Parallel Standard Cell Placement on a Cluster of Workstations*

Faris H. Khundakjie, Patrick H. Madden
Nael B. Abu-Ghazaleh and Mehmet Can Yildiz
State University of New York at Binghamton
Computer Science Department
nael@cs.binghamton.edu

Abstract

In this paper we report experiences on a parallel implementation of a standard cell placement algorithm on a cluster of myrinet connected PCs. The proposed algorithm is based on a recently developed placement tool (called *Feng Shui*) that extends recursive bisection placement to incorporate global aspects of the design. This is achieved using an efficient and novel optimization called iterative deletion. We investigate several algorithmic and system-level optimizations. Contrary to previous attempts at parallelizing placement algorithms, initial experimental results show significant performance improvement with small reduction in the placement quality. Furthermore, the reduction in the placement quality does not increase with the number of processors.

1 Introduction

With advances in VLSI fabrication technology, the size of circuits of interests has become extremely large and is continuously expanding. Physical design automation tools are needed to aid in design and layout of such circuits. However, the size of the circuits presents a similar challenge to the design automation tools: they must be able to provide good quality layouts with acceptable run times. In this paper, we consider VLSI standard cell placement – an important and difficult problem in physical design automation. The placement impacts circuit areas and wire delays profoundly; a poor placement may prevent a circuit from operating at an acceptable speed, or make too large for the design floor-plan. Furthermore, if the placement algorithm has high complexity, we will be unable to obtain a placement in an acceptable amount of time. This in turn may force us to sacrifice the placement quality to achieve faster placement time.

Parallel processing offers the promise of increasing the performance and capacity of placement tools, enabling them to provide better solutions in faster times. The emergence and commercial success of clustering technologies is perhaps the most exciting development yet in the field of parallel processing: it finally allows scalable cost-effective parallel processing machines to be built [1, 3, 29]. Clusters approach the performance of custom parallel machines by using high-performance Local/System area networking technologies and standards (such as Myrinet [5], SCI [10,

*This research was supported in part by NSF under awards CCR-9988222 and EIA-991099

17] and others [4, 36, 37]) and low-overhead user-level space communication protocols (such as the Basic Interface for Parallelism (BIP) [15], Illinois Fast Messages (IFM) [28] and others [11, 27]). Because they use commodity components, clusters are affordable, scalable and easy to build [32].

This paper presents our experiences in parallelizing a state-of-the-art placement tool on a cluster of myrinet connected workstations. The algorithm uses standard iterative partitioning but augments that with an iterative deletion step that incorporates global aspects of the partition in addition to the local optimizations obtained through the iterative bisection step.

The contributions of this paper are the following: (i) the initial sequential algorithm is state-of-the-art both from a run-time complexity and solution quality perspectives; (ii) the parallel implementation provides significant speedup while maintaining very high quality solutions regardless of the number of processors used. In contrast, existing parallel implementations generally suffer significant degradation in solution quality; (iii) most published solutions are specific to an architecture and are not directly portable to a cluster environment. To our knowledge, this is the first study that investigates this important problem on a modern cluster environment; and (iv) we investigate several algorithmic optimizations as well as system-level tradeoffs in the implementation. This includes a study of the effect of the Myrinet network [5] running the BIP message passing library relative to using 100 Mbit/sec switched Ethernet for communication.

The remainder of this paper is organized as follows. Section 2 the placement problem and related work. Section 3 discusses the sequential placement algorithm in more detail. Section 4 presents the details of the parallel implementation. Section 5 presents the experimental setup and results. Finally, Section 6 presents some concluding remarks.

2 Overview and Related Work

Placement algorithms have been extensively studied; objectives such as area minimization, wire length minimization, and timing optimization are common. For all but trivial problem sizes, we have only heuristic methods; optimization is usually performed on only a small subset of the circuit at any given time. If we perturb a subset of circuit elements, we can optimize a placement from a *local* perspective, but have no guarantee that our modifications are appropriate from a *global* perspective.

We consider the problem of standard cell placement: the objective is to place rectilinear circuit elements (cells) into one or more horizontal rows, minimizing total wire length. There are a number of established approaches to the placement problem. For a comprehensive survey, the reader is referred to the following paper [33]. **Force Directed** or **LP** based approaches repeatedly solve systems of equations, determining cell locations iteratively (for example,

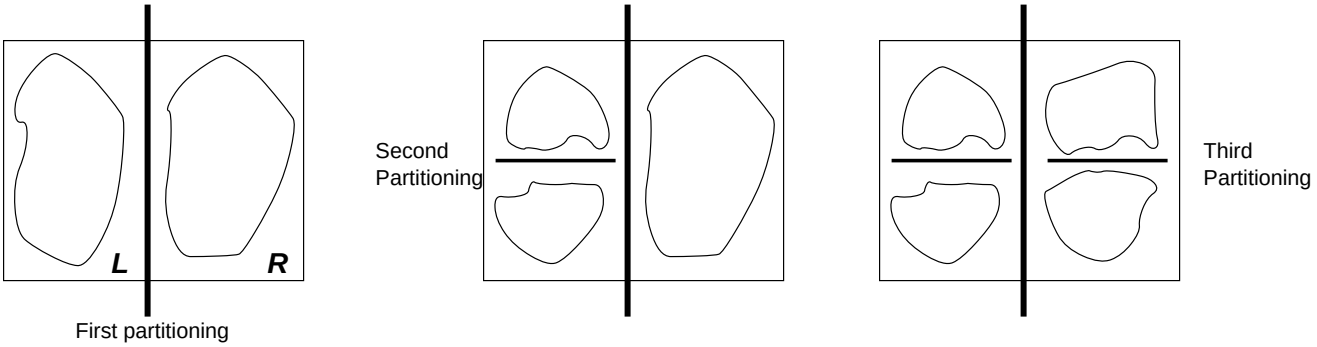

Figure 1: Order of bisections in a top-down recursive approach. The partitioning of *L* influences the solution for *R*, and vice versa.

[13]). This approach is popular in commercial placement tools. **Simulated Annealing** based approaches obtain cell placements by swapping positions of cells randomly, guided by a probabilistic acceptance function. A number of current commercial placement engines utilize this approach; efficient cost estimates allow the consideration of large numbers of intermediate states. A well known example of this approach is TimberWolf[35]. **Partitioning** based approaches determine cell locations by recursively dividing an initial area (region) with successive bisections or quadrisections. This approach has become more attractive recently; advances in partitioning research have provided a number of fast algorithms which produce extremely good results. This is the approach used in this paper.

Breuer[6] utilized repeated graph bisections to obtain a circuit placement; this approach is shown in Figure 1. The bisections divide the circuit netlist into a hierarchy of cells, with the resulting hierarchy roughly mapping into a rectilinear grid. Dunlop and Kernighan[12] extended this approach, through the use of an improved partitioning method[20]. Moving beyond simple bisections, Suaris and Kedem[34] explored the use of quadrisection (a four way partitioning). Huang and Kahng[16] also apply quadrisection, utilizing a multi-level clustering based partitioning algorithm, and considering minimum spanning tree lengths, rather than the simple min-cut metric.

Dunlop and Kernighan [12] also introduce *terminal propagation*. When partitioning a region, we can expect a number of connections to be required to cells or pads outside of the region. Terminal propagation provides a simple method to insert fixed "dummy" vertices, so that the partitioning considers these external connections (Figure 2). With terminal propagation, the partitionings of regions become interdependent; if we begin with two regions, *L* and *R*, and partition *L* first, this impacts the optimal solution for *R*. Partitioning *R* first might result in a different solution, and neither of these might be globally optimal, even if the individual partitionings were. To address the order dependence of the partitioning, both [34] and [16] employ repeated partitioning at each level. We might wish to partition *L*, followed by *R*, and then partition *L a second time*. Repeated partitionings do not, however, change a local optimization process into a global one.

Because of the computational complexity of placement, there has been significant interest in parallelizing place-

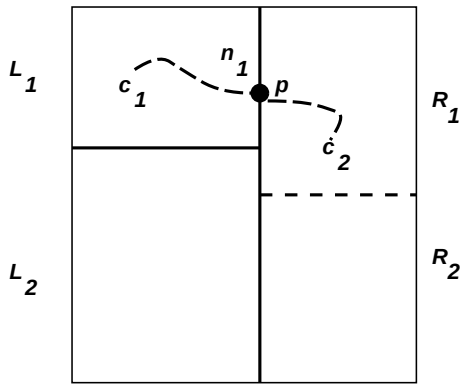

Figure 2: Terminal propagation as performed by Dunlop and Kernighan. When recursively partitioning a netlist, we can insert *dummy terminals* to influence the partitioner. If a net spans more than one region, the location of dummy terminals can improve the placement quality.

ment algorithms since the 1980's (see for example [8, 9, 18, 21, 22, 23, 24, 26]; a good summary is available here [2]). These studies have experimented with parallelizing most of the placement algorithms on a variety of parallel architectures. Most of these studies are quite old and are specific to the architectures that were used to test them; it is difficult to compare performance directly with them. In addition, the objective functions that the different studies optimize are generally different. Furthermore, research in placement suffers from the lack of uniform metrics for reporting the results – wire lengths are reported, but the results can vary by a multiplicative factor depending on the underlying assumptions (such as cell spacing and the method of measuring wire length). This makes it difficult to fairly compare the algorithms, even when the same models are used. However, while direct comparison is difficult, comparison of relative measures between the sequential and parallel versions of each algorithm (such as speedup, and quality degradation) are still possible. In the following paragraph we will overview some of the most recent of these works. In Section 5 they are also reviewed as we compare our results to them.

Banerjee's research group has done the most extensive work in the area of parallel placement algorithms. For example, within the ProperPLACE CAD tool they discuss a parallel placement algorithm based on simulated annealing [21]. In contrast to other parallel placement implementations, they work with an abstract parallel machine model allowing the implementation to be directly ported across different architectures. They study shared memory and distributed memory implementations. The speedup across different implementations were reported on the ISCAS benchmarks. Koide et al present a parallel implementation of their POPINS timing driven placement tool [23]. They use a bisection approach similar to our own, with non-linear programming used in a second phase optimization. The parallel implementation uses shared memory. In this study, the drop of quality in the parallel implementation was significant (an average of 14%).

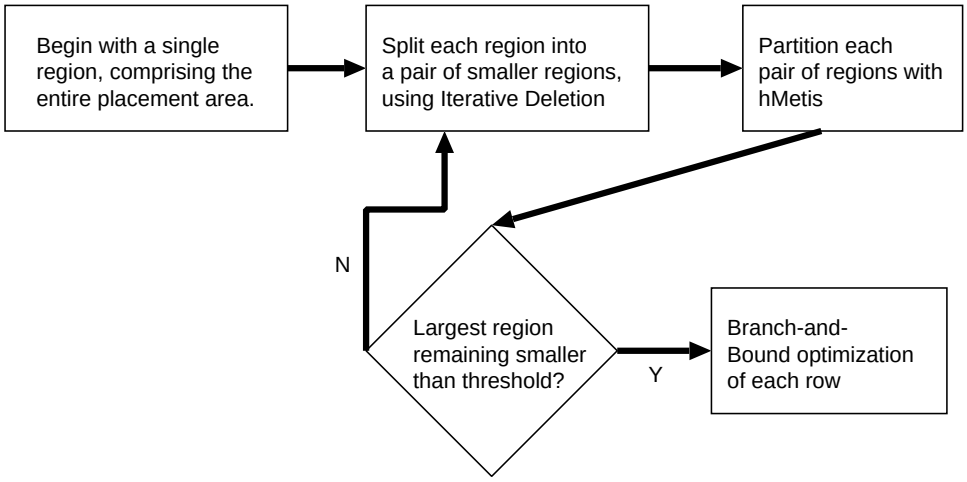

Figure 3: Flowchart of the placement approach. The partitioning and branch-and-bound improvement steps are well known. New to the approach is the pre-processing performed by *iterative deletion.*

3 Algorithm Formulation

This paper considers parallelizing a placement tool (called *Feng Shui* [38]) developed by two of the authors. *Feng Shi* adapts a recently presented partitioning approach for *k*-way partitioning. It differs from that solution in that it uses a technique called *iterative deletion* [25] to allow some global issues to be captured resulting in improved placement quality at significantly lower cost than *k*-way partitioning. *Feng Shui* integrates a variant of *k*-way partitioning approach into a traditional framework. The general flow of our placement tool is as shown in Figure 3. When faced with large numbers of regions to bisect, *iterative deletion* is applied first (to obtain a good quality global solution), followed by repartitioning of regions with a traditional bisection approach (to improve solution quality from a local perspective). The remainder of this section describes these steps in more detail.

3.1 Bisection

The framework for the placement tool is a textbook implementation of the approach of Dunlop and Kernighan[12]. The circuit is repeatedly divided by either horizontal or vertical cut lines. A recent multi-level clustering based partitioning algorithm hMetis[19], version 1.5.3 is used. At each partitioning, we attempt to obtain a nearly exact bisection if cutting vertically. If the cut line is horizontal (splitting a number of rows), the rows are split as evenly as possible. If the region being bisected contains an odd number of standard cell rows, fixed and weighted dummy vertices are added to allow a nearly exact bisection to be mapped into the available space. The partitioning objective is *min-cut*; the hMetis partitioner attempts to minimize the number of cut hyperedges.

The recursive bisection process divides placement regions into progressively smaller areas, ultimately assigning each cell to a single row, but possibly having several cells remaining within a region. To establish positions for each cell, the cells are ordered by region location within each row; the cells are packed together without spaces or overlap. The positions of cells which were within the same region will be arbitrary at this point; they were not ordered by the partitioning process. To optimize these positions, we apply branch-and-bound reordering, modifying the positions of a small set of consecutive cells in a single row.

Feng Shui allows the specification of a "window size," controlling the number of cells involved in any branch-and-bound optimization. This window passes over each cell row (in order), traveling along each row at steps of half the window size. At each step, the optimal order for cells found under the window is determined. The number of passes over the placement, and the size of the window, are both parameters which can be controlled by the user. In practice, we find that window sizes of 6 to 8 cells, and 4 passes of improvement, are sufficient for good overall performance. Increasing the window size may impact run times substantially (as the complexity of the branch-and-bound procedure is $O(w!)$ worst case, where w is the size of the optimization window). In [7], a number of ways to implement branch-and-bound reorderings efficiently were explored.

3.2 *k*-Way Partitioning

The focus of our work has been on the *global* aspects of the placement problem. With partitioning, we can optimize the number of edges cut within a region effectively, but have no way of knowing if this *local* optimization is appropriate from a *global* perspective. Similarly, our branch-and-bound reordering is also a *local* optimization.

A careful examination of placement by recursive bisection reveals a number of instances where global objectives may be lost. The example in Figure 4 shows a simple case where local optimization is insufficient; there are four regions to bisect, each with two cells. If the problem is approached as a series of independent bisections, a number of configurations which are both stable and suboptimal can be encountered. The sub-optimality of the global solution is not related to the quality of the bisections of each region; simply improving the bisection algorithm will not improve the global configuration.

As we progress through the placement process, the number of regions increases, doubling repeatedly. If there are k regions that are to be split (obtaining $2k$ new regions), the traditional approach is iterative, bisecting a single region at a time. Instead, *Feng Shui* attempts bisection of all regions at the same time, obtaining a solution that is of good quality *globally*. The method used to perform this massive bisection is based on partitioning by iterative deletion[25].

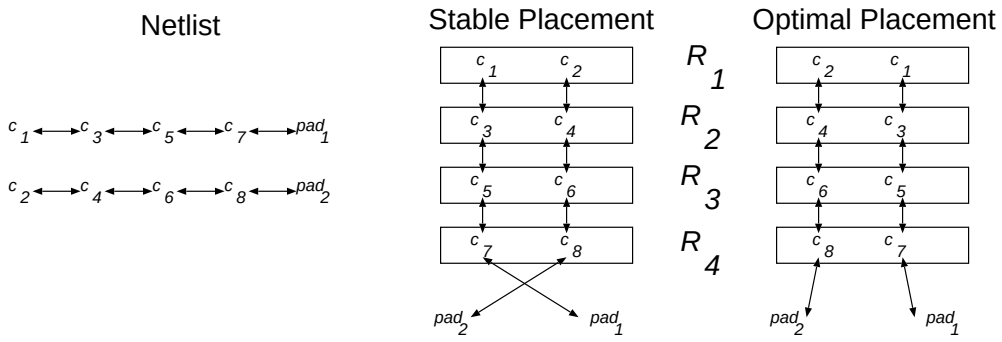

Figure 4: Given the netlist above, we may assign pairs of cells to the regions shown. If we partition or apply branch-and-bound reordering to the four regions, we may be unable to find an optimal solution. If we determine the ordering in region R_1 first, we arrive at a stable and suboptimal solution. Repeated local optimization (through repartitioning) will fail to find the globally optimal solution.

3.3 New Formulation – Iterative Deletion

To capture global objectives effectively, a variant of *multi-way partitioning* is used (rather than as a series of bipartitions). We partition all regions *simultaneously*, with the intermediate state of each region influencing the others. We are concerned with partitioning very large numbers of regions, and our cost objective is wire length rather than *min-cut*. The problem we consider is *given a set of regions (with physical constraints) and a set of elements mapped to these regions, bisect all regions to minimize the resulting bounding-box wire length.* Solution of this problem optimizes the circuit from a *global* perspective. Multi-way partitioning has proven quite challenging[31]; for traditional objectives such as *min-cut*, the greatest success has been obtained with recursive partitioning.

In [25], hypergraph partitioning was considered. A new method based on *iterative deletion* was presented; in this approach, vertices are duplicated, with one instance of each vertex being assigned to a partition. Redundant elements are removed one at a time until no duplicates remain. While the approach was relatively simple, it proved effective in some areas where traditional methods had difficulty. For bipartitioning, cut sizes from a single linear-time pass were comparable to many passes of a traditional FM[14] algorithm. Multi-way cut sizes were superior to a direct flat multi-way partitioning algorithm[31]. For problems with a variety of hyperedge weights, a combination of iterative deletion and FM partitioning proved substantially more effective than FM partitioning alone. The approach is computationally attractive: with integer hyperedge weights, it may be implemented in $O(n)$ time.

Our variation of the iterative deletion approach for placement operates in the following manner. Each cell in a region is assigned to *both* subregions; if there is more than a single instance of a cell, it is considered to be *redundant*. We repeatedly remove redundant cells from subregions which have high utilization, and select the highest cost cell for removal.

In the existing implementation of the placement engine, the cell cost is evaluated based on the center of mass for

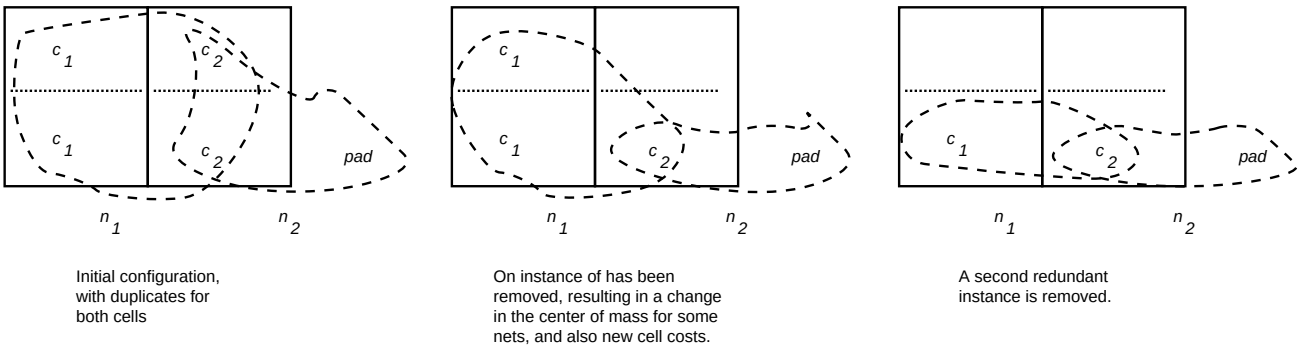

Figure 5: Two instances of each cell in the netlist are generated, and assigned to both subregions of any region. Cell costs are based on the center of mass for the vertices; high cost cells are removed one at a time *from any region of the entire placement problem*. In this way, the partitioning is performed on a global basis, rather than with only a pair of subregions at a time.

the component nets. For each net n_i, the center of mass for this net is the average X and Y location of the cells which it connects. The cost of any cell c_i is the sum of the distances between the cell and the center of mass of each net to which the cell is connected. In this way, a cell which is far from the center of mass of each net to which the cell is connected has high cost. Each region has a number of cells assigned to it, and an available capacity; redundant cells (those introduced by iterative deletion) with high costs are removed from the region which has the highest ratio of cell area to capacity.

Heaps are used to maintain the ordering of cells within any given region and the ordering of regions. In this way, maintenance and cell selection are both at worst $O(\log n)$ for each cell removed. As the number of redundant elements to be removed in any pass is n, each pass is $O(n \log n)$. Region sizes decrease by a factor of 2 with each pass, resulting in a logarithmic number of passes required. Thus, the *iterative deletion* portion of the algorithm is at worst $O(n \log^2 n)$.

To illustrate the iterative deletion process, we present Figure 5. In this figure, we duplicate cells c_1, in region R_1, and cell c_2 in region R_2, and assume that a net connects c_1 to c_2, and that a second net connects c_2 to a pad.

The order of cell deletions in this figure is as follows. Note that other cell deletions may be interspersed with the following; we focus only on these cells to clarify the process.

- The center of mass for the nets connected to cell c_2 is closest to the pad; we remove the instance of c_2 which is furthest from this location (as this is the instance which has highest cost).
- The center of mass for nets connected to cell c_2 is recalculated, and this is propagated to the other cells.
- Net n_1 now has two instances of c_1 and one instance of c_2 connected to it: the center of mass for this net changes, influencing the cost for each instance of c_1.
- An instance of c_1 is removed.

4 Parallel Cell Placement

The optimized sequential implementation of the placement algorithm discussed above was instrumented to isolate the portions that are most computationally demanding. The bulk of the execution time was spent in partitioning and reordering phases – iterative deletion updates contributed very little overhead after each partitioning pass. The parallel implementation and optimizations to it (both those we have already implemented and ones that are planned) will be discussed in this section. The implementation was performed using MPICH running on top of the Basic Interface for Parallelism (BIP) [30] on a myrinet connected PC cluster running Linux.

4.1 Parallel Partitioning

In order to maintain the global optimization achieved by iterative deletion, parallel processing is restricted to the growing list of regions to be partitioned in each pass. In the first pass the main region is partitioned by a master process into two. In the second pass the resulting two regions are partitioned into four in parallel by the master and a slave and so on until the processor limit is reached. As the list grows, regions above a certain threshold are distributed evenly by the master among all processes. An example is shown in Figure 6.

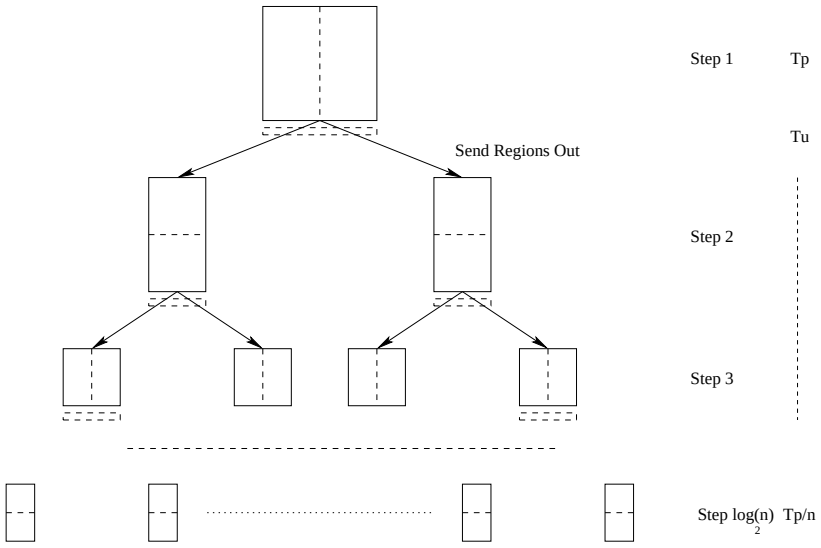

Figure 6: Parallel Partitioning

Initially, we implemented the data distribution using a message per work unit (i.e. region). Thus, the master walks the list and sends a message for each region to the slaves in a *round-robin* fashion. We optimized this implementation by generating a message per each batch of work units using MPI's *vector scatter-gather*. Obviously, the latter

outperforms the first because it increases the computation granularity by requiring the master to communicate fewer messages with larger size. Load balance is maintained by the aforementioned even distribution of regions and by the fact that the partitioning engine produces balanced region cuts as well.

Because VLSI CAD applications require large memory space, it is necessary to utilize memory efficiently. In order to do so, unicast master-to-slave and broadcast communication messages are used to instruct slaves to allocate only the space required for storing and processing their workload in the round robin and vector scatter-gather implementations respectively. Such messages incur slight overhead that is negligible for non-trivial placement problems.

An approximate analysis of the parallel partitioning follows. Consider a region of area R to be partitioned into N regions just below the threshold. Assuming a linear run time complexity of the partitioning engine and a homogeneous distribution of circuit elements, if it takes time T_p to partition R, then partitioning a region of area $\frac{R}{n}$ requires $\frac{T_p}{n}$. The progression of region partitioning is modeled by the tree in Figure 6. A sequential implementation performs n steps, with step time $\frac{T_p}{n}$, at each pass to partition n regions into 2n regions with a total time of T_p. The number of passes is log_2N, and so the total amount of partitioning time is $T_p \cdot log_2 N$. On the other hand, using P processors, the first $log_2 P + 1$ passes have $n <= P$ and each takes $\frac{T_p}{n}$ to complete (i.e. a maximum of one region per processor) while the rest have $n > p$ and each takes ceil(n/p) x T_p/n to complete (i.e. ceil(n/p) regions per processor). The total time for the first group of passes is:

$$\sum_{i=0}^{log_2 P} \frac{T_p}{2^i} \tag{1}$$

and the total time for the rest is:

$$\sum_{i=log_2 P+1}^{log_2 N-1} \frac{T_p}{P} = \frac{T_p}{P} \cdot (log_2 N - log_2 P - 1) \tag{2}$$

For machines with small P, over a wide range of problem size, N is much larger than P (i.e. the number of passes with $n <= p$ is much less than those with $n > P$), and if $log_2 N >> log_2 P + 1$, then the total time is given by the second formula and is approximated by $\frac{T_p}{p} \cdot log_2 N$. The best case speedup (sequential/parallel time) is hence linear in the number of processors.

Cost update and communication times could be incorporated by time T_u after each pass. The total of this overhead is the geometric series and is given by:

$$\sum_{i=0}^{log_2 N-1} 2^i \cdot T_u = T_u \cdot \frac{(2^{(log_2 N)} - 1)}{(2-1)} = (N-1) \cdot T_u \tag{3}$$

Since T_u is larger in the parallel implementation due to communication time, the actual speedup cannot be expected to be linear.

4.2 Parallel Reordering

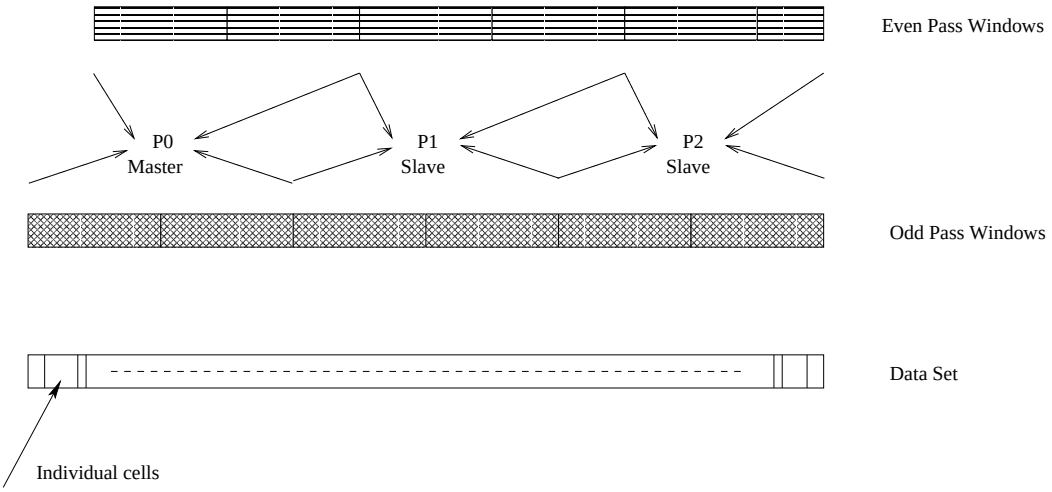

Figure 7: Parallel Reordering Passes

Branch and bound cell reordering is a second phase optimization that is performed after the placement step to improve the solution quality. A reordering window of a prespecified width in number of cells is slid across the each row in steps of half the window size. All combinations of reorderings of the cells within the current window are considered and the best window kept. Thus, the smallest work unit is the block of consecutive cells defined by the reordering window. The larger the window size the larger the block and the large the number of cells considered for reordering at a time, but the smaller the number of blocks since rows are of fixed length. All cells except half a window's worth at the beginning of each row are considered for reordering twice. To keep this key for quality in the parallel implementation, only non-overlapping blocks are reordered in parallel. Thus, each row undergoes two passes of reordering where a window now moves at steps of full size. The first pass considers even blocks only and the second considers odd blocks only or vice versa.

The same principles for partitioning load balance and communication are followed here; blocks are distributed evenly among all processors using vector scatter-gather as shown in Figure 7. Because there is often a large number of rows in a placement, it is crucial to optimize memory reallocation; allocated memory is not released unless the space required to hold blocks of a new row is larger than the available space.

5 Experiments

The parallel implementation of the placement tool was evaluated for execution time and quality of solution on a

Benchmark	Rows	Cells	Nets
fract	6	149	163
struct	21	1952	1920
primary1	17	833	904
primary2	22	3014	3029
biomed	44	6514	7052
industry2	69	12637	13419
industry3	52	15433	21967
avqsmall	79	21918	30038
avqlarge	83	25178	33298
golem3	117	100312	217362

Table 1: The placement benchmarks considered. The first four benchmarks are the ISCAS Benchmark circuits. The number of rows used was determined by the TimberWolf placement and routing tools.

Benchmark	Wire Length		Run Time (sec)			
	Sequential	Parallel (ratio)	Sequential	2 Processors	4 Processors	8 Processors
fract	66242	69498.4(1.05)	2.6	7	7	6.9
struct	775636	797647.6(1.03)	41.9	30.5	21.5	16.6
primary1	1053258	1064441(1.01)	18.6	16.9	13.2	11.8
primary2	3747715	3855615.4(1.03)	80.2	57.1	39.2	30.1
biomed	3382200	3514514(1.04)	162	111.8	79.4	64.5
industry2	15629154	16508315(1.06)	359.4	227.2	155.5	117.8
industry3	45088174	47757192.8(1.06)	538.6	338.5	218.1	157
avqsmall	6041243	6626165.6(1.1)	925.1	746.8	598.9	566.3
avqlarge	6344469	7053757.8(1.11)	1006	788.2	645.5	603.9
golem3	88959019	92830917(1.04)	3235.9	2191.2	1243.1	918.1

Table 2: Wire Lengths and Run Times for the Parallel Implementation Using Scatter/Gather on Myrinet

cluster of eight 550MHz Pentium III workstations. The cluster is interconnected using a Myrinet network; the Myrinet LAN cards use a LANai 7 33MHz processor and 33MHz, 32-Bit PCI bus. The machines are also connected via switched 100Mbit/sec Ethernet. In [38], *Feng Shui*'s performance was compared against commercially available tools and found to produce better results with smaller run times. In this paper, we only focus on the performance of the parallel version. Unless we indicate otherwise, all results are the average of five runs with five different seeds. Table 1 shows the benchmarks used in the experiments. These benchmarks span a wide range of circuits, up to and including the largest available public domain circuits.

Table 2 shows the execution times of the parallel algorithm (using scatter/gather). These results were about 30% faster than our initial implementation with round robin region distribution. The quality of the parallel version is not affected by the number of processors. The degradation in quality is due to the fact that the region processing is done in parallel and in phases. This is in contrast to the sequential version where the regions are updated sequentially, with every subsequent region using the updated values of the regions processed before it. We note that the smallest model (fract) suffers a slowdown in the parallel implementation; its size makes the communication overhead dominate the execution time. Note that even though the table shows the results for 2, 4 and 8 processors only, the algorithm is not

restricted to power of two processor configurations.

The results use a reorder window size of 6. As we will show later, a window size of 6 was found to provide the best quality to execution time point (bigger window size took much longer to execute and provided marginal improvement). The drop in quality was small for most benchmarks (lower than 6%). The exception was avqlarge and avqsmall which suffered around 10% reduction in solution quality. These two benchmarks also did not achieve good speedup despite their large size. We are currently looking more closely at their behavior to try to gain insight into their behavior; other studies report difficulties with these benchmarks [23]. The reduction in quality is significantly smaller than that reported in other parallel placement studies. For example, in one study the quality degradation of the parallel solution reached 30% (with an average of 14%) [23] on a 4 processor study. In another study, the quality drop also reached over 30% with an average of around 20%. Both studies only report results on a subset of the benchmarks used in this study.

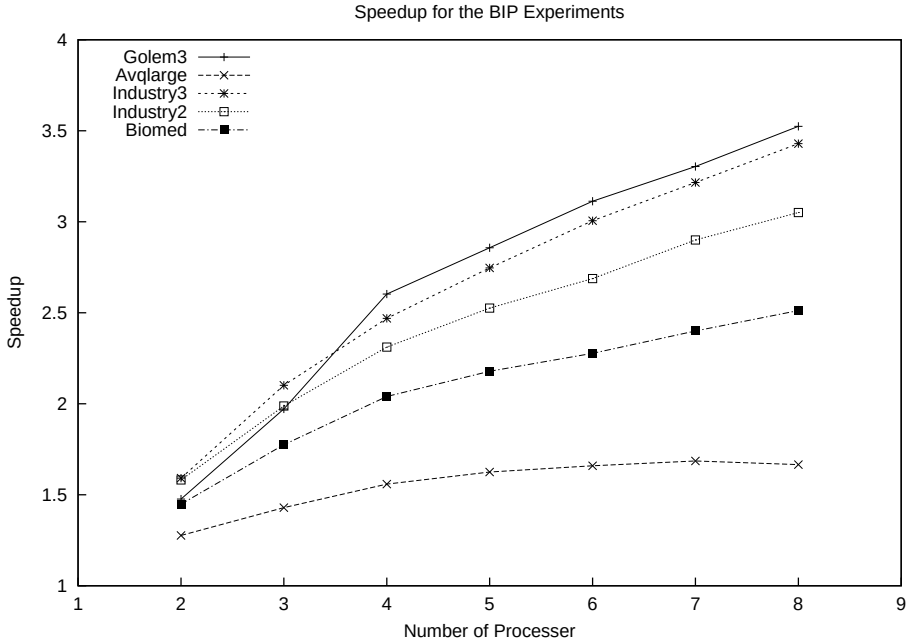

Figure 8: Speedup for the Myrinet/BIP Experiments

Figure 8 shows the speedup obtained by the parallel version on selected benchmarks. We note that the speedup generally increases with the size of the model; the larger the model the larger the granularity of the computation. The only exception is avqlarge – despite being the second largest model, it benefits least of the 5 models shown.

Table 3 shows the run times for selected benchmarks using ethernet communication. The quality results were identical to the Myrinet version. Clearly, the performance is significantly worse than the Myrinet version. This can more clearly be seen in Figure 9.

Benchmark	Run Time (sec)			
	Sequential	2 Processors	4 Processors	8 Processors
industry2	359.4	283	229.7	212.6
industry3	538.6	402.9	309.1	263.9
avqsmall	925.1	894.6	816	772.9
avqlarge	1006	946.2	872.6	842.3
golem3	3235.9	2328	1983	1635

Table 3: Wire Lengths and Run Times for the Parallel Implementation Using Scatter/Gather on Ethernet

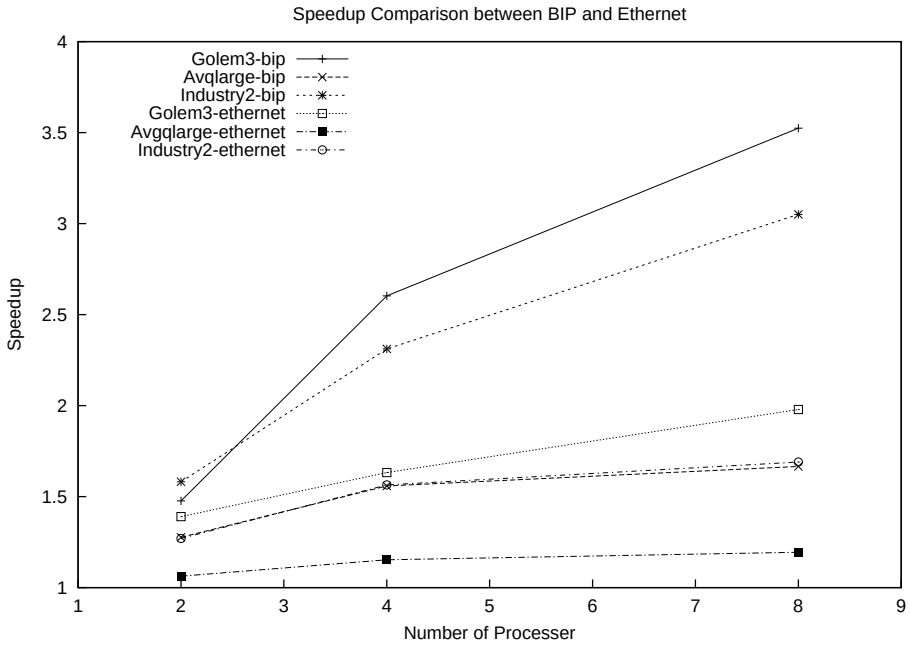

Figure 9: Speedup Comparison Myrinet/BIP vs. Ethernet

Iterative deletion proved useful in the sequential version of the tool, yielding a few percent improvement on average with a small increase in execution time. We attempted to verify whether iterative deletion will have the same effect on the parallel version. Table 4 presents the results of this study. While iterative deletion resulted in 0.7% improvement on average, it did perform worse for a number of benchmarks. Most notably, the two problematic benchmarks in terms of speedup and quality degradation (avqsmall and avqlarge) suffered the biggest degradation from using iterative deletion. Without those two, the improvement becomes 1.4%.

We also investigated the effect of the size of the reorder window (from the default 6 to 8). The execution time rose sharply for all benchmarks, but the efficiency of the parallelism for the reorder portion of the algorithm raised the overall speedup (as can be seen in Table 5). We note however that the gain in quality is marginal; we would be better off to execute with window size 6 sequentially rather than window size 8 – the gain from the higher window size is smaller than the loss due to parallelism.

Benchmark	Wire Length		Improvement (%)
	Without ID	With ID	
fract	68502	69498.4	-1.4
struct	804123	797647.6	0.8
primary1	1084639	1064441	1.9
primary2	3914142	3855615.4	1.5
biomed	3506130	3514514	-0.2
industry2	16488752	16508315	-0.1
industry3	48973501	47757192.8	2.5
avqsmall	6479309	6626165.6	-2.3
avqlarge	6947883	7053757.8	-1.5
golem3	96020944	92830917	3.3

Table 4: Effect of Iterative Deletion on the Parallel Execution

Benchmark	Window Size					
	6			8		
	Time	Speedup	Wire Length	Time	Speedup	Wire Length
struct	16.7	2.5	789777	34.3	3.32	782683
primary2	30.2	2.65	3863183	77.8	3.69	3852182
biomed	64.3	2.52	3494020	110	3.22	3472616
industry2	117.9	3.04	16444527	301.1	4.1	16380765
avqsmall	561.1	1.64	6670242	749.7	2.88	6603005

Table 5: Effect of Reording Window Size – 8 Processors

These results represent what is very much a work-in-progress. We expect to further refine the existing implementation in the following ways. Currently, the region list is relayed back to the master at the end of every partitioning step to update the dependencies, before resending the results back out to the slaves. This causes a lot of redundant communication as well as a sequential walk of the list. The next step is to allow a parallel walk of the list and update of the regions by doing an all to all exchange, instead of the "reduction" back to the master process. From the quality perspective, we are working on maintaining the sequential dependencies using a "wavefront pipeline" scheme. This formulation should produce the same quality as the sequential version, although the speedup will not be as high as the current formulation.

6 Concluding Remarks

With the exponential growth in the size of circuits under fabrication, efficient physical design tools are needed. Parallel processing offers the promise of increasing the performance and capacity of these tools. The emergence of clusters as cost-effective scalable high-performance computing platform brings the long overdue promise of parallel processing to the mainstream. The investigation of parallelization of physical design tools using clusters is timely.

In this paper, we presented experiences with parallelizing a state-of-the-art placement tool on a cluster of work-

stations. The sequential tool provides wire lengths comparable to those of a well known commercial tool, and results reported for the tool *Capo* indicate that placements are not difficult to route. The sequential implementation is efficient; the growth in run times is nearly linear with the size of the circuit. The tool uses a novel iterative deletion approach to allow the consideration of global objectives from within a traditional top-down placement framework. For more details please refer to the following paper [38].

Contrary to other efforts at parallel placement, we were able to obtain significant improvement in performance with minimal degradation in the quality of the layout. The degradation in the quality of the solution does not increase with the degree of parallelism. We explored several algorithmic and system optimizations and evaluated the implementation on a myrinet as well as a switched Ethernet network. We are still in the process of optimizing the implementation and are hopeful of achieving higher speedup in time for the final version of this paper.

We expect to continue to refine this tool both from an implementation and functionality perspectives. For example, timing driven placement is a significant concern for modern design. We are currently working with an industry research group to evaluate the performance of our approach on large designs under realistic delay rules. We note that delay optimization is perhaps more of a global phenomena than wire length minimization: meeting timing objectives may require modifications in many areas of a placement, and reductions in delay for some nets may require increased delay in others.

References

[1] T. Anderson, D. Culler, and D. Patterson. The case for NOW (network of workstations). *IEEE Micro*, 15(1), February 1995.

[2] P. Banerjee. *Parallel Algorithms for VLSI Computer-Aided Design*. Prentice Hall, Englewood Cliffs, New Jersey, 1994. (Chapter 3).

[3] D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, and C. Packer. BEOWULF: A parallel workstation for scientific computation. In *International Conference on Parallel Processing*, 1995.

[4] M. Blumrich, R. Alpert, Y. Chen, D. Clark, S. Damianakis, C. Dubnicki, E. Felten, L. Iftode, K. Li, M. Martonosi, and R. Shillner. Design choices in the shrimp system: An empirical study. In *Proceedings of the* 25^{th} *Annual ACM/IEEE International Symposium on Computer Architecture*, June 1998.

[5] N. Boden, D. Cohen, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1), February 1995.

[6] M. A. Breuer. A class of min-cut placement algorithms. In *Proc. Design Automation Conf*, pages 284–290, 1997.

[7] A. E. Caldwell, A. B. Kahng, and I. L. Markov. Optimal partitioners and end-case placers for standard cell layout. In *Proc. Int. Symp. on Physical Design*, pages 90–96, 1999.

[8] A. Casotto and A. Sangiovanni-Vincentelli. Placement of standard cells using simulated annealing on the connection machine. In *Proceedings of the International Conference on Computer-Aided Design*, pages 350–353, November 1987.

[9] F. Darema, S. Kirkpatrick, and V. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development*, May 1987.

[10] Dolphin Internconnects. The Dolphin SCI interconnect – whitepaper, February 1996. Available from `http://208.179.47.35/pdf_filer/T-WhitePaper.pdf`.

[11] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: Efficient support for reliable, connection-oriented communication. In *Hot Interconnects V*, August 1997.

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

[13] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proc. Design Automation Conf*, pages 269–274, 1998.

[14] Charles M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[15] P. Geoffray, L. Prylli, and Bernard Tourancheau. BIP-SMP: High performance message passing over a cluster of commodity SMPs. In *Proceedings of Supercomputing (SC99)*, November 1999.

[16] D. J.-H. Huang and A. B. Kahng. Partitioning based standard cell global placement with an exact objective. In *Proc. Int. Symp. on Physical Design*, pages 18–25, 1997.

[17] M. Ibel, K. Schauser, C. Scheiman, and M. Weis. High performance cluster computing using SCI. In *Hot Interconnects V*, August 1997.

[18] R. Jayaraman and R. Rutenbar. Floorplanning by annealing on a hypercube multiprocessor. In *Proceedings of the International Conference on Computer-Aided Design*, pages 346–349, November 1987.

[19] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in VLSI domain. In *Proc. Design Automation Conf*, pages 526–529, 1997.

[20] Brian W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[21] S. Kim, B. Ramkumar, J. Chandy, S. Parkes, and P. Banerjee. ProperPLACE: a portable parallel algorithm for standard cell placement. In *Proceedings of the 8th International Parallel Processing Symposium (IPPS'94)*, April 1994.

[22] R. Kling and P. Banerjee. Concurrent ESP: a placement algorithm for execution on distributed processors. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–357, November 1987.

[23] T. Koide, M. Ono, S. Wakabayashi, and Y. Nishimaru. Par-POPINS: a timing-driven parallel placement method with the elmore delay model for row based VLSIs. In *Proceedings of the Design Automation Conference (DAC'97)*, 1997.

[24] S. Kravitz and R. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 6(4):534–549, June 1987.

[25] P. H. Madden. Partitioning by iterative deletion. In *Proc. Int. Symp. on Physical Design*, pages 83–89, 1999.

[26] S. Mohan and P. Mazumder. Wolverines: Standard cell placement on a network of workstations. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 12(9):1312–1326, September 1993.

[27] M-VIA: Virtual interface architecture for linux, 2001. `http://www.nersc.gov/research/FTG/via/`.

[28] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing (SC'95)*, 1995.

[29] G. Pfister. *In Search of Clusters, 2nd Ed.* Prentice Hall, 1998.

[30] L. Prylli. BIP messages user manual, 1998. Available at `http://lhpca.univ-lyon1.fr/BIP-manual/index.html`.

[31] L. A. Sanchis. Multiple-way network partitioning with different cost functions. *IEEE Trans. on Computers*, 42(22):1500–1504, 1993.

[32] Scalable Computing Lab. SCL cluster cookbook: Building your own clusters for parallel computation, 1998. `http://www.scl.ameslab.gov/Projects/ClusterCookbook`.

[33] K. Shahookar and P. Mazumder. Vlsi cell placement techniques. *ACM Computing Surveys*, 23(2):143–220, June 1991.

[34] P. R. Suaris and G. Kedem. An algorithm for quadrisection and its application to standard cell placement. *IEEE Trans. on Circuits and Systems*, 35(3):394–303, 1988.

[35] W.-J. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits andSystems*, 14(3):349–359, 1995.

[36] Virtual interface architecture (VIA) specification, 2001. `http://www.viarch.org`.

[37] M. Welsh, A. Basu, and T. von Eicken. Atm and fast ethernet network interfaces for user-level communication. In *Proceedings of the Third High-Performance Computer Architecture Conference (HPCA'97)*, February 1997.

[38] M. Yeldiz and P. H. Madden. Global objectives for standard cell placement. In *Design Automation Conference (DAC'2000)*, 2000.