ENERGY EFFICIENT FLOATING-POINT UNIT DESIGN

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Sameh Galal

November 2012

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mark Horowitz, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

William Dally, Co-Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Oyekunle Olukotun

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mark Horowitz) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(William J. Dally)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Kunle Olukotun)

Approved for the University Committee on Graduate Studies.

Abstract

Energy-efficient computation is critical for increasing performance in power limited systems. Floating point performance is of particular interest because of its importance in scientific computing, graphics and multimedia processing. For floating-point applications that have large amounts of data parallelism one should optimize the throughput/mm^2 given a power density constraint. We present a method for creating a trade-off curve that can be used to estimate the maximum floating-point performance given a set of area and power constraints. These throughput optimized designs turn out to be different from latency optimized ones and more energy efficient. Looking at floating-point multiply-add units and ignoring register and memory overheads, we find that in a 90nm CMOS technology at 1W/mm^2, one can achieve a performance of 27GFlops/mm^2 single-precision, and 7.5GFlops/mm^2 double-precision. Adding register file overheads reduces the throughput by less than 50% if the compute intensity is high. Since the energy of the basic gates is no longer scaling rapidly, to maintain constant power density with scaling requires moving the overall FP architecture to a lower energy/performance point using lower supply voltage, shallower pipelines and more relaxed gate sizing. A 1W/mm^2 design at 90nm is a "high-energy" design, so scaling it to a lower energy design in 45nm still yields a 7× performance gain, while a more balanced 0.1W/mm^2 design only speeds up by 3.5× when scaled to 45nm. Performance scaling below 45nm rapidly decreases, with a projected improvement of only 2-3 for both power densities when scaling to a 22nm technology.

On the other hand, some floating point units employed for single threaded performance such as CPU designs are latency sensitive. For such designs a different optimization in the implementation of fused floating-point multiply-add operations

can be utilized. By realizing that the average latency of all operations going through the unit is what matters most, an optimized cascade design can reduce the accumulation dependent latency by 2× over a fused design, at a cost of a 13% increase in non-accumulation dependent latency. A simple in-order execution model shows this design is superior in most applications, providing 12% average reduction in FP stalls, and improves performance by up to 6%. Simulations of superscalar out-of-order machines show 4% average CPI improvement in 2-way machines and 4.6% in 4-way machines. This feat is achieved by a design architecture called *cascade*, where the addition operation is cascaded after multiplication in comparison to traditional architectures. The cascade design has the same area and energy budget as a traditional FMA.

Acknowledgments

It's hard to sum up how an amazing and a transformative experience the past few years have been at Stanford. The great people I have had the opportunity to work and interact with are truly exceptional. I would like to thank professor Mark Horowitz, my advisor for the amazing mentorship, help, patience and support. Especially in difficult times through this journey, Mark was the best and most supportive advisor I could ever hope for. I would like also to thank professors Bill Dally and Kunle Olukotun for serving on my reading and defense committee and their comments and feedback on this dissertation.

I had the pleasure to work and be friends with a great group of fellow grad students. In particular I would like to thank : Ofer Shacham, the generators guru with which I enjoyed working on converting this work to a useful generator; Megan Wachs, my thesis boot camp buddy with whom I spent countless hours writing this dissertation; Omid Azizi, who mentored me into the optimization field; Zain Asgar, Pete Stevenson and all the rest of the chip generator group. Special thanks also to Stephen Richardson for his support and help improving this dissertation; and Stuart Oberman for the discussions we had on floating point design.

I would like to thank the great friends that I had throughput the years here: Bernd Bandemer, Luis Adarve, Hai Nguyen, Vitali Brand, Alex Becka, Atlal Laouar and Mohamed Zaghoo. I am also quite indebted to Annuschka and Rajat Deb who were my family away from home whose kindness and support made a huge difference on me. Finally I would like to thank my parents who supported me all along from thousands of miles away.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer performance has been increasing exponentially in the last half century, driven by improvements in architecture, circuit design and technology. Historically, chip design was focused on maximizing performance within a constrained die area with the niche of mobile and battery held devices focused on low power design. This decade, even high-performance designs have transitioned from being transistor/complexity limited to being power limited. This change in design constraints has had a significant impact on system design as increase in performance can come only from lowering the energy of operations. This is now a serious issue since the energy savings offered by scaling has slowed down dramatically in recent technology nodes.

Interestingly, the energy per operation depends on performance (ops/sec): as the required performance increases, the energy to perform each operation also increases. This energy-performance relationship is one of the factors driving the trend towards chip multiprocessors. By reducing the peak performance of each processor, we can decrease its energy/instruction. Thus for the same power, we can execute more instructions /sec. Of course, to make it more energy-efficient each processor has lower peak performance than before; so to achieve the power limited instruction issue rate, we need to integrate more processors on to the die. The resulting machine, for parallel applications, can deliver more performance at the same power than the previous uniprocessor designs.

This dissertation explores how to optimize floating-point (FP) functional units in

this energy constrained design space. Floating point unit designs have been studied extensively and are the backbone of scientific computation and computer graphics. This work looks specifically at floating point units based on the fused multiply add operation and its variants. Currently, FPUs exist in two large differentiated segments: CPU's and GPU's. Ever since the integration of x87 floating point coprocessor in the Intel 486, the floating point unit has been an integral part of CPU performance. The design of an FPU for CPUs is highly latency sensitive as it is designed for single threaded operation. On the other hand, recent GPUs employ thousands of FPU's working together on highly parallel work loads. This high parallelism allows GPUs to have generally superior energy efficiency and higher floating point performance than CPUs. For such designs, the total throughput of the aggregate parallel units is more important than the latency of each individual FPU since latency can be hidden by interleaving of the execution of parallel threads. Both designs pose different challenges and design questions.

In the first part of the dissertation, we look at how to design an energy efficient throughput system. Chapter 3 introduces the rationale for optimizing throughput machines. For parallel systems, the latency or even the throughput rate per processor is not the critical optimization parameter, since changing the design changes the number of units we can fit on the die. Instead we optimize the number of results/sec/mm^2 remembering that very small, slower units might be the best solution. Thus, for parallel systems, the main tradeoff is between energy/op and ops/sec/mm^2, so power density becomes a critical design metric. Chapter 4 presents the tradeoffs in FPU design for throughput and the overheads in area and energy for its associated register files. Finally, implemented designs for throughput tradeoffs in 90nm and 45nm show interesting trends of uneven scaling of high power density and low power density designs. This prompts an extended study of scaling of throughput tradeoffs down to 16nm in Chapter 5. The study explores how designs need to change with scaling to track most energy efficient designs.

In the second part of the dissertation, FPU design is approached from the angle of applications that don't have enough parallelism and are latency sensitive such as in CPUs. Here a quantitative study is presented that looks at the different latencies

embedded with design choices in fused multiply add design. Based on this study, a cascade implementation that favors very short accumulation latency over other latencies offers almost 20% improvement in average latency over state of the art design with no overhead in energy or area. Such a design pushes the envelope of energy efficiency tradeoffs by improving performance for same amount of energy.

Chapter 2

Background

2.1 It's a Power Limited World

In 1965, Gordon Moore famously noted that the number of transistors that can be placed inexpensively on an integrated circuit doubles approximately every two years. This trend has been holding remarkably well for almost half a century through the downscaling of transistor dimensions. In 1974, Dennard outlined the theory for scaling MOSFET transistors which the industry has followed consistently until recently [33]. The theory outlined in Table 2.1 stipulates that scaling all the transistor dimensions of a circuit as well as the supply voltage with some scaling factor $1/\kappa$ maintains constant electric fields throughout the devices. This results in linear improvement in circuit delay ($1/\kappa$) and quadratic improvement in power ($1/\kappa^2$) while maintaining constant power density. Moore's law and Dennard scaling have since defined technology scaling and the whole electronics industry.

Despite the constant power delivered by classical Dennard Scaling, the power of microprocessors has been continually rising. Power increased as designers used an ever increasing transistor budget to build more sophisticated architectures, and scaled operating frequencies even faster than Dennard scaling using deeply pipelined architectures. However in the beginning of this decade, designs started hitting a power wall around 130W where system design became increasingly hard in terms of cooling and power delivery as shown in Figure 2.1. This means that not only low

Device or Circuit Parameter	Scaling Factor
Device dimension t_{ox}, L, W	$1/\kappa$
Doping concentration N_a	κ
Voltage V	$1/\kappa$
Current I	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit VC/I	$1/\kappa$
Power dissipation/circuit VI	$1/\kappa^2$
Power density VI/A	1

Table 2.1: Scaling results for circuit performance. κ is unitless scaling factor. reproduced from Dennard et al. [33]

power mobile designs are limited in their power dissipation but also high performance designs have become power limited too. Making matters worse, the scaling of energy per operation predicted by Dennard scaling started slowing down beyond 90nm as it became increasingly hard to lower supply voltage without taking a big performance or leakage current penalty. Scaling supply voltage down requires lowering transistor threshold voltage to get acceptable transistor performance which in turn increases leakage currents exponentially. The net result is that supply voltage is staying roughly constant and the energy/op now only scales proportional to the scaling factor, and consequently power limited performance scales linearly.

2.2 Energy Efficient Design

Since all designs are power limited and power is the product of performance (operations/second) and energy/operation, then the only way to increase performance is to be more energy efficient. First, energy waste in the design has to be eliminated. For example, clock gating prevents gates in a logic block from switching during cycles when their output is not used, reducing clock power and logic dynamic power. Power gating shuts off power supply from design portions when unused for longer periods of time, reducing idle leakage power. Once the strategies for eliminating waste are exhausted, reducing energy comes at the expense of performance as faster designs

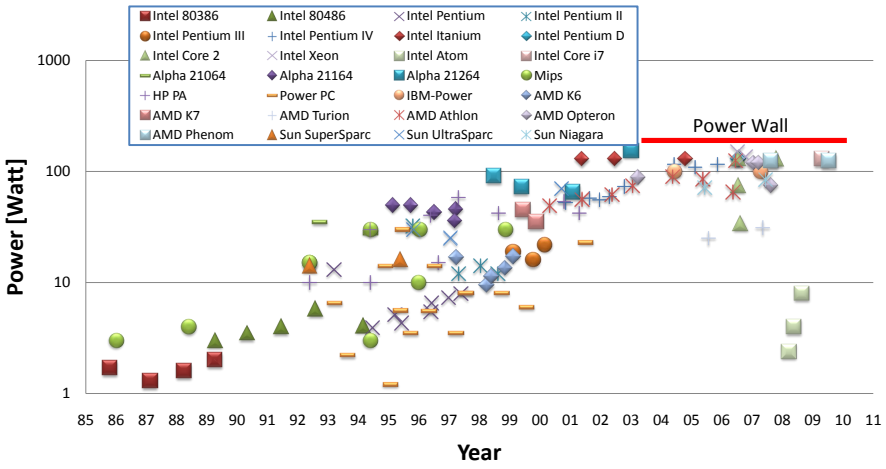


Figure 2.1: Processor power has hit a wall in the last decade.

require faster circuits that employ big transistors and operate at higher voltages, creating a tradeoff between performance and energy per operation. Such a tradeoff even exists for historical data of processor energy and performance normalized for technology as shown in Figure 2.2. The goal of the designer is to choose the best designs that lie on the efficient frontier achieving least energy for a performance target, or achieving maximum performance for a certain power budget.

2.2.1 Design Parameters Optimization

From among the set of all possible designs, designers need to choose those that lie on the efficient frontier of energy versus performance. To do that, they need to tune all their design parameters carefully to get the minimum energy for their performance target. As such, an optimization process is needed to guide the choice of design parameters to achieve a certain position on the efficient frontier. A sensitivity analysis of the different design parameters to energy and performance can be used to assist

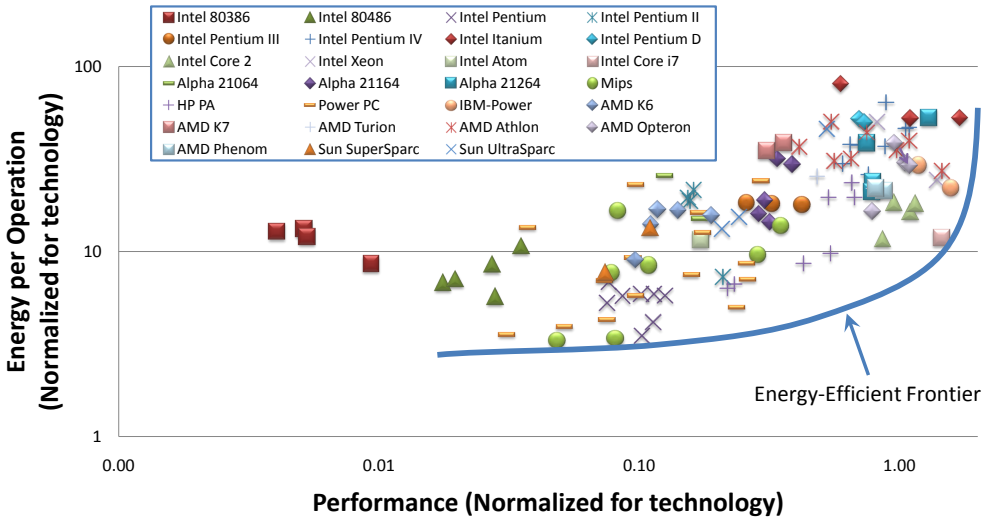


Figure 2.2: Plot of historical processors in the energy-performance space. Designs that maximize performance (to the right) and minimize energy (to the bottom) are preferred, resulting in a trade-off curve.

in this design space search. The sensitivity for an optimization variable x for energy delay tradeoff is defined by S_x; the incremental percent energy (E) per percent reduction in delay (τ) as given by equation (2.1).

$$S_x = -\frac{\tau}{E}\frac{\frac{\partial E}{\partial x}}{\frac{\partial \tau}{\partial x}} \tag{2.1}$$

In an optimal design, the sensitivities of all design parameters have the same value ($S_{V_{dd}} = S_{V_{th}} = ... = S$) and minimize the weighted energy delay product $E\tau^S$ [43]. For example adjusting the design parameters of voltage, threshold and sizing to have sensitivity of 1% increase in energy for every 1% decrease in delay will minimize the product $E\tau$ while optimizing parameters to have sensitivity of 10% increase in energy for every 1% decrease in delay will minimize the product $E\tau^{10}$. Varying S allows the traversal of the efficient frontier from high performance points (large S) to low energy designs (small S).

2.2.2 Push for Parallelism

The tradeoff between energy and performance is one of the factors driving the trend towards chip multiprocessors. If a targeted application can be parallelized to run on several processors instead of one, one can reduce the peak performance of each processor, and consequently its energy/instruction. Thus for the same power, we can execute more instructions per second. Of course, to maximize the power limited instruction issue rate, we need to integrate more processors on the die. The resulting machine, for parallel applications, can deliver more performance at the same power than the previous uniprocessor designs as illustrated in Figure 2.3. One such application class consists of throughput oriented workloads such as the ones handled by GPU. The inherent parallelism of these applications has allowed GPUs over the last decade to have an order of magnitude higher floating point performance than even multicore CPUs as shown in Figure 2.4

Taking the argument of parallelism to its extreme, if the targeted application is inherently parallel and can be divided between as many processors as possible, the

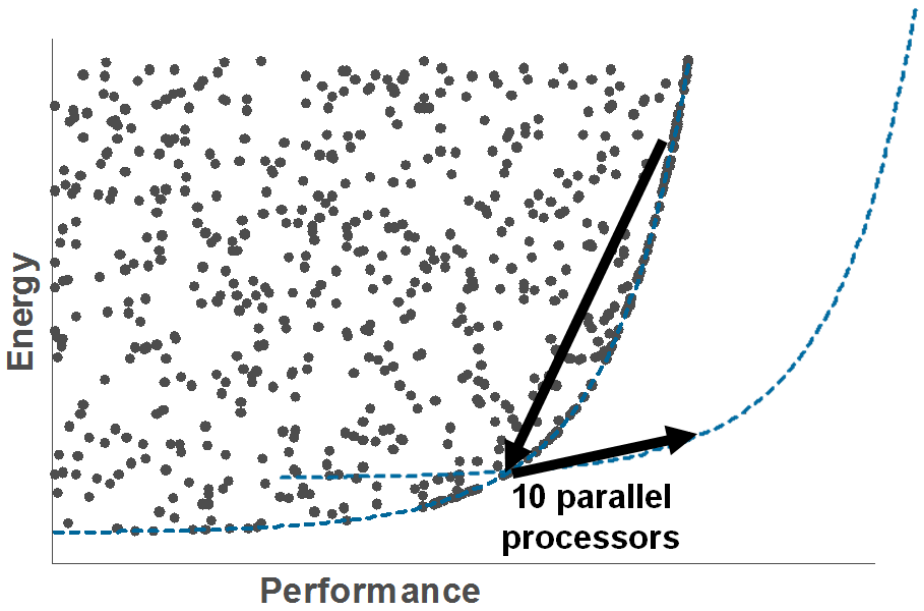


Figure 2.3: Pushing the frontier by exploiting parallelism; if dropping performance by 2× increases efficiency 5×, a 5× improvement in total performance at the same power is achieved by having 10 parallel machines running at half original performance.

resulting efficient design will be the absolute minimum energy design (the lowest to the most left on the efficient frontier of Figure 2.3) with sea of very slow processors and huge area overhead. Such a design seems to be very uneconomical and as such area efficiency should be taken into consideration as well. Chapter 3 discusses how to create energy efficient designs in this highly parallel space for both resource limited and performance limited designs.

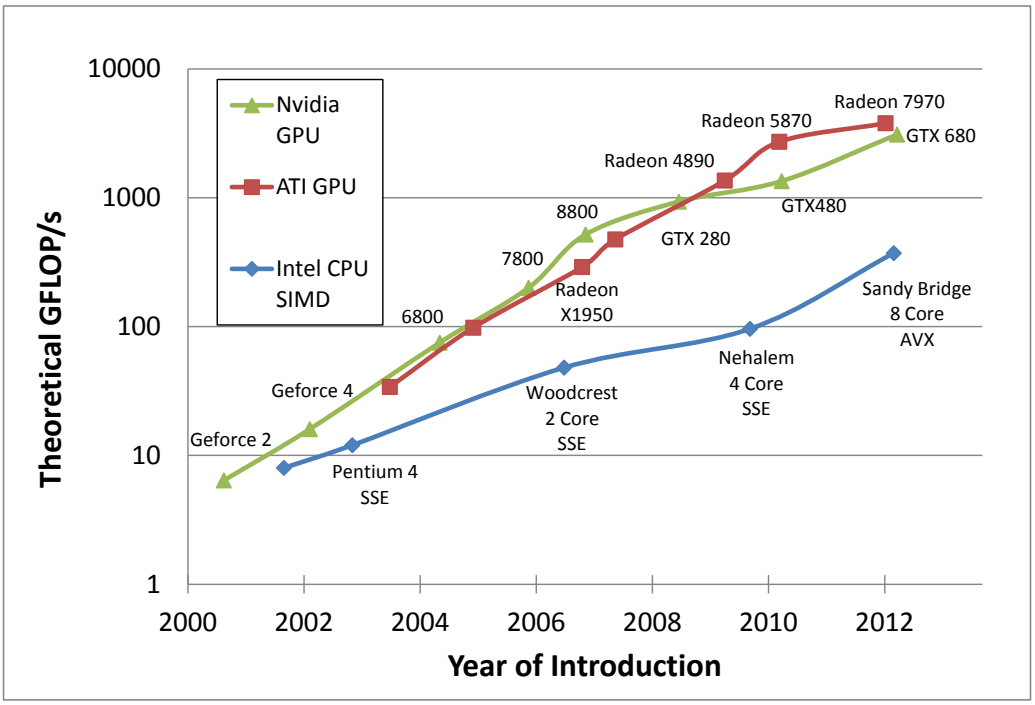


Figure 2.4: Historical Floating-Point Performance for CPUs and GPUs

Chapter 3

Optimizing Throughput Machines

The last chapter introduced the tradeoff between delay and energy per operation. Such a tradeoff suggests that backing off the minimum delay design can yield significant saving in energy. Based on this tradeoff, Chandrakasan proposed the microarchitectural techniques of pipelining and parallel datapaths to achieve lower power designs for a given throughput constraint [7]. Pipelining and parallelism allow the same throughput to be achieved using lower clock frequency as illustrated in Figure 3.1. The increased delay of the circuits allows the use of lower supply voltage, higher threshold voltage and reduced gate sizes resulting in energy savings of 40-70% [27].

Several studies investigated the optimal parallelism and pipelining parameters to minimize total power given a throughput constraint and the limits of its applicability. Using an analysis based on the NTRS technology roadmap parameter predictions, Bhavnagarwala looked at parallel datapaths designs across different technology nodes and concluded that the optimal number of parallel datapaths for minimum power will decrease from 4 in 0.25 μm technology to 2 in 0.05 μm node with power savings shrinking from 80% to 20% [9]. The smaller reduction in total power and the fewer required parallel datapaths required for such reduction are caused by the shallower underlying energy-delay tradeoffs of smaller technologies due to their lower V_{dd}/V_{th} ratios. Markovic also found parallelism to be only useful in minimizing total energy for high performance targets above the minimum energy-delay (ED) product design point (the point at which the marginal cost of energy and performance are equal)

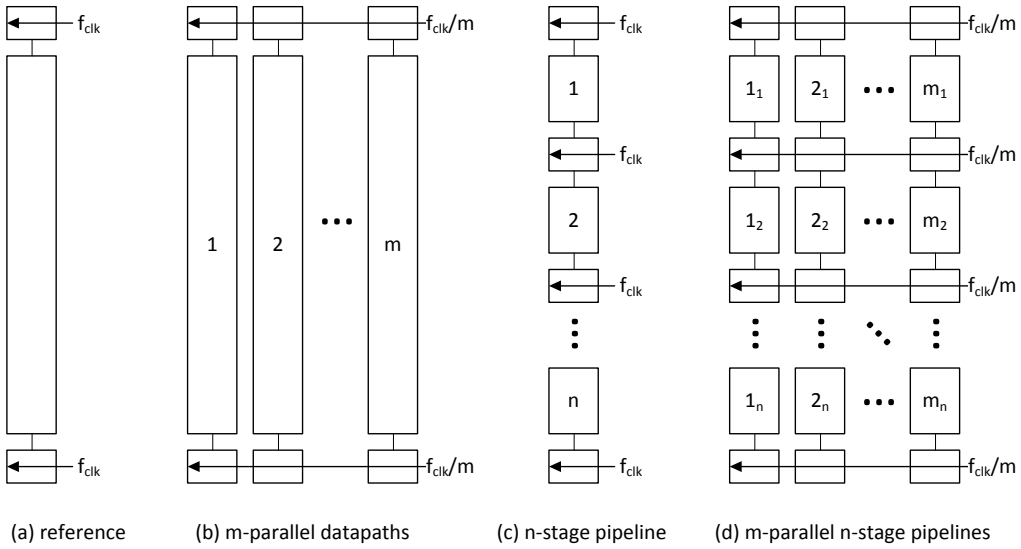


Figure 3.1: Throughput equivalent designs employing parallelism and pipelining

[27]. As for these high performance targets, energy savings due to parallelism are substantial in comparison to the overheads. For optimal pipelining, Hartstein found that unpipelined microprocessor designs are minimum E and minimum ED designs while pipelined microprocessors with pipeline depth of 22.5 FO4 minimized the ED^2 product [19].

While energy savings from parallelism and pipelining are similar, parallel datapaths consume more area than pipelining. Markovic et al. introduced a methodology for minimizing power and area given throughput and latency constraints that employ parallelism, pipelining and time-multiplexing [26]. The latency constraints provide an upper bound on the delay that allows the choosing of a minimum energy design that meets the latency bound and then the microarchitectural alternatives are compared to choose the design with the least area. We have also found that even for latency optimization only, pipelining is a powerful tool for minimizing leakage energy by dividing the leakage power cost across several operations. Figure 3.2 illustrates how the minimum energy design for latency targets of 5ns in 90nm single-precision floating-point fused multiply add unit is actually a 3 stage pipelined design. While the dynamic energy of such a design is increased due to the inserted pipelining flip-flops,

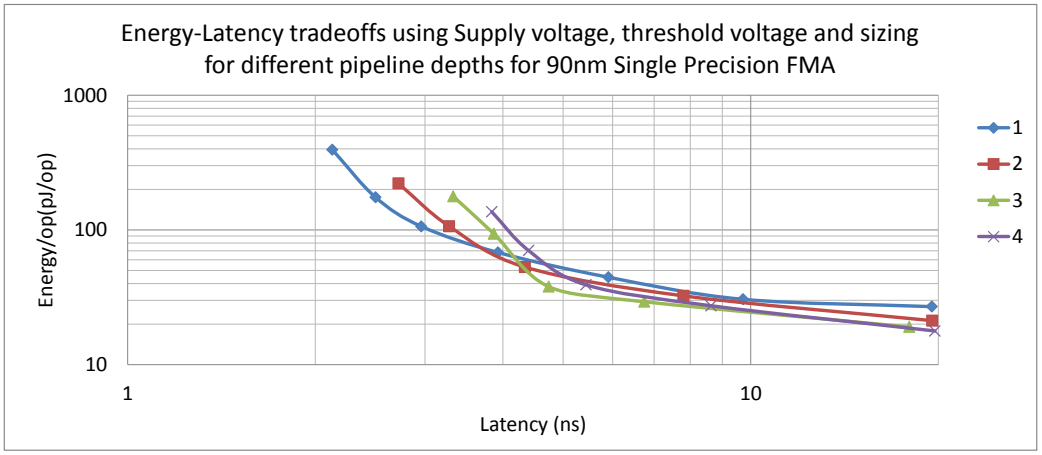


Figure 3.2: Energy latency tradeoff shows that pipelining is a useful optimization tool even for latency. For example a 3 stage pipeline has lower total energy than an unpipelined design for the 5ns latency target due to lower leakage energy.

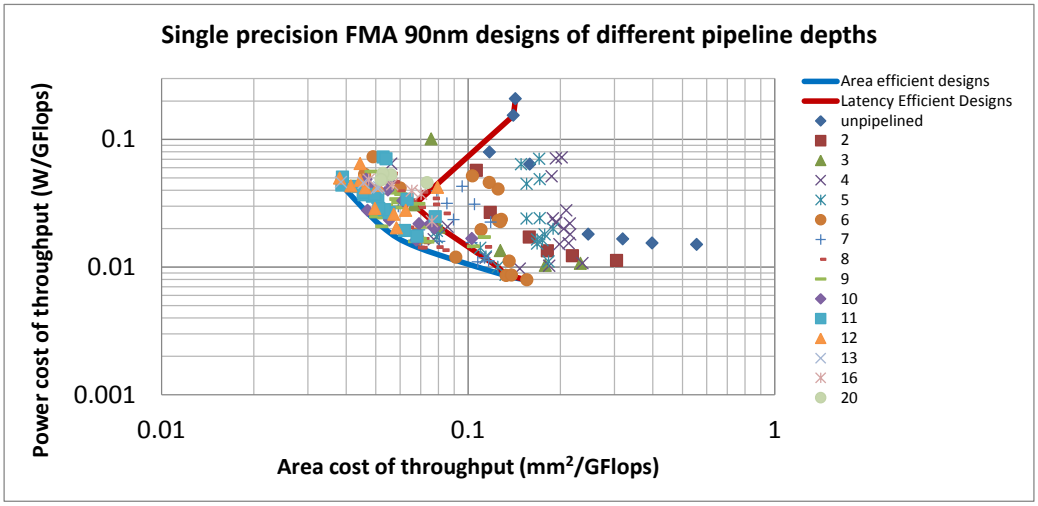
the decrease in leakage energy due to sharing the leakage power across three operations outweighs such an increase and the total energy is lower than an unpipelined design.

Given a throughput target and absent any latency constraints, one could try to find the lowest power designs as has been illustrated in earlier works, however such low power designs will be very slow designs that will be at the expense of total design area. Similarly if one optimizes only for area, one will get deeply pipelined fast-tick machines that would consume higher power. For example, a 1 TeraFlop single-precision floating-point throughput can be achieved in 90nm technology by the following two ends of the design spectrum:

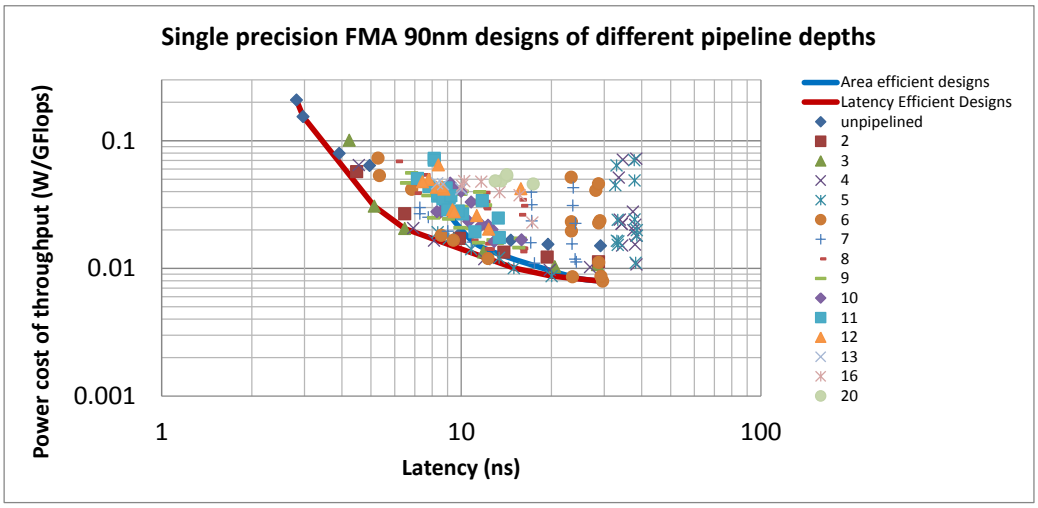
- Power efficient design: using 2500 low power 4-stage FMA units operating at 200 MHz. The units use low 1V supply voltage and high threshold voltage transistors to minimize consumed power. The total power is only 9W at the cost of 132.5 mm^2 of area.[1]

- Area efficient design: employing only 294 high performance 10-stage FMA units operating at 1.7GHz. The units use high 1.2V supply voltage and low threshold

[1]FMA operation is counted as 2 floating-point operations



(a)



(b)

Figure 3.3: Energy per operation vs. (a) Area cost of throughput (b) Latency for single-precision FMA in 90nm. Throughput optimal designs are different from latency optimal.

transistors to improve speed. The total area is only 35mm^2 at the expense of 46W of power consumption.

So we have a 3-dimensional search space of throughput, power and area. Given any two dimensions, one can optimize for the third. For example, given a certain throughput target and area budget, one can minimize total power. Alternatively, for a certain area and power budget, one can seek the maximum throughput design. Luckily the power and area of parallel designs are to a first order a linear function of throughput. Take for instance a design achieving throughput T at cost of total power P and total area A, then using two such designs we achieve 2T throughput, and the needed power and area are 2P and 2A respectively. Therefore, one can normalize the power and area to be per unit of throughput, reducing the search space to a 2-dimensional space of power efficiency ϵ_P (P/T in W/GFlops which is also the energy per operation) and area efficiency ϵ_A (A/T in mm^2/GFlops). Figure 3.3(a) plots different FMA designs with varying supply and threshold voltages, pipeline depth and target frequency in this design space. The plot shows that there exists actually an efficient frontier for tradeoff between ϵ_P and ϵ_A. The area efficient frontier is not the same as the latency minimum frontier as shown in Figure 3.3(b). This shows the need for different design methodology for throughput designs than latency designs.

So in conclusion, for applications that have abundant parallelism (e.g. visual computing, Internet routing and web search), the key performance metric is the aggregate number of operations performed by the entire machine. Whether we have 5 or 20 processors does not matter; all we care about are the overall throughput, power and area. For a given throughput, the true costs we are trying to optimize, whether it is a chip or a server room, are chip area (or floor space for a bigger machine) and power. By normalizing area and power cost to throughput, we find there is a set of optimal designs that tradeoff power for area necessary to achieve a certain throughput. However, we still need a criteria for choosing one design from this set. We can find such criteria by looking at the hard constraints of a throughput system. Some systems have hard resources constraints, while others have hard performance constraints. The tradeoff curve of Figure 3.3(a) is sufficient in choosing the optimal design for both resource and performance constrained throughput systems.

3.1 Resource Constrained Throughput Systems

In designing resource constrained throughput systems, we are trying to maximize throughput given a set of fixed power, area and thermal constraints. These are often single chip systems such as GPU or mobile devices. In these systems the goal is to:

$$\text{maximize } T$$

subject to:

$$P/A < D_{max}$$
$$P < P_{max}$$
$$A < A_{max}$$

where T is total throughput in GFlops, P is total power in W, A is total area in mm^2, D_{max} is maximum power density in W/mm^2

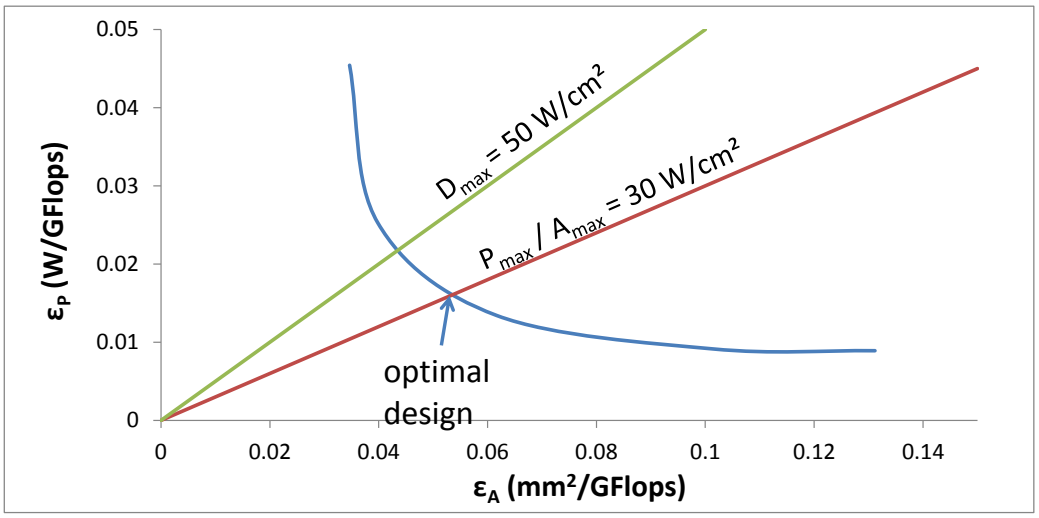
Using an ϵ_P-ϵ_A trade-off curve, we can easily find the optimal maximum throughput that conforms to area, power, and power density constraints by substituting $P=\epsilon_P T$ and $A=\epsilon_A T$. The solution to this problem is the point(ϵ_A,ϵ_P) on the efficient frontier satisfying the condition:

$$\epsilon_P/\epsilon_A=\min(P_{max}/A_{max},D_{max})$$

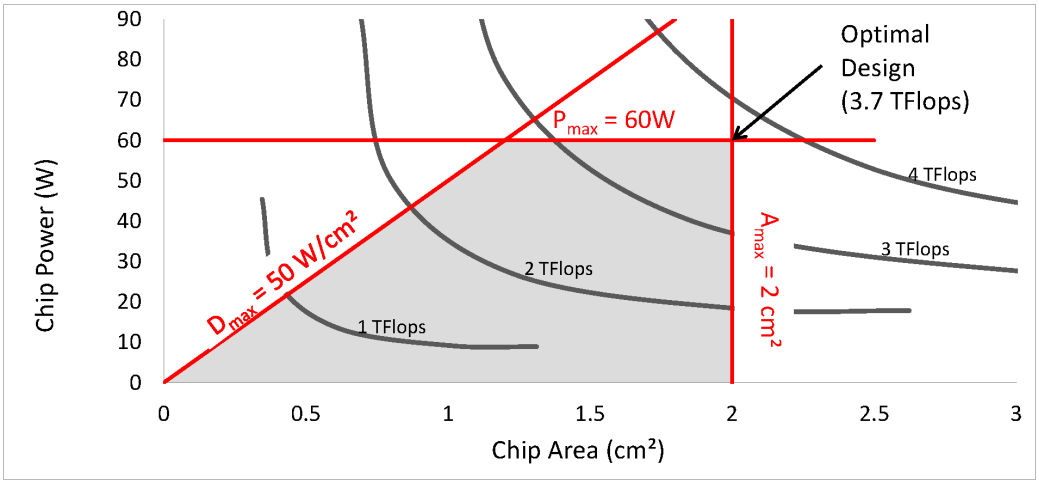
Such a design achieves the maximum throughput of:

$$T=\min(A_{max},P_{max}/D_{max})/\epsilon_A$$

Figure 3.4(a) illustrates graphically how to find the optimal design using an existing ϵ_P-ϵ_A trade-off curve for an example constraints of $A_{max} = 2$ cm^2, $P_{max} = 60$ W, and $D_{max} = 50$ W/cm^2. The intersection of the P_{max}/A_{max} constant power density line with the tradeoff curve is the optimal design since P_{max}/A_{max} in this example is a tighter constraint than D_{max}. The optimal FPU design is a 1.67 GFlops design



(a)



(b)

Figure 3.4: (a) Determining optimal design point from throughput-energy trade-off curve and constraints (b) Contour map of achievable throughputs versus area and power. Constraints of A_{max} = 2 cm^2, P_{max} = 60 W, and D_{max} = 50 W/cm^2 are indicated.

with an area of 0.09 mm^2 and power of 27 mW.[2] Integrating 2,222 such FPUs on a chip we achieve a total throughput of 3.7 TFlops at 60W and 2cm^2.

Figure 3.4(b) views the data in a slightly different way. First, we take the ϵ_P-ϵ_A trade-off and multiply the curve by several values of throughput (say 1, 2, 3, 4 TFlops) generating the required chip area and chip power required for such throughputs. Drawing these curves together in Chip Power versus Chip Area space gives us a contour map of efficient throughput designs for any value of chip power and area. Overlaying the resource constraints on the graph (the red lines), we obtain the shaded allowed design space with highest throughput design at the intersection of the area and power constraints achieving 3.7 TFlops; the same as obtained previously using Figure 3.4(a).

3.2 Performance Constrained Throughput Systems

The second class of systems has hard performance constraints, and these systems generally use many individual processing units in parallel to achieve their total throughput requirement. As such, both system energy and total "chips" area are flexible, as long as one can meet the throughput performance target. Minimizing the total cost of ownership (TCO) is the optimization goal of such a system. Several studies have looked at building a detailed cost model for datacenters that incorporates all the costs of building, running and maintaining a datacenter [28, 11]. The model separates the costs into two categories:

- Capital expenditures: including cost of real estate, buildings, datacenter machines, power delivery and cooling equipment. The compute equipment is usually amortized over 3 years while the building facilities are amortized over 15 years.

- Operational expenditures: including electricity cost for powering and cooling,

[2]The throughput, area, and the power of the building blocks can't be deduced using only the trade-off curve. It only says the optimal design has figures of merit of ϵ_A of 0.054 mm^2/GFlops and ϵ_P of 0.016 W/GFlops. The throughput information is retrieved from the stored design information for such a design point.

personnel and software expenses

The total cost function is related to the area and power efficiency metrics by the prices of chip area and electricity respectively. The optimization problem can be summarized as follows:

$$\text{minimize } \phi(\epsilon_A, \epsilon_P)$$

subject to:

$$\epsilon_P < D_{max}\epsilon_A$$

(where $\phi(\epsilon_A, \epsilon_P)$ is the throughput cost efficiency in \$/GFlops per year as a function of energy/op (ϵ_P) in W/GFlops and area efficiency (ϵ_A) in mm^2/GFlops, D_{max} is maximum power density in W/mm^2)

In minimizing the cost of throughput constrained systems, two extreme cases are easy to see:

- **System energy is free**: In this case, all we care about is a design that maximizes the throughput per chip area. This is exactly how chips were designed in the early days of scaling where area efficiency was the overriding design goal.
- **Chip area is free**: In this case, all we care about is minimizing energy consumption. We then choose the most power-efficient system, which generally leads to systems with a large number of very slow units. Numerous studies have shown that minimum energy solutions generally operate at low Vdd, which cause the units to operate in the subthreshold region and have very low performance per unit area.

In real situations, however, neither energy nor area is free so both need to be considered in the context of minimizing the total cost of ownership. Eq. (3.1) shows an example cost function that incorporates different possible cost components: a

power cost ϕ_{power} as function of energy efficiency, a hardware cost $\phi_{hardware}$ as function of area efficiency, and a cooling cost $\phi_{cooling}$ as function of power efficiency and power density.

$$\phi(\epsilon_A, \epsilon_P) = \phi_{hardware}(\epsilon_A) + \phi_{power}(\epsilon_P) + \phi_{cooling}(\epsilon_A, \epsilon_P) \tag{3.1}$$

At the optimal point, the marginal cost of incremental energy and area will match, since if they were not the same, we could lower cost by "selling" the expensive one, and "buying" the cheaper one. If the hardware and power cost are linear on area and energy, the ratio of the \$/mm^2 and \$/W sets the constant cost curves which are straight lines in the W/GFlops versus mm^2/GFlops space. If the costs are nonlinear, the constant cost curves will still exist, but will no longer be straight lines. The point where the trade-off curve is tangent to the constant cost curve will minimize the total cost of the system as illustrated in Fig. 3.5.

$$\phi(\epsilon_A, \epsilon_P) = c_A\epsilon_A + c_P\epsilon_P + constant \tag{3.2}$$

If the marginal energy and area costs are relatively constant, we can use the simplified linear cost equation (3.2) employing area cost c_A (¢/mm^2 per year) and power cost c_P(¢/W per year). A such, the slope of the constant cost lines becomes c_A/c_P. We can then convert the minimum TCO problem to finding the optimal design at a given power density by relating the slope of the trade-off curve at each point to the power density at this point. Fig. 3.6 shows the result for the trade-off curves in 90 and 45nm. As intuitively expected, higher energy prices (low c_A/c_P ratio) results in low-energy designs with low performance/mm^2 and low-power density, while high hardware cost (high c_A/c_P ratio) results in high-energy high-performance/mm^2 designs with higher power density and less energy efficiency. It is notable that the optimal power density for 45nm is twice the optimal power density of 90nm designs which indicates that energy efficiency is scaling slower than area efficiency from 90nm to 45nm. Another interesting observation is the quadratic relation between power density and slope of tradeoff curve. This quadratic relation can be used to derive an approximation model for the tradeoffs to better understand its properties by noticing

Example cost model assumptions:

- Hardware cost: assume to be proportional to the area (e.g. 100 mm^2 chip costs \$100 while 200 mm^2 chip cost \$200)
  - Area cost $= \frac{1\ \$/mm^2}{3\text{ years depreciation period}} = 33\ ¢/mm^2$ (per year)
- Electricity cost: assume an electricity price of 10 ¢/KWh
  - Power cost $= \frac{10\ ¢}{\text{KWh}} \frac{24\text{ hours}}{1\text{ day}} \frac{365\text{ days}}{1\text{ year}} \frac{1\text{ KWh}}{1000\text{ W}} = 87.6\ ¢/\text{W}$ (per year)
- Power delivery and Cooling cost is significant cost as well
  - 0.5W consumed for every 1W of operation power
  - High power density designs require more expensive cooling systems
- Throughput cost = Area cost $\frac{\text{Area}}{\text{Throughput}}$ + (Power & Cooling costs) $\frac{\text{Power}}{\text{Throughput}}$
  - $\frac{¢}{\text{GFlops.Year}} = \frac{¢}{mm^2\text{.year}} \frac{mm^2}{\text{GFlops}} + \frac{¢}{\text{W.year}} \frac{\text{W}}{\text{GFlops}}$

Final Cost equation:

$$\phi(\epsilon_A, \epsilon_P) = 33\epsilon_A + 87.6\epsilon_P(1.5 + \frac{\epsilon_P}{\epsilon_A})$$

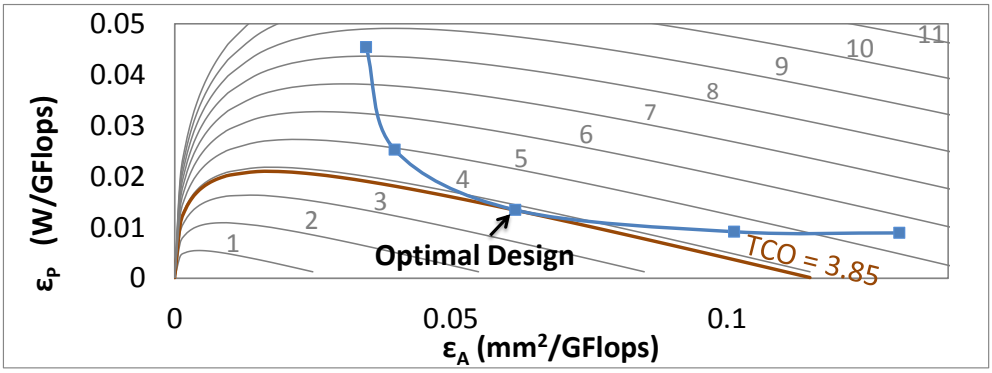


Figure 3.5: Power/throughput versus area/throughput trade-off overlaid on example constant TCO contours in ¢/GFlops per year. The minimum cost design achieves TCO of 3.85¢/GFlops per year. The cooling costs of the system are proportional to power density which accounts for the nonlinearity of the constant cost contours.

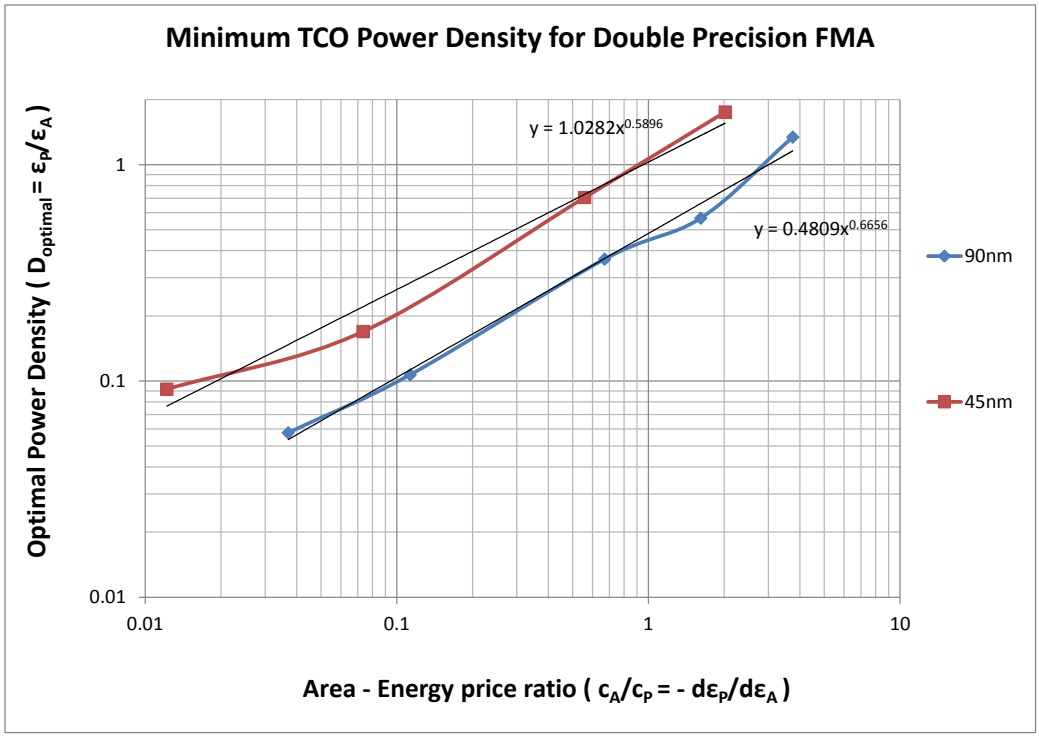


Figure 3.6: Optimal Power density of system ($\frac{\epsilon_P}{\epsilon_A}$) depends on the slope of the tradeoff curve ($\frac{d\epsilon_P}{d\epsilon_A}$) which is equal to the relative cost of area and energy in an optimal performance-constrained system. This relationship was generated for the 90nm and 45 nm tradeoffs data for the linear cost function $\phi(\epsilon_A, \epsilon_P) = c_A\epsilon_A + c_P\epsilon_P$. Note that high-power densities are only cost-effective when the area to power costs ratio is large.

that:

$$\frac{d\boldsymbol{\epsilon_P}}{d\boldsymbol{\epsilon_A}} + c\left(\frac{\boldsymbol{\epsilon_P}}{\boldsymbol{\epsilon_A}}\right)^2 = 0$$
$$\frac{d\boldsymbol{\epsilon_P}}{\boldsymbol{\epsilon_P}^2} + c\frac{d\boldsymbol{\epsilon_A}}{\boldsymbol{\epsilon_A}^2} = 0$$
$$\frac{1}{\boldsymbol{\epsilon_P}} + \frac{c}{\boldsymbol{\epsilon_A}} = d$$

So the FMA tradeoff curve can be approximated by two degrees of freedom: the minimum achievable W/GFlops (ϵ_{Pmin}) and the minimum mm^2/GFlops (ϵ_{Amin}). They determine the approximation curve and its slope as given by Eqs. (3.3) and (3.4). The knee of the curve which balances the power and area efficiency occurs at the point $(2\epsilon_{Amin}, 2\epsilon_{Pmin})$ with the characteristic power density of that point being $\epsilon_{Pmin}/\epsilon_{Amin}$

$$\boxed{\frac{\epsilon_{Pmin}}{\boldsymbol{\epsilon_P}} + \frac{\epsilon_{Amin}}{\boldsymbol{\epsilon_A}} = 1} \tag{3.3}$$

$$\frac{d\boldsymbol{\epsilon_P}}{d\boldsymbol{\epsilon_A}} = -\frac{\epsilon_{Amin}}{\epsilon_{Pmin}}\left(\frac{\boldsymbol{\epsilon_P}}{\boldsymbol{\epsilon_A}}\right)^2 = -\frac{\epsilon_{Amin}\epsilon_{Pmin}}{(\boldsymbol{\epsilon_A} - \epsilon_{Amin})^2} = -\frac{(\boldsymbol{\epsilon_P} - \epsilon_{Pmin})^2}{\epsilon_{Amin}\epsilon_{Pmin}} \tag{3.4}$$

Figure 3.7 shows that Eq. (3.3) provides a good approximation to the FMA throughput tradeoffs between mm^2/GFlops and W/GFlops. The minimum achievable mm^2/GFlops (ϵ_{Amin}) scales 10.76× from 0.114 mm^2/GFlops in 90nm to 0.0106 mm^2/GFlops in 45nm. The minimum achievable W/Gflops (ϵ_{Pmin}) however scales by only 2.37× from 17.6 mW/GFlops in 90 nm to 7.4 mW/GFlops in 45nm. This results in the characteristic power density increasing from 0.16 W/mm^2 in 90nm to 0.7 W/mm^2 in 45nm.

3.3 Sensitivity Analysis of Throughput Tradeoffs

Having generated tradeoff curves for throughput, now we look closely at how to make the different design choices to achieve the optimal designs on the efficient frontier. In

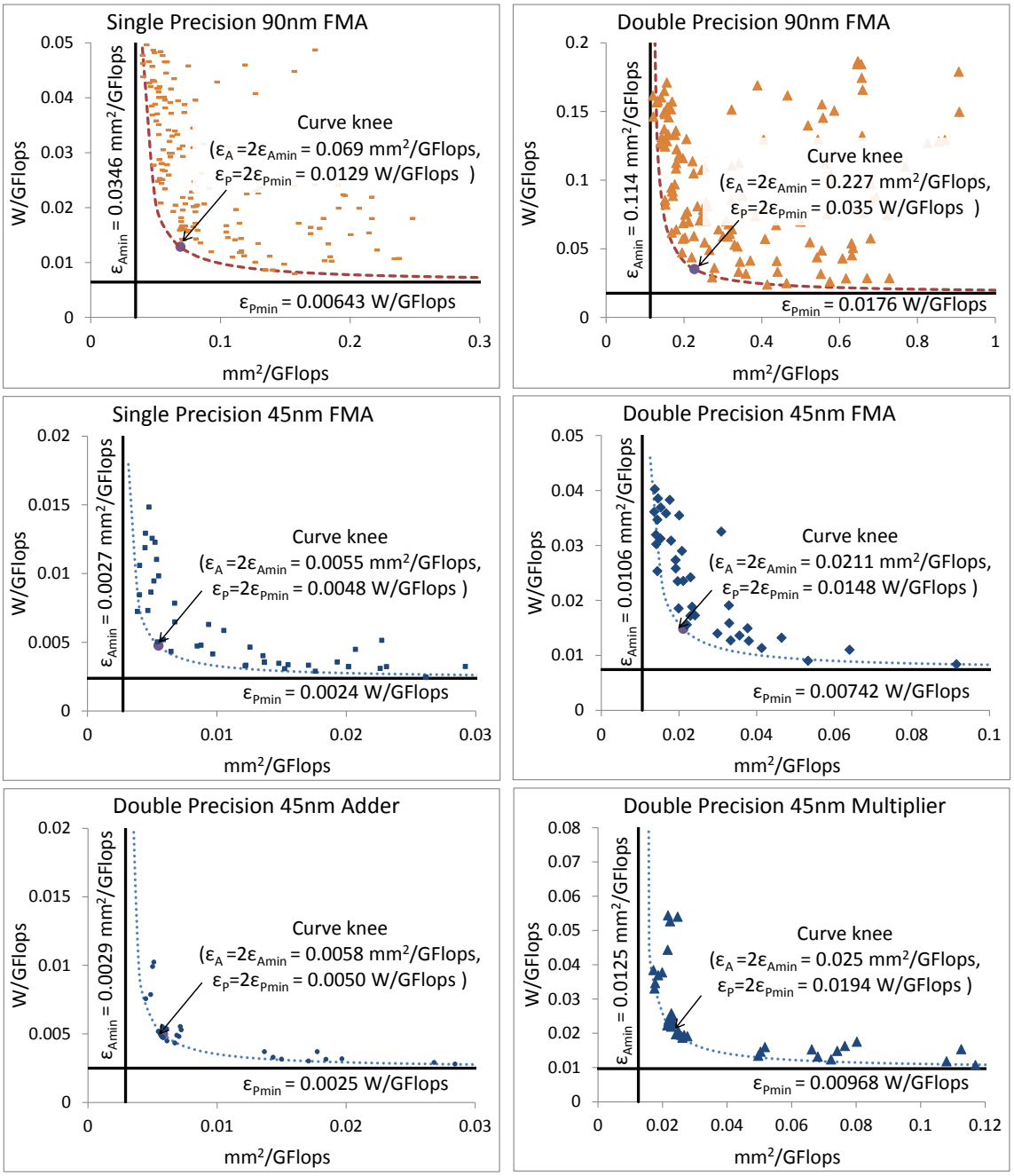


Figure 3.7: Floating point FMA, adder and multiplier designs for 90 and 45nm processes with their efficient frontiers fitted through the relationship $\frac{\epsilon_{Pmin}}{\epsilon_P} + \frac{\epsilon_{Amin}}{\epsilon_A} = 1$. The 45nm process has higher characteristic power density ($\epsilon_{Pmin}/\epsilon_{Amin}$) of 0.7W/mm^2 vs 0.16W/mm^2 for 90nm due to limited scaling of power efficiency

this section, we look at sensitivity analysis for the optimization variables of throughput tradeoffs to choose the values of optimization knobs. We derive energy marginal costs of mm^2/throughput from energy marginal costs of latency which were introduced in Section 2.2.1.

In the case of throughput, the tradeoff is between energy per operation (E) and area per throughput ($\tau A/N$) where τ is the latency delay, A is the area and N is the pipeline depth of the design. We define the sensitivity for the throughput case (U_x) of an optimization variable x as:

$$U_X = -\frac{\tau A}{NE}\frac{\frac{\partial E}{\partial x}}{\frac{\partial(\frac{\tau A}{N})}{\partial x}} \tag{3.5}$$

In the following sections we derive the sensitivities for our four optimization variables: supply voltage V_{dd}, threshold voltage V_{th}, pipeline depth and transistor sizing.

3.3.1 Supply and Threshold Voltage Sensitivities

Supply voltage and threshold voltage neither affect the area (A) nor the pipeline depth of the circuit (N). Therefore $\frac{\partial(\frac{\tau A}{N})}{\partial x}$ reduces to $\frac{A}{N}\frac{\partial \tau}{\partial x}$ and Eq. (3.5) reduces to Eq. (2.1) and their throughput sensitivities are the same as their latency sensitivities which are derived in detail in [27].

$$U_{V_{dd}} = S_{V_{dd}} \tag{3.6}$$

$$U_{V_{th}} = S_{V_{th}} \tag{3.7}$$

3.3.2 Circuit Sizing Sensitivity

For circuit sizing we only consider the sizing of the overall circuit and not of an individual component of the circuit. That is because the sensitivities of all internal circuits are equal to the sensitivity of the containing block in an optimal design. So Eq. (3.8) defines sizing sensitivity to the sizing parameter W as:

$$
\begin{aligned}
U_W &= -\frac{\tau A}{NE}\frac{\frac{\partial E}{\partial W}}{\frac{\partial(\frac{\tau A}{N})}{\partial W}} \\
&= -\frac{\tau A}{NE}\frac{\frac{\partial E}{\partial W}}{\frac{A}{N}\frac{\partial \tau}{\partial W}+\frac{\tau}{N}\frac{\partial A}{\partial W}} \\
&= -\frac{\frac{\partial E}{\partial W}/E}{\frac{\partial \tau}{\partial W}/\tau+\frac{\partial A}{\partial W}/A} \\
\boxed{U_W &= -\frac{1}{\frac{1}{S_W}-\frac{E}{A}\frac{\frac{\partial A}{\partial W}}{\frac{\partial E}{\partial W}}}} \qquad (3.8)
\end{aligned}
$$

However if we look closely to the denominator in Eq. (3.8) we find that the second term ($\frac{E}{A}\frac{\frac{\partial A}{\partial W}}{\frac{\partial E}{\partial W}}$) is around 1 from experimental data as energy and area are proportional to each other given that all other parameters unchanged. This basically means that percent increase in area almost matches percent increase in energy per operation if only circuit sizing is changed. We can use this fact to approximate throughput sensitivity U_W in terms of energy-delay sensitivity S_W to be

$$U_W \approx \frac{S_W}{1-S_W} \qquad (3.9)$$

This relationship is interesting because it says that the circuit sizing knob saturates when energy-delay sensitivity S_W reaches 1 which is the ED minimum design point. Therefore for throughput efficient designs with sensitivities greater than 1, the circuits should not be sized as aggressively as possible, since parallelism can yield better results than increased sizing of the building block, and other knobs such as supply or threshold voltage should be used further as tuning knobs. Therefore minimum delay sizing which may be efficient for high performance latency designs is never efficient for throughput efficient designs.

3.3.3 Pipeline Depth Sensitivity

Pipelining is another important knob for throughput. To understand the effects of pipelining on throughput, area and energy, we model the total area, delay and energy for a system with N-stage pipeline in Eqs. (3.10-3.14). We start with an unpipelined design with delay $\tau_{unpipelined}$, area $A_{unpipelined}$ and energy $E_{unpipelined}$. We make the simplifying assumption that we can take an average datapath width to represent the overhead, although the number of registers in every pipeline stage could be different from one stage to another. Therefore a pipeline stage can be modeled to have a delay τ_{stage}, an average area A_{stage}, and energy E_{stage}.

$$\tau = \tau_{unpipelined} + N\tau_{stage} \tag{3.10}$$

$$A = A_{unpipelined} + NA_{stage} \tag{3.11}$$

$$E_{dynamic} = E_{unpipelined\_dynamic} + NE_{stage\_dynamic} \tag{3.12}$$

$$\begin{aligned} E_{Leakage} &= P_{leakage} \times \frac{\tau}{N} \\ &= (P_{unpipelined\_leakage} + NP_{stage\_leakage})(\frac{\tau_{unpipelined}}{N} + \tau_{stage}) \end{aligned} \tag{3.13}$$

$$\begin{aligned} E &= E_{dynamic} + E_{leakage} \\ &= E_{unpipelined\_dynamic} + P_{unpipelined\_leakage}\tau_{stage} + P_{stage\_Leakage}\tau_{unpipelined} \\ &+ N(E_{stage\_dynamic} + P_{stage\_leakage}\tau_{stage}) + \frac{P_{unpipelined\_leakage}\tau_{unpipelined}}{N} \end{aligned} \tag{3.14}$$

Adding the N$^{\text{th}}$ pipeline stage to such a design leads to interplay of several effects on the delay, area and energy:

- Increases latency by stage timing overhead (τ_{stage})
- Decreases cycle time by $\frac{\tau_{unpipelined}}{N-1} - \frac{\tau_{unpipelined}}{N} = \frac{\tau_{unpipelined}}{N^2-N}$
- Increases area by pipelining area overhead (A_{stage})
- Increases dynamic energy by the switching and clocking energy needed for the

extra stage ($E_{stage\_dynamic}$)

- Decrease leakage energy consumed per operation on combinational logic because of decreased cycle time by $\frac{P_{unpipelined\_leakage}\tau_{unpipelined}}{N^2-N}$
- Increases leakage energy by $\frac{P_{stage\_leakage}\tau}{N}$

So it is clear that pipelining improves throughput ($\frac{N}{\tau}$), and degrades area (A), however it is not clear how it affects the energy because it has both positive and negative effect on the energy per operation (E). Investigating this further we will find out that pipelining indeed improves energy efficiency for low values of N and then saturates as an optimization knob for throughput when the throughput improvement of pipelining is matched by the area overhead of pipelining.

Having modeled the system delay (τ) in Eq. (3.10), area (A) in Eq. (3.11) and energy per operation (E) in Eq. (3.14), we derive the marginal costs and the throughput sensitivity (U_N) as a function of pipeline depth (N) in Eqs. (3.15-3.17). These equations are plotted for empirical 90nm single precision FMA data in Figure 3.8.

$$\frac{\partial(\frac{\tau A}{N})}{\partial N} = \tau_{stage}A_{stage} - \frac{\tau_{unpipelined}A_{unpipelined}}{N^2} \tag{3.15}$$

$$\frac{\partial E}{\partial N} = E_{stage\_Dyn} + P_{stage\_leakage}\tau_{stage} - \frac{P_{unpipelined\_leakage}\tau_{unpipelined}}{N^2} \tag{3.16}$$

$$U_N = -\frac{\tau A}{NE}\frac{\frac{\partial E}{\partial N}}{\frac{\partial(\frac{\tau A}{N})}{\partial N}} \tag{3.17}$$

The minimum number of pipeline stages for throughput efficient designs is 5 stages in Figure 3.8. As unpipelined designs suffer from big leakage energy component, adding a pipeline stage adds a small energy cost but achieves bigger savings in leakage energy. The minimum pipeline depth N_{min} occurs when the marginal energy cost of adding a pipeline stage is zero, i.e. when $\frac{\partial E}{\partial N} = 0$ (savings in leakage energy matches pipelining energy overhead) and is given by Eq. (3.18). Similarly Eq. (3.19) gives an upper bound N_{max} on the utility of pipelining when the increased throughput

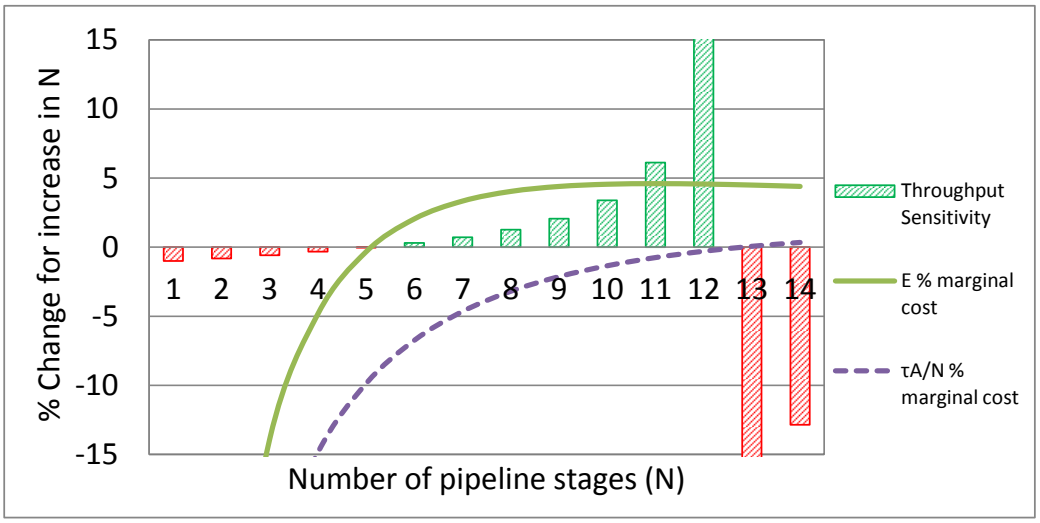


Figure 3.8: Pipelining sensitivity and marginal costs for single-precision 90nm FMA designs: green bars indicate useful area of tradeoff

achieved through pipelining is matched by the area overhead, i.e. when $\frac{\partial(\frac{\tau A}{N})}{\partial N} = 0$. Table 3.1 illustrates the useful tradeoff interval ($N_{min}<N<N_{max}$) where the throughput sensitivity of pipelining U_N is positive and increased computational density comes at the cost of energy efficiency.

$$N_{min} = N(where \frac{\partial E}{\partial N} = 0) = \sqrt{\frac{P_{unpipelined\_leakage}\tau_{unpipelined}}{E_{stage\_Dyn} + P_{stage\_leakage}\tau_{stage}}}\Bigg|_{U=0} \tag{3.18}$$

$$N_{max} = N(where \frac{\partial(\frac{\tau A}{N})}{\partial N} = 0) = \sqrt{\frac{\tau_{unpipelined}}{\tau_{stage}}\frac{A_{unpipelined}}{A_{stage}}}\Bigg|_{U=\infty} \tag{3.19}$$

Optimal Logic Density

It is more useful to look at pipelining in terms of logic depth in fan-out-of-fours (FO4) between pipeline stages rather than the number of pipeline stages N , since the number of pipeline stages depend on the function while the logic depth can give an insight if another function other than our canonical FMA example is used. To do this we model the design using the model parameters of the basic gate of the technology

	Area / throughput marginal cost $(\frac{\partial(\frac{\tau A}{N})}{\partial N})$	Energy / op marginal cost $(\frac{\partial E}{\partial N})$	Pipelining Depth Sensitivity (U_N)	Notes
$N<N_{min}$	-	-	-	Pipelining reduces both energy per operation and area per throughput so it is never energy efficient to have an unpipelined design
$N_{min}<N<N_{max}$	-	+	+	The useful tradeoff area of pipelining where increased pipelining reduces area/throughput at the cost of increased energy per operation
$N>N_{max}$	+	+	-	Additional pipelining hurts performance as the improvement in throughput is less than the increase in the area thereby increasing area/throughput.

Table 3.1: Pipeline Depth Sensitivity Intervals. Green shading denotes improvement through reduction in energy or area per throughput, while red color denotes increase in energy or area per throughput.

assuming a design where every pipeline stage uses W gates in parallel cascaded H times and pipelined using W registers. Eq. (3.20) summarizes the model parameters.

$$W : \text{Logic width} \tag{3.20a}$$

$$H : \text{Logic depth} \tag{3.20b}$$

$$N : \text{Pipeline depth} \tag{3.20c}$$

$$\tau_{unpipelined} = NH\tau_{gate} \tag{3.20d}$$

$$\tau_{stage} = \tau_{register} \tag{3.20e}$$

$$T_{clk} = \frac{\tau}{N} = H\tau_{gate} + \tau_{register} \tag{3.20f}$$

$$A_{unpipelined} = NWHA_{gate} \tag{3.20g}$$

$$A_{stage} = WA_{register} \tag{3.20h}$$

$$E_{unpipelined\_dynamic} = NWHE_{gate\_dynamic} \tag{3.20i}$$

$$E_{stage\_dynamic} = WE_{register\_dynamic} \tag{3.20j}$$

$$P_{unpipelined\_leakage} = NWHP_{gate\_leakage} \tag{3.20k}$$

$$P_{stage\_leakage} = WP_{register\_leakage} \tag{3.20l}$$

Using the model parameters and Eqs. (3.18), (3.19), we can estimate the maximum clock period for least energy designs (T_{max}) and the minimum clock period for maximum computational density designs (T_{min}) as done in Eqs. (3.21), (3.22) respectively. Tables 3.2, 3.3 calculates the maximum and minimum clock periods for 90nm and 45nm xor gates and standard flip flops library parameters. The maximum clock period for 90nm is 98 FO4 while for 45nm, it is 170 FO4, which closely matches the empirical tradeoff data from table 4.2 of 111 FO4 for 90nm and 146 FO4 for 45nm.

	90nm (V_{dd}=0.9, V_{th}=high)	45nm (V_{dd}=0.72, V_{th}=high)
τ_{gate} (ns)	0.18	0.17
$\tau_{register}$ (ns)	0.43	0.29
$E_{register\_dynamic}$ (fJ)	18	3
$P_{gate\_leakage}$ (nW)	207	17
$P_{register\_leakage}$ (nW)	267	21
FO4(ps)	44.5	34.2
T_{max} (FO4)	**98**	**170**

Table 3.2: Maximum Logic Density for 90nm and 45nm Technologies using low supply voltage and high threshold voltages based on Eq. (3.21)

	90nm (V_{dd}=1.08, V_{th}=low)	45nm (V_{dd}=0.9, V_{th}=low)
τ_{gate} (ns)	0.095	0.051
$\tau_{register}$ (ns)	0.2	0.086
A_{gate} (μm^2)	8.8	1.8
$A_{register}$ (μm^2)	16.5	3.5
FO4(ps)	25.3	13.5
T_{min} (FO4)	**15.4**	**13.4**

Table 3.3: Minimum Logic Density for 90nm and 45nm Technologies using high supply voltage and low threshold voltages based on Eq. (3.22)

$$T_{max} = \frac{\tau}{N_{min}} = \tau_{register} + \sqrt{\frac{\tau_{gate}}{P_{gate\_leakage}}(E_{register\_dynamic} + P_{register\_leakage}\tau_{register})}\Bigg|_{U=0} \tag{3.21}$$

$$T_{min} = \frac{\tau}{N_{max}} = \tau_{register} + \sqrt{\tau_{gate}\tau_{register}\frac{A_{register}}{A_{gate}}}\Bigg|_{U=\infty} \tag{3.22}$$

3.3.4 Optimization Parameters: Putting it All toghether

Having derived their sensitivities and the expected behavior, Figure 3.9 summarizes the trends of the optimal values of the optimization knobs discussed in this section (V_{dd},V_{th}, N, sizing) for progressing power density requirements for double precision FMA in 90nm. As expected, the range of useful pipelining is between 3 and 12 and

the FO4 per stage is between 30 and and 140.

3.4 Summary

For throughput applications with abundant parallelism, pipelining and parallelism can be used to achieve high-throughput energy-efficient designs. Without hard latency constraints, energy-efficient designs exhibit a high area cost with a marginal improvement in energy efficiency. As such, the trade-off between energy/op, measured in W/GFlops and computational density measured in GFlops/mm^2 is the correct trade-off for throughput designs. It allows balancing out power efficiency (W/GFlops) and area efficiency (mm^2/GFlops) to achieve the optimal design that minimizes the total cost of operation (TCO). Sensitivity analysis of different design parameters shows that throughput optimal designs are distinct from latency optimal ones and that only performance gains that can't be achieved more cheaply using parallelism are useful for throughput optimal designs. For example aggressive circuit sizing for a latency minimum design is not optimal for throughput where parallel units of moderately sized circuits can provide more throughput for the same power and area budgets.

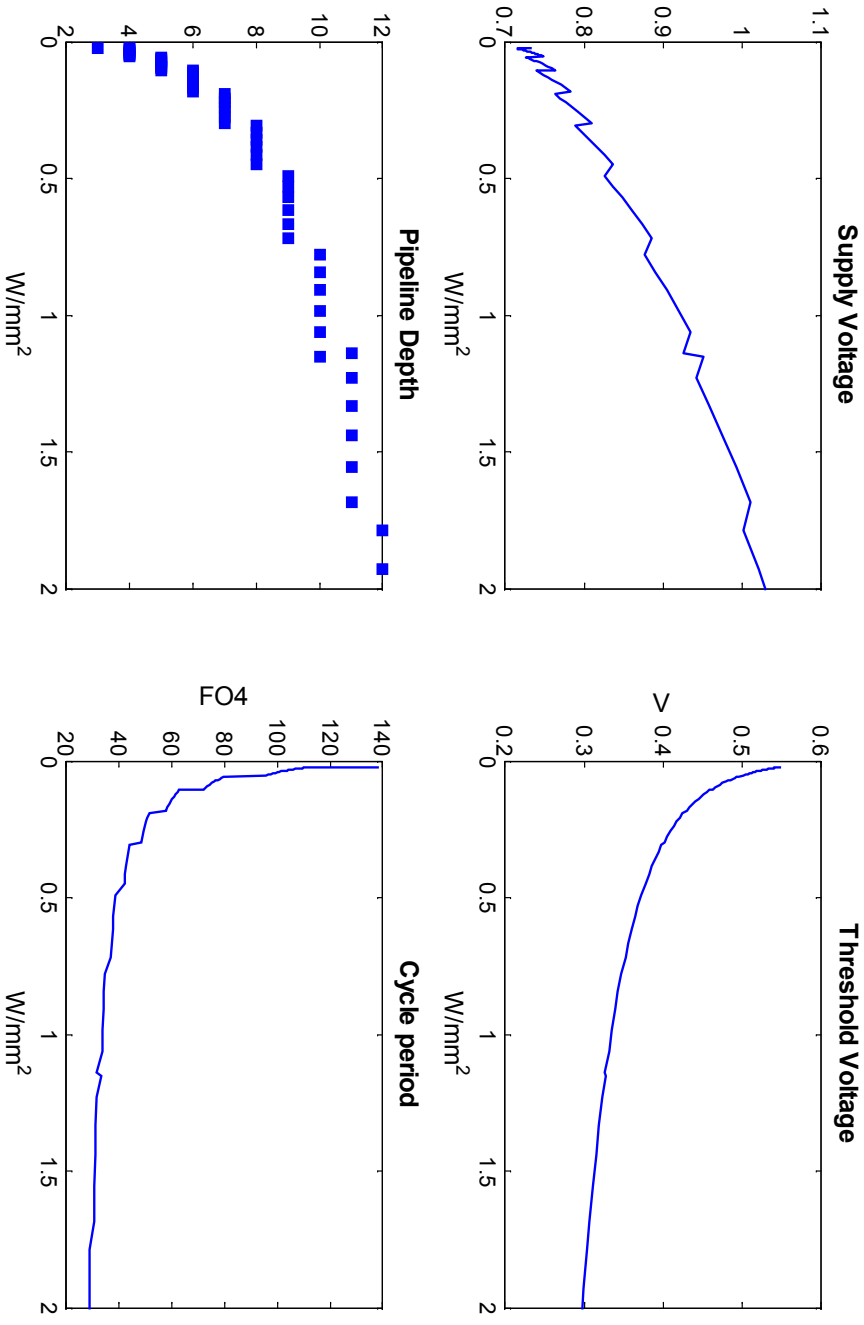


Figure 3.9: Optimal Design knobs at different power densities for 90nm Technology

Chapter 4

FPU Design Evaluation for Throughput

4.1 Floating Point Background

Floating point (FP) numbers are a computer approximation of real numbers. They are similar in concept to normalized scientific notation which takes the form of $a \times 10^b$ where the exponent b is an integer, and the mantissa coefficient a is a real number between 1 and 10. For example, the number 4335.34 is represented by 4.33534×10^3 in normalized scientific notation. In contrast, binary floating point numbers represent $(-1)^s 2^e (b_0.b_1 b_2 ... b_{p-1})$ where s is the sign bit, $b_i \in \{0, 1\}$ and e is any integer where $emin \leq e \leq emax$. Therefore floating point numbers represent a subset of real numbers characterized by precision and exponent range. Floating point arithmetic was incorporated in the earliest computers [16, 8]. While initially each manufacturer had their own standard for floating point number representation, in the 1980s the IEEE standardized the floating-point format and operations in the IEEE 754 standard [5]. This standard included a number of different rounding modes to enable one to bound round-off errors, and also defined denormal numbers (denorms), representations for numbers that are smaller in magnitude than what would otherwise be the smallest valid FP number (2^{emin}). For normalized numbers ($e > emin$), $b_0 = 1$ and therefore only the fractional part f of the remaining bits ($b_1 b_2 ... b_{p-1}$) is stored

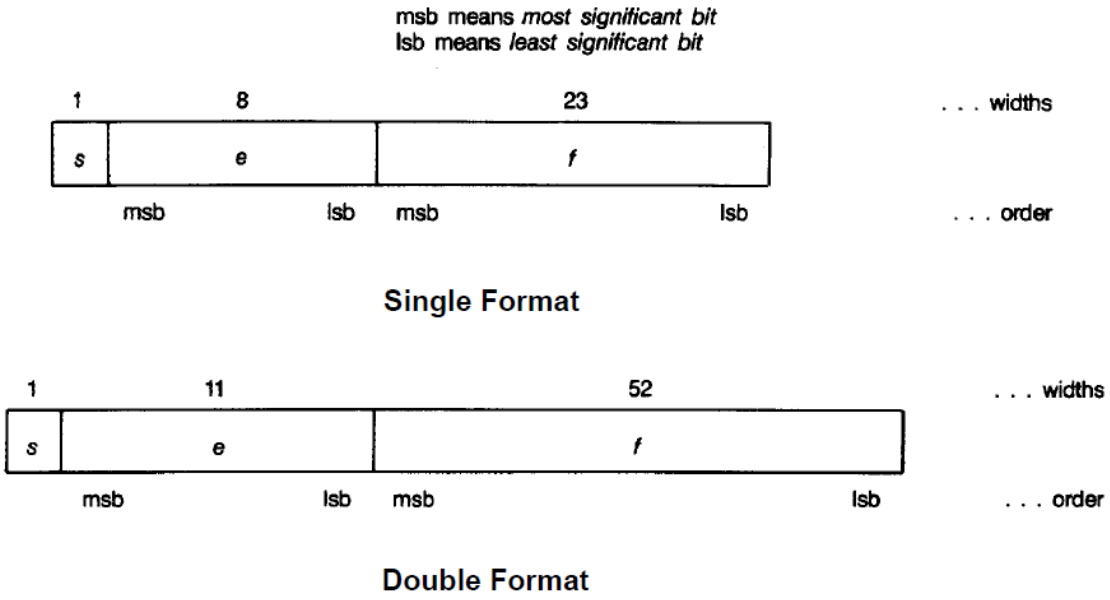


Figure 4.1: Single and double precision format according to IEEE754

to represent the number $(-1)^s 2^e (1.f)$. However for denormal numbers ($e = emin$), $b_0 = 0$ and the resulting number is $(-1)^s 2^{emin+1} (0.f)$. The standard also defines 32-bit single precision and 64-bit double precision formats as shown in Figure 4.1.

Floating point arithmetic is crucial to computer performance. It forms the backbone of scientific computation. Additionally it is used extensively for graphics and media applications. Graphics Processing Units (GPUs) exploit parallelism inherent in graphics applications and have thousands of floating point units processing parallel streams of data. For such applications, latency is not the most critical parameter. Optimizing floating point throughput per mm^2 for such designs maximizes performance. Traditionally, FPU designs have used separate floating point adders and multipliers. However, recent designs incorporate combined floating-point multiply-add instructions that implement the (A $\times$ C) + B operation; these units offer better accuracy and improved performance. We present the two most common multiply-add implementations, which we use to explore the energy/performance space of these units. The fused multiply-add (FMA) design performs operand alignment in parallel

with the multiplication, which leads to the shortest overall latency, but to accomplish this parallelism, it requires a very large variable shifter and large intermediate result datapath width. The cascade multiply-add (CMA), on the other hand, performs the multiply first, and then aligns the operands for the FP adder. While the overall latency of this structure is longer, it requires a less wide datapath, so it might be better for throughput applications. While these architectures are by no means exhaustive of all the possible multiply-add architectures, these were the "best" architectures we tested when energy becomes a first-order issue. The reason is that they don't incorporate any speculative hardware for improving latency, and no energy is wasted on pre-computed results that get discarded. In addition to these designs, we implemented many other designs that claimed some performance advantage. All of these were much worse when area and energy were considered. We also compared designs that conformed to the IEEE standard, supporting all rounding modes and denorms versus those without this support.

4.2 Fused Multiply Add

Since its introduction in IBM's RS/6000 FPU in 1990 [20], the fused Multiply add (FMA) unit has become a common implementation in recent FP multiply-add designs [38] [21]. This operation has been recently added to the IEEE floating-point arithmetic standard, IEEE741-2008. The standard defines fusedMultiplyAdd(A, C, B) as the operation that computes (A $\times$ C) + B initially with unbounded range and precision, rounding only once to the destination format. As a result, fused multiply-add has lower latency and higher precision than a multiplication followed by an addition. This design has the shortest latency compared to any other design, with aggressive designs such as the Cell Processor achieving a single precision latency of around 60 FO4. Since this base design offers the shortest latency, many innovations have been proposed to shorten its latency further, however, they have large area and power overheads that would not be appropriate when trying to optimize FLOPs/mm^2 or FLOPs/W. This design achieves its short latency by aligning the addend significand (S_A) in parallel with multiplication of S_B and S_C. This removes the conventional

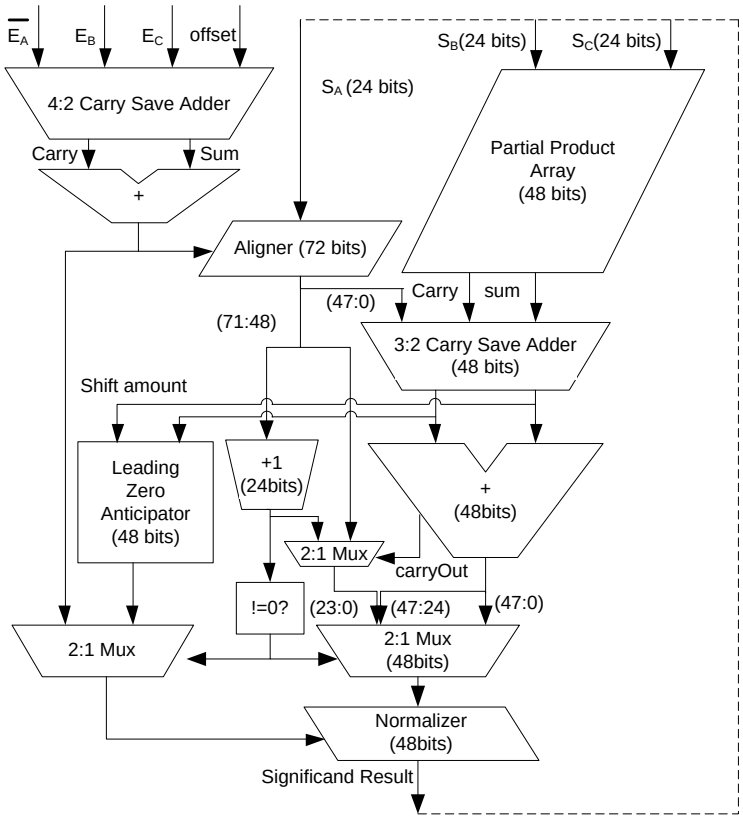


Figure 4.2: Block diagram for a single precision fused multiply-add unit. A is shifted to align it to the result of the multiply.

alignment step from the critical path of the FMA. However, since the exponent of the addend might be smaller or larger than the sum of multiplicands exponent, the addend significand can be shifted from all the way to the left of the multiplier result to all the way to the right, resulting in a wide 72 bit shifting operation in the case of single precision operation. Therefore, the datapath width for the adder and normalize stages are around 72 bits for single precision. Figure 4.2 shows the dataflow of traditional FMA, with the dashed lines showing the forwarding paths. For more detailed information on FMA design and implementation issues, please see the article by Schwarz [35].

4.3 Cascade Multiply Add

Some recent designs still prefer a cascaded design of an FP multiplier followed by an FP adder over the FMA design, especially in embedded graphics application [9][10]. In a cascade design, the partial products coming from the multipliers are combined using an adder before being fed to the aligner. The aligner then swaps its two inputs based on which significand has a smaller exponent and then shifts it to align the numbers. Finally, the aligned results are added and normalized. The datapath width for single precision CMA is around 48 bits for the aligner, adder and normalizer. Figure 4.3 illustrates the datapath of a CMA design, with the dashed lines showing the forwarding path for a dependent accumulate operation, which is shorter than the forwarding path for an operation that is using the multiplier (the dotted lines). The latency of the forwarding path for dependent accumulation is, in fact, even smaller than in the FMA design. For certain operations such as dot products, the total latency of the operation might be shorter in a CMA design than a FMA design.

4.4 Optimization Flow

Since both metrics we are studying, energy/op and ops/s/mm^2, are dependent on circuit and architecture parameters, we consider both issues by constructing different circuit level designs for the datapath portion, and use a memory simulator for estimating the register file energy and area costs. For datapath optimization, we start by synthesizing a design using standard cell libraries. The standard flow minimizes power and area for a certain delay target. The results of such latency optimized designs are not usually throughput optimal as well. This difference requires us to iterate over a wide range of frequencies, pipeline depths, and supply and threshold voltages to measure many different solutions. We can guide our exploration by understanding how each of our basic knobs affects the area, power and throughput of the design.

- **Supply and threshold voltages**: these knobs tradeoff throughput against energy/op without affecting area which leads to a straightforward tradeoff between energy/op and area/throughput.

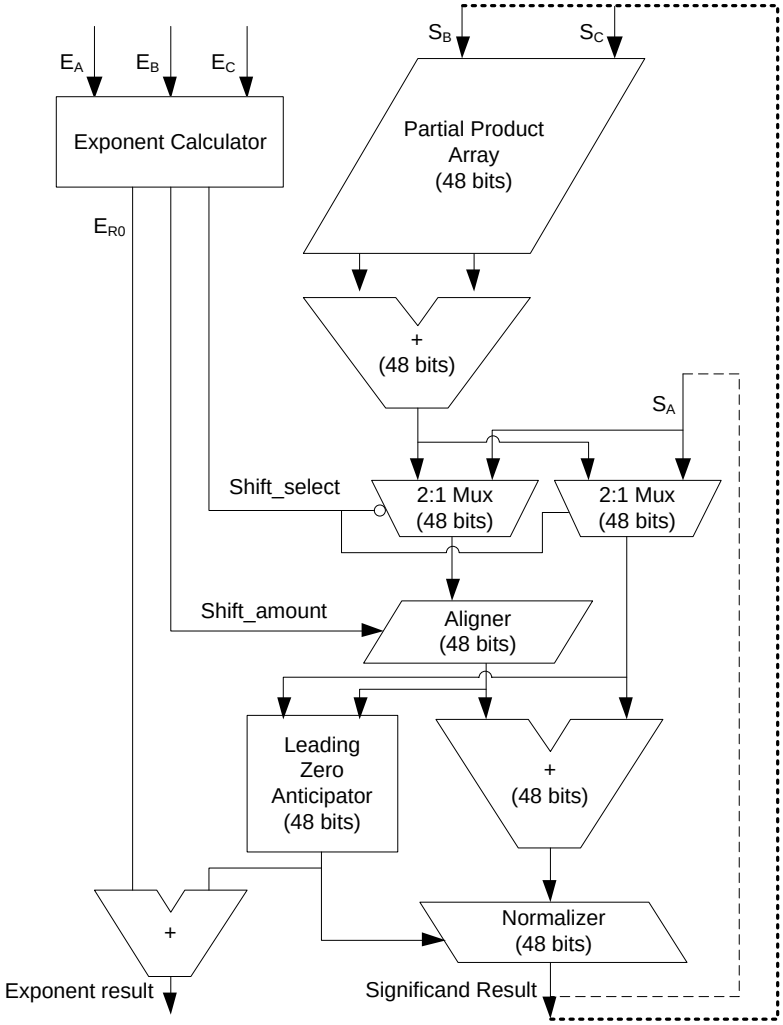


Figure 4.3: Block diagram for a single precision cascade multiply-add unit. In this design the multiply is performed first, and then the smaller of the accumulator or the product is shifted and added to generate the result.

- **Pipelining**: adding a pipeline stage (without circuits resizing) leads to an interplay of several effects on performance, energy and area, as was explained in Section 3.3.3

- **Circuit sizing**: the sizing of the circuits is controlled indirectly by setting the frequency and the pipeline depth of the design. Increasing sizing increases all throughput, area and energy/op of the design. The ability to change the energy/throughput trade-off by circuit sizing is usually smaller than in the latency optimized designs, since the relative increase in throughput due to aggressive sizing is partially offset by the increase in area the larger transistors require. This area increase reduces the improvement in ops/s/mm^2.

The datapath optimization flow starts by synthesizing a design for a certain timing constraint, inserting pipeline registers and doing register retiming to pipeline the design. Then the resulting design is placed and routed and the required clock network is generated. After the design is routed, the design is reoptimized and parasitics are extracted and annotated to the netlist. Activity factors for dynamic power calculations are derived using random input vectors and assuming full utilization of the FPU. The timing and power of the design are then reported using the Primetime timing tool. This procedure is repeated over a wide range of supply voltages, threshold voltages, clock periods and pipeline depths. After generating the data, the points on the efficient frontier are extracted from data points to generate tradeoffs as shown in Figure 3.3.

As intuitively expected, deeply-pipelined high-voltage high-frequency designs maximize computational density (ops/s/mm^2), while shallow-pipelined low-frequency low-voltage designs maximize energy efficiency (ops/s/W). Designs that mixed these traits, for example high Vdd and shallow pipelines, were never efficient choices, since we could decrease the voltage and increase the pipelining to maintain the same performance, while reducing the energy. We have used this flow with 90 nm standard cell libraries operating at Vdd values of between 1-1.2V and 45 nm libraries with 0.8-1V operating points. We have experimented with a larger voltage range as well as shown in Figure 4.4, but found that it is only helpful for extreme power densities

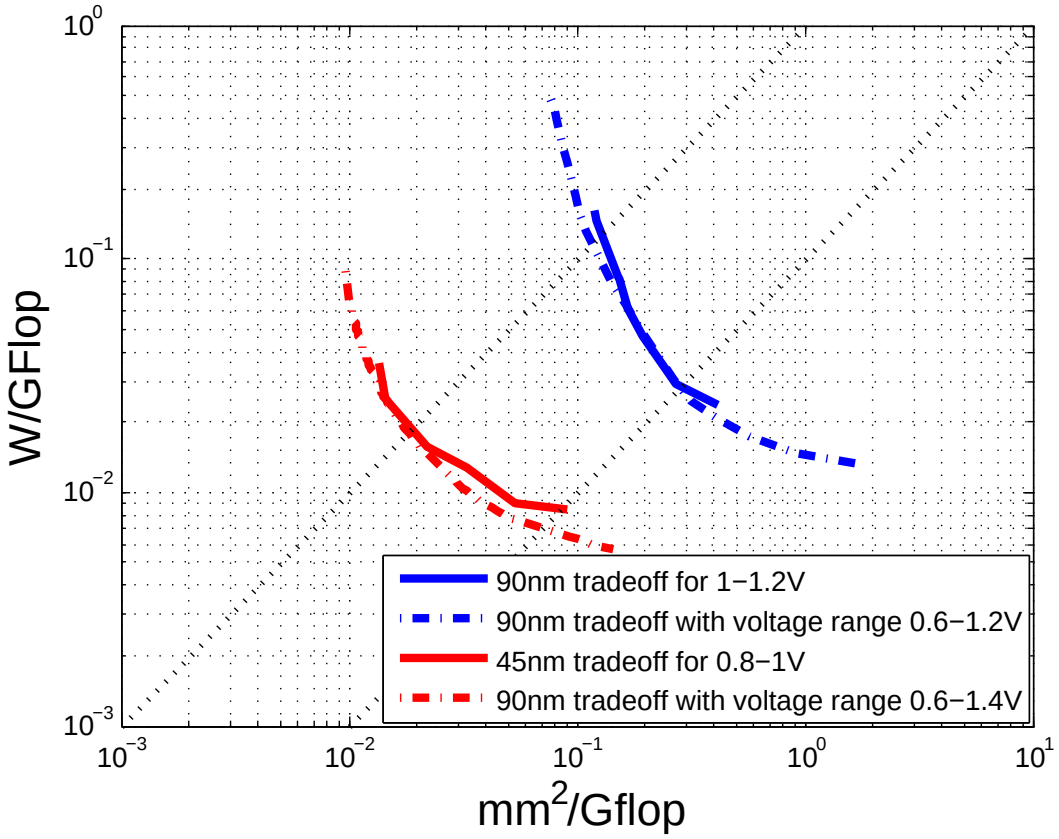


Figure 4.4: For double precision FMA tradeoffs, the voltage range of 1-1.2V in 90nm and 0.8-1V in 45nm is the optimal range for power densities between 0.1W/mm^2 and 1W/mm^2 (the two diagonally dotted lines)

that are not practical for most applications; therefore, we think that these voltage ranges satisfy most of the desired power density ranges.

4.5 Exploring Multiply-Add Architectures

We started exploring the multiply-add unit by trying to determine the most efficient architecture among cascade and fused designs. Initially, we started with designs that left out IEEE denormals and supported only truncation rounding as done in many designs used for multimedia processing. Building these two designs, it turned out that both have very similar power area tradeoffs as shown in Figure 4.5. While both designs achieve the same performance metrics in terms of W/GFlops and mm^2/GFlops, the cascade design has longer latency. For example, 3.2 GFlops throughput can be achieved by both designs at 0.036 mm^2/GFlops and 0.046 W/GFlops, but the cascade design will have a latency of 12 cycles while the fused design will take only 10 cycles.

IEEE compliance requires support for more rounding modes and input and output of denormal numbers. For support of different rounding modes an extra incrementer is added at the output of the normalizer resulting in 20% degradation in latency but only 5% degradation in energy. For supporting denormal numbers, the unit has to be modified to accept denormal number inputs and produce correct denormal results when needed. Traditionally denormal number calculations have been implemented using software traps but recent research has shown the feasibility of hardware implementations [34]. For supporting denormal inputs, the exponent difference has to be slightly modified to calculate the shift amount for the aligner correctly. The exponent field of a denormal number is 0 while the implied biased exponent of that number is 1, similar to the smallest normal number. Therefore the exponent difference needs to be modified to be incremented by +/-1 depending on which operands are denormal. Such calculation can be easily done by using carry-in signals in the exponent difference adders and therefore does not need additional energy. Supporting denormal outputs is a little bit more involved. It requires modifying the leading zero anticipator (LZA) responsible for determining the shift amount for normalization. For results that become denormals the normalization should not shift beyond what would be

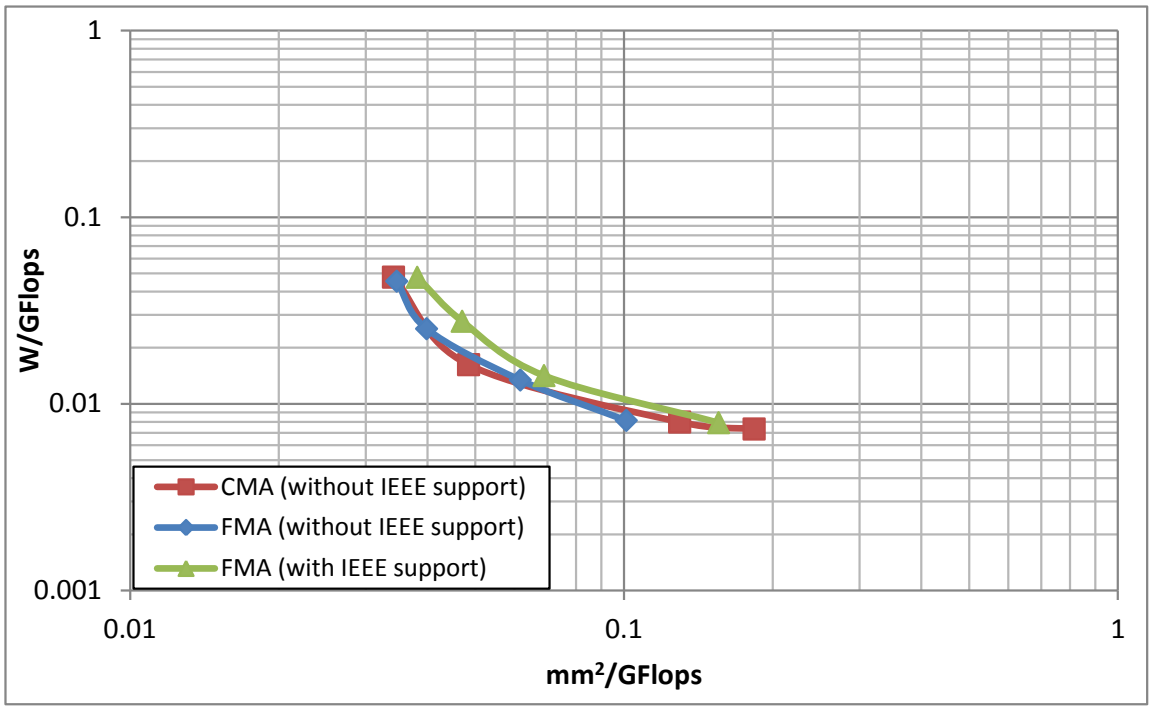


Figure 4.5: Comparison between two different FP single precision multiply-add designs, fused and cascade multiply-add. As the graph shows there is little difference between the two different designs. Also shown is an IEEE compliant unit as well. The overhead for IEEE compliance is small.

lower than the lowest exponent E_{min}. To achieve that without affecting latency an encoder produces a string of zeros whose count is equal to the maximum shift amount followed by ones. This encoded signal is "OR"ed with the original LZA string that gets fed to the leading zero detector, adding only one gate delay to the latency. Figure 4.5 shows the total cost of IEEE compliance for throughput. An IEEE compliant implementation has an overhead of 5-10% over the range of different power densities in terms of throughput performance.

Examining the effect of precision on performance, we found that double precision required approximately 3X more resources than single precision as illustrated in Figure 4.6; the area and power of the multiplier trees grow quadratically with the size of the operands (a 4X increase) while the rest of the datapath grows linearly (a 2X increase). This results in the multiplier share of area and power growing from 31% in single precision design to 45% in the double precision design.

4.6 The Energy Cost of the Fused Operation

The fused multiply-add operation requires unlimited range and precision for intermediate results between multiplication and addition. However, this increased precision increases the energy because the addition has almost 3x wider datapath than normal addition operation. Therefore a sequence of separate multiplication and addition consumes around 1.5-2X less energy than fused multiply-add as shown in Figure 4.7.

4.7 Storage Overhead

To consider register file overhead effects on FMA performance, we start by assuming that there are enough parallel threads that can be interleaved for execution to achieve full throughput, and that the interleave factor is equal to the pipeline depth so each thread does not see any data dependencies. For example, a 6 stage datapath has to interleave at least 6 threads to keep the FPU busy all the time. Therefore the minimum register file size is proportional to pipeline depth, a size sufficient for applications that

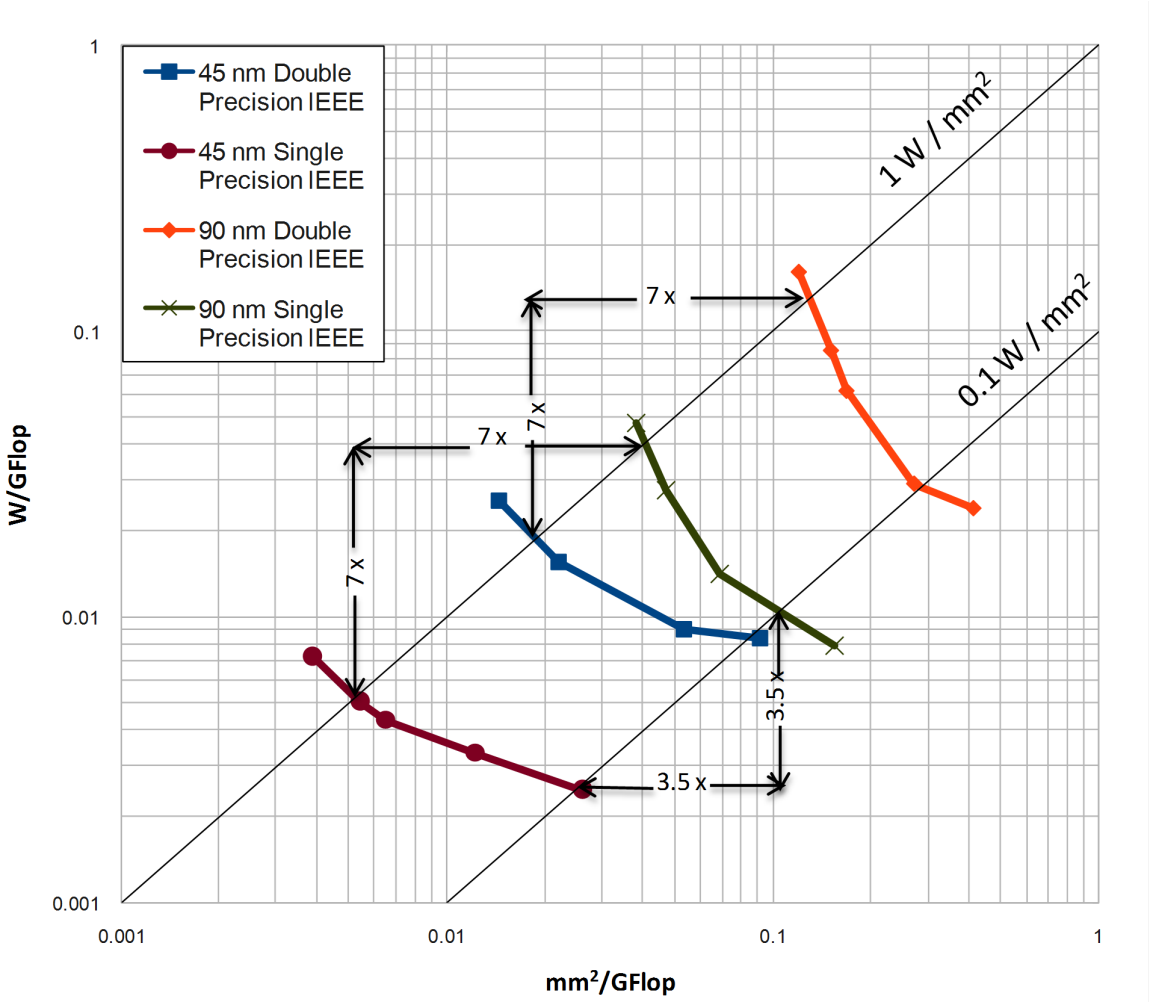


Figure 4.6: Scaling of FMA single and double precision designs from 90nm to 45nm. The performance gain depends on the power density allowed.

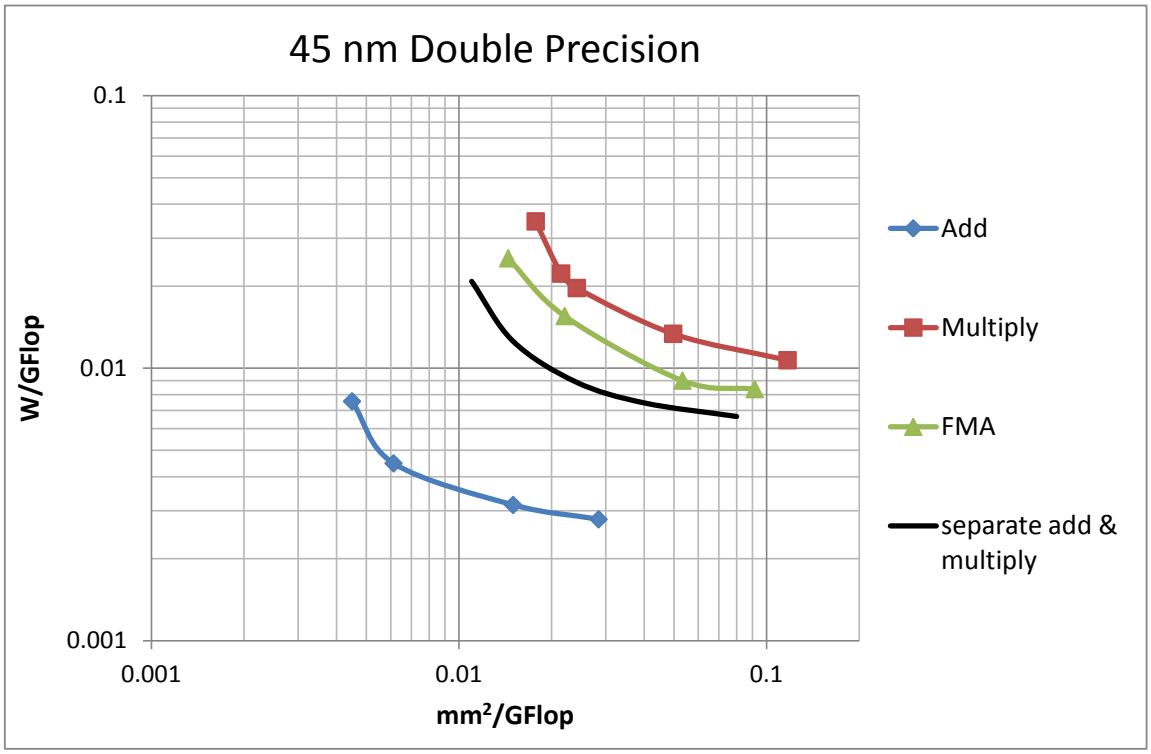


Figure 4.7: Throughput Tradeoffs for Separate Multiply and Add Units and FMA. Doing multiplication and addition in separate units exhibit around 50% less energy per operation than FMA due to lower intermediate precision. Note that FMA operation is counted as 2 Flops while multiplication and addition are counted as one.

have large arithmetic intensity - a large number of floating point operations per memory load or store. Applications with higher memory demands require larger register files to hide the latency of the memory fetch, and we explore these situations as well. The large number of required threads to hide the datapath latency makes the size of the required register files much larger than traditional CPU latency-optimized designs. This can make a straightforward 3-read 1-write register file (required for FMA designs) unwieldy both in terms of energy and area. Fortunately, since every thread accesses only its own subset of the register file, the multiported register file is usually implemented as a multibanked memory made of single ported or 1-read 1-write banks connected to the read and write ports through a crossbar [23] [25] [41]. Many memory parameters such as pipelining, hierarchical bitlines and the number of banks are part of the optimization setup. For modeling a multibanked memory system, we use the HP labs developed CACTI, a cache and memory model for estimating timing, power and area [37]. Using memory designs generated by CACTI, we augment our datapath data to generate tradeoffs that include register file accesses as well.

The FMA unit requires a multiported register file that holds enough register state for at least the number of threads equal to the datapath latency. The number of ports of the register file is equal to the product of the number of datapath ports and the ratio of the register file cycle time to the datapath cycle time. Unhooking the two clocks allows the register file to trade parallelism versus pipelining of register file access to achieve the least energy solution. In building our multibank register files, we constrain the number of banks to be at least equal to the number of ports of the register file. Additionally we explore the possibility of SIMD (single instruction multiple data) execution, which allows the use of wider words in the construction of the register file, which can result in more compact register files. Using all these design parameters we generate design space of all possible combinations using the CACTI modeling tool. Once the required number of threads and thread storage is determined from application characterization, we search the register and datapath design space to find the most optimal designs in terms of energy/op and ops/s/mm^2.

Regardless of the application characteristics however, the minimum storage needed for utilization of the FPU is dictated by the latency of the datapath itself. Assuming

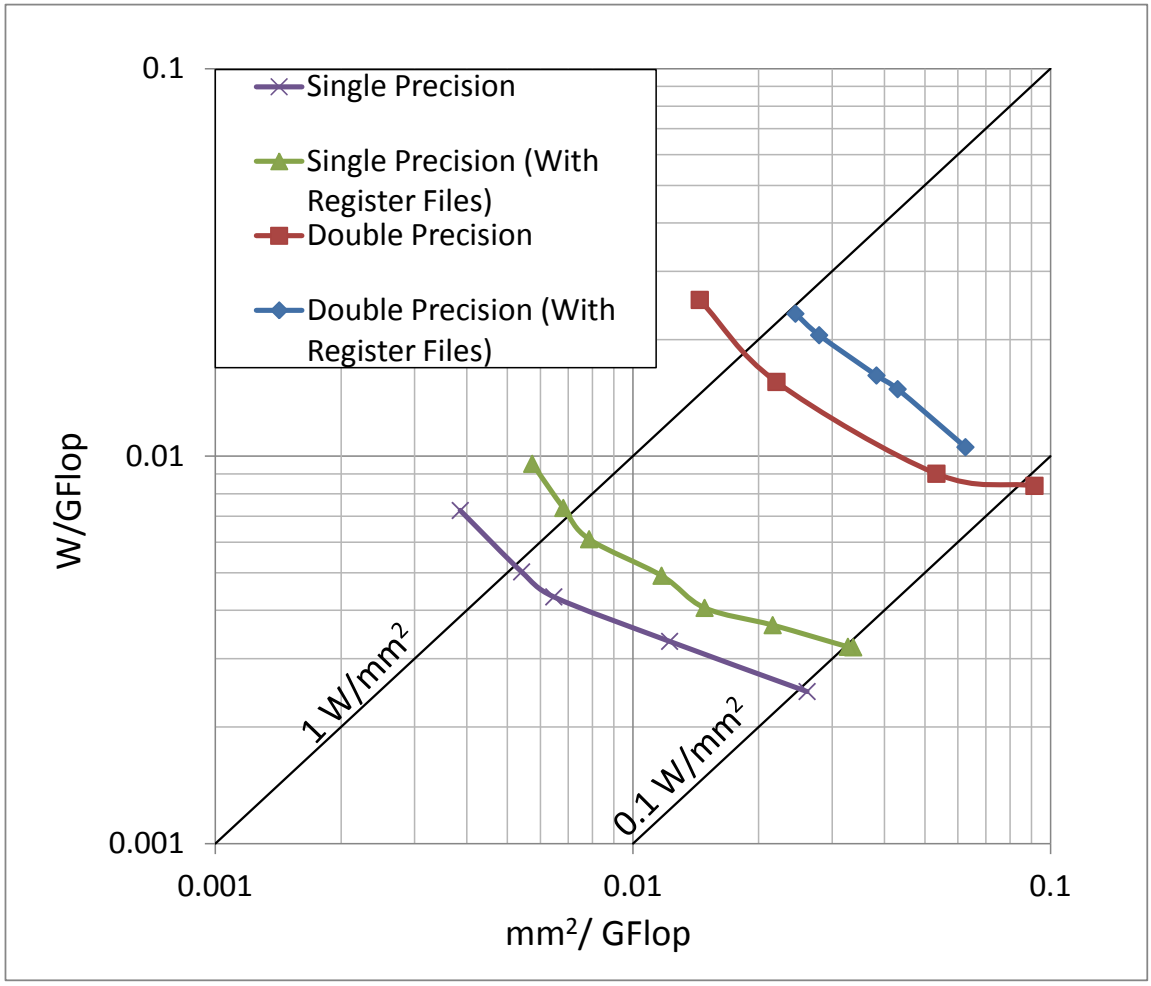


Figure 4.8: Energy throughput trade-offs for 45nm single and double precision FMA units showing the overhead of the required register file on performance. For constant power density, the required area overhead is under 30%.

datapath Parameters						
pipeline depth	4	5	5	6	8	8
Vdd	0.72	0.81	0.81	0.81	0.81	0.9
Clock period (ns)	3.04	1.78	1.53	1.05	0.82	0.48
Area (mm^2)	0.035	0.037	0.039	0.042	0.049	0.059
Energy (pJ)	16.8	18	25.4	27.9	31.1	60.4
Register file parameters						
Size (bytes)	512	1024	1024	1024	1024	1024
clock period (ns)	0.75	0.44	0.37	0.26	0.2	0.24
Access cycles	1	2	2	2	2	2
# of ports	1	1	1	1	1	2
# of banks	1	1	1	1	1	4
Area (mm^2)	0.006	0.011	0.011	0.011	0.011	0.026
Energy (pJ)	3.07	4.38	4.38	9.95	9.58	17.7
Total system metrics						
mm^2/GFlops	0.062	0.043	0.038	0.028	0.024	0.021
W/GFlops	0.011	0.015	0.016	0.02	0.023	0.039
W/mm^2	0.169	0.346	0.422	0.735	0.952	1.9

Table 4.1: Design parameters for the efficient frontier of 45 nm double precision FMA with register file.

a minimum of 16 registers required per thread, the 45 nm single and double precision FMA with latencies of 3-6 cycles is well served using 512 and 1024 bytes register files respectively. Due to the relatively small size of the register file, the access time is very small and therefore a single ported RAM operated at higher frequency than the datapath is most efficient. Table 4.1 illustrates the parameters of throughput efficient designs for the double precision FMA. These designs are different from the efficient designs identified when studying the FMA without the register file. Registers add an energy and area overhead of around 25% for single precision and 20% for double precision in 45 nm design, as illustrated in Figure 4.8. The overheads are larger for 90nm design, since the number of pipeline stages is larger to obtain the same power density.

Of course, this is the minimum overhead and assumes that all references fit into the register file. We estimate the effects of the memory system by looking at how the performance of a FMA changes with the arithmetic intensity of the application, using

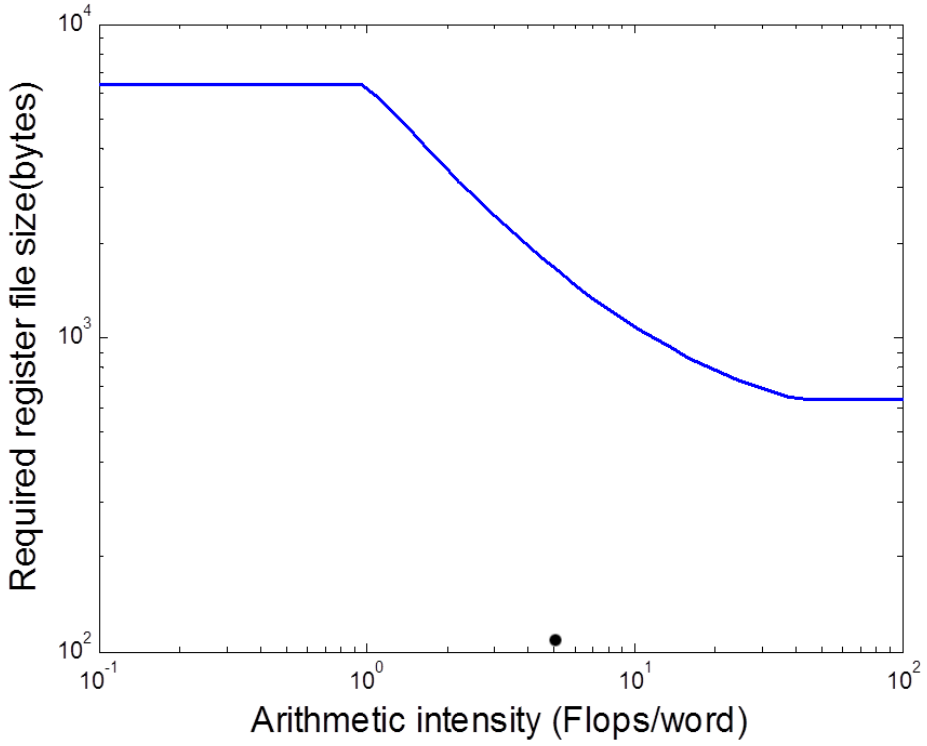


Figure 4.9: Example register file size vs. arithmetic intensity relationship for a double precision 5 stage 2 GFlops system with 100 ns of memory latency.

a very simple single level memory model. Assuming that the arithmetic intensity represents the average number of FP operations between memory fetches, Figure 4.9 shows how the size of the register file - required to feed the floating point unit with enough work - changes with arithmetic intensity; at lower arithmetic intensity levels, one needs more contexts to keep the FMA units busy. Probably more important is the energy cost of a DRAM access. Current high performance graphics DRAM (GDDR5) run around 1 nJ per double precision word fetch [3]. Given our estimates of around 25pJ/Flop in 45nm, it means one needs a ratio of over 40 Flops/double word load for the memory not to dominate the overall system energy. This is in line with current graphic systems which support over 500 GFlops of double-precision computation with 150 GB/s (19 GW/s) of memory bandwidth [1].

4.8 Effects of Technology Scaling

This analysis can be used to explore the expected gains in floating-point performance with scaling. We first compare the results of using 90 nm technology with the results using a 45 nm library, and Table 4.2 shows the detailed results of the efficient frontier data points. Figure 4.10 illustrates how throughput scales for resource constrained throughput and how minimum TCO cost scales for performance constrained throughput:

Resource Constrained Throughput Scaling If the power and area constraints remain the same, we should look at how the designs change for a fixed power density. The combination of shrinking area per functional unit and constant power density means that each functional unit must dissipate less energy. Since the energy consumed by logic gates does not scale fast enough due to slower Vdd scaling, we see that the architectures move to simpler, less pipelined designs. This means that the performance gain depends on the performance cost for moving to more energy-efficient designs. For example, 1 W/mm^2 efficient designs achieve 7× improvement since the trade-off curve was steep at this point in the 90 nm technology, so the required energy savings did not cost much in performance. Scaling designs at 0.1 W/mm^2 improve only 3.5× , since they reside on a flatter part of the trade-off curve. Unfortunately, in 45 nm even the 1 W/mm^2 designs are on a less steep part of the curve, indicating that further technology scaling will yield smaller performance gains.

Performance Constrained Throughput Cost Scaling The design that minimizes the total cost of ownership is the one which is tanget to the constant cost curves. For cost assumptions of 1 \$/mm^2, electricity cost of 10 ¢/KWh and power delivery and cooling cost of 0.5W overhead for every Watt of computation, the optimal design moves from being 0.2W/mm^2 design achieving a TCO of 11.6 ¢/GFlops per year in 90nm to the 45nm design with the double power density of 0.4W/mm^2 achieving 4.8× reduction in cost to 2.42 ¢/GFlops per year. The reason for the differing optimal power density is that the 45nm tradeoff is

Pipeline Depth	V_{th}	V_{dd}	Frequency (GHz)	Area (μm^2)		Power (mW)		FO4 (ps)	Cycle Time (FO4)	Latency (FO4)	W/ Gflops	mm^2/ Gflops	W/ mm^2
				Combinational	Total	Leakage	Dynamic						
90 nm Single Precision FMA													
12	low	1.08	1.54	66303	117325	80.1	72.2	25	26	308	0.0477	0.0381	1.25
10	normal	1.08	1.2	68578	113288	13.3	55.5	29	28	283	0.0277	0.047	0.59
8	high	1.08	0.66	53207	91206	1.5	7.4	35	46	369	0.0142	0.0689	0.21
6	high	0.9	0.2	44063	62925	0.54	2.63	42	111	665	0.0079	0.1554	0.05
90 nm Double Precision FMA													
14	low	1.08	1.43	219294	344275	290	200	25	28	387	0.1615	0.1205	1.34
11	normal	1.08	1.03	211645	311874	37	169	29	33	363	0.0855	0.1513	0.57
9	normal	1.08	0.75	175247	253832	35.7	58.4	29	45	408	0.0617	0.1688	0.37
7	high	1.08	0.39	159132	213689	4	19.2	33	78	545	0.0291	0.2725	0.11
7	high	0.9	0.28	147218	195243	3.6	10	45	79	555	0.0239	0.263	0.09
45 nm Single Precision FMA													
6	low	0.9	2.08	11258	16077	1.2	30.9	13	36	214	0.0072	0.0039	1.88
5	low	0.81	1.32	9945	14241	0.55	12.9	16	48	239	0.005	0.0054	0.93
4	low	0.81	0.98	9715	12670	0.58	8.09	16	64	257	0.0043	0.0065	0.67
3	normal	0.72	0.5	9415	12117	0.16	3.16	25	81	242	0.0033	0.0122	0.27
3	high	0.72	0.2	7735	10619	0.036	0.95	34	144	431	0.0025	0.0261	0.09
45 nm Double Precision FMA													
6	low	0.9	1.81	38444	49839	4.6	95.9	13	41	247	0.0253	0.0145	1.75
6	low	0.81	0.95	30964	42019	1.8	29.2	16	66	396	0.0155	0.0221	0.70
4	normal	0.72	0.33	28252	35058	0.4	5.6	25	122	486	0.009	0.0533	0.17
4	high	0.72	0.2	29610	36747	0.13	3.23	34	146	582	0.0084	0.0914	0.09

Table 4.2: Summary of Scaling Results for FMA Unit

flatter for the same power density and therefore it is cheaper to move to lower mm^2/GFlops without affecting the energy efficiency much thereby minimizing the total cost of the system while increasing the design power density.

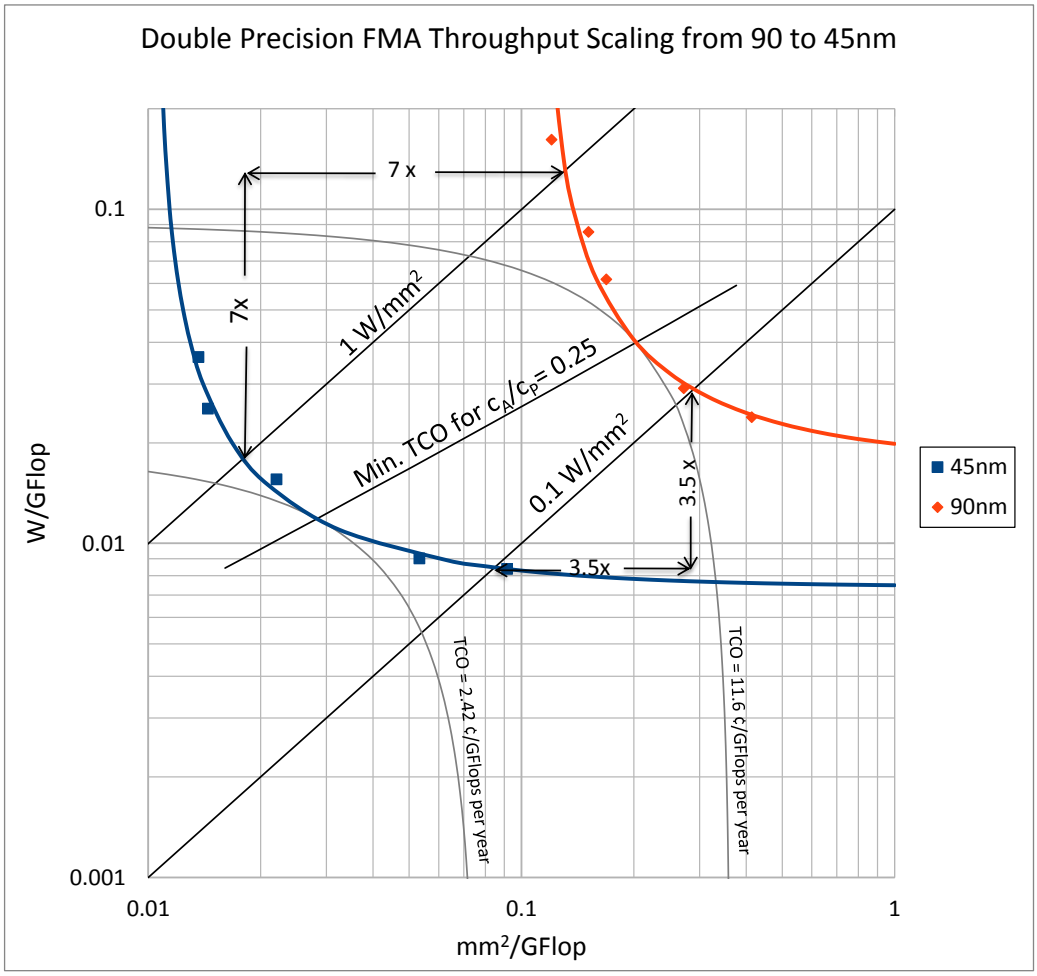


Figure 4.10: Scaling of FMA double-precision designs from 90 to 45 nm. For resource constrained designs, the performance gain depends on the power density allowed. For performance constrained designs, the minimum TCO design moves to higher power density due to flatter tradeoff curve. The cost equation is $\phi(\epsilon_A, \epsilon_P) = 32\epsilon_A + 128\epsilon_P$

Chapter 5

Scaling of Throughput

As mentioned earlier, classic Dennard scaling provides linear scaling of circuit delay and quadratic scaling of power while maintaining the power density constant. That translates to cubic improvement in power efficiency (W/GFlops) and area efficiency (mm^2/GFlops). However, as we found out in the last chapter, scaling designs from 90nm to 45nm, the power density of scaled designs increases and scaling of throughput performance varied between 7× for high power density designs to 3.5× for low power density designs instead of the 8× expected by the theory. This is because voltage scaling has slowed down from the 2× factor suggested by ideal scaling to 1.25× lowering the energy scaling to 3.125×. This is a consequence of the stalling of threshold voltage scaling due to unacceptable leakage power and underlying subthreshold slope. To estimate how the throughput tradeoffs will scale for technology nodes beyond 45nm for which we don't have standard cell libraries, we created a technology independent model of FMA with delays expressed in FO4, area expressed in λ^2 and energy normalized by CV^2. We then mapped these technology independent parameters to real performance area and energy using technology parameters extracted from SPICE simulations to generate tradeoffs. Section 5.1 introduces the parameters of technology independent FMA model and section 5.2 introduces how the technology metrics are extracted from SPICE simulation.

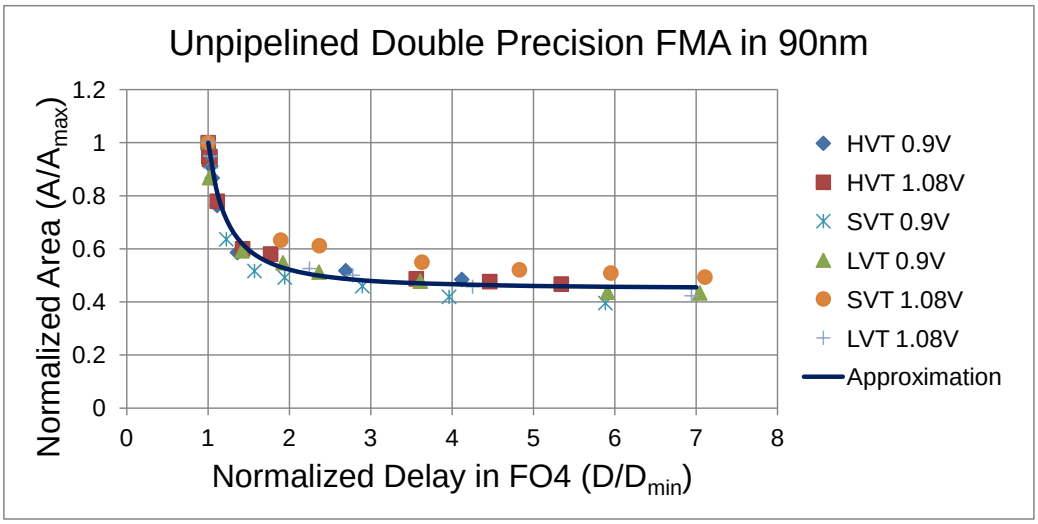


Figure 5.1: Area-Delay tradeoff for 90nm FMA is closely approximated by $0.45(\frac{A_{max}}{A}) + 0.55(\frac{D_{min}}{D})^2 = 1$

5.1 Modeling of technology independent FMA unit

The basic model for an FMA unit is composed of:

- An unpipelined design: which can be sized for different target delays. An unpipelined FMA design was synthesized for different target delays using different Vdd, Vth and gate sizing. All these designs are then normalized and plotted in Figure 5.1. By normalizing the area of the design to the max area and the delay to the delay of FO4 inverter, all the designs follow the same tradeoff curve. The delay is calculated using the minimum achievable unpipelined delay τ_{min} given in FO4, the area A_{max} given in feature size area λ^2, and the extracted shape function f which relates delay to area intensity (A/A_{max}) in equation (5.1).

$$\tau = \tau_{min} f(\frac{A}{A_{max}}) \tag{5.1}$$

- N pipeline stages: inserted at equal delay intervals D/N to the unpipelined design to increase clock frequency and improve throughput. However each additional pipeline stages add area overhead (A_{stage}) and delay overhead (D_{stage})

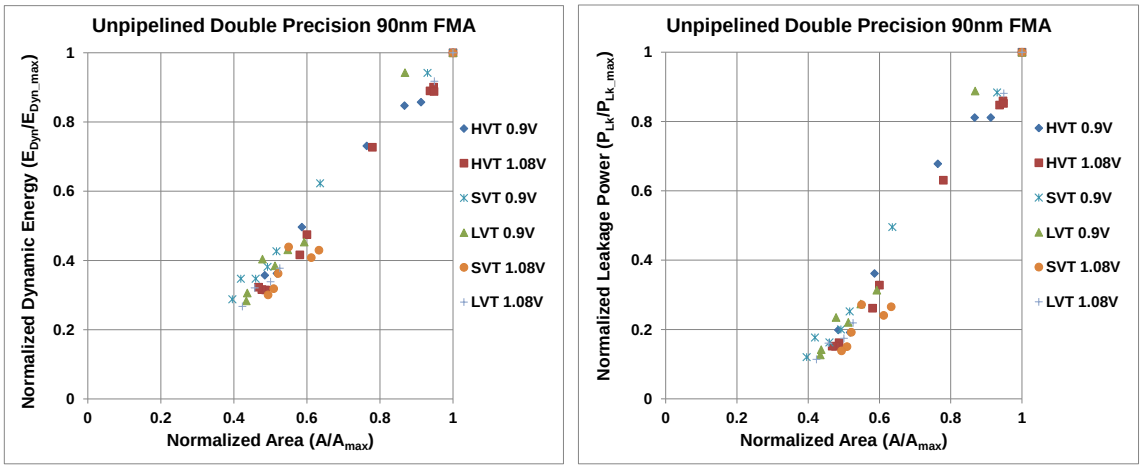


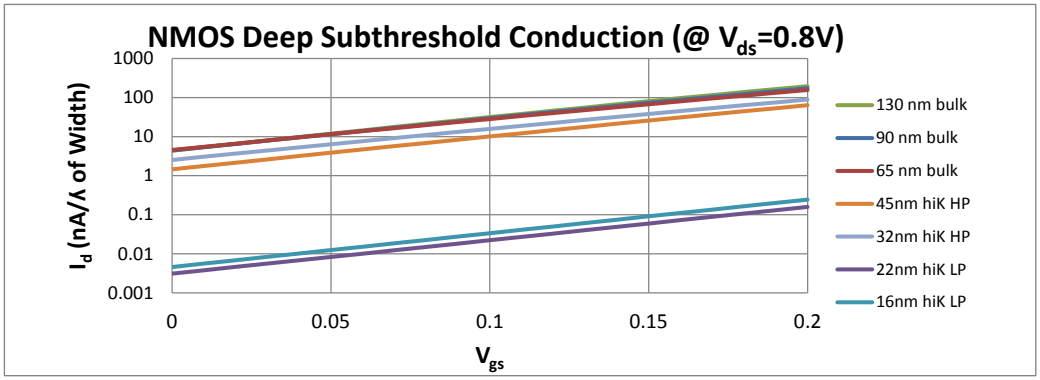
Figure 5.2: Leakage power and dynamic energy are roughly proportional to area for same supply and theshold voltages in synthesized FMA unit

which are model parameters.

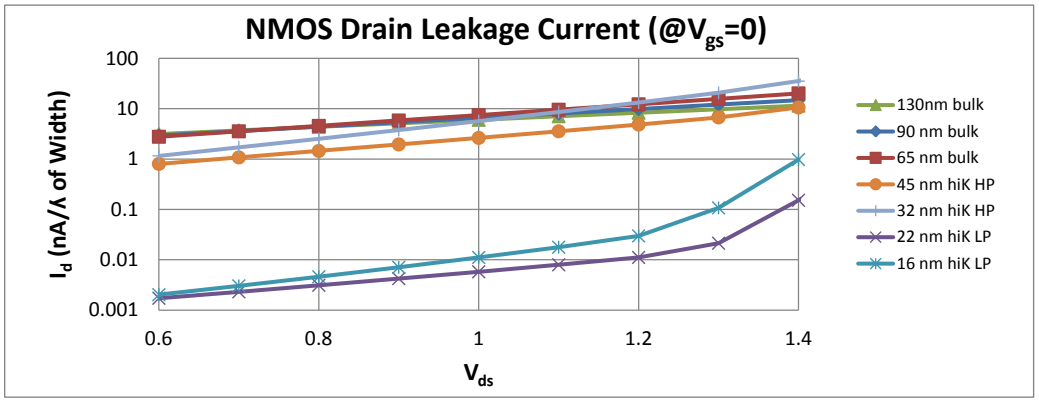
As for leakage power and dynamic energy, they are roughly propotional to design area for same supply and threshold voltages as seen from synthesized designs in figure 5.2. The final dynamic energy is decided by the CV_{dd}^2 decided by optimization and capacitance parameters as well as activity factor parameters $\alpha_{unpipelined}$ and α_{stage}. Leakage power is scaled by design area after being calculated in the optimization phase from V_{dd}, V_{th} and technology parameters.

5.2 Modeling of Technology Parameters

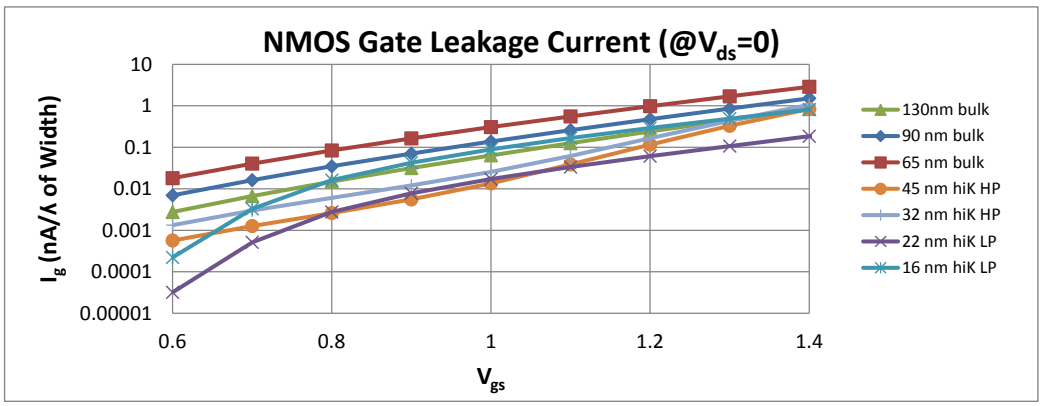
We used predictive technology models for planar CMOS technology from 180nm to 16nm, including high K gate technologies, high performance (HP) technologies and low power (LP) technologies [4, 42]. The SPICE models are based on predictions from the ITRS roadmap of 2007 [6]. Spice test circuits were used to extract the key technology metrics.



(a) Subthreshold Conduction Current Dependence on V_{GS}-V_{TH}



(b) Drain Leakage Current Dependence on Drain Voltage



(c) Gate Leakage Current Dependence on Gate Voltage

Figure 5.3: Subthreshold Conduction and Leakage Dependence on Gate and Drain Voltages

5.2.1 Leakage, Subthreshold Current and DIBL

Drain leakage in modern CMOS technologies depends on many factors. The drain leakage current is a subthreshold conduction current that is exponentially related to threshold voltage. The inverse slope of this exponential relationship S is an important parameter for measuring the effect of changing threshold voltage (e.g. decreasing threshold voltage by S increased leakage current 10x). Additionally for short channel CMOS transistors, threshold voltage is dependent on drain voltage and is lower for high voltages by a phenomonen called DIBL (Drain Induced Barrier Lowering). Equation (5.2) models the leakage drain current based on these factors. The inverse subthreshold slope S is estimated from simulation data in Figure 5.3(a) and used to estimate the DIBL parameter η in conjunction with leakage drain voltage dependence graphs of Figure 5.3(b).

$$I_{DrainLeakage} = I_{d0}10^{\frac{V_{gs}-V_{th}}{S}} = I_{d0}10^{\frac{V_{gs}-(V_{th0}-\eta V_{ds})}{S}} = I_{d0}10^{\frac{-(V_{th0}-\eta V_{dd})}{S}} \tag{5.2}$$

The other component of leakage current is gate leakage. Gate leakage due to tunneling exhibits an exponential dependence on gate voltage as shown in 5.3(c). The data is used to estimate base gate leakage current (I_{G0}) and exponential gate voltage leakage dependence slope (A) in equation (5.3).

$$I_{GateLeakage} = I_{g0}e^{a_g V_{gs}} = I_{g0}e^{aV_{dd}} \tag{5.3}$$

5.2.2 FO4 Delay

Fanout of 4 inverter chains are simulated at different power supply voltages to extract FO4 delay plots of Figure 5.4. The mobility is modeled using the accurate BSIM model Eq. (5.4) [17]. Velocity saturation model is used to estimate t_{FO4} using a fitting parameter t_{FO4_0} in Eq. (5.7). The resulting approximation fits the SPICE data very well as seen in Figure 5.4.

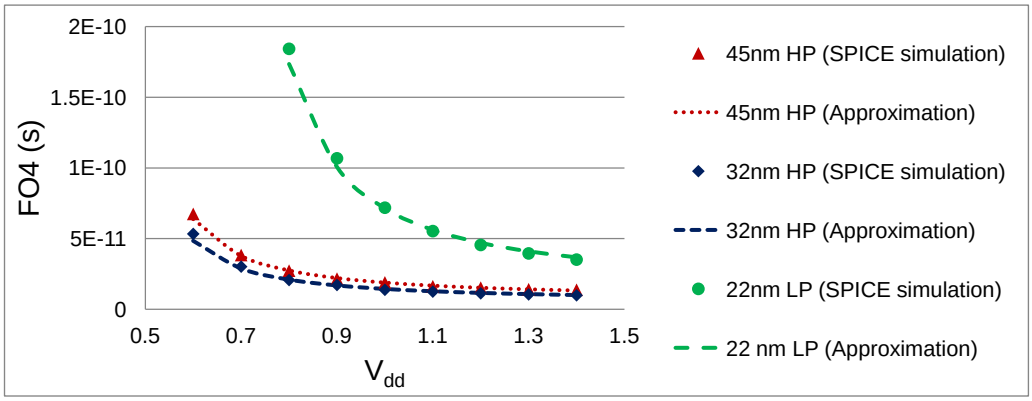


Figure 5.4: FO4 dependence on Vdd is approximated well by short channel model

$$\mu_{eff} = \frac{\mu_0}{1 + U_a(\frac{V_{gs}+V_{th}}{TOX}) + U_b(\frac{V_{gs}+V_{th}}{TOX})^2} \tag{5.4}$$
$$(for NMOS : U_a = 6E-10 m/V, U_b = 1.2E-18 (m/V)^2$$
$$for PMOS : U_a = 2E-9 m/V, U_b = 5E-19 (m/V)^2)$$

$$E_{sat} = \frac{2\upsilon_{sat}}{\mu_{eff}} \tag{5.5}$$

$$V_{th} = V_{th0} - \eta V_{dd} \tag{5.6}$$

$$t_{FO4} = t_{FO4_0} \frac{V_{dd}(V_{dd} - V_{th} + E_{sat}L)}{(V_{dd} - V_{th})^2} \tag{5.7}$$

5.2.3 Capacitance

Dennard scaling expects capacitance per transistor to scale linearly with device scaling making the capacitance per unit width of the device constant. However in recent modern technologies, as thickness of transistor gates reached just few atomic layers,

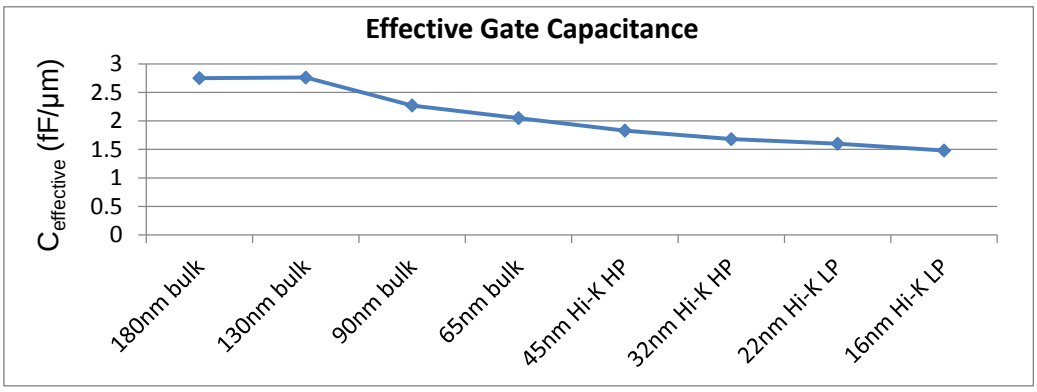


Figure 5.5: Effective gate Capacitance

scaling of gate thickness has slowed down due to increased gate tunneling. This results in the gate capacitance per μm of width dropping from around 2.5 fF/μm to around 1.5 fF/μm as shown in Figure 5.5. The scaled capacitance values are used to give a more accurate prediction of dynamic energy of the design.

Here is a summary of the scaling model design and technology parameters and how the equations tie them together to estimate scaled designs.

Optimization Parameters:

- V_{dd}: supply voltage
- V_{th0}: long-channel threshold voltage
- N: number of pipeline stages
- I_A: area intensity or ratio of logic area to maximum logic area A_{max}

FMA Parameters:

- τ_{min}: minimum logic delay in FO4's
- A_{max}: maximum logic area associated with τ_{min} in λ^2 's
- τ_{stage}: pipelining stage delay in FO4's
- A_{stage}: pipelining stage area in λ^2 's
- α_{logic}: activity factor for logic circuits
- α_{stage}: activity factors for pipeline stage

Technology Parameters:

- λ: feature size
- μ_0: low field mobility
- υ_{sat}: saturation velocity
- η: DIBL
- $C_{\mu m}$: capacitance per μm
- S: inverse subthreshold slope in mV/decade
- I_{g0}: base gate leakage current
- a_g: gate leakage exponential voltage slope
- I_{d0}: base drain leakage current when V_{th0}=$V_{th0TECH}$
- $V_{th0TECH}$: technology V_{th0}

Model:

$$A = (I_A A_{max} + N A_{stage})\lambda^2 \tag{5.8}$$

$$\mu_{eff} = \frac{\mu_0}{1 + U_a(\frac{V_{gs}+V_{th}}{TOX}) + U_b(\frac{V_{gs}+V_{th}}{TOX})^2} \tag{5.9}$$

$$E_{sat} = \frac{2\upsilon_{sat}}{\mu_{eff}} \tag{5.10}$$

$$V_{th} = V_{th0} - \eta V_{dd} \tag{5.11}$$

$$t_{FO4} = t_{FO4_0}\frac{V_{dd}(V_{dd} - V_{th} + E_{sat}L)}{(V_{dd} - V_{th})^2} \tag{5.12}$$

$$\tau = (\tau_{min} f(A_{intensity}) + N\tau_{stage})t_{FO4} \tag{5.13}$$

$$E_{Dyn} \propto (\alpha_{logic} I_A A_{max} + \alpha_{stage} N A_{stage}) C_{\mu m} \lambda V_{dd}^2 \tag{5.14}$$

$$P_{Lk} \propto A V_{dd}(I_{d0} 10^{\frac{V_{th0TECH} - V_{th}}{S}} + I_{g0} e^{a_g V_{dd}}) \tag{5.15}$$

$$E = E_{Dyn} + P_{Lk}\frac{\tau}{N} \tag{5.16}$$

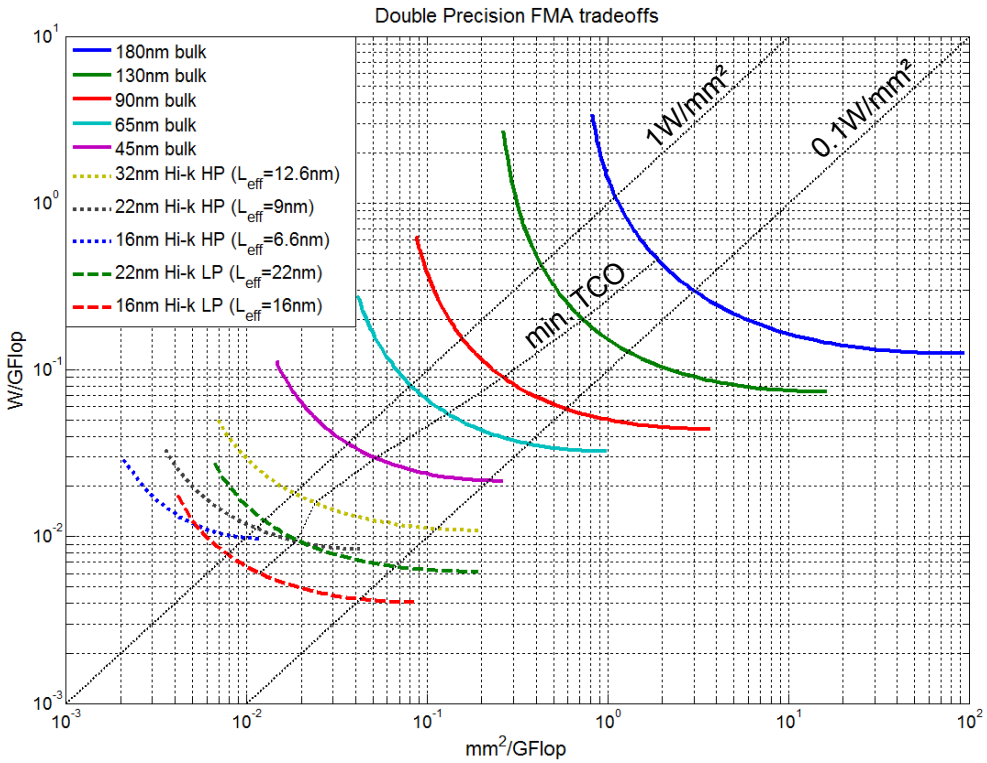


Figure 5.6: Double Precision FMA scaling from 180nm down to 16nm

5.3 Planar CMOS scaling

Using an optimization framework in Matlab, the technology and design parameters of the model were used to extract throughput efficient designs for different technology nodes. The results are shown in Figure 5.6. We notice that for bulk technologies the tradeoffs of smaller technology nodes keep going more to the left implying that energy scaling is slower than area scaling. This necessitate moving to lower energy technologies. This transition first happens around 45-32nm from bulk technologies to hi-K gates and later from high performance technologies (HP) to low power technologies (LP) around 16nm. One also notices that the distance between the tradeoff curves are getting smaller as gains from scaling keep getting smaller. Figure 5.7 shows the incremental improvement over previous generation to shrink from 2.57× in scaling from 180nm to 130nm to 1.44× in scaling from 22nm to 16nm.

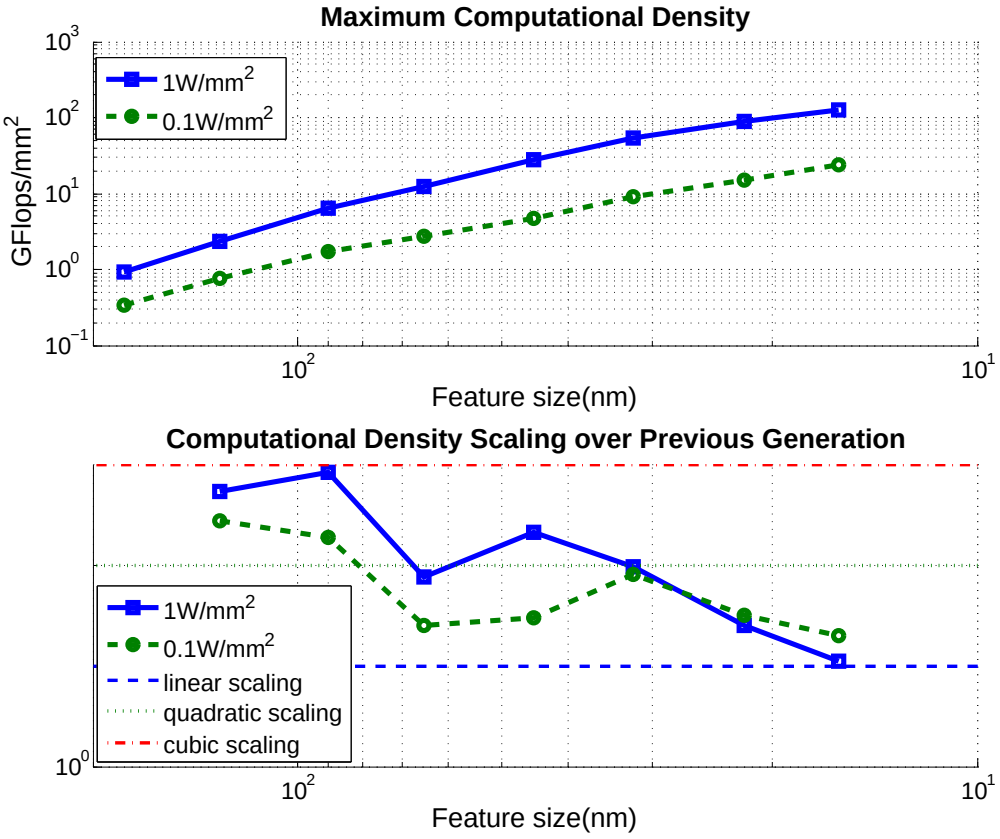


Figure 5.7: Double Precision FMA scaling from 180nm down to 16nm for 1W/mm^2 and 0.1W/mm^2 optimal designs

Looking at optimal designs for 1 W/mm^2 we notice the stalling of supply voltage scaling. While designs scale linearly from 1.712V to 1.232V in transition from 180nm to 130nm, it stays roughly constant around 0.7V for 22nm and 16nm. This results in energy scaling cubically from 180n to 130nm and linearly from 22nm to 16nm. In the intermediate interval between 90nm and 32nm, for higher energy 1W/mm^2 designs, additional scaling results from moving to lower energy designs resulting in an intermediate quadratic scaling. First the optimal voltage can be slowly scaled at the expense of slower scaling of intrinsic technology speed at ∼40ps FO4 delay. Additionally, optimal designs move to shallower pipelines and more logic per stage to decrease energy at the expense of area efficiency. On the other hand, 0.1 W/mm^2 designs are already operating at the shallow part of the tradeoff curve between area and power efficiency (low sensitivity for changing design parameters), and as such the voltage does not scale and stays around 0.7V while the designs stay relatively the same or move slowly to shallower designs as well. In effect the scaling of 0.1W/mm^2 designs is always less than 1W/mm^2 designs as shown in Figure 5.7(b).

Supply voltage scaling is highly dependent on threshold voltage scaling which in turn depends on the subthreshold slope of the transistor, which indicates how much voltage is needed for a decade change in subthreshold conduction current. To ensure low enough off leakage, V_{th} must be at least 3× (4×) the subthreshold slope for 1W/mm^2 (0.1W/mm^2) as shown in Figure 5.8(b). This setting of V_{th} results in leakage energy of about 20-30% as shown in Figure 5.8(d).

Under classical Dennard scaling, no design changes are needed for optimal throughput designs. Supply voltages and technology speed (FO4) scales linearly while energy per operation scales cubically. That allows a 2× scaled down designs to run unchanged at 2× higher clock speeds while operating at the same power density. However when supply voltage is slowly scaling due to leakage issues, the energy per operation scales slower than area. Thus design changes are required for a design to stay within the same power density. Therefore optimal logic per stage increases from 25 FO4 in 180nm to around 30 FO4 in 16nm for 1W/mm^2 designs which results in savings in dynamic energy of around 20% as shown in Figure 5.9 (a) and (f).

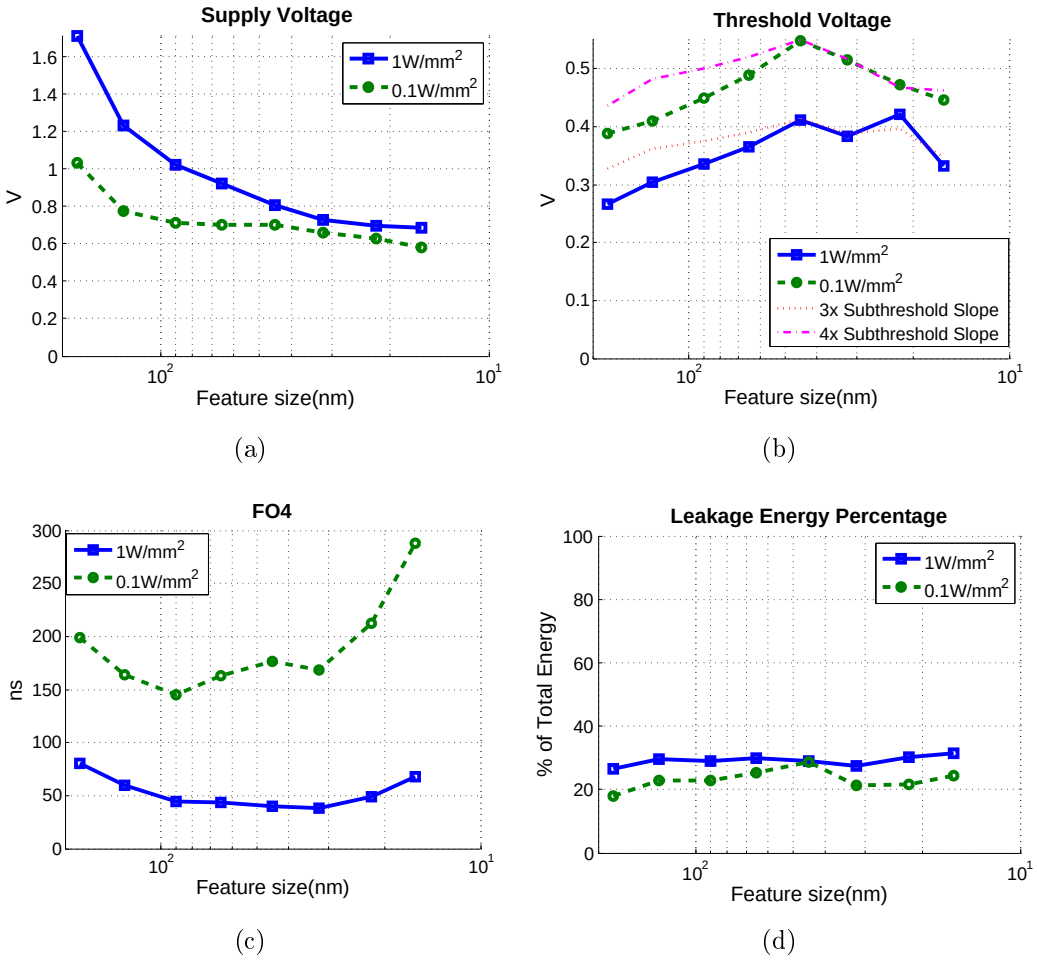
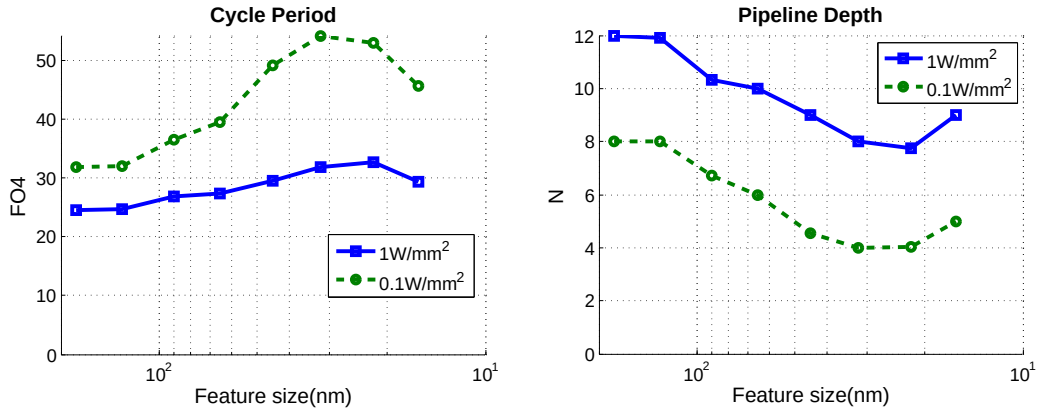
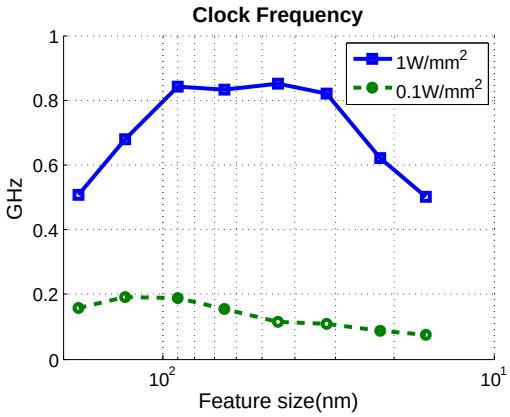


Figure 5.8: Voltage and Energy Scaling Parameters for 1W/mm^2 and 0.1W/mm^2

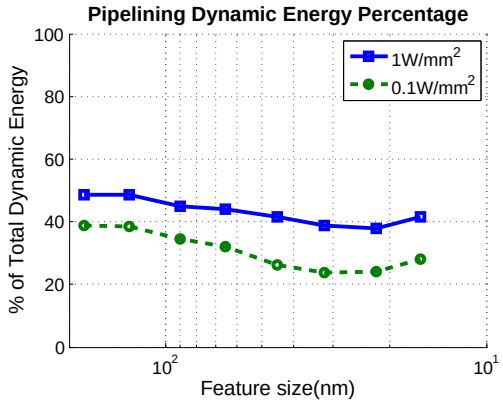


(a)

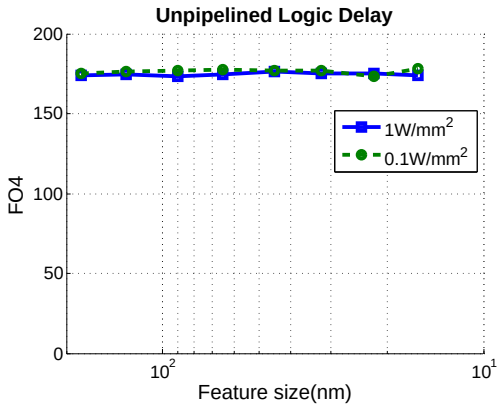
(b)



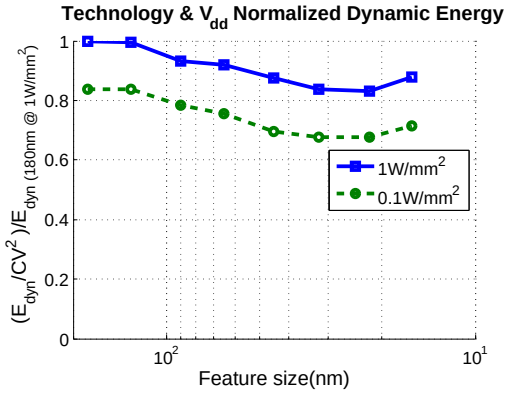
(c)



(d)



(e)



(f)

Figure 5.9: Timing Scaling Parameters for 1W/mm^2 and 0.1W/mm^2

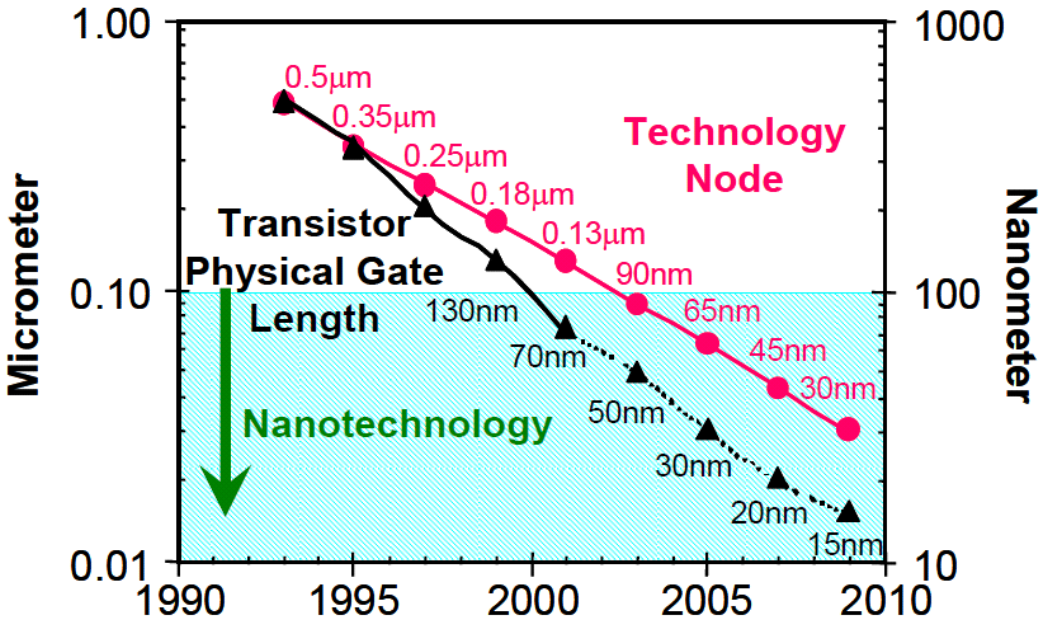


Figure 5.10: Aggressive Physical Gate Length Scaling. Reproduced from Intel[15]

5.4 Future Scaling

In the past few years physical gate length scaling has deviated from traditional Dennard scaling trends and has been more aggressively scaled than the rest of the dimensions. For example Intel 65nm technology features 35nm physical gate length [10]. Figure 5.10 illustrates Intel projection for physical gate length scaling extracted from [15]. The ITRS roadmap for scaling of 2007 took the same projections for HP technologies down to the 16nm which predicted a physical gate length of 6.3nm. Clearly the generated tradeoffs for such technology nodes has unacceptable energy efficiency such that it is more throughput efficient to use LP technologies with physical gate length of 16nm rather than using HP technologies even for high power density of $1W/mm^2$ as shown in Figure 5.6. This is due to the poor electrostatics of these technologies as they suffer from high subthreshold slope. The high ratio between effective oxide thickness (EOT) and effective gate length (L_{eff}) induces a high subthreshold slope as illustrated in Figure 5.11. HP 16nm has a subtheshold slope of 0.145 V/decade while LP 16nm has a subthreshold slope of 0.115V/decade. This translates

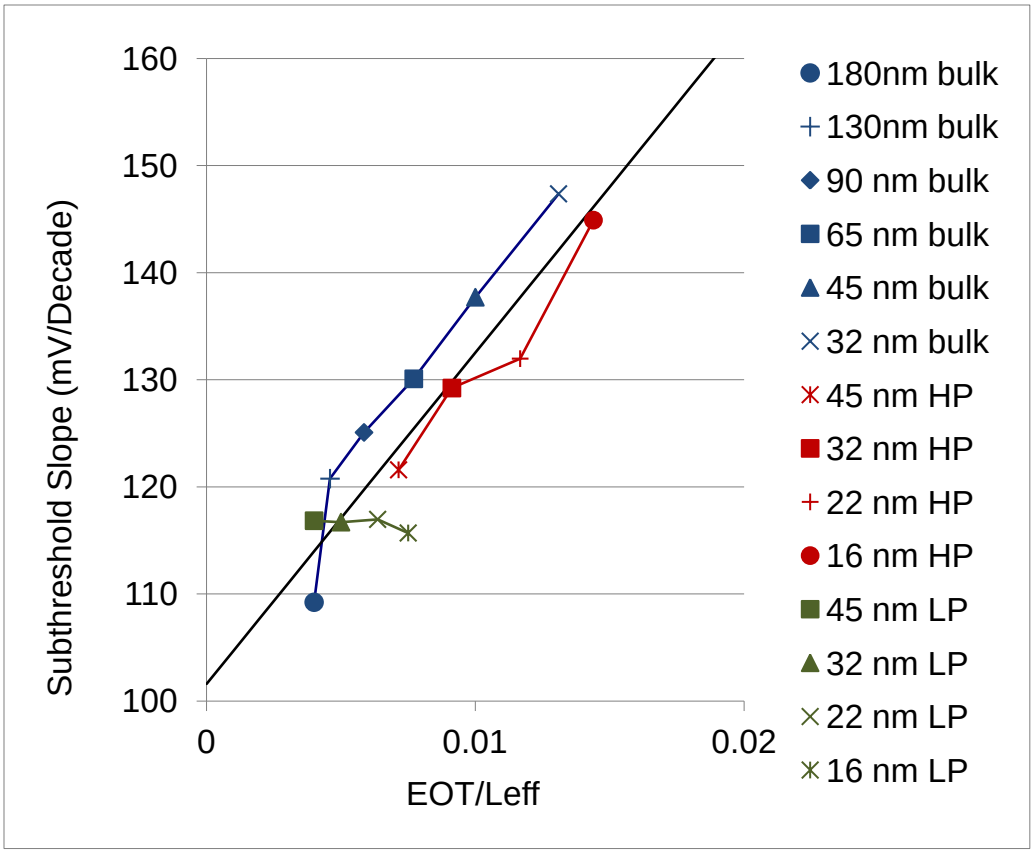


Figure 5.11: Effect of aggressive Leff Scaling on Subthreshold Slope

to effective threshold voltages of 0.42 and 0.33 for 1W/mm^2 optimal designs with the reduced leakage making the LP design 20% more energy efficient. Recent studies have shown similar prediction for need to back down from aggressive gate length scaling for future scaling [14] [22]. Also the ITRS roadmap of 2009 now has a prediction of 12.8nm for 15nm technology in departure of the early aggressive target of 6.3nm from 2007. So in a nutshell aggressive gate length scaling is dead and move to lower energy technologies is needed in the short term consistent with the trends of moving to lower energy designs.

Another attempt at addressing these scaling issues is through the use of ultra thin silicon layers, which have better electrostatics. Intel has announced finFETs that have improved transistor operation by having 3D structure of the gate that better

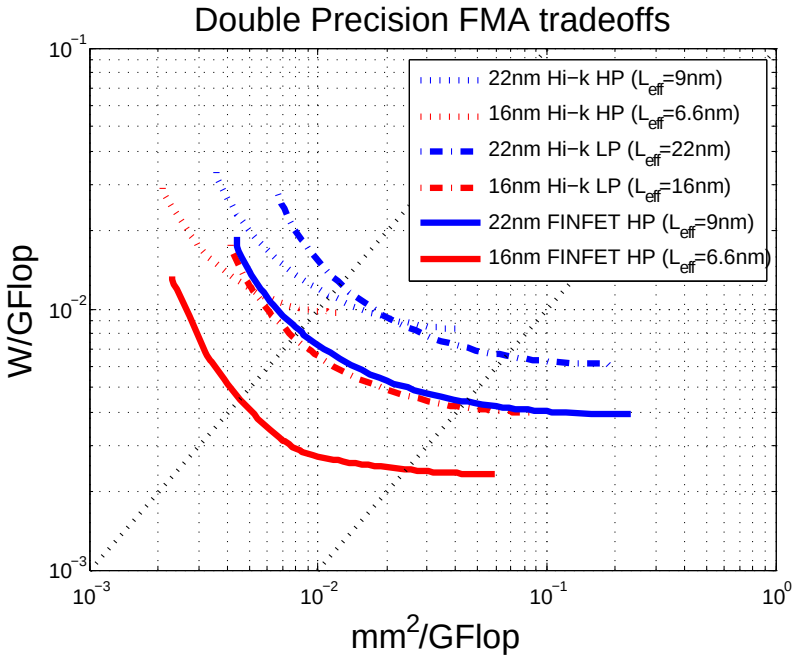


Figure 5.12: FINFET technologies impact on scaling

control the channel conductivity. Using details extracted from Intel finFET disclosure [24], the technology seems to have an advantage for throughput scaling illustrated in Figure 5.12. An optimal FINFET design in 16nm can run on 0.6V supply voltage in comparison of 0.7V in 16nm LP bulk technology shaving 20% dynamic energy. Additionally leakage energy goes down from 28% to 5% of total energy. All in all, it provides 1.75× energy scaling over traditional CMOS technology for 1W/mm^2 power density.

5.5 Summary

While feature size and area per function continues to scale as expected by Moore's law, the energy per operation scaling is slowing down from cubical to linear. The implications of this change in scaling nature are that design and process changes across technology generations are needed to achieve the throughput optimal designs

for a given area and power budgets. Lower energy designs with shallower pipelines, slower clock frequency and longer channel length are needed to eke out some extra energy savings to the ones supplied by scaling. Technology innovations to produce transistors with better electrostatics and subthreshold slopes such as the finFET technology are key to continued scaling of throughput.

Chapter 6

Latency Sensitive FMA Design

Having explored throughput designs in the previous chapter, in this chapter we focus on optimizing the design of FPUs in CPUs, which are more latency sensitive than GPU designs. We evaluate the design alternatives using the SPEC CFP2000 floating point benchmark suite [2]. To understand how FP latency affects these applications, we classify FMA dependencies according to where the result is used in a subsequent instruction, as shown in Figure 6.1:

- **Accumulation Dependency**: the result is accumulated in a subsequent FADD or FMADD instruction (bypass through f_B).
- **Multiply-Add Dependency**: the result goes through a fused multiply and then an add (bypass through f_A or f_C).
- **Other Dependencies**: the dependent instruction is not FMADD, FADD or FMUL.

A tuple notation is used to indicate the latency for the different kind of dependencies to compare different designs. For example, a (3,7,8) design has a 3 cycle accumulation latency, 7 cycle multiply-add latency and an 8 cycle latency for other non-FMA dependent instructions.

Traditional FMA design does not make a distinction between the latency of accumulation and multiply-add, resulting in designs that have equal latencies for all

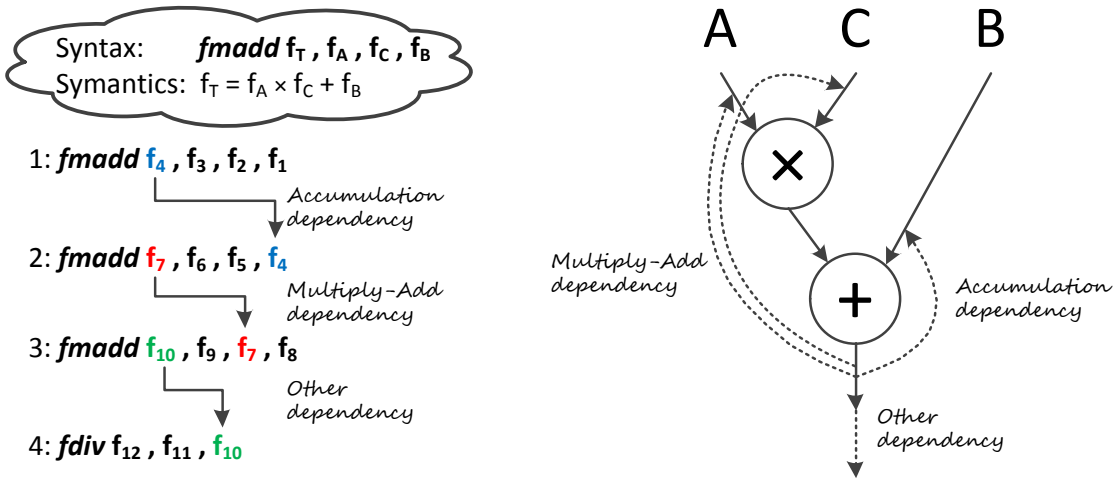


Figure 6.1: FMA Latency Types showing on the left the types of dependencies as they occur in instruction sequences and their corresponding data feedback path on the logical implementation

dependencies. For example, the IBM Power5 FMA is a (6,6,6) design, but the Power6 FMA is (6,6,7), because the design is optimized to handle forwarding of dependent instructions before the rounding stage [38]. We review such a design in Section 6.1.1 and use it as a reference for a state of the art FMA design. We then introduce our cascade implementation of the FMA instruction (CMA) which has been optimized for accumulation dependencies with a small effect on the other latencies. CMA allows the accumulation operand f_B to enter the pipeline much later than in a traditional FMA implementation, allowing for shorter accumulation latency. We then optimize this path by introducing overlapping bypass paths for exponent and significand to make the accumulation dependent latency as short as possible. We demonstrate how a CMA can achieve a (3,7,8) latency at the same clock rate of an FMA(6,6,7). Figure 6.2 shows the FMA and CMA pipelines and their bypass paths and how these bypass paths reduce the effective latency of the instructions

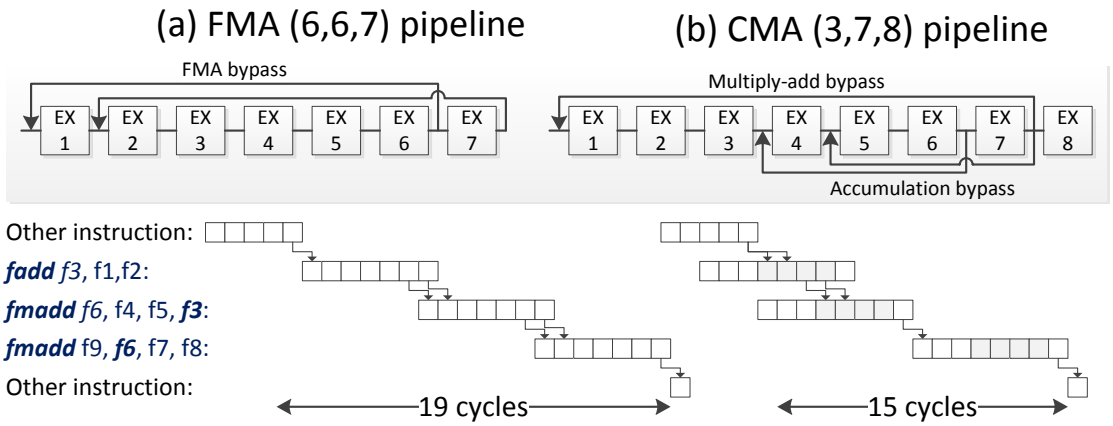


Figure 6.2: Block diagram of FMA and CMA pipelines with their respective bypass paths, and a timing diagram of an example instruction trace for both pipelines. The CMA architecture has shorter accumulation latency than FMA.

6.1 Evaluated FMA Design Variations

In this section, different design variations of FMA are presented that have different microarchitectures and latency tradeoffs. These designs are evaluated for overall performance and energy consumptions in later sections.

6.1.1 Traditional FMA Architecture FMA(6,6,7)

The Power6 FMA is a recent IEEE-compliant 7 cycle 13 FO4 design with a 6 cycle latency for dependent instructions (Figure 6.3). It achieves the reduced dependency latency by forwarding the unrounded results with special control signals to indicate if the result is to be incremented. Special terms added in the multiplier tree are used to generate the correct product. For example, if A is forwarded and Increment signal is asserted, an additional A term is added in the multiplication tree to produce $A \times C + A = (A+1) \times C$. Such a design has (6,6,7) latency by the metrics introduced earlier. This FMA design is used as the standard design for comparison because it is IEEE-compliant and has the shortest latency of FMA architecture for the least area and energy.

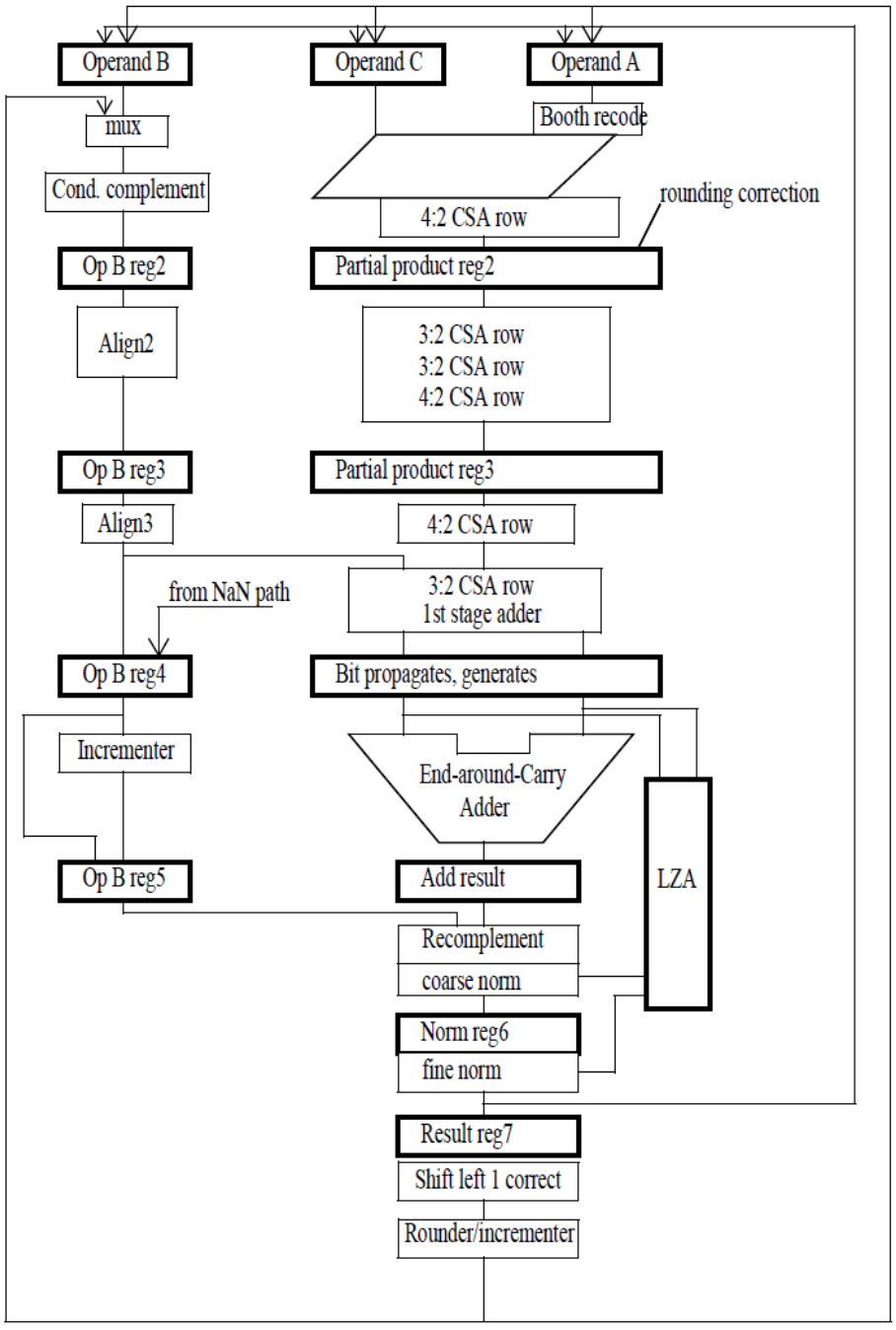


Figure 6.3: Power6 FMA Significand Datapath (reproduced from [38])

6.1.2 Cascade Multiply Add architecture CMA(3,7,8)

One can compute a multiply add by simply cascading the addition operation after multiplication. However because of the requirement of unlimited precision for intermediate results of FMA instructions, the multiplier and adder are different from traditional floating point adders/multipliers. For example, a double precision CMA design contains the following stages:

- A multiplier that takes 2 double-precision operands A,C to generate the result $A \times C$ in "quad" precision (106 bit mantissa, 13 bit exponent)

- An asymmetric adder that takes a double precision operand B and the "quad" precision multiplier output to produce a double-precision result $(A \times C) + B$

Thus, CMA is just an implementation variant of FMA that produces exactly the same result for FMADD instructions with unlimited intermediate precision and rounding only once at the end.The add portion can be optimized to be very fast using parallel paths algorithms where either alignment or normalization steps are saved [18] which might make up for the slight increase in overall latency. The overall latency increases because the multiplier tree outputs are combined using an adder before being fed to the cascaded adder. Since the add operations start "late" in the overall pipeline, forwarded results cause less stall time than would occur in a normal FMA. Figure 6.4 illustrates the datapath of the significand of the CMA design we have developed. It employs an adder with far path datapath for calculating the sum or difference when the exponent difference is greater than 1 and a close path datapath that calculates the difference when the exponent difference is ≤ 1, which is the only case where there could be massive cancellation and a need for a big normalizing shifter. The design has been optimized to shorten accumulation latency and handle forwarding of unrounded results (with increment signals) to shave a cycle off the accumulation and multiply-add latencies as was done in the FMA design. The next two sections discuss the details of these optimizations.

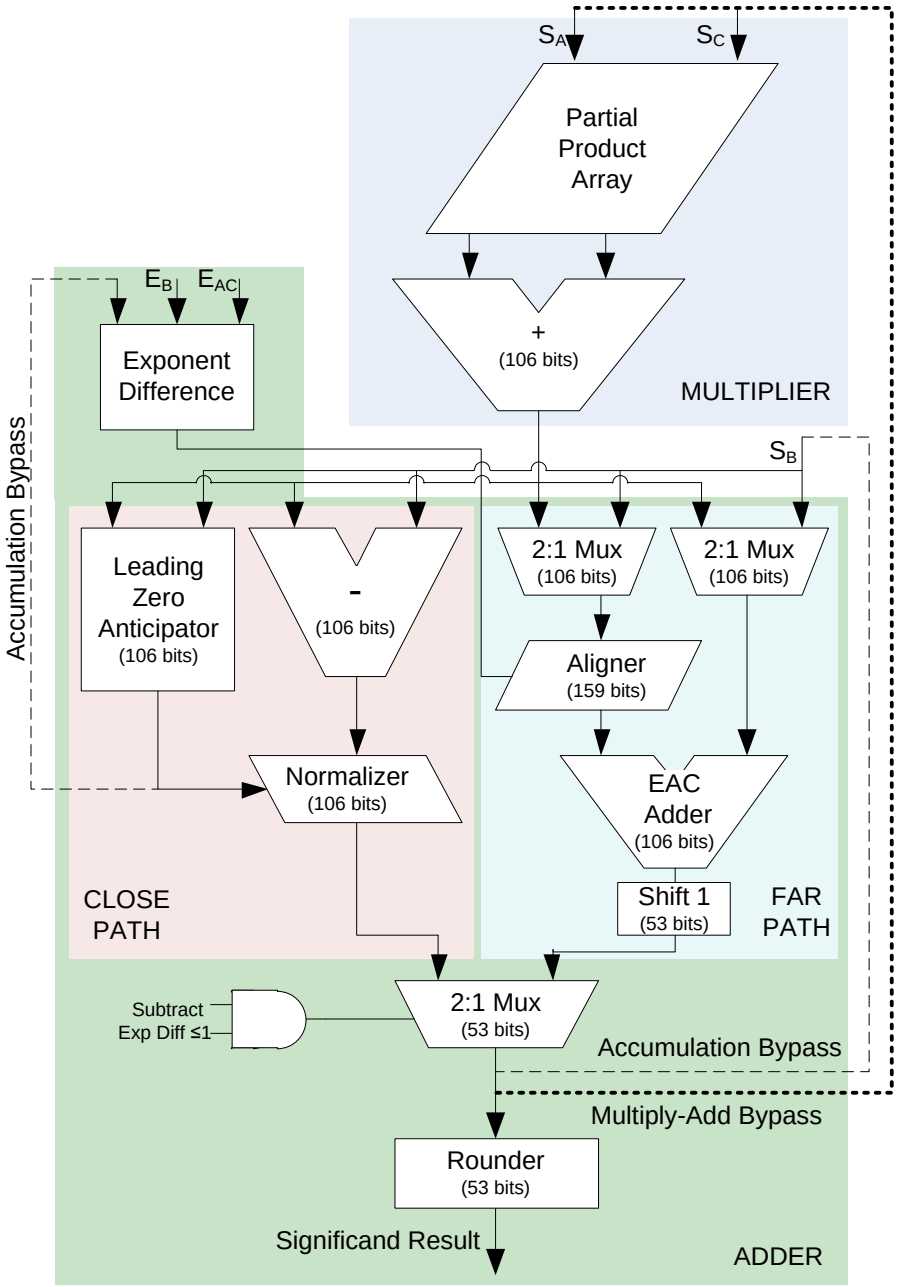


Figure 6.4: Simplified CMA significand datapath (multiplier; adder: far path, close path) with accumulation bypass path shown as dashed line and multiply-add bypass path shown as dotted line.

Removing Rounding Latency Overhead

To reduce the overall latency of dependent instructions, our CMA design implements a bypass path for dependent instructions that forwards the unrounded result and an increment signal. Implementing the bypass for the multiplier inputs A , C is similar to the Power6 design. We modify the multiplier tree to have one extra term that can be either S_A if Inc_A signal is asserted, or S_C if Inc_C is asserted. As for the input B, the adder part has been modified to accept the inputs S_B, Inc_B and $S_{A\times C}$. The idea is to merge the incrementation of B with the addition to $A \times C$ using carry save adders. The implementation of the close path and far path adders that support the increment signal is done as follows:

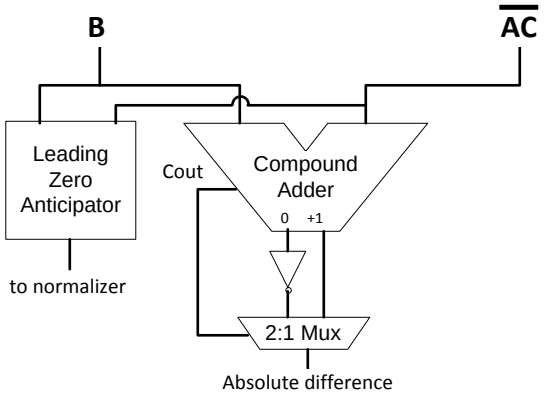
Close Path The close path handles the subtraction case of $S_{A\times C}$ (106 bits) and S_B (53 bits) which are aligned on the MSB. The absolute difference of two binary numbers x, y is usually calculated as follows:

$$abs(x-y) = \begin{cases} x - y = x + \overline{y} + 1 & , y < x \\ -(x - y - 1) - 1 = \overline{x + \overline{y}} & , y \geq x \end{cases} \tag{6.1}$$

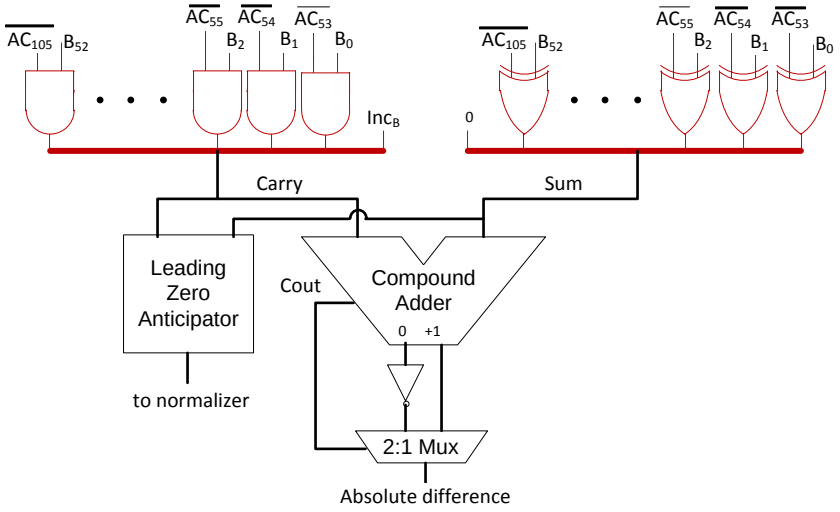
Therefore, the operation can be implemented using a compound adder fed by x and $\overline{y}$ to produce $(x+\overline{y})$ and $(x+\overline{y}+1)$, which are muxed out to produce the correct absolute difference based on the adder carry out from $(x+\overline{y})$.

Additionally, S_B needs to be incremented before the absolute difference operation if Inc_B is asserted. It is straightforward to merge the incrementation of S_B with the absolute difference operation by introducing a half adder at the input to produce sum and carry vectors for the compound adder and leaves a bit position at LSB where Inc_B is inserted as shown in Figure 6.5.

Far Path The far path handles addition and subtraction when the exponent difference is greater than 1 (Figure 6.6). The addend with the bigger exponent (S_{big}) can be as wide as 106 bits for double precision inputs. The addend with the smaller exponent (S_{small}) is shifted right by the amount of exponent difference and becomes



(a) Original Close Path Datapath



(b) Modified Close Path Datapath

Figure 6.5: Modifying Close Path to Support Incrementation Signal $\mathrm{Inc_B}$. The inputs are added using a half adder, which frees up one of the LSB inputs to put the $\mathrm{Inc_B}$ signal in

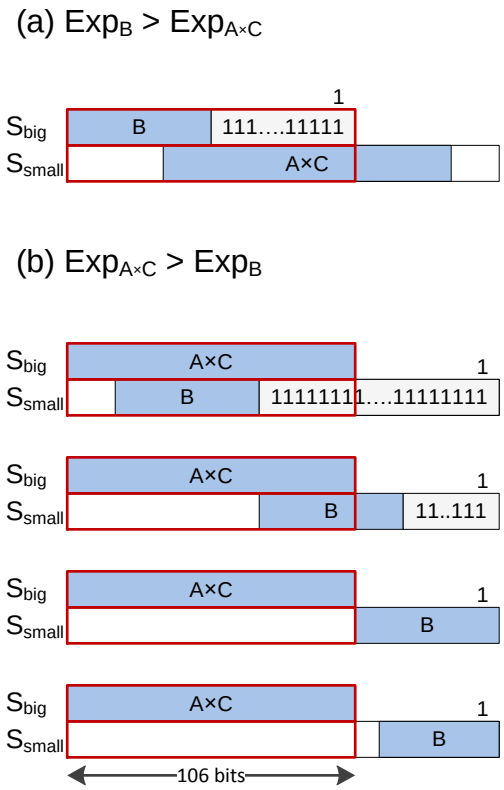


Figure 6.6: Far Path addition of mantissa of B and $A \times C$ with Inc_B asserted. The boxes indicate the portion of the fractions that are fed to the adder. The padded ones and 1 added at the least significant bit produce the equivalent of increment of B. In case $Exp_{A\times C} > Exp_B$: Carry in to the 106 bit adder is carry in to the effective 159 bit adder ANDed with the 53 LSBs, since they all need to be 1 for the carry to propagate to the upper 106 bits.

159 bits wide after shifting. In case of subtraction, S_{small} is inverted before being fed to the adders. A compound adder of 106 bits summing S_{big} and S_{small}[158:53] produces sum and sum+1 which is sufficient for calculating the sum and difference [35]. Finally, only the uppermost 53 bits of the result is retained after normalization (possible right shift in case of addition and left shift in case of subtraction) and guard and sticky bits are calculated. To support incrementation of S_B, the design is modified by having an adder that produces sum, sum+1, and sum+2. Choosing between the three results gives the equivalent result of incrementing S_B before the add operation. The correct result is chosen according to the following rules:

- When $Exp_B > Exp_{A\times C}$ (Figure 6.6(a)): S_B is right padded with Inc_B. and:

  $S_{big} = S_B, \{(53)\{Inc_B\}\}$
  $S_{small} = \{S_{A\times C}, 53'b0\} >> (Exp_B - Exp_{A\times C})$

  If Inc_B is asserted, the result of addition becomes sum+1, while the result of subtraction becomes sum+2.

- When $Exp_{A\times C} > Exp_B$ (Figure 6.6(b)): S_B is the smaller fraction, and in case of incrementation, we need to add 1 to the LSB of S_B which is then fed to the alignment shifter. To combine the incrementation with alignment and add operation we pad the lower bits with Inc_B so that after shifting, adding 1 to the LSB is still equivalent to incrementing S_B before shifting. Logically for S_{small} we will create a 159 operand to feed into the adder, and we will add the carry at the LSB. So

  $S_{big} = S_{A\times C}$
  $S_{small} = \{S_B, (106)\{Inc_B\}\} >> (Exp_{A\times C} - Exp_B)$

  Since S_{big} is zero for the 53 LSBs, carry-in to the 106 bit adder is generated by carry-in ANDed with the lower 53 bits of S_{small} which is used to choose between sum and sum+1 in the case of addition. This handles all the shift cases.

As for subtraction, S_{small} is inverted before being fed to the adder. Since $S_{small} = \overline{S_{small} + 1}$, then the result of subtraction is always sum if Inc_B is asserted.

Figure 6.7 is a block diagram illustrating the above-described combining of shifting and addition in the far path.

Optimizing the Accumulation Loop

The accumulation loop can be reduced by noticing that the result exponent is known to within ±1 in advance of the result mantissa in carry save format as an output of the adder. In the near path, the exponent is the difference between the larger exponent and the leading zero anticipator (LZA) count. In the far path, the exponent is just the bigger exponent of the two addends, but might be incremented if a late right shift is needed in case of addition or decremented if a late left shift is needed in case of subtraction. Figure 6.9 illustrates the exponent datapath implementation to achieve reduced accumulation latency. An exponent difference unit takes as input E_{now}, LZA, and $E_{AC(next)}$. It computes: $abs(E_{now} + LZA - E_{AC(next)} + x)$, where $x = -1, 0, 1$. corresponding to the exponent difference if the last result is normalized to the left, not shifted or normalized to the right. A late select based on normalization of the mantissa is used to select the correct exponent difference for next stage.

The mantissa datapath is architected to start operation after the exponent difference is found, resulting in overlapping bypass loops of the exponent datapath and mantissa datapath, as shown in Figure 6.8. This late mantissa datapath design has several advantages. First, the exponent difference is done in parallel with the multiplication, removing the exponent difference stage from the critical path between the multiplier and adder; thereby shortening the total latency of CMA design and making it roughly the same as FMA one. Second, the critical path for an accumulation dependent instruction is improved from 4 cycles to 3 cycles without noticeably affecting the latency of independent instructions. Finally, since exponent difference is performed first, power optimizations such as fine-grained clock gating of the far/near path of the adder based on exponent difference can be introduced, although no such optimization was implemented in the presented power figures.

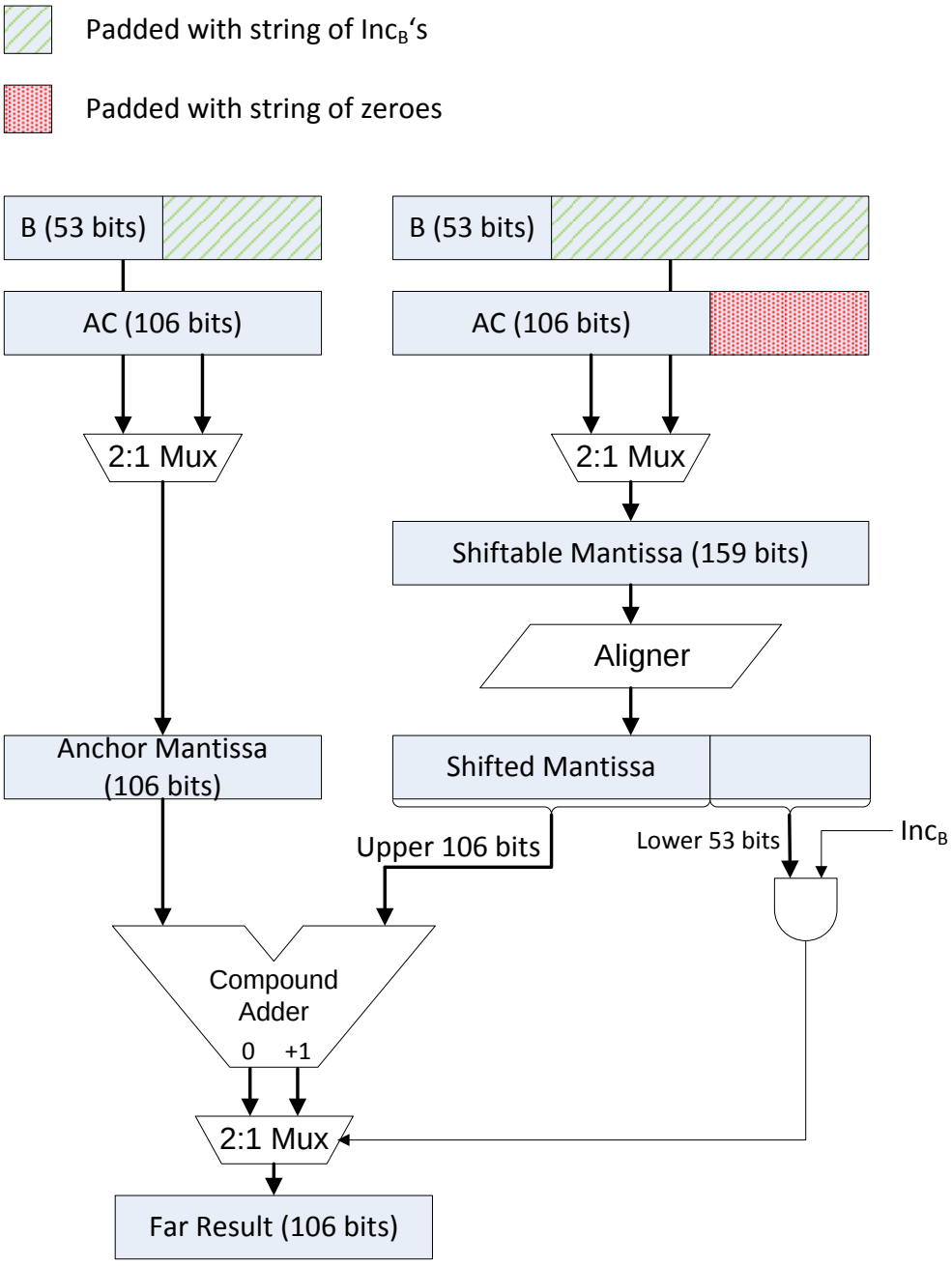


Figure 6.7: Far path addition implementation of mantissa datapath with support for incrementation signal $\mathrm{Inc_B}$

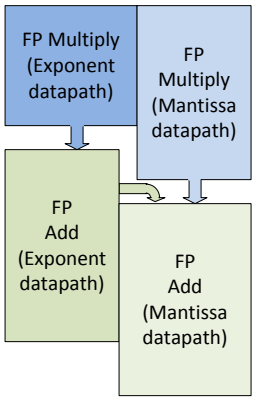


Figure 6.8: Block diagram of CMA mantissa and exponent datapaths showing the staggered timing of the exponent and mantissa

6.1.3 Cascade Multiply Add architecture with multiplier outputs in Carry Save format CMA2(4,6,7)

Several other design modifications to improve latency have been proposed. Parallel path designs that compute different datapaths in parallel and select the correct answer based on different cases have been proposed, but have large area overhead [32, 36]. Some FMA designs also aim to improve the accumulation latency as well. Intel demonstrated an 80-core throughput chip that employed an 11-stage multiply-accumulate unit with single cycle accumulation latency [40] [39]. Unfortunately, this design is not an IEEE FMA operation, because it does not preserve intermediate precision. A Bridge FMA design has been proposed to add FMA functionality by adding a bridge unit to slightly modified adder and multiplier designs [31]. The area of this bridge FMA unit is nearly as large as a separate FMA and adder units, which makes this approach less appealing. Energy efficient implementation of fused operation has been proposed employing shifting one multiplicand before multiplication based on the exponent difference with the addend. However the shifted out bits results in loss of precision for intermediate results. Therefore it is not an IEEE compliant implementation and is more suitable for signal processing applications that can tolerate such rounding errors [30, 29].

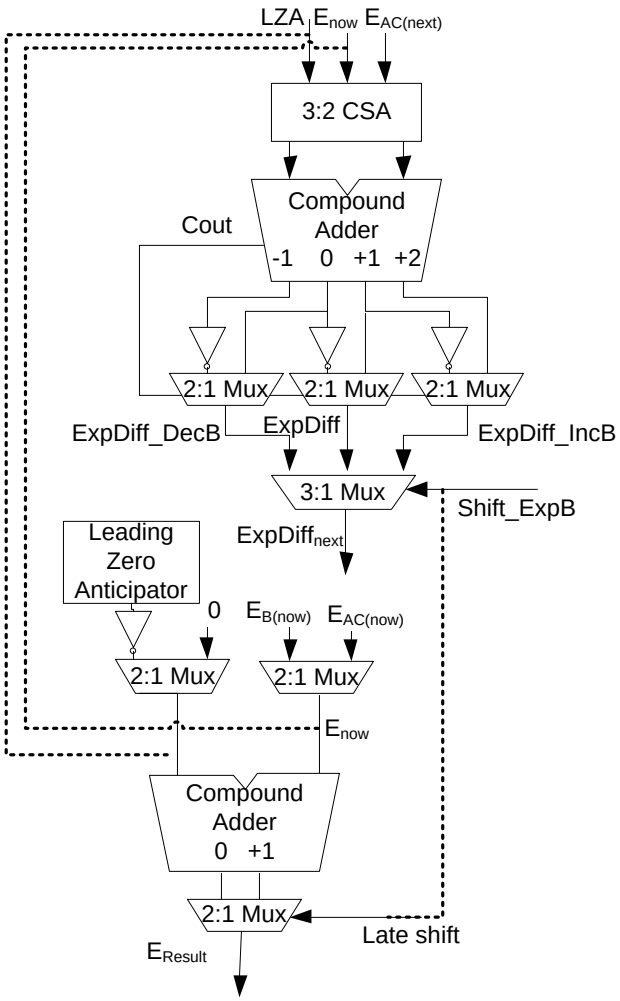


Figure 6.9: Simplified exponent datapath indicating the feedback loops. Since we don't know the output of the final normalization (ShiftExpB) we take the output of the current operation (E_{now}) and the output of the LZA and combine them with the next multiplier output ($E_{AC(next)}$) to compute the next exponent difference ($ExpDiff_{next}$). Since $E_{now} + LZA$ can be off by one, we need to compute both options, and we need to compute the absolute value of the result (the 2-1 mux driven by C_{out})

Another FMA design tries to improve the latency of additions by separating addition cases into two groups. One, where the exponents are far apart, does not require normalization, and the alignment is done after multiplication. The other, where the exponents are close, skips the shifter, which gives time for post addition normalization [13]. That design also keeps the multiplier output in carry save format to shorten the total latency. However this comes at the expense of added energy and the accumulation latency is degraded because of the extra carry save adder and wider datapath required. This design has roughly (4,6,7) latency. Figure 6.10 illustrates the datapath of the significand of the this design which we denote by CMA2 because it is conceptually very similar to cascade design with the only difference that multiplier outputs are kept in carry save format.

6.2 Application Study

The effect of the different instruction dependencies in FPU design and their respective latencies is application dependent, since for applications with parallelism, data dependencies can be hidden by interleaving execution of parallel (non dependent) work to keep the machine busy during the "stall" time. For example, on a 6 stage FPU, interleaving the execution of 6 threads will keep the unit busy and hide any data dependencies. This technique is used in GPU designs. For such parallel workloads, W/GFlops and mm^2/GFlops are the critical parameters to optimize as was shown in Chapter 3. For applications with less parallelism, the performance effect of these latency changes are important, and depend on the amount of parallelism that the processor can extract from the application: only when FP operations are on this critical path will the latency changes matter. We first studied a simple single-issue, in-order model to quickly explore the frequencies of the different dependency paths, and to gain intuition for the types of trade-offs that might exist. To provide this information, we modified the M5 architecture simulator [12] built for the PowerPC architecture to count the three different FP latency stalls. The modified simulator stored the total number of stalled cycles for every design and calculated the average

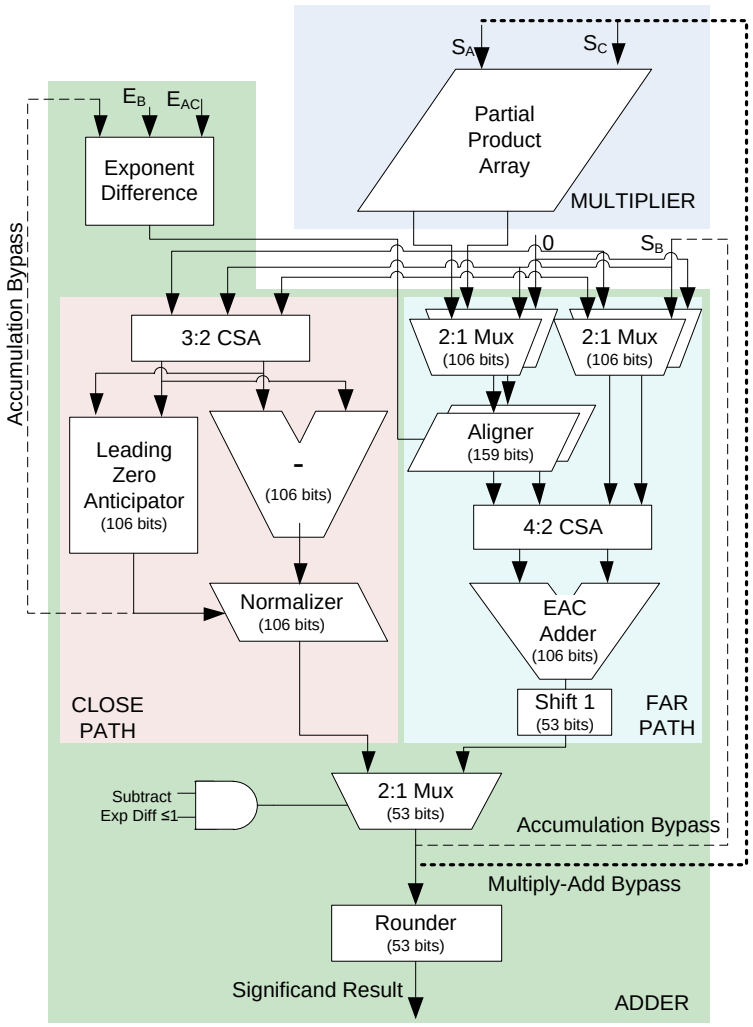
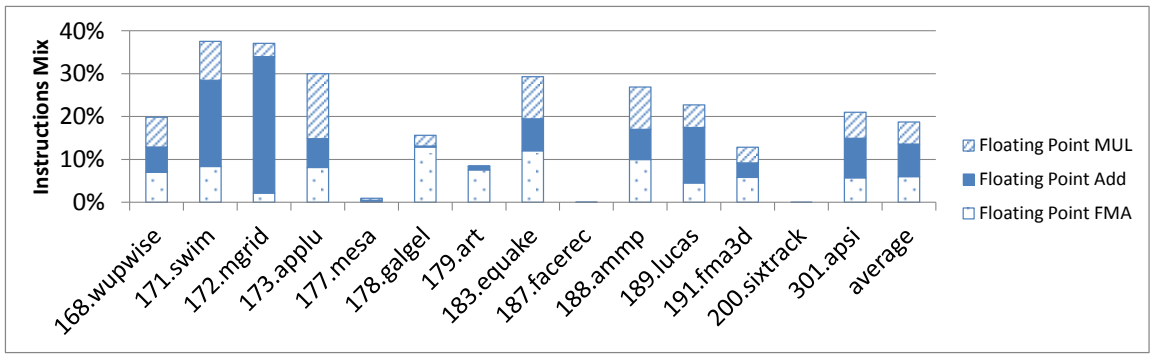


Figure 6.10: Simplified CMA2 significand datapath (multiplier; adder: far path, close path) with accumulation bypass path shown as dashed line and multiply-add bypass path shown as dotted line. Notice the duplicated Aligner and Mux units and extra CSA adders in the adder datapath

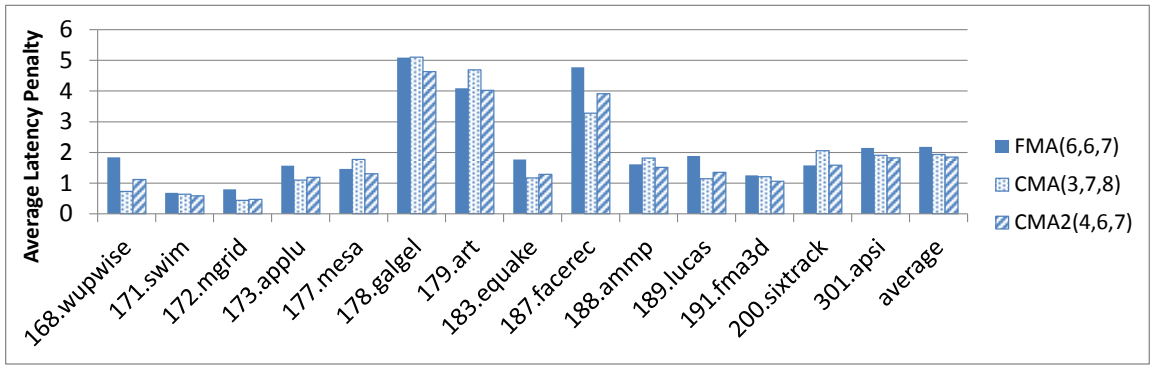
latency penalty by dividing by the total number of FMADD, FMUL and FADD instructions. Finally, we calculated the average time penalty by dividing the average latency penalty by the clock frequency.

This study revealed the importance of the accumulation latency, so we focused on creating a design which maximized the overall performance (at small power changes) using asymmetric latencies. In the end we compared FMA (6,6,7), CMA (3,7,8), and CMA2(4,6,7). We simulated the reference set of CFP2000 benchmarks using the gcc PowerPC cross compiler with the -O3 optimization directive. The PowerPC architecture was chosen because it has had the FMA instruction for a long time and has more mature FMA compiler support. The compiler optimizes for a 6 cycle FPU, which matches our base FMA architecture. Figure 6.2 shows the in-order model results. On average, FMADD, FMUL and FADD instructions make up around 20% of these application's instructions, but are much smaller in three (mesa, facerec, and sixtrack). We ignore these applications in the averages in Figure 6.2(b) and (c) since FP performance is not critical for them. Figure 6.2(b) shows the average latency penalty for each application. CMA(3,7,8) achieves an average latency penalty of 1.81 cycles across the benchmark which is 13% lower than the 2.07 average latency penalty incurred by the FMA(6,6,7) design. CMA2(4,6,7) achieves a slightly better average latency penalty of 1.73, but in this simple model, the change in the two latencies essentially balances out. Figure 6.2(c) shows the performance loss from FP stalls. FMA(6,6,7), CMA(3,7,8) and CMA2(4,6,7) incur total performance penalties of 41%, 33.7% and 33.1% respectively. Therefore, CMA(3,7,8) and CMA2(4,6,7) architectures will be 5-6% faster than an FMA(6,6,7) architecture at the same clock frequency, if the average instructions per cycle (IPC) of all non-FP instructions is one.

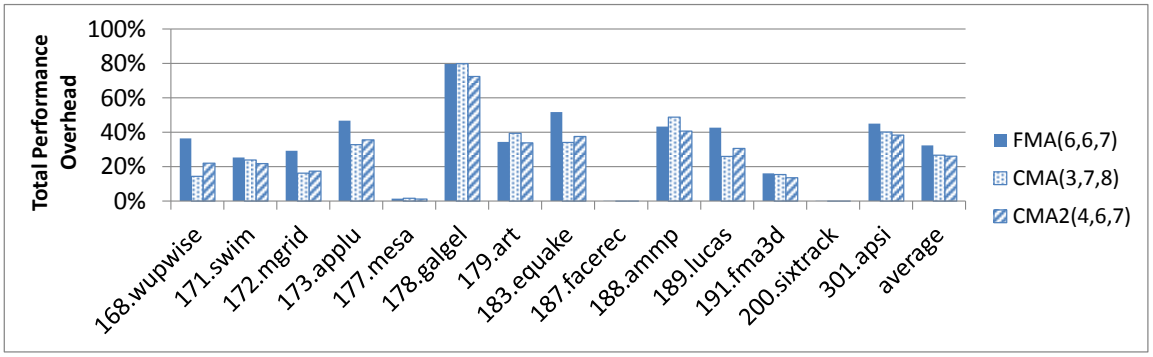
An in-order machine is very latency sensitive, as any subsequent dependent instructions stalls the pipeline execution until the floating point instruction has finished. Out of order superscalar designs are less latency sensitive because they exploit instruction level parallelism (ILP) to find non-dependent instructions to issue while waiting for executing instructions, resulting in higher IPC. However, long FPU latency still affects performance when the available ILP is not enough to keep the functional units busy, resulting in stalls. To test the effectiveness of the proposed cascade design in out



(a)



(b)



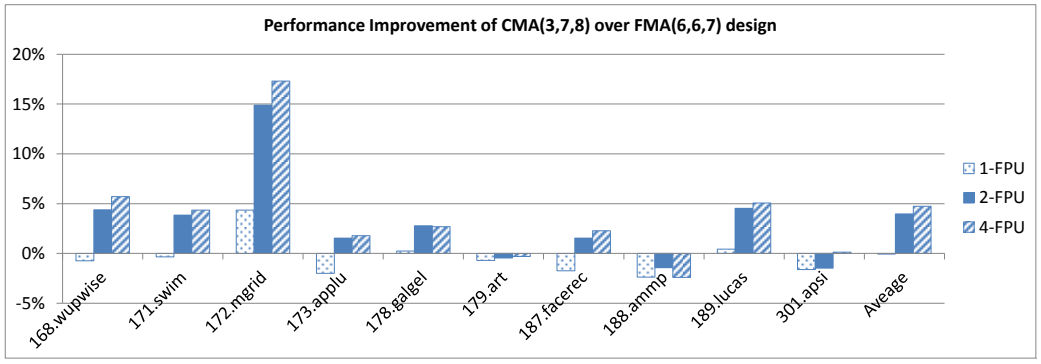
(c)

Figure 6.11: CFP 2000 benchmark on a simple single-issue in-order model. (a) Floating point instruction mix as percentage of total number of instructions. (b) Average latency penalty (c) Total performance overhead (assuming IPC=1 except for FP operations) for FMA(6,6,7), CMA(3,7,8) and CMA2(4,6,7) designs.

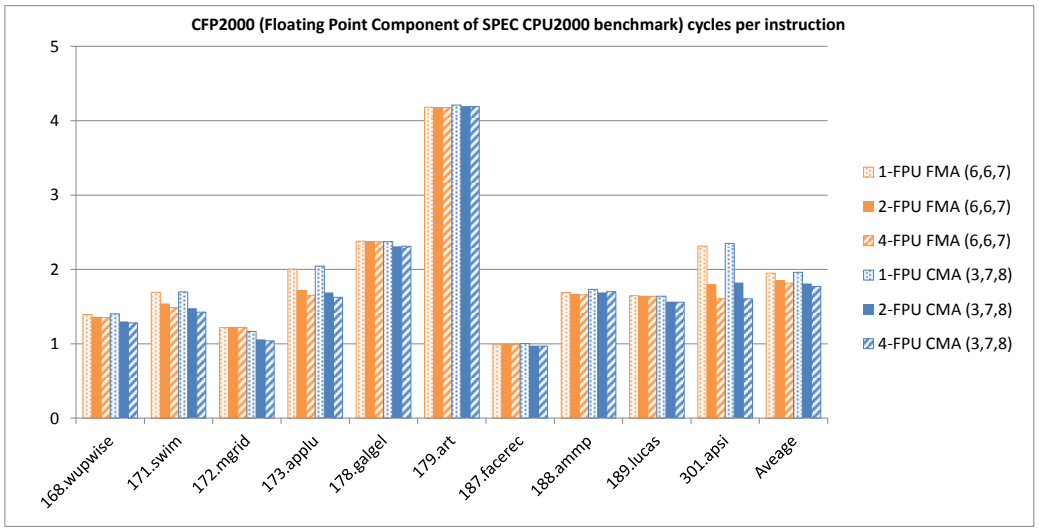
of order machines, we modified the scheduler of the out of order model of the M5 simulator to support the FMA(6,6,7) and CMA(3,7,8) architectures. For the CMA(3,7,8) design, the scheduler was modified to allow fadd and accumulation-dependent fmadd instructions to issue up to 5 cycles earlier if the critical operand was produced by preceding fmadd, fmul or fadd instructions and up to 3 cycles earlier if produced by other instructions. Additionally, dependent fmul and multiply-add dependent fmadd are issued up to one cycle earlier. On the other hand, for the FMA(6,6,7) scheduler, any accumulation dependent or multiply-add dependent FMADD, FMUL or FADD instructions are issued up to 1 cycle earlier. Using the modified model, the CFP 2000 benchmarks were run with 1-FPU, 2-FPU and 4-FPU configurations to see how the performance improvement scales with increased number of functional units, which should increase the sensitivity to FPU latency. The results of the floating point rich benchmarks are summarized in Table 6.1. The CMA design shows an average reduction in cycles per instruction (CPI) over FMA of 3.97% for the 2-FPU case and 4.62% for the 4-FPU machine as illustrated in Figure 6.12. As for the single FPU design case, the machine turns out to be not latency sensitive as the function unit turns out to be busy most of the time and is resource limited.

If the results are normalized by the percentage of floating point component in the benchmark, we find that we have on average 4% improvement in the 2-FPU and 4.6% for the 4-FPU because they are more latency sensitive. On the other hand, no improvement at all happens in the case of single FPU because performance there is limited by the number of FPUs rather than the latency of the FPU. For example in the 173.applu application, the busy rate of the FPU (the proportion of times an instruction was not issued because the FPU was already fed another instruction) in the single case was 58% while the rate was 16% and only 1.4% for 2-FPU and 4-FPU respectively.

In summary, the proposed CMA(3,7,8) design achieves an average performance improvement of 4-6% for a wide spectrum of designs that are latency sensitive ranging from simple in-order single issue designs to out of order superscalar designs. We proceed next to analyze the area and power cost of such design in comparison to traditional FMA design.



(a)



(b)

Figure 6.12: CPI Reduction in CFP 2000 Benchmarks for Out of Order Machines with 1,2 or 4 Floating Point Units

	Instruction Mix				CPI (Cycles Per Instructions)					
				Total	1-FPU		2-FPU		4-FPU	
Benchmark	fma	fadd	fmul	Floating	CMA	FMA	CMA	FMA	CMA	FMA
168.wupwise	9%	7%	10%	25%	1.404	1.393	1.301	1.358	1.282	1.355
171.swim	8%	19%	9%	36%	1.698	1.692	1.479	1.536	1.425	1.487
172.mgrid	3%	42%	3%	48%	1.166	1.217	1.061	1.219	1.04	1.22
173.applu	11%	9%	22%	41%	2.045	2.004	1.689	1.715	1.625	1.654
178.galgel	45%	3%	5%	53%	2.374	2.379	2.311	2.375	2.312	2.374
179.art	9%	2%	0%	11%	4.211	4.181	4.196	4.177	4.19	4.177
187.facerec	6%	10%	4%	19%	1.003	0.986	0.975	0.99	0.969	0.991
188.ammp	7%	5%	7%	19%	1.732	1.691	1.689	1.665	1.701	1.66
189.lucas	3%	13%	5%	22%	1.640	1.647	1.568	1.639	1.559	1.638
301.apsi	6%	15%	12%	33%	2.351	2.313	1.823	1.796	1.605	1.607
Aveage	11%	13%	8%	31%	1.96	1.95	1.81	1.85	1.77	1.82
					Weighted Average Performance Improvement					
					-0.07%		3.98%		4.74%	

Table 6.1: Out of order performance results for CFP2000 benchmark

	FMA	CMA	CMA2
Accumulation Latency (ns)	2.14	1.03	1.29
Multiply-Add Latency (ns)	2.14	2.4	2.28
Average Latency (ns)	2.14	1.715	1.785
Area (μm^2)	33149	36660	41429
Energy/op (pJ)	17.9	19.3	21.864

Table 6.2: Unpipelined Latencies for Different FMA Designs

6.3 Timing, Power and Area

An FMA, a CMA design and a CMA2 with multiplier outputs in carry save format have been implemented and verified using SystemVerilog and synthesized using TSMC 45nm technology libraries. To determine the relative latencies, unpipelined versions of the designs were synthesized. Table 6.2 summarizes the result. CMA has the least accumulation latency while FMA has the least multiply-add latency. These latencies were the basis for choosing the latency cycles we evaluated in our application study.

For comparing the delay and energy of the designs, the FMA design and the CMA2 were synthesized using a 7-stage pipeline while for the CMA design an 8-stage

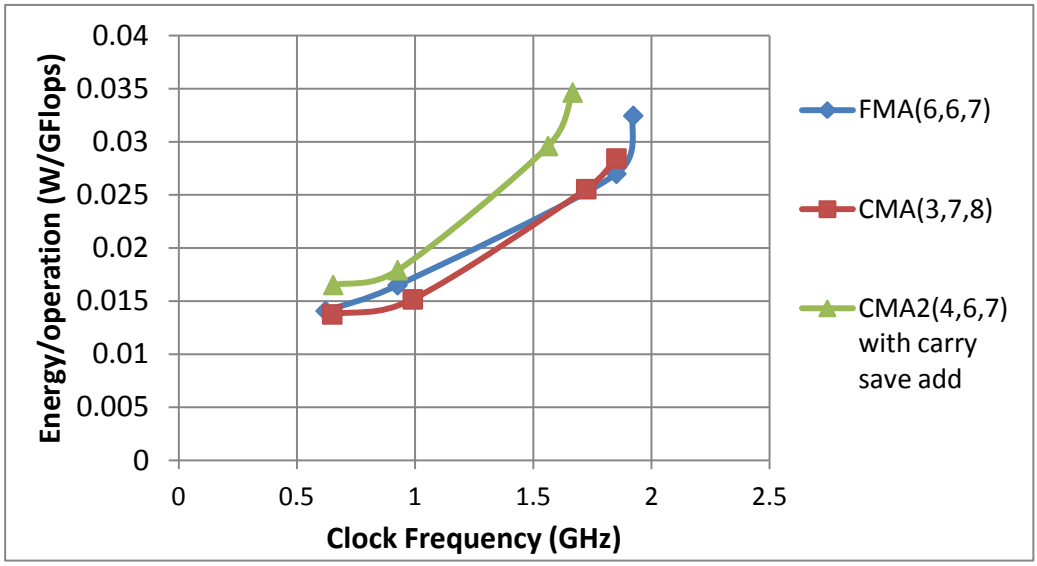


Figure 6.13: Energy efficiency tradeoff curves of different fused multiply-add architectures.

pipeline was synthesized. The datapath optimization flow starts by synthesizing a design for a certain timing constraint, inserting pipeline registers and doing register retiming to pipeline the design. Then the resulting design is placed and routed and the required clock network is generated. After the design is routed, the design is re-optimized and parasitics are extracted and annotated to the netlist. Activity factors for dynamic power calculations are calculated for random input vectors and assuming full utilization of the FPU. The timing and power of the design are then reported using Primetime timing tool. This procedure is repeated over a wide range of supply voltages, threshold voltages, and clock periods to choose the most energy efficient designs. After generating the data, the points on the efficient frontier of minimum energy/op designs for a certain performance targets are extracted from data points and are plotted in Figure 6.13. Table 6.3 provides the power, area and design parameters of these efficient frontiers. Examining the data, FMA(6,6,7) and CMA(3,7,8) have very similar energy and area cost, while CMA2(4,6,7) requires roughly 20% more energy and area.

V_{dd}	V_{th}	Freq-uency (GHz)	Area (μm^2)	Power (mW)		FO4 (ps)	Cycle Time (FO4)	W/ Gflops (FO4)	mm^2/ Gflops
				Dyn-amic	Lea-kage				
FMA(6,6,7)									
0.72	standard	0.62	47269	17.9	0.9	24	67	0.038	0.014
0.81	low	0.93	43651	30.5	2.3	17	64	0.024	.016
0.9	low	1.92	71089	204	7	14	37	0.018	0.032
CMA(3,7,8)									
0.72	standard	0.65	49571	17.4	0.9	24	64	0.038	0.014
0.8	low	0.99	44950	28	2.2	17	59	0.023	0.015
0.9	low	1.72	54578	96.7	4.7	14	41	0.016	0.026
0.	low	1.85	61133	134	6.1	14	39	0.017	0.028
CMA2(4,6,7) with multiplier outputs in Carry Save format									
0.72	standard	0.65	58990	20.9	1.1	24	64	0.045	0.016
0.81	low	0.93	52357	32.9	2.7	17	64	0.028	0.018
0.9	low	1.56	63530	110.2	6.4	14	46	0.02	0.03
0.9	low	1.67	81944	64.3	5	14	43	0.025	0.035

Table 6.3: Efficient Frontier Designs (Energy/Op vs. Frequency) for Different Double Precision FMA Architectures in 45nm TSMC technology

6.4 Summary

When optimizing an FMA design, it is critical to understand that the effective latency of the operation depends on which unit (multiplier or adder) will consume the output, and whether latency matters at all. For applications with abundant parallelism, the latency penalty will be zero and throughput oriented metrics such as W/GFlops and mm^2/GFlops should be the optimization target. For more latency sensitive applications, a cascade design provides a number of parameters that can be optimized, and in particular it allows one to create a design with very low effective latency between operations with a sum dependence. The reduction in latency depends on two main optimizations: forwarding of unrounded results and tightening the accumulation bypass path by staggering the exponent and mantissa datapath of the adder. Building and synthesizing the design reveals it does not incur area or energy overheads over existing FMA designs. Using an architectural simulator and SPEC2000 FP benchmark we found the CMA design to have 6% performance gain for a simple single issue

in-order designs and 4-4.5% gain for out of order superscalar designs with Multiple FPUs.

Chapter 7

Conclusion

With modern scaling where we get more gates but not with low enough energy, power has become the problem that needs to be addressed. Floating point operations are critical for many applications and are running into energy limits today. This dissertation explored FP units and created methods to create both throughput and latency optimized designs.

For throughput designs, the floating point application has a lot of parallelism that allows one to reduce energy and increase performance by duplicating many low energy cores; Latency then becomes just an optimization parameter instead of hard performance target. However, extra parallelism comes at a high area cost and marginal energy improvement. Therefore, a tradeoff between energy/op and computational density (ops/s/mm^2) ties all energy, area and throughput parameters. As such, power density becomes the critical design choice. For example high performance designs might use 1W/mm^2 design target while low power mobile GPU might use 0.1W/mm^2 design. Even better, the target power density might be calculated from a larger optimization to minimize the total cost of operation which balance system cost (mm^2) with energy costs. Since energy is scaling slower than area scaling, design must change to lower energy point to maintain constant power density scaling. Table 7.1 illustrates 2× energy savings from throughput optimal design over latency optimal one and another 2× benefits from optimized scaling from 90nm to 45nm. More trouble lies ahead for future scaling beyond 45nm as the throughput performance scales

	90nm Latency Optimal	90nm Throughput Optimal	90nm Throughput Optimized Design Scaled to 45nm	45nm Throughput Optimal
Clock Frequency	580 MHz	660 MHz	1.37 GHz	500MHz
Pipeline Depth	3	8	8	3
Area	0.078 mm^2	0.091 mm^2	0.0185 mm^2	0.012 mm^2
Latency	5.13 ns	12.08ns	5.84ns	6.06ns
mm^2/GFlops	0.067	0.069	0.0067	0.012
W/GFlops	0.031	0.014	0.0065	0.0033
Power Density	0.46	0.21	0.96	0.27

Table 7.1: Double Precision FMA Design Recap: throughput optimized 90nm design (third column) is more than 2× more energy efficient than latency optimized design (second column) for the same area and throughput. When this throughput optimal design is scaled down to 45nm (fourth colunmn) it is 2× less energy efficient than design reoptimized for 45nm process

linearly instead of cubically. Technologies for reducing the leakage energy component by using longer channel lengths or using transistors with better electrostatics such as finFETS are some of the proposed mitigations.

On the other hand, when applications don't have enough parallelism, latency becomes again the traditional performance bottleneck. Simple in-order designs for example are more latency sensitive than out of order designs as out of order designs can exploit instruction level parallelism. For FPU designs based on fused multiply add, one has to closely look at the different latencies of the different operation sequences to realize that optimizing for accumulation latency using our proposed cascade architecture can give total system performance increase of up to 6% in latency sensitive designs. The cascade design as such proves to be a viable alternative to traditional design.

Bibliography

[1] ATI Radeon™HD 5870 GPU Feature Summary.

[2] CFP2000 (floating point component of spec CPU2000).

[3] Hynix 1Gb (32Mx32) GDDR5 SGRAM H5GQ1H24AFR datasheet.

[4] Predictive technology models.

[5] IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*, 1985.

[6] The International Technology Roadmap for Semiconductors 2005 Edition, System Drivers, 2007.

[7] S. Sheng A. P. Chandrakasan and R. W. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(2):473–484, 1992.

[8] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers. The ibm system/360 model 91: Floating-point execution unit. *IBM Journal of Research and Development*, 11(1):34 –53, jan. 1967.

[9] Keith A. Bowman Azeez J. Bhavnagarwala, Blanca L. Austin and James D. Meindl. A minimum total power methodology for projecting limits on cmos gsi. *IEEE Transactions on VLSI*, 8(3):235–251, June 2000.

[10] P. Bai, C. Auth, S. Balakrishnan, M. Bost, R. Brain, V. Chikarmane, R. Heussner, M. Hussein, J. Hwang, D. Ingerly, R. James, J. Jeong, C. Kenyon, E. Lee, S.-H. Lee, N. Lindert, M. Liu, Z. Ma, T. Marieb, A. Murthy, R. Nagisetty,

S. Natarajan, J. Neirynck, A. Ott, C. Parker, J. Sebastian, R. Shaheed, S. Sivakumar, J. Steigerwald, S. Tyagi, C. Weber, B. Woolery, A. Yeoh, K. Zhang, and M. Bohr. A 65nm logic technology featuring 35nm gate lengths, enhanced channel strain, 8 cu interconnect layers, low-k ild and 0.57 mu;m2 sram cell. In *Electron Devices Meeting, 2004. IEDM Technical Digest. IEEE International*, pages 657 – 660, dec. 2004.

[11] Luiz André Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[12] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. The m5 simulator: Modeling networked systems. *Micro, IEEE*, 26(4):52 –60, july-aug. 2006.

[13] J.D. Bruguera and T. Lang. Floating-point fused multiply-add: reduced latency for floating-point addition. In *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, pages 42 – 51, june 2005.

[14] L. Chang, D.J. Frank, R.K. Montoye, S.J. Koester, B.L. Ji, P.W. Coteus, R.H. Dennard, and W. Haensch. Practical strategies for power-efficient computing technologies. *Proceedings of the IEEE*, 98(2):215 –236, feb. 2010.

[15] R. Chau. Benchmarking nanotechnology for high-performance and low-power logic transistor applications. In *Nanotechnology, 2004. 4th IEEE Conference on*, pages 3 – 6, aug. 2004.

[16] International Business Machines Corporation. *IBM 704 electronic data-processing machine: manual of operation*. International Business Machines Corp., 1955.

[17] Weidong Liu et al. *BSIM3v3.2.2 MOSFET Model Users' Manual*. University of California, Berkeley, 1999.

[18] P. M. Farmwald. *On the design of high performance digital arithmetic units.* PhD thesis, Stanford University, 1981.

[19] A. Hartstein and Thomas R. Puzak. Optimum power/performance pipeline depth. In *Proceedings of the 35th Annual International Symposium on Microarchitecture*, November 2003.

[20] E. Hokenek, R.K. Montoye, and P.W. Cook. Second-generation risc floating point with multiply-add fused. *Solid-State Circuits, IEEE Journal of*, 25(5):1207 –1213, oct 1990.

[21] Hwa-Joon Oh et al. A fully pipelined single-precision floating-point unit in the synergistic processor element of a CELL processor. *IEEE Journal of Solid-State Circuits*, 41:759–771, April 2006.

[22] H. Iwai. Si mosfet roadmap for 22nm and beyond. In *Computers and Devices for Communication, 2009. CODEC 2009. 4th International Conference on*, pages 1 –4, dec. 2009.

[23] K. Johguchi, Y. Mukuda, K. Aoyama, H.J. Mattausch, and T. Koide. A 2-stage-pipelined 16 port sram with 590gbps random access bandwidth and large noise margin. *IEICE Electronics Express*, 4(2):21–25, 2007.

[24] J. Kavalieros, B. Doyle, S. Datta, G. Dewey, M. Doczy, B. Jin, D. Lionberger, M. Metz, W. Rachmady, M. Radosavljevic, U. Shah, N. Zelick, and R. Chau. Tri-gate transistor architecture with high-k gate dielectrics, metal gates and strain engineering. In *VLSI Technology, 2006. Digest of Technical Papers. 2006 Symposium on*, pages 50 –51, 0-0 2006.

[25] J.E. Lindholm, M.Y. Siu, S.S. Moy, S. Liu, and J.R. Nickolls. Simulating multiported memories using lower port count memories, March 4 2008. US Patent 7,339,592.

[26] Dejan Markovic, Borivoje Nikolic, and Robert W. Brodersen. Power and area efficient vlsi architectures for communication signal processing. In *Proceedings*

of the IEEE International Conference on Communications, Vol. 7, pages 3323–3328, June 2006.

[27] Dejan Marković, Vladimir Stojanović, Borivoje Nikolić, Mark A. Horowitz, and Robert W. Brodersen. Methods for true energy-performance optimization. *IEEE Journal of Solid-State Circuits*, (8), Aug 2004.

[28] Chandrakant D. Patel and Amip J. Shah. Cost model for planning, development and operation of a datacenter. Technical Report HPL-2005-107, Internet Systems and Storage Laboratory,HP Laboratories, Palo Alto, June 2005.

[29] R.V.K. Pillai, D. Al-Khalili, and A.J. Al-Khalili. Low power architecture for floating point mac fusion. *Computers and Digital Techniques, IEE Proceedings* -, 147(4):288 –296, jul 2000.

[30] R.V.K. Pillai, S.Y.A. Shah, A.J. Al-Khalili, and D. Al-Khalili. Low power floating point mafs-a comparative study. In *Signal Processing and its Applications, Sixth International, Symposium on. 2001*, volume 1, pages 284 –287 vol.1, 2001.

[31] E. Quinnell, E.E. Swartzlander, and C. Lemonds. Bridge floating-point fused multiply-add design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(12):1727 –1731, dec. 2008.

[32] E. C. Quinnell. *Floating-Point Fused Multiply-Add Architectures*. PhD thesis, The University of Texas at Austin, 2007.

[33] H. N. Yu V. L. Rideout E. Bassous R. H. Dennard, F. H. Gaensslen and A. R. LeBlanc. Design of ion-implanted mosfets with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256Ű268, 1974.

[34] E.M. Schwarz, M. Schmookler, and S.D. Trong. Fpu implementations with denormalized numbers. *Computers, IEEE Transactions on*, 54(7):825 – 836, july 2005.

[35] Eric M. Schwarz. Binary floating-point unit design. In Vojin G. Oklobdzija and Ram K. Krishnamurthy, editors, *High-Performance Energy-Efficient Microprocessor Design*, Integrated Circuits and Systems, pages 189–208. Springer US, 2006.

[36] P.-M. Seidel. Multiple path ieee floating-point fused multiply-add. In *Circuits and Systems, 2003 IEEE 46th Midwest Symposium on*, volume 3, pages 1359 – 1362 Vol. 3, dec. 2003.

[37] T. Shyamkumar et al. Cacti: 5.0 an integrated cache timing, power, and area model. *HP Laboratories Palo Alto, Technical Report HPL-2007-167*, 2007.

[38] Son Dao Trong, M. Schmookler, E.M. Schwarz, and M. Kroener. P6 binary floating-point unit. In *Computer Arithmetic, 2007. ARITH '07. 18th IEEE Symposium on*, pages 77 –86, june 2007.

[39] S.R. Vangal, Y.V. Hoskote, N.Y. Borkar, and A. Alvandpour. A 6.2-gflops floating-point multiply-accumulator with conditional normalization. *Solid-State Circuits, IEEE Journal of*, 41(10):2314 –2323, oct. 2006.

[40] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *Solid-State Circuits, IEEE Journal of*, 43(1):29 –41, jan. 2008.

[41] L. Yue, J.W. Berendsen, K.M. Abdalla, R.M. Bastos, and R. Danilak. Architecture for compact multi-ported register file, February 10 2009. US Patent 7,490,208.

[42] Wei Zhao and Yu Cao. New generation of predictive technology model for sub-45nm early design exploration. *IEEE Transactions on Electron Devices*, 53(11):2816–2823, November 2006.

[43] Victor Zyuban and Philip Strenski. Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels. In *ISLPED*

'02: Proceedings of the 2002 international symposium on Low power electronics and design, pages 166–171, New York, NY, USA, 2002. ACM.