Profile-Guided Microarchitectural Floor Planning for Deep Submicron Processor Design

Mongkol Ekpanyapong, *Member, IEEE*, Jacob Rajkumar Minz, *Student Member, IEEE*, Thaisiri Watewai, *Student Member, IEEE*, Hsien-Hsin S. Lee, *Member, IEEE*, and Sung Kyu Lim, *Member, IEEE*

***Abstract*—As very large scale integration (VLSI) process technology migrates to nanoscale with a feature size of less than 100 nm, global wire delay is becoming a major hindrance in keeping the latency of intrachip communication within a single cycle, thus substantially decaying performance scalability. In addition, an effective microarchitectural floor planning algorithm can no longer ignore the dynamic communication patterns of applications. This article, using the profile information acquired at the microarchitecture level, proposes a "profile-guided microarchitectural floor planner" that considers both the impact of wire delay and the architectural behavior, namely, the intermodule communication, to reduce the latency of frequent routes inside a processor and to maintain performance scalability. Based on the simulation results here, the proposed profile-guided method shows a 5%–40% average instructions per cycle (IPC) improvement when the clock frequency is fixed. From the perspective of instruction throughput in billion instructions per second (BIPS), the floor planner is much more scalable than the conventional wirelength-based floor planner.**

***Index Terms*—Floor planning, microarchitectural design.**

I. Introduction

According to the projection of the International Technology Roadmap for Semiconductors (ITRS), deep submicron process technology will soon be able to integrate more than one billion transistors onto a single monolithic die. Given the continuing and fast miniaturization of transistor feature size, global wirelength is becoming a major hindrance in keeping performance scalable since its relative delay to the gate delay gradually worsens as technology continues to shrink. Local wirelength, on the other hand, will scale with a marginal impact in adding an extra delay with respect to the same process technology [1]. Despite the use of different materials, device structures, circuit techniques, and novel architectures including nanotechnology, this global interconnect limit still persists due to the nature of device physics and inflicts a substantial performance impact for chips manufactured with deep submicron technology. In particular, microprocessors that keep pushing the envelope of high performance as the primary design objective are especially more vulnerable to the wire delay problem.

For the last decade, due to the dramatic advancement of microelectronics and manufacturing technology, computer architects were able to improve processor performance simply by adding more computing capability and increasing resource capacity by, for example, increasing cache sizes and hierarchy, enlarging reorder buffer, widening issue width, improving speculation with very complex branch predictors, to name a few. All of these architecture enhancements effectively resulted in higher processor performance in the past. On the other hand, computer-aided design (CAD) tool developers and circuit designers try to reduce the clock period as much as possible, paying little attention to the entire design at the architectural level. With an increasing impact of global wirelength, however, such design methods could lead to less optimal designs, if not totally ineffective, due to the huge intrachip communication latency, and need to be largely changed by taking the wires into account. While it is true that many modern CAD tools consider the impact of interconnect on area, performance, power, and signal integrity, these tools fail to consider the dynamic behavior of applications running on target designs. This approach in turn may mislead the optimization process by letting it focus on a set of wires that are not used often during the execution of applications.

In this article, we advocate the coalition of architecture design and physical design. By considering both simultaneously, we expect to achieve a much better overall performance improvement for microprocessors designed using deep submicron technology. Here, we propose profile-guided microarchitectural floor planning for architectural designers to consider physical location information during the design phase. The contribution of this article is twofold.

1) We propose a design methodology that builds a bridge between microarchitectural design and physical design to address wire delay issues more effectively. Our design flow identifies a set of wires that are frequently used during a cycle-accurate architectural simulation and guides the subsequent module floor planning process to effectively optimize these critical wires.
2) Our profile-guided floor planner obtains a) 5%–40% average instructions per cycle (IPC) improvement when the clock frequency is fixed, and b) higher throughput, measured in terms of billion instructions per second (BIPS), than the conventional wirelength-based floor planner.

Manuscript received July 30, 2004; revised December 30, 2004. This work was supported by the National Science Foundation under Grants CCF-0326396, CNS-0325536, and CNS-0411149. This paper was recommended by Associate Editor M. D. F. Wong.

M. Ekpanyapong, J. R. Minz, H.-H. S. Lee, and S. K. Lim are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA.

T. Watewai is with the Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720 USA.

Digital Object Identifier 10.1109/TCAD.2005.855971

The rest of this article is organized as follows. Section II discusses some related work. Section II-B overviews the implication of IPC and clock speed, and FF insertion. Section III presents the overall design flow and problem formulation. Section IV introduces our profile-guided floor planning and its mathematical foundation. A description of our microarchitectural framework follows in Section V. Then we show our experimental results in Section VI. Finally, we conclude this work in Section VII.

II. Preliminaries

A. Related Work

With a growing concern in global wire delay, there exists a number of research efforts focusing on different aspects including circuits, microarchitectures, and a collaboration between logic synthesis and physical design. Agarwal *et al.* [2] raised an issue of wirelength impact in designing conventional microarchitecture. They showed that reducing the feature size and increasing the clock rate do not necessarily imply an overall performance improvement for deep submicron processor designs. Cong *et al.* [3] confirmed their observation and showed that without considering clock speed, IPC, a widely used performance metric in architecture research, can be misleading in evaluating the performance of next-generation processors. Thus, the number of flip-flops used for interconnect pipelining has been estimated at the full-chip level in [4] and [5] in order to address wire problems effectively.

More recently, Sankaralingam *et al.* [6] proposed a new data-bypassing mechanism that enhances the performance when multicycle bypassing delays are needed in a processor. Cong *et al.* [7] proposed a grid-based microarchitecture that supports multicycle on-chip communication. They also proposed layout-driven architectural synthesis algorithms for multicycle communication, including scheduling-driven placement and placement-driven simultaneous scheduling with rebinding. In logic synthesis, novel techniques [8] were proposed to improve the performance by applying wiring-aware logic synthesis. These techniques provide a location information in the phase of logic synthesis, leading to an overall performance improvement. However, postponing the optimization until the end of the logic synthesis phase can be time consuming and inapplicable to custom designs. Casu and Macchiarulo [9] performed FF insertion for throughput improvement using a distance-based cost function in simulated-annealing-based floor planning. Long *et al.* [10] developed an efficient table-lookup-based model to quickly estimate IPC with interconnect pipelining to guide their simulated-annealing-based floor planner.

B. Wire Delay Issues

Ho *et al.* [1] classified wires into three categories based on their delay impact: 1) wires that scale in length, such as local wires within logic blocks; 2) wires that do not scale in length and is superlinear when the feature size is reduced; and 3) repeated wires, i.e., long wires with repeated buffers inserted on them. The propagation delay of a repeated wire can be represented by

$$D = 0.7n\left[\frac{R_d\left[w(\beta+1)(C_d+C_g)+lC_w\right]}{w} + l^2\frac{R_wC_w}{2} + lR_ww(\beta+1)C_g\right]$$

where R_d is the driver resistance, w is the width of the driver transistor, C_d and C_g are diffusion and gate capacitance per unit width, R_w and C_w are wire resistance and capacitance per unit length, l is the repeater segment length, and β is the pMOS to nMOS sizing ratio.

In next-generation deep submicron processor design, it is likely that repeaters will be inserted frequently on global wires to prevent the wire delays from becoming nonlinear. In this article, we assume that repeated wires are dominant, and we examine their performance impact from the perspective of floor planning. Based on the predicted values of resistance, capacitance, and other parasitic parameters from [1] and [11], repeated wire delay is approximated to be 80 pS/mm for the 30-nm technology. This is used as the baseline for our discussion in Section IV-A. Note that a FO4 gate delay for 30 nm is approximately 17 pS.

Flip-flop insertion is a technique to alleviate the impact of wire delay for achieving the target clock frequency. A deeper pipelining enabled by flip-flop insertion results in a higher clock frequency and higher BIPS [5]. Nevertheless, the improvement cannot always be anticipated especially for designs with a small feature size. The reason is that flip-flop insertion may cause IPC degradation from its increased latency, as shown in Fig. 1. Therefore, inserting flip-flops without a meticulous measure does not guarantee an overall performance improvement. It is possible that a change in the number of flip-flops on wires may require a redesign of the control units and an update on the microarchitectural floor planning. We solve this problem by allocating a larger size for the control units such that there is a room to accommodate any additional change on the control unit design.

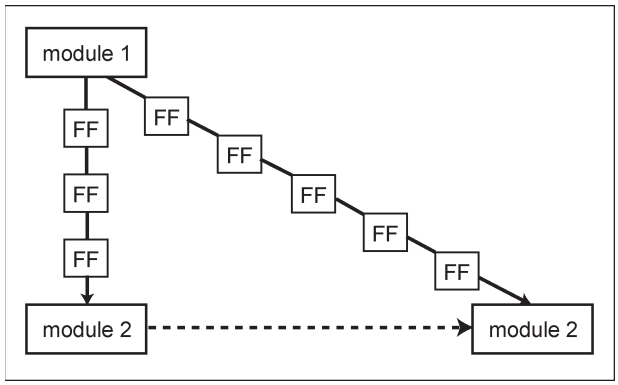


Fig. 1. Impact of wire delay and FF insertion on module access latency. Moving module 2 from left to right causes the length of wire to increase. This requires more FFs to be inserted in order to meet the clock period constraint, which in turn causes the access latency to increase.

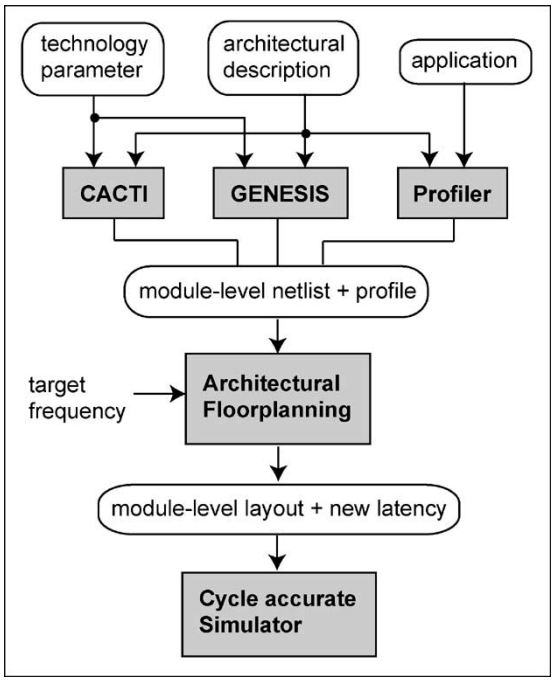


Fig. 2. Overview of our profile-guided microarchitectural floor planning framework.

III. Problem Formulation

A. Design Flow

An overview of our profile-guided microarchitectural floor planning is shown in Fig. 2. Our framework combines technology scaling parameters and the execution profiling information of applications to guide the floor planning step of a given microarchitecture design. First, a machine description is provided as an input to the microarchitecture simulator, in which profiling counters were instrumented for bookkeeping module-to-module communication. Then, a cycle-accurate simulation is performed to collect and extract the amount of interconnection traffic between modules for a given benchmark program. For cache-like or buffer-like structures, the area and module delay are estimated using an industry tool from HP Western Research Labs called CACTI [12]. For scaling other structures such as ALUs, we use GENESYS [13] developed at Georgia Institute of Technology.

After the timing and area information of each module is collected, we feed the module-level netlist, statistical interconnection traffic, and a processor target frequency into our profile-guided floor planner. The purpose is to generate a floor plan from which all the intermodule latency values of the given microarchitecture are derived. With the new latency values, the architecture performance simulation is performed to obtain more realistic and accurate IPC and BIPS data.[1]

CACTI is an integrated simulator for estimating access time, cycle time, area, aspect ratio, and power model for cache/buffer-like structures. The inputs of CACTI include cache size, block size, number of associativity, number of read/write ports, and technology parameters. By exploiting technology parameters, CACTI can observe the impact of different technologies on performance. GENESYS is also developed to study the impact on each architectural module under different technology scalings. However, unlike the cache structure, a noncache structure is harder to model, and the corresponding circuit itself can be changed based on a different transistor size. GENESYS assumes no change in circuit design and estimates module area and delay based on a set of empirical modeling equations. The inputs of GENESYS include the number of transistors, critical path delay, and technology parameters. We use the Verilog model of ARM processor to access the number of transistors in each module. We collect the scaling improvement ratio from GENESYS and store it in our internal table. Note that CACTI and GENESYS are used because of its simplicity in finding module size and delay. More importantly, they allow us to study the impact of technology scaling on the performance of microarchitectural designs.

For profiling, we use the Simplescalar 3.0 tool suite [14] to collect the number of accesses when each module is accessed by another module. After the profiling is completed, we normalize the traffic values so that they range from [0, 1]. The inputs of the profiling simulator include a target application and a train input set. Note that we could use multiple input sets to collect multiple access patterns and use their average behavior for more accurate frequency values.

B. Problem Formulation

Given a set of microarchitectural modules and a netlist that specifies the connectivity among these modules, our profile-guided microarchitectural floor planner tries to place each module such that 1) there is no overlap among the modules and 2) a user-specified clock period constraint is satisfied. Our objective is to minimize the overall execution time of a given processor. We use BIPS for this purpose, which is an average number of billion instructions that can be issued in 1 s. IPC represents an average number of instructions that can be issued in one clock cycle. In very large scale integration (VLSI) circuit design, clock period is used to evaluate the quality of logic synthesis and physical design solutions, which equals to the longest path delay from these solutions. F (clock frequency), which denotes the total number of cycles per second, is the reciprocal of clock period. Finally, $\mathrm{BIPS} = \mathrm{IPC} \times F$ if F is in the gigahertz range. In this article, we maximize IPC under a clock frequency constraint so that the overall BIPS is maximized.

IV. Profile-Guided Floor Planning

In this section, we present mathematical programming models for our profile-guided microarchitectural floor planner. First, we present a mixed integer nonlinear programming (MINP) model that minimizes the weighted wirelength under a performance constraint. Since finding an optimal solution for an MINP model is NP-hard, we apply various techniques to

[1]Note that this entire design flow can be repeated multiple times to evaluate multiple microarchitectural designs in terms of IPC, BIPS, and clock cycle, thereby enabling microarchitectural design space exploration. This is the focus of our ongoing work.

MINP Formulation

$$\text{Minimize} \sum_{(i,j)\in E} [U_1 \cdot \lambda_{ij} z_{ij} + U_2 \cdot (X_{ij} + Y_{ij})] + U_3 \cdot X_m \quad (1)$$

Subject to:

$$\begin{aligned}
z_{ij} &\geq \frac{g_i + \alpha(X_{ij} + Y_{ij})}{L},\ (i,j) \in E && (2)\\
X_{ij} &\geq x_i - x_j,\ (i,j) \in E && (3)\\
X_{ij} &\geq x_j - x_i,\ (i,j) \in E && (4)\\
Y_{ij} &\geq y_i - y_j,\ (i,j) \in E && (5)\\
Y_{ij} &\geq y_j - y_i,\ (i,j) \in E && (6)\\
z_{ij} &\geq f_{ij},\ (i,j) \in E && (7)\\
X_m &\geq x_i\ i \in N && (8)\\
A \cdot X_m &\geq y_i,\ i \in N && (9)\\
x_i + w_i &\leq x_j - w_j + M(c_{ij} + d_{ij}),\ i < j \in N && (10)\\
x_i - w_i &\geq x_j + w_j - M(1 + c_{ij} - d_{ij}),\ i < j \in N && (11)\\
y_i + \frac{a_i}{4w_i} &\leq y_j - \frac{a_j}{4w_j} + M(1 - c_{ij} + d_{ij}),\ i < j \in N && (12)\\
y_i - \frac{a_i}{4w_i} &\geq y_j + \frac{a_j}{4w_j} - M(2 - c_{ij} - d_{ij}),\ i < j \in N && (13)\\
w_{min,i} &\leq w_i \leq w_{max,i},\ i \in N && (14)\\
x_i, y_i &\geq 0,\ i \in N && (15)\\
c_{ij}, d_{ij} &\in \{0,1\},\ i < j \in N && (16)\\
z_{ij} &\in \mathbb{N},\ (i,j) \in E && (17)
\end{aligned}$$

Fig. 3. MINP formulation of our profile-guided microarchitectural floor planning.

convert the MINP model to a mixed integer linear programming (MILP) model and then to a linear programming (LP) model.

A. Mixed Integer Programming Model

We model the input module-level netlist with a directed graph $G(N, E)$, where N denotes the set of all flexible modules and E denotes the set of directed edges. A directed edge (i, j) represents a wire from module i to module j. Each multipin net in the netlist is decomposed into a set of source–sink module pairs. Let α be the repeated wire delay per 1 mm as discussed in Section II-B, and λ_{ij} be the statistical traffic on wire (i, j), i.e., the normalized access counts from module i to module j. g_i is the delay of module i. $w_{\min,i}$ and $w_{\max,i}$ denote the minimum and maximum half width of module i. The area of module i is denoted by a_i. Finally, f_{ij} is the number of flip-flops on wire (i, j) in the given netlist.

Let L $(= 1/\text{clock\_frequency})$ denote the target cycle period of the given microarchitectural design, which is an input to our floor planner. In the MINP model, we need to determine the values for the following decision variables: x_i, y_i, w_i, h_i, and z_{ij}. Let (x_i, y_i) denote the location of the center of module i in $\mathbb{R}^2_+$ space. X_{ij} and Y_{ij} represent $|x_i - x_j|$ and $|y_i - y_j|$ between module i and j, respectively. z_{ij} is the number of flip-flops on wire (i, j) after FF insertion. w_i and h_i denote the half width and the half height of module i. To avoid overlapping between two modules i and j, where $i < j$, we need a set of binary variables so that at least one of the following holds:

$$\begin{aligned}
x_i + w_i \leq x_j - w_j, &\qquad i \text{ is on the left of } j\\
x_i - w_i \geq x_j + w_j, &\qquad i \text{ is on the right of } j\\
y_i + \frac{a_i}{4w_i} \leq y_j - \frac{a_j}{4w_j}, &\qquad i \text{ is below } j\\
y_i - \frac{a_i}{4w_i} \geq y_j + \frac{a_j}{4w_j}, &\qquad i \text{ is above } j.
\end{aligned}$$

We thus let c_{ij} and d_{ij} be the binary variables such that

$$(c_{ij}, d_{ij}) = \begin{cases} (0,0), & \text{if } i \text{ is on the left of } j\\ (0,1), & \text{if } i \text{ is on the right of } j\\ (1,0), & \text{if } i \text{ is below } j\\ (1,1), & \text{if } i \text{ is above } j. \end{cases}$$

Fig. 3 shows the MINP formulation of our profile-guided microarchitectural floor planning. A weighted delay of an edge (i, j) is defined to be $\lambda_{ij} \times z_{ij}$, where the weight λ_{ij} is based on module access frequency. z_{ij} is the number of FFs on the wire. X_m and Y_m are the maximum values among all x_i and y_i, respectively. $A = Y_m/X_m$ is the aspect ratio of the chip. Since the area objective $X_m \cdot Y_m$ is nonlinear, we linearize it by minimizing X_m while maintaining $A \cdot Y_m > y_i$ for all modules. Thus, the objective of our MINP formulation [$=$ (1) in Fig. 3] is to minimize the weighted sum of the 1) weighted

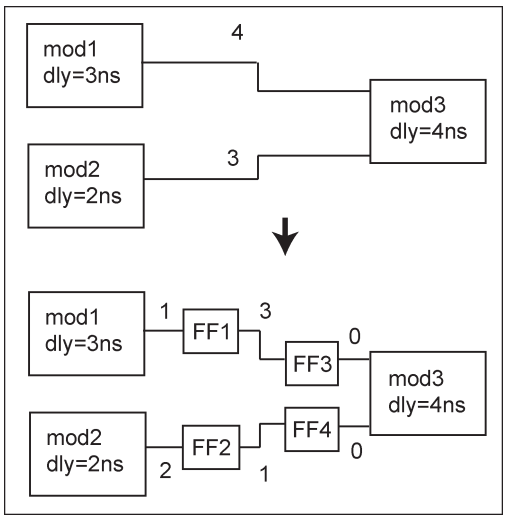


Fig. 4. Number of FF constraint. The numbers on the wires represent their delay values. Assuming that the clock period constraint is 4, constraint (2) decides to put two FFs on both wires [$\lceil(3+4)/4=2\rceil$ and $\lceil(2+3)/4=2\rceil$]. FF3 and FF4 are added so that they provide their values to module 3 at the same time (= input synchronization).

delay among all wires, 2) total wirelength, and 3) total area. U_1, U_2, and U_3 are user-specified weighting constants.

Constraint (2) in Fig. 3 is obtained from the definition of latency (Fig. 4). If there is no FF on a wire (i,j), the delay of this wire is calculated as $d(i,j)=\alpha(X_{ij}+Y_{ij})$. Then, $g_i+d(i,j)$ represents the latency of module i accessing module j. Since L denotes the clock period constraint, $(g_i+d(i,j))/L$ denotes the minimum number of FFs required on (i,j) in order to satisfy the clock period constraint.[2] Nonoverlapping constraints are given in (3)–(6). Constraint (7) requires that we do not remove any existing FFs from the wires. Constraints (8) and (9) are related to area minimization as mentioned previously. Constraints (10)–(13) represent relative positions among the modules. Constraint (14) specifies the possible range of the half width of each module. Constraint (15) is a nonnegative constraint for the module location. Constraint (16) states that (c_{ij}, d_{ij}) are binary variables. Finally, constraint (17) specifies that the number of flip-flops must be an integer. Also note that M is a sufficiently large number.

In our floor planning, we allow w_i (half-width) and h_i (half-height) to vary but require that a_i (area) remain fixed, i.e., $4w_i \cdot h_i = a_i$. We linearize this nonlinear relation [= constraint (12) and (13)] by letting $h_i = m_i \cdot w_i + k_i$ instead of $h_i = a_i/4w_i$, where $m_i = a_i/(4w_{\min,i} \cdot w_{\max,i})$ and $k_i = a_i/(4w_{\max,i} + 4w_{\min,j})$. An illustration is shown in Fig. 5. Note that this approximation guarantees that the solution in the approximated model is feasible due to an overapproximation. It is possible to better approximate h_i by using multiple linear lines instead of one. However, this more accurate model comes at a cost of more constraints in our MILP model. Since area is not our primary concern, we do not attempt this more accurate model

[2]In fact, this formula includes one extra FF that is inserted at the end of the wire unlike the conventional retiming [15]. In case a module has multiple fan-in wires, these extra FFs will ensure that the data transfer on these wires are synchronized so that the input values for the module are available at the same time.

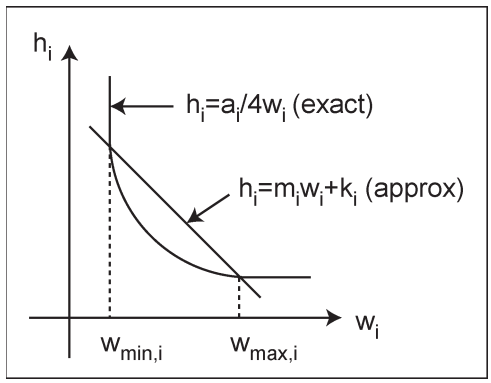


Fig. 5. Module height approximation for MINP to MILP conversion, where $m_i = a_i/(4w_{\min,i} \cdot w_{\max,i})$ and $k_i = a_i/(4w_{\max,i} + 4_{\min,j})$.

MILP Formulation

Minimize $\sum_{(i,j)\in E} [U_1 \cdot \lambda_{ij} z_{ij} + U_2 \cdot (X_{ij} + Y_{ij})] + U_3 \cdot X_m$

Subject to (2)—(11), (14)—(17) and the following:

$$y_i + m_i w_i + k_i \leq y_j - m_j w_j - k_j + M(1 - c_{ij} + d_{ij}),\ i < j \quad (18)$$

$$y_i - m_i w_i - k_i \geq y_j + m_j w_j + k_j - M(2 - c_{ij} - d_{ij}),\ i < j \quad (19)$$

Fig. 6. MILP formulation of our profile-guided microarchitectural floor planning.

during our optimization. However, we perform compaction as a postprocess as discussed in Section IV-B to fine tune the overall floor plan area. The resulting MILP formulation is shown in Fig. 6.

B. LP Model

Even with the MILP conversion, the floor planning problem still remains NP-hard. Specifically, z_{ij}, c_{ij}, and d_{ij} are integer variables. To remedy this problem, we further relax MILP into the LP model.[3] We adopt a partitioning method similar to the one described in [16]. To relax the integrality while maintaining the feasibility and staying close to the optimal solution, we first relax the integrality of z_{ij} to be a real number. We also solve several LP problems to determine the relative positions among the modules, i.e., c_{ij} and d_{ij}. If these c_{ij} and d_{ij} are known and z_{ij} can take real values, our MILP model shown in Fig. 6 becomes LP.

Our LP-based slicing floor planning algorithm consists of multiple iterations, where at each iteration a cutline is inserted to divide a region (alternatively called a block) into two subblocks. We start the algorithm by creating a large block containing all modules. At each iteration, we choose a block, divide it into two subblocks, and perform module floor planning again so that the objective is further minimized. After the floor plan, the modules in the chosen block should be enclosed by

[3]If a near-optimal solution is required, MILP is the better approach than LP. However, MILP in general requires an excessive amount of computational power to solve.

LP-based Slicing Floorplanning

Initialize $B(1) = \{1\}, M_1(1) = N$;
for ($u = 1$ to $log_2 N$)
  for (*count* =1 to *run*)
    Choose a block j and divide it into two;
    Specify $S_{jk}(u), (\bar{x}_{jk}, \bar{y}_{jk})$;
    Solve LP(u);
    Update $B(u+1), M_j(u+1), r_j, l_j, t_j, b_j$;
  Project best solution for $u+1$;
Obtain c_{ij}, d_{ij} from prior slicing floorplan;
Solve MILP;
return x_i, y_i, w_i, h_i, z_{ij};

Fig. 7. Description of our LP-based slicing floor planning algorithm. We perform a top-down recursive bipartitioning and solve LP-based floor planning at each iteration. We then solve MILP again after the last iteration using the slicing floor planning result.

LP Formulation

$$\text{Minimize} \sum_{(i,j)\in E} (U_1 \cdot \lambda_{ij} z_{ij} + U_2 \cdot (X_{ij} + Y_{ij}))$$

Subject to (2)—(7), (14)—(15) and the following:

$$x_i + w_i \leq r_j,\ i \in M_j(u), j \in B(u) \quad (20)$$
$$x_i - w_i \geq l_j,\ i \in M_j(u), j \in B(u) \quad (21)$$
$$y_i + m_i w_i + k_i \leq t_j,\ i \in M_j(u), j \in B(u) \quad (22)$$
$$y_i - m_i w_i - k_i \geq b_j,\ i \in M_j(u), j \in B(u) \quad (23)$$

And for $k \in \{1,2\}, j \in B(u)$

$$\sum_{i\in S_{jk}(u)} a_i x_i = \sum_{i\in S_{jk}(u)} a_i \times \bar{x}_{jk} \quad (24)$$
$$\sum_{i\in S_{jk}(u)} a_i y_i = \sum_{i\in S_{jk}(u)} a_i \times \bar{y}_{jk} \quad (25)$$

Fig. 8. LP formulation of our profile-guided microarchitectural floor planning. This LP is used to perform floor planning at iteration u of the main algorithm shown in Fig. 7.

the block boundaries and the area-weighted mean (= center of gravity) among all modules in each subblock should correspond to the center location of the subblock. In addition, the user-specified clock period L constraint needs to be satisfied, i.e., the longest combinational path delay should be less than L. We terminate the iteration when each block contains exactly one module. Lastly, we obtain the relative positions among the modules from the slicing floor planning result and solve MILP (shown in Fig. 6) again. This time, however, the MILP formulation becomes an LP since c_{ij} and d_{ij} are already determined and z_{ij} are still allowed to have noninteger values. Note that each iteration can be repeated multiple times to obtain different slicing floor plans. This is due to the fact that there exists multiple solutions that satisfy the boundary and center of gravity constraints during each bipartitioning. Thus, we perform each bipartitioning several times and pick the best solution in terms of the total wirelength for the next iteration.

Fig. 7 shows a description of our LP-based slicing floor planning algorithm. $B(u)$ denotes the set of all blocks at iteration u and $M_j(u)$ denotes the set of all modules currently in block j at iteration u. $S_{jk}(u)$ is the set of modules assigned to the center of subblock k ($k \in \{1,2\}$) contained in block j at iteration u. We denote the center of subblock k contained in block j by $(\bar{x}_{jk}, \bar{y}_{jk})$. We note that balancing the area of the two subblocks at each bipartitioning helps reduce the overall floor plan area. Thus, each cutline bisects the initial block, and $\bar{x}_{jk}$ and $\bar{y}_{jk}$ correspond to the center location of the two subblocks. Finally, let r_j, l_j, t_j, b_j denote the right, left, top, and bottom boundary of block j. Fig. 8 shows the LP formulation we solve at each iteration of the slicing floor planning.[4] The block boundary constraints (20)–(23) require that all modules in the block be enclosed by these block boundaries. The center of gravity constraints (24), (25) require that the area-weighted mean (= center of gravity) among all modules in each subblock corresponds to the center of the subblock. Fig. 9 shows an example of our LP-based floor planning algorithm.

The main objective of our slicing floor planning is to determine c_{ij} and d_{ij}, i.e., the relative position among the modules. Note that the recursive bipartitioning scheme may cause multiple relative positions between a pair of modules. In Fig. 10, module a can be either on the left of module b, i.e., $(c_{ab}, d_{ab}) = (0,0)$, or above module b, i.e., $(c_{ab}, d_{ab}) = (1,1)$. However, one has to choose between (0, 0) and (1, 1) to be used in our MILP formulation (see Fig. 6). Note that this decision has a nontrivial impact on the overall floor plan area and thus needs to be done carefully. In our area compaction heuristic, we make a decision based on the first cut that separates a pair of module. Modules a and b are first separated by the vertical cut (cut $n-1$) in Fig. 10, and then by the horizontal cut (cut n). In this case, we choose $(c_{ab}, d_{ab}) = (0,0)$, i.e., a is on the left of b since it is the vertical cut that first separates a and b during the top-down recursive bipartitioning process. Our related experiment indicates that this scheme generates highly a compact floor plan.

V. Simulation Infrastructure

Simplescalar 3.0 tool suite [14] was used as our architecture simulator for both communication profile collection and performance simulation. The detailed microarchitecture is illustrated in Fig. 11 and four microarchitecture configurations used in our experiments are listed in Table I.[5] Each functional block in Fig. 11 represents a module used by our floor planner. In order to facilitate the physical-design-driven microarchitectural exploration, a few new features were introduced on top of the baseline machine model. First, the pipeline depth was made configurable. The instruction fetch unit of the simulator was

[4]Note that we dropped the area objective ($= U_3 \cdot X_m$ in Fig. 6) from our LP formulation. Having three competing objectives may distract the optimization process, so we focus on performance and wirelength during the optimization and handle area with our compaction scheme during a postprocess. We observed that this approach produced better results than optimizing three objectives simultaneously.

[5]We believe that architects will use more and more on-chip L3 caches [17] in future designs due to the ever-increasing number of transistors available and the ever-increasing disparity between memory access and on-chip cache access latency. Our cfg6 and cfg7 are designed to study the impact of on-chip L3 cache.

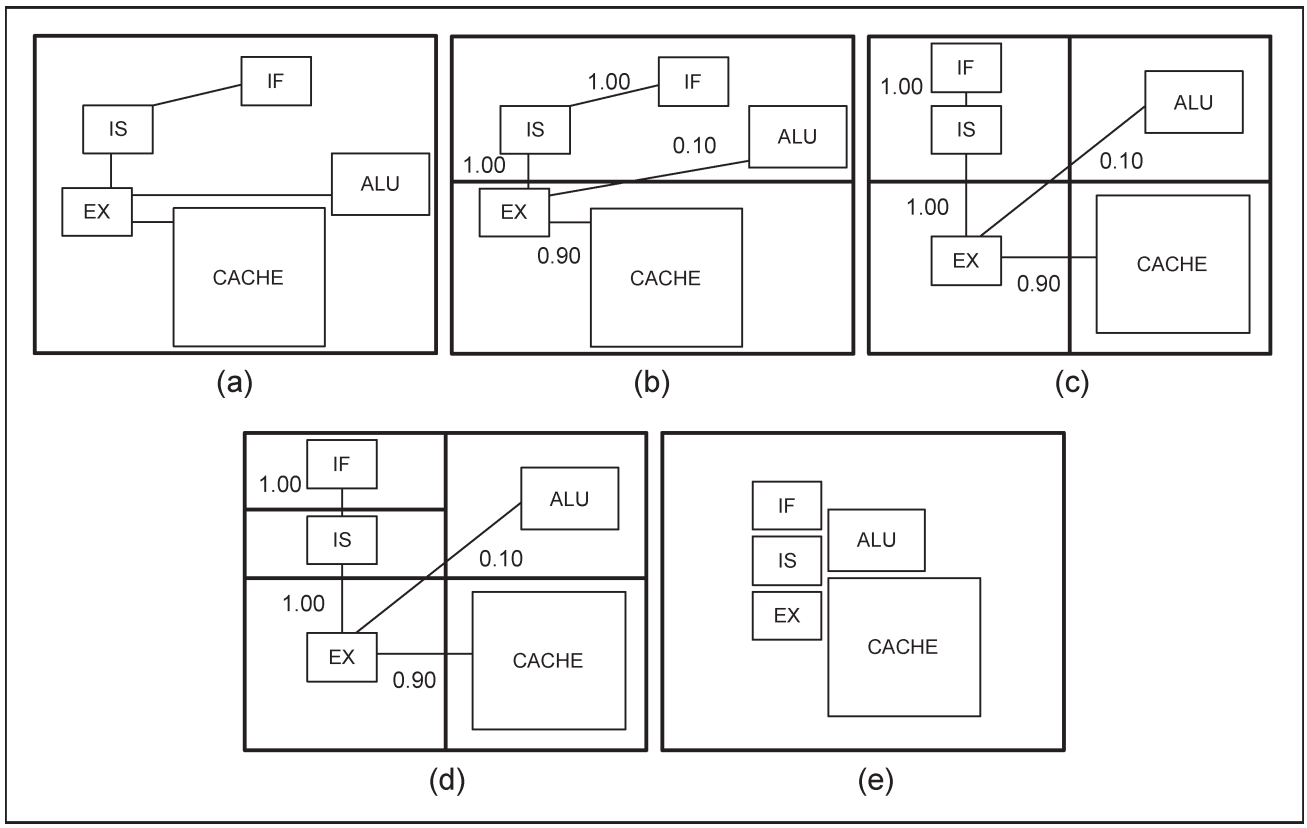


Fig. 9. Illustration of recursive bipartitioning for the relaxation of integer constraint. Recursive bipartitioning is performed until each module is placed in one block. The objective is to minimize the weighted wirelength under the boundary, center of mass, and clock period constraints. After the recursive bipartitioning is done, the relative positions among the modules (left, right, above, and below) are fed to our integer programming for the final iteration.

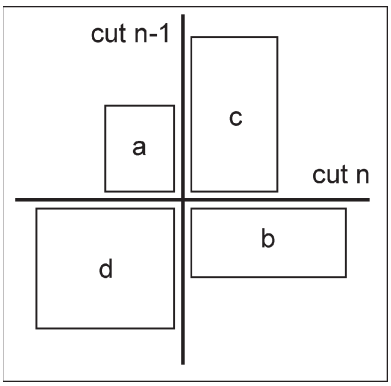


Fig. 10. Illustration of our area compaction scheme.

modified to accommodate any desired pipeline depth, providing a capability to study the effects of lengthening the processor pipeline. In terms of performance, the pipeline depth has a direct impact on the operational frequency of the processor. For example, depending on the placement of each functional block by the floor planner, signal routing might take more than one pipeline stage to complete. The number of pipeline stages is also affected by our extension of the functional units. In this work, the pipeline depth is varied from 15 to 22 stages. In the modified simulator, the functional units or execution resources are completely configurable in terms of operation and issue latency, and can be specified as a configuration input. For instance, the same functional unit design (e.g., integer ALU) can have nonuniform latencies depending on each individual unit's final placement. In addition, a third-level cache was added. The primary microarchitectural parameters are elaborated as follows.

- Machine width: The maximum number of issuable IPC.
- Bpred: The branch predictor. In this study, we used a hybrid branch predictor containing a 2-bit counter-based predictor and a Gshare predictor with a choice meta predictor [18]. Each number of Bpred entry in Table I is the number of entries (2^N) of the 2-bit counter array and the meta table, where N is the size of the global branch history.
- BTB: The branch target buffer. Indexed by the program counter, each BTB entry keeps the branch instruction address and its associated branch target address for accelerating instruction fetching.
- RUU: The register update unit [19], which combines the functionality of a register alias table, a reservation station, and a reorder buffer for maintaining instruction ordering and supporting precise interrupt. Each RUU entry also contains a physical register for enabling register renaming.
- Caches: The first level cache contains a split instruction and data cache while the second and third levels are both unified caches.
- TLB: The translation look-aside buffer for fast address translation. Split instruction and data TLBs are assumed.
- ALU and FPU: The ALU only executes integer operations while the FPU performs floating-pointing arithmetic.

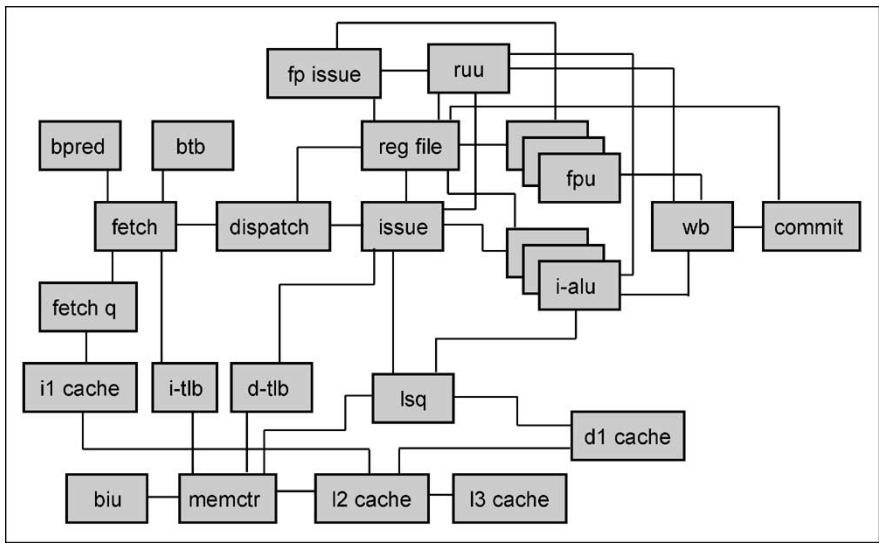


Fig. 11. Processor microarchitecture model.

TABLE I
MICROARCHITECTURE CONFIGURATIONS USED IN OUR STUDY

μ-arch modules	Size configuration							# of bits
	cfg1	cfg2	cfg3	cfg4	cfg5	cfg6	cfg7	
M-width	2	4	4	8	8	8	8	-
Bpred	128	512	512	512	512	512	512	2
BTB	128	512	512	512	512	512	512	96
RUU	64	128	256	256	256	512	512	168
L1 I-$	8K	64K	64K	64K	64K	8K	8K	512
L1 D-$	8K	64K	64K	64K	64K	8K	8K	512
L2 U-$	64K	512K	512K	512K	512K	128K	128K	1024
L3 U-$	-	-	-	-	-	2M	2M	1024
ITLB	32	128	128	128	128	128	128	112
DTLB	32	128	128	128	128	128	128	112
ALU	2	4	4	4	6	8	8	-
FPU	1	2	2	2	3	4	4	-
LSQ	16	64	128	128	128	128	128	84
M-port	1	4	4	4	6	8	8	-

- LSQ: The load/store queue for enabling load speculation and disambiguating memory addresses.
- Mem port: The number of ports available in the first level data cache.

Some applications exploit some particular microarchitecture components (e.g., caches, TLBs, or BTB) better than the others, but there is no guarantee that increasing the sizes of these modules will lead to an overall execution time improvement measured in billions of instructions per second (BIPS). Note that the IPC is an architectural metric while the clock period is determined largely by both the degree of pipelining and the targeted process technology. Increasing the sizes of BTBs and TLBs may improve IPC; nevertheless, it could also lengthen the clock period due to the elongated wires of larger structures, thus leading to an overall increase in the total execution time. In order to have a simultaneous view of the frequency and the IPC for attaining optimal performance, we must explore the floor plan and processor's microarchitecture configuration together.

For accurate performance prediction and optimization, wires can no longer be isolated from architecture-level evaluation but must be modeled as units that consume power and have delays. Therefore, provisions were also made to consider wire delays in our simulator. The existing simulator assumes that the communication latency between functional blocks is always one cycle, which no longer holds while operating at extremely high frequency given the increased wire delays and ever-growing die areas. For example, the Pentium 4 processor design has dedicated two pipeline stages for moving signal across the chip due to wire delay [20]. The wire parameters can be captured for architectural optimization through floor planning to a reasonable degree of accuracy.

As aforementioned, in our modified simulator, the intercommunication latencies between function units or data forwarding latencies among different pipeline stages were made configurable, enabling a more realistic IPC projection. For performance evaluation, we use the information provided by the floor planner to derive essential simulation parameters such as pipeline depth and communication/forwarding latencies. The interfunctional unit latency is a function of the distance between units in the floor plan and the number of flip-flops between modules. If the floor plan has been optimized for clock speed, the pipeline depth of the processor reflects it. In our experiments, we expect an improvement in performance (in architectural simulation) if the frequency of forwarding traffic between units is included in our floor plan formulation. The forwarding frequency-driven floor planning tries to place highly communicated units closer together, minimizing their latencies as a function of distance.

Table II illustrates the range of intermodule access latency values measured in terms of clock cycle. These values are averaged over the wire latency values we obtained from multiple profile-guided floor planning solutions. Let l_i denote the total number of possible latency values wire i can have. For example, v_1 for the first communication wire (= branch predictor penalty) from Table II is 7. Then, the size of the solution space for module floor planning is $\prod_{i=1}^{M} l_i$ for M wires. Therefore, a brute-force-based search for finding an optimal solution proves to be nonpractical due to the exponential size of the solution space.

TABLE II
ACCESS LATENCY RANGE (IN CLOCK CYCLES) AMONG THE MICROARCHITECTURE MODULES

communication	range
Branch predictor penalty	5 - 11
Pipeline Depth	9 - 33
Inter ALU communication	1-8
Inter FPU communication	4-20
RUU access from ALU	1-7
RUU access from FPU	4-20
ALU access from RUU	2-11
FPU access from RUU	2-15
IL1 access	2-11
DL1 access	2-11
L2 access	5-20
L3 access	16-79

VI. EXPERIMENTAL RESULTS

Our floor planning algorithms are implemented in C, compiled with gcc with -O3, integrated with Simplescalar tool set, and executed on Pentium IV 2.4-GHz machines. We use lp_solve [21] to solve our linear programs. We performed experiments on ten SPEC2000 benchmarks (gzip, vpr, mcf, gap, bzip2, twolf, swim, art, equake, lucas), six from the integer suite and four from the floating point one. The training input set was used for profile collection while IPC performance results were gathered using the reference input set. Each simulation was fast-forwarded by 200 million instructions and then simulated for 100 million instructions. The maximum execution time spent in both floor planning and simulation was less than 1 h for each benchmark. We use the following algorithms in our experiment.

1) WL: wirelength-driven floor planning. We obtain this algorithm by setting $U_1 = 0$ and $U_2 > 0$ in Fig. 8. Area compaction is not used.
2) AWL: area/wirelength-driven floor planning. We use area compaction as a postprocess.
3) PGFi: pure profile-guided floor planning. We obtain this algorithm by setting $U_1 > 0$ and $U_2 = 0$ in Fig. 8. Area compaction is not used.
4) PGF: profile-guided floor planning with area/wirelength optimization. We obtain this algorithm by setting $U_1 > 0$ and $U_2 > 0$ in Fig. 8. Area compaction is used.

We report the average IPC, BIPS, wirelength, and area results among all ten benchmark applications for each of the seven microarchitectural configurations shown in Table I.

Fig. 12 shows the IPC comparison between WL and PGFi floor planning algorithms. We observe that PGFi consistently obtains higher IPC values than WL for all seven configurations, where the improvement ranges from 11%–39%. In addition, the IPC improvement is more visible from more complex designs (cfg6 and cfg7). This is a strong evidence that exploiting the dynamic intermodule interconnect traffic information during floor planning is crucial in improving the performance of microarchitecture designs measured by IPC. The main reason behind this success is the shorter access latency among heavily communicated functional modules. Fig. 13 shows the IPC comparison between AWL and PGF floor planning algorithms, where we use area compaction for both algorithms. We note that the IPC advantage of PGF over AWL is reduced to 7%–19% compared with PGFi versus WL. This is due to the tradeoff existing between performance and area, where PGF lost more performance than AWL did from the additional area/wirelength objectives. In fact, PGF obtained 40% better wirelength and 50% better area results compared to PGFi as revealed in the following discussions.

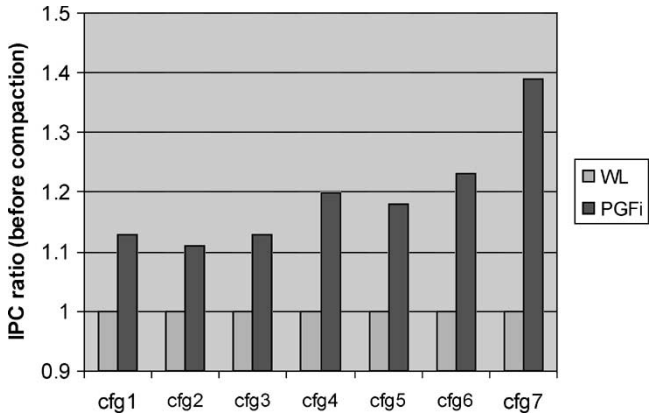


Fig. 12. IPC comparison between wirelength-driven (WL) and profile-guided (PGFi) floor planning algorithm. Area compaction is not used.

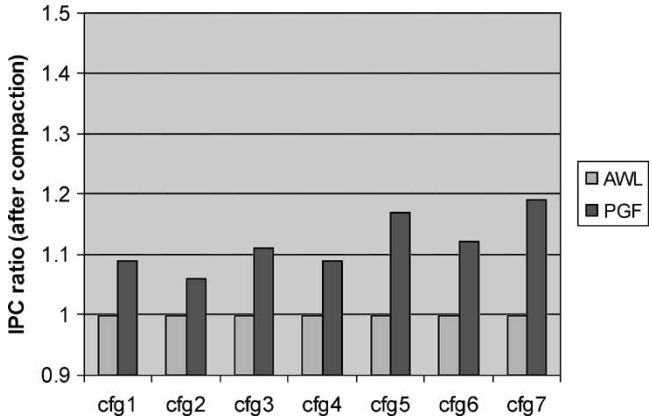


Fig. 13. IPC comparison between area/wirelength-driven (AWL) and profile/area/wirelength optimization (PGF) floor planning algorithm. Area compaction is used.

Fig. 14 shows wirelength comparison among WL, AWL, and PGF floor planners. These results are normalized to the PGFi algorithm. From the PGF results, we see that the wirelength has reduced by almost 40% on average compared to PGFi. This is primarily due to the wirelength objective used in PGF. In addition, area compaction had a positive impact on reducing the wirelength further. It is interesting to note that AWL obtained approximately 10% better wirelength results than WL, suggesting a positive correlation between area and wirelength in microarchitectural floor planning. Compared with AWL, our PGF obtained wirelength results within a tolerable range: 10%–30% worse on average.

Fig. 15 shows area comparison among WL, AWL, and PGF floor planners. These results are normalized to the PGFi algorithm. From the PGF results, we see that area has reduced by almost 50% on average compared to PGFi. This is primarily due to the area compaction we used. In addition, our wirelength objective had a positive impact on reducing the area further.

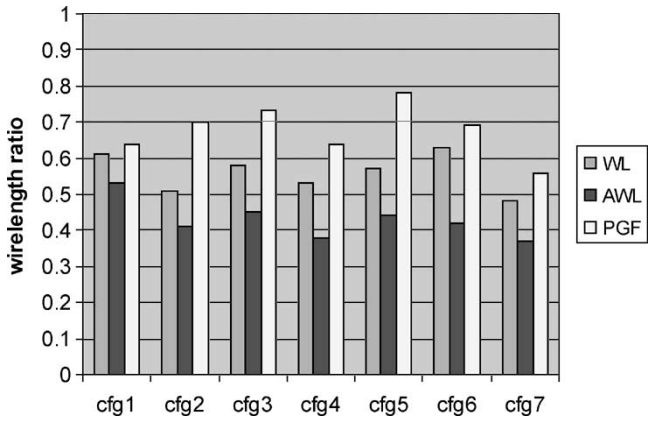


Fig. 14. Wirelength comparison among wirelength-driven (WL), area/wirelength-driven (AWL), and profile/area/wirelength-driven (PGF) floor planners. These results are normalized to profile-only (PGFi) algorithm.

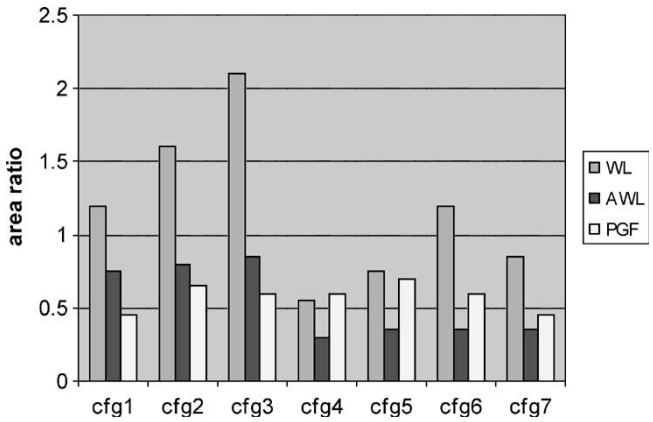


Fig. 15. Area comparison among wirelength-driven (WL), area/wirelength-driven (AWL), and profile/area/wirelength-driven (PGF) floor planners. These results are normalized to profile-only (PGFi) algorithm.

Our area compaction scheme worked well with AWL and generated consistent area improvement. From the AWL versus PGF comparison, we note that PGF obtained better area results for the small configurations (cfg1 to cfg3) while loosing on big configurations (cfg4 to cfg7). However, the gap between AWL and PGF for big configuration is decreasing as the design size increases. We note that the module size is better balanced in a bigger configuration, which helps ease the burden of the floor planner for area optimization. This leads us to believe that partitioning bigger modules into smaller submodules may help on area optimization.

Fig. 16 shows the BIPS comparison among AWL, PGF, and OPT for several future technologies. These results are normalized to the WL algorithm for 5-GHz design. In our OPT algorithm, the delay of all wires are set to zero to obtain upper bounds on BIPS. OPT is not based on any floor planning since the wire delay is completely ignored. The overall trend indicates that the gap between AWL and PGF (BIPS advantage of our PGF) increases as the clock frequency increases. This clearly indicates that the profile information of the target application must be considered to improve the overall throughput of the system. When we move into a higher clock frequency, the gap between PGF and OPT also increases. This shows that there

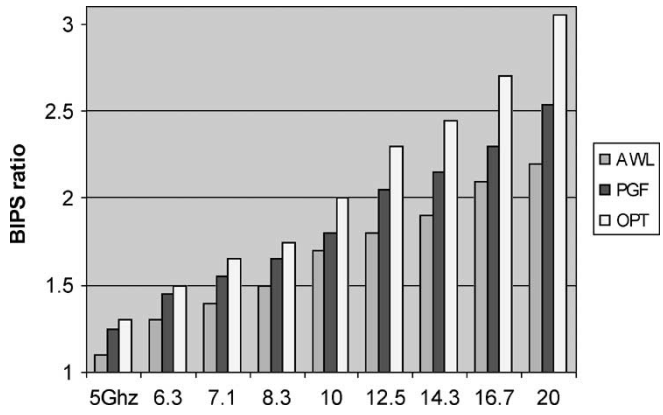


Fig. 16. BIPS comparison among area/wirelength-driven (AWL), profile/area/wirelength-driven (PGF), and lower bound with no wire-delay (LOW) for several future technologies. These results are normalized to wirelength-only (WL) algorithm for 5-GHz design.

TABLE III
RUNTIME BREAKDOWN IN SECONDS

step	runtime
size estimation	2.9
profiling	1133.1
floorplanning	212.6
simulation	998.6

is still a room for improvement either by better floor planning algorithm or additional circuit techniques that alleviate the ever-worsening wire problem.

Table III show a breakdown of the average runtime among various steps involved in our framework. The "size estimation" is the runtime of CACTI and GENESYS combined. The "profiling" is the runtime for collecting access frequency information using a cycle-accurate simulation. The "floor planning" is the runtime for our LP-based floor planning algorithm. Finally, the "simulation" is the runtime for computing the final IPC values from another cycle-accurate simulation. All runtime results are reported in seconds. A snapshot of the final PGF floor plan is shown in Fig. 17.

VII. CONCLUSION

An effective microarchitectural floor planning algorithm can no longer ignore the dynamic communication patterns of applications and the impact of wires on the overall performance. In this article, we proposed a profile-guided microarchitectural floor planner that considers both the impact of wire delay and architectural behavior to reduce the latency of frequent routes inside a processor and to maintain performance scalability. Results show that our profile-guided method obtains huge improvement on IPC and BIPS under the clock period constraint compared to the conventional area/wirelength-based floor planner.

One future research direction is to further optimize the performance by partitioning each functional module into finer submodules. For example, one can partition the register file into several disjoint modules based on the access frequency acquired from the execution profiling. With this partitioning, our floor planner could potentially generate a floor plan with different

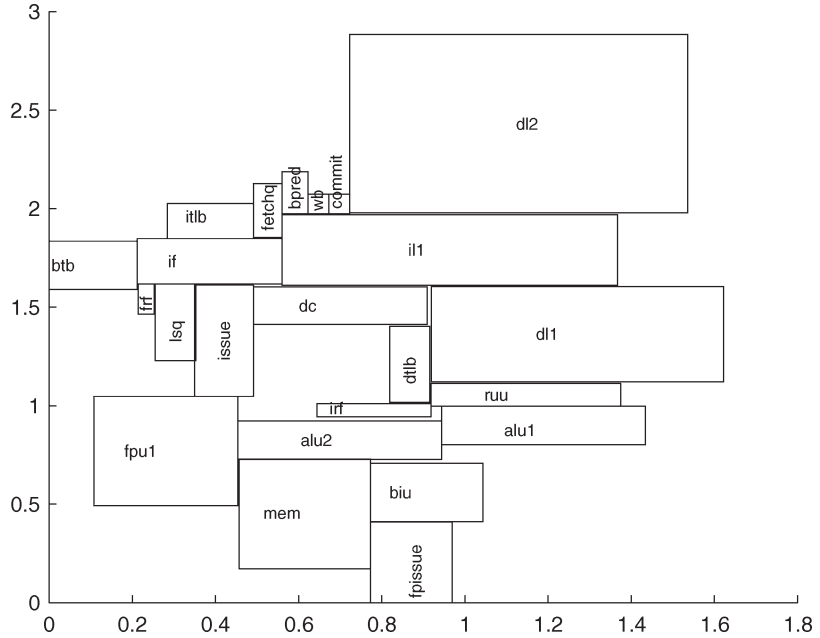


Fig. 17. Snapshot of PGF floor plan.

access latencies to these submodules while attempting to minimize the latency of the most frequently accessed registers. This leads to a new opportunity to explore the tradeoff in latency, area, and partitioning for processor resources. The tradeoff study and its overall impact can be performed for the other microarchitecture components such as the reorder buffer, the branch target buffer, and caches. In addition, resource allocation with respect to what-to-allocate in the faster submodules will also be a subject of many research interests.

Some wires at microarchitectural level are "invisible," such as control wires among the modules and wires connecting Power/Ground (= P/G) pins of the module to global P/G pins. The implication of these invisible control wires is not yet known. Some recent studies [22], [23], however, show that considering the P/G connection at circuit module floor planning is important to guarantee high-quality power supply. Thus, we plan to consider this issue for microarchitectural modules. In addition, we plan to use our PGF result as an initial solution for a subsequent iterative improvement with an efficient IPC estimation [10]. Lastly, we are developing a more efficient compaction algorithm for area minimization.

REFERENCES

[1] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.

[2] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus IPC: The end of the road for conventional microarchitectures," in *Proc. IEEE Int. Conf. Computer Architecture*, Vancouver, Canada, 2000, pp. 248–259.

[3] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis, "Microarchitecture evaluation with physical planning," in *Proc. ACM Design Automation Conf.*, Anaheim, CA, 2003, pp. 32–35.

[4] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2002, pp. 268–273.

[5] W. Liao and L. He, "Full-chip interconnect power estimation and simulation considering concurrent repeater and flip-flop insertion," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 2003, pp. 574–580.

[6] K. Sankaralingam, V. A. Singh, S. W. Keckler, and D. Buger, "Routed inter-ALU networks for ILP scalability and performance," in *Proc. IEEE Int. Conf. Computer Design*, San Jose, CA, 2003, pp. 170–177.

[7] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multi-cycle communication," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 4, pp. 550–564, Apr. 2004.

[8] W. Gosti *et al.*, "Wireplanning in logic synthesis," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, 1998, pp. 26–33.

[9] M. Casu and L. Macchiarulo, "Floorplanning for throughput," in *Proc. Int. Symp. Physical Design*, Phoenix, AZ, 2004, pp. 62–69.

[10] C. Long, L. Simonson, W. Liao, and L. He, "Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects," in *Proc. ACM Design Automation Conf.*, San Diego, CA, 2004, pp. 640–645.

[11] SIA, *National Technology Roadmap for Semiconductors*, 2001.

[12] P. Shivakumar and N. P. Jouppi, "CACTI 3.0: An integrated cache timing, power, and area model," HP Western Research Labs, Palo Alto, CA, Tech. Rep. 2001.2, 2001.

[13] J. C. Eble, V. K. De, D. S. Wills, and J. D. Meindl, "A generic system simulator (GENESYS) for ASIC technology and architecture beyond 2001," in *Proc. Int. Application-Specific Integrated Circuit (ASIC) Conf.*, Rochester, NY, 1996, pp. 193–196.

[14] T. M. Austin, *SimpleScalar Tool Suite*. [Online]. Available: http:/www.simplescalar.com

[15] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[16] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 10, no. 3, pp. 356–365, Mar. 1991.

[17] D. Weiss, J. Wuu, and V. Chin, "The on-chip 3-MB subarray-based third-level cache on an Itanium microprocessor," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1523–1529, Nov. 2002.

[18] S. McFarling, "Combining branch predictors," Western Research Lab., Digital Equipment Corp., Palo Alto, CA, Tech. Rep. TN-36, Jun. 1993.

[19] G. Sohi and S. Vajapeyam, "Instruction issue logic for high performance interruptable pipelined processors," in *Proc. 14th Annu. Int. Symp. Computer Architecture*, Pittsburgh, PA, 1987, pp. 27–34.

[20] P. N. Glaskowsky, "Pentium 4 (partially) previewed," *Microprocess. Rep.*, vol. 14, no. 8, pp. 11–13, Aug. 2000.

[21] E. U. of Technology, *LP_solve*. [Online]. Available: ftp:/ftp.es.ele.tue.nl/pub/lp_solve/

[22] S. Zhou, S. Dong, X. Wu, and X. Hong, "Integrated floorplanning and power supply planning," in *Proc. Int. Conf. Application-Specific Integrated Circuit (ASIC)*, Shanghai, China, 2001, pp. 194–197.

[23] I. Liu, H.-M. Chen, T.-L. Chou, A. Aziz, and D. Wong, "Integrated power supply planning and floorplanning," in *Proc. Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, 2001, pp. 589–594.

Mongkol Ekpanyapong (S'03–M'05) received the B.E. degree from the Computer Engineering Department, Chulalongkorn University, Bangkok, Thailand, in 1997, the M.E. degree from the Computer Science Department, Asian Institute of Technology, Pathumthani, Thailand, in 2000, the M.S. degree from the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, in 2003, and is currently working toward the Ph.D. degree in physical design for microarchitectures at the Georgia Institute of Technology.

He is a Graduate Research Assistant at the School of Electrical and Computer Engineering, Georgia Institute of Technology. His research interests include physical very large scale integration (VLSI) design, VLSI design, computer architecture, and compiler.

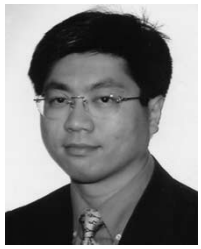
Jacob Rajkumar Minz (S'05) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, India, in 2001, and is currently working toward the Ph.D. degree at the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta.

He was with the Advanced VLSI Design Lab, IIT, Kharagpur, India, for a year, where he was involved in the design of digital chips. His areas of interest are physical design automation and algorithms for electronic computer-aided design (CAD).

Thaisiri Watewai (S'05) received the B.Eng. degree (*summa cum laude*) in industrial engineering from Chulalongkorn University, Bangkok, Thailand, in 2001, the LL.B. degree from Ramkhamraeng University, Thailand, in 2001, the M.S. degree in operations research from the Georgia Institute of Technology, Atlanta, in 2002, the M.A. degree in statistics from the University of California, Berkeley, in 2005, and is currently working toward the Ph.D. degree in industrial engineering and operations research at the University of California, Berkeley.

His current research interests include stochastic control and optimization, robust dynamic optimization, ambiguity modeling, financial engineering, and statistical learning.

Mr. Watewai was awarded the Anandamahidol Foundation Fellowship, the most prestigious fellowship in Thailand, by His Majesty the King of Thailand to pursue graduate studies in the USA.



Hsien-Hsin S. Lee (M'96) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 2001.

Prior to joining the academia, he was a Senior Computer Architect at Intel Corporation (1995–2001), and later an Architecture Manager of the StarCore DSP Technology Center, Agere Systems (2001–2002). He is an Assistant Professor at the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta. He holds four U.S. patents. His research interests include computer architecture, low power circuits, information security, and design optimization tools.

Dr. Lee is a member of Tau Beta Pi, Sigma Xi, and the Association for Computing Machinery (ACM). His Ph.D. dissertation was awarded the Horace H. Rackham School Distinguished Dissertation Award at the University of Michigan. He has authored two papers that won the Best Paper Award at the International Symposium on Microarchitecture (2000) and the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (2004).

Sung Kyu Lim (M'01) received the B.S., M.S., and Ph.D. degrees all from the Computer Science Department, University of California, Los Angeles (UCLA), in 1994, 1997, and 2000, respectively.

From 2000 to 2001, he was a Post-Doctoral Scholar at UCLA and a Senior Engineer at Aplus Design Technologies, Inc. He joined the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, as an Assistant Professor in August 2001, and the College of Computing as an Adjunct Assistant Professor in September 2002. He is currently the Director of the Georgia Tech Computer Aided Design Laboratory. His research focus is on the physical design automation for three-dimensional (3-D) circuits, 3-D system-on-packages, microarchitectural physical planning, field programmable analog arrays, and quantum cell automata.

Dr. Lim has been on the advisory board of the Association for Computing Machinery (ACM)/SIGDA since 2003. He is currently serving the technical program committee of the IEEE International Symposium on Circuits and Systems, ACM Great Lakes Symposium on VLSI, IEEE International Conference on Computer Design, ACM International Symposium on Physical Design, and ACM/IEEE Asia and South Pacific Design Automation Conference. He has been awarded the ACM/SIGDA DAC Graduate Scholarship in June 2003.