An Algorithm for Dynamically Reconfigurable FPGA Placement

Guang-Ming Wu[1], *Jai-Ming Lin*[2], *and Yao-Wen Chang*[3]

[1]Department of Information Management, Nan-Hua University, Chiayi, Taiwan
gmwu@mail.nhu.edu.tw
[2]Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan
gis87808@cis.nctu.edu.tw
[3]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
ywchang@cc.ee.ntu.edu.tw

Abstract

In this paper, we introduce a new placement problem movitated by the Dynamically Reconfigurable FPGA (DRFPGA) architectures. Unlike traditional placement, the problem for DRFPGAs must consider the precedence constraints among logic components. For the placement, we develop an effective metric that can consider wirelength, register requirement, and power consumption simultaneously. With the considerations of the new metric and the precedence constraints, we then present a three-stage scheme of partitioning, initial placement generation, and placement refinement to solve the new placement problem. Experimental results show that our placement scheme with the new metric achieves respective improvements of 17.2%, 27.0%, and 35.9% in wirelength, the number of registers, and power consumption requirements, compared with the list scheduling method.

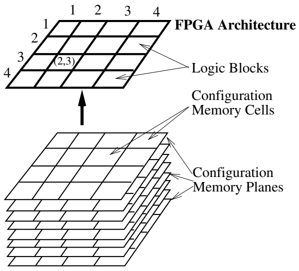

Figure 1: The Xilinx DRFPGA configuration model.

1 Introduction

Improving logic efficiency by time-sharing, Dynamically Reconfigurable FPGAs (DRFPGAs) have gained much attention recently. In a DRFPGA, a virtual large design is partitioned into multiple *stages* (or partitions) to share the same smaller physical device than that occupied by a traditional FPGA. Various architectures have been proposed, e.g., the Xilinx model [12], the Dynamically Programmable Gate Array [2], and the Virtual Element Gate Array [7]. In these models, on-chip SRAM bits are programed to record the configuration of each stage. Dynamic reconfiguration of logic blocks and wire segments can be performed by reading the on-chip SRAM bits of each configuration in order.

Figure 1 shows the Xilinx DRFPGA configuration model [12]. The Xilinx DRFPGA emulates a single large design in multiple configurations. Each configuration can be stored in a *configuration memory plane* (CMP) which consists of a two-dimensional array of *configuration memory cells* (CMCs). In each *micro cycle*, the SRAM bits of the corresponding configuration are loaded into the DRFPGA, and the configurable logic blocks (CLBs) are reused to evaluate combinational logic. One pass through all micro-cycles is called a *user cycle*. The target architecture consists of an array of augmented XC4000E-style CLBs [12]. Each CLB includes a set of *micro registers* (*MRs*) to hold the CLB results between configurations. Every CMC of the original FPGA is packed by eight inactive memory cells. MRs not only store the intermediate values of combinational logic for use in later micro-cycles, but also hold latch values for use in the next user cycle. A micro-cycle starts with saving all the CLB results of the previous micro-cycle in MRs, and then a new configuration is loaded into the active configuration memory. The loading process is called *flash reconfiguration*.

Due to the reuse of logic and interconnect, the placement problem for DRFPGAs is quite different from the traditional one. Unlike traditional FPGAs, the order of the execution of nodes must satisfy the precedence constraints in a DRFPGA. We refer to the *lifetime* of a node in a DRFPGA as the duration from the stage where the node is assigned to the stage when it is last used. The intermediate value of a node must be stored in an MR during its lifetime. The values of several nodes can be stored in the same MR if the lifetimes of the nodes do not overlap. In contrast, if there are two combinational or latch nodes placed in the same position on different memory planes and their lifetimes overlap, then their results cannot be stored in the same memory space of an MR. Besides, the number of nodes whose lifetimes overlap in the same position cannot exceed the MR capacity—the *MR-capacity constraint*.

For the DRFPGA placement, we develop in this paper a new metric that can simultaneously consider wirelength, MR usage, and power consumption under the precedence constraints. With the considerations of the new metric and the precedence constraints, we then present a three-stage scheme of partitioning, initial placement generation, and placement refinement to solve the new placement problem for DRFPGAs. The first stage partitions a circuit into k sub-circuits without violating the precedence constraint, where k is the number of CMPs in a DRFPGA. The k-way DRFPGA partitioning method is an extension of the FM [5] balanced bipartitioning. In the partitioning, we reduce the length of lifetime for each node as much as possible, since the length of lifetime is closely related to the number of MRs required. The second stage employs a constructive method to obtain an initial placement; nodes are placed in the decreasing order of the percentages that their neighbors are already placed. The last stage applies a simulated annealing approach to improve the initial placement. Experiments with the benchmark circuits used in [4] show that our placement scheme with the new metric achieves respective improvements of 17.2%, 27.0%, and 35.9% in wirelength, the number of registers, and power consumption requirements, compared with the list scheduling method.

The remainder of this paper is organized as follows. Section 2 formulates the new placement problem. Section 3 presents the new metric for the DRFPGA placement. Section 4 proposes the three-stage placement algorithm. Section 5 shows the experimental results, and finally conclusions are given in Section 6.

2 Problem Formulation

In this paper, all circuits are preprocessed by a lookup table based (LUTs) technology mapper [11] and thus the circuit components are composed of lookup tables, latches, and netlists. We represent a circuit by a directed hypergraph $G = (V, E)$, where V is the set of LUTs and latches and E is the set of nets. We denote a net e by $e = (v_1 \to < v_2, v_3, \ldots, v_n >)$, where v_1 is the *fanout* node whose output signal is the input signal to v_j $(2 \leq j \leq n)$, and v_j $(2 \leq j \leq n)$ is the *fanin* node whose input signal is the output signal from v_1. The set E can be divided into two subsets E_c and E_f according to the type of fanout nodes. A net $e \in E_c$ (E_f) if the fanout node of e is an LUT (latch) node.

For a DRFPGA, a circuit is placed into several CMPs such that the logic in different CMPs temporally shares the same physical CLBs by setting the CMPs active in order. To ensure the correct results of a circuit in a user cycle, the nodes must be evaluated in the proper order. According to the Xilinx architecture, the following *precedence constraints* must be satisfied:

- Each LUT node must be placed in a CMP no later than all its output nodes.
- Each latch node must be placed in a CMP no earlier than all its input nodes. (This ensures that latch input values are calculated before they are stored.)
- Each latch node must be placed in a CMP no earlier than all its output nodes. (This ensures that all of the nodes use the same value of the latch—the value of the latch from the previous user cycle.)

The above constraints define a partial temporal ordering on the nodes in the circuit. Let $Pre(v)$ be the precedence of a node v. For two nodes v and u, we define $Pre(v) \preceq Pre(u)$ if v must be placed no later than u. Let $s(v) = i$ if node v is assigned to the CMP i. $s(v) \leq s(u)$ if $Pre(v) \preceq Pre(u)$. The placement with the precedence constraint is called *precedence-constrained placement* (*PCP*).

Two nodes are said to be *related* if they are placed in the same CLB of two different CMPs. The *lifetime* of a node is the duration from the CMP where it is assigned to the CMP where it is last used. A node in its lifetime is called a *live node*, i.e., the data of the node must be stored in an MR for later use. For the fanout node v_1 of a net $e = (v_1 \rightarrow< v_2, v_3, \ldots, v_n >)$, and if $e \in E_c$, the lifetime of v_1 is from the CMP $s(v_1)$ to the CMP $\max\{s(v_j)|2 \leq j \leq n\}$, and if $e \in E_f$, the lifetime of v_1 is from the CMP $s(v_1)$ to the last CMP and from the CMP 1 to the CMP $\max\{s(v_j)|2 \leq j \leq n\}$, because the output of a latch node is used in the next user cycle. If there does not exist any net whose fanout node is the node v_1, v_1 has no lifetime. It implies that the data of v_1 does not have to be stored for later use. The result of a node must be stored in an MR during its lifetime; several related nodes can share the same MR if their lifetime do not overlap. In other words, the results of two related nodes must be stored in different MRs if their lifetime overlap.

The power consumption during reconfiguration can be very high. Therefore, the PCP shall consider power consumption. For two nodes u and v, if $Pre(u) \preceq Pre(v)$ and u and v are placed in the same CLB of different CMPs, node v can get the result of u immediately from the MR in its own CLB after flash reconfiguration, e.g., the case of nodes v_1 and v_2 in Figure 2. If $Pre(u) \preceq Pre(v)$ and u and v are placed in different CLBs of different CMPs, the result of the node u must be passed to the node v by an extra connection during flash reconfiguration (e.g., the case of nodes v_3' and v_4' in Figure 2); this will increase the the power consumption of the system. The nodes v_3' and v_4' are called a *power-consumption pair*. Considering the power consumption in the DRFPGA placement, we prefer to place nodes in the same CLB of CMPs if they have data dependency.

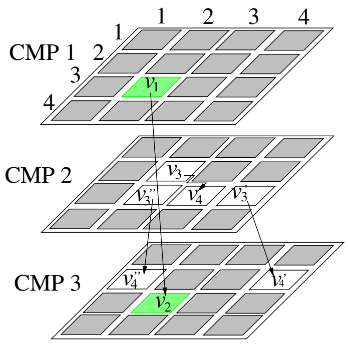

Figure 2: Three cases of placing net $(v_3 \rightarrow< v_4 >)$ to empty CMCs, with the other net $(v_1 \rightarrow< v_2 >)$ having been preplaced. Case 1: v_3 and v_4 are placed in the same CMP, and thus the cost is due to the wirelength alone. Case 2: v_3' and v_4' have wirelength and power consumption penalties, but it has no MR memory penalty. Case 3: v_3'' and v_4'' have wirelength, memory and power penalties, assuming $|m_{3,2}|$ is the largest among all $|m_{i,j}|, 1 \leq i, j \leq n$.

We use the following notations throughout this paper.

- $c(u, v)$: A power-consumption pair for nodes u and v.
- C: The set of all power-consumption pairs in the placement.
- $|C|$: The number of power-consumption pairs in C.
- $B = (P, M)$: P is the set of configuation memory cells (CMCs) and M is the set of MRs.
- $D(B)$: A DRFPGA, where B is the set of $n \times n$ CLBs in the DRFPGA.
- $b_{i,j} \in B$ $(1 \leq i, j \leq n)$: The CLB at the grid location (i, j) in D.
- $p_{k,i,j}$: The CMC at the grid location (i, j) in CMP k.
- $P_k = \{p_{k,i,j}|1 \leq i, j \leq n\}$: The set of CMCs in CMP k.
- $P = \bigcup_{i \in \{1,2,\ldots,r\}} P_i$, where r is the number of stages (CMPs) in the DRFPGA.
- $m_{i,j} \in M$: The set of MRs needed in $b_{i,j}$.
- $|m_{i,j}|$: The size of $m_{i,j}$, i.e., the number of MRs needed in $b_{i,j}$.
- $p^t_{k,i,j}$: An LUT cell in $p_{k,i,j}$.
- $p^l_{k,i,j}$: A latch cell in $p_{k,i,j}$. (Each CMC $p_{k,i,j}$ consists of an LUT cell and a latch cell.)

The Precedence-Constrained Placement (PCP) problem is defined as follows.

- **Instance:** A DRFPGA $D(B)$ and a circuit graph $G(V, N)$.
- **Problem:** Assign each LUT node (latch node) to a unique CMC $p^t_{k,i,j}$ $(p^l_{k,i,j})$, where $1 \leq k \leq r$ and $1 \leq i, j \leq n$ so that

  (1) the total wirelength,

  (2) $\max\{|m_{i,j}||1 \leq i, j \leq n\}$, and

  (3) $|C|$

  are simultaneously minimized, and for any nodes v_1 and v_2, $s(v_1) \leq s(v_2)$ if $Pre(v_1) \preceq Pre(v_2)$.

The first objective considers wirelength. Unlike the traditional placement problem, the estimation of the wirelengths in the PCP must consider two cases. One is that all nodes of a net are assigned to the same CMP. In this case, the wirelength is estimated by the geometric (Manhattan) distance of the net, same as the traditional measurement. The other is that the nodes of a net are assigned to different CMPs. For this case, we must project all nodes to the same CMP and then estimate the wirelength as in the previous case. The second objective tries to minimize $\max\{|m_{i,j}||1 \leq i, j \leq n\}$, facilitating the design to fit into a CLB with fewer MRs. Note that the MRs in the CLBs of a DRFPGA are all identical. The third objective is intended to minimize power consumption.

3 Metric for the PCP

In this paper, we are first concerned with the problem of finding an effective metric to guide the low-power precedence-constrained placement. By effective, we mean one that can simultaneously minimize wirelength, MR count, and power consumption for the problem being considered.

In PCP, to achieve good performance, the metric must consider the three issues: (1) wirelength, (2) micro register requirement, (3) power consumption. The metric presented in this paper is defined as follows. Let $x(e)$ be the placement of a net e which satisfies the precedence constraints. The cost for $x(e)$, $\Phi(x(e))$, is given by

$$\Phi(x(e)) = \alpha w(x(e)) + \beta h(x(e)) + \gamma o(x(e)), \quad (1)$$

where $w(x(e)), h(x(e))$, and $o(x(e))$ represent the respective cost functions for wirelength, MR count, and power consumption, and α, β, and γ are user-specified parameters. Here, $\alpha + \beta + \gamma = 1$ and $\alpha, \beta, \gamma \geq 0$. Example 1 illustrates several cases of a placement of a two-terminal net and estimates their cost by our metric.

Example 1 *In the example shown in Figure 2, we assume that the net $(v_1 \rightarrow< v_2 >)$ has been preplaced and $|m_{3,2}|$ is the largest among $|m_{i,j}|$ $(1 \leq i, j \leq n)$ presently. Assuming $\alpha = \beta = \gamma = \frac{1}{3}$, we consider the three cases of the placement for net $e = (v_3 \rightarrow< v_4 >)$. In the first case, v_3 is placed in $p_{2,2,2}$ and v_4 is placed in $p_{2,3,3}$; we get $\Phi(x(e)) = \frac{2}{3}$ since it only spends two units (a unit represents the distance* *between two adjacent cells) of wirelength. In the second case,* v_3 *is placed in* $p_{2,3,4}$ *and* v_4 *is placed in* $p_{3,2,4}$*; we get* $\Phi(x(e)) = \frac{2}{3}$ *since it generates a power-consumption pair* (v_3', v_4') *and wirelength=1. In the third case,* v_3 *is placed in* $p_{2,3,2}$ *and* v_4 *is placed in* $p_{3,2,1}$*; we get* $\Phi(x(e)) = \frac{5}{3}$ *(wirelength = 2; it contributes one memory space to* $|m_{2,2}|$ *and generates a power-consumption pair* (v_3'', v_4'')*).*

4 Our Approach

In this section, we present the algorithm for PCP. We consider partitioning and placement simultaneously in our method.

Figure 3 shows the framework for our placement algorithm. The first step is a precedence-constrained partitioning that partitions a circuit into r stages (associated with the CMPs) and minimizes the length of lifetimes of nodes, since the lengths of lifetimes affect the number of MRs needed for a DRFPGA. Once the partitioning is done, placement is performed for each CMP. We apply an iterative two-stage algorithm for each CMP i: an initial constructive method for CMP i followed by a simulated annealing method for CMPs 1 to i.

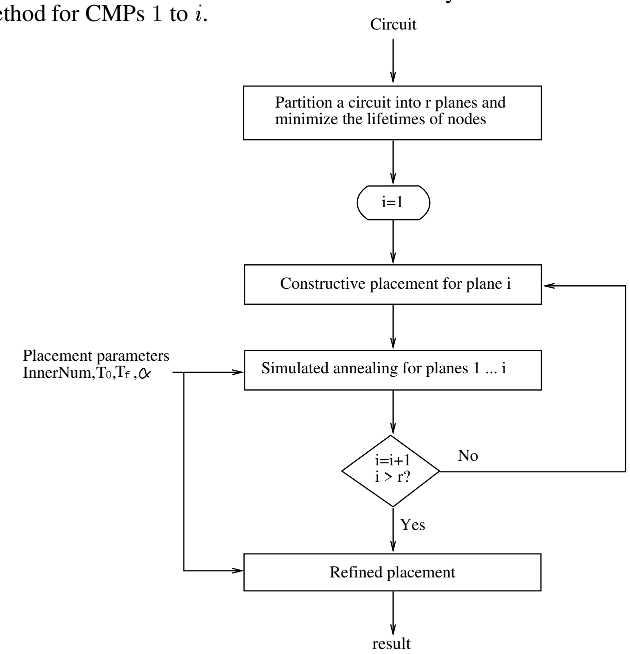

Figure 3: High-level view of our placement algorithm.

4.1 Precedence Constraint Partitioning

In the PCP, we first partition a circuit into r sub-circuits and then apply the precedence-constrained placement algorithm to map each sub-circuit to the corresponding CMP. In order to reduce the number of the MRs required for a circuit, we minimize the maximum density of live nodes. In this subsection, we propose an effective partitioning algorithm to shorten the lifetimes of nodes, which affect the number of MRs needed directly.

Our algorithm begins with an initial feasible partitioning which is usually the result of the ASAP and/or ALAP [6] scheduling or is produced by using a constructive partitioning method. A node may be assigned to any CMP if the precedence constraints are not violated.

Given an initial partitioning, our algorithm improves the quality of the partitioning iteratively by selecting a set of tuples with the maximum accumulative gain, where the tuple is used to record the move of nodes and is represented by $tuple(node, CMP)$. $tuple(v, i)$ is selected when the node v is moved into CMP i. Specifically, in an iteration, we select the $tuple(v, i)$ which

(1) has the maximum gain,

(2) satisfies the precedence constraints, and

(3) satisfies the balance criterion,

and a tentative move of the corresponding node is made. Then the gains associated with all neighbors of v are updated. A $tuple(v, i)$ cannot be selected twice in an iteration. We repeat the selection process described above until all tuples are selected. In each iteration, all selected tuples and the resulting gains are recorded in order. The *partial sum* of the ith tuple is the total gains of the first i tuples. At the end of an iteration, the corresponding nodes of the maximum partial sum are moved. Repeat the above action of an iteration until the maximum partial sum of an iteration is not greater than zero. This scheme is similar to the algorithm proposed by Fidducia and Mattheyses [5].

The gain function in our precedence-constrained partitioning is described in the following. The goal for the precedence-constrained partitioning is to minimize the maximum size of $cut(k)$, where $cut(k)$ denotes the set of MRs needed between CMP k and CMP $k+1$. If a node v_i is moved from CMP j to CMP k, then only the $cut(x)$, $\min\{j,k\} \le x < \max\{j,k\}$ may be changed. Therefore, if node v is moved from CMP j to CMP k, the gain function $g_v(j,k)$ is given as follows:

$$g_v(j,k) = \max_{\min\{j,k\} \le z < \max\{j,k\}} \{cut(z) \cdot \Delta_v(j,k)\}, \tag{2}$$

where $\Delta_v(j,k)$ denotes the change of the maximum $cut(z)$, $\min\{j,k\} \le z < \max\{j,k\}$; it is defined by

$$\Delta_v(j,k) = \max_{\min\{j,k\} \le z < \max\{j,k\}}\{cut_v(z)\} - \max_{\min\{j,k\} \le z < \max\{j,k\}}\{cut_{\overline{v}}(z)\},$$

where $cut_v(z)$ is the size of $cut(z)$ on condition that v is still in CMP j, and $cut_{\overline{v}}$ is the size of $cut(z)$ on condition that v is moved from CMP j to CMP k.

Before computing the gain values, we must check whether the move of a node violates the precedence constraints. In each iteration, the gain value is calculated for each unlocked tuple.

4.2 Placement of Each CMPs

After we have partitioned a circuit into r CMPs, a placement algorithm is then applied to the circuit. Our placement algorithm consists of two stages: the constructive method followed by the simulated annealing-based iterative improvement method.

To obtain an initial placement, we use a constructive algorithm rather than a random one. Given a circuit $G(V, E)$, we first perform a random placement of all I/O nodes in the circuit. It provides anchor points for the other unplaced nodes. For each unplaced node v, we define *p_degree(v)* and *np_degree(v)* as follows. *p_degree(v)* is the number of placed neighbors of v; in contrast, *np_degree(v)* is the number of unplaced neighbors of v. We set the priority of a node v depending on the ratio $\frac{p\_degree(v)}{p\_degree + np\_degree(v)}$ (called p_ratio). A node with higher p_ratio implies that most of its neighbors are placed; in other words, there is more precise information while it is placed. The placement ordering of nodes is according to the priorities derived from their p_ratios. (We called this placement as *p_ratio constructive placement*.) For a node v, we place v at the position associated with the arithmetic mean of all the placed neighbors of v. Due to the use of the neighborhood information from the placed nodes, our constructive method for PCPs leads to a superior initial placement for the simulated annealing method than that produced by a random method; as shown in Figure 4, the improvement is more significant when the size of DRFPGAs is increasing.

Following the constructive method for PCPs, a simulated annealing based method [8, 10] is applied to improve the initial placement. The simulated annealing method is one of the most well-developed and widely used iterative techniques for solving combinatorial optimization problems.

The key parameters used in a simulated annealing method, such as initial temperature (T_0), the number of moves per temperature (called *Inner_Num*($|V|$)), the temperature update factor ($\delta(T)$), and the termination temperature (T_f), determine the quality of the resulting PCP. In the simulated annealing method, if T_0 is too high, the information of the initial placement produced by the constructive algorithm in the last subsection will be missed; if it is set too low, a near optimal solution will not be found.

We describe the ingredients of the simulated annealing method in the following.

Initial placement: The result of the p_ratio constructive placement serves as the initial placement.

Cost function: We apply the metric Φ as the cost function in the simulated annealing process.

Perturbation functions: We perturb a feasible solution to another feasible solution by using the following three moves.

Circuit	Wirelength			$\|C\|$			$\max\{\|m_{i,j}\|\}$			Time
	Random	List	Ours	Random	List	Ours	Random	List	Ours	(sec.)
s5378	18129	5655	4315	1093	1055	549	5	4	3	551
s9234	15543	7953	5873	788	763	531	4	4	3	847
s13207	34562	17014	14208	1721	1615	908	6	5	3	1284
s15850	47948	13360	9873	2215	2060	1328	7	6	4	2048
s35932	105233	33858	27409	6657	6073	4267	6	6	5	3997
s38417	121954	58048	49026	8613	8059	5382	7	6	4	3871
s38584	139273	60325	51784	8711	8066	4795	8	6	5	4073
Total	482642	196213	162488	29798	27691	17760	43	37	27	16671
Improv.(%)	+66.3%	+17.2%		+40.4%	+35.9%		+37.2%	+27%		

Table 1: Results for the 8-stage DRFPGA placement.

M1: Swap two nodes of the same type in a CMP.

M2: Replace a node to an empty cell with no precedence constraints.

M3: Swap two nodes of the same type in the different CMPs with no precedence constraints.

Here, we assign the M1 move the highest probability 0.7, the M2 move the probability 0.2, and the M3 move the lowest probability 0.1. For each solution perturbation, we check if the placement after the perturbation satisfies the precedence constraints.

Annealing schedule: The cooling schedule is given by

$$T_{i+1} = \delta(T) \times T_i \tag{3}$$

where $\delta(T)$ is the cooling rate parameter.

Inner loop criterion: The value of $Inner\_Num(|V|)$ depends on the size of the circuit, e.g., 150 moves per temperature for a 300-node circuit and 800 moves per temperature for a 4000-node circuit.

The simulated annealing process stops when the temperature reaches the specified value T_f.

5 Experimental Results

Our algorithm has been implemented in the C++ language on a PC with a Pentium II 300 microprocessor and 512 MB RAM. We use the same test suite as [4], in which only the larger sequential circuits of the ISCAS'89 benchmarks are chosen. These circuits are translated by technology mapper [11] into 4-LUTs.

In our experiments, we estimate the wirelength by the *minimum spanning tree* method, which is very closely to the actual routing length. We implemented the p_ratio constructive placement and compared the wirelength with `List` and random initialization on DRFPGA of various sizes. The sizes of an array-based FPGA range from 70×70 to 200×200. As shown in Figure 4, the average wirelength of the initial placements generated by the p_ratio constructive placement is substantially smaller than those generated by `List` and random initializations. In addition, the difference is more significant as the sizes of DRFPGAs increase. List scheduling labels each node with a priority and each node is greedily placed into a cell in an order according to the priorities. A placed node influences the priority of its neighbors. For the precedence constraints, most of the related research [4, 3, 13] applies the list scheduling heuristic with different priority models. But they only considered the partitioning problem. In our experiment, we adapt the list scheduling for the PCP. We compared our method with the list scheduling placement `List` and the randomly generated placement `Random` on the Xilinx DRFPGA model, in which a circuit was placed into 8 CMPs. The size of the DRFPGA is set to 25×25. The results are shown in Table 1. Columns 2–4 list the total wirelength of `Random`, `List` and our results (ours), respectively. Columns 5–7 list the number of power-consumption pairs of `Random`, `List` and Ours, respectively. Columns 8–10 list the maximum size of micro registers needed between two successive CMPs. The runtimes of our algorithm are shown in Column 11. The last row in Table 1 reports the average improvement. The improvement for the `List(Random)` is calculated by $\frac{\texttt{List(Random)} - Ours}{\texttt{List(Random)}} \times 100\%$. Overall, our method reduces the total wirelength, $|C|$ and $\max\{|m_{i,j}|\}$ by 17.2% (66.3%), 35.9% (40.4%) and, 27% (37.2%), respectively, compared with `List` (`Random`).

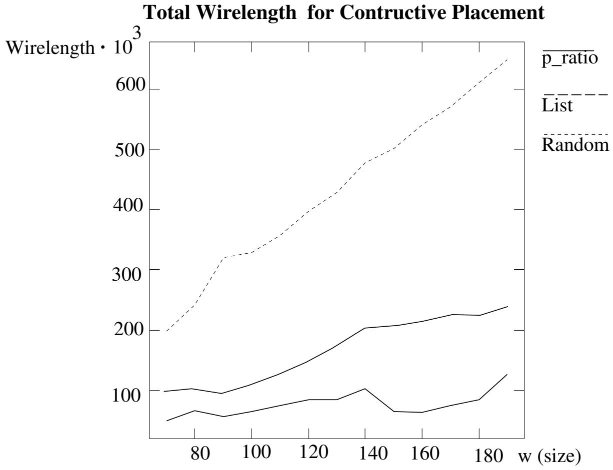

Figure 4: Comparison of the wirelengths by using the p_ratio constructive placement, `LIST`, and randomly generated placement on DRFPGA of different sizes.

6 Conclusions

In this paper, we have formulated a new precedence constrained placement problem for Dynamically Reconfigurable FPGAs and presented an efficient algorithm to solve it. We proposed a new metric which can simultaneously consider wirelength, utilization of micro registers, and power consumption. We also presented a new constructive algorithm for initial placement. Experimental results show that, with the new metric, our algorithm outperforms `List` by a large margin.

References

[1] F. Brglez, "ACM/SIGDA Design Automation Benchmarks: Catalyst or Anathema?," *IEEE Design and Test*, Sep., 1993.

[2] J. Brown, et al. "DELTA: Prototype for a first-generation dynamically programmable gate array," Transit Note 112, MIT, 1995.

[3] D. Chang and M. Marek-Sadowska, "Buffer minimization and Time-multiplexed I/O on Dynamically Reconfigurable FPGAs," *Proc. FPGA'97.*

[4] D. Chang and M. Marek-Sadowska, "Partitioning Sequential Circuits on Dynamically Reconfigurable FPGAs," *Proc. FPGA'98.*

[5] C. M. Fidducia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions", *Proc. DAC'82.*

[6] C. Y. Hitchcock III and D. E. Thomas, "A Method of Automatic Data Path Synthesis," *Proc. DAC'83.*

[7] D. Jones and D. M. Lewis, "A time-multiplexed FPGA architecture for logic emulation, " *Proc. CICC'95.*

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, May, 1983.

[9] G. G. Lemieux and S. D. Brown and D. Vranesic, "On Two-Step Routing for FPGAs," *Proc. ISPD'97.*

[10] C. Sechen, and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, Apr., 1985.

[11] E. M. Sentovich, and K. J. Singh, et al. "SIS: A system for sequential circuit synthesis. Report M92/41," University of California, Berkeley, 1992.

[12] S. Trimberger, "A Time-Multiplexed FPGA," in *Proc. FCCM'97.*

[13] S. Trimberger, "Scheduling Design into a Time-Multiplexed FPGA," in *Proc. FPGA'98.*