Low Power GPGPU Computation with Imprecise Hardware

A Thesis

Presented to the Faculty of

School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

in

Computer Engineering

by

Hang Zhang

December 2014

Abstract

Massively parallel computation in GPUs significantly boosts the performance of compute-intensive scientific and engineering applications due to improved floating point performance, but creates power and thermal issues that could limit further performance scaling. The high power consumption for general purpose computation in GPU (GPGPU) majorly comes from two sources: the memory system and the computation units, which can reach as much as 250W for high end GPUs due in large part to the sheer complexity of the IEEE-754 compliant hardware and its high activity factors during floating point operations. This thesis focuses on lowering GPGPU computation power consumption by using the imprecise hardware (IHW) methodologies. Imprecise Hardware is an emerging design paradigm that targets at high power computation units and seeks to improve power, area, latency, and related nonfunctional metrics through the use of quality tradeoffs in application domains that can tolerate relaxed correctness specifications. Power saving techniques such as dynamic voltage frequency scaling (DVFS) and power gating are capable of effectively reducing power consumption, but these approaches essentially trade-off between power and performance. The use of imprecise hardware (IHW) effectively shifts the design paradigm from a power-performance tradeoff to a power-quality tradeoff with little or no degradation in performance. The use of IHW is orthogonal to DVFS, power gating, and other hardware or software power optimization techniques, and can be combined with these techniques to further reduce the power consumption for GPGPU computation.

This thesis demonstrates significant GPGPU power savings by relaxing application accuracy requirements and enabling the use of low power imprecise floating point and special function units, which are typically the most frequently exercised and the most power hungry arithmetic components in many high performance (high power) GPGPU computations. A set of novel imprecise floating point arithmetic units is presented and their non-functional metrics are obtained through post-layout SPICE level simulations. GPU performance simulator GPGPU-Sim and power-energy model GPUWattch are used to estimate the impacts of IHW units on output quality as well as the GPU system-level power consumption, providing a power-quality tradeoff framework for application-specific power optimizations on GPGPU platform. Experimental results in a 45nm process show up to 32% power savings with negligible impacts on output quality.

Acknowledgement

First of all, I greatly appreciate the significant guidance, patience, and help from my thesis advisor Professor John Lach. It would be almost impossible to achieve this work without all the effort and support from Professor Lach. Before starting my graduate study at UVa, I barely understand how to conduct research, how to write a paper, and how to think critically when reading papers and solving research problems. Professor Lach provided me a great opportunity to explore a new and exciting research area, the imprecise hardware design and methodologies. And it is also Professor Lach's encouragement that lead me on to focusing the work on the GPU platform, making a new way to tackle a challenging but interesting problem: reducing GPU power consumption by using imprecise hardware. The work turns out with two major conference publications, with one of them in one of the most prestigious conferences. The advices, encouragement, and guidance from Professor Lach are so valuable and will keep helping me along the rest of my career and life.

I would also like to thank my committee members: Professor Mircea Stan and Professor Gabriel Robins. Their time and effort on my work has also been tremendous. I want to especially thank Professor Mircea Stan for his advice on a lot of other projects I worked on as well. Those experiences are so valuable and greatly helped the way I work on research problems, and provided with opportunities to learn different skills and come up with new ideas.

I would also like to thank my colleagues such as Xinfei Guo, Wei Zhang, Runjie Zhang, He Qi, and others. They made my life here fun and exciting. And I really appreciate the friendship.

Last but not least, I would like to thank my parents for all these years' unconditional support and love. You have always been the greatest person in my life.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 An Opportunity to Relax Quality for Lower Power in GPGPU Computation

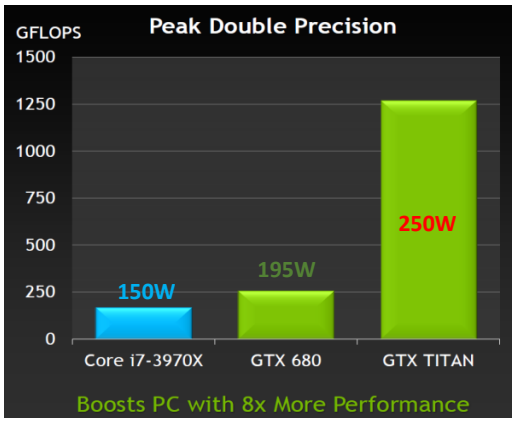


Figure 1 Floating Point Performance Improvement between high end Intel CPU and NVIDIA GPUs

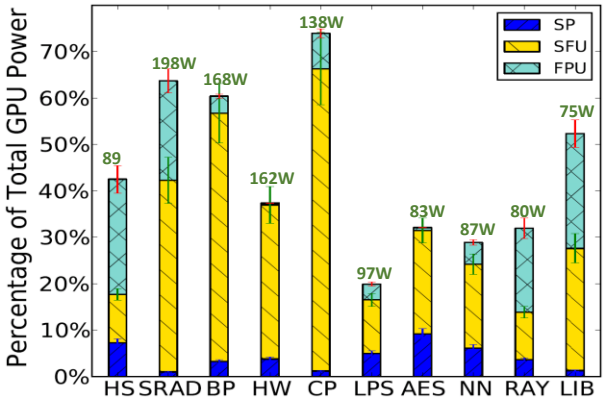


Figure 2 Arithmetic power consumption for compute intensive benchmarks from Rodinia and ISPASS2009 (obtained from GPUWattch with GTX480[17]**)**

Many scientific and engineering applications can achieve significant gains in performance by migrating from conventional multi-core CPUs to general purpose GPUs (GPGPU). This performance gain comes from the massively parallel computation units or processing cores with shared front end in a SIMD architecture. As significant amount of hardware resources are dedicated computation units for parallel computation, the performance gain can be easily scaled up for more than an order of magnitude. Figure 1 shows the significant floating point performance (in peak double precision throughput) increase in terms of GFLOPS between a high-end CPU and two generations of GPUs. However, the price for this performance boost comes as a dramatic increase in the power consumption, reaching as much as 250 W for some high-end GPUs. In addition, as technology scales, GPUs are inevitably hitting the infamous “power wall”, which prevents

GPU vendors from integrating more compute cores and hardware resources or increasing operating frequency to further improve the performance.

GPU architecture consists of a large number of homogeneous processing cores that can perform concurrent thread execution, and unlike CPUs, contains significantly more arithmetic circuits per unit area. For instance, NVIDIA's Fermi architecture contains up to 512 general purpose CUDA cores, with each core consisting of an ALU and FPU. These homogeneous cores are grouped into 16 streaming multiprocessors (SM) [1]. Each SM also contains schedule, dispatch, load/store, and four special function units (SFU) for performing fast floating point elementary functions. Our preliminary study (results in Figure 2) revealed that the portion of power consumed by floating point arithmetic operations in compute-intensive benchmarks can reach more than 70% of the total GPU power consumption. As shown in Figure 2, the most power hungry components in GPU computation units comes from floating point computations in floating point units (FPUs) and special function units (SFUs), while the integer units consumes less than 10% of total GPU power consumption.

The floating point number system in IEEE-754 format are widely used in most scientific and engineering applications. It provides high precision, a wide dynamic range of real values, and a much simplified programming model compared to the fixed point system. The performance of floating point operations in a system, often represented as FLOPS (floating point operations per second), has become one of the most important performance metrics in modern computing system such as multi-core CPU processors, GPUs, and even supercomputers. For example, a high end Intel Core i7 processor can reach around 187 GFLOPS with a designed power consumption of 150W [2]. NVIDIA's Kepler

GK110 architecture contains 192 single precision and 64 double precision IEEE-754 compliant FPUs for each streaming multiprocessor (SMX, 15 SMX in total), capable of achieving 1 TFLOPS double precision throughput, however, with a high power consumption of 250W [3]. As GPU architecture contains significantly more floating point arithmetic system per unit area with shared front end compared to CPU, the floating point system performance are significantly higher but also draws significant amount of power consumption.

Many scientific and engineering applications contain significant amount of floating point instructions, as it is desirable to have a wide range of representable values, where they would all be represented as the same two extreme values (maximum or minimum) in a fixed point number system. However, it is often neglected that the accuracy requirements of these applications could be relaxed because they either don't require a 100% accurate result, or the results are acceptable or good enough at certain levels of estimation. For example, voice recognition applications are based on approximation algorithms, and therefore inherently error tolerant. Image processing and communication algorithms usually have to deal with a lot of noises which makes the results probabilistic. Some other applications only cares about the relative temporal or spatial differences, therefore an erroneous shift in the results have minimum effect on the delta function of interests. In addition, the IEEE-754 floating point number system is inherently imprecise. And the IEEE-754 standard specifies the requirements of a rounding unit with different rounding modes that results in a power hungry component in a floating point unit. The absolute error of a floating point number grows exponentially as the exponent of the represented value gets larger and larger. These facts present a unique opportunity to utilize power-quality

trade-off design methodologies that approximates the individual floating point arithmetic result at the instruction and micro-architecture level for significant lower power consumptions.

1.2 The Promising Paradigm of Imprecise Hardware

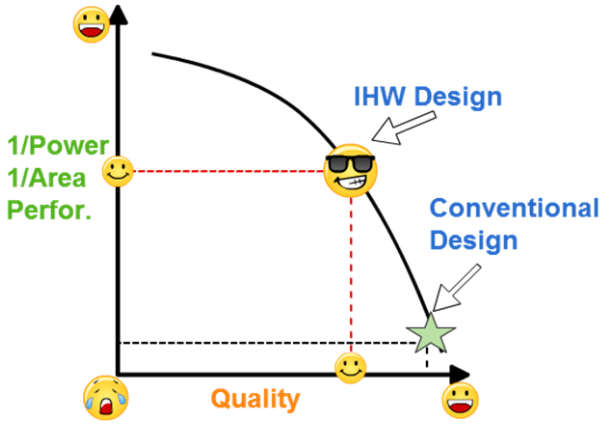


Figure 3 Imprecise Hardware (IHW) Conceptual power-quality trade-off design space

Imprecise hardware (IHW) is an emerging design methodology that seeks to improve power, area, latency, and related nonfunctional metrics through the use of quality tradeoffs in application domains that can tolerate relaxed correctness specifications. Conventional arithmetic designs often try to achieve "100%" accuracy for general computing purposes which result in high power, area, and very limited performance. Admittedly, many applications in the scientific and engineering do require extremely high accuracies, such as various models in financial engineering where a small error would

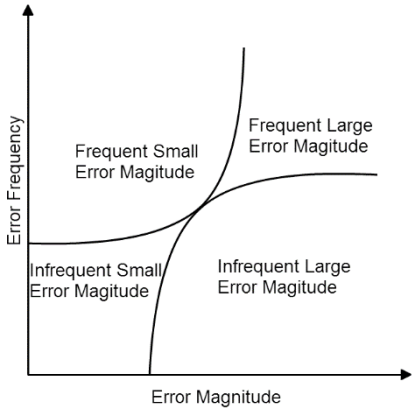


Figure 4 Imprecise hardware taxonomy based on error frequency and magnitude

results in millions of dollars difference. For other applications where there are only limited power and area budget, once such strict requirements on computation accuracy are relaxed, one could find significant benefits in power, performance, and area (P.P.A) measures with negligible and acceptable quality loss. As shown in Figure 3, the imprecise hardware design paradigm allows a reasonable amount of quality sacrifices, but in return it gains significant improvement on non-functional metrics in power, area, and performance. The quality loss sometime can even be negligible to the application, giving imprecise hardware a significant advantage comparing against the conventional "precise" hardware. Figure 5 shows an example studied in previous work by [4], the proposed IHW adder was used in a JPEG decompression algorithm, the resulting image in the middle suffered negligible quality loss, while achieving 24% energy delay product (EDP) savings.

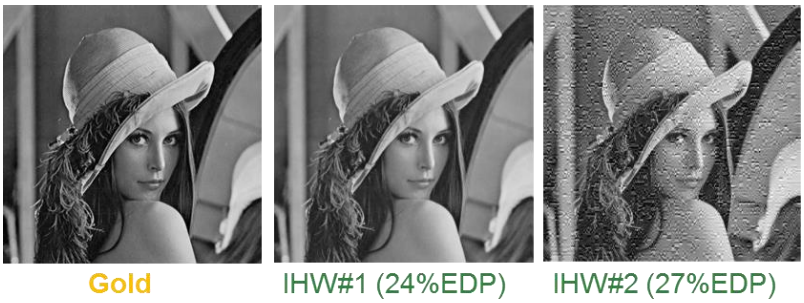


Figure 5 IHW used in JPEG decompression algorithm that results in minimal quality loss but significant EDP gain IHW categorized by error frequency and magnitude [4]

However, achieving significant impacts on system level nonfunctional metrics with IHW is difficult. Nonfunctional metric improvements are limited by the rate at which operations that could be executed on IHW are found in the application. Even if arithmetic operations constitute the majority of runtime cycles in an application, the upper bound of the system-level impact is set by the collective power contributions of the executing hardware modules. While the benefits of applying IHW design methodology are most readily realized in specialized ASICs, the heterogeneity of CPU architectures ultimately limits the potential success of IHW in these general-purpose platforms. However, the massively parallel homogeneous computation cores in GPU provided a tremendous opportunity as it contains significantly higher arithmetic units per unit area, and the arithmetic computation is a significant power consumer in the GPU architecture.

There are different design techniques and design spaces for IHW depending on its error characteristics. Figure 4 shows a taxonomy of IHW from a quality perspective based on its relationship between error magnitude and error frequency. In general, an IHW with its error characteristics can fall into one of these four categories: Frequent Small Magnitude (FSM), Frequent Large Magnitude (FLM), Infrequent Small Magnitude (ISM), and

Frequent Large Magnitude (FLM). Among these four categories, FLM usually produces the worst quality, but it tends to be the most powerful in the power saving domain. Infrequent small and large magnitude IHW designs has the characteristics that the error tends to be sparsely distributed depending on the input distribution, such as the work in [5]. This thesis focuses on floating point designs where errors happen frequently due to the bit-truncation wise techniques, and the error magnitude can be both small and large. Huang's dissertation contains a more complete taxonomy on IHW based on various characteristics [6].

Conventional IHW design and methodology explorations have been focused on integer hardware[4][5]. However, many floating point applications in multimedia and scientific computing can tolerate imprecise computations, and the large power consumption of floating point multipliers presents an opportunity for significant power savings. In fact, many floating point applications can operate with either double or single precision and even with bit truncations [8]. This provides a significant opportunity for applying imprecise hardware methodologies to make floating point operations imprecise and even accuracy-configurable so that dramatic power reductions could be achieved by adopting the lowest accuracy configuration that still meets the application specific quality constraint with no loss in the wide range of representable real values which is one of the most important benefits from using a floating point number system.

1.3 Major Contributions

The main contributions of this thesis are summarized as follows:

- A set of novel imprecise FPUs and SFUs with improved nonfunctional metrics is presented and evaluated in a 45 nm process.
- A methodology for error analysis and characterization is presented and is applied to guide the selection of imprecise components based on application specific quality requirements.
- A power-quality evaluation framework is presented to efficiently determine the quality impact and estimate the system level power savings for compute-intensive GPGPU applications. Results of the developed evaluation framework are presented for several compute-intensive scientific applications, demonstrating power-quality tradeoffs with IHW components.
- A novel design of low-power accuracy-configurable floating point multiplier is proposed and is capable of achieving significant same-delay power reductions for single and double precisions compared to the IEEE-754 compliant counterparts.
- An analysis demonstrates that it is suboptimal to directly apply bit truncation schemes, which is the conventional technique in floating point multiplication, to the power hungry mantissa multiplication in the power-quality trade-off design space.
- Studies on a set of CPU and GPU Benchmarks that demonstrate the significant opportunities to leverage the proposed imprecise floating point multiplier for dramatic power savings with minimal application-level quality impact.

Chapter 2 Related Work

Various studies on imprecise arithmetic designs [4] [5][7][9][10] have shown that significant power savings and performance improvement can be achieved over traditional "precise" designs with acceptable impact on output quality. In [5], the authors proposed a variable latency speculative adder called Almost Correct Adder, it assumes that the carry propagation chain along the datapath of a Kogge-Stone adder is less than a certain threshold. The adder therefore is probabilistic in nature and can achieve $\log(\log^n)$ performance compared to $\log^n$ of traditional high performance Kogge-Stone adder, where n is the number of bits in operands. In [4], the authors proposed an error balancing adder with dynamic error mitigation capabilities to push the Pareto optimal surface in the energy-quality design paradigm. The proposed adder is studied with a common JPEG image decompression algorithm and demonstrates minimum quality degradation with 24% energy delay product (EDP) savings. An imprecise integer adder was also proposed in [7] such that the design utilizes multiple copies of smaller adders to configure different datapath with speculated carry-in values. This accuracy configurable integer adder achieves approximately 30% of power reduction compared to the conventional pipelined adder.

The emergence of various imprecise integer units and their novel designs significantly stimulated the research in the imprecise hardware designs and methodologies. Work in [11][12] provides a systematic methodological study of integer imprecise arithmetic units in the quality-efficiency design paradigm. The methodology can also be applied to floating point imprecise arithmetic units. The error modeling framework for imprecise arithmetic circuits proposed in [13] provides a foundation for the error

characterization framework in this work. However, a limited amount of work has been focused on imprecise floating point or special function units where power consumptions can be significantly larger than integer units. Wires et al. [14] proposed an imprecise floating point multiplier, which truncates the less significant bits (LSBs) in the multiplication matrix, resulting in a less than 1 unit in the last place (ULP) error and 37% power savings. A low power probabilistic floating point multiplier that uses combined voltage scaling and LSBs truncation techniques achieved 31% energy savings with negligible quality degradation in the studied RayTracing application [15]. A bit width reduction technique was used for a floating point multiplier in [8]. Even though significant power savings were achieved in these studies, we argue that more aggressive power savings are necessary to have a big impact at the system level. While some attempts have been made on low power floating point adder, divider, and special function units [8][14][15], little effort has been spent in the power-quality tradeoff design paradigm.

At the system level, previous work has shown significant power/energy savings from using IHW on DSPs, where arithmetic power/energy consumption dominates, such as the work presented in [15]-[16]. An energy-precision tradeoff has been studied for mobile graphic processing units, which demonstrates over 23% energy savings with acceptable result accuracy [17]. The energy consumption is lowered by reducing the precision of arithmetic operations. However, the work is limited to the vertex transformation stage in a multi-stage graphics processor pipeline and was not applied to the more promising GPGPU paradigm.

This work uses linear approximation and algorithmic and logic level simplification and considers all floating point arithmetic operations as candidates for approximate

operations. However, we justify this approach by arguing that it constitutes the best-case nonfunctional/worst-case functional tradeoff to facilitate studying the limits of the impacts of approximate computing in GPU architectures. This work is the first to consider the tradeoffs of relaxed precision floating point operations in the GPGPU paradigm and to evaluate the limits of system-level impacts. While some assumptions made in this work may limit its practical implementation, its results should be considered as a set of design goals to make a significant impact on system level metrics in future GPU designs.

There are a few previous works that leverage quality as a design knob for reducing power consumptions in floating point multipliers [6][8][9][10]. Most of these works use intuitive bit truncations and voltage scaling that results in a relatively large percentage of power/energy savings. Gupta et al. [8] proposed a low power probabilistic floating point multiplier that uses combined voltage scaling and intuitive bit truncation in less significant bits (LSBs) of the mantissas, achieving 31% energy savings. The probabilistic nature, however, makes the multiplier less predictable in terms of power-quality trade-off. Wires et al. [9] also used intuitive bit truncation schemes to directly truncate LSBs in the mantissa multiplication matrix, resulting in a less than 1 unit in the last place (ULP) error and 37% power savings. Energy quality trade-off using intuitive bit truncations was studied in [6] for mobile graphic applications, achieving up to 36% energy savings. The work in [5] explored bit truncation schemes more aggressively in the mantissa multiplication block and achieved up to 66% power savings. It also studied a variety of floating point applications that suffers no quality degradation when up to 12 bits of mantissas were truncated. This work presents even more aggressive power reduction techniques that can achieve order of magnitude power reductions with graceful quality degradations.

Despite the emergence of various imprecise and accuracy configurable integer arithmetic units [3][4][11], a limited amount of work has been focused on reconfigurable floating point multipliers. Previous work such as [12][13] presents the reconfigurability between single and double precisions instead of accuracies. To the best of our knowledge, this work presents the first accuracy configurable floating point multiplier that has multiple accuracy configurations and is capable of achieving an order of magnitude power reductions without any sacrifice on performance and with acceptable impacts on quality.

Chapter 3 Imprecise floating point arithmetic units[1]

3.1 Proposed set of imprecise floating point unit and special function units

Table 1 A set of proposed imprecise hardware function for common floating point operations

Function	Imprecise Function	Range	ε_{max}
$y = \frac{1}{x}$	$y = 2.823 - 1.882 \times x$	$x \in [0.5, 1]$	5.88%
$y = \frac{1}{\sqrt{x}}$	$y = 2.08 - 1.1911 \times x$	$x \in [0.5, 1]$	11.11%
$y = \sqrt{x}$	$y = x \times (2.08 - 1.1911 \times x)$	$x \in [0.25, 1]$	11.11%
$y = log_2 x$	$y = exp + 0.9846 \times x - 0.9196$	$x \in [1, 2)$	Unbounded
$y = \frac{a}{b}$	$y = a \times (2.823 - 1.882 \times b)$	$b \in [0.5, 1]$	5.88%
$y = a \times b$	$(1 + M_a)(1 + M_b) \approx 1 + M_a + M_b$	N/A	25%
$y = a \pm b$	Structural Parameter TH	$TH \in [1, 27]$	Unbounded
$y = a \times b \pm c$	Imprecise $\times$ and $\pm$	N/A	Unbounded

This thesis presents the design and evaluation of a set of imprecise FPU and SFU components whose "precise" counterparts are frequently used in compute-intensive applications and rank among the highest power consumers in a GPU. Results of a preliminary study presented in Figure 2 show that SFU and FPU operations together account for approximately 38% on average of total power consumption in a GPU, under compute-intensive benchmarks from Rodinia [16] and ISPASS2009 [17]. Additionally, these units are especially conducive to IHW design, because they are only used in arithmetic operations, meaning that if the output quality of these units is degraded, essential control and memory operations will not be affected. In contrast, the integer unit (ALU), which is used for such essential operations, accounts for only 4% on average of the total

[1] Part of the content in this chapter has been published in [33] and [34]

power consumption, and was not modified in this study. The set of imprecise FPU and SFU functions developed in this work are shown in Table 1.

The proposed imprecise reciprocal, inverse square root, square root, log2, and floating point division functions are based on linear approximation with the range reduction technique. These functions are commonly grouped in the special function unit of the microprocessor, as they could be configured to share hardware resources and using similar binary algorithms. In general, there are three approaches for calculating "precise" elementary functions, the lookup table based approach, the iterative approximation approach based on the Newton-Raphson (NR) method or the Goldschmidt's algorithm and a combined table-lookup and approximation approach proposed as the Tang's method [18][19]. Conventional table look-up methods such as multipartite require storing tables in the memory and perform an additional multi-operand multiplication and additions. It keeps a table of pre-computed values for every subintervals of the operand. The number of table entries is exponential in the number of bits representing the subinterval. Iterative methods do not need additional memory but require several multiplication-addition iterations. In order to achieve a "precise" result that is normally within one unit in the last place (ULP) error, both approaches can result in long propagation delay and significantly large power consumption [20][21].

This thesis aims at achieving significant power savings by sacrificing as much accuracy as possible under the constraint that the output quality is still acceptable to the application evaluated by application specific quality metrics. To approximate these power hungry elementary functions aggressively and achieve significant power savings, a range reduction is firstly performed on the operand to fit into a specific range and then the reduced

operand is applied onto a linear approximation function with no iterations. The third column of Table 1 shows the range of the operand as a result of range reduction. For instance, the single operand of the reciprocal function will be reduced to the range of [0.5,1). And then the reciprocal of the reduced operand will be approximated using the described linear equation. In hardware implementation, this requires minimum overhead as only a right shift is needed for reducing the mantissa of the operand which is already in the range of [1,2). The range reduction technique is commonly used in approximating floating point functions in hardware combined with quadratic approximations for high performance and high accuracy requirements [22]. The goal of imprecise hardware designs is to minimize power consumption while sacrificing accuracy according to application specific error tolerance, therefore, only a linear approximation is used here for best-case power reduction and limited accuracies as compared to commonly used quadratic approximations using Lagrange or least square approximations with high accuracy but also very high power consumption.

Using IEEE-754 floating point number representation, the range reduction can be implemented easily by replacing the exponent with the predefined constant exp_bias – 1 and a conceptually right shift of the mantissa (in hardware, it is passed through with a different alignment of bits), where exp_bias equals 127 and 1023 in single and double precision, respectively. In the case of log2, the exponent is replaced with exp_bias. The approximation functions can also be used as initial approximation functions for conventional iterative methods such as the Newton-Raphson (NR) method or the Goldschmidt's algorithm [23]. The simpler linear approximation eliminates the need for additional memory in table-based methods or multiple iterations in iterative methods, with

a reasonable amount of accuracy sacrifice, as demonstrated in Section 5. The coefficients of each linear approximation function are obtained using curve fitting techniques with the goal to minimize the mean absolute error. Also shown in the table is the error information related to each imprecise function. For instance, the imprecise reciprocal function has an ε_{max} of 5.88%, where ε_{max} represents the maximum absolute error percentage. The maximum error percentage is an important metric for an imprecise hardware design. It can be obtained either by formal mathematical proofs or numerical analysis with significant large uniformly distributed random input vectors. The details of error analysis will be presented in Chapter 5.

When designing the imprecise floating point addition, subtraction, and multiplication units, we take a different approach by simplifying and restructuring these functions at the algorithmic and logic level. An algorithmic-level simplification for floating point multiplication $z = a \times b$ is shown as follows:

$$a = S_a \times 2^{\text{expa}} \times (1 + M_a);\ 0 \le M_a < 1 \qquad (1)$$

$$b = S_b \times 2^{\text{expb}} \times (1 + M_b);\ 0 \le M_b < 1 \qquad (2)$$

then:

$$S_z = S_a\ xor\ S_b \quad (3)$$

$$expz = expa + expb + cin \qquad (4)$$

$$M_z = (1 + M_a) \times (1 + M_b)$$

$$\approx \begin{cases} 1 + M_a + M_b & ;(M_a + M_b < 1) \\ \frac{1+M_a+M_b}{2} & ;(M_a + M_b \ge 1) \end{cases} \qquad (5)$$

$$cin = \begin{cases} 0; & (M_a + M_b < 1) \\ 1; & (M_a + M_b \geq 1) \end{cases} \qquad (6)$$

In equation (5), the imprecise floating point multiplication approximates the result by neglecting the additional term $M_a * M_b$. In the hardware circuit for a 32-bit floating point multiplier, this means that the 24x24-bit mantissa multiplication can be effectively replaced with a 25x25-bit addition. Since the result is already an approximation, no IEEE-754 compliant rounding circuit is needed. Subnormal numbers are set to zero by default so that additional hardware for handling subnormal numbers in rare situations can be ignored. Infinities and *NaN*s are still supported. As a result of the restructuring and simplification, a 5X latency reduction and 25X power savings are observed. The detailed results are shown in Chapter 5 and 6.

The same approach is applied to the floating point adder as well. A threshold value *TH* (a design-time structural parameter) is set on the exponent difference. During the mantissa alignment, if the exponent difference exceeds the threshold *TH*, the mantissa of the smaller operand is effectively set to zero. This means that in the hardware circuit for a floating point adder, only a *TH* bit right shifter and a (*TH+1*)-bit adder is needed, as opposed to the 27-bit right shifter and adder in the IEEE-754 compliant floating point adder. For example, if *TH=3*, *expa – expb = 1*, and $b = 1.x_1x_2x_3x_4x_5 \times 2^{expb}$, then operand *b* after the shift-and-align stage will become:

$$b' = 0.1x_1x_2000 \times 2^{expa} \qquad (7)$$

If however, when *expa – expb = 4*, which is greater than the threshold of 3, the mantissa of operand *b* after shift-and-align will be zero, and the addition result will be equal to operand *a*. IEEE-754 compliant rounding circuits are ignored and subnormal numbers

are set to zero as well. For a 32-bit floating point adder with $TH = 8$, the maximum error percentage is only 0.78% for effective addition operations with about 70% power savings and 25% performance improvement. This combination of algorithmic and logic level simplification and restructuring of floating point functional units result in a significant power savings with small quality degradations.

3.2 A low-power accuracy-configurable floating point multiplier

This section details an improvement that could be made on the imprecise floating point multiplier discussed in the previous chapter, where 25X power reduction could be achieved at a maximum error magnitude of 25%. The power error quality trade-off could be significantly improved as shown in this chapter so that 26X power reduction could be achieved at only 11.4% for single precision floating point multiplier and 49X power reduction at only about 18.07% maximum error magnitude for double precision operations. The improved floating point multiplier is based on Mitchell's algorithm and can be configurable based on different accuracy levels and application dependent error tolerance requirements.

3.2.1 Mitchell's Algorithm for Fixed Point Multiplications

A simple binary to logarithm conversion algorithm for approximating fixed point multiplication and division was proposed in [7], which has been commonly referred to as the Mitchell's Algorithm (MA). The approximation algorithm involves three major steps. The first step is to convert each operand to its log2 based logarithm values by applying a piecewise linear approximation in the range of zero to one. The logarithm values are then added or subtracted according to the intended operation. And the last step is to perform another piecewise linear approximation to find the antilogarithm value from previous step, which gives the result of the approximated multiplication or division. A brief mathematical summary of the approximation algorithm is shown as follows:

Assume D is an integer number and can be represented in the binary form as:

$$D = (-1)^s \sum_{i=0}^{k} 2^i Z_i \quad i \in N, Z_i \in \{0,1\} \qquad (8)$$

Where i represents the bit position of each binary digit, Z_i represent the binary value of either *"0"* or *"1"*, and k is the most significant bit position of the leading *"1"* in $D's$ binary representation. Before performing the piecewise linear approximation, a 2^k can be factored out:

$$D = (-1)^s 2^k (1 + \sum_{i=0}^{k-1} 2^{i-k} Z_i) = 2^k (1 + x), x \in [0,1) \quad (9)$$

$x = \sum_{i=0}^{k-1} 2^{i-k} Z_i$ is now a fixed point decimal number between 0 and 1. Applying linear approximation on the straight line curve for $\log_2(1 + x), x \in [0,1)$ function yields the following approximation equation:

$$\log_2 D \approx k + x \qquad (10)$$

Therefore, the product of D1 and D2 could be approximated by:

$$D_1 \times D_2 \approx \begin{cases} 2^{k_1+k_2} \times 2^{x_1+x_2}, x_1 + x_2 \in [0,1) \\ 2^{k_1+k_2+1} \times 2^{x_1+x_2-1}, x_1 + x_2 \in [1,2) \end{cases} \quad (11)$$

Finally, the product can be approximated by another linear approximation function using $2^x \approx 1 + x$ for $x \in [0,1)$:

$$D_1 \times D_2 \approx \begin{cases} 2^{k_1+k_2} \times (1 + x_1 + x_2), x_1 + x_2 \in [0,1) \\ 2^{k_1+k_2+1} \times (x_1 + x_2), x_1 + x_2 \in [1,2) \end{cases} \quad (12)$$

Figure 6 shows the conventional hardware implementation of Mitchell's Algorithm. The binary to log conversion approximation is calculated by the leading one detector (LOD) and the Barrel left shifter. The log to binary conversion after the addition is performed by a simple decoder. A '1' is inserted into the appropriate position of the approximated binary result.

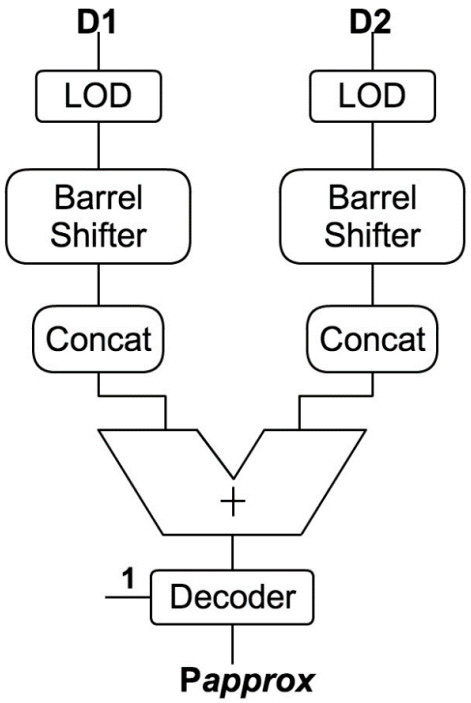


Figure 6 MA hardware implementation for fixed point

- Mitchell's algorithm for binary fixed point addition
  - Let N_1, N_2 be two n-bits binary multiplicands, P_{approx}:2n-bits approximated product
  - Calculate K_1: leading '1' position of N_1
  - Calculate K_2: leading '1' position of N_2
  - Shift N_1, N_2 to the left by $n - K_1$ and $n - K_2$ bits respectively and concatenate after K_1, K_2 respectively in S_1, S_2 (binary to log estimation)
  - Calculate S: Adding S_1, S_2
  - Decode S and insert a '1' in its position of P_{approx} (2^x approximation)
  - $N_1 * N_2 \approx P_{approx}$

3.2.2 Design of low-power accuracy-configurable floating point multiplier

IEEE-745 standard floating point (FP) multiplication contains a mantissa multiplication block, which is usually the most power hungry component in a FP multiplier. Significant amount of work on low power FP multipliers have focused on the mantissa multiplier, as mentioned in the related work section, and relatively large amount of power/energy savings could be achieved. However, to the best of our knowledge, no previous work has applied Mitchell's Algorithm to the floating point multiplier attempting to dramatically reduce power consumption and evaluate the implementation within the power-quality trade-off design paradigm.

An intuitive method to apply Mitchell's Algorithm to the floating point multiplication is simply replacing the mantissa multiplier with an MA multiplier of the same bitwidth. However, this method leads to a fixed accuracy design with a maximum error magnitude of 11.11%. In fact, various algorithms have different error sensitivities, some of which may require a much higher accuracy than others. Therefore, a fixed accuracy design will limit the FP multiplier to a small number of applications. To enable more accuracy configurations, an algorithmic transformation can be performed on the mantissa multiplication, which could reduce the maximum error to only 2.04% without significant hardware costs. In addition, bit truncations can still be applied on top of the algorithmic transformation, providing a wide range of accuracies for maximum amount of power savings while still producing acceptable quality of results (QoR) to specific applications. The algorithmic transformation for a FP multiplication $Z = a \times b$ is shown as following:

$$a = S_a \times 2^{expa} \times (1 + M_a); M_a \in [0,1) \quad (13)$$

$$b = S_b \times 2^{expb} \times (1 + M_b); M_b \in [0,1) \quad (14)$$

Then:

$$Z = (S_a \oplus S_b) \times 2^{expa+expb-BIAS} \times (1 + M_a) \times (1 + M_b) \quad (15)$$

$$Z = (S_a \oplus S_b) \times 2^{expa+expb-BIAS} \times (1 + M_a + M_b + M_a \times M_b) \quad (16)$$

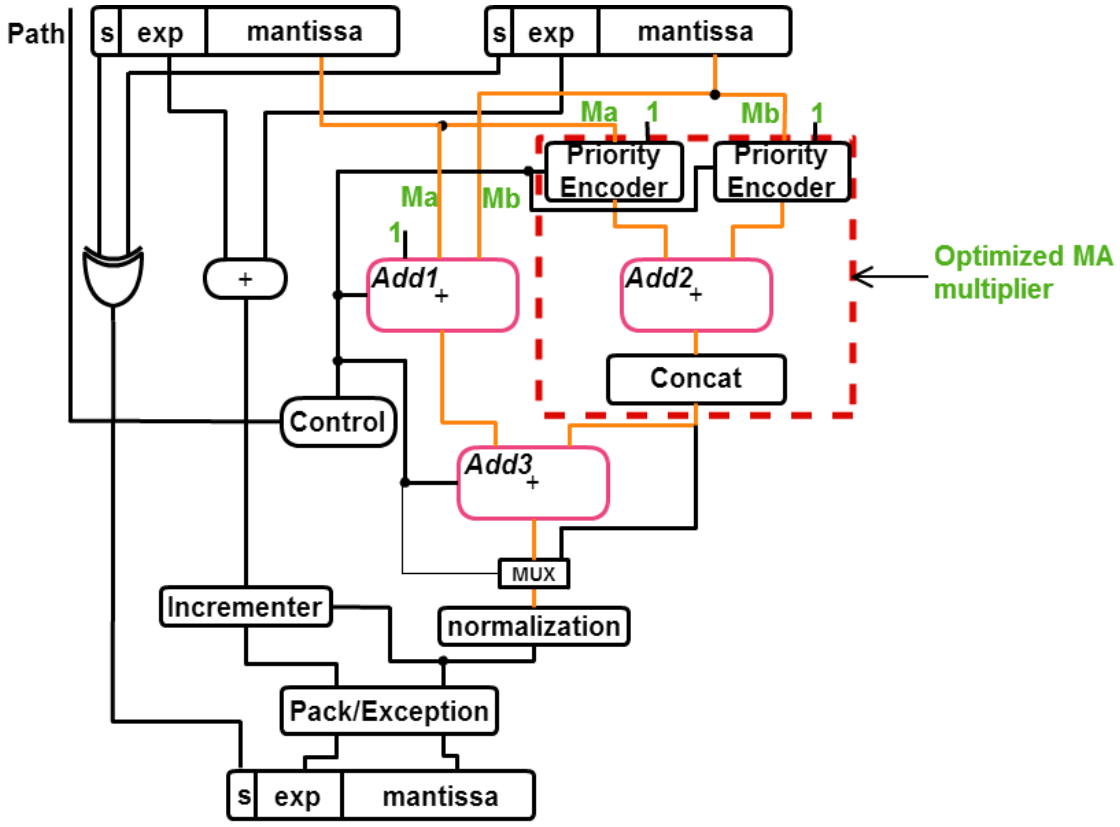


Figure 7 Low-power accuracy configurable floating point multiplier based on Mitchell's Algorithm

An observation can be made such that the MA multiplier can be applied on either the mantissa multiplication $(1 + M_a) \times (1 + M_b)$ (the Log Path) or the fraction multiplication of $M_a \times M_b$ (the Full Path) with an additional adder for $(1 + M_a + M_b)$. The log path is effectively the intuitive replacement of the mantissa multiplier by an MA multiplier with a maximum error magnitude of 11.11%. The full path has a much lower

maximum error magnitude of 2.04%. This effectively replaces the multiplier in the IEEE-754 standard implementation with three adders, and the large accuracy range between the two configurations allows bit truncation to be applied on both datapath, enabling a wide range of accuracy configurations.

This accuracy configurability allows the FP mulitplier to be used by a variety of error tolerant applications. By truncating 19 bits in the single precision mantissa multiplication, the proposed FP multiplier is able to achieve 26X power reductions with a maximum error percentage of 18%, while truncating 21 bits directly in the "Precise" FP multiplier produces about 21% maximum error but only gains 2.3X power reduction. Chapter 5 provides a formal error analysis and Chapter 6 provides the detailed power measurement as well as application level quality evaluation results.

Figure 7 shows the micro-architecture of the proposed accuracy configurable FP multiplier using Mitchell's Algorithm. Instead of implementing a power hungry Barrel left shifter and LOD, a priority encoder can be used, which has much simpler structure and helps minimize the power consumption. As seen from the figure, the mantissa multiplier is now replaced with a MA multiplier and two adders. The adder *Add1* performs addition of $(1 + M_a + M_b)$ for the full path configuration. *Add2* in the MA multiplier can be configured to calculate either the log path or the $(M_a \times M_b)$ for the full path. And the $(1 + M_a + M_b)$ and $(M_a \times M_b)$ can be added by *Add3*. When the multiplier is configured to the log path, *Add1* and *Add3* are set idle by multiplexing all inputs to constant 0. And *Add3* is effectively bypassed by a set of multiplexers. In this case, *Add1* and *Add3* consume only the leakage power. During the operation of the full path configuration, all three adders are switching, providing about 2X the power reduction compared to its single precision

IEEE-754 counterpart. It's necessary to point out that part of the power savings of the accuracy configurable FP multiplier comes from the elimination of the rounding unit, which consumes up to 18% power consumption according to [8]. Since the proposed FP multiplier is inherently imprecise and not compliant to the IEEE-754 standard, no rounding unit is needed.

Chapter 4 Error Analysis and Characterization[2]

Error analysis of imprecise hardware functions show their error probability (frequency), maximum error magnitude or percentage, and mean error magnitude or percentage. These are important quality metrics for designers and programmers to choose the appropriate configurations. Error analysis is usually done by formal mathematical analysis and proofs. For certain imprecise configurations where formal mathematical analysis is not possible or very challenging, numerical analysis can be performed by feeding the imprecise unit with a significant large set of uniformly distributed input vectors. Error characterization shows the error distribution of an imprecise hardware function for a given input distribution. It can show the error probability or error frequency for a specified interval of interest. The results of error analysis and characterization can be useful in guiding the selection of different imprecise hardware configurations.

4.1 Error Analysis

4.1.1 Error Analysis for Imprecise Floating Point Adder

The maximum absolute error percentage, represented as ε_{max} for each imprecise function is shown in Table 1. This sub chapter presents the formal error analysis for the imprecise floating point adder as well as the improved accuracy configurable floating point multiplier. Error analysis for other floating point and special function units are essentially the same. As described in the previous chapter, the output accuracy of the imprecise floating point adder is determined by the design time structural parameter *TH*. We assume both operands a and b are normal floating point numbers, and they can be represented using

[2] Part of the content in this chapter has been published in [33] and [34]

equations (1) and (2). We also assume that the exponent difference d between a and b are non-negative so that the compare-and-swap step can be ignored to simplify the analysis process. Let

$$y = a \pm b; \quad y, a, b \in \mathbb{R} \qquad (17)$$

$$a \pm b = S_a \times 2^{expa} \times (1 + M_a) \pm S_b \times 2^{expb} \times (1 + M_b); \qquad (18)$$

$$expa - expb = d; d \in N; 0 \le TH \le 27; \qquad (19)$$

Depending on the effective operation, we need to consider four different cases:

$$\begin{cases} y = a + b; & d \ge TH \\ y = a + b; & 0 \le d \le TH \\ y = a - b; & d \ge TH \\ y = a - b; & 0 \le d \le TH \end{cases} \qquad (20)$$

a) $y = a + b; \quad d \ge TH$; In this case, b will be 0 after alignment and shifting, therefore,

$$\varepsilon_{\max} = \frac{(1 + M_a) \times 2^{expb}}{(1 + M_a) \times 2^{expa} + (1 + M_a) \times 2^{expb}} = \frac{1}{\frac{1 + M_a}{1 + M_b} \times (2^{expa - expb}) + 1}$$

$$= \frac{1}{\frac{1 + M_a}{1 + M_b} \times 2^d + 1} < \frac{1}{2^{d-1} + 1} < \frac{1}{2^{TH-1} + 1}; (\frac{1.M_a}{1.M_b} > \frac{1}{2}, d \ge \text{TH})$$

When $TH = 8$, $\varepsilon_{max} < \frac{1}{2^7+1} \approx 0.775\%$

b) $y = a + b; \quad 0 < d < TH;$

$$\varepsilon_{max} = \frac{2^{-TH} \times 2^{expb}}{(1 + M_a) \times 2^{expa} + (1 + M_a) \times 2^{expb}}$$

$$< \frac{1}{2^{TH}(2^d + 1)} < \frac{1}{2^{TH+1}};$$

When $TH = 8$, $\varepsilon_{max} < \frac{1}{2^9} \approx 0.2\%$

c) $y = a - b;$ $\quad d \geq \text{TH};$

$$\varepsilon_{max} = \frac{1.M_b \times 2^{expb}}{1.M_a \times 2^{expa} - 1.M_b \times 2^{expb}} < \frac{1}{2^{TH-1} - 1};$$

When $TH = 8$, $\varepsilon_{max} < \frac{1}{2^7 - 1} \approx 0.785\%$

d) $y = a - b;$ $\quad 0 < d < TH;$

$$\varepsilon_{max} = \frac{2^{-TH} \times 2^{expb}}{1.M_a \times 2^{expa} - 1.M_b \times 2^{expb}}$$

$$= \frac{1}{2^{TH} \times (1.M_a \times 2^d - 1.M_b)};$$

In the first three cases (a)-(c), the ε_{max} are bounded by a percentage that is smaller than 0.785% when $TH = 8$. In case (d), however, when the effective operation is a subtraction and the exponent difference is less than the predefined threshold, the maximum error percentage ε_{max} explodes. This happens when the two operands are very close to each other on the real line and are likely to produce a subnormal number as a result of subtraction. In this case, both the "precise" result and the approximated result can be very close to zero while producing a very large error percentage. However, due to the small absolute quantity, this will have minimum effect on the output quality of the application, despite the large relative error percentage.

4.1.2 Error Analysis for Imprecise Accuracy Configurable Floating Point Multiplier

A detailed formal error analysis for Mitchell's Algorithm fixed point multiplication is provided in [7], proving that the maximum error magnitude for a MA multiplier is 11.11%. The analysis also applies to the log path for the proposed multiplier as the entire mantissa multiplication is essentially replaced by the MA multiplier. This section provides the formal error analysis for the full path configuration, when the mantissa multiplication is approximated by $1 + M_a + M_b + MA(M_a, M_b)$, where $MA(M_a, M_b)$ stands for using Mitchell's Algorithm for approximating $M_a \times M_b$.

Since $MA(M_a, M_b)$ is essentially a fixed point operation in binary, approximation equation (12) applies to $M_a \times M_b$. Let's define:

$$M_a = 2^{k_a}(1 + x_a); \quad x_a \in [0,1), k_a \leq -1$$

$$M_b = 2^{k_b}(1 + x_b); \quad x_b \in [0,1), k_b \leq -1$$

$$MA(M_a, M_b) = \begin{cases} 2^{k_a+k_b} \times (1 + x_a + x_b), & x_a + x_b \in [0,1) \\ 2^{k_a+k_b+1} \times (x_a + x_b), & x_a + x_b \in [1,2) \end{cases} \quad (21)$$

Let ε be the error magnitude percentage, then:

$$\varepsilon = \frac{(1 + M_a)(1 + M_b) - (1 + M_a + M_b + MA(M_a, M_b)}{1 + M_a + M_b + M_a \times M_b}$$

$$= \frac{M_a \times M_b - MA(M_a, M_b)}{1 + M_a + M_b + M_a \times M_b}$$

① Assume that $x_a + x_b \in [0,1)$:

$$\varepsilon = \frac{2^{k_a+k_b}(1 + x_a)(1 + x_b) - 2^{k_a+k_b}(1 + x_a + x_b)}{1 + 2^{k_a}(1 + x_a) + 2^{k_b}(1 + x_b) + 2^{k_a+k_b}(1 + x_a)(1 + x_b)}$$

$$= \frac{1}{\frac{(1+2^{-k_a})(1+2^{-k_b})}{x_a x_b} + \frac{2^{-k_a}+1}{x_a} + \frac{2^{-k_b}+1}{x_b} + 1}$$

$$= f(k_a, k_b, x_a, x_b)$$

Since $k_a, k_b \leq -1$, then:

$$\varepsilon_{\max} = \lim_{k_a \to -1} \lim_{k_b \to -1} f(k_a, k_b, x_a, x_b)$$

$$= \frac{1}{\frac{9}{x_a x_b} + \frac{3}{x_a} + \frac{3}{x_b} + 1}$$

$$= \frac{1}{g(x_a, x_b)}$$

Let $x_a + x_b = \alpha,\ 0 < \alpha < 1$:

$$g(x_a, \alpha) = \frac{9}{x_a(\alpha - x_a)} + \frac{3}{\alpha - x_a} + \frac{3}{x_a} + 1$$

$$Let\ \frac{\partial g}{\partial \alpha} = 0 \xrightarrow{yields} x_a = \frac{\alpha}{2}$$

$$\varepsilon_{\max} = \frac{1}{g_{min}} = \frac{1}{\lim_{\alpha \to 1} g(x_a, \alpha)}\Big|_{x_a = \frac{\alpha}{2}} = \frac{1}{49} \approx 0.0204$$

② Assume $x_a + x_b \in [1,2)$:

$$\varepsilon = \frac{2^{k_a+k_b}(1+x_a)(1+x_b) - 2^{k_a+k_b+1}(x_a+x_b)}{1 + 2^{k_a}(1+x_a) + 2^{k_b}(1+x_b) + 2^{k_a+k_b}(1+x_a)(1+x_b)}$$

$$= \frac{1}{\frac{(1+2^{-k_a}+x_a)(1+2^{-k_b}+x_b)}{(1-x_a)(1-x_b)}}$$

$$= f(k_a, k_b, x_a, x_b)$$

Then:

$$\varepsilon_{\max} = \lim_{k_a \to -1} \lim_{k_b \to -1} f(k_a, k_b, x_a, x_b)$$

$$= \frac{1}{\frac{(3 + x_a)(3 + x_b)}{(1 - x_a)(1 - x_b)}}$$

$$= \frac{1}{g(x_a, x_b)}$$

Let $x_a + x_b = \alpha, \alpha \geq 1$:

$$g(x_a, \alpha) = \frac{(3 + x_a)(3 + \alpha - x_a)}{(1 - x_a)(1 - \alpha + x_a)}$$

$$Let\ \frac{\partial g}{\partial \alpha} = 0 \xrightarrow{yields} \alpha = 1$$

$$\varepsilon_{\max} = \frac{1}{g_{min}} = \frac{1}{\lim_{x_a \to \frac{1}{2}} (\frac{12}{x_a - x_a^2} + 1)} |_{\alpha=1} = \frac{1}{49} \approx 0.0204$$

The above analysis shows the maximum error bound is 2.04% for the full path configuration in the proposed FP multiplier when no bitwidth truncation is applied. It applies to both single and double precisions. The maximum error percentage for truncated configurations is very challenging to obtain using formal analysis. Therefore, they are obtained using statistical analysis during the process of error characterization.

4.2 Error Characterization

Maximum error percentage is an important metric for an imprecise floating point multiplier. However, using the error bound alone does not provide a complete picture of

its behavior under a dynamic wide range of input vectors. IHW error characterization provides some statistical insight, such as error rate and error magnitude distribution, to applications and system designers. The error characterization shows the error sensitivity of each component under no application context. When doing quality tuning, it is helpful to look at the error characteristics to determine which component to be deployed. It is very necessary in imprecise arithmetic component design as it shows multiple quality properties and determines the applicability of the designed component to applications. It can serve as a guide for quality tuning. It is inherently and implicitly reflected in simulation results since it is the intrinsic property of the imprecise component.

Compared to fixed point arithmetic operations, error characterization for floating point arithmetic is difficult because of the large range and non-discrete nature of floating point numbers. However, as the proposed imprecise floating point algorithm has no effect on the accuracy of the exponent addition, the range between 0.0 and 1.0 on the real line can provide a good coverage for characterizing the error percentage distribution produced only from the mantissa multiplication. In addition, obtaining a uniform distribution along the real line using the conventional pseudo-random number based Monte Carlo simulation method would result in an extremely large sample space, making the characterization processes very slow and producing biased results. Instead, we use the quasi-Monte Carlo method [24], which uses a low discrepancy sequence to generate correlated numbers that could provide a better uniformity in the specified range.

Figure 8 shows the probability mass function (PMF) of error distributions for 32-bit imprecise functions proposed in Chapter 3 using the quasi- Monte Carlo method with 200 million random inputs. Each bar indicates a non-zero error probability or error

frequency. The sum of all bars represents the error rate of the component. The x axis is $log2$ based and represents the upper bound of error magnitude percentage obtained using the following formula:

$$x = \lceil \log_2 |ERR\%| \rceil$$

For example, for 32 bit *fpadd* in Figure 8, a bar on top of the -2 marker indicates that there is a 5% probability that the error percentage is bounded between $2^{-3}\%$ and $2^{-2}\%$. It can be seen that the floating point adder and the log2 function are dominated by frequent small magnitude (FSM) error. The error magnitude explosion problem analyzed in the previous section for the floating point adder has a probability very close to zero when the error magnitude is larger than 8%. For other imprecise functions, there is an increasing probability towards larger error magnitude, but the error magnitude is bounded by the theoretical maximum error, shown in Table 1.

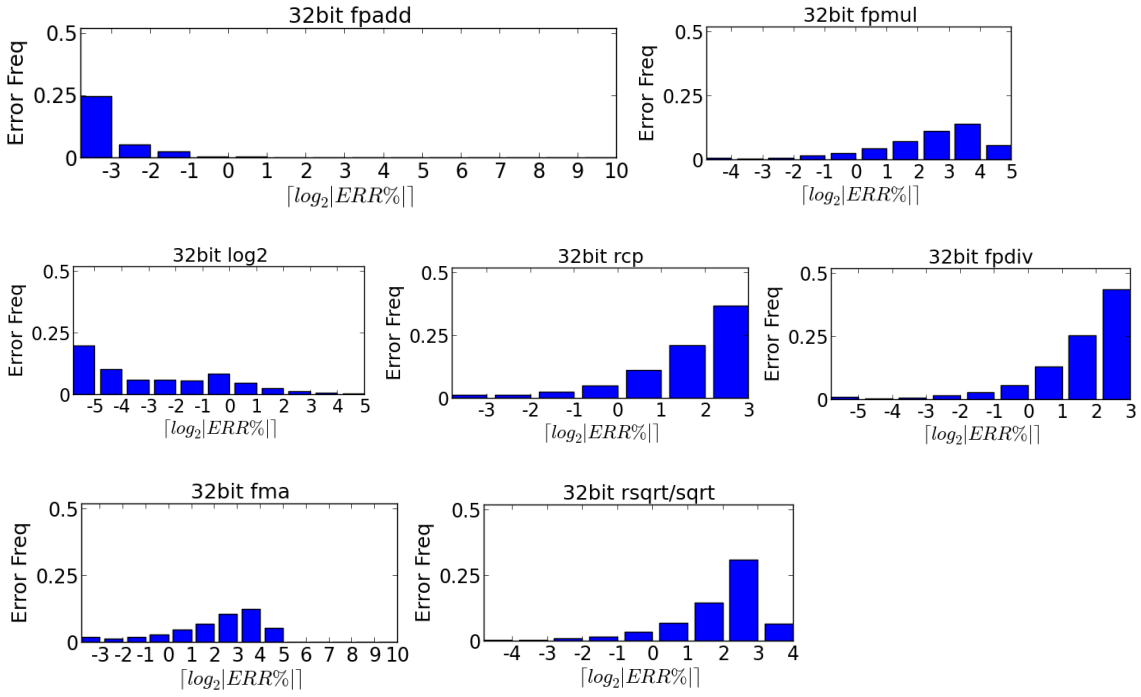


Figure 8 Proposed imprecise hardware error characterization with 200 million random

Figure 9 shows the probability mass function (PMF) of error distributions for the improved floating point multiplier with both the full path configuration and the log path configuration and several bit-truncation schemes applied on top of both datapath. It can be seen that as the number of truncation bits increases, the error probability tends to be clustered to the right but not the rightmost interval. This indicates that even though the maximum error percentage seems to be large for a configuration, most input vectors will have an error percentage far below the maximum error bound. With each configuration characterized by a large set of input vectors, we can use this information to guide the choice of a particular configuration during application specific quality tuning processes. For instance, in Figure 9, there is only a small difference between “Log Path Tr17” and “Log Path Tr18”. However, a noticeable difference appears between 18 and 19 bits truncation as the highest error probability interval is shifted to the right.

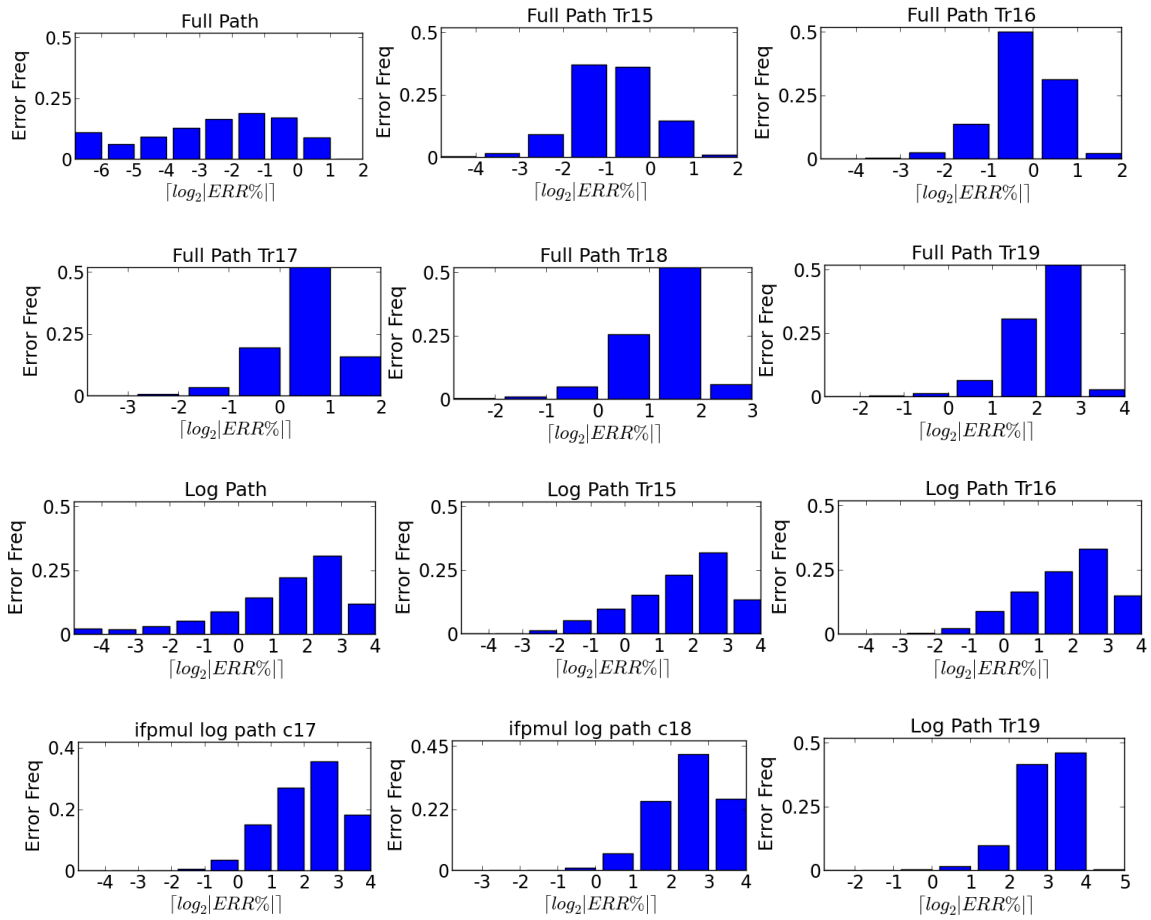


Figure 9 Improved imprecise floating point multiplier error characterization with 200 Million random inputs

Chapter 5 Experimental Methodology and Results[3]

5.1 Power-quality tradeoff frame work

To study IHW arithmetic components on GPU, we propose a power-quality tradeoff framework based on CAD synthesis tools, GPGPU-Sim, and GPUWattch simulation models to quickly evaluate the impact of IHW on the output quality and estimate the impact on GPU system level power consumption. GPGPU-Sim is a cycle-accurate simulator that models GPU architectures similar to the NVIDIA Fermi series [25]. GPUWattch is an energy model based on McPAT [26]. It models GPU power consumption by fetching the performance counters from GPGPU-Sim during the simulation process. The modified McPAT with GPU-specific architectural components then computes the estimated static and dynamic power using per-access energy acquired from synthesis. For the Fermi architecture, the modeling error on power consumption is 9.7% [17].

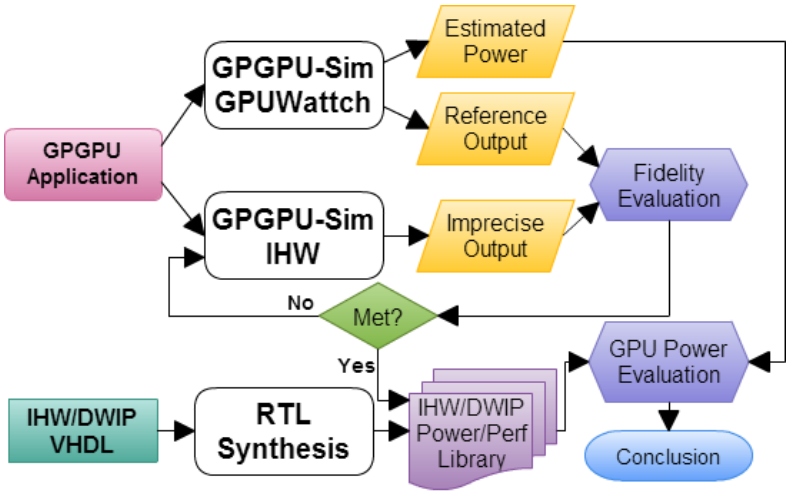


Figure 10 Methodology flow for estimation GPGPU system level power quality trade-off with IHW

[3] Part of the content in this chapter has been published in [33] and [34]

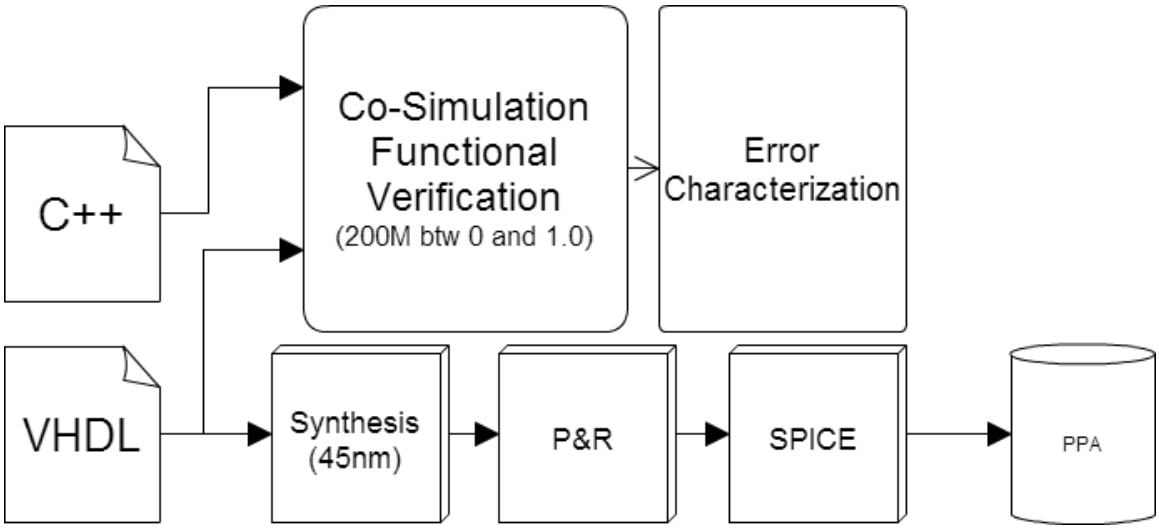


Figure 11 Top-down Power synthesis, functional verification and error characterization flow

Figure 10 shows the methodology flow used in this study. All the imprecise components were first implemented as functional models in C++. The correctness of the functional models was verified against hardware models written in VHDL through simulation. The functional models were then imported into the GPGPU-Sim simulator. A knob was created for allowing the simulation to run in either the precise or the imprecise mode. Each imprecise hardware unit can be enabled or disabled individually, along with the tunable structural parameter, as detailed in Chapter 3. A GPGPU application is run on GPGPU-Sim with GPUWattch enabled to collect the power and performance statistics and the reference (baseline) application output. We then run a functional simulation with imprecise arithmetic units enabled to obtain the imprecise output. The reference and the imprecise outputs are then compared and evaluated using an application specific quality metric against a predefined fidelity constraint. If the constraint is not met, the structural parameter is adjusted or some imprecise components are disabled based on their

application-specific error sensitivity and error characteristics from previous characterization analysis to improve the output quality. The output is then re-evaluated for the updated imprecise output. The iterative quality tuning process is complete once the quality constraint is satisfied.

To obtain non-functional metrics for proposed IHW components, we developed a synthesized HDL library. Due to the fact that implementation details of components in a GPU are not publicized, we compared our proposed imprecise components against the standard IEEE-754 compliant counterparts in the Synopsys DesignWare IP (DWIP) library. All proposed components and DWIPs were synthesized in Synopsys Design Compiler using 45nm FreePDK library. SPICE netlists were extracted after place and route in Encounter. The post-layout netlists were then sent to HSIM for SPICE simulations. The power consumptions were measured in HSIM with 500 random input vectors. The process is shown in Figure 11 together with the co-simulation and error characterization processes. With the power statistics obtained by GPUWattch for each benchmark application, we evaluated the impacts of the IHW design approach on nonfunctional metrics at the system level by applying the hardware synthesis results to the power simulation results from GPUWattch. The system-level GPU power improvement is the additive result of the power improvement in both FPU and SFU.

To estimate the system level power impact of IHW, we used a similar estimation approach to GPUWattch by using per-access energy acquired from synthesis. The latency for each operation is calculated by assuming a continuously operating pipeline with no stalls. We also assume that other idle components in the execution units are power-gated with no effect on dynamic power consumption. A 700 MHz core clock frequency of the

execution pipeline was used, which is the same as GPUWattch. The application specific average power consumptions from IHW and DWIPs are calculated by dividing the total average energy (sum of average energy of each operation) by total latency spent in the functional units. Then the application level power savings from IHW can be obtained by comparing against the application specific power consumption from DWIPs. The percentage power savings for FPU and SFU is then applied to the percentage of power consumption obtained from GPUWattch to calculate the system level power savings. Due to the limitation of existing GPU power modeling tools and the difficulty of verifying against real GPU power numbers, these results serve as an estimation of the relative system level impact from IHW.

```
init_perf_acc(); // read in all performance counters
init_syn_res();  // initialize nonfunctional metrics matrix
for each op in op_list:
acc = get_perf_counter(op);
(ihw_pwr,ihw_lat)= get_syn_res(op, imprecise_mode[op]);
(dw_pwr,dw_lat)  = get_syn_res(op, imprecise_mode=False);
i_pipe_lat = [acc - 1 + ceil(ihw_lat/CLK_FREQ)]/CLK_FREQ;
dw_pipe_lat = [acc - 1 + ceil(dw_lat/CLK_FREQ)]/CLK_FREQ;
if op ϵ FPU:
ihw_fpu_eng += ihw_pwr * i_pipe_lat;
dw_fpu_eng  +=  dw_pwr * dw_pipe_lat;
elif op ϵ SFU:
ihw_sfu_eng += ihw_pwr * i_pipe_lat;
dw_sfu_eng  +=  dw_pwr * dw_pipe_lat;
end for;
ihw_fpu  = ihw_fpu_eng / tot_ihw_lat;
dw_fpu  = dw_fpu_eng / tot_dw_lat;
…
avg_fpu_pwr_impr = | (dw_ fpu_pwr- ihw_fpu_pwr) | / dw_ fpu_pwr
avg_sfu_pwr_impr = | (dw_ sfu_pwr- ihw_sfu_pwr) | / dw_ sfu_pwr
sys_pwr_impr = fpu_pwr * avg_fpu_pwr_impr + sfu_pwr * avg_sfu_pwr_impr;
```

Figure 12 Simplified algorithm for estimating system level power savings

5.2 Non-Functional Metrics

Table 2 Nonfunctional metrics of 32-bit IHW components (normalized against DWIP components, lower is better)

Function	Power(mW)	Latency(ns)	Area(GE)	Energy(pJ)	EDP(pJ*ns)
ifpadd	0.31	0.74	0.39	0.23	0.17
ifpmul	0.040	0.218	0.103	0.009	0.002
ifpdiv	0.84	0.85	0.64	0.71	0.60
ircp	0.20	0.34	0.25	0.07	0.02
isqrt	1.16	0.33	1.04	0.39	0.13
ilog2	0.30	0.79	0.36	0.24	0.19
ifma	0.08	0.70	0.14	0.05	0.04
irsqrt	0.061	0.109	0.087	0.007	0.001

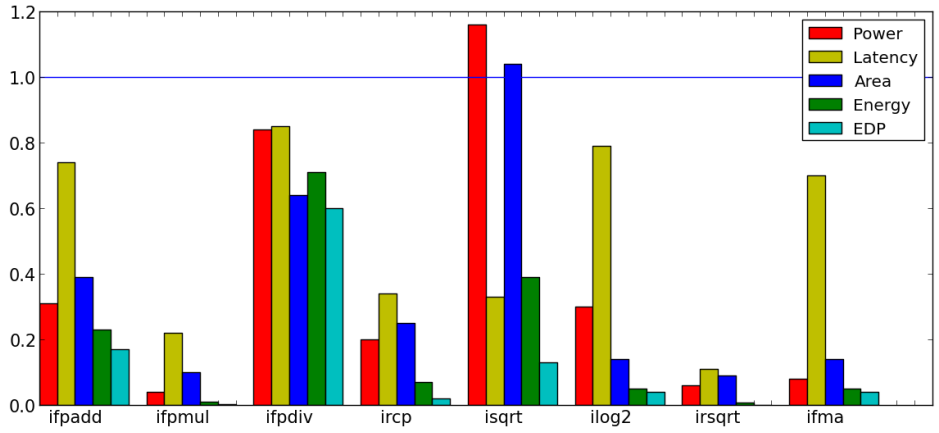


Figure 13 Normalized Non-functional Metrics of 32bit IHW vs. DWIPs

Table 3 Integer Adder vs. Integer Multiplier

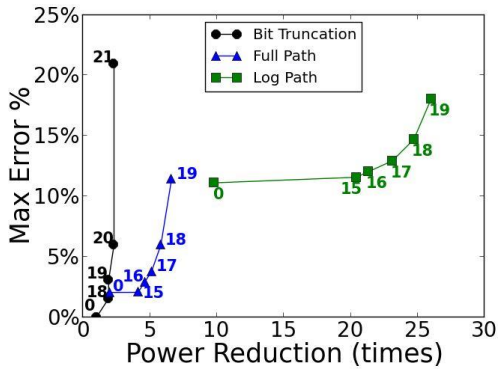
Function	Power(mW)	Latency(ns)
25bit Add	0.24	0.31
24bit Mult	8.50	0.93

A library consisting of all proposed imprecise units was implemented in VHDL. We also use the highly optimized and industrial standard soft IPs from the IEEE-754 compliant Synopsys DesignWare IP library (DWIPs) as baselines for comparison. All IHW

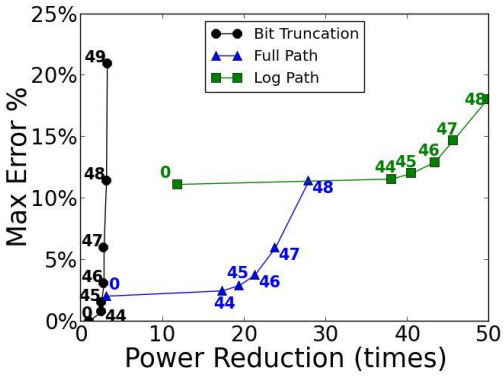
components and DWIPs were synthesized using Synopsys's Design Compiler with 45nm FreePDK using the top-down power flow described in Figure 11. The power, performance, and area (P.P.A) results were stored into a matrix for system level power evaluation for each application. Table 2 shows the normalized nonfunctional metrics compared against each corresponding DesignWare IP component. And Figure 13 is a bar chart representing the same set of data for better interpretation. For example, the 32-bit floating point adder was synthesized with structural parameter *TH*=8. Along with other logic-level simplifications such as ignoring rounding circuits, a 69% power savings and a 26% percent latency improvement were achieved. The proposed imprecise floating point multiplier demonstrates the most significant improvement among all IHW components: about 96% power reduction and 78% performance improvement. This is not a surprising result, as such significant improvements are mainly achieved by replacing the 24x24-bit multiplier with a 25x25-bit carry save adder. A closer study of these two components shows a difference in power consumption of approximately 35 times and approximately 3 times in performance, as shown in Table 3. Table 2 and Figure 13 also demonstrate other proposed IHW designs achieving significant power and performance improvements as well. Even though the *isqrt* component has 16% higher power consumption, the EDP savings can reach about 87%.

Figure 14 shows the power-quality trade-off design space for both single precision and double precision FP multipliers. The quality metric max error percentage is used here to be consistent with previous error analysis. Other inherent quality metrics such as mean error distance (MED) and worst case error distance (WED) [27] follow the same trends and therefore are not shown here. As seen from Figure 4(a), the log path in the single

precision imprecise multiplier can achieve more than 25X power reductions when 19 bits are truncated in the mantissa. However, the intuitive bit truncation schemes can only achieve approximately 2.5X power reduction at a higher max error percentage (21%). Similarly for the double precision multipliers, the imprecise multiplier can achieve 49X power reduction while truncating 48 bits in mantissa. This demonstrates that even though intuitive bit truncation schemes are simple and require little hardware modifications, they are far from optimal in the power-quality design space when aggressive power reductions are desirable.



(a) 32bit imprecise FP multiplier



(b) 64bit imprecise FP multiplier

Figure 14 Power-quality trade-off of accuracy configurable FP multiplier

Table 4 Non-functional metrics of imprecise FP multiplier

Configuration	Power(mW)	Latency(ns)	Area(um^2)
DW_fp_mult_32	36.63	1.7	19551.5
ifpmul32*	17.93	1.7	7671.2
ifpmul32°	18.59	1.4	9209.6
DW_fp_mult_64	119.9	2.0	66817.5
ifpmul64*	38.17	2.0	28447.1
ifpmul64°	39.65	1.8	32784.4

* The proposed FP multiplier with full bitwidth at same latency
° The proposed FP multiplier with full bitwidth at minimum latency

5.3 Application level power-quality trade-off

5.3.1 System level power-quality trade-off with GPGPU applications

We apply the power-quality tradeoff framework presented in Chapter 5.1 to study three compute-intensive CUDA benchmark applications from Rodinia and ISPASS2009 benchmark suits: HotSpot, SRAD, and RayTracing. These three benchmark applications were chosen based on their representative application domains, the dominance of floating point operations in FPU and SFU, and the availability of outputs for quality evaluation with specific quality metrics. Other compute-intensive benchmark applications, such as the CFD solver which is an unstructured grid finite volume solver for compressible flow used in fluid dynamics, were not studied because of the lack of functional output for quality evaluations and the lack of application specific quality metrics.

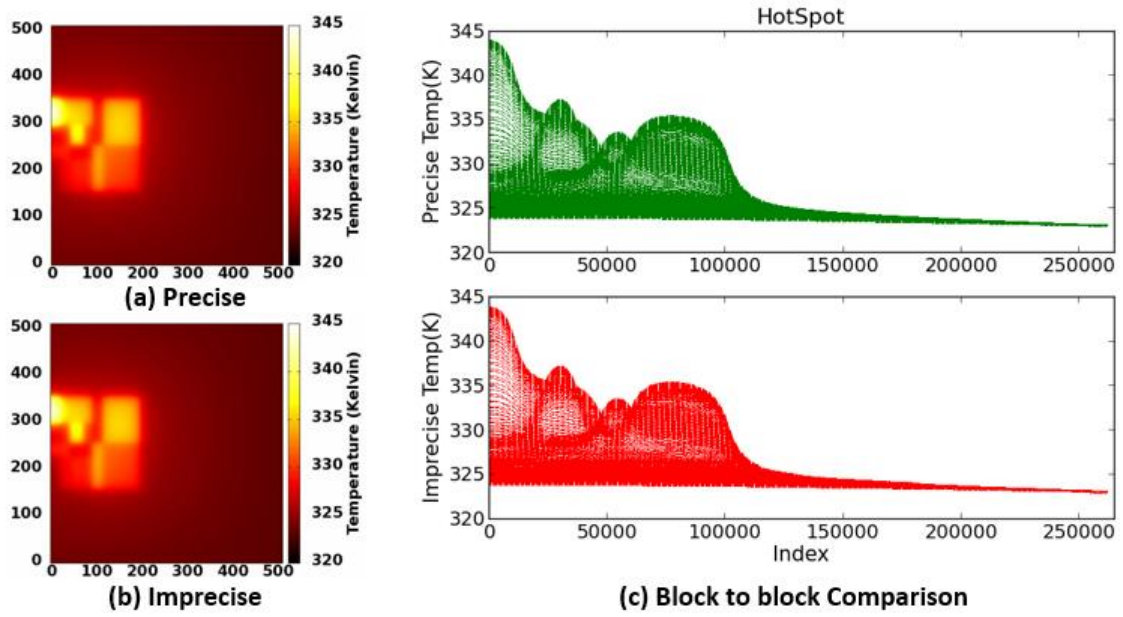


Figure 15 Functional simulation result from Hotspot with estimated 32% GPU power savings

The Hotspot benchmark [28] is a processor temperature simulation model based on an architectural floor plan and simulated power measurements. The thermal simulation iteratively solves a series of differential equations. The compute-intensive kernel consumes nearly 35% of total power consumption by FPUs and SFUs, as shown in Figure 1. The algorithm tends to iteratively average out errors, and by performing functional simulation using GPGPU-Sim with all proposed IHW components enabled, there is almost no perceptible quality degradation, with a mean absolute error of 0.05 Kelvin, and mean square error of 0.003 K for all temperature blocks. Figure 15 (a)-(b) show the temperature "hot spots" in a 512 by 512 block processor for both the "precise" and imprecise results. Figure 15(c) compares the simulated temperature block by block. Each peak represents a local temperature "hot spot" that is the area of interest. The two temperature distributions have almost identical temperature peaks. Using the system-level power evaluation model, the application of IHW in HotSpot achieves about 32.06% power savings by employing all

proposed IHW components, due in large part to the 91.54% power savings from the SFU and FPU, as shown in Table 5.

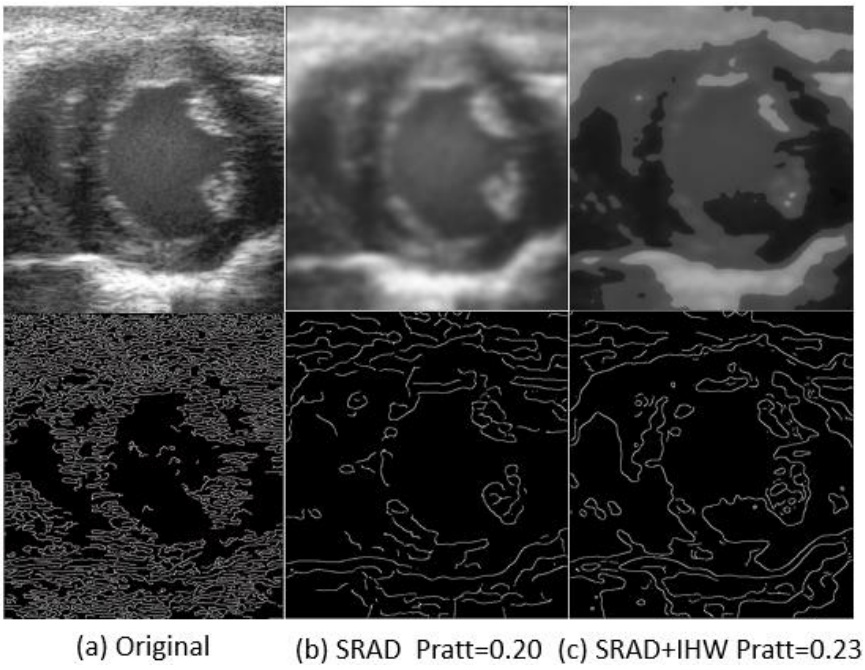


Figure 16 SRAD with an estimated 24% GPU power savings

SRAD (Speckle Reducing Anisotropic Diffusion) [29] is a diffusion method for ultrasonic and radar imaging applications based on partial differential equations (PDEs). It is used to remove locally correlated noise, known as speckles, without destroying important image features. Figure 2 shows that the computational kernel of SRAD consumes around 27% of total power by FPUs and SFUs. The application was simulated on GPGPU-Sim with all IHW components enabled following the methodology outlined in Chapter 6.1. The images on the top row of Figure 16 are ultrasonic images, while the images on the bottom row are binary-edged segmentation maps that are the basis for quality evaluation, as measured by Pratt's figure of merit (from 0 to 1) in the original SRAD work [30]. Figure 16(a) represents the original ultrasonic image and the ideal segmentation, 16(b) shows the image and segmentation after precise SRAD processing, and 16(c) shows the same results

when utilizing the proposed IHW. The precise version has a Pratt's figure of merit of 0.20, while the imprecise results in a slightly higher 0.23, showing that the noise of the imprecise processing is dwarfed by the real-world image noise. The power evaluation model reports a significant 24.23% system level power savings for the GPU.

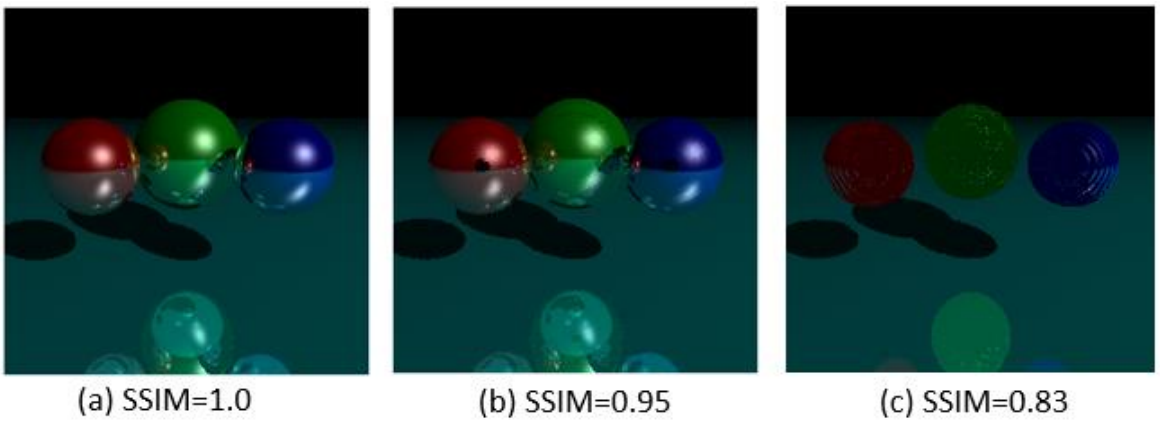


Figure 17 Ray tracing algorithm with an estimated 10% (b) and 12% (c) GPU power savings

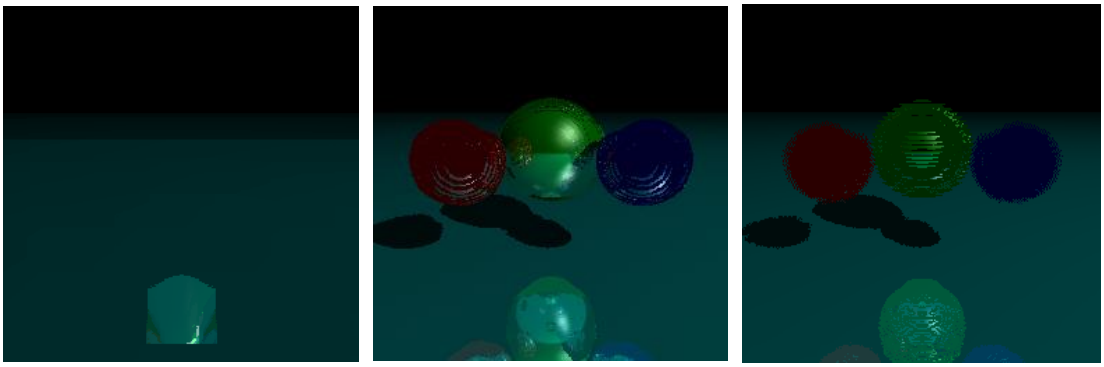
The third application is a ray tracing algorithm for 3D graphics applications. The application is obtained from the ISPASS benchmark included in the GPGPU-Sim simulation tool. The total power consumption from FPU and SFU is about 28%. Since the output is a graphic image with 3D objects, we used the structural similarity index (SSIM) as the quality metric for better capturing the structural content in the image [31]. SSIM is expressed as a value between 0 and 1, with 1 corresponding to perfect quality. Figure 17 shows the quality degradation based on the different IHW components proposed in Table 1. When only reciprocal, floating point addition/subtraction, and square root functions are used as in Figure 17(b), the SSIM is as high as 0.95 and the system-level GPU power savings is about 10.24% as shown in Table 5. When the imprecise inverse square root function is added, the quality drops to 0.83, but the power savings improved to about 11.5%.

Due to the nature of error compounding in the ray tracing algorithm, the application is not as error resilient as the other two, and 11.5% power savings is about the maximum that can be achieved using imprecise components proposed in Table 1 without significantly degrading the structural content of the image. It is found out that ray tracing is very sensitive to floating point multiplication operations as significant amount of floating point multiplication are present during various dot product and cross product calculations during reflection angle and surface normal computation that have significant impacts on the image quality.

Table 5 System level power savings for some compute intensive GPU applications

Applications	Holistic Power Savings	Arith. Power Savings
Hotspot	32.06%	91.54%
SRAD	24.23%	90.68%
RAY(rcp,add,sqrt)	10.24%	36.14%
RAY(rcp,add,sqrt,rsqrt)	11.50%	40.59%
RAY(rcp,add,sqrt,fpmul_fp*)	13.56%	47.86%

* The full path configuration of improved floating point multiplier



(a) (b) (c)

Figure 18 Power-quality improvement for Ray tracing application using the improved floating point multiplier

(a) Orignal ifpmul design in Table 1
(b) rcp,fpadd,sqrt,full path fpmul tr=0; SSIM = 0.85,13.56% System level power savings
(c) rcp,fpadd,sqrt,full path fpmul tr=15; SSIM = 0.79,15.37% System level power savings

Figure 18 shows the output of the RAY tracing algorithm when the imprecise floating point multiplication function proposed in Table 1 is applied together with the configuration in Figure 17(b), which has turned on imprecise floating point addition/subtraction, reciprocal, and square root functions. As it could be seen from the Figure 18(a), RayTracing algorithm is very sensitive to the precision of floating point multiplication, as significant amount of floating point multiplications are used in ray reflection and surface normal calculations. The large error presented in the multiplication operations could result in wrong ray direction and surfaces calculations. Since the rays are reflected multiple times before hitting the predefined end point in the 3D space, the errors can accumulate very quickly. This presents a strong barrier and challenge in achieving further power quality tradeoff. Even though we could achieve only slight quality degradation when only floating point addition/subtraction, reciprocal, and square root are applied, when the original imprecise floating point multiplication is added, the quality significantly drops, in which most of the spheres are no longer visible, and the image is most likely unacceptable to any applications. The large error magnitude from the original proposed imprecise floating point multiplier therefore limits the possibility of power quality tradeoff for RayTracing algorithm. Figure 18(b) shows the generated image when we apply the improved floating point multiplier with full path configurations. The image quality slightly degrades from 0.95 to 0.85, but the holistic power savings improved from 10% to 13.56%. The improved floating point multiplier was also applied to previously studied Hotspot and SRAD. However, since these two applications are not as sensitive to floating point multiplications as RayTracing, the system level improvement on quality and power savings are less 1%.

The next section presents more detailed study on the improved imprecise floating point multiplier based on Mitchell's algorithm and shows that conventional intuitive bit truncations schemes are far less optimal in the power – quality tradeoff design space for application sensitive to floating point multiplication inaccuracies.

5.3.2 Application level study for the improved floating point multiplier

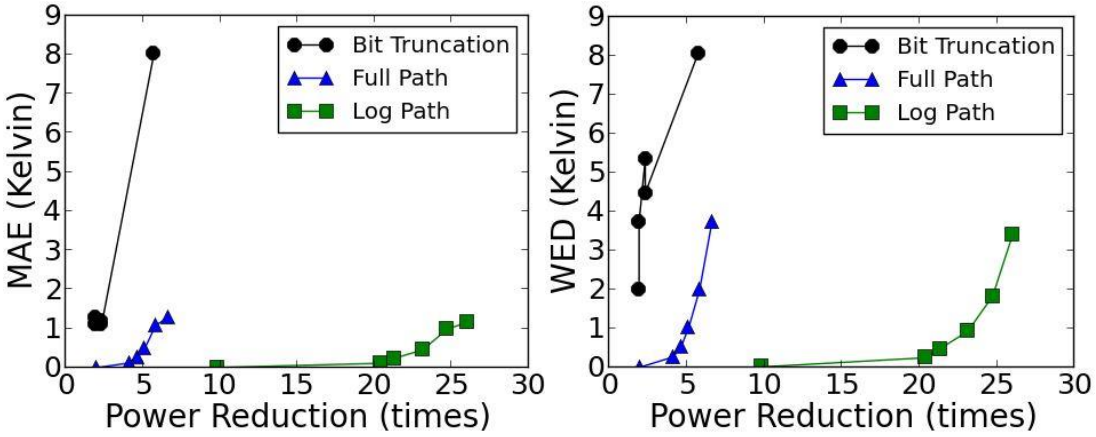
Table 6 CPU and GPU Benchmarks Summary

Benchmarks	Single Precision	Double Precision	Quality Metric	Application Domain
Hotspot°	3.7M(100%)*	0	MAE,WED	Physics Simulation
CP°	32.9M(80%)	0	MAE,WED	Ion placement
RayTracing°	12.4M(36%)	0	SSIM	3D Graphics
172.art•	0	3.17B(89%)	Vigilance⋆	Neural Network
435.gromacs•	0	5.9B(100%)	Err%	Molecular Dynamics
482.sphinx•	3K(100%)	15.6B(100%)	Accuracy†	Voice Recognition

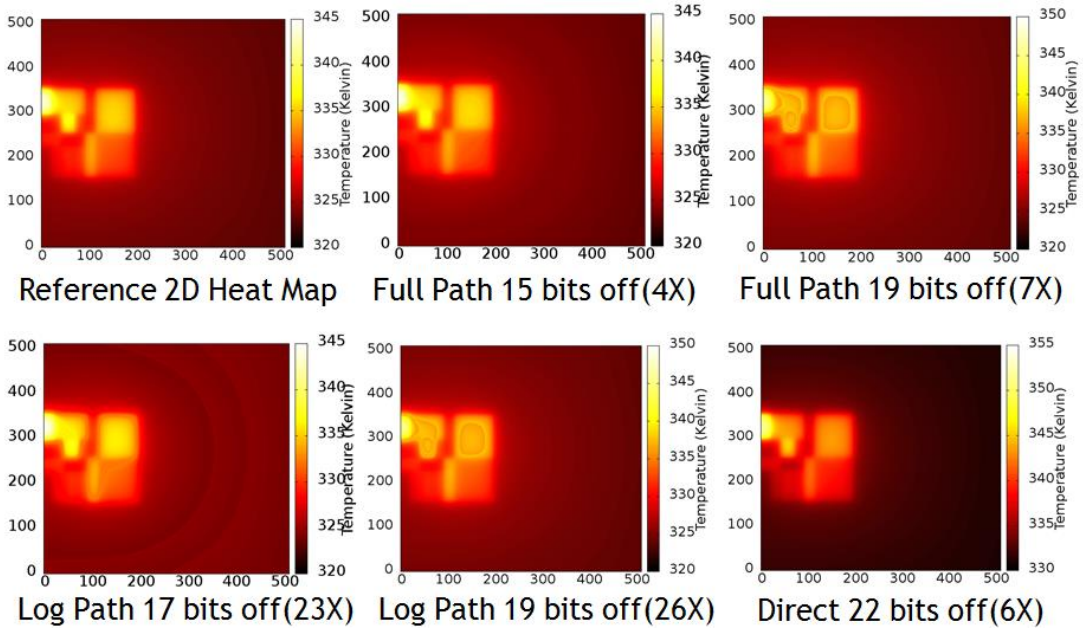
° GPU benchmark in CUDA, • CPU benchmark
* Percentage of operations using accuracy configurable FP multiplier
⋆ Confidence of an object match, † Number of words correctly recognized

To demonstrate the applicability of the proposed accuracy configurable FP multiplier to different algorithms and computation platforms under various accuracy configurations and show that it is significantly better than direct bit truncation schemes used previously, we studied the quality of results with three representative GPU benchmarks from Rodinia benchmark suit [16] and ISPASS2009 [17], and also three CPU benchmarks from SPEC2000 and SPEC2006. A high-level functional model of the proposed multiplier is written in C and CUDA. The CUDA version is used for GPU benchmaks and the C version is used for CPU benchmarks. The correctness of the functional model was verified against the hardware behavior model through extensive functional simulations. The benchmarks are picked based on their dominance of floating

point multiplications as well as the availability of application outputs for quality evaluation. Table 6 lists a summary of all six benchmarks studied in this work.



(a) MAE vs. Power Reduction (b) WED vs. Power Reduction



(c) Heatmap generated by various configuration of accuracy configurable fp Multiplier

Figure 19 HotSpot power-quality tradeoff with improved accuracy configurable floating point multiplier

Hotspot [28], as introduced previously, is a processor temperature simulation model based on an architectural floor plan and simulated power measurements which iteratively solves a series of differential equations. The compute-intensive benchmark kernel has about 3.7 Million single precision floating point multiplications. The imprecise simulation outputs were obtained by replacing all floating point multiplications in the kernel with the functional model written in CUDA. For each configuration, the mean absolute error (MAE) and worst error distance (WED) for all temperature blocks were calculated. The 26X power reduction configuration (19 bits truncated in log path) produces a MAE of 1.2 Kelvin, while the 22 bits intuitive bit truncation on a "precise" multiplier has about 8 times larger MAE with only 6X power reductions. As seen from Figure 19(a) and 19(b), the proposed multiplier shows significant better quality vs. power trade-off compared to the bit truncation schemes. Figure 19(c) show the 2D temperature maps from three worst-case quality but lowest-power configurations.

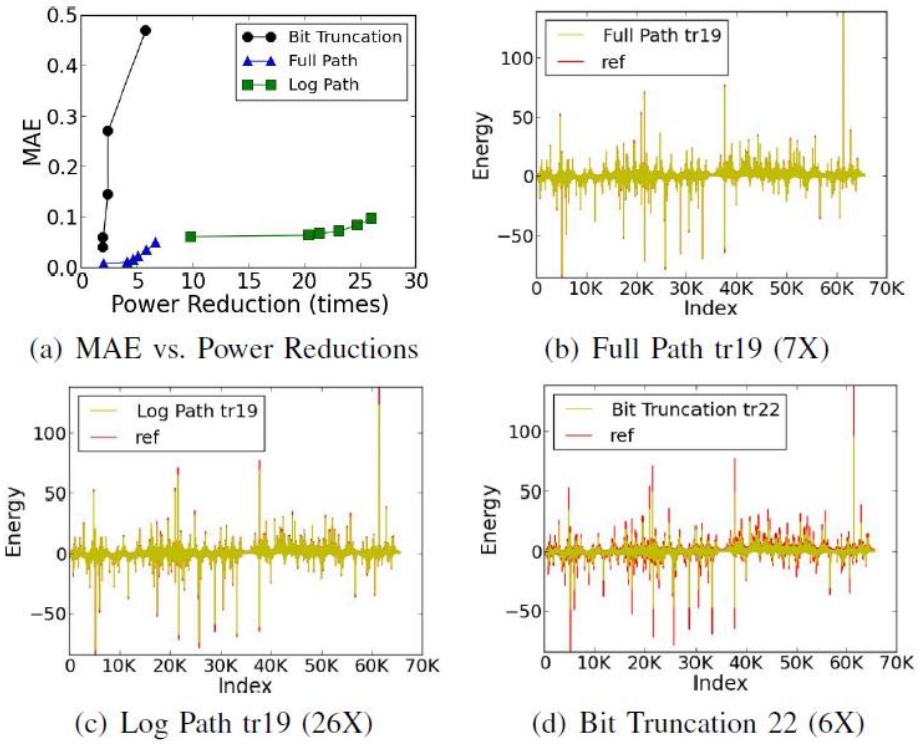
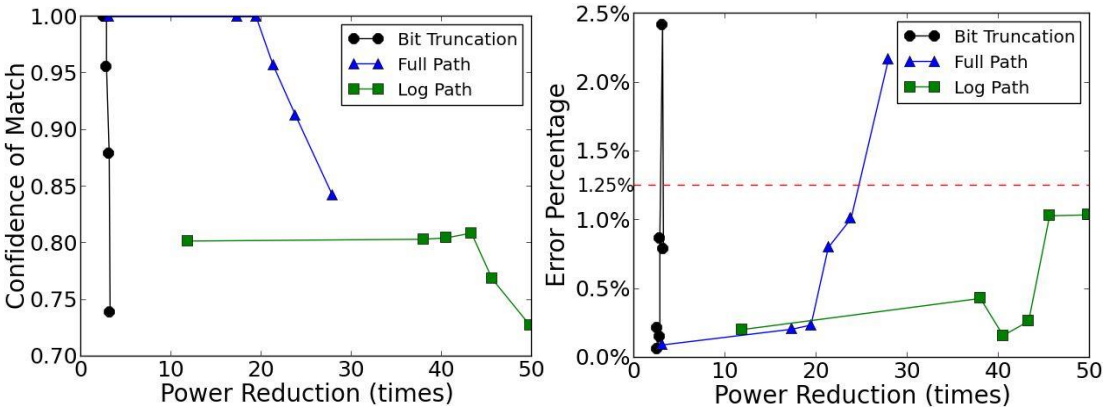


Figure 20 CP benchmark power-quality trade-off

CP (Coulomb Potential) is a GPGPU application used for placing counterions near a biological molecule in preparation for molecular dynamics simulations. Same as Hotspot, mean absolute error (MAE) was used as the figure of merit for CP. Figure 20(a) shows the MAE for all accuracy configurations. As can be seen from the figure, the proposed FP multiplier has a consistently lower MAE and larger power reduction across all configurations. Figure 20(b)-(d) demonstrates the resulting energy differences between the specific configuration and the reference for each atom. It is worth-noting that out of all floating point multiplication operations, about 20% was kept precise as these were used for determining the coordinates of each atom in a 2D grid.

179.art (The Adaptive Resonance Theory 2 (ART 2) neural network) is a floating point CPU benchmark from SPEC2000 used for recognizing objects in a thermal image.

As seen from Table II, the benchmark contains a total of 3.17 Billion double precision floating point multiplications. The input to the benchmark contains thermal images of a helicopter and an airplane. The output contains the coordinates of the recognized object as well as a confidence of match (vigilance). Therefore, we choose to use the confidence of match as the quality metric for this application given the correct object coordinates. Figure 21(a) shows that the vigilance from intuitive bit truncation drops abruptly as the more bits were truncated, while a slow slope can be obtained by using the proposed accuracy configurable FP multiplier. With 26X power reduction, the accuracy configurable FP multiplier can still maintain a confidence of match above 0.8.



(a) 179.art (b) 435.gromacs

Figure 21 179.art, 435.gromacs: power-quality trade-off

435.gromacs is a floating point CPU benchmark from SPEC2006 performing a simulation of the protein Lysozyme in a solution of water and ions. The input to the benchmark is set to be the default 6000 iteration steps. The output contains the average potential energy and system temperature. According to SPEC2006 documentation [19], molecular dynamics simulations by default are chaotic processes, therefore an error

percentage within 1.25% for the outputs is considered to be correct. The error percentage compared to the reference value given by the benchmark therefore is used here as a quality metric. The results from various accuracy configurations as well as the bit truncation schemes are shown in Figure 21(b). All points below the dashed line at 1.25% can be considered correct. In this benchmark, the log path of the FP multiplier has a better QoR than the full path. This is counter-intuitive and might be caused by the randomness within the application.

Table 7 482.Sphinx3: quality of results

Config	Accuracy†	Config	Accuracy	Config	Accuracy
bt_44	24/25	fp_tr0	25/25	lp_tr0	25/25
bt_45	24/25	fp_tr44	24/25	lp_tr44	21/25
bt_46	24/25	fp_tr45	24/25	lp_tr45	21/25
bt_47	25/25	fp_tr46	24/25	lp_tr46	21/25
bt_48	25/25	fp_tr47	25/25	lp_tr47	23/25
bt_49	22/25	fp_tr48	24/25	lp_tr48	24/25

† Number of words correctly recognized

482.sphinx3 is a voice recognition benchmark adapted from the speed recognition system Sphinx-3 [32]. For a reasonable and feasible simulation time, we randomly chose 5 raw audio streams (an391 - an395) from the AN4 database that comes with the benchmark. The 5 raw audio streams contains a total of 25 words. Since the quality of the result depends on the words interpreted by the system, we use the number of words correctly recognized as a quality metric. The results are shown in Table III. The intuitive bit truncation schemes has a good accuracy until 49 bits are truncated in the double precision multiplication. However, the full path configuration miss recognized at most one word across all six accuracy configurations. Even though these two configurations has similar quality characteristics, the full path configuration has a much larger power

reductions. The log path does not perform very well in this application compared to the other two.

Chapter 6 Conclusions and Future Work

In this thesis, we demonstrated that:

1. Significant GPU system level power savings could be achieved by applying floating point IHW for compute-intensive applications with negligible quality degradation.
2. Traditional bit truncation schemes employed in floating point multiplication is sub-optimal in the power-quality design paradigm with inferior quality and limited power savings.

The significant impact from IHW is mainly contributed by the following three factors:

- The error resiliency of floating point arithmetic in compute-intensive GPGPU applications, which enables the use of IHW.
- The high execution frequency of these arithmetic operations, which sets the power consumption upper bound at the system level.
- The massively parallel homogeneous computation cores in the GPGPU architecture, which provide a multiplicative effect for system level power savings.

The collective impact of these three factors was demonstrated with the evaluation of IHW under several benchmarks in scientific computing, achieving up to 32%, 24%, and 13% system-level power savings respectively, while maintaining an acceptable level of output quality for each. The system level power savings from IHW can be further improved by combining with DVFS and power gating techniques. In addition, the proposed improvement on the low power accuracy configurable floating point multiplier based on Mitchell's Algorithm demonstrates that dramatic (49X) power reductions could be achieved when applying bit truncations on top of the algorithmic transformed FP multiplier.

As shown in the application level analysis, the wide range of accuracy configurations allows a fine grained quality tuning for various error tolerant applications.

Future work includes refining the system level power quality estimation framework, enabling more structural parameters of IHW components to expand the design space, and adding more control knobs for tuning output quality. One limitation of the proposed floating point multiplier is that it is inherently imprecise. Therefore, for applications that are partially error tolerant such as RayTracing, a “precise” floating point multiplier may be required for obtaining a good quality of result (QoR). Some future work include integrating the “precise” mode into the floating point multiplier and developing an automatic quality tuning model for applications that are partially error tolerant.

References

[1] NIVIDA, “Whitepaper NVIDIA’s Next Generation CUDA Compute Architecture” pp. 1–22. [Online] Available: http://www.nvidia.com/content/pdf/fermi_white_papers/nvidia_fermi_compute_architecture_whitepaper.pdf

[2] Intel, “Intel ® Core i7-3900 Desktop Processor Extreme Edition Series GHz,” 2011. [Online]. Available: http://download.intel.com/support/processors/corei7ee/sb/core_i7-3900_d_x.pdf.

[3] NVIDIA, “NVIDIA’s Next Generation CUDA Compute Architecture: Kepler GK110” [Online]. Available: http://www.nvidia.com/content/PDF/kepler/NVIDIA-kepler-GK110-Architecture-Whitepaper.pdf

[4] M. Weber, M. Putic, and H. Zhang, “Balancing Adder for error tolerant applications” *ISCAS, pp. 3038-3041, May 2013*.

[5] A. K. Verma, P. Brisk, and P. Ienne, “Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design” *DATE*, pp. 1250–1255, Mar. 2008

[6] J. Huang, “A Digital System Design Methodology for Efficiency-Quality Tradeoffs Using Imprecise Hardware” Ph.D. Dissertation, Computer Engineering, University of Virginia, Charlottesville, VA, 2012.

[7] A. B. Kahng and S. Kang, “Accuracy-Configurable Adder for Approximate Arithmetic Designs” *DAC*, pp. 820–825. 2012.

[8] J. Ying, F. Tong, D. Nagle, and R. A. Rutenbar, “Reducing Power by Optimizing the Necessary Precision / Range of Floating-Point Arithmetic” *IEEE Transactions on VLSI* ,vol. 8, no. 3, pp. 273–286, 2000.

[9] K. Du, P. Varman, and K. Mohanram, “Static Window Addition : A New Paradigm for the Design of Variable Latency Adders” *ICCD*, pp. 455–456, 2011.

[10] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, “IMPACT : IMPrecise adders for low-power Approximate CompuTing” *ISLPED*, pp. 409–414, 2011.

[11] J. Huang and J. Lach, “Exploring the fidelity-efficiency design space using imprecise arithmetic” *ASP-DAC*, pp. 579–584, Jan. 2011.

[12] J. Huang, J. Lach, and G. Robins, “A methodology for energy-quality tradeoff using imprecise hardware” *DAC*, p. 504, 2012.

[13] J. Huang, J. Lach, and G. Robins, “Analytic Error Modeling for Imprecise Arithmetic Circuits” *Silicon Errors in Logic - System Effects (SELSE)*, 2011.

[14] K. E. Wires, M. J. Schulte, and J. E. Stine, "Variable-Correction Truncated Floating Point Multipliers" *Signals, Systems and Computers,* no. 1, pp. 1344–1348.

[15] A. Gupta, S. Mandavalli, V. J. Mooney, K.-V. Ling, A. Basu, H. Johan, and B. Tandianus, "Low Power Probabilistic Floating Point Multiplier Design" *2011 IEEE Comput. Soc. Annu. Symp. VLSI*, pp. 182–187, Jul. 2011.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing" *2009 IEEE Int. Symp. Workload Charact.*, pp. 44–54, Oct. 2009.

[17] J. Leng, T. Hetherington, A. Eltantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch : Enabling Energy Optimizations in GPGPUs Categories and Subject Descriptors" *ISCA*, pp. 487-498, June 2013.

[18] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. San Francisco, Calif: Morgan Kaufmann ;Oxford :Elsevier Science, 2004.

[19] P. Tak and P. Tang, "Table-Driven Implementation of the Logarithm Function in IEEE Floating-Point Arithmetic" *ACM Transactions on Mathematical Software (TOMS)*, vol. 16, no. 4, pp. 378–400, 1990.

[20] J. E. Stine and M. J. Schulte, "The Symmetric Table Addition Method for Accurate Function Approximation" *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 12, pp. 1–12, 1999.

[21] W. Liu and A. Nannarelli, "Power Efficient Division and Square Root Unit" *IEEE Trans. Comput.*, vol. 61, no. 8, pp. 1059–1070, Aug. 2012.

[22] D. De Caro, N. Petra, and A. G. M. Strollo, "A High Performance Floating-Point Special Function Unit Using Constrained Piecewise Quadratic Approximation" *IEEE Transactions on Circuits and Systems*, pp. 472–475, 2008.

[23] M. a. Cornea-Hasegan, R. a. Golliver, and P. Markstein, "Correctness proofs outline for Newton-Raphson based floating-point divide and square root algorithms" *Proc. 14th IEEE Symp. Comput. Arith.* pp. 96–105.

[24] R. E. Caflisch, "Monte Carlo and quasi-Monte Carlo methods" *Acta Numer.*, vol. 7, p. 1, Nov. 2008.

[25] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator" *2009 IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 163–174, Apr. 2009.

[26] S. Hong and H. Kim, "An integrated GPU power and performance model" *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 280, Jun. 2010.

[27] J. Han and M. Orshansky, "Approximate Computing : An Emerging Paradigm For Energy-Efficient Design" *IEEE European Test Symposium (ETS)*, p. 1-6, 2013

[28] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture" *ISCA*, p. 2, 2003.

[29] Y. Yu and S. T. Acton, "Speckle reducing anisotropic diffusion" *IEEE Trans. Image Process.*, vol. 11, no. 11, pp. 1260–70, Jan. 2002.

[30] A. J. Pinho, D. Electrnica, and T. Inesc, "Figures of merit for quality assessment of binary edge maps" *ICIP*, pp. 591–594, 1996.

[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity" *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–12, Apr. 2004.

[32] "CMU sphinx." [Online]. Available: http://cmusphinx.sourceforge.net/.

[33] H. Zhang, M. Putic, and J. Lach, "Low Power GPGPU Computation with Imprecise Hardware" *DAC*, pp. 1–6, 2014.

[34] H. Zhang, W. Zhang, and J. Lach, "A Low-Power Accuracy-Configurable Floating Point Multiplier" *ICCD*, Oct. 2014.