Dynamic Power Redistribution in Failure Prone CMPs

Paula Petrica
Computer Systems Laboratory
Cornell University

Jonathan A. Winter
Google, Inc.

David H. Albonesi
Computer Systems Laboratory
Cornell University

Abstract

Future chip multiprocessors (CMPs) will be capable of deconfiguring faulty units in order to permit continued operation in the presence of wear-out failures. However, the unforeseen downside is *pipeline imbalance* due to other portions of the pipeline now being overprovisioned with respect to the deconfigured functionality. We propose *PowerTransfer*, a novel CMP architecture that *dynamically redistributes* the chip power under pipeline imbalances that arise from deconfiguring faulty units. Through rebalancing – achieved by temporary, *symbiotic deconfiguration* of additional functionality within the degraded core – power is harnessed for use elsewhere on the chip. This additional power is dynamically transferred to portions of the multi-core chip that can realize a performance boost from turning on previously dormant microarchitectural features. We demonstrate that a realistic PowerTransfer manager achieves chip-wide performance improvements of up to 25% compared to architectures that simply deconfigure faulty units without regard to the resulting inefficiency.

1 Introduction

Future multicore microprocessors built in sub-32nm technologies will present major design challenges for computer architects. One serious concern is the possibility of *aging defects* due to various transistor and wire failure mechanisms which become more prominent with technology scaling [3]. The possibility of wear-out failures and manufacturing defects is already forcing multicore architects to include the capability of deconfiguring various features that may become faulty, to permit the system to operate in a degraded state in the event of a hard error. While the most obvious redundancy to exploit in a multicore microprocessor is at the core level, with many failures possible over the lifetime of a product in the future – and some failures even present at product shipment [3] – the trend is towards finer grain levels of redundancy that permit each core to operate in a degraded state.

One issue that has not yet been addressed is the *pipeline imbalance* that arises within a core with deconfigured functionality. A modern pipeline is highly tuned to create a good balance of hardware features within an individual stage, and from stage to stage. Similarly, memory hierarchies are carefully designed to be well-balanced in terms of bus and cache parameters at every level of the hierarchy. Deconfiguration of a full unit (such as an ALU) or part of a unit (such as a way of a cache) may now make another unit to be *overprovisioned* in terms of the extent of its features, but the degree of overprovisioning is application dependent and not obvious to exploit. Thus, current fault-resilient microarchitectures simply deconfigure the faulty unit and live with the resulting hardware imbalance that may arise.

But for some applications, this imbalance may be severe. As we show later in Section 4, the loss of functionality due to a fault and subsequent deconfiguration can lead to significant imbalance in some applications, while others see little or no effect. For the former, eliminating the imbalance through *symbiotic deconfiguration* of additional units saves significant power that can be better used elsewhere on the chip.

In this paper, we present *PowerTransfer*, a novel multicore architecture that exploits both the pipeline imbalances that arise due to deconfiguration of a faulty structure, and the deconfiguration capability built-in to overprovisioned structures to permit failover. PowerTransfer eliminates the power inefficiencies that arise due to hardware deconfiguration through additional deconfiguration of hardware that is no longer efficiently used by the current running application. Because chips operate within a maximum power limitation, the power saved in this particular area of the chip can be *transferred* to another portion of the die that can achieve a performance boost with the added power. Our results demonstrate the potential for large performance gains over designs that simply deconfigure faulty units without concern for the resulting imbalance and loss of power efficiency.

2 PowerTransfer Architecture

Figure 1 shows an overview of the PowerTransfer system. During application execution, the four step cycle shown in Figure 1 is repeated periodically as the hardware and software characteristics change. The PowerTransfer Runtime Manager (PTRM) is informed of faulty units that are deconfigured. During program execution, the PTRM determines which symbiotic deconfigurations can save power with minimal performance costs. The saved power is then used to boost performance. The simplest approach is to boost the performance of the affected core, but this might not be the best use of chip-wide power. Rather, PowerTransfer considers other possible chip-wide candidates for the additional power.

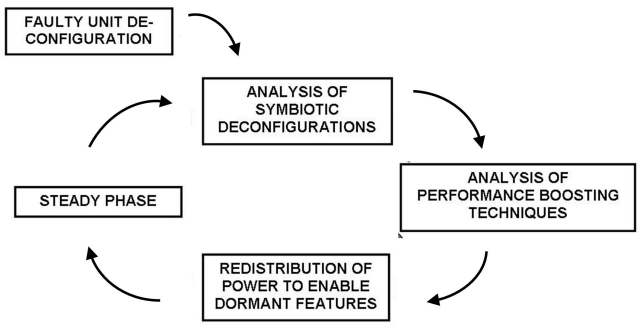

Figure 1: PowerTransfer system.

In actuality, the situation is much more challenging than in this simple example as there may be faults in each core and several possible performance boosting techniques from which to choose on each core. Moreover, the effectiveness of symbiotic deconfiguration and performance boosting is application-dependent, which implies that new decisions must be made at the same time granularity as the Operating System scheduler time slice, which can be on the order of 100ms.

2.1 Symbiotic Deconfiguration

We assume a baseline CMP that can detect the onset of a permanent failure, determine the faulty unit, and deconfigure the entire unit or some portion of the unit in order to keep the affected core operational. The inherent redundancy of processor structures permits partial deconfiguration of faulty units in order to ensure continued, albeit degraded, operation in the presence of a permanent failure.

Processor pipelines are carefully tuned to create a good balance of hardware features within an individual stage, and from stage to stage. Thus, the deconfiguration of a faulty unit may lead to pipeline imbalances due to other fully functional parts of the processor now being overprovisioned with respect to the deconfigured units. If these overprovisioned units can be accurately identified, then they can be *symbiotically deconfigured*, or partially or entirely deconfigured even though they are fault-free, with little expected performance impact, and often larger power savings.

The physical process of symbiotically deconfiguring additional hardware is straightforward; we can simply leverage the deconfiguration capability built-in at design time for fault tolerance. The more challenging task is identifying hardware pairs that have correlated performance and thus are good candidates for symbiotic deconfiguration in terms of performance loss versus power savings. In PowerTransfer, the front-end, integer and floating point back-ends, and the load/store unit are deconfigurable by horizontal slices through the pipeline or *lanes*[1]. For instance, for a four-way front-end with four lanes, a hard error in one lane reduces the front-end to three-way. This coarse-grain approach serves as a middle ground between deconfiguring an entire core – for which the performance cost is exceedingly high – and deconfiguring at fine granularity – for which the built-in overhead for deconfiguration and identification of the faulty portion of the faulty unit is also very high.

Figure 2 illustrates an example of symbiotic deconfiguration. A fault in a particular unit causes an entire lane – encompassing a horizontal slice through all the units within the same region (FE, BE, or LSQ) – to be deconfigured. Symbiotic deconfiguration of the other two regions also occurs at lane-level granularity.

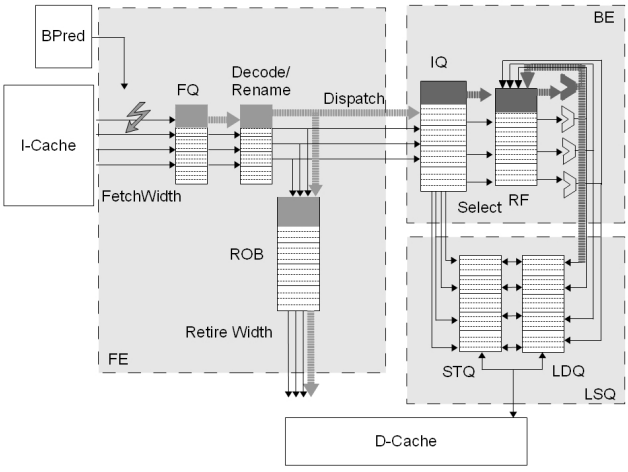

Figure 2: Example fault and symbiotic deconfiguration for a fault in fetch. The front-end lane associated with the fault is initially disabled, and the PTRM symbiotically deconfigures a lane of the back-end.

[1]We also include a sub-bank of the associated queues within a lane, even though they are not technically part of the pipeline "width."

2.2 Performance Boosting

Once a margin of additional available power has been accumulated by exploiting both permanent unit deconfiguration (due to a fault) and symbiotic deconfiguration (for pipeline rebalancing), this power is distributed among the chip components in order to boost performance. This is accomplished by temporarily enabling previously dormant hardware features within the limits of the global power budget.

By definition, a performance boosting technique does not improve performance for most applications; otherwise, the technique would be built-in to the design by default. Rather, these techniques improve what might be deemed *performance corner cases*, snippets of particular applications. While the speedup may be significant in these situations, in most cases, the power cost exceeds the performance gain such that the technique is not enabled by default.

In PowerTransfer, several boosting techniques are implemented that collectively cover a range of performance corner cases such that there is ideally always some worthwhile boosting technique to engage given some harnessed power no matter what mix of applications are running.

While there are many potential approaches, we discuss three techniques that cover the spectrum of CPU, cache hierarchy, and memory performance-bound applications.

2.2.1 Boosting CPU Performance: DVFS

A well-known technique for boosting CPU performance is to scale up voltage and frequency. Microprocessors typically include multiple frequency and voltage domains, and recent research has shown the merits of separate domains for each core [7, 9]. The most straightforward approach is to use the saved power locally within the core with the faulty unit to compensate for the loss in IPC performance. Instead, PowerTransfer adds any locally saved power to the chip-wide pool of accumulated power for potential use in boosting other cores' performance. While this requires more complex chip-level power management algorithms, a performance gain closer to the global optimal can be found. For instance, it would be more worthwhile to boost the frequency and voltage of another core running a high IPC thread when the local core is running a memory-bound thread.

2.2.2 Boosting Cache Hierarchy Performance: Speculative Cache Access

For applications that are more limited by L1 cache misses, improvements in the access time of lower levels of the cache hierarchy would be more beneficial than DVFS. Speculative cache access can be a very effective means to boost performance at reasonable cost.

L2 caches are typically accessed in sequence after L1 lookup. To do so otherwise would greatly increase power consumption for relatively little overall performance gain. A performance boosting technique that requires little added hardware complexity is to speculatively send L1 requests to the L2 cache simultaneously in order to reduce the delay penalty in case of an L1 miss. We expect good improvements for applications that miss in the L1 but hit in the L2.

The main drawback of this technique is the substantial additional power requirement, which mainly comes from unnecessarily accessing the L2 cache and amounts to increasing a core's power usage by 60% on average. In order to reduce this latter power consumption, we add a Load Miss Predictor, implemented as a two-bit saturating counter that is updated with L1 hit/miss information. This low-overhead predictor has little performance penalty yet reduces wasted L2 access power by 90% on average.

A similar cost-effective performance boosting technique is to access the L2 tags and data in parallel. Lower level caches such as those in the Itanium II and Alpha 21164 access the

tag and data arrays in sequence due to power concerns. Given additional harnessed power due to symbiotic deconfiguration, and an application that can achieve significant performance gains from parallel tag and data access, the L2 cache can be switched into parallel mode.

2.2.3 Boosting Memory Performance: Clear

Many speculative techniques have been proposed to boost the performance of memory-bound applications, but these may come at a prohibitive power cost. When there is potential benefit (i.e., many long latency loads) and sufficient power has been harnessed, we engage Clear mode [10]. In this mode, the registers are checkpointed, stores are buffered in the store queue, loads are speculatively early retired, and the predicted values are supplied to their destination registers. Through these mechanisms, dependency chains following a long latency load complete early, and processor resources are freed for use by non-dependent instructions.

Although Clear can significantly boost application performance, power consumption may increase significantly due to the additional structures required to implement the mechanism and the large boost in processor utilization it affords.

2.3 PowerTransfer Runtime Manager

The PowerTransfer Runtime Manager (PTRM) receives information on deconfigured components with hard faults, determines what components should be symbiotically deconfigured to save additional power, and allocates the harnessed power to boosting mechanisms on the different cores. In order to adapt to dynamic program behavior, the PTRM operates at a time granularity of tens to hundreds of milliseconds.

Operation at this time granularity also allows the PTRM to coordinate with the Global Power Manager (GPM). The GPM controls the frequencies and voltages of each core to maintain the chip-wide power budget. In order to address chip-wide power budget constraints, prior work has proposed coordinating power management decisions at a global level [7, 14]. The GPM acts as a fail-safe mechanism in instances where the PTRM underestimates the additional power cost of enabling a performance boosting techique. Such overshoots occur only 5% of the time in the sampling-based PTRM that we evaluate in Section 4.

3 Methodology

In order to evaluate PowerTransfer we use a highly modified version of the SESC [11] simulator augmented with Wattch [4], Cacti [13], and HotLeakage [16] to model both static and dynamic power consumption. We also modified the simulator to dynamically account for temperature dependent leakage power.

3.1 CMP Architecture, Workloads, and Degraded Configurations

To ensure that the baseline processor core was appropriately sized, we performed an exhaustive design space study to create a balanced baseline core microarchitecture with the parameters shown in Table 1. Deconfiguring a portion of this baseline design in the absence of a fault results in more than a 10% performance loss for multiple benchmarks.

We use this baseline to model a four-core CMP, each of which runs one of 13 SPEC CPU2000 benchmarks. We fast-forward each benchmark five billion instructions and run for a total time of 100ms, the granularity at which we periodically engage PowerTransfer. We create 100 randomly chosen four-benchmark workloads that run on 100 four-core configurations, each with a random single fault chosen from the three possible coarse-grain errors (FE, BE, LSQ). Benchmarks are not repeated for any given workload as this would tend to accentuate the benefit of our approach, but the same fault may occur in more than one core.

Front End	Branch Predictor: gshare + bimodal 64 entry RAS, 2KB BTB, 128 entry ROB fetch/decode/rename/retire 4-wide
Execution Core	Out-of-order, issue/execute 4-wide 80 Integer Registers, 80 FP Registers 32 entry Integer Queue, 24 entry FP Queue 32 entry Load Queue, 16 entry Store Queue 4 Integer ALUs, 1 Integer Mult/Div Unit 1 FP ALU, 1 FP Mult/Div Unit
On-chip Caches	L1 ICache: 8KB, 2-way, 2 cycle access latency L1 DCache: 8KB, 2-way, 2 cycle access latency L2 Cache: 2MB, private, 8-way, 10 cycle latency
Memory	200 cycle latency
Operating Parameters	1V Vdd, 4.0 GHz frequency

Table 1: Architectural parameters.

3.2 Symbiotic Deconfiguration

The core is divided into three regions: Front End (FE), Back End (BE) and Load Store Queue (LSQ), each of which has four lanes. The pipeline resources associated with each region are shown in Table 2. As an example, an error in one of the decoders disables an entire FE lane (one quarter of the fetch/decode/rename/retire widths, Fetch Queue, and ROB). The BE and/or the LSQ are symbiotically deconfigured if the power-performance tradeoff is attractive.

Front End	Back End	Load Store Queue
Fetch Width	Issue Queues	Load Queue
Fetch Queue	ALUs	Load Queue Ports
Decode Width	Select	Store Queue
Rename Width	Wakeup	Store Queue Ports
ROB	Register Files	
Retire Width		

Table 2: The pipeline regions and their affected structures.

3.3 Performance Boosting Techniques

We model three performance boosting techniques that cover CPU, cache hierarchy, and memory-bound applications.

Boosting voltage and frequency: Similar to Intel's Turbo Boost [15], we increase the operating voltage and frequency within the constraints of the overall power budget. We use four voltage and frequency levels above the baseline frequency and voltage, in 2.5% Vdd increments.

Speculative cache access: We implement two techiques for boosting cache hierarchy performance: performing L1 and L2 access in parallel, and performing L2 tag and data array access in parallel. We augment the energy per read/write hit and miss calculations in Cacti to reflect the difference between accessing the L2 tag and data sequentially or in parallel. We also enhanced SESC to allow for dynamically changing the access pattern of the cache hierarchy, that is, to either send L1 and L2 requests sequentially (only misses in L1 go to L2) or to speculatively send requests to the L1 caches to the L2 cache at the same time. This technique complements the DVFS core boosting technique in that it provides a much better performance benefit / power cost ratio but only for a subset of the benchmarks. Many of these fall into the mid IPC category for which boosting frequency and voltage has only a moderate performance improvement.

Clear: We augmented our simulator to correctly execute mispath instructions and implemented Checkpointed Early Load Retirement [10]. We model a Prediction Queue of 48 entries, up to four checkpoints, and a checkpoint allocation threshold of seven loads.

4 Results

4.1 Pipeline Imbalance and Symbiotic Deconfiguration

We first evaluate the pipeline imbalance that may occur due to a fault and subsequent deconfiguration, as well as the viability of symbiotically deconfiguring additional functionality to save power with little added performance cost. Symbiotic deconfiguration is effective if in the presence of a fault, the additional deconfiguration yields little additional performance loss relative to the added power savings; this indicates that the fault creates pipeline imbalance that, when corrected through symbiotic deconfiguration, permits significant power to be harnessed relative to the performance loss.

We evaluate power harnessing opportunities within a degraded core using coarse-grain deconfiguration as in Table 2.

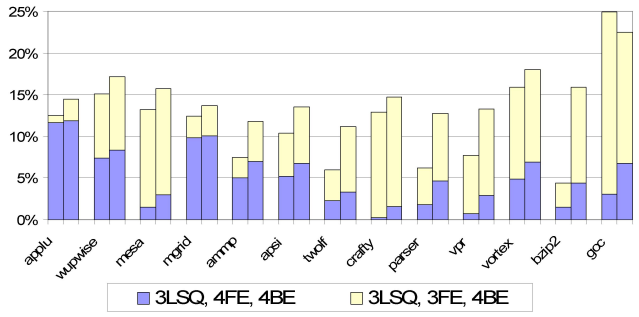

Figure 3: Performance loss (left bars) and power savings (right bars) due to an initial fault in the LSQ and with symbiotic deconfiguration of a FE lane.

Figure 3 shows the performance loss and power savings (due to gating the lane of the affected region) for an initial fault in the Load Store Queue, as well as the effect of symbiotically deconfiguring a lane in the Front End. The left bars show the performance loss while the right bars show the power savings. The lower sections of the bars denote the performance cost and power savings from deconfiguring one lane within the faulty region, while the upper stacked sections show the effects of symbiotic deconfiguration. Similar results were obtained for the Front End and the Back End.

We make two major observations from these results. First, the initial deconfiguration due to the faulty unit yields significant performance losses for some benchmarks, but also appreciable power savings in many cases. In most cases, the power/performance ratio is much less than two, indicating that the unit is not overprovisioned to begin with. However, given a fault, the power saved by deconfiguring the affected unit can be used to boost performance by some other means, even without symbiotic deconfiguration.

Second, additional symbiotic deconfiguration can yield a large power savings for a small additional performance loss (much greater than two to one), but for only a subset of the benchmarks (e.g. for bzip2 but not for gcc). In other words, large performance losses can be incurred by blindly deconfiguring additional units without regard for the characteristics of the running application. On the other hand, judicious symbiotic deconfiguration in cases of pipeline imbalance can be an effective means of harnessing additional power that can be used elsewhere.

4.2 Performance Boosting Techniques

In this section, we evaluate the characteristics of the three power boosting techniques described in Section 3.3, and compare them in terms of performance and power.

Figure 4 shows the percent performance improvement and

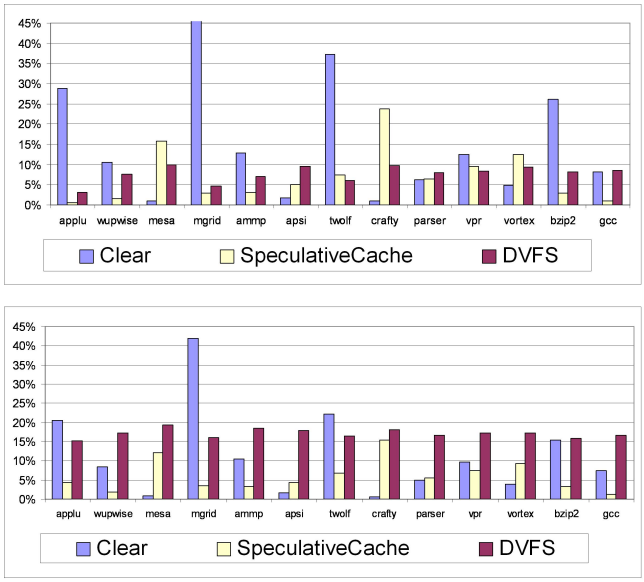

Figure 4: Performance improvement (top) and power consumption (bottom) for the performance boosting techniques.

percent power increase for the three techniques for each benchmark. No one technique is clearly superior across all benchmarks. While Clear has the greatest performance benefit for some benchmarks (applu, mgrid, twolf, bzip2), Speculative Cache and DVFS are the best techniques for other benchmarks (mesa, crafty, applu for the former, and apsi, parser for the latter). Thus, the decision of which combination of performance boosting techniques to engage is workload dependent.

The Power Performance Ratio (PPR), the ratio of the percent power increase to percent performance gain (relative to no boosting) is an effective metric for making this decision. The smaller the PPR, the more efficiently the technique uses the accumulated power. Even though Clear appears to be the most power hungry technique, its PPR is advantageous (average of 1) for our PowerTransfer system. On the other hand, DVFS has a high PPR relative to Speculative Cache and Clear. Therefore, as we show in the next section, DVFS largely serves as a backup technique to the more effective Clear and Speculative Cache boosting approaches.

4.3 PTRM Results

To evaluate the PTRM, we assume that the power budget for any particular benchmark-core combination is the total power used by that benchmark on that core in the absence of faults. We also assume that the maximum power budget of the four-way CMP is the sum of the power of all current benchmarks running on the cores in the absence of faults. This approach limits our results to only show improvements from the techniques we propose, rather than artificially introducing improvements from choosing a good power management technique. We evaluate our architecture with respect to the four-core configuration with the initial random errors without applying any performance boosting techniques.

4.3.1 Understanding the Fundamental Tradeoffs

In order to gain insight into the effectiveness of the different performance boosting techniques, the interactions between symbiotic deconfiguration and performance boosting, and the impact of locally versus globally managing the accumulated power, we developed a number of offline PowerTransfer managers. Given a set of initial deconfigurations (due to hard faults) and applications for the four cores, these managers have *a priori* knowledge of the performance benefits and power costs for each possible symbiotic deconfiguration and perfor-

mance boosting possibility. We model this perfect knowledge by calculating the global BIPS for all combinations of symbiotic deconfigurations and power boostings for the same 100ms time quantum, and pick the configuration that maximizes the geometric mean of all the cores' BIPS with respect to the baseline. It is important to note that the geometric mean acts as both a performance and a fairness metric at the same time. Thus we avoid penalizing low IPC applications to benefit high IPC ones.

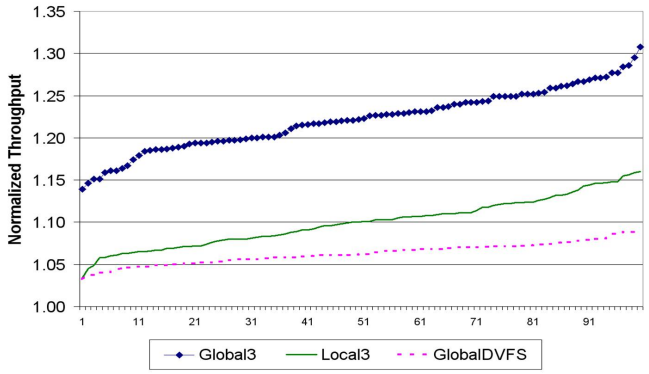

Figure 5: Throughput improvement with only DVFS used globally (GlobalDVFS), all three techniques used locally (Local3), and with all three boosting techniques used globally (Global3).

Figure 5 compares the improvement in throughput for 100 four-core configurations with random initial errors for a manager that globally employs only DVFS, one that globally employs all three performance boosting techniques (DVFS, Speculative Cache Access, and Clear), and one that locally employs all three techniques, i.e., any harnessed power from a core is only applied to boosting that core's performance.

The results of Figure 5 confirm the intuition from Figure 4 that DVFS is not sufficient to reap the available performance benefits. Rather, a number of techniques in combination is necessary to boost different application classes. With all three techniques, the chip-wide throughput is improved on average by 22.2%, while using DVFS as the sole boosting technique achieves only a 6.3% average increase in chip-wide throughput. The individual Speculative Cache and Clear techniques also fall far short, increasing chip-wide throughput by an average of 9% and 13%, respectively.

We assess the benefit of accumulating a global pool of power and applying it to the best combination of chip-wide boosting techniques by comparing Global3 with Local3, in which each core makes local symbiotic deconfiguration and boosting decisions. The 10% average throughput improvement for Local3 is significantly less than the 22.2% average improvement achieved by accumulating and distributing the power globally.

140 FE Errors

	GlobalDVFS	Global3	Local3
BEDeconf	63	98	66
LSQDeconf	18	20	9
NoDeconf	59	22	65

Table 3: Symbiotic deconfiguration decisions given an initial error, the available boosting techniques, and whether decisions are made locally or globally for best configurations.

Table 3 shows the deconfiguration decisions as a function of the initial error, the available boosting techniques, and whether the decision is made locally or globally. Note that the symbiotic deconfigurations made by the managers are highly application and fault dependent. For example, out of the 100 initial 4-core configurations (total of 400 cores), 140 of them were randomly picked to have a Front End fault. In 98 of the 140 cases the best decision is to symbiotically deconfigure the Back End, in 20 cases the LSQ is deconfigured, and in 22 cases no symbiotic deconfiguration is performed. While there is often a bias towards the symbiotic deconfiguration of one region over another depending on the initial fault, it is not a clear-cut decision to engage symbiotic deconfiguration all the time.

Finally, we implemented an offline manager that decouples the symbiotic deconfiguration decision from the boosting decision. The manager first accumulates the largest amount of power possible using only symbiotic deconfigurations with a PPR of at least 2 (i.e., a minimum 2% power accumulation for a 1% performance loss). The manager then finds the combination of boosting techniques that maximizes performance within this accumulated power budget. We found that this decoupled offline manager achieved an average performance of 20.4%, which is very close to the 22.2% achieved by Global3.

The results from this section demonstrate that in order to reap the full benefits of PowerTransfer:

- Symbiotic deconfiguration decisions must account for the characteristics of the running applications; however, these decisions can be decoupled from the decisions of which boosting techniques to engage;
- Alternative CPU boosting techniques with a better PPR than DVFS should be included;
- Multiple performance boosting techniques should be implemented to account for a range of application types;
- A global pool of power should be accumulated and distributed to boosting techniques in a global fashion.

4.3.2 Online PowerTransfer Management

For the offline managers, each core can be configured in 160 ways: 4 deconfiguration decisions, 5 DVFS configurations (one of four DVFS levels or no frequency boosting), 4 ways to employ Speculative Cache Access, and 2 ways to employ Clear (on/off). Moreover, since there are four cores in each configuration, this results in a total number of 160^4 chip-wide combinations, which clearly indicates that runtime exhaustive exploration of the space is impractical. In this section, we devise a Power Transfer Runtime Manager (PTRM) capable of making new decisions every OS time slice interval by sampling the space in a pragmatic way based on the insights gained from the offline analysis.

We identify a sequence of sampling steps that significantly reduces the search space. The first step is to make a symbiotic deconfiguration decision on each core by sampling deconfiguring a lane in each of the two remaining pipeline regions in turn. Second, based on our previous observations, we only sample 8 boosting configurations, corresponding to the combination of enabling Speculative Cache Access and Clear. Using this information, the best chip-wide configuration that meets the power budget is chosen. The remaining power (if any) is distributed using DVFS across the cores in a greedy fashion: the frequency of the highest IPC core is boosted to its maximum value, then the next highest IPC core, etc, until the power budget is exhausted.

Based on the need for only 10 samples, and our experimentation with different sampling sizes, we chose a sampling interval of 1ms within a 100ms time quantum. Note that it is possible that the configurations deemed best in the sampling phase would actually exceed the power budget in a 100ms quantum. To address this possibility, the GPM uses DVFS to reduce the frequency and voltage when the power budget might be exceeded. This only occurred in 5 out of the 100 runs and is reflected in our performance results.

To differentiate between the performance loss due to the limited number of samples, and that due to the use of a 1ms sample to infer the behavior of the full 100ms time quantum, we also implemented a version of the online manager (*OnlineExhaustive*) that can sample all possible combinations of symbiotic deconfigurations and performance boosting without cost. That is, we run all possible combinations for 1ms each and pick the best to run over the entire 100ms quantum.

Figure 6 shows the results. The OnlineExhaustive manager improves throughput on average by 17.3% compared to the 22.2% improvement when decisions are made based on the entire 100ms time quantum, indicating some loss due to the 1ms sampling interval. The PTRM achieves very close to OnlineExhaustive (average of 15.3% compared to 17.3% over the baseline), indicating a modest impact from decoupling symbiotic deconfiguration from performance boosting.

From these results, we conclude that a significant performance improvement – as much as 25% over the baseline – can be achieved with a PTRM that samples the deconfiguration in a logical fashion based on the insight obtained from the offline analysis.

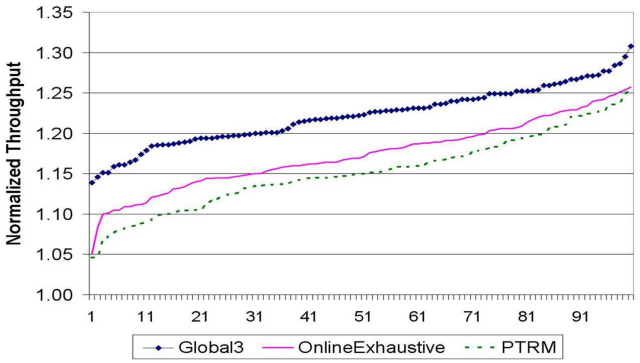

Figure 6: Throughput improvement of PTRM, the online exhaustive manager (OnlineExhaustive), and Global3 over the baseline.

5 Related Work

Processor Adaptivity: Prior work investigated architectural techniques that adapt the processor components to workload variation save power. Albonesi et al. propose Complexity-Adaptive Processors which dynamically disable underutilized hardware to improve performance or power efficiency [1]. Iyer and Marculescu develop a run-time profiling technique to detect program hotspots and adapt the processor configuration to match the hotspots demands [8].

Most work in this area examines a particular processor structure and makes it more efficient through adaptation. Buyuktosunoglu et al. coordinate adaptive fetch and issue [5]. Folegnani and Gonzalez also develop a variable sized issue queue [6]. Bahar and Manne [2] examine a more coarse granularity adaptation which disables an entire back-end execution cluster to save power, when an application does not need it. While previous research developed mechanisms which sacrificed a little performance to save a lot of power, our work deconfigures units that would otherwise be wasting power due to hardware faults, and intelligently redistributes the power to other other components to boost performance.

CMP Power Management : There has been much work on power management for CMPs, but here we focus on the most related work where performance is maximized under a chip-wide power constraint. Isci et al. [7] developed the per-core maxBIPS algorithm. Sharkey et al. extend this work by exploring algorithms based on both DVFS and fetch toggling, and explore a number of tradeoffs such as local versus global management [12]. Teodorescu and Torellas [14] consider global power management in the presence of process variations and propose using linear optimization to efficiently find a near optimal allocation.

6 Conclusions

Future CMPs built in highly-scaled technologies face the prospect of having to deconfigure hardware units in the face of manufacturing defects and aging-related faults. Such deconfiguration may lead to application-specific pipeline imbalances that reduce the power-performance efficiency of the formerly well-balanced pipeline. We propose a novel CMP microarchitecture, PowerTransfer, that dynamically identifies such imbalances and rebalances the pipeline by proactively deconfiguring additional units. Doing so in an application-specific way yields additional power savings at little performance cost. The harnessed power is used to improve chip-wide performance by enabling previously dormant performance boosting techniques. We demonstrate that the use of a small number of boosting techniques enabled across the entire workload yields up to a 25% performance improvement over simply deconfiguring the faulty units.

Acknowledgements

The authors would like to thank Meyrem Kırman and Nevin Kırman for their help integrating Clear in our simulation infrastructure. This research is supported by NSF grants CCF-0916821 and CCF-0811729.

REFERENCES

[1] D. Albonesi, R. Balasubramonian, S. Dropsho, S. Dwarkadas, E. Friedman, M. Huang, V. Kursun, G. Magklis, M. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. Cook, and S. Schuster. Dynamically Tuning Processor Resources with Adaptive Processing. In *IEEE Computer*, 2003.

[2] R. Bahar and S. Manne. Power and Energy Reduction Via Pipeline Balancing. In *ISCA*, 2001.

[3] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. In *IEEE Micro*, 2007.

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

[5] A. Buyuktosunoglu, T. Karkhanis, D. Albonesi, and P. Bose. Energy Efficient Co-Adaptive Instruction Fetch and Issue. In *ISCA*, 2003.

[6] D. Folegnani and A. Gonzalez. Energy-Effective Isssue Logic. In *ISCA*, 2001.

[7] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proc. IEEE/ACM 39th Annual International Symposium on Microarchitecture*, 2006.

[8] A. Iyer and D. Marculescu. Microarchitecture-level Power Management. In *IEEE Transactions on Very Large Scale Integration Systems*, 2002.

[9] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *Proc. 14th IEEE Symposium on High Performance Computer Architecture*, 2008.

[10] N. Kirman, M. Kirman, and J. Martinez. Checkpointed Early Load Retirement. In *Proc. IEEE/ACM 38th Annual International Symposium on Microarchitecture*, 2005.

[11] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC Simulator. http://sesc.sourceforge.net. 2005.

[12] J. Sharkey, A. Buyuktosunoglu, and P. Bose. Evaluating Design Tradeoffs in On-Chip Power Management for CMPs. In *International Symposium on Low Power Electronics and Design*, 2007.

[13] D. Tarjan, S. Thoziyoor, and N. Jouppi. Cacti 5.3. In *HP Laboratories Palo Alto Technical Report*, 2005.

[14] R. Teodorescu and J. Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In *ISCA*, 2008.

[15] Whitepaper. Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors. http://download.intel.com/design/processor/applnots/320354.pdf. 2008.

[16] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. The University of Virginia, Department of Computer Science, Technical Report CS-2003-05. 2003.