# **Rapid Prototyping of CMP Floorplans: A Technical Report**

UVA Dept. of Computer Science TR CS-2012-02, March 30, 2012

Gregory G. Faust, Brett H. Meyer<sup>1</sup> and Kevin Skadron Department of Computer Science, University of Virginia, Charlottesville, VA {gf4ea, skadron}@cs.virginia.edu, brett.meyer@mcgill.ca

#### Abstract

The Computer Architecture literature is now replete with papers concerned with the change in architectural direction from ever more complex single cores to single chip multi-core designs. Along with this opportunity come major challenges. Among them is the sheer size of the space of possible designs. The investigation of this space is far from complete. What is needed to aid in this task is an integrated suite of tools that provides support throughout the design life-cycle, from early prototyping to final design. Here we present a floorplan tool targeted towards early prototyping of pre-RTL CMP design concepts. As such, it acts as a complement to traditional floorplan tools that are more appropriate later in the design process. Early phase CMP design investigations into the distribution of power and temperature, pin allocation, core/cache cluster size, and NoC design trade-offs are examples of experiments that can benefit from CMP layout information without many of the design details needed to drive a traditional floorplanner. We use two such studies to validate the benefit of the tool in rapid prototyping.

The floorplan is specified using a model similar to that supported by GUI toolkits such as Java Swing or Windows Presentation Foundation. The floorplan design is comprised of a hierarchy of components placed within containers that provide a variety of layout services. These services include support for geographic hints for component placement, generalized grid layouts, and other layout algorithms. The tool can also be integrated with other tools in the suite by absorbing area information from tools such as McPAT, and producing output for ingestion by tools such as HotSpot. In addition, the current services can act as the basis for building more specific layout algorithms such as those targeting a certain type of NoC configuration, cache partitioning strategy, or SIMD design. Finally, the architecture is flexible enough to allow for the inclusion of a traditional floorplan tool such as ParquetFP to support detailed floorplanning once enough design information is available. This tool, which we call ArchFP, can be downloaded from http://lava.cs.virginia.edu/archfp.

## **1. Introduction**

The advance of Moore's Law has increased the number of transistors available on a chip faster than can be profitably utilized by single cores of ever increasing complexity. Instead, the focus of chip design has shifted towards the inclusion of many cores on a chip leading to the production of Chip Multi-Processors (CMPs). The size and complexity of the design space for CMPs is staggering. It spans multiple dimensions such as the number, type, and complexity of the cores, the size of on-chip cache, the cache sharing model, the type of on-chip network interconnect between components, local vs. global time synchronization, the type and number of memory controllers, etc.

<sup>&</sup>lt;sup>1</sup> Brett Meyer is now in the ECE Dept. at McGill University.

While there is a combinatorial explosion of the possible system architectures to contemplate, there is also a number of increasingly hard to overcome constraints that must be dealt with. These include pin count, power density, temperature density, and total die size. Each of these are worthy of study in their own right. But it is also becoming increasingly clear that making system wide architectural decisions while attending to one or two of these constraints at a time can often lead to sup-optimal designs [10].

In order to investigate the large CMP design space in a comprehensive fashion, increased emphasis must be placed on integrated tool suites capable of modeling single chip multiprocessors and their on-chip support systems. In addition, to allow rapid early stage investigation, the suite must contain tools capable of modeling systems at different levels of detail. An example of such a tool is McPAT [4]. It contains hierarchical power, area, and timing models for various components which include three different levels of detail; Architectural, Circuit, and Technology. Modeled components include cores, router and crossbar based NoCs, caches, memory controllers, clocking circuitry, etc. McPAT has been used to investigate a number of core cluster configurations in terms of their total area, power, and NoC latency without considering the actual layout of the various configurations.

However, many other CMP design investigations do require layout information. For example, in several studies [3, 22, 23] conducted at the University of Virginia, the effect of CMP layout on peak chip temperature was investigated for a variety of possible chip configurations and target uses. It was shown that the layout of the CMP, and in particular the relative placement of components that tend to run hot and those that run cold, can have a marked impact on temperature. In addition, it was shown that the severity of the temperature different is exacerbated in applications such as laptops in which the size of any cooling apparatus is severely limited. Finally, the lack of temperature-aware floorplanning can force runtime throttling of the voltage and/or clock speed thereby effecting performance.



Figure 1 -- Extended McPAT Model Hierarchy. The blue rectangle encloses the domain for a traditional floorplan tool, while the red rectangle encloses the domain for the layout algorithms presented and proposed here. Note that there is an area of overlap between the two.

The UVA temperature studies provide a motivating example for the inclusion in a tool suite of a CMP level floorplanner that can operate at a high level of abstraction. Traditional floorplan tools typically operate with detailed information about the components and wires which comprise the design. However, this level of detail may not be appropriate to early stage CMP design. In addition, as discussed below, traditional floorplanners do not account for the challenges facing

layout at the CMP level which is dominated by different design concerns than layout lower in the hierarchy. Therefore, we present a novel approach to CMP floorplan layout. Figure 1 shows the McPAT levels of hardware description augmented with a higher "CMP" level. The overlay shows the targeted domain of traditional floorplan algorithms, and those described here.

More specifically, we borrow the model of Graphical User Interface design toolkits such as Java Swing [6] and Windows Presentation Foundation [7] in the construction of a novel framework for floorplanning. After all, such toolkits are also designed to layout (rectangular) shapes in a 2D space. In this model, components such as cores, caches, crossbars, etc. are placed within containers. Associated with each container is a layout algorithm called a "layout manager" (LM). Containers are themselves components; therefore the model is inherently hierarchical. Finally, the model is implemented as a class library, allowing for extensibility of the model with additional layout algorithms of arbitrary generality or specificity. In particular, a traditional floorplan algorithm can be included smoothly within the architecture, as can very specific knowledge-based LMs. To the best of our knowledge, no one has previously proposed this model of hierarchical containment with differing layout algorithms for hardware floorplanning.

As proven in many other contexts, a very powerful tool paradigm is to have a few simple primitives that operate well in conjunction with one another. Therefore, we have started by implementing a small collection of such LMs first. Currently implemented LMs include one that is driven by geographic hints about where components should be placed, another that supports repeating grids, and a third that loads in pre-existing floorplans from a file. These simple, intuitive, easy to use models of layout can be used in combination to produce interesting floorplans. For example, the designer can use the geographic LM to create component clusters which are then replicated across the chip in a grid; a pattern that appears often in CMP designs. In addition, such a set of primitives acts as the foundation upon which to build more complex LMs, especially ones specific to important CMP patterns such as NoC topologies, cache sharing models, or more exotic CMP designs. The framework presented here acts as an organizing framework into which such LMs can be added and used in combination in a consistent fashion.

The remainder of this document is organized as follows. Section 2 will discuss related work in traditional floorplan algorithms and GUI frameworks. Section 3 will provide details of the layout algorithms currently implemented. In section 4 we will use the new floorplan tool in two CMP design scenarios as case studies to evaluate the benefits of the approach. Section 5 will present next steps towards turning the current implementation into a more complete tool that is also better integrated into a tool suite. Section 6 is the conclusion.

# 2. Background and Related Work

#### **2.1 Floorplans**

Placement and Floorplanning are two related activities that have been part of chip design for decades. While optimal floorplan design is an NP-Hard problem, various approximation techniques have been used to provide practical designs. Floorplanning is a rich and complex topic that cannot be fully covered here. Classic texts that include chapters on these topics include Gerez [1] and Sarrafzadeh and Wong [2]. These traditional floorplan algorithms take as input a collection of components and the wires between them. They then try to find a non-overlapping 2D layout of the components that minimizes the value of an objective function such as a linear combination of the total area and the total wire length, while staying close to square in shape. Optimization algorithms use graph or tree representations, and use linear programming or

simulated annealing to approximate optimal solutions. An example floorplan of an Alpha EV6 computer at the architectural level of detail is shown in Figure 2.

FPMap1	FPMap2	IntMap	In	tO	RF1	RF2		
FPMul1	FPMul2			~~				
FPRF1 FPRF2	FPRF3 FPRF4	4 FPQ	Lď	StQ		ALU		
FPAdd1	FPAdd2		ITB1	ITB	2			
Bpred1	Bpred2	Bpred3	DTB	1	DTB2	DTB3		
	Icache				Dcache	•		

Figure 2 – Example Floorplan for the EV6 version of the Alpha chip at what the McPAT hierarchy would call the Architectural level of detail. Floorplans for more detailed levels in the hierarchy are significantly messier with a much wider disparity in component sizes, many very small pieces of a few transistors in size needed to sew the larger pieces together, and "white space" or areas of the floorplan in which no transistors are located. This is a "slicing" floorplan in that one can recursively partition this floorplan into pieces by drawing a line completely through the remaining area at each level of the recursion.

For example, a popular technique is to represent the current floorplan as a tree of rectangular areas, with hardware components as the leaves, and larger composite rectangles as one goes up the tree. A "slicing" floorplan is formed by such a tree which is binary. As the name implies, the remaining area is sliced into two not necessarily equal parts either horizontally or vertically at each level of the layout. The space of possible layouts is searched via simulated annealing. At each step, a "move" is randomly generated that takes the current floorplan and turns it into a new floorplan. Legal moves are local perturbations of the tree structure, and include rearranging the children of a node, moving children between nodes, laying a block on its side, etc. After each move, the objective function is evaluated to determine if the move results in a floorplan which is better or worse than the previous one. As with any simulated annealing algorithm, moves that

result is worse floorplans are accepted randomly with decreasing probability as the "temperature" of the system goes down. Eventually, towards the end of the run, only moves that produce improved scores are likely to be accepted. Throughout the process, the globally best configuration is remembered and returned as the answer at the end of a specified number of moves. One advantage of this approach is that all elements in the floorplan are handled in a simple and consistent fashion. The use of simulated annealing as the search algorithm means that traditional floorplanning can take a lot of runtime, and produce in any given run, a floorplan of uncertain quality.

An added complexity in this process is that some individual components may have a fixed rectangular shape and others not. Non-fixed components have a range of potential rectangular shapes specified as a range of allowed "aspect ratios" (AR), which is the ratio of the width/ height. When present, such components lead to an enlarged search space for the floorplanner, but also often provide flexibility that can lead to better floorplans. An additional move for such a floorplanner is to change the shape of one or more components within their AR constraints.

The use of a tree structure in these algorithms seems to imply a hierarchical description of the design space. However, this is not really the case. The tree structure is just a convenient representation in which to generate moves, and the hierarchy has neither stability nor semantic meaning. Many floorplanners do support the notion of a "macro" which is a pre-layed out standard component that essentially acts as a leaf in the current level of the design. However, the actual space being explored is still essentially a flat collection of leaf components.

Some modern floorplanners such as ParquetFP [9] and Fast-SA [20] do support another type of hierarchy in their layout algorithm. As a prepass to the algorithm described above, they first find subsets of components that are strongly connected by many wires. They then form one component for each cluster, allow it some AR flexibility, then layout just these clusters as indivisible units. This results in a set of blocks with fixed ARs that form the top level of the layout. These blocks then have their interiors layed out one at a time within their specified AR. This requires "fixed-outline floorplanning". The added complexity is that many generated moves may violate the fixed outline constraint. Such an algorithm must either temporarily allow such floorplans, or are likely to quickly fall into a locked position in which no move will be allowed. Chen et al. [21] proposes to apply such techniques recursively.

Still, the hierarchy represented in the model presented here is fundamentally different. The hierarchical inclusion of components in containers is stable and has direct impact on the resultant design. Components do not hop between containers during layout. More importantly, completely different layout algorithms can be, and typically are, applied at different levels in the hierarchy. Therefore, the inclusion of a component in a level of the hierarchy does have semantic meaning in the design space. As more specific layout algorithms are added to the system that contain domain specific knowledge about how their children should be layed out, the semantic content of component placement in containers goes up. This model has the potential advantage of resulting in better layouts. However, it does place upon the user the added burden of specifying which components go into which containers, and which layout algorithms to use in those containers.

Several observations are relevant to evaluating the relative merits of these two models. First, as a traditional floorplanner can be included as one layout algorithm, the presented framework can be no worse than the current standard. As is now done, one would use bulk methods for loading large numbers of components and wires into such a container by reading them in from a file. Second, the layout algorithms either presented or proposed here for CMP level design are not

meant to handle large numbers of disparate components. As stated earlier, they are meant for the higher layers in Figure 1 in which there are either fewer total components, or fewer types of different components, or both. Finally, the current objective functions of traditional floorplanners are not well matched for the design concerns that predominate at the CMP level.

Current floorplanners optimize for total wire length, total area, and perhaps target AR. However, at the CMP level it might be more important to optimize for other factors. One such factor might be minimizing the number of hops in the worst case route of the NoC. In addition, the designer may well know that the latency of some wires is more crucial to the design than others. And for some wires, the latency need not be minimized so long as it is below some critical value. While it is possible to include such factors into an objective function, the more factors there are in the function the wider the design space that must be searched. This would lead to even longer runs times to obtain reasonable floorplans.

Instead of changing the objective function in the floorplanner, many CMP studies have wrapped an extra evaluation function around the results of the floorplanner. We have already referred to the UVA work on temperature aware floorplanning [3, 22, 23] and the use of McPAT [4] to investigate of the power and area trade-offs for various cluster sizes in grid NoC configurations. Later we will use these two examples as case studies for our floorplanner. But there are many additional examples. Kumar et al. [5] did a detailed investigation of the area, power, and performance ramifications of several CMP NoC organizations, using NoC latency as a primary measure of floorplan optimization. Benini [11] did a similar study for application specific NoC design for wireless communication SoCs. Murali et al [12] studied NoC topology for 3D SoCs optimizing, among other things, the placement of Through Silicon Vias in the generated floorplans. Meyer et al [13] studied the optimization of system reliability and cost in application specific SoCs using various NoC configurations. Meyer has stated [private communication] that over 90% of the runtime in the exploration of the relevant design space was spent in the floorplanning component. Pande et al [8] also looked at area, power, and performance of various NoC topologies. Here they chose to investigate 8 different packet based NoC topologies instead of busses and crossbars. They floorplans they used were extremely rudimentary, and they calculated most of their wire lengths from analytical models, not actual layouts.

In the model presented here, such investigations can be facilitated by the creation of domain specific layout algorithms that attend to the factors important to the particular CMP design space under investigation. This can be done either by building domain specific knowledge into a novel LM, or by using existing LMs in a fashion directed by the designer's intuition about which organizations are likely to result in the desired attributes. The downside of this approach is that the designer has to take the time to write these custom layout strategies.

Current floorplan algorithms are measured by their performance in standard floorplan benchmarks such as GSRC from UC Santa Clara. The metrics used include total area, percent of white space, total wire length, and runtime. As the floorplanning problems presented in such a benchmark relate to post-RTL floorplanning, it is not the domain of the floorplan algorithms presented here. Therefore, we have not pursued testing against such benchmarks.

#### **2.2 GUI Design Toolkits**

The idea of using hierarchical descriptions in the specifications of GUI designs has been around for a long time, and it is not our intention to recapitulate this history here. Modern GUI toolkits such as Java Swing [6] or Windows Presentation Foundation (WPF) [7] are 3<sup>rd</sup> generation

systems built upon the lessons learned in previous systems. Java Swing first shipped in 1998 has the exact same architecture of components, containers, and layout managers that we are proposing to use here. The current version of Swing contains about 10 different LMs, supporting horizontal, vertical, grid, and box-and-spring placement of components. In addition, as is typical in Java, the contract for the LM is defined as an interface. Therefore, anyone can create their own LMs either on top of the base set or completely independently and have them participate in the overall Swing architecture in a consistent fashion. In fact, there are many 3<sup>rd</sup> party Swing LMs available on the web.

WPF is the latest in a series of component/container GUI models from Microsoft. It is intended to span the creation of Windows apps, as well as web-based apps, and it therefore built on top of the .NET infrastructure. Therefore, LMs are also defined in terms of interfaces, with 3<sup>rd</sup> parties providing custom LMs. It first shipped in 2006 with 6 LMs, such as stack, wrap, grid, etc. These have very similar functionality to their Swing counterparts. It is interesting to note that in the WPF architecture, layout is done recursively in two passes. In the first, all of the components are queried for their sizes. In the second, the actual placement is performed. In our system, layout also proceeds in these same 2 passes, one to get the total area needed by the components if layed-out with no white space, followed by a recursive descent in which upper layer LMs dictate a target AR to lower layer LMs. A difference is that on the way back up from the layout pass, floorplan LMs are expected to perform fix-ups of their own size and shape if their inferiors were unable to meet the expected area and/or AR goals.

Unlike Swing or WPF, our system is built in C++. Therefore, the LM abstraction is defined in terms of an abstract base-class with virtual methods for component addition, layout, outputting to a file, etc.

There is a long tradition of graphical (CAD) tools for both HW design and GUI design. Such tools allow designers to directly translate their ideas about layout (and other design issues) into portions of a specification for the system being developed. WPF provides such a graphical tool to specify layouts, but Java Swing does not. A discussion of such tools is beyond the scope of this paper. No CAD tool for the LMs presented here is currently contemplated.

# **3. Current Implementation**

A key goal of the floorplanner is to integrate with the growing suite of tools under development at UVA for CMP design space investigation. The tool suite already contains tools such as MV5 [15], McPAT [4], HotSpot [3], and ParquetFP [9]. Most of these tools are written in C++ which is why C++ was also chosen as the implementation language of the floorplanner. Eventually, the floorplanner will be better integrated with the rest of the tool suite (see Section 5). At that time, the leaf components in the floorplan hierarchy will be components from MV5. However, as of this writing, the MV5 components do not contain area information which is instead provided by McPAT. Therefore, the floorplanner currently includes a leaf component wrapper class meant to latter be replaced by the appropriate MV5 component base class. This wrapper class contains the following information:

• Component type. Future LMs that are specific to various CMP structures will take the type of the components into account when doing layout. In the current, more general LMs, the type is ignored during layout. However it is used to provide information for output.

• Minimum and maximum aspect ratios. For components that have fixed ARs, the minimum and maximum will be the same. The minimum AR need not be square.

All components in the system are derived from a base class, which is very similar to a component in one of the GUI frameworks.

- Components store their (x, y) location information relative to their container, not the overall floorplan. That is, relative positioning is used, not absolute positioning.
- Components also have a specified area.
- Components have a width and height. However the actual width and height of the component is often not known until after layout has occurred. This is so that the recursive specification of AR information can flow from top to bottom during the layout process. After layout, all components have a known width and height.

All LMs in the system are derived from an abstract container class. The container class contains little more than its list of inferiors. However, the methods of the LMs are where the work of the system is performed. There are two abstract methods on the container base class that all LMs must implement. The first is to output the layout of the LM in HotSpot format. The reasons this output format was chosen are twofold. First, HotSpot is an important recipient of layout information in the tool suite. Second, it has the simplest possible format in which each element simply specifies it name, width, height, and (x, y) location. The second important method for LMs is of course the layout method. It takes a target AR as a goal. Often specific LMs use their target AR in conjunction with information about the number and size of their inferiors to dictate the target ARs for their inferiors. The flow of information during layout is as follows:

- 1. Each container, starting at the top of the hierarchy, calculates their target area as the sum of the target areas of their inferiors. These requests for area information will flow from the top down, while the actually sums are performed on the way back up. After this stage, the top container knows its goal area if no white space is needed.
- 2. Next the container starts applying its layout to its inferiors, often calling those inferiors to lay themselves out according to a target AR. Leaf components are also able to lay themselves out by checking their minimum and maximum AR and complying as closely as possible with the request from above.
- 3. Once an inferior is layed out, the LM checks to see if the inferior's resultant width and height is as requested. If not, it is the LMs responsibility to adjust accordingly.
- 4. As the LM finishes the layout for a given inferior, it then sets that inferior's location.
- 5. Finally, once all inferiors are layed out, the container calculates its own width and height based on the actual location and size of its inferiors.

The system currently implements four LMs. All of them currently produce slicing floorplans. The simplest LM to understand is the grid LM (GLM). It contains a single inferior (of arbitrary nested complexity), and a total number of grid elements. The actual dimensions of the grid are not specified. Instead, during layout, the GLM will determine the best dimensions for the grid based on its requested AR. For example, if the grid layout method is called with a target AR of 2 (meaning twice as wide as high), and the grid contains 8 elements, the GLM will set its grid dimensions to 2 rows by 4 columns. This allows the inferior to be layed out with the least extreme AR targets (closest to a square). The GLM does not duplicate its inferior, but rather the output method asks its inferior to output itself over and over again with different (x, y) locations.

- 1. geogLayout \* dCacheStack = new geogLayout();
- 2. dCacheStack->addComponentCluster(Control, 1, 4, 10., 1., Top);
- 3. dCacheStack->addComponentCluster(L1, 4, 9, 3., 1., Bottom);
- 4. geogLayout \* CoreCluster = new geogLayout();
- 5. CoreCluster->addComponentCluster(ICache, 5, 1, 10., 1., Left);
- 6. CoreCluster->addComponent(dCacheStack, 1, Left);
- 7. CoreCluster->addComponentCluster(RF, 4, 1, 10., 1., Top);
- 8. CoreCluster->addComponentCluster(Core, 16, 3, 2., 1., Bottom);
- 9. geogLayout \* L2Stack = new geogLayout();
- 10.L2Stack->addComponentCluster("EBC", 1, 3.166, 3., 1., Top);
- 11.L2Stack->addComponentCluster("C2C", 1, 3.166, 3., 1., Bottom);
- 12.L2Stack->addComponentCluster(MemCtrl, 2, 3.166, 3., 1., TopBottom);
- 13.L2Stack->addComponentCluster("DMA", 2, 3.166, 3., 1., TopBottom);
- 14.L2Stack->addComponentCluster(L2, 4, 9.5, 2., 1., Center);
- 15.geogLayout \* WholeChip = new geogLayout();
- 16.WholeChip->addComponent(L2Stack, 1, Left);
- 17.WholeChip->addComponentCluster(L2, 12, 9.5, 2.0, 1., Left);
- 18.WholeChip->addComponent(CoreCluster, 2, TopBottomMirror);
- 19. WholeChip->Layout(AspectRatio, 1.0);
- 20.WholeChip->OutputHotSpotLayout("TRIPS.txt");

EBC1			leS	Control1	RF1	RF2	RF3	RF4
MemCtr11 DMA1	L2_15	L2_16	the4 ICach	L1_4	Core13	Core14	Core15	Core16
12_4	1.2_13	L2_14	Cache3 ICac	L1_3	Core9	Core10	Corel 1	Core12
			ache2 I	L1_2	Core5	Core6	Core7	Core8
1.2_3	12_11	12_12	ICachel IC	L1_1	Core1	Core2	Core3	Core4
L2_2	L2_9	L2_10	I Cache6	L1_5	Core17	Core18	Core19	Core20
L2_2	L2_9	L2_10	ICache7 ICache6	L1_5 L1_6	Core17 Core21	Core18	Core19 Core23	Core20 Core24
12_2	L2_9 L2_7	L2_10 L2_8	9 ICache8 ICache7 ICache6	L1_5 L1_6 L1_7	Core17 Core21 Core25	Core18 Core22 Core26	Core19 Core23 Core27	Core20 Core24 Core28
12_2 12_1 DMA2	L2_9 L2_7	L2_10 L2_8	ICache9 ICache8 ICache7 ICache6	L1_5 L1_6 L1_7	Core17 Core21 Core25	Core18 Core22 Core26	Core19 Core23 Core27	Core20 Core24 Core28
1.2_2 1.2_1 DMA2 MemCtrl2	L2_9 L2_7 L2_5	L2_10 L2_8 L2_6	he 10 I Cache9 I Cache8 I Cache7 I Cache6	L1_5 L1_6 L1_7 L1_8	Core17 Core21 Core25 Core29	Core18 Core22 Core26 Core30	Core19 Core23 Core27 Core31	Core20 Core24 Core28 Core32

Figure 3 – TRIPS CMP level floorplan. The blue overlays (added manually for emphasis) show the various portions of the layout put together by the use of hierarchical containment of multiple Layout Managers.

The most flexible of LM is the Geographic LM (GeoLM). To reduce the number of levels of hierarchy that the user of the GeoLM needs to deal with, inferiors added to the GeoLM can take a repeat count. The GeoLM will then automatically create a GLM as an inferior to contain the repeating group. In addition, inferiors to the GeoLM are specified with a geographic hint that indicates where the component is to be placed. The list of currently supported geographic hints includes Left, Right, Top, Bottom, Center, LeftRight, and TopBottom. During layout, the GeoLM takes its inferiors in order, and allocates all remaining space along the specified location to the current component. LeftRight and TopBottom are different from the others in that they expect to be applied to a repeating group of size that is a multiple of 2. They cut the group in half and put each half in the specified location. In addition, the LeftRight and TopBottom hints have a mirroring option and a 180 degree rotation option.

These two layouts alone can be used to produce some interesting floorplans. Consider the following example code snippet, and the resultant floorplan shown in Figure 3. The bottom portion of Figure 3 shows the Architectural layout for the TRIPS CMP [16]. The CMP contains 2 core clusters highlighted by the two blue rectangles on the right hand side of the floorplan. The left hand side is a NUCA L2 cache array plus some off-chip communication components in the upper and lower left corners.

The addComponentCluster method takes a component type, a repeat count, each component's area, the max and min AR constraints for the component, and the geographic layout hint. Lines 1-8 define the core cluster as a combination of a vertical stack of Control and L1 Cache (lines 1-3) and 3 repeating groups of components, ICache (line 5), Register Files (line 7) and Cores (line 8). Line 6 includes the GeoLM from line 1 into the core cluster. Lines 10-14 define the left most column of off-chip components and part of the L2 array. Lines 15-18 put the whole chip together. Of particular note is line 18 in which the entire core cluster is duplicated and mirrored with one statement. Lines 19 and 20 call the layout and output methods on the top-most container in the hierarchy. The picture of the floorplan was produced by converting the HotSpot layout format into PDF using third party tools.

The remaining two LMs are a bag LM which, as its name implies, takes an arbitrary collection of components without any layout hint information. It lays them out from largest to smallest in size within its target AR. The resultant floorplans are almost always one dimensional, because it expects to have a rectangle remaining after each inferior component is layed out. Finally, there is a fixed LM that does no layout, but instead allows its inferiors to decide the size, shape, and location. It is largely used to load in layouts from existing HotSpot files as will be seen in the next section.

## 4. Two Case Studies

The original goal of this project, and a critical next step for this research, is to use the floorplanner in an architectural study investigating some aspect of the CMP design space. Here, to help validate the framework, we will look at two previous CMP design explorations as case studies of how this floorplanner could have been used. The first of the two studies investigated the impact of CMP floorplans on overall chip temperature [3, 22, 23]. A synopsis of this research appears in Section 1. Here we will focus on the floorplans that were considered, and how they could have been produced with the current floorplanner implementation. Figures 4, 5, and 6 show side by side depictions of floorplans as presented in the original paper [22] (on the left) and as produced by the new floorplanner (on the right). The left hand side pictures are color coded to

show hotter temperatures in red and cooler temperatures in blue. In the generated floorplans, the names of the components have been suppressed to enhance clarity.

The core component is the same Alpha EV6 as shown in Figure 2. In all of the floorplans, the core is replicated four times and surrounded by cache. The cores naturally run hotter than the cache, and the cache can act as a cooling buffer for the cores. In addition, different parts of the cores run hotter than others, so the orientation of the cores relative to one another when in close proximity can also materially impact the resultant temperature.

In Figure 4, the cores are placed in a conventional arrangement with mirror reflections in both the x and y planes. This unfortunately places the hottest running portions of the cores, namely the register files and ALUs, in close proximity to each other, and the resultant temperature is significantly higher. Figure 5 shows the same arrangement of cores and caches, but with the cores reoriented to keep the hottest components away from each other. This results in substantial temperature reduction. Finally, in Figure 6, the cores are surrounded by cooler caches, resulting in the best temperature profile of the three floorplans at the expense of slightly higher communication latencies between the cores.



Figure 4 – Four Alpha cores surrounded by cache oriented with their Register Files and ALUs in close proximity. The red color in the center of the picture on the left shows the resultant high temperature.

To produce these floorplans, the fixed LM was used to load in the floorplan for the Alpha core from an existing HotSpot file. The remainder of the layout was straightforward use of GeoLM in Figures 4 and 5 using TopBottomMirror in Figure 4, and TopBottom180 in Figure 5. In fact, that one change is the only difference between the code for the two layouts. In Figure 6, TopBottomMirror is again used to place the cores, while slightly more work is required to calculate the area of the surrounding cache to maintain the total CMP ratio of 3 times as much cache area as core area. Overall, the code for each of these configurations was shorter than that required for the TRIPS example above.



Figure 5 – The same four alpha cores reoriented to keep the Register Files and ALUS farther apart.



Figure 6 – The same four alpha cores, now with cache acting as a cooling buffer between cores.

In a follow on study [23], larger number of cores were included in different core/cache ratios to investigate the effects of checkerboard like interspersing of cores and cache blocks. The floorplans used in this study did not include actual core models, but rather uniform heat generators of the appropriate size.

The study found that the largest factor effecting chip temperatures in these checkerboard-like configurations was the core/cache ratio. Therefore, the Regular-50 pattern, in which half the area is devoted to core and half to cache, was the hottest. The remaining layouts shown lower the core area usage to 25%. The second largest factor was the location of cores near the edge of the die. Such cores do not benefit from the cooling effects of the surrounding cache on one or two sides. This effect was particularly pronounced in systems, such as laptops, without significant off-chip

heat sinks. The Regular-25 was the hottest in such scenarios due to its many cores on the edges of the chip. Alternate-25 ran noticeably cooler with no large heatsink. Finally, the effect of core orientation was investigated in the Rotated-25 configuration, which had temperature characteristics very similar to Regular-25.

The current floorplanner is capable of easily generating all of the investigated floorplans as show in Figure 7. None of these floorplans required more than 25 lines of code. Rotated-25 was the hardest do to its larger repeat pattern. The names of cache components was suppressed to enhance readability.

				1								I			1	1	
		Care25		G	re27		Care29		Core31		Core25		Core27		Core29		Core31
Cor	e26		Core28			Core30		Core32		Core26		Core28		Core30		Core32	
		Core17		G	re19		Core21		Core23		Core17		Core 19		Core21		Care23
Cor	el\$		Core20			Care22		Core24		Core18		Core20		Core22		Core24	
		Carel		C	rel l		Care13		Core15		Core9		Core11		Core13		Care15
Cor	e10		Core12			Care14		Core16		Core10		Core12		Core14		Core16	
		Corel		0	me3		Core5		Core7		Core1		Core3		Core5		Core7
Co	re2		Core4			Core6		Core8		Core2		Core4		Coreő		Core8	
		Core25		Core2	,		Core29		Core31		Core18		Cose19		Care26		Care27
	Core26	Core25	c	Core2	,	Core34	Core29	Core33	Core31	Core17	Corel8	Core20	Core19	Care25	Care26	Care28	Core27
	Care26	Core25 Core17	c	Core2 xre23 Core1	,	Core34	Core29 Core21	Core33	Core31	Corel 7	Core18	Core20	Core19 Core21	Care25	Care26	Care28	Care27
	Core26	Core25	a	ore28 Core2 core28 Core11 core20	•	Core34	Core29 Core21 core21	Core3	Core31	Core17	Core18	Corr20	Core19	Core25	Care26	Core:3 Core:30	Care27
	Care26	Core25 Core17 Core9	α α	xe28 Core1 xe29 Core1		Core3	Core29 Core21 Core13	Care3	Core31 Core23 Core15	Care17 Care23	Core18 Core24 Core2	Corc20	Core19 Core21 Core3	Core31	Carc26	Core28	Core29
	Care26	Core25 Core17 Core9		xe28 Core2 xe28 Core3 xe20 Core3 xe20 Core3		Corel Corel	Corc29 Corc21 Corc21 Corc21 Corc13	Core33	Core31 Core23 Core35	Care17 Care23 Care2	Core18 Core2	Cont20 Cont20 Cont22 Cont22 Cont22	Core21 Core3	Care25 Care31 Care9	Care26	Core12	Core29 Core11
	Core26	Core25 Core17 Core1 Core1		Core2           xr23           xr24           xr25           xr26           xr27           xr28           xr29           xr29           xr21           xr21           xr21           Xr21           Xr21		Core2	Carc29 Carc21 Carc21 Carc13 Carc13 Carc5	Contil	Core31 Core23 Core55 Core7	Core17 Core23 Core1	Core18 Core2 Core2 Core2	Carc20 Carc2	Core19 Core21 Core3 Core3	Core31 Core31	Carc26 Carc20 Carc10 Carc10 Carc10 Carc10 Carc10	Care28 Care29 Care12	Core27
	Care16	Core25 Core17 Core17 Core4 Core4		xr28 Core2 xr28 Core1 xr20 Core1 xre12 Core1		Core3	Carc29 Carc29 Carc21 Carc21 Carc21 Carc23 Carc23 Carc25 Ca		Core31 Core23 Core25 Core25 Core7	Core17 Core23 Core1	Core18 Core24 Core2 Core2 Core2 Core2	Core20 Core2 Core4	Core19 Core21 Core3 Core5	Core25	Care26 Care27 Care20 Ca	Cere18 Cere10 Cere12	Core29 Core13 Core13

Figure 7 – Checkerboard-like layouts used to study effects of temperature dissipation in various core-cache configurations. Clockwise from upper left, they are Regular-50, Regular-25, Alternate-25, and Rotated-25.

Ľ	Can	±57	e Co	ne58	Co	ne59	2 0	are60	Cas	n61 52	Ca	re62	C 4	re63	Ca	re64	50	Core	7 Const	Core58	9	Cor	e59 Emm	Co	re60	c c	ore61	Co	me62	g	Core63	Co	xe64
NoC	12	57	<sup>2</sup> 12	8	ц	وں	Ľ	2_60	12	61 ¥	12	.02	Ľ	_63	20 L2	_64	NaC	12.5	7	12_58	2	12	59	1.2	60	Ľ	2_61	L	2_62	M	12_63	Ľ	2_64
\$	Can	e49	, Co	ne50	Co	re51	2 0	are52	Con	:53	Ca	ne54	, Co	re55	e Ca	re5ð	2	Core	9 Cross	Core50	2	Cor	ත්l - Cross	Co Bar26	re52	C	ore53 Error	Co	are54		Core55	Co	we56
Noo	12	49	2	<b>3</b> 0	12	J	ž Ľ	222	12	53 <b>2</b>	12	я	- L	55	2 L2	56	NoC	12_4	9	12_50	ž	12	_51	1.2	22	Ľ	2_53	L	2_54	2	12_55	L	2_56
ļ	Con	-11	, co	re-12	_ Co	re13	, G	xe44	Con	M5 9	Co	re16	, Co	æ17	са 8	re48	21	Core	11	Core42	8	Cor	el3 Com	Co	re44 F	0	ore45 Cross	C	are46	3	Core47	C	xe48
8	12	41	2 12	-2	ц	<u>.</u> 0	Ľ	2.44	12	.45 <sup>2</sup>	12	46	- L	un -	2 L2	_48	Nec	12_4	1	12_42	2	12	43	1.2	.44	ž I	2_45	L	2_46	2	L2_47	L	2_48
-	Can	-33	, co	<b>e</b> 34	_ Co	re35	<u>م</u>	are36	Con	37	Co	re38 S		e39	2 Ca	re40	4	Core	3	Core34	_	Cor	e35	Co	re36	, c	ore37 Cross	C	ore38	8	Core 39	Ca	xe40
NoC	12	33	12	9	ц	US -	έ Ľ	256	12	37	12	.8	L 1	<b>"</b>	2 L2	_40	NoC	12,3	3	12_34	ž	12	35	1.2	36	ž I	2_37	L	2_38	2	12_39	L	2_40
5	Con	as 1	, Co	re26	Co	re27		xe28	Con		Co	re30	- Co	re31	, Ca	re32	5	Core	5	Core26	-	Cor	e27	Co	re28	_ c	ore29	Co	ore30	2	Core31	Co	ne32
NeC	12	25	12	26	ц	27	ž L	23 2	12	29 \$	Ľ	<b>30</b>	1	UL	2 L2	32	NoC	12_2	5	12_26	2	12	27	12	28	Ľ	2_29	L	2_30	2	12_31	L	2_32
1	Cas	e17	- Co	elt :	Co	re19	2 0	xe20	Con	-21	Ca	ne22	6	ce23	5 Ca	re24	8	Corel	7 Const	Core18		Cor	e19 Cross	Co	re20	_ c	ore21 Cros	Co	ore22		Core23	Ce	ne24
NeC	12	17	12	18	Ľ	• س	έ L	200	12	21	Ľ	22	Ľ	23	E 12	24	NoC	12_1	7	12_18	2	12	_19	12	20	ž I	2_21	L	2_22	2	12_23	L	2.24
	Ca	-	2 00	ne 10	Co	ee11	, a	are12	Can	-13 Z	Ca	re14	2 00	æ15	e Ca	re16	2	Core	9 Crost	Core10		Cor	ell Cros	Co Bar6	re12	c	ore13 Cro	Co Bar7	mel4		Core15	Co scBar8	we16
No	12	,	<u>د</u> 1	uo 🖡	12	u i	ž L	2.12	12	13	12	и	1 L	US I	2 L2	_16	Not	123	,	L2_10	2	12	'n	1.2	12	Ľ	2_13	Ľ	2_14	××	12_15	L	2_16
ļ	Ca	el	, α	ne2	Ca	are]	, c	iare4	C	ಕ	Ca	reð	0	ae7	e Ca	reŝ		Core	1 Emet	Core2	-	Co	re3 Erros	Co Bar2	xe4	, (	iore5 Cro	C	ore6		Core7	C	ore8
λų Λ	ц	. 1	ž Ľ	2		23	٤ ١	2.4	12	5 ¥	Ľ	6	Ľ	27	Ľ	U	Not	12	1	12_2	Ŷ	12	0	Ľ	2_4	ž ı	25	L	2_6	ž	12_7	L	2_8
																								-									
			_	_																													
	c	0.00	0.0	0.00	0.0	~ "		~ "					0.0	0.0	0.0	<b>.</b>	Π																
	Core49	Core50	Care51	Core52	Core53	Care54	Care55	CareSé	Care57	Care58	Core 59	Core60	Carefi	Care\$2	Core63	Care64		Care49 C	are50	Core51 Core	2 0	æ53	Core54	Core55	Care56	Care57	Care58	Care 59	Core60	Cosed	Core62	Core63	Core64
NeC!	Core49	Care50	Care51	CoreS2 Cross	Core53 Bar7	Care54	Care 55	Core56	Care57	Core58	Core 59	Cores60 Cros	Core61 sBar8	Care52	Core63	Core64	NaCH	Care49 C	crossB	Core51 Core ar13	2 C	ar53 (	Core54 Cross	Core55 Bar14	Care56	Core57	Core58 Cross	Care59 Bar15	Core60	Cored	Coref2	Core63 Bar16	Corró4
No.7	Core49	Core:50	CareSI	Core52 Cross L2_52	Core53 iBar7 1.2_53	Care54	Care55	Core56 2 2 12_56	Care57	Care58	Core:59	Corres0 Cross 1.2_60	Core61 sBar8 12_61	Care52	Core63	Core64	NaCID.	Core49 C	crossB 2_50	Core51 Core #13 12_51 12_5	2 CA	2_53	Core54 Crossi 12_54	CoreSS Bar14 L2_SS	Care56	Core57	Core58 Cross L2_58	Core59 Bar15 12_59	Core60	Correl L2_6	Cosef2 Cross	Coreti3 Bar16 L2_63	Com64
No.7	Core49	Core50	CareS1	Core52 Cross L2_52 Core36	Core53 iBar7 12_53 Core12	Core54	Care55	Cards	Care\$7	Core58	Core 59	Core60 Cross 12_60	Core61 aBar8 L2_61	Care52	Core63	Care64	Necto	Cone49 C	crossB 2_50	Core51 Core #13 12_51 12_5	2 CA	2,53	Core54 Crossi 12_54	Core55 Bart4 L2_55	Core56	Care57	Core58 Cross L2_58	Core59 Bar15	Cxe60	Cored	Cross	Core63 Bar16 L2_63	Core64
No.	Core49 L2_49 Core33	Core50 1.2_50 Core34	CareS1	Core52 Cross L2_52 Core36	Core53 Bar7 12_53 Core37	Core54 1.2_54 Core38	Care55 1.2_55 Care39	CoreS6	Care57 12_57 Care41	Care58 L2_58 Care42	Core59 L2_59 Core43	Core60 Cross 1.2_60 Core44	Core61 nBar8 L2_61 Core45	Care52 1.2_62 Care46	Cored3 L2_63 Core47	Care64 1.2_64 Care48	NoCID	Care49 C	crossB 2_50 cre34	aris 12.51 12.5 Core35 Core	2 CA 2 Z 2 L 46 CA	2_53 2_53	Core54 Crossi L2_54 Core38	Core55 Bar14 L2_55 Core39	Care56	Care57 12_57 Care41	Core58 Cross L2_58 Core42	Care59 Bar15 1.2_59 Care43	Core60	Cored L2_6 Coret	Cross Cross L2_62 i Core46	Core63 Bar16 L2_63 Coref7	Core64 1.2_64 Core48
NoC5 NoC7	Core49 1.2_49 Core33	Core50 12_50 Core34	CareS1	Core52 Cross L2_52 Core36 Cross	Core53 Bar7 1.2_53 Core37 Bar5	Core54	Care55 L2_55 Care39	Care56	Care\$7	Core58 1.2_58 Core42	Core59 L2_59 Core43	Cores0 Cross L2_60 Core44 Cross	Core61 aBar8 L2_61 Core45 aBar6	Care62	Core63 L2_63 Core47	Care64 1.2_64 Care48	NeC9 NeC13	Core49 C	crossB 2_50 crossE crossE	Core51 Core ar13 12_51 12_5 Core35 Core	2 CA 100% 2 L' 36 CA	253 253	Core54 Crossi 12_54 Core38 Crossi	Core55 Bar14 L2_55 Core39 Bar10	Care56	Care57	Core58 Cross L2_58 Core42 Cross	Core59 Bar15 1.2_59 Core43 Bar11	Corres0	Cored L2_61	Core42	Core63 Bar16 L2_63 Coref7 Bar12	Core64
NoC5 NoC7	Core49 1.2_49 Core33 1.2_33	Core:50	Care\$1	Core52 Cross L2_52 Core36 Cross	Core53 iBar7 12_53 Core37 iBar5	Core34 12_54 Core38	Care55 L2_55 Care39	Corr56	Care57	Core58 L2_58 Core42 L2_42	Core59 L2_59 Core43 L2_43	Cores60 Cross 1.2_60 Cores44 Cross 1.2_44	Core61 aBar8 L2_61 Core45 aBar6	Care62 L2_62 Care46	Core63 L2_63 Core47 L2_47	Care64 1.2_64 Care48 L2_48	NoC9 NoC13	Core49 C	CrossB 2,50 CrossB 2,20 CrossB	Core51 Core #13 12_51 12_5 Core35 Core har9	2 CA 100 2 L2 2 L2 46 CA 100 100 100 100 100 100 100 10	2,53 2,33 2,37	Core54 Crossi L2_54 Core38 Crossi L2_38	Core55 Bar14 L2_55 Core39 Bar10 L2_39	Care56	Care57 12_57 Care41 12_41	Core58 Cross L2_58 Core42 Cross L2_42	Core59 Bar15 L2_59 Core43 Bar11 L2_43	Core60	Cored L2_6 L2_6 L2_4	Cross	Core63 Bar16 L2_63 Coref7 Bar12 L2_47	Core64
NACS NACT	Core49 1.2_40 Core33 1.2_33 Core17	Core:50	Care\$1	Core52 Cross L2_52 Core36 Cross L2_35 Core30	Core33 (Bar7) 12_53 Core37 (Bar5) 12_37 Core21	Core34 12_54 Core38 12_38	Core55	CareS6 2 2 12_56 Care40 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Care\$7	Core58 L2_58 Core42 L2_42 Core26	Core59 L2_59 Core43 L2_43 Core27	Cores60 Cross 1.2_60 Cores44 Cross 1.2_44 Cores28	Core61 aBar8 1.2_61 Core45 sBar6 1.2_45 Core29	Care62 1.2_62 Care46	Core63 L2_63 Core47 L2_47 Core31	Care64 1.2_64 Care48 1.2_48 Care32	NoC9 NoC13	Corretion Co L2_40 L Corret33 C L2_33 L Corret34 C	CrossB CrossB 2,50 CrossB CrossB	Core31 Core #13 12_51 12_5 Core35 Core har9	2 C	2,53 22,53 22,57	Core54 Crossl L2_54 Core38 L2_38	Core55 Bar14 L2_55 Core39 Bar10 L2_39	Care56 12_56 Care40 12_49	Care57 12_57 Care41 12_41	Core58 Cross L2_58 Core42 Coss	Care59 Bar15 12_59 Care43 Bar11 12_43	Core60	Cored L2_61 L2_61 L2_41	Cross Cross L2_62 Cross L2_46 Cross	Core63 Bar16 L2_63 Core47 Bar12 L2_47	Corre64
NoC3 NeC7	Core49 1.2_49 Core33 1.2_33 Core17	Core50 1.2_50 Core34 1.2_34 Core18	Care51 12_51 Care35 12_35 Care19	Core52 Cross L2_52 Core36 Core36 L2_36 Core20	Core33 iBar7 1.2_53 Core37 1.2_37 Core21	Core54	Core55 L2_55 Core39 L2_39 Core23	Core56 2 12_56 Core40 2 12_40 Core24	Care57	Core58	Core59 L2_59 Core43 L2_43 Core27	Core60 Cross 12_60 Core44 L2_44 Core28	Core61 12_61 Core45 12_45 Core29	Care62 L2_62 Care46 L2_46 Care30	Core63 L2_63 Core47 L2_47 Core31	Core64 12_64 Core68 12_48 Core62	NeCO NeCO	Correto C 1.2_40 L Corret3 C 1.2_33 L Corret7 C	CrossB 2_50 crc34 ( CrossB 2_34	Core31 Core ar13 12_51 12_5 Core35 Core bar9 12_35 12_5 Core19 Core	2 C 100 2 L 2 L 46 C 100 100 100 100 100 100 100 10	xe3 ( 2_53 xe37 ( 2_37 xe21 (	Core54 Cross1 12_54 Core38 Coss2 Core22	Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core23	Care\$6	Core57 12_57 Core41 12_41 Core25	Core58 Cross L2_58 Core42 Cross L2_42 Core26	Core59 Bar15 12_59 Core43 Bar11 12_43 Core27	Corre60	Correl 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Crost L2_62 Crost L2_62 Crost L2_62 Crost L2_62 Crost L2_65 Crost	Core63 Bar16 L2_63 Core47 Bar12 L2_47 Core31	Corr64 L2_64 Corr68 L2_48 Corr63
NOC3 NOC3 NOC3	Corred9 1.2_49 Corre33 1.2_33 Corre17	Core50 12_50 Core34 12_34 Core18	Care51	Core32 Cross L2_52 Core36 L2_36 Core30 Core30	Core33 Bar7 L2_53 Core37 Bar5 L2_37 Core21 Bar3	Core54	Core55 1.2_55 Core39 1.2_39 Core33	Core56 7 12_56 Core60 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Care57	Core58 12_58 Core42 12_42 Core26	Core59 L2_59 Core43 L2_43 Core27	Core60 Cross 12_60 Core44 Cross 12_44 Core28 Cross	Core61 12_61 Core45 sBar6 L2_45 Core29	Care62 12_62 Care46 12_46 Care30	Coref3 L2_63 Coref7 L2_47 Core31	Care64 12_64 Care65 12_48 Care62	Necs Necs Nect	Core49 C	cresB 2_50 cresB cresB cresB cresB cresB	Core31 Core #13 12_51 12_1 Core35 Core #49 12_35 12_1 Core19 Core #45	2 C 7 C	xe3 ( 2_53 xe37 ( 2_37 xe21 (	Core54 Cross L2_54 Core38 L2_38 Core22 Cross	Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core23 Bar6	Care56	Care57	Core58 Cross L2_58 Core42 L2_42 Core26 Core	Core59 Bar15 L2_59 Core43 Bar11 L2_43 Core27 SBar7	Correl0 12_60 12_44 12_44	Core6	Crose	Core63 Bar16 L2_63 Core47 Bar12 L2_47 Core31 Bar8	Corr64 L2_64 Corr68 L2_48 Core32
NOS NOCI NOCI	Corres9 122_49 Corre33 122_33 Corre17 122_17	Core:50	Care51 12_51 Care35 12_35 Care19 12_19	Core52 Cross Core36 Core36 Core36 Core30 Core30 Core30 Core30	Core53 12_53 Core37 Dar5 (Core37 12_37 Core21 Bar3	Core54	Caess L2_55 Caes99 L2_39 Caes23	Core56	Care57	Core58	Core59 L2_59 Core43 L2_43 Core27	Corredo Cross L2_60 Corred L2_44 Core23 Cross L2_24	Core61 a3ar8 12_61 Core45 a3ar6 L2_45 Core29 a3ar4	Care62	Core63 L2_63 Core67 L2_67 Core31	Care64 12_64 Care68 12_48 Care632 12_32	Nects Nects Nects	Core49 C 12.49 L Core33 C Core33 L Core17 C 12.17 L	cres0 ( CrossB 2_50 ( CrossB 2_34 ( CrossB 2_34 ( CrossB 2_34 ( CrossB	Core31 Core ar13 12_51 12_1 Core35 Core har3 12_35 12_1 Core19 Core har5	2 C 2 C 2 C 2 C 4 C	me3) ( 2,53 me37) ( 2,37 me21) ( 2,21	Core54 Crossl L2_54 Core38 Core38 L2_38 Core22 Cross Core22	Core35 Bar14 L2_55 Core39 Bar10 L2_39 Core23 Bar6 L2_23	Core56	Care57	Core33 Cross L2_58 Core42 Cross L2_42 Core26 Cross	Core59 Bar15 Core43 Bar11 1.2_43 Core27 sBar7 1.2_77	Core60	Cored 12_61 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Cross Cr	Core63 Bar16 Core47 Bar12 L2_47 Core31 L2_11	Corr64 L2_64 Corr68 L2_48 Core32
NGS NGS NG	Correl9 1.2_49 Correl3 1.2_33 Correl7 1.2_17 Correl	Corr50 12_50 Corr34 12_34 Corr18 12_18 Corr2	Care51 L2_51 Care35 L2_35 Care19 L2_19 Care19	Core52 Cross L2_52 Core36 Cross L2_36 Core20 Core20 Core20 Core20 Core52	Core53 38ar7 12_53 Core37 38ar5 Core37 38ar5 Core31 38ar5 12_27 38ar5	Core54 12_54 Core38 12_38 Core22 12_22 Core65	Care55 12_55 Care39 12_39 Care23 12_23 Care7	Correld 2 12_56 Correl0 2 2 2 2 2 2 2 2 2 2 2 2 2	Care\$7 12_57 Care\$1 12_41 Care\$2 12_25 Care\$5 Care\$	Core58 L2_58 Core62 L2_62 Core62 Core25 Core50	Core59 L2_59 Core43 L2_43 Core27 L2_27 Core11	Core60 Cross L2_60 Core44 Cross L2_44 Core23 Cross L2_28 Core12 Cross L2_28 Core12	Core61 12_61 12_61 12_6 aBar6 12_6 Core29 aBar6 12_29 Core13	Care62	Correl3 L2_63 Correl7 L2_77 Correl1 L2_31 Correl5	Care64	NeC5 NeC9 NeC13	Correll C 1.2_40 L 1.2_33 L 1.2_33 L Correll C 1.2_17 L 	crest ( CrossE 2_50 : crest ( CrossE 2_34 : crest ( CrossE 2_18 :	Case 51         Case 51           at13         12_51         12_5           Core 15         Core 15         Core 15           Core 15         L2_55         L2_5           Core 16         Core 16         Core 16           Mar 5         Core 17         L2_55		xe33 ( 2_53 xe37 ( 2_37 xe21 ( 2_21	Core54 Cross1 L2_54 Core38 Core38 L2_38 Core22 Cross L2_32 Cross	Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core23 Bar6 L2_23	Core56	Core57	Core58 Cross L2_58 Core42 L2_42 Core26 Core26 L2_26	Core59 Bar15 Core43 Bar11 12_59 Core43 Bar11 12_43 Core27 Bar7 12_27	Care60	Courd 12_61 12_61 12_4: 12_4: 12_4: 12_4: 12_5 12_5 12_5	Cross Cr	Core31 Ear16 Core47 Ear12 Core47 L2_67 Core31 EBar8 L2_51 L2_51	Correl4 L2_64 Correl48 L2_48 Correl32 L2_32
NOC3 NOC3 NOC3	Correl9 1.2_40 Correl3 1.2_33 Correl7 1.2_17 Correl	Core50 12_50 Core34 12_34 Core18 12_18 Core2	Care31 L2_51 Care35 L2_35 Care19 L2_19 Care3	Core52 Cross L2_52 Core36 Core36 Core30 Core20 Core30 Core30 Core30 Core30 Core30	Core53 aBar7 12_53 Core37 aBar5 Core31 aBar5 Core51 Core5	Core34 12_54 Core38 12_38 Core32 12_22 Core6	Core55 L2_55 Core39 L2_39 Core23 L2_23 Core7	Correls 2 12_56 Correl0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	Care\$7 12_57 Care\$1 12_41 Care\$2 12_55 Care\$9	Core58 L2_58 Core42 L2_42 Core12 L2_26 Core10	Core59 L2_59 Core43 L2_43 Core27 L2_27 Core11	Core60 L2_60 Core44 L2_44 Cros L2_44 Cros L2_28 Cros	Core61 aBar6 12_61 core45 sBar6 12_65 Core39 sBar4 12_29 Core13	Care62 1.2_62 Care46 L2_46 Care30 L2_30 Care14	Correl3 L2_63 Correl7 Correl1 L2_31 Correl5	Care64 1.2_64 1.2_64 1.2_48 1.2_48 1.2_48 1.2_48 Care32 Care36	NeCS NeCP NeCP	Care#9 C 12_40 L Care33 C Care33 L Care33 L Care17 C Care17 C Care1 C	creste ( Cresste 2,50 Cresste 2,34 Cresste Cresste 2,34 Cresste 2,218	Cone51 Core 1813 12,51 12,5 Core15 Core 12,35 12,5 L2,35 12,5 Ker5 Core19 Core Core 10 Core	2 C 2 C	arc3 ( 2,53 arc37 ( 2,37 arc21 ( 2,21 arc5	Core54 Cross L2_54 Core38 L2_38 Core22 Core22 L2_22 Core6	Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core33 Bar6 L2_23 Core7	Corr55	Care57 12_57 Care41 12_41 Care25 12_25 Care9	Core58 Core58 L2_58 Core42 Core42 Core56 L2_42 Core56 Core56 Core56	Core59 12_59 Core43 Bar11 12_43 Core27 12_77 Core11	Core60 12_60 Core44 12_44 12_44 12_28 Core12	Coret 12_61 12_61 12_61 12_61 12_71 12_71 12_72 12_72	Crose 22 Crose 22 Cro	Core63 Bar16 L2_63 Core47 L2_47 Core31 L2_51 L2_51 Core15	Core54 L2_64 Core48 L2_48 Core32 L2_32 Core16
NACI NACI NACI	Corres9 1.2_49 Corre33 1.2_33 Corre17 1.2_17 Corre1	Core50 12_50 Core34 12_34 Core18 12_18 Core2	Core31 12_51 Core35 Core19 L2_19 Core3	Core52 Cross L2_52 Core36 Cross L2_35 Core20 Cross Core20 Cross Cross Core4 Cross	Core53 Bar7 12_53 Core37 Bar5 Core37 Bar5 12_37 Core5 Bar1	Core54 12_54 Core38 L2_38 Core22 L2_22 Core6	Care35 L2_55 Care39 L2_39 Care33 L2_23 Care7	Corress 2 2 2 2 2 2 2 2 2 2 2 2 2	Care57 12_57 Care41 12_41 Care55 12_55 Care5	Care58 1.2_58 Care42 1.2_42 Care26 L2_26 Care10	Core:59 L2_59 Core:43 L2_43 Core:27 L2_27 Core:11	Core60 Cross L2_60 Cross L2_44 Core28 L2_28 Cross Cros	Core61 12_61 Core45 sBar6 12_65 12_65 sBar6 12_75 core39 sBar6 12_29 Core13 sBar2	Care62 1.2_62 Care46 Care46 Care30 L2_30 Care14	Correl3 L2_63 L2_67 Correl7 L2_17 Correl3 L2_31 Correl5	Care64	NeCI NeCI NeCI	Careil C 12_40 L Careil C Careil C Careil C Careil C	CrossB 2,50 CrossB 2,50 CrossB 2,34 CrossB 2,34 CrossB 2,18 CrossB	Coned3 Core #13 12_51 12_5 Coned3 Core Core13 Core Core19 Core Sar5 Core19 Core Sar5 Core		2,53 2,53 2,37 2,37 2,37 2,27 2,21 2,21	Core54 Crossl Core38 Core38 Core38 L2_38 Core22 Core62 Core6 Core6	Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core23 Bar6 L2_23 Core7 Bar2	Care56	Care\$7	Core58 Cross L2_58 Core42 L2_42 Core36 Core36 Core36 Core36 Core36 Core36 Core36 Core36 Core36 Core38	Core59 Bar15 1.2_59 Core43 Bar11 1.2_43 Core17 1.2_77 Core11 Core11 Core11	Core60	Correl 12_61 12_61 12_61 12_61 12_71 12_71 12_72 12_72 12_72	Cross Cross L2_62 Cross Cr	Core63 Bac16 L2_63 Core47 L2_67 L2_67 L2_67 L2_67 L2_67 L2_67 L2_67 L2_61 Core15 Core15 Core15	Corr64 L2_64 Corr68 L2_48 Corr68 L2_32 Corr66
Noci Noci Noci Noci	Correl9 12_49 Correl3 12_33 Correl7 12_17 Correl 12_1 12_1	Core50 12_50 Core34 12_34 Core13 12_18 Core2 12_2	Care51 12_51 Care55 12_35 Care19 12_19 Care1 12_3	Core52 Cress L2_52 Core36 Core36 Core30 Core30 Core4 Core4 Core4 Core4 Core52	Core53 Bar7 12_53 Core37 Bar5 Core21 Bar4 Core5 Bar4 Core5 Bar4	Core54 12_54 Core38 12_38 Core22 Core6 12_6	Core35 L2_55 Core39 L2_39 Core32 L2_23 Core7 L2_23 Core7	Care56 2 2 2 2 2 2 2 2 2 2 2 2 2	Care\$7	Core58 L2_58 L2_62 Core52 L2_76 Core10 L2_10	Core59 L2_59 Core43 L2_43 Core27 L2_27 Core11	Correlo Correlo L2_60 Correlo L2_44 Correlo L2_78 Correlo Correlo Correlo Correlo	Care61 1.2_61 1.2_61 1.2_63 1.2_65 1.2_65 1.2_65 1.2_65 1.2_79 0.00013 0.0001 1.2_13 0.0001 0.000 0.00	Core52 L2_62 Core46 L2_46 Core30 L2_30 Core14	Core63 L2_63 Core67 L2_17 Core31 L2_31 Core15	Care64 12_54 12_54 12_48 12_48 12_48 Care32 12_32 Care16 12_16	Nect Nect Nect Nect	Core49 C 1.2_49 L Core13 C 1.2_13 L Core17 C Core1 C Core1 C L2_1 I L2_1 I	xxe50         (           CrossB         (           2_50         (           xxe14         (           CrossB         (           2_34         (           xxe15         (           CrossB         (           CrossB         (           xxe15         (           xxe15 <td>Carefs Care #13 12,51 12,51 12,5 Carefs Care Carefs Carefs Care Carefs Carefs Care Carefs Carefs Carefs Care Carefs Carefs Carefs Care Carefs Carefs C</td> <td></td> <td>2,53 2,53 2,57 2,77 2,27 2,21 2,21 2,21 2,21</td> <td>Core54 Coost 12_54 Core38 Core38 Core38 Core38 Core38 Core5 Core6 Core6 Core6</td> <td>Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core23 Bar6 L2_23 Core7 Bar2 Core7 Bar2</td> <td>Cont6</td> <th>Core57 12_57 12_57 12_41 12_41 12_11 12_12 Core25 12_12 12_12</th> <td>Core38 Cross L2_58 Core42 L2_42 Core26 Core26 Core26 Core26 Core20 Core20 Core20 Core20 Core22</td> <td>Care 59 Barl 5 12_59 Care 43 Barl 1 12_43 Care 43 12_77 Care 11 12_77 Care 11 12_77 Care 11 12_77</td> <td>Core60</td> <td></td> <td>Conet2           Cross           L2_92           Cross           Cross</td> <td>Cored3 Bar16 L2_63 Core47 L2_67 L2_67 Core31 L2_51 Core15 cRar4</td> <td>Com54 L2_64 Com48 L2_48 Com32 L2_32 Com16 L2_16</td>	Carefs Care #13 12,51 12,51 12,5 Carefs Care Carefs Carefs Care Carefs Carefs Care Carefs Carefs Carefs Care Carefs Carefs Carefs Care Carefs Carefs C		2,53 2,53 2,57 2,77 2,27 2,21 2,21 2,21 2,21	Core54 Coost 12_54 Core38 Core38 Core38 Core38 Core38 Core5 Core6 Core6 Core6	Core55 Bar14 L2_55 Core39 Bar10 L2_39 Core23 Bar6 L2_23 Core7 Bar2 Core7 Bar2	Cont6	Core57 12_57 12_57 12_41 12_41 12_11 12_12 Core25 12_12 12_12	Core38 Cross L2_58 Core42 L2_42 Core26 Core26 Core26 Core26 Core20 Core20 Core20 Core20 Core22	Care 59 Barl 5 12_59 Care 43 Barl 1 12_43 Care 43 12_77 Care 11 12_77 Care 11 12_77 Care 11 12_77	Core60		Conet2           Cross           L2_92           Cross	Cored3 Bar16 L2_63 Core47 L2_67 L2_67 Core31 L2_51 Core15 cRar4	Com54 L2_64 Com48 L2_48 Com32 L2_32 Com16 L2_16

Figure 8 – McPAT configurations of 64 cores in clusters that contains (clockwise from upper left) 1, 2, 4, and 8 cores per cluster. Notice the scaling of the crossbars in the clusters that make this CMP configuration hard to justify in term of area budget above 4 cores per cluster.

For our second case study we consider the CMP design space investigation used to validate the McPAT tool [4]. In this study, they modeled a 2D mesh topology of core/cache clusters containing 64 Niagara-like cores. Each cluster contained a 1-1 ratio of cores and cache banks, connected via a crossbar switch. In addition, each cluster also contained a NoC router that supports communication between clusters. The crossbar was double pumped to reduce the increase in crossbar area from the square of the number of cores per cluster to half that rate. And, the single core cluster configuration needs no crossbar at all. In addition, all configurations supported the same bisection bandwidth of the 2D NoC mesh. Therefore, for non-square grid configurations (for example 2 cores per cluster in a 4x8 2D mesh), the NoCs were scaled to support the needed bandwidth through the smallest of the two grid dimensions. Because of the super-linear scaling of the crossbar area with cluster size, Li et al. conclude that this CMP design

is not justified beyond 4 cores per cluster when including cost in the evaluation metric (EDAP). When cost is not taken into account, the 8 core per cluster arrangement has the best performance and area as measured by EDP.

The McPAT paper does not mention floorplans for these configurations. However, we have tried to match the NoC scaling, crossbar scaling, and relative areas of cores and caches as presented in that work. Figure 8 shows the resultant floorplans for four cluster configurations with 1, 2, 4, and 8 cores per cluster. To show the ease of building parameterized layout configurations on top of the general purpose LMs, we wrote a 50 line subroutine that takes the number of clusters, the number of cores per cluster, and does the component scaling and the floorplan generation for the desired CMP configurations. The cluster layout can be handled by a single GeoLM which is then placed into a grid LM. Admittedly, these layouts assume that the NoC component can take on ARs beyond what may be possible. If the AR of the NoC components is constrained, the resultant floorplans will contain a fair amount of white space to accommodate ill fitting shapes unless the NoC component can be placed between the cores/caches along with the crossbar. In addition, the assumption is made that the core AR is also a bit malleable, an assumption we believe was also made in the original work.

### **5. Future Directions**

Important future directions for this work fall into two categories. First, there are additional features and capabilities that can and should be added to the floorplanner. Second, the floorplanner model presented here must be validated by beneficial use in a CMP design space investigation. Inclusion of a traditional floorplanner as an additional LM in the system helps both of these goals. It acts as an important functionality for the tool suite going forward. It also acts as the current standard against which any benefits provided by the new LMs used in a CMP design study should be gauged. ParquetFP is a full-featured modern floorplanner that has been used in several of the studies mentioned here, and has source available on the web for research use. It is expected that this is the traditional floorplanner that will be added to the system.

Other potential improvements to the current system include the following:

- 1. It is important that the floorplanner be better integrated with other tools in the tool suite. First, the leaf components should be the actual components modeled in other tools in the suite such as MV5 and/or McPAT. Second, floorplans are currently created by writing C++ code to create and connect the LMs in the hierarchy. A textual specification language that can be used as input to the floorplanner would allow the tool to be used without needing to edit the source code.
- 2. The current LMs are not sufficiently robust when their inferiors are not able to lay themselves out in the requested AR. Fix ups on the way back up the recursion stack are important in this hierarchical model. The current implementation of the GeoLM is particularly fragile in this way.
- 3. It would be helpful to have a way to say that the next component to be layed out in the GeoLM should not take all the remaining space along one side of the current layout rectangle. Currently the user must add an additional level of hierarchy to the design to achieve this as was seen in the TRIPS floorplan specification. There are several ways this could be done, all of which involve specifying additional hint information, perhaps about the linkage between two direct inferiors of the GeoLM. Swing contains such an LM called the SpringLayout. Adding this requires that components in the layout have unique identifiers (currently, just the C++ pointers are

used for this) and the layout must be able to handle the case in which the remaining area is not rectangular. Still, the simple EV6 floorplan shown in Figure 2 currently requires 8 different GeoLMs to be specified because of this limitation. Any new feature(s) of this type would help.

- 4. Connectivity (wires) between components is not currently modeled in any of the existing LMs. However, it is unlikely that a very general wiring model will be added to the system as this type of floorplan constraint is already well handled by traditional floorplanners. Instead, it is expected that this will be handled with more specific LMs such as ones that model NoC topologies, or with the idea below.
- 5. Skadron et al. [17] point out the desirability for a floorplanner to be able to create pre-RTL architectural level layouts for cores using information about the pipeline flow between the processor elements. They suggest the use of "adjacency matrix" floorplan specifications. Such an LM is likely to have wider applicability. The features proposed in 3 above can help to provide the underpinnings for such an LM, and the inclusion of such an LM in the floorplanner is a way to avoid the need to specificy wires in certain scenarios.
- 6. Non rectangular components are not handled by any of the current LMs.

More important than any of the above suggested improvements is the use of the floorplanner in a novel CMP design study. There are several possibilities for such a study. One possibility is the ongoing UVA study to investigate the requirements for power distribution across a CMP chip based on the power needs of the various components and the location of those components in the specific CMP topology [unpublished]. Another possibility is to more completely investigate the CMP design trade-offs suggested by Humenay et al. [18]. They point out that within-die systematic process variation can lead to particularly undesirable effects in CMP design when it causes different cores on the chip to have different performance characteristics. Such within-die systematic variation can be minimized by placing cores near each other in the CMP floorplan. However, the close proximity of the cores can cause other undesirable effects such as higher core temperature, which in turn can cause dynamic frequency and/or voltage throttling, thereby reducing performance. While the authors propose an analytical model for a metric to apply in such situations, which was later greatly refined [19], they investigated a limited number of actual floorplan configurations.

### 6. Conclusion

The work presented here has made several contributions.

- An argument has been made in favor of a new approach to pre-RTL floorplanning at the architectural and CMP level of abstraction. Numerous examples of CMP level design investigations have been cited that could have benefitted from such a floorplanner. This high level early stage layout capability should be included in any comprehensive CMP design tool suite.
- A novel hierarchical architectural framework, repurposed from GUI design, has been suggested for floorplanning that makes it possible to add new floorplan algorithms (LMs) in a consistent fashion. This in turn allows for the inclusion in a single system of many different layout mangers, including current traditional floorplan algorithms, fairly general purpose but high level LMs targeted at CMP layouts, and specific knowledge-based LMs for particular NoC topologies, core-cache clusters, or other important CMP constructs.

To the best of our knowledge, no one has previously proposed this architectural model for hardware floorplanners.

- An initial set of four LMs have been provided that are simple, intuitive, easy to use, yet powerful when used in combination, for the creation of CMP floorplans.
- The existing capabilities of the floorplanner were demonstrated in two case studies of previous CMP design space investigations.
- ArchFP can be downloaded from http://lava.cs.virginia.edu/archfp. It is made available under a BSD-type open-source license.

# **Bibliography**

- 1. Sabih H. Gerez, Algorithms for VLSI Design Automation, New York: John Wiley and Sons, 1999.
- 2. M. Sarrafzadeh, C. K. Wong, *An Introduction to VLSI Physical Design*, McGraw Hill Series in Computer Science, New York: McGraw Hill, 1996.
- 3. Karithik Sankaranarayanan, Sivakumar Velusamy, Mircea Stan, Kevin Skadron, "A Case for Thermal-Aware Floorplanning at the Microarchitectural Level", *The Journal of Instruction-Level Parallelism*, vol. 7, Oct. 2005
- 4. Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, Norman P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", *Micro '09*, New York, NY, December 12-16, 2009
- Rakesh Kumar, Victor Zyuban, Dean M. Tullsen, "Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling", *Proceedings of the 32<sup>nd</sup> International* Symposium on Computer Architecture (ISCA '05)
- 6. David Geary, *Graphic Java 2, Volume 2, Swing (3<sup>rd</sup> Edition)*, The Sun Microsystems Press Java Series, Prentice Hall, March 22, 1999
- 7. Chris Anderson, *Essential Windows Presentation Foundation (WPF)*, Addison-Wesley Professional, April 27, 2007
- Partha Pratim Pande, Cristian Grecu, Micahel Jones, Andre Ivanov, and Resve Saleh, "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures", *IEEE Transactions on Computers*, VOL. 54, NO. 8, August 2005
- 9. Saurabh N. Adya, and Igor L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design",
- 10. Yingmin Li, Benjamin Lee, David Brooks, Zhigang Hu, and Kevin Skadron, "CMP Design Space Exploration Subject to Physical Constraints", IEEE 2006
- 11. Luca Benini, "Application Specific NoC Design", in the Proceedings of the 2009 Conference on Design, Automation, and Test in Europe, DATE'09, April 2009.
- 12. Srivivasan Murali, Ciprian Seiculescu, Luca Benini, and Giovanni De Micheli, "Synthesis of Networks on Chips for 3D System on Chips", IEEE 2009
- 13. Brett H. Meyer, Adam S. Hartman, and Donald E. Thomas, "Cost-effective Slack Allocation for Lifetime Improvement in NoC-based MPSoCs," in the Proceedings of the 2010 Conference on Design, Automation, and Test in Europe, DATE'10, March 2010.
- 14. Brett H. Meyer, Adam S. Hartman, and Donald E. Thomas, "Slack Allocation for Yield Improvement in NoC-based MPSoCs," in the Proceedings of the 11th annual International Symposium on Quality Electronic Design, ISQED'10, March 2010.
- 15. Jiayuan Meng and Kevin Skadron, "Avoiding Cache Thrashing due to Private Data Placement in Lastlevel Cache For Manycore Scaling" in Proceedings of ICCD 2009.
- 16. Mark Gebhart, Bertrand A. Maher, Katherine E. Coons, Jeff Diamond, Paul Gratz, Mario Marino, Nitya Ranganathan, Behnam Robatmili, Aaron Smith, Janes Burrill, Stephen W. Keckler, Doug Burger, and Kathryn S. McKinley, "An Evaluation of the TRIPS Computer System", *Proceedings of the Fourteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2009.
- Kevin Skadron, Mircea Stan, Marco Barcella, Amar Dwarka, Wie Huang, Ungmin Li, Yong Ma, Amit Naidu, Dharmesh Parikh, Paolo Re, Garrett Rose, Karhik Sankaranarayanan, Ram Suryanaranay, Sivakumar Velusamy, Hao Zhang, and Yan Zhang, "HotSpot: Techniques for Modeling Thermal

Effects at the Processor-Architecture Level", Proceedings of the 2002 International Workshop on Thermal Investigations of ICs and Systems (THERMINIC), pp. 169-172, October 2002.

- Eric Humenay, David Tarjan, and Kevin Skadron, "Impact of Process Variations on Multicore Performance Symmetry", Proceedings of the ACM/IEEE/EDAA/EDAC 2007 Conference on Design, Automation and Test in Europe (DATE), pp. 1653-1658, Apr. 2007.
- Smruti R. Sarangi, Brian Greskamp, Radu Teodorescu, Jun Nakano, Abhishek Tiwari, and Josep Torrella, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects", *IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING*, VOL. 21, NO. 1, February 2008.
- 20. Tung-Chieh Chen, and Yao-Wen Chang, "Modern Floorplanning Based on Fast Simulated Annealing", *Proceedings of the 2005 International Symposium on Physical Design*, pp. 104-112, 2005.
- Tung-Chieh Chen, Yao-Wen Chang, and Shyh-Chang Lin, "A new Multilevel Framework for Large-Scale Interconnect-Driven Floorplanning", Computer-Aided Design of Integrated Circuits and Systems, Vol. 27, Issue 2, pp. 286-294, Feb. 2008.
- Karithik Sankaranarayanan, Mircea R. Stan, and Kevin Skadron, "Microarchitectural Floorplanning for Thermal Management: A Technical Report", Tech Report CS-2005-08, Univ. of Virginia Dept. of Computer Science, May 2005.
- 23. Karithik Sankaranarayanan, Brett H. Meyer, Wei Huang, Robert Ribondo, Hossein Haj-Hariri, Mircea Stan, and Kevin Skadron, "Architectural Implications of Spatial Thermal Filtering", In submission.