Multi-Product Floorplan and Uncore Design Framework for Chip Multiprocessors

Marco Escalante[1], Andrew B. Kahng[2], Michael Kishinevsky[1],
Umit Ogras[3] and Kambiz Samadi[4]

[1]Intel Corp., Hillsboro, OR, [2]ECE and CSE Departments, University of California at San Diego
[3]School of ECEE, Arizona State University, [4]Qualcomm Research, San Diego, CA

ABSTRACT

Chip multiprocessors (CMPs) for server and high-performance computing markets are offered in multiple classes to satisfy various power, performance and cost requirements. As the number of processor cores on a single die grows, resources outside the "core", such as the distributed last-level cache, on-chip memory controllers and network-on-chip (NoC) interconnecting these resources, which constitute the "uncore", play an increasingly important role. While it is crucial to optimize the floorplan and uncore of each product class to achieve the best power-performance tradeoff, independent optimization may greatly increase the design effort, and undermine the savings ultimately achieved with a given total amount of optimization effort. This paper presents a novel multi-product optimization framework for next generation CMPs. Unlike traditional chip optimization techniques, we optimize the floorplan of multiple product classes at once, and ensure that the smaller floorplans can be obtained from larger ones by optimally removing, i.e., *chopping*, the unused parts.

1. INTRODUCTION

Due to the diversity of market demand, modern CMPs are offered in multiple versions also known as SKUs (stock-keeping units). For example, the Intel Xeon Server processor code-named Haswell had 27 different SKUs, with number of cores ranging from 4 to 18, announced or launched in the third quarter of 2014 [12]. Typical product classes include a low-cost, low-power model which targets mobile or low-end markets, a high-performance model which targets high-end markets, and multiple medium-cost SKUs to fill the spectrum in between. Different product classes share the same resources, e.g., CPU cores, memory controllers, common cache structures, NoC topology and NoC routers, as the building blocks. However, they differ in the number of cores, amount of on-chip memory, I/O bandwidth and NoC dimensions. These differences lead to changes in the chip floorplan and result in extra design effort, which increases both cost and time-to-market.

SLIP '15 San Francisco, CA USA

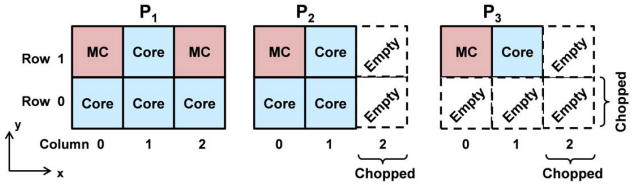

Figure 1: Example CMP floorplans for three different products. Chopped parts are shown with dotted lines for illustration purposes.

During the design of large CMPs, different resources, e.g., cores, memory controllers, memory channels, I/O, and NoC, are laid out in rows and columns in a grid, as shown in Figure 1. Each tile in the grid contains either one of the aforementioned resources, or it is empty, i.e., not occupied by any functional resource. In this work, we focus on the uncore and chip-level floorplanning problem of CMPs, i.e., the floorplan of each individual resource is done *a priori* by allocating space for the NoC links. The goal is to *simultaneously* optimize the chip-level floorplan across multiple CMP products subject to area and power constraints. We propose a novel uncore and floorplan co-optimization framework for the product classes of a given CMP to minimize the design effort. The resulting floorplans satisfy *choppability* constraints, i.e., a smaller configuration can be obtained from a larger configuration by simply chopping and shifting the floorplan, as illustrated in Figure 1. In this example, there are three product classes. Product P_1 is composed of four cores and two memory controllers. Product P_2 is obtained from P_1 by first removing the memory controller and core resources in column 2, and then chopping out the empty tiles. Similarly, P_3 is obtained from P_2 by removing the resources in the bottom row, and then chopping the entire row.

Current practice is to focus on the product class with the largest market share, and optimize it during the early design process. The floorplans of both bigger and smaller product classes are derived from the optimized design. After the architecture and features freeze and front-end design is finished, the full-chip layout of the largest configuration is designed first. Then, the layout of each smaller configuration is obtained by chopping the biggest floorplan.

In this paper, we propose an efficient integer-linear programming (ILP) approach to solve the multi-product CMP floorplanning problem. Unlike traditional chip floorplanning approaches [1, 2, 3, 4, 5], our approach simultaneously optimizes the floorplans of multiple CMP products such that the floorplans of smaller products can be easily derived from those of the larger products via *chopping* oper-

ations. The major contributions of our work are as follows.

- We define the *choppability property* for a given CMP product such that the floorplan of smaller CMP products can be obtained by chopping operations.
- We propose an efficient ILP-based approach to simultaneously optimize the floorplan of multiple CMP products subject to design area and power constraints.
- We extend our baseline problem formulation to enable efficient design space exploration of CMPs under given power and area budgets.
- We support heterogeneous resources by considering different width and height values for cores, memory controllers, and memory channels.
- We provide several realistic examples with varying number of resources, and show that our approach efficiently provides choppable floorplans across all products.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 and Section 4 respectively present preliminaries and the proposed multi-product floorplan optimization approach. Section 5 provides three extensions to our original approach to enable efficient design space exploration. Finally, Section 6 presents our experimental results and Section 7 concludes the paper.

2. RELATED WORK

Physically-aware NoC link allocation for CMPs is addressed in [7], while physical planning of large CMPs for architectural exploration is discussed in [8]. Likewise, the authors of [9] analyze the impact of memory controller placement on the performance. However, all of these studies consider only one product class. We refer the reader to recent surveys [10, 11] for more detailed reviews of the NoC literature. To our knowledge, the only prior work that considers floorplan optimization of multiple products is [6], which proposes a simulated annealing based approach that packs the building blocks used in each product class. The authors of [6] also extend their cost function to include a weight for each product to represent its importance relative to other products. However, this approach does not consider choppability of the floorplans, which enables a great increase in productivity due to design reuse. Similarly, power budget and performance constraints of individual product classes are not considered by prior work. Unlike prior work, our framework enables multi-product CMP floorplan optimization considering both the full-chip layout and power consumption constraints.

3. PRELIMINARIES AND NOTATION

Let $P_1, P_2, \cdots, P_S$ denote S product classes targeting different market segments, and let K be the number of different resource types (CPU, cache bank, controllers, etc.) used in the CMP. We use $k = 1$ to represent the core, $k = 2$ for the memory controller, and $k = 3$ for memory channels, which are the most common resources. We support an arbitrary number of resource types $1 \leq k \leq K$, where the encoding for the remaining resources can be arbitrary. To facilitate the definition of choppability, we introduce $k = 0$ to represent the *empty* type, i.e., the lack of a functional resource, as shown in Figure 1. Each product P_i, $1 \leq i \leq S$, can be represented by the K-tuple $< n_{i0}, n_{i1} \cdots, n_{iK} >$, where n_{ik} shows the number of type k resources used in product P_i. Then, the product classes can be represented as follows:

$$\begin{aligned} P_1 &= < n_{10}, n_{11}, \cdots, n_{1K} > \\ P_2 &= < n_{20}, n_{21}, \cdots, n_{2K} > \\ &\vdots \\ P_S &= < n_{S0}, n_{S1}, \cdots, n_{SK} > \end{aligned} \quad (1)$$

Definition 1. For any $1 \leq i, j \leq S$, product class P_i *encompasses* class P_j ($P_i \succ P_j$) if and only if $n_{ik} \geq n_{jk}\ \forall\ 1 \leq k \leq K$.

That is, if $P_i \succ P_j$, P_i has all the resources found in P_j and possibly additional ones. Without loss of generality, we index product classes such that $P_1 \succ P_2 \succ \cdots \succ P_S$. In a CMP product i with R^i rows and C^i columns (i.e., $R^i \times C^i$ tiles), the binary variable u^i_{rck} denotes whether the tile (r,c) contains a resource of type k ($u^i_{rck} = 1$) or not ($u^i_{rck} = 0$), where $0 \leq r \leq R^i - 1$, $0 \leq c \leq C^i - 1$, $0 \leq k \leq K$. As an example, we show a matrix representation of product P_S.

$$\mathbf{U^S_{R^S C^S 1}} = \begin{bmatrix} u^S_{001} & \cdots & u^S_{0(C^S-1)1} \\ \vdots & \ddots & \vdots \\ u^S_{(R^S-1)01} & \cdots & u^S_{(R^S-1)(C^S-1)1} \end{bmatrix}, \quad (2)$$

$$\vdots$$

$$\mathbf{U^S_{R^S C^S K}} = \begin{bmatrix} u^S_{00K} & \cdots & u^S_{0(C^S-1)K} \\ \vdots & \ddots & \vdots \\ u^S_{(R^S-1)0K} & \cdots & u^S_{(R^S-1)(C^S-1)K} \end{bmatrix}$$

The example in Figure 1 shows three different products P_1, P_2, P_3, with three resources: Empty ($k = 0$), core ($k = 1$), and memory controller ($k = 2$). Using our notation, the encoding for P_2 in this example is:

$$P_2: \quad u^2_{001} = 1,\ u^2_{011} = 1,\ u^2_{111} = 1,\ u^2_{102} = 1$$
$$u^2_{020} = 1,\ u^2_{120} = 1, \text{the rest of the variables are 0}$$

Simultaneous optimization across multiple products requires deriving the floorplans of smaller products from the larger ones to minimize the implementation effort. Removing resources arbitrarily does not result in an optimal design for smaller products even when the larger design is optimal. Our goal is to remove the resources in such a way that the final layouts of the larger products can be *chopped* to obtain those of the smaller products. Chopping operations simply remove an entire row or column. In other words, all the resources in the chopped row or column are converted to empty tiles. In the following, we formally define the chopping operation, as well as the choppability property for a given product. Removing only a subset of the tiles in a row or column would result in whitespace.

Definition 2. The *row chopping operation* in product i for row r^*, $0 \leq r^* < R^i$, is defined as

$$\forall c \text{ set } u^i_{r^*ck} = 0 \text{ for } k > 0, \text{ and } u^i_{r^*c0} = 1$$

That is, the resource type in every tile of the chopped row is marked as empty ($k = 0$). Likewise, the *column chopping operation* for column c^*, $0 \leq c^* < C^i$, is

$$\forall r \text{ set } u^i_{rc^*k} = 0 \text{ for } k > 0, \text{ and } u^i_{rc^*0} = 1$$

Definition 3. If there is a sequence of chopping operations that transforms P_i to P_j, then we say that P_i *can be chopped to* P_j, and denote this by $P_i \rightsquigarrow P_j$.

4. MULTI-PRODUCT FLOORPLAN OPTIMIZATION

4.1 Basic Problem Formulation

Let the chip height and width of product class P_i be H_i and W_i, respectively. We define the aspect ratio as $A_R = \frac{H_i}{W_i}$, and constrain it as $1/R_{max} \leq A_R \leq R_{max}$. Next, we introduce the objective function and constraints used in our optimization formulation.

Objective function. In our formulation, we minimize the sum of half-perimeter of all products, instead of total area, to keep the objective function linear.

$$\textbf{Minimize} \sum_i (H_i + W_i) \tag{3}$$

The half perimeter can be obtained as $H_i + W_i = W_i(A_R + 1)$. We note that a rectangle of given perimeter has minimum area when its aspect ratio is maximum, i.e., $A_R = R_{max}$. The minimum area is $A_{min} = W_i^2 R_{max}$, while the corresponding half perimeter is $W_i(R_{max} + 1)$. On the other hand, the maximum area with the same perimeter is obtained when both sides are equal to $W_i(R_{max} + 1)/2$. Hence, $A_{max} = W_i^2(R_{max} + 1)^2/4$ and the ratio between the maximum and minimum area for a given half perimeter is $(R_{max}+1)^2 : 4R_{max}$. For example, if $R_{max} = 2$, the worst-case ratio is $9 : 8$, which means minimizing half-perimeter can end up with 12.5% more area than the minimum. To reduce deviation from area minimization, we can impose an upper bound on aspect ratio.

Constraints on the number of resources. Each tile can be occupied by only one type of resource, and each product P_i has a specified number of instances of the k^{th} resource type (N_k^i). These constraints are specified as:

$$\forall i, r, c \;\; \sum_k u_{rck}^i = 1 \tag{4}$$

$$\forall i, \;\; \forall k > 0 \;\; \sum_r \sum_c u_{rck}^i = N_k^i \tag{5}$$

Monotonicity constraints. Suppose $P_i \rightsquigarrow P_j$ (P_i can be chopped to P_j). If a tile in product P_i is occupied, then the corresponding tile in P_j can either be occupied by the same resource or be empty.

$$\forall i, r, c \;\; u_{rck}^i \geq u_{rck}^j \;\text{ for }\; k > 0 \tag{6}$$

However, if a tile in product P_i is empty, the corresponding tile in P_j must also be empty.

$$\forall i, r, c \;\; u_{rc0}^i \Rightarrow u_{rc0}^j, \; i.e., \;\; u_{rc0}^i \leq u_{rc0}^j \tag{7}$$

Placement constraints. Certain resources, such as memory channels and I/O controllers, may need to be placed at the boundary of the chip. Let the set of these resources be $S_{boundary}$.

$$\forall i, \; k \in S_{boundary}, \;\; u_{rck}^i = 0, \;\; \text{for} \\ 0 < r < R^i - 1, \;\; 0 < c < C^i - 1 \tag{8}$$

Height and width computations. Chip height H_i and width W_i are functions of the resources used in each product. Let $used_r^i$ and $used_c^i$ denote whether row r and column c are used in P_i. Row r in product P_i is chopped only if all the tiles within that row are empty, i.e., when $used_r^i = 0$. Similarly, column c in product P_i is chopped if $used_c^i = 0$. We can express $used_r^i$ and $used_c^i$ as:

$$\forall i, r \;\; used_r^i = \begin{cases} 1 & \text{if } \sum_{0 \leq c \leq C^i - 1} \;\; \sum_{1 \leq k \leq K} u_{rck}^i \geq 1 \\ 0 & otherwise \end{cases} \tag{9}$$

$$\forall i, c \;\; used_c^i = \begin{cases} 1 & \text{if } \sum_{0 \leq r \leq R^i - 1} \;\; \sum_{1 \leq k \leq K} u_{rck}^i \geq 1 \\ 0 & otherwise \end{cases} \tag{10}$$

If any one of the tiles in a given row (column) is occupied by a resource, we mark that row (column) as used. Hence, we can obtain H_i and width W_i by counting the number of used rows and columns multiplied by tile height h and width w, respectively:

$$\forall i \;\; H_i = h \times \sum_{0 \leq r \leq R^i - 1} used_r^i \tag{11}$$

$$\forall i \;\; W_i = w \times \sum_{0 \leq c \leq C^i - 1} used_c^i \tag{12}$$

In this formulation, the tile height and width are assumed to be the same for all resources. This assumption will be relaxed in Section 5.

4.2 Handling Resources Occupying Contiguous Tiles

In a given CMP product, the boundary tiles are allocated for memory channels (MCh) and I/O devices, as illustrated in Figure 2. Memory channels, which often occupy more than one contiguous tile, are called memory channel groups. Since MCh groups are connected to the memory controllers, their placement affects the placement of memory controllers.

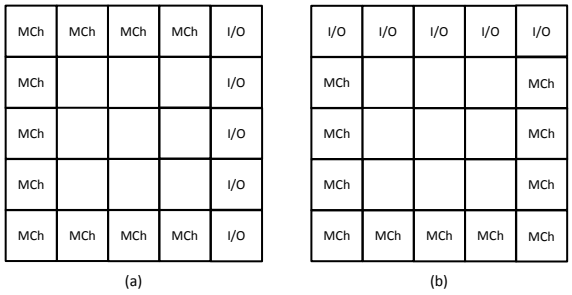

Figure 2: Two possible memory channel and I/O placements at the boundary of the design.

For a given product class, let G be the number of memory channel groups and $g \in \{1, 2, \ldots, G\}$ denote each group. The size of the g^{th} memory channel group, w_g is the number of contiguous tiles that it occupies. For example, there are two memory channel groups with sizes 1 and 2 in Figure 3. The direction along which the memory channel extends is given by $d \in \{\uparrow, \rightarrow\}$. Note that the other directions (i.e., $\downarrow, \leftarrow$) are symmetric to the directions in d. The binary variable v_{rcdg} is used to mark the starting tile of memory channel g, where $v_{rcdg}^i = 1$ denotes that tile (r, c) is the starting tile of memory channel g in the direction d in product P_i.

To ensure that different memory channel groups do not overlap with each other, we enumerate all the pairwise combinations of dif-

ferent memory channel groups and add the following constraints:

$$\forall i, r', c', d', g', r'', c'', d'', g'' \quad (g' \neq g'')$$
$$v^i_{r'c'd'g'} + v^i_{r''c''d''g''} \leq 1 \tag{13}$$
$$\forall i, g \quad \sum_r \sum_c \sum_d v^i_{rcdg} = 1 \tag{14}$$

Constraint (13) ensures that the starting tile of each memory channel group is different, while Constraint (14) guarantees that the specified number of memory channels are placed in each product. In addition to these constraints, we also need to ensure that the starting tile of a memory channel group does not overlap with another memory channel. To illustrate this, we consider the memory channels shown in Figure 3, where there are two memory channel groups of sizes one and two (i.e., $w_1 = 1$ and $w_2 = 2$). For the sake of clarity and space considerations, we *only* give the constraints corresponding to g_2 as shown in Figure 3:

$$v^i_{20\to 2} + v^i_{20\to 1} \leq 1 \tag{15}$$
$$v^i_{20\to 2} + v^i_{20\uparrow 1} \leq 1$$
$$v^i_{20\to 2} + v^i_{21\to 1} \leq 1 \tag{16}$$

Assuming the location and size of g_2 in Figure 3, the first and second constraints ensure that the starting points of g_1 cannot be tile $(2, 0)$ irrespective of its direction. Constraint (16) introduces one more constraint that prohibit g_1 from starting at tile $(2, 1)$, since $(2, 1)$ is already occupied by g_2.

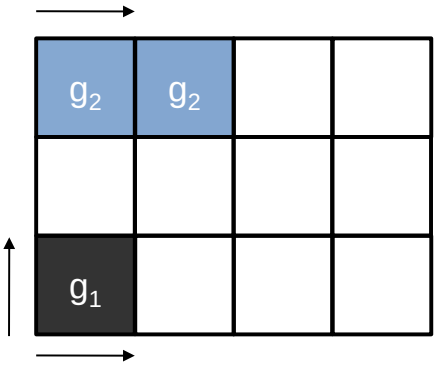

Figure 3: An example of a design with two memory channel groups.

Adjacency constraints. Since the placements of memory channels affect the placements of memory controllers, we have to add necessary constraints to guarantee that a memory controller tile is adjacent to a memory channel group. Consider the configuration shown in Figure 2(a), where the memory channels can be placed at the leftmost column, top row and bottom row. Memory controllers need to be adjacent to one of these three sides. For example, consider the top row, $r = R^i - 1,\ 0 \leq c \leq C^i - 1$. If there is a memory controller in $(R^i - 2, c)$, i.e., one row below the top row, then there has to be a memory channel MCh in $(R^i - 1, c)$. This constraint can be written as:

$$\forall i, 0 \leq c \leq C^i - 1, \quad \sum_g v^i_{(R^i-1)c\to g} \geq u^i_{(R^i-2)c2} \tag{17}$$

Similarly, the corresponding constraints for the first column and bottom row can be written as:

$$\text{Leftmost column: } \forall i, 0 \leq r \leq R^i - 1, \ \sum_g v^i_{r0\uparrow g} \geq u^i_{r12} \tag{18}$$

$$\text{Bottom row: } \forall i, 0 \leq c \leq C^i - 1, \ \sum_g v^i_{0c\to g} \geq u^i_{1c2} \tag{19}$$

Constraints (17), (18) and (19) ensure that whenever there is a memory controller, there will also be a memory channel adjacent to it. To implement a logical OR operation between the two given configurations, we need nonlinear constraints which degrades the performance of our proposed approach. Hence, we construct two separate constraints files corresponding to Figure 2(a) and 2(b) configurations, and solve these two problems separately.

Additional placement constraint. Since the memory channels and I/O devices are placed at the boundary of the design, core and memory controller blocks can only be placed in the inner tiles (see the white tiles in Figures 2(a) and 2(b)). We add the following constraints to avoid the placement of cores and memory controllers on the chip boundary:

$$\forall i, r \in \{0, R^i - 1\}, c \in \{0, C^i - 1\}$$
$$u^i_{rck} = 0 \ \text{ for } k \notin S_{boundary} \tag{20}$$

where $S_{boundary}$ is the set of resource types that can be in the boundary.

In the next section, we propose additional constraints to enable effective floorplan design space exploration across multiple products.

5. POWER-PERFORMANCE-DRIVEN DESIGN SPACE EXPLORATION

Thus far, we have co-optimized floorplans of multiple CMP products to minimize the sum of the half-perimeters of all product classes. To achieve efficient uncore design space exploration under power and performance constraints, we now add the following extensions to the proposed framework.

- We allow the number of cores and memory controllers for each product to vary in a given range.
- We add constraints on maximum number of memory controllers in a given row or column.
- We consider different width and height values for different resource types.

5.1 Extension 1: Power Exploration

The complete design specification is often not available early in the design cycle. In particular, the "landing zone" of target thermal design power (TDP) and memory bandwidth is known, but precise targets evolve through time. In addition, the number and definition of product classes also change during the design cycle. Hence, there is a need for a fast design space exploration capability that can generate the uncore floorplan for multiple products quickly as the design parameters evolve. Therefore, our first extension is to allow the number of cores and memory channels to vary in a given range for each product. To achieve this, we define the TDP for product P_i, denoted as P^i_{TDP}. If we denote the thermal design power of resource k as p_k, the TDP constraint can be expressed as:

$$\forall i \sum_{k=1}^{K} p_k \sum_r \sum_c u^i_{rck} \leq P^i_{TDP} \tag{21}$$

This means that the number of resources in each product class is not fixed. Hence, the optimization tool is free to select the number of resources under the TDP constraint given in Equation 21, instead of using the constraint given by Equation 5. In addition to ensuring that each product class meets its TDP constraint, we can take advantage of Equation (21) to push the performance limit. For example, assume that $p_{core} = 2W$, $p_{MC} = 1W$, and $P^1_{TDP} = 8W$. Figures 4(a) and (b) show two possible configurations for P_1. We prefer the configuration in Figure 4(b) since it utilizes the power budget more efficiently by utilizing an extra core without exceeding the power budget. Therefore, we modify our original objective such that it minimizes the sum of half-perimeter and the number of empty tiles across all products:

$$\textbf{Minimize:} \sum_i (H_i + W_i + \sum_r \sum_c u^i_{rc0}) \tag{22}$$

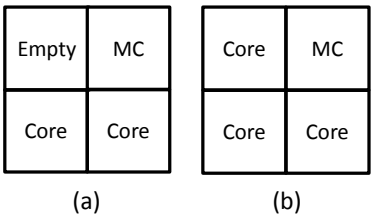

Figure 4: Two possible configurations for a given product.

5.2 Extension 2: Performance Enhancement

Memory controllers receive all the cache miss traffic directed by the last level cache to the memory. Hence, they become hotspots when there are many cache misses. Therefore, placing many memory controllers on a given row (column) causes congestion on the corresponding row (column), and results in performance degradation. To alleviate this problem, we add constraints on the maximum number of memory controllers that can be placed on a given row or column.

$$\forall i, r \ \sum_c u^i_{rc2} \leq \text{Max number of MC's per row} \tag{23}$$

$$\forall i, c \ \sum_r u^i_{rc2} \leq \text{Max number of MC's per column} \tag{24}$$

where $u^i_{rc2} = 1$ implies that there is a memory controller in row r, column c of product class i.

5.3 Extension 3: Heterogeneous Resource Support

Finally, we consider different width and height for core, memory controller, and memory channel tiles to support heterogeneous resources. More specifically, we add Constraints (25) and (26) to achieve the maximum height (width) in given row (column). Then, we modify our area computation to capture the difference in core and memory controller dimensions using Constraints (27) and (28):

$$\forall i, r, 0 \leq c < C^i, \ \ h^i_r \geq u^i_{rc1} \times h_{core} \tag{25}$$
$$\forall i, r, 0 \leq c < C^i, \ \ h^i_r \geq u^i_{rc2} \times h_{MC}$$
$$\forall i, c, 0 \leq r < R^i, \ \ w^i_r \geq u^i_{rc1} \times w_{core} \tag{26}$$
$$\forall i, c, 0 \leq r < R^i, \ \ w^i_c \geq u^i_{rc2} \times w_{MC}$$
$$\forall i, r, c, \ \ H_i = \sum_r h^i_r \tag{27}$$
$$\forall i, r, c, \ \ W_i = \sum_c w^i_c \tag{28}$$

where h_{core}, w_{core}, h_{MC}, and w_{MC} denote core height and width, and memory controller height and width, respectively. h_r and w_c also denote height of row r and width of column c, respectively. Note that our minimization objective ensures that the smallest possible values for h^i_r and w^i_c are obtained. In the following section, we describe our experimental setup, and discuss our results.

6. EXPERIMENTAL RESULTS AND ANALYSIS

We now describe the infrastructure developed for the proposed multi-product CMP floorplan optimization framework, and discuss our experimental results.

6.1 Experimental Setup

We developed a Perl script (∼2000 lines of code) to read in a floorplan description file, and generate the corresponding ILP constraints. We feed the ILP formulation to CPLEX [13] to obtain the uncore floorplan description for each product class. Finally, we generate a visual representation for each product class. The input floorplan description file includes the following information: (1) the grid size, (2) minimum (resp. maximum) number of cores and memory controller tiles for each product, (3) maximum number of memory controller tiles in a given row or column of a product, (4) core and memory controller dimensions, and (5) building block power values, and power budget for each product class. Grid size is set to be greater than the total number of tiles in the largest product class. Our script generates an ASCII file which contains the corresponding ILP constraints for the given floorplan description file. To show the chopping operations that have taken place from P_i to P_j, we generate visual representations of all the intermediate products between P_i and P_j.

6.2 Representative Example

To validate the proposed approach, we investigate the example problem shown in Figure 5, which has two product classes: (1) P_1 with 14 cores and 2 MCs, and (2) P_2 with 6 cores and 2 MCs. In this example, h_{core} and h_{MC} are 3 and 1 units, and w_{core} and w_{MC} are both 4 units. For P_1, we have to make sure to place the MCs on the boundary. However, placing both of them in the first or last column will not help minimize the area for P_2, since the memory controller width is the same as that of core. In addition, placing the memory controllers in different rows is not advantageous, since we would not benefit from the smaller height of the memory controller to minimize area when chopping rows. On the contrary, we may achieve a configuration with a smaller row after corresponding chopping operations, if we place both of the memory controllers in either the top or the bottom row, as shown in Figure 5(a).

Figures 5(b), (c), and (d) show possible configurations for P_2. Note that there are other symmetric solutions which are identical

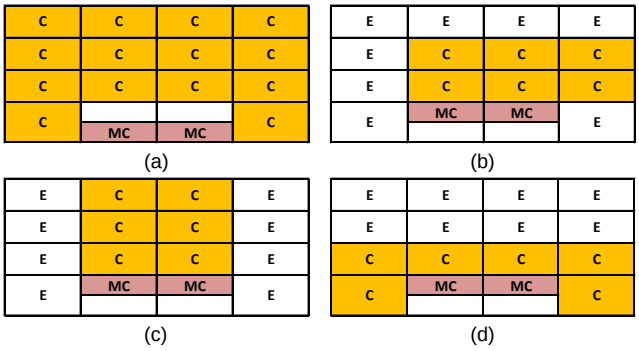

Figure 5: An example testcase with two products.

to the configurations shown in the figures. Given core and memory controller width and height values, the configuration in Figure 5(c) achieves the smallest half-perimeter (respectively area) among all three configurations. The proposed method has appropriately picked the solution shown in Figure 5(c). In addition to the optimal solution, our approach enables the designers to obtain and inspect all the possible solutions from the pool of solutions derived by CPLEX such that they can be compared. We have also verified our approach against real industry products and observed that our proposed solutions match those developed by the designers, but are obtained in a much shorter amount of time. This productivity gain is significant, since multi-product floorplan optimization is repeated many times due to changes in the core architecture and NoC design.

6.3 Significance Assessment

We performed additional experiments using three testcases with varying number of cores, memory controllers, and memory channels, as summarized in Table 1. In our testcases, core and MC tiles have both 3 units of width, while the core and MC heights are 2 units and 1 unit, respectively. The width (height) of a column (row) is determined by the width (height) of the largest building block in that column (row). As mentioned before, memory controllers can reduce chip height if all them are placed in the same row.

The floorplan of each product is simultaneously obtained after solving the associated ILP problem. In our problem formulation, the grid size is independent from total number of tiles in the largest product. For instance, if the largest product has 20 tiles, the grid size does not have to be 4×5 or 5×4, but can be any size that contains the largest product.[1] This allows us to efficiently explore different solutions with heterogeneous resources.

Table 1: Our experimental testcases.

		Num. of Cores				Num. of MCs				Num. of MChs			
Product		1	2	3	4	1	2	3	4	1	2	3	4
Testcase	1	24	16	10	6	6	4	2	2	6	4	2	2
	2	36	27	18	-	8	6	4	-	8	6	4	-
	3	48	32	24	16	8	6	4	2	8	6	4	2

[1]Selecting a very large grid size will increase the runtime due to additional constraints for the extra tiles.

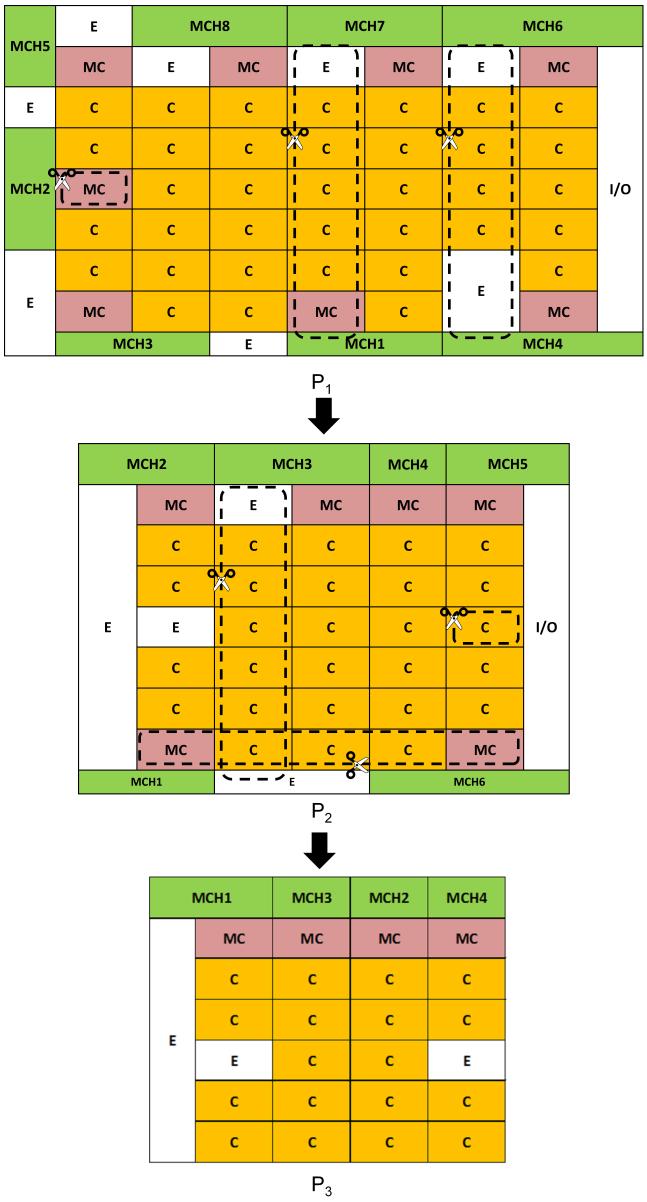

Figure 6: Testcase 2 with three different products and varying number of cores, memory controller, and memory channels.

For illustration purposes, we show the three product classes of testcase 2 in Figure 6. Tiles denoted with C, MC, MCh, and E represent cores, memory controllers, memory channels, and empty tiles, respectively. Figure 6 shows the final floorplan of all three product classes of testcase 2. Product class P_2 is derived from P_1 by three consecutive chopping operations. First, the 5^{th} and 7^{th} columns are chopped, as illustrated in Figure 6 using the dotted lines. The resulting floorplan would have one extra memory controller than the specification summarized in Table 1. Hence, the memory controller in the middle of the second column and the corresponding memory channel are chopped. Finally, we note that the remaining columns are shifted to obtain the floorplan of P_2 depicted in Figure 6. Similarly, the third column, second row, and an additional core in P_2 are chopped away to obtain the product class P_3, as illustrated in Figure 6.

Table 2 shows the number of binary variables, number of constraints, and the CPU runtime to solve the corresponding ILP problem. We observe that our approach has good scalability with respect to the number of resource types used in a given design. All of our testcases represent future-generation CMPs; runtimes for smaller testcases are on the order of a few seconds to a few minutes. In addition, our method can be easily run on multiple computing resources if multiple product configurations need to be explored.

Table 2: Complexity and runtime of our approach.

Testcase	#binary variables	#constraints	CPU runtime (sec)
1	595	3014	687
2	896	6204	4744
3	1089	7218	14936

7. CONCLUSIONS

In this paper, we have proposed a simultaneous floorplan optimization framework for CMPs across multiple products. We define the concept of a *choppable floorplan* which enables us to easily derive the floorplan of smaller products from those of larger ones through simple *chopping operations*. Our approach supports (1) multiple resource types (i.e., core, memory controller, memory channel, etc.), (2) design space exploration of achievable floorplans under given power and performance budgets, and (3) heterogeneous resources (i.e., different height and width values for each resource type. We observe that our approach efficiently finds choppable floorplans across multiple products to reduce re-design costs and shortens time-to-market.

References

[1] S. N. Adya and I. Markov, "Fixed-Outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. VLSI*, 11(6) 2003, pp. 1120–1135.

[2] K. Sankaranarayanan, S. Velusamy, M., L. Charles and K. Skadron, "A Case for Thermal-Aware Floorplanning at the Microarchitectural Level," *Journal of ILP*, (8) 2005, pp. 1–16.

[3] A. M. Smith, G. A. Constantinides and P. Cheung, "Integrated Floorplanning, Module-Selection, and Architecture Generation for Reconfigurable Devices," *IEEE Trans. VLSI*, 16(6) 2008, pp. 733–744.

[4] S. Sutanthavibul, E. Shragowitz and J. B. Rosen, "An Analytical Approach to Floorplan Design and Optimization," *IEEE Trans. CAD*, 10(6) 1991, pp. 761–769.

[5] D. F. Wong and C.-L. Liu, "A New Algorithm for Floorplan Design," *Proc. DAC*, 1986, pp. 101–107.

[6] Q. Ma and M. D. F. Wong, "Configurable Multi-Product Floorplanning," *Proc. ASPDAC*, 2010, pp. 549–554.

[7] N. Nikitin, S. Chatterjee, J. Cortadella, M. Kishinevsky and U. Y. Ogras, "Physical-Aware Link Allocation and Route Assignment for Chip Multiprocessing," *Proc. NOCS*, 2011, pp. 125–134.

[8] J. de San Pedro, N. Nikitin, J. Cortadella and J. Petit, "Physical Planning for the Architectural Exploration of Large-scale Chip Multiprocessors," *Proc. NOCS*, 2013, pp. 1–2.

[9] D. Abts, N. D. Enright Jerger, J. Kim, D. Gibson and M. H. Lipasti, "Achieving Predictable Performance Through Better Memory Controller Placement in Many-Core CMPs," *ACM SIGARCH Computer Architecture News*, 37(3) 2009, pp. 451–461.

[10] R. Marculescu and P. Bogdan, "The Chip Is the Network: Toward a Science of Network-on-Chip Design," *Foundations and Trends in Electronic Design Automation*, 2(4) 2009, pp. 371–461.

[11] U. Y. Ogras and R. Marculescu, *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*, Springer Science & Business Media, 2013.

[12] Intel ARK, "Haswell Products", `http://ark.intel.com/products/codename/42174/Haswell\#@Server`.

[13] ILOG CPLEX, `http://www.ilog.com/products/cplex/`.