Energy-Efficient Floating-Point Unit Design

Sameh Galal, *Student Member*, *IEEE*, and Mark Horowitz, *Fellow*, *IEEE*

Abstract—Energy-efficient computation is critical if we are going to continue to scale performance in power-limited systems. For floating-point applications that have large amounts of data parallelism, one should optimize the $\mathrm{throughput/mm^2}$ 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. Looking at FP multiply-add units and ignoring register and memory overheads, we find that in a 90 nm CMOS technology at $1\ \mathrm{W/mm^2}$, one can achieve a performance of $27\ \mathrm{GFlops/mm^2}$ single precision, and $7.5\ \mathrm{GFlops/mm^2}$ double precision. Adding register file overheads reduces the throughput by less than 50 percent 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. A $1\ \mathrm{W/mm^2}$ design at 90 nm is a "high-energy" design, so scaling it to a lower energy design in 45 nm still yields a $7\times$ performance gain, while a more balanced $0.1\ \mathrm{W/mm^2}$ design only speeds up by $3.5\times$ when scaled to 45 nm. Performance scaling below 45 nm rapidly decreases, with a projected improvement of only $\sim 3\times$ for both power densities when scaling to a 22 nm technology.

Index Terms—Arithmetic and logic structures, high-speed arithmetic, floating point, fused multiply-add, throughput/$\mathrm{mm^2}$ optimization.

1 Introduction

Computer performance has been increasing exponentially in the last half century, driven by improvements in architecture, circuit design, and technology. While the power per function has been dropping exponentially, for the past 25 years it has been falling at a slower rate than the performance growth. The net result was a growth in power that computing systems consumed, and the associated issues with power distribution and heat removal. This decade, high-performance designs transitioned from being transistor/complexity-limited to being power-limited. Power constraints are only going to become more severe in the future, since Dennard scaling [1]—scaling of voltages with feature size—has ended. Historically, we scaled Vdd and Vth of our CMOS technology with feature size, which decreased the energy/op by the scaling factor cubed. Lowering Vth now exponentially increases leakage power, so it has stopped scaling. If Vth does not scale, scaling Vdd will make the gates slower, so its scaling has dramatically slowed. The net result is that the energy/op now only scales roughly proportional to the scaling factor, so the energy/op of the underlying design limits our peak performance.

Interestingly, the energy per operation depends on the 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 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 paper explores how to optimize FP functional units in this energy-constrained, parallel design space. We choose floating-point computation for a number of reasons. First, most FP applications can be parallelized, and a large number of these applications already run on parallel machines like GPUs. This gives us useful data to work with, including assurances that many applications have enough parallelism for the hardware to exploit. Second, since FP operation rates have been growing rapidly, especially on GPUs; we can use this analysis to estimate how we should expect their performance to scale in the future. Third, and finally, FP design has been well studied, so we have a wide variety of architectures, pipeline structures, and logical implementations we can explore.

To examine the design of energy-efficient FP units, we create an optimization framework that includes both circuit and system-level issues. 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 $\mathrm{results/sec/mm^2}$—remembering that very small, slower units might be the best solution. Thus, for parallel systems, the main trade-off is between energy/op and $\mathrm{ops/s/mm^2}$, so power density becomes a critical design constraint. As we will show later, for our 90 nm technology, tiling a chip with just FPUs (no register or memory) designed to be optimal for power density of $1\ \mathrm{W/mm^2}$ will yield 27 $\mathrm{GFlops/mm^2}$ single precision and 7.5 $\mathrm{GFlops/mm^2}$ double precision. However, the poor energy scaling of modern technologies means that to maintain this power density as we scale technology, we need to move the FP design to what was a lower power density design point. Thus, the improvements we get with scaling depend on the shape of the throughput/$\mathrm{mm^2}$ versus energy curve, and

• *The authors are with the Department of Electrical Engineering, 353 Serra Mall, Gates Building, Stanford University, Stanford, CA 94305-9030. E-mail: {sameh.galal, horowitz}@stanford.edu.*

Manuscript received 24 Aug. 2009; revised 3 Mar. 2010; accepted 10 May 2010; published online 4 June 2010.
Recommended for acceptance by E.M. Schwarz.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-08-0431.
Digital Object Identifier no. 10.1109/TC.2010.121.

decline as we are forced to move to lower and lower power density architecture and circuits.

The next section will briefly review FPU design, focusing on multiply-add units, since we use that structure for the examples in the paper. Section 3 then presents the design problems for throughput-based systems, and introduces the flow used to generate trade-off curves for floating-point multiply-add units which are used to solve these design problems. In Section 4, we look at the effect of technology scaling on floating-point performance. The effects of register and memory overhead are described in Section 5, and Section 6 concludes our paper.

2 Floating-Point Background

Floating-point (FP) numbers have their origins back in the earliest computers. They allow the machine to cover a wide range of results using a limited number of bits by representing each number by a mantissa, often between one and two, and an exponent. 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. 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 ($2^{\mathrm{minExponent}}$).

There has been an extensive research to improve the performance of floating-point calculations [3], [4], though most were done without considering energy efficiency. Even when energy was considered, the work focused on energy versus latency, and not throughput [2]. In contrast, this work focuses on optimizing floating-point throughput per mm^2 for applications that have abundant parallelism. While latency is not the most critical parameter, as we will explain in Section 5, it is still an issue because it affects minimum memory/registers requirements of the system.

Traditionally, FPU designs have used separate floating-point adders and multipliers. However, recent designs incorporate combined floating-point multiply-add instructions that implement the $\mathrm{A} + \mathrm{B} \times \mathrm{C}$ operation; these units offer better accuracy and improved performance. We present the two most common multiply-add implementations since we used both 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-accumulate (FPMAC) 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 precomputed 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, and those without this support.

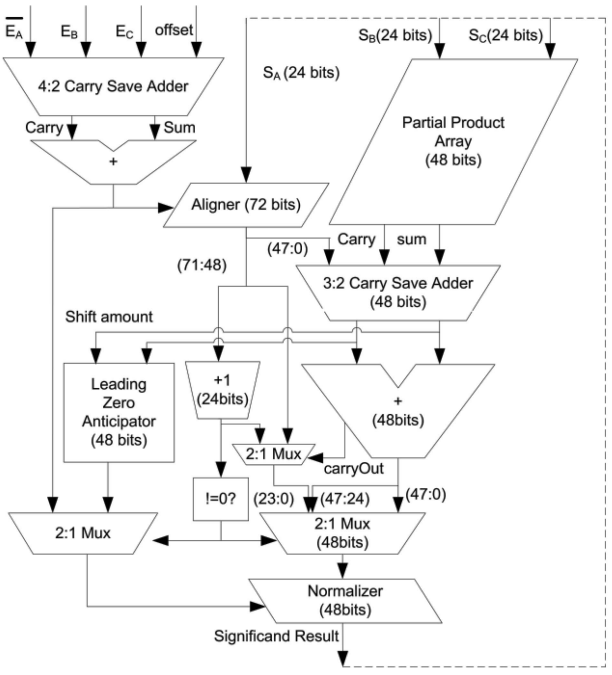

Fig. 1. Block diagram for a single-precision fused multiply-add unit. A is shifted to align it to the result of the multiply.

2.1 Fused Multiply-Add

Since its introduction in IBM's RS/6000 FPU in 1990 [6], the FMA unit has become a common implementation in recent FP multiply-add designs [7], [8]. 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 $\mathrm{FLOPs/mm}^2$ or FLOPs/W.

This design achieves its short latency by aligning the addend significand ($\mathrm{S_A}$) in parallel with multiplication of $\mathrm{S_B}$ and $\mathrm{S_C}$. This removes the conventional 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. Fig. 1 shows the data flow of traditional FMA, with the dashed lines showing the forwarding paths. For more detailed information on FMA design and implementation issues, please see the paper by Schwarz [11].

2.2 Cascade Multiply-Add

Some recent designs still prefer a cascaded design of an FP multiplier followed by an FP adder over the FMA design,

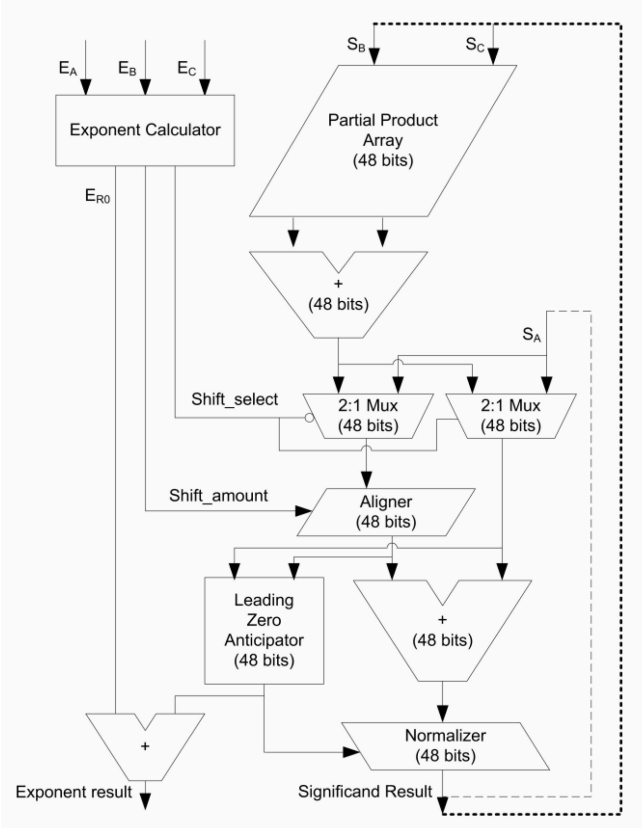

Fig. 2. Block diagram for a single-precision cascade multiply-add. In this design, the multiply is performed first, and then the smaller of the accumulator or the product is shifted and added to generate the result.

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 bit for the aligner, adder, and normalizer. Fig. 2 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 an FMA design. This is another argument in favor of CMA for these types of calculations.

3 Optimizing Throughput-Based Designs

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/sec, 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. We can characterize a throughput system by its hard constraints. Some systems have hard resource constraints, while others have hard performance constraints. As we will see, they both reduce to similar problems.

- **Resource-constrained throughput systems**: These systems are trying to maximize throughput given 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:

$$\text{maximize } T$$
$$\text{subject to:}$$
$$P/A < D_{max}$$
$$P < P_{max}$$
$$A < A_{max}$$

(where P is total power in W, A is total area in mm², T is total throughput in GFlops, D_{max} is maximum power density in W/mm²)

- **Performance-constrained throughput systems**: These have hard performance constraints, and generally use many individual processing units to achieve their throughput requirement. In this type of application, both system energy and total "chips" area are flexible, so one can meet the throughput performance target. Minimizing the total cost of ownership (TCO) is the optimization goal of such a system [17]. This cost includes both capital expenses and operational expenses and is a function of the area and energy per op of the system.

$$\text{minimize } \varphi(\varepsilon_A, \varepsilon_P)$$
$$\text{subject to:}$$
$$\varepsilon_P < D_{max}\,\varepsilon_A$$

(where $\varphi(\varepsilon_P, \varepsilon_A)$ 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 mm²/GFlops, D_{max} is maximum power density in W/mm²)

The key insight to find the optimal solution to these systems is to realize that power efficiency and area efficiency are usually conflicting goals. The easiest example is to consider what happens when you lower supply voltage: gates go slower, so the area/ops/s will increase, but the energy/op decreases, since it is proportional to supply voltage squared. Other knobs have similar effects; e.g., deeply pipelining a function unit usually decreases area/ops/s, but increases the energy/op since each result must flow through more flops. By utilizing circuit parameters such as sizing, supply voltage, and threshold voltage as well as microarchitectural parameters such as pipeline depth, we can generate many designs and then explore the trade-off between the power efficiency (power/throughput ε_P) and area efficiency (area/throughput ε_A) of the overall system. The next section describes the specific method we used to generate these trade-offs.

3.1 Optimization Flow

Since both metrics we are studying, energy/op and $\mathrm{ops/s/mm^2}$, are dependent on circuit and architecture parameters, we consider both issues by constructing

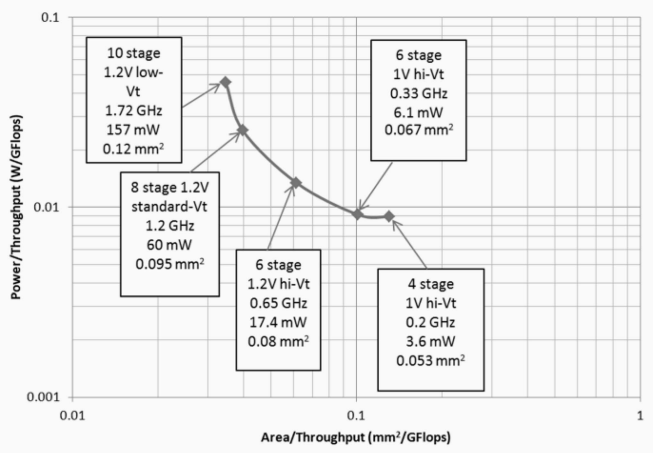

Fig. 3. Power/throughput versus area/throughput trade-off for 90 nm single-precision floating-point fused multiply-add unit using supply voltage, threshold voltage, circuit sizing, and pipeline depth as optimizing parameters. Energy per Flop, ranges from 10 to 100 pJ, and the silicon area needed to achieve 1 GFlops ranges from 0.035 to $0.15\ \mathrm{mm}^2$.

different designs on circuit level 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 the throughput optimal designs. 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 trade off throughput against energy/op without affecting area which leads to a straightforward trade-off between energy/op and area/throughput.
- **Pipelining**: adding a pipeline stage (without circuits resizing) leads to an interplay of several effects on performance, energy, and area, as it:
  - Increases throughput (i.e., decreases cycle time).
  - Increases area by pipeline stage overhead. For an efficient design, the relative increase in area should be less than relative increase in throughput, otherwise duplicating the design would achieve the required throughput with less total area. This puts an upper limit on the number of pipeline stages to put in the design.
  - "Increases" energy/op: while dynamic energy/op always increases, leakage energy/op is both increased by pipelining overhead and decreased by amortizing the leakage power over multiple operations in the pipeline. This explains why even the lowest energy designs still use three or four pipeline stages: the pipelined design turns out to be more energy-efficient than an unpipelined design which will be dominated by leakage power.

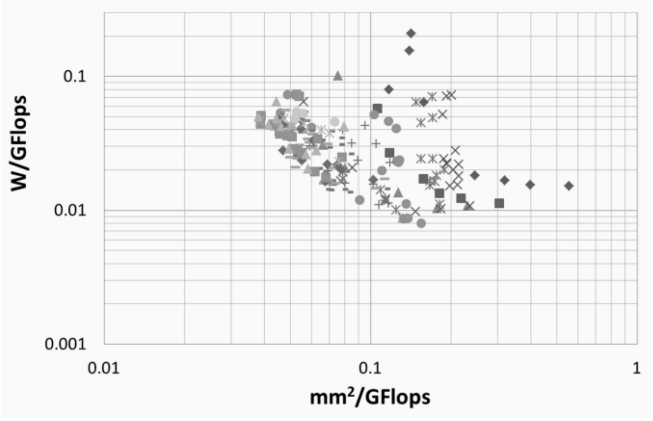

Fig. 4. Result of design space exploration of different supply voltages, threshold voltages, cycle time, and pipelining stages for 90 nm single-precision fused multiply-add unit.

- **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 $\mathrm{ops/s/mm}^2$.

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 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, clock periods, and pipeline depths. After generating the data, the points on the efficient frontier are extracted from data points shown in Fig. 4 to generate trade-offs as shown in Fig. 3.

As intuitively expected, deeply pipelined high-voltage high-frequency designs maximize computational density ($\mathrm{ops/s/mm}^2$), while shallow-pipelined low-frequency low-voltage designs maximize energy efficiency (ops/s/W). Designs which 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 1-1.2 V and 45 nm libraries with 0.8-1 V operating points. We have experimented with a larger voltage range as well, but found that it is only helpful for extreme power densities that are not practical for most applications; therefore, we think that these voltage ranges satisfy most of the desired power density ranges.

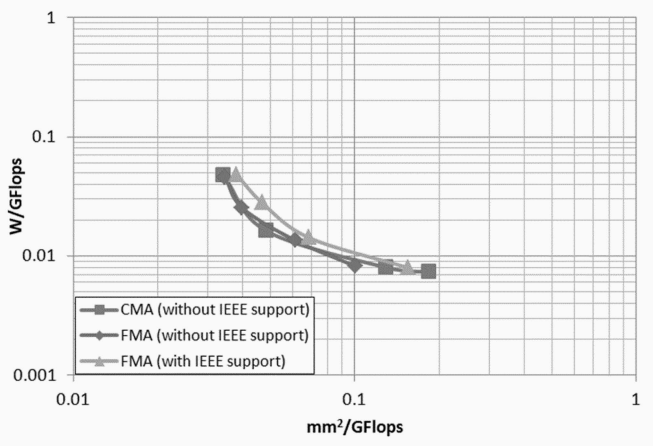

Fig. 5. Comparison between two different FP single-precision multiply-add designs, fused and cascade multiply-add. As the graph shows there is little difference between the two different designs. Also shown is an IEEE compliant unit as well. The overhead is small.

3.2 Exploring Multiply-Add Architectures

We started exploring the multiply-add unit by trying to determine the most efficient architecture considering the two most common approaches that were described in Section 2: the fused multiply-add, and the cascade multiply-add. 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 trade-offs as shown in Fig. 5.

While both designs achieve the same performance metrics in terms of W/GFlops and $\mathrm{mm}^2/\mathrm{GFlops}$, the cascade design has longer latency. For example, 3.2 GFlops throughput can be achieved by both designs at $0.036\ \mathrm{mm}^2/\mathrm{GFlops}$ 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. Since lower latency is a desirable and a "free" feature of fused design, we present the rest of study based on fused design. Fig. 5 also shows the cost of the IEEE compliance. An IEEE compliant implementation has an overhead of 5-10 percent over the range of different power densities.

Examining the effect of precision on performance, we found that double precision required approximately $3\times$ more resources than single precision as illustrated later in the paper in Fig. 9; the area and power of the multiplier trees grow quadratically in the size of the operands (a $4\times$ increase) while the rest of the datapath grows linearly (a $2\times$ increase). This results in the multiplier share of area and power growing from 31 percent in single-precision design to 45 percent in the double-precision design.

Using the trade-off data, we generated for FMA designs between the power efficiency, ε_P, in W/GFlops, and the area efficiency, ε_A, in $\mathrm{mm}^2/\mathrm{GFlops}$, we can proceed to find maximum throughput solution to the resource-constrained systems and minimum cost solution for the performance-constrained systems we posed earlier in this section.

3.3 Resource-Constrained System

We can easily find the optimal maximum throughput that conforms to area, power, and power density constraints by

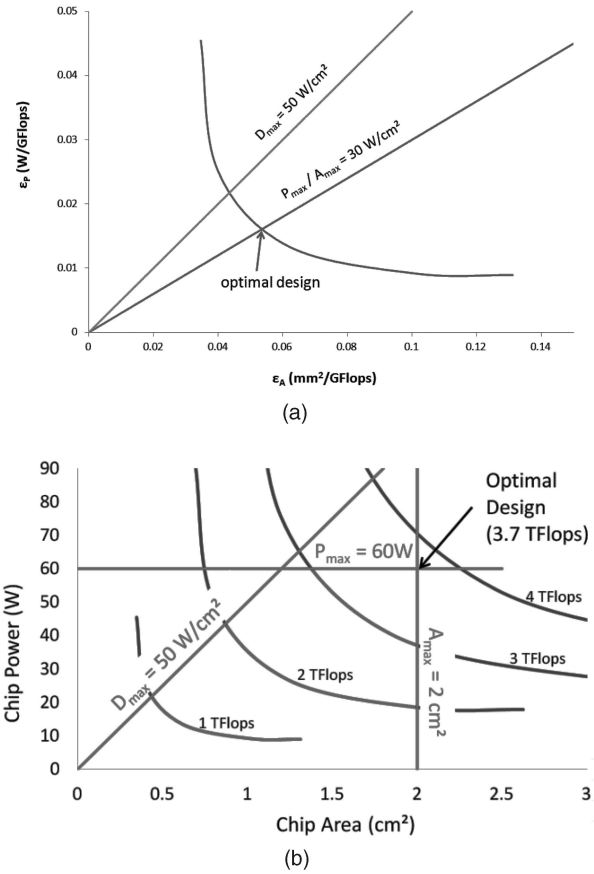

Fig. 6. (a) Determining optimal design point from throughput-energy trade-off curve and constraints, (b) Contour map of achievable throughputs versus area and power. Constraints of $\mathrm{A_{max}} = 2\ \mathrm{cm}^2$, $\mathrm{P_{max}} = 60\ \mathrm{W}$, and $\mathrm{D_{max}} = 50\ \mathrm{W/cm}^2$ are indicated.

substituting $\mathrm{P} = \varepsilon_\mathrm{P}\ \mathrm{T}$ and $\mathrm{A} = \varepsilon_\mathrm{A}\ \mathrm{T}$. The solution to this problem is the point$(\varepsilon_\mathrm{A}, \varepsilon_\mathrm{P})$ such that

$$\varepsilon_\mathrm{P} = \min(\mathrm{P_{max}}/\mathrm{A_{max}}, \mathrm{D_{max}})\varepsilon_\mathrm{A}$$

with a maximum throughput achieved $\mathrm{T} = \min(\mathrm{A_{max}}, \mathrm{P_{max}}/\mathrm{D_{max}})/\varepsilon_\mathrm{A}$. Fig. 6a illustrates graphically how to find the optimal design using an existing $\varepsilon_\mathrm{P} - \varepsilon_\mathrm{A}$ trade-off curve for an example constraints of $\mathrm{A_{max}} = 2\ \mathrm{cm}^2$, $P_{\max} = 60\ \mathrm{W}$, and $\mathrm{D_{max}} = 50\ \mathrm{W/cm}^2$. The intersection of the $\mathrm{P_{max}}/\mathrm{A_{max}}$ constant power density line with the tradeoff curve is the optimal design since $\mathrm{P_{max}}/\mathrm{A_{max}}$ in this example is a tighter constraint than $\mathrm{D_{max}}$. The optimal FPU design is a 1.67 GFlops design with an area of $0.09\ \mathrm{mm}^2$ and power of 27 mW.[1] Integrating ~2,222 such FPUs on a chip we achieve a total throughput of 3.7 TFlops at 60 W and $2\ \mathrm{cm}^2$.

Fig. 6b views the data in a slightly different way. First, we take the $\varepsilon_\mathrm{P} - \varepsilon_\mathrm{A}$ 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.

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

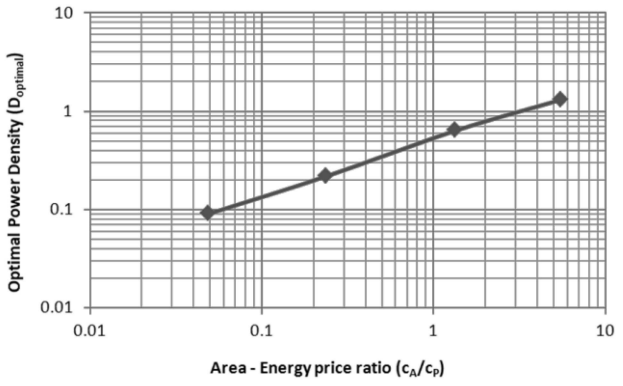

Fig. 7. Optimal Power density of system depends on the relative cost of area and energy in a performance-constrained system. This relationship was generated for the 90 nm data shown in Fig. 3 for the linear cost function $\varphi(\varepsilon_A, \varepsilon_P) = c_A\varepsilon_A + c_P\varepsilon_P$. Note that high-power densities are only cost-effective when the area to power costs ratio is large.

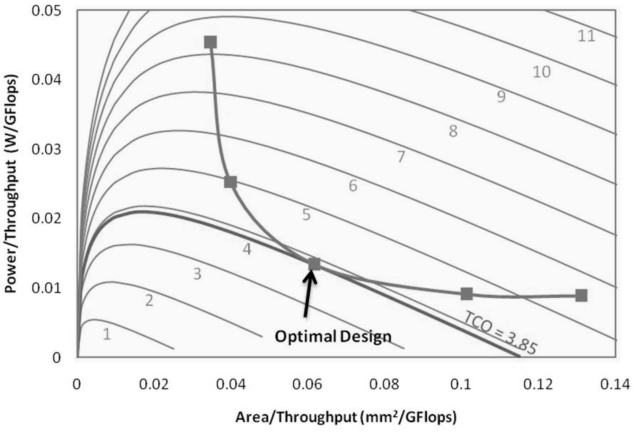

Fig. 8. Power/throughput versus area/throughput trade-off overlaid on example constant TCO contours in ¢/GFlops per year. The minimum cost design achieves TCO of ¢/GFlops per year. The cooling costs of the system are proportional to power density which accounts for the nonlinearity of the constant cost contours.

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 using Fig. 6a.

3.4 Performance-Constrained System

In these systems, we are trying to minimize the cost of the resulting system. Clearly, 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 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 minimization problem. Equation (1) shows an example cost function that incorporates different possible cost components: a power cost as function of energy efficiency, a hardware cost as function of area efficiency, and a cooling cost as function of power efficiency and power density.

$$\varphi(\varepsilon_A, \varepsilon_P) = \varphi_{hardware}(\varepsilon_A) + \varphi_{power}(\varepsilon_P) + \varphi_{cooling}(\varepsilon_P, \varepsilon_A). \quad (1)$$

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 $\$/\mathrm{mm}^2$ and $\$/\mathrm{W}$ 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. 8.

If we know the slope of the constant cost lines (i.e., when the ratio of marginal energy and area costs are relatively constant), we can convert this 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. 7 shows the result for the trade-off curve shown in Fig. 3. As intuitively expected, higher energy prices (low $\mathrm{c_A}/\mathrm{c_P}$ ratio) results in low-energy designs with low performance/mm^2 and low-power density, while high hardware cost (high $\mathrm{c_A}/\mathrm{c_P}$ ratio) results in high-energy high-performance/mm^2 designs with higher power density and less energy efficiency.

4 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 first using 90 nm technology with the results using a 45 nm library, and the detailed results of the efficient frontier data points are shown in Table 1 and Fig. 9. We assume that the system constraints remain the same and therefore we see how the designs change for a fixed power density. The combination of shrinking area/functional unit and constant power density means that each functional unit must dissipate less energy. Since the energy consumed by logic gates does not naturally scale fast enough, 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, $1\ \mathrm{W}/\mathrm{mm}^2$ efficient designs achieve $7\times$ 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 $0.1\ \mathrm{W}/\mathrm{mm}^2$ improve only $3.5\times$, since they reside on a flatter part of the trade-off curve. Unfortunately, in 45 nm even the $1\ \mathrm{W}/\mathrm{mm}^2$ designs are on a less steep part of the curve, indicating that further technology scaling will yield smaller performance gains.

We can also use the energy throughput curves to estimate how performance will continue to scale with

TABLE 1
Summary of Scaling Results for FMA Unit

Pipeline Depth	V_{th}	V_{dd}	Frequency (GHz)	Area (μm²)		Power (mW)		FO4 (ps)	Cycle Time (FO4)	Latency (FO4)	W/GFlops	mm²/GFlops	Power Density (W/mm²)
				Combinational	Total	Leakage	Dynamic						
90 nm Single Precision FMA													
12	low	1.08	1.54	66303	117325	80.1	72.2	24	27	323	0.0477	0.0381	1.25
10	standard	1.08	1.2	68578	113288	13.3	55.5	27	31	312	0.0277	0.047	0.59
8	high	1.08	0.66	53207	91206	1.5	7.4	35	43	345	0.0142	0.0689	0.21
6	high	0.9	0.2	44063	62925	0.54	2.63	42	116	699	0.0079	0.1554	0.05
90 nm Double Precision FMA													
14	low	1.08	1.43	219294	344275	290	200	24	29	406	0.1615	0.1205	1.34
11	standard	1.08	1.03	211645	311874	37	169	27	36	401	0.0855	0.1513	0.57
9	standard	1.08	0.75	175247	253832	35.7	58.4	27	50	450	0.0617	0.1688	0.37
7	high	1.08	0.39	159132	213689	4	19.2	35	73	509	0.0291	0.2725	0.11
7	high	0.9	0.28	147218	195243	3.6	10	42	83	582	0.0239	0.263	0.09
45 nm Single Precision FMA													
6	low	0.9	2.08	11258	16077	1.2	30.9	14	34	204	0.0072	0.0039	1.88
5	low	0.81	1.32	9945	14241	0.55	12.85	17	45	225	0.005	0.0054	0.93
4	low	0.81	0.98	9715	12670	0.58	8.09	17	60	241	0.0043	0.0065	0.67
3	standard	0.72	0.5	9415	12117	0.16	3.16	24	85	256	0.0033	0.0122	0.27
3	high	0.72	0.2	7735	10619	0.0358	0.952	40	122	367	0.0025	0.0261	0.09
45 nm Double Precision FMA													
6	low	0.9	1.81	38444	49839	4.6	95.9	14	39	235	0.0253	0.0145	1.75
6	low	0.81	0.95	30964	42019	1.8	29.2	17	62	372	0.0155	0.0221	0.7
4	standard	0.72	0.33	28252	35058	0.4	5.6	24	128	514	0.009	0.0533	0.17
4	high	0.72	0.2	29610	36747	0.13	3.23	40	124	496	0.0084	0.0914	0.09

technology. The key is to realize that the shape of the trade-off curve remains roughly the same with scaling, so the curves only shift as a result of technology scaling. Moving from 45 to 22 nm should decrease the capacitance by $2\times$, and thus should decrease the energy/op by $2\times$ as well. Since the devices get smaller, scaling should increase the $\mathrm{Flops/mm^2}$ by $4\times$—four units will fit in the same area as one unit today. If gate speed remains constant, then scaling a 45 nm solution to 22 nm will increase performance by $4\times$, but will also increase power density by $2\times$, since the energy of each operation scaled down by only $2\times$. This means if we want to end up at $1\ \mathrm{W/mm^2}$, we need to look at $0.5\ \mathrm{W/mm^2}$ in 45 nm. These solutions have about 1.5 times the area of the $1\ \mathrm{W/mm^2}$ solutions today, so a 22 nm $1\ \mathrm{W/mm^2}$ solution will only be 4/1.5 or 2.7 times faster than a $1\ \mathrm{W/mm^2}$ solution today.

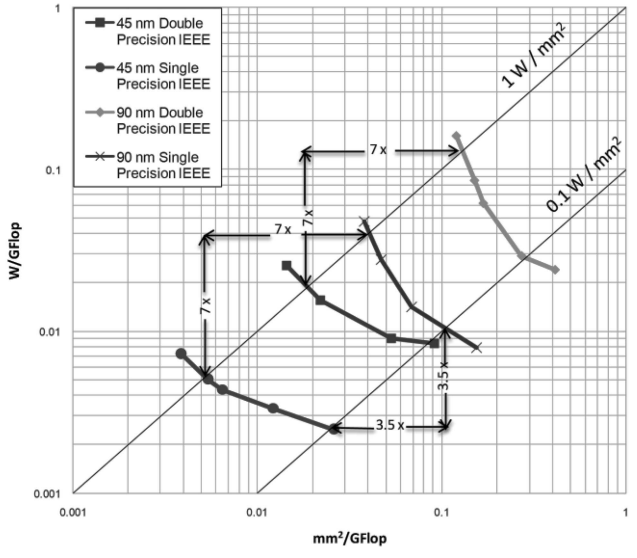

Fig. 9. Scaling of FMA single and double-precision designs from 90 to 45 nm. The performance gain depends on the power density allowed.

If gate speed scales, then moving to a 22 nm solution should increase performance by $8\times$ ($4\times$ in density and $2\times$ in frequency), but will increase power density by $4\times$. Again, if we are targeting $1\ \mathrm{W/mm^2}$, this means we need to start at $0.25\ \mathrm{W/mm^2}$ in 45 nm. These solutions have about a $2\times$ area overhead, so will yield a $4\times$ overall performance improvement moving to 22 nm. Notice that the trade-off curves get pretty flat below around $0.25\ \mathrm{W/mm^2}$, so further scaling of performance will yield even smaller gains in the future.

To help answer the question of how gates speed will scale, we built a technology-independent Matlab model for the FMA using the previous 90 and 45 nm results, and combined it with predictive transistor models [18] to estimate future scaling gains. This tool allowed us to estimate trade-off curves down to a 16 nm generation, and include new high-K technologies and technologies designed for HP and LP. Fig. 10 illustrates the estimated scaling data, giving the best curve for each technology node. As expected, in the past, classic Dennard scaling gave us $\sim 8\times$ improvement when scaling from 180 to 90 nm. Scaling from 90 to 45 nm, we still achieve around $7\times$ only by moving to lower energy designs. Going forward, a $1\ \mathrm{W/mm^2}$ design scales by only $3\times$ from 45 to 22 nm. Furthermore, for the power density range between 0.1 and

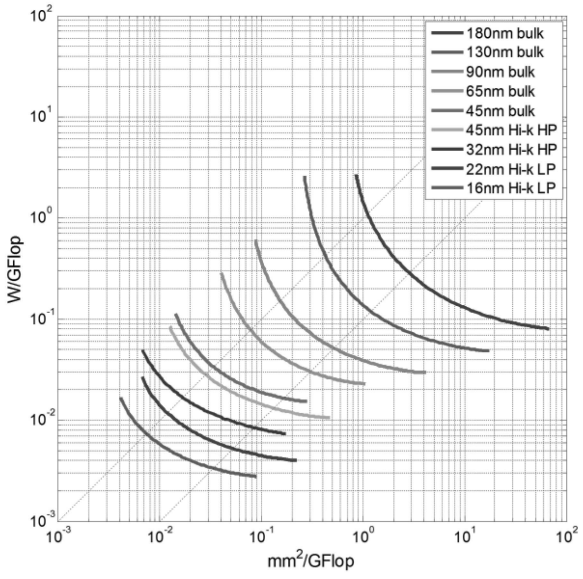

Fig. 10. Estimated double-precision FMA trade-off curves for different technology nodes from 180 nm bulk CMOS down to 16 nm high-K node.

$1\ \mathrm{W/mm^2}$, below 32 nm, low-power technologies employing higher effective oxide thickness (EOT) and longer effective transistor length (Leff) become the optimal technologies to use because of the need to continue to move to effectively lower energy density circuits to keep the actual energy density constant.

5 Storage Overhead

To consider register file overhead effects on FMA performance, we assume 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 six stage datapath has to interleave at least six threads to keep the FPU busy all the time. Therefore, the minimum register file size is proportional to pipeline depth, and the number of registers sufficient for applications that 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 ported 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 [12], [13], [14]. 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 CACTI, a timing, power and area cache and memory model developed by HP Labs [15]. Using memory designs generated by CACTI, we augment our datapath data to generate trade-offs that include register file accesses as well.

TABLE 2
Design Parameters for the Efficient Frontier of 45 nm Double-Precision FMA with Register File

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

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 single instruction multiple data (SIMD), 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 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 $\mathrm{ops/s/mm^2}$.

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 a minimum of 16 registers required per thread, the 45 nm single and double-precision FMA with latencies of three to six cycles is well served using 512 and 1,024 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 are most efficient. Table 2 illustrates the parameters of throughput-efficient designs for the double-precision FMA. These designs are different from the efficient designs

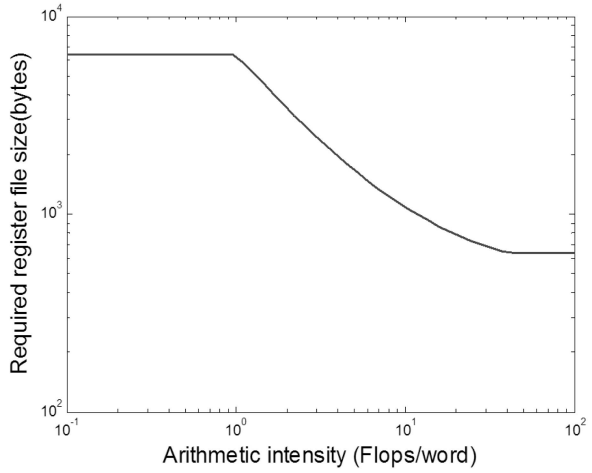

Fig. 11. Example register file size versus arithmetic intensity relationship for a double-precision five stage 2 GFlops system with 100 ns of memory latency.

identified when studying the FMA alone. Registers add an energy and area overhead of around 25 percent for single precision and 20 percent for double precision in 45 nm design, as illustrated in Fig. 12. The overheads are larger for 90 nm 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 FPMAC changes with the arithmetic intensity of the application, using a very simple single-level memory model. Assuming that the arithmetic intensity to represent the average number of FP operations between memory fetches, Fig. 11 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 [19]. Given our estimates of around 25 pJ/Flop in 45 nm, it means one needs a ratio of over 40 Flops/double word load for the memory not to dominate the overall system. 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 [20], but is likely to be an issue for future systems.

6 Conclusion

For throughput applications with abundant parallelism, the correct trade-off to consider is energy/op, measured in W/GFlops, and computational density, measured in $\mathrm{GFlops/mm^2}$. Computing this trade-off curve provides information about asymptotic performance that FP units can achieve, and how this performance is likely to scale. Our results show that current 45 nm double precision units can achieve around 40 $\mathrm{GFlops/mm^2}$, at a $\mathrm{1W/mm^2}$ power density. While this performance will continue to scale, it is unlikely to improve by more than $4\times$ at the 22 nm node, and is likely to only increase by $3\times$ at constant power density. While these give impressive numbers, the 25 pJ computing energy can be easily dwarfed by the nJ required for DRAM access. Thus, unless the application has very high arithmetic intensity, or new low-power memory technology is created, floating-point systems performance will be limited by memory power.

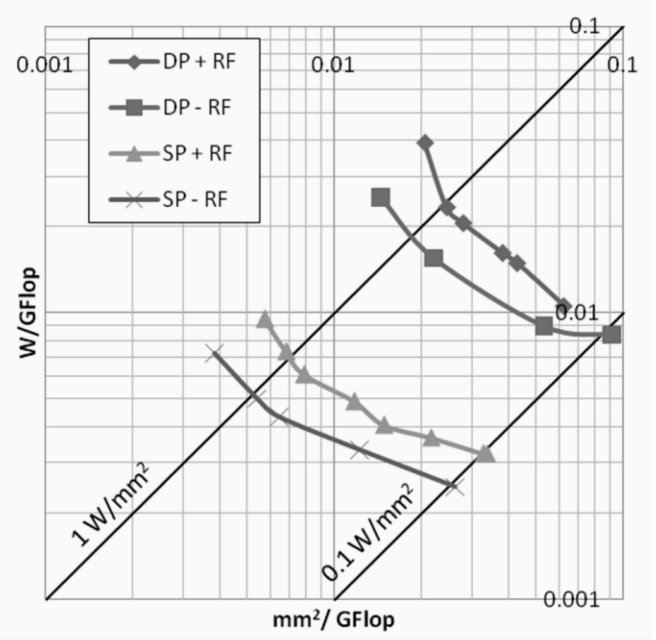

Fig. 12. Energy throughput trade-offs for 45 nm single and double-precision FMA units showing the overhead of the required register file on performance. For constant power density, the required area overhead is under 30 percent.

Acknowledgments

This work was supported by a FCRP grant through the C2S2 center.

References

[1] R.H. Dennard, F.H. Gaensslen, L. Kuhn, and H.N. Yu, "Design of Micron MOS Switching Devices," *Proc. IEEE Int'l Electron Devices Meeting*, pp. 168-170, 1972.

[2] D. Patil, O. Azizi, and M. Horowitz, "Robust Energy-Efficient Adder Topologies," *Proc. 18th IEEE Symp. Computer Arithmetic (ARITH '07)*, pp. 16-28, 2007.

[3] P.-M. Seidel and G. Even, "Delay-Optimized Implementation of IEEE Floating-Point Addition," *IEEE Trans. Computers*, pp. 97-113, vol. 53, no. 2, Feb. 2004.

[4] T. Lang and J.D. Bruguera, "Floating-Point Fused Multiply-Add: Reduced Latency for Floating-Point Addition," *Proc. 17th IEEE Symp. Computer Arithmetic (ARITH '05)*, pp. 42-51, 2005.

[5] P.M. Seidel, "Multiple Path IEEE Floating-Point Fused Multiply-Add," *Proc. 46th Int'l IEEE Mid-West Symp. Circuits and Systems (MWSCAS)*, 2003.

[6] E. Hokenek, R.K. Montoye, and P.W. Cook, "Second-Generation RISC Floating Point with Multiply-Add Fused," *IEEE J. Solid-State Circuits*, vol. 25, no. 5, pp. 1207-1213, Oct. 1990.

[7] H.-J. Oh et al., "A Fully Pipelined Single-Precision Floating-Point Unit in the Synergistic Processor Element of a CELL Processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 4, pp. 759-771, Apr. 2006.

[8] S. Dao Trong, M.S. Schmookler, E.M. Schwarz, and M. Kroener, "P6 Binary Floating-Point Unit," *Proc. 18th IEEE Symp. Computer Arithmetic (ARITH '07)*, pp. 77-86, 2007.

[9] N. Ide et al., "2.44-GFLOPS 300-MHz Floating-Point Vector-Processing Unit for High-Performance 3D Computer Graphics Computing," *IEEE J. Solid-State Circuits*, vol. 35, no. 7, pp. 1025-1033, July 2000.

[10] D.R. Lutz and C.N. Hinds, "A New Floating-Point Architecture for Wireless 3D Graphics," *Proc. 38th Asilomar Conf. Signals, Systems and Computers (ACSSC '04)*, vol. 2, pp. 1879-1883, Nov. 2004.
[11] E.M. Schwarz, "Binary Floating-Point Unit Design: The Fused Multiply-Add Dataflow," *High-Performance Energy-Efficient Microprocessor*, V.G. Oklobdzija and R.K. Krishnamurthy, eds., Springer, 2006.
[12] K. Johguchi, Y. Mukuda, K. Aoyama, H.J. Mattausch, and T. Koide, "A 2-Stage-Pipelined 16 Port SRAM with 590 Gbps Random Access Bandwidth and Large Noise Margin," *IEICE Electronics Express*, vol. 4, no. 2, pp. 21-25, 2007.
[13] J.E. Lindholm, M.Y. Siu, S.S. Moy, S. Liu, and J.R. Nickolls, "Simulating Multiported Memories Using Lower Port Count Memories," US Patent US 7,339,592 B2, Nvidia Corporation, Mar. 2008.
[14] L. Yue, J.W. Berendsen, K.M. Abdalla, R.M. Bastos, and R. Danilak, "Architecture for Compact Multi-Ported Register File," US Patent US 7,339,592 B2, Nvidia Corporation, Mar. 2008.
[15] S. Thoziyoor, N. Muralimanohar, and N.P. Jouppi, "CACTI 5.0: An Integrated Cache Timing, Power, and AreaModel," technical report, HP Laboratories, 2007.
[16] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips, "GPU Computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879-899, May 2008.
[17] C. Patel et al., "Cost Model for Planning, Development and Operation of a Datacenter," http://www.hpl.hp.com/techreports/2005/HPL-2005-107R1.pdf, 2009.
[18] Predictive Transistor Models, http://ptm.asu.edu/, 2010.
[19] Hynix 1 Gb (32Mx32) GDDR5 SGRAM H5GQ1H24AFR Datasheet, http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf, 2010.
[20] ATI Radeon HD 5870 GPU Feature Summary, http://www.amd.com, 2010.
[21] R.W. Brodersen, M.A. Horowitz, D. Markovic, B. Nikolic, and V. Stojanovic, "Methods for True Power Minimization," *Proc. IEEE/ACM Int'l Conf. Computer Aided Design (ICCAD)*, pp.35-42, Nov. 2002,

Sameh Galal received the BS degree in electronics engineering and computer science from the American University in Cairo in 2005 and the MS degree in electrical engineering from Stanford University in 2008. He worked as a research engineer in Nokia research center and is currently a PhD student at Stanford University. His research interests include circuit design, optimization, computer architecture, computer arithmetic, and embedded systems. He is a student member of the IEEE.

Mark Horowitz received the BS and MS degrees in electrical engineering from MIT in 1978, and the PhD degree from Stanford in 1984. He is the chair of the Electrical Engineering Department and the Yahoo Founder's Professor of the School of Engineering at Stanford University. In addition, he is a chief scientist at Rambus Inc. He has received many awards including the 1985 Presidential Young Investigator Award, the 1993 ISSCC Best Paper Award, the ISCA 2004 Most Influential Paper of 1989, and the 2006 Don Pederson IEEE Technical Field Award. He is a fellow of the IEEE and the ACM and is a member of the National Academy of Engineering and the American Academy of Arts and Science. His research interests are quite broad and span using EE and CS analysis methods to problems in molecular biology to creating new design methodologies for analog and digital VLSI circuits. He has worked on many processor designs, from early RISC chips to creating some of the first distributed shared memory multiprocessors, and is currently working on on-chip multiprocessor designs. Recently, he has worked on a number of problems in computational photography. In 1990, he took leave from Stanford to help start Rambus Inc, a company designing high-bandwidth memory interface technology, and has continued work in high-speed I/O at Stanford. His current research includes multiprocessor design, low-power circuits, high-speed links, computational photography, and applying engineering to biology.