Power Dissipation Challenges in Multicore Floating-Point Units

Wei Liu and Alberto Nannarelli

Dept. of Informatics & Mathematical Modelling
Technical University of Denmark
Kongens Lyngby, Denmark

***Abstract*—With increased densities on chips and the growing popularity of multicore processors and general-purpose graphics processing units (GPGPUs) power dissipation and energy consumption pose a serious challenge in the design of system-on-chips (SoCs) and a rise in costs for heat removal. In this work, we analyze the impact of power dissipation in floating-point (FP) units and we consider different alternatives in the implementation of FP-division that lead to substantial energy savings. We compare the implementation of division in a Fused Multiply-Add (FMA) unit based on the Newton-Raphson approximation algorithm to the implementation in a dedicated digit-recurrence unit. The results show a significant reduction of energy in a typical scientific application when the division digit-recurrence unit is used. In addition, we model the thermal behavior of the considered FP-units.**

***Index Terms*—floating-point; fused multiply-add; division; low power; thermal analysis.**

I. Introduction

Technology scaling allowed a dramatic increase of chip densities and a consequent clock frequency increase especially from the mid-nineties and for about ten years. The typical CPU clock rate rose from about 300 MHz in 1996 to 3 GHz in 2003. The drawback of this trend was the increased dynamic power dissipation to levels that would soon become unmanageable. Increased power dissipation results in excessive heat dissipation with a rise in cooling costs (heat sinks, fans, etc.) and the risk of compromising chips (electromigration).

To have high computing power by keeping the power dissipation to an acceptable level, computer evolved from single high-rate clock processor to parallel multi-processor (multicore) systems, on the same chip, with clock rates in the range 1 to 3 GHz. In addition to general purpose CPUs, in recent years, Graphic Processing Units (GPUs) for General Purpose computing (GPGPUs) have emerged as a cheap and viable solution for massive parallel computing.

Although the clock rates have reached a plateau, the power consumption of state-of-the-art multicores and GPGPUs is still quite high and poses problems on cooling and power budgets especially for large data centers [1].

Moreover, transistor's leakage power, due to sub-threshold currents, has become significant in today's nanometric technologies. This leakage power increases with the chip temperature. For this reason, thermal management of large Systems on Chip (SoCs) has become an important design aspect.

In this work, we analyze the impact on power dissipation of the Floating-Point (FP) units implemented in multicores and GPUs and propose some alternatives to reduce the power dissipated and to mitigate the thermal profile of large SoCs.

We model the main FP operations (on binary64[1] operands) by using VHDL descriptions of units similar to those implemented in state-of-the-art processors, and extract post-synthesis power dissipation and thermal profiles of those FP-units.

The unit considered for FP addition and multiplication is a Fused Multiply-Add (FMA) unit which is a popular choice in multicores [3] and GPUs [4]. For FP division we considered the multiplicative approach implemented by the FMA unit, and the digit-recurrence approach with a unit similar to that of [5].

In addition, we propose a low power solution for division based on an architecture derived from [6] and [7].

The FP-units presented are not intended to be the best in performance or power consumption, but to provide realistic models to evaluate the latency, area and power trade-offs of FP-units populating large SoCs.

II. Fused Multiply-Add Unit

A Fused Multiply-Add (FMA) unit implements the floating-point operation

$$A + B \times C \ .$$

The main advantages of the fused implementation over the separate implementation of multiplication and addition are:

- The high occurrence of expressions of that type in scientific computation, and the consequent reduction in overhead to adjust the operands from the IEEE format to the machine internal representation (de-normalization, etc.).
- Improvement in precision, as the result of multiplication is added in full precision and the rounding is performed on $A + B \times C$.

The drawback is that if a large percentage of multiply and add cannot be fused, the overhead in delay and power is large especially for addition.

[1]The double-precision format is now called binary64 in the IEEE 754-2008 standard [2].

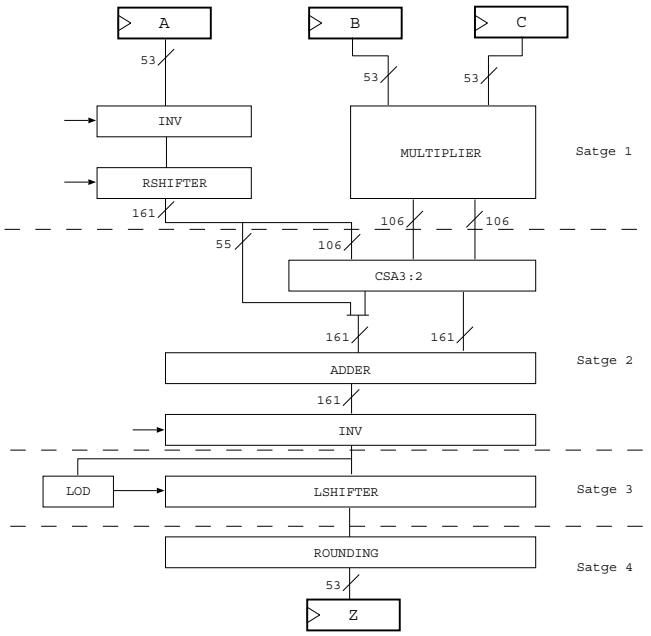

Fig. 1. Scheme of a FMA unit

The architecture of the FMA unit, shown in Fig. 1, is derived from [8] and [9]. In the figures, we omit the processing of exponents and signs to simplify the drawings.

The operation performed is $Z = A + B \times C$. In order to prevent shifting both A and the product of $B \times C$, A is initially positioned two bits to the left of the most significant bit (MSB) of $B \times C$ so that only a right shift is needed to align A and the product $B \times C$. The zero bits inserted in the two least-significant (LS) positions are used as the guard and round bits when the result significand is A. The amount of shift depends on the difference between the exponents of A and $B \times C$. Moreover, A is conditionally inverted when the effective operation is subtraction.

A Booth encoded tree multiplier computes the product of $B \times C$ and the result is in carry-save format to be added with the shifted A.

Since the product has 106 bits, only the 106 LSBs of the shifted A are needed in the carry-save adder (CSA). The 55 MSBs of the shifted A are concatenated with the sum of the CSA to form the input to the adder. Since the carry output of the CSA has 106 bits, only one of the inputs to the adder has 161 bits.

Consequently, the leftmost 55 bit portion of the adder is implemented as an incrementer with the carry-out of the lower part as the increment input. The adder also performs end-around-carry adjustment for effective subtraction. As the result might be negative, an array of inverters is required at the output of the adder.

Once the result of the addition is obtained, the amount of the normalization shift is determined by the leading one detector (LOD). No right shift for normalization is required due to the initial position of A.

To increase the throughput, the FMA unit is implemented in a four-stage pipeline. The position of the pipeline registers is indicated with dashed horizontal lines in Fig. 1.

III. Division by Multiplication

The quotient q of the division

$$q = \frac{x}{d} + rem$$

can be computed by multiplication of the reciprocal of d and the dividend x

$$q = \frac{1}{d} \cdot x$$

This is implemented by the approximation of the reciprocal $R = 1/d$, followed by the multiplication $q = R \cdot x$.

By determining $R[0]$ as the first approximation of $1/d$, R can be approximated in m steps by the Newton-Raphson (NR) approximation [8]

$$R[j+1] = R[j](2 - R[j]d) \qquad j = 0, 1, \ldots, m$$

Each iteration requires two multiplications and one subtraction. The convergence is quadratic and the number of iterations m needed depends on the initial approximation $R[0]$ (implemented by a look-up table in our case).

Once $R[m]$ has been computed, the quotient is obtained by an additional multiplication $Q = R[m] \cdot x$.

To have rounding compliant with IEEE standard, extra iterations are required to compute the remainder and perform the rounding according to the specified mode [8]:

- $rem = Qd - x$
- $q = ROUND(Q, rem, mode)$.

The NR algorithm for binary64 division ($m = 2$) is summarized below.

```
R[0] = LUT(d);
FOR i := 0 TO 2 LOOP
    W = 2 - d * R[i];
    R[i+1] = R[i] * W;
END LOOP;
Q = x * R[3];
rem = x - d * Q;
q = ROUND(Q,rem,mode);
```

Although division by iterative multiplication is expensive in power, it has been chosen to implement division in AMD processors [10], NVIDIA GPUs [4], and in Intel Itanium CPUs [11].

The original FMA unit needs to be augmented with a look-up table and several mutiplexers and registers in order to achieve the initial approximation and implement the division algorithm. The implementation of the multiplicative method based on FMA unit is shown in Fig. 2. The FMA components not used for division are shown in dashed lines. Control signals to these disabled units are carefully designed so that they dissipate static power only.

A look-up table is generated using the midpoint reciprocal method [12], of which the entries are the reciprocals of

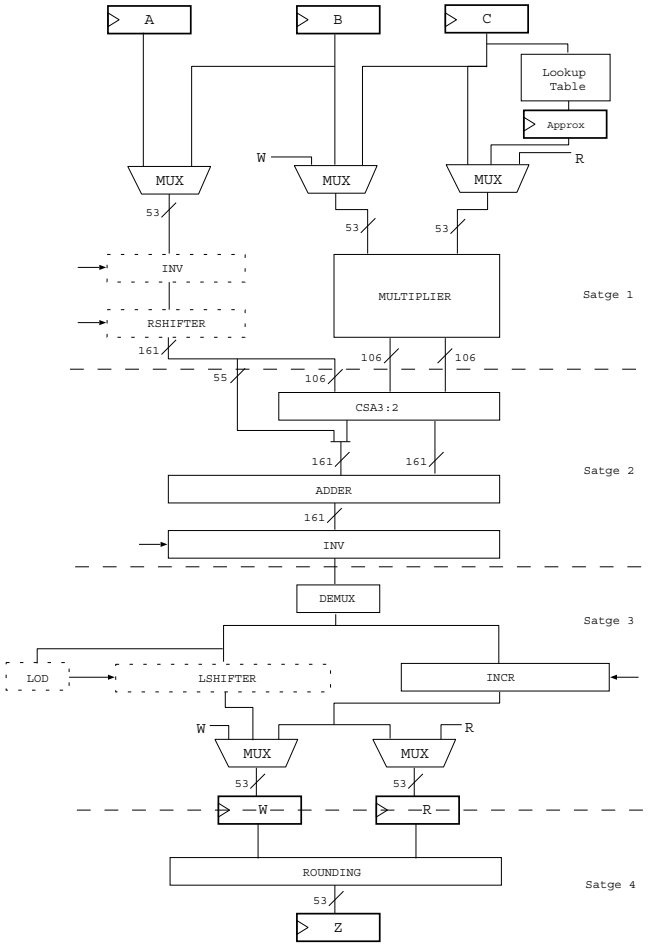

Fig. 2. Scheme of the modified FMA unit to support division.

midpoints of the input intervals. The dividend x is stored in register B and divisor d in register C.

The first cycle is to obtain the initial 8-bit approximation $R[0]$. After that, the operations performed in the 4-stage pipelined unit of Fig. 2 are the following (Stage 1 is abbreviated S1, etc.):

- S1 The initial approximation $R[0]$ is multiplied by d using the tree multiplier.
- S2 The product is complemented to obtain $2 - R[0]d$. This is achieved by using the array of inverters in this stage and the incrementer in next stage.
- S3 The $2 - R[0]d$ operation is completed in the incrementer. The demux directs data to the incrementer only so that the switching activity in the LOD and LSHIFTER of Fig. 2 are minimized. The result is stored in register W ($W[1] \leftarrow (2 - R[0]d)$).
- S1 $W[1]$ is multiplied by $R[0]$.
- S2 The multiplication $W[1]R[0]$ is continued.
- S3 The new approximation $R[1] \leftarrow W[1]R[0]$ is stored in register R. The new approximated reciprocal has a precision of 16 bits.

The above 6 steps have to be repeated two more times to have $R[3]$ with the precision necessary for binary64 division.

Once the correct approximation of $1/d$ has been computed, another two iterations in the multiplier are required to compute:

1) the non-rounded quotient: $Q = x \cdot R[3]$;
2) the remainder: $rem = Q \cdot d - x$ necessary for IEEE compliant rounding.

Finally, Q is rounded according to the remainder and the specified rounding mode

$$q = ROUND(Q, rem, mode) \ .$$

Summarizing, the number of clock cycles required for the implementation of the division algorithm with the unit of Fig. 2 is 26 as detailed in Table I.

	cycles
initial approx. $R[0]$	1
three NR iterations	$3 \times 6 = 18$
non-rounded quotient $Q = x \cdot R[3]$	3
remainder $rem = Q \cdot d - x$	3
rounding	1
Total cycles	26

TABLE I
CYCLES FOR BINARY64 DIVISION IN FMA UNIT.

IV. DIVISION BY DIGIT-RECURRENCE

The digit-recurrence algorithm [13] is a direct method to compute the quotient of the division

$$q = \frac{x}{d} + rem$$

The radix-r digit-recurrence division algorithm for binary64 (double-precision) significands is implemented by the residual recurrence

$$w[j+1] = rw[j] - q_{j+1}d \qquad j = 0, 1, \ldots, n$$

with the initial value $w[0] = x$ and the quotient-digit q_{j+1}, normally in signed-digit format, which provides $\log_2 r$ bits of the quotient at each iteration. The quotient-digit selection is

$$q_{j+1} = SEL(d_\delta, y) \qquad q_{j+1} = [-a, a]$$

where d_δ is d truncated after the δ-th fractional bit and the estimated residual, $y = rw[j]_t$, is truncated after t fractional bits. The residual $w[j]$ is normally kept in carry-save format to have a shorter cycle time.

The divider is completed by a on-the-fly convert-and-round unit [13] which converts the quotient digits q_{j+1} from the signed-digit to the conventional representation, and performs the rounding.

The digit-recurrence algorithm is quite a good choice for the hardware implementation because it provides a good compromise latency-area/power and rounding is simple (the remainder is computed at each iteration). A radix-4 division scheme is implemented in Intel Pentium CPUs [5], in ARM processors [6] and in IBM FPUs [14].

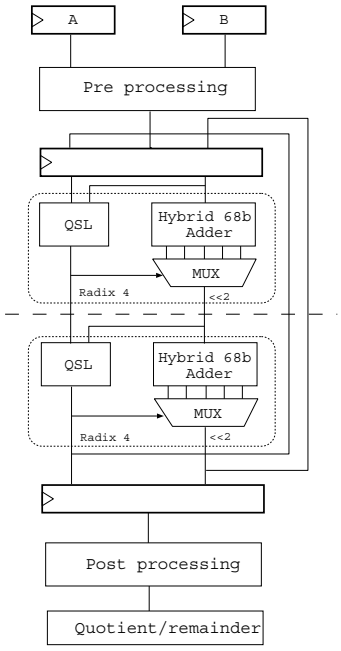

Fig. 3. Architecture of Penryn divider.

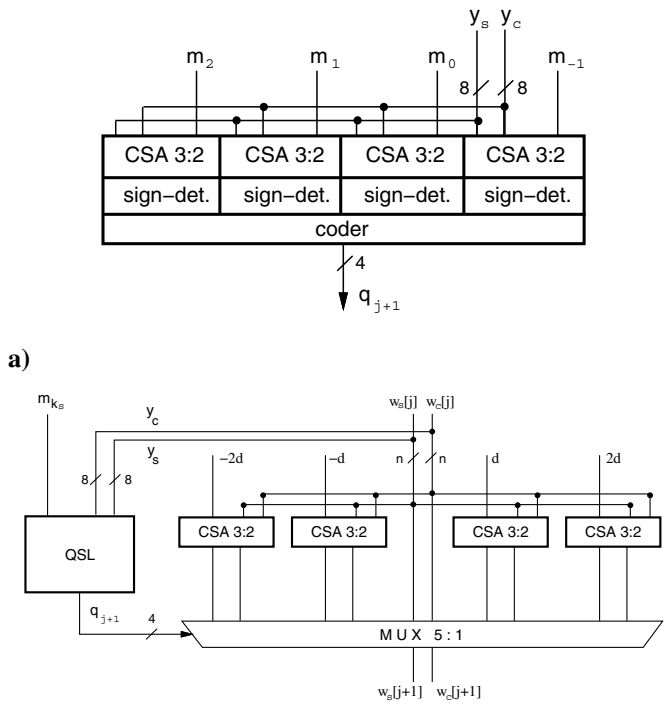

Fig. 4. **a)** Selection by comparison (QSL). **b)** Single radix-4 division stage.

A. Intel Penryn division unit

The division unit implemented in the Intel Core2 (Penryn) family is sketched in Fig. 3 [5]. It implements IEEE binary32/binary64 compliant division, plus extended precision (64 bits) and integer division. The unit consists of three main parts: the pre-processing stage necessary to normalize integer operands to ensure convergence; the recurrence stage; and the post-processing stage where the rounding is performed.

The recurrence is composed of two cascaded radix-4 stages synchronized by a two-phase clock to form a radix-16 stage (4 bits of quotient computed) over a whole clock cycle. Each radix-4 stage is realized with a scheme similar to that of [6] shown in Fig. 4.

We now briefly summarize the algorithm of [6]. The radix-4 recurrence is

$$w[j+1] = 4w[j] - q_{j+1}d \qquad j = 0, 1, \ldots, n$$

with $q_{j+1} = \{-2, -1, 0, 1, 2\}$.

The quotient-digit q_{j+1} is determined by performing a comparison of the truncated residual $y = \widehat{4w[j]}$ (carry-save) with the four values (m_k) representing the boundaries to select the digit for the given d. That is,

$$\begin{array}{rcccl} & y & \geq & m_2 & \rightarrow \quad q_{j+1} = 2 \\ m_1 & \leq & y & < & m_2 \rightarrow \quad q_{j+1} = 1 \\ m_0 & \leq & y & < & m_1 \rightarrow \quad q_{j+1} = 0 \\ m_{-1} & \leq & y & < & m_0 \rightarrow \quad q_{j+1} = -1 \\ & & y & < & m_{-1} \rightarrow \quad q_{j+1} = -2 \end{array}$$

This selection can be implemented with a unit (QSL) similar to that depicted in Fig. 4.a where four 8-bit comparators (sign-det.) are used to detect in which range y lies. The coder then encodes q_{j+1} in 1-out-4 code which is suitable to drive multiplexers.

In parallel, all partial remainders $w^k[j+1]$ are computed speculatively (Fig. 4.b), and then one of them is selected once q_{j+1} is determined. The carry-save output of the radix-4 stage is than hardwired to shift-left 2 bits (multiplication by 4).

This scheme was selected because of the reduced logical depth. The critical path of the unit in Fig. 4 is

$$t_{REG} + t_{QSL} + t_{buf} + t_{MUX} + t_{su} \qquad (1)$$

where t_{REG} and t_{su} are the w-register propagation delay and set-up time, t_{QSL} is the delay of the QSL block ($t_{CSA} + t_{8b-CPA} + t_{coder}$), t_{buf} is the delay of a buffer necessary to drive the high load of the multiplexer select, and t_{MUX} is the delay of the multiplexer.

However, the speculation on the whole w-word (54 bits for [6], 68 bits for the Core2 format) is quite expensive in terms of area and power dissipation.

According to [5], a maximum of 6+15=21 cycles are required to perform a division on binary64 (double-precision) operands.

B. Low power radix-16 division unit

An alternative to the Penryn solution, is to have a radix-16 divider obtained by overlapping (and not cascading) two radix-4 stages. In this scheme, the speculation is applied to the narrower y-path as explained next. Examples of radix-16 dividers by radix-4 overlapping are reported in [13] and [7].

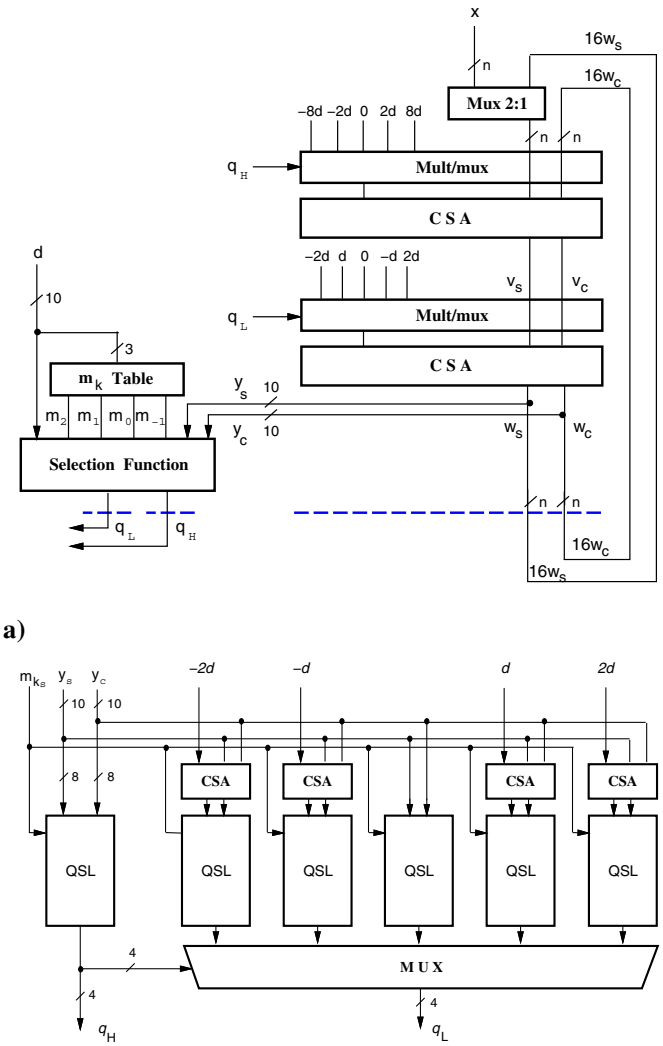

Fig. 5. **a)** Recurrence radix-16. **b)** Overlapped selection function.

The radix-16 retimed recurrence, illustrated in Fig. 5.a, is

$$\begin{aligned} v[j] &= 16w[j-1] - q_{Hj}(4d) \\ w[j] &= v[j] - q_{Lj}d \end{aligned}$$

with $q_{Hj} \in \{-2,-1,0,1,2\}$, $q_{Lj} \in \{-2,-1,0,1,2\}$, and $w[0] = x$ (eventually shifted to ensure convergence). In Fig. 5.a, the position of the registers is indicated with a dashed horizontal line. The recurrence is retimed (the selection function is accessed at the end of the cycle) to increase the time slack in the bits of the wide w-path (at right) so that these cells can be redesigned for low power [7].

The block QSL in Fig. 5.b is the same as that of Fig. 4.a. In this case, while q_H is computed, all five possible outcomes of q_L are computed speculatively. Therefore, the computation of q_L is overlapped to that of q_H, and q_L is obtained with a small additional delay.

Unit	T_c $[ns]$	Cycles	Latency $[ns]$	Area $[\mu m^2]$	P_{ave} $[mW]$
FMA ADD only	0.75	4	3.0	129,807	57.0
FMA MUL only					265.1
FMA MA fused					291.4
FMA DIV	0.75	26	19.5		124.1
Penryn	0.80	18	14.4	22,900	13.27
r16div	0.75	18	13.5	16,110	7.78

P_{ave} is average power measured at 1 GHz.

TABLE II
RESULTS OF IMPLEMENTATIONS.

The critical path of the unit in Fig. 5 is

$$t^{qH}_{REG} + t_{buf} + t^{H}_{MUL} + t^{H}_{CSA} + t^{L}_{CSA} + t_{QSL} + t^{SEL}_{MUX} + t^{qL}_{su}$$

Because of the retiming, only the 10 MSBs of w_s and w_c are on the critical path, while the rest can be optimized for low power during synthesis.

The total number of iterations to compute a binary64 division, including initialization and rounding, is 3+15=18.

V. IMPLEMENTATION OF THE FP-UNITS

To analyze the impact on power dissipation of the different units and to evaluate the different approaches to division, we implemented the three units for binary64:

- **FMA** is the fused multiply-add unit of Fig. 2 modified to execute the NR division algorithm.
- **r16div** is the radix-16 divide unit of Fig. 5 completed with convert-and-round unit and sign and exponent computation and update.
- **Penryn** is the division unit of Fig. 3 modified to handle binary64 only. That is, the recurrence is composed by two cascaded radix-4 stages (Fig. 4.b) plus the same initialization, convert-and-round unit and sign and exponent processing as the **r16div**.

The units are synthesized by Synopsys's Design Compiler with a 65 nm standard cell library and are laid-out by Synopsys's IC Compiler. All units are synthesized to obtain the maximum speed. Because in our flow we do not use two-phase clocks, for the **Penryn** implementation we cascaded the two radix-4 stages of Fig. 3 into a single clock cycle.

We used Synopsys's Power Compiler to estimate the power based on randomly generated input vectors conformed to IEEE 754 binary64 format. The synthesis results are summarized in Table II. The power dissipation data for the FMA are divided by operation.

The **Penryn** unit cannot be clocked at $T_C = 0.75\ ns$ (corresponding to a frequency of 1.33 GHz) as its critical path (1) is now

$$t_{REG} + 2 \cdot (t_{QSL} + t_{buf} + t_{MUX}) + t_{su}\ .$$

The proposed **r16div** scheme is the one with the lowest latency for division. It can probably be clocked with the same scheme used in Intel Core2 FP-units and provide the same throughput at reduced area and power dissipation.

The power dissipation is normalized for all units at 1 GHz to have a fair comparison. For the three operations: ADD, MUL and MA fused, the power was measured with the pipeline full to get the worst case power dissipation necessary to characterize the thermal behavior (Section VII) of the units. For division, being an iterative algorithm, the power was measured per operation. This explains why the value P_{ave} for FMA DIV is smaller than the other FMA cases.

As for floating-point division, from the data of Table II it is clear that the digit-recurrence approach (**Penryn** and **r16div**) is much more convenient in terms of latency, area and power dissipation. The only argument in favor of the FMA DIV is that division is much less frequent than addition and multiplication to justify the longer latency and the ten-fold power dissipation.

VI. Energy Consumption in FP-operations

In [15], the average frequency of floating-point operations in the SPECfp92 benchmark suite is reported. The most common instructions are multiply and add with MULT accounting for 37% and ADD for 55%. Moreover, the FP adder consumes nearly 50% of the multiply results which explains why fused multiply-add units are often used in modern processors. Table III summarizes the instruction mix. The first column shows the mix when none of the MULT and ADD instructions are fused. The second column shows the updated mix when 50% of the MULT instructions are fused with ADD.

To compare the energy consumption of units with different latencies a suitable metric is the energy consumption per operation (E_{op}) which is defined as

$$E_{op} = P_{ave} \times latency = P_{ave} \cdot n \cdot T_C \quad [J] \ .$$

The values of E_{op} (at 1 GHz) for the FP-units implemented are listed in Table IV.

Due to the significant reduction in E_{op} for division, we propose to use a digit-recurrence (**Penryn** or **r16div**) divider for FP-division. To compare the energy savings between division by FMA and by digit-recurrence, we use the SPICE benchmark which has a rather high percentage of divisions [16].

We list the results of the comparison in Table V to show the upper and lower bound of the energy consumption. In Table V, the results are obtained by assuming none of the MULT instructions can be fused with the ADD (top) and by assuming all MULT can be fused with the ADD (bottom). In all three cases, the MULT and ADD operations are implemented by the FMA unit. The DIV operation is implemented either in **Penryn**, **r16div** or FMA.

Note that the comparison is based on the four FP operations (ADD, MULT, Fused MA, DIV). Therefore, *Percentage* in Total is not 100%. Due to the reduction in the number of instructions by fusing MULT and ADD, there is a reduction in the total amount of instructions which is reflected in the *Percentage* in Total in Table V (bottom).

	not fused	*fused*
ADD	55.0%	44.8%
MULT	37.0%	22.7%
FMA	0%	22.7%
DIV	3.0%	3.7%
OTHERS	5.0%	6.1%

TABLE III
INSTRUCTION MIX.

Unit	n	P_{ave} $[mW]$	E_{op} $[pJ]$
FMA ADD only	4	57.0	228.11
FMA MUL only	4	265.1	1060.41
FMA MA fused	4	291.4	1165.58
FMA DIV	26	124.1	3226.57
Penryn	18	13.27	238.79
r16div	18	7.78	139.98

P_{ave} is average power measured at 1 GHz.

TABLE IV
ENERGY-PER-OPERATION.

In both cases (fused MA or not) there is significant reduction (around 40%) in energy consumption by using a digit-recurrence divider to implement binary64 division.

VII. Thermal Analysis

To perform the thermal analysis, we use the model proposed in [17], which consists of a conventional RC model of the heat conduction paths around each thermal element. The differential equation modeling heat transfer according to the Fourier's law is solved by first transforming it into a difference equation, and using the electrical simulator SPICE to solve the equivalent RC circuit.

A circuit is meshed into three-dimensional thermal cells. The z direction is discretized into 9 layers and on each layer x and y directions are both discretized into 20 units which result in a grid of $20 \times 20 \times 9 = 3600$ cells in total. This provides us accurate temperature estimations at standard cell

	Percentage	*Penryn* $[pJ]$	*r16div* $[pJ]$	*FMA* $[pJ]$
DIV	8.0%	19.1	11.2	258.1
ADD	45.0%	102.7		
MUL	26.0%	275.7		
Fused MA	0.0%	–		
Total	79.0%	397.5	389.6	636.5

	Percentage	*Penryn* $[pJ]$	*r16div* $[pJ]$	*FMA* $[pJ]$
DIV	8.0%	19.1	11.2	258.1
ADD	19.0%	43.3		
MUL	0.0%	–		
Fused MA	26.0%	303.1		
Total	53.0%	365.5	357.6	604.5

TABLE V
ENERGY CONSUMPTION IN SPICE WITHOUT (TOP) AND WITH (BOTTOM) FUSED MULTIPLY-ADD.

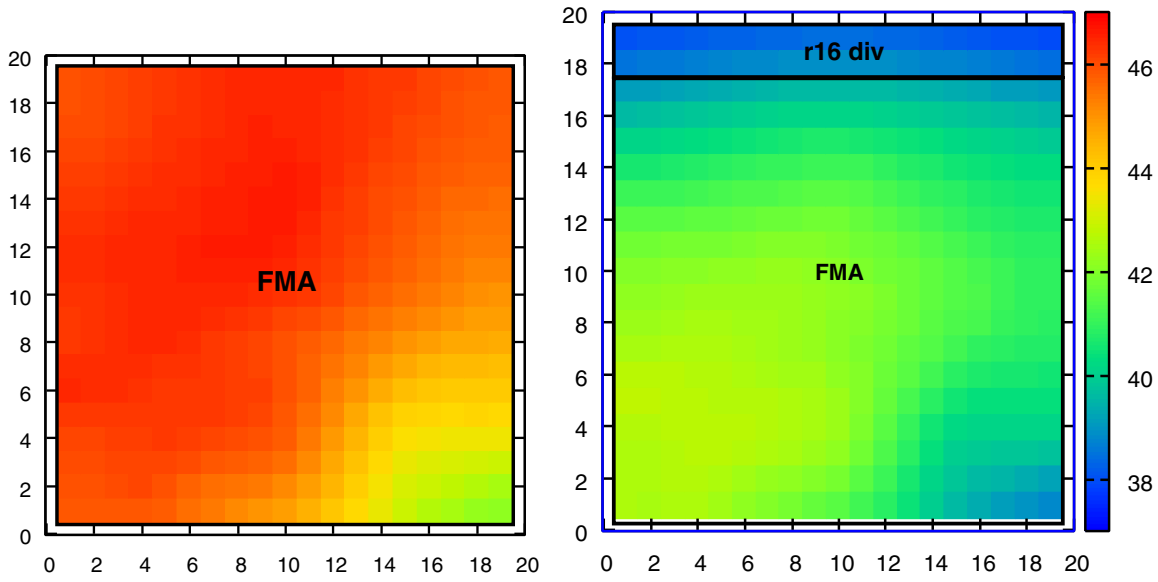

Fig. 6. Comparison of thermal profiles: FMA alone (left) and FMA plus **r16div** (right). Temperatures are in °C.

level with a reasonable simulation overhead. Cells inside the grid are connected to each other while cells on the boundary are connected to voltage sources that model the ambient temperature. In our thermal model, we adopted the thermal conductivities of different layers from [18].

Because the radix-16 division unit (**r16div**) has a much lower power consumption than the FMA, by having a separate division unit will not only save energy, but also reduce the peak temperature since the **r16div** unit can act as a heat spreader. Similar floorplan strategies can be found in high-end multicore processors where caches are placed beside cores [19] to partly take away the excessive heat generated inside the cores.

Fig. 6 shows the impact on temperature distribution when the **r16div** is placed next to the FMA. Temperatures shown in the figure indicate the rise above the ambient temperature. The left figure is a thermal map of the FMA unit alone and the right figure shows the thermal map when a **r16div** unit is placed above the FMA. Power consumption in both units are estimated based on workload characterized by the instruction mix with fused MA as shown in Table III. The size of the thermal maps of Fig. 6 is scaled to the physical dimensions (area), but both maps are determined on the same grid of thermal cells (i.e. the planar grid 20×20 of the right figure is coarser).

In the left thermal map in Fig. 6, the upper left region corresponds to the tree multiplier in the FMA unit which occupies most of the area and also has the highest power consumption. This is reflected by the high temperature colored in red. The peak temperature reached is 47°C over the ambient temperature. In the right thermal map, by putting the divider beside the multiplier the hot-spot becomes smaller. The area of the circuit is increased and as a result thermal resistance to the ambient is reduced. The peak temperature dropped from 47°C to 43°C and the average temperature is also reduced by about 4°C.

In high-end processors where there are multiple floating-point execution units, the peak temperature can become extremely high due to the increased thermal coupling between these units. The low power division units like **r16div** can be used to partly mitigate the thermal problem. By placing the division units in between the FMA units, we can reduce the thermal coupling and thus reduce peak temperature. In nanometer technologies where static power becomes dominant, even a slight reduction in temperature can have positive impacts on the increasing leakage power. Table IV and Table V show that the **r16div** unit also saves energy when the effective operation is division. However, the overhead is an increase in total area.

VIII. Conclusions

In this paper, we compared the power dissipation and energy consumption in floating-point units which are typically used in modern multicore processors and GPGPUs. Implementing the division operation in a Fused Multiply-Add (FMA) unit consumes much more power and energy than in a dedicate divider based on the digit-recurrence algorithm. By using a digit-recurrence divider together with the FMA, we found a significant reduction of energy in a typical scientific application over the implementation of all floating-point arithmetic operations in FMA. Moreover, adding a separate divider also reduces the peak temperature rise in the FMA by about 4°C according to our thermal analysis.

These results suggest to implement digit-recurrence based FP-division units in multicores and GPUs. Sacrificing a little area – the **r16div** area is 1/8 that of the FMA – it would be possible to reduce both power dissipation and chip temperature at same, or improved, performance. The division-by-FMA approach has been preferred because the low percentage of

divisions in programs allows multiple issues of more frequent multiply and add operations in FMAs. However, in GPUs where several FP-units are grouped in clusters (e.g. in [4]) it might be reasonable to include a digit-recurrence division (and square-root) unit in each cluster to reduce the power dissipation when the percentage of divisions is not negligible.

REFERENCES

[1] X. Fan, W.-D. Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-sized Computer," *Proc. of ACM International Symposium on Computer Architecture*, June 2007.

[2] *IEEE Standard for Floating-Point Arithmetic*, IEEE Computer Society Std. 754, 2008.

[3] S. M. Mueller *et al.*, "The Vector Floating-Point Unit in a Synergistic Processor Element of a CELL Processor," *Proc. of 17th Symposium on Computer Arithmetic*, pp. 59–67, June 2005.

[4] NVIDIA. "Fermi. NVIDIA's Next Generation CUDA Compute Architecture". Whitepaper. [Online]. Available: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[5] H. Baliga, N. Cooray, E. Gamsaragan, P. Smith, K. Yoon, J. Abel, and A. Valles, "Improvements in the Intel Core2 Penryn Processor Family Architecture and Microarchitecture," *Intel Technology Journal*, pp. 179–192, Oct. 2008, http://www.intel.com/technology/itj/2008/v12i3/3-paper/1-abstract.htm.

[6] N. Burgess and C. Hinds, "Design Issues in Radix-4 SRT Square Root and Divide Unit," *Proc. 35th Asilomar Conference on Signals, Systems and Computers*, pp. 1646–1650, 2001.

[7] A. Nannarelli and T. Lang, "Low-power division: Comparison among implementations of radix 4, 8 and 16," *Proc. of 14th Symposium on Computer Arithmetic*, pp. 60–67, 1999.

[8] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[9] T. Lang and J. D. Bruguera, "Floating-point multiply-add-fused with reduced latency," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 988–1003, Aug. 2004.

[10] S. F. Oberman, "Floating-point division and square root algorithms and implementation in the AMD-K7 microprocessor," *Proc. of 14th Symposium on Computer Arithmetic*, pp. 106–115, 1999.

[11] H. Sharangpani and H. Arora, "Itanium processor microarchitecture," *IEEE Micro*, vol. 20, no. 5, pp. 24–43, Sep/Oct 2000.

[12] D. DasSarma and D. W. Matula, "Measuring the Accuracy of ROM Reciprocal Tables," *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 932–940, Aug. 1994.

[13] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publisher, 1994.

[14] G. Gerwig, H. Wetter, E. M. Schwarz, and J. Haess, "High performance floating-point unit with 116 bit wide divider," *Proc. of 16th Symposium on Computer Arithmetic*, pp. 87–94, 2003.

[15] S. Oberman and M. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, pp. 154–161, February 1997.

[16] J. L. Hennessy and D. A. Patterson, *Computer Architecture: a Quantitative Approach*, 2nd ed. Morgan Kaufmann Publishers Inc., 1995.

[17] W. Liu, A. Calimera, A. Nannarelli, E. Macii, and M. Poncino, "On-chip Thermal Modeling Based on SPICE Simulation," *Proc. of 19th International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS 2009)*, pp. 66–75, Sept. 2009.

[18] T. Sato, J. Ichimiya, N. Ono, K. Hachiya, and M. Hashimoto, "On-chip thermal gradient analysis and temperature flattening for SoC design," *Proc. of Design Automation Conference Asia and South Pacific (ASP-DAC)*, vol. 2, pp. 1074–1077, Jan. 2005.

[19] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora, "A 45 nm 8-Core Enterprise Xeon Processor," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 1, pp. 7–14, Jan. 2010.