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[1] 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.

[1] 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 multi-processors 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.

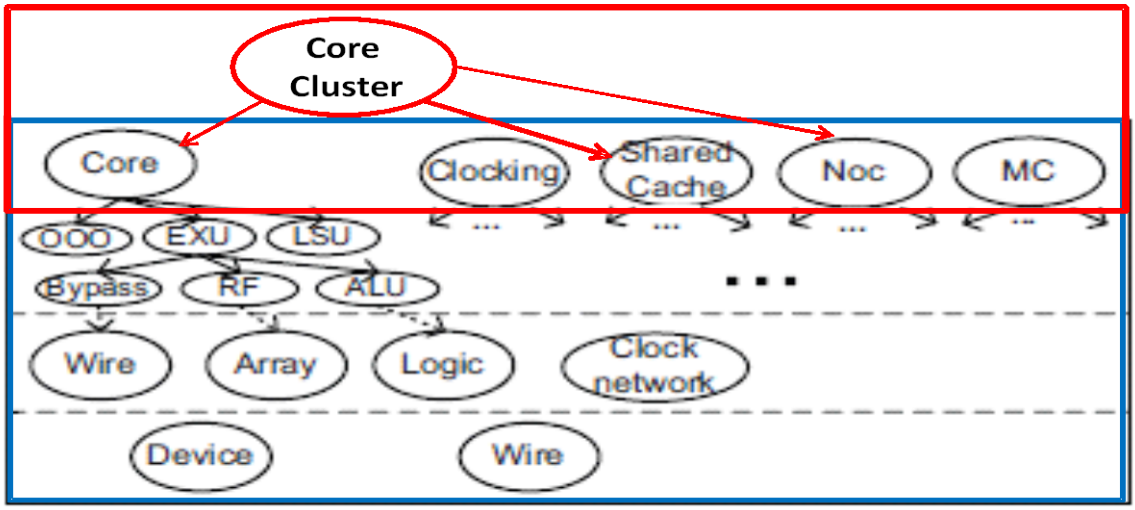

Figure 1 -- Extended McPAT Model Hierarchy. The blue rectangle encloses the domain for a traditional floorplan tool, while the red rectangle encloses the domain for the layout algorithms presented and proposed here. Note that there is an area of overlap between the two.

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.

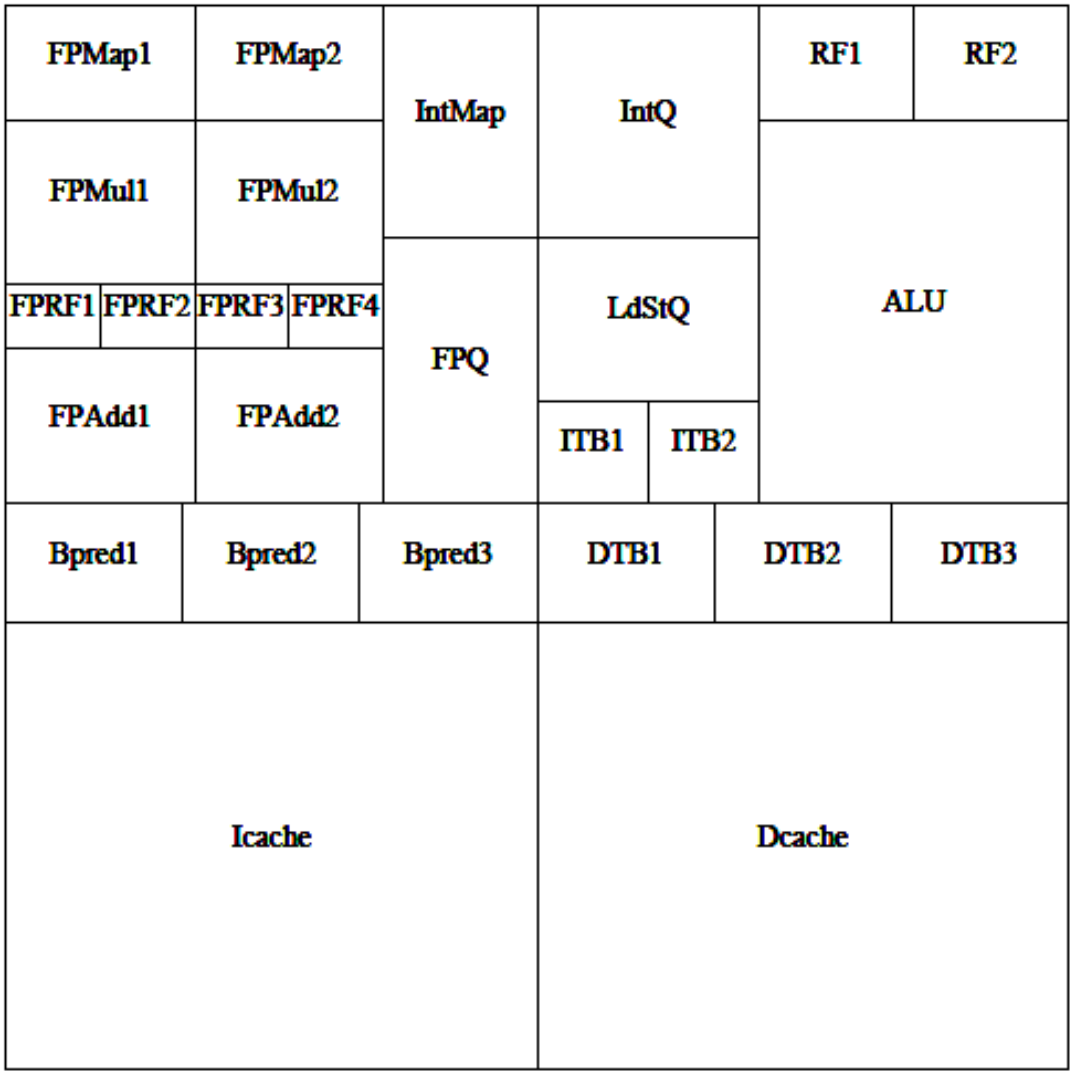

Figure 2 – Example Floorplan for the EV6 version of the Alpha chip at what the McPAT hierarchy would call the Architectural level of detail. Floorplans for more detailed levels in the hierarchy are significantly messier with a much wider disparity in component sizes, many very small pieces of a few transistors 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 3rd 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 3rd 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 3rd 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");
```

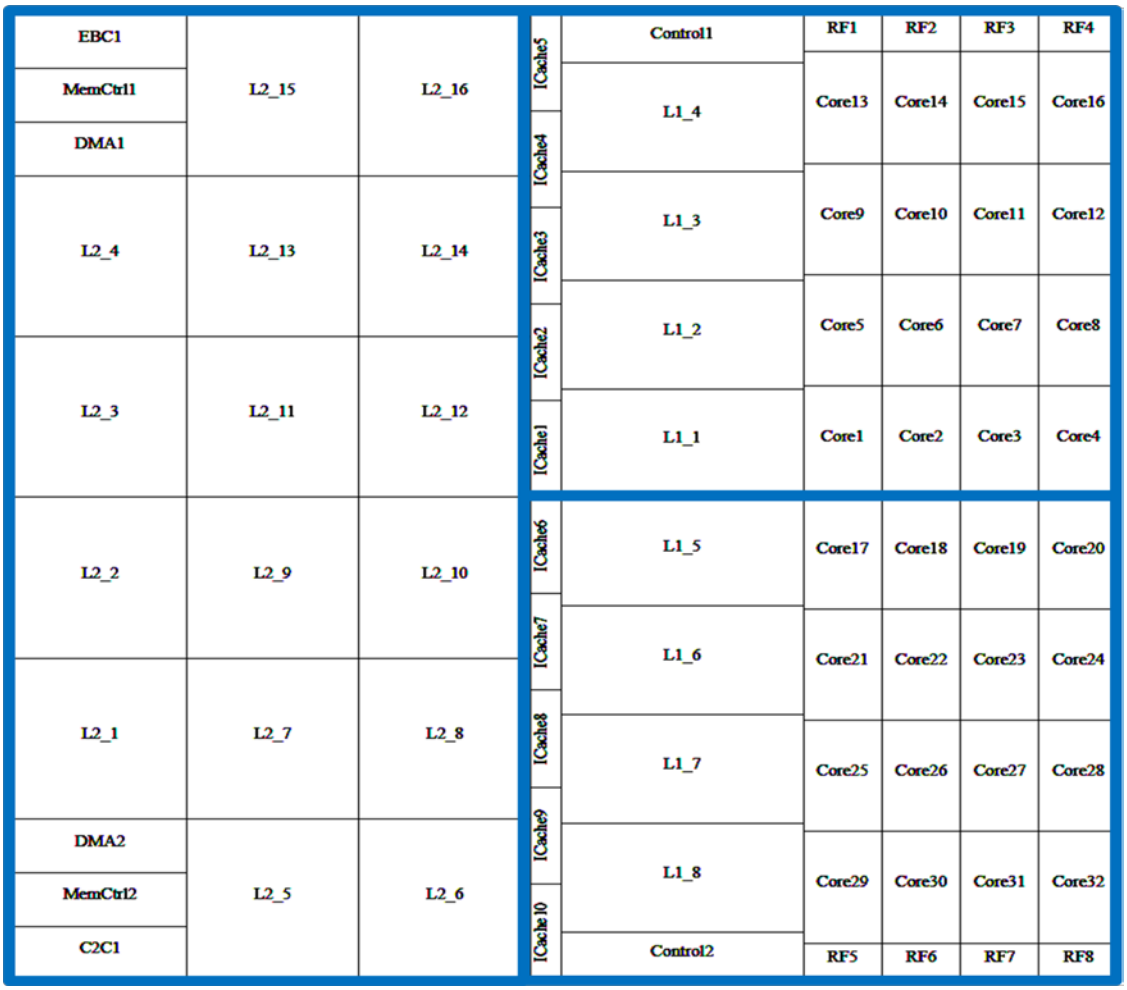

Figure 3 – TRIPS CMP level floorplan. The blue overlays (added manually for emphasis) show the various portions of the layout put together by the use of hierarchical containment of multiple Layout Managers.

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.

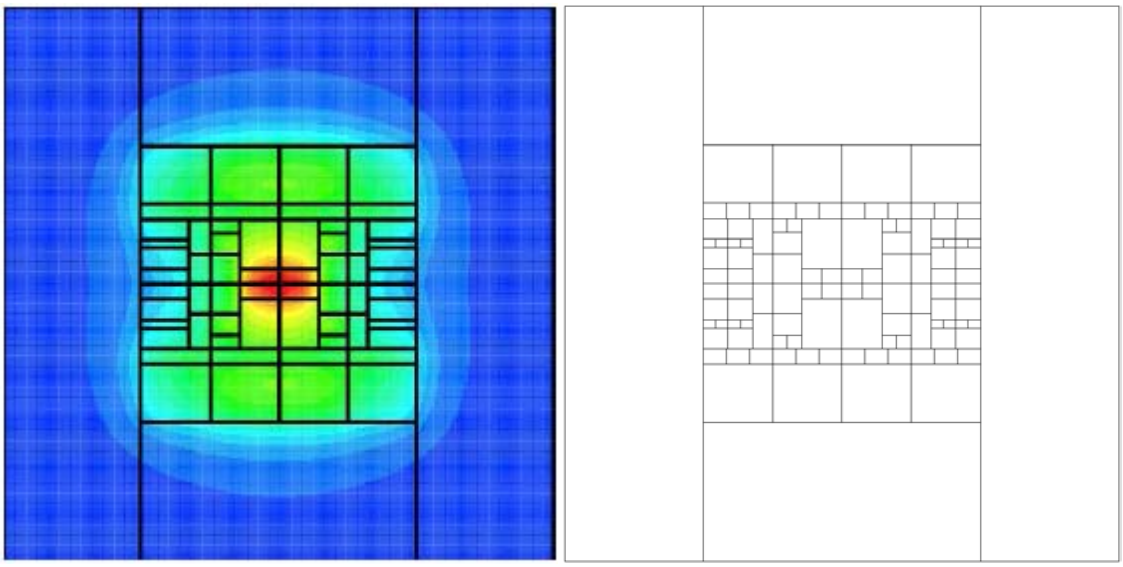

Figure 4 – Four Alpha cores surrounded by cache oriented with their Register Files and ALUs in close proximity. The red color in the center of the picture on the left shows the resultant high temperature.

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.

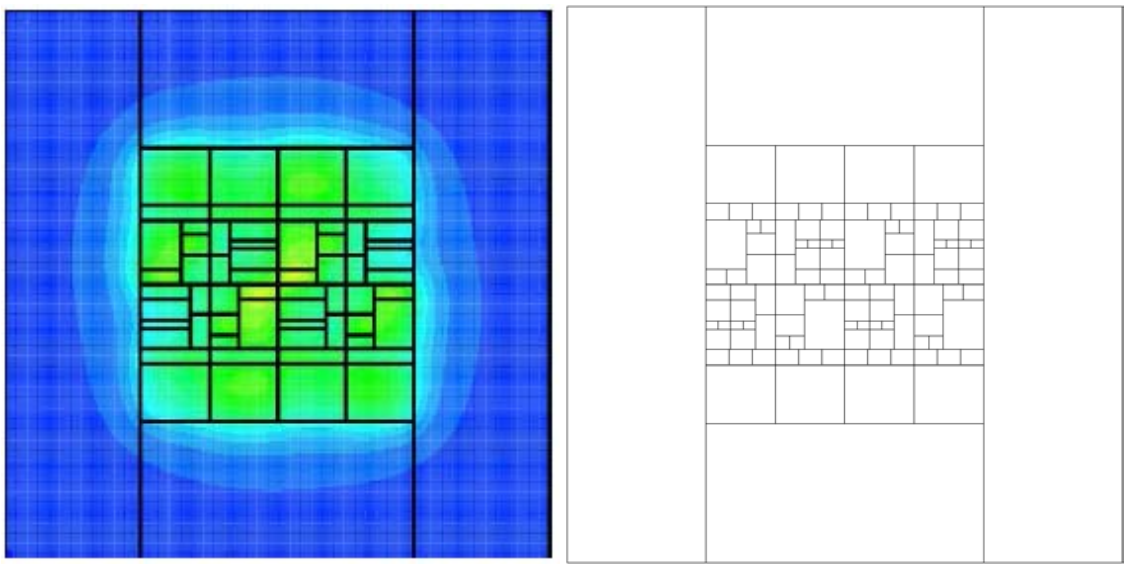

Figure 5 – The same four alpha cores reoriented to keep the Register Files and ALUS farther apart.

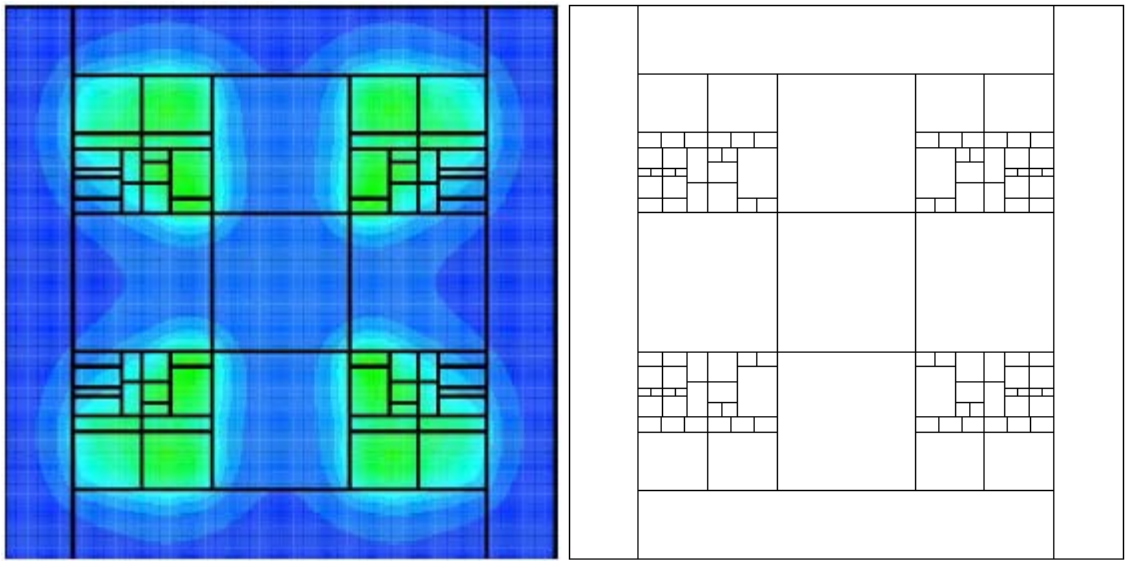

Figure 6 – The same four alpha cores, now with cache acting as a cooling buffer between cores.

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.

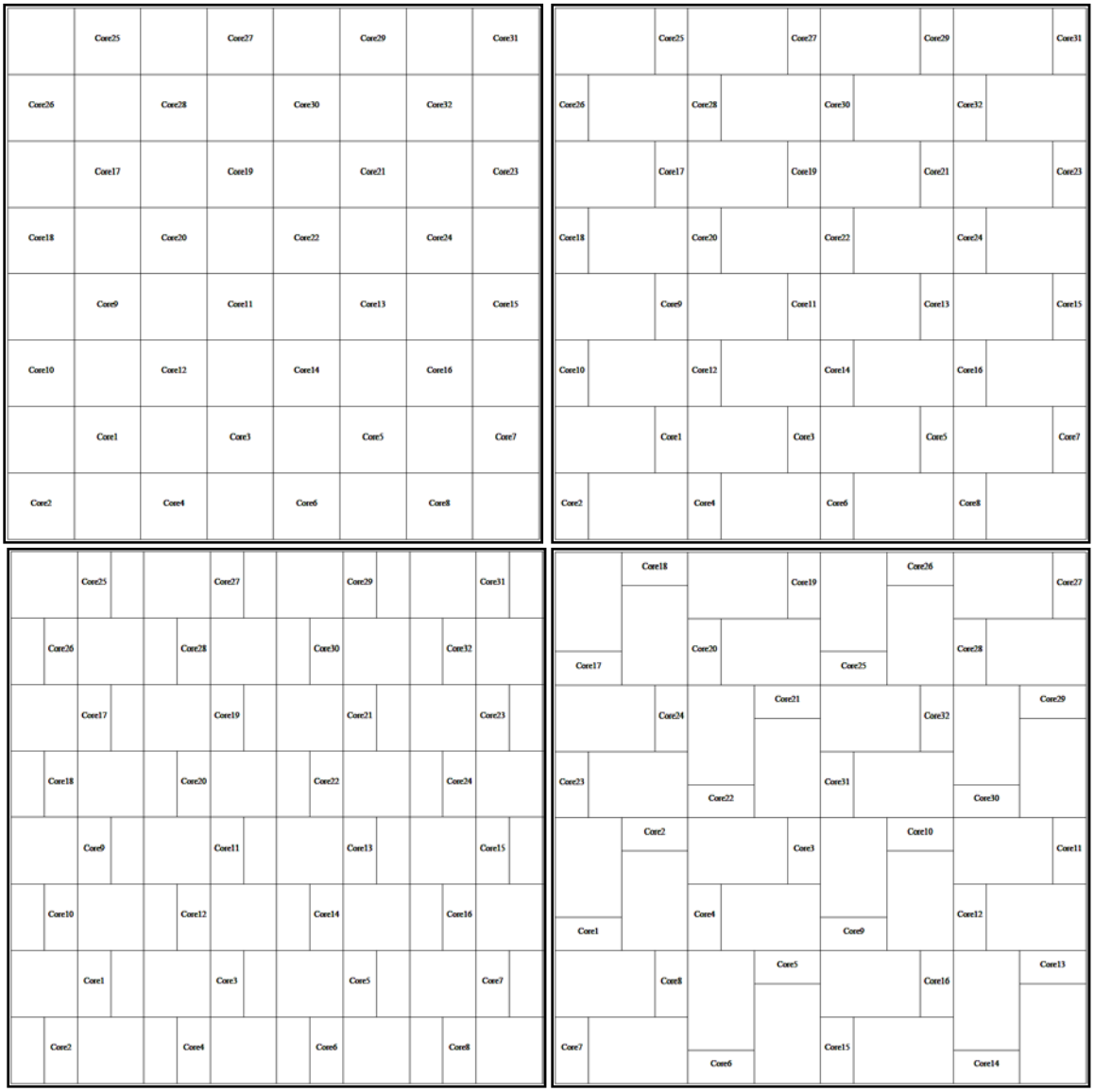

Figure 7 – Checkerboard-like layouts used to study effects of temperature dissipation in various core-cache configurations. Clockwise from upper left, they are Regular-50, Regular-25, Alternate-25, and Rotated-25.

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 lay-out 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
5. Rakesh Kumar, Victor Zyuban, Dean M. Tullsen, "Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling", *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA '05)*
6. David Geary, *Graphic Java 2, Volume 2, Swing (3rd 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
8. 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 Last-level 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.
17. 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.
18. 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.
19. 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.
21. 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.
22. 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.