ArchFP: Rapid Prototyping of pre-RTL Floorplans

Gregory G. Faust*, Runjie Zhang*, Kevin Skadron*, Mircea R. Stan† and Brett H. Meyer‡,
*Department of Computer Science, University of Virginia, Charlottesville, VA, USA
Email: {gf4ea, rz3vg, skadron}@cs.virginia.edu
† Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA, USA
Email: mircea@virginia.edu
‡Department of Electrical and Computer Engineering, McGill University, Montréal, QC, Canada
Email: brett.meyer@mcgill.ca

***Abstract*—There has been a fundamental shift from ever more complex single cores to single chip multi-core (CMP) designs. Along with this opportunity come major challenges; notably the sheer size of the CMP design space. An integrated suite of tools is needed that provides life-cycle support from early prototyping to final design. Here we present ArchFP, a floorplanning tool targeted towards prototyping of pre-RTL CMP design concepts. As such, it is complementary to traditional floorplanners that are more appropriate later in the design cycle.**

An ArchFP floorplan is specified using a model similar to that supported by GUI toolkits such as Java Swing and Windows Presentation Foundation. The floorplan design is comprised of a hierarchy of components placed within containers, called Layout Managers (LMs), that provide a variety of layout algorithms. Current LMs include a generalized grid LM, one that supports geographic hints for component placement, and a fixed layout of imported subcomponents. ArchFP is easy to extend with additional LMs that leverage these initial layout algorithms to support more specific design elements such as NoC configurations, cache partitioning strategies, SIMD design, etc. To the best of our knowledge, no one has previous used this approach for floorplanning. ArchFP is written in C++ for UNIX systems, and is free for download from http://lava.cs.virginia.edu/archfp.

We demonstrate the utility of ArchFP in the study of power delivery and temperature constraints in likely CMP configurations of Penryn-like cores over four technology scales.

I. Introduction

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. 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, and total die size. Each of these are worthy of study in their own right, but making system-wide architectural decisions while attending to one or two of these constraints at a time can often lead to suboptimal designs [1].

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 CMPs and their on-chip support systems. To allow rapid early stage investigation, the suite should contain tools capable of modeling systems at different levels of detail. An example of such a tool is McPAT [2] which contains hierarchical power, area, and timing models for various HW components. 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, at the University of Virginia, the effect of CMP layout on peak chip temperature was investigated for a variety of CMP configurations and target uses [3], [4]. It was shown that the lack of temperature-aware floorplanning can force runtime throttling of the voltage and/or clock speed thereby affecting performance. Also, Kumar et al. [5] studied area, power, and performance of various NoC topologies for CMPs, and found that a hierarchical bus NoC can result in reduced area and power consumption.

These studies and others like them motivate the creation of a pre-RTL floorplanner targeted at CMP design. While traditional floorplanners operate well at the three levels of description supported by McPAT (*Architectural*, *Circuit*, and *Technology*), ArchFP works well at the *Architectural* and a higher *CMP* level of design in which clusters of *Architectural* level components can be combined into reusable/repeatable elements. Traditional floorplanning tools typically operate with detailed post synthesis information about the components and wires which comprise the design. However, this level of detail is not appropriate for early pre-RTL CMP design space exploration. In addition, traditional floorplanners do not attend to the metrics of concern to layout at the CMP level which are different than those later in the design cycle.

ArchFP retargets the model of Graphical User Interface (GUI) design toolkits such as Java Swing [6] and Windows Presentation Foundation [7] to provide a novel framework for floorplanning. 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 floorplanning 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 LMs for hardware floorplanning. Previously, in a Technical Report, we retrospectively showed the utility of ArchFP for CMP design investigations like those above [8]. Below we present a more recent example involving the study of Power Delivery Networks (PDNs).

The remainder of this document is organized as follows. Section II discusses related work in traditional floorplanning algorithms and GUI frameworks. Section III provides details of the layout algorithms currently implemented. In section IV we describe an investigation of Power Distribution Networks using ArchFP. Section V is the conclusion.

II. Background and Related Work

A. Floorplanning

Traditional floorplanners input a set of components and the wires between them, and search for a non-overlapping 2D layout that minimizes the value of an objective function of total area and total wire length. Because optimal floorplan design is NP-Hard, practical systems use various approximation techniques [9], [10]. For example, one technique to is to represent the current floorplan as a (binary) tree of rectangular areas, with HW components as the leaves, and larger composite rectangles further up the tree. The space of possible layouts is searched via simulated annealing which randomly perturbs the tree by a series of "moves" such as 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 new floorplan is better or worse than the previous one. One advantage of this approach is that all components are handled in a simple and consistent fashion. But this approach can be slow, and produce in any given run, a floorplan of uncertain quality.

The use of a tree structure in these algorithms does not imply a hierarchical specification of the HW design, and the hierarchy has neither stability nor semantic meaning. In contrast, in the ArchFP model the hierarchical inclusion of components in containers is stable and has direct impact on the resultant design. More importantly, different layout algorithms can be, and typically are, applied at different levels in the hierarchy. Furthermore, the semantic relevance of component placement in LMs is increased when domain-specific LMs use specialized knowledge about how to lay out their children,

The objective functions of traditional floorplanners are too inflexible for CMP level design. Therefore, many CMP studies have wrapped an extra evaluation function around the results of a traditional floorplanner. For example, Meyer et al. [11] studied system reliability and cost in application-specific SoCs using various NoC configurations. It was necessary to generate a very large number of floorplans to find the few that scored well on the desired metrics. Meyer found that over 90% of the runtime used in design space exploration was spent in the floorplanning component.

We believe ArchFP has several advantages over traditional floorplanners for these CMP studies. First, the current set of LMs are quite general purpose and easy to use, and as they do not examine a large search space, have trivial runtimes. Second, as discussed below, the requirements on an LM are minimal, making the system very easy to extend. Together these two features allow ArchFP to support the direct creation of layouts that have desirable features for the CMP design space under investigation. The investigator can either build domain-specific knowledge into a novel LM, or combine existing LMs using their intuition about layouts likely to have needed attributes. This can result in faster and more focused investigation of large CMP design spaces. Finally, the ArchFP system design is flexible enough to allow a traditional floorplanner to be directly included as an LM, thereby ensuring floorplanning capability no worse than the current standard.

Floorplanners are often evaluated using benchmarks such as GSRC [12]. These metrics relate to post-RTL floorplanning, thus we have not tested ArchFP against such benchmarks.

B. GUI Design Toolkits

Modern GUI toolkits such as Java Swing [6] and Windows Presentation Foundation (WPF) [7] are 3rd generation systems built upon the lessons learned in previous systems. Java Swing and WPF both have the same architecture of components, containers, and LMs that we propose to use for floorplanning. Both toolkits contain many different LMs supporting horizontal, vertical, grid, box-and-spring and other placement algorithms. In both systems, the contract for an 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 architecture in a consistent fashion. ArchFP is written in C++ and offers the same level of extensibility through the use of an abstract base-class with virtual methods for component addition, layout, generating output to a floorplan file, etc. In WPF, layout is done using the same flow of information between LMs as in ArchFP as we discuss in the next section.

III. ArchFP Implementation

A key goal of the floorplanner is to integrate with the growing suite of tools used for CMP design space investigation. The tool suite already contains tools such as gem5 [13], McPAT [2], HotSpot [14], and ParquetFP [15]. The area information for components needed by ArchFP to produce floorplans comes from McPAT. ArchFP can both consume and produce HotSpot floorplan files. This format was chosen to integrate with HotSpot, but also because it is extremely simple; each element specifies it name, location, width and height.

A. ArchFP Components

All components in the system are derived from a C++ base class, which is very similar to a component in one of the GUI frameworks. While LMs act as containers, they are also components. This allows the arbitrary containment

hierarchy to be consistent. Components contain the following information:

- *Component type*. In the current general purpose LMs, the type is ignored during layout. However it is used to provide information for output. Future LMs that are specific to various CMP structures will take the type of the components into account when doing layout.
- *MinimumAR* and *MaximumAR*. For fixed-shape components, these have the same value.
- *Location*. Components store their (x, y) location information relative to their container, not the overall floorplan.
- *Area*. Basic components have an intrinsic area. Containers discover their area as part of the layout process.
- *Width* and *Height*. The actual width and height of the component is often not known until after layout because the recursive specification of AR information flows from top to bottom during the layout process. After layout, all components have a known width and height.

B. ArchFP Layout Managers

All ArchFP LMs are derived from a C++ abstract container class, which contains its list of inferiors, and the attributes it inherits from the component class. 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; one to perform the layout, and one to output the layout in HotSpot floorplan format.

Information flow during layout is as follows.

1) The layout method of the top level LM is given a target AR as a goal. Specific LMs use their target AR and the number and size of their inferiors to dictate the target ARs to those inferiors.
2) Recursively, each LM, starting at the top of the hierarchy, requests the area of their inferiors. As noted above, leaf components are given an area when they are created, halting the recursive descent. LMs then calculate their own area as the sum of their inferiors' areas.
3) Next the LM uses its target AR to determine the required AR of each inferior, then calls the layout method on the inferior specifying that target AR. Therefore, during this phase, components discover their width and height from the bottom up. Leaf components are also able to lay themselves out by checking their minimum and maximum AR and complying as closely as possible with the AR request from above.
4) Next, 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.
5) As each LM finishes the layout for a given inferior, it then sets that inferior's relative location.
6) Finally, the LM calculates its own width and height based on the actual location and size of its inferiors.

C. Initial set of LMs

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:

- *GridLM*. A GridLM contains a single inferior (of arbitrary nested complexity), and a count of grid elements. The actual dimensions of the grid are not specified. Instead, during layout, the GridLM will determine the best dimensions based on its requested AR. For example, if the grid layout method is called with a target AR of 2 (twice as wide as high), and the grid contains 8 elements, the GridLM will set its grid dimensions to 2 rows by 4 columns. This allows layout of the inferior with a target AR closest to a square.
- *GeoLM*. This is the most flexible LM. To reduce the number of levels of hierarchy that the user of ArchFP must specify, inferiors added to the GeoLM can take a repeat count. The GeoLM will then automatically create a GridLM as an inferior to contain the repeating group. In addition, inferiors to the GeoLM are specified with a Geographic Location Hint (GLH) 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 border(s) to the current component. LeftRight and TopBottom are different from the others in that they expect to be applied to a repeating group of even size. The group is cut in two and each half put in the specified location. In addition, the LeftRight and TopBottom hints have a Mirroring option and a 180-degree-rotation option.
- *FixedLM*. This powerful LM allows any pre-existing layout in a HotSpot floorplan file to be reused, with automatic rescaling, in any ArchFP floorplan. It does no layout; its inferiors know their size, shape, and location.
- *BagLM*. A BagLM contains an arbitrary collection of components. It lays them out from largest to smallest placing each in turn along the shortest remaining side.

As shown below, these simple, intuitive, easy to use LMs can be used in combination to produce surprisingly complex floorplans. In addition, such a set of primitives acts as the foundation upon which to build more sophisticated LMs, especially ones specific to important CMP patterns such as NoC topologies, cache sharing models, SIMD layouts, 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.

IV. Case Study

As power density rises with each new generation of chip technology, temperature and power delivery will soon become design constraints on further CMP scaling. In this paper we focus on reliability issues associated with electromigration (EM) in C4 pads and IR drop resulting in increased latencies. The study of these physical constraints requires an integrated tool suite. We use a combination of new and pre-existing tools;

McPAT for power and area information, HotSpot to calculate local temperatures, a novel PDN model specifically designed for this experiment to calculate local IR drop and C4 pad currents [16], and ArchFP for layout information.

A. Floorplan Specification

Using a 45nm Intel Penryn-like core as a baseline (described in IV-C), we create a series of scaled CMPs down to 16nm and study the resulting power delivery noise and chip temperature. We model 2, 4, 8, and 16 core configurations. Figure 1 shows the C++ specification for the 8-core floorplan shown in Figure 2. To create a hierarchical description of a floorplan, we must first create the components near the leaves of the hierarchy so that we can later include them in the higher levels.

The specification for this floorplan works as follows.

1) First, the layout of the Penryn-like core is loaded as a "macro" from a HotSpot-style floorplan file which was also created in ArchFP from 45nm scale *Architectural* level components. Due to the subcomponent diversity, this required a few dozen lines of code. But once created, it can be reused in any ArchFP layout. In this case, it is rescaled to 22nm when loaded for the 8-core configuration. In Figure 2 the core forms the unlabled complex areas near the top and bottom of the CMP (with subcomponent names elided for clarity).
2) Second, the cluster of L2 cache and NoC is created.
3) Third, the Memory Controllers, core, and L2 clusters are each added in implicit grids of 8 each to the CMP level of the LM hierarchy. Note the use of the TopBottom GLH and mirroring of the core component that lead to the symmetric structure of the final floorplan.
4) Fourth, the layout is performed and the result is output.

```
// 1. Load and scale the pre-existing Penryn core.
fixedLayout * core;
core = new fixedLayout("Penryn45.flp", 22./45.);
// 2. Create the L2/NoC cluster.
geogLayout * L2 = new geogLayout();
L2->addCluster(L2, 1, L2area, 20., 1., Right);
L2->addCluster(NoC, 1, NoCarea, 50., 1., Left);
// 3. Now add all the remaining components.
geogLayout * CMP = new geogLayout();
CMP->addCluster(MC, 8, MCarea, 20., 1., TopBottom);
CMP->addComponent(core, 8, TopBottomMirror);
CMP->addComponent(L2, 8, Center);
// 4. Perform the layout and output the result.
CMP->layout(AspectRatio, 1);
CMP->outputHotSpotLayout("CMP8Core.flp");
```

Fig. 1. ArchFP C++ code to build the layout shown in Figure 2. The arguments to the addCluster method are component, repeat count, area, AR limits, and a GLH. The arguments to addComponent are a predefined cluster (thereby adding hierarchy to the layout), a repeat count, and a GLH. The layout method takes a target AR, in this case 1 (square).

B. Floorplan Layout

ArchFP calculates the layout for the above floorplan specification as follows.

1) First all components calculate their total area as described in section III-B.

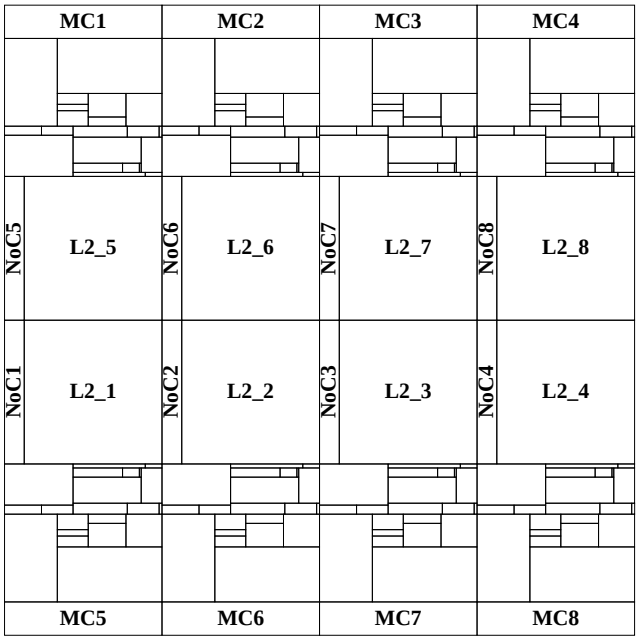

Fig. 2. CMP floorplan of 8 Penryn-like cores with L2 cache, NoCs and Memory Controllers added.

2) The CMP GeoLM is the top component in the hierarchy, and we request it lay itself out in a square. It takes its children *in order of specification* and lays them out in the requested locations. It starts with the 8 MCs, and due to the TopBottom GLH divides them into two implicitly generated 4-element GridLMs. Each GridLM is allocated the entire width of the CMP square for layout at their respective top and bottom locations. Given the size and AR constraints, the GridLMs choose 1x4 grids, and determine the resultant width of each MC. The MC then lays itself out to fit its width.
3) Next the CMP GeoLM lays out the core components. Again, two 4-element GridLMs are formed, and each is allocated the entire width of the CMP square as above, but their y-locations are now inboard of the MCs. Due to the TopBottomMirror GLH, the bottom GridLM is told to reflect its component through the x-axis. The fixed core component need not lay itself out because it uses the information read from the Penryn45.flp file.
4) Next the CMP GeoLM lays out the L2 cluster. There is one 8-element GridLM created. Given the shape of the Center space to fill, it chooses a 2x4 grid configuration.
5) Finally, the L2 cluster lays itself out. As a GeoLM, it lays out its components in order. First the L2 is allocated the full height of the right side of its grid location to lay itself out, followed similarly by the NoC on the left.

C. Scaling and Power Modeling

Our scaling and power assumptions are as follows.

1) Multicore scaling: We chose an Intel 45nm Penryn-like out-of-order processor [17] as our baseline design. It has two 32-bit 4-way out-of-order cores and each core contains a 32KB

L1 instruction cache and a 32KB L1 data cache. The core runs at 3.7GHz. Unified L2 caches are private to each core and are each 3MB. For each technology node, we hold the processor architecture constant but assume that the number of cores (and therefore the number of L2s) doubles. We also assume that L2 cache is always private. We use mesh-based network-on-chip (NoC) structure across all technology nodes.

2) Power modeling and CMP floorplanning: To get chip-wide power consumption data for all the technology nodes, we use McPAT [2]. Table 1 shows the area and peak power (including leakage power) results for our Penryn-like multicore designs in each technology.

Table 1
Area and power of multicore processors with Penryn-like cores

Tech Node(nm)	45	32	22	16
# of Cores	2	4	8	16
Area(mm^2)	116.44	124.78	131.48	149.25
Supply Voltage(V)	1.0	0.9	0.8	0.7
Peak Total Power(W)	74.62	100.48	116.76	148.49

To estimate the worst-case power consumption for each system, we conducted performance simulations and activity factor analyses to extract an empirical reasonable worst-case switching activity. Based on these simulations, we use 80% of McPAT's theoretical peak power as our best estimate for chip practical peak power consumption. McPAT calculates this theoretical peak power by assuming maximum switching activity, therefore requiring that functional blocks be active every cycle. For most of the structures like L2 cache or NoC, this is not achievable nor sustainable.

As discussed above, our CMP floorplans were generated by ArchFP. Chips at different technology nodes share the same single core structure and the area of each functional block is calculated by McPAT. During the construction of all the CMP floorplans, we place MCs on the chip periphery and NoCs between cores as long and thin blocks.

3) Power delivery and temperature modeling: Taking our floorplans as input, we use HotSpot [14] to model chip temperature and a steady-state power delivery model described in [16] to model on-chip power delivery network. Table 2 lists the major physical parameters we used in these two models. For power delivery, we choose on-chip metal parameters to approximate an Intel 45nm metal stack [18]. C4 pad spacing was selected so that our pad density matches ITRS projections. Package resistance comes from [19]. For temperature, we simulated both air cooling and liquid cooling systems and their effective thermal resistance can be found in product datasheets.

D. Results

1) Electromigration on C4 Pads: EM is one of the major failure mechanisms that deserve designers' attention. According to [20], aluminum and copper metal wires, commonly used for on-chip interconnections, can carry two orders of magnitude higher current density than solder joints. This suggests that C4 solder bumps are more vulnerable to EM. For this reason, we calculate the max current density on C4 pads, illustrated by the line in Figure 3. In order to determine the upper bound of the PDN capacity (or the lower bound of PDN noise), we assume that all pads are used for power or ground (and that each type is distributed uniformly). While this is an unrealistic assumption for a real system, it allows us to determine the best-case trend in PDN behavior. In the event that the PDN imposes constraints on the rest of the design under this best case, clearly any design under more realistic assumptions will be constrained by the PDN as well.

Table 2
PDN and cooling system parameters selected for scaling study

Power Delivery Network	
Top Layer Metal Pitch (μm)	30
Top Layer Metal Width (μm)	6
Top Layer Metal Thickness (μm)	5
Top Layer Metal Resistivity (ρ)	1.68e-8
C4 Pad Diameter (μm)	130
C4 Pad Pitch (μm)	285
C4 Pad Resistivity (ρ)	1.46e-7
Package Resistance ($m\Omega$)	0.03
Cooling System Thermal Resistance (K/W)	
High end air cooling	0.25
Liquid cooling	0.1

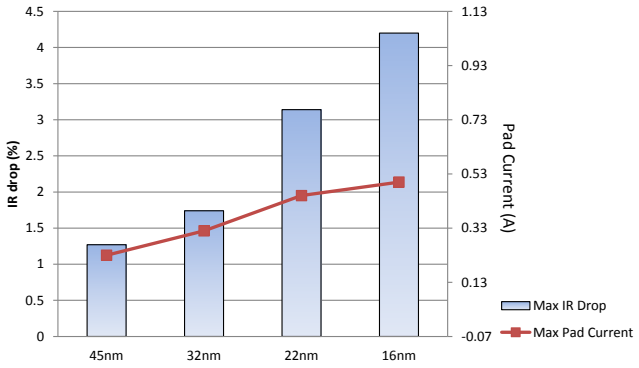

Fig. 3. Max pad current and max on-chip IR drop at each technology node. The upper range of the right Y-axis is the threshold current value for EM (at 100°C). For IR drop, we do not set an explicit threshold value but a 3.8% IR drop could cause as high as 51% delay increase [21]. IR drop therefore poses a more significant risk to failure than EM.

In [20], the author gives an EM threshold current density for SnPb solder. At 100°C, the maximum current density that a solder joint can carry without electromigration damage is $8.5 \times 10^3 A/cm^2$. Combined with our pad diameter assumption, we calculate the per pad current limit as 1.13A.

The maximum value of the right Y-axis in Figure 3 indicates the current limit; it is obvious that even though the maximum pad current increases as the technology scales, the absolute value is still far away from the EM threshold. This could be an indication that under ITRS's projections for total pad count, there may be enough guard band for EM in C4 pads for at least the near future.

2) Steady-State IR drop: IR drop is an important PDN metric because it is directly related to silicon delay increase and frequency degradation. As technology scales, the impact

of IR drop would increase due to higher currents. Similar to the previous section, we dedicate all potential pad locations to power and ground pads and no pads to I/O signals. We then use the model to find the maximum on-chip IR drop ratio for each technology. This gives a lower bound on IR drop and the results are shown in Figure 3. The reported IR drop value combines both voltage drop from power plane and ground bounce on ground plane.

IR drop, unlike EM, does not directly result in immediate failure when a threshold current has been crossed, but results in performance degradation instead. Previous work [21] suggests that a 0.05V voltage drop at $0.13\mu m$ with 1.35V power supply would cause a 15% average and up to 51% maximum delay increase. The bars in Figure 3 show that the IR drop increases as the power density increases with technology scaling, and that the IR-drop ratio value reaches above 4% at 16nm—resulting in non-trivial performance degradation. For a more realistic scenario where not all pads were dedicated to power and ground, the problem would be even worse.

3) Temperature vs. IR drop: Both IR drop and temperature are physical design constraints that closely relate to chip power density. A robust system should be designed with both factors in mind. Figure 4 combines chip max temperature with max IR drop. The temperature results are based on both an air cooling system and a liquid cooling system. Our results to date indicate similar trends for both power delivery and thermal limits. Starting from 16nm, air cooling will be insufficient for the modeled CMP. Even though switching to liquid cooling can keep chip temperature under the limit, power delivery will become a new bottleneck. The platform we built provides an infrastructure for future studies.

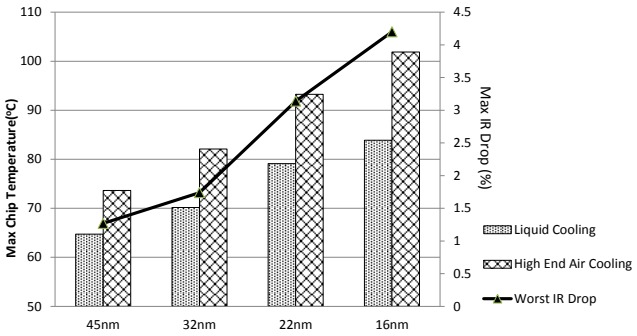

Fig. 4. A comparison between chip max temperature and worst IR drop across different technologies

V. Conclusion

To facilitate investigation of the very large design space for CMPs, tools that can be used to rapidly prototype pre-RTL CMP designs are becoming increasingly important. We present ArchFP, a floorplanning tool specifically designed to operate with *CMP* and *Architectural* components before the *Circuit* and *Technology* details have been determined. ArchFP retargets the flexible and extensible component/container/LM model of GUI design tools to floorplanning. To the best of our knowledge, this approach to floorplanning is completely novel. ArchFP is complementary to traditional floorplanners; it is superior for early prototyping, while traditional floorplanning techniques are necessary later in the design life-cycle. We previously showed that ArchFP could have been useful to several CMP design investigations. Here, we show the utility of ArchFP as part of an integrated tool suite in the study of power delivery and temperature constraints in likely CMP configurations of Penryn-like cores for four technology nodes. We conclude that IR drop may pose a more significant risk to PDNs than EM in C4 pads. In addition, starting with 16nm chips, power delivery is likely to become a more limiting factor than temperature in applications using liquid cooling.

Acknowledgment

This work was supported in part by the NSF under grants CRI-0551630 and MCDA-0903471 and by the SRC under task 1972, as well as grants from AMD and Intel.

References

[1] Y. Li et al., "CMP design space exploration subject to physical constraints," in *HPCA*, Feb. 2006, pp. 17 – 28.

[2] S. Li et al., "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, Dec. 2009, pp. 469 –480.

[3] K. Sankaranarayanan et al., "A case for thermal-aware floorplanning at the microarchitectural level," *Journal of ILP*, vol. 7, 2005.

[4] K.Sankaranarayanan et al., "Architectural implications of spatial thermal filtering," *Integration, the VLSI Journal*, Online, Dec. 2011.

[5] R. Kumar et al., "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," in *ISCA*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 408–419.

[6] D. M. Geary, *Graphic Java 2 : mastering the JFC*, 3rd ed., ser. The Sun Microsystems Press Java series. Palo Alto, CA, USA: Sun Microsystems, 1999, v. 2. Swing.

[7] C. Anderson, *Essential Windows Presentation Foundation*, ser. Microsoft .NET development series. Upper Saddle River, NJ, USA: Addison-Wesley, 2007.

[8] G. Faust et al., "Rapid prototyping of CMP floorplans: A technical report," University of Virginia, Tech. Rep. CS-2012-02, March 2012.

[9] S. H. Gerez, *Algorithms for VLSI design automation*. Chichester, NY, USA: Wiley, 1999.

[10] M. Sarrafzadeh et al., *An Introduction to VLSI Physical Design*, ser. McGraw-Hill series in computer science Computer engineering. New York, NY, USA: McGraw Hill, 1996.

[11] B. H. Meyer et al., "Cost-effective slack allocation for lifetime improvement in NoC-based MPSoCs," in *DATE*, March 2010, pp. 1596 –1601.

[12] "GSRC floorplan benchmark suite." [Online]. Available: http://www.cse.ucsc.edu/research/surf/GSRC/GSRCbench.html

[13] N. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug 2011.

[14] K. Skadron et al., "Temperature-aware microarchitecture," in *ISCA*, San Diego, CA, USA, June 2003.

[15] S. N. Adya et al., "Fixed-outline floorplanning: enabling hierarchical design," *VLSI, IEEE Transactions on*, vol. 11, no. 6, pp. 1120 –1135, Dec. 2003.

[16] R. Zhang et al., "Some limits of power delivery in the multicore era," in *WEED*, Oregon, USA, Jun 2012.

[17] V. George et al., "Penryn: 45-nm next generation Intel Core 2 processor," in *ASSCC*, Nov 2007, pp. 14 –17.

[18] N. H. Weste et al., *CMOS VLSI Design A Circuit and Systems Perspective*, 4th ed. Addison-Wesley, 2011.

[19] *Intel Pentium 4 Processor in the 423 pin package / Intel 850 Chipset Platform*. Intel, 2002.

[20] Y. T. Yeh et al., "Threshold current density of electromigration in eutectic SnPb solder," *Applied Physics Letters*, vol. 86, no. 20, May 2005.

[21] M. Shao et al., "IR drop and ground bounce awareness timing model," in *IEEE Computer Society Symposium on VLSI*, May 2005, pp. 226 – 231.