A hierarchical approach for generating regular floorplans

Javier de San Pedro Jordi Cortadella Antoni Roca
Universitat Politècnica de Catalunya
Barcelona, Spain

ABSTRACT

The complexity of the VLSI physical design flow grows dramatically as the level of integration increases. An effective way to manage this increasing complexity is through the use of regular designs which contain more reusable parts. In this work we introduce *HiReg*, a new floorplanning algorithm that generates regular floorplans. HiReg automatically extracts repeating patterns in a design by using graph mining techniques. Regularity is exploited by reusing the same floorplan for multiple instances of a pattern, as long as neither area, wire length or existing hierarchy constraints are violated or compromised. The proposed scheme is targeted towards early system-level design of chip multiprocessors (CMPs). Experiments show the scalability of the method for many-core CMPs and competitive results in area and wire length.

1. INTRODUCTION

The computational complexity of the floorplanning problem highly depends on the number of components of the system. For large systems, a flat view makes the floorplanning problem intractable. For this reason, hierarchical methods [16, 15] has been proposed and successfully used to reduce this complexity. Hierarchical methods divide the floorplanning problem into multiple subproblems that may be either fully or partially independent from each other, thereby enhancing scalability.

An important metric often disregarded during floorplanning is *regularity*, which is known to lead to efficient and economical designs [14]. Large-scale systems have significant amounts of regular patterns than can be exploited (on-chip memories, many-core CMPs, etc.). The design cost of such systems can be brought down by reducing the number of distinct subcircuits to be designed, and then replicating the pre-designed subcircuits as many times as possible. To allow this reduction, a regular floorplan uses the exact same layout for all replications of a subcircuit. To reduce complexity of timing closure, it is also desirable for all of the adjacent components to be placed in similar relative positions, so that the interconnect geometries are regular and timing analysis is similar.

Table 1: Comparison of related work.

	Hierarchy	Regularity
[5]	No	Arrays only
REGULAY [17]	Yes	Tiles only
DeFer [16]	Yes	No
CompaSS [4]	By similarity	No
[15]	Yes	No
ArchFP [8]	Manual	Manual
HiReg	Yes	Yes

In many-core CMPs, tiled layouts [3] are often used to exploit regularity. The design is split into homogeneous tiles that are only floorplanned once and then replicated. However, with industry moving towards heterogeneous CMPs [11], it is no longer possible to assume that most CMPs will have only a single type of tile. Integrated graphics, accelerators and I/O blocks are some of the special types of co-processors that introduce heterogeneity.

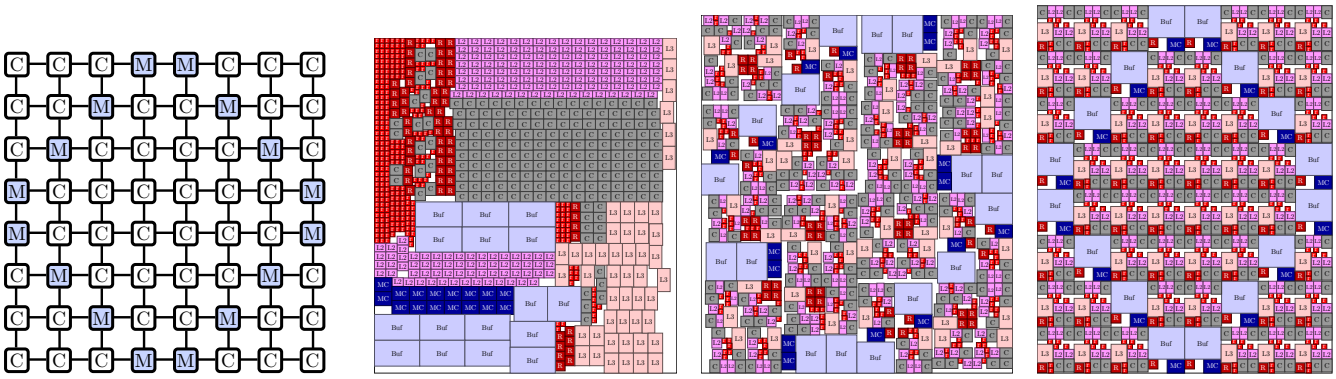
On the other hand, enforcing regularity in a design may compromise other floorplan metrics such as area or wiring. Existing designs are often hierarchical in nature. CMPs with hierarchical topologies are a good example. Preserving the pre-defined hierarchy may result in a better wiring quality (e.g. by reducing the number of wires that cross between different subcircuits). Very often, hierarchy is manually enforced by designers to split the design and assign components to different design teams. Thus, breaking the existing hierarchy could be counterproductive to the goal of simplifying design. Another example is the concept of *choppability* [13]. In a choppable floorplan, large functional blocks can be chopped away, reducing the total die size and varying performance/power metrics in order to construct multiple versions of a product from the same basic design.

This paper presents a new floorplanning algorithm, *HiReg*, that considers area, wiring, regularity and hierarchy as floorplanning objectives. Very often these objectives are conflicting, e.g., reducing final area compromises regularity and vice versa. To deal with this issue, HiReg uses a new method that can trade-off hierarchy and regularity constraints. Both hierarchy and regularity are automatically discovered from the block netlist.

1.1 Related work

There has been little work in the area of regular floorplans. Regularity is more common in the area of physical design for analog circuits, where it is often a strict requirement due to the peculiarities of analog design [2]. However, most of

978-1-4799-6278-5/14/$31.00



(a) Diamond pattern (b) CompaSS (c) DeFer (d) HiReg

Figure 1: Example CMP floorplans generated using different floorplanning strategies.

the techniques in analog design involve symmetry properties that are not relevant for maximizing design reusability.

[5] acknowledges the importance of regular designs in CMPs and describes a simulated annealing-based floorplanner that organizes groups of similar blocks in regular arrays. However, the technique does not fully exploit regularity since adjacent components may not be placed in aligned locations that enable regular interconnection geometries. The blocks that are to be placed in regular groups must also be manually selected by the designer.

In System-on-Chip design, *REGULAY* [17] also mentions the importance of preserving regularity and hierarchy. *REGULAY* discovers the optimal mapping of heterogeneous tiles into a regular grid arrangement, and does not consider the floorplanning of the individual tiles themselves.

On the other hand, the advantages of using hierarchy during floorplanning are not new [6]. Nonetheless, most existing work uses hierarchy only to improve the scalability of the floorplanning problem, allowing efficient generation of floorplans with large numbers of components, and differ in the methodologies used to discover hierarchy.

DeFer [16] uses graph bipartitioning to generate a binary tree of balanced netlist partitions, a method similar to the one proposed in this work for hierarchy discovery. This hierarchy tree is then used to generate a slicing tree, reducing the number of floorplans that need to be explored during the search. *CompaSS* [4] automatically clusters blocks with similar or identical shapes, and then creates grid floorplans for them. However, *CompaSS* ignores connectivity information. [15] applies a recursive slice-and-partition method derived from cell placement strategies.

All three examples use slicing floorplans and bounding curves that will also be used in this work to efficiently represent floorplans. Slicing floorplans are not able to represent the entire set of optimal floorplans. However, this difference is minimal given a large number of soft blocks [18], as it often occurs in CMPs.

ArchFP [8] describes a different strategy that can produce floorplans that are both regular and hierarchical. However, *ArchFP* assumes that a designer will construct, previously to the floorplanning process, a manual hierarchy of the CMP components and will choose a floorplanning approach for each group of components. Our work extracts hierarchy and regularity in an automated way, without any previous knowledge of the topology of the input netlist. Table 1 summarizes the differences between these strategies.

2. OVERVIEW

We start this section with an example to illustrate the trade-offs between regularity and hierarchy. Figure 1 shows the result of floorplanning the same netlist using HiReg and two other hierarchical floorplanners. Table 3 contains whitespace and wire length results. This netlist represents a hierarchical tiled CMP, with 192 cores. It contains 64 tiles, with 48 *processing* tiles containing 4 cores each, and 16 *memory controller* tiles. This CMP uses a hierarchical Network-on-Chip topology. An 8×8 mesh interconnects all tiles. Inside each tile, a ring provides connectivity.

The mapping of processing and memory controller tiles has been selected to match the *diamond* pattern (Fig. 1(a), [1]) which maximizes off-chip memory performance. Thus, this configuration is representative of a potential many-core CMP design. For the sake of easy visualization, both types of tiles have similar area requirements in this example. In general, each tile may have a different area constraint.

In addition to cores (C), processing tiles contain L2 caches private to each core, a tile-shared L3 cache block, ring routers for intra-tile communication (r) and mesh routers for inter-tile communication (R). The memory controller tiles each contain a buffer (Buf), mesh router, and a memory controller itself (MC). Physical information is described in Table 2. Cores come in several hard aspect ratios, but we assume memories to be flexible within a limited range. We also assume every net represents a link with 1024 wires.

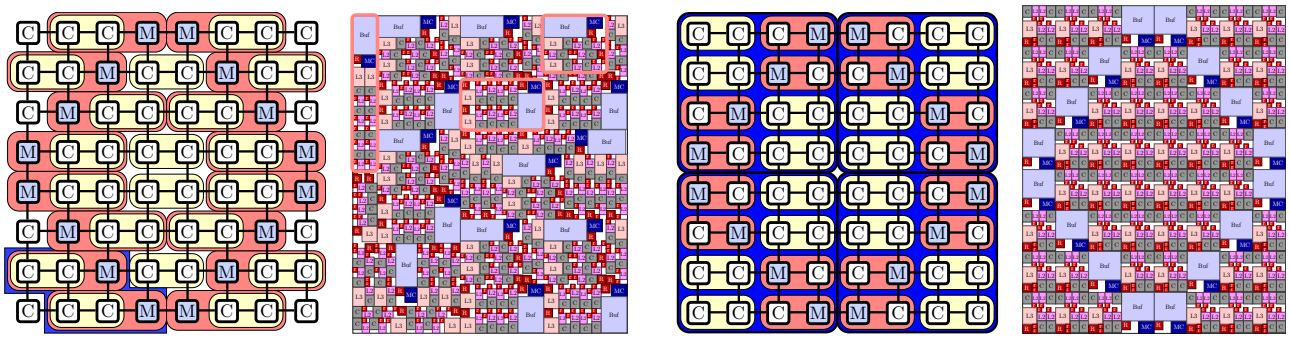
In Fig. 1(b), CompaSS groups blocks by similarity and creates arrays in order to improve packing quality, thus resulting in floorplans that have some regularity. However,

Table 2: Physical information for Fig. 1.

Component	Area	Aspect ratio
Core (C)	1.38 mm^2	0.8 or 1.25
L2 cache	1 mm^2	$0.5 \div 2$
L3 cache	3 mm^2	$0.5 \div 2$
Ring router (r)	0.27 mm^2	1
Mesh router (R)	0.99 mm^2	1
Memory controller (MC)	2.5 mm^2	0.8 or 1.25
Buffer (Buf)	12 mm^2	$0.5 \div 2$

Table 3: Floorplanning results for Fig. 1.

	Whitespace	HPWL (m)
CompaSS	6.3%	6801
DeFer	8.2%	630
HiReg	12.6%	516



(a) Regularity without hierarchy (b) Floorplan generated from (a) (c) Regularity preserving hierarchy (d) Floorplan generated from (c)

Figure 2: Example floorplans with and without hierarchy constraints.

connectivity information is not considered, resulting in floorplans with good area metrics but poor wire length.

DeFer minimizes area and wire length, and uses hierarchy to floorplan efficiently. In Fig. 1(c), we disabled compaction in order to easily visualize the effects of hierarchy, but it was enabled for obtaining the results in Table 3. Because of hierarchy, the floorplan is divided in 4 quadrants, with each quadrant also divided in 4 quadrants, and so on. However, small differences in the floorplans used for every quadrant prevent reusing the same design for all sub-quadrants. The construction of hierarchy from connectivity information results in a 2% area increase compared to CompaSS, but generates a significantly reduced wire length.

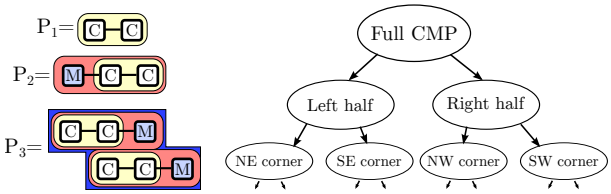
HiReg, on the other hand, constructs a floorplan that exploits the regularity and hierarchy inherent to a tiled CMP design. It is able to extract additional regularity by grouping cores inside tiles in blocks of 2. This tiled structure is discovered despite HiReg not having any previous knowledge of the interconnect topology. The use of hierarchy and regularity causes an additional 4% area increase over DeFer results, but generates a 20% reduction in wire length. Because of regularity, the entire CMP can now be constructing by replicating the two types of tiles. At the same time, two cores in every processing tile can be constructed by replication, resulting in significant design time savings.

2.1 Discovering regularity

In order to create regular floorplans, HiReg automatically finds repeating patterns in the input netlist. We define a pattern as a subgraph from the netlist. We consider a pattern P to be repeated if there is at least one additional subgraph in the netlist isomorphic to P. We call all the repetitions of P the *instances* of P.

An example of the way HiReg extracts regularity is shown in Fig. 4(a). The initial netlist contains 4 instances of the same pattern (composed of C, $L2$ and r each). After identifying this pattern, HiReg *compresses* the graph, replacing every instance with a new vertex, representing the compressed instance. The process iterates until no additional patterns can be found.

Because of this iterative process, the result of regularity discovery is actually a directed acyclic graph (Fig. 4(b)). In this DAG, there is a leaf node (with no exit edges) for each component type in the netlist. Every other node represents a pattern. An edge between two patterns P_a, P_b indicates that pattern P_a contains an instance of P_b. The root pattern (with no entry edges) represents the entire original netlist.



(a) Patterns used in Fig. 2(a) (b) Discovered hierarchy tree

Figure 3: List of patterns and hierarchy tree.

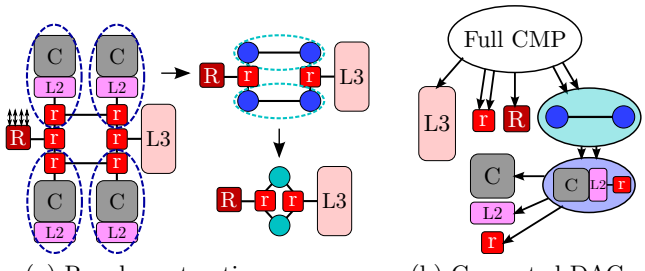
HiReg uses this DAG to apply a divide-and-conquer strategy. Instead of floorplanning the entire netlist, the problem is split into floorplanning every pattern. To ensure regularity, HiReg enforces using the same or similar floorplans for all instances of a pattern, albeit this restriction may be relaxed if better area or wire length results are required.

2.2 Trading off regularity and hierarchy

An important contribution of this paper is the importance of preserving existing hierarchy when discovering regularity.

Our initial approach completely disregarded hierarchy and centered on regularity as defined in Section 2.1. The regularity extraction process is primarily based on local decisions and lacks a global vision of the entire netlist. By centering on regularity only, the results may contradict existing design hierarchy, which can be counterproductive to the goal of reducing design time.

A visual example is shown in Fig. 2. This example shows a CMP design identical to the one in Fig. 1, containing processing (C) and memory controller (M) tiles. In (a) and (b) floorplanning is performed using discovered regularity only,



(a) Regular extraction process (b) Generated DAG

Figure 4: Regular extraction process and example generated DAG.

without preserving hierarchy. (c) and (d) show the results of floorplanning using both regularity and hierarchy discovery.

The set of repeating patterns that have been extracted to construct (a) are shown in Fig. 3(a). Because the regularity discovery process lacks global vision of the netlist, it discovers a set of patterns that break the natural tile hierarchy of the CMP. Despite patterns P_1 and P_2 being frequent patterns, using these to compress the netlist limits further extraction of regularity. The only remaining patterns left after compressing the netlist with P_1 and P_2 are combinations that do not respect the mesh topology, such as P_3. P_3 groups a non-rectangular set of tiles. In these situations, good area and wire length metrics cannot be obtained if the same layout must be strictly replicated for all instances of P_3. Thus, the regularity of the design is compromised, as shown in Fig. 2(b).

In HiReg, existing netlist hierarchy is automatically discovered using recursive graph bisection (similar to [16], described in Section 3.1). Alternatively, it may be provided by the designer. Using recursive bisection, we can build a binary tree representing existing hierarchy on a design and guide the regularity discovery process so that no pattern instances can cross boundaries delimited by hierarchy. In Fig. 2(c), no pattern instance was allowed to extend to more than one CMP quadrant, based on the hierarchy tree discovered by bisection (Fig. 3(b)) which separates the four CMP quadrants. This restriction reduces the number of P_1 instances, but eventually allows a larger number of more regular patterns to be found. In this way, maintaining hierarchy adds a global vision to the regularity extraction process.

3. ALGORITHM

The algorithm can be divided in 5 stages, as seen in Fig. 5. The first 3 stages of the algorithm perform *hierarchy discovery* and *regularity discovery* based on the input netlist. During *hierarchy tree collapsing*, trade-offs between hierarchy and regularity are explored. Instead of generating a single regular hierarchy, between stages 3 and 4 we store a set of candidate regular hierarchies, delaying the selection on which hierarchy is most optimal until after all hierarchies have been evaluated.

The latter two stages perform actual floorplanning for all the candidate hierarchies. Stage 4 (*bounding curve construction*) enumerates all possible floorplans for each of the hierarchies, and stores the outlines efficiently as a single bounding curve. After this stage, the outlines for each possible floorplan are known and the designer can select a smaller subset based on physical metrics such as aspect ratio. Stage 5 (*floorplan construction*) constructs the selected floorplans.

Algorithm 1 contains a formal definition of this multiple stage flow, showing all 5 stages. The stages will be explained in detail during this section.

3.1 Hierarchy discovery

The first stage discovers the existing hierarchy in the input design. This is performed in order to ensure that an existing high-level topology in the design is preserved. Respecting existing design hierarchies is not only desirable from a reusability point of view, but also generates results with improved wire length when compared to results that create layouts which ignore the existing topology.

The method proposed in this section is based on hypergraph partitioning, an extension of the methodology proposed in [16].

Algorithm 1 General overview of the algorithm

```
function RegularFloorplanning(G)
    CandidateHierarchies ← ∅
    HierarchyTree ← HierarchyDiscovery(G)
    for threshold in {0 ... n} do
        CollapsedHierarchyTree ←
            CollapseHierarchy(HierarchyTree, threshold)
        RegularHierarchyDAG ←
            RegularityDiscovery(CollapsedHierarchyTree)
        append RegularHierarchyDAG to CandidateHierarchies
    ▷ CandidateHierarchies contains a set of candidate
      regular hierarchy DAGs
    Γ ← empty bounding curve
    for all RegularHierarchyDAG ∈ CandidateHierarchies do
        Γ ← Γ ∪
            ConstructBoundingCurve(RegularHierarchyDAG)
    ▷ Γ contains the bounding curve of all possible floor-
      plans for all of the candidate hierarchies
    SelectedPoints ← select desired outlines from Γ
    return ConstructFloorplans(SelectedPoints)
```

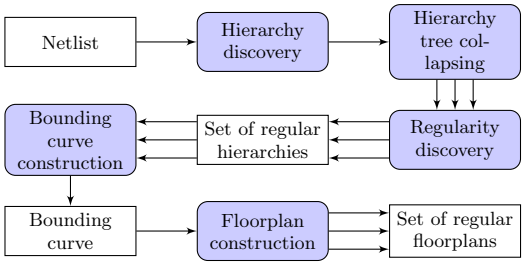


Figure 5: High-level flow of the algorithm.

We assume that the input netlist, represented as the hypergraph G, has a natural number of partitions. For example, a CMP with 8×8 tiles would have 64 natural partitions. A 63-way or 65-way partition would result into more interconnections between the different subcircuits than the natural 64-way partition. The goal of the algorithm is to discover these natural partitions.

The input netlist G is partitioned into two smaller circuits, minimizing the total number of interconnections between the two partitions, as long as both partitions have an area imbalance $\leq \frac{2}{3}$. Such imbalance margin allows handling values of k that are not power of 2, by dividing the circuit into a partition with $\frac{2}{3}$ of the area and one with $\frac{1}{3}$. Between multiple bipartitions with the same number of interconnections, the most balanced partition is preferred.

The process is recursively applied to the two generated partitions, until all partitions contain a low enough number of blocks so that further bisectioning is not required (*MinSize*). This process is described in Algorithm 2.

During the process a binary tree is created where every leaf node is a subcircuit (with a number of components $<$ *MinSize*), and every other node is a bipartition, the root node being a bipartition of the input netlist G. We call this tree the *hierarchy tree* (Fig. 3(b)).

3.2 Hierarchy tree collapsing

The trees generated during hierarchy discovery are used to constrain the regularity discovery procedure and ensure that existing circuit hierarchy is preserved. However, strictly preserving all hierarchy would prevent the floorplanner from finding regularity, as discussed in Section 2. Thus, this stage

Algorithm 2 Hierarchy discovery algorithm

```
function HierarchyDiscovery(G)
    ▷ G is the input netlist
    if |G| < MinSize then
        ▷ Trivial case if there are too few elements left
        return G
    G1, G2 ← bipartition of G minimizing number of
                edges between G1 and G2 with
                AreaImbalance(G1, G2) ≤ 2/3
    T1 ← HierarchyDiscovery(G1)
    T2 ← HierarchyDiscovery(G2)
    return CreateTree(T1, T2)

function AreaImbalance(G1, G2)
    return Max(Area(G1),Area(G2)) / (Area(G1)+Area(G2))
```

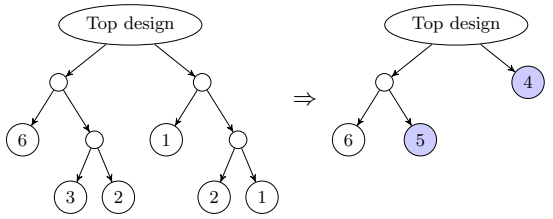


Figure 6: Tree collapsing with a threshold of 4 mm^2. Labels in leaf nodes indicate block area (in mm^2).

generates hierarchies that have been *relaxed*, giving more flexibility to the posterior regularity discovery procedure.

It is often the case that only the high-level hierarchy is significant. For example, it is important to ensure that tile boundaries are honored in a CMP, but the contents of the tiles themselves often have a less well defined hierarchy, and a reduced connectivity impact if such hierarchy is not preserved. For this reason, it is preferable to relax hierarchy at the leaves of the hierarchy tree.

Algorithm 3 shows the details of the algorithm. *threshold* is an input parameter, specified as an absolute area value. For tree nodes where the total area is less than this threshold, the tree node is *collapsed*: all of its descendants are enumerated, combined into a single subcircuit, and the tree node is replaced by the new leaf subcircuit node. Thus, the resulting collapsed tree will have fewer nodes than the input hierarchy tree, with larger leaf nodes containing more components. See Fig. 6 for a visual example.

Multiple possible hierarchies are generated by this process by automatically testing for several values of the *threshold* parameter. This way, the trade-off between hierarchy and regularity is explored. The selection of the best threshold is thus deferred until floorplans are generated and area, wire length and additional metrics are available.

Algorithm 3 Tree collapsing

```
function CollapseTree(HierarchyTree, threshold)
    ▷ any nodes representing parts of the netlist with
      less area than threshold are collapsed
    if BlockArea(Tree) > threshold then
        ▷ Keep this node intact; continue walking the tree
        T1 ← CollapseTree(HierarchyTree.leftChild,
                          threshold)
        T2 ← CollapseTree(HierarchyTree.rightChild,
                          threshold)
        return CreateTree(T1, T2)
    else
        T ← combine all descendants of H into single node
        return T
```

3.3 Regularity discovery

An essential stage in the floorplanning algorithm is finding repeated patterns of blocks in the netlist. The methodology proposed in this work is based on the ideas of frequent subgraph discovery, a popular research area within the domain of data mining [10]. The goal of subgraph discovery is to identify repeated subgraphs in a graph.

We consider two distinct subgraphs G_1, G_2 of a graph G to be a repetition if they are isomorphic. In such case, we call both G_1 and G_2 *instances* of the same repeating *pattern* P. In our formulation, each type of block in the netlist (core, router, memory module, etc.) has a different *label*. Two vertices of a graph are considered to be isomorphic only if they have the same label.

The algorithm is shown in Algorithm 4. RegularityDiscovery starts from a *FlattenedTree* as input. At every iteration of the inner loop the most frequent pattern is found, and then the netlist graph is compressed with all the instances of such graph. This iterative process generates the regularity DAG as seen in Fig. 4.

The outer loop ensures that the existing hierarchy indicated by *FlattenedTree* is preserved. Instead of finding the most frequent pattern of the entire netlist, we initially limit our search to the subcircuits in nodes *FlattenedTree* whose depth is equal to the maximum depth of the entire tree.

Only if no repeating patterns are found in those subcircuits, the search proceeds by enlarging the search are to include all subcircuits in nodes with fewer depth, decreasing the minimum allowed depth (*CurMinDepth*) by one. This way, the search algorithm ensures that patterns that are fully contained inside the partitions marked by hierarchy boundaries are preferred before patterns that do not.

Procedure FindMostFrequentPattern implements frequent subgraph discovery based on [10]. It is based on a constructive beam search model [12]. At every iteration, we keep a list L of candidate patterns. This list is initialized with all trivial patterns of size 1 (that is, every subgraph with a unique label). At every iteration, every pattern P in L is tested to check which of its instances can be extended by including an adjacent vertex. Each possible extension is stored in L_{new}. The extended patterns in L_{new} are sorted by their Value and only a subset of them is selected according to their best value. The number of surviving patterns is determined by b (beam width). The algorithm finishes when no further extensions for any pattern in L can be found.

The Value function is used to discriminate between valid patterns when more than one is found for a given graph G. In HiReg the following function is used:

$$\text{Value}(P) = \text{“number of instances of } P \text{ in } G\text{”} \times |G| + |P|$$

This value ensures all patterns are ordered firstly by their frequency. When comparing two patterns with the same number of repetitions, the pattern with the largest vertex count ($|P|$) is preferred.

3.4 Bounding curve construction

In this stage, a bounding curve is constructed for each one of the regular hierarchy trees discovered by the previous stage. The bounding curve is constructed by a post-order walk in the hierarchy tree. This is similar to the techniques used in other hierarchical floorplanners [4, 16].

Every regular hierarchy tree is a directed acyclic graph where every non-leaf node is a subcircuit representing a reg-

Algorithm 4 Frequent subgraph discovery

```
function FindMostFrequentPattern(G)
    G ← input graph
    L ← {∀ label l ∈ G : a subgraph with a single node
         v ∈ G with label(v)= l}
    b ← beam width
    P_best ← Front(L)
    while ¬ Empty(L) do
        L_new ← ∅
        for all P ∈ L do
            for all vertex u ∉ P adjacent to v ∈ P do
                L_new ← L_new ∪ Extend(P, u)
        Sort(L_new by descending Value())
        L ← first b elements of L_new
        if Value(Front(L)) > Value(P_best) then
            P_best ← Front(L)
    return P_best

function RegularityDiscovery(CollapsedTree)
    PatternList ← ∅
    CurMinDepth ← maximum depth of CollapsedTree
    while CurMinDepth ≥ 0 do
        G ← contents of all nodes from CollapsedTree
             with depth >= CurMinDepth
        repeat
            P ← FindMostFrequentPattern(G)
            append P to PatternList
            G ← Compress(G, P)
        until no repeating patterns in G
        CurMinDepth ← CurMinDepth −1
    return PatternList
```

ular pattern, and and edge between P_1 and P_2 indicates that P_1's definition contains an instance of P_2. Only once the bounding curves for every children of a pattern P have been constructed the algorithm can proceed to construct the bounding curve of P itself.

To construct the bounding curve of a pattern P, two different search strategies are used depending on the number of blocks n. For patterns where n is less than a threshold N, an exhaustive branch-and-bound search algorithm is used. This algorithm explores every possible slicing floorplan.

If $n \geq N$, a more efficient heuristic search based on simulated annealing and slicing trees is used. This heuristic search generates a much reduced number of results than the branch-and-bound approach, but is capable of handling patterns with a much larger number of components. The threshold N depends on the specifications of the host computer, such as the amount of available memory.

The bounding trees for each hierarchy tree are combined into a single bounding curve that represents the outlines of all floorplans found. Because many hierarchy trees will contain similar sets of patterns, HiReg uses memoization in order to decrease the runtime of this stage.

3.5 Floorplan construction

After selecting a subset of points from the final bounding curve (Γ), the final stage of the algorithm constructs floorplans starting from the outlines specified by these points.

A single point in a bounding curve can represent multiple floorplans with the same outline, including floorplans that only differ in mirroring and simple block swapping (but also floorplans with entirely different layouts that happen to share the same outline). Because the outline is fixed, the selection of these floorplans cannot affect whitespace, but it may have a significant impact on the wire length and regularity metrics.

The algorithm in Algorithm 5 implements a greedy search that finds a combination of floorplans for a selected point P of the bounding curve that minimizes a given cost function comparing wire length and regularity. By manipulating the cost function, the designer is able to guide the search to either enforce more regular floorplans, or on the other hand prefer floorplans with increased connectivity quality.

An important technique used during this process is *terminal propagation* [7]. When selecting a floorplan for a pattern T, we know the layout and positions of all instances of other subpatterns $t_1, t_2, \ldots, t_n$ contained in T. Thus, for all nets that have a terminal in a child subpattern t_i, but also have other terminals in any other subpatterns of T, the algorithm can propagate the approximate terminal positions of those terminals outside t_i. This allows calculation of the wire length when selecting floorplans for t_i, even for nets external to t_i.

Algorithm 5 Floorplan construction

```
function ConstructFloorplans(P, T)
    ▷ P is the selected point of Γ
    ▷ T is the hierarchy tree that was used
    F ← floorplans from Γ with size P
    for all t in Children(T) do
        propagate terminal positions from F to t
        p ← shape of t in F
        F_t ← ConstructFloorplans(p, t)
    select combination of F_1, F_2, ..., F_n that minimizes
           Cost(F, F_1, F_2, ..., F_n)
    expand F with F_1, F_2, ..., F_n
    return F

function Cost(F, F_1, F_2, ..., F_n)
    return (Wirelength(F_1)+Wirelength(F_2)+...) /
           (number of floorplans in F_1, F_2, ... with the same layout)
```

To increase floorplan quality or in order to generate more than one result, HiReg combines this algorithm with the technique of beam search [12]. Instead of keeping a single current solution, the b best solutions are kept.

4. EXPERIMENTAL RESULTS

We implemented HiReg in C++ and tested it on a set of design examples. All the experiments in this section were run on a Intel Xeon 2.8Ghz CPU with 32GB of RAM. While the implementation can make use of multiple cores, it was limited to a single thread for fair comparisons. METIS [9] was used to generate the graph bisections required for hierarchy discovery.

Since there is little previous work on regularity-constrained floorplanning for multi-processors, there are not many available benchmarks designed to compare the quality of regular floorplans. Commonly used floorplanning public domain benchmarks are mostly from old designs that do not contain much regularity. Thus, during this section, we will use artificial benchmarks based on many-core CMP designs.

It is hard to give a numerical metric for regularity in a floorplan. For HiReg generated floorplans, we can provide an estimation of regularity based on the regularity DAG used to generate them. Every node in the DAG with multiple input edges represents a sublayout that has been replicated. Thus, a regularity metric can be built by comparing the area of all nodes in this DAG with an expanded version where none of the layouts are replicated:

$$Regularity = 1 - \frac{area\ of\ DAG}{area\ of\ equivalent\ expanded\ tree}$$

Because other tools do not target the creation of regular floorplans, we cannot provide similar regularity metrics for non-HiReg floorplans.

4.1 Heterogeneous tiled CMP example

We start this section by mentioning the results presented during Section 2, the heterogeneous tiled CMP. We also show the differences in wire length from a floorplanning that preserves hierarchy versus one that does not.

The netlist used represents a CMP containing 64 tiles, with 48 tiles being processing tiles, containing 4 cores each, and 16 memory controller tiles. The tiles are arranged according to the *diamond* pattern from [1]. The benchmark has a total of 816 blocks. The physical information for these blocks is shown in Table 4.

For this example CMP configuration, HiReg generates a floorplan (Fig. 2(d)) with 12.6% whitespace and a HPWL of 516 m. Up to 81% of the floorplan area is regular. This floorplan was used by combining hierarchy and regularity. HiReg automatically prefers floorplans where hierarchy is preserved to around the tile level, as those provide the best regularity with minimal loss in other metrics. For comparison, the floorplan in Fig. 2(b), which was created without hierarchy constraints, has a slightly worse whitespace (13.5%), worse regularity (66.6% of area) and a much worse HPWL (1076 m). These hierarchy-less floorplans were explored but discarded by HiReg because of the lower metrics.

When compared to other tools (Table 3), HiReg provides results that are competitive in wire length but slightly less in area. We configured DeFer to optimize for area and wire length given a maximum aspect ratio constraint of $\frac{4}{5}$. CompaSS was configured to optimize for area within the same aspect ratio constraint. HiReg used 15.4 seconds to generate that example floorplan. 13% of the time was spent during hierarchy and regularity discovery, 27% of the time was spent floorplanning all the discovered patterns, and 60% generating the final floorplans. Both DeFer and CompaSS took less than one second to generate the floorplans.

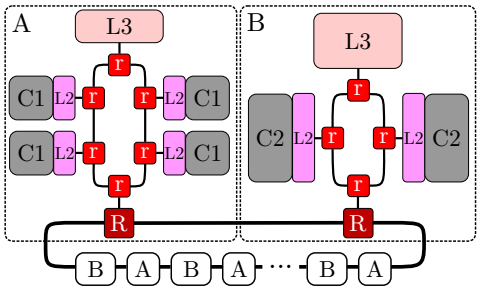


Figure 7: Netlist used for the scalability and quality experiments.

Table 4: Physical information for Fig. 7.

Component	Area	Aspect ratio
Global ring routers (R)	0.27 mm^2	1
Clusters of type A		
4 × Core ($C1$)	1.38 mm^2	0.8 or 1.25
4 × L2 cache	1 mm^2	0.5 ÷ 2
1 × L3 cache	3 mm^2	0.5 ÷ 2
6 × Local ring router (r)	0.27 mm^2	1
Clusters of type B		
2 × Core ($C2$)	3.75 mm^2	0.8 or 1.25
2 × L2 cache	2 mm^2	0.5 ÷ 2
1 × L3 cache	6 mm^2	0.5 ÷ 2
4 × Local ring router (r)	0.27 mm^2	1

4.2 Scalability and floorplan quality

This experiment measures the loss of optimality in area and wire length caused by the use of regularity, as well as measure the execution time required for the implemented algorithm. We compare HiReg, configured to optimize for maximum regularity, to DeFer, configured to optimize for both area and HPWL.

All the test cases were generated based on the configuration shown in Fig. 7. This configuration is a *ring of rings*, where a global ring connects a set of *clusters* of two alternating types. The physical characteristics are detailed in Table 4. For this example, every net contains 1024 wires, and the wire pitch is 0.1µm. This configuration is used in this testcase because the total number of clusters in the ring can be easily parameterized, providing multiple testpoints with different numbers of components.

Figure 8 shows the results of the comparison. For a highly regular netlist such as the one used in this experiment, the runtime growth of both hierarchical floorplanners (proposed and DeFer) is close to linear. However, a purely hierarchical floorplanner such as DeFer is still much faster. On the other hand, the results show that both the whitespace and HPWL of floorplans generated by HiReg are comparable to the results provided by DeFer.

4.3 Trading off hierarchy and regularity

Figure 9 shows two different potential configurations for the ring configuration described in Fig. 7. These two configurations differ in the organization of the global ring. In configuration (a), clusters A and B appear in alternating order in the global ring. Configuration (b) contains the same

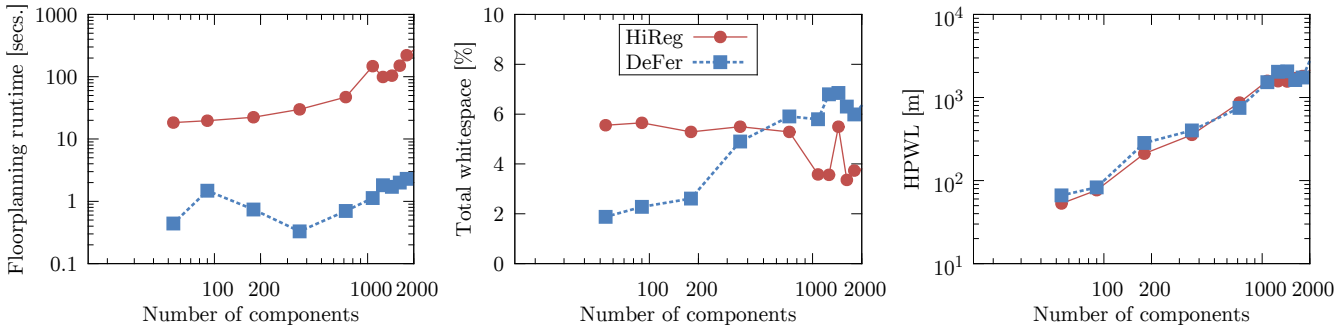


Figure 8: Comparison of runtime, area and wire length.

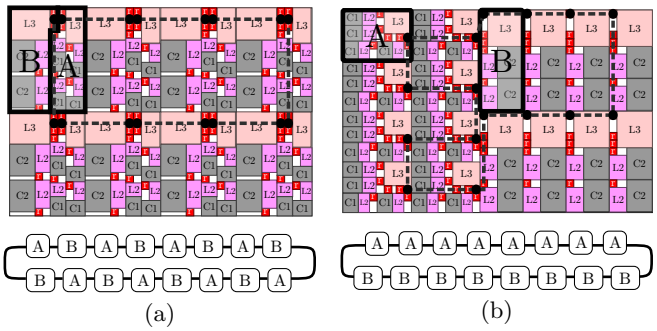


Figure 9: Regular floorplans for two different ring configurations.

clusters but configured so that clusters of the same type are grouped together. The net for the global ring is shown in both figures as a dashed line.

For Fig. 9(b), HiReg generates a regular floorplan where the different cluster types are strictly separate. The floorplan provides good whitespace metrics because the similar clusters are being packed into arrays. However, in case (a), such packing would not be possible without impacting wire length. Therefore, the best floorplan found by HiReg relaxes hierarchy a bit, and combines the two cluster types into a single repeating pattern, providing better packing.

HiReg can also be configured to further relax regularity when improved wire length or other metrics are preferable. In figure 10, we show three different floorplans for the configuration in Fig. 9(b). Figure 10(b) and (c) are generated by varying the cost function mentioned from Section 3.5, while (a) is generated using DeFer. The plot shows how by trading off regularity, HPWL can be improved, approaching the HPWL of the floorplan generated by DeFer for the same configuration. As DeFer does not generate regular floorplans, the plot uses only its HPWL as baseline for comparisons.

5. CONCLUSIONS

This work has presented a new floorplanning method that generates regular floorplans while preserving the inherent regularity of the design. The method is specially suited for CMPs with many cores and can handle systems with heterogeneous tiles. The method delivers layouts with high regularity and acceptable area, and also reduces wire length when compared to other hierarchical approaches.

6. REFERENCES

[1] 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. In *Proc. of the Annual International Symposium on Computer Architecture*, pages 451–461, 2009.

[2] F. Balasa and K. Lampaert. Symmetry within the sequence-pair representation in the context of placement for analog design. *IEEE Trans. Comput.-Aided Design Integrated Circuits*, 19(7):721–731, Nov. 2006.

[3] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proc. of the Annual International Conference on Supercomputing*, pages 187–198, 2006.

[4] H. H. Chan and I. L. Markov. Practical slicing and non-slicing block-packing without simulated annealing. In *Proc. of the Great Lakes Symposium on VLSI*, pages 282–287, 2004.

[5] X. Chen, J. Hu, and N. Xu. Regularity-constrained floorplanning for multi-core processors. *Integration, the VLSI Journal*, 47(1):86–95, 2014.

[6] W. Dai. Hierarchical placement and floorplanning in BEAR. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(12):1335–1349, 1989.

[7] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(1):92–98, Jan. 1985.

[8] G. G. Faust, R. Zhang, K. Skadron, M. R. Stan, and B. H. Meyer. ArchFP: rapid prototyping of pre-RTL floorplans. In *Proc. of the International Conference on VLSI and System-on-Chip*, pages 183–188, 2012.

[9] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, Dec. 1998.

[10] N. S. Ketkar, L. B. Holder, and D. J. Cook. Subdue: compression-based frequent pattern discovery in graph data. In *Proc. of the Open Source in Data Mining Workshop*, pages 71–76, 2005.

[11] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *IEEE Computer*, 38(11):32–38, 2005.

[12] S. M. Rubin and R. Reddy. The Locus model of search and its use in image interpretation. In *Proc. of the International Joint Conference on Artificial Intelligence*, volume 2, pages 590–595, 1977.

[13] K. Samadi. *Accurate estimators and optimizers for networks-on-chip*. Ph.D., UC San Diego, 2010.

[14] D. Sreenivasa Rao and F. J. Kurdahi. On clustering for maximal regularity extraction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(8):1198–1208, Aug. 1993.

[15] R. Wang and N. Shah. Scalable hierarchical floorplanning for fast physical prototyping of systems-on-chip. In *Proc. of International Symp. on Physical Design*, pages 187–192, 2012.

[16] J. Z. Yan and C. Chu. DeFer: deferred decision making enabled fixed-outline floorplanning algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(3):367–381, Mar. 2010.

[17] T. T. Ye and G. D. Micheli. Physical planning for on-chip multiprocessor networks and switch fabrics. In *Proc. of the International Conference on Application-Specific Systems, Architectures, and Processors*, pages 97–107, June 2003.

[18] F. Y. Young and D. F. Wong. How good are slicing floorplans? In *Proc. of the International Symposium on Physical Design*, pages 144–149, 1997.

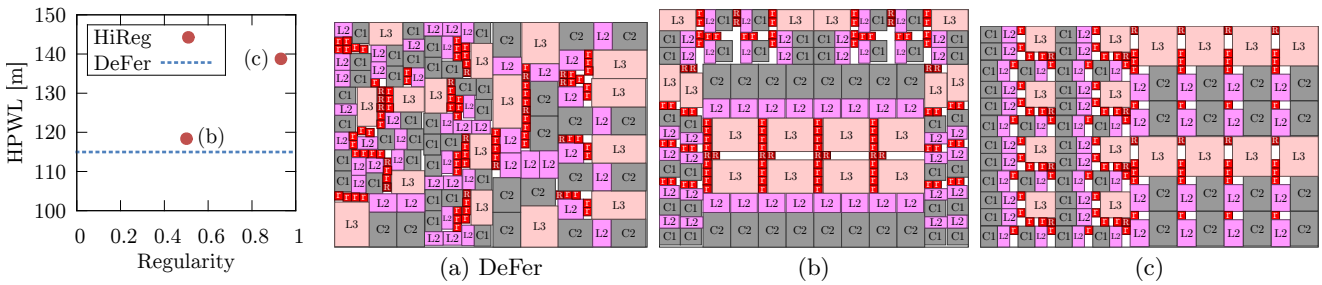


Figure 10: Trading off regularity and wire length in configuration described in Fig. 9(b)