Thermal Optimization in Multi-Granularity Multi-Core Floorplanning

Michael B. Healy, Hsien-Hsin S. Lee, Gabriel H. Loh†, and Sung Kyu Lim
School of Electrical and Computer Engineering, †College of Computing; Georgia Institute of Technology
{mbhealy, leehs, limsk}@ece.gatech.edu, †loh@cc.gatech.edu

Abstract—**Multi-core microarchitectures require a careful balance between many competing objectives to achieve the highest possible performance.** ***Integrated Early Analysis*** **is the consideration of all of these factors at an early stage. Toward this goal, this work presents the first adaptive multi-granularity multi-core microarchitecture-level floorplanner that simultaneously optimizes temperature and performance, and considers memory bus length. We include simultaneous optimization at both the module-level and the core/cache-bank level. Related experiments show that our methodology is effective for optimizing multi-core architectures.**[1]

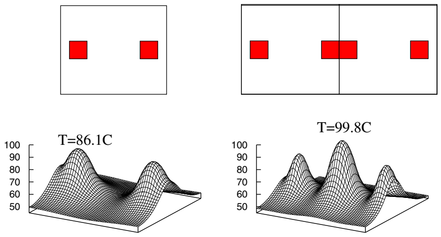

Fig. 1. The effect of core tiling on the thermal profile. The power dissipation per-core remains constant, but thermal coupling causes the maximum temperature to be higher for the tiled floorplan.

I. Introduction

The shift to the multi-core paradigm has been driven by increasing power dissipation and the need for ever-increasing throughput. Increases in power dissipation cause increases in operating temperatures, which in turn can cause device failure or reduction of reliability. Many techniques, such as dynamic voltage and frequency scaling, are used to reduce power dissipation in modern microprocessors. Most of these techniques, however, also reduce performance. Therefore, any design methodology (such as the one presented in this paper) that can statically decrease operating temperatures will allow increases in performance by reducing the need for dynamic power dissipation limiting.

Darringer [6] cites many challenges for the EDA community regarding the multi-core era. One overall challenge is that of *Integrated Early Analysis*. Microarchitectural-level floorplanning is a critical piece of Integrated Early Analysis. Microarchitectural floorplanning influences the overall operating temperature of a microprocessor by changing module-to-module thermal coupling. By reducing this module-to-module thermal coupling, the maximum temperature experienced by a design can be reduced. In multi-core floorplans, modules from neighboring cores will affect the thermal profile just as much as modules within the same floorplan. Therefore, floorplanners of microprocessor cores intended for use in multi-core systems must account for the effects of thermal coupling from neighboring cores. A motivating example is shown in Figure 1. If the floorplan from the left of Figure 1 is tiled horizontally then the thermal profile increases to that of the floorplan on the right.

Additionally, the heirarchical nature of multi-core systems allows for interesting innovations in algorithm design. Multi-core systems are typically designed with only one or two types of processor cores that are stamped out multiple times and connected together with a communication fabric. Exploiting the multi-level nature of these systems is one of the focuses of this paper. Core-level and module-level floorplan changes have different impacts on the total system performance and both must be taken into account during optimization.

The contributions of this work are as follows:

- We present the first multi-core microarchitectural floorplanner that extends and generalizes traditional floorplanning techniques to effectively optimize operating temperatures in multi-core systems without negatively impacting performance.
- We propose and evaluate a number of multi-granularity floorplanning techniques for temperature optimization targeted at architectures with multiple cores and L2 cache banks. Our optimization accounts for routing of the memory bus connecting cores and cache banks.
- We explore multi-core floorplanning of homogeneous architectures with heterogeneous floorplans to exploit differences in the thermal environment of separate parts of the chip.

Multi-core systems designed with multiple core types (called heterogeneous coreplans) present an interesting area for optimization exploration. Our work provides an efficient early design analysis tool that helps system designers exploit the full potential of the multi-core paradigm. Typically, the core-level and module-level design cycles are decoupled. However, our unified core-level and module-level methodology should produce module-plans that are more generally reusable for integration in multiple types of multi-core systems. This is because module-plans produced with our methodology can be optimized to deal with thermal-coupling from cores in several or all directions.

II. Preliminaries

A. Previous Works

Most work in the multi-core arena focuses on dynamic behavior and the microarchitectural aspects of a multi-core design. Li *et al.* [12] analyzed a networked memory subsystem by considering thermal effects and examined the impact of exchanging bank locations with core locations in a 3D chip-stack. Chaparro *et al.* [2] studied the thermal implications of a 16-core architecture and explored various thermal management schemes, such as activity migration. Ogras *et al.* [18] proposed a methodology for customized communication-architecture synthesis that minimizes total energy consumption while satisfying many communication pattern requirements. In contrast, our work focuses on the static physical design of multi-core systems, and we do not consider dynamic thermal-management schemes which are orthogonal to our contributions. The improvements resulting from

[1] The authors acknowledge the support of the Focus Center for Circuit & System Solutions (C2S2), one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation Program.

TABLE I
A SUMMARY OF THE TERMS USED IN THIS PAPER.

term	explanation
module-plan	arrangement of modules in a single core
coreplan	arrangement of cores and cache banks
multi-core floorplan	arrangement of modules and cores (= module-plan + coreplan)
homogeneous coreplan	coreplan with only one type of module-plan
heterogeneous coreplan	coreplan with multiple module-plans
template-based heterogeneous coreplan	hand-optimized heterogeneous coreplan
multi-granularity floorplanning	optimize both module- and coreplan

our optimizations are achieved without architectural performance degradation.

For multi-core works that focus on physical design, Murali *et al.* [15] describe the design of an ASIC NOC that includes floorplan information and layout. However, they only consider IP blocks as floorplan members and do not include thermal considerations. We consider the floorplanning of the modules inside each core to optimize the thermal profile more effectively. Mukherjee and Memik [14] present two physical-aware frequency selection algorithms that focus on reducing hotspots while maintaining high performance. They use task mapping to accomplish their temperature reduction. In contrast, we focus on physically designing the architecture to have limited hotspots.

Traditional physical design tools that focus on temperature aware floorplanning include Cong *et al.* [5] and Healy *et al.* [9]. Cong *et al.* focus on decreasing the execution time of temperature calculations during simulated annealing using a novel bucket structure and limiting assumptions about heat flow while Healy *et al.* focus on simultaneous temperature and performance improvement using a combined linear programming and simulated annealing technique. This work differs in that it specifically targets issues that are unique to the multi-core paradigm, such as core-level floorplan changes.

Other microarchitectural-level floorplanning works include Cong *et al.* [4], Ekpanyapong *et al.* [8], Nookala *et al.* [17], and Long *et al.* [13]. None of these works, however, explore the design tradeoffs inherent in multi-core-aware design.

B. Terminology

A summary of the terms used in this paper is given in Table I. Note that module-planning is the only optimization used in homogeneous floorplanning and fixed template-based heterogeneous floorplanning. The six types of template-based heterogeneous coreplans considered in this paper are shown in Figure 2. These coreplans do not represent fixed outline examples. The coreplans in Figure 2 merely show the relative location and orientations of the corresponding cores and cache banks. Optimized forms of heterogeneous coreplans (see Figure 9) may not follow any of these "templates."

C. Module-Planning Basics

The module-planning algorithm presented in this work uses Simulated Annealing with the Sequence Pair [16] floorplan representation. We use three move types during annealing. The first move swaps the position of two modules in both sequence pairs. The second move swaps the position of two numbers in a single sequence. The final move changes the shape of one module [3]. In the case of optimizing module-plans in homogeneous coreplans, a single sequence pair is enough. In the case of heterogeneous coreplans with k types of cores, k separate sequence pairs are managed. During annealing, a core type is first selected for performing a candidate move on. Next, the move type is selected for that core type. This allows the simultaneous optimization of all module-plans in the heterogeneous coreplan.

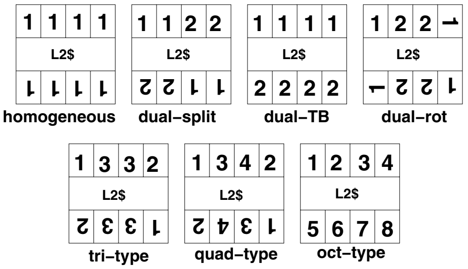

Fig. 2. Homogeneous and template-based heterogeneous coreplans for an 8-core system. Note that module-planning is the only optimization used in these coreplans.

The objectives of the optimizations being made are to increase performance and decrease maximum and average temperatures. Performance is optimized by using profile-weighted wirelength [8] as a part of the cost function used during annealing. Access frequencies of major architectural wires are collected during architectural simulations of multi-core benchmark suites. The normalized averages of these access frequencies (profile) are then used as wire weights during annealing. The cost function optimized during module-planning is:

$$Cost = \alpha * area + \beta * wwl + \gamma * temp$$

where α, β, and γ are user-defined weighting constants, $area$ is the total area of the coreplan, wwl is the sum of the profile-weighted wirelength of all core types in the system and $temp$ is related to maximum and average temperatures as described next.

D. Temperature Optimization

Floorplanning is only capable of reducing the temperature of an architecture by changing module-to-module thermal coupling. For the purpose of lowering these module-to-module interactions, and thus the operating temperatures, a temperature-weighted block distance cost is added to the annealing cost function. Temperature-weighted distance for blocks i and j for the temperature profile of benchmark k is calculated as follows:

$$Temp\_Dist = \frac{T_{i,k} * T_{j,k}}{d_{i,j}}$$

Where $d_{i,j}$ is the distance between blocks i and j and $T_{n,k}$ is the temperature of block n under the power profile of benchmark k. The addition to the cost function is calculated as the temperature-weighted distance between each pair of modules in the floorplan.

Calculating the temperature at every annealing move is too costly in terms of execution time. Several options were explored for the purpose of reducing execution time. Experiments indicated that a simple scheme was sufficient. In this scheme the temperature update was modified to occur once out of every n annealing moves. During the period between temperature updates the previously simulated temperature was used for calculating the temperature-weighted block distance. Several values of n were tested and $n = 20$ resulted in the best tradeoff between execution time and final temperature values.

The temperature-weighted distance cost tends to optimize average temperature more highly than maximum temperature. To deal with this issue the maximum temperature of the floorplan is used as the $temp$ term after a temperature update. In updates where the temperature is not recalculated the $temp$ value is weighted by the ratio between the temperature-weighted block distance of the floorplan being considered and the temperature-weighted block distance of the floorplan that the temperature was last calculated on. This technique effectively minimizes both the *average* and *maximum* temperature of the resulting floorplan.

III. Coreplan Optimization

Coreplanning involves optimizing the location and orientation of cores and cache banks. If one were to allow unrestricted coreplan moves during annealing, then a large number of highly unacceptable coreplans would be examined. To combat this issue, we consider a fixed coreplan space that consists of a square grid. Specifically, a 4×4 grid is used for the 8-core system with 8 L2-cache banks examined here. The grid does not specify any sort of fixed outline, it merely fixes the relative positions of cores and banks. Each point on the grid may be occupied by either a core or a bank of the L2 cache. Each core and bank is also associated with an orientation. The "top" of the core or cache is the connection point for the bus that connects all the cores to each bank.

During coreplanning, there are two types of moves allowed on the fixed grid. The first move swaps the location of two objects (either a core or a bank) at two randomly chosen grid points. The second move rotates the object at a randomly chosen grid point to another orientation. The temperature is recalculated after every coreplan move because of the large change to the power map.

The final cost function used for coreplanning is similar to that used for module-planning. The only difference is the addition of a $(\delta * bus\_area)$ term, where δ is a user defined weighting constant, and bus_area is the total routing area occupied by the memory bus (discussed below).

IV. Bus Routing

One issue that is unique to coreplanning is dealing with the memory bus that connects the cores and the cache banks. In a fixed template-based heterogenous coreplan, the bus length will change only in a minor way during optimization. Conversely, in a non-template-based optimization, moving cores and caches around can change bus length drastically. Because the bus is a major architectural issue, it must be considered during the coreplanning process. Because there are only a small number of memory buses in the banked system, we use a simple router to perform full bus routing after each coreplan annealing move [11].

There are 8 buses in the 8-core system considered here. One bus runs from each cache bank to the L1 caches of all cores. The banks are divided by address so there is no need for communication between banks. The bus router begins by selecting a bus and then choosing the shortest segment to connect any two terminals. The algorithm always analyzes new segments by starting with a zero bend segment, then considering a one bend segment. The router greedily adds the shortest segments available until all terminals of the bus are connected. Then, the router moves on to the next bus. In the case where a full route is not found for each bus a simple overlap-unaware trunk router is used to estimate the total bus area. The trunk router returns a consistently higher bus area than that of the segment-based router, so it effectively forces the annealer to choose coreplans that are routable using the segment router.

V. Floorplanning Algorithm

Multi-core floorplanning in this paper refers to the combination of both module-planning and coreplanning. The goal is to simultaneously optimize both the module and core locations in order to maximize the performance and minimize the maximum temperature. The temperature of a floorplan is highly dependant on the module-to-module thermal coupling that occurs between neighboring cores. The two different move granularities used in this floorplanner impact the cost function and the final floorplanning results in different ways. For example, a coreplan move will not change the area of each of the module-plans or the weighted wirelength within each module-plan. To save execution time, the floorplanner exploits this difference by calculating the area and wirelength objectives only once if a set of coreplan moves is going to be evaluated. However, if the coreplan and module-plan moves are mixed together then the full set of costs is recalculated for each move.

A. Static Multi-Granularity Floorplanning

There are several options to consider when combining coreplanning and module-planning moves. One option is to consider them to be equivalent moves and allow either to be randomly selected during annealing. However, this option ignores the fact that moving a core to a different section of the chip will completely change its thermal environment. Using this option would require careful selection of a temperature update policy and the ratio of selection between coreplan and module-plan moves. This method is referred to as the *Random* method.

A second option is to perform a single coreplan move followed by a large number of module-plan moves. The coreplan move is either accepted or rejected and then the module-plan moves begin. The module-plan moves are then accepted or rejected based on the cost function as they are made. The most obvious issue with this method is that it limits the amount of optimization that each coreplan is allowed to receive before the next coreplan is evaluated. This can result in inefficient optimization speed. This method is referred to as the *Inner Loop* method.

The final non-adaptive method analyzed in this work uses a number of coreplan moves followed by a number of module-plan moves. For every annealing temperature level a number of coreplan moves are first accepted or rejected and then a number of module-plan moves are accepted or rejected. This method is referred to as the *Two Sets* method.

B. Adaptive Multi-Granularity Floorplanning

Additionally, adaptive methods may be used to intelligently choose between the two types of moves available. If one type of move is consistently failing to provide a good avenue of investigation then it may be beneficial to switch to the other. Two adaptive methods are discussed and evaluated here.

The first adaptive method springs from medium access control methods used in networking. This method relies on credits to determine the success rate of each type of move. Each type of move (coreplan and module-plan) is initialized with a number of credits at each annealing temperature level. Whenever a move is completed, the number of credits for that move type is increased if the move is accepted or decreased if the move is rejected. When a move type runs out of credits then the other type of move is selected instead. Effectively, when one type of move is doing badly, the algorithm switches to the other type. The number of credits that are given to each move type is based on the acceptance percentage of the previous annealing temperature level. When more moves are accepted the number of credits given is low, when fewer moves are accepted the number of credits given is high. This method is referred to as the *Adaptive Credits* method.

The second adaptive method considered is based on the Inner Loop fixed floorplanning method described above. For each annealing

TABLE II
THE ARCHITECTURE USED IN THE EXPERIMENTS.

Parameters	Values
Number of cores	8 cores
Clock Frequency	5 GHz
Fetch width	4-wide
Issue/Commit width	4-wide
Branch predictor	Combining: 16K entry Metatable Bimodal: 16K entries 2-Level: 11 bit BHR, 16K entry PHT
BTB	2-way, 2048 sets
L1 I- and D-Cache	32KB 2-way 32B Line
I- and D-TLB	64 entry
L2 Cache	8MB 8-way Unified 32B Line
L1/L2 Latency	2 cycles / 12 cycles
Main Memory Latency	500 cycles
LSQ Size	64 entries
ROB Size	176 entries
Functional Units	4 Int ALUs, 2 FP ALUs

temperature level a limit for the number of bad module-planning moves per coreplan move is allocated. This limit is again based on the previous annealing temperature-level's acceptance percentage. For a high acceptance percentage, a small limit is used. For a low acceptance percentage, a large limit is used. Whenever this limit of bad module-plan moves is reached, the inner loop ends and the next coreplan move is attempted. The intuition is that a large number of bad module-plan moves may be indicative of larger problems at the coreplan level. This method is referred to as the *Adaptive Loop Limit* method.

VI. Experimental Setup

A. Performance Simulation

Performance simulation of multi-core processors is a time-intensive task. As a result it is important to choose a simulator that runs quickly. Performance simulations in this work are run using a version of SESC [19] modified to be floorplan-aware by adding tunable latencies to various microarchitectural operations. For example, a tunable, cycle-level delay is added between the fetch and issue stages of the pipeline. After the fetch function of the simulator is called, the instruction object is stored and delayed for the set number of cycles before being released to the issue unit. The delay used for each wire is calculated based on the target cycle time and the wire length in the floorplan under consideration [8]. The delay of each of the architectural modules is assumed to match that of the target cycle time (in this case, 5 GHz).

The architectural parameters used in this work are summarized in Table II. The sizes of the microarchitectural modules in the multi-core floorplan are estimated using CACTI [20] and GENESYS [7]. SESC includes a power model that is based on Wattch [1]. These power numbers that are output by SESC are used in this work as the steady-state power dissipation for each module in the floorplan. We assume a total power budget of 150 Watts. Temperature calculation is a large bottleneck in thermal-aware floorplanning. The block model available in HOTSPOT [10] is used during optimization and the more accurate (but slower) grid model is used to report the final results below.

B. Data Reporting

Select programs from the SPLASH-2 [21] parallel benchmark suite are used in this work. The IPC value for a benchmark is calculated by adding together the IPC values of each core in the multi-core architecture over an entire benchmark run to represent the overall throughput. Our performance optimizations are identical between the single-core and multi-core cases. Therefore, the only differences between IPC values come as a result of differing amounts of optimization as a result of thermal environment changes, or because of our stochastic optimization techniques. In our experiments it was observed that the IPC values between all of the single-core and multi-core cases varied randomly. This indicates that the thermal optimizations did not adversely effect the performance of the microarchitecture of one case over the others. Different runs with the same input parameters produced floorplans with IPC values that varied around a nominal point. All of the IPC values generated by our experiments are very similar to those reported in Section VII-A.

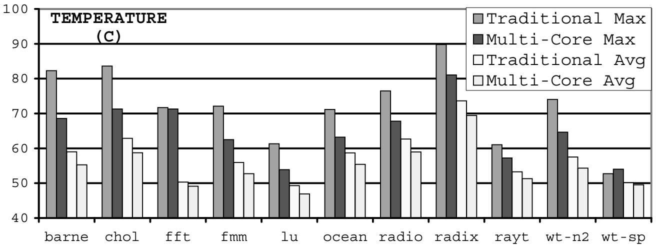

Fig. 3. A temperature comparison between traditional-style and multi-core-aware homogeneous module-planning

The reported maximum temperature of a floorplan is calculated in the following way. First, HOTSPOT's grid model is run for the power profile of each benchmark in the benchmark set. Next, the maximum block temperature for each benchmark is saved. The maximum of these block temperatures among all the benchmarks is used as the maximum temperature of a floorplan. The average temperature of a multi-core floorplan is calculated as follows. The maximum temperature of each block is averaged together over each benchmark's power profile. Next, the maximum of the averages among the benchmarks is used as the floorplan's average temperature.

The execution time for each of the experimental runs depends significantly on the number of independant cores being optimized and the speed of the thermal analyzer. This work does not attempt to address execution-time minimization so these results are not reported here. There are many published techniques targeted at addressing execution-time minimization and almost all are useable in the context of this work. In general the execution time of the implementation of our algorithm used for these experiments was approximately one hour per-core for module-planning and two hours per-core for combined core-planning and module-planning. This long execution time is the result of the 11 temperature calculations that need to be completed each time the temperature is updated and the efficiency of the temperature calculation.

VII. Experimental Results

A. Homogeneous Module-Planning

First, we examine the benefits of the most basic move to multi-core floorplanning. In traditional-style homogeneous module-planning, the temperature is evaluated in a single-core environment. In multi-core-aware homogeneous module-planning, the temperature is evaluated in a multi-core environment. Traditional-style module-planning ignores heat coupling with neighboring cores.[2] A comparison of the maximum and average temperatures between traditional-style and multi-core-aware module-planning is presented in Figure 3. IPC results for the same set are given in Figure 4. For these results floorplanning is performed using the fixed homogeneous template from Figure 2

[2] This traditional module-planning serves as the baseline for all subsequent experiments that report IPC and temperature ratios.

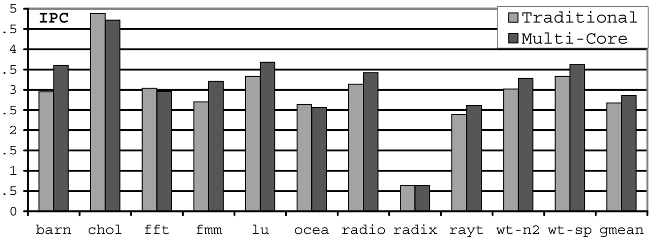

Fig. 4. An IPC comparison between traditional-style and multi-core-aware homogeneous module-planning

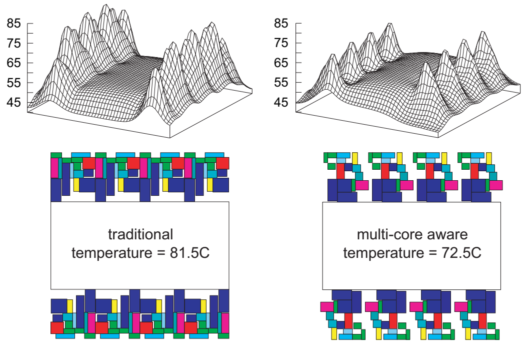

Fig. 5. A temperature profile comparison between traditional-style and multi-core-aware module-planning. The traditional-style module-plan did not consider thermal coupling with nearby cores. Hot modules near the core boundaries cause the average and maximum temperature to be higher.

The IPC result of the multi-core-aware module-planning is nearly the same as that of the traditional-style module-planning. However, the maximum temperature for the multi-core-aware module-planning is 10% lower and the maximum average temperature is 6% lower. A comparison of the temperature profiles from both floorplans, shown in Figure 5, explains the improvement. We note that the traditional-style module-planning produces a temperature distribution that has hot modules near the chip boundaries. In multi-core-aware module-planning, the neighboring cores will tend to push their hot modules away from the boundaries of the chip. Figure 6 shows the module-plan of the single cores used in the temperature profiles of Figure 5. The distribution of the ALUs and the register files demonstrate some of the reasons for the temperature profile differences between the two approaches. These results emphasize that multi-core-aware module-planning is better than traditional-style module-planning for homogeneous (= single core-type based) multi-core systems.

B. Heterogeneous Module-Planning

Next, we compare the six fixed template-based heterogeneous coreplan types shown in Figure 2. For the purpose of fairness among the different numbers of core types, each coreplan type is given the same per-core optimization time budget. For example, an optimization using 2 core types would be given twice the total time budget of an optimization with a single core type. We also include the results from the traditional-style homogeneous module-planning and multi-core-aware homogeneous module-planning presented in Section VII-A for comparison. The baseline is traditional-style homogeneous module-planning.

Figure 7 shows the temperature results for this optimization. As expected, with the largest design effort comes the best results, and the 8-type coreplan has the lowest temperature values. The largest drop in temperature values occurs with the move from 2 core types to 3 core types. The IPC of the different coreplans were observed to vary as a result of thread-to-core assignments that we do not control in this work, as well as module-plan differences that induce variation in microarchitectural delays. No pattern in the IPC values is readily discernible. Though these coreplans conflict with the decreased design effort benefit of the move to multi-core they show that there is some temperature benefit to be gained.

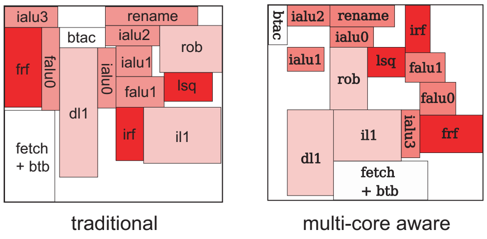

Fig. 6. Module-plans for the cores in Figure 5.

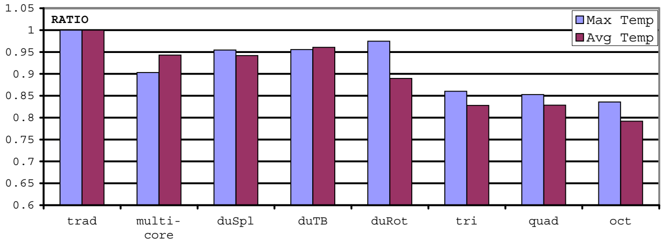

Fig. 7. Comparison between several template-based heterogeneous coreplan types.

C. Multi-Granularity Algorithm Comparison

Next, we show a comparison of the results between the 5 different styles of multi-granularity floorplanning discussed in Section V. The three fixed and two adaptive algorithms were run with one type of core in the coreplan (homogeneous style). The maximum and average temperature of the results for each algorithm are shown in Figure 8.

The lowest temperatures are achieved using the Adaptive Loop Limit method, which provides a 19% and 27% drop in maximum and average temperature, respectively, when compared to the traditional-style single-floorplan results. Because the Adaptive Loop Limit method provides the best temperature results, all subsequent experiments use this method. An example homogeneous floorplan and the resulting bus routing from the Adaptive Loop Limit method is shown in Figure 9.

D. Floorplanning with Cache Banks

In this section we study the impact of floorplanning with cache banks. The L2 cache is partitioned into 8 banks so that the area of each bank is similar to that of a core. Because each cache bank has an area similar to that of a core the 8 cores and 8 banks can be easily arranged into a 4×4 grid. This grid is not of a fixed size. The grid merely fixes the relative positioning of the cores and banks, that is, this core is above/to the right of this bank, and so on. We use our Adaptive Loop Limit method to optimize the floorplan (= both module-plan *and* coreplan) of these 16 blocks. We use two hand-optimized coreplans, named *edges-type* and *checker-type*, as shown in Figure 10, for comparison. The edges-type coreplan is meant to provide a comparison to the fixed homogeneous case presented in Section VII-A.

Figure 11 shows a comparison of the average and maximum temperatures. For this comparison, our Adaptive Loop Limit is based on a single floorplan for all cores (= homogeneous). As expected, the checkers-type coreplan results in the lowest temperature, and also the highest memory bus length. Our Adaptive Loop Limit floorplanning method results in similarly low temperature levels; however, the memory bus length is more comparable to the edges-type coreplan.

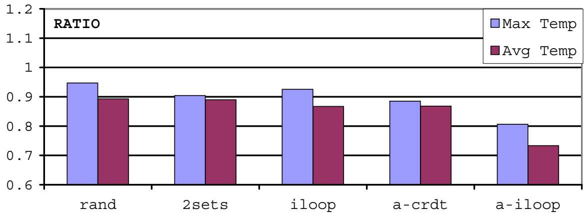

Fig. 8. Comparison among the 5 multi-granularity floorplanning algorithms.

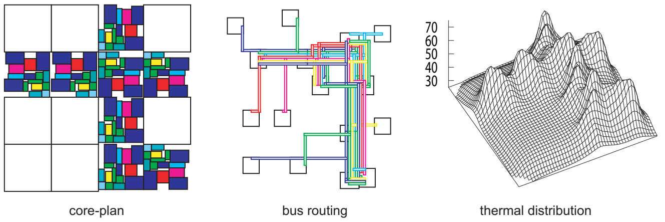

Fig. 9. An example of a homogeneous floorplan from the Adaptive Loop Limit method, the resulting memory bus routing, and the thermal profile from a high-activity benchmark. Temperatures are in degrees Celcius.

This shows that our Adaptive Loop Limit floorplanning algorithm effectively provides both the temperature benefits of the checker-type coreplan and the lower bus length of the edges-type coreplan.

VIII. Conclusions

Design for multi-core systems presents a new challenge for the EDA community. Early phases of the design cycle, such as microarchitectural-level floorplanning, must take into account many factors. This paper presents a microarchitectural floorplanner targeted at multi-core systems. Our methodology successfully reduces operating temperatures while maintaining system performance by taking into account module-to-module interactions with neighboring cores. In the simplest approach our multi-core design achieves a 10% decrease in maximum temperature and a nearly 6% decrease in average temperature with a negligible change in IPC relative to traditional-style floorplanning.

Our Adaptive Loop Limit multi-granularity multi-core floorplanning method achieves a 19% reduction in maximum temperature and a 27% reduction in average temperature compared to traditional single-core floorplanning. We also present results for heterogeneous floorplan types in a multi-core system with a homogeneous architecture. All improvements in this paper come without physical-aware scheduling algorithms or dynamic temperature control schemes.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proc. IEEE Int. Symp. on Computer Architecture*, 2000.

[2] P. Chaparro, J. Gonzalez, G. Magklis, Q. Cai, and A. Gonzalez. Understanding the thermal implications of multi-core architectures. *IEEE Trans. on Parallel and Distributed Systems*, 18:1055–1065, 2007.

[3] J. C. Chi and M. C. Chi. An effective soft module floorplanning algorithm based on sequence pair. In *Proc. IEEE Int. ASIC/SOC Conf.*, 2002.

[4] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis. Microarchitecture evaluation with physical planning. In *Proc. ACM Design Automation Conf.*, 2003.

[5] J. Cong, J. Wei, and Y. Zhang. A thermal-driven floorplanning algorithm for 3D ICs. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, 2004.

[6] J. A. Darringer. Multi-Core Design Automation Challenges. In *Proc. ACM Design Automation Conf.*, 2007.

[7] J. C. Eble, V. K. De, D. S. Wills, and J. D. Meindl. A Generic System Simulator (GENESYS) for ASIC Technology and Architecture Beyond 2001. In *Int'l ASIC Conference*, 1996.

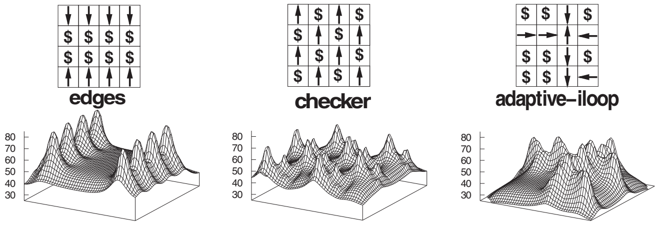

Fig. 10. Hand-optimized coreplans with cache banks and the resulting thermal profiles compared with our algorithmic coreplan. Temperatures are in degrees Celcius.

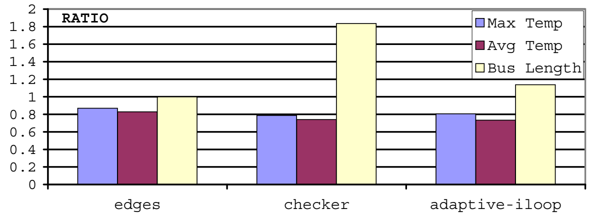

Fig. 11. Comparison of the average and maximum temperatures among edges-type, checker-type, and our Adaptive Loop Limit algorithm. Our Adaptive Loop Limit algorithm shows comparable temperatures to checker-type but maintains much lower memory bus length.

[8] M. Ekpanyapong, J. Minz, T. Watewai, H.-H. Lee, and S. K. Lim. Profile-guided microarchitectural floorplanning for deep submicron processor design. In *Proc. ACM Design Automation Conf.*, 2004.

[9] M. Healy, M. Vittes, M. Ekpanyapong, C. Ballapuram, S. K. Lim, H.-H. S. Lee, and G. H. Loh. Multi-objective microarchitectural floorplanning for 2d and 3d ics. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(1):38–52, 2007.

[10] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *IEEE Trans. on VLSI Systems*, 14(5):501–513, 2006.

[11] D. H. Kim and S. K. Lim. Bus-aware microarchitectural floorplanning. In *Proc. Asia and South Pacific Design Automation Conf.*, 2008.

[12] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir. Design and management of 3d chip multiprocessors using network-in-memory. In *Proc. IEEE Int. Symp. on Computer Architecture*, 2006.

[13] C. Long, L. Simonson, W. Liao, and L. He. Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects. In *Proc. ACM Design Automation Conf.*, 2004.

[14] R. Mukherjee and S. O. Memik. Physical aware frequency selection for dynamic thermal management in multi-core systems. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, 2006.

[15] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo. Designing application-specific networks on chips with floorplan information. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, 2006.

[16] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Vlsi module placement based on rectangle-packing by the sequence-pair. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, 1996.

[17] V. Nookala, Y. Chen, D. J. Lilja, and S. S. Sapatnekar. Microarchitecture-aware floorplanning using a statistical design of experiments approach. In *Proc. ACM Design Automation Conf.*, 2005.

[18] U. Y. Ogras and R. Marculescu. Energy- and performance- driven noc communication architecture synthesis using a decomposition approach. In *Proc. Design, Automation and Test in Europe*, 2005.

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

[20] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. CACTI 4.0. Technical Report 2006.86, HP Western Research Labs, 2006.

[21] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proc. IEEE Int. Symp. on Computer Architecture*, 1995.