Minimizing the Power Consumption of a Chip Multiprocessor under an Average Throughput Constraint

Mohammad Ghasemazar, Ehsan Pakbaznia, Massoud Pedram
University of Southern California
Department of Electrical Engineering
Los Angeles, CA 90089 U.S.A.
{*ghasemaz,pakbazni,pedram*}*@usc.edu*

ABSTRACT - In a multi-core system, power and performance may be dynamically traded off by utilizing power management (PM). This paper addresses the problem of minimizing the total power consumption of a Chip Multiprocessor (CMP) while maintaining a target average throughput. The proposed solution relies on a hierarchical framework, which employs core consolidation, coarse-grain dynamic voltage and frequency scaling (DVFS), and task assignment at the CMP level and fine-grain DVFS based on closed-loop feedback control at the individual core level. Our experimental results are very favorable showing noticeable average power saving compared to a baseline technique, and demonstrate the high efficacy of the proposed hierarchical PM framework.

KEYWORDS

Chip multiprocessor, Power minimization, Hierarchical power management, Closed-loop control.

1. INTRODUCTION

With the increase in demand for high performance processors, *Chip Multiprocessor* (CMP) architectures have been introduced to enable continued performance scaling in spite of the slow-down of the CMOS technology scaling. At the same time the demand for higher processing power is causing the need for power and energy efficient design of multi-core processing platforms. As technology continues to scale to smaller feature sizes, power dissipation and die temperature have become the main design concerns and key performance limiters in processor design.

The problem of power efficient multiprocessor design has been extensively studied in the literature. Prior studies propose dynamic power/thermal management for homogeneous [1]-[4] or heterogeneous multicore architectures [5][6]. The real-time power management techniques include local responses at the core-level [2][7][8] or global task scheduling heuristics [6][9]-[11]. Typically, the problem formulations target performance optimization under a power/energy budget [1][3] or a thermal constraint [7][12]-[14], or attempt to minimize a composite cost function in the form of energy per throughput [5][12]. Minimization of the total power consumption of a general-purpose CMP system while meeting a total throughput constraint [4][15] is an equally interesting problem, which is the focus of the present paper. Our solution framework solves the power management problem for such a CMP system through concurrent core consolidation, task assignment to cores, and core-level DVFS.

Dynamic Voltage and Frequency Scaling (DVFS) for single processor systems is well understood and standardized [16]. However, due to key differences between single-core and multicore systems, there are a number of options in applying DVFS to CMP platforms [3][17][18]. In particular, DVFS in such systems can be applied in one of two ways: chip-wide [2][3] or per-core [1][18][19]. Moreover, DVFS may be combined with power gating (shutdown) to a portion of the chip. Finally, performance of the CMP system is strongly influenced by the task to core assignment, and thus, DVFS should be combined with (or at least solved in light of) task assignment [9].

In [4], the authors address the problem of finding a chip-wide operating voltage-frequency (*v-f*) setting as well as finding the number of active cores that minimize power consumption of a CMP under a performance constraint. The proposed method uses an offline characterization of the system power and performance for target application and a hill-climbing search method to find the optimal solution, and therefore is costly to be a general purpose runtime power management technique. Reference [15] formulates the problem of minimizing total power consumption of a multi-core system subject to a throughput constraint by means of dynamic voltage scaling and task scheduling, and proves it to be NP-hard. A heuristic is then presented for the case of queued tasks, which is based on performing exhaustive search in the state transition space at each task execution point. The shortcomings of this work include the high complexity of the proposed solution, and the fact that it does not utilize core-shut down as a way of saving power. In [2], the authors deploy a control theory based controller (PI controller) to perform DVFS in CMPs at runtime. Similarly, the limitation of this work is that it does not consider the potential power saving of changing the number of the active cores. In [1], the authors introduce the concept of a global power manager which senses the per-core power and performance of a CMP and sets the operating *power mode* of each core while meeting a target power budget. One of the limitations of this work is also that independent of the amount of the workload that is given to the CMP, the number of the active cores is always fixed. This results in sub-optimal

This research was sponsored in part by a grant from the National Science Foundation under award number CNS-0615437.

power consumption values especially when the CMP workload is low. Also, the premise that each core is dedicated to run a specific application forever limits the practicality of this approach.

In this paper, we address the problem of minimizing the total power consumption of a CMP while maintaining a CMP-level average throughput target for tasks running on the CMP. The minimum power solution is achieved by applying DVFS, core consolidation, and task assignment in the introduced hierarchical global power manager that is comprised of three tiers. The top-tier PM unit performs core consolidation and coarse-grain DVFS based on the information/prediction about the current/future tasks provided by the workload manager unit. The mid-tier PM assigns the tasks, which are assumed to be independent, to available cores considering server and task affinities. The low-tier PM employs a closed-loop feedback implementing DVFS technique at core level which senses a core's performance at periodic intervals and sets the operating frequency level of the core to enforce adherence to known chip level throughput requirements.

The novelties of this paper may be summarized as follows:

- It solves the problem of CMP power optimization under an average throughput constraint by means of core consolidation and closed loop DVFS.
- It proposes to append a workload analyzer to the CMP power management unit to perform coarse grain DVFS depending on the type of task e.g., memory intensive vs. CPU intensive.
- It uses a high level simulation tool that simulates a CMP by emulating core consolidation, task scheduling, task queuing, and DVFS.

The remainder of this paper is organized as follows. In section 2, we provide background on CMP and throughput models used in this paper. Section 3 describes the problem of minimizing the power consumption of a CMP given a throughput constraint. In section 4 we present our heuristic method in detail. Section 5 is dedicated to the experimental results while section 6 concludes the paper.

2. PRELIMINARIES

2.1. System Model

We consider an N-way homogeneous CMP system (a.k.a. multiprocessor system-on-chip, MPSoC). Such a system is composed of *N* homogenous processing cores which are independent except that they share the L2 cache and the interface to the Main Memory [20]. Each core has a separate supply voltage and clock generation module so that they can run at different voltage-frequency (*v-f*) settings. Note that utilizing per-core DVFS [18] makes cores in CMP operate heterogeneously, in terms of their power and performance.

Application programs and/or the operating system generate requests/tasks and send them to the CMP. Similar to [11], we assume these tasks are independent of each other, and each task runs on a single core without the need for inter-core communication. The problem of optimally assigning dependent tasks in multiprocessor systems has been studied in the literature [10][21][22], but is outside the scope of the present paper. Note however that the only modification needed for addressing task dependencies is to utilize a task assignment policy in the task dispatching unit that handles the structure of a task graph; any performance losses due to one task waiting for the results of a predecessor task can thus be accounted for.

General characteristics of tasks, including their *expected job size, s*, and *memory access rate* (MAR) values, are assumed to be known. Note that *MAR* is not a precise micro-architectural performance metric, such as the *cache miss ratio*; instead it denotes an approximation of the number of accesses to memory that cause pipeline stalls and delay penalties. Roughly speaking, *MAR* value indicates if a given task is *CPU-intensive* or *memory-intensive*. This information can be collected from history-based profiling data of the tasks generated by certain applications (see for example published data about characteristics of various tasks generated by E-Commerce, Banking, or Support applications [23]) or by dynamic profiling with the aid of built-in performance monitoring units (e.g. a core's IPS value can be measured on the fly by using the retired instruction count measured by Hardware Performance Counters during a time epoch; for instance, MSR_PERF_FIXED_CTR0 register in Intel Xeon processors reports the number of instructions that retire execution [24]). In either method, estimated values are prone to error which affects the result of decisions made based on these uncertain values, and the power manager must employ a technique to take care of this uncertainty and/or inaccuracy of exact task characteristics; in our approach it is done through using a feedback control loop (cf. section 3).

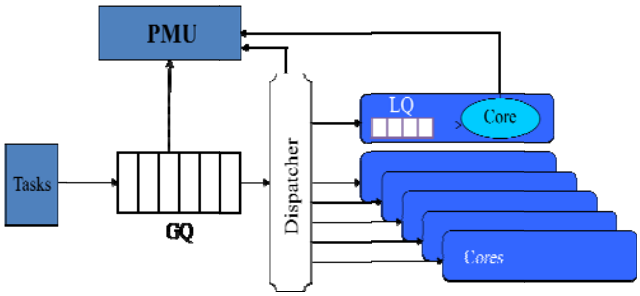

Figure 1. System model with global and local queues.

Figure 1 shows an abstract block diagram of the CMP system considered in this paper. A *Power Management Unit* (PMU), which sets the working *v-f* levels of different CMP cores and provides the *Task Dispatching Unit* (TDU) with the input data needed for task assignment, acts as the global controller for the system. The CMP has a single *Global Queue* (GQ), in which the incoming tasks are held. The TDU assigns the tasks in the GQ to the available cores periodically. Each CMP core has a *Local Queue* (LQ), which is used to hold tasks that are assigned to the core.

The PMU may be implemented through either a centralized hardware unit, such as a separate embedded microcontroller, or as a piece of high-priority software which is being executed on one of the cores. The former realization can

become a bottleneck for the system due to PMU's limited bandwidth for collecting runtime data about cores and the growing overhead of detailed data processing and decision making as the number of cores goes up. The latter realization helps with the scalability of the PM framework with respect the number of cores in the system. In addition in our proposed hierarchical framework, the top tiers of PM perform a quick (low overhead) global data processing and decision making at the system level, whereas the low-tier PM performs detailed decisions at the core level.

The disparate applications are assigned to the cores by the TDU, which is a part of the OS code. Depending on the size of the CMP, i.e., number of cores in the system, the TDU can be realized in a centralized or distributed manner. In this paper, we assume a centralized TDU implementation. The GQ is typically implemented in software as part of the OS kernel while the LQ's are implemented as part of the local power management codes that run on the individual cores.

2.2. Throughput Model

Throughput of a processor core is defined as the average number of executed instructions per second and is denoted by instructions per second or *IPS* for short. If a core that is running at frequency f executes task j with known characteristics, then the time t_0 needed to run I_0 instructions can be estimated by equation (1), in which the first term represents the computation time and the second term accounts for the delay of accessing higher level caches.

$$t_0 = \frac{I_0}{IPC_j^{ncm} \cdot f} + \alpha_c \cdot CMF_j \cdot I_0 \qquad (1)$$

where CMF_j denotes *cache miss frequency*, i.e. the proportion of instructions that cause an L1 cache miss while executing task j; α_c is a fixed parameter representing average cache miss penalty which captures the core's expected stall time when a cache miss occurs. The value of α_c depends on parameters such as the pipeline implementation, cache size, cache management policy, and speed of the L2 cache and main memory. IPC_j^{ncm} denotes the *no-cache-miss instruction per cycle* of the task; it is defined as the IPC value under a condition that there are no cache misses, e.g. very large cache that has all the application data pre-fetched, and thus no misses occur.

Recall *CMF* is a micro-architecture level parameter that indicates the number of memory accesses of a task missing in the L1 cache. In fact, it can be interpreted as a translation of high level *MAR* in the architecture level; in general, a CPU-intensive task, i.e. low *MAR*, has a low *CMF* value while a memory-intensive task, i.e. high *MAR*, exhibits a high *CMF* (although, a memory-intensive task may have a low *CMF* due to its special memory access pattern). Here, we use *CMF* and *MAR* interchangeably to distinct the memory-intensive and CPU-intensive tasks. Also, note that CMF_j in (3) represents average cache miss frequency due to both instruction and data cache misses (denoted by CMF_j^{inst} and CMF_j^{data} respectively):

$$CMF_j = CMF_j^{inst} + \pi_d \cdot CMF_j^{data} \qquad (2)$$

where π_d is the fraction of instructions accessing data memory (typical value between 0.1 and 0.6).

Referring to the definition of throughput, throughput of the core i is calculated as follows using (3):

$$IPS_i(f) = IPC_j(f) \cdot f$$
$$IPC_j(f) = \frac{IPC_j^{ncm}}{1 + \alpha_c \cdot IPC_j^{ncm} \cdot CMF_j \cdot f} \qquad (3)$$

where $IPC_j(f)$ denotes the *actual* IPC value of the task running on the core.

2.3. IPS Saturation Effect

Figure 2 shows the relationship between IPS and frequency as captured in equation (3) for different types of tasks. Figure 2-a corresponds to three low-*CMF* tasks with high, medium and low IPC^{ncm} values, while Figure 2-b shows three high-*CMF* tasks with high, medium and low IPC^{ncm} values.

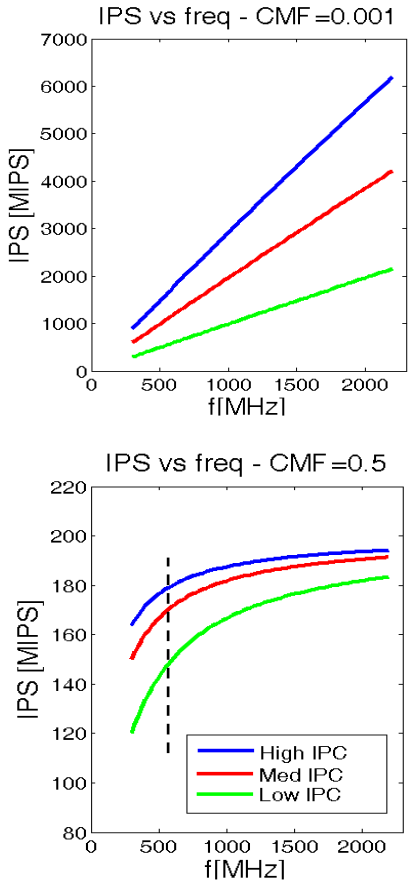

Figure 2. Throughput-frequency relationship for (a) low CMF tasks (b) high CMF tasks.

From the Figure 2-b, domain of the IPS function of high CMF tasks can be divided into two regions: a frequency region where IPS rises rapidly with an increase in f and another where rate of change of IPS with f is low. We define a *unit-slope frequency* separating these two regions:

$$f_{unsl} = f_{\left|\frac{\partial IPS}{\partial f}=1\right.} \qquad (4)$$

where $\frac{\partial IPS}{\partial f}$ is the partial derivative of the IPS with respect to frequency (normalized appropriately to produce a unit-slope

value). For example in Figure 2-b, f_{unsl} for high *CMF* and high IPC^{ncm} tasks is about 710MHz, which is illustrated by a dashed line. For different combinations of IPC^{ncm} and *CMF*, the unit-slope frequency may be calculated for the corresponding task type. In practice, there is uncertainty about the predictive values of IPC^{ncm} and *CMF* of an incoming task, and hence f_{unsl} cannot be calculated accurately for a task in future. In practice, a single average f_{unsl} is assumed for all memory intensive tasks to lower the complexity. Note that if a task has a high *CMF* value, much of the time the core is waiting idle for the memory response, and hence, clock frequency can be set to a relatively low value, to reduce power/energy with no or very little performance loss. Therefore to reduce the runtime of the consolidation and coarse-grain DVFS steps, we will limit the clock frequency of a core running specific type of task to frequencies below the f_{unsl} (cf. section 4.1).

3. Problem Statement

Consider an N-way CMP as described in section 2.1. The PMU seeks to minimize the total power consumption of the CMP subject to achieving a service rate whereby a GQ overflow does not occur. This means that on average, CMP service rate must be greater than or equal to the rate of the incoming tasks. This is equivalent to imposing a lower bound constraint on the average throughput of the CMP.

The problem statement can be written as follows:

$$Min\{P_{CMP}\}\ s.t.\ \mu \geq \lambda \qquad (5)$$

where P_{CMP} denotes the CMP power (see equation (7)), λ is the rate of the incoming tasks (arrival rate of the tasks in the GQ), and μ is the CMP service rate (departure rate of the tasks from the GQ). To solve this problem, the power management algorithm needs to decide on the optimum number of the processing cores that are required to service the tasks, determine the *v-f* setting of each active core, and assign and schedule the tasks in the GQ to different cores. Moreover, the predictive input information of the system, such as the task characteristics (as described in section 2.1) are prone to uncertainty and inaccuracy, and a mechanism needs to be adopted to opt out the effect of inaccurate data. Due to the real time nature of the problem, conventional mathematical optimization approaches do not result in a robust solution to this problem. We want to utilize an efficient (light and thin) and robust algorithm to solve it..

To estimate the power consumption of the CMP, we use a power model which is the summation of the intra-core power dissipation and the CMP-level power contribution of the core. The intra-core power dissipation is comprised of a dynamic power which is cubically dependent on the core's clock frequency (assuming that the frequency f is directly proportional to the core's supply voltage level V) and an v-f setting dependent idle component, $P_{idle}(f)$. The second component of CMP's core power dissipation is $P_{common,chip}$ (also denoted by P_C) which is comprised of power consumption of the shared resources in the CMP system, most importantly the L2 cache and I/O interface. This power component is independent of the frequency of any core.

$$P_{core,intra} = Q_D \cdot f^3 + P_{idle}(f)$$
$$P_{common,chip} = P_{L2} + P_{I/O} \qquad (6)$$

where Q_D in the $P_{core,intra}$ expression is a constant (implementation and CMP platform-dependent) term while $P_{idle}(f)$ is the idle power consumption for each core which is a function of the frequency. $P_{idle}(f)$ at different frequencies, f values, can be measured offline and the values can be kept in a lookup table. P_{L2} and $P_{I/O}$ –that are frequency independent - denote constant terms capturing the power dissipation of the L2 cache and I/O interface of the CMP. We have:

$$P_{CMP} = \sum_{i=1}^{M} active(i) \cdot P_{core,intra}(i) + P_{common,chip} \qquad (7)$$

where $active(i)$ is a pseudo-Boolean variable set to 1 exactly if the i^{th} core is active. In this model, it is assumed that at least one core is active in the CMP, executing arrived tasks and the PMU application.

4. Proposed Solution

We introduce an efficient strategy that solves the policy optimization problem described above. The proposed solution relies on a 3-tier hierarchical DPM approach (that we call ***3T-PM***) where the original problem is broken into three optimization problems based on the significance and the granularity level of the decisions that must be made. Higher level DPM sets values of the input parameters of the lower levels. Decisions at the top level are made based on coarse-grain information about the target task set (e.g., predicted *MAR* value for tasks in the GQ) whereas the lower level decisions are made based on characteristics of individual tasks.

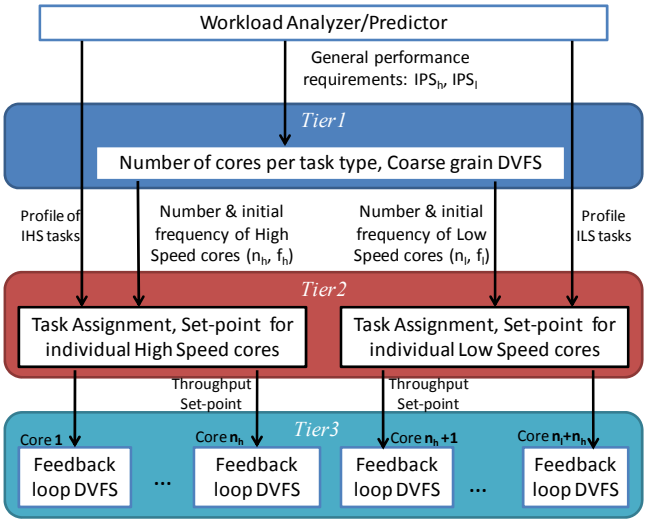

Figure 3. Block diagram of the proposed three-tiered PM.

Figure 3 shows the block diagram of the proposed hierarchical PMU. The PMU attempts to minimize the CMP power consumption while ensuring that the CMP throughput is higher than a minimum threshold value. This is done by:

a) Choosing the optimum number of the cores required to maintain the required throughput and turning the rest of the cores off (see tier 1 in Figure 3);

b) Dividing the total active cores in two groups: high speed and low speed cores where the target working frequencies for high speed and low speed cores are set (we call this optimization *core consolidation and coarse-grain DVFS*, see tier 1 in Figure 3);
c) Assigning tasks from the GQ into the LQ of different active cores (this *task assignment* step is done separately for high and low speed cores, see tier 2 in Figure 3);
d) Setting the target average throughput value (so-called "set point") for each core considering the task assignments, such that the system-level throughput constraint –in the form of task processing rate- is satisfied (see tier 2 in Figure 3);
e) Dynamically tuning voltage-frequency level of each active core by using a local control feedback loop for each core (we call this step *fine-grain DVFS*, see tier 3 in Figure 3).

Decisions at each tier of the PM hierarchy are performed regularly, but with different frequencies. Tier 1 decisions are made at each *decision epoch*, T_d. Task assignment is done as part of the second tier optimization at each *allocation window* T_a, where $T_a < T_d$. The third tier decision making is done with period of T_s, which denotes the *sampling period* of the digital feedback control loop of each core. Typically $T_s < T_a$, such that the lower level controller iterates for enough sampling periods and becomes stable within the T_a period. This means the stability of two tiers are independent as long as they are operating according to the specification. Furthermore, note that the hierarchical structure of the solution implies that a higher level PM makes a decision that sets the target (aspiration level) for lower levels, and decisions lower levels only satisfy these targets i.e. they cannot damage higher level decisions, as long as target points are feasible.

4.1. Workload Analyzer

The task of Workload Analyzer (WA) is monitoring the incoming tasks at the GQ to (i) classify them based on their IPC characteristics, and (ii) predict the future workload both in terms of its arrival rate and its *IPC* characteristics. The decision about the amount of workload that needs to be processed at each decision epoch is also made at this time. This decision is made so that, on average, queue overflow is avoided. The WA aims to keep the average queue occupancy of the GQ at a constant level, which of course implies that the service rate μ matches the demand rate λ. In fact, if this condition is held, CMP has supplied just enough performance to satisfy the throughput requirement of the system and save power as much as possible.

4.1.1. Task Classification

As mentioned earlier, *MAR* indicate if the task is a *CPU-intensive* or *memory-intensive* task. Two classes of tasks are defined based on their *MAR* values on the given cores: *Intrinsically Low Speed* (ILS, or *l* for short) and *Intrinsically High Speed* (IHS, or *h* for short) tasks. Task classification is done based on the value of the task *MAR*, i.e.,

$$C(task) = \begin{cases} C_l & MAR_{task} \geq MAR_{th} \\ C_h & otherwise \end{cases} \quad (8)$$

where $C(task)$ is an enumerated type describing the class of the task and MAR_{th} is a threshold value used to partition the tasks. When the apriori information about a task is not available, WA assigns it to default class ILS, which allows the task to run more power efficiently. Meanwhile, the WA monitors and records its *MAR* for later reference. The *MAR* values of tasks are recorded in a table, with least recently used (LRU) replacement policy to limit the table size.

4.1.2. Workload Analysis and Prediction

The WA monitors and predicts the required throughput for each task set, IPS_h and IPS_l, and the average characteristics of tasks, e.g. IPC_{avg}, to provide to the tier-one PM in order to manage the core consolidation and coarse-grain DVFS choices at each decision epoch. The prediction method used can be a history-based prediction technique, whereby a moving-window average of the task arrival rates and their *IPC* values over the last few decision epochs is used as estimates of the task arrival rate and *IPC* value in the next decision epoch.

Next, based on the current state of the GQ and prediction about task arrival rate, the WA determines the number of tasks, *W*, in the GQ to be dispatched to cores in each allocation window. The WA sets *W* such that the occupancy level of the GQ remains nearly constant at some target level, e.g. 50% (c.f. [18] for detailed analysis). This value is found to be energy efficient for a single processor system, however, the CMP can also be seen as a processor that is N-times faster, and the incoming task rate is thus N times higher too.

The WA creates the ILS and IHS task sets running during the decision period and calculates required throughput for each set:

$$IPS_C = \frac{\sum_{\text{task } j \in C} s_j}{T_d} \quad (9)$$

where T_d denotes the duration of the decision period and s_j denotes the expected job size of task *j*.

4.2. Tier-One PM

The job of Tier-One PM includes first finding the optimum number of cores to run each class of tasks so as to minimize P_{CMP}. Then, it must assign a single target voltage and frequency level to all the cores that are assigned to one class (the *v-f* setting would be fine tuned by tier-three.)

4.2.1. Core Consolidation and Coarse-Grain DVFS

Armed with the task classification, the PMU allocates the optimum number of cores to each class of tasks, and sets the coarse-grain frequency (and hence, the supply voltage level) of each core. The objective is to minimize the total power consumption while satisfying the throughput constraint for the task set in each class. Let n_l, n_h and *N* denote the number of cores assigned to the ILS and IHS tasks and the CMP core count. Tier-one power minimization problem can be formulated as follows:

$$Min\ P_{CMP} = \left(n_l \cdot f_l^3 + n_h \cdot f_h^3\right) Q_D + P_{common,chip} + \left(n_l \cdot P_{idle}(f_l) + n_h \cdot P_{idle}(f_h)\right) \quad (10)$$

$$s.t. \qquad (11)$$

$$\begin{cases} n_l, n_h \geq 0, n_l + n_h \leq N \\ f_{min} \leq f_l \leq f_{unsl} \\ f_{min} \leq f_h \leq f_{max} \\ n_l \cdot IPC_{l,avg}(f_l) \cdot f_l \geq IPS_l \\ n_h \cdot IPC_{h,avg}(f_h) \cdot f_h \geq IPS_h \end{cases}$$

where f_l and f_h are the coarse-grain working frequencies for the ILS and IHS cores, respectively. These two frequencies together with the number of cores, n_l and n_h, assigned to each task class are the optimization variables to be determined. The first constraint limits the number of cores. Under the low workload conditions, it may be prudent from a power-saving perspective to turn off some of cores -this is why the summation of the two types of cores can be less than *N*. The second and third constraints bound f_l and f_h in the ranges (f_{min}, f_{unsl}) and (f_{min}, f_{max}). The last two constraints are throughput constraints for each task class. Here $IPC_{C,avg}$ denotes the average actual *IPC* values of all current tasks in the corresponding class, *C*, assumed to be equal to measured *IPC* value in the recent past by hardware performance counters [25].

Notice also that IPS_l and IPS_h have already been determined by the WA from equation (9). This is a *Non Linear Integer Programming* problem. Fortunately, since the range of independent variables is small (few available frequency levels and a limited number of cores on the chip,) a Branch and Bound search method, as described below, is attractive and computationally feasible. On line 12, the algorithm searches for the best variable values (n_l, n_h, f_l, f_h) that minimize the power dissipation.

```
1.  S = {};
2.  for (m = 0 to N; m++) do
3.    for (f_l = f_min to f_unsl; f_step++) do
4.      n_l=⌈IPS_l/(IPC_{l,avg} · f_l)⌉;      // from the 4th constraint
5.      n_h = m – n_l;                        // from the 1st constraint
6.      f_h = IPS_h/(n_h · IPC_{h,avg})       // from the 5th constraint
7.      calculate P_CMP from (10) ;
8.      s = (n_l, n_h, f_l, f_h, P_CMP);
9.      S = S ∪ {s};
10.   end for
11. end for
12. s_min = find_min (S);
13. return s_min;
```

It can be shown that the complexity of our proposed algorithm is O(N*F) due to two nested loops of lines 2 and 3, where *F* is the number of frequency steps between f_{min} and f_{unsl}.

4.3. Tier-Two PM

After classifying tasks into IHS and ILS, and deciding about the number of high and low speed cores and their corresponding coarse-grain *v-f* settings in the top-level PM, *TDU* now assign the tasks to individual cores. It also determines the target throughput for individual cores in high speed and low speed categories.

4.3.1. Task Assignment

The task assignment scheme is shown in Figure 4. The tasks in the GQ are passed through a switch where ILS and IHS tasks are distinguished from each other and will be sent to the corresponding Round Robin (RR) switches. Each RR switch assigns its input tasks to LQ of an available core using the round robin scheduling technique [11]. At each instance of time, both RR switches have a list of the current available cores. We define an available core as an active with Queue Occupancy (QO) level less than a threshold value. If the next core in the list of RR switch is not available, the RR switch simply ignores that core and looks for the next available core.

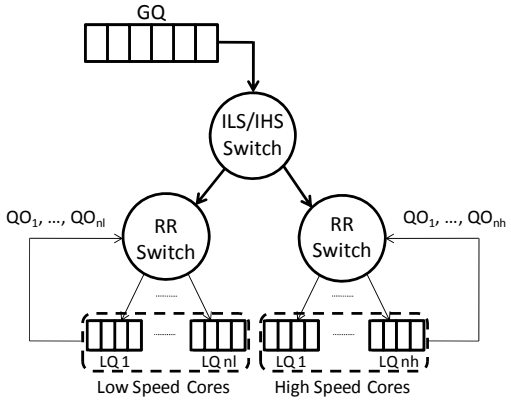

Figure 4. Tier-2 task assignment scheme

4.3.2. Determining Target Throughput of Cores

Once tasks are assigned to cores, based on the set of tasks that are assigned to each individual core, the mid-level PM calculates the target throughput that must be used as the set point in the feedback loop controller of the core (tier three PM). The target throughput of a core, IPS_{target}, is equal to the sum of the expected number of instructions in the assigned tasks divided by the allocation period length. The calculation for each core uses a similar equation as equation (9) except that the task set is restricted to the tasks assigned to that core, and T_d is replaced by T_a. That is:

$$IPS_{target} = \frac{\sum_{\text{task } j \in C} s_j}{T_a} \qquad (12)$$

Note that if the execution time of a task exceeds T_a, it is not feasible to execute the complete task in a single allocation period, and the corresponding core must continue running such task for the next period. However, in order to calculate the target throughput value for the core that is running this task (with large expected execution time), the task is virtually divided into two or more subtasks to be executed in subsequent allocation periods. Therefore, only the portion of task that is executed during each T_a period is considered in the target throughput calculation of the core for that period.

4.4. Tier-Three PM

To maintain a target throughput, IPS_{target}, for each core, we use the feedback control theory [26]. More precisely, we model a processor core as a system, called G_s, whose input vector is the *v-f* settings and whose output is the resulting throughput of the core, *IPS*, as shown in Figure 5. The

controller, shown by G_c in the figure, assigns a *v-f* setting for the core. The system then employs this *v-f* setting, and the resulting throughput is measured by means of the built-in performance monitoring units (a core's IPS value can be measured on the fly by using the retired instruction count measured by Hardware Performance Counters [24] in a time interval). If the measured throughput is less than the target throughput, the controller will increase the *v-f* setting value, which results in higher throughput. On the other hand, if the measured throughput is greater than the target throughput, the controller will reduce the *v-f* setting value to match the required throughput. This technique reduces power consumption by performing DVFS to deliver only the required throughput.

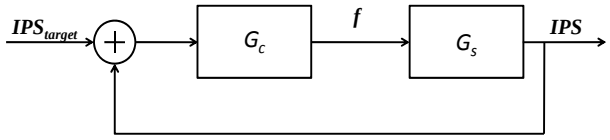

Figure 5. Closed loop system representation.

We can model the throughput of a core, given by (3), as a linear function of its frequency, i.e., the input-output relationship of the system, G_s, can be represented with a linear function. Consequently, we can apply the linear control techniques which are simple, effective, and accurate enough for our purpose. Recall the controller needs to be embedded in the PMU, and hence complex implementations are to be avoided. There are many options for the type of linear controller here. In this paper, we use a *Proportional Integral* (PI) controller [26]. A PI controller is a special case of *Proportional Integral Derivative* (PID) controllers that are very easy to implement, and usually easy to design for a first-order system [2][8]. The derivative component of the general PID controllers may amplify the effect of noise, and thus it is not used in this work. To design the PI controller, we follow the well established control theory techniques [26].

To use the linear control techniques, we first linearize the relationship between throughput of a core and its frequency around a frequency f_0, by replacing $IPC(f)$ in (3) with its maximum value at f_0 as a fixed value (using the maximum value is to guarantee stability of the closed loop system):

$$IPS(f) = IPC(f_0) \cdot f \tag{13}$$

where $IPC(f_0)$ is defined by (14), if the set of tasks in the local queue have an average expected IPC of IPC_{avg}^{ncm} and average *CMF* of CMF_{avg}. The value of $IPC(f_0)$ is approximated at design time for worst case.

$$IPC(f_0) = \max\left(\frac{IPC_{avg}^{ncm}}{1 + \alpha_c \cdot IPC_{avg}^{ncm} \cdot CMF_{avg} \cdot f_0}\right) \tag{14}$$

The transfer function of a PI controller in the z-domain is:

$$G_c(z) = K_p + K_I \frac{z}{z-1} \tag{15}$$

where K_p and K_I are coefficients to be determined based on the desired characteristics of the closed loop system. Hence the transfer function of the closed-loop control system may be written as:

$$G(z) = \frac{G_c(z)G_s(z)}{1 + G_c(z)G_s(z)} \tag{16}$$

with a corresponding characteristic equation [26] as follows:

$$z^2 + \big((K_I + K_P)IPC_{avg} - 2\big)z + 1 - K_P \cdot IPC_{avg} = 0 \tag{17}$$

The solutions to the above equation are the closed-loop poles of system, whose placement in the z-plane determines the main characteristics of the system, such as its steady state error, response time, overshoot and stability. To guarantee stability of the loop, the poles should be placed inside the Unit Circle of z-plane in the Root Locus of system [26]. For our problem, the best placement of poles is found to be at $0.5 \pm 0.1i$ to generate a relatively fast and low-overshoot step response as shown in Figure 6.

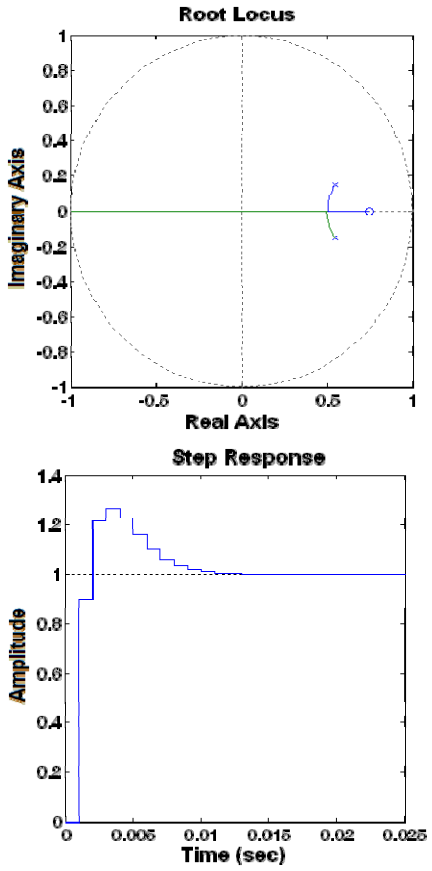

Figure 6. (a) Root Locus and (b) Step Response of the places poles.

5. EXPERIMENTAL RESULTS

We have developed a real-time simulator in C++ to implement and evaluate the proposed power management technique. The simulator uses an N-way CMP with shared L2 cache. The simulator is an event driven simulator, in which the triggering events are task arrival, task departure, decision, allocation, and sampling points. It emulates execution of the tasks on the cores based on their size, IPC^{ncm} and *CMF;* however, PMU decisions are made only by knowing size and *MAR* of tasks, and estimating $IPC_{C,avg}$'s on-line. The PMU, GQ, LQ's, and TDU are implemented in software. Configuration of the cores is as described in Table 1, which is

based on the configuration of UltraSPARC T2 (Niagra2) processor [27]. The dynamic and idle power consumption of CMP are modeled as presented in equation (6) with the coefficients matching the target processor's power characteristics. The proposed PM technique shown in Figure 3 (called **3T-PM**) was implemented, as well as another baseline PM algorithm. For the purpose of comparison, no prior work is found that tackles the same power management problem that we have solved. For example, reference [1] that is the closest to our problem, ignores core consolidation and also is based on life-time fixed task to core assignment. References [2][3] are also lacking dynamic task to core assignment phase. Therefore, a direct comparison with a specific prior work is not possible. However, to evaluate 3T-PM's performance, we compare it to a baseline PM which can be seen as a modified version of the work presented in [1]. The baseline PM does not support core consolidation; neither does it classify the tasks into IHS and ILS. Round robin is used for task assignment in the baseline PM. Also, to study the efficiency of control-theory feedback loop, the baseline PM comes with per-core open loop DVFS capability, which indeed is different from non-control-theory closed loop DVFS of [1]. In order to realize DVFS, the baseline PM utilizes information about tasks to determine the required frequency that satisfies system throughput, and uses a higher core frequency value, as a safety margin, to take into account the uncertainty of those values.

5.1. Task Generation

Tasks are randomly generated and sent to the CMP. The expected job size and inter-arrival time of tasks are assumed to be two independent random variables with exponential distributions with mean values of E(s) and 1/E(λ), respectively. Note that in order to avoid overflow, the average of task arrival rate is set to less than or equal to the maximum processing capacity of CMP. In other words, the mean value of task inter-arrival time is greater than or equal to the expected execution time (mean of expected job size, divided by product of the average IPC of tasks and the maximum core frequency) divided by *N*, total number of cores. Each incoming task is assumed to be generated by one of the nine applications (benchmarks) given in Table 2. The *MAR* values of tasks are assigned based on *MAR* of SPEC2000 benchmarks [28]. The detailed micro-architecture task characteristics used to emulate task execution in our simulator, i.e. IPC^{ncm} and *CMF* are extracted by SimpleScalar simulations and shown in Table 2. In order to choose the parent application of each incoming task, we use a discrete uniform random variable that selects from the nine benchmarks given in Table 2 with equal probability, p=1/9. This means that the characteristic values for incoming tasks are picked from the values given in Table 2 with equal probability of p=1/9. In order to model the uncertainty of the information about the tasks, we apply a ±20% uniform disturbance to the values of task characteristics at runtime, before issuing the task to CMP.

Table 1. Configurations of the cores in CMP system

Pipeline stages	8 (int), 12 (fp)
Execution units	2 INT units, 1 FP unit
Issue queue size	20
Load/Store queue	32/32
L1 instruction/data cache	16KB, 8-way/8KB, 4-way/LRU
L2 unified cache	N*512KB, 16-way, 64B line
Technology node/Vdd	65nm, 1.5V
Frequency	{200:200+:1600} MHz
Typical Dynamic Power	8.9W @ f_{max}
Typical Leakage Power	2.9W

Table 2. Average characteristics of benchmarks used to generate tasks

Benchmark	MAR	IPC^{ncm}	CMF
Art	16%	0.816	0.0488
Bzip	18%	0.902	0.0685
Equake	7%	1.850	0.0065
Gcc	14%	0.876	0.0392
Go	9%	0.773	0.0188
Gzip	26%	0.869	0.0865
Mcf	23%	2.221	0.0629
Mesa	13%	1.923	0.0272
Twolf	8%	1.205	0.0172

Table 3 summarizes the parameters used in the simulation.

Table 3. Simulation parameters

Number of Cores	**N= {4, 8, 16}**
*f*unsl	800MHz
LQ / GQ	8 / 40,80,160
T_d / T_a / T_s	50 / 10 / 2ms
MAR_{th}	%15
Q_d	60%
E(s)	1 M instructions
E(λ)	1.0N, 1.9N [1/ms]
Number of simulated task	5000

5.2. Results

For the purpose of comparison, we compare the proposed 3T-PM algorithm to the baseline power management algorithm described earlier in this section. It does not support core consolidation, task classification, or control-theory feedback loop. Figure 7 shows the average power consumption of the CMP system under the baseline solution and the 3T-PM solution. The experiments were done for three different CMP configurations with *N*=4, *N*=8, and *N*=16 cores, and under two system throughput constraints, low and high corresponding to 30% and 80% of the maximum processing capacity of CMP, respectively. On average, 3T-PM consumes **23**% less power compared to baseline PM.

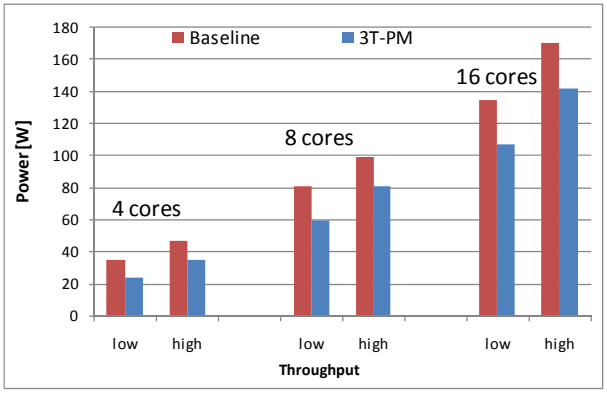

Figure 7. Power consumption of 3T-PM solution vs. baseline PM for three CMP configurations and arrival rates.

Figure 8 depicts waveforms of frequency set by DVFS method of 3T-PM and the baseline PM for one core to compare the effect of PI controller-based DVFS with the open loop DVFS. The throughput constraint for both systems is the same and is shown in the figure with blue color, and none of the PM techniques violate the throughput constraint. It can be seen from the figure that the core frequency level used by 3T-PM is (on average) around **7%** lower than the frequency level used by the baseline technique. Note that in this example, 3T-PM is about **17%** more power efficient compared to the baseline system.

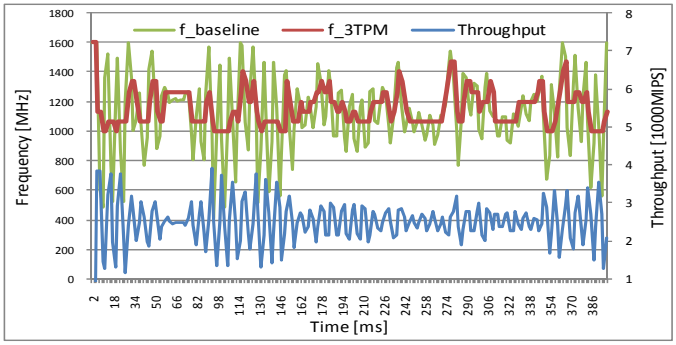

Figure 8. Frequency waveforms used by 3T-PM and baseline PM for the same throughput constraint.

In addition to power minimization, our PM performs better in terms of performance compared to an improved version of the baseline which now employs closed loop DVFS as explained in section 4.4. In particular, we considered very high task arrival rates that pushed the CMP to its processing capacity limit, hence resulting in sizable task drop rate at the global queue of the CMP system. Under this scenario, our method shows an average of **18%** lower task drop rate, with **7%** lower power consumption. Note that the size of GQ was set to the same value in both cases. The reason lies in the separation of IHS and ILS tasks to run on different cores in our method, which prevents unnecessary wait of the IHS tasks for the ILS tasks in the GQ.

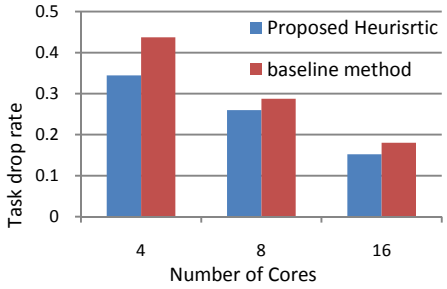

Figure 9. Task loss rate improvement due to the task classification step in the 3T-PM solution.

Figure 9 shows this fact, which can also be interpreted as the higher quality of service (QoS) of the 3T-PM solution compared to the baseline one with feedback, under very high task arrival rate. Finally, in the experiments we performed for various core counts (up to 16) and workload configurations, the power and performance overheads of 3T-PM are negligible. More precisely, the 3T-PM runtime at each tier is negligible compared to the epoch length, i.e., it is less than 1%. Since the algorithm is software based, its power consumption overhead is linearly related to the ratio of execution time of PMU code to that of the applications; hence, the power dissipation overhead of 3T-PM is also insignificant, the same as its runtime overhead.

6. CONCLUSION

We formulated the problem of minimizing the power consumption of a chip multiprocessor system under an average throughput constraint. DVFS and core consolidation along with task assignment methods are employed as part of our solution framework. In particular, we introduced a hierarchical global power manager comprised of three tiers performing core consolidation and coarse-grain DVFS at top tier, assigning the tasks to available cores considering server and task affinities at mid-tier, and closed-loop feedback based per-core DVFS at the low-tier.

The proposed PM suffers from a number of limitations which can be summarized as follows: (i) it focuses on independent tasks and ignores communication between the tasks, and also parallel and multi-threaded tasks, which will be studied as the future work and (ii) relying on a centralized TDU limits the scalability of the proposed 3T-PM to large number of cores. The reason of using a centralized dispatch mechanism is to have better control on task assignment, while a distributed task fetch mechanism would give better scalability with cost of less controllability over optimality of task assignment. Furthermore it does not consider heterogeneous multi-core processors, which is again our future work. Comparison of this technique to a baseline one showed 23% power saving for our technique, which also resulted in some 18% lower task drop rate under stringent throughput constraints for the target CMP system. Considering the promising simulation results, we would like to implement 3T-PM algorithm into the kernel of an open-source operating system and evaluate its performance and power saving in a physical CMP.

Finally, a more sophisticated task assignment algorithm, based on the *Limited Minimum Intervening* (LMI) task scheduling policy of [29], can be developed to further improve the power efficiency by considering *cache affinity* to reduce *CMF*. More precisely, we will build a weighted graph of tasks and cores, and define task-to-core and inter-task affinity factors. Next we will develop an algorithm, based on Fiduccia-Mattheyes heuristic solution to the VLSI CAD partitioning problem [30], to optimally assign tasks to cores considering the given cache affinity factors.

REFERENCES

[1] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," *Proc. IEEE/ACM int'l Symp. on Microarchitecture,* 2006.

[2] S. Herbert, D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," *Proc. of Int'l Symp. on Low Power Electronics and Design*, 2007.

[3] J. Sharkey, A. Buyuktosunoglu, and P. Bose, "Evaluating Design Tradeoffs in On-Chip Power Management for CMPs," *Proc. of Int'l Symp. on Low Power Electronics and Design*, 2007.

[4] J. Li, J. F. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," *Proc. Int'l Symp. on High-Performance Computer Architecture,* 2006.

[5] R. Kumar, D. M. Tullsen, N. P. Jouppi, P. Ranganathan, "Heterogeneous Chip Multiprocessors", *IEEE Computer*, 38(11):32–38, 2005.

[6] S. Ghiasi, T. Keller, F. Rawson, "Scheduling for heterogeneous processors in server systems," *Proc. of the 2nd Conf. on Computing Frontiers* 2005.

[7] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors," *Proc. of Int'l Conf. on Computer-Aided Design*, 2008.

[8] F. Xia, Y.-C. Tian, Y. Sun, J. Dong, "Control- Theoretic Dynamic Voltage Scaling for Embedded Controllers," *IET Computers and Digital Techniques*, 2008.

[9] H. Aydin, Q. Yang, "Energy-Aware Partitioning for Multiprocessor Real-Time Systems," *Proc. Int'l Symp. on Parallel and Distributed Processing*, 2003.

[10] Y. Xie, W. Wolf, "Allocation and scheduling of conditional task graph in hardware/software co-synthesis," *Proc. conf. on Design Automation and Test in Europe,* 2001.

[11] M. Harchol-Balter, M. E. Crovella, C. Murta, "On choosing a task assignment policy for distributed server system," *IEEE Journal of Parallel and Distributed Computing*, vol59, 1999.

[12] M. Annavaram, E. Grochowski, J. Shen, "Mitigating Amdahl's Law through EPI Throttling," *Proc. of 32nd Annual int'l Symp. on Computer Architecture*, 2005.

[13] I. Yeo, C.C. Liu, E.J. Kim, "Predictive dynamic thermal management for multicore systems," *Proc. of the 45th Annual Design Automation Conference,* 2008.

[14] M. Gomaa, M.D. Powell, T. Vijaykumar, "Heat-and-run: leveraging SMT and CMP to manage power density through the operating system," *SIGOPS Operating System Review,* 2004.

[15] G. Qu, "Power Management of Multicore Multiple Voltage embedded Systems by Task Scheduling," *Proc. Int'l Conf. on Parallel Processing Workshops*, 2007, pp. 78-83.

[16] K. Choi, R. Soma and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," *Proc. of Int'l Symp. on Low Power Electronics and Design,* Aug. 2004, pp. 174-179.

[17] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," *SIGARCH Computer Architecture News,* 2006.

[18] P. Juang, Q. Wu, L. Peh, M. Martonosi, D.W. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors," *Proc. of int'l Symp. on Low Power Electronics and Design*, 2005.

[19] W. Kim, M. Gupta, G. Y. Wei, D. Brook, "System level analysis of fast, per-core DVFS using on-chip switching regulators," *Proc. Int'l Symp. on High-Performance Computer Architecture*, 2008.

[20] http://www.intel.com/products/processor_number/chart/xeon.htm [online]

[21] P. Rong and M. Pedram, "Energy-aware task scheduling and dynamic voltage scaling in a real-time system," *Int'l Journal of Low Power Electronics, American Scientific Publishers*, Vol. 4, No. 1, Apr. 2008.

[22] S. L. Hary, and F Ozguner, "Precedence-Constrained Task Allocation onto Point-to-Point Networks for Pipelined Execution," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, no. 8, Aug. 1999.

[23] SPEC Web2009, [online] http://www.spec.org/web2009

[24] Intel Corp, Intel® 64 and IA-32 Architectures Software Developer's Manual, 2009, [online] http://www.intel.com/products/processor/manuals/

[25] M. Zagha, et al., "Performance analysis using the MIPS R10000 performance counters," *Proc. Conf. on Supercomputing* , 1996.

[26] R. C. Dorf, R. H. Bishop, *Modern Control Systems*, Prentice Hall, 2008.

[27] M. Shah, et al., "UltraSPARC T2: A Highly-Threaded, Power-Efficient, SPARC SOC," *Proc. of Asian Solid-State Circuits Conference*, Nov. 2007.

[28] SPEC2000, [online] http://www.spec.org/

[29] M. S. Squilante and E. D. Lazowska, "Using processor-cache affinity information in shared-memory multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, Feb. 1993.

[30] M.A. Breuer, *Design Automation of Digital Systems: Theory and Techniques*, Prentice Hall, 1975.