Profile-Guided Microarchitectural Floorplanning for Deep Submicron Processor Design

Mongkol Ekpanyapong Jacob R. Minz Thaisiri Watewai† Hsien-Hsin S. Lee Sung Kyu Lim

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
{*pop, jrminz, leehs, limsk*}*@ece.gatech.edu*

†Dept. of Industrial Engineering and Operations Research
University of California
Berkeley, CA 94720
thaisiri@uclink.berkeley.edu

Abstract

As process technology migrates to deep submicron with feature size less than 100nm, global wire delay is becoming a major hindrance in keeping the latency of intra-chip communication within a single cycle, thus decaying the performance scalability substantially. An effective floorplanning algorithm can no longer ignore the information of dynamic communication patterns of applications. In this paper, using the profile information acquired at the architecture/microarchitecture level, we propose a "profile-guided microarchitectural floorplanner" that considers both the impact of wire delay and the architectural behavior, namely the inter-module communication, to reduce the latency of frequent routes inside a processor and to maintain performance scalability. Based on our simulation results, the profile-guided method shows a 5% to 40% IPC improvement when clock frequency is fixed. From the perspective of instruction throughput (in BIPS), our floorplanner is much more scalable than a conventional wire length based floorplanner.

1. INTRODUCTION

According to the projection of the International Technology Roadmap for Semiconductors (ITRS) [13], deep submicron process technology will be able to integrate more than one billion transistors onto a single monolithic die very soon. Given the continuing and fast miniaturization of transistor feature sizes, global wire length is becoming a major hindrance in keeping performance scalable as its relative delay to the gate delay gradually worsens as technology continues to shrink. Local wire length, on the other hand, will scale with a marginal impact in adding extra delay with respect to the same process technology [7]. Despite the use of different materials, device structures, circuit techniques or novel architectures including nanotechnology, this global interconnect limit still persists due to the nature of device physics and inflicts a substantially higher performance impact for chips manufactured with deep submicron technology, particularly, for microprocessors which keep pursuing ever-higher performance as the primary design objective.

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, for example, increasing cache sizes and hierarchy, enlarging reorder buffer, widening issue and commit width, improving speculation by employing very complex branch predictors, to name a few. All of these architecture enhancement effectively resulted in higher processor performance in the past. On the other hand, CAD tool developers and circuit designers try to reduce the cycle time as much as possible, without any knowledge of the entire design at the architectural level. With the increasing impact of global wire length, however, such design methods could lead to less optimal designs, if not totally ineffective, due to the intra-chip communication latency, and need to be largely changed by taking the wires into account. In this paper, we advocate the collaboration between architecture design and physical design. By considering both simultaneously, we expect to achieve a much better performance improvement for microprocessors designed using deep submicron technology.

The rest of this paper is organized as follows. The next section discusses some related work. Section 3 overviews the implication of IPC and clock speed and outlines our approach. Section 4 introduces our profile-guided floorplanning and its mathematical foundation. The description of our architectural framework follows in the next section. Then we show our experimental results in Section 6. Finally, we conclude this work and point out some future directions in Section 7.

2. RELATED WORK

With the growing concern in global wire delay, there are a large number of researches focusing on this area that attempt to address this issue from different aspects including circuits, microarchitectures, and the combination between logic synthesis and physical design. Agarwal et al. in [1] raised the issue of the 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. [4] confirmed their observation and showed that without considering clock speed, Instruction Per Cycle (IPC), widely used in architecture research, can be misleading in evaluating the performance of next generation processors.

More recently, Sankaralingam et al. [11] proposed a new data bypassing mechanism that enhances performance

DAC'04, June 7–11, 2004, San Diego, California, USA.

when multi-cycle bypassing delays are preset in processor designs. In logic synthesis, novel techniques [3, 6] were proposed to improve the performance by applying wiring aware logic synthesis. These techniques provide more information regarding locations in the phase of logic synthesis, leading to overall performance improvement. However, postponing the optimization after logic synthesis phase is completed can be time-consuming, and is inapplicable to custom design.

3. WIRE DELAY ISSUES

Ho, Mai and Horowitz in [7] classified wires into three categories based on their delay impact: 1) wires that scale in length, such as local interconnected wire within logic blocks; 2) wires that do not scale in length and is super-linear when feature size is reduced; 3) repeated wire, i.e. long wire with repeated buffers along the wire. The propagation delay of a repeated wire can be represented as Equation (1), where R_d is the driver resistance, w is the width of the driver transistor, C_d and C_g are diffusion and gate resistances per unit width, R_{wire} and C_{wire} are wire resistance and capacitance per unit length, l is the repeater segment length, and *beta* is the PMOS to NMOS sizing ratio. In next generation deep submicron processor design, it is likely that repeaters will be inserted frequently for global wires to prevent the wire delays from becoming non-linear. In this paper, we assume repeated wires to be the dominant wire and examine their performance impact from the perspective of floorplanning. Based on predicted value of resistor, capacitance and other parameters from [13, 7] repeated wire delay is approximated to be 80pS/mm for 30nm technology, which we use as the baseline for our discussion in this paper. Note that a FO4 gate delay for 30 nm is approximately 17pS.

$$D = 0.7n\left[\frac{R_d\left[w(\beta+1)(C_d+C_g)+lC_{wire}\right]}{w} + l^2\frac{R_{wire}C_{wire}}{2} + lR_{wire}w(\beta+1)C_g\right] \quad (1)$$

Flip-flop insertion is a technique to alleviate the impact of wire delay for achieving target clock frequency. As shown in [9], by inserting flip-flops between modules, clock frequency can be increased through deeper pipelining and results in a higher billion instructions per second or BIPS. Nevertheless, the improvement cannot always be anticipated especially for designs with a small feature size as flip-flop insertions will cause IPC degradation from increasing the latency as shown in Figure 1, in which if module 2 is moved toward the righthand side from the lefthand side, more flip-flops need to be inserted, leading to longer latency. Therefore, inserting flipflops without a meticulous evaluation, even though we can accelerate the clock rate, it does not guarantee an overall performance improvement since the latency of communication between modules might now take longer.

4. PROFILE-GUIDED FLOORPLANNING

As shown in Figure 2, we propose an unified framework that combines technology scaling parameters and the execution profiling information of applications to guide the floorplanning of a given processor architecture design. First of all, a machine description is provided as the input for 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 given benchmark programs. For cache-like or buffer-like structures, the area size and module delay are estimated using a industry tool from HP Western Research Labs called CACTI [12]. For scaling the other structures such as ALUs, we use GENESYS [5] developed at Georgia Tech which has a baseline using a .35 μm Verilog model. After the timing and area information of each module is collected, the statistical interconnection traffic and the processor target frequency range are fed into our profile-guided floorplanner to generate a floorplan for deriving the timing model (i.e. inter-module latency) of the microarchitecture as a result of the generated floorplan. With the new latencies, the architecture performance simulation is performed for simulating the IPC numbers, and later the actual performance in BIPS can be calculated together with the cycle time. We keep iterating the entire methodology for exploring alternate architecture designs by increasing or decreasing the module size, or even splitting the module geometry for achieving the performance goal. Results from our profile-guided microarchitectural floorplanning can be used to guide the final global floorplanning such that performance is not degraded.

Now we formulate the mathematical programming models for our floorplanner. First, we present a mixed integer nonlinear programming (MINP) model which minimizes the product of the cycle time and the weighted wire length. Since to find the optimal solution of the MINP model is

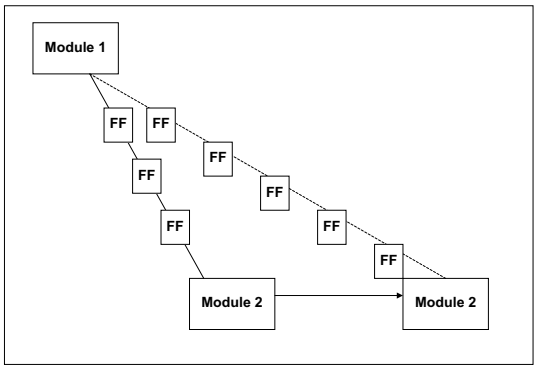

Figure 1: Impact of Wire Delay on Latency

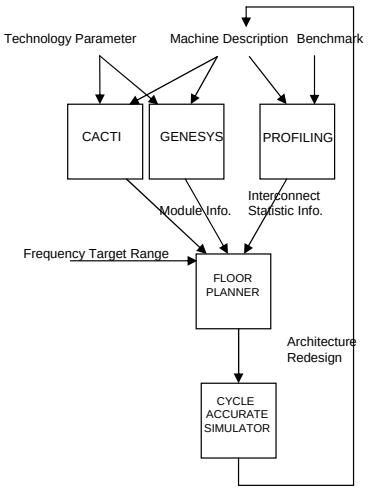

Figure 2: Profile-Guided Floorplanning

NP-hard, we then linearize and relax integral constraints in such a way as to stay closed to the optimal solution within a reasonable runtime. Finally, we demonstrate the complete floorplanning algorithm.

4.1 Mixed Integer Non-linear Programming Model

The parameters used in the model MINP are defined as follows. Let N denote the set of all flexible modules, and E denote the set of directed edges where a directed edge (i, j) represents a wire from module i to module j. Also, let α be the repeated delay per mm, and λ_{ij} be the statistical traffic on wire (i, j). g_i is the gate delay of module i. $w_{min,i}$ and $w_{max,i}$ denote the minimum half width and the maximum half width of module i respectively. The area of module i is denoted by a_i. Finally, f_{ij} is the minimum number of flip-flops on wire (i, j).

In the model, we need to determine the values of the following decision variables: Let L denote the cycle time of the system (1/clock_frequency), and (x_i, y_i) denote the location of the center of module i on $\mathbb{R}^2_+$ space. X_{ij} and Y_{ij} represent $|x_i - x_j|$ and $|y_i - y_j|$ respectively. z_{ij} is the number of flip-flops on wire (i, j). w_i denotes the half width of module i. To avoid overlapping between any two modules i and j, where $i < j$, we also need a set of binary variables so that at least one of the followings holds.

$$\begin{aligned} x_i + w_i &\le x_j - w_j && i \text{ is on the left of } j \\ x_i - w_i &\ge x_j + w_j && i \text{ is on the right of } j \\ y_i + \frac{a_i}{4w_i} &\le y_j - \frac{a_i}{4w_i} && i \text{ is below } j \\ y_i - \frac{a_i}{4w_i} &\ge y_j + \frac{a_i}{4w_i} && i \text{ is above of } j \end{aligned}$$

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

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

The constraints for non-overlapping among modules are given from inequality (9) to (12) in the MINP formulation illustrated in Figure 3 wherein the objective function (2) is to minimize the net weighted delay. Constraint (3) is obtained by the definition of latency. (4) to (7) define the absolute values. Constraint (8) represents the minimum number of flip-flops required. (13) states the possible range of each module's half width. (14) are non-negative constraints of module's location. (15) manifests that (c_{ij}, d_{ij}) are binary. Finally, (16) specifies that the number of flip-flops must be integer. Also note that M is a sufficiently large number.

4.2 Mixed Integer Linear Programming Model

To avoid non-linearity of (11) and (12), we linearize the half height of module i (h_i) using linear approximation as shown in Figure 4. Note that this approximation will guarantee that the solution in the approximated model is feasible (due to over approximation). Also, in (3) we can eliminate the non-linearity by fixing $L = L_0$ and iteratively solve the problem with different L values. This, however, doesn't require too much effort since the set of the L candidates is not large based on achievable target frequency. The resulting mixed integer linear programming formulation, MILP, is shown in Figure 5.

4.3 Linear Programming Model

Despite the fact that we can convert MINP into MILP, the problem still remains NP-hard for two dimensional bin-packing. Hence, it will limit our search space by MILP running time for a large number of modules. To remedy this problem, we further relax MILP into Linear Programming (LP) as shown in Figure 6. We derive a method similar to those described in [8, 2] by recursively bi-partitioning a space into smaller subregions.

To relax the integrality while maintaining the feasibility and staying closed to the optimal solution, we first relax the integrality of z_{ij} to be a real number. We also solve several linear programming problems to determine the non-overlapping relationship — (c_{ij}, d_{ij}).

We start the algorithm by creating a large block containing all modules. In each iteration we divide each current block into two smaller subblocks by specifying the centers of each subblock as well as their corresponding boundaries. Then we randomly assign modules currently in the block into one of the centers of these two subblocks. For the next iteration, the modules currently in the block must remain in the same block ((22)-(25)), and all the modules assigned to the same subblock have center of gravity of area at the center of that subblock ((26)-(27)). Note that even if a module assigned to a particular subblock, it can be moved across the subblock boundary as long as the preceding two

MINP

$$\text{Minimize} \quad L \times \sum_{(i,j)\in E} \lambda_{ij} z_{ij} \tag{2}$$

Subject to

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

Figure 3: MINP Formulation

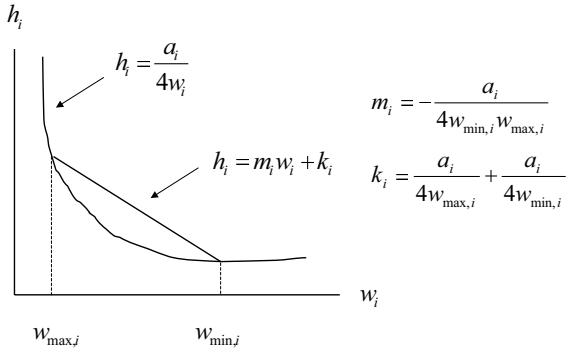

Figure 4: Block Area Constraint Approximation

MILP

$$\text{Minimize} \quad \sum_{(i,j)\in E} \lambda_{ij} z_{ij} \tag{17}$$

Subject to (4)—(10), (13)—(16) and the following

$$z_{ij} \geq \frac{g_i + \alpha(X_{ij} + Y_{ij})}{L_0} \qquad (i,j) \in E \tag{18}$$

$$y_i + m_i w_i + k_i \leq y_j - m_j w_j - k_j + M(1 - c_{ij} + d_{ij}) \qquad i < j \in N \tag{19}$$

$$y_i - m_i w_i - k_i \geq y_j + m_j w_j + k_j - M(2 - c_{ij} - d_{ij}) \qquad i < j \in N \tag{20}$$

Figure 5: MILP Formulation

LP(u): Minimizing net weighted delay

$$\text{Minimize} \quad \sum_{(i,j)\in E} \lambda_{ij} z_{ij} \tag{21}$$

Subject to (18), (4)—(8), (13)—(14) and the following

$$x_i + w_i \leq r_j \qquad i \in M_j(u), j \in B(u) \tag{22}$$

$$x_i - w_i \geq l_j \qquad i \in M_j(u), j \in B(u) \tag{23}$$

$$y_i + m_i w_i + k_i \leq t_j \qquad i \in M_j(u), j \in B(u) \tag{24}$$

$$y_i - m_i w_i - k_i \geq b_j \qquad i \in M_j(u), j \in B(u) \tag{25}$$

$$\sum_{i \in S_{jk}(u)} a_i x_i = \sum_{i \in S_{jk}(u)} a_i \times \bar{x}_{jk} \qquad k \in \{1,2\}, j \in B(u) \tag{26}$$

$$\sum_{i \in S_{jk}(u)} a_i y_i = \sum_{i \in S_{jk}(u)} a_i \times \bar{y}_{jk} \qquad k \in \{1,2\}, j \in B(u) \tag{27}$$

Figure 6: Minimize Net Weighted Delay

conditions are satisfied. Once the locations of all modules are determined at the current iteration (by LP(u) as shown in Figure 6), each subblock will become a block in the next iteration. We terminate the algorithm when each subblock contains exactly one module. The following notations are defined before we detail the whole algorithm. Let $B(u)$ denote the set of all blocks at iteration u, and $M_j(u)$ denote the set of modules currently in block j at iteration u. Also, let $S_{jk}(u)$ be 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 $(\bar{x}_{jk}, \bar{y}_{jk})$. Finally, let r_j, l_j, t_j, b_j denote the right, left, top, and bottom boundary of block j. The algorithm is summarized in Figure 7.

Note that the LP(u) will be feasible if we start from an initial block that is large enough, and at each iteration each subblock is created according to the total of areas of modules assigned to that subblock.

Once the algorithm is terminated with locations of all modules, we investigate the non-overlapping relationship (9) to (12). If for each pair (i,j) exactly one of four possible pairs (c_{ij}, d_{ij}) is satisfied, we fix such (c_{ij}, d_{ij}), otherwise we randomly fix one of the two possible relationships — one from (9) to (10) and the other one from (11) to (12). By fixing all (c_{ij}, d_{ij}), we solve MILP without (16) so that it is a linear programming problem. Note that each time we perform a random procedure, we repeat many times with different random values and select the best result.

We also have another LP model in which we minimize the total wire length for benchmarking. We apply the similar approach as our model. The LP$'(u)$ for wire length minimization is described in Figure 8.

5. SIMULATION INFRASTRUCTURE

Bi-Partitioning LP Algorithm

For each L in target frequency range

Step 0: Initialize $B(1) = \{1\}, M_1(1) = N$.

Step u: for *count* =1 to *run*
Specify $S_{jk}(u), (\bar{x}_{jk}, \bar{y}_{jk})$. Solve LP($u$).
Update $B(u+1), M_j(u+1), r_j, l_j, t_j$, and b_j.
Goto final step if each $M_j(u+1)$ contains exactly 1 module
end for
Project best obj. into next iteration $u+1$

Final step: Solve MILP by fixing c_{ij}, d_{ij} obtained from Recursive Bipartitioning.
Project best L and z_{ij}

Figure 7: Recursive Bi-partitioning LP Algorithm

LP$'$(u): Minimizing total wirelength

$$\text{Minimize} \quad \sum_{i,j \in N} (X_{ij} + Y_{ij}) \tag{28}$$

subject to (4)—(7), (22)—(25), (13)—(14), (26)—(27)

Figure 8: Minimize Total Wirelength

Simplescalar 3.0 tool suite is used as our architecture simulator for both profile collection and performance simulation. The detailed processor microarchitecture evaluated in Section 6 is illustrated in Figure 9 and their variations are listed in Table 1. Each functional block in Figure 9 represents a module used by our floorplanner. In order to facilitate physical-design driven micro-architectural exploration we added some new features. First, we extended the Simplescalar to include configurable pipeline depth. The fetch unit of the simulator was efficiently modified to accommodate pipelines of desired depth. This gives us a handle to study the effects of lengthening the processor pipeline. In terms of performance, pipelining has a direct impact on the operational frequency of the processor. The number of pipeline stages is also affected by our extension of the functional units. 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 configuration input.

Provisions were also made to consider wire delays in the simulator. The existing simulator assumes that the forwarding latency between blocks is always single cycle which is less reasonable while operating at extremely high frequency, given the increased wire delays and larger die areas. In our modified simulator, the forwarding latencies of the function units to other units or pipe stages were made configurable, enabling a more realistic IPC projection.

For accurate prediction and optimization of performance, the wires can no longer be isolated from architecture-level evaluation but must be modeled as *units* that consume power and have delays. Wires, as much as other logical, storage, and control units of a processor must be "designed" appropriately. In order to have a simultaneous view of the frequency and the IPC, we have to explore the floorplan and processor organization together. The wire parameters can be captured for architectural optimization through floorplanning, to a reasonable degree of accuracy. The wires directly affect the forwarding latency and the pipeline depth of the processor. For performance evaluation, we use the information provided by the floorplanner to derive essential simulation parameters such as pipeline depth and forwarding latency. The forwarding latency is a function of the distance between units in the floorplan and the number of flip-flops between modules. If the floorplan

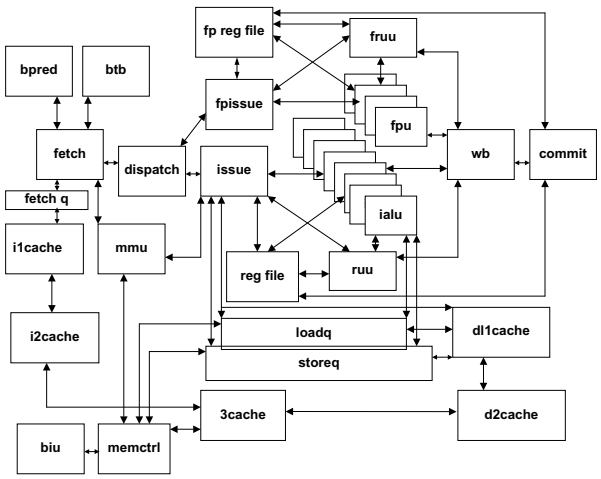

Figure 9: Processor Microarchitecture Model

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 are included in our floorplan formulation. The forwarding frequency driven floorplanning tries to put heavy traffic units closer together, minimizing their latencies as a function of their distance. Our floorplanning algorithms described in Section 4 were implemented in C, compiled with gcc with -O3, integrated with Simplescalar tool set, and executed on Pentium III 750MHz machines. We use lp_solve [10] to solve our linear programs.

6. EXPERIMENTAL RESULTS

We perform experiments on ten SPEC2000 benchmarks, six from the integer suite and 4 from the floating point. We performed profiling on training input sets. Our results are reported by running on the reference input sets by fast forward for 200 million instructions and simulation for another 100 million instructions. The maximum execution time combining both floorplanning and simulation is less than two hours for each benchmark.

Figure 10 shows the IPC improvement based on the same clock frequency with different microarchitecture configurations for the baseline (WL) and our profile-guided floorplanning algorithm (PGF). Note that the baseline floorplan is generated by minimizing the total wire length. The results show that the baseline algorithm will no longer deliver higher IPCs as the number of the modules and/or the size of the modules are increased due to the limitation of wire length. On the other hand, our profile-guided floorplanner continues to increase the IPCs since the dynamic information of interconnection traffic between modules is taken into account during floorplanning. For the most complex microarchitecture (Configuration 4), the processor design based on our technique outperforms the baseline by 40%.

Figure 11 evaluates the impact in terms of total wire length using our technique. Instead of minimizing wire length, our algorithm attempts to improve overall performance and could potentially result into a longer total wire length. As shown in the figure, our technique does not actually incur any significant increase in wires for less complexity processors, in some cases it is even better than the baseline[1]. For the outlyer case shown in Config3 of bzip2,

[1]Note that the baseline itself has some randomness involved and is also a heuristics, not an optimal solution.

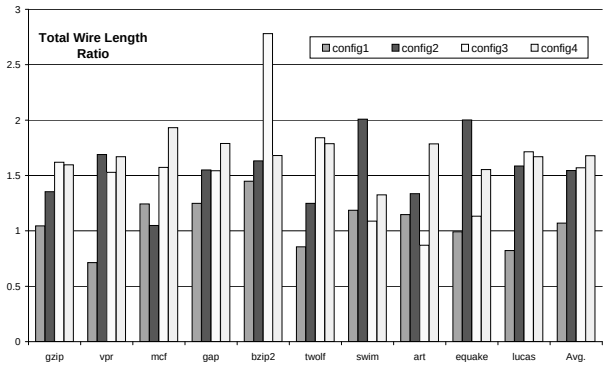

Figure 11: Impact on Total Wire Length (Cfg3)

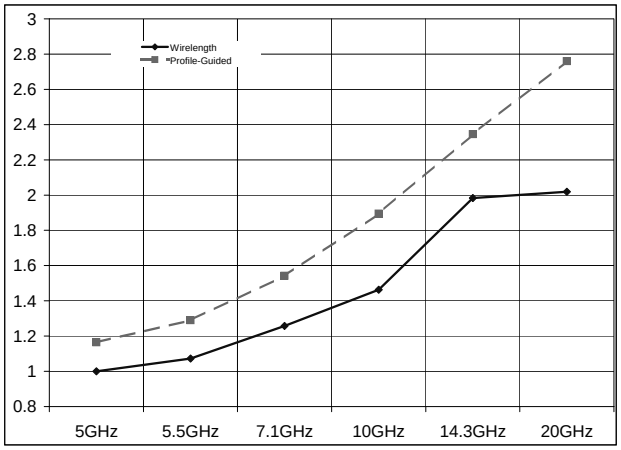

Figure 12: Performance versus Frequency Scaling

data are mostly forwarded from RUU to ALUs with very little inter-ALU communication, hence the only constraint imposed is from the RUU. Whereas in Config4 case, there are more data forwarding between ALUs, hence the imposed constraint do not allow each ALU to move freely. For complex machines such as the configuration 4, we increase the total wire by 68% in average. Even though the total wire length is increased, it only affects the inter-module latencies of those less frequently used routes.

We present the overall performance in normalized BIPS for different clock frequencies in Figure 12, in which the clock period is decreased from 0.2ns (5GHz), 0.18ns (5.5GHz) down to 50ps per cycle (20GHz). The results, averaged across all benchmarks, are normalized to the BIPS delivered by 5GHz baseline machine using the floorplanner that minimizes the wirelength. As shown, the advantage of using a profile-guided method significantly stands out when the clock speed is increased. When clock frequency is quadFor a 20GHz processor, our technique shows almost 3 times improvement in BIPS over the baselne, whereas minimizing wirelength can gain only twice. Hence our floorplanner is more scalable than a conventional approach. Eventhough we didn't show the comparison between timing driven floorplaning here, it can be approximatedly considered as minimizing wirelength on higher clock frequency. As shown in the Figure, our approach can be as good as next generation processor and sometimes it is even better.

For optimal cycle time precdiction, our algorithm can predict two out of eight correctly and another four are in the close range. Since we relax the original MINP into LP,

This can be seen more clearly in small solution space such as in Config1

Micro-arch Structure	Cfg 1		Cfg 2		Cfg 3		Cfg 4		# of Bits
	Size	Lat	Size	Lat	Size	Lat	Size	Lat	
Bpred	128	2	512	2	512	2	512	2	2
BTB	128	2	512	3	512	3	512	4	96
RUU	64	2	128	3	512	4	512	6	168
Int RF	32	2	32	2	32	2	32	3	64
FP RF	32	2	32	2	32	2	32	3	64
L1 Icache	8KB	3	64KB	3	8KB	4	8KB	4	512
L1 Dcache	8KB	2	64KB	5	8KB	5	8KB	5	512
L2 Ucache	64KB	3	512KB	10	128KB	10	128KB	10	1024
L3 Ucache	None	N/A	None	N/A	2MB	59	2MB	59	1024
ITLB entries	32	4	128	4	128	5	128	5	112
DTLB entries	32	4	128	4	128	5	128	5	112
ALU	2	2	4	2	4	2	8	2	N/A
FPU	1	7	2	7	2	7	4	7	N/A
LSQ	16	4	64	5	128	6	128	6	84
Mem port	1	N/A	4	N/A	8	N/A	8	N/A	N/A
Machine width	2	N/A	4	N/A	8	N/A	8	N/A	N/A

Table 1: Microarchitecture Configurations

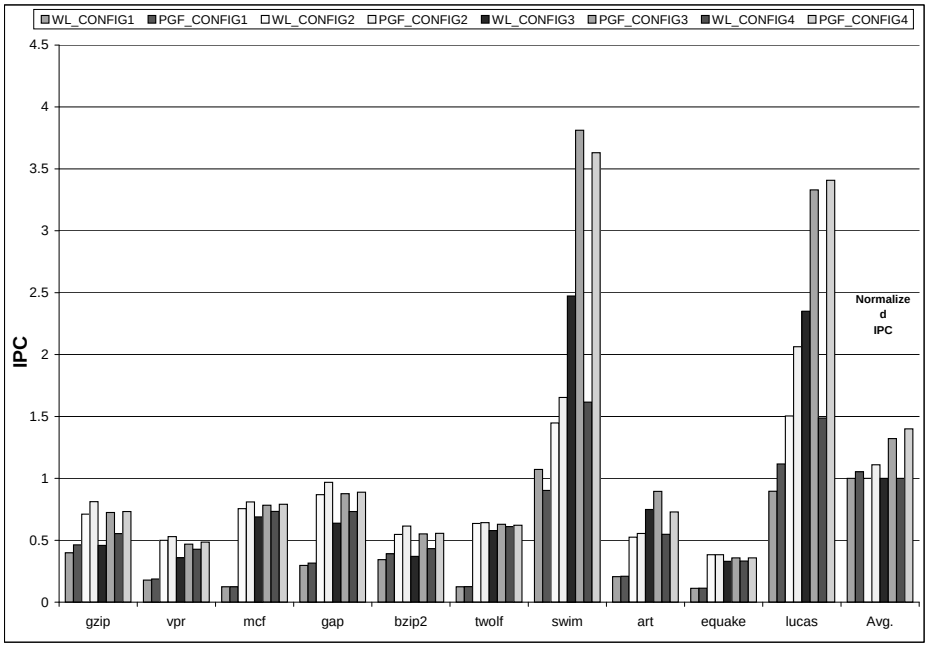

Figure 10: Performance Improvement for a 10GHz Processor (WL: Wirelength, PGF: Profile-Guided Floorplan)

we cannot guarantee the optimal solution. In addition, it is not easy for mathematical model to completely capture architecture behavior. However we can use this as a guideline for faster search time. We can use predict cycle time as the starting point and hence perform iterative search by running floorplanner and simulator alternately.

7. CONCLUSIONS AND FUTURE WORK

Due to the miniaturization of feature size, global wire delay needs to be addressed and considered as an architectural entity for microprocessor designs. In this work, we proposed a profile-guided floorplanning algorithm which uses both the technology scaling parameters and the information of dynamic internnection traffic between microarchitectural modules to guide the floorplanning for performance optimization. Based on the statistics of the intermodule communication, our design methodology generates given a target clock frequency range. Our results show our proposed method can effectively improve the overall applications' performance by up to 40% over a conventional wire length based floorplanner.

One future research direction of this work is to optimize the performance by further partitioning each functional unit 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 floorplanner could potentially generate a floorplan with different 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 trade-offs in latency, area, and parititioning for processor resources. The trade-offs 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.

8. REFERENCES

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *ISCA*, 2000.

[2] N. Sehgal B. Halpin, C.Y.R. Chen. Timing Driven Placement using Physical Net Constraints. In *DAC01*, 2001.

[3] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang. Architectural Synthesis Integrated with Global Placement for Multi-Cycle Communication. In *ICCAD*, 2003.

[4] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis.

Microarchitecture Evaluation with Physical Planning. In *DAC*, 2003.

[5] 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 *Int'l ASIC Conference*, 1996.

[6] W. Gosti and et al. Wireplanning in Logic Synthesis. In *ICCAD*, 1998.

[7] R. Ho, K. W. Mai, and M. A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 2001.

[8] J. Kleinhaus and et al. GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization. *IEEE Tran. on CAD*, 1991.

[9] S. Liao and L. He. Full-Chip Interconnect Power Estimation and Simulation Considering Concurrent Repeater and Flip-flop Insertion. In *ICCAD*, 2003.

[10] Eindhoven University of Technology. LP_solve. ftp:/ftp.es.ele.tue.nl/pub/lp_solve/.

[11] K. Sankaralingam, V. A. Singh, S. W. Keckler, and D. Buger. Routed Inter-ALU Networks for ILP Scalability and Performance. In *ICCD*, 2003.

[12] P. Shivakumar and N. P. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model. Technical Report 2001.2, HP Western Research Labs, 2001.

[13] SIA. National Techonology Roadmap for Semiconductors, 2001.