5.1}$$

where $C_0^e(g)$ is the computation energy on the host.

When the data is offloaded to an accelerator, the new execution energy E_1 (energy with one accelerator) is

$$E_1(g) = O_1^e(g) + L_1^e(g) + C_1^e(g) \tag{5.2}$$

where $O_1^e(g)$ is the host overhead energy in offloading 'g' bytes of data to the accelerator, $L_1^e(g)$ is the link or communication energy and $C_1^e(g)$ is the computation energy in the accelerator to process data of granularity 'g'.

We assume that an accelerator with acceleration 'A^e' can decrease, in the absence of overheads, the algorithm's computation energy on the host by a factor of 'A^e', i.e., the accelerator and host use algorithms with the same complexity. Thus, the computation energy on the accelerator will be $C_1^e(g) = \frac{C_0^e(g)}{A^e}$. This reduction in the computation energy results in energy gains, and we quantify these gains with efficiency, the ratio of the un-

accelerated and accelerated energy:

$$Efficiency(g) = \frac{E_0(g)}{E_1(g)} = \frac{C_0^e(g)}{O_1^e(g) + L_1^e(g) + C_1^e(g)} \tag{5.3}$$

We also assume that the computation energy is a function of the computational index 'C^e' and granularity, i.e., $C_0^e(g) = C^e * f(g)$, where $f(g)$ signifies the complexity of the algorithm. We also assume that $f(g)$ is power function of 'g', i.e., $\mathcal{O}(g^\beta)$.

Earlier for the performance model, our analysis is based on two alternative assumptions about latency, i.e., it can be granularity independent or dependent. However for the energy model, the assumption of granularity independent latency is not likely to be valid. Since communicating large blocks will proportionally consume more link energy and overlapping various computations can not save energy. That is why the link energy will always be granularity dependent and for simplicity we assume it to be a linear function of granularity i.e., $L_1^e(g) = L^e * g$. We also assume that energy overheads are granularity independent, i.e., $o_1^e(g) = o^e$. With these assumptions, efficiency will be given by:

$$Efficiency(g) = \frac{C^e * g^\beta}{o^e + L^e * g + \frac{C^e * g^\beta}{A^e}} \tag{5.4}$$

For large granularities, equation (5.4) reduces to:

$$\lim_{g \to \infty} Efficiency(g) = \left(\frac{A^e}{\frac{A^e}{C^e * g^\beta} * (L^e * g) + 1} \right) < \frac{C^e}{L^e} * g^{\beta - 1} \tag{5.5}$$

Since the above equation is similar to equation (4.10) from the performance model, all the results and implications from our analysis in Section 4.2.3 are also applicable for the energy model. Thus, we do not discuss this further in detail and just list the equations for

energy metrics for reference.

$$g_1^e = \frac{C^e * (\beta - 1) * (A^e - 1) + A^e * o^e}{C^e * \beta * (A^e - 1) - A^e * L^e} \tag{5.6}$$

$$g_{\frac{A^e}{2}}^e = \frac{C^e * (\beta - 1) + A^e * o^e}{C^e * \beta - A^e * L^e} \tag{5.7}$$

The above equations highlight the fact that for achieving any energy efficiency, the computation energy on the host should be greater than the link energy. Similarly, for achieving an energy efficiency of half of the peak value, the computation energy on the accelerator should be greater than the link energy.

5.3 Speedup-Efficiency Product

In the early days of low-power designs, a number of proposed techniques saved power at the cost of low performance. Thus, making it difficult for architects to optimize for both energy and performance. To overcome this problem, Horowitz et al. [95] argued for a single metric *energy-delay* product (EDP). EDP helps in comparing different low-power design techniques by exposing the trade-offs between lower energy and increased delays.

Similarly, we argue that both speedup and efficiency should be considered while making a design decision for accelerators. Therefore, we propose a similar metric, *speedup-efficiency* product (SEP), which takes into account both speedup and efficiency. Mathematically,

$$SEP = \frac{Time_{unaccelerated}}{Time_{accelerated}} * \frac{Energy_{unaccelerated}}{Energy_{accelerated}} = \frac{EDP_{unaccelerated}}{EDP_{accelerated}} \tag{5.8}$$

The above equation shows that *SEP* is the ratio of unaccelerated and accelerated EDPs. A

higher (and increasing) SEP signifies an energy-efficient design, whereas a lower SEP implies that the performance gains are not accompanied by reasonable energy improvements, i.e., an energy-inefficient design. Thus, SEP can be helpful in comparing performance and energy trade-offs in accelerator designs similar to EDP.

5.4 Results

In this section, we present our results and show that the energy counterpart of LogCA follows the observed energy efficiency behavior for both off and on-chip accelerators. We also observe that these results closely match the results from our performance analysis in the previous chapter. This is not surprising, as we mentioned earlier in Chapter 3 that we measure energy consumption of a kernel by multiplying the average power by its execution time. Therefore, the energy consumption is a scaled version of the execution time and consequently, the efficiency curves are more or less similar to the speedup curves.

5.4.1 Linear Complexity Kernels

Figure 5.3 shows the curve-fitting of LogCA^e for accelerators connected through different interfaces, i.e., ranging from PCIe bus to special instructions. We observe that the off-chip accelerators and APU, unlike on-chip accelerators, are energy efficient only at very large granularities. For example, g_1^e for GPU and APU are 8MB and 4MB, respectively. This observation shows that the powerful (and power hungry) cores in the discrete GPU make it energy-inefficient for smaller granularities. It also shows that despite being an integrated GPU and not connected to the PCIe bus, APU spends considerable energy in copying data from the host to device memory.

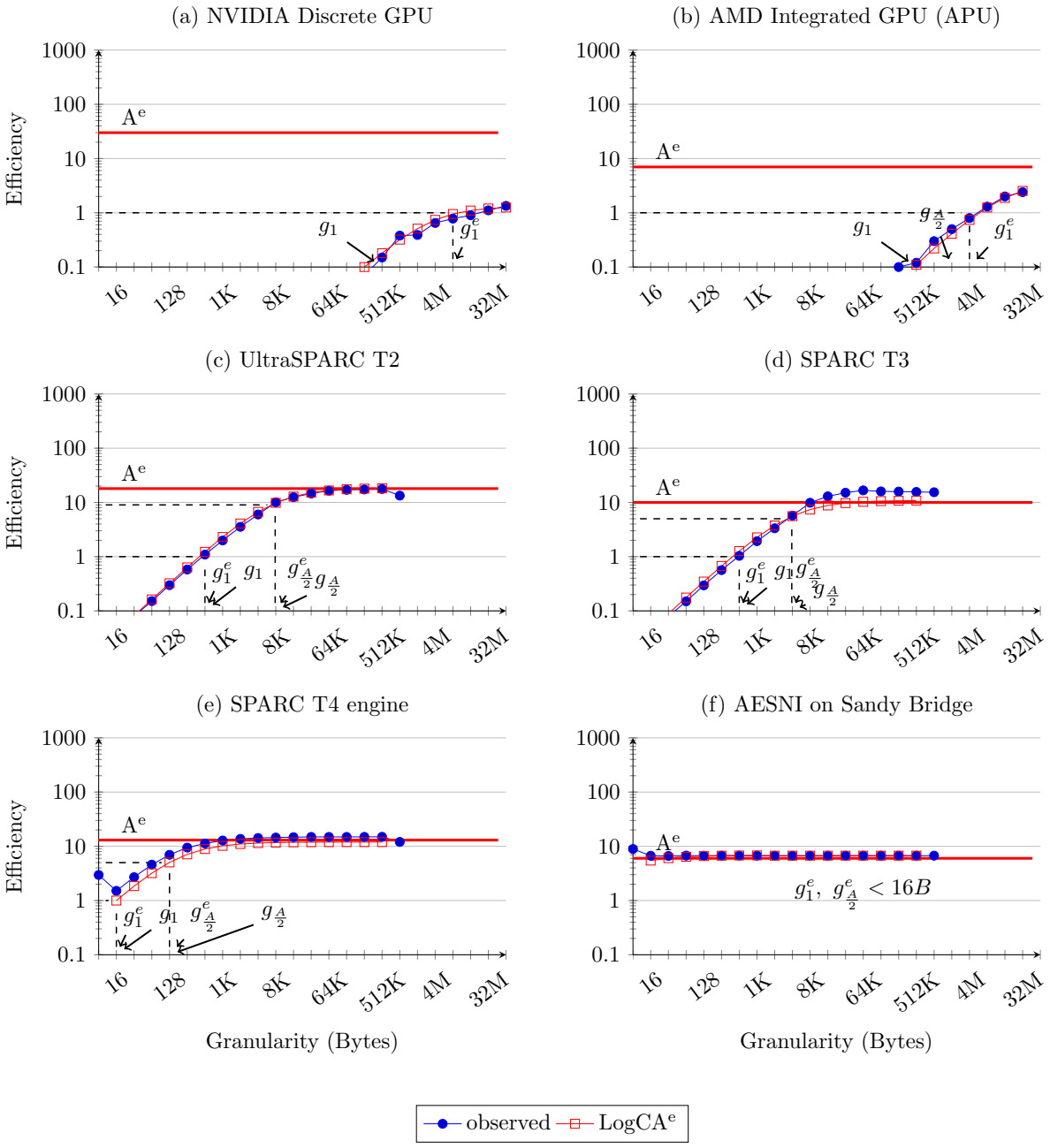


Figure 5.3: Efficiency curve fittings plots comparing LogCA^e with the observed energy of AES [2] over a range of granularities. Performance metrics (g_1 , $g_{\frac{A}{2}}$) are also labelled for a comparison.

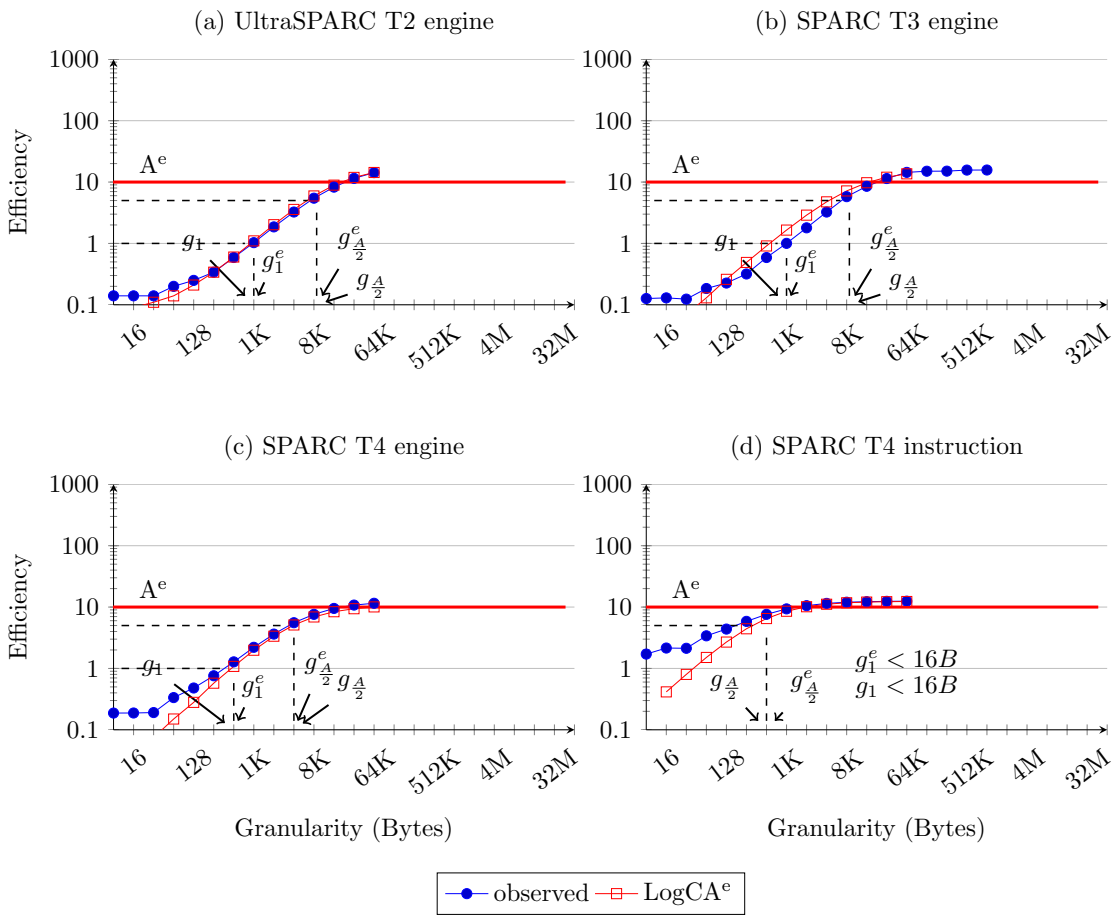


Figure 5.4: Efficiency curve fittings plots comparing LogCA^e with the observed energy of SHA256 [3] over a range of granularities. Performance metrics (g_1 , $g_{\frac{A}{2}}$) are also labelled for a comparison.

Figure 5.3 also shows that UltraSPARC T2, SPARC T3 and T4 provide higher efficiency than Sandy Bridge, but break-even at a higher granularity. Sandy Bridge, on the other hand, breaks-even at very small granularity but provides limited efficiency.

Figure 5.4 shows the curve fitting for SHA on various on-chip cryptographic accelerators. Similar to AES, g_1^e and $g_{\frac{A}{2}}^e$ do exist as all of these are on-chip accelerators. It is also interesting to note that g_1^e for SPARC T4 is slightly lower than g_1 (as shown in Figure 4.5 in Chapter 4),

and signifies an energy-efficient design.

Figure 5.5 shows that LogCA^e does not follows the observed energy efficiency for radix sort for smaller granularities. LogCA^e starts following the observed efficiency after crossing the break-even point. This observation is similar to the performance results of radix sort in the previous chapter. LogCAe is also unable to predict the performance correctly because of the irregular memory access pattern of radix sort.

5.4.2 Super-Linear Complexity Kernels

Figures 5.6 and 5.7 show the speedup curve fitting plots for matrix multiplication and FFT kernels on discrete GPU and APU. We observe that Matrix multiplication (Figure 5.6) with higher complexity achieves higher efficiency as compared to Sort (Figure 5.5) and FFT (Figure 5.7) with relatively lower complexities. This observation supports earlier implication that achievable efficiency (speedup) of higher-complexity algorithms is higher than lower-complexity algorithms.

5.4.3 Sub-Linear Complexity Kernels

We do not present energy efficiency results for the sub-linear complexity kernels as we have observed earlier in the previous chapter (Figure 4.10) that—with granularity dependent latency—kernels with sub-linear complexity do not provide any speedup.

5.4.4 Speedup-Efficiency Product

To show how SEP helps in choosing an energy and performance-efficient design, we plot speedup-efficiency product for AES running on five different accelerators in Figure 5.8. This figure provides some key observations: First, the fixed-function accelerators provide

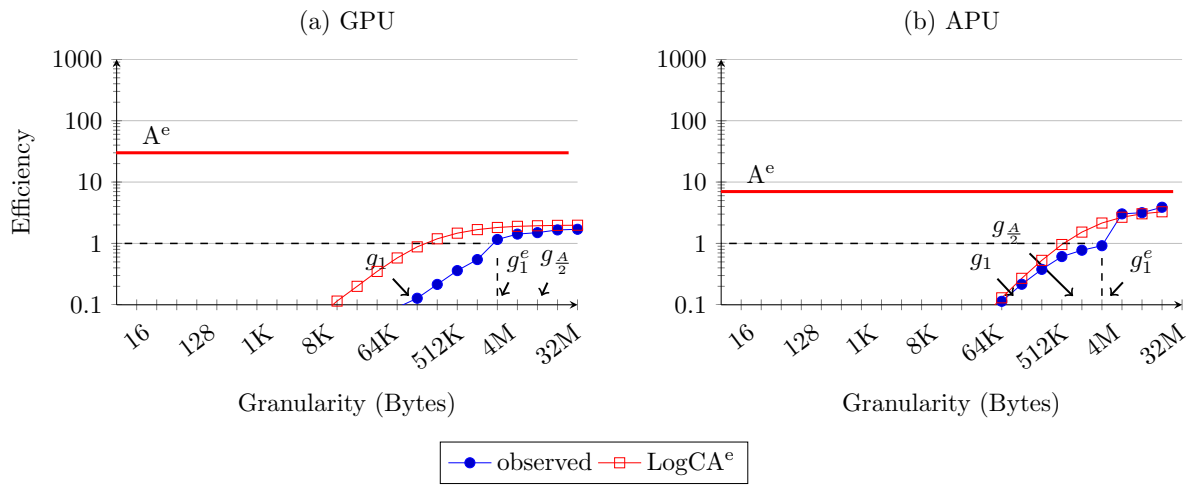


Figure 5.5: Efficiency curve fittings plots comparing LogCA^e with the observed energy of Radix Sort over a range of granularities. Performance metrics (g_1 , $g_{\frac{A}{2}}$) are also labelled for a comparison.

higher SEP than the off-chip accelerators. Thus, representing designs where performance gains are accompanied with comparable energy efficiency.

Second, among the fixed-function accelerators, granularity determines a better energy-efficient design. For example, AESNI instructions on Sandy Bridge provide the highest speedup-efficiency product for small granularities, i.e., till 128B, whereas for larger granularities, i.e., till 16KB, SPARC T4 has higher SEP. We also observe that for granularities larger than 16KB, UltraSPARC T2 is the most energy-efficient solution. This happens because at large granularities, the energy consumption due to overheads is no longer a bottleneck, so UltraSPARC T2 provides comparable results to SPARC T3 and T4.

Third, for the programmable accelerators, we observe that GPU and APU have comparable SEPs. This is surprising because we observe in Figure 5.3 that GPU is less energy-efficient than APU. However, despite low efficiency, GPU provides higher speedup than APU, resulting in similar SEP.

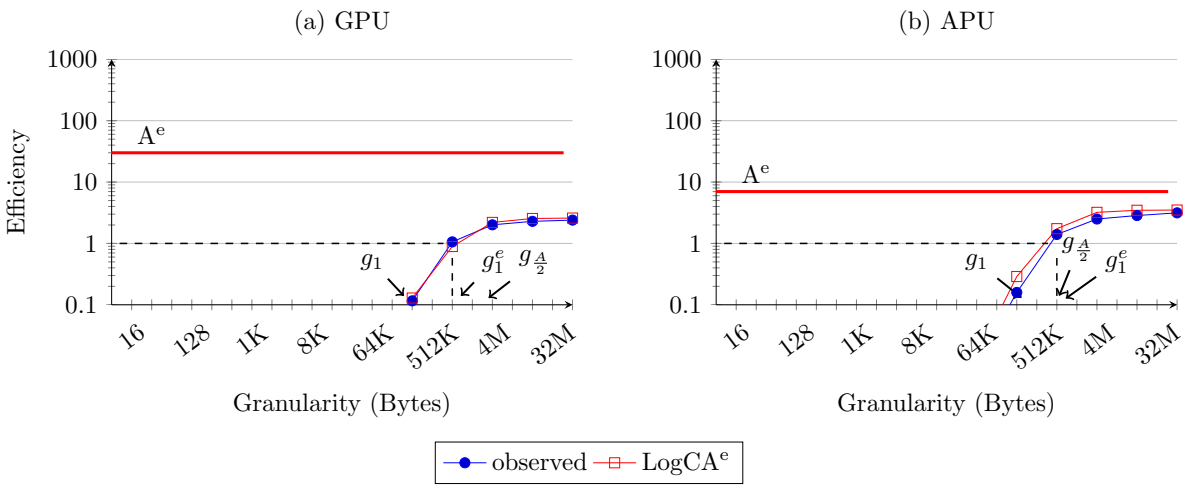


Figure 5.6: Efficiency curve fittings plots comparing LogCA^e with the observed energy of matrix multiplication over a range of granularities. Performance metrics (g_1 , $g_{\frac{A}{2}}$) are also labelled for a comparison.

We observe similar results for running SHA on on-chip accelerators in Figure 5.9, i.e., SPARC T4 with special encryption instructions has the highest SEP and all accelerators provide comparable SEP at very large granularities.

5.5 Summary

With the recent trend towards energy-efficient computing, energy has become an important design criterion. Consequently, architects can no longer make design decisions while focussing only on performance. Therefore, in this chapter, we have developed an energy counterpart—LogCA^e—for the LogCA model. Similar to LogCA, LogCA^e abstracts away low-level details using five key parameters. These parameters signify the link energy (L^e), energy for overheads (o^e), granularity (g) of the data, computational complexity (C^e) of the kernel and the peak energy efficiency of an accelerator (A^e).

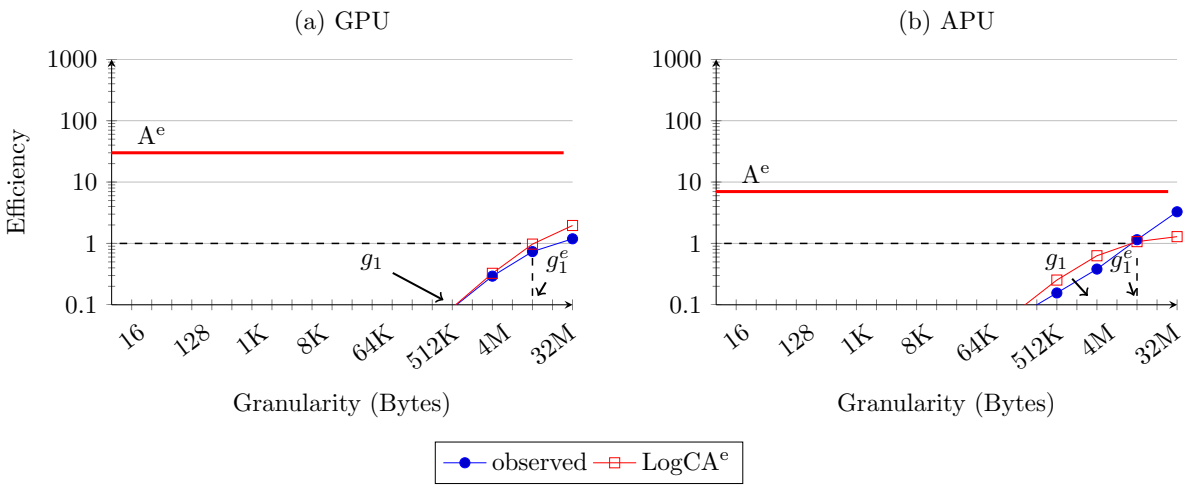


Figure 5.7: Efficiency curve fittings plots comparing LogCA^e with the observed energy of FFT over a range of granularities. Performance metric (g_1) is also labelled for a comparison.

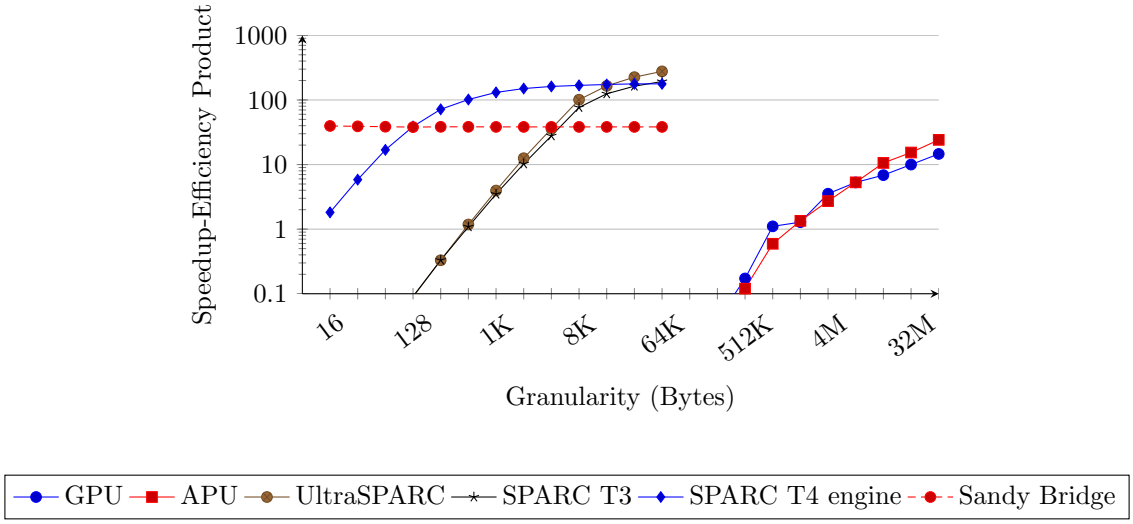


Figure 5.8: Speedup-Efficiency Product (SEP) plot for AES on various accelerators.

For evaluating trade-offs between performance and energy efficient design, we have also defined a metric—speedup-efficiency product. We have observed that the speedup-efficiency product of the fixed-function accelerators is higher than the programmable accelerators. We have also observed that for discrete GPUs, with powerful processing cores,

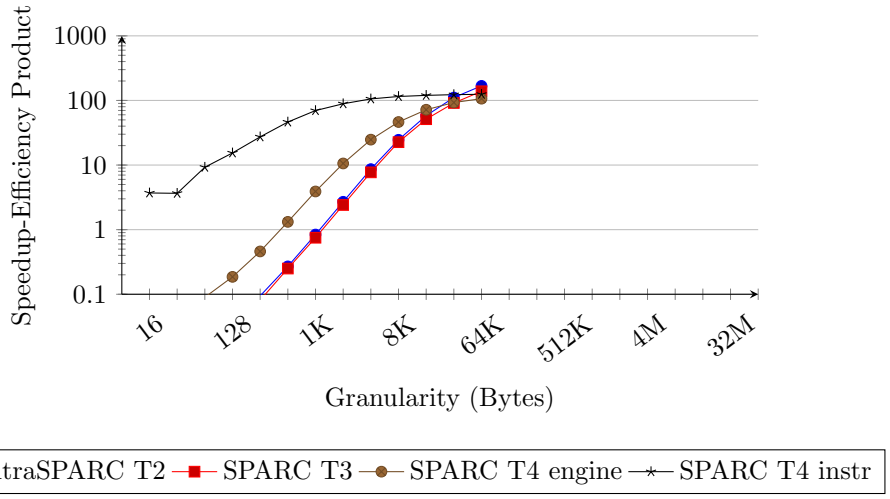


Figure 5.9: Speedup-Efficiency Product (SEP) plot for SHA on various accelerators.

lower energy efficiency results in lower speedup-efficiency product.

We have validated our model across a diverse range of kernels on both on-chip and off-chip accelerators. However, similar to LogCA, the accuracy of the energy model is limited by its inability to model caches.

6 ACCELEROMETER: A SENSITIVITY ANALYSIS OF THE LOGCA PARAMETERS

6.1 Introduction

LogCA, presented earlier in Chapter 4, provides sufficient simplicity for programmers and architects to easily reason about accelerators, yet it does not provide many insights on the design bottlenecks and how to mitigate these bottlenecks. For example, consider three accelerators, SPARC T4, UltraSPARC T2 and a discrete GPU, running an encryption kernel in Figure 6.1. These accelerators have different break-even points and speedup characteristics.

For the given example, LogCA helps programmers in determining the minimum granularity to offload and architects in choosing an accelerator design based on the expected operating granularity. It also helps in capturing the effect of granularity on the speedup for a given design and deciding whether a proposed design meets the desired speedup or

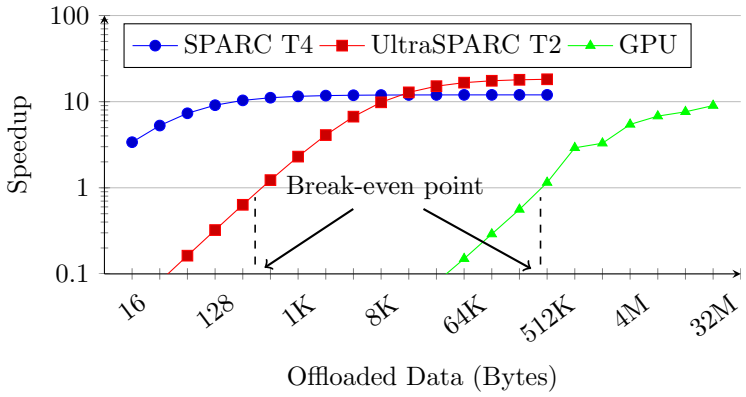


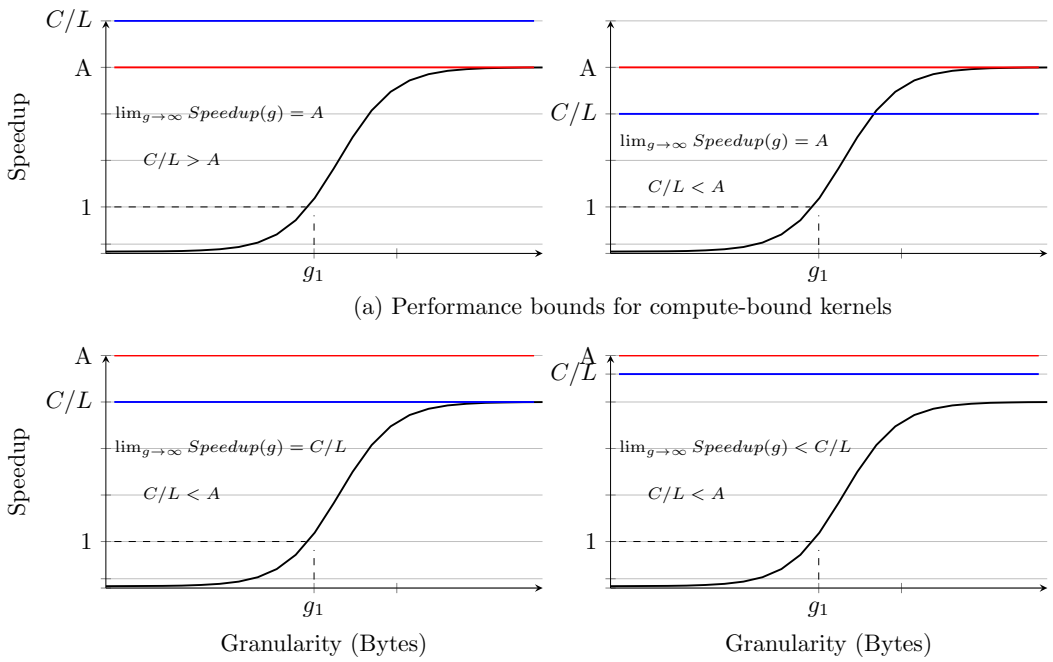
Figure 6.1: Running Advanced Encryption Standard (AES) kernel on three different cryptographic accelerators.

not. However, if the desired speedup is not achievable, LogCA does not help in exposing the design bottlenecks and how to alleviate them.

Considering the AES encryption example in Figure 6.1, programmers and architects would greatly benefit from understanding: what bottlenecks cause UltraSPARC T2 and GPU to under-perform for small data sizes? Which optimizations on UltraSPARC T2 and GPU would result in similar performance to SPARC T4? Which optimizations are programmer-dependent and which are architecture-dependent? What are the trade-offs in selecting one optimization over the other?

To answer these questions, we take inspiration from the *Roofline* model [1] and extend LogCA to develop *Accelerometer* in this chapter. Accelerometer is a tool which exposes the design bottlenecks using the sensitivity analysis of the LogCA parameters. Accelerometer provides insights about the accelerator's interface by exposing the design bounds and bottlenecks. It also suggests optimization—using visually identifiable regions—to alleviate these bottlenecks. These visually identifiable optimization regions also help programmers and architects to qualitatively assess the trade-offs among various optimizations.

The rest of the chapter is organized as follows. In Section 6.2, we develop accelerometer and describe the performance bounds, sensitivity analysis and totem plots. Section 6.3 describes the retrospective case studies, and we discuss parallelizing and pipelining computations in Section 4.5. Then, we discuss how programmers and architects can use accelerometer in Section 6.4. We conclude this chapter by summarizing the key points in Section 6.5.



(a) Performance bounds for compute-bound kernels

(b) Performance bounds for latency-bound kernels

Figure 6.2: Accelerometer helps in visually identifying (a) compute and (b) latency bound kernels.

6.2 Accelerometer

In this section, we describe Accelerometer and show how it leverages LogCA for visually identifying the performance bounds. We also show the efficacy of the sensitivity analysis of the LogCA parameters in exposing design bottlenecks and suggesting possible optimizations to alleviate them.

6.2.1 Performance Bounds

We have observed earlier in Chapter 4 that for granularity-independent latency speedup is bounded by acceleration, i.e.,

$$\lim_{g \to \infty} Speedup(g) = A \tag{6.1}$$

whereas for granularity-dependent latency, speedup is bounded by the product of computational intensity and $g^{\beta-1}$

$$\lim_{g \to \infty} Speedup(g) = \frac{C}{L} * g^{\beta-1} \tag{6.2}$$

Depending on the complexity of the kernel, the above equation can be further expanded as:

$$\lim_{g \to \infty} Speedup(g) = \begin{cases} < \frac{C}{L} & \text{if } \beta < 0 \\ = \frac{C}{L} & \text{if } \beta = 1 \\ > \frac{C}{L} & \text{if } \beta > 1 \end{cases} \tag{6.3}$$

Using these observations, we classify kernels either as compute-bound or latency-bound. For compute-bound kernels, the achievable speedup is bounded by acceleration, whereas for the latency-bound kernels, the speedup is bounded by computational intensity. Based on this classification, a compute-bound kernel can either be running on a system with granularity-independent latency or has super-linear complexity while running on a system with granularity-dependent latency. Figure 6.2-a illustrates these bounds for compute-bound kernels.

On the other hand, a latency-bound kernel is running on a system with granularity-dependent latency and has either linear or sub-linear complexity. Figure 6.2-b illustrates

these bounds for compute-bound kernels.

Programmers and architects can visually identify these bounds and use this information to invest their time and resources in the right direction. For example, for compute-bound kernels—depending on the operating granularity—it may be beneficial to invest more resources in either increasing acceleration or reducing overheads. However, for latency-bound kernels, optimizing acceleration and overheads is not that critical, but decreasing latency and increasing computational index maybe more beneficial.

6.2.2 Sensitivity Analysis

To identify the design bottlenecks, we perform a sensitivity analysis of the LogCA parameters. We consider a parameter a design bottleneck if a 10x improvement in it provides at least 20% improvement in speedup. A 'bottlenecked' parameter also provides an optimization opportunity. To visually identify these bottlenecks, we introduce optimization regions. As an example, we identify design bottlenecks in UltraSPARC T2's cryptographic accelerator by varying its individual parameters in Figure 6.3 (a)-(d).

Figure 6.3 (a) shows the variation (or the lack of) in speedup with the decrease in latency. The resulting gains are negligible and independent of the granularity, as it is a closely coupled accelerator.

Figure 6.3 (b) shows the resulting speedup after reducing overheads. Since the overheads are one-time initialization cost and independent of granularity, the per byte setup cost is high at small granularities. Decreasing these overheads considerably reduces the per byte setup cost and results in significant gains at these smaller granularities. Conversely, for larger granularities, the per byte setup cost is already amortized, so reducing overheads does not provide much gains. Thus, overhead is a bottleneck at small granularities and

provides an opportunity for optimization at these granularities.

Figure 6.3 (c) shows the effect of increasing the computational index. The results are similar to optimizing overheads in Figure 6.3 (b), i.e., significant gains for small granularities, and a gradual decrease in the gains with increasing granularity. With the constant overheads, increasing computational index increases the computation time of the kernel and decreases the per byte setup cost. For smaller granularities, the reduced per byte setup cost results in significant gains.

Figure 6.3 (d) shows the variation in speedup with increasing acceleration. The gains are negligible at small granularities and become significant for large granularities. As mentioned earlier, the per byte setup cost is high at small granularities and it reduces for large granularities. Since increasing acceleration does not reduce the per byte setup cost, optimizing acceleration provides gains only at large granularities.

We group these individual sensitivity plots in Figure 6.4 to build the optimization regions. As mentioned earlier, each region indicates the potential of 20% gains with 10x variation of one or more LogCA parameters. For the ease of understanding, we color these regions and label them with their respective LogCA parameters. For example, the blue colored region labelled 'oC' (16B to 2KB) indicates an optimization region where optimizing overheads and computational index is beneficial. Similarly, the red colored region labelled 'A' (32KB to 32MB) represents an optimization region where optimizing acceleration is only beneficial. The granularity range occupied by a parameter also identifies the scope of optimization for an architect and a programmer. For example, for UltraSPARC T2, the overhead parameter occupies most of the lower granularity region, suggesting an opportunity for architects to improve the interface. Similarly, the absence of the latency parameter suggests little benefit for optimizing latency. These regions can also help programmers

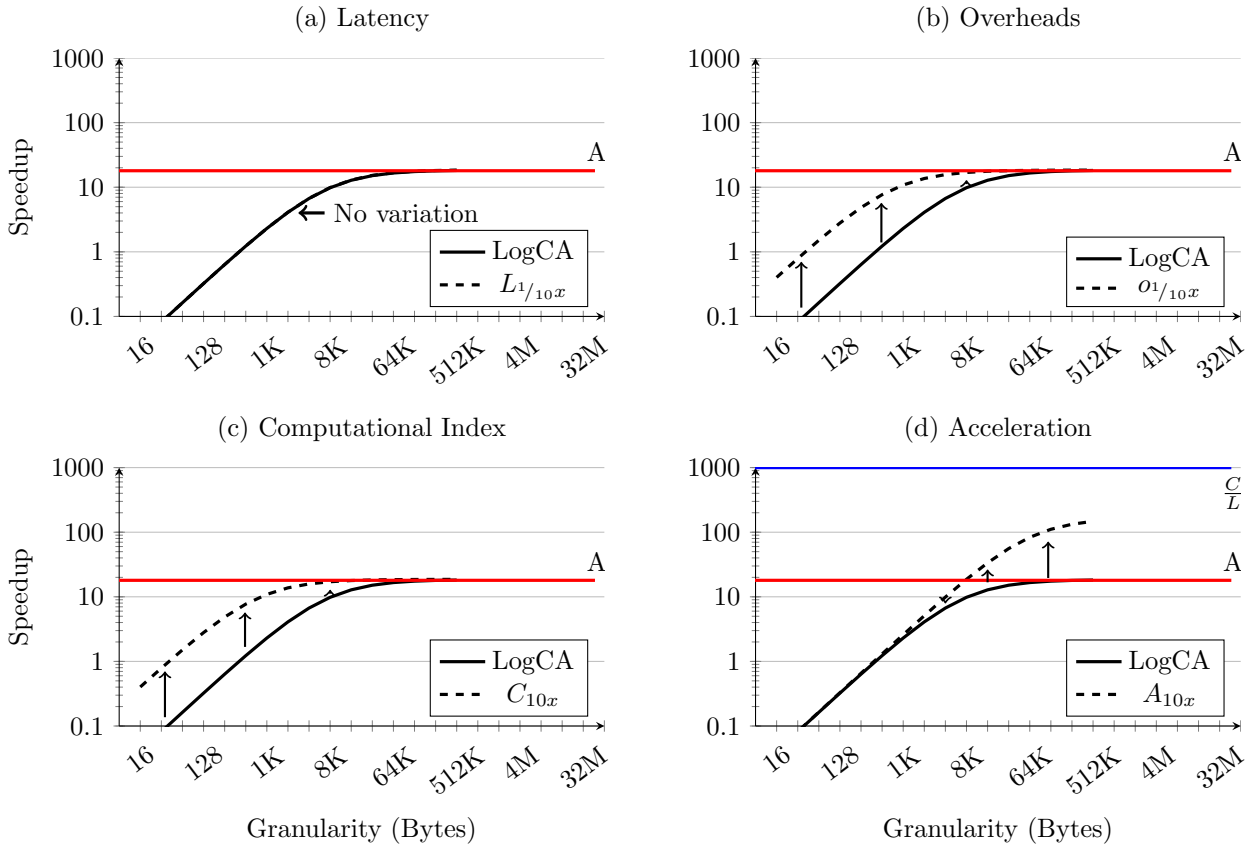


Figure 6.3: The effect on speedup of 10x improvement in each LogCA parameter. The base case is the speedup of AES [2] on UltraSPARC T2.

and architects in prioritizing optimizations.

We also add horizontal arrows to the optimization regions in Figure 6.4 to demarcate the start and end of granularity range for each parameter. For example, optimizing acceleration starts providing benefits from 2KB while optimizing overheads or computational index is beneficial up till 32KB. These arrows also indicate the cut-off granularity for each parameter. These cut-off granularities provide insights to architects and programmers about the design bottlenecks. For example, the high cut-off granularity of 32KB suggests high overheads and thus a potential for optimization.

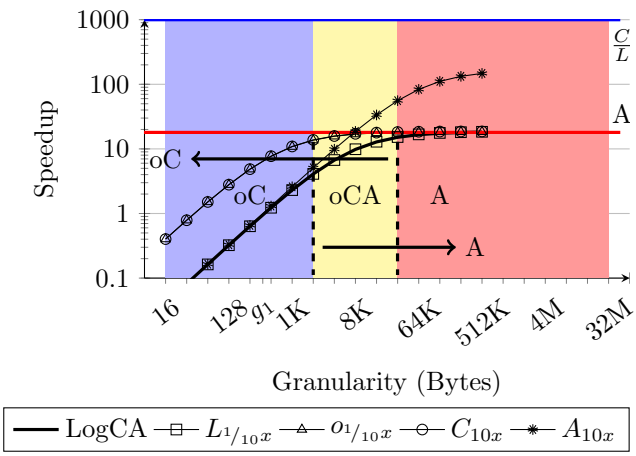


Figure 6.4: Optimization regions for UltraSPARC T2. The presence of a parameter in an optimization region indicates that it can at least provides 20% gains. The horizontal arrow indicates the cut-off granularity at which a parameter provides 20% gains.

Figure 6.4 shows that the achievable performance of running an AES kernel is 16. This performance can only be increased by replacing the current accelerator with an accelerator of higher acceleration. It also shows that encryption on UltraSPARC T2 is a compute-bound operation. The not so small g_1 signifies reasonable overheads and thus a potential for an optimization.

Figure 6.4 also shows that a particular region can have multiple optimization possibilities. For example, both overheads and acceleration can provide gains from 2KB to 32KB. An architect has to make a decision in choosing an optimization as there may be pros and cons associated with each optimization option.

6.2.3 Totem Plots

In the previous section, we have considered optimizations with 10x variation of each parameter. The resulting plots help architects and programmers in identifying preferable optimization early in the design stage. But these plots may not be helpful in answering

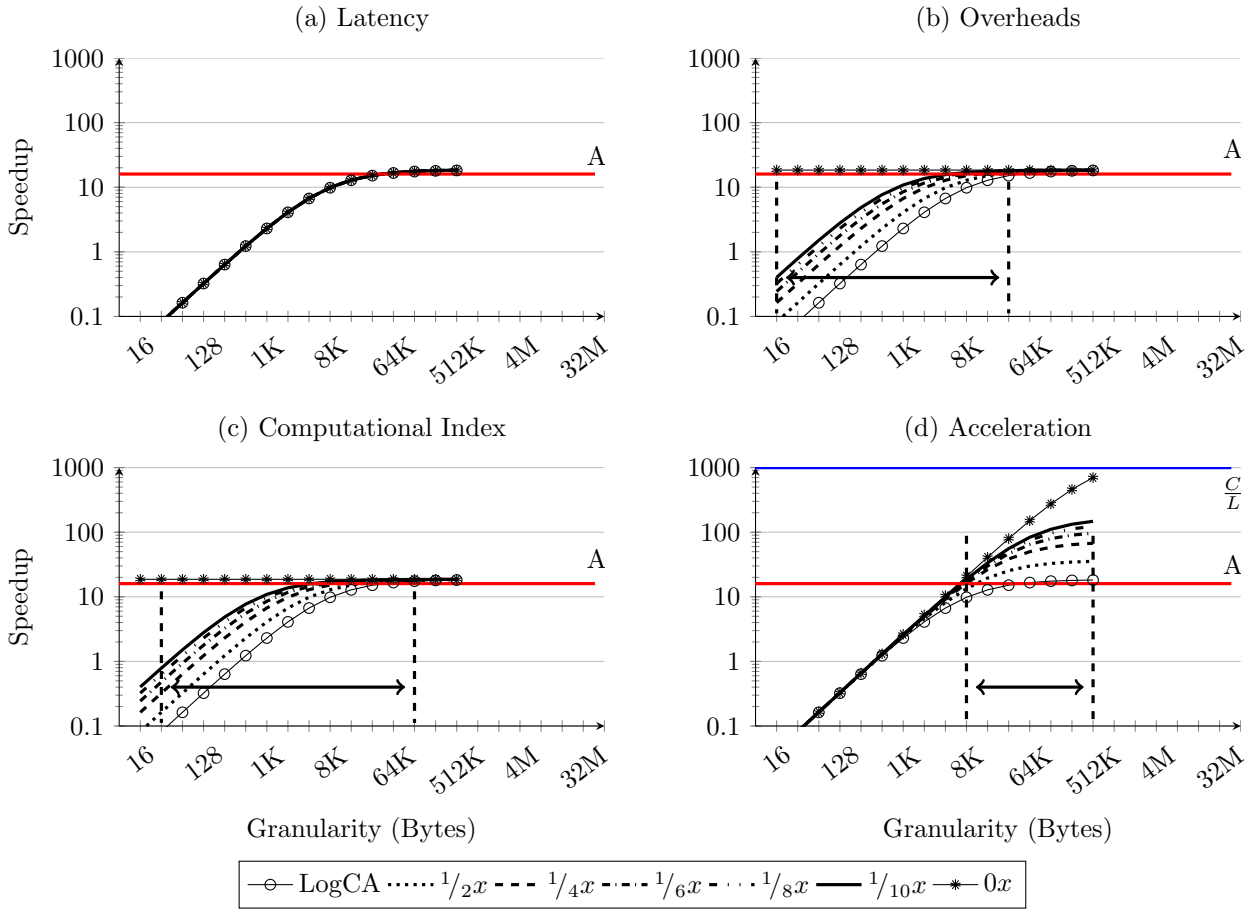


Figure 6.5: Totem plots for AES on UltraSPARC T2. We vary each LogCA parameter from 2x to 10x. We also consider the hypothetical case of decreasing (increasing) each parameter to zero (infinity).

these *what-if* questions: 1) What if the architects and programmers do not have the freedom for a parameter variation of 10x? 2) Instead of investing large resources on optimization, what if the same gains can be achieved by smaller variation of the parameters?

To answer these questions, and analyze the effect of smaller improvements we introduce *totem plots* in this section. Totem plots investigate whether comparable gains can be achieved by smaller variations in LogCA parameters. We draw the totem plots by varying each parameter from 2x-10x. To determine an upper bound on the resulting gains, we also

consider two hypothetical cases with extreme values, i.e., increasing a parameter to infinitely large value and decreasing a parameter to zero. Figures 6.5 - 6.10 show these plots for all machines in our analysis while running AES.

Figures 6.5 shows the totem plots for an encryption kernel on the cryptographic accelerator on UltraSPARC T2. Figure 6.5 (a) shows the totem plots for the latency. As we note earlier, a 10x reduction in latency on this machine does not provide any gains, it is not surprising that there are no gains for 2x-8x reduction either. But it is interesting to note that the achievable gains with the extreme case of zero latency are not different either. So for designs similar to UltraSPARC T2, latency should have the lowest priority for optimization.

Figure 6.5 (b) shows the totem plots for overheads. We observe that for smaller granularities—the effective optimization region for overheads—reducing overheads even by 2x or 4x provide significant gains. Further reducing the overheads does not provide considerable gains. This observation can guide an architect that significant gains can be achieved by allocating limited resources for optimizing the overheads.

Totem plots for computational index in Figure 6.5 (c) show a similar result. If operating at smaller granularities, a programmer can use this insight to get significant gains by moderate increase of computational index.

Figure 6.5 (d) shows that for small granularities, even with a infinite acceleration, the gains are not significant. While for large granularities 2x and 4x optimization of acceleration provides significant gains but further increase in acceleration provides diminishing returns. We also observe that systems with granularity independent latency, speedup with infinite acceleration is limited by the computational intensity. This observation provides an upper bound on the achievable performance.

Figures 6.6 and 6.7 show totem plots for the encryption kernel on Sandy Bridge and

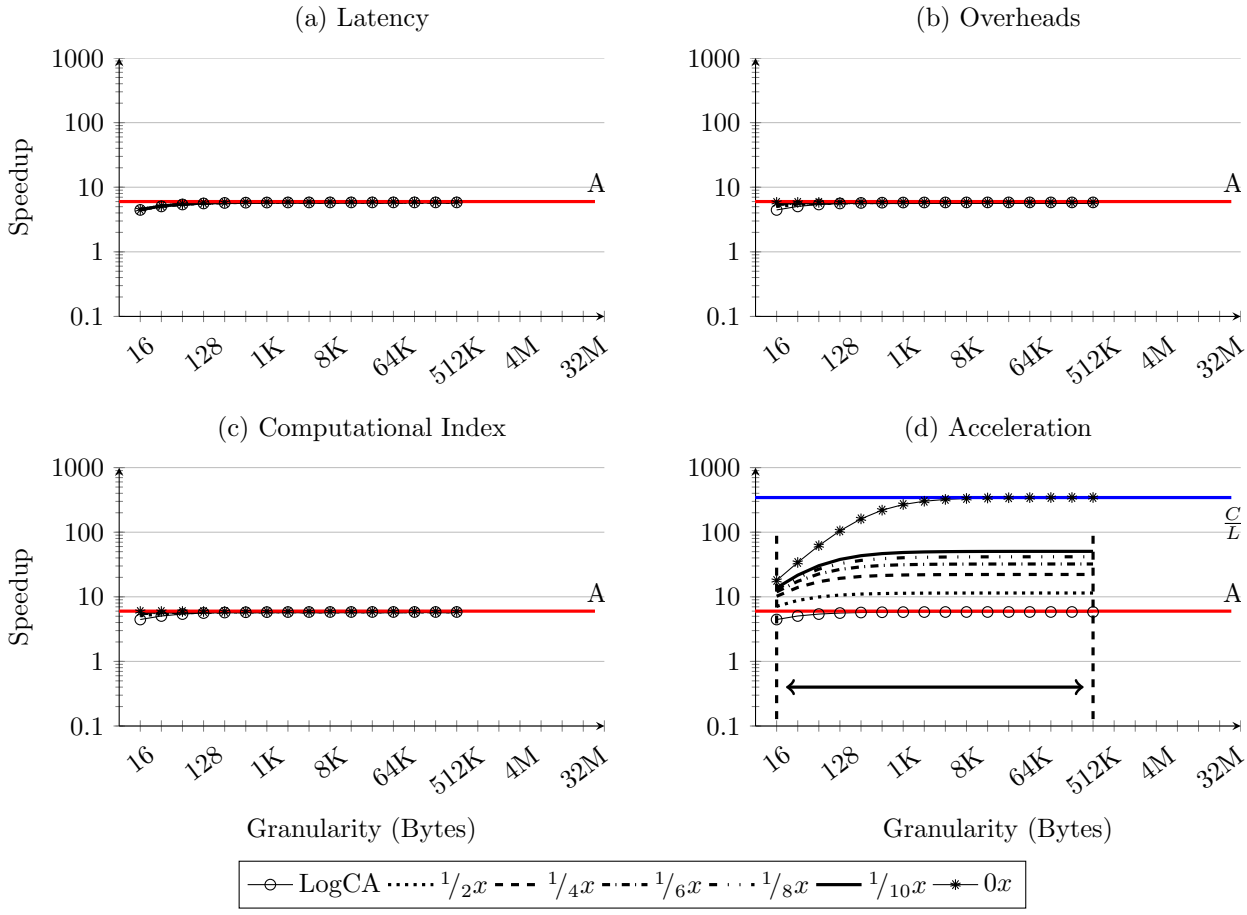


Figure 6.6: Totem plots of AES on Sandy Bridge. We vary each parameter (a) Latency (b) Overhead (c) Computation Index and (d) Acceleration, from 2x to 10x. We also consider the hypothetical case of decreasing (increasing) each parameter to zero (infinity).

SPARC T4 respectively. These machines represent on-chip accelerators and in this respect are similar to UltraSPARC T2 design (Figure 6.5). This behavior is also evident as the latency parameter does not provide any opportunity for optimization. However, in contrast to UltraSPARC T2 design, they use low-overhead special instructions for encryption which results in almost no optimization opportunity for overheads. With overheads and latency bottlenecks already addressed for these machines, optimizing acceleration is logically the next optimization opportunity. This opportunity is also evident from the plots as the

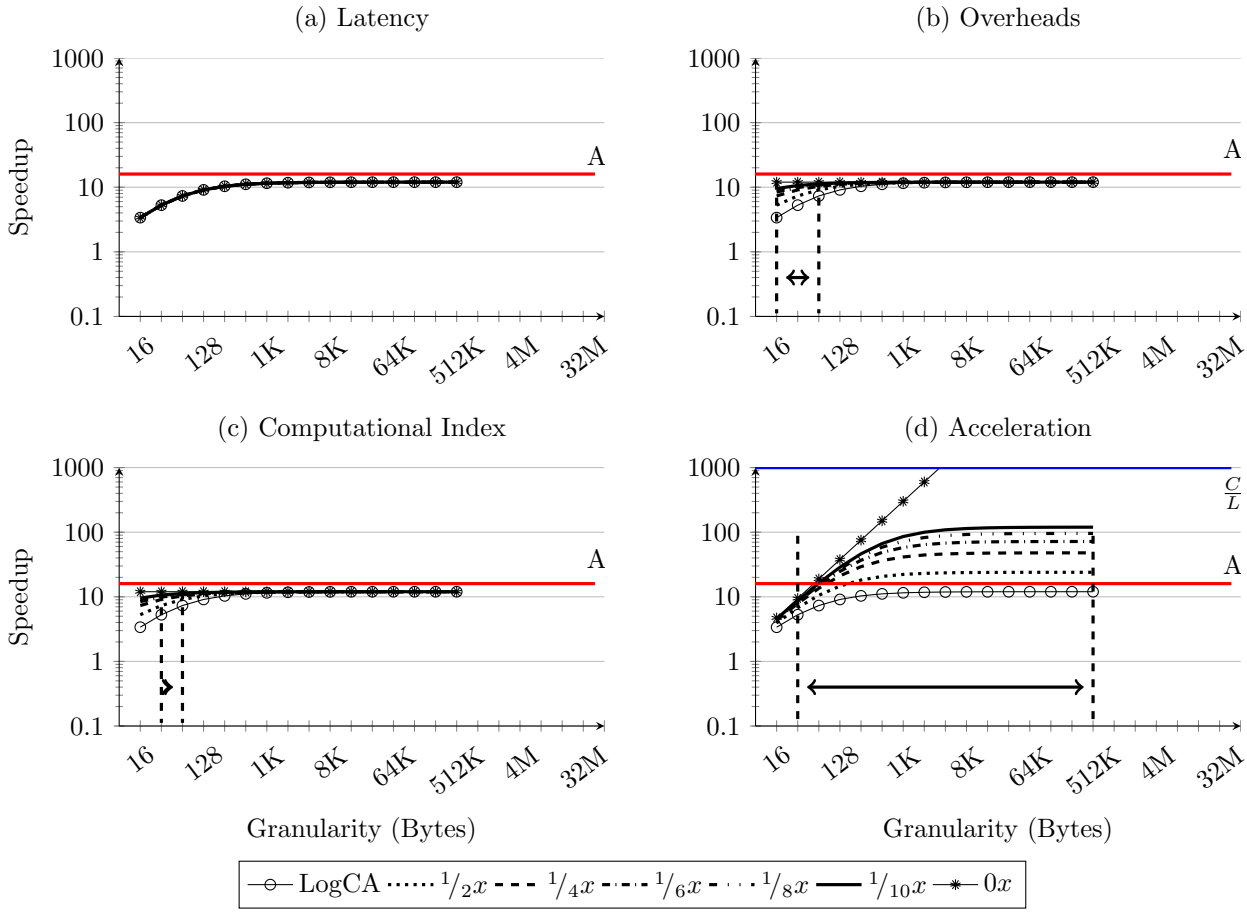


Figure 6.7: Totem plots for AES on T4 instr. We vary each parameter (a) Latency (b) Overhead (c) Computational index and (d) Acceleration, from 2x to 10x. We also consider the hypothetical case of decreasing (increasing) each parameter to zero (infinity). The marked region indicates optimizations which provide at least 20% gains

optimization range for acceleration parameter encompass the whole granularity range for both Sandy Bridge and SPARC T4.

Figures 6.8 and 6.9 show the totem plots for AES on a PCIe connected accelerator and discrete GPU respectively. In contrast to on-chip accelerators, both of these machines represent off-chip accelerators and this is also evident from the large optimization opportunity for latency. Computational index also provides a comparable optimization opportunity

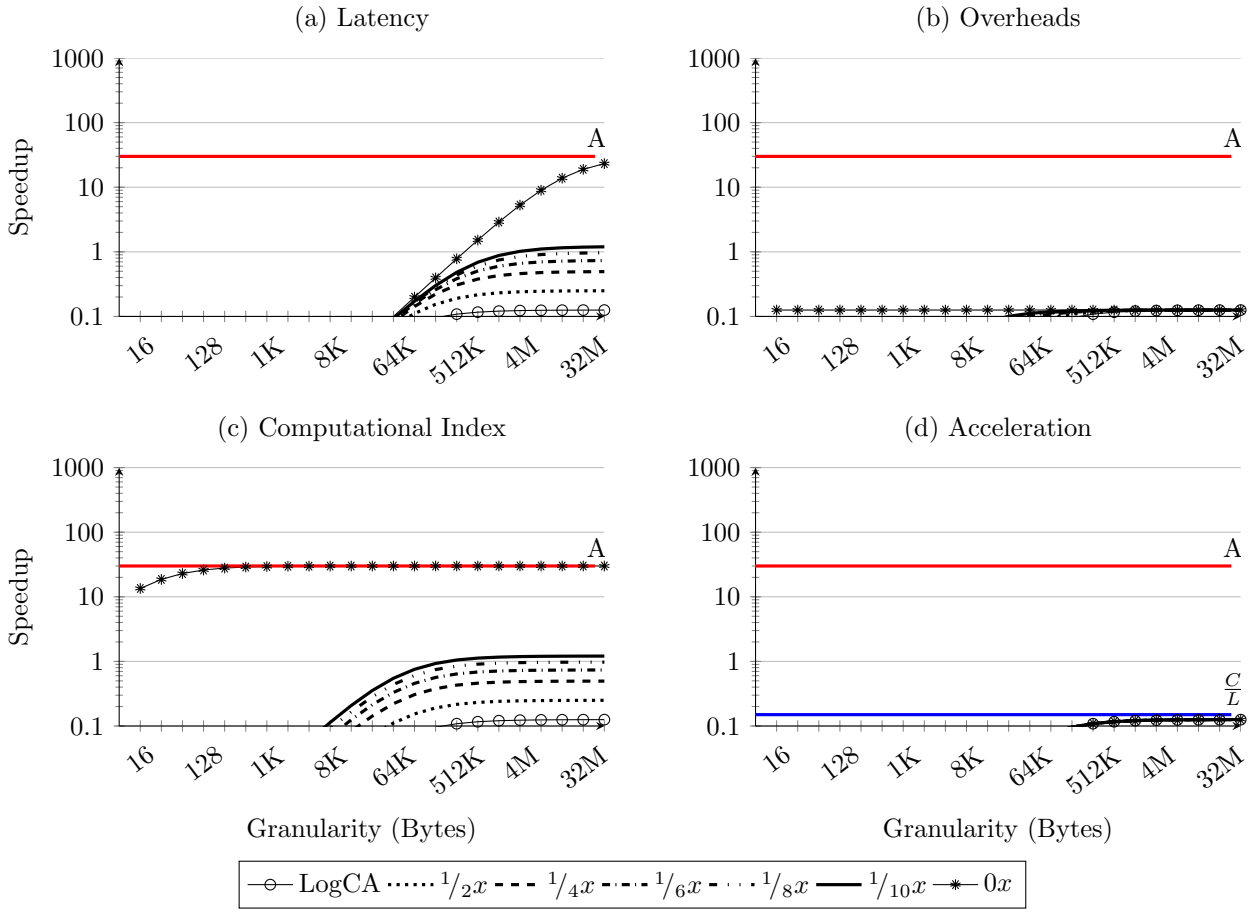


Figure 6.8: Totem plots for AES on PCIe accelerator as we vary each LogCA parameter from 2x to 10x. We also consider the hypothetical case of decreasing (increasing) each parameter to zero (infinity). The marked region indicates optimizations which provide at least 20% gains

which is tantamount to reducing the latency. We also observe that acceleration does not provide substantial benefits at large granularities, even for the hypothetical case of infinite acceleration. Thus signifying low optimization opportunity for acceleration in off-chip accelerators.

Figure 6.10 shows the totem plots for AES on the APU. These plots are similar to those of the discrete GPU. This behavior is not surprising as despite being an integrated accelerator,

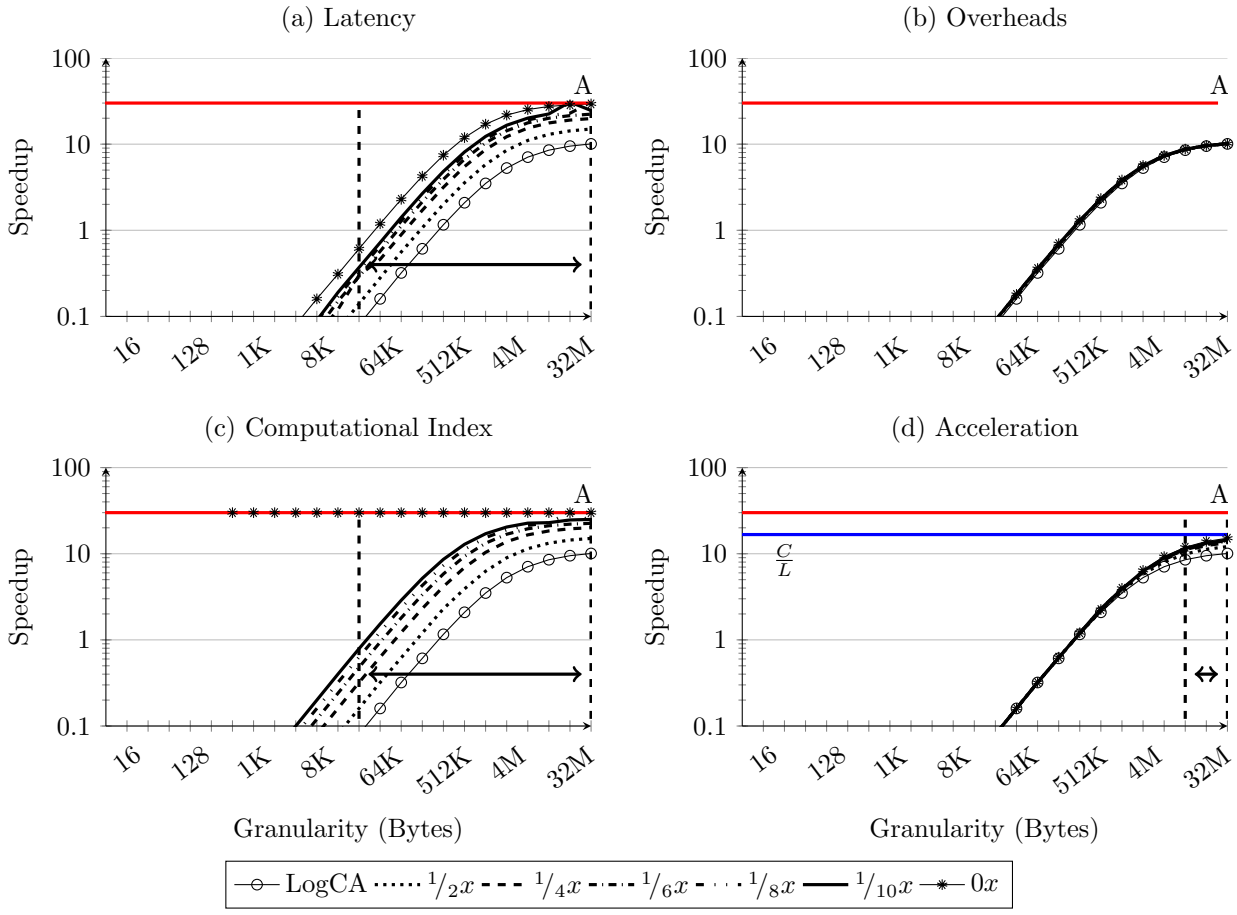


Figure 6.9: Totem plots of AES on NVIDIA GPU. We vary each parameter (a) Latency (b) Overhead (c) Computational index and (d) Acceleration, from 2x to 10x. We also consider the hypothetical case of decreasing (increasing) each parameter to zero (infinity).

APU requires explicit copying between the host and device memories. Similar to discrete GPU, latency and computational index provides a lot of optimization opportunity. We also observe that optimizing acceleration yields slightly better optimization opportunity than discrete GPU.

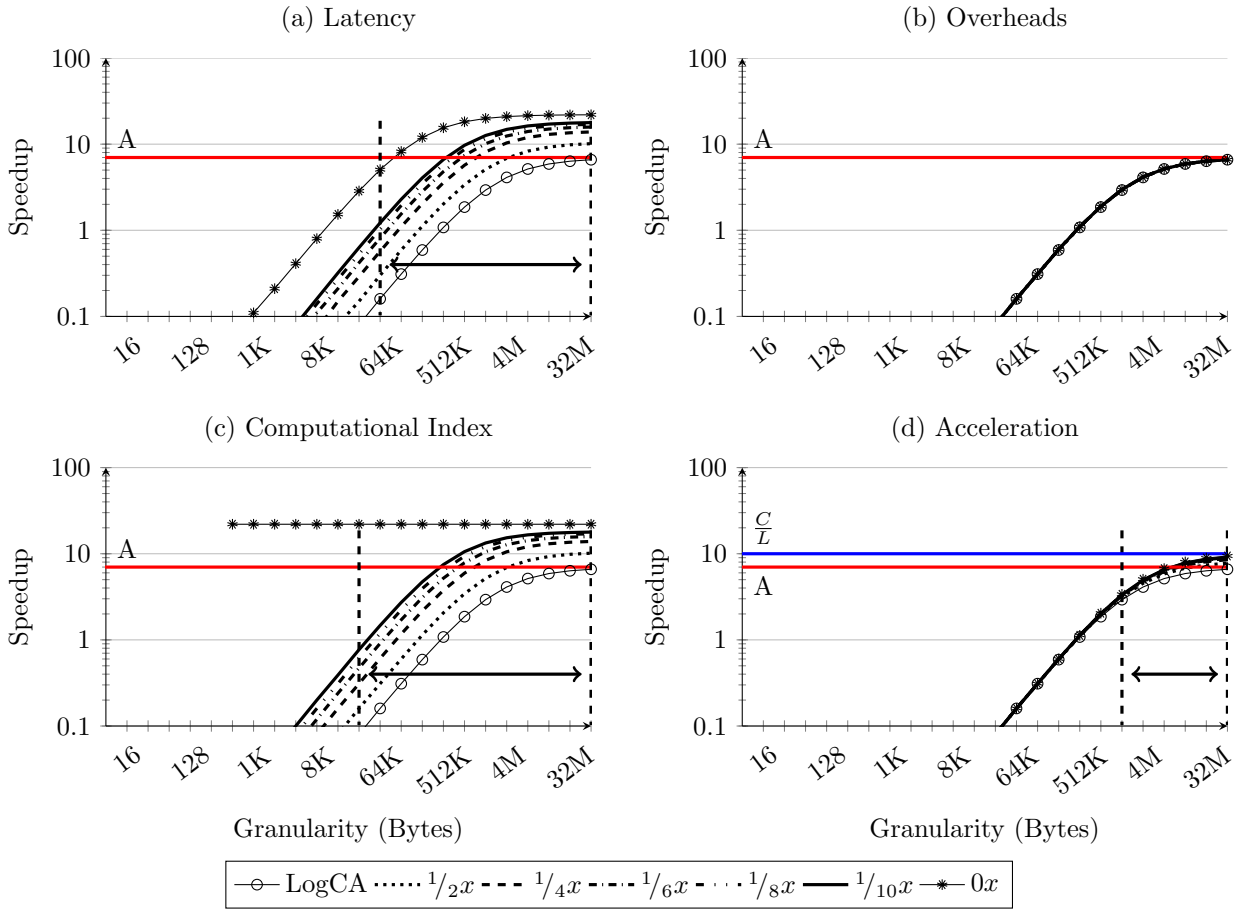


Figure 6.10: Totem plots of AES on AMD APU. We vary each parameter (a) Latency (b) Overhead (c) Computational index and (d) Acceleration, from 2x to 10x. We also consider the hypothetical case of decreasing (increasing) each parameter to zero (infinity).

6.3 Case Studies

In this section, we present two case studies to present the efficacy of our model. For the first case, we consider cryptographic accelerators connected through different interfaces, and discuss the evolution of these designs in Sun/Oracle SPARC machines. For the second case, we consider three different GPUs architectures. In both cases, we elaborate Accelerometer using the insights it provides about accelerator's interface, the design bottleneck it exposes

and optimization opportunities it suggests.

6.3.1 Cryptographic Accelerators

Figure 6.11 shows the evolution of cryptographic accelerators in SPARC architectures from the off-chip cryptographic accelerators in pre-Niagara (Figure 6.11 (a)) to cryptographic accelerators integrated within the pipeline in SPARC T4 (Figure 6.11 (e)). We observe that latency is absent in the on-chip accelerators' optimization regions, as these accelerators are closely coupled with the host. We also note that the optimization region with overhead parameter—representing the complexity of an accelerator's interface—shrinks while the optimization regions with acceleration parameter expand from Figure 6.11 (a-e). For example, for the off-chip cryptographic accelerator, the cut-off granularity for overheads is 256KB, whereas it is 128B for the SPARC T4, suggesting a much simpler interface.

Figure 6.11 (a) shows the optimization regions for the off-chip cryptographic accelerator connected through the PCIe bus. We note that the overhead and latency parameter occupy most of the optimization regions, indicating high overhead OS calls and high-latency data copying over the PCIe bus as the bottlenecks. As mentioned earlier, these bottlenecks also suggest potential for optimization.

Figure 6.11 (b) shows the optimization regions for UltraSPARC T2. The large cut-off granularity for overheads at 32KB suggests a complex interface, indicating a high overhead OS call creating a bottleneck at small granularities. The cut-off granularity of 2KB for acceleration suggests that optimizing acceleration is beneficial at large granularities. Similar to UltraSPARC T2, Figure 6.11 (c) shows the optimization regions for SPARC T3. These regions closely resemble those of UltraSPARC T2 in Figure 6.11 (b), and suggests that on-chip accelerator on SPARC T3 uses a similar interface.

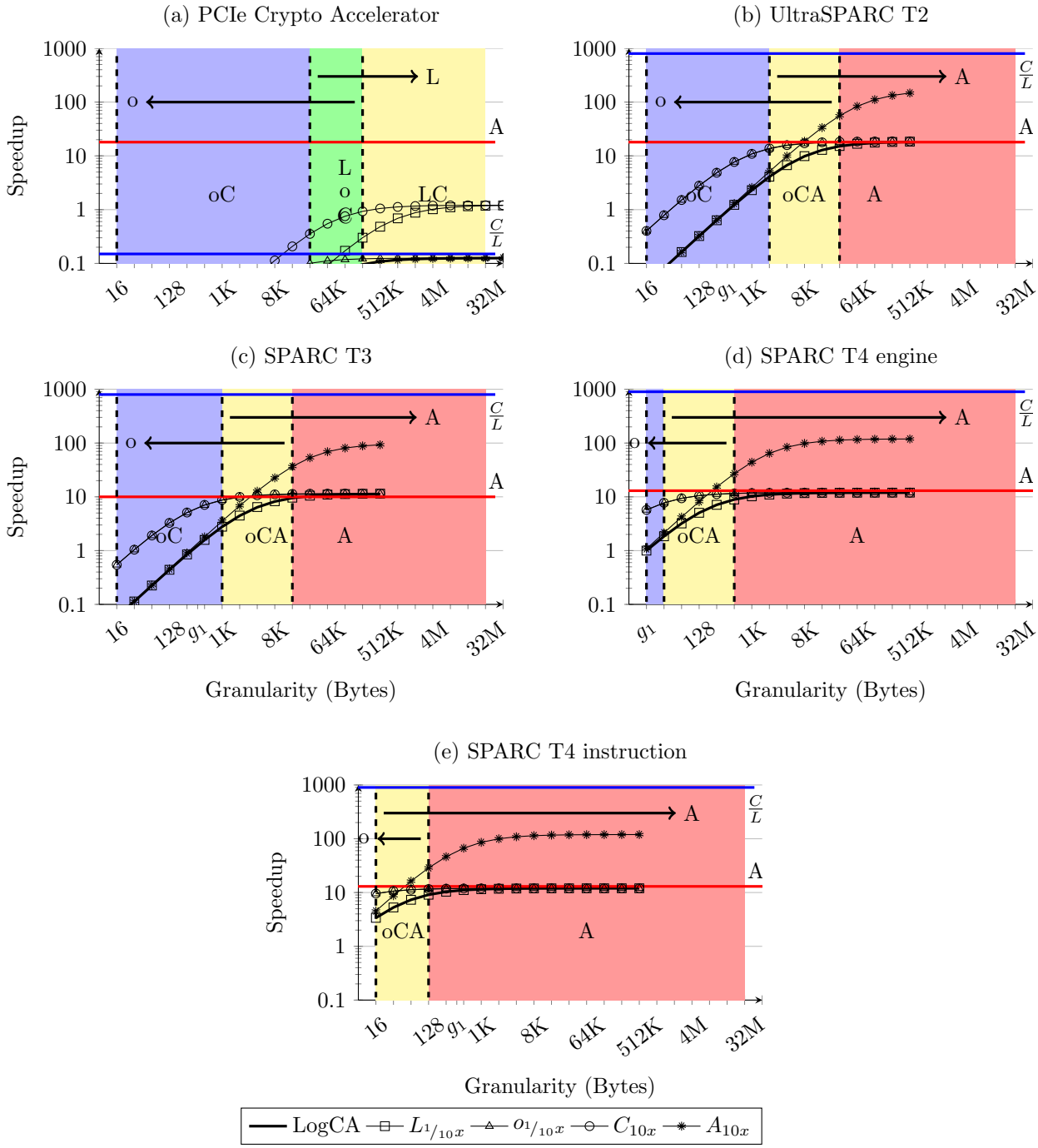


Figure 6.11: LogCA for performing Advanced Encryption Standard [2] on various cryptographic accelerators. LogCA identifies the design bottlenecks through LogCA parameters in an optimization region. The bottlenecks which LogCA suggests in each design is optimized in the next design.

Figure 6.11 (d) shows optimization regions for an on-chip accelerator on SPARC T4. There are three optimization regions, with the cut-off granularity for overhead now reduced to only 512B. This observation suggests a considerable improvement in the interface design over SPARC T3 and it is also evident by a smaller g_1. We also note that the cut-off granularity for acceleration now decreases to 32B, showing an increase in the opportunity for optimizing acceleration.

Figure 6.11 (e) shows optimization regions for cryptographic instructions on SPARC T4. We observe that unlike earlier designs, it has only two optimization regions and the speedup approaches the acceleration at a small granularity of 128B. In contrast, UltraSPARC T2 and SPARC T3 provide no gains at this granularity. We also observe that the cut-off granularity for overheads further reduces to 128B, suggesting some opportunity for optimization at very small granularities. The model also shows that the acceleration occupies the maximum range for optimization. For example, optimizing acceleration provides benefits for granularities greater than 16B. The low overhead access is due to the non-privileged instruction SPARC T4 uses to access the cryptographic unit, which is integrated within the pipeline.

The reduced overheads which Accelerometer indicates in SPARC T4 are due to the *fast-path* access introduced by Sun in their SPARC design codenamed *Rainbow Falls* [96]. Unlike earlier designs, the source (host) and destination (accelerator) pages are pre-pinned during initial access to an accelerator. Thus, an application does not have to go through the high-overhead OS calls on each access to an accelerator, resulting in lower overheads. We should mention that Sun claims [96] to enable the *fast-path* in SPARC T3. However, we observe this behavior only in SPARC T4.

We note that although all SPARC machines use the same accelerator, yet Figure 6.11

shows these accelerators have different acceleration. For example, UltraSPARC T2 with 18, and SPARC T4 with 13. This happens because of the way we calculate the acceleration. As mentioned earlier, we compute the acceleration by computing speedup at very high granularity. So a machine operating at high frequency will have lower acceleration and vice-versa. Since UltraSPARC T2 is operating at the lowest frequency, it has the highest acceleration.

Summary: We show in Figure 6.11 (a)-(e) that the adopted optimizations in each design is similar to Accelerometer's suggestion. For example, Accelerometer suggests that for small granularities, resources should be spend on improving the interface and for large granularities, resources should be spend on improving acceleration. Since encryption jobs are mostly smaller in size, over the years we have mainly observed improvement in the interface design. For the future designs, our model suggests that more resources should be allocated for improving acceleration.

6.3.2 General-Purpose Accelerators

Now we discuss Accelerometer for different GPUs architectures and discuss the insights it provides about these architectures. We choose kernels with varying complexity to observe the effect of kernel complexity on the optimization regions.

Figure 6.12 shows Accelerometer for performing matrix multiplication on a discrete NVIDIA GPU, an AMD integrated GPU and an integrated AMD GPU with HSA support. Figure 6.12 shows that matrix multiplication is compute bound on all three GPUs. We also observe a higher value of g_1 for the discrete GPU and APU as compared to HSA GPU, signifying a complex interface—with high overheads and latencies—for the discrete GPU and APU.

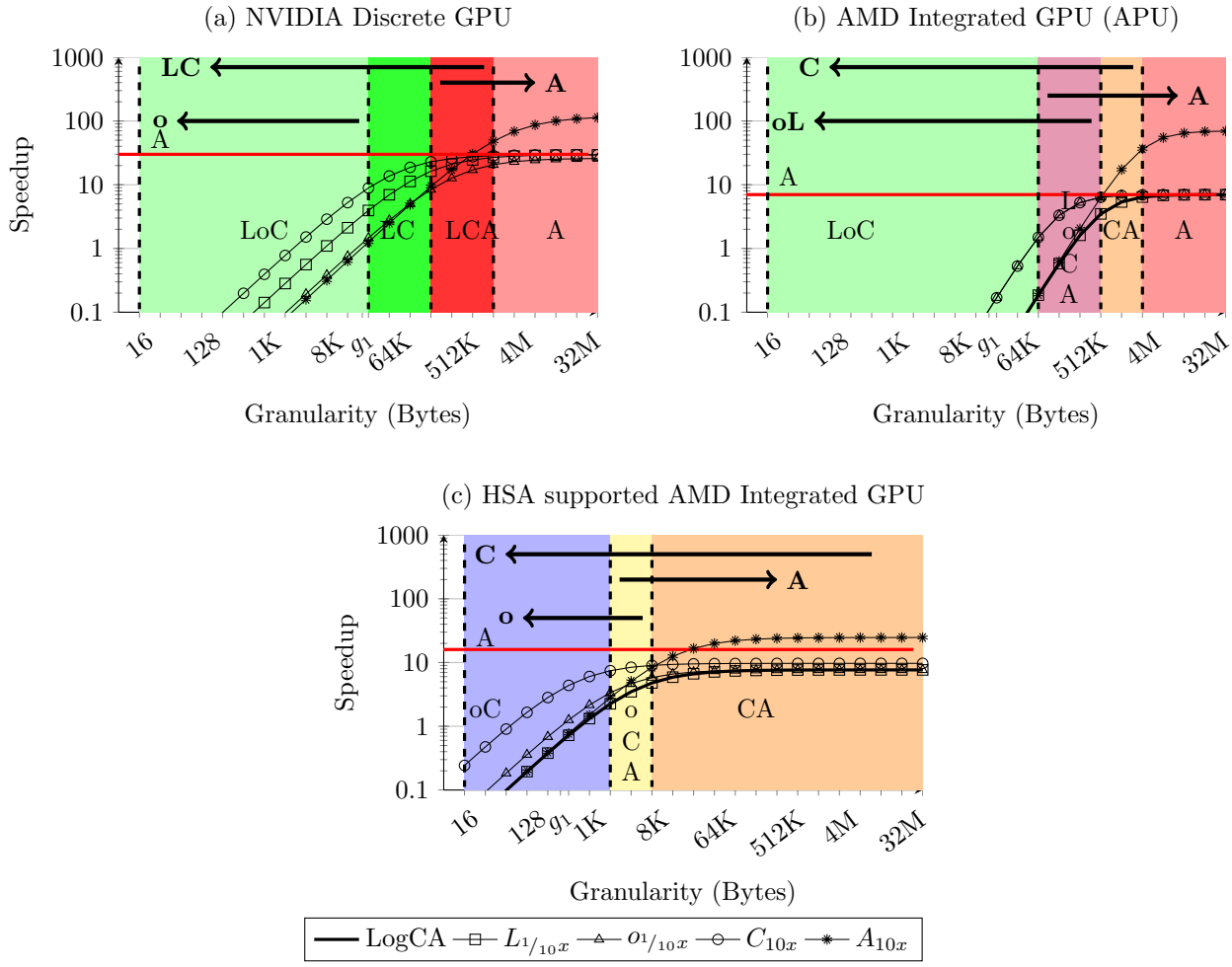


Figure 6.12: Various optimization regions for matrix multiplication over a range of granularities on (a) NVIDIA discrete GPU, (b) AMD APU and (c) HSA Supported GPU.

Figure 6.12 shows the evolution of memory interface design in GPU architectures. It shows the optimization regions for matrix multiplication on a discrete NVIDIA GPU, an AMD integrated GPU (APU) and an integrated AMD GPU with HSA support. We observe that the computational index occupies most of the regions which signifies maximum optimization potential for programmers.

The discrete GPU has four optimization regions (Figure 6.12 (a)). Among these, latency

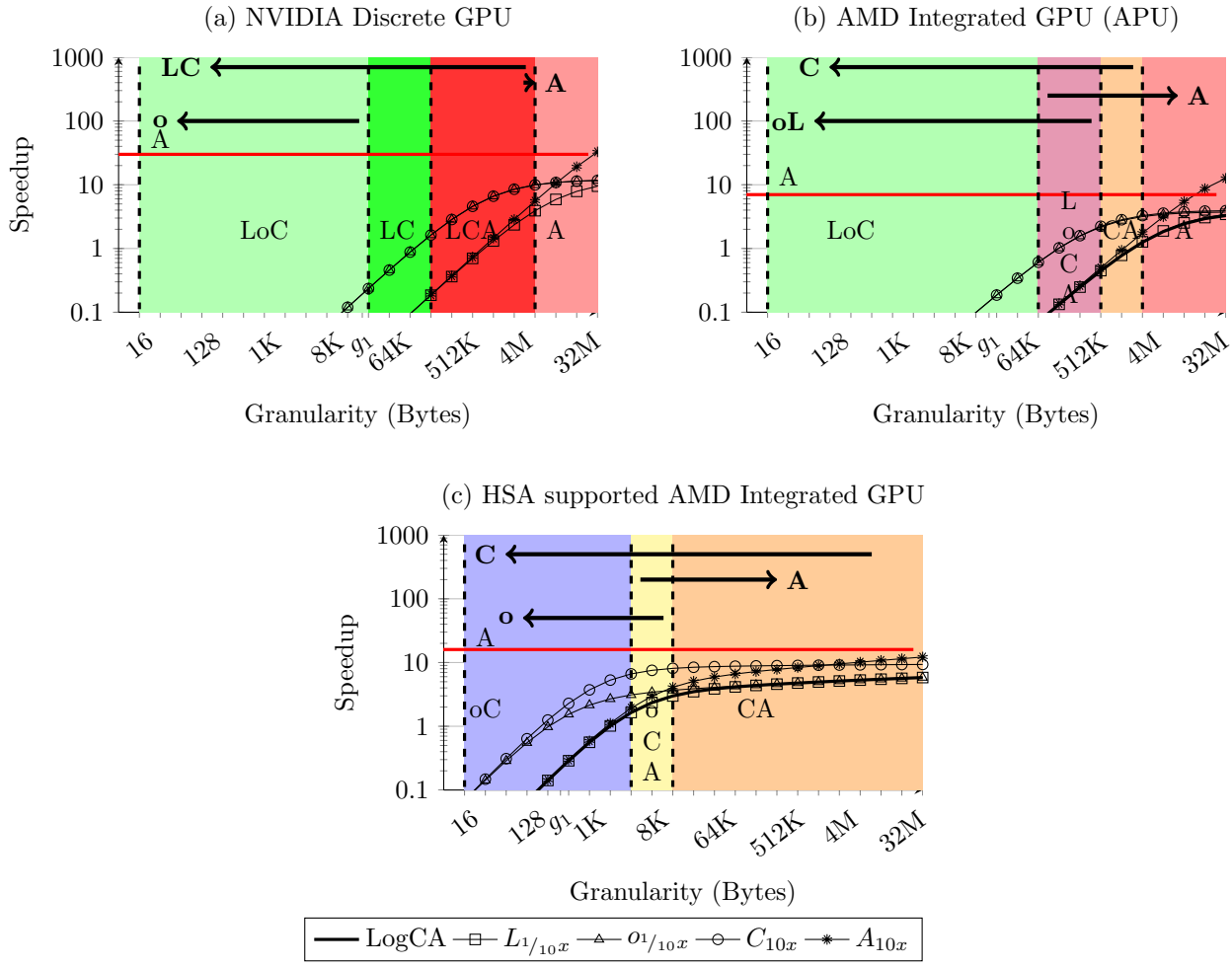


Figure 6.13: Various Optimization regions for FFT over a range of granularities on (a) NVIDIA discrete GPU, (b) AMD APU and (c) HSA Supported GPU.

dominates most of the regions, signifying high-latency data copying over the PCIe bus and thus maximum optimization potential. The high cut-off granularity for overheads at 32KB indicates high overhead OS calls to access the GPU. Similarly with highly aggressive cores, acceleration has high cut-off granularity of 256KB indicating less optimization potential for acceleration.

Similar to the discrete GPU, the APU also has four optimization regions (Figure 6.12 (b)).

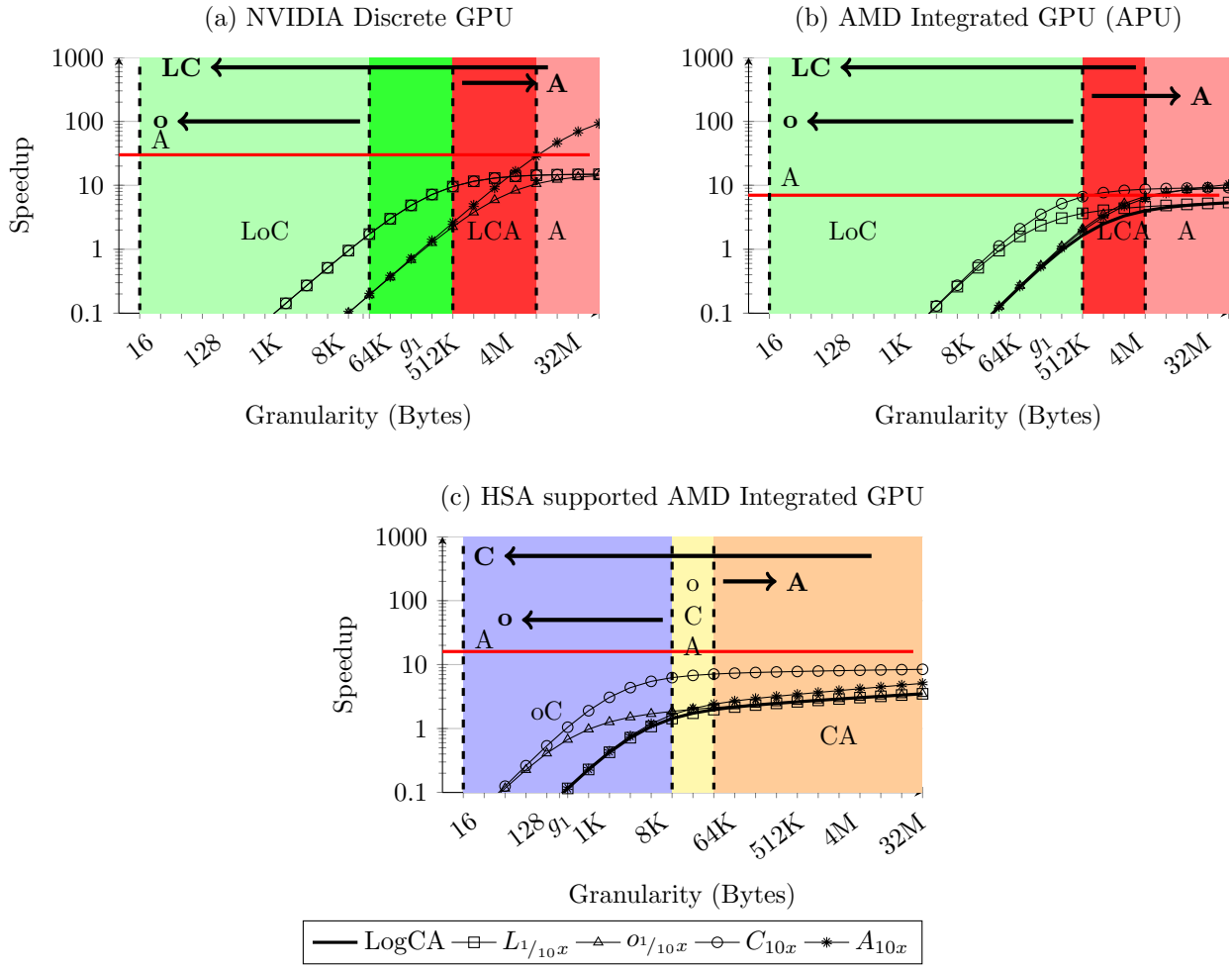


Figure 6.14: Various Optimization regions for Radix Sort over a range of granularities on (a) NVIDIA discrete GPU, (b) AMD APU and (c) HSA Supported GPU.

There are few notable differences from the discrete GPU. For example: with the elimination of data copying over the PCIe bus, the cut-off granularity for latency reduces to 512KB; the cut-off granularity for overheads is still high suggesting high overhead OS calls to access the APU and with less aggressive cores, the cut-off granularity for acceleration reduces to 64KB, implying more optimization potential for acceleration.

Figure 6.12 (c) shows three optimization regions for the HSA-enabled integrated GPU.

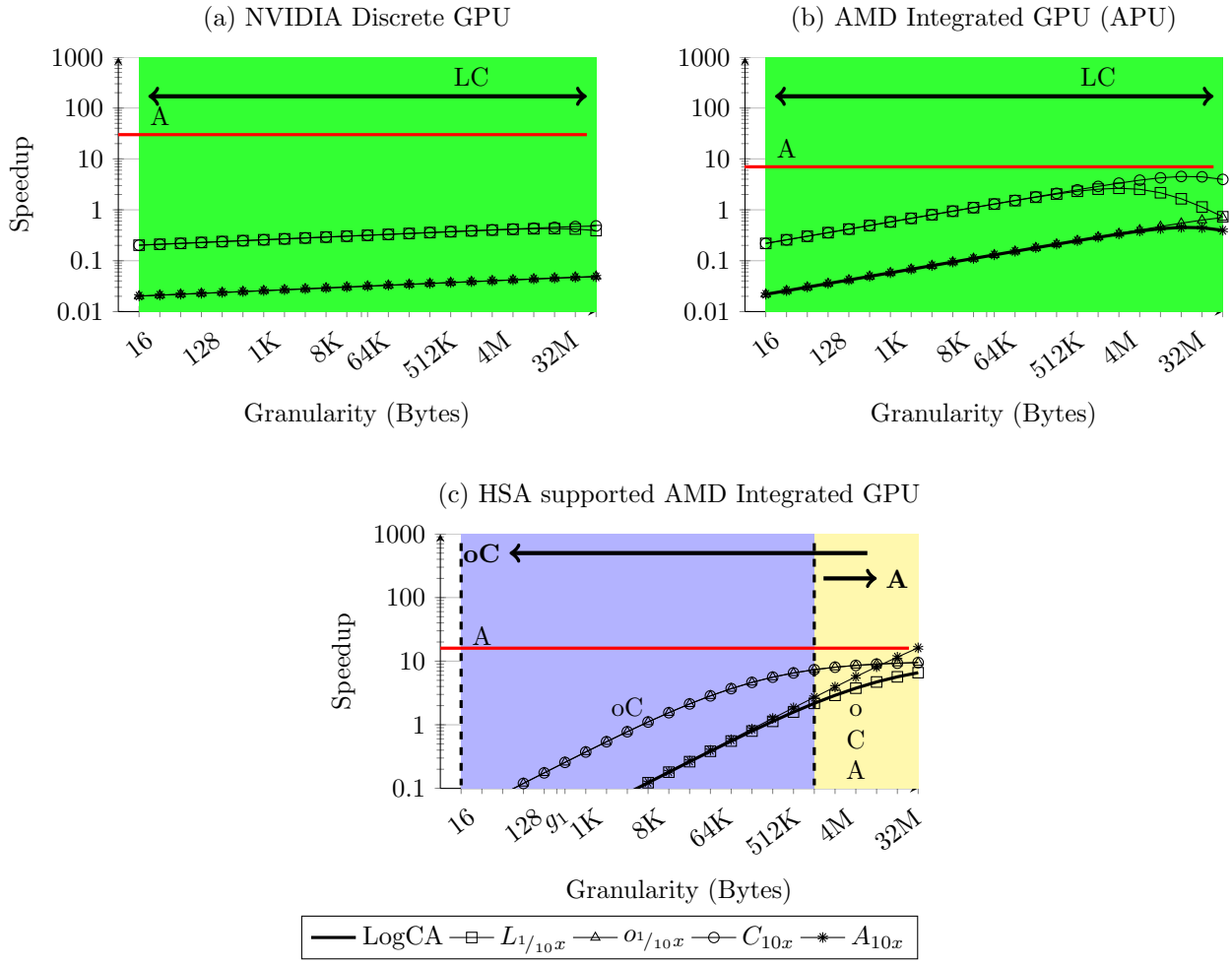


Figure 6.15: Various Optimization regions for Binary Search over a range of granularities on (a) NVIDIA discrete GPU, (b) AMD APU and (c) HSA Supported GPU.

We observe that latency is absent in all regions and the cut-off granularity for overhead reduces to 8KB. These reductions in overheads and latencies signify a simpler interface than the discrete GPU and APU. We also observe that the cut-off granularity for acceleration drops to 2KB, suggesting higher potential for optimizing acceleration.

Figures 6.13 and 6.14 show the sensitivity analysis plots for DCT and radix sort respectively. We observe that these plots are similar to matrix multiplication sensitivity plots

with some minor differences. These plots have similar optimization regions. However, the width of optimization regions change with the kernel complexity, i.e., as the complexity of the kernel decreases, the cut-off for optimizing overhead and latency increase. This behavior is not surprising, as we have noted earlier in chapter 4 that higher complexity kernels are less sensitive to latency and overheads.

The sensitivity analysis of binary search, Figure 6.15 (a-b), with sub-linear complexity on discrete GPU and APU, provide an interesting result. We already know from LogCA results in chapter 4 that sub-linear complexity kernels are not a good candidate for offloading on systems with granularity-dependent latencies. The sensitivity analysis further shows that optimizing computational index and latency provide optimization opportunities for both GPU and APU. However, for the discrete GPU, optimizing latency provides comparable benefits to computational index, whereas, on the APU optimizing latency is counter-productive at large granularities.

Figure 6.15-c shows the sensitivity analysis for the binary search on a system with granularity-independent latency. On such systems, optimizing overheads and computational index are beneficial for a range of granularities, whereas optimizing acceleration is beneficial only for very large granularities.

6.4 Discussion

In this section, we discuss how programmers and architects can use Accelerometer with LogCA to make decisions early in the design cycle. We consider different scenarios to elaborate our case.

Figures 6.16 and 6.17 show flow charts for programmers and architects, respectively. These flow charts are similar to the LogCA flow chart from Chapter 4 with some minor

changes. Specifically, we add extra steps for optimizing the LogCA parameters when the target speedup is not achievable. We describe these steps below and later explain them with a working example.

In the first step, consult the performance bound plots to determine the upper bounds on the performance. Then, leverage the sensitivity analysis plots to determine the optimization region for the operating granularity. In the next step, narrow down the scope of optimization by selecting an optimization parameter. Subsequently, determine whether a 10x optimization of that particular parameter results in the target speedup or not. If it does, then consult the totem plots to determine whether similar gains can be achieved by a smaller variation of the parameter. However, if the target speedup is still not achievable, consult the totem plots to determine the optimization potential for greater than 10x improvement.

For illustration purposes, we consider a design similar to UltraSPARC T2 in Figure 6.18, with a target speedup of 12. We also assume that the operating granularity is fixed at 4KB. The sensitivity analysis in Figure 6.18 shows that this operating granularity falls under the 'oCA' region. As we have noted earlier in Chapter 4 that for programmers,optimization opportunity lies in optimizing computational index, whereas for architects optimization opportunity lies in improving overheads, latencies and acceleration. So for the optimization region 'oCA', both programmers and architects can potentially optimize this design, and we discuss these scenarios below.

A programmer can offload a bigger chunk of data (16KB) to achieve the target speedup. But with the fixed granularity constraint, offloading larger data is not an option. In that case, a programmer can consult the sensitivity analysis plots and determine whether a 10x optimization of the computational index achieves the target speedup. In the next step, the

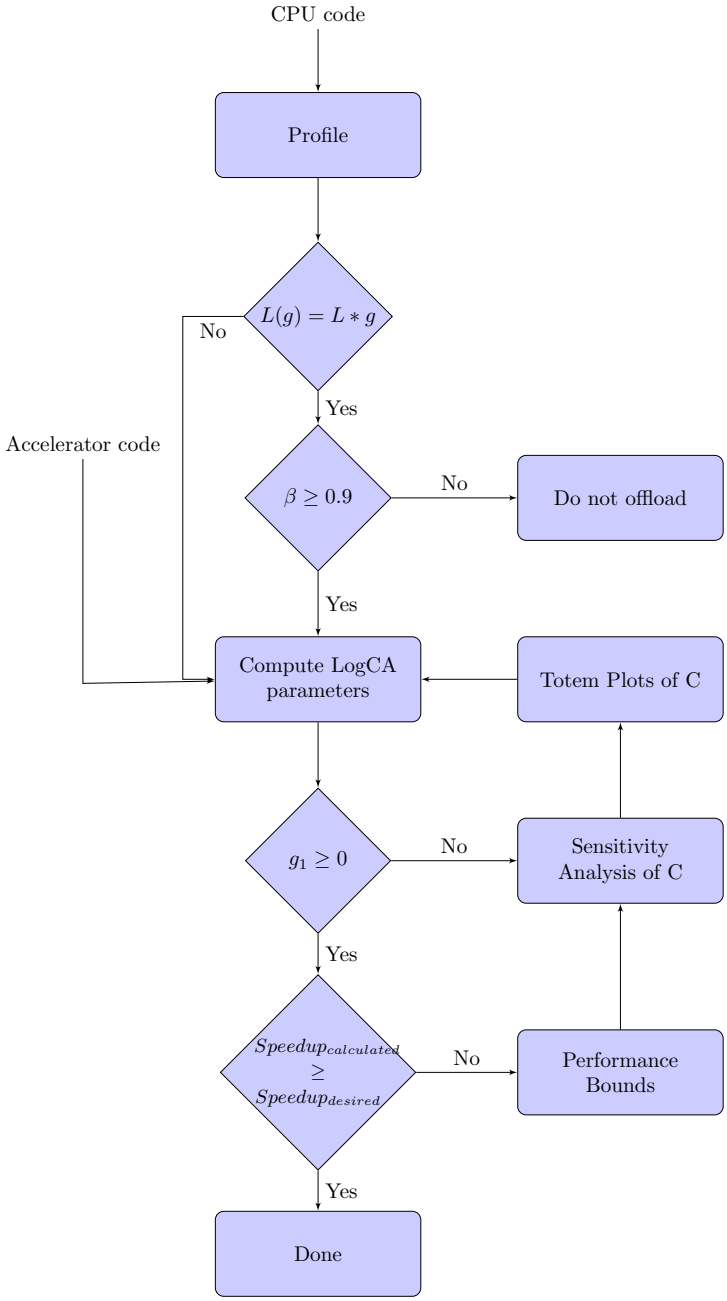


Figure 6.16: A flowchart for programmers for using Accelerometer with LogCA early in the design stage.

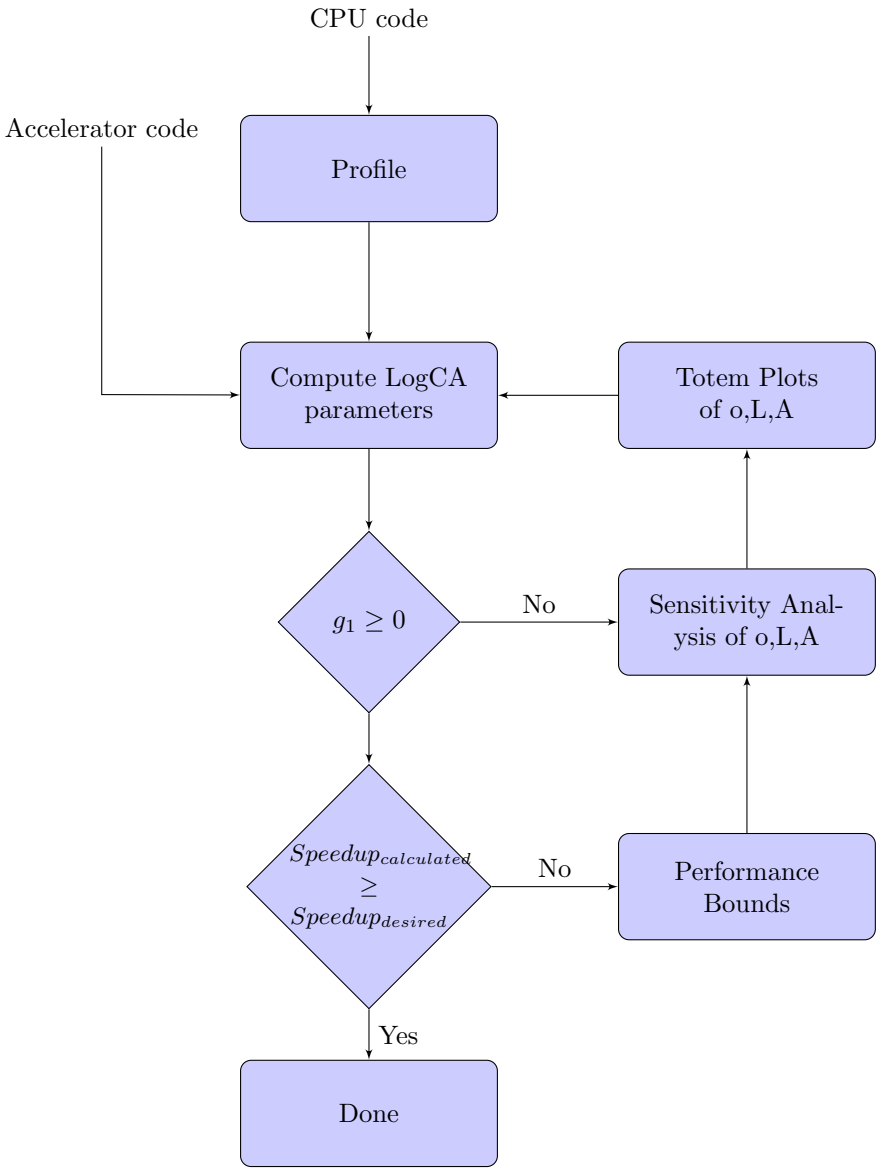


Figure 6.17: A flowchart for architects for using Accelerometer with LogCA early in the design stage.

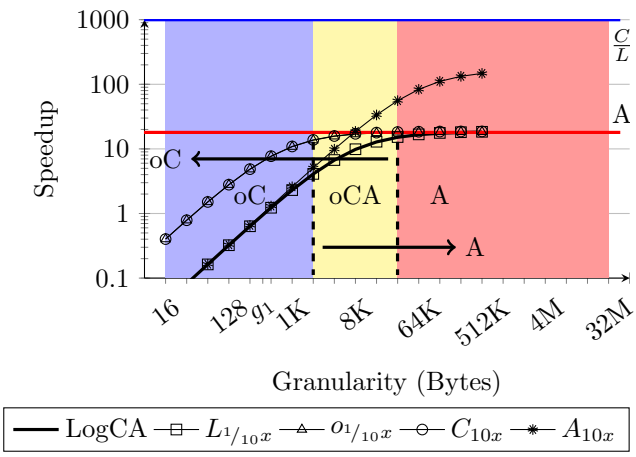


Figure 6.18: Optimization regions for UltraSPARC T2. The presence of a parameter in an optimization region indicates that it can at least provides 20% gains. The horizontal arrow indicates the cut-off granularity at which a parameter provides 20% gains.

programmer can refer the totem plots to determine whether similar gains can be achieved by a smaller variation in the computational index.

Similarly, an architect can consult the sensitivity analysis plots and determine that optimizing acceleration does not provide the target speedup. However, optimizing overhead provides the desired speedup. As we have mentioned in the above paragraph, the next step is to consult the totem plots and determine whether smaller variations in overheads results in similar gains. However, if for some reason, the overheads can not be reduced, i.e., interface can not be changed, an architect can look at the totem plots to determine whether a larger variation in acceleration achieves the target speedup or not.

6.5 Summary

In this chapter, we leverage LogCA and take inspiration from the Roofline model of multicore architectures to develop a visual performance model for accelerators. Our model

provides high-level insights, early in the design stage, to both architects and programmers. The model uses “bound and bottleneck" analysis to expose design limitation and suggest optimizations to overcome these bottlenecks.

For providing performance bounds, we have classified kernels as either compute or latency bound. We have done a sensitivity analysis of the LogCA parameters, and have partitioned the granularity range with this analysis. We have also demarcated the start and end of an optimization region for a particular parameter. We have also defined totem plots for a fine-grained sensitivity analysis of the parameters. The aforementioned techniques help in visually identifying the optimal strategy for optimization.

Using case studies of Sun/Oracle SPARC cryptographic accelerators and GPU architectures, we have shown that accelerometer captures the evolution of interface in these designs. We have also outlined a methodology which programmers and architects can employ for making decisions early in the design stage.

7 CONCLUSIONS

In this dissertation, we seek to answer this question—how effective are simple analytical models in predicting performance of hardware accelerators? To this end, we have made the following contributions.

7.1 Summary of Contributions

- In Chapter 4 we proposed LogCA—a high-level performance model for hardware accelerators. LogCA provided architects with insights about the interface of hardware accelerators. We defined performance metrics which helped programmers in deciding the right amount of data to offload. We showed that simple models are helpful for algorithms with regular access patterns.
- In Chapter 5 we complemented LogCA, with its energy counterpart, LogCA^e. We showed that a performance efficient design does not imply an energy efficient design. We also defined a metric—speedup-efficiency product—for evaluating trade-offs between performance and energy efficient designs.
- In Chapter 6 we proposed Accelerometer—a sensitivity analysis of the LogCA parameters. Accelerometer visually identified the performance bounds and bottlenecks associated with an accelerator design. We also defined various optimization regions and the potential gains associated with these regions.

7.2 Limitations and Directions for Future Work

Here, we discuss the limitations of our work and pointers to multiple research directions to overcome these limitations. A number of studies [98, 99, 100] have discussed various possible issues and future directions for accelerator-rich architecture. Our work can also be improved by building on insights from these studies.

The applicability of our modeling framework can be limited by the simplifying assumptions. For example, to keep our model simple, we abstracted aways caches and memories through the interface abstraction, and we modeled this interface with a single parameter. With this simplification, we do not capture the effect of caches and different memory access patterns. To overcome this limitation, our work can be complemented with similar analytical cache models [43, 101, 102].

For most of our analysis, we assumed a single accelerator system. However, there have been recent proposals for future heterogeneous architectures, featuring “sea-of-accelerators" [34]. Moreover, researchers have looked into partitioning resources for such architectures [103, 104, 105]. In these scenarios, our modeling framework will not be applicable as we do not explicitly model contention among resources.

We have also not considered the effect of pipelining and parallelization from an energy perspective. This is because as a first-order effect, these techniques result in higher energy consumption. However, the idea of pipelining and parallelizing computations is not new to the architecture community. Over the years, architects have proposed various optimization techniques—including analytical models [57, 61]—for reducing energy consumption in multicore architectures. Similar to these models, our modeling framework can be extended for optimizing parallel computations on accelerators.

In this dissertation, our design space, for the fixed-function accelerators, has focused

on encryption and hashing kernels on real systems. Expanding the design space on either real system or simulators, can be a potential future work.

For the sensitivity analysis in Chapter 6, we lacked a cost model to evaluate the trade-offs among multiple optimizations which provide similar gains. If each optimization provides equal benefits then the "cost"—in terms of finances or time duration—of each optimization may be the deciding factor. As it is quite possible that an optimization may provide more gains than the other but the cost for implementing that optimization over-weighs the potential benefits. A cost model, similar to the work of Debardelaben et al. [106] and Nguyen et al. [107], may help in answering these questions. But a cost model is beyond the scope of this dissertation and we leave it as a possible future work.

BIBLIOGRAPHY

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, pp. 65–76, 2009.

[2] AES, "Advanced Encryption Standard (AES)." `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`, 2001.

[3] " Secure Hash Standard." `http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf`, 2008.

[4] C. T. Morrison and R. T. Snodgrass, "Computer Science Can Use More Science," *Communications of the ACM*, vol. 54, p. 36, jun 2011.

[5] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, pp. 82–85, 1998.

[6] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Solid-State Circuits Newsletter*, vol. 12, 2007.

[7] ITRS, "International Technology Roadmap for Semiconductors, Executive Summary," 2011.

[8] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: Reducing the energy of mature computations," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, pp. 205–218, 2010.

[9] R. Merritt, "ARM CTO: power surge could create 'dark silicon'." `http://www.eetimes.com/document.asp?doc_id=1172049`, Oct. 2009.

[10] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceeding of the 38th annual international symposium on Computer architecture - ISCA '11*, (New York, New York, USA), p. 365, ACM Press, 2011.

[11] P. Bose, "Is Dark Silicon Real?: Technical Perspective," *Commun. ACM*, vol. 56, p. 92, feb 2013.

[12] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, pp. 6–15, 2011.

[13] M. Taylor, "Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pp. 1131–1136, 2012.

[14] S. Patel and W.-m. W. Hwu, "Accelerator Architectures," *IEEE Micro*, vol. 28, pp. 4–12, jul 2008.

[15] C. Cascaval, S. Chatterjee, H. Franke, K. J. Gildea, and P. Pattnaik, "A taxonomy of accelerator architectures and their programming models," *IBM Journal of Research and Development*, vol. 54, pp. 5:1–5:10, 2010.

[16] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," *Proceedings of the 37th annual international symposium on Computer architecture - ISCA '10*, p. 37, 2010.

[17] V. Govindaraju, C. H. Ho, and K. Sankaralingam, "Dynamically specialized datapaths for energy efficient computing," in *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 503–514, 2011.

[18] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, "QsCores: Trading Dark Silicon for Scalable Energy Efficiency with Quasi-Specific Cores," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-44 '11*, p. 163, 2011.

[19] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the Walkers: Accelerating Index Traversals for In-memory Databases," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, (New York, NY, USA), pp. 468–479, ACM, 2013.

[20] M. Lavasani, H. Angepat, and D. Chiou, "An FPGA-based In-Line Accelerator for Memcached," *IEEE Comput. Archit. Lett.*, vol. 13, pp. 57–60, jul 2014.

[21] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, "Q100: The Architecture and Design of a Database Processing Unit," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, (New York, NY, USA), pp. 255–268, ACM, 2014.

[22] S. Kumar, N. Vedula, A. Shriraman, and V. Srinivasan, "DASX: Hardware Accelerator for Software Data Structures," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, (New York, NY, USA), pp. 361–372, ACM, 2015.

[23] H. Franke, J. Xenidis, C. Basso, B. M. Bass, S. S. Woodward, J. D. Brown, and C. L. Johnson, "Introduction to the wire-speed processor and architecture," *IBM Journal of Research and Development*, vol. 54, pp. 3:1–3:11, 2010.

[24] J. Stuecheli, "POWER8," in *25th Hot Chip Symposium*, 2013.

[25] S. Phillips, "M7: Next Generation SPARC," in *26th Hot Chip Symposium*, 2014.

[26] Y. S. Shao and D. Brooks, *Research Infrastructures for Hardware Accelerators*. Morgan & Claypool Publishers, nov 2015.

[27] B. Wile, "Coherent Accelerator Processor Interface (CAPI) for POWER8 Systems," tech. rep., IBM, 2014.

[28] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, "CAPI: A Coherent Accelerator Processor Interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 1—-7, 2015.

[29] "Cache Coherent Interconnect for Accelerators (CCIX)." `http://www.ccixconsortium.com/`.

[30] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, (New York, NY, USA), pp. 451–460, ACM, 2010.

[31] C. Gregg and K. Hazelwood, "Where is the data? Why you cannot debate CPU vs. GPU performance without the answer," in *International Symposium on Performance Analysis of Systems and Software*, pp. 134–144, IEEE, apr 2011.

[32] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley \& Sons, 1990.

[33] L. Eeckhout, "Computer Architecture Performance Evaluation Methods," *Synthesis Lectures on Computer Architecture*, vol. 5, pp. 1–145, dec 2010.

[34] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures," in *International Symposium on Computer Architecture (ISCA)*, 2014.

[35] J. Cong, Z. Fang, M. Gill, and G. Reinman, "PARADE: A Cycle-Accurate Full-System Simulation Platform for Accelerator-Rich Architectural Design and Exploration," in *2015 IEEE/ACM International Conference on Computer-Aided Design*, (Austin, TX), 2015.

[36] N. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, p. 1, 2011.

[37] Y. C. Tay, *Analytical Performance Modeling for Computer Systems*. Morgan & Claypool Publishers, 2nd ed., 2013.

[38] J. D. C. Little and S. C. Graves, "Little's law," in *Building intuition*, pp. 81–100, Springer, 2008.

[39] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proceedings of the April 18-20, 1967, spring joint computer conference on - AFIPS '67 (Spring)*, p. 483, 1967.

[40] M. D. Hill and M. R. Marty, "Amdahl's Law in the Multicore Era," *Computer*, vol. 41, pp. 33–38, jul 2008.

[41] S. Fortune and J. Wyllie, "Parallelism in Random Access Machines," in *Proceedings of the tenth annual ACM symposium on Theory of computing - STOC '78*, (New York, New York, USA), pp. 114–118, ACM Press, 1978.

[42] D. Culler, R. Karp, D. Patterson, and A. Sahay, "LogP: Towards a realistic model of parallel computation," in *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 1–12, 1993.

[43] N. Beckmann and D. Sanchez, "Cache Calculus: Modeling Caches through Differential Equations," *Computer Architecture Letters*, vol. PP, no. 99, p. 1, 2016.

[44] M. Shoaib Bin Altaf and D. A. Wood, "LogCA: A Performance Model for Hardware Accelerators," *Computer Architecture Letters*, vol. PP, no. 99, p. 1, 2014.

[45] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, pp. 13–24, IEEE Press, 2014.

[46] P. Gupta, "Xeon+FPGA Platform for the Data Center," in *The Fourth Workshop on the Intersections of Computer Architecture and Reconfigurable Logic (CARL)*, 2015.

[47] D. Sheffield, "IvyTown Xeon + FPGA : The HARP Program," in *International Symposium on Computer Architecture (ISCA): Tutorial*, 2016.

[48] "Microsoft Goes All in for FPGAs to Build Out AI Cloud." `https://www.top500.org/news/microsoft-goes-all-in-for-fpgas-to-build-out-cloud-based-ai/`.

[49] "Intel Marrying FPGA, Beefy Broadwell for Open Compute Future." `https://www.nextplatform.com/2016/03/14/intel-marrying-fpga-beefy-broadwell-open-compute-future/`.

[50] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, pp. 272–281, jan 2015.

[51] E. S. Chung, P. a. Milder, J. C. Hoe, and K. Mai, "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?," *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 225–236, dec 2010.

[52] M. Hempstead, G.-Y. Wei, and D. Brooks, "Navigo: An early-stage model to study power-constrained architectures and specialization," in *ISCA Workshop on Modeling, Benchmarking, and Simulations (MoBS)*, 2009.

[53] M. R. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole, "Modeling and predicting performance of high performance computing applications on hardware accelerators," *International Journal of High Performance Computing Applications*, vol. 27, pp. 89–108, 2013.

[54] M. Daga, A. M. Aji, and W.-c. Feng, "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing," in *2011 Symposium on Application Accelerators in High-Performance Computing*, pp. 141–149, Ieee, jul 2011.

[55] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, vol. 38, pp. 280—-289, 2010.

[56] K. Kumar, J. Liu, Y. H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, pp. 129–140, 2013.

[57] D. H. Woo and H. H. S. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, pp. 24–31, 2008.

[58] K. W. Cameron and R. Ge, "Generalizing Amdahl's law for power and energy," *Computer*, vol. 45, pp. 75–77, 2012.

[59] X. H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 183–188, 2010.

[60] D. Cohen, F. Petrini, M. D. Day, M. Ben-Yehuda, S. W. Hunter, and U. Cummings, "Applying Amdahl's Other Law to the data center," *IBM Journal of Research and Development*, vol. 53, pp. 5:1–5:12, 2009.

[61] S. Cho and R. G. Melhem, "Corollaries to Amdahl's law for energy," *IEEE Computer Architecture Letters*, vol. 7, pp. 25–28, 2008.

[62] B. Juurlink and C. H. Meenderinck, "Amdahl's law for predicting the future of multicores considered harmful," *ACM SIGARCH Computer Architecture News*, vol. 40, p. 1, 2012.

[63] A. Marowka, "Extending Amdahl's Law for heterogeneous computing," in *Proceedings of the 2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2012*, pp. 309–316, 2012.

[64] S. Nilakantan, S. Battle, and M. Hempstead, "Metrics for Early-Stage Modeling of Many-Accelerator Architectures," *Computer Architecture Letters*, vol. 12, pp. 25–28, jan 2013.

[65] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, vol. 37, pp. 152–163, 2009.

[66] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 673–686, 2013.

[67] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 382–393, 2011.

[68] J. Lai and A. Seznec, "Break down GPU execution time with an analytical method," in *Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation Methods and Tools - RAPIDO '12*, (New York, New York, USA), pp. 33–39, ACM Press, 2012.

[69] J. Meng, V. A. Morozov, K. Kumaran, V. Vishwanath, and T. D. Uram, "GROPHECY: GPU Performance Projection from CPU Code Skeletons," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, (New York, New York, USA), p. 1, ACM Press, 2011.

[70] J. Sim, A. Dasgupta, H. Kim, and R. Vuduc, "A Performance Analysis Framework for Identifying Potential Benefits in GPGPU Applications," in *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, (New York, NY, USA), pp. 11–22, ACM, 2012.

[71] J. Lai and A. Seznec, "TEG: GPU Performance Estimation Using a Timing Model," tech. rep., INRIA, 2011.

[72] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, pp. 661–672, 2013.

[73] C. Nugteren and H. Corporaal, "The Boat Hull Model : Enabling Performance Prediction for Parallel Computing Prior to Code Development Categories and Subject Descriptors," in *Proceedings of the 9th Conference on Computing Frontiers*, pp. 203—-212, ACM, 2012.

[74] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, and Y. Li, "GPURoofline: A Model for Guiding Performance Optimizations on GPUs," in *Proceedings of the 18th International Conference on Parallel Processing*, Euro-Par'12, (Berlin, Heidelberg), pp. 920–932, Springer-Verlag, 2012.

[75] C. Nugteren and H. Corporaal, "A Modular and Parameterisable Classification of Algorithms," tech. rep., Eindhoven University of Technology, 2011.

[76] J. Lai and A. Seznec, "Performance upper bound analysis and optimization of SGEMM on Fermi and Kepler GPUs," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2013*, 2013.

[77] M. Shah, J. Barren, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Sana, D. Sheahan, L. Spracklen, and A. Wynn, "UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC," in *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*, pp. 22–25, nov 2007.

[78] S. Patel, "Sun's Next-Generation Multithreaded Processor: Rainbow Falls," in *21st Hot Chip Symposium*, 2009.

[79] M. Shah, R. Golla, G. Grohoski, P. Jordan, J. Barreh, J. Brooks, M. Greenberg, G. Levinsky, M. Luttrell, C. Olson, Z. Samoail, M. Smittle, and T. Ziaja, "Sparc T4: A dynamically threaded server-on-a-chip," *IEEE Micro*, vol. 32, pp. 8–19, 2012.

[80] S. Gueron, "Intel Advanced Encryption Standard (AES) Instructions Set," tech. rep., Intel Corporation, 2012.

[81] "vpn 1401, for Std. PCI-sockets." `http://soekris.com/products/vpn-1401.html`.

[82] P. Rogers, "Heterogeneous system architecture overview," in *Hot Chips*, 2013.

[83] "OpenSSL, Cryptography and SSL/TLS Toolkit." `https://openssl.org`.

[84] "APP SDK - A Complete Development Platform." `http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/`.

[85] "PKCS11 Engine Patch for OpenSSL." `https://blogs.oracle.com/janp/entry/pkcs_11_engine_patch_including`.

[86] N. Sun and C.-C. Lin, "Using the Cryptographic Accelerators in the UltraSPARC T1 and T2 processors," tech. rep., 2007.

[87] "How to tell if SPARC T4 crypto is being used?." `https://blogs.oracle.com/DanX/entry/how_to_tell_if_sparc`.

[88] Wattsup, "WattsUp?." `https://www.wattsupmeters.com/secure/products.php?pn=0&wai=212&spec=4`.

[89] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A View of the Parallel Computing Landscape," *Commun. ACM*, vol. 52, pp. 56–67, oct 2009.

[90] G. Wang, Y. Lin, and W. Yi, "Kernel Fusion: An Effective Method for Better Power Efficiency on Multithreaded GPU," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pp. 344–350, dec 2010.

[91] S. Tabik, G. Ortega, and E. M. Garzón, "Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study," *The Journal of Supercomputing*, vol. 70, pp. 577–587, nov 2014.

[92] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*. 2006.

[93] E. W. Weisstein, "Newton's Method." From MathWorld – A Wolfram Web Resource. `http://mathworld.wolfram.com/NewtonsMethod.html`.

[94] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, pp. 52–58, 2001.

[95] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *Proceedings of IEEE Symposium on Low Power Electronics*, pp. 8–11, IEEE, 1994.

[96] L. Spracklen, "Sun's 3rd generation on-chip UltraSPARC security accelerator," in *21st Hot Chip Symposium*, 2009.

[97] D. Lustig and M. Martonosi, "Reducing GPU Offload Latency via Fine-Grained CPU-GPU Synchronization," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 354–365, IEEE, feb 2013.

[98] R. Hou, L. Zhang, and M. Huang, "Efficient data streaming with on-chip accelerators: Opportunities and challenges," in *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pp. 312—-320, IEEE Computer Society, 2011.

[99] R. Iyer, "Accelerator-Rich Architectures: Implications, Opportunities and Challenges," *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pp. 106–107, 2012.

[100] B. Reagen, G.-Y. Wei, and D. Brooks, "How Hardware Accelerators Trade-Off Pipelining and Parallelism to Maximize Efficiency," in *Boston Area Architecture Workshop (BARC)*, 2015.

[101] N. Beckmann and D. Sanchez, "Talus: A Simple Way to Remove Cliffs in Cache Performance," in *Proceedings of the 21st international symposium on High Performance Computer Architecture (HPCA-21)*, 2015.

[102] N. Beckmann and D. Sanchez, "Modeling cache performance beyond LRU," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 225–236, IEEE, mar 2016.

[103] T. Zidenberg, I. Keslassy, and U. Weiser, "Multi-Amdahl : Optimal Resource Sharing with Multiple Program Execution Segments," pp. 1–7.

[104] T. Zidenberg, I. Keslassy, and U. Weiser, "MultiAmdahl: How Should I Divide My Heterogenous Chip?," *IEEE Computer Architecture Letters*, vol. 11, pp. 65–68, jul 2012.

[105] T. Zidenberg, I. Keslassy, and U. Weiser, "Optimal resource allocation with multi-amdahl," *Computer*, vol. 46, pp. 70–77, 2013.

[106] J. A. Debardelaben, V. K. Madisetti, and A. J. Gadient, "Incorporating cost modeling in embedded-system design," *IEEE Design and Test of Computers*, vol. 14, pp. 24–44, 1997.

[107] T.-V.-A. Nguyen, S. Bimonte, L. D'Orazio, and J. Darmont, "Cost Models for View Materialization in the Cloud," in *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, (New York, NY, USA), pp. 47–54, ACM, 2012.