PCOUNT: A Power Aware Fetch Policy in Simultaneous Multithreading Processors

Lichen Weng, Gang Quan and Chen Liu
Electrical and Computer Engineering Department
Florida International University
Miami FL 33174
{*lichen.weng,gang.quan,chen.liu*}*@fiu.edu*

***Abstract*—The Simultaneous Multithreading (SMT) architecture improves the resource efficiency via scheduling and executing concurrent threads in the same core. Moreover, fetch policies are proposed to assign priorities in the fetch stage to manage the shared resources. However, power consumption study is omitted in most fetch policies. On the other hand, the power management schemes nowadays are focused on multi-core processors. Given the growing demands to manage the power consumption of processors and the fully shared system resources in SMT environment, it requires detailed research to develop the power management in an SMT processor.**

This paper proposes a power aware fetch policy PCOUNT, which evaluates the power consumption for two categories in SMT: computation resources and memory accessing resources. PCOUNT fetches from the thread with lowest evaluated power consumption in every CPU cycle, in order to reduce overall power consumption. Furthermore, this paper justifies studied fetch polices using power efficiency, which is calculated as evaluated power consumption per unit system throughput. As a result, PCOUNT improves power efficiency over ICOUNT by 26% and over DWarn by 31% on average. Meanwhile, PCOUNT is able to achieve better overall system throughput and average thread improvement than ICOUNT and DWarn.

I. Introduction

The multithreading architecture, especially the SMT, provides the processors with significant parallelism. SMT architecture has been defined as fully shared system resources, i.e., computation resources and memory accessing resources, by several concurrently executing threads in the same core [1]. With SMT employed in the processor, both horizontal waste and vertical waste are minimized [2]. However, resource sharing leads to the uneven distribution among threads as well as performance unfairness, which might not be optimized until the well-designed scheme is carried out. Long-latency load might make the shared resources occupied by some threads inefficiently, and thus is one of the major obstacles towards higher parallelism and better performance [3]. As a result, some instruction fetch policies [2, 4], are designed to allocate the priorities in the fetch stage, expecting to alleviate the impact of long-latency load. However, their results are mainly focused on the system throughput, e.g., Instruction Per Cycle (IPC), while power consumption is omitted in their study.

On the other hand, there is growing demand to control the power consumption. It is less likely that better performance can be pursued efficiently without consideration of such an issue. For example, 48 cores are successfully implemented on a single chip to form the Single-Chip Cloud computer (SCC) [5] by Intel®. Nevertheless, Pentium™ class IA-32 cores introduced in 1994 are employed on SCC in 2010, such that the power consumption can be restrained to 125W at 50°C. New designs ought to take into consideration the power consumption before they are physically validated. Consequently, great importance should be attached to the power consumption of different designs, such that the results are more fair and practical.

The power consumption issue is taken into consideration by power management schemes, e.g., [6] and [7] in the architecture level. They are able to adjust the execution parameters, e.g., process priority, shared time slices and core frequency to guarantee the power management goal. Nevertheless, their research is mainly focused on the multi-core architecture, while little attention is paid to SMT. The simultaneous threads may have different power and thermal characteristics, sharing the same execution resources. The approach to treat them identically may not lead to the optimal solution in terms of power consumption.

The goal of this study is to develop a power aware fetch policy in SMT processors, such that the power consumption of different threads is evaluated independently. Moreover, more resources are allocated to the threads consuming fewer power via increasing their priority in the fetch stage. The proposed fetch policy expects such threads to generate more throughput with lower power consumption. It will further result in the improvement of overall power efficiency, meaning power consumption is reduced with little harm to the performance. The paper is organized as follows: some related work is discussed in Section 2, and the proposed fetch policy is introduced in Section 3. Furthermore, the experimental results are presented in Section 4, while conclusions are drawn in Section 5.

978-1-4577-1221-0/11/$26.00 © 2011 IEEE

II. Related Work

Kumar et al. [6] evaluated process temperature according to utilization. If the temperature resulting from a thread is more than a threshold, the thread's priority is reduced in scheduling; if the overall temperature is too high, clock gating is applied to the whole system. It requires pre-knowledge about the contribution of various resources to the temperature, which limits the generality of their design. Furthermore, the counters specified to every unit and every thread will introduce great hardware overhead. Employing the frequency of different cores, Wang et al. [7] analyzed the relationship via Recursive Least Square regression. Based on the regression model, Dynamic Voltage and Frequency Scaling is employed to manage the power. However, this approach works only for multicore processors, since simultaneous threads in the same core work at the same frequency. DeVyust et al. proposed the scheduling policy for chip multiprocessors with SMT [15]. Their policy pairs threads in multiple cores according to data from performance counters. Nevertheless, their policy is multicore scheduling policy, rather than the fetch policy coordinating threads in the same core.

Fetch policies are designed to handle priorities among threads in SMT. ICOUNT by Tullsen et al. [2] assigns higher priority to the thread that has less instructions in the pipeline. The thread with more in-flight instructions is considered as low-throughput thread. As a result, they are able to utilize more shared resources to improve overall performance. Long-latency load issue is believed as the major inefficient phenomenon in SMT processors [2]. It generally refers to a miss in Last Level Cache (LLC), e.g., L2. Hence, a thread with loads missed in the L2 cache is not favored by STALL [2] policy, which stops the thread with long-latency load from further fetching to improve the overall resource utilization. Because it introduces overhead to stall a thread completely in the pipeline, DCache Warn (DWarn) policy [4] minimizes the overhead by reducing thread priority. Furthermore, the authors argue that L1 data cache (DCache) misses are a clear indicator to L2 data misses, i.e., long-latency load. To respond in advance, it reduces the threads' priority as long as there are L1 DCache misses associated with them. In summary, they work on better performance, but power consumption is rarely studied in their work.

In fact, there are not enough counters to evaluate thread power consumption in real-time. Hence a universal design should not rely on beforehand analysis. The resources consumed by threads are divided into two categories in the SMT architecture: Computation resources (com) and Memory accessing resources (mem) [8], and Cycle Per Instruction (CPI) portions are used to measure thread resource demands by Zhu et al. in [9]. The pipeline and L1 cache belong to computation resources, while lower level memory hierarchy is memory accessing resources. This paper appreciates such an idea in terms of the categories of SMT resources. It offers an abstract view to observe the resource utilization in the SMT architecture. Hence the power consumption can be evaluated in two categories for different threads, instead of employing hardware counters to introduce considerable overhead to the system.

III. Power Aware Fetch Policy PCOUNT

In this section, the power evaluation metric is introduced. Then the power aware fetch policy is proposed employing such evaluated power consumption results.

A. Power Consumption Evaluation

Power consumption is modeled by Wang et al. [7] in multicore processors as:

$$P(t + \Delta t) = P(t) + \sum_i \omega_i \Delta f_i(\Delta t) \tag{1}$$

where $P(t)$ is the overall power consumption, Δf_i is the frequency changes, Δt is the sampling period, and i is the core context. Their model assumes a linear relationship between power and frequency changes. Frequency suggests the core utilization and it is closely related to the power consumption. In SMT processors, however, simultaneous threads share the same frequency, so it is not practical to evaluate thread power consumption using frequency.

Sharing the similar concept, Kumar et al. [6] evaluated the temperature based on the utilization of microarchitectural units:

$$T_{overall} = \omega_{const} + \sum_m \omega_m \frac{u_m}{t_{total}} \tag{2}$$

where ω is the weight from off-line analysis, u_m is the actual utilization in terms of CPU cycles, t_{total} is the total sampling CPU cycles in off-line analysis, m is for different architectural units, and T is the temperature.

Although the simultaneous threads share the same frequency, their utilization of system resources may be different, which leads to various contribution to the power consumption. Therefore, the power consumption of simultaneous threads can be evaluated with respect to the resource utilization. Following above researchers [6, 7], we assume a monotonically increasing relationship between resource utilization and power consumption in this study. It is important to emphasize that we are not focused on a precise model, but rather an effective indicator for the architecture.

Meanwhile, the resources consumed by simultaneous threads are divided into two categories: Computation resources and Memory accessing resources. It is not necessary to analyze and observe all the architectural units exhaustively, which would introduce great hardware overhead to the system, e.g., sensors, storage and computation. As a result,

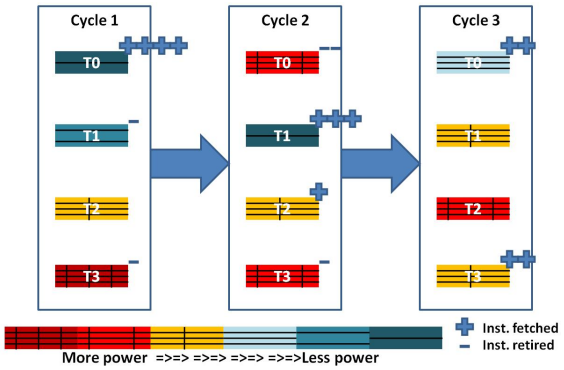

Figure 1. PCOUNT example in an SMT processor

following the concept of Eq. 1 and 2, the power consumption evaluation is proposed as:

$$eP(t+\Delta t)_j = \Delta U_{com,j} + \Delta U_{mem,j} + eP(t)_j \quad (3)$$

where eP is the evaluated power consumption, j is the thread context, U is the utilization and Δ means the changes over Δt. All in-flight instructions are considered as utilization of computation resource, i.e., $U_{com,j}$, while the number of data requests lower than L1 DCache represents the utilization of memory accessing resource, i.e., $U_{mem,j}$. It is defined by [9] and advocated by [2] and [4], etc. Consequently, the power consumption is referred as eP value in this paper.

B. Power Aware Fetch Policy

It is common practice for SMT fetch policy to employ thread's current behavior as an indicator of its performance or resource demand, as illustrated by ICOUNT [2] and DWarn [4]. Following similar practice, the goal of the proposed PCOUNT fetch policy is to improve the overall power efficiency via adjusting priority in the fetch stage. Every cycle each thread is evaluated and prioritized based on its eP value. Thus, PCOUNT minimizes the overall system power consumption by fetching from the thread with lower power consumption, i.e., smaller eP value. In summary, the proposed power aware fetch policy is expressed using pseudo code as:

$eP_j \leftarrow evaluatedPower(U_{com,j}, U_{mem,j})$
$SORT(Thread_j, eP_j)$
while FetchWidth **do**
  $FETCH(Thread_j)$
end while

In order to further explain the proposed policy, an SMT processor is employed as an example, which can fetch four instructions from four threads per cycle. Some legends are used to illustrate the power consumption levels shown in Fig. 1. In Cycle 1, the system is aware of the power consumption of four threads using Eq. 3. Hence, $T0$ has the least power consumption and thus the highest priority in the fetch stage; while $T1$ has the second least and thus the second highest priority, and so on. Then the system fetches from $T0$, and fortunately it is able to fulfill the fetch width. Because some instructions are retired in the first cycle, their utilization values are changed correspondingly.

In Cycle 2 the evaluated power consumption is shown in Figure 1. Now the system should fetch from $T1$, but it does not have four ready instructions due to branch boundary [3]. Thus the system fetches one instruction from $T2$ to guarantee the concept of SMT. Furthermore, considering some instructions retired in the second cycle, the power consumption is as in Cycle 3. The system fetches from $T0$ first to obey the PCOUNT, and then $T1$ or $T3$. When $T1$ and $T3$ have the same priority, the system may fetch from them alternatively. Because thread priority is closely related to the number of instructions fetched, a single thread is less likely to monopolize the system resources. As more instructions are fetched from one thread, its evaluated power consumption is expected to increase, and thus in the next cycle, its priority would be lower than other threads. Hence, thread priority is closely related to the number of instructions fetched, and a single thread is less likely to monopolize the system resources. As a result, the fairness is guaranteed by PCOUNT.

If ICOUNT or DWarn is implemented in the SMT processor in Fig. 1, it would not fetch from the thread with lower power consumption. ICOUNT is supposed to favor high IPC threads [10]. It may fetch from $T1$ in Cycle 1 because it has least instructions in pipeline. It underestimates the power consumed by loads from $T1$, which increases the processor power consumption. On the other hand, DWarn is focused on the memory accessing resources. It may decide to fetch from $T2$ in Cycle 1 in the example because it is the only thread without unsolved L1 DCache miss. However, other instructions from $T2$ are utilizing other resources in the processor, e.g., Integer ALU and ReOrder Buffers.

Regarding the hardware implementation, the proposed PCOUNT policy does not incur significant hardware overhead. Its hardware requirement is similar to the combination of ICOUNT [2] and DWarn [4]. It collects instruction information in pipeline and unsolved L1 DCache misses for all the threads. Then in the fetch stage, such three fetch policies share the same behavior, i.e., to sort a few registers and then fetch correspondingly. In summary, PCOUNT has the overhead at the same level with ICOUNT and DWarn, and obviously less than policies involving many sensors, registers and regression algorithms.

IV. Experimental Results

A. Simulation Methodology

Benchmarks from SPEC CPU2000 [11] are used to form multithreading workloads and simulated on the SuperEscalar

Table I
WORKLOADS IN SIMULATION

Configuration	Category	Benchmark List
2 threads	ILP	*gzip, bzip2*
	MIX	*gzip, twolf*
	MEM	*mcf, twolf*
4 threads	ILP	*gzip, bzip2, crafty, gcc*
	MIX	*gzip, twolf, bzip2, mcf*
	MEM	*mcf, twolf, equake, parser*
6 threads	ILP	*wupwise, apsi, gzip, gcc, crafty, bzip2*
	MIX	*gzip, wupwise, crafty, mcf, twolf, parser*
	MEM	*swim, art, equake, mcf, parser, twolf*

Table II
BASELINE PARAMETERS

Parameter	Value
IF/IR Width	8/9
Instruction Window Size	160 INT, 64 FP
Function Unit	3 Load/Store
	3 INT Mul/Div, 5 FP Mul/Div
ReOrder Buffer Size	320 entries
L1 D-Cache	32KB, 4-way
L1 I-Cache	32KB, 4-way
L2 Cache	512KB, 8-way associative
L1 Cache Hit Latency	2 cycles
L2 Cache Hit Latency	9 cycles
Main Memory Hit Latency	469 cycles

Simulator (SESC) platform [12]. The workloads employed in this study are shown in Table I.

Using the configuration in Table II, the architecture is equipped with SMT ability, i.e., two threads, four threads and six threads. Every thread is simulated for 100 million instructions in the representative region [13]. ICOUNT [2] policy is used as the baseline scheme, and DWarn is used for further comparison.

B. Performance Measurement

The average Instruction Per Cycle ($avgIPC$) used in [11] is one critical method to measure the overall system throughput.The formula of $avgIPC$ is:

$$avgIPC = \frac{\sum_{i=1}^{N} IPC_i}{N} \tag{4}$$

where N is the number of threads.

Nevertheless, in justifying the performance of the new architecture for a multi-threading workload, Sazeides et al. [14] proposed the Average Baseline Weighted IPC ($abwIPC$). It is calculated as the average thread improvement in the new architecture over the old architecture, which is also a good reference to observe the average thread performance. The formula of $abwIPC$ is:

$$abwIPC = \frac{\sum_{i=1}^{N} \frac{IPC_{new,i}}{IPC_{baseline,i}}}{N} \tag{5}$$

Furthermore, we propose power efficiency to justify the power consumption of different fetch polices. First of all, average eP in execution is collected with respect to different

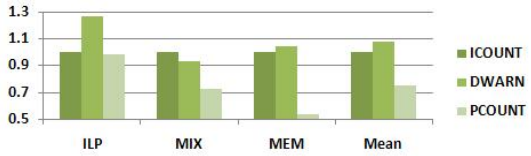

Figure 2. Power per throughput

Figure 3. Power per thread improvement

policies. Secondly, such values are normalized to ICOUNT. Finally, the power efficiency would be the power consumed by unit throughput, i.e., $avgIPC$ or $abwIPC$. Therefore, two metrics suggesting the power efficiency of the policies are defined:

$$avgP = \frac{normalized\ eP}{avgIPC} \tag{6}$$

$$abwP = \frac{normalized\ eP}{abwIPC} \tag{7}$$

Obviously, better power efficiency means lower power consumption per unit throughput, i.e., smaller $avgP$ and $abwP$ values.

C. Power Efficiency

A critical property of PCOUNT is to take power consumption of different threads into consideration. Take the load as an example. A load that hits in the L1 cache is more power-efficient than the load that misses in caches, because the former load finishes its job with minimal access to unnecessary resources. In detail, the latency for an L1 cache hit is 2 cycles, which means it increases eP by one for 2 cycles. The long-latency load will increase eP by one for 469 cycles as the main memory latency. Consequently, such consistency between power consumption and eP value ensures the overall power efficiency. As a result, the power efficiency is improved in PCOUNT by 25.3% over ICOUNT and 30.7% over DWarn in terms of $avgP$; and 26.1% over ICOUNT and 31.4% over DWarn with respect to $abwP$ as shown in Fig. 2 and 3.

Furthermore, it is obvious that long-latency load consumes power significantly more than other instructions. On the other hand, MEM benchmarks have higher cache miss rates than ILP benchmarks [9], such that they are facing long-latency load issue more seriously than others. In the workloads consisting of MEM benchmarks, PCOUNT is able to make the decision according to the evaluated power consumption, such that the power consumed inefficiently is minimized significantly in the MEM workloads. On the contrary, other

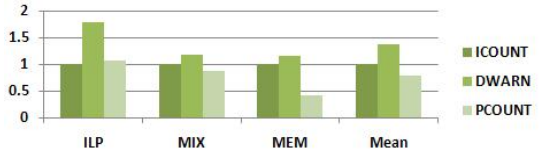

Figure 4. Power per overall throughput 2-threading workloads

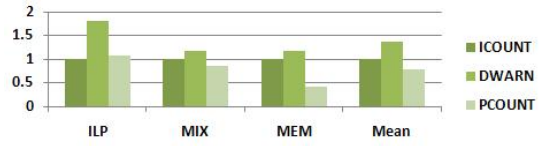

Figure 5. Power per thread improvement in 2-threading workloads

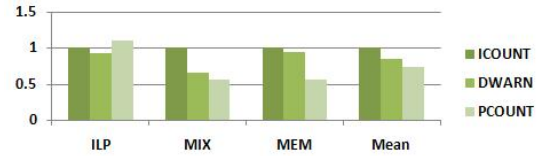

Figure 6. Power per overall throughput in 4-threading workloads

studied policies do not consider power consumption at all, and they favor high performance threads in terms of IPC. Therefore, the power efficiency in those policies is not guaranteed. As a result, in the overall power efficiency improvement, MEM workloads generate the best result with respect to both $avgP$ and $abwP$.

The detailed power efficiency is shown in Fig. 4 – 9. A unique situation observed is ILP workloads do not have best power efficiency in PCOUNT policy all the time. By definition, $avgP$ and $abwP$ are not only dependent on power consumption, but also system throughput. Actually, in the studied ILP workloads, PCOUNT does not necessarily generate satisfying system throughput compared with ICOUNT and DWarn. In PCOUNT policy, some performance is sacrificed for power efficiency in some workloads. However, considering the overall superior power efficiency results and satisfactory throughput discussed later, around 10% reduction in power efficiency than ICOUNT is acceptable in the unique case. Furthermore, as the number of threads increases, the competition for the shared resources becomes severer. Hence it requires a better designed scheme to achieve power efficiency in the system. As a result, PCOUNT is able to completely win in the six-threading workloads in terms of power efficiency.

D. System Throughput

The system throughput results in terms of IPC are presented in this section. Overall, PCOUNT is superior in terms of overall throughput and average thread performance. It improves $avgIPC$ by 12.6% over ICOUNT and 8.6% over

Figure 7. Power per thread improvement in 4-threading workloads

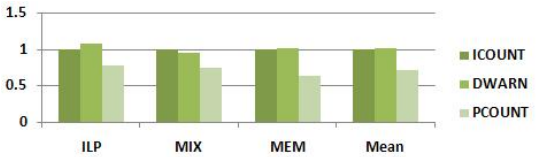

Figure 8. Power per overall throughput in 6-threading workloads

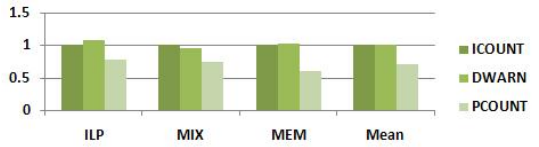

Figure 9. Power per thread improvement in 6-threading workloads

Figure 10. Overall throughput improvement ($avgIPC$) by PCOUNT

DWarn, and $abwIPC$ by 14.1% over ICOUNT and 8.6% over DWarn as shown in Fig. 10 and 11. The improvement in $avgIPC$ is because PCOUNT policy minimizes the long-latency load issue in SMT, such that the system latency is reduced. On the other hand, $abwIPC$ takes fairness into consideration in justifying system throughput. Because it is defined as average thread improvement in the new policy over the old, higher $abwIPC$ is achieved when most threads are improved fairly, rather than a single thread.

However, in some ILP workloads, PCOUNT does not generate best results in terms of system throughput. the main reason is because the nature of ILP workloads. From the perspective view of PCOUNT, they are utilizing more resources and thus consuming more power. Their progress in the pipeline is regulated by PCOUNT for lower power consumption. However, MEM threads are able to compensate such issue to improve the system throughput of PCOUNT. Therefore, the overall throughput and average thread performance by PCOUNT both increases as the workloads switch from ILP to MIX and MEM.

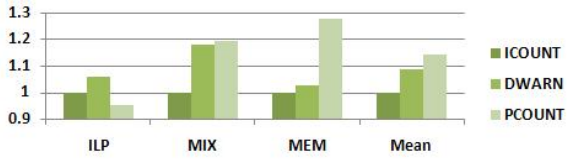

Figure 11. Average thread improvement ($abwIPC$) by PCOUNT

V. Conclusion and Future Work

SMT architectures try to keep the resources fully utilized by the threads in the same core, expecting improved resource efficiency. On the other hand, the demand to manage the power consumption in processors is growing rapidly. However, most power management schemes are focused on the multicore architectures, while the unique property of SMT, i.e., the execution resources are fully shared among simultaneous threads, is more or less ignored by those schemes. Consequently, there do exist the demands to manage the power consumption in the SMT environment.

In this paper, we propose to study the power consumption spent on two categories: Computation resources and Memory accessing resources. By analyzing the utilization of different threads in such resources, the power consumption is evaluated for every thread. In order to validate the design in a straightforward way, the power consumption is calculated as the number of in-flight instructions, i.e., consuming computation resources, plus the number of data requests to memory hierarchy lower than L1, i.e., consuming memory accessing resources. Consequently, the power aware fetch policy, which is called PCOUNT, is proposed to minimize power consumption and thus to improve power efficiency, via fetching from the thread with the fewest evaluated power consumption. As a result, PCOUNT improves power efficiency by 25.3% in terms of $avgP$ and 26.1% in terms of $abwP$ over ICOUNT; while it is better than DWarn by 30.7% in terms of $avgP$ and 31.4% in terms of $abwP$. Meanwhile, PCOUNT is able to perform better in overall system throughput and average thread performance than other studied fetch policies.

Regarding the future work, attention might be paid to the balance between in-flight instructions and the data requests to lower cache system. It is also interesting to develop the proposed policy to thermal aware management.

References

[1] D.M. Tullsen, S.J. Eggers and H.M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism". In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 392–403, June, 1995.

[2] D.M. Tullsen, S.J. Eggers, J.S. Emer, H.M.Levy, J.L. Lo and R.L. Stamm, "Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor". In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp. 191–202, May, 1996.

[3] D.M. Tullsen and J.A. Brown, "Handling long-latency loads in a simultaneous multithreading processor". In *proceedings of the 34th International Symposium on Microarchitecture*, pp. 318–327, December, 2001.

[4] F.J. Cazorla, A. Ramirez, M. Valero and E. Fernández, "D-Cache warn: an I-fetch policy to increase SMT efficienty". In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pp. 74–83, April, 2004.

[5] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, et al., "A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS". In *International Solid-State Circuits Conference*, pp. 108–109, March, 2010.

[6] A. Kumar, L. Shang, L. Peh and N.K. Jha, "System-level dynamic thermal management for high-performance microprocessors". In *IEEE Transactions on Computer-aided Design of Integrated Circuites and Systems*, vol. 27, NO. 1, pp. 96–108, January 2008.

[7] Y. Wang, K. Ma and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation". In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, pp. 304–314, June, 2009.

[8] A. Kagi, J.R. Goodmand and D. Burger, "Memory bandwidth limitations of future microprocessor," In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp. 78–89, May, 1996.

[9] Z. Zhu and Z. Zhang, "A performance comparision of DRAM memory system optimizations for SMT processors". IN *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pp. 213–224, 2005.

[10] A. El-Moursy and D.H. Albonesi, "Front-end policies for improved issue efficiency in SMT processors". In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, pp. 31–40, Feburary, 2003.

[11] J.L. Henning, "SPEC CPU 2000: measuring CPU performance in the new millennium". In *Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[12] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, et al., "SESC Simulator," http://sesc.sourceforge.net, 2005.

[13] E. Perelman, G. Hamerly and B. Calder, "Picking statistically valid and early simulation points". In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, pp. 244–255, October, 2003.

[14] Y. Sazeidex and T. Juan, "How to compare the performance of two SMT microarchitectures". In *The IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 180–183, August, 2002.

[15] M. DeVuyst, R. Kumar and D. M. Tullsen, "Exploiting unbalanced thread scheduling for energy and performance on a CMP of SMT processors," In *The 20th International Parallel and Distributed Processing Symposium*, pp. 10 – 19, April, 2006.