Reducing the Latency and Area Cost of Core Swapping through Shared Helper Engines

Anahita Shayesteh[†], Eren Kursun[†], Tim Sherwood[‡], Suleyman Sair[§], Glenn Reinman[†]
[†] Computer Science Department, University of California, Los Angeles
[‡] Computer Science Department, University of California, Santa Barbara
[§] Department of Electrical and Computer Engineering, North Carolina State University
{anahita, kursun, reinman}@cs.ucla.edu, sherwood@cs.ucsb.edu, ssair@ncsu.edu

Abstract

Technology scaling trends and the limitations of packaging and cooling have intensified the need for thermally efficient architectures and architecture-level temperature management techniques. To combat these trends, we explore the use of core swapping on a microcore architecture, a deeply decoupled processor core with larger structures factored out as helper engines. The microcore architecture presents an ideal platform for core swapping thanks to helper engines that maintain the state of each process in a shared fabric surrounding the cores, reducing the impact of core swapping 43% on average while showing promising thermal reduction. It also has favorable performance when compared to other thermal management techniques.

Furthermore, we evaluate alternative approaches to spending the area overhead of the additional microcore, including larger microcores, CMP cores, and SMT cores with different thermal management techniques.

1 Introduction and Motivation

Thermal characteristics of contemporary processors are creating significant challenges to microprocessor design. Various trends threaten to make things even worse: the number of on-chip transistors are dramatically increasing, feature sizes are dropping to deep submicron levels, and supply voltage reduction is expected to slow down as it approaches noise margin barriers. As a result, power densities and on-chip temperatures are expected to increase even faster for the next generation of processors.

Thermal issues have gained significant importance in the past few years. Processor heating raises number of problems that threaten vital aspects of the microprocessor design, such as proper functionality, reliability, cost, and performance.

In recent years, dynamic thermal management (DTM) [5, 8, 9] has become an integral part of microprocessor design to adapt to increasing on-chip temperatures. DTM usually targets the removal of excessive heat from the processor after a certain temperature threshold is reached. Thermal management can cause performance degradation, as a result of reduced clock frequency, voltage or temporarily shutting down the entire chip. Therefore, efficient thermal management techniques with less impact on processor performance are extremely desirable.

Activity migration (i.e. core swapping) was recently proposed as an efficient technique for power density reduction [7]. While core swapping has the potential to drastically alleviate thermal problems, it can be plagued with two main drawbacks: 1) the latency overhead of performing a swap and 2) the area overhead of retaining additional resources to perform the swap. The latency overhead stems from both the time to propagate resources required for computation to migrate from one core to another and the warmup time for various predictors that may not be migrated from one core to another.

In this paper we further explore the use of swapping an application between multiple cores when a given core exceeds thermal threshold, specifically attacking both the latency and area overhead of swapping. Our cores feature a small, fast pipeline augmented with helper engines [16]. All large structures are factored out of this *microcore* and are relocated as helper engines, taking advantage of locality in the first level structures. The helper engines buffer state during core swaps and help reduce the overhead of swapping. We compare this approach to current DTM techniques.

Our contributions include:

- We propose a thermally-triggered core-swapping technique as a DTM for dual-microcore architectures. Our architecture solves the switching overhead problem inherent to core swapping by buffering the state in helper engines. We compare the performance of core swapping with other DTMs such as global clock gating and dynamic frequency scaling.

- We study the area overhead of dual-microcore architecture and investigate different architectures with comparable area including SMT and CMP cores for a two application workload.

The rest of this paper is organized as follows. In Section 2

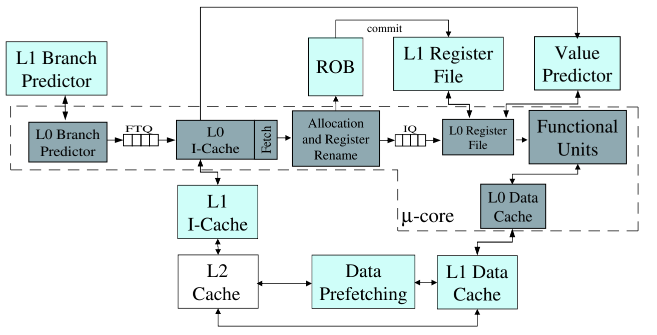

Figure 1: The factored microcore architecture. Helper engines (L1 Data Cache, L1 Instruction Cache, L1 Branch Predictor, L1 Register File, Value Predictor and Data Prefetching) are shown in lighter shade.

we discuss the prior work, followed by an introduction of the architectures we investigate in Section 3. Section 4 presents the methodology. We present the experimental results in Section 5 and concluding remarks are in Section 6.

2 Related Work

The circuit design community has proposed a great deal of work on dynamic power optimization techniques, which are also used as dynamic thermal management techniques in microprocessors in various forms. Such techniques include dynamic voltage scaling (DVS) and dynamic frequency scaling (DFS). In this section we will focus on the studies that are close to our own and specifically target microprocessor power/thermal optimization.

The Pentium 4 incorporates the existing *stopclock*, an architectural low-power logic mechanism that halts the clock signal to the bulk of the processor [6]. whenever any of the thermal sensors indicate that the die is hotter than critical temperature.

Heo, Barr and Asanovic [7] proposed an activity migration technique for power density reduction. Activity migration reduces the temperature by moving the computation between multiple replicated blocks. They analyze multiple configurations with some of the microprocessor units replicated or shared. The study concludes that the best configuration has a shared instruction cache, data cache, rename table, and issue queue. Ghiasi et al. [12] suggested migration between asymmetrical cores instead of symmetrical cores.

3 Core Swapping on a Microcore

We examine core swapping on a dual core version of a microcore architecture, where a set of larger structures are moved out of the cores as helper engines. This results in a smaller core size, with only the components necessary to maintain performance included in the microcore. Components that are latency tolerant become helper engines that are shared between the microcores.

Figure 1 illustrates our microcore architecture. The level one data and instruction caches are moved out of the core processor pipeline and replaced with a smaller L0 cache. The L0 extends the cache hierarchy, and therefore the L1 data cache is accessed on an L0 miss.

Our architecture makes use of a basic block target buffer (BBTB) [18], a branch address predictor that predicts an entire basic block each cycle. The microcore design has a reduced size BBTB in the core pipeline and adds a second level BBTB as done in [11]. On a first level BBTB miss, the second level BBTB is probed and fetch stalls until a response is received from the second level.

A multi-level register file is used as proposed in [2]. The basic differences are that they model an inclusive register file hierarchy where the second level register file (RF1) includes all the state contained in the first level register file (RF0). On a branch misprediction, the second level register file recovers the state of the first level register file. The register file is extended with a second level structure, but the commit hardware and ROB are completely decoupled as a helper engine, with only tag allocation in the ROB impacting the core timing.

The architecture includes a stream buffer guided by a stride-filtered markov predictor as proposed in [14]. There is also a hybrid value predictor [17], predicting only load instructions. The address and value predictors are moved further away from the core pipeline of the microcore.

The microcore design provides a suitable framework for core swapping for two main reasons:

- Performance efficiency: State buffering in helper engines shared between cores reduces the core swapping overhead significantly.
- Area efficiency: Resource duplication is limited to the smaller microcores, while larger structures are shared between cores.

Core swapping can impact processor performance significantly. On a core swap, we flush the pipeline similar to a branch misprediction. We need to propagate the first level register file state, the store buffer, and the dirty cache blocks in the L0 cache. Register file and store buffer state is copied to the other core, and dirty cache blocks are written back to the level one cache (the helper engine), which is shared between the cores. We could also have used a writethrough policy with our L0 data cache. We overlap copying register file and store buffer state and writing back dirty blocks with the restart of the pipeline on the new core. Our core swaps are triggered by thermal sensors, removing the overhead of unnecessary core swaps. When one core exceeds a thermal threshold, the application workload is swapped to the other core.

The cold start effect of caches and predictors causes an even more severe impact on the second core. These structures need to warm up and depending on their size, there is an overhead involved. The microcore architecture, with less state in the core and more buffering between the cores, provides a very tolerant framework for core swapping, while decreasing the cost

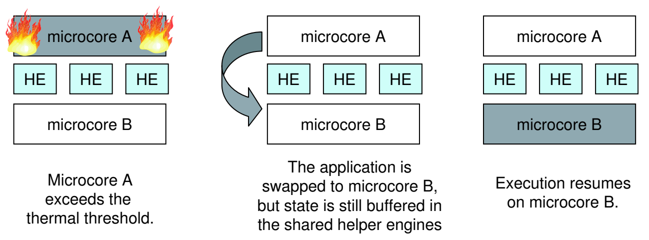
microcore A
HE
HE
HE
microcore B
Microcore A exceeds the thermal threshold.
microcore A
HE
HE
HE
microcore B
The application is swapped to microcore B, but state is still buffered in the shared helper engines
microcore A
HE
HE
HE
microcore B
Execution resumes on microcore B.

Figure 2: Core Swapping

of core replication to a small factored core. We evaluate these features in Section 5 by studying the performance and thermal behavior of selected architectures with comparable area, including core swapping.

4 Methodology

The simulator used in this study was derived from the SimpleScalar/Alpha 3.0 tool set [4], a suite of functional and timing simulation tools for the Alpha AXP ISA. The timing simulator executes only user-level instructions. Simulation is execution-driven, including execution down any speculative path until the detection of a fault, TLB miss, or branch misprediction. Our processor operates at a 5.6 GHz clock frequency.

We used 8 floating point and 8 integer benchmarks from SPEC2000 set for our experiments. The programs were compiled on a DEC Alpha AXP-21164 processor using the DEC C and C++ compilers under OSF/1 V4.0 operating system using full compiler optimization (`-O4-ifo`). We simulate 100 Million instructions after fast-forwarding an application-specific number of instructions as proposed by Sherwood et. al in [13]. All benchmarks were simulated using the *ref* inputs.

4.1 Architectural Model

We have made significant modifications to SimpleScalar to model the various speculative techniques and different configurations in this study. In addition to modeling all of the structures and latencies in the microcore architecture, we have extended SimpleScalar to include a cycle accurate, execution driven model of simultaneous multithreading (SMT) and chip multiprocessing (CMP)

Table 1 presents the simulation parameters for the microcore architectures we explore in this paper. Cache and register file access latencies are extracted from Cacti [15] for a *70nm* Technology at 5.6 GHz frequency. We modeled a 32 entry issue window and an 8K entry gshare branch predictor.

4.2 Power and Thermal Simulator

A complete analysis of the static and dynamic power consumption and resulting temperature characteristics of different architectures is crucial to our study. We used process parameters for a *70nm* process at 5.6GHz with 1V supply voltage, in order to have a better understanding of next generation submicron, low supply voltage, aggressively clocked microprocessors.

	Microcore	
	L0	Helper Engines
Instruction Window and Physical RF	256 entry ROB 100 entry RF0	256 entry RF1
BBTB	256-entry 4-way SA	2048-entry 4-way SA
L1 Data Cache	8KB 4-way SA, dual port, 32 byte block size, 2 cycle lat	32KB 4-way SA, single port, 32 byte blocksize, 5 cycle lat
L1 Instruction Cache	8KB 2-way SA , single port, 32 byte block size, 2 cycle lat	32KB 2-way SA, single port, 32 byte block size, 4 cycle lat
Value Predictor (1 prediction/cycle)	none	2K-entry stride 8K-entry L2 markov
Address Predictor (1 prediction/cycle)	none	2K-entry stride 4K-entry markov
Stream Buffer	none	32-entry FA buffer
Core Width	8-way issue, 4-way decode, 4-way commit	
Memory and L2 Cache	150 cycle memory lat, 512KB 4-way SA unified cache with a 64 byte block size and 12 cycle lat	

Table 1: Simulation parameters.

We have incorporated Wattch [3] models for dynamic power analysis of the microprocessor blocks. The experimental results we present are extracted with the most aggressive conditional clocking strategy, where the dynamic power scales linearly with access to the ports.

For submicron technologies, such as 70nm, leakage power constitutes a significant portion of the overall power. We adapted leakage models from Hotleakage [19] in our power/thermal simulator. The public version of Hotleakage only provides a software implementation of the leakage models for the data cache. We have extended and modified the tool significantly to accommodate other caches and cache-like structures in the microprocessor. We also used leakage parameters from Hotleakage's predetermined values specific to the 70nm process technology.

We use Hotspot's [8] thermal resistance/capacitance models and RC solvers for our analysis. Our power/thermal simulator also incorporates the thermal runaway phenomena enabled by the Hotleakage and Hotspot models. We used area values for our various architectural blocks based on our analysis with CACTI [15].

Heo, Barr and Asanovic [7], argue that most heat is dissipated vertically on the microprocessor chip, as the wafer thickness is much smaller than the chip area. Therefore, they assume infinite lateral resistances, although it leads to the worst case temperature gradients. We follow their example, and tune HotSpot to only consider the vertical component of temperature. Lateral modeling, while possible with HotSpot, is unrealistic without a more accurate floorplan of the various architectures we consider.

4.3 Dynamic Thermal Management Techniques

We used some thermal threshold values for the dynamic thermal management techniques. Critical thermal threshold, is the maximum tolerated temperature for proper functionality (timing data errors are likely after this value). DTM needs to be activated at this temperature. Safety thermal threshold, is the safety temperature for DTM to deactivate during the cooling down. We assume that the critical thermal threshold is 82°C and the safety thermal threshold is 79°C for the 70nm technology process we are investigating according to the ITRS [1] projections and results from [8].

We have incorporated an idealized version of dynamic frequency scaling for the experimental analysis. Our DFS has two different frequency settings: 5.6GHz for the normal operation and 4GHz for thermal relief, which gets activated as soon as on-chip temperatures reach the 82°C critical thermal threshold. Usually there is a large latency (on the order of μsecs) incurred every time the frequency is adjusted, which results in significant performance penalties in dynamic frequency scaling schemes. Skadron et al. [8] report 10 μsec for the non-idealized version of DFS. In our dynamic frequency scaling implementation there is no overhead, delay or penalty involved with changing the frequency of the processor.

We implemented global clock gating similar to Pentium 4 as discussed in Section 2. The global clock signal is shut down, whenever on-chip temperatures exceed the critical thermal threshold of 82°C. The processor resumes normal operation after the chip temperatures cool down below the safety threshold of 79°C.

Our thermally-triggered core swapping mechanism gets activated when a core reaches 82°C. The runs with this architecture assume an extra core (identical to the main core) that can be used to offload an application when one core overheats. The computation is migrated to the cooler core until the active core heats above the critical thermal threshold and another swap is required.

5 Experimental Results

In this section we compare the performance impact of core swapping to other DTMs on the microcore architecture. In particular, we examine the ability of the microcore to buffer state when core swapping, resulting in minimal degradation in performance. We further study the area overhead of duplicating the microcore, and investigate different architectures with comparable area. We use the power/thermal simulator framework discussed in Section 4 to explore different alternatives of simultaneous multithreading (SMT) and chip multiprocessing (CMP).

5.1 Core Swapping Performance Overhead

Core swapping can impact processor performance significantly. Figure 3 illustrates how this impact can be reduced by buffering state in helpers that are shared between two cores.

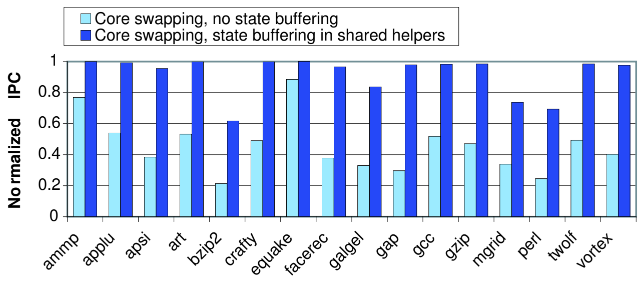

Figure 3: Core swapping overhead with and without buffering

Core swapping is thermally triggered and its impact directly depends on the characteristics of the benchmark. For benchmarks that heat up more, swapping occurs more frequently and results in more degradation. Benchmarks such as `bzip2`, `mgrid`, and `perl` have higher temperatures and suffer more from core swapping, as we will show in next section.

The first bar presents the performance impact when helpers are flushed on every core swap. Values are normalized based on a similar architecture with no core swapping. The average 53% degradation reflects the cold start effect of caches and predictors after a core swap. The second bar demonstrates the efficiency of buffering state in our shared helpers. The impact of core swapping is reduced to an average of merely 10%.

5.2 Single Application Workload

Core swapping has an area overhead from the duplication of core resources. The microcore design reduces this cost by sharing larger structures among cores and limiting the duplication to a small microcore. We extracted area numbers for most of the architectural blocks using CACTI [15], and used [10] for structures that could not be easily modeled with CACTI. Our data indicates that duplicating the microcore only increases our overall chip area by 25%. In this section we study alternative architectural designs with a comparable increase in area as well as alternative thermal management techniques.

Figure 4 compares the performance and thermal behavior of our microcore architecture in presence of different DTM techniques, including core swapping. The upper half of the figure shows the performance in BIPS for different benchmarks, and the lower half illustrates the heating behavior of the investigated architectures. This latter component shows the percentage of cycles for which at least one block exceeds the indicated temperatures: 75°C, 79°C, 82°C and 85°C. Darker colors in the lower graphs indicate higher temperatures. The rest of the figures in this section are similarly constructed.

Our detailed thermal analysis considers all of the possible overheating blocks. Although some of the hotspots were common among different benchmarks, such as the register file, load-store queue, etc, others varied across the different benchmarks and configurations. Even though the location of hotspots can provide a level of insight, the thermal behavior of the architecture can also be captured by the number of cycles that any of the blocks exceed a given thermal threshold.

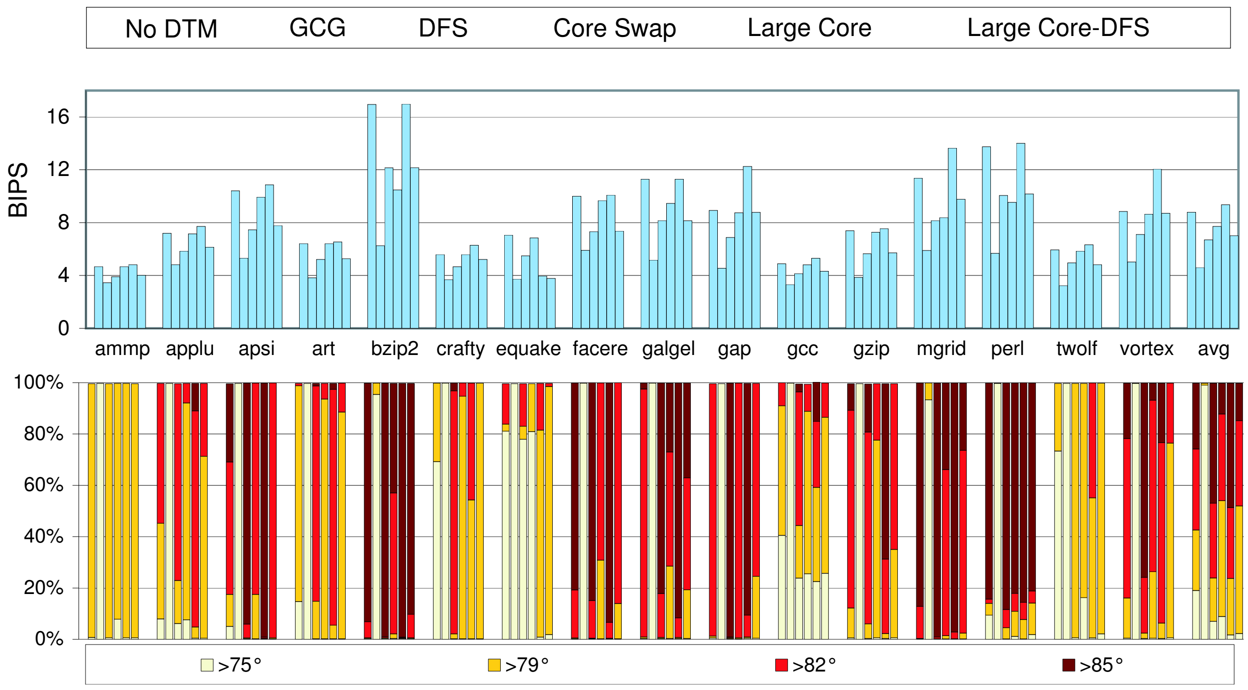

Figure 4: Thermal and Performance behavior of different architectures with and without DTM

As mentioned earlier in Section 1, performance degradation is commonly experienced with dynamic thermal management techniques. The degradation usually comes from various sources such as frequency decrease, voltage reduction, clock gating. Performance degradation might be quite significant depending on the DTM technique.

The first and fifth bars demonstrate results without thermal management of any kind. The first bar is our default microcore and the fifth bar is the microcore with larger resources. This latter bar doubles critical processor resources including the issue window (64-entry issue window), the first level data and instruction caches, and the first level branch predictor. The second and third bars show our microcore with global clock gating and idealized dynamic frequency scaling. The fourth bar presents core swapping results. The final bar represents the larger microcore with DFS.

For example benchmarks `bzip2` and `mgrid` see temperature greater than 85°C almost all the time when no thermal management is applied, and all DTMs impact their performance significantly.

Note that for many benchmarks, temperature frequently exceeds the thermal threshold, 82°C. These results should be considered as an upper bound for performance that can not be achieved. It would require sustained operation at a temperature beyond the critical thermal threshold, and a processor operating under such conditions would likely have timing, data and reliability complications.

Although global clock gating seems to be more effective in reducing the temperature in most benchmarks than DFS, it has a very significant performance penalty from frequently disabling the global clock signal. On average global clock gating sees 47% degradation on performance, running under critical threshold all the time.

Thanks to state buffering in the helper engines, core swapping is able to come close to the performance of the architecture without any DTM in most cases, seeing an average of 12% impact on performance. Core swapping is extremely effective at thermal management, reducing the temperature below 79°C at least 80% of the time for all benchmarks and well above 95% of the time for many benchmarks. Even `galgel`, which spends over half its execution time over 82°C is able to reduce its temperature below 79°C around 93% of the time using core swapping, with only an 8% degradation in BIPS.

Our results with core swapping, indicate that helper engines rarely heat up to critical temperatures, due to fewer number of accesses to those structures. For benchmarks we simulated, the block or blocks overheated were always inside the microcore. This is crucial for the effectiveness of our core swapping method in reducing the temperature.

For many applications, temperatures are still above the threshold with DFS, such as `bzip2`, `galgel` and `perl`. This indicates that our DFS strategy requires an even lower frequency to provide thermal relief to these applications, but at an even greater cost to performance. Despite a 70% drop in performance `bzip` is still above 85°C around 90% of the time with DFS. The benchmark `gap` operates in lower frequency mode almost 99% of the time in order to reduce the temperature, yet it is still above the 85°C temperature threshold 97% of the time. Our DFS on average, sees 23% degradation on performance with 14% of execution cycles above critical threshold of 85°C.

The actual scaling of frequency in DFS can have a significant performance impact if used often. However, we have used an idealized DFS implementation (see Section 4) that does not see this impact when transitioning between frequencies. Despite this advantage, core swapping is still able to outperform DFS.

The configurations with larger resources demonstrate one alternative use of the area overhead of core swapping. In some cases, the use of this overhead to instead increase the structures of the microcore has a clear performance advantage, 6% speed up over our baseline architecture on average – but this does nothing to alleviate the thermal problem. Even with idealized DFS on these larger resources, the performance impact is either too great or the thermal alleviation is not sufficient for safe operation.

5.3 Two Application Workload

In this section we consider the trade-off between using this extra core to run a second application instead of core swapping. It is important to note that the design space for chip capable of supporting two threads is quite large, but we believe that the results we present in this section highlight some interesting observations that will be explored in future work. Each application runs on a separate microcore with its own distinct set of helper engines. This approach (CMP) does not make use of core swapping, but can use dynamic frequency scaling (DFS) to *independently* scale the frequency of either core.

We compare this CMP approach to a microcore that has been enhanced with SMT to run two applications on a single core. This architecture also has a different set of helper engines per thread. One alternative is to simply have one SMT core running both threads without any DTM. Another is to have two SMT cores, but use core swapping to migrate both threads together from one core to the next. A final option is to have only one SMT core, but make it proportionately larger SMT (to compensate for the added area of the second core), and explore this with and without DFS.

Figure 5 displays the comparison between the SMT and CMP architectures. We constructed 16 set of benchmarks. The first bar shown on the graph is a single SMT core with no thermal management. The second bar shows two SMT cores that use core swapping for thermal management – both applications exist on only one core at a time and migrate together. The third bar is a larger single core SMT with double size issue window, caches, and branch predictor to compensate for the area overhead of core swapping. Again, this provides an alternative way to spend that area overhead. The fourth bar is this same architecture with DFS – both applications and set of helpers run in same frequency at all times. The last two bars show two single threaded CMP cores without any thermal managment and with our idealized DFS respectively.

CMP is able to outperform SMT for a number of benchmarks when no DTM is used on either architecture. The benchmark mixes of `bzip2.galgel` and `perl.lucas` are two examples of this. This is because these cores have comparable issue widths and resources (except for the larger SMT core). When SMT resources are increased to account for second core on CMP, the larger SMT can outperform CMP for many benchmarks. On average there is 17% performance speed up for both these alternative designs compared to baseline SMT architecture. Unfortunately, all of these alternatives see a significant number of cycles of thermal violation. SMT has lower temperatures than CMP for some applications – this is directly related to the improvement in BIPS seen by CMP. As the difference in BIPS grows, so does the difference in the temperature profile.

However, when thermal management is applied, core swapping shows the lowest impact of an average of 3% on performance for successful thermal alleviation. The other alternatives with DFS are either not able to retain comparable performance or not able to sufficiently alleviate the thermal problem. SMT with larger resources, sees an impact of %20 degradation due to frequency scaling, while 12% of cycles are above critical threshold. Similarly frequency scaling on CMP, degrades performance 19% on average, leaving 15% of cycles above 85°C. Note that our implementation of DFS allows the processor to run at temperatures higher than the critical 82°C-threshold in the lower frequency. A more realistic implementation would further impact the performance of DFS, but would allow more thermal alleviation.

6 Summary

We have explored the use of core swapping on a microcore architecture, a deeply decoupled processor core with larger structures factored out as helper engines. Core swapping is complicated by two factors: the cost of migrating an application from one core to another and the area overhead of the additional core for thermal management. Microcores enable efficient core swapping by buffering processor state in shared helper engines that reduce startup costs when switching to a new core. And they are small enough to reduce the area overhead of core replication.

Our results demonstrate that our microcore reduces the impact of core swapping significantly, on average 43% while showing promising thermal reduction ability. It also has favorable performance (as measured in BIPS) when compared to other DTM techniques such as GCG and an idealized version of DFS.

Additionally, we evaluated alternative approaches to spending the area overhead of the additional microcore, including larger microcores, CMP cores, and SMT cores, all with DFS. Our results indicate that while additional core on CMP and larger resources on SMT can improve performance, even an idealized version of DFS can hurt their performance significantly.

Acknowledgments

We would like to thank the anonymous reviewers for providing useful comments on this paper. This work was funded by an NSF CAREER grant.

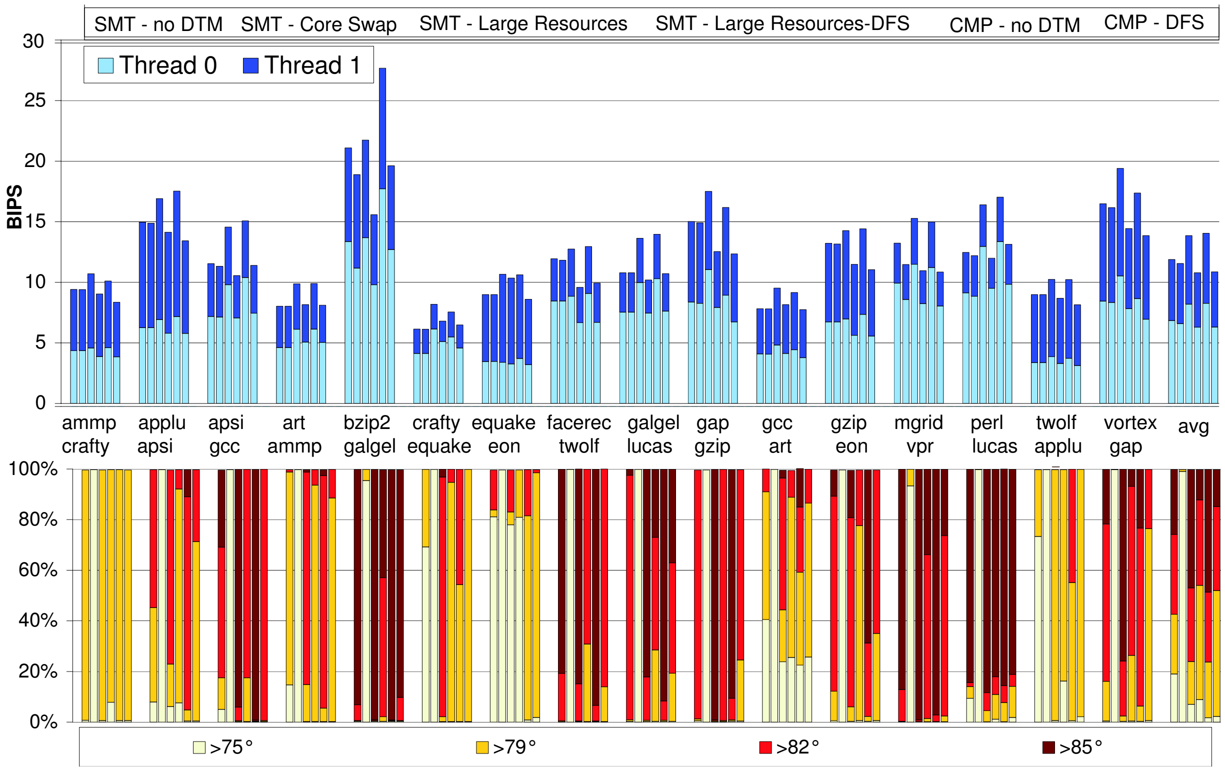

Figure 5: Thermal and Performance behavior of different architectures for two-thread workloads with and without DTM.

References

[1] In *International Technology Roadmap for Semiconductors*, 2003.

[2] R. Balasubramonian, S. Dwarkadas, and D. Albonesi. Reducing the complexity of the register file in dynamic superscalar processors. In *Proceedings of the 34th Annual International Symposium on Microarchitecture*, Dec. 2001.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimization. In *27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[4] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-97-1342, U. of Wisconsin, Madison, June 1997.

[5] D.Brooks and M.Martonosi. Dynamic thermal management for high-performance microprocessors. In *International Symposium on High-Performance Computer Architecture (HPCA-7)*, pages 171–182, Jan. 2001.

[6] S. Gunther, F. Binns, D. Carmean, and J. Hall. Managing the impact of increasing microprocessor power consumption. In *Intel Technology Journal Q1*, 2001.

[7] S. Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. In *International Symposium on Low Power Electronics and Design*, Aug. 2003.

[8] K.Skadron, M.Stan, W. Huang, S.Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *30th Annual International Symposium on Computer Architecture*, pages 2–13, June 2003.

[9] C.-H. Lim, W. Daasch, and G.Cai. A thermal-aware superscalar microprocessor. In *International Symposium on Quality Electronic Design*, pages 517–522, Mar. 2002.

[10] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 206–218, June 1997.

[11] G. Reinman, T. Austin, and B. Calder. A scalable front-end architecture for fast instruction delivery. In *26th Annual International Symposium on Computer Architecture*, May 1999.

[12] S.Ghiasi and D.Grunwald. Thermal management with asymmetrical dual core designs. Technical Report CU-CS-965-03, University of Colorado, 2003.

[13] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[14] T. Sherwood, S. Sair, and B. Calder. Predictor-directed stream buffers. In *33rd International Symposium on Microarchitecture*, Dec. 2000.

[15] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. In *Technical Report*, 2001.

[16] J. E. Smith. Instruction-level distributed processing. *IEEE Computer*, 34(4):59–65, Apr. 2001.

[17] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *30th Annual International Symposium on Microarchitecture*, pages 281–290, Dec. 1997.

[18] T. Yeh and Y. Patt. A comprehensive instruction fetch mechanism for a processor supporting speculative execution. In *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pages 129–139, Dec. 1992.

[19] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. In *University of Virginia Dept of Computer Science Tech Report CS-2003-05*, Mar. 2003.