LogCA: A High-Level Performance Model for Hardware Accelerators

Muhammad Shoaib Bin Altaf *
AMD Research
Advanced Micro Devices, Inc.
shoaib.altaf@amd.com

David A. Wood
Computer Sciences Department
University of Wisconsin-Madison
david@cs.wisc.edu

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 don't always live up to their expectations and may underperform 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 paper, we propose LogCA—a high-level performance model for hardware accelerators. LogCA helps both programmers and architects identify performance bounds and design bottlenecks early in the design cycle, and provide insight into which optimizations may alleviate these bottlenecks. We validate our model across a variety of kernels, ranging from sub-linear to super-linear complexities on both on-chip and off-chip accelerators. We also describe the utility of LogCA 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 LogCA's suggested optimizations. We argue that architects and programmers can use insights from these retrospective studies for improving future designs.

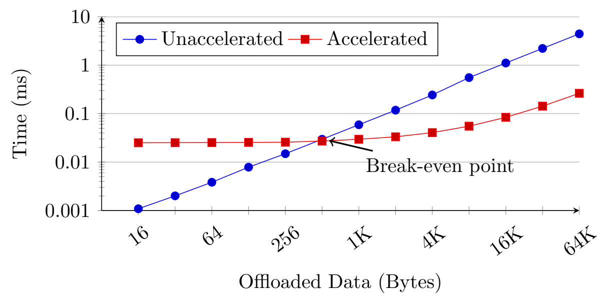
CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; • **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Hardware** → **Hardware accelerators**;

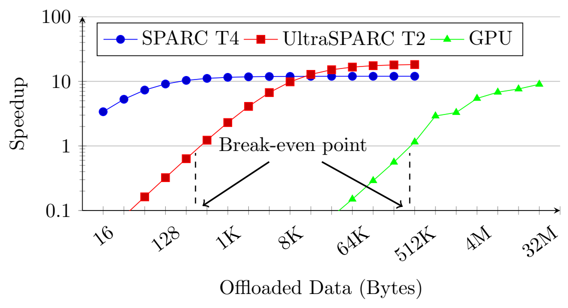
KEYWORDS

Analytical modeling, Performance, Accelerators, Heterogenous architectures

[*]This work was done while a Ph.D. student at Wisconsin.



(a) Execution time on UltraSPARC T2.



(b) Variation in speedup for different crypto accelerators.

Figure 1: Executing Advanced Encryption Standard (AES) [30].

ACM Reference format:
Muhammad Shoaib Bin Altaf and David A. Wood. 2017. LogCA: A High-Level Performance Model for Hardware Accelerators. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 14 pages.
https://doi.org/10.1145/3079856.3080216

1 INTRODUCTION

The failure of Dennard scaling [12, 49] over the last decade has inspired architects to introduce specialized functional units such as accelerators [6, 36]. These accelerators have shown considerable performance and energy improvement over general-purpose cores for some applications [14, 16, 23, 25, 26, 50, 51, 55]. 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 [13, 37, 45].

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 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. Figure 1 illustrates such an outcome for the crypto accelerator in UltraSPARC T2 running the Advanced Encryption Standard (AES) kernel [30].

Figure 1 provides two key observations. First, accelerators can under-perform as compared to 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 1-a). Second, different accelerators—while executing the same kernel—have different break-even points, e.g., SPARC T4 breaks even for smaller offloaded data while UltraSPARC T2 and GPU break even for large offloaded data (Figure 1-b).

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. Considering the AES encryption example, 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 result in similar performance to SPARC T4? Which optimizations are programmer dependent and which are architect dependent? What are the trade-offs in selecting one optimization over the other?

To answer these questions, programmer and architects can employ either complex or simple modeling techniques. Complex modeling techniques and full-system simulation [8, 42] can provide highly accurate performance estimates. Unfortunately, they often require 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 and provide key insights early in the design cycle that are useful for experts and non-experts alike [2, 5, 19, 27, 48, 54].

For an insightful model for hardware accelerators, this paper presents LogCA. LogCA derives its name from five key parameters (Table 1). These parameters characterize the communication latency (L) and overheads (o) of the accelerator interface, the granularity/size (g) of the offloaded data, the complexity (C) of the computation, and the accelerator's performance improvement (A) as compared to a general-purpose core.

LogCA is inspired by LogP [9], 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 [28]. With LogCA, our goal is to develop a simple model that supports the important implications (§2) of our analysis and use 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*.

LogCA helps programmers and architects reason about an accelerator by abstracting the underlying architecture. It provides insights about the accelerator's interface by exposing the design bounds and bottlenecks, and suggests optimizations to alleviate these bottlenecks. The visually identifiable optimization regions help both experts and non-experts to quantify the trade-offs in favoring one optimization over the other. 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.

We validate our model across on-chip and off-chip accelerators for a diverse set of kernels, ranging from sub-linear to super-linear complexities. We also demonstrate the utility of our model using two retrospective case studies (§5). In the first case study, we consider the evolution of interface in the cryptographic accelerator 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) [38] supported GPU. In both case studies, we show that the adopted design optimizations for these machines are similar to LogCA's suggested optimizations. We argue that architects and programmers can use insights from these retrospective studies for improving future designs.

This paper makes the following contributions:

- We propose a high-level visual performance model providing insights about the interface of hardware accelerators (§2).
- We formalize performance metrics for predicting the "right" amount of offloaded data (§2.2).
- Our model identifies the performance bounds and bottlenecks associated with an accelerator design (§3).
- We provide an answer to *what-if* questions for both programmers and architects at an early design stage (§3).
- We define various optimization regions and the potential gains associated with these regions (§3.2).
- We demonstrate the utility of our model on five different cryptographic accelerators and three different GPU architectures (§5).

2 THE LogCA MODEL

LogCA assumes an abstract system with three components (Figure 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 1.

Figure 2 (b) illustrates the overhead and latency model for an un-pipelined accelerator where computation 'i' is returned before requesting computation '$i+1$'. Figure 2 (b) also shows the breakdown of time for an algorithm on the host and accelerator. We assume that

Table 1: Description of the LogCA parameters.

Parameter	Symbol	Description	Units
Latency	L	Cycles to move data from the host to the accelerator across the 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

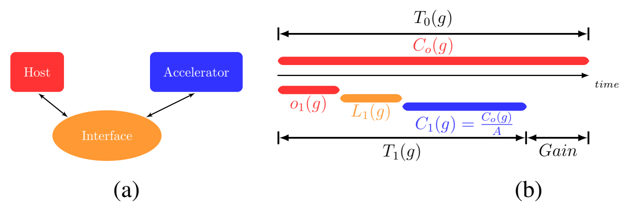


Figure 2: Top level description of the LogCA model (a) Shows the various components (b) Time-line for the computation performed on the host system (above) and on an accelerator (below)

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 $T_0(g) = C_0(g)$, 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 $T_1(g) = O_1(g) + L_1(g) + C_1(g)$, 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:

$$Speedup(g) = \frac{T_0(g)}{T_1(g)} = \frac{C_0(g)}{O_1(g) + L_1(g) + C_1(g)} \tag{1}$$

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 [4], 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 (§5).

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 independent 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 (§ 2.3). With these assumptions,

$$Speedup(g) = \frac{C * f(g)}{o + L + \frac{C*f(g)}{A}} = \frac{C * g^{\beta}}{o + L + \frac{C*g^{\beta}}{A}} \tag{2}$$

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 [47, 52]—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.

2.1 Effect of Granularity

A key aspect of LogCA is that it captures the effect of granularity on the accelerator’s speedup. Figure 3 shows this behavior, i.e., speedup increases with granularity and is bounded by the acceleration ‘A’. At

[1] not to be confused with operational intensity [54], which signifies operations performed per byte of DRAM traffic.

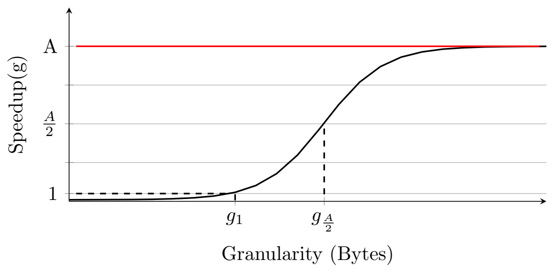


Figure 3: A graphical description of the performance metrics

one extreme, for large granularities, equation (2) becomes,

$$\lim_{g\to\infty} Speedup(g) = A \tag{3}$$

While for small granularities, equation (2) reduces to:

$$\lim_{g\to 0} Speedup(g) \simeq \frac{C}{o+L+\frac{C}{A}} < \frac{C}{o+L} \tag{4}$$

Equation (4) is simply *Amdahl's Law* [2] for accelerators, demonstrating the dominating effect of overheads at small granularities.

2.2 Performance Metrics

To help programmers decide *when* and *how much* computation to offload, we define two performance metrics. These metrics are inspired by the vector machine metrics N_v and $N_{1/2}$[18], 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 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 (2) for g_1 gives:

$$g_1 = \left[\left(\frac{A}{A-1}\right) * \left(\frac{o+L}{C}\right)\right]^{\frac{1}{\beta}} \tag{5}$$

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

For reducing g_1, the above implication guides an architect to invest 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 (2) for $g_{\frac{A}{2}}$ gives:

$$g_{\frac{A}{2}} = \left[A * \left(\frac{o+L}{C}\right)\right]^{\frac{1}{\beta}} \tag{6}$$

Using equation (5) and (6), g_1 and $g_{\frac{A}{2}}$ are related as:

$$g_{\frac{A}{2}} = (A-1)^{\frac{1}{\beta}} * g_1 \tag{7}$$

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 this effect is more pronounced 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 (7) 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 (7) 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 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 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.

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 (1) reduces to:

$$Speedup(g) = \frac{C * g^{\beta}}{o + L * g + \frac{C * g^{\beta}}{A}} \tag{8}$$

For large granularities, equation (8) reduces to:

$$\lim_{g\to\infty} Speedup(g) = \left(\frac{A}{\frac{A}{C*g^{\beta}} * (L*g) + 1}\right) < \frac{C}{L} * g^{\beta-1} \tag{9}$$

Unlike equation (3), 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 (8) reduces to:

$$\lim_{g \to 0} Speedup(g) \simeq A * \frac{C}{A*(o+L)+C} \tag{10}$$

Similar to equation (4), the above equation exposes the increasing effects of overheads at small granularities. Solving equation (8) for g_1 using Newton's method [53]:

$$g_1 = \frac{C*(\beta-1)*(A-1)+A*o}{C*\beta*(A-1)-A*L} \tag{11}$$

For a positive value of g_1, equation (11) 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.

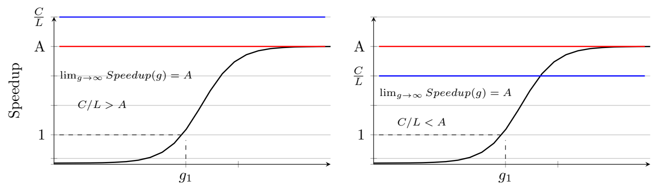
Similarly, solving equation (8), using Newton's method, for $g_{\frac{A}{2}}$ gives:

$$g_{\frac{A}{2}} = \frac{C*(\beta-1)+A*o}{C*\beta-A*L} \tag{12}$$

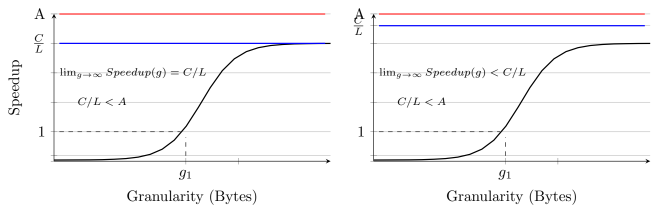
For a positive value of $g_{\frac{A}{2}}$, equation (12) must satisfy $\frac{C/A}{L} > \frac{1}{\beta}$. Thus, for achieving a speedup of $A/2$, $\frac{C}{L}$ should be at least 'A' for linear algorithms. However, for super-linear algorithms a speedup of $\frac{A}{2}$ can achieved at values of $\frac{C}{L}$ 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.



(a) Performance bounds for compute-bound kernels



(b) Performance bounds for latency-bound kernels

Figure 4: LogCA helps in visually identifying (a) compute and (b) latency bound kernels.

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 (8) for g_A gives:

$$g_A = \frac{C*(\beta-1)+A*o}{C*\beta-A*L} \tag{13}$$

Surprisingly, we find that the above equation is similar to equation (12), 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 (12), 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.

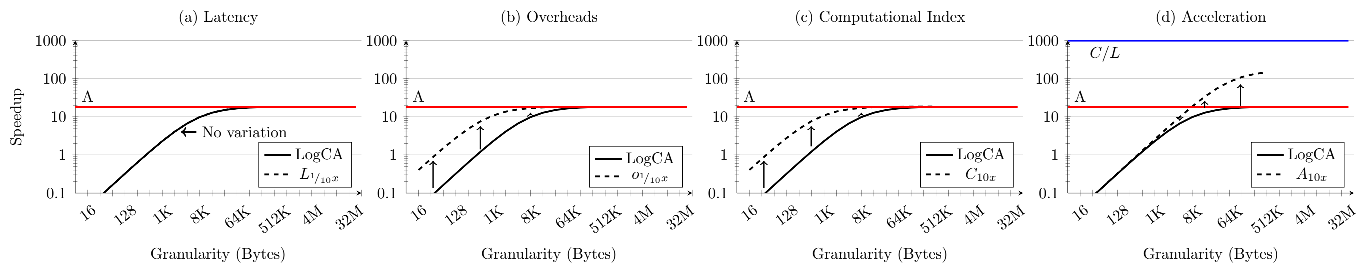


Figure 5: The effect on speedup of 10x improvement in each LogCA parameter. The base case is the speedup of AES [30] on UltraSPARC T2.

3 APPLICATIONS OF LogCA

In this section, we describe the utility of LogCA for visually identifying the performance bounds, design bottlenecks and possible optimizations to alleviate these bottlenecks.

3.1 Performance Bounds

Earlier, we have observed that the speedup is bounded by either acceleration (equation 3) or the product of computational intensity and $g^{\beta-1}$ (equation 9). 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 4-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 4-b illustrates these bounds for latency-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.

3.2 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 lest 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 crypto accelerator by varying its individual parameters [2] in Figure 5 (a)-(d).

[2]We elaborate our methodology for measuring LogCA parameters later (§ 4).

Figure 5 (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 5 (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 provide an opportunity for optimization.

Figure 5 (c) shows the effect of increasing the computational index. The results are similar to optimizing overheads in Figure 5 (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 5 (d) shows the variation in speedup with increasing peak 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 peak acceleration does not reduce the per byte setup cost, optimizing peak acceleration provides gains only at large granularities.

We group these individual sensitivity plots in Figure 6 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 peak 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 *overheads* occupy most of the lower granularity, suggesting opportunity for improving the interface. Similarly, the absence of the latency parameter suggests little benefits for optimizing latency.

We also add horizontal arrows to the optimization regions in Figure 6 to demarcate the start and end of granularity range for each

Table 2: Description of the Cryptographic accelerators

Crypto Accelerator	PCI Crypto	UltraSPARC T2	SPARC T3	SPARC T4	Sandy Bridge
Processor	AMD A8-3850	S2	S2	S3	Intel Core i7-2600
Frequency	2.9 GHz	1.16 GHz	1.65 GHz	3 GHz	3.4 GHz
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	Linux2.6.32-504

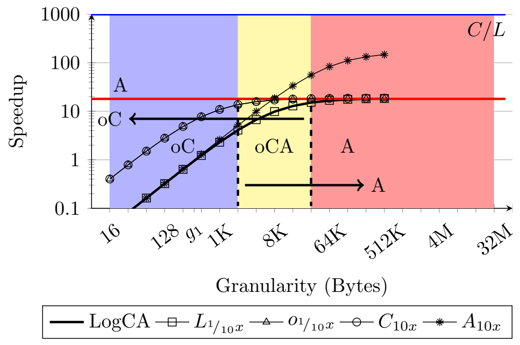


Figure 6: Optimization regions for UltraSPARC T2. The presence of a parameter in an optimization region indicates that it can at least provides 20% gains. The horizontal arrow indicates the cut-off granularity at which a parameter provides 20% gains.

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, high cut-off granularity of 32KB suggests high overheads and thus a potential for optimization.

4 EXPERIMENTAL METHODOLOGY

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

Our experimental setup comprises of on-chip and off-chip crypto accelerators (Table 2) and three different GPUs (Table 3). The on-chip crypto accelerators include cryptographic units on Sun/Oracle UltraSPARC T2 [40], SPARC T3 [35], SPARC T4 [41] and AES-NI (AES New Instruction) [15] on Sandy Bridge, whereas the off-chip accelerator is a Hifn 7955 chip connected through the PCIe bus [43]. The GPUs include a discrete NVIDIA GPU, an integrated AMD GPU (APU), and HSA supported integrated GPU.

For the on-chip crypto accelerators, each core in UltraSPARC T2 and SPARC T3 has a physically addressed crypto unit which requires privileged DMA calls. However, the crypto unit on SPARC T4 is integrated within the pipeline and does not require privileged DMA calls. SPARC T4 also provides non-privileged crypto instructions to access the crypto unit. Similar to SPARC T4, sandy bridge provides non-privileged crypto instruction—AESNI.

Considering the GPUs, 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 partitioned between host and GPU memory. This eliminates the PCIe bottleneck of data copying but it still requires copying data between memories. Unlike discrete GPU and APU, HSA supported GPU provides a unified and coherent view of the system memory. With the host and GPU share the same virtual address space, explicit copying of data between memories is not required.

Our workloads consist of encryption, hashing and GPU kernels. For encryption and hashing, we have used advanced encryption standard (AES) [30] and standard hashing algorithm (SHA) [31], respectively from OpenSSL [34]—an open source cryptography library. For GPU kernels, we use matrix multiplication, radix sort, FFT and binary search from AMD OpenCL SDK [1]. Table 4, 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. 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—unlike others—computations are performed on matrices and 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. We also observe that for the granularity range of 16B to 32MB, $\beta = 0.11$ provides a close approximation for $\log(g)$.

Table 3: Description of the GPUs

Platform	Discrete GPU	Integrated APU	AMD HSA
Name	Tesla C2070	Radeon HD 6550	Radeon R7
Architecture	Fermi	Beaver Creek	Kaveri
Cores	16	5	8
Compute Units	448	400	512
Clock Freq.	1.5 GHz	600 MHz	720 MHz
Peak FLOPS	1 T	480 G	856 G
Host:			
Processor	Intel	AMD	AMD
	Xeon E5520	A8-3850	A10-7850K
Frequency GHz	2.27	2.9	1.7

For calculating execution times, we have used Linux utilities on the crypto accelerators, whereas for the GPUs we have used NVIDIA and AMD OpenCL profilers to compute the setup, kernel and data transfer times, and we report the average of one hundred executions. For verifying the usage of crypto accelerators, we use built-in counters in UltraSPARC T2 and T3 [46]. SPARC T4, however, no longer

Table 4: Algorithmic complexity of various kernels with number of elements and granularity. The power of g represents β for each kernel.

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	$\mathcal{O}(\log n)$	$\mathcal{O}(g^{0.14})$

Table 5: Calculated values of LogCA Parameters.

		LogCA Parameters			
Device	**Benchmark**	**L** (cycles)	**o** (cycles)	**C** (cycles/B)	**A**
Discrete GPU	AES	3×10^3	2×10^8	174	30
	Radix Sort			290	
	GEMM			2	
	FFT			290	
	Binary Search			116	
APU	AES	15	4×10^8	174	7
	Radix Sort			290	
	GEMM			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 instr.	AES	4	111	32	12
	SHA		1,638	32	10
Sandy Bridge	AES	3	10	35	6

supports these counters, so we use Linux utilities to trace the execution of the crypto instructions [3]. We use these execution times to determine LogCA parameters. We calculate these parameters once and can be later used for different kernels on the same system.

For computational index and β, we profile the CPU code on the host 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. For overheads, we use the observation that for very small granularities the execution time for a kernel on an accelerator is dominated by the overheads, i.e., $\lim_{g\to 0} T_1(g) \simeq o$. For acceleration, we use different methods for the on-chip accelerators and GPUs. For on-chip accelerators, we calculate acceleration using equation (3), and the observation that the speedup curve flattens out and approaches acceleration for very large granularities. However, for the GPUs, we do not use equation (3) 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. We use the ratio of peak GFLOPs on CPU and GPU, i.e., $A = \frac{Peak\,GFLOP_{GPU}}{Peak\,GFLOP_{CPU}}$. Similar to acceleration, we use two different techniques for calculating latency. For the on-chip accelerators, we run micro-benchmarks and use execution time on host and accelerators. On the other hand, for the GPUs, we compute latency using peak memory bandwidth of the GPU. Similar to Meswani et al. [29], we use the following equation for measuring data copying time for the GPUs: $L = \frac{1}{BW_{peak}}$

Earlier, we develop our model using assumptions of granularity independent and dependent latencies. In our setup, we observe that the on-chip crypto accelerators and HSA-enabled GPU represent accelerators with granularity independent latency while the off-chip crypto accelerator and discrete GPU/APU represent the granularity dependent accelerators. For each accelerator we calculate the speedup and performance metrics using the respective equations (§2).

5 EVALUATION

In this section, we show that LogCA closely captures the behavior for both off and on-chip accelerators. We also list the calculate LogCA parameters in Table 5. To demonstrate the utility of our model, we also present two case studies. In these studies, we consider the evolution of interface in SUN/Oracle's crypto accelerators and three different GPU architectures. In both cases, we elaborate the design changes using the insights LogCA provides.

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

Figure 7 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 acceleration for on-chip accelerators. This observation supports earlier implication on the limits of speedup for granularity independent and dependent latencies in equation (3) and (9), respectively.

Figure 7 also shows that UltraSPARC T2 provides higher speedups than 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. However, 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 7 which help programmers and architects identify the complexity of the interface. For example, g_1 for crypto 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 (12) that $g_{\frac{A}{2}}$ for these designs does not exist.

We also observe that g_1 for the crypto-card connected through the PCIe bus does not exist, showing that this accelerator does not break-even even for large granularities. Figure 7 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,

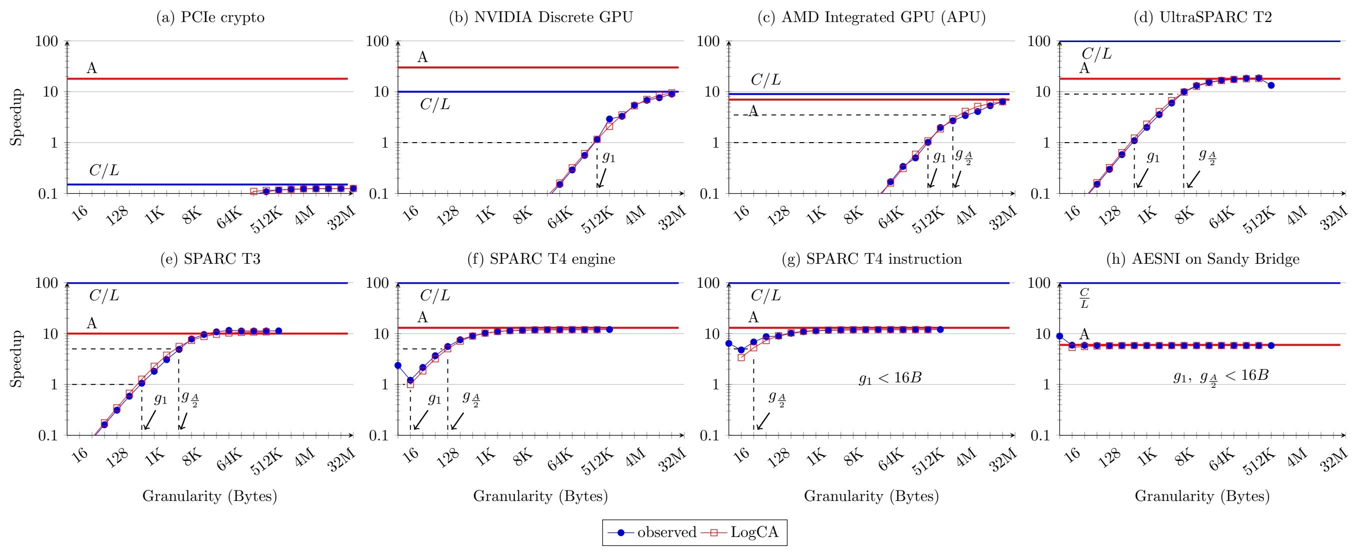


Figure 7: Speedup curve fittings plots comparing LogCA with the observed values of AES [30].

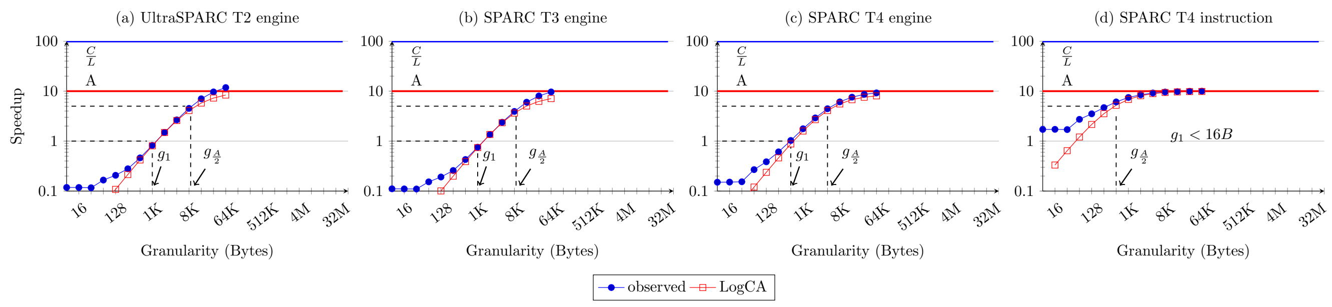


Figure 8: Speedup curve fittings plots comparing LogCA with the observed values of SHA256 [31]. LogCA starts following observed values after 64B.

APU spends considerable time in copying data from the host to device memory.

Figure 8 shows the curve fitting for SHA on various on-chip crypto 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 9-a shows the speedup curve fitting plots for Radix sort. We observe that LogCA does not follow observed values for smaller granularities on GPU. Despite this inaccuracy, LogCA accurately predicts g_1 and $g_{\frac{A}{2}}$. We also observe that $g_{\frac{A}{2}}$ for GPU is higher than APU, and this observation supports equation (7) that increasing acceleration increases $g_{\frac{A}{2}}$.

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

Figures 9-b and 9-c show the speedup curve fitting plots for super-complexity kernels on discrete GPU and APU. We observe that matrix multiplication with higher complexity ($\mathcal{O}(g^{1.7})$) achieves higher speedup than sort and FFT with lower complexities of $\mathcal{O}(g)$ and $\mathcal{O}(g^{1.2})$, respectively. This observation corroborates results from equation (9) 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 (12) that for $g_{\frac{A}{2}}$ to exist for FFT, $\frac{C}{L}$ should be greater than $\frac{A}{1.2}$. However, Figure 9-c shows that $\frac{C}{L}$ is smaller than $\frac{A}{1.2}$ for both GPU and APU.

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

Figure 9-d shows the curve fitting for binary search which is a sub-linear algorithm ($\beta = 0.14$). We make three observations. First, 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

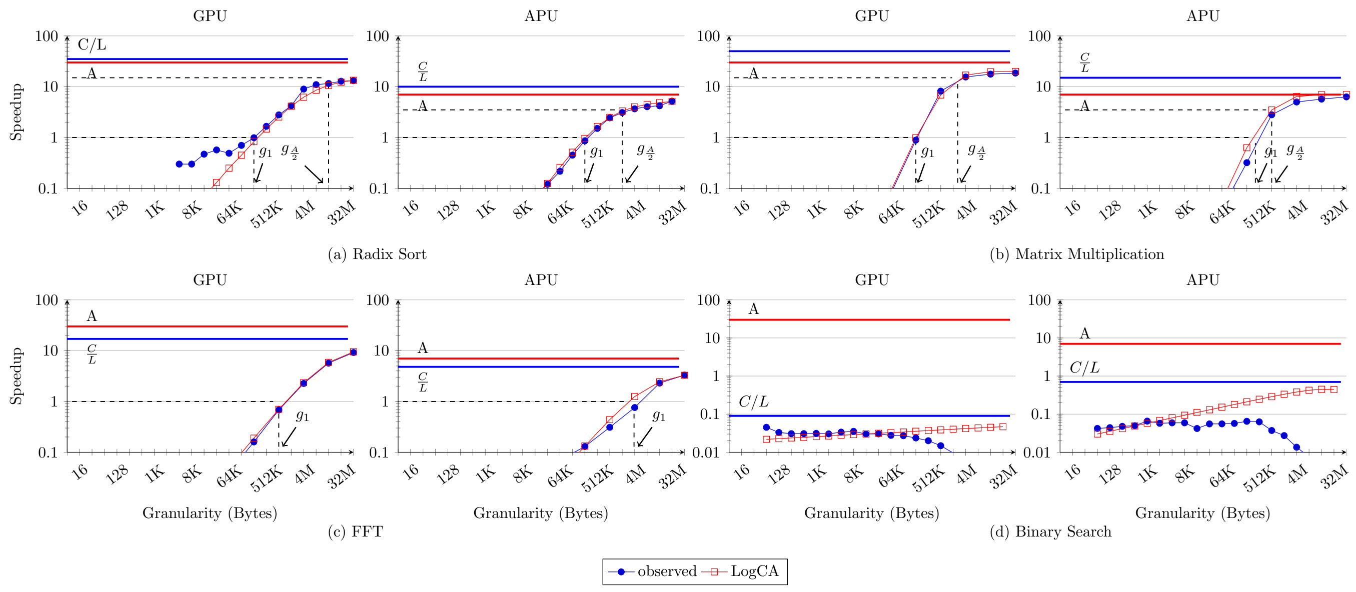


Figure 9: Speedup curve fittings plots comparing LogCA with the observed values of (a) Radix Sort (b) Matrix Multiplication (c) FFT and (d) Binary Search.

of $\beta = 0.14$, $\frac{C}{L}$ should be greater than 7 to provide any speedup. Second, 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 decreases. Third, 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.

5.4 Case Studies

Figure 10 shows the evolution of crypto accelerators in SPARC architectures from the off-chip accelerators in pre-Niagara (Figure 10 (a)) to accelerators integrated within the pipeline in SPARC T4 (Figure 10 (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 *overheads*—representing the complexity of an accelerator's interface—shrinks while the optimization regions with *acceleration* expand from Figure 10 (a-e). For example, for the off-chip crypto accelerator, the cut-off granularity for overheads is 256KB, whereas it is 128B for the SPARC T4, suggesting a much simpler interface.

Figure 10 (a) shows the optimization regions for the off-chip crypto accelerator connected through the PCIe bus. We note that *overheads* and *latencies* occupy most of the optimization regions, indicating high overhead OS calls and high-latency data copying over the PCIe bus as the bottlenecks.

Figure 10 (b) shows the optimization regions for UltraSPARC T2. The large cut-off granularity for overheads at 32KB suggests a complex interface, indicating 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.

Figure 10 (d) shows optimization regions for 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 cut-off granularity for acceleration now decreases to 32B, showing an increase in the opportunity for optimizing acceleration.

Figure 10 (e) shows optimization regions for crypto instructions on SPARC T4. We observe that unlike earlier designs, it has only two optimization regions and the speedup approaches the peak acceleration at a small granularity of 128B. In contrast, UltraSPARC T2 and SPARC T3 do not even provide any 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 which LogCA shows is due to the non-privileged instruction SPARC T4 uses to access the cryptographic unit, which is integrated within the pipeline.

Figure 11 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 matrix multiplication for all three architectures is *compute bound*

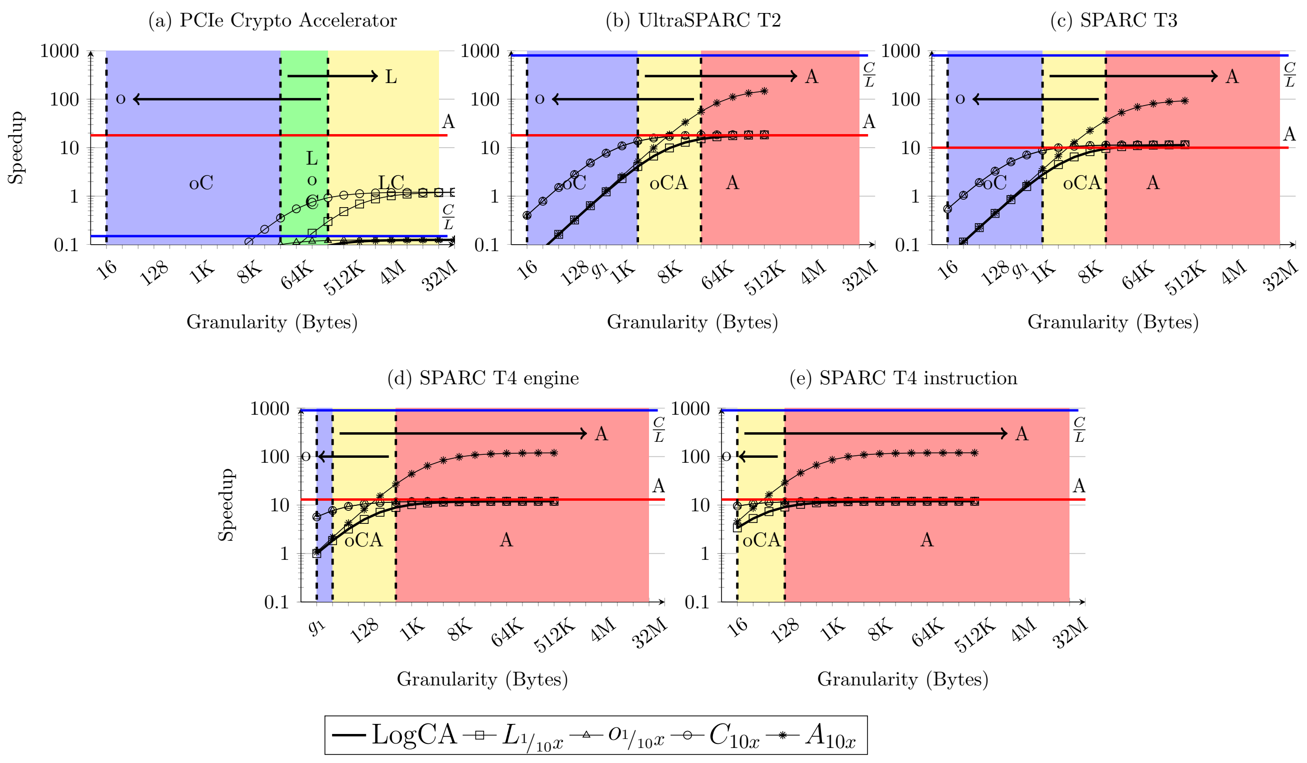


Figure 10: LogCA for performing Advanced Encryption Standard on various crypto accelerators. LogCA identifies the design bottlenecks through LogCA parameters in an optimization region. The bottlenecks which LogCA suggests in each design is optimized in the next design.

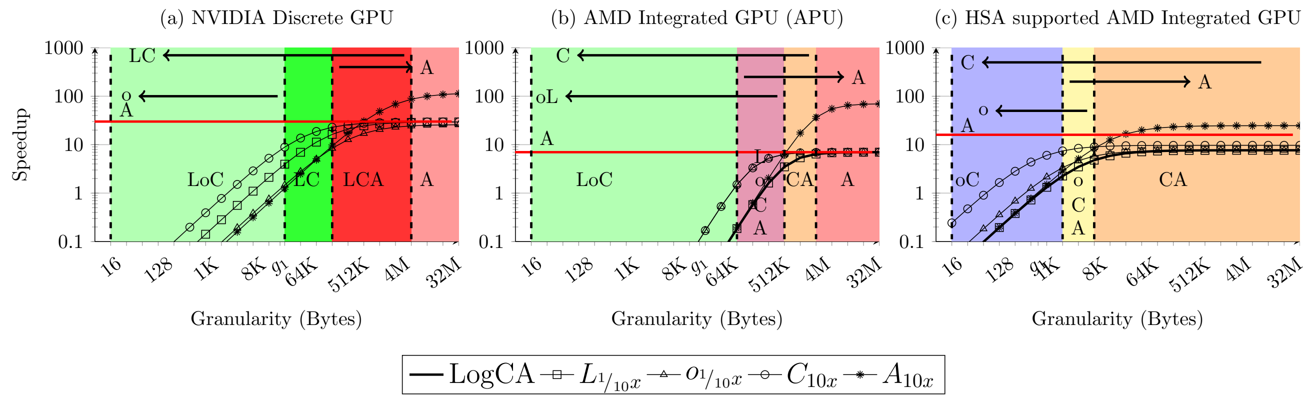


Figure 11: Various Optimization regions for matrix multiplication over a range of granularities on (a) NVIDIA discrete GPU, (b) AMD APU and (c) HSA Supported GPU.

(§3.1). We also observe that the computational index occupies most of the regions which signifies maximum optimization potential.

The discrete GPU has four optimization regions (Figure 11 (a)). Among these, latency 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 11 (b)). There are few notable differences as compared to the discrete GPU: The cut-off granularity for latency reduces to 512KB with the elimination of data copying over the PCIe bus, the overheads are still high suggesting high overhead OS calls to access the APU; with less aggressive cores, the cut-off granularity for acceleration reduces to 64KB, implying more optimization potential for acceleration.

Figure 11 (c) shows three optimization regions for the HSA enabled integrated 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 as compared to the discrete GPU and APU. We also observe that the cut-off granularity for acceleration drops to 2KB, suggesting higher potential for optimizing acceleration.

6 RELATED WORK

We compare and contrast our work with prior approaches. Lopez-Novoa et al. [28] provide a detailed survey of various accelerator modeling techniques. We broadly classify these techniques in two categories and discuss the most relevant work.

Analytical Models: There is a rich body of work exploring analytical models for performance prediction of accelerators. For some models, the motivation is to determine the future trend in heterogeneous architectures: Chung et al. [7], in a detailed study, predict the future landscape of heterogeneous computing; Hempstead et al. [17] propose an early-stage model, Navigo, that determines the fraction of area required for accelerators to maintain the traditional performance trend; Nilakantan et al. [32] propose to incorporate communication cost for early-stage model of accelerator-rich architectures. For others, the motivation is to determine the right amount of data to offload and the potential benefits associated with an accelerator [24].

Some analytical models are architecture specific. For example, a number of studies [20, 21, 44, 57] predict performance of GPU architectures. Hong et al. [20] present an analytical performance model for predicting execution time on GPUs. They later extend their model and develop an integrated power and performance model for the GPUs [21]. Song et al. [44] use a simple counter based approach to predict power and performance. Meswani et al. [29] explore such models for high performance applications. Daga et al. [11] analyze the effectiveness of Accelerated processing units (APU) over GPUs and describe the communication cost over the PCIe bus as a major bottleneck in exploiting the full potential of GPUs.

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.

Roofline Models: In terms of simplicity and motivation, our work closely matches the Roofline model [54]—a visual performance model for multi-core architectures. Roofline exposes bottlenecks for a kernel, and suggests several optimizations which programmers can use to fine tune the kernel on a given system.

A number of extensions of Roofline have been proposed [10, 22, 33, 56] and some of these extensions are architecture specific. For example, targeting GPUs [22], vector processors [39], and FPGAs [10, 56].

Despite the similarities, roofline and its extensions cannot be used for exposing design bottlenecks in an accelerator's interface. The primary goal of roofline models has been to help programmers and compiler writer while LogCA provides more insights for architects.

7 CONCLUSION AND FUTURE WORK

With the recent trend towards heterogeneous computing, we feel that the architecture community lacks a model to reason about the need of accelerators. In this respect, we propose LogCA—an insightful visual performance model for hardware accelerators. LogCA provides insights, early in the design stage, to both architects and programmers, and identifies performance bounds, exposes interface design bottlenecks and suggest optimizations to alleviate these bottlenecks. We have validated our model across a range of on-chip and off-chip accelerators, and have shown its utility using retrospective studies describing the evolution of accelerator's interface in these architectures.

The applicability of LogCA can be limited by our simplifying assumptions, and for more realistic analysis, we plan to overcome these limitations in our future work. For example, We also assume a single accelerator system and do not explicitly model contention among resources. Our model should handle multi-accelerator and pipelined scenarios. For fixed function accelerators, our design space is currently limited to encryption and hashing kernels. To overcome this, we are expanding our design space with compression and database accelerators in Oracle M7 processor. We also plan to complement LogCA with an energy model as energy efficiency is a prime design metric for accelerators.

ACKNOWLEDGEMENTS

We thank our anonymous reviewers, Arkaprava Basu and Tony Nowatzki for their insightful comments and feedback on the paper. Thanks to Mark Hill, Michael Swift, Wisconsin Computer Architecture Affiliates and other members of the Multifacet group for their valuable discussions. We also thank Brian Wilson at University of Wisconsin DoIT and Eric Sedlar at Oracle Labs for providing access to SPARC T3 and T4 servers, respectively. Also, thanks to Muhammad Umair Bin Altaf for his help in the formulation. This work is supported in part by the National Science Foundation (CNS-1302260, CCF-1438992, CCF-1533885, CCF- 1617824), Google, and the University of Wisconsin-Madison (Amar and Balindar Sohi Professorship in Computer Science). Wood has a significant financial interest in AMD and Google.

REFERENCES

[1] Advanced Micro Devices 2016. *APP SDK - A Complete Development Platform*. Advanced Micro Devices. http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk/.
[2] Gene M. Amdahl. 1967. 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)* (1967), 483. https://doi.org/10.1145/1465482.1465560
[3] Dan Anderson. 2012. *How to tell if SPARC T4 crypto is being used?* https://blogs.oracle.com/DanX/entry/how_to_tell_if_sparc.
[4] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John

Wawrzynek, David Wessel, and Katherine Yelick. 2009. A View of the Parallel Computing Landscape. *Commun. ACM* 52, 10 (oct 2009), 56–67. https://doi.org/10.1145/1562764.1562783

[5] Nathan Beckmann and Daniel Sanchez. 2016. Cache Calculus: Modeling Caches through Differential Equations. *Computer Architecture Letters* PP, 99 (2016), 1. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7366753$\delimiter"026E30F$npapers3://publication/doi/10.1109/LCA.2015.2512873

[6] C. Cascaval, S. Chatterjee, H. Franke, K. J. Gildea, and P. Pattnaik. 2010. A taxonomy of accelerator architectures and their programming models. *IBM Journal of Research and Development* 54 (2010), 5:1–5:10. https://doi.org/10.1147/JRD.2010.2059721

[7] Eric S. Chung, Peter a. Milder, James C. Hoe, and Ken Mai. 2010. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture* (dec 2010), 225–236. https://doi.org/10.1109/MICRO.2010.36

[8] Jason Cong, Zhenman Fang, Michael Gill, and Glenn Reinman. 2015. 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.

[9] D Culler, R Karp, D Patterson, and A Sahay. 1993. LogP: Towards a realistic model of parallel computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 1–12. http://dl.acm.org/citation.cfm?id=155333

[10] Bruno da Silva, An Braeken, Erik H D'Hollander, and Abdellah Touhafi. 2013. Performance Modeling for FPGAs: Extending the Roofline Model with High-level Synthesis Tools. *Int. J. Reconfig. Comput.* 2013 (jan 2013), 7:7—-7:7. https://doi.org/10.1155/2013/428078

[11] Mayank Daga, Ashwin M. Aji, and Wu-chun Feng. 2011. On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing. In *2011 Symposium on Application Accelerators in High-Performance Computing*. Ieee, 141–149. https://doi.org/10.1109/SAAHPC.2011.29

[12] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In *Proceeding of the 38th annual international symposium on Computer architecture - ISCA '11*. ACM Press, New York, New York, USA, 365. https://doi.org/10.1145/2000064.2000108

[13] H. Franke, J. Xenidis, C. Basso, B. M. Bass, S. S. Woodward, J. D. Brown, and C. L. Johnson. 2010. Introduction to the wire-speed processor and architecture. *IBM Journal of Research and Development* 54 (2010), 3:1–3:11. https://doi.org/10.1147/JRD.2009.2036980

[14] Venkatraman Govindaraju, Chen Han Ho, and Karthikeyan Sankaralingam. 2011. Dynamically specialized datapaths for energy efficient computing. In *Proceedings - International Symposium on High-Performance Computer Architecture*. 503–514. https://doi.org/10.1109/HPCA.2011.5749755

[15] Shay Gueron. 2012. *Intel Advanced Encryption Standard (AES) Instructions Set*. Technical Report. Intel Corporation. https://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf

[16] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. 2010. Understanding sources of inefficiency in general-purpose chips. *Proceedings of the 37th annual international symposium on Computer architecture - ISCA '10* (2010), 37. https://doi.org/10.1145/1815961.1815968

[17] Mark Hempstead, Gu-Yeon Wei, and David Brooks. 2009. Navigo: An early-stage model to study power-constrained architectures and specialization. In *ISCA Workshop on Modeling, Benchmarking, and Simulations (MoBS)*.

[18] John L Hennessy and David A Patterson. 2006. *Computer Architecture, Fourth Edition: A Quantitative Approach*. 704 pages. https://doi.org/10.1.1.115.1881

[19] Mark D Hill and Michael R Marty. 2008. Amdahl's Law in the Multicore Era. *Computer* 41, 7 (jul 2008), 33–38. https://doi.org/10.1109/MC.2008.209

[20] Sunpyo Hong and Hyesoon Kim. 2009. 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. 152–163. https://doi.org/10.1145/1555815.1555775

[21] Sunpyo Hong and Hyesoon Kim. 2010. An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, Vol. 38. 280—-289. https://doi.org/10.1145/1816038.1815998

[22] Haipeng Jia, Yunquan Zhang, Guoping Long, Jianliang Xu, Shengen Yan, and Yan Li. 2012. GPURoofline: A Model for Guiding Performance Optimizations on GPUs. In *Proceedings of the 18th International Conference on Parallel Processing (Euro-Par'12)*. Springer-Verlag, Berlin, Heidelberg, 920–932. https://doi.org/10.1007/978-3-642-32820-6_90

[23] Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. 2013. Meet the Walkers: Accelerating Index Traversals for In-memory Databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, USA, 468–479. https://doi.org/10.1145/2540708.2540748

[24] Karthik Kumar, Jibang Liu, Yung Hsiang Lu, and Bharat Bhargava. 2013. A survey of computation offloading for mobile systems. *Mobile Networks and Applications* 18 (2013), 129–140. https://doi.org/10.1007/s11036-012-0368-0

[25] Snehasish Kumar, Naveen Vedula, Arrvindh Shriraman, and Vijayalakshmi Srinivasan. 2015. DASX: Hardware Accelerator for Software Data Structures. In *Proceedings of the 29th ACM on International Conference on Supercomputing (ICS '15)*. ACM, New York, NY, USA, 361–372. https://doi.org/10.1145/2751205.2751231

[26] Maysam Lavasani, Hari Angepat, and Derek Chiou. 2014. An FPGA-based In-Line Accelerator for Memcached. *IEEE Comput. Archit. Lett.* 13, 2 (jul 2014), 57–60. https://doi.org/10.1109/L-CA.2013.17

[27] John D C Little and Stephen C Graves. 2008. Little's law. In *Building intuition*. Springer, 81–100.

[28] U Lopez-Novoa, A Mendiburu, and J Miguel-Alonso. 2015. A Survey of Performance Modeling and Simulation Techniques for Accelerator-Based Computing. *Parallel and Distributed Systems, IEEE Transactions on* 26, 1 (jan 2015), 272–281. https://doi.org/10.1109/TPDS.2014.2308216

[29] M. R. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole. 2013. Modeling and predicting performance of high performance computing applications on hardware accelerators. *International Journal of High Performance Computing Applications* 27 (2013), 89–108. https://doi.org/10.1177/1094342012468180

[30] National Institute of Standards and Technology 2001. *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.FIPS.197.

[31] National Institute of Standards and Technology 2008. *Secure Hash Standard*. National Institute of Standards and Technology. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[32] S Nilakantan, S Battle, and M Hempstead. 2013. Metrics for Early-Stage Modeling of Many-Accelerator Architectures. *Computer Architecture Letters* 12, 1 (jan 2013), 25–28. https://doi.org/10.1109/L-CA.2012.9

[33] Cedric Nugteren and Henk Corporaal. 2012. 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*. ACM, 203—-212.

[34] OpenSSL Software Foundation 2015. *OpenSSL, Cryptography and SSL/TLS Toolkit*. OpenSSL Software Foundation. https://openssl.org.

[35] Sanjay Patel. 2009. Sun's Next-Generation Multithreaded Processor: Rainbow Falls. In *21st Hot Chip Symposium*. http://www.hotchips.org/wp-content/uploads/hc

[36] Sanjay Patel and Wen-mei W. Hwu. 2008. Accelerator Architectures. *IEEE Micro* 28, 4 (jul 2008), 4–12. https://doi.org/10.1109/MM.2008.50

[37] Stephen Phillips. 2014. M7: Next Generation SPARC. In *26th Hot Chip Symposium*.

[38] Phil Rogers. 2013. Heterogeneous system architecture overview. In *Hot Chips*.

[39] Yoshiei Sato, Ryuichi Nagaoka, Akihiro Musa, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, and Hiroaki Kobayashi. 2009. Performance tuning and analysis of future vector processors based on the roofline model. *Proceedings of the 10th MEDEA workshop on MEmory performance DEaling with Applications, systems and architecture - MEDEA '09* (2009), 7. https://doi.org/10.1145/1621960.1621962

[40] 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. 2007. UltraSPARC T2: A highly-treaded, power-efficient, SPARC SOC. In *Solid-State Circuits Conference, 2007. ASSCC '07. IEEE Asian*. 22–25. https://doi.org/10.1109/ASSCC.2007.4425786

[41] Manish Shah, Robert Golla, Gregory Grohoski, Paul Jordan, Jama Barreh, Jeffrey Brooks, Mark Greenberg, Gideon Levinsky, Mark Luttrell, Christopher Olson, Zeid Samoail, Matt Smittle, and Thomas Ziaja. 2012. Sparc T4: A dynamically threaded server-on-a-chip. *IEEE Micro* 32 (2012), 8–19. https://doi.org/10.1109/MM.2012.1

[42] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *International Symposium on Computer Architecture (ISCA)*.

[43] Soekris Engineering 2016. *vpn 1401, for Std. PCI-sockets*. Soekris Engineering. http://soekris.com/products/vpn-1401.html.

[44] Shuaiwen Song, Chunyi Su, Barry Rountree, and Kirk W. Cameron. 2013. 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*. 673–686. https://doi.org/10.1109/IPDPS.2013.73

[45] Jeff Stuecheli. 2013. POWER8. In *25th Hot Chip Symposium*.

[46] Ning Sun and Chi-Chang Lin. 2007. *Using the Cryptographic Accelerators in the UltraSPARC T1 and T2 processors*. Technical Report. http://www.oracle.com/technetwork/server-storage/solaris/documentation/819-5782-150147.pdf

[47] S. Tabik, G. Ortega, and E. M. Garzón. 2014. Performance evaluation of kernel fusion BLAS routines on the GPU: iterative solvers as case study. *The Journal of Supercomputing* 70, 2 (nov 2014), 577–587. https://doi.org/10.1007/s11227-014-1102-4

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

[49] MB Taylor. 2012. Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*. 1131–1136. https://doi.org/10.1145/2228360.2228567

[50] G Venkatesh, J Sampson, N Goulding, S Garcia, V Bryksin, J Lugo-Martinez, S Swanson, and M B Taylor. 2010. Conservation cores: Reducing the energy of mature computations. In *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*. 205–218. https://doi.org/10.1145/1736020.1736044

[51] Ganesh Venkatesh, Jack Sampson, Nathan Goulding-Hotta, Sravanthi Kota Venkata, Michael Bedford Taylor, and Steven Swanson. 2011. 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*. 163. https://doi.org/10.1145/2155620.2155640

[52] Guibin Wang, Yisong Lin, and Wei Yi. 2010. 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)*. 344–350. https://doi.org/10.1109/GreenCom-CPSCom.2010.102

[53] Eric W. Weisstein. 2015. *Newton's Method*. From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/NewtonsMethod.html.

[54] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52 (2009), 65–76. https://doi.org/10.1145/1498765.1498785

[55] Lisa Wu, Andrea Lottarini, Timothy K Paine, Martha A Kim, and Kenneth A Ross. 2014. 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)*. ACM, New York, NY, USA, 255–268. https://doi.org/10.1145/2541940.2541961

[56] Moein Pahlavan Yali. 2014. *FPGA-Roofline: An Insightful Model for FPGA-based Hardware Accelerators in Modern Embedded Systems*. Ph.D. Dissertation. Virginia Polytechnic Institute and State University.

[57] Yao Zhang and John D. Owens. 2011. A quantitative performance analysis model for GPU architectures. In *Proceedings - International Symposium on High-Performance Computer Architecture*. 382–393. https://doi.org/10.1109/HPCA.2011.5749745