RC23890 (W0602-197) February 24, 2006
Computer Science

IBM Research Report

Global Power Management Policies for Multi-Core Processors

Canturk Isci, Alper Buyuktosunoglu, Pradip Bose, Chen-Yong Cher
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Global Power Management Policies for Multi-Core Processors

1. Introduction

Chip-level power dissipation limits constitute a fundamental design constraint in future high-performance microprocessors. Recently, there has been an industry-wide trend of shifting towards lower frequencies and multiple cores to meet next generation performance targets at affordable power. Nonetheless, even with reduced per-core frequency, the chip-level performance targets for such a multi-core processor are aggressive, and meeting the power budget at those performance levels continues to present a major design challenge. On-chip, dynamic power management is therefore a design feature that is likely to be an integral part of the overall architecture. Initial analysis has shown that clock-gating alone is not enough to meet the chip-level power budget for a high-end multi-core design. Both average and maximum power must be managed to remain within acceptable limits, dictated by power-related maintenance cost budgets and cooling/packaging solution cost limits.

Previous research and our own specific power analysis studies have shown that individual techniques for dynamic management of power (be it of the active, passive or both kinds), invariably have the following characteristics: (a) the average power savings is a strong function of the input workload; in fact in some cases, the net power savings may even be zero or negative; (b) depending on the technique, there is usually an associated performance penalty – either in terms of cycle time (frequency) or cycles per instruction (CPI); (c) there is an area overhead that is paid as a price for any expected power reduction; (d) there is no direct or easy way to bound the worst-case power that is consumed: usually, the only way is an empirical pre-silicon evaluation using an assumed worst-case workload.

As such, given a "bag of tricks" to reduce power, it is clear that ideally, one would like to invoke a subset of those tricks, depending on the workload or the particular phase of the workload. This implies the need for some kind of a "monitor-and-control" facility that would sense the workload demand and microarchitectural activity and dynamically invoke particular mechanisms from within the full repertoire of architected techniques for power reduction and control. A detailed description of a hierarchical power-performance monitor-and-control architecture that can be used to ensure adherence to stipulated power budgets, while maximizing chip-level performance, has been documented elsewhere [1]. In general terms, such a hierarchical controller consists of distributed monitor-and-control units local to each core and non-core storage and interconnect resource, with direct communication paths to a global on-chip controller. The latter (global) controller is responsible for assigning appropriate power optimization modes to each monitored core (or other non-core resource), such that chip-level power budget is not exceeded. Of course, the objective is to maximize performance, while adhering to chip-level power budgets.

In this report, we focus on the problem of devising efficient algorithms for the *global* controller in such a hierarchical monitor-and-control structure. We limit our attention to the core processors in the multi-core chip. Briefly stated, the problem at hand is: given a total chip power budget, and an input set of application programs, how to (dynamically) assign power modes to each processor

core, such that the chip-level performance is maximized, without violating the total chip power budget. The rest of the report is organized as follows. In Section 2, we present a brief overview of the assumptions and objectives governing the global controller function. In Section 3, we describe a fast static global power management analysis tool, *GPAT*, to evaluate different power saving techniques. In Section 4, we describe the global power management policies, with variations for different objectives such as prioritization, fairness and optimized throughput. We perform experiments including transition overhead costs.

2. Global Power-Performance Management: Basic Function and Objectives

For the purposes of this report, let us consider a global on-chip hardware controller, that manages power consumption across multiple cores in the processor chip. We shall call the global power-performance management unit (GMU). In its most general form, such a GMU could be architected to cross-optimize across various different metrics, like power, performance, temperature, reliability, and perhaps even be used to enhance effective chip yield through setting each core to operate nominally at the "right" voltage and frequency (in the face of process variability). But, for this report, we limit the function of the GMU to one of managing power and performance only. The basic objective is to maximize chip-level performance while adhering to a preset chip power budget. Depending on the operating environment, the performance metric to be optimized may be different, and one or more issues like thread (task) priority, fairness, single-thread latency or net chip throughput may be the dominant criteria.

2.1 Power Modes

The GMU optimizes chip power/performance by assigning power modes for each core, based on application behavior. Therefore, a first step in the design of GMU control policies relies on the definition of these power modes.

Specifically, for the purposes of this report we consider three power modes for each processor core: *Turbo, Eff1 and Eff2*, corresponding to three different speed settings. Our target in each power saving mode is to achieve a ***ΔPower Savings : ΔPerformance Degradation*** ratio of 3:1. Under this target, *Eff1* represents a conservative power savings mode with relatively small performance degradation; *Eff2* represents high power saving significant performance degradation. In Table 1, we summarize the design targets for the three modes.

Mode	**Power Savings**	**Performance Degradation**
Turbo	NONE	NONE
Eff1	15%0	5%
Eff2	45%	15%
General Target	**3X**	**1X**

Table 1. Target ΔPower:ΔPerformance ratios for the three modes.

For this report, we consider dynamic voltage and frequency scaling (DVFS) as the basic method for defining modes. In our exploration, we choose a *linear* DVFS scenario, where both voltage and

frequency are scaled linearly. This choice falls within the range of previous product and circuit analysis datasets. For linear DVFS, we define our modes as follows:

- **Turbo**: NO DVFS (Vdd,f)
- **Eff1**: 95% Vdd and 95% f
- **Eff2**: 85% Vdd and 85% f

A very useful property of DVFS is, power and performance behavior can be approximately estimated with simple calculations. Note that this simplifies the design of the power/performance tradeoff table. Power consumption follows the CV^2f equation, and therefore has a cubic relation to DVFS scaling, under our assumption of linear dependence between voltage and frequency. (The presence of leakage power, which also depends on V, does not cause major perturbation to the cubic law governing power and performance). The upper bounds for power and performance, based on these estimations, for our mode choices are shown in Table 2. Note that, actual performance behavior is expected to be better, as asynchronous memory latencies are not scaled with DVFS.

Power Saving			**Performance Degradation**		
Turbo	Eff1	Eff2	Turbo	Eff1	Eff2
0	**~14%**	**~38%**	**0**	**<5.3%**	**<17.7%**

Table 2. Computed power savings and performance degradation upper bounds

In these analyses we determine performance degradation from increases in execution time. Overall, these initial estimates look very promising in terms of achieving 3:1 ΔPower:ΔPerformance ratio. We show in Figure 1, the average power saving and performance degradation for the whole SPEC suite, for each mode. In our experiments with Turandot, we use a nominal frequency of 1.1GHZ and Vdd of 1.3V. In the figure, we also show the IPC in different modes. This shows an increase in IPC as frequency is scaled down, due to asynchronous memory latencies.

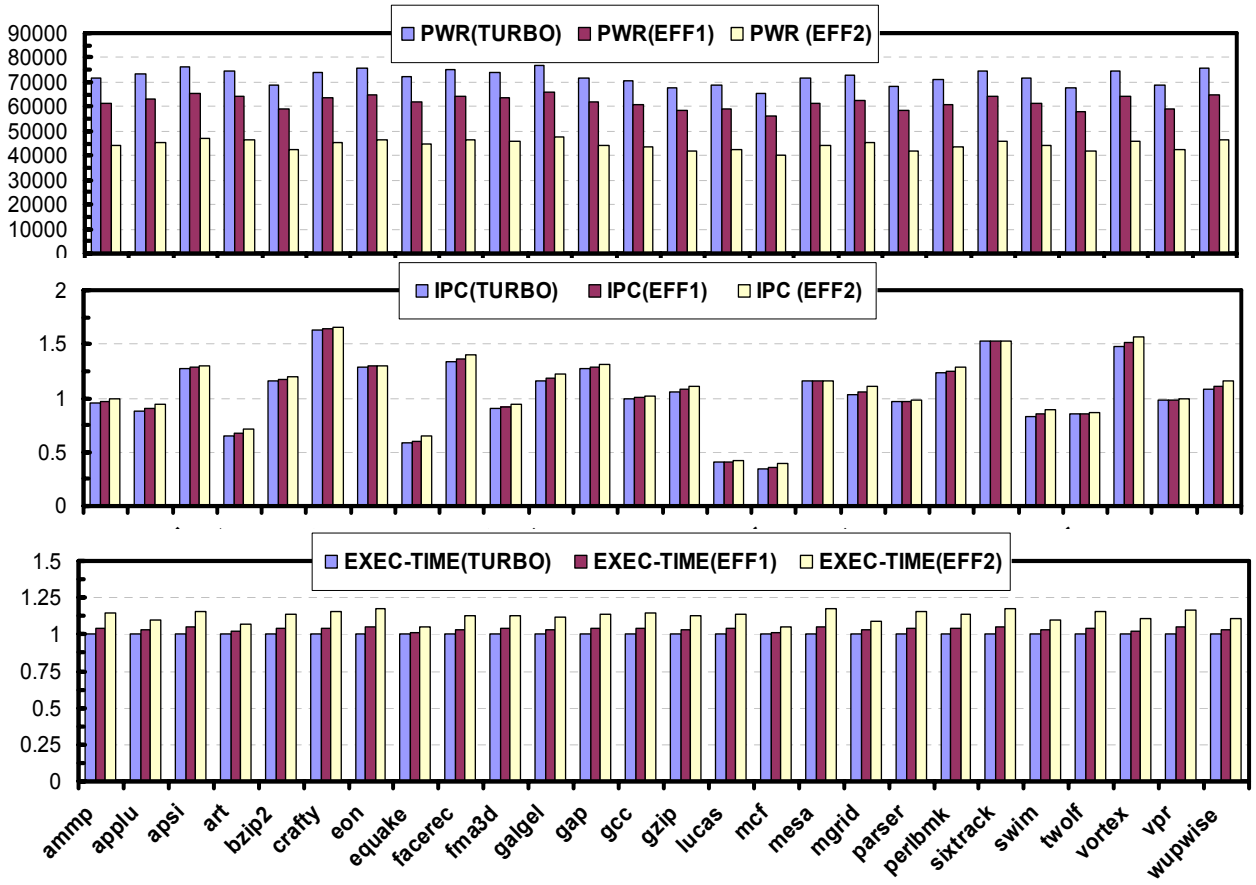


Figure 1. Power and performance of different modes defined by DVFS. (power shown in a.u. arbitrary units)

In Figure 2, we show the aggregate power/performance behavior for the whole SPEC suite. In both plots, *ideal* represents the computed values, without latency scaling. The leftmost bars show the actual values obtained from Turandot simulations. These results show, DVFS is a good candidate for achieving the 3:1 ratio, and the computations for power/performance estimates at different modes track very closely with actual behavior. We make use of this fact later in our global power management policy designs.

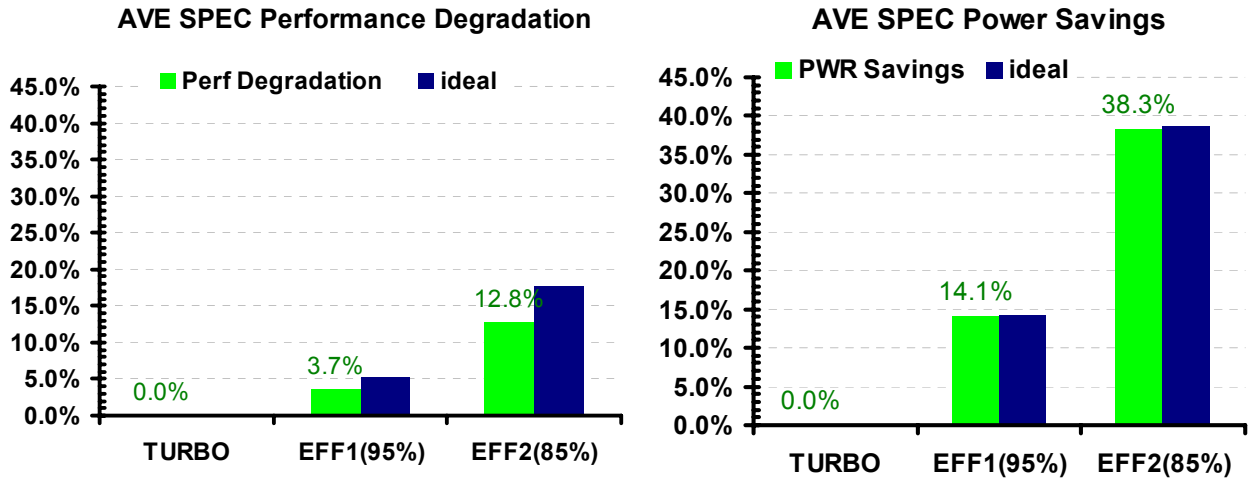


Figure 2. Aggregate SPEC results for power and performance under DVFS.

In the remainder of our work, we use DVFS modes for our power mode definitions, and build upon these for GMU-driven power management. We consider nominal Vdd of 1.3V and frequency of 1.1GHz. We consider leakage as 33% of total unconstrained power and no Vdd gating applied.

3. GPAT: Global Power-Performance Analysis Tool

In order to evaluate GMU-driven power management techniques under a multi-core (CMP) environment, we developed a static analysis tool, GPAT. GPAT simulates different CMP constructs with a loose integration with Turandot. It provides a very fast, perfectly scalable interface to develop and evaluate different policies for GMU, as well as providing benchmark statistics for single threaded and static mode definitions. It also enables us efficiently explore different design objectives such as increasing cores per chip or considering yield improvement with asymmetric mode processors.

GPAT simulation environment uses single threaded Turandot results for each benchmark as input, and performs a CMP simulation by simultaneously progressing over Turandot traces for different benchmarks assigned to different cores. Figure 3 depicts the general data flow for GPAT. For each benchmark, we use Turandot to generate three traces for the three power modes: Turbo, Eff1, Eff2. These traces are based on fixed instruction granularities. Therefore, when a core switches its power mode, GPAT identifies the instruction number for that switch and switches to the input trace that corresponds to the new mode for that benchmark. The mode switches are performed simultaneously at each core, at execution points we refer to as *explore times*. After an explore time, all mode switches are performed and GPAT continues CMP simulation until next explore time. During this period it simply evaluates the power/performance statistics based on the current input trace. GPAT updates its simulation stats every *delta sim time*. That is, at each delta sim time, it reevaluates per core and overall chip statistics. These statistics change, as GPAT progresses through a delta instruction sample and moves to next sample on an input trace. As the output, GPAT produces a single combined trace with delta sim time steps, showing the power/performance behavior of each core and overall chip. It also produces the track of which core operates at which mode.

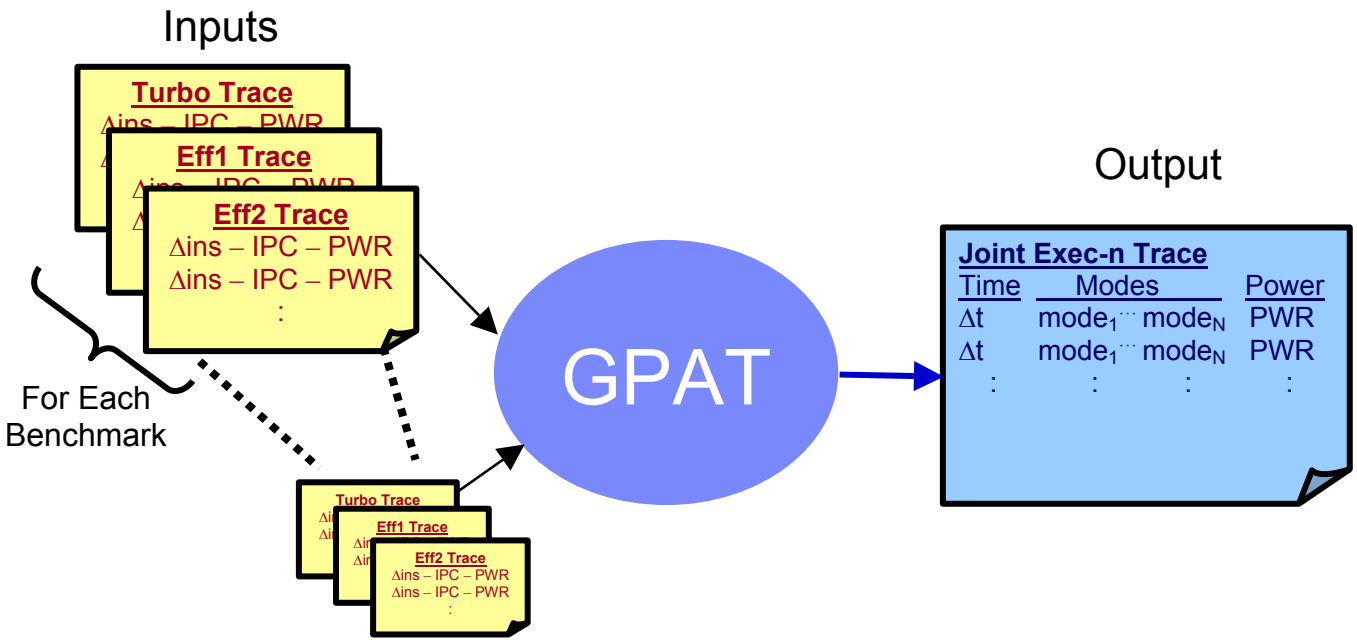


Figure 3. GPAT data flow.

In addition to the simulation timeline, GPAT also produces a large number of statistics, including native execution statistics for each benchmark, power savings and performance degradations with respect to baseline Turbo execution, mode transition statistics, time distributions of modes and transition overhead computations.

GPAT can explore a large number of cores from 2 to 64, with different power budgets, simulation termination conditions. Policy descriptions and different design constraints can easily be modified, added and evaluated under GPAT structure. Inevitably, as GPAT is a static evaluation, it does not include shared L2 bus and address conflicts. Although this should not have significant effect when the most of the chosen applications are not highly L2 and memory intensive, further research is needed for a quantitative evaluation. The major advantage of GPAT is, it provides a fast interface to policy and design evaluation. Although the values may differ between dynamic and static cases, the inter-relations are mostly consistent.

Here we provide a brief example to GPAT utilization with an evaluation of higher on chip integration with DVFS power modes. (one of the possible GMU objectives) In Figure 4, we show the evaluation of a higher core integration for a fixed power budget. In the figure, we show a single chip with 4,5 and 6 cores respectively. For each case, as we include a new core, we also introduce a new highly cpu-bound benchmark. While introduction of each new core/benchmark incurs performance penalties per core, overall system throughput is improved for the same budget. In this example we use a simple, fair policy that tries to balance power consumption of each core. As most of these benchmarks—except for mcf, art and parts of ammp—are cpu-bound, we expect similar conclusions with a dynamic CMP evaluation.

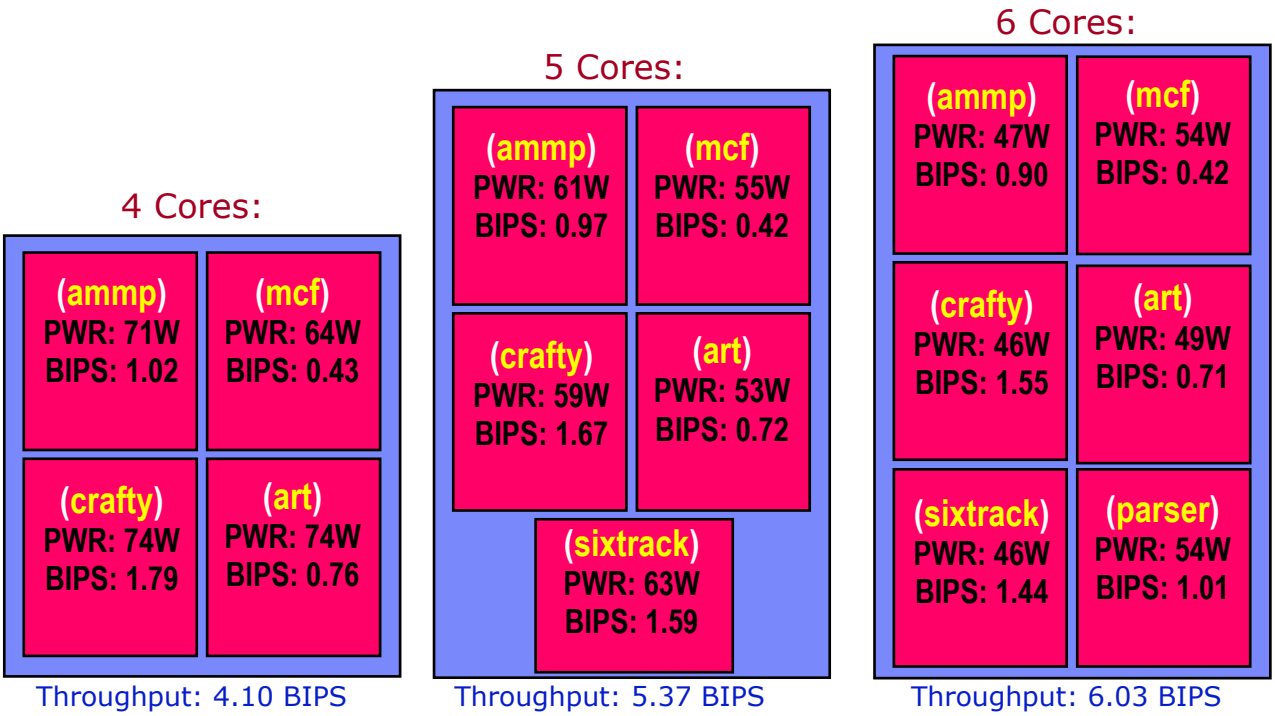


Figure 4. Evaluation of more cores/chip with DVFS modes using GPAT.

4. Power Management Policies for Global Controller

In this section we discuss different GMU policies with a common constraint. We devise policies that are subject to meeting a specific global power budget by adjusting power modes of individual cores. Besides this common constraint, different policies target different objectives such as core prioritization of cores/benchmarks, fairness, power balancing among cores and optimizing system throughput. We assume that each core reports monitored power and performance information for each core to the GMU. The latter delivers global power management actions to cores based on the global constraint and derived policies. In the analysis with GPAT, we use the same parameters throughout the experiments. We use delta sim times of 50us and explore times at 500us. GPAT input traces are generated from Turandot with 100K instruction samples. We experiment with budgets from 150W to 300W for a 4 core CMP, and 300W to 600W for an 8 core case. (The watt value limits assumed are somewhat arbitrary, and do not necessarily reflect any real chip designs). The termination condition is, set when one of the benchmarks reach completion. Thus, all cores are utilized for the experimented regions.

In our evaluations, we use accounting based on (global) time sampling, since, under the multiple frequency domains, synchronization should be based on a global clock or wall clock. Similarly, for performance metrics, we use BIPS instead of IPC.

4. 1 Experimented Policies

Here, we introduce three of the experimented policies for different objectives. The first policy, labeled *priority,* assigns different priorities to different tasks. In our implementation, core4 has the highest priority and core1 has the lowest priority (in a 4-core chip). Therefore, in policy implementation, it tries to run the last core as fast as possible, while preferring to slow down core1 first in case of a budget overshoot. Second policy, *pullHipushLo*, tries to balance the power consumption of each core, by slowing down the core that has the highest power in case of a budget overshoot and vice versa. Last, *MaxBIPS*, targets at optimizing the system throughput, by predicting and choosing the power mode combination that maximizes the throughput at each

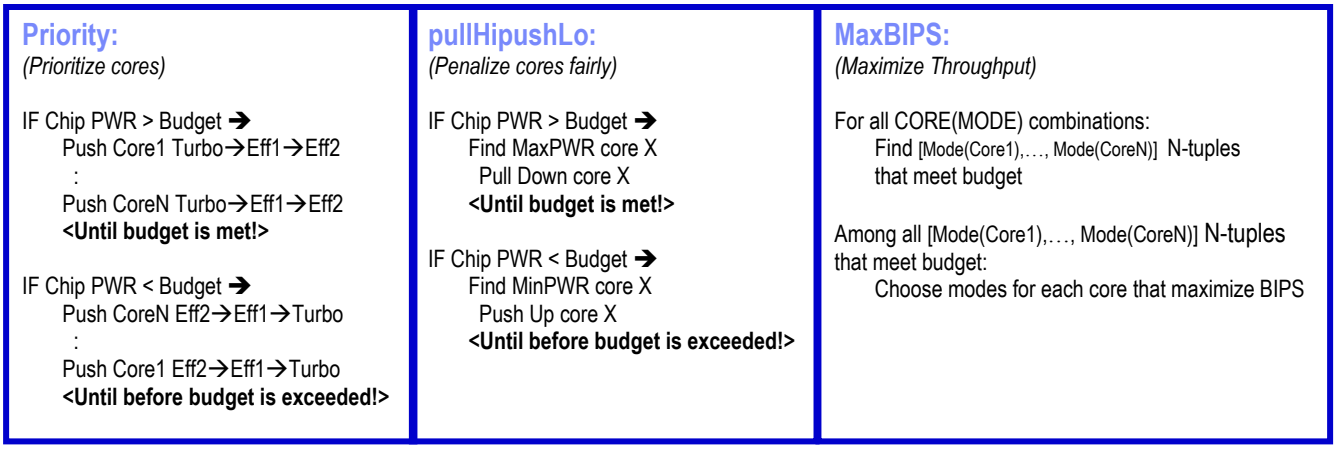


Figure 5. GMU power management policies

explore time. In Figure 5, we summarize the algorithmic implementations of the three policies.

In Figure 6, we show the power/performance behavior of different benchmarks for a 4 core CMP with the three described policies. The top figures plot average performance degradation for each benchmark for several power budget targets. The bottom plots show the average power saving for the same range of budgets. As the target budget is increased towards the worst case corner, all benchmarks converge to zero performance degradation and power saving, as they all can operate safely in Turbo mode. In all these experiments, core1 to core4 execute ammp, mcf, art and crafty. Spanning a range of execution behavior, mcf is highly memory bound, followed by art. Ammp is an average scale benchmark, with both cpu bound and memory bound regions. Finally crafty is a highly cpu bound benchmark.

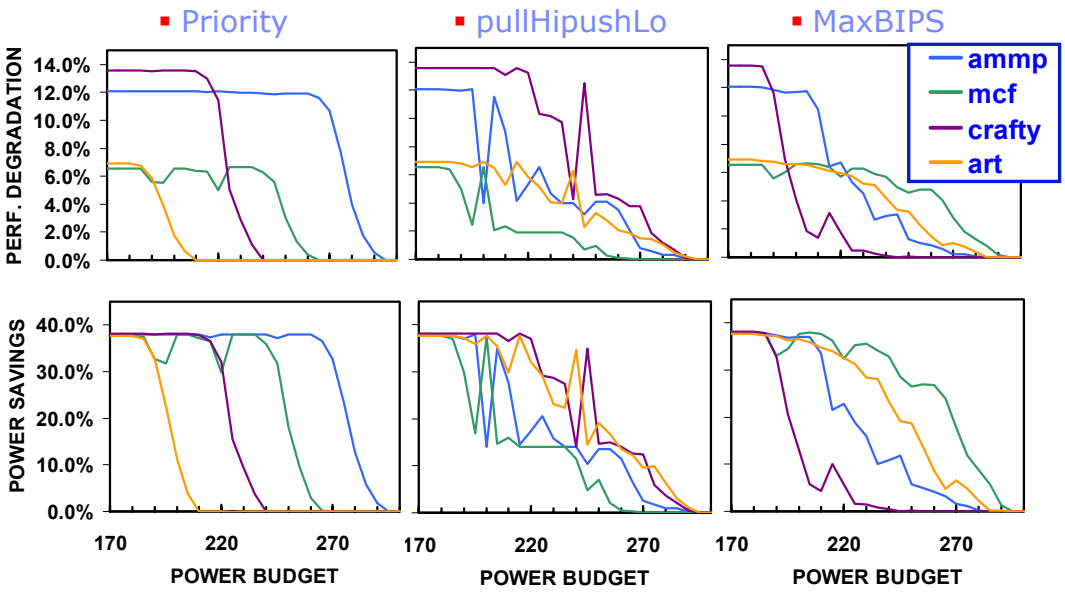


Figure 6. Power/performance results for the three policies for a 4 way CMP.

In Figure 6, in the plots on the left side, we see the straightforward interpretation of priority (prioritization in this example is set as: ammp < mcf < crafty < art). As the power budget is increased, first core4 (art) then core3 (crafty) is released from Eff2 to Turbo, until all cores can execute in Turbo mode. In this scheme, *priority* pushes down mcf to Eff2, as it has the smallest delta power to fit the budget, then as a higher budget is used, the benchmark in the priority order can actually find enough slack to move to Eff1. In the middle plots, pullHipushLo tries to balance the power of each core, therefore the benchmarks move somewhat closer together as the budget is increased. However, to have fair power distribution, pullHipushLo actually employs kind of a prioritization as well, where benchmarks are preferred in their memory boundness order (mcf > art > ammp > crafty). Moreover, once again this policy can choose non-monotonic behavior with different budgets, as for different budgets different amounts of slack can enable different choices that satisfy better power balancing. On the right-hand side, we show the MaxBIPS policy, which actually optimizes for throughput. In this case, we once again see fairly regular behavior, but once again, this policy inherently assumes a priority. It actually prefers the benchmarks in the order of their cpu-boundness (crafty > ammp > art > mcf). Therefore, it performs in an almost completely inverse way to pullHipushLo.

In Figure 7, we show a final overall comparison of all the experimented policies. We show these in terms of *policy curves*, where for each policy we draw the overall system performance degradation—with respect to all Turbo execution—vs. different power budgets. Therefore, the policy which leads to least degradation for a given budget works better in that budget scenario. In addition to the policy curves, we also show the total powers for each budget in comparison to the target budget to fit (dotted straight line). This serves as a 'sanity check' to confirm the policies actually fit into the given power budget.

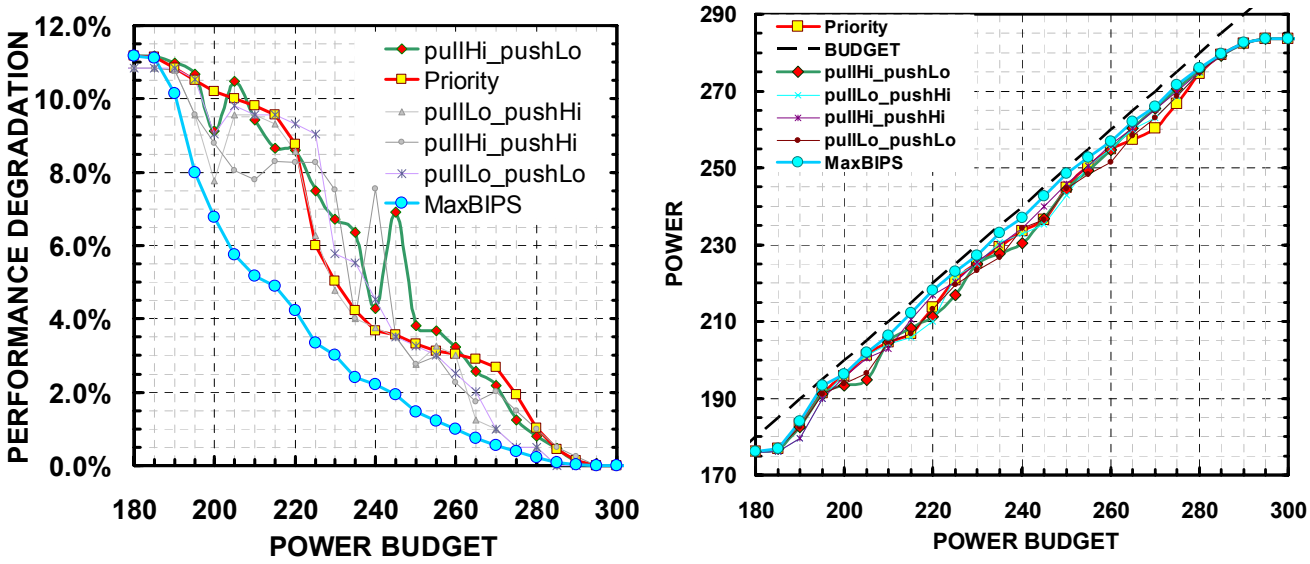


Figure 7. Policy curves and power for different policies.

The curves of Figure 7 show, for all the given budgets, MaxBIPS performs significantly superior for performance—as its target is tailored for throughput. Also all policies fit closely to the assigned power budgets.

In addition to the policy curves, here we also revisit our initial target of 3:1 ΔPower:ΔPerformance ratio. In Figure 8, we show the ΔPower:ΔPerformance ratios curves for the three described policies. In each plot, we also show the target 3:1 ratio curve as the dotted straight line. Our DVFS based power modes and experimented policies all show very good ΔPower:ΔPerformance ratios, right on the 3:1 ratio target. In the case of MaxBIPS, GMU actually achieves significantly better than the 3:1 ratio with DVFS.

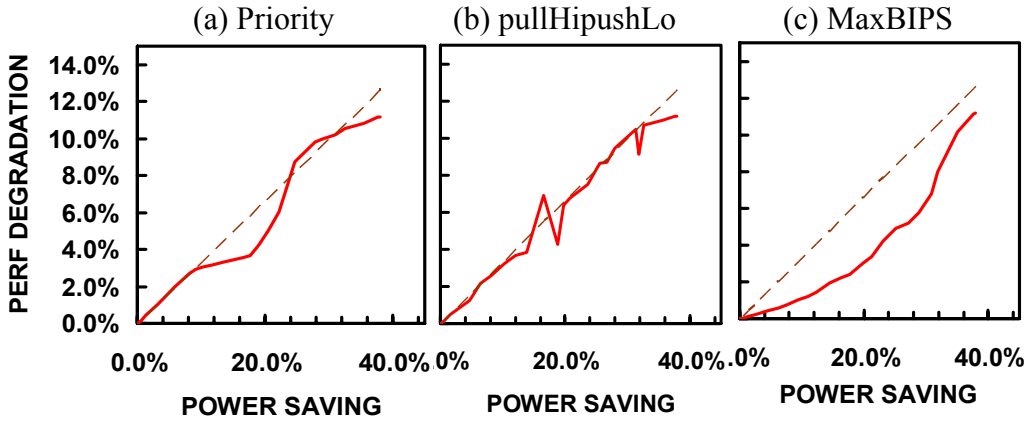


Figure 8. Hitting 3:1 ΔPower:ΔPerformance ratio with GMU policies.

Finally, in Figure 9, we show an example budget case, with the MaxBIPS benchmark, demonstrating how GMU works to meet power budgets, while optimizing for throughput. At each explore time, the best mode combination is computed that meets the budget. However, there can be small regions in each case that can exceed the budget at small locations as the unprecedented

application behavior changes can lead to increased instantaneous system powers, until the next explore time. In this example, we also see how MaxBIPS favors crafty and penalizes mcf.

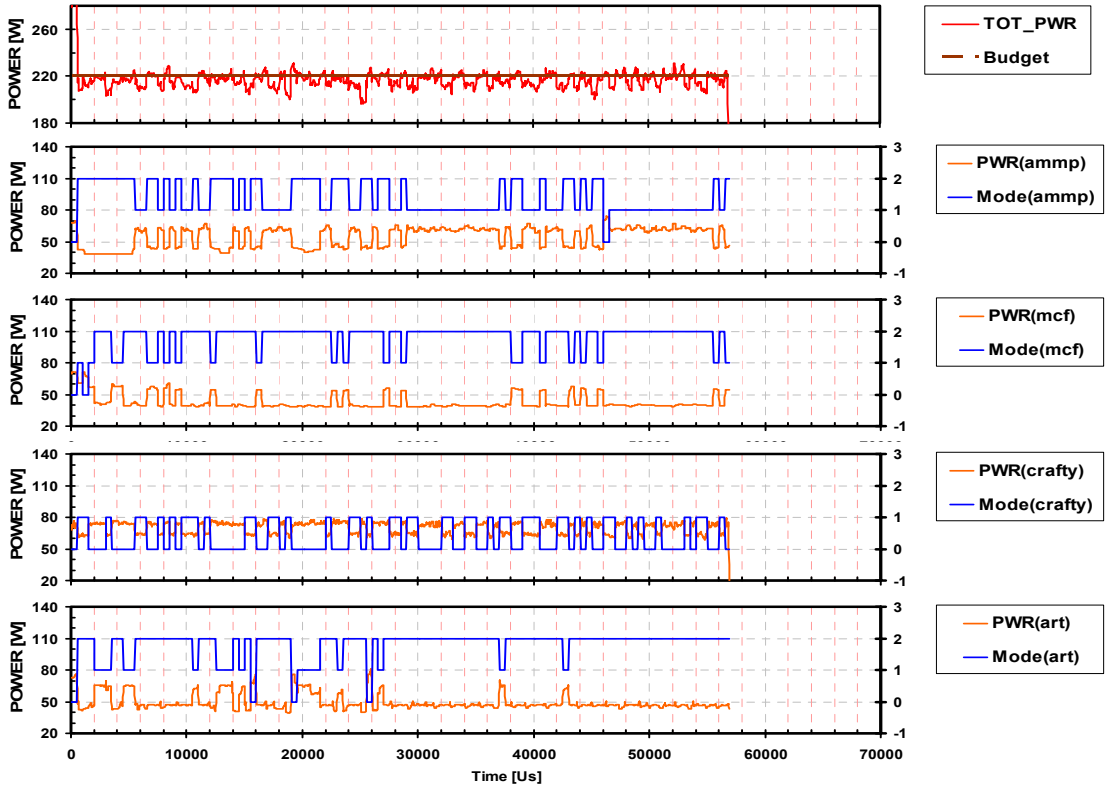


Figure 9. Execution timeline of 4 benchmarks and total power with MaxBIPS policy, for a 220W power budget. Right-hand axes represent modes with Turbo=0, Eff1=1 and Eff2 =2.

4.2 Estimating Different Mode Behavior

In the policy discussions of previous section, we have inherently assumed that the GMU has the knowledge of power/performance behavior of applications in different modes. So that, based on this information, policies can choose among different mode combinations based on their merits. However, this knowledge is not directly available to GMU. The general approach in many dynamic management schemes is to perform small-scale "explorations" of each mode to deduce the application behavior in these modes, or to somehow predict this behavior from past history. For a heavy-handed application like DVFS, this exploration approach is essentially prohibitive. Overheads lead to diminishing returns. The alternative approach is to assume a previously seen behavior history in a specific mode as the persistent behavior in that mode. However, this has unreliable outcomes, as there is no guarantee that a mode is previously visited under the given policies—unless there exist periodic explorations—and since relying on past history can be dangerously misleading for a strict power budget constraints.

As we have previously shown, DVFS has a very attractive property; different mode behaviors can be *a priori* estimated with reasonable accuracy with the analytical relations. Therefore, in all our analyses, we actually use this computational approach to estimate behavior in different modes.

This alleviates any necessity for exploration, and provides good confidence in our mode selection for budget fitting.

We represent the characteristics of each mode in terms of *Power* and *BIPS Matrices* as shown in Table 3. For an N-core CMP system with three power modes two Nx3 matrices can completely characterize all mode behaviors. For the Power Matrix, different power behaviors of modes can be characterized by applying the cubic scaling relation. For the BIPS Matrix, BIPS values can be computed with a linear scaling.

Power Matrix			
	Turbo	Eff1	Eff2
Core1	$P1_T=P1/(0.95^3)$	P1	$P1_T*(0.85^3)$
:	:	:	:
CoreN	PN_T	$PN_T*(0.95^3)$	$PN_T*(0.85^3)$

BIPS Matrix			
	Turbo	Eff1	Eff2
Core1	$B1_T=B1/(0.95)$	B1	$B1_T*(0.85)$
:	:	:	:
CoreN	BN_T	$BN_T*(0.95)$	$BN_T*(0.85)$

Table 3. Power and BIPS Matrices with DVFS.

For our original mode definitions, power and BIPS values of different modes can be estimated by scaling with 0.95 and 0.85 as shown in Table 3. For example, for if core1 is in Eff1 mode, corresponding Turbo and Eff2 power and BIPS values can be approximated as shown in the core1 row of the two matrices. As the power modes and number of cores are known in design time, all these values can be hardwired. Thus, the matrix computation can be done in parallel fashion with GMU specifying which column propagates to other modes.

Inevitable, these estimations are prone to errors. Over the SPEC suite, this is relatively insignificant with 0.1-0.3% estimation errors. For BIPS values, the errors are 2-4%. The higher errors for BIPS are due to asynchronous memory latencies. For powers, the errors stem from slightly changing utilizations at delta sim cycles in different modes. As the more important estimate is power—due to budget constraints, this approach works very well in our DVFS scenario. In our experiments we use this approach to choose appropriate power modes.

4.3 Upper Bounds in Policy Efficiency with Oracle Mode Selection

To assess the efficiency of the policies from an optimal throughput perspective, here we describe the upper bounds to achievable efficiency with "oracular" knowledge. For the oracle based mode selection, at each explore period, we look at the future execution until the next explore time, i.e., 500us forward. We generate our oracle BIPS and Power matrices based on this knowledge, and then choose the maximum performance mode combination that satisfies the power budget.

To be stricter with the oracle behavior, we actually experiment with cases where oracle fits a power envelope slightly higher than the power budget, we denote this with a *SHIFT*. For example, an oracle with SHIFT 5, *SemiOracleShft5*, tries to fit power into an envelope of POWER_BUDGET+SHIFT Watts. The reason for this approach is, fitting to budget every 500us can be more stringent, for overall execution. We demonstrate this with in the power slack plots of Figure 10. Here, oracles subject to different shifts are compared to the power slack of MaxBIPS. An oracle with shift 8 exceeds budget for all reasonable budgets, while some other oracles have actually more slack than MaxBIPS at certain budgets.

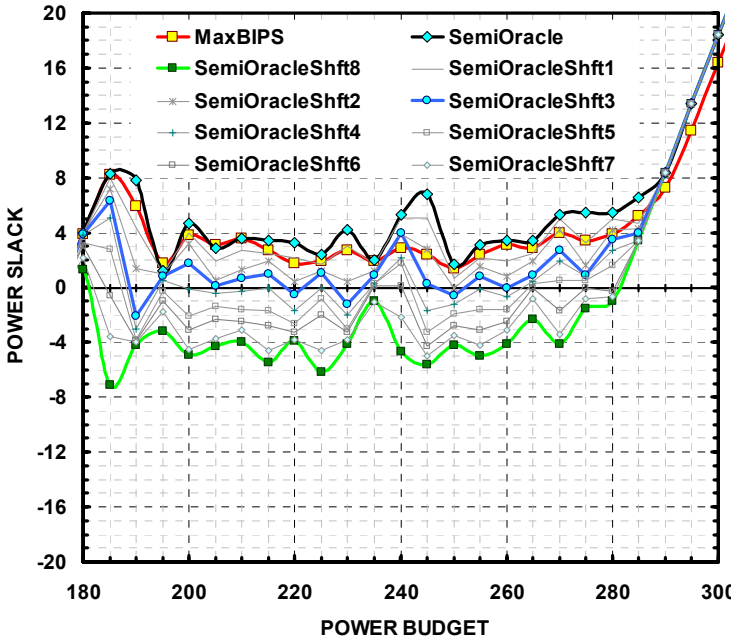


Figure 10. Power slack for oracles with different shifts, and for MaxBIPS.

Overall, there is no single shift value,that minimizes the slack for all budgets. Therefore, for the strict upper bound comparison, we merge these oracles with shifts 0 to 8 such that, at each budget, we choose the one that minimizes slack, but doesn't overshoot. We refer to this final representation as *OracleMERGE*. In Figure 11, we show the power slack for OracleMERGE, and a few of the previous policies. In this case, oracle provides a strictly tighter fit to budgets than all policies. In the figure, we also show the policy curve for the OracleMERGE, together with the previous policy curves. Although it performs slightly better than all policies, it is easily seen that MaxBIPS lies within 1% of this strict oracle. This shows, our MaxBIPS policy, with throughput optimized policy selection and predictive Power and BIPS matrices perform comparably to a strict oracle case in power management.

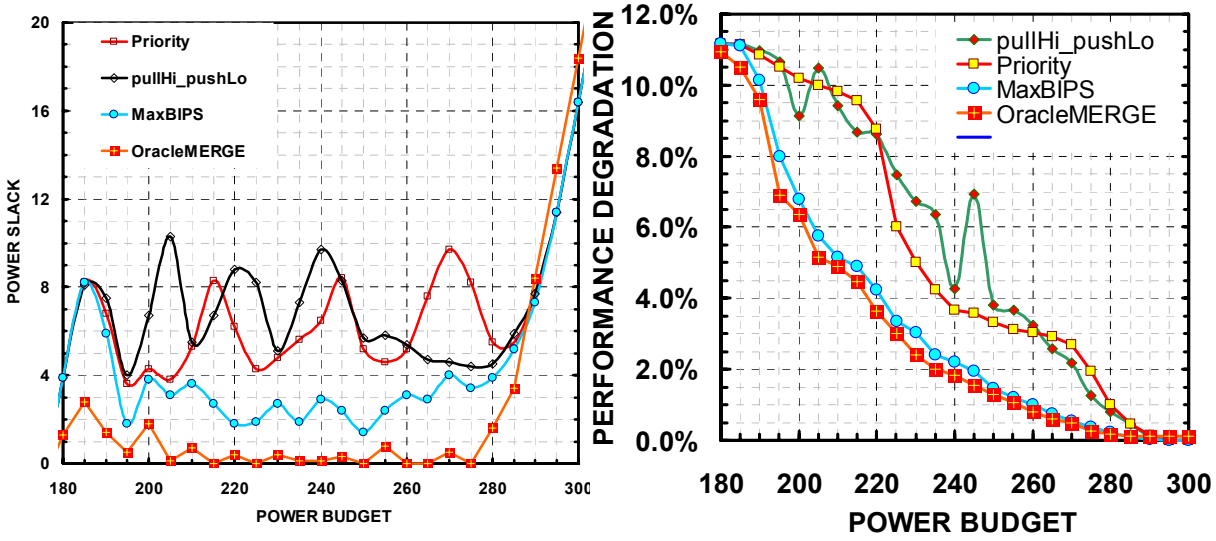


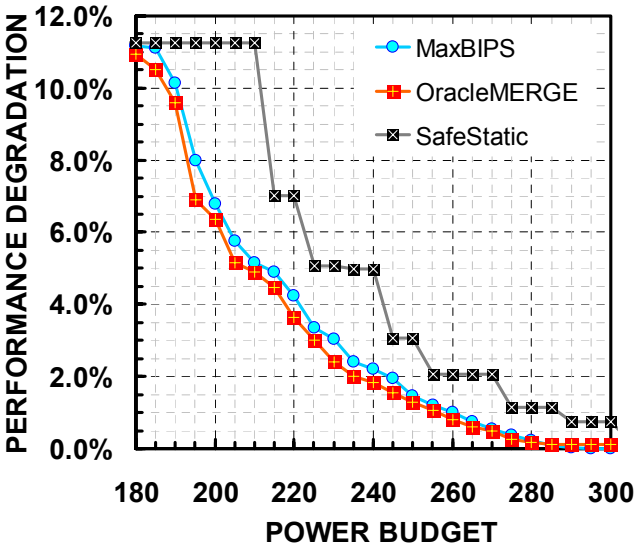
Figure 11. Power Slack and policy curve for OracleMERGE in comparison to policies.

4.4 Lower Bounds with Static Mode Assignments

One extreme method of the global power management consists in a *static* power assignment for each core. This can be explored in two different ways. First, we compare separate static assignments for each core, still based on 'oracle' knowledge and optimal core-benchmark assignments. We call this, *idealized static management*. Second, a more realistic scenario is considering a 'heterogeneous' core with different power-mode cores and investigate thread assignments. We call this *realistic static management.*

To explore the first scenario, idealized static case, we look at the overall native executions of each benchmark. Then, for each budget, we choose the mode combinations that maximize throughput, while satisfying budget requirements. This "idealistic" case relies on several assumptions. It assumes OS knows how to schedule each to which core for optimized results. We assume the underlying cores are always available for the optimal mode combination. For example, for one budget, we assume we have 3 Eff2 and 1 Turbo core, for another budget, we assume we have 3 Turbo and 1 Eff1 core. Inherently, as we schedule based on overall native executions, this approach entails oracle knowledge of whole application execution behavior.

In Figure 12, we show the policy curve for such an idealistic static case, *SafeStatic*. We also show, for each budget, the assumed underlying core configurations. Even under such considerations, static case performs significantly worse. This shows the impact of dynamic management potentials.



BUDGET	CORE1(ammp)	CORE2(mcf)	CORE3(crafty)	CORE4(art)
205.0000W:	Eff2	Eff2	Eff2	Eff2
210.0000W:	Eff2	Eff2	Eff2	Eff2
215.0000W:	Eff2	Eff2	Eff1	Eff2
220.0000W:	Eff2	Eff2	Eff1	Eff2
225.0000W:	Eff2	Eff2	Turbo	Eff2
230.0000W:	Eff2	Eff2	Turbo	Eff2
235.0000W:	Eff1	Eff2	Eff1	Eff2
240.0000W:	Eff1	Eff2	Eff1	Eff2
245.0000W:	Eff1	Eff2	Turbo	Eff2
250.0000W:	Eff1	Eff2	Turbo	Eff2
255.0000W:	Turbo	Eff2	Turbo	Eff2
260.0000W:	Turbo	Eff2	Turbo	Eff2
265.0000W:	Turbo	Eff2	Turbo	Eff2
270.0000W:	Turbo	Eff2	Turbo	Eff2
275.0000W:	Turbo	Eff2	Turbo	Eff1
280.0000W:	Turbo	Eff2	Turbo	Eff1
285.0000W:	Turbo	Eff2	Turbo	Eff1
290.0000W:	Turbo	Eff2	Turbo	Turbo
295.0000W:	Turbo	Eff2	Turbo	Turbo
300.0000W:	Turbo	Eff2	Turbo	Turbo
305.0000W:	Turbo	Eff1	Turbo	Turbo
310.0000W:	Turbo	Eff1	Turbo	Turbo
315.0000W:	Turbo	Turbo	Turbo	Turbo

Figure 12. Policy curve for idealistic static management and underlying core configurations at each budget.

The more realistic static management scenario, on the other hand, operates for a fixed core configuration for a given set of benchmarks. Therefore, it can operate only on smaller range of power budgets. The benchmark assignments can be considered as random, or scheduled by OS. Therefore, such core-benchmark pairings can only change at task switch intervals, on the order of milliseconds This is a higher level of granularity than GMU, and is considered as part of OS. Without oracle knowledge, the static benchmark assignments can lead to drastically different power/performance behavior. In Figure 13, we show an exhaustive case for a 4 core CMP, where the cores are assigned to [Turbo, Eff1, Eff2]. In this case, the benchmark assignments can cause varying performance degradation between 2 to 8%.

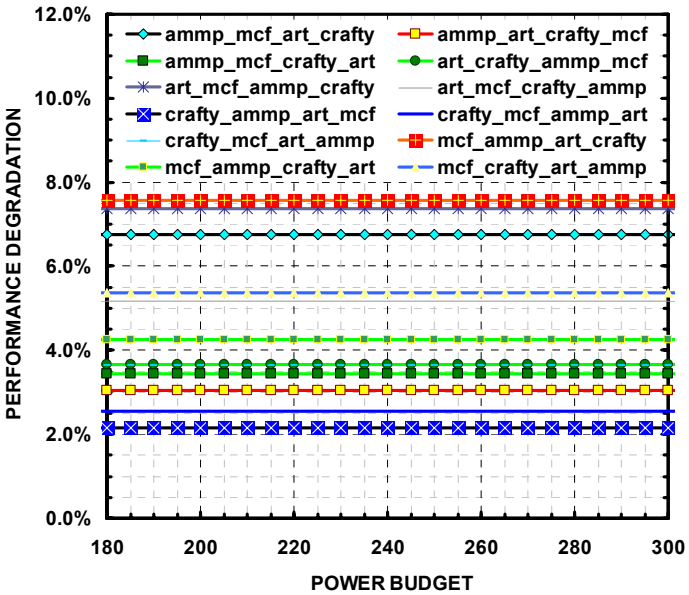


Figure 13. Effect of task scheduling to different cores on performance degradation.

In Figure 13, we see that assigning the memory bound application, mcf, to core1, and cpu-bound application Crafty, to core4 leads to worst performance degradation (highest line). The opposite of this (with crafty on core4 and mcf on core4) produces minimal performance degradation (lowest line). There exists a significant range among different benchmark assignments.

As rescheduling can be done only at OS switching granularities, without oracle knowledge, OS can realize a bad assignment at the end of a switching interval, and can switch tasks or reconfigure cores. This leads to higher time penalties compared to 500us GMU time granularity. As an approach like MaxBIPS is indifferent to benchmark-core pairings, initial scheduling decisions do not affect the management outcomes.

4.5 Chip-Wide DVFS

Last, we compare our results with a chip-wide DVFS scenario. Chip-wide DVFS has very appealing features for implementation, as there is no synchronization problem across cores which simplifies both software and hardware implementation. For this case, all cores transition together into Turbo, Eff1 or Eff2 modes at each explore time based on budget constraints.

In Figure 14, we show the application of chip-wide DVFS for a 250W budget on a 4-core CMP with ammp, mcf, crafty and art. In Figure 15, we show the same experiment with ammp, crafty, art and sixtrack.

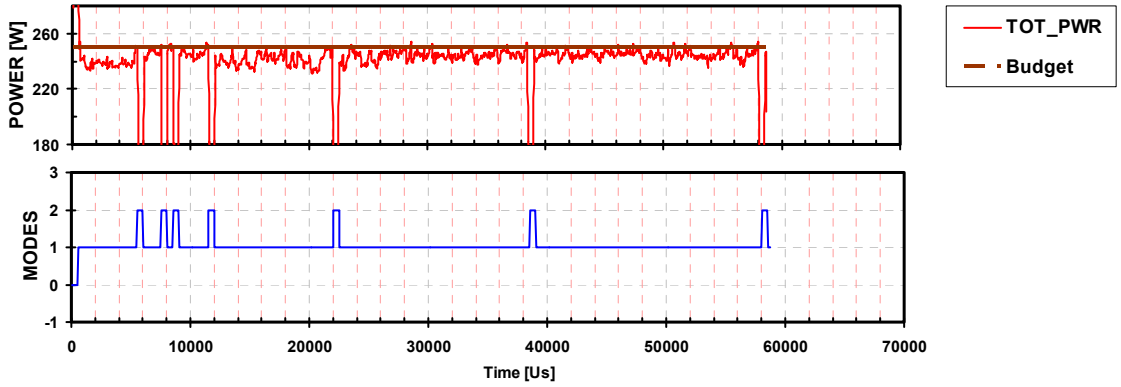


Figure 14. Chip-wide DVFS for 250W budget with ammp, mcf, crafty, art.

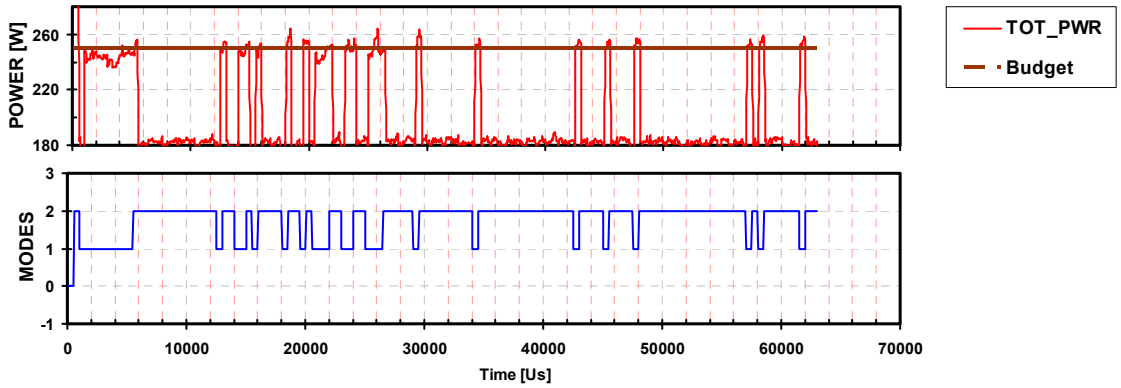


Figure 15. Chip-wide DVFS for 250W budget with ammp, crafty, art, sixtrack.

As seen in Figures 14 and 15, chip-wide DVFS has completely different implications for different set of benchmarks. For the first case, it fits the power envelop really well, as the total chip power meets around 250W for the first set of benchmarks when they run in Eff1 mode. In the second case, same budget results in a drastically reduced power/performance behavior, with one change in the benchmark set. In the second case, as chip power is usually slightly higher than 250W, all cores are penalized tremendously to run in Eff2 mode. In both cases, the downside of chip-wide DVFS is clearly conveyed, with CMP systems, you pay a huge penalty for small budget overshoots. Obviously this kind of monolithic global management has linearly growing negative impact with more aggressive scale-out scenarios. Also a comparison of the two figures shows, global DVFS cannot guarantee generic good performance for a budget with varying workload combinations. This effect is also emphasized with scale-out due to more combinations of variations.

Finally, in Figure 16, we show how the policy curve with chip-wide DVFS compares to previous cases, for the ammp, mcf, crafty, art workload combination. In the policy curve and budget plots, both the increased performance degradation and power slack are clearly demonstrated.

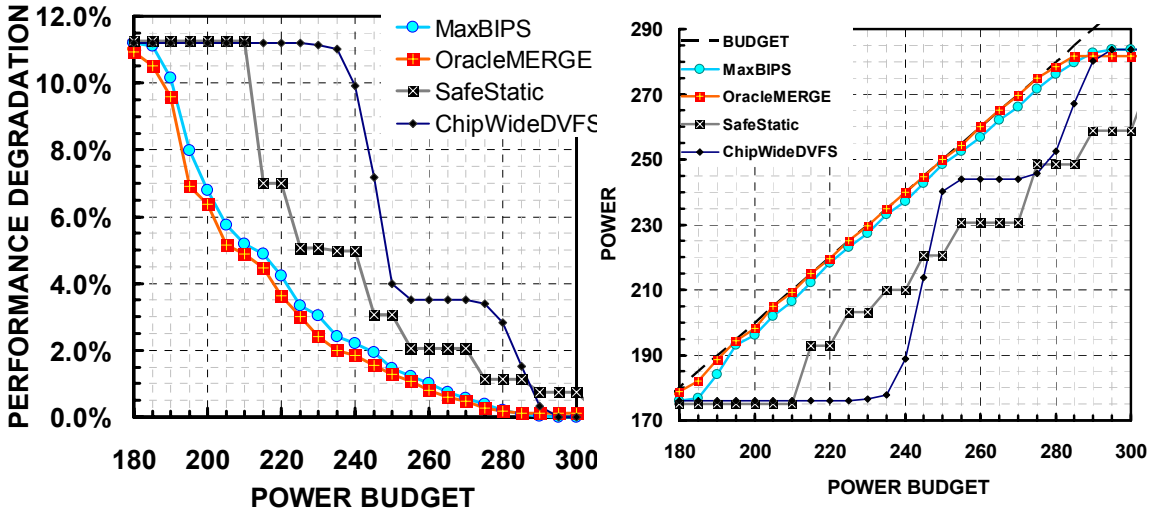


Figure 16. Policy and power budget curves for chip-wide DVFS.

Note that, despite the unpromising results for the presented cases, chip-wide DVFS has certain advantages in terms of implementation simplicity. Especially for small scale-out cases such as a 2 way CMP chip, it might be possible to get comparable results with this approach, probably by introducing more power modes. Further research is needed to characterize these benefits and to evaluate the feasibility of having more voltage and frequency domains.

4.6 Incorporating Transition Overheads

In our previous analyses, we have not discussed the overheads of mode-transitions. Although the overheads do not overwhelm the benefits for our large scale 500us explore times, they have observable effects in the power/performance behavior. Here, we describe the overheads and refine our methods to incorporate their effects.

Based on specifications from previous products and research, we choose a realistic DVFS transition rate of 10mV/us. Thus, the transition overheads for our three power modes are computed as shown in Table 4.

Turbo Vdd (f=1.1GHz)	Eff1 Vdd (f=1.045GHz)	Eff2 Vdd (f=0.935GHz)
1.3V	1.3*0.95=1.235V	1.3*0.85=1.105V
Δ=65mV → **6.5(7) Us**		
Δ=195mV → **19.5(20) Us**		
	Δ=130mV → **13 Us**	

Table 4. DVFS transition overheads for the three power modes.

Table 4 shows, the transition overheads for the three modes are on the order of 10us. Therefore, compared to our 500us explore times, these have relatively low overheads ranging from 1 to 4 %. However, despite the small magnitude of overheads, they can still inhibit the improvements with our policies. For example, for the greedy case of MaxBIPS, a possible scenario is as follows. Assume current mode combination is [Turbo Eff1 Eff2 Eff1] with a throughput of 3.6 BIPS. At the explore time, MaxBIPS decides to switch to a new combination [Eff2 Turbo Turbo Eff2] with a predicted throughput of 3.7 BIPS. However, considering the overheads of transitioning the actual observed BIPS goes down to 3.56, thus overriding projected benefits.

To accurately account for this effect, we devise a new policy, *BestCostBIPS*, which incorporates the overheads into optimization objective. The flow for this policy is depicted in Figure 17.

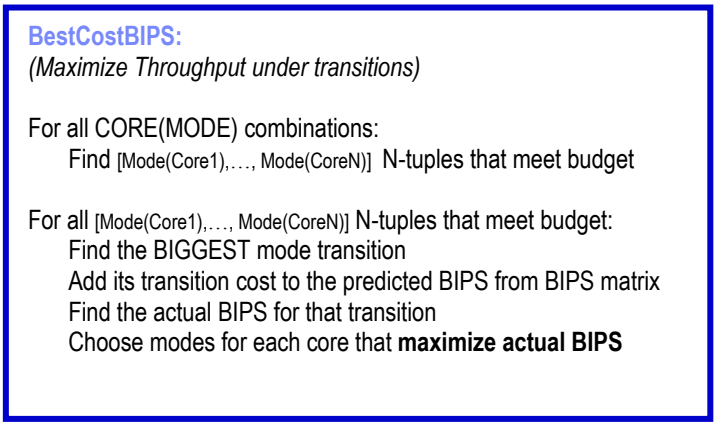


Figure 17. BestCostBIPS policy description.

The implementation of BestCostBIPS is not significantly different from MaxBIPS. Following from similar arguments, at design time, the cost of switching among modes is well known as well as the BIPS and Power Matrix scalings. Therefore, same parallel BIPS matrix computation can be performed, only with additional scaling factors for BIPS values such as 500/507, 500/513 and 500/520 for our choice of parameters. For example, referring to Table 3, instead of scaling $P1_T$ with 0.85 for BIPS matrix, Eff2 estimate, we scale with 0.85*(500/520).

To evaluate the effect of overheads and the efficiency of BestCostBIPS policy, we consider a conservative transition scenario. At each explore time, if there is a mode transition, we find the longest transition cost among all cores and assume all cores are stalled during this period. We assume chip produces 0 BIPS—attributed to benchmark execution—during transitions, while power remains the same. Such a power assumption is more realistic to avoid swings that can lead to di/dt hazards. Penalizing all cores with the largest penalty is not the most efficient approach, but much more beneficial to keep cores synchronized for explorations.

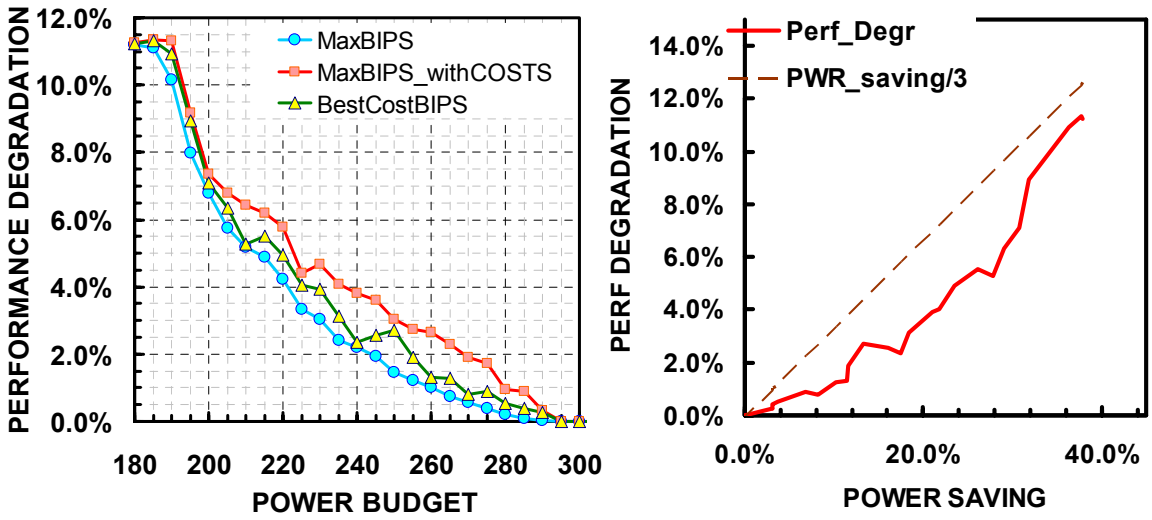


Figure 18. Policy curves and ΔPower:ΔPerformance curve under transition costs.

In Figure 18, we show the policy curves for MaxBIPS and BestCostBIPS with transition costs. We also include the original zero overhead policy for reference. Overall, we see the impact of transitions on performance degradation with two MaxBIPS curves. In comparison, BestCostBIPS consistently performs better than the costed MaxBIPS, and it comes within 1% of the zero overhead MaxBIPS. In addition to these, we also show the 3:1 ΔPower:ΔPerformance ratio with the BestCostBIPS policy, which shows we still over-achieve 3:1 ratio with DVFS based power management with GMU These results show, first, the impact of transition costs is not drastic, and our GMU power management methods still perform well; second, our new policy, BestCostBIPS performs superior under realistic transition overheads.

5. Conclusion

Our presented research describes a new perspective to dynamic power management for CMP systems, under the context of a global power management unit (GMU). The GMU is exposed to the activity of all cores in a system, and therefore, can decide upon per-core actions to meet global chip constraints and for both local and global objectives.

Our experiments investigate different power management implementations, and demonstrate a DVFS approach achieves a 3:1 ΔPower:ΔPerformance ratio for power savings and performance degradation. We describe a fast and scalable CMP power/performance analysis tool to investigate global power management policies. Our analyses show, global management with GMU provides a good balance between power/performance under varying workload behavior. Such dynamic global

power management can avoid the inefficiency of worst-case designs by reducing power envelopes or increasing on chip core integration, with reasonable single-threaded performance sacrifice. Our policy experiments show the different trade-offs between fairness and throughput. Our best performing policy, *MaxBIPS*, achieves performance degradations within 1% of a conservative oracle, exceeding 3:1 ΔPower:ΔPerformance ratio in most operating regions. We show, dynamic management policies perform significantly better than static power management or chip-wide DVFS. Under transition overhead considerations, our best performing policy, *BestCostBIPS,* comes within 1% of MaxBIPS without costs, showing the feasibility of GMU actions under realistic overhead constraints. In all the policies experimented, the predictability of DVFS based techniques alleviate any overheads related to mode explorations. Our mode predictor structures, *Power* and *BIPS Matrices* can estimate the behavior across different modes of an application with .2% and 3% errors respectively.

Overall, this research investigates dynamic management possibilities with a dedicated GMU architecture. Our results show the potentials and applicability of such a method. We believe as more aggressive chip- and system-level scale-out strategies are followed, the benefits of global power management will be correspondingly emphasized, encouraging the incorporation of such global management techniques in the next generation systems.

6. Future Research Directions

There exist several additional avenues of research that will enlarge the depth and breadth of our presented research. First, for the power mode explorations, an important alternative is to consider core-wide resource scaling. In such a case, all possible array structures, functional units and queues can be scaled for Eff1 and Eff2 mode definitions. In such a case, it might be compulsory to integrate Vdd gating to baseline to achieve 3:1 ΔPower:ΔPerformance ratios.

Second, an alternative can be explored by considering fixed, heterogeneous cores. These can be developed as similar cores with different voltage and frequency islands, or with different size cores. One fundamental difference with this approach is, as the cores are fixed, global management can be done by shifting threads. For this purpose, GMU functionality might have to be shifted to OS, but due to granularity differences, a smaller scale solution might evolve that can be implemented via GMU.

Third, besides described experiments, a global controller can also participate in dynamic reliability management. Under today's process variability, there exist significant probabilities that certain modes of a core can be non-functional at given (V,f) setting. That is, a certain core might not work at Turbo for a given frequency. In such case GMU can be configured to avoid those mode combinations and can help binning of yield into different available-modes categories. The implementation and evaluation of such a scenario is straightforward with GPAT and our Power and BIPS Matrices. Simply, an unachievable mode results in reducing initial mode combinations to an available subset, and our policies can base their decision based on both budget and mode constraints.

Finally, a further progressive step in extending this research is to implement the presented scenarios on a dynamic CMP simulation environment. In such a case, further policy evolutions can emerge by considering also various address and bus conflict constraints.

REFERENCES

1. A Method and System of Reducing and Limiting Maximum and Average Power in a Chip Through a Power and Performance Control Unit. A. Buyuktosunoglu, P. Bose, C. Cher, P. Kudva. IBM Pending Patent, dated as 02/04/2005.