Architecting a MOS Current Mode Logic (MCML) Processor for Fast, Low Noise and Energy-Efficient Computing in the Near-Threshold Regime

Yuxin Bai, Yanwei Song, Mahdi Nazm Bojnordi, Alexander Shapiro, Engin Ipek, and Eby. G. Friedman
Department of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627, USA
Emails: {yuxin.bai, yanwei.song, m.nazmbojnordi, alexander.shapiro}@rochester.edu,
ipek@cs.rochester.edu, friedman@ece.rochester.edu

***Abstract*— Near-threshold computing (NTC) is an effective technique for improving the energy efficiency of a CMOS microprocessor, but suffers from a significant performance loss and an increased sensitivity to voltage noise. MOS current-mode logic (MCML), a differential logic family, maintains a low voltage swing and a constant current, making it inherently fast and low-noise. These traits make MCML a natural selection to implement an NTC processor; however, MCML suffers from a high static power regardless of the clock frequency or the level of switching activity, which would result in an inordinate energy consumption in a large scale IC. To address this challenge, this paper explores a single-core microarchitecture for MCML that takes advantage of C-slow retiming technique, and runs at a high frequency with low complexity to save energy. This design principle is opposite to the contemporary multicore design paradigm for static CMOS that relies on a large number of gates running in parallel at modest speeds. When compared to an eight-core static CMOS processor operating in the near-threshold regime, the proposed processor exhibits $3\times$ higher performance, $2\times$ lower energy, and $10\times$ lower voltage noise, while maintaining a similar level of power dissipation.**

I. Introduction

Near-threshold computing (NTC) is a promising approach to significantly improve the energy efficiency of a microprocessor, in which the supply voltage is scaled down to a level near the threshold voltage of the transistors. However, these energy savings come at a ten times performance cost [1], which confines NTC to low-power applications. In addition, design issues such as the increased sensitivity to voltage noise [2] and process variations [1], make it challenging to apply NTC to high-speed designs.

MOS current mode logic (MCML) is a fast, low noise differential logic style, which has garnered recent interest as a replacement for static CMOS circuits in noise sensitive applications [3]–[5]. MCML is the CMOS successor of the bipolar emitter coupled logic (ECL), which has been used in high speed applications since the 1970s. The Cray-1 [6], the IBM Enterprise System/9000™ [7], and the IBM System/390® [8] all used ECL and differential current switching to achieve high speeds with low noise.[1] MCML maintains the benefits of traditional ECL, such as high speed, a reduction in di/dt noise, and common mode noise rejection, without relying on bipolar transistors [3]. It requires a constant current independent of frequency, making this technology particularly appropriate for high speed integrated circuits (ICs). These traits make it possible to address both the speed and the signal integrity challenges in an NTC processor. Unfortunately, MCML suffers from a constant static power dissipation, which, if left unmanaged, would result in an inordinate energy requirement in large scale ICs operating at moderate speeds (*e.g.*, 1 to 2 GHz). This limitation heretofore has prevented the use of MCML as a replacement for static CMOS in mainstream multicore processors; however, speed, signal integrity and power delivery challenges, as well as the increasingly large CMOS static power due to sub-threshold leakage, are now tilting the balance in favor of MCML—particularly in single core processors operating at high frequencies.

This paper explores the use of MCML as an alternative to static CMOS circuits in near-threshold computing for throughput-oriented microprocessors, thereby improving the signal integrity, performance, and energy efficiency of future high performance systems. Mitigating static power requires exploiting the unique circuit level power characteristics of MCML at the architecture level, and breaking from the contemporary multicore design paradigm, which relies on a large number of gates running in parallel at modest speeds. A complexity effective, deeply pipelined, multithreaded architecture specific to MCML is designed, and synthesized at 22nm using an MCML standard cell library. The proposed MCML processor outperforms an 8-core static CMOS processor operating in the near threshold regime by $3\times$ in execution time and $2\times$ in energy, while exhibiting $10\times$ lower voltage noise. Moreover, the operation of the MCML processor is robust under both systematic and random variations in transistor threshold voltage and effective channel length.

II. Background and Motivation

The growing importance of signal integrity and power delivery challenges, compounded by the significant performance loss in near-threshold computing as well as the increasingly large CMOS static power due to sub-threshold leakage, are now diminishing the advantages of static CMOS design over other logic families. This section provides an overview of near-threshold computing, the signal integrity problems in CMOS ICs, and the design principles behind MCML. Detailed descriptions of MCML are beyond the scope of this paper; interested readers can find more information on MCML in the literature [3], [5], [10]–[12].

Near-Threshold Computing. The operation of a CMOS transistor is typically divided into three regions according to

[1]The low noise environment has allowed the Cray-1 to use an entirely unregulated power supply [9].

978-1-4673-7166-7/15/$31.00 ©2015 IEEE

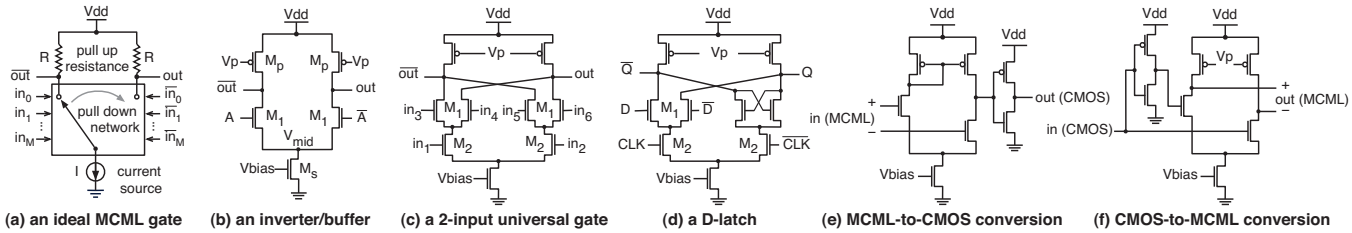

Fig. 1. Exemplary MCML gates and interfacing circuitry.

the range of supply voltage. Typical high performance processors in static CMOS nominally operate at the super-threshold region. As the supply voltage scales down to a level near the threshold voltage of transistors (near-threshold region), a significant energy improvement can be achieved with some performance loss. When the supply voltage continues to scale below the threshold voltage (sub-threshold region), however, the device becomes increasingly slow and energy inefficient due to the exponential increase in leakage and gate delay. Among these regions, near-threshold region is considered to be most energy efficient, making near-threshold computing appealing for energy-constrained applications. To take advantage of NTC, however, the performance loss must be compensated, and the increased sensitivity to process variations and noise [2], [13] must be overcome.

Signal Integrity. Electrical noise can be categorized into device and switching noise. Device noise is intrinsic to transistors, including thermal, shot, and flicker noise. Switching noise results from the simultaneous switching of devices, which is a primary concern in conventional CMOS digital circuits, and is typically two to three orders of magnitude greater than device noise [14]. It is coupled to the imperfect power delivery networks (PDN) through resistive IR-drops and inductive Ldi/dt noise, and can be coupled to the interconnect through crosstalk. CMOS digital circuits are traditionally considered tolerant to noise due to their high noise margins [14]. This condition, however, has changed due to scaled power supply and threshold voltages. High-frequency signal transitions exacerbate the magnitude of the coupling noise, which makes signal integrity a significant challenge in high-speed digital ICs. As a result, microprocessor designers are forced to incorporate expensive hardware solutions and pessimistically employ worst-case designs to ensure reliable operation. For example, the guard-bands must be adjusted for the worst-case voltage droop, which results in a significant energy loss ($2\times$ energy loss due to pessimistic PDN design in [2]). Microarchitecture techniques have been proposed [15]–[18] to address inductive noise and IR drops at the expense of performance, power, and area. The design of future microprocessors therefore requires innovative solutions to compensate for noise in highly scaled technologies.

	CMOS	MCML
Delay	$\frac{N\times C\times V_{dd}}{\frac{k}{2}(V_{dd}-V_{th})^{\alpha}}$	$\frac{N\times C\times V_{swing}}{I}$
Power	$N\times C\times V_{dd}^2\times f$	$N\times V_{dd}\times I$
Energy	$N\times C\times V_{dd}^2$	$N^2\times C\times V_{swing}\times V_{dd}$
EDP	$\frac{N^2\times C^2\times V_{dd}^3}{\frac{k}{2}(V_{dd}-V_{th})^{\alpha}}$	$\frac{N^3\times C^2\times V_{swing}^2\times V_{dd}}{I}$

TABLE I. COMPARISONS BETWEEN CMOS AND MCML. ASSUMING N IDENTICAL GATES ARE CONNECTED IN A LINEAR CHAIN. α AND k ARE PROCESS DEPENDENT PARAMETERS; C, I, f ARE THE LOAD CAPACITANCE, TAIL CURRENT, AND FREQUENCY RESPECTIVELY.

MOS Current-Mode Logic. MCML was introduced by Yamashina *et al.* in 1992 as an inherently high speed, low noise, and analog friendly digital logic family for mixed signal environments [11], and has been used in applications such as optical communication transceivers and high-speed ring oscillators [3], [19]–[21]. An MCML gate (Figure 1 (a)) comprises three major parts: a pair of pull up load resistors (typically implemented with PMOS transistors, as shown in an MCML inverter in Figure 1 (b)), a differential pull down logic network that generates true and complementary forms of the output signals, and a constant current source (typically a single saturated NMOS transistor). The current source is controlled by a separate voltage, V_{bias}, and is steered between the branches, but the total current is kept constant, resulting in far less switching noise but a higher static power than that of static CMOS gates. The voltage swing, $V_{swing} = I \times R$, can be adjusted by tuning the tail current as well as the gate and body voltages of the PMOS loads. The voltage swing usually is not rail-to-rail and can be 2-10$\times$ lower than V_{dd}. A comparison between CMOS and MCML gates in terms of delay, power, energy, and energy-delay product (EDP) is shown in Table I. One appealing property of MCML is that the gates do not exhibit theoretically minimum EDP [3]. Both the energy and the EDP benefit from the low voltage swing, but the energy efficiency deteriorates with logic depth N (*e.g.*, the number of gates connected in series in a pipeline stage). It is therefore important for MCML systems to maintain a shallow logic depth in each pipeline stage. This quality implies that MCML circuits with constant power consumption can be significantly more energy efficient than CMOS circuits (dynamic power increases with frequency) when operating at high speeds [3].

C-Slow Retiming. C-slow retiming [22], [23] is a pipelining technique to improve the throughput of a circuit with feedback loops. It consists of an initial C-slowing step in which every register is replaced with a sequence of C registers, followed by a retiming step that balances the logic delay between pipeline registers. In the ideal case, this optimization improves the throughput of the pipeline by a factor of C, while introducing additional latency for a single data stream. If C interleaved, independent data streams (*e.g.*, threads) enter the pipeline on a round robin basis, no new combinational logic is needed to create new feedback loops between the stages. In this paper, C-slow retiming is combined with simultaneous multi-threading to permit a digital processor to operate at high frequencies. In C-slow retiming, the performance and power can both be improved by employing a low complexity circuit (superpipelining without C-slow retiming on an MCML processor would improve performance at the expense of increased complexity and static power). C-slow retiming is particularly important for MCML rather than static CMOS in terms of energy efficiency, since the power of MCML is static and independent of frequency.

III. BUILDING BLOCKS

The fundamental principles behind the design of the MCML gates and libraries used in the evaluation are described

in this section. To implement a baseline for comparison, the static CMOS gates from the NCSU FreePDK 45nm standard cell library are scaled to 22nm. The most energy efficient voltage level is found to be 0.55V (near threshold region) for most of these gates. A 22nm MCML standard cell library (19 gates at 0.55V with a 100mV voltage swing) is designed to study the system-level characteristics. The design space for each cell is searched by varying parameters such as the transistor sizes, the voltage swing, the tail current, and the supply voltage. Detailed procedure is shown in Table II.

Given variables: d_t, V_{swing}, W_1, L_1, W_p, L_p, W_s, L_s, I, V_p, V_{mid}, V_{bias}
Minimize: power consumption P
Subject to:
(1) Performance target, $t_d \leq d_t$
(2) Supply voltage, $0.4V \leq V_{dd} \leq 0.8V$
(3) Signal slope ratio (defined as rise or fall time/prorogation delay), $SSR < 5$
(4) Voltage swing ratio (defined as I_{on}/I), $VSR > 0.95$
(5) Within die variations (δ/μ), 9% for V_{th}, 4.5% for L_{eff}
(6) Gain and noise margin, $A_v > 2$, $NM > 0.4V_{swing}$
(7) Load resistance R, tuned with load PMOS transistors W_p, L_p, V_p
(8) Voltage swing, $V_{swing} = I \times R$, $0.05V \leq V_{swing} \leq V_{th_1}$
(9) Constant current, $I = W_s v_{sat} C_{ox}(V_{bias} - V_{th_s} - A_s V_{dsat_s})$
(10) Bias voltage, $V_{th_s} \leq V_{bias} \leq V_{mid} + V_{th_s}$
(11) Tail transistor, $V_{dd} - V_{th_1} - V_{swing} \leq V_{mid} \leq V_{dd} - V_{th_1}$

TABLE II. THE DESIGN SPACE EXPLORATION PROCEDURE FOR A SINGLE-LEVEL MCML GATE. I IS THE TOTAL CURRENT, AND I_{on} IS THE CURRENT THROUGH THE ON BRANCH ONLY. VARIABLES WITH SUBSCRIPT "1" ARE FOR NMOS TRANSISTORS M_1 IN THE DIFFERENTIAL PAIR, WITH SUBSCRIPT "s" ARE FOR THE TAIL NMOS TRANSISTOR, AND WITH SUBSCRIPT "p" ARE FOR LOAD PMOS TRANSISTORS AS SHOWN IN FIGURE 1 (B) IN THE PAPER.

Similar to static CMOS gates, process variations should be considered when MCML gates are operating at the near-threshold voltage. This paper takes into account the process variation in the design by considering a $\pm 12\%$ die-to-die threshold voltage variation, a within-die threshold voltage variation (σ/μ) of 9%, and a within-die effective channel length variation of 4.5% [24]. The basic gates are upsized according to the $\sigma \propto 1/(\sqrt{area})$ law [25], and transistor stacking is strictly restricted to at most two levels to avoid functional failures. Delay, area, and power comparisons between the MCML library and the static CMOS library for seven representative cells are shown in Table III.

Basic Cells	Delay (ps)		Area (μm^2)		Power (μW)	
	MCML	CMOS	MCML	CMOS	MCML	CMOS
INVX1	8.50	22.70	0.67	0.34	1.05	0.29
NAND2X1	10.10	40.77	1.30	0.45	0.89	0.31
NOR2X1	17.25	53.11	1.30	0.56	0.89	0.38
XOR2X1	20.02	54.60	1.30	1.22	1.51	0.73
MUX2X1	15.20	51.38	1.30	1.19	1.51	0.54
DFFX1	49.00	129.01	3.12	1.91	4.11	0.84
LATCHX1	34.02	92.05	1.59	1.23	1.56	0.55

TABLE III. DELAY, AREA, AND POWER COMPARISON OF MCML AND STATIC CMOS (0.55V) STANDARD CELL LIBRARIES. ASSUME A 100% SWITCHING ACTIVITY FOR STATIC CMOS CELLS OPERATING AT 1 GHZ, $17ps$ INPUT TRANSITION TIME, $1fF$ LOAD CAPACITANCE, AND THE SAME INPUT SWITCHING PATTERNS FOR ALL THE PRESENTED CELLS.

Universal Gates. A universal gate is a standard logic gate that can implement one of multiple logic functions by exchanging input signals. For example, the circuit in Figure 1 (c) can implement the AND, OR, XOR, MUX functions, as well as their complementary forms. Different input combinations are listed in Table IV to construct various logic functions with the same gate. The cell library used in the evaluation leverages this universal gate.

Storage Structures. MCML-based storage cells (Figure 1 (d) shows a D-Latch) maintain the low noise benefits of MCML, but suffer from a constant static power dissipation,

Logic function	in_1	in_2	in_3	in_4	in_5	in_6
XOR/NOXR	$\overline{A}$	A	B	$\overline{B}$	$\overline{B}$	B
MUX/NMUX	S	$\overline{S}$	D_1	$\overline{D_1}$	D_0	$\overline{D_0}$
AND/NAND	A	$\overline{A}$	B	$\overline{B}$	A	$\overline{A}$
OR/NOR	B	$\overline{B}$	B	$\overline{B}$	A	$\overline{A}$

TABLE IV. INPUT CONNECTIONS FOR A 2-INPUT UNIVERSAL GATE TO CONSTRUCT DIFFERENT LOGIC FUNCTIONS.

which can be problematic for large structures with low per-cell activities, such as caches. Therefore, SRAM cells at 0.8V nominal voltage (due to the reliability and energy efficiency concerns [13]) are used for L1 and L2 caches, register files, and TLBs, and appropriate interfacing is provided between the MCML and SRAM domains.

Interfacing Circuit. Different voltage domains are interfaced to each other via level shifters. Single-ended (static CMOS) and differential (MCML) domains communicate via interfaces that provide the required conversion. As shown in Figure 1 (e), conversion from MCML to static CMOS requires a differential to single-ended amplifier with a small gain (no more than 10), plus a static CMOS inverter that amplifies the low voltage swing beyond the switching threshold of a static CMOS gate. To convert from static CMOS to MCML domains (Figure 1 (f)), a static CMOS inverter and an MCML inverter are sufficient to generate and balance the differential outputs. Area and power depend on the total number of signals that need to cross the boundary between static CMOS and MCML modules, which is not significant according to the evaluation.

IV. DESIGN PHILOSOPHY

Power dissipation in MCML circuits is roughly proportional to the number of logic gates, but is independent of the switching frequency. As a result, the intuition behind an energy-efficient, current mode microprocessor is (1) to avoid using a large number of gates, and (2) to maintain a shallow logic depth in each pipeline stage to minimize the time each gate remains idle within a clock period. Maintaining energy efficiency in a current mode processor therefore requires a complexity-effective design that operates at a high frequency to reduce the execution time and static energy. A challenge that often plagues a deeply pipelined datapath is the problem of overcoming data dependences to keep the pipeline highly utilized. A secondary problem that is particularly important in the case of MCML is the static power overhead of the extra pipeline latches and control logic, which can offset the energy-efficiency benefits of a deep pipeline. To make a high-speed, deep pipeline viable for a current mode processor, this paper takes advantage of two synergistic techniques: C-slow retiming and multithreading. In this paper, each of the original pipeline stages (except writeback) in a 5-stage baseline design is partitioned into C stages. The resulting pipeline (4C+1 stages) works correctly without any new forwarding paths or control logic as long as threads are interleaved, and instructions from a given thread are restricted to issuing every C cycles.

Choosing the Optimum C. A critical parameter that determines the energy-efficiency of the proposed C-slowed MCML processor is C. It is possible to derive a simple analytical model that expresses system energy as a function of C and other design parameters to understand the tradeoffs involved in selecting C optimally. Different from models for static CMOS to find the optimal pipeline depth [26], the model for MCML should consider the static nature of its power.

Assume that a baseline MCML processor has a static power dissipation of P_0, and that the static power of the C-slowed pipeline is P_c. Let the marginal increase in static power (mainly extra MCML latch power) when going from a C-slowed design to a (C+1)-slowed design be P_δ for the entire pipeline; thus

$$P_c = P_0 + (C^\beta - 1)P_\delta \quad (1)$$

where the factor β indicates how the power overhead increases with the increased pipeline depth (*e.g.*, linear when $\beta = 1$, or superlinear when $\beta > 1$) [27], [28]. The baseline critical path delay is split into two components: the combinational delay d_0, and the pipeline latch delay d_δ. Since the combinational portion is split into C stages by C-slow retiming, the critical path delay of the new pipeline stage, d_c, will be $\frac{d_0}{C} + d_\delta$. The static energy consumed in one cycle by the C-slowed pipeline, E_c, will be

$$E_c = P_c d_c = (P_0 + (C^\beta - 1)P_\delta)(\frac{d_0}{C} + d_\delta) \quad (2)$$

For example, if the latch overhead increases linearly with C ($\beta = 1$), then the energy can be expressed as,

$$E_c = \frac{(P_0 - P_\delta)d_0}{C} + CP_\delta d_\delta + (P_0 d_\delta + P_\delta d_0 - P_\delta d_\delta) \quad (3)$$

This expression indicates that the energy to run an application on a C-slowed MCML pipeline depends on two competing factors: (1) the lower static energy wasted in each pipeline stage as C is increased, and (2) the lower latch energy overhead as C is decreased. Setting the derivative of E_c (with respect to C) to zero yields the optimum C value, C_{opt}, that minimizes static energy per cycle:

$$C_{opt} = \sqrt{(\frac{P_0}{P_\delta} - 1)\frac{d_0}{d_\delta}} \quad (4)$$

When the latch overhead (P_δ and d_δ) is negligible compared to the baseline pipeline (P_0 and d_0), C_{opt} is large; in this regime, the deeper the pipeline, the lower is the energy, and practical limitations on the achievable energy efficiency would be the maximum reachable pipeline depth and the available thread-level parallelism. If the latch overhead becomes comparable to the energy of the baseline pipeline, however, the energy efficiency starts to decrease beyond C_{opt}. For example, when $P_0 \approx 8P_\delta$ and $d_0 \approx 9d_\delta$, C_{opt} is 8, indicating that a 33-stage ($4C + 1$) pipeline is the most energy efficient design. In a case where the latch overhead increases super-linearly (*i.e.*, β is greater than 1), the optimum C would be smaller, and the energy efficiency would primarily be constrained by the pipeline overhead and the robustness requirement.

V. Pipeline Organization

The proposed complexity-effective current-mode microprocessor is a C-slowed version of a single-issue, in-order, multithreaded core with five stages. Each of the fetch, decode, execute, memory access, and writeback stages constitutes a major pipeline stage, and each of the four major pipeline registers is replaced with C pipeline registers using C-slow retiming. This converts the baseline core into a deeply pipelined processor with 4C pipeline registers and 4C+1 stages (*e.g.*, 33 stages in Figure 2). To avoid the need for extra forwarding and control logic, a restriction is placed on when a thread can enter a major pipeline stage. Specifically, each clock cycle is assigned a *time slot ID* between 0 and C-1, such that a clock cycle with a time slot ID of t is followed by a clock cycle with a time slot ID of $(t+1)$ mod C. Each thread is statically assigned to a time slot. For example, if C = 8, and a thread is assigned to time slot 1, that thread is eligible to enter the pipeline on cycles 1, 9, 17, 25, and so on. Interleaving the threads obviates the need for extra forwarding paths: from the point of view of a single thread, the processor is logically equivalent to the baseline five-stage pipeline, running at $\frac{1}{C}$ of the actual clock rate. This concept is illustrated in Figure 3 (b) and is compared to a conventional, more complex pipeline with extra forwarding paths in Figure 3 (a).

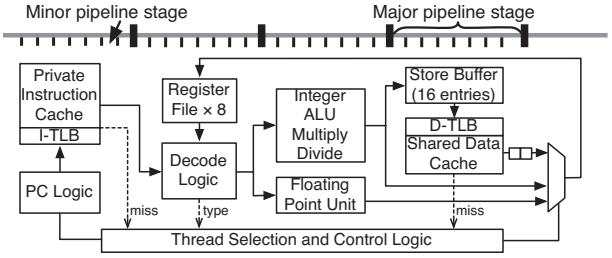

Fig. 2. Illustration of the proposed C-slowed (C=8) pipeline structure for the current mode processor.

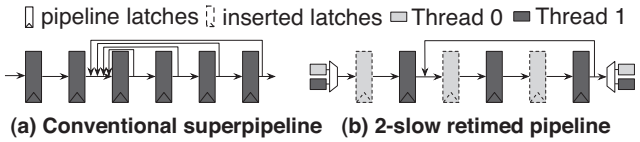

Fig. 3. Illustration of scheduling multiple threads in a conventional deep pipeline and a 2-slowed pipeline: (a) consecutive instructions from the same thread need a full set of forwarding paths; (b) instructions from interleaved threads use a simple forwarding path.

A. Instruction Fetch

The instruction fetch stage implements a fair fetch policy that allocates threads to each time slot in a 2-level round-robin fashion. Each time slot is identified by a time slot ID (Figure 4 (a)), and assigned a pre-designated set of hardware thread contexts. Figure 4 shows how threads are interleaved. In this example, threads 0, 1 and 9 are available for execution, while other threads are stalled for memory. Thread 0 has been statically assigned to time slot 0, while threads 1 and 9 have been assigned to time slot 1. Therefore, threads 1 and 9 take turns sharing time slot 1. The rest of the slots carry NOPs, waiting for one of the threads assigned to them to be ready.

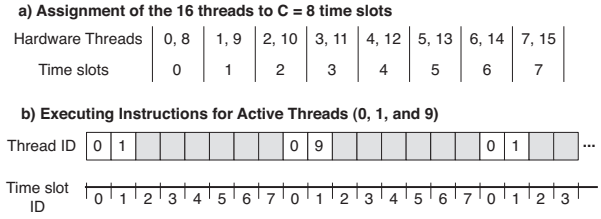

Fig. 4. Illustration of instruction assignment to time slots.

Program Counters. Each hardware thread is assigned a 32-bit program counter (PC). Given T hardware threads, a set of $\frac{T}{C}$ PC registers are statically allocated to each time slot. Every cycle, the fetch unit selects one of the ready PCs using a two-level round-robin selection policy. First, the time slot ID is used to select the corresponding set of PCs in round-robin order; next, within the selected set, one of the ready threads is selected (also in round-robin order).

Instruction Cache. The instruction cache consists of C 16KB cache banks. Each cache bank is dedicated to a particular time slot and provides a throughput of one access every C cycles. All threads assigned to the same time slot share the same cache bank. Since a bank can only be accessed in its dedicated time slot (*i.e.*, every C cycles), there is never a bank conflict among threads. One 32-entry, four-way set associative instruction TLB is used for each of the C cache banks.

B. Instruction Decode

The decode stage involves accessing both the instruction decoder and the register files.

Decoder. The decoder implements the MIPS ISA. It detects hazards and updates the state of the current thread. Since the architecture guarantees that a thread is not assigned to more than one time slot, and the original five-stage pipeline handles the necessary operand bypassing, no new hazards exist in the C-slowed pipeline.

Register Files. The processor incorporates C separate register files, each of which is dedicated to one time slot. If the number of hardware threads is T, every register file is partitioned into $\frac{T}{C}$ banks that share two read ports and a write port. As shown in Figure 5, 16 hardware threads are assigned to eight time slots. Each register file is enabled every eight cycles through shared input and output buses. Two banks of the same register file (*e.g.*, the banks for thread 0 and 8 in register file 0) are never accessed within the same eight-cycle period, and share the same read and write ports. This eliminates the need for extra register files ports, large selection logic, and routing for the inputs and outputs between register file banks.

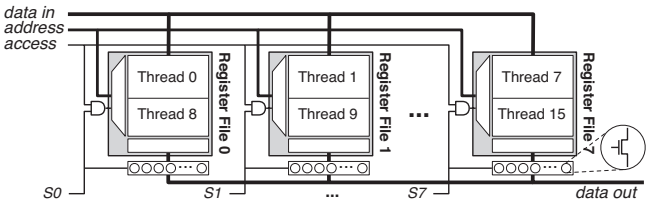

Fig. 5. Illustrative example of accessing the register files (8-slowed core with 16 hardware threads). T0-T7 denote the 8 time slot enable signals.

C. Execute

The execute stage includes an integer ALU, an FPU, a branch resolution unit, and the forwarding logic.

Integer ALU. The integer ALU consists of a logic unit, an adder/subtracter, a shifter, and a multiplier/divider. The multiplier/divider implements a 32-stage Booth algorithm, which allows for a high clock rate with a relatively small number of gates. Recall that, for an MCML circuit, a smaller gate count translates to lower static power; hence, a small, highly utilized unit delivers higher energy efficiency. The multiplier/divider is C-slow retimed, which results in a latency of 32C clock cycles. Since it is pipelined, consecutive multiply and divide instructions from different threads, as well as independent multiply/divide instructions from the same thread, do not need to wait. However, if the next instruction from the same thread is not a multiply or a divide, that thread is blocked to prevent out-of-order completion. This event results in a thread switch within the corresponding time slot.

Floating Point Unit. The FPU is based on the open source OR1K floating point unit [29], and implements a 4-stage, IEEE 754 compliant, single precision pipeline. Each original FPU stage takes 2 baseline clock cycles (2C cycles in the C-slowed pipeline); this amounts to 8C cycles for conversion, addition, and subtraction operations. For multiply and divide, a 32-stage Booth algorithm is used to minimize gate count; after C-slow retiming, this translates to 38C cycles of delay.

Branch Resolution Unit. Branches are resolved in the execution stage, which requires nullifying one over-fetched instruction on a taken branch. This strategy results in a shorter critical path, and consequently a higher frequency and better overall performance than resolving branches in the decode stage of the proposed architecture.

D. Data Memory Access

The memory stage implements load and store instructions by interfacing to a shared store buffer and a shared L1 data cache. The L1 data cache comprises C cache banks, each supported by a 32-entry, four-way set associative data TLB. A memory operation is initiated in a single cycle, and completes within C cycles on a cache hit. A data TLB or data cache miss results in a context switch to another thread assigned to the same time slot. Since L1 data cache banks are shared, there exists a possibility of bank conflicts among different threads when the clock period is shorter than the cache cycle time, which can happen when C is large. When a bank conflict is detected, the entire pipeline freezes for typically one or two minor cycles, until the conflicting bank becomes available. (The impact of bank conflicts was investigated and found not to constitute a performance bottleneck; the conflict rate is 6.4% when C=8.) After the conflict is resolved, the blocked request is sent to the data cache bank, and the pipeline restarts.

Store Buffer. The load-store unit implements a store buffer with one entry per hardware context. By checking the physical tags in the store buffer, a read-after-write (RAW) hazard between loads and stores can be detected, and resolved by store-load forwarding.

E. Register Writeback

If necessary, the retiring thread writes its result to the register file in the writeback stage. The time slot ID and the thread ID are used to determine the destination register file and the corresponding bank, respectively.

VI. Experimental Setup

A baseline single-issue, 5-stage, in-order MIPS processor ($C = 1$) is implemented in Verilog RTL, and retimed ($C = 2, 4, 8$) and synthesized with both MCML and CMOS cell libraries. Table V shows the design flow. A multicore version ($C = 1, 2, 4, 8$) is also created for both MCML and static CMOS. C represents the C-slowing factor as well as number of cores. Configurations with the same C are designed with the same amount of computation and memory resources. The per-instruction energy of each type of instructions for CMOS processors is derived from netlist simulation and synthesis results. The power of MCML is calculated as the weighted sum of the measured static power for each type of gate or interface (weighted by gate count). Due to the differential nature of MCML, making a fair area comparison requires place and

route (area in 22nm is scaled from the 45nm results[2]). A "fat-wire" approach [30] is used to estimate the differential routing overhead of MCML (this approach doubles the pitch width of wires, and honors minimal spacing rules).

Task flow	Tools
Cell design & noise simulation	22nm Predictive Technology Model [31], Cadence Virtuoso 6.1.5
RTL & netlist simulation	Verilator 3.841, Cadence NCSim
C-Slow retiming	Customized C-slow retiming script in Perl
Synthesis	Synopsys Design Compiler F-2011.09
Power evaluation	Synopsys PrimeTimePT & CACTI 6.5 [32]
Place & Route	Cadence Encounter 9.1
Architectural simulation	SESC Simulator [33]

TABLE V. STEPS AND TOOL CHAIN USED IN THE EVALUATION.

Components	Static CMOS	MCML (single core)
1C	1 core @667MHz	1-slow @2GHz, 5-stage
2C	2 cores @667MHz	2-slow @3.7GHz, 9-stage
4C	4 cores @667MHz	4-slow @6.8GHz, 17-stage
8C	8 cores @667MHz	8-slow @13GHz, 33-stage
L1 caches (per C)	16KB direct-map I-cache 16KB 4-way D-cache MESI coherent protocol	16KB direct-map I-cache bank 16KB 4-way D-cache bank
L2 cache	8MB, 16-way set associative, 8 banks, 64B line size	
Memory	DDR2-1066 [34], 2 channels, FR-FCFS scheduling	

TABLE VI. EXPERIMENTAL SETUP FOR CMOS AND MCML PROCESSORS.

A. System-Level Noise Modeling

A system-level noise model is employed to mimic the essential components in the noise generation process [35]. It decouples the linear (power and ground network), and nonlinear (transistors) components of the system (Figure 6). The power and ground networks are modeled as resistive networks with parasitic capacitors, and the non-linear switching blocks (*e.g.*, decoders and functional units) are modeled as distributed current generators connected in parallel between power and ground networks. In this paper, current generators are implemented with 32-bit adder circuits, and the current is proportional to the power of the block it represents [35].

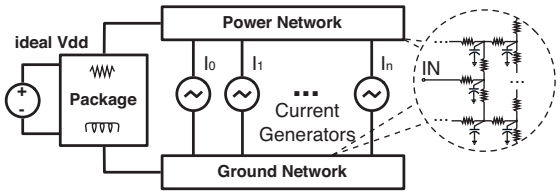

Fig. 6. The model for system-level noise evaluation.

B. Architecture Modeling and Applications

A heavily modified SESC simulator [33] is used to model both static CMOS and MCML processors. A C-slowed ($C = 1, 2, 4, 8$), single-core MCML processor is compared to a C-core static CMOS processor; both support a maximum of $T = 64$ hardware threads. With the same C for a C-slowed processor and a C-core processor, the total cache capacity and per-thread resources are kept the same for a fair comparison (Table VI). A mix of 14 parallel applications from Phoenix [36], SPLASH-2 [37], SPEC OpenMP [38], and NAS [39] benchmark suites, as well as a Batch Self-Organizing Map (BSOM) application [40], are evaluated and average results of them are shown in later sections (Table VII).

[2]FreePDK provides design rules at only 45nm technology, but not 32nm or 22nm.

Benchmarks	Suite	Input
BSOM	NA	census
Histogram	Phoenix	small
Linear Regression	Phoenix	50MB key file
CG	NAS OpenMP	Class A
FT	NAS OpenMP	Class S
MG	NAS OpenMP	Class A
Swim-Omp	SPEC OpenMP	MinneSpec-Large
Equake-Omp	SPEC OpenMP	MinneSpec-Large
Barnes	SPLASH-2	16K Particles
Cholesky	SPLASH-2	tk 15.0
FFT	SPLASH-2	1M points
LU	SPLASH-2	512×512 matrix, 16×16 blocks
Ocean	SPLASH-2	514×514 ocean
Water-NSquared	SPLASH-2	512 molecules

TABLE VII. APPLICATIONS AND DATA SETS.

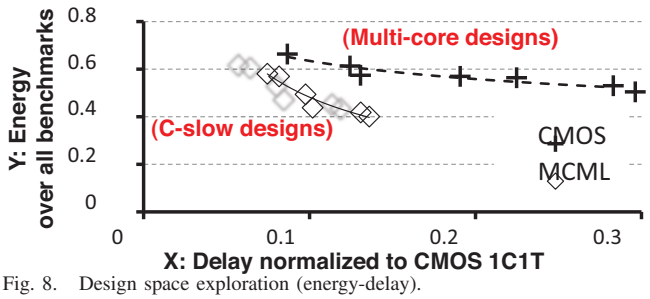

Fig. 8. Design space exploration (energy-delay).

VII. EVALUATION AND DISCUSSION

This section presents a quantitative comparison between MCML and CMOS processors in near-threshold computing.

A. Design Space Exploration

The design space of different architectural choices is explored for MCML and static CMOS in this section. Four parameters are involved: (1) supply voltage ($0.8V$ to $0.55V$), (2) C-slowing factor ($C = 1, 2, 4, 8$), (3) the number of cores ($1, 2, 4, 8$), and (4) the number of hardware threads (T up to 64). Figure 8 hides the non-optimal points and only shows the Pareto curve (a point corresponds to the average energy-delay over all benchmarks for a configuration). For static CMOS, the Pareto boundary includes only the multicore designs, because C-slowed designs require a high frequency that would significantly increase the power consumption. In the case of MCML, the C-slowed designs (rather than multicore designs) are Pareto-optimal, and are presented on the Pareto boundary. This is because the MCML core power is not significantly affected by frequency, and the additional static power due to the larger number of cores for multicore designs is greater than that of increasing the pipeline depth for C-slowed designs.

B. Evaluations on Proposed Architectures

Upon determining the most energy-efficient designs (multi-core for static CMOS, and C-slowed single-core for MCML), a series of studies on the performance, energy, and noise advantages of MCML processors are discussed in this section.

Power, Performance, and Energy. Without the proposed architectural optimizations, the baseline single-core, single-thread processor ($1C1T$ in Figure 7) in MCML outperforms the baseline in static CMOS by three to four times in performance. However, because of the higher static power of MCML, the energy consumption is 20% higher. The performance and energy of MCML and static CMOS processors are further studied by varying C and T. The performance (*i.e.*, $\frac{1}{ExecutionTime}$)

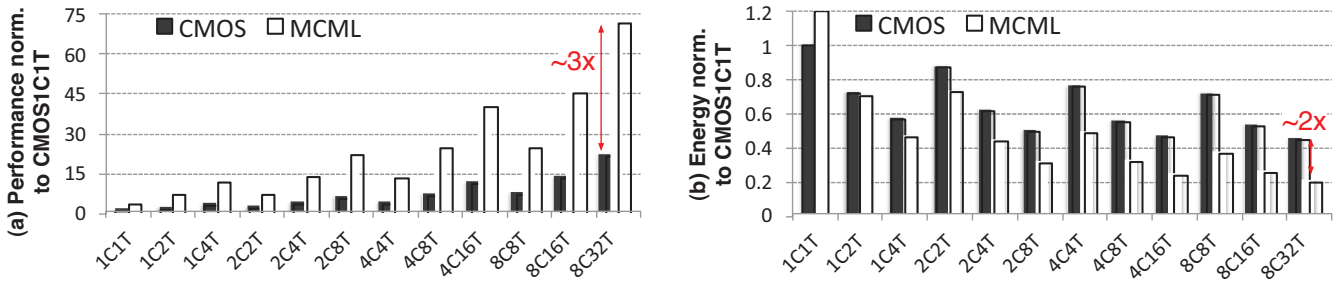

Fig. 7. Comparison of performance (a) and energy (b) when increasing the configured throughput (C respectively represents number of cores for static CMOS, and C-slowing factor for MCML. T is the number of total hardware threads). Numbers are averaged over all 14 benchmarks.

improvement of MCML processors over static CMOS processors is greater than $3\times$ (Figure 7 (a)). As C and T increase, the MCML becomes increasingly energy efficient, ultimately achieving $2\times$ lower energy than the static CMOS multi-core design (Figure 7 (b)). The performance gap mostly comes from the speed difference of different logic styles, and the improved energy efficiency of MCML is due to the effective static power reduction using architectural techniques—notably, C-slow retiming. The best energy efficiency is achieved by designs with the highest throughput for a given C (*e.g.*, $1C4T, 2C8T, 4C16T$ and $8C32T$).

System Energy and Area Breakdown. To better understand the bottlenecks of the system, energy and area breakdowns are analyzed in Figure 9. Static rather than dynamic energy dominates in the MCML core, whereas dynamic energy is more significant in static CMOS (Figure 9 (a)). Memory consumes about 50% of the total system energy, and more than 50% of the total area (Figure 9 (b)) for both MCML and static CMOS processorss. The $1C4T$ MCML processor occupies a larger area than the baseline static CMOS processor due to the differential wiring, relatively larger gates, and the additional interfaces for MCML. As C increases, however, this area gap is reduced; for example, when $C = 8$, the MCML core consumes $1.5\times$ less area than eight static CMOS cores, and the entire system occupies 9% less area than the 8-core static CMOS processor.

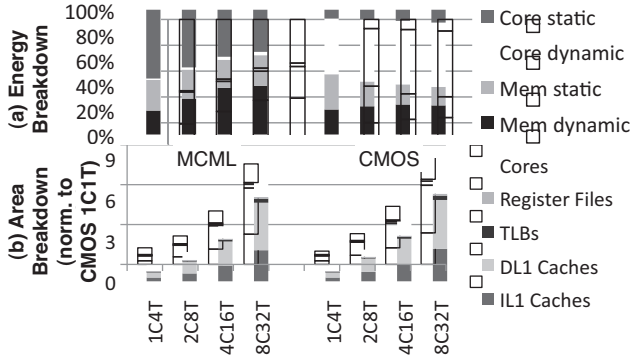

Fig. 9. Energy (a) and area (b) breakdown for the most energy-efficient configurations.

C. Impact of Limited Thread-Level Parallelism

When the number of active software threads is less than the number of hardware contexts, the energy efficiency of the MCML processor can degrade. Figure 10 shows an example of the $1C8T$ configuration for both MCML and static CMOS: MCML exhibits energy savings over static CMOS when more than a half of the hardware contexts $(4/8)$ are used. For $4C16T$ (a more aggressive configuration), the energy break-even point occurs at a quarter of the hardware contexts $(4/16)$. Similar results were observed for other configurations, indicating that a limited thread-level parallelism does not necessarily nullify the energy advantage of MCML (even for a larger C and T), and a more aggressive configuration with a higher hardware utilization results in larger energy savings. This makes C-slowed MCML processors favorable for highly utilized throughput-computing systems, where an abundant of threads can keep the pipeline busy and hide the memory stalls effectively.

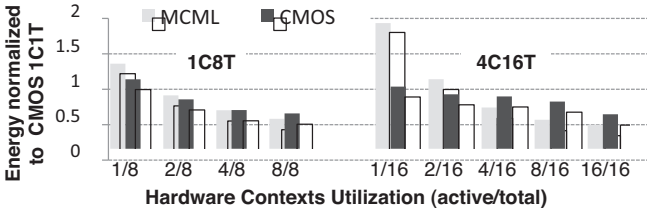

Fig. 10. Energy for static CMOS and MCML processors when hardware contexts are underutilized.

D. Signal Integrity

An MCML processor draws a nearly constant current, and exhibits small voltage fluctuations, which adds potential for further reducing the PDN design margins for better energy efficiency. At the same supply voltage $(0.55V)$, the baseline core in MCML generates $10\times$ lower peak-to-peak voltage noise than in static CMOS (Figure 11) using the system-level noise model in Section VI-A.

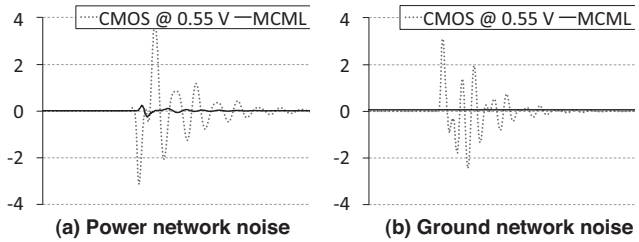

Fig. 11. The voltage noise (percentage of Vdd) over time for the baseline processors in MCML and CMOS.

E. Process Varaibility

Process variations, as well as voltage and temperature fluctuations, are challenges for near-threshold designs in both static CMOS and MCML. The variational effects of MCML circuits can be mitigated by tuning bias voltages with an on-chip voltage generator. Due to the differential nature of these circuits, it is necessary to consider transistor mismatches between the different branches of an MCML circuit. To study the effect of random and systematic variations in threshold voltage and channel length at the system-level, 2000 standard

cell libraries were generated and evaluated via Monte Carlo simulations (distribution parameters see Section III). With these statistical libraries, the baseline processor is synthesized in both static CMOS and MCML. The mean chip frequency for the 8-slowed MCML processor is 12.85 GHz, and the standard deviation is 4% of the mean. The results also show that $8C$ configuration suffers more variational effects than $4C$ and $2C$. MCML suffers less from variational effects on power (16%), but more from variations in critical path delay (43%), compared to static CMOS (25% in power and 24% in delay).

To mitigate the static power of MCML, a high clock rate—13GHz—is assumed with the 8-slowed configuration, which could make the pipeline vulnerable to clock skews. Besides pursuing better clock network designs, such as resonant clock distributions [41], [42], less aggressive configurations can be clustered or power gated [43] to achieve similar benefits (the 4-slowed MCML core provides a significant gain over static CMOS in energy).

VIII. Conclusions

We have studied a 22nm MCML processor from gate, circuit, RTL and architecture levels in the near-threshold region, and the design space of MCML is explored on the system level. By applying appropriate architecture techniques—aggressive C-slow retiming and interleaved multithreading, the static power of MCML can be effectively mitigated, and an 8-slowed single-core MCML processor is able to deliver $3\times$ higher performance, $2\times$ lower energy, and 9% less area than an 8-core static CMOS processor. The best energy efficiency is achieved at designs with the highest throughput, making MCML more favorable for highly utilized throughput-computing systems. Similar to static CMOS at near-threshold voltage levels, an MCML based NTC design faces challenges such as process variations that should be considered in the design process. A system-level $10\times$ voltage noise reduction is observed for MCML, which holds the potential for reducing the complexity of PDN designs and further improving the energy efficiency.

References

[1] R. Dreslinski *et al.*, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[2] X. Zhang *et al.*, "Characterizing and evaluating voltage noise in multi-core near-threshold processors," in *ISLPED*, Sept 2013.

[3] J. Musicer *et al.*, "MOS current mode logic for low power, low noise CORDIC computation in mixed-signal environments," in *ISLPED*, 2000.

[4] I. Savidis, S. Kose, and E. G. Friedman, "Power noise in TSV-based 3-D integrated circuits," *Solid-State Circuits, IEEE Journal*, 2013.

[5] S. Badel, "MOS Current-Mode logic standard cells for high-speed low-noise applications," Ph.D. dissertation, EPFL, 208.

[6] *Cray X1E Supercomputer*, http://www.cray.com/products/x1e/.

[7] A. E. Brown *et al.*, "Improved performance of ibm enterprise system/9000 bipolar logic chips," *IBM J. Res. Dev.*, 1992.

[8] E. Eichelberger and S. Bello, "Differential current switch–high performance at low power," *IBM Journal of Research and Development*, 1991.

[9] J. Kolodzey, "Cray-1 computer technology," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, 1981.

[10] M. Alioto and G. Palumbo, "Power-aware design techniques for nanometer mos current-mode logic gates: a design framework," *Circuits and Systems Magazine, IEEE*, 2006.

[11] M. Mizuno *et al.*, "A GHz MOS adaptive pipeline technique using MOS current-mode logic," *IEEE Journal of Solid-State Circuits*, 1996.

[12] H. Hassan *et al.*, "MOS current mode circuits: Analysis, Design, and Variability," *IEEE Transactions of VLSI*, 2005.

[13] G. Chen *et al.*, "Yield-driven near-threshold sram design," *IEEE Transactions of VLSI*, vol. 18, no. 11, pp. 1590–1598, Nov 2010.

[14] E. Salman and E. G.Friedman, *High Performance Integrated Circuit Design*. McGraw-Hill Professional, 2012.

[15] R. Zhang, K. Skadron, B. H. Meyer, M. R. Stan, and W. Huang, "Some limits of power delivery in the multicore era," 2012.

[16] V. J. Reddi, M. S. Gupta, K. K. Rangan, S. Campanoni, G. Holloway, M. D. Smith, G. yeon Wei, and D. Brooks, "Voltage noise: Why its bad, and what to do about it."

[17] M. D. Powell and T. N. Vijaykumar, "Pipeline damping: A microarchitectural technique to reduce inductive noise in supply voltage," *SIGARCH Comput. Archit. News*, 2003.

[18] V. Reddi *et al.*, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *MICRO 2010*.

[19] J. Cao *et al.*, "Oc-192 transmitter and receiver in standard 0.18um cmos," *Solid-State Circuits, IEEE Journal of*, 2002.

[20] A. Tanabe *et al.*, "0.18mu CMOS 10-Gb/s multiplexer/demultiplexer ICs using current mode logic with tolerance to threshold voltage fluctuation," *Solid-State Circuits, IEEE Journal*, 2001.

[21] M. Houlgate *et al.*, "Adaptable MOS current mode logic for use in a multi-band RF prescaler," in *ISCAS 2004*.

[22] F. C. Leiserson and J.Saxe, "Optimizing synchronous circuitry by retiming," *Third Caltech Conference on VLSI*, May 1993.

[23] N. Weaver *et al.*, "Post-placement C-slow Retiming for the Xilinx Virtex FPGA," in *Proceedings of the 2003 ACM/SIGDA 11th International Symposium on Field Programmable Gate Arrays*, 2003.

[24] S. Sarangi *et al.*, "VARIUS: A model of process variation and resulting timing errors for microarchitects," *Semiconductor Manufacturing, IEEE Transactions on*, 2008.

[25] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers, "Matching properties of mos transistors," *IEEE J. Solid-State Circuits*, 1989.

[26] A. Hartstein *et al.*, "Optimum power/performance pipeline depth," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003.

[27] V. Srinivasan *et al.*, "Optimizing pipelines for power and performance," in *MICRO-35*, 2002.

[28] A. Hartstein *et al.*, "Optimum power/performance pipeline depth," in *MICRO-36, 2003*.

[29] "Floating Point Unit," http://opencores.org/project,fpu.

[30] S. Badel *et al.*, "A Generic Standard Cell Design Methodology for Differential Circuit Styles," in *Design, Automation and Test*, 2008.

[31] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *ISQED*, 2006.

[32] N. Muralimanohar *et al.*, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *MICRO*, 2007.

[33] J. Renau *et al.*, "SESC simulator," Jan 2005.

[34] *512Mb DDR2 SDRAM: MT47H128M4B6-25*, Micron, Mar 2006.

[35] H. Chen and D. Ling, "Power Supply Noise Analysis Methodology for Deep-submicron VLSI Chip Design," in *DAC*, 1997.

[36] R. M. Yoo *et al.*, "Phoenix rebirth: Scalable MapReduce on a large-scale shared-memory system," in *Proceedings of IEEE International Symposium on Workload Characterization*, 2009.

[37] S. C. Woo *et al.*, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA*, 1995.

[38] L. Dagum *et al.*, "OpenMP: An industry-standard API for shared-memory programming," *Computational Science & Engineering*, 1998.

[39] D. H. Bailey *et al.*, "NAS parallel benchmarks," NASA Ames Research Center, Tech. Rep., March 1994, tech. Rep. RNR-94-007.

[40] R. D. Lawrence *et al.*, "A scalable parallel algorithm for self-organizing maps with applications to sparse data mining problems," *Data Mining and Knowledge Discovery*, pp. 171–195, 1999.

[41] J. Rosenfeld and E. G. Friedman, "Design methodology for global resonant h-tree clock distribution networks," *Transaction on VLSI*, 2007.

[42] S. Chan *et al.*, "Uniform-phase uniform-amplitude resonant-load global clock distributions," *Solid-State Circuits, IEEE Journal of*, 2005.

[43] A. Cevrero *et al.*, "Power-Gated MOS Current Mode Logic: A power aware DPA-resistant standard cell library," in *IEEE 48th DAC*, 2011.