Architecture-aware Task-scheduling: A thermal approach

Artur Podobas
KTH Royal Institute of Technology
Email: podobas@kth.se

Mats Brorsson
KTH Royal Institute of Technology and
SICS Swedish Institute of Computer Science
Email: matsbror@kth.se

***Abstract*—Current task-centric many-core schedulers share a "naive" view of processor architecture; a view that does not care about its thermal, architectural or power consuming properties. Future processor will be more heterogeneous than what we see today, and following Moore's law of transistor doubling, we forsee an increase in power consumption and thus temperature. Thermal stress can induce errors in processors, and so a common way to counter this is by slowing the processor down; something task-centric schedulers should strive to avoid. The Thermal-Task-Interleaving scheduling algorithm proposed in this paper takes both the application temperature behavior and architecture into account when making decisions. We show that for a mixed workload, our scheduler outperforms some of the standard, architecture-unaware scheduling solutions existing today.**

***Index Terms*—OpenMP , Tasks , Power, Thermal, Temperature, Scheduling, Many-core, Tilera**

INTRODUCTION

This papers focuses on the thermal properties of applications that uses the task-based programming model. Increased temperature have long been a bottleneck for processor designs, as they induce errors and limitations in the device [13], [11]. Among the errors that can occur are electronmigration, stress-migration and gate-oxide breakdown. These are considered as hard errors, and usually render the device useless as they occur. Also soft errors exist, that cause the processor the execute incorrectly due to e.g. electrical noise. To counter the thermal related errors, processors usually come with a fan and a heat sink designed to dissipate the generated heat. Larger fans contribute to the indirect cost of using the processor, as more power is required to drive them and keep the processor at the designed thermal level.

Another way of constraining the temperature in processors is by using a dynamic frequency or voltage scaling. Since the power consumption of semiconductors is proportional to the frequency and voltage of the processor, scaling these are an effective way to reduce temperature. Processors such as the Nehalem uses DFS to successfully control the power consumption of cores. Decreasing the frequency to control the power consumption also decreases the performance of the processor.

In this study, we propose a way to schedule work in a way that minimizes the performance throttling mechanisms of the underlying architecture. We do this by taking application specific behavior into account. More specifically, we propose an algorithm for the task-centric programming model where application specific power consumption is balanced to control the temperature of the chip. The task-centric programming model exposes available application parallelism in the form of tasks. A task is a user-exposed workload that can be run in parallel on any availible core. The exposed tasks are scheduled onto the system resources by the scheduler. Task-scheduling differ from OS-scheduling in several ways. Tasks can have dependencies among each other, and are usually composed of work that is much more fine-grained than OS-threads. Tasks are currently used in user-space, where a programmer exposes different application workloads to the task-scheduler, whose job is to schedule these in a efficient way. Many current scheduling algorithms are completely un-aware of the architecture they run on. Although there are locality aware schedulers (e.g. Cilk), they are designed in such abstract way that locality comes as a bonus. To give a concrete example of this, given a task-graph, the Cilk scheduler will try to distribute task closest to the root of the task-graph to cores, and the cores transverse down the leafs from the root task. All tasks spawned as the scheduler transverse down the tree are put into the cores private task-queue, and thus locality is achieved in this sense, since cores will first perform work in their own queues rather than go stealing from others. Although this does increase locality, there is no detailed architecture awareness.

Our understanding is that future many-core processors will be more heterogeneous and the task schedulers need to incorporate architectural details inside it to use it effectively. This paper investigate how to incorporate thermal-awareness into a task-centric scheduler, and how it would perform under a thermally stressed environment. To this day, we are unaware of any work that has been performed in the realm of task-centric scheduling that address the issue of temperature.

RELATED WORK

Temperature prediction: is an attempt to calculate the temperature of an area in advance, thus giving the scheduler opportunities to proactively counter it. Coskun et al. [4] have proposed to use an ARMA model to predict future temperatures. Their scheduling technique reduces up to 60% of the hotspot occurrences with the UltraSPARC T1.

Other approaches to predict a cores future temperature is proposed by Yeo et al [15], [16]. In [15] they Yeo et al. propose to group applications with similar thermal characteristics together, and use these to predict the thermal behavior of an unknown application. In [16] they propose to combine an application-specific thermal model with a core-based thermal model when predicting the temperature of a core. Their model shows predictive errors as low as 1.6% compared to the thermal sensor of their architecture.

OS-based Thermal scheduling: Scheduling tasks and threads in an OS environment has been subject to much research concerning thermal behavior. Choi et al [3] introduces a kernel task which aims to reduce the temperature of a core. These "cool-tasks" are used together with task migration to reduce the temperature of the processor.

Coskun et al [5], [4] proposes two scheduling solutions to the thermal reducing problem. In [5] they formulate the problem as a ILP with objectives for reducing amongst other hot spots and gradients. They shows that their proposed solution reduces hotspot better than existing OS load-balancing schedulers. Merkel et al [10] created an algorithm that is based on the OS queue's power and thermal capabilities and show that by setting a throttling limit, a thermal-aware scheduler outperforms some common Linux schedulers. Although their method closely resemble ours, we have focused on the task-centric model while Merkel et al. focuses on the OS domain.

Objective based Thermal scheduling: is when tasks delivered to the scheduler not only have thermal but also other constraints. Cui et al. [6] performed a study that concerns real-time temperature behavior on a CMP processor. They propose a heuristic scheduling algorithm which consider the lowest weight of a idle core when scheduling a real-time task. The weight is primary based on the maximum temperature a core would experience if the task was to be allocated to that particular core. Juan et al [2] have investigated how to reduce energy using DVFS on OpenMP static loop scheduling. They implemented two different schedulers that reduces the power consumption under a energy-constraint environment for up to 160 processors.

Thermal-Task Interleaving

To counter thermal limitations in modern processors, we propose an algorithm which controls the power consumption of cores in order to avoid elevated temperatures. The algorithm relies on a scheduler-embedded power and a thermal model that will be considered when scheduling tasks onto cores, something that is lacking in today's task-centric schedulers.

To give an intuitive feeling of the logic behind our scheduling algorithm we first consider how the Breadth-first (BF) scheduler behaves thermally. The BF scheduler consists of a single global queue. A queue in the task-centric model is a placeholder into which tasks are spawned and made available for execution. In the BF scheduler, an idle core will take work from the head of the queue and spawn more work to the tail of the queue. Figure 1 shows a thermal trace of BF executing two applications with different thermal behavior using a single core. Due to the way the BF works, we execute the first application, and then execute the second benchmark. Under a thermally constrained environment, the application is prone to be throttled ($330.5K$in the figure) for the entire duration of the first benchmark; something that clearly can be avoided as the other benchmark is much more cooler than the first.

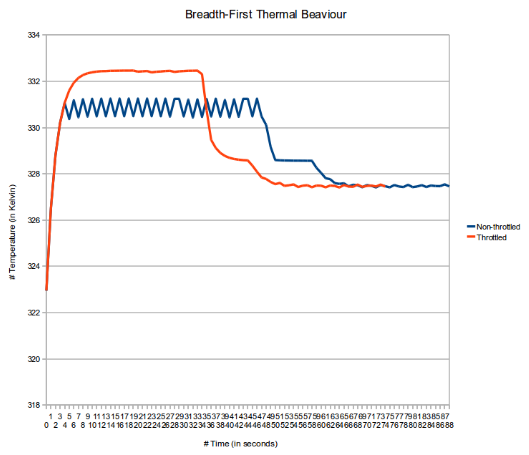

Figure 1. BF thermal behavior under a no-throttle and throttled (330.5 K) environment

Based on this knowledge, we propose a thermal-aware scheduling algorithm we call TTI (Thermal-Task-Interleaving). The objective the TTI-scheduler is to try to keep a user-specified temperature (T_{user}) throughout the program execution. We make two assumptions about our application behavior:

1) Power-consumption of previous tasks can be used to predict the power-consumption of future tasks of the same kind.
2) The application exposes enough parallelism so that both power-hungry and non-power-hungry tasks are present.

Thermal interleaving is achieved by using a multiple global queue approach. Each root-task spawned will allocate one global queue that will hold all potential children. A root-task is defined here as a task with no parent. We monitor the power-consumption of each individual task executed, and update the associated queue average power-consumption to reflect the power behavior of future tasks spawned in the queue. The scheduler monitors the temperature of the current core, and pick tasks from the appropriate queues to control temperature. Should the temperature rise above T_{user}, the scheduler will try to pick those tasks that consume the least power. If the temperature is below T_{user}, the scheduler will try to execute power-hungry tasks. To quickly find the appropriate tasks, the scheduler keeps a per-power sorted index list. The list is sorted so that most-power consuming tasks are indexed at the end and the least-power consuming tasks at the beginning. Locating a power-hungry task is simply transversing the sorted list from the end to the beginning and the other way around for a not so power consuming task. The list is continuously updated each second. Our scheduler also contains a probing mode to keep each queue's IPC up-to-date. The probing mode is necessary to intercept and register application specific power consumption changes. Pseudo-code for the scheduler from a cores perspective is seen in algorithm 1. The scheduling algorithm was implemented as a plugin to the Nanos++ framework operating under the OmpSs infrastructure.

Algorithm 1 TTI scheduler implementation

```
Schedule_Loop:
if (ProbeMode==ON)
        {
                Clear_HW_Counters();
                Get_Task_To_Probe();
                Execute_Task();
                Calculate_Task_Power();
                Update_Queue_Power();
                ProbeMode = OFF;
        }
if (CurrentTemp>TargetTemp) && (ProbeMode==OFF)
        {
                Get_Low_Power_Task();
                Execute_Task();
        }
        else
        {
                Get_High_Power_Task();
                Execute_Task();
        }
goto Schedule_Loop;
```

Methodology

Power modeling

To calculate power-consumption of an electronic device you would ideally need a very detailed description of the device in question. Everything from the parameters of the semiconductor that was used (e.g. doping-levels, transistor widths), electrical details (e.g. parasitic capacitance's, resistances, leakage etc) to micro-architectural details. This information is not always available to us and even if it were it would not be feasible to calculate this in a run-time-system due to the overhead this would induce.

A more common approach is to derive the power-consumption of a core using the Instruction-Per-Cycle metric. Instruction-Per-Cycle (IPC) gives the amount of CPU activity compared to memory accesses (or other stalling mechanism). IPC has previously been shown [9], [1], [14] to have a strong correlation to the power consumed by the processor. To verify this correlation, we sequentially executed four different benchmarks on our target architecture and recorded their IPC. We then statically analyzed the type of instructions executed using a simulator trace, and calculated the power consumption using McPAT [8]. McPAT is system power, timing and area analyzing tool, that divides the processor into blocks of functional units and calculates the dynamic and static power consumption of the system using a user-supplied execution trace. McPAT was setup to model the power consumption of a processor with capabilities that resemble the TilePro64. The workloads consisted of four benchmarks: Fibonacci, n-Queens, SparseLU and matrix multiplication. The workloads were compiled with different optimization flags to receive different IPC ratings. Since the architecture we modeled resembled the TilePro64 architecture, compiling with an optimization level of $-O0$ will generate code that does not take full advantage of the VLIW nature of the TilePro64, and thus reduce the IPC compared to more aggressive forms of optimization.

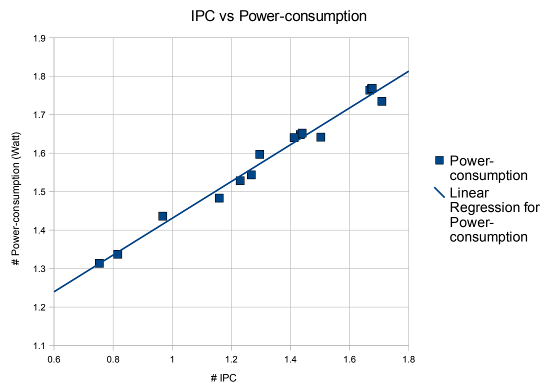

Figure 2. IPC vs Power-consumption scatter plot.

Figure 2 shows the scatter plot of IPC versus the (by McPAT) estimated power-consumption of the workloads running on one core. The graph verifies a strong correleation between IPC and power consumption. The power model that we used to estimate can be summarized as the equation: $P_{core} = k * IPC + p_{static}$ where p_{static} is the static power consumption of the core and IPC is the instruction-per-cycle value for the current frame.

Thermal modeling

Some of the power consumed by the processors is converted to thermal energy. Since our experimental architecture does not contain any thermal sensors or diodes, the power to temperature calculation is performed within our run-time system. We also require that the calculation is fast enough to reduce the overheads and performance footprints[1] on the system. We used a method that models the temperature as a lumped RC model, consisting of thermal capacitances and resistors. This method has been used in previous studies [12], [6] to model the temperature. Heat can be described as power (current source) that is flowing through a resistance, and the temperature will thus be the "voltage" across the thermal resistor. HotSpot [12] is another thermal modelling tool that uses an advance model for this, to model heat dissipation to nearby areas and different layers of the chip. Our model is a simple RC circuit due to the increased computational complexity needed to solve a large RC-network.

We validated our model by running comparing our thermal-model against HotSpot's, assuming the same chip dimensions. Figure 3 shows how HotSpot and our simplified model reacts to a constant power pressure of 1.6 Watt. Although our model estimates a higher temperature than HotSpot due to the simplification we have made, we consider it detailed enough for this study.

[1]By performance footprints we mean corruption of other cores caches or accessing shared resources that can induce penalties for other cores.

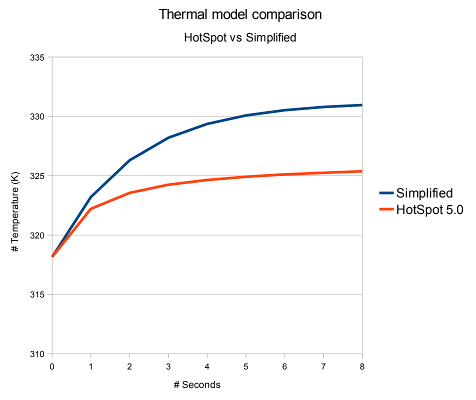

Figure 3. Thermal model validation

Benchmarks

We used a combination of benchmarks from the Barcelona OpenMP Task Suite for our baseline evaluation [7]. The benchmarks provide variability in terms of power consumption and functionality. The benchmarks are also customizable so that the degree of exposed parallelism can be controlled. We chose to run the benchmark combination for up to 20 cores. Each benchmark combination was executed three times and the median was chosen to represent the runs. The benchmark combination was implemented as a single application; not as two seperate OS processes. Speedup was calculated normalized to the execution time of the BF scheduler under thermal stress. The user-specified temperature threshold, T_{user}, is set to $329K$ in all runs concerning the TTI scheduler.

A brief description of the benchmarks we used is given:

n-Queens: is a search-and-prune parallel benchmark that tries to solve the N-Queens problem. Given a nxn sized chessboard and n queens, in how many ways can all queens be put on the chessboard so that no queen threatens another queen.

Floorplan: is a search-and-prune benchmark that strives to optimize the placement of blocks on a die to reduce area required to fit all blocks. Given n blocks, in what way can they be arranged so that the area is minimal. Unlike other benchmarks, the generated task-graph for this application is not always the same. This is because each task will check if its current area is larger than what has been reported by other tasks (that successfully placed the blocks), and prune the search if the condition is true.

Multisort: is a parallel sorting algorithm that is composed of two stages: sort and merge. The sort stage recursively divides the unsorted array into sub-arrays. The sub-arrays are sorted using combination of insertion- and quicksort. Once the sub-arrays have been sorted, the algorithm merges the resulting sub-arrays into a complete sorted array.

Strassen: is a parallel matrix multiplication based on Strassen's algorithm. The algorithm subdivides the array into smaller arrays, and multiplies the sub-matrix separately. There are two parameters to control the benchmark execution: sub-matrix size and task cutoff. Task-cutoff limits the depth of the generated task-graph and sub-matrix size determine the matrix block sizes to be used by the algorithm.

We choose a combination of application so that each power-hungry benchmark was coupled with one with lower power consumption. The combinations can be seen in table I.

Table I
WORKLOADS USED FOR EVALUATING THE SCHEDULER

Workload	Parameters
Sort + Strassen	80 MB Sort // 1024x1024 Strassen
Floorplan + Strassen	15 Floorplanned blocks // 1024x1024 Strassen
Floorplan + n-Queens	15 Floorplanned blocks // N-Queen 14x14
n-Queens + Strassen	N-Queen 14x14 // 1024x1024 Strassen

Schedulers: We compared our results with the default and breadth first scheduler found inside the Nanos++ / OmpSs OpenMP framework.

Architecture

Our evaluation platform is the TilePro64 processor by Tilera. The TilePro64 processor is a many-core processor composed of 64 RISC-like cores. Each core runs at a clock frequency of 700 MHz and of VLIW type. Each core has a private 16 kB L1 data+instruction cache, and the L2 is distributed across the chip. The description of the L2 coherence protocol is out of the scope for this study, but is highly customizable. The cores are connected to each other using a Network-on-Chip (NoC). The NoC is enhanced mesh-like interconnect that support several types of data traffic.

RESULTS

To examine the impact of our scheduler compared to other thermally unaware schedulers, we examined how well the TTI scheduler scales compared to BF and default. In the execution runs, we set the thermal throttling level to $330.5K$ respectively $330K$ to illustate benefits of using our scheduler compared to other schedulers under different throttling threshold.

Figure 4 shows all the speedup for all the combations. For Sort+Strassen our TTI scheduler performs 4.3% better than the default scheduler, and 26.6% better compared to BF. For Floorplan+Strassen the increase in speedup is 9.5% compared to default and 27% compared to BF. Note that in these combinations, Strassen is the power-intensive benchmark while Floorplan/Sort is used to balance the power consumption. For the combination n-Queens+Floorplan, the speedup increase using our TTI scheduler is 14.7% compared to default and over 80% increase compared to BF. n-Queens is considered to be a power-intensive benchmark in these runs.

The combination Strassen+n-Queens is also shown to illustrate the weakness of our scheduler. Strassen and n-Queens are both power-intensive benchmarks and thus our TTI scheduler is unable to find any workload to balance the power-consumption with. There is therefore no speedup increase compared to default (0.5% decrease even) since both schedulers suffer from performance throttling. This scenario can also occur in the other combinational benchmarks. Figure

5 shows the limitations of our TTI scheduler using the workloads Strassen+n-Queens and Floorplan+Sort . In the case n-Queens+Strassen, after 23 seconds the TTI has exhausted all low-power tasks and is forced to only execute the remaining power-intensive tasks with an thermal throttling penalty. In the case Floorplan+Sort, the power generated by the benchmarks is simply not enough to even hit the user-set thermal threshold of $329K$; in this case, TTI scheduling is obsolete and leads only to additional run-time overheads.

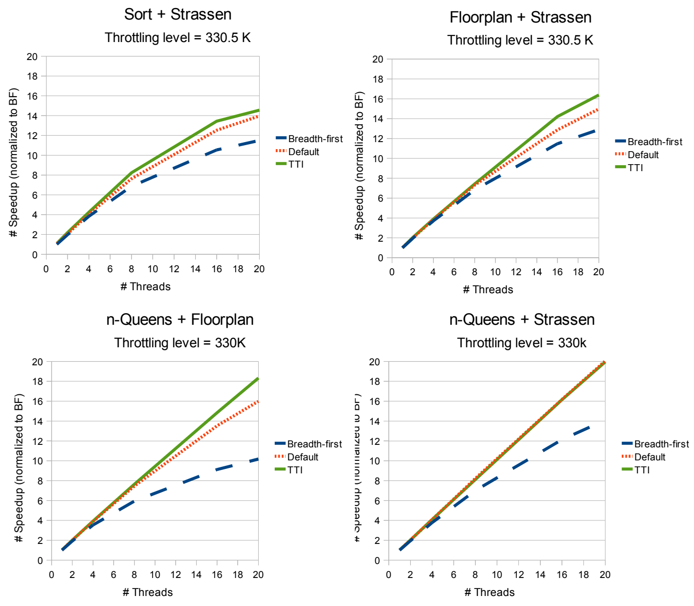

Figure 4. Speedup graphs the benchmark combinations.

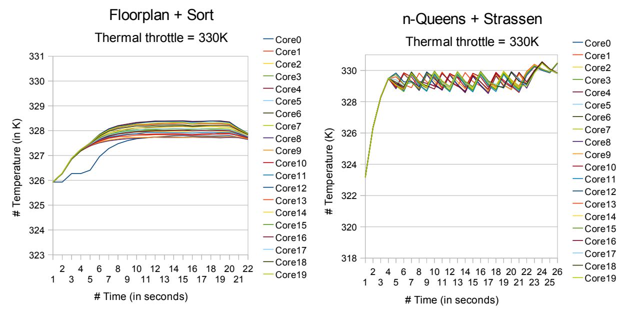

Figure 5. TTI example limitations illustrated using a temperature trace.

Conclusion

Preliminary results show that our thermal-aware scheduler, the Thermal-Task-Interleaving scheduler, balances availible task workloads in a way that tries to satisfy the user-set temperature limit. Additionally, we show that if the system is thermally constrained so that the processors are throttled when the temperature reaches the threshold, our scheduler outperforms two standard task-centric schedulers. The speedup can be as high as 80% compared to the Breadth-First scheduler and 14.7% compared to the default scheduler.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the ENCORE Project (www.encore-project.eu), grant agreement nr. 248647. The authors are members of the HiPEAC European network of Excellence (http://www.hipeac.net). We would also like to thanks the McPAT team for the support given regarding McPAT.

References

[1] Jianwei Chen, M. Dubois, and P. Stenstrom. Integrating complete-system and user-level performance/power simulators: the simwattch approach. *Performance Analysis of Systems and Software, IEEE International Symmposium on*, 0:1–10, 2003.

[2] Juan Chen, Yong Dong, Xuejun Yang, and Panfeng Wang. Energy-constrained openmp static loop scheduling. *High Performance Computing and Communications, 10th IEEE International Conference on*, 0:139–146, 2008.

[3] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Henrdrik Hamann, Alan Weger, and Pradip Bose. Thermal-aware task scheduling at the system software level. In *Proceedings of the 2007 international symposium on Low power electronics and design*, ISLPED '07, pages 213–218, New York, NY, USA, 2007. ACM.

[4] Ayse Kivilcim Coskun, Tajana Simunic Rosing, and Keith Whisnant. Temperature aware task scheduling in mpsocs. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '07, pages 1659–1664, San Jose, CA, USA, 2007. EDA Consortium.

[5] Ayse Kivilcim Coskun, Tajana Simunic Rosing, Keith A. Whisnant, and Kenny C. Gross. Temperature-aware mpsoc scheduling for reducing hot spots and gradients. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, ASP-DAC '08, pages 49–54, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[6] Jin Cui and Douglas L. Maskell. Dynamic thermal-aware scheduling on chip multiprocessor for soft real-time system. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, GLSVLSI '09, pages 393–396, New York, NY, USA, 2009. ACM.

[7] Alejandro Duran, Xavier Teruel, Roger Ferrer, Xavier Martorell, and Eduard Ayguade. Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp. *Parallel Processing, International Conference on*, 0:124–131, 2009.

[8] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 469–480, New York, NY, USA, 2009. ACM.

[9] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. *SIGMETRICS Perform. Eval. Rev.*, 31:160–171, June 2003.

[10] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40:403–414, April 2006.

[11] Francisco Javier Mesa-Martinez, Ehsan K. Ardestani, and Jose Renau. Characterizing processor thermal behavior. *SIGARCH Comput. Archit. News*, 38:193–204, March 2010.

[12] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1:94–125, March 2004.

[13] Jayanth Srinivasan, Sarita V. Adve, Pradip Bose, and Jude A. Rivers. The case for lifetime reliability-aware microprocessors. *Computer Architecture, International Symposium on*, 0:276, 2004.

[14] M. Valluri and L.K. John. Is compiling for performance == compiling for power? In *5th Annual Workshop of Interaction between Compilers and Computer Architecture*, 2001.

[15] Inchoon Yeo and Eun Jung Kim. Temperature-aware scheduler based on thermal behavior grouping in multicore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 946–951, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[16] Inchoon Yeo, Chih Chun Liu, and Eun Jung Kim. Predictive dynamic thermal management for multicore systems. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 734–739, New York, NY, USA, 2008. ACM.