A Hybrid Local-Global Approach for Multi-Core Thermal Management

Ramkumar Jayaseelan Tulika Mitra
Department of Computer Science
National University of Singapore
{ramkumar,tulika}@comp.nus.edu.sg

ABSTRACT

Multi-core processors have become an integral part of mainstream high performance computer systems. In parallel, exponentially increasing power density and packaging costs have necessitated system level thermal management solutions for multi-core systems. Dynamic thermal management (DTM) techniques monitor on-chip temperature continuously and typically employs dynamic voltage and frequency scaling (DVFS) to lower the temperature when it exceeds a pre-defined threshold. State-of-the-art DTM solutions for multi-core systems include distributed DVFS (where each core can scale the voltage/ frequency individually) and global DVFS (where all cores scale voltage/frequency simultaneously). Distributed DVFS generally offers higher performance than global DVFS, but it is hard to implement and has major scalability issues.

We propose a hybrid local-global thermal management approach for multi-core systems that offers better performance than distributed DVFS, while maintaining the simplicity of global DVFS. We employ global DVFS across all the cores but locally tune the performance of each core individually through architectural adaptations. We exploit easily reconfigurable micro-architecture parameters such as instruction window size, issue width, and fetch throttling in per-core thermal management. Our hybrid solution is easy to implement and highly effective towards temperature management. The key challenge is appropriate choice of configurations at runtime to provide optimal performance under thermal constraints. We formulate it as a configuration search problem and design an efficient software-based solution that selects the appropriate configuration. Our hybrid method, though simpler to implement, achieves 5% better throughput compared to distributed DVFS.

Categories and Subject Descriptors

C.1.0 [**Processor Architectures**]: General

General Terms

Design, Performance, Reliability

Keywords

Dynamic Thermal Management (DTM), Architecture Adaptation, Multi-Core, Global DVFS

ICCAD'09, November 2–5, 2009, San Jose, California, USA.

1. INTRODUCTION

Exponentially rising power density and on-chip temperatures are some of the key challenges in micro-processor design. While packaging techniques are the first line of defence against rising on-chip temperatures, modern processors are already pushing the limits of what cost-effective packaging solutions can offer. Hence, there is widespread interest in thermal management at different levels of system design. The goal of thermal management techniques is to maximize the performance of a computing system while maintaining its temperature below the threshold.

System and micro-architecture level dynamic thermal management (DTM) techniques have gained prominence in recent years. DTM is an on-line technique where the on-chip temperature is continuously monitored (during system execution) through temperature sensors. When the temperature exceeds a pre-defined threshold, appropriate mechanisms are invoked to reduce it sufficiently. Dynamic voltage and frequency scaling, clock gating, fetch gating [5] are some of the most popular mechanisms employed by state-of-the-art DTM solutions to reduce the temperature.

Initially, most thermal management solutions were proposed in the context of single core systems. The advent of multi-core architectures — where a number of physical processors are integrated in the same chip — calls for innovative DTM techniques that can exploit the unique and additional opportunities offered by such systems. A detailed overview of thermal management options for multi-core processors is presented in [9].

DTM techniques for multi-core architectures can be broadly classified into distributed techniques (which operate at individual core level) and global techniques (which operate globally across all the cores on chip). For instance, dynamic voltage/frequency scaling can be employed in a distributed fashion (*distributed DVFS)* where each core can scale its voltage/frequency independently. Alternatively, a global approach constrains all the cores to scale their voltage/frequency uniformly and simultaneously (*global DVFS*).

Multi-core systems also provide thread migration opportunity to manage temperature. Depending on the workload executing on a multi-core system, there can be substantial variation in temperature among the different physical cores on the same die. Migration-based DTM techniques exploit this temperature differential by periodically moving the threads away from the hot cores to the cold cores and thereby balancing the temperature of the cores.

Let us consider a heterogenous workload comprising of four SPEC benchmarks `wupwise, art, gcc` and `crafty` executing on a 4-core system. Figure 1 shows the temperature profiles of the four cores without DTM and with the previously proposed DTM techniques[1]. We assume a threshold temperature of 82.5^oC. The temperatures of all the four cores remain above the threshold

[1]The experimental setup for the plots will be detailed in Section 6.

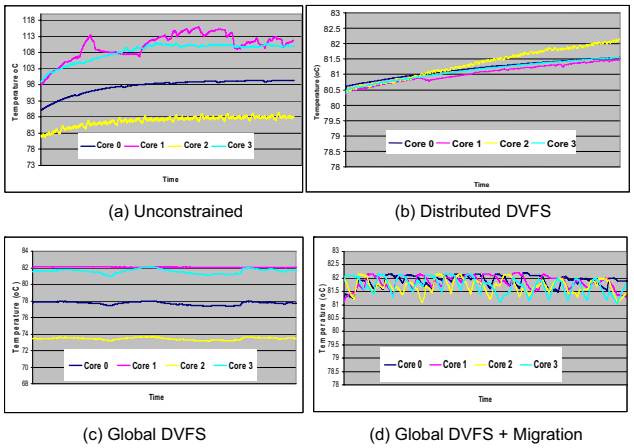

Figure 1: Temperature profiles for a workload on multi-core (core 0: wupwise, core 1: gcc, core 2: art, core 3: crafty). Thread to core mapping is not applicable for migration.

for the entire duration of execution when no thermal management is deployed (see Figure 1(a)). However, we observe a large variation in temperature between the core executing the hottest thread (`gcc`) and the core executing the coldest thread (`art`).

Global DVFS scales down the operating frequency of all the cores from 3.6GHz to 2.92GHz in an effort to lower the temperature of the hottest core (core 1) below the threshold. Clearly, this scheme is unfair for the cold threads (e.g., `art`) as their performance penalties are at par with those of the hot threads. Distributed DVFS addresses this fairness issue by allowing each core to choose its operating frequency independently based on the workload. Thus, distributed DVFS selects a higher operating frequency for the cold thread (3.4 GHz) and a lower operating frequency (2.86 GHz) for the hot thread resulting in substantially better throughput for the overall system.

Clearly, the additional flexibility of being able to manage the temperature of each core independently results in significant performance advantage. However, this advantages for distributed DVFS (which is the only distributed multi-core DTM in the literature) comes at the cost of escalating design complexity. First, allowing each core to have its own supply voltage creates multiple voltage islands on-chip and suitable mechanisms need to be built in for communication among the voltage islands. Secondly, voltage regulators have to be provided per core (so that each core can scale its supply voltage independently) increasing the complexity of the power delivery network [14]. Finally, as the communication among the cores need to be verified for each possible voltage state of each core, the verification complexity for the chip increases exponentially.

Thread migration can mitigate, to some extent, the performance impact of global DVFS without the additional hardware complexity of multiple voltage islands [10, 15]. As threads are periodically migrated among the cores, any single core is less likely to get heated up significantly. Global DVFS coupled with thread migration help to smoothen out the temperature difference across the cores and hence boost the operating frequency of the chip. The thermal profile of global DVFS + migration is shown in Figure 1(d). Employing thread migration along with global DVFS has enabled the entire chip to operate at 3.14GHz as opposed to 2.92GHz for global DVFS alone. However, migration has its own limitations. First, migration does not reduce the total power dissipated in the system; it simply moves the hot spots around instead of eliminating the hot spots. Secondly, the huge penalty associated with switching a task from one core to another constraints the time scale at which migration can be performed. Finally, migration is not very scalable as it has quadratic complexity in the number of cores.

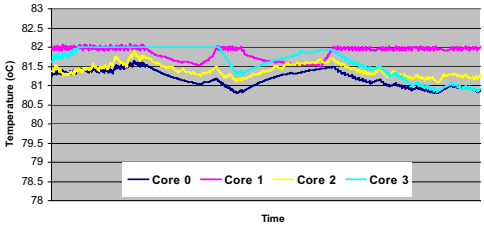

Figure 2: Temperature profiles for hybrid DTM on multi-core (core 0: wupwise, core 1: gcc, core 2: art, core 3: crafty).

In this paper, we propose a novel two-level hybrid DTM scheme for multi-core systems. We observe that per-core control is quite powerful in achieving high performance as long as it does not add substantially to the design complexity. Clearly, DVFS is only suitable at a global level for the entire chip. We need a different set of knobs to control the performance and power per core independently. We recently proposed non-DVFS techniques such as fetch gating and architecture adaptations (runtime reconfiguration of instruction window size and issue width) for dynamic thermal management of single cores [11]. These techniques are easy to implement at individual core level as they are largely localized and do not create complexity in terms of communication among the cores and multiple voltage islands. At the same time, they provide effective response (though gentler than DVFS) to thermal stress.

Our hybrid scheme thus combines *local non-DVFS techniques* (fetch gating and architecture adaptation) at per core level with *global DVFS*. Each core can chose an appropriate configuration (fetch gating level and architectural parameters) to maintain its own temperature. Thus the global operating frequency need not be lowered to keep the hottest thread below the threshold. This is illustrated in Figure 2, where our hybrid DTM scheme employs non-DVFS techniques at individual core level to balance the core temperatures and the global supply frequency is no longer limited by the temperature of the hottest thread.

The major challenge for our hybrid DTM technique is that it should be accompanied by an efficient *runtime* mechanism that can select the appropriate per-core settings and the global setting so as to control the temperature below the threshold while achieving near-optimal performance. We rephrase the thermal management problem as a configuration search problem and design an efficient *software* based mechanism to adapt the local and global thermal management parameters at runtime depending on the workload. Our thermal management strategy results in 12% better throughput than global DVFS plus migration based scheme and 5% better throughput than the more complicated distributed DVFS scheme.

2. RELATED WORK

Dynamic thermal management leverages on on-chip temperature sensors to control the processor temperature dynamically. Dynamic control of processor temperature can be either implemented in hardware or software. Hardware based DTM schemes employ hardware controllers to monitor the temperature continuously and employ appropriate mechanisms to lower the temperature when it exceeds a certain threshold. Commonly employed mechanisms in hardware DTM include DVFS [18], fetch gating [5] and others.

With the advent of multi-core processors, thermal management of multi-core systems has received a lot of attention. Donald et al. [9] classify thermal management approaches into distributed

(per core) and global schemes and show that distributed schemes can outperform global schemes. Migration based thermal management schemes exploit the heterogeneity in thermal characteristics among threads executing on different cores of a multi-core architecture. The temperature of the cores are controlled by periodically migrating threads and thus changing the mapping of threads to cores [10, 15, 9].

In this paper, we propose a hybrid two-level thermal management scheme for multi-core system. We show that employing a combination of simple local schemes and global DVFS results in simple and effective thermal management solutions. We have recently proposed runtime reconfigurable architectural parameters (instruction window size and issue width) as an effective mechanism for single-core DTM [11]. Applying multiple knobs simultaneously for thermal management of multi-cores is more challenging because (a) all the cores are constrained to use identical operating frequency preempting the possibility of choosing per-core parameters independently, and (b) the configuration and workload of a core has significant impact on the temperature of the neighboring cores due to lateral coupling (lateral heat transfer among adjacent cores). In this work, we design a software based thermal management framework that adapts the local and global thermal management parameters at runtime.

3. OVERVIEW OF HYBRID DTM

In this section, we provide an overview of our hybrid local-global thermal management framework.

3.1 Adaptive Multi-Core Architecture

We assume that each core in our multi-core architecture employs out-of-order execution engine. The only difference between our adaptive multi-core architecture and a normal multi-core is the presence of three runtime reconfigurable micro-architectural parameters: the issue width, the instruction window size and the fetch gating level. We chose these parameters because they can directly impact the temperature of the hotspots within the processor and are easily configurable at runtime [20]. Unlike DVFS, the above mentioned schemes are easier to implement at per core level as these knobs are localized within a core and do not affect communication among different cores.

The issue width can be scaled down by disabling the appropriate selection tree [3]. Fetch unit can be controlled by setting the appropriate fetch gating level. When the fetch gating level is set to T, the fetch unit is deactivated once after every T cycles. When the gating level is set to 0, the fetch unit is active for all cycles (no fetch gating). The instruction window has four equal partitions and each partition can be enabled/disabled separately [7]. Changing the window size involves a pipeline flush followed by resetting the window to the appropriate new size. As our adaptivity is coarse grained (large sampling interval in the order of milliseconds), such pipeline flush has negligible impact on performance.

We employ DVFS globally in conjunction with the above mentioned per-core knobs. For a time varying multi-programmed workload, the challenge is to determine, at runtime, the appropriate setting for all these knobs that maximizes performance of the system while maintaining the temperature of all the cores below the threshold. Next we formally define the thermal management problem as a configuration search problem.

3.2 Problem Formulation

The goal of our thermal management framework is to determine the optimal parameters for each core (issue width, window size and fetch gating level) as well as the global supply voltage/frequency that maximizes performance (in billion instructions per second or BIPS) while maintaining the temperature of all the cores below the threshold. To state formally, given N cores, we would like to choose a configurations for each core C_i ($1 \leq i \leq N$, $1 \leq C_i \leq M$) where M is the total number of configurations per core and a global operating frequency F such that

- The system performance $Perf = F \times \Sigma_{i=1}^{N} IPC_i$ is maximized where IPC_i is the instructions per cycle of core i for configuration C_i
- For all cores $1 \leq i \leq N, Temp_i < Th$ where $Temp_i$ is the temperature of core i and Th is the threshold temperature.

The configuration C_i for each core consists of three parameters: issue width (IW_i), window size (W_i), and fetch gating level (T_i).

There are two main challenges to solving the configuration search problem. Given a specific configuration for each core and global operating frequency, we need to determine if this configuration is thermally safe and evaluate the performance for this configuration. In our framework, we use a neural classifier to determine if a configuration is thermally safe and a performance prediction model to evaluate the performance of a given configuration.

Secondly, the large configuration space and the online nature of our scheme makes exhaustive search of all configuration points infeasible. For instance, given a 4-core architecture with five possible issue widths, four possible window sizes and eight possible fetch gating levels at each core and eight global frequency levels, there is a total of $160^4 \times 8$ configuration points. We devise a three-phase search strategy that efficiently solves this configuration search problem.

3.3 Thermal Management Framework

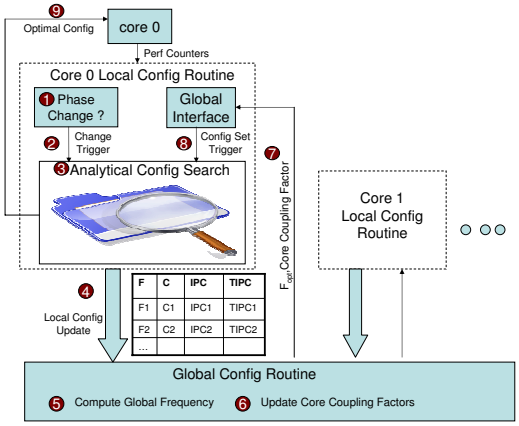

Figure 3: Overview of our hybrid DTM framework.

Figure 3 presents the structure of our hybrid local-global thermal management framework. During execution, each core periodically (0.1 ms) samples the on-chip thermal sensors. In addition, it collects the performance counter values and the instruction signatures once every 3 ms. The instruction signatures are used to determine if there is a phase change [8] in the application executing on the core. A phase change will trigger a configuration search process to better fit the architecture according to the new workload. Note that local phase change in the workload of processor P_i might result in a new configuration for processor P_j ($i \neq j$) that did not encounter phase change itself.

The central component of our framework is an efficient multi-phase configuration search strategy. As mentioned earlier, the configuration space is too large to be searched at runtime. On the other

hand, we cannot nicely partition the search space at core boundaries as the frequency has to be selected globally and the configuration selected for a core impacts the temperature of the cores in the neighborhood due to lateral transfer to heat. Our search strategy breaks this cyclic dependency by employing multiple phases: (1) a local phase suggests optimal configuration at every global frequency level based on minimal global information, (2) a global phase follows that selects the global operating frequency, and (3) another local phase concludes the search by selecting appropriate local configurations armed with more global information.

The first search phase proceeds locally (i.e., independently) for all the cores that detect phase changes in their workload. It suggests for each core the optimal safe configuration C_i^f (highest throughput without exceeding temperature threshold) at every frequency level f and the expected instructions per cycle (IPC) for this configuration. This phase uses an estimate of lateral coupling as minimal global information is known at this point. Our local search algorithm is a modified version of the binary search and examines a very small portion of the entire search space (presented in Section 4). It employs a neural classifier to determine if a configuration is thermally safe for a workload. As we predict the thermal safety of a configuration, we must provision for a failsafe mechanism in case of rare misprediction that may result in the system exceeding thermal threshold. Thus, our technique is backed up by global clock gating where all the cores are made inactive (by hardware) for one sampling interval of thermal sensors (0.1 msec) when the temperature of any core hits the threshold.

The second phase is global in nature. It takes in the results from the local configuration searches as input and determines the global operating frequency F that maximizes throughput. In addition, it estimates the core coupling factor. The temperature of a core depends on configuration and workload of all the cores in the system. The core coupling factor (defined in Section 4.1) reflects the operating conditions of the other cores in the system.

Now that more accurate lateral coupling information is available, we need one final local phase for each core to check the thermal safety and optimality of the suggested configuration at frequency level F. If the previously suggested configuration (with limited global information) is either unsafe (more thermal impact from neighbors) or sub-optimal (less thermal impact from neighbors allowing higher performance configuration), the core can choose an appropriate safe and optimal configuration C_i.

Once all the three phases are over, the global operating frequency is set to F and the reconfigurable components of each core are scaled according to the selected parameters.

4. LOCAL CONFIGURATION SEARCH

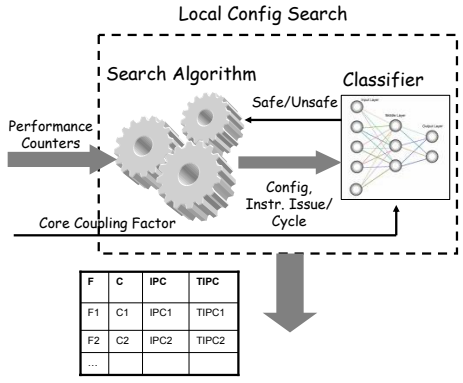

Figure 4: Overview of local configuration search

An overview of the local configuration search is presented in Figure 4. It has two main components: the configuration search algorithm and the classifier.

The highest performing configuration that is feasible at a given frequency depends on the workload running on a specific core. Clearly, determining this configuration requires us to evaluate the IPC and the thermal safety of the configuration. We use an analytical performance model to quickly estimate the IPC for a {configuration, workload} pair using the sampled performance counters as input. We use a neural classifier to predict if a given {configuration, workload, frequency} tuple is thermally safe.

Next we present our neural classifier. The performance prediction model is presented in Section 4.2.1 and the search strategy is presented in Section 4.2.2.

4.1 Neural Network Classifier

We model the problem of determining if a particular {configuration, workload, frequency} tuple is thermally safe as a neural classifier problem. The neural classifier accepts parameters reflecting the configuration of the cores, workload and frequency as input and predicts if the corresponding execution is thermally safe.

Parameters. Our neural classifier is shown in Figure 5. In a multi-core system, the temperature of a core depends both on the power dissipation of the core as well as power dissipation of the other cores in the system. In our adaptive multi-core architecture, the temperature of a core $Core_i$ depends on the workload, configuration of $Core_i$, the workload and configuration of the other cores $Core_j$ $(j \neq i,\ 1 \leq j \leq N)$, as well as the global operating frequency F of the system. Our classifier includes both local and global parameters to reflect this requirement.

An example classifier running on core 0 (C0) of a four-core system is shown in Figure 5. The top four inputs correspond to the workload on this core in the form of number of instructions of different classes issued per cycle. The instruction classes (integer, floating point, load/store, and branches) are chosen based on the observation that these parameters strongly affect the utilization of the execution core and the branch predictor — the hottest units in a core. These values are obtained from the performance prediction model. This is followed by the configuration of the core and the global operating frequency. We reduce the configuration space consisting of three variables (issue width, widow size and throttling level) into a single variable based on the observations from the performance model. The value of this single variable is input to the classifier to reflect the configuration.

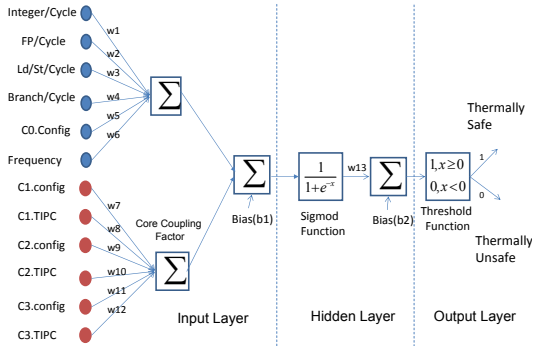

Figure 5: Neural network classifier.

The remaining parameters correspond to global effect. The global power dissipation (and hence the lateral coupling) is captured through two parameters per core, namely, the total instructions issued per

cycle ($TIPC_j$) and the configuration (C_j) of the core. We use multiple parameters to reflect the workload of the local core, while we use only a single parameter for the other cores as temperature depends more strongly on local parameters than remote parameters. The global phase updates the neural classifier at each core with the global parameters. As explained in Section 5, the structure of the classifier ensures that the global phase can pre-compute and transfer a single value to reflect the global parameters corresponding to all the other cores.

Architecture. We use a back propagation network with a single hidden layer and one neuron in the hidden layer as shown in Figure 5. We chose this configuration as it provides a good tradeoff between classification accuracy and complexity. The input layer performs the following computation.

$$Out_{L1}=w_1\times I_1+w_2\times I_2+...+w_6\times I_6+CF+b_1 \quad (1)$$

$$CF=w_7\times I_7+w_8\times I_8+...+w_{12}\times I_{12} \quad (2)$$

where Out_{L1} is the output of the first layer (input layer) of the neural classifier, $w_1 - w_{12}$ are the weights in the neural classifier, $I_1 - I_6$ are the inputs corresponding to the local workload and configuration parameters, $I_7 - I_{12}$ are the inputs corresponding to the workload and configuration of the other cores in the system. CF is the core coupling factor that can be computed by the global configuration routine and transferred to the local configuration routine (see Section 5).

The hidden layer applies the sigmod function and the output layer applies the threshold function as shown in Figure 5. The final output of the classifier is a binary output which predicts if the combination of configuration of the cores, workload and global operating frequency is thermally safe.

Training and Accuracy. We train our neural classifier using a set of microbenchmarks. Each micro-benchmark comprises of a loop body with 100 randomly generated instructions. The instructions are chosen from the input instruction classes and the random selection of instructions is skewed to represent the instruction mix found in real benchmarks. We generate thirty such micro-benchmarks. We then generate 3,000 different training examples, each consisting of a randomly generated workload mix of four micro-benchmarks and randomly chosen core configurations. Each of these 3,000 examples are executed to observe if the temperature of any core hits the threshold. After generating the training set, we use the Levenberg-Marquardt [1] training algorithm to train the classifier. The training algorithm is an iterative process that adjusts the weights and bias values in the neural network to minimize the number of miss-classifications in the training set. We also verified the accuracy of our classifier for the workload mix presented in Table 1 and found that it has an accuracy of over 95%.

4.2 Search Algorithm

The configuration search algorithm examines points in the frequency, architecture adaptation space to determine the best performing thermally feasible configuration at each frequency level. It encompasses a performance model to predict the performance at a given configuration.

4.2.1 Performance Prediction Model

Our model takes in the performance counters at a specific configuration as input and evaluates the IPC and total instructions issued per cycle at a different configuration. The performance counters collected are the following (i) Number of committed instructions N_{inst}, (ii) Number of cycles in the interval $Cycles$, (iii) Number of committed instructions of type X: N^X_{inst} where X can be of type integer, floating point, branch, or load/store, (iv) Instruction cache misses IC_{miss}, Data cache misses DC_{miss}, and Branch mispredictions Br_{miss}.

Our performance prediction model is adapted from the interval analysis [13]. Interval analysis is based on the notion that a superscalar model has a sustained background level performance that is interrupted by miss events such as branch mispredictions and cache misses. Thus the cycles per instruction (CPI) can be expressed as

$$CPI=CPI_{steady}+CPI_{miss} \quad (3)$$

$$CPI_{miss}=CPI_{brmiss}+CPI_{icmiss}+CPI_{dcmiss} \quad (4)$$

where CPI_{steady} is the background sustainable performance when there are no miss events and CPI_{brmiss}, CPI_{icmiss}, CPI_{dcmiss} represent the performance loss from branch miss-predictions, instruction cache misses and data cache misses, respectively.

CPI_{miss} can be computed from the individual miss events and their miss penalties as follows

$$CPI_{miss}=\frac{IC_{miss}\times IC_{penalty}+DC_{miss}\times DC_{penalty}+Br_{miss}\times Br_{penalty}}{N_{useful}} \quad (5)$$

where IC_{miss}, DC_{miss}, and Br_{miss} are number of I-cache misses, D-cache misses and branch mispredictions over a particular interval and N_{useful} is number of useful instructions committed over the same interval. The penalties ($IC_{penalty}$, $DC_{penalty}$, $Br_{penalty}$) are estimated using the first order superscalar model [13].

We assume that CPI_{miss} remains constant across configurations. Thus CPI at a specific configuration $\mathcal{C}$ can be expressed as

$$CPI(\mathcal{C})=CPI_{steady}(\mathcal{C})+CPI_{miss} \quad (6)$$

We now proceed to examine how the different points in the configuration space affect CPI_{steady}. The steady state IPC (which is the inverse of CPI) at window size W and issue width IW can be expressed as [13]

$$IPC_{steady}(IW,W)=min(IW,\sqrt{W}) \quad (7)$$

At fetch gating level T, the fetch unit can deliver at most $\frac{T}{T+1}\times FW$ instructions per cycle, where FW is the fetch width. In the steady state, the number of instructions fetched per cycle should be equal to the number of instructions issued per cycle. So the steady IPC at configuration $\mathcal{C}=\langle T, IW, W\rangle$ can be expressed as

$$IPC_{steady}(\mathcal{C})=min(\tfrac{T}{T+1}\times FW,\ IW,\ \sqrt{W}) \quad (8)$$

This equation assumes that all instructions are unit latency. If we account for the variable instruction latencies [13]

$$IPC_{steady}(\mathcal{C})=min(\tfrac{T}{T+1}\times FW,\ IW,\ \tfrac{\sqrt{W}}{L}) \quad (9)$$

where L is the average instruction latency. L is an application dependent parameter. However we simply set the value of L to the highest average instruction latency across all our benchmarks, which turns out to be 1.77.

Equation 9 presents the relationship between our configuration parameters and performance. Clearly the steady IPC can be limited by either the issue width, window size or gating level. Hence, given a specific value of steady IPC, it can be sustained by a balanced architecture. Given a specific value for the steady IPC (S_{ipc}),

$$IW=S_{ipc};\ W=S^2_{ipc}L^2;\ T=\frac{S_{ipc}}{(FW-S_{ipc})} \quad (10)$$

Over-dimensioning the architecture along any one of the configuration parameters does not increase performance significantly. Thus our micro-architecture configuration space comprising of three parameters can be reduced to a single parameter.

Given a specific configuration presented by steady IPC S_{ipc}, the corresponding architectural parameters can be derived from Equation 10. The actual CPI (including miss events) at the particular configuration S_{ipc} is given by

$$CPI(S_{ipc}) = \frac{1}{S_{ipc}} + CPI_{miss} \quad (11)$$

where CPI_{miss} can be computed from the sampled miss events (performance counters) using Equation 5.

However, our neural classifier accepts the total number of issued instructions (right path and wrong path instructions) of different classes as input because power consumption depends on the issued instructions. We modify the performance prediction model [12] to estimate the total number of issued instructions per cycle $TIPC$. The number of instructions issued per cycle of different types is

$$TIPC^X(S_{ipc}) = \frac{N^X_{inst}}{N_{inst}} \times TIPC(S_{ipc}) \quad (12)$$

where N_{inst} is the total number of committed instructions in the previous sampling period, and N^X_{inst} is the total number instructions of type X (integer, floating point, branch, or load/store) committed in the previous sampling interval.

4.2.2 Search Process

The configuration search process proceeds along two axes: the operating frequency and the steady IPC. For each {frequency level, steady IPC} pair, it first uses the performance prediction model to determine the actual IPC and the total instructions issued per cycle (TIPC). These values and the core coupling factor (from global phase) are given as input to the neural classifier, which determines if the {frequency, steady IPC} pair is thermally safe. Using this process, the highest performing thermally safe configuration at each operating frequency level can be determined.

The search process can be optimized further based on the observation that any increase in steady IPC (wider configuration) results in an increased performance and temperature. Thus, if at a given frequency f, steady IPC S_{ipc} is infeasible, then all steady IPC values higher than S_{ipc} are also infeasible. Exploiting this property, our configuration search can proceed as a binary search along the steady IPC axis. It first examines the midpoint of the steady IPC space. If this point is feasible, it proceeds to the upper half of the search space; otherwise it proceeds to the lower half of the search space. Given $\mathcal{F}$ frequency levels and $\mathcal{S}$ steady IPC levels, the total number of points examined is at most $\mathcal{F} \times ln(\mathcal{S})$.

In our experiments, we have eight frequency levels between 3.6 GHz and 2.2 GHz and 9 steady IPC levels between 2 and 6 resulting in at most 32 search points. The configuration search routine takes about 8,000 cycles in the worst case. Even if the routine is invoked once every phase detection sample (3 msec), it represents an overhead of only 0.8%. In reality the overheads are less than 0.1% as the routine is invoked infrequently.

5. GLOBAL CONFIGURATION SEARCH

The global configuration routine is responsible for determining the global operating frequency and the core coupling factors.

Inputs. The input to the global configuration routine is a set of tables: one from each of the local search routines. The table for each core is indexed by the different operating frequency levels. At each frequency level f, the entries contain the highest performing configuration C_i, the IPC (IPC_i) and the total instructions (correct + wrong path) issued ($TIPC_i$) in this configuration.

Operating Frequency. The aim of the framework is to maximize the system performance which is defined by $P = F \times \sum_{i=1}^{N} IPC_i$, where F is the operating frequency, N is the number of cores, and IPC_i is the IPC of the i^{th} core. The global configuration routine can easily determine the system performance by making a linear scan of all the tables at a particular frequency level. It then selects the frequency level with the maximum system performance as the global operating frequency. Note that the global configuration routine can be easily modified to maximize other metrics such as weighed speedup, harmonic speedup, etc.

Core Coupling Factor. In section 4.1, we presented how the core coupling factor approximates the global effects on the temperature of a particular core. The coupling factor for each core is a weighed sum of the configurations and the total IPC of other cores in the system. For example, the coupling factor for core 0 is

$$CF_0 = wc_{01}config_1 + wt_{01}tipc_1 + wc_{02}config_2 + wt_{02}tipc_2 + wc_{03}config_3 + wt_{03}tipc_3$$

The weights in the coupling factor are the classifier weights determined during off-line training of the classifier (Section 4.1). Once the optimal frequency is calculated, the global configuration routine determines the coupling factor for all the cores using the configuration and the total instructions per cycle from the tables.

Final Tuning. The computed optimal global frequency F and the core coupling factor are updated for the individual cores. Each core then re-runs the local configuration search only for the optimal frequency F with the new coupling factor to determine the configuration for execution.

Overheads. Our global search algorithm performs a maximum of $O(N^2 + \mathcal{F})$ multiplications and $O(N\mathcal{F} + N^2)$ additions where N is the number of cores and $\mathcal{F}$ is the number of distinct frequency levels. We implemented and profiled our algorithm and determine the overhead in the worst case to be 1,215 cycles for a four-core system with eight different frequency levels. We assume worst case overhead for each invocation of our algorithm in our results.

6. EXPERIMENTAL EVALUATION

We now proceed to evaluate our DTM technique for multi-cores.

Experimental Settings. We use a modified multi-core version of SimpleScalar [4] toolkit for experimental evaluation. We model a 4-core architecture and obtain the performance numbers through detailed cycle-accurate simulation. Each core is configured similar to the Alpha 21264 out-of-order processor with a 128 entry ROB, peak issue width of six instructions per cycle. The cache sizes are configured to 64 KB of L1 data cache and instruction cache and a 2MB unified L2 cache.

We use the Wattch [6] libraries to compute the power dissipated every cycle. We use the linear scaling of Wattch [6] to obtain the power consumption at 1.4V and 3.6GHz at 100 nm (voltage and frequency levels for Pentium 4). We employ eight different frequency levels between 3.6GHz and 2.2GHz for DVFS and assume a penalty of $10\mu s$ per voltage transition [9].

We extend our performance and power models to handle architectural adaptivity, that is, configuring issue width, window size and fetch throttling level. We modify Wattch power models to reflect an adaptive instruction window implementation described in [7] and other core logic from [17].

The power values are averaged over the sampling interval (0.1 msec) and the resulting power trace is fed to the thermal modeling libraries from HotSpot-3.0 [19] to calculate the temperature of each micro-architecture unit within each core. The temperature values are input to the DTM module to guide the thermal management decisions (frequency scaling, architecture adaptation, clock

gating, etc.) and these decisions are communicated back to the core pipeline models.

We use a four-core floorplan similar to previous work [9, 10] containing four identical cores and L2 cache occupying the rest of the die-area. We assume an ambient temperature of 45^oC, heat sink convection resistance of $1.0^oC/W$, and a maximum tolerable temperature of 85^oC [19]. After adjusting for placement and sensor errors, we get a threshold of 82.5^oC [19].

Label	Benchmarks	Label	Benchmarks
T1	bzip,parser,wupwise,gcc	T2	crafty,mesa,gcc,vortex
T3	bzip2,gzip,equake,vortex	T4	eon,gcc,art,wpwise
T5	gzip,vortex,art,parser	T6	fma3d,parser,facerec,bzip2
T7	wupwise,gcc,art,crafty	T8	parser,gzip,bzip,gcc
T9	voretx,bzip,eon,equake	T10	mesa,vortex,facerec,parser

Table 1: Multi-programmed workloads used for evaluation

Workload. We use multi-programmed workloads each comprising of four threads. Each thread is chosen from one of fifteen SPEC CPU 2000 [2] benchmarks. For each benchmark, we fast forward to the first simulation point specified by [16] and then perform detailed simulation. Our simulation includes architectural warmup and thermal warmup [19] phases after which the statistics are collected. The workload mix is shown in Table 1.

DTM Techniques. We compare our proposed technique, called *Hybrid*, that combines global DVFS with local architectural adaptivity against three other state-of-the-art multi-core DTM techniques.

- *Global DVFS:* The frequency/voltage of all cores are scaled equally in response to a thermal stress on any of the cores. We use feedback control (PI controller) to set the appropriate frequency level for a given thermal stress [9].
- *Distributed DVFS:* Each core can chose its own frequency and voltage settings. We use feedback control (distributed PI controller) to set the appropriate frequency levels for a given thermal stress [9].
- *Global DVFS + Migration:* We compared a number of approaches for migration in conjunction with global DVFS to avert thermal stress [10, 15, 9]. We observed that the multi-loop based method combining an inner feedback loop for global DVFS with an outer feedback loop for migration performed the best among all the migration techniques investigated and we present the results for the same. We assume $100\mu s$ overhead per migration [9].

In addition, we include a variation of our *Hybrid* DTM technique, called *Hybrid-Simple* in the evaluation. *Hybrid-Simple* is a simplified version of *Hybrid* where we engage global voltage scaling with local fetch gating to control the temperature. This mechanism is suitable when per core adaptivity cannot be supported and a simpler technique needs to be employed at per-core level.

The throughput of different DTM schemes (from left to right Distributed DVFS, Global DVFS, Global DVFS + Migration, Hybrid, Hybrid-Simple) is shown in Figure 6.

Throughput. The execution of a heterogenous workload generally exhibit large variation in temperature across the cores. Performance of Global DVFS is limited by the temperature of the hottest core, while Distributed DVFS can select appropriate frequency for each core. Employing migration in conjunction with DVFS can bridge the performance gap between global and distributed DVFS to some extent. On an average, Distributed DVFS has 18% higher throughput than Global DVFS. Migration helps boost throughput of Global DVFS by 7.25%.

Our proposed technique Hybrid, where adaptive architecture is employed locally in conjunction with global DVFS, can outperform the more complex Distributed DVFS resulting in 5% better

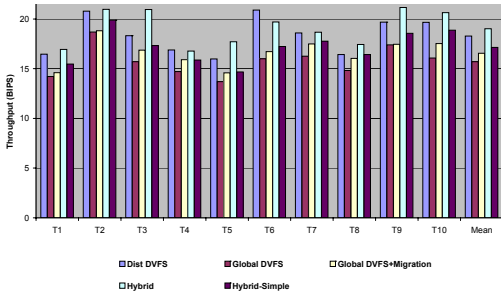

Figure 6: Comparison of throughput for different DTM schemes.

throughput. It has 12% better throughput than Global DVFS+ Migration. Even a simpler version of our scheme Hyrbid-Simple, which only uses fetch gating per core along with global DVFS, performs comparably with Distributed DVFS (4% lower) and better than Global DVFS + Migration (3% higher). The results show that there is significant performance advantage for hybrid DTM techniques that combines global DVFS with local adaptivity.

7. CONCLUSIONS

In this paper we presented a two-level hybrid thermal management scheme for multi-core systems. Our scheme employs a combination of non-DVFS techniques (architecture adaptation and fetch throttling) at a per core level and global DVFS for thermal management. It is simpler to implement and outperforms previously proposed multi-core thermal management schemes.

8. REFERENCES

[1] Matlab Neural Network Toolbox. `www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf`.

[2] SPEC CPU 2000 Benchmark Suite. `http://www.spec.org/cpu2000/`.

[3] D. H. Albonesi et al. Dynamically Tuning Processor Resources with Adaptive Processing. *IEEE Computer*, 36(12), 2003.

[4] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, 35(2), 2002.

[5] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *HPCA 2001*.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-level Power Analysis and Optimizations. In *ISCA 2000*.

[7] A. Buyuktosunoglu et al. A Circuit Level Implementation of an Adaptive Issue Queue for Power-Aware Microprocessors. In *GLSVLSI*, 2001.

[8] A. S. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. *SIGARCH Computer Architecture News*, 30(2), 2002.

[9] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *ISCA 2006*.

[10] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to manage power density through the operating system. In *ASPLOS*, 2004.

[11] R. Jayaseelan and T. Mitra. Dynamic thermal management via architectural adaptation. In *DAC 2009*.

[12] T. S. Karkhanis and J. E. Smith. Automated Design of Application Specific Superscalar Processors: An Analytical Approach. In *ISCA 2007*.

[13] T. S. Karkhanis and J. E. Smith. A First-Order Superscalar Processor Model. In *ISCA 2004*.

[14] W. Kim et al. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, 2008.

[15] C. A. Kivilcim, R. T. Simunic, and G. Kenny C. Proactive temperature balancing for low cost thermal management in mpsocs. In *ICCAD*, 2008.

[16] E. Perelman, G Hamerly, and B Calder. Picking statistically valid and early simulation points. In *PACT*, 2003.

[17] R. Sasanka, C. J. Hughes, and S. V. Adve. Joint Local and Global Hardware Adaptations for Energy. In *ASPLOS 2002*.

[18] K. Skadron. Hybrid Architectural Dynamic Thermal Management. In *DATE 2004*.

[19] K. Skadron et al. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM TACO*, 1(1), 2004.

[20] J. Srinivasan and S. V. Adve. Predictive Dynamic Thermal Management for Multimedia Applications. In *ICS 2003*.