SUBMITTED TO TCAD, DO NOT DISTRIBUTE !

Micro-Architecture and Floorplanning Co-Optimization

Changbo Long, Lucanus J. Simonson, Weiping Liao and Lei He

Abstract

Interconnect pipelining has a great impact on system performance, but it has not been considered by microprocessor floorplanning until recently. Considering interconnect pipelining, we first study the floorplanning optimization problem to minimize system CPI (cycle-per-instruction) for given micro-architecture configurations. We propose two algorithms, one based on access ratio without computing CPI and the other using a trajectory piecewise-linear (TPWL) model to calculate CPI. Compared to the conventional floorplanning to minimize area and wire length, our TPWL-based floorplanning can reduce CPI by up to 72.4% with a small area overhead less than 5.0%. Compared to the TPWL-based method, the access ratio method takes less time to build model but obtains slightly worse CPI. Furthermore, we develop microarchitecture and floorplan co-optimization by expanding the TPWL model to consider micro-architecture changes (the access ratio based model is not able to accomodate micro-architecture changes). Efficiently exploring over 1,000,000 micro-architecture configurations, we find micro-architecture and corresponding floorplanning with an IPC (instructions per cycle) 26.9% better than a recent paper that was only able to consider a limited number of micro-architecture configurations.

I. Introduction

In this paper, we study the problem of maximizing throughput by co-optimization of micro-architecture and floorplanning. The throughput of a system is the product of average instruction-per-cycle (IPC) and clock rate. To maximize throughput, the traditional design flow separates micro-architecture design to determine IPC and floorplanning to determine clock rate, which is based on the assumption that inter-block communication can be finished within one clock cycle. Because of the existence of multiple cycles for inter-block communication in the state of the art designs [1], [2], [3], we develop co-optimization of micro-architecture and floorplanning to maximize IPC based on a fixed clock rate for a single core microprocessor in this paper.

Maximizing IPC involves the utilization of instructions level parallelism (ILP). An example is the SuperScalar micro-architecture [4], [5], which executes multiple instructions simultaneously. Quantitatively determining IPC for a particular SuperScalar micro-architecture requires the details of architectural configurations, including issue width, branch prediction, cache/TLB sizes and number of function units. As shown in [6], micro-architecture configurations significantly impact performance and resizing critical components improves performance over 20% for Power5 processors. Cycle accurate simulations over a set of benchmarks are often used to measure IPC. Due to the large number of reasonable configurations (beyond 1,000,000 as shown in Section V) and the fact that cycle-accurate simulation for one configuration may take hours to finish, it is very difficult to find optimal configurations in general. Statistical micro-architecture simulations have been developed for the purpose of fast exploration of micro-architecture configurations[7], [8] but automatic micro-architecture exploration method was not well studied.

Complicating matters, floorplanning affects IPC significantly. For instance, increasing the latency between the fetch logic and L1 instruction cache from one cycle to two can increase the average CPI by 12.6% for the microprocessor similar to Alpha 21264 in Table I for the 100nm generation. In fact, the delay of a wire crossing a leading edge chip in the $70nm$ generation can be over five clock cycles[9]. As shown by [2], the number of flip-flops used for interconnect pipelining is expected to increase exponentially between generations of microprocessors. The impact of floorplanning should be considered by automatic micro-architecture exploration.

In this paper, we first study microprocessor floorplanning considering interconnect pipelining. We explore two methods to evaluate the impact of floorplanning on system performance: access ratio based model and trajectory piecewise-linear (TPWL) CPI model. The TPWL model computes CPI explicitly and access ratio model assigns weights to global interconnects without explicit CPI computation. Experiment results show that CPI-aware floorplanning can reduce CPI by up to 72.4% with a

small area overhead less than 5.0% compared to traditional floorplanning. We then explore over 1,000,000 micro-architecture configurations to find the optimum based on the TPWL model. This is in sharp contrast with the previous work [10], [11], which simply enumerate a limited number of micro-architecture candidates. Our experiments show that the average error of the TPWL model considering micro-architecture configuration and pipelined global interconnects is about 3.0% compared to cycle accurate estimation. Based on the accurate TPWL model, we find micro-architecture configurations and corresponding floorplannings that are less than 20.0% away from the ideal case, which counts no performance loss caused by insufficient parallelism, branch and cache missing penalty, and lack of function units. Our solutions are 26.9% better than [11] that was able to consider interconnect pipeline but only a limited number of micro-architecture configurations.

Related to our work, floorplanning for interconnect optimization has been studied for ASIC and System-On-Chip (SOC) designs. Early work on interconnect-driven floorplanning [12], [13], [14], [15] for ASIC chips focus on buffer block planning without interconnect pipelining. [16] for SOC designs targets at minimizing throughput degradation while preserving functionality when interconnects are pipelined by inserting throughput in the floorplanning objective function.

Floorplanning optimization for microprocessors has also been studied. [17] studied bus-driven floorplanning to optimize the routability and timing of buses in a given micro-architecture. [10] developed micro-architecture and floorplanning co-optimization, but no interconnect pipelining was considered. Furthermore, the micro-architecture was modeled by cycle-accurate micro-architecture level simulation and stored in tables, therefore [10] was able to consider only a limited number (thirty-two) of micro-architecture configurations compared to 1,000,000 micro-architecture to be explored in this paper. Micro-architecture floorplanning with interconnect pipelining was studied by [11] and the earlier version [18] of the paper to be presented here. [11] profiled module-to-module communication and solved an interconnect-pipelining aware floorplanning using mixed integer non-linear programming (MILP). Iterations between profiling and MILP are needed to guarantee the convergence of the overall design flow. Again, the micro-architecture configurations was limited as only four candidates was considered in the paper.

In the rest part of this paper, we introduce background knowledge in Section II, and present the TPWL model in Section III. We develop microprocessor floorplanning considering pipelined interconnects in Section IV, and co-optimization of micro-architecture and floorplanning in Section V. We report experiment results in Section VI and conclude the paper in Section VII.

II. Background

A. Bus Latency Vectors

Our experiment for floorplanning is based on two out-of-order SuperScalar implementations of MIPS instruction set. One is similar to the Alpha 21264 implemented in the ITRS [9] 100nm generation with an issue width of four and the other is implemented in the ITRS 70nm generation with an issue width of eight. A summary of the microprocessor configurations is presented in Table I. We group these modules into blocks that are each treated as an independent unit during floorplanning. We assume that interconnects between modules within the same block will not be pipelined. Blocks that are composed of multiple

Generation	100nm	70nm
ISA	MIPS	MIPS
SuperScalar	Width 4	Width 8
Functional Units	3 Integer ALU 1 Integer Mult. 1 FP Adder 1 FP Mult.	6 Integer ALU 2 Integer Mult. 2 FP Adder 2 FP Mult.
Register Update Unit	64 Instructions	128 Instructions
Load Store Queue	32 Instructions	64 Instructions
Fetch Queue	8 Instructions	16 Instructions
Clock Frequency	3 GHz	6 GHz
FF Insertion Length	$2000\mu m$	$707\mu m$

TABLE I

PROCESSOR USED IN EXPERIMENTS. THE FOUR-WAY SUPERSCALAR IS SIMILAR TO ALPHA 21264.

modules are the RUU block including Register Update Unit and Load Store Queue, Decode block including Fetch Queue and

the Decoder, Branch block including Fetch Unit and Branch Predictor, DL1 block including the Level 1 Data Cache and the DTLB, and the IL1 block including the Level 1 Instruction Cache and the ITLB. The L2 unified cache and all functional units are treated as independent blocks. We summarize the block area in Table II.

Block	Area (mm^2)	Block	Area (mm^2)
IALU	1.00	IMULT	1.00
F_ADD	1.94	F_MULT	2.07
RUU	3.04	Decode	1.44
Branch	2.27	L2	75.6
IL1	8.99	DL1	10.03

TABLE II

AREA OF LOGICAL BLOCKS IN THE 100NM GENERATION. THE AREA FOR THE 70NM GENERATION IS SCALED DOWN BY A FACTOR OF $(100/70)^2$.

The lengths of the interconnects between two blocks in Table II are computed according to the Manhattan distance, without loss of generality, between the centers of two blocks in the floorplan[1]. We treat the latency of each such interconnect as an independent variable. Changing the latency of one of these interconnects is effectively a change in the micro-architecture and will impact performance. In Table III we specify these interconnects with respect to their terminal blocks[2].

Bus id	Terminal blocks	Access ratio	Bus id	Terminal blocks	Access ratio
1	IALU, RUU	0.953	6	IL1, L2	0.001
2	IMULT, RUU	0.002	7	DL1, L2	0.046
3	FPAdd, RUU	0.137	8	Branch, IL1	0.391
4	FPMul, RUU	0.078	9	Decode, Branch	0.390
5	LSQ, DL1	0.409	10	Decode, RUU	0.390

TABLE III

BUSES THAT POTENTIALLY AFFECT CPI.

We define a bus latency vector ($\vec{B}$) for characterizing a floorplan as a vector containing the latency of each interconnect in Table III. If, for a given floorplan, Bus 1 has a latency of 3, Bus 2 has a latency of 4, Bus 3 has a latency of 7 etc. The $\vec{B}$ for that floorplan would be $\vec{B} = \{3, 4, 7, ...\}$. These latencies can be determined from the floorplan by dividing the length of each interconnect by the flip-flop (FF) insertion length as computed by interconnects and device parameters and simultaneous buffer and FF insertion algorithm in [3].

B. Cycle accurate simulation

For a given $\vec{B}$ we use out-of-order issue, cycle accurate simulation in the SimpleScalar 3.0 [19] framework to measure CPI for ten SPEC2000 benchmarks. These benchmarks includes *equake, mesa, gzip, art, bzip2, parser, vpr, gcc, go* and *mcf*, representing both integer and floating-point workloads.

In order to summarize the performance of a floorplan as described by $\vec{B}$ we take the arithmetic mean of the CPI for the above benchmarks. To obtain results more quickly we fastforward 200 million and simulate 100 million instructions. We observe that in our experiments fastforwarding 200 million instructions is enough to skip the initial stage and warm up the architecture structure such as caches and branch predictors. In addition, simulating 100 million instructions is long enough to obtain a steady-state estimation

To specify the latency of various interconnects in SimpleScalar in some cases we can simply modify latency values for modules in the configuration file. These cases include the buses for L1 and L2 data cache. The penalty for L2 instruction miss is different from the penalty for L2 data miss because the bus lengths are different for the two types of misses.

We insert queues between modules to buffer data for cases including the buses for instruction L1 cache, fetch to dispatch, and dispatch to issue. The L1 instruction cache interconnect latency is modeled by a fifo queue placed between the fetch unit and the cache. A branch cannot be identified by the fetch unit until it moves through the queue therefore prefetching proceeds

[1]Note that our method can be applied to the exact bus length and forked bus if such design information is provided.

[2]L2 cache is composed by three banks in our experiment and we consider the worst case latency.

speculatively to consecutive memory locations. When a branch is taken the prefetched contents of the queue are flushed and fetch proceeds from the target location. The latencies between the fetch and dispatch units and between the dispatch and issue units are modeled by a fifo queue between the fetch and dispatch units with length equal to the sum of the latencies of the two buses. This is because the dispatch stage is completely self contained and the effect of latency that comes immediately before dispatch is identical to that of latency that comes immediately after dispatch.

C. Floorplanning

Floorplanning determines the positions and shapes of blocks in a chip subject to the minimization of a cost function, which is usually a combination of area and total wire length, such as,

$$\alpha \cdot \frac{area}{area_{norm}} + \beta \cdot \frac{wire\_length}{wire\_length_{norm}}, \tag{1}$$

where $area$ and $wire\_length$ are the area and total wire length of the floorplan, respectively, α and β are user-defined weights. Because the metrics of $area$ and $wire$ are in different magnitude, we normalize them by typical vale of $area_{norm}$ and $wire\_length_{norm}$ in the objective function.

A category of floorplanning tools uses simulated annealing (SA) algorithm to solve the floorplanning problem[20], [21], [22]. SA starts with an initial floorplan and $moves$ to a new one by changing the positions or shapes of blocks. In each iteration the cost of the new floorplan is evaluated and the $move$ is accepted if the cost of the new floorplan is smaller than the old one. The $move$ may also be accepted regardless of cost with a probability dictated by the simulated "temperature" of the annealing. A $move$ that increases the cost is more likely to be accepted at a higher temperature. The temperature is decreased throughout the annealing based upon a schedule so that by the end only $moves$ that reduce the cost are likely to be accepted.

III. Trajectory piecewise-linear model

The TPWL model was originally proposed to model nonlinear dynamic systems [23]. In [23], a trajectory is generated by performing a single simulation of the system for a fixed "training" input. Then, a piecewise-linear model is built around the "training input". At first glance, one may think the TPWL model can be accurate only around the trajectory. As shown in [23], however, in terms of accuracy, TPWL can easily outperform other recently developed techniques based on quadratic reduction when the model is away from the trajectory.

In this paper, we solve the floorplanning (and later on with micro-architecture co-optimization) by simulated annealing (SA) [20], [21], [22]. We apply the TPWL model in floorplanning optimization by treating the optimization process of SA during floorplanning as a "training input". Specifically, the trajectory of a "training" SA process for floorplanning optimization is identified then a piecewise-linear CPI model is built around this "training" trajectory. We call this CPI model as trajectory piecewise-linear (TPWL) CPI model.

In this work we have improved the original TPWL model by introducing a TPC problem and iterations. The TPC problem reduces the cost to build the piecewise-linear model and iterations are employed to improve the accuracy of the TPWL model. The details of these two techniques will be discussed in this section.

The original TPWL model was designed for model order reduction for continuous phenomena. In essence, the TPWL model tries to pinpoint the trajectory in the solution space that SA optimization process covers, and then build a piecewise-linear model that is accurate for the neighborhood of the trajectory. It works for both continuous cases, such as model order reduction, and non-continuous cases, such as floorplanning optimization.

1) Construction of the TPWL model: The construction of the TPWL model consists of the following phases.

- $Sampling$: We sample an SA optimization process to obtain a trajectory in the solution space for bus vector $\vec{B}$. Note that a bus latency vector $\vec{B} \in \mathbf{I}^n$ represents a floorplan and a point in the $n-$dimension space ($n = 12$ in this paper). Specifically, we sample the floorplanning optimization process in the access ratio based approach.

- $Collecting$: We collect the sampled points of the trajectory in as few "balls" as possible in the $collecting$ phase. Shown in Fig. 1 is an illustration of using "balls" to collect points in the 2-dimension space, where the x-axis represents the latency of $bus1$ and y-axis represents the latency of $bus2$.

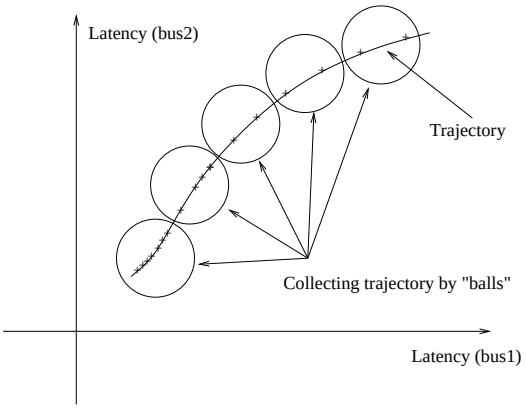

Fig. 1. Illustration of collecting trajectory in 2-dimension space.

We formulate the problem of minimizing the number of "balls" required to collect the points in a given trajectory as follows.

Formulation 1: **Trajectory points collecting (TPC)**: Given a set of points $\mathcal{P} \subset \mathbf{I}^n$ and radius $r \in \mathbf{I}$, find $\mathcal{C} \subset \mathbf{I}^n$ with minimum $|\mathcal{C}|$ while satisfying

$$\min_{c_j \in \mathcal{C}} \parallel p_i - c_j \parallel \leq r, \quad \forall p_i \in \mathcal{P}. \tag{2}$$

With respect to Fig. 1, c_j is the center of a ball and the minimum $|\mathcal{C}|$ leads to the smallest number of balls. One can see that TPC is NP-hard via analogy to the $set\text{-}cover$ problem [24]. Note that the TPC procedure was not precisely formulated in [23]. To improve model accuracy, we formulate the TPC procedure and approach it by the greedy algorithm developed for the $set\text{-}cover$ problem [25], [26]. The idea is to consecutively find a ball c which covers as many points in $\mathcal{P}$ as possible. We define a termination criteria T such that when the number of remaining points is smaller than T, the algorithm terminates.

- $Simulating$: By solving the TPC problem via the greedy algorithm, we obtain a set of points $\mathcal{C} \subset \mathbf{I}^n$, which covers most points in the trajectory with a given radius r. In fact, each point $c_j \in \mathcal{C}$ represents a bus latency vector $\vec{B}$, which specifies the latency for all buses in Table III. We obtain CPI by cycle accurate simulation for each $\vec{B}$ (See Section II-B) and build a CPI table indexed by the bus latency vector $\vec{B}$.

2) CPI estimation under TPWL model: Assuming the size of $\vec{B}$ is n and there are m entries in the CPI table, the ith table entry is represented as ($\vec{B}_i$, CPI_i). The CPI value of a target $\vec{B}$ can be estimated by each entry in the table. In general, the estimation is more accurate if the entry is closer to $\vec{B}$. Therefore, we first compute the distance between each entry and $\vec{B}$, and then employ an exponential function of the distance as a weight to compute the average estimation. The distance d_i between $\vec{B}$ to each $\vec{B}_i$ is computed as as

$$d_i = \parallel \vec{B} - \vec{B}_i \parallel_2, \quad i = 1, \cdots, m. \tag{3}$$

Then, we compute the weight of each entry by d_i:

$$\hat{w}_i = e^{-\beta d_i / \overline{d}}, \quad i = 1, \cdots, m, \tag{4}$$

where

$$\overline{d} = \min d_i, \quad i = 1, \cdots, m. \tag{5}$$

Note that β is a positive constant and is set to 25 same as [23]. Afterward, we compute the normalized weights as

$$w_i = \hat{w}_i / \sum_{i=1}^{|\mathcal{C}|} \hat{w}_i, \quad i = 1, \cdots, m. \tag{6}$$

Actually, from each entry in the CPI table, we can estimate the CPI value for the target $\vec{B}$ by the first order expansion,

$$CPI^i_{\vec{B}} = CPI_i + \overrightarrow{\nabla CPI_i} \cdot (\vec{B} - \vec{B}_i), \tag{7}$$

where

$$\overrightarrow{\nabla CPI_i} = \left[\begin{array}{c} \frac{\partial CPI}{\partial \vec{B}(1)} \\ \vdots \\ \frac{\partial CPI}{\partial \vec{B}(n)} \end{array} \right]_{|\vec{B}=\vec{B}_i}, \tag{8}$$

and $\frac{\partial CPI}{\partial \vec{B}(j)}$ is computed as follows

$$\frac{\partial CPI}{\partial \vec{B}(j)} = \frac{CPI(\vec{B}_i + \Delta) - CPI(\vec{B})}{\Delta}. \tag{9}$$

Note that $CPI(\vec{B}_i + \Delta)$ should be obtained from cycle accurate simulation. However, it is time-consuming to compute $\overrightarrow{\nabla CPI}$ for each entry in the CPI table. In this paper, we compute the average $\overrightarrow{\nabla CPI}$ and use it for all entries in the CPI table. The average $\overrightarrow{\nabla CPI}$ is obtained as follows:

$$\begin{aligned} \Delta_{max}(j) &= CPI(\vec{B}_{max-min}(j)) - CPI(\vec{B}_{max}), \\ \Delta_{min}(j) &= CPI(\vec{B}_{min}) - CPI(\vec{B}_{min-max}(j))), \\ \overrightarrow{\nabla CPI}(j) &= \frac{\Delta_{max}(j) + \Delta_{min}(j)}{2 \cdot D}, \quad j = 1, \cdots, n, \end{aligned} \tag{10}$$

where $\vec{B}_{max}$ and $\vec{B}_{min}$ is the bus latency vector with maximum and minimum latency on all buses, respectively. In this paper, we assume the maximum and minimum latency of a bus is 10 and 0, respectively, and denote the difference between them as D, i.e. $D = 10$. Also, $\vec{B}_{max-min}(j)$ is the same as $\vec{B}_{max}$ except the latency of the jth bus is the minimum. Similarly, $\vec{B}_{min-max}(j)$ is the same as $\vec{B}_{min}$ except the latency of the jth bus is the maximum. It can be seen from above that $\overrightarrow{\nabla CPI}(j)$ actually represents the sensitivity of the bus.

The estimated CPI value for $\vec{B}$ is computed as the weighted sum of $CPI^i_{\vec{B}}$, i.e,

$$CPI_{\vec{B}} = \sum_{i=1}^{m} w_i \cdot CPI^i_{\vec{B}}. \tag{11}$$

For convenience, we summarize the computations to estimate CPI for a bus vector $\vec{B}$ as in Fig. 2.

Compute CPI for $\vec{B}$
$d_i = \parallel \vec{B} - \vec{B}_i \parallel, \quad i = 1, \cdots, m.$
$\overline{d} = \min d_i, \quad i = 1, \cdots, m.$
$\hat{w}_i = e^{-\beta d_i / \overline{d}}, \quad i = 1, \cdots, m., \beta = 10.$
$CPI^i_{\vec{B}} = CPI_i + \overrightarrow{\nabla CPI_{\vec{B}_0}} \cdot (\vec{B} - \vec{B}_i)$
$CPI(\vec{B}) = CPI_{\vec{B}} = \sum_{i=1}^{m} w_i \cdot CPI^i_{\vec{B}}.$

Fig. 2. CPI estimation under the TPWL model.

IV. Microprocessor floorplanning considering pipelined interconnects

We study microprocessor floorplanning considering interconnect pipelining under designated micro-architecture configurations in this section. The microprocessor floorplanning studied in this paper is based on Simulated Annealing (SA) algorithm to minimize a given floorplanning objective with CPI included. The CPI value can be explicitly computed by TPWL model, or substituted by correlated metrics such as the access ratio of global interconnects. We discuss both methods in this section.

A. Microprocessor floorplanning

For microprocessor floorplanning studied in this paper, we additionally consider CPI in the objective function as,

$$\alpha \cdot \frac{area}{area_{norm}} + \beta \cdot \frac{wire\_length}{wire\_length_{norm}} + \gamma \cdot \frac{CPI}{CPI_{norm}}, \tag{12}$$

where $area$ and $wire\_length$ are the area and total wire length of the floorplan, respectively, α and β are user-defined weights, CPI, CPI_{norm}, and γ are the CPI value of the floorplan, the normalization value of CPI, and user-defined weights for CPI, respectively. For simplicity, we denote the objective combining area and total wire length as AL, and combining all area, total wire length and CPI as ALC.

The CPI value in (12) is computed by TPWL model as shown in Section III and an access ratio based approach. *Access ratio* of an interconnect (bus) is defined as the number of bus access over the total clock cycles for a benchmark. The arithmetic mean of the access ratio over all benchmark set is presented in Table III. Intuitively, the heavier for a bus to be accessed, the greater that the latency of this bus affects performance. Therefore, access ratio indicates the CPI-critical level for a bus.

The access ratio based approach weights CPI-critical interconnects by the arithmetic means of the access ratio as presented in Table III, and represent CPI in (12) by the weighted sum of interconnect latencies for all CPI-critical interconnects. Note that access ratios of interconnects are obtained by cycle accurate simulation.

An independent study in [11] has employed an similar idea to assign weights to interconnects between modules and solve the microprocessor floorplanning problem by mixed integer linear programming. Specifically, profiling counters are first used in microarchitecture simulator to record module-to-module communication, and then a cycle-accurate simulation is performed to obtain interconnect traffic between modules, which eventually determines the weights of interconnects between modules.

B. Microprocessor floorplanning with integrated TPWL model

We integrate the TPWL model with floorplanning based on the $Parquet$ package [22]. Shown in Fig. 3 is the overview of the CPI-aware floorplanning process. It starts with floorplanning to generate a trajectory with access ratio based approach to estimate CPI. Then, the set of balls $\mathcal{C}$ is found by solving the TPC problem on the trajectory. Accurate CPI value for all balls in $\mathcal{C}$ are found by cycle accurate simulation to generate the CPI table.

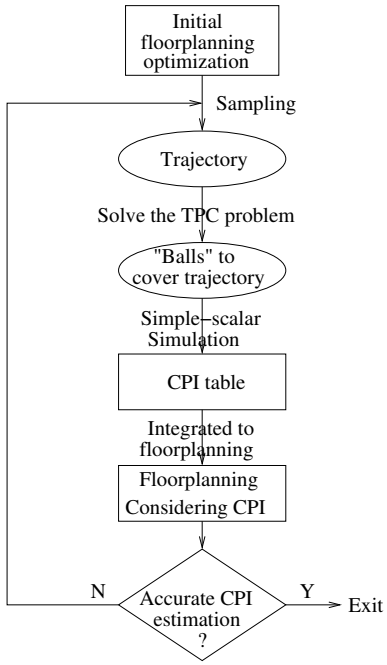

Fig. 3. Overview of CPI-aware floorplanning.

As shown in [22], the objective of AL is a linear combination of area and total wire length. In [22], after each move in SA, Δ of the objective is computed as the weighted sum of the changes of area and total wire length. In this paper, we optimize

ALC in the floorplanning by changing the objective function. Similar to [22], in our floorplanning, we compute Δ of the objective as the weighted sum of the changes in area, total wire length and CPI after each move. I.e.,

$$\Delta_{area} = \frac{area_{new} - area_{old}}{\sum area_{block}} \tag{13}$$

$$\Delta_{wire} = \frac{wire_{new} - wire_{old}}{wire_{old}} \tag{14}$$

$$\Delta_{CPI} = \frac{CPI_{new} - CPI_{old}}{CPI_{old}} \tag{15}$$

$$\Delta = w_{area} \cdot \Delta_{area} + w_{wire} \cdot \Delta_{wire} + w_{CPI} \cdot \Delta_{CPI}, \tag{16}$$

where $\sum area_{block}$ is the area of all blocks in the floorplan, and w_{area}, w_{wire} and w_{CPI} are the weights for area, total wire length and CPI, respectively. Note that in SA, a move is accepted if and only if

$$\begin{cases} \Delta < 0 \\ R < exp(\frac{-\Delta \cdot t_i}{t_c}) \quad \Delta \geq 0 \end{cases} \tag{17}$$

where R is a random value between 0 and 1, t_i and t_c are initial temperature and current temperature, respectively.

As shown in Fig. 3 that when the CPI estimation is not accurate enough, we return to the sampling phase for iterations. The TPWL model built in the previous iteration is employed to estimate CPI during floorplanning. Each iteration improves the accuracy of the TPWL model by expanding the CPI table with new entries obtained from the new trajectory in the current round[3]. The stop criteria is that the estimated CPI difference given by two consequent trajectories is smaller than a given threshold. In fact, our experiment results show that two extra iterations can improve accuracy significantly[4].

We employ iterations to improve the accuracy of TPWL model. The TPWL model is built on the trajectory that is sampled from a "training" SA optimization process in the floorplanning. One can imagine that CPI will be taken into account later on and the course of the SA optimization process will change. When the sampled "training" trajectory is off the course of the SA optimization with consideration of CPI, the CPI estimation is relatively inaccurate and the final quality of the floorplan suffers. Under this circumstance we introduce iterations to improve accuracy.

The first iteration of SA optimization is conducted without CPI minimization. From the second iteration on, CPI minimization is considered and new trajectories are generated based on the SA optimization trajectory. The accuracy of CPI estimation is improved with each iteration and the trajectories eventually converge. In practice we employ only several iterations and obtain the desired accuracy.

V. Micro-architecture and floorplanning co-optimization

In this section we extend the TPWL model to explore micro-architecture configurations thus to achieve co-optimization of micro-architecture and floorplanning. Specifically, we build a TPWL CPI model that models changes in floorplanning as well as 1,000,000 micro-architecture configurations. This not only explores more design freedoms but also produces better results, which demonstrates the generality of the TPWL model.

A. Variables of micro-architecture configurations

In this paper, we consider issue width, branch prediction, cache/TLB size and number of function units as variables of micro-architecture configurations.

1) Issue width: Issue width, also called machine width, is one of the most important parameters determining IPC. It is defined as the number of instructions that can be issued to the execution stage of the pipeline in a single cycle. As shown in [27], CPI can be approximated as

$$CPI = CPI_{ss} + CPI_{brmiss} + CPI_{imiss} + CPI_{dmiss}, \tag{18}$$

where CPI_{ss} is the maximum performance sustainable from background setting, and CPI_{brmiss}, CPI_{imiss}, and CPI_{dmiss} are the additional CPI caused by branch misprediction, instruction miss and data cache miss, respectively. Note that (18) shows that the upper bound of IPC is $1/CPI_{ss}$[5].

[3]A new entry with a small enough distance to an entry in the CPI table in fact is not calculated.

[4]The computation cost limit the number of iterations.

[5]We assume on average, $CPI = 1/IPC$ in this paper.

μ-arch modules	issue width 2	issue width 4	issue width 8
RUU size	**32**	**64**	**256**
LSQ size	**16**	**32**	**128**
Bpred (KB)	1, 2,**4**, 8, 16	1, 2, **4**, 8, 16	1, 2, **4**, 8, 16
L1 Icache (KB)	4, 8, **16**, 32, 64	8, 16, **32**, 64, 128	32, 64, **128**, 256, 512
L1 Dcache (KB)	4, 8, **16**, 32, 64	8, 16, **32**, 64, 128	32, 64, **128**, 256, 512
L2 Ucache (KB)	16, 32, **64** , 128, 256	32, 64, **128**, 256, 512	64, 128, **256**, 512, 1024
ITLB (entry)	16, 32, **64**, 128, 256	16, 32, **64**, 128, 256	32, 64, **128**, 256, 512
DTLB (entry)	16, 32, **64**, 128, 256	16, 32, **64**, 128, 256	32, 64, **128**, 256, 512
IALU	1, **2**,3,4	1,**2**,3,4	4, 5, **6**, 7, 8
FPALU	**1**, 2, 3	1, **2**,3,4	1, **2** 3, 4
IMult	**1**, 2, 3	**1**, 2, 3	1, **2** 3, 4
FPMult	**1**, 2, 3	**1**, 2, 3	1, **2** 3, 4

TABLE IV

THE RANGES OF CONFIGURATION VARAIABLES.

It is observed in [28] that the total number of instructions that can issue per cycle is roughly the square root of the number of instructions in the issue window. Therefore, issue width determines the size of RUU (Register Update Unit [29]) and LSQ. In this paper, we explore three issue width values of 2, 4 and 8. This range covers most current architectures.

2) Branch prediction: Equation (18) shows that branch miss-events increase CPI. A branch misprediction event causes fetching of useless instructions into pipeline with five to ten cycles misfetch delay penalty in a pipeline with five to nine front-end depth, as shown in [27].

The missing rate of branch prediction depends on the prediction method. We assume a combination of bimodal and 2-level method [30] in this study. [30] shows that this combinational method has a good performance in practice. The BTB size of the bimodal and 2-level methods is the input parameter to configure. In general, a larger size of BTB leads to a more accurate prediction.

3) Cache and TLB size: The cache missing event is another major factor that increases CPI, as shown in (18). Cache missing events cause stalling of the pipeline to bring data from next level caches and introduce delay penalties. For example, the L1 instruction missing penalty is about eight cycles in a pipeline with five to nine front-end depth[27]. The occasion of cache missing events depends on the size and organization of caches. In this study, we assume a smaller associativity for small cache/TLB sizes and a larger one for large cache sizes, and treat the cache/TLB size as the parameter to configure. The range of cache/TLB size is from 4K to 2048K in this study.

The physical size of cache/TLB in the floorplanning is calculated by Cacti[31] under the ITRS 100nm generation. Cacti takes the size and organization of a cache/TLB as input and estimates the physical size of the cache/TLB with high accuracy.

4) Number of function units: Insufficient number of function units can significantly affect performance. Our study shows that one less or more integer ALU can cause over 20% difference in IPC. However, redundant function units waste area and power. To a certain degree, the purpose of exploring the number of function units is to find the appropriate number of function units that just satisfies the requirement for computation resources based on other parameters such as issue width, cache/TLB size and so on. Because the number of function units can not exceed issue width, in this study it is limited between one and eight. Also, we consider four types of function units: integer ALU (IALU), integer multiplier (IMult), floating point ALU (FPALU), and floating point multiplier (FPMult).

B. Configuration candidates

Assuming six options for the size of cache/TLB, and five options for branch prediction and the number of function units, by simple multiplication of these options, we obtain that the total number of configuration options is over 1,000,000. This is in sharp contrast with previous work of [11], [10], which enumerate no more than 32 candidates. In this work, we treat all parameters in micro-architecture configurations independently and explore the aforementioned over 1,000,000 configurations.

As shown in (18), it is difficult to further improve performance by increasing cache/TLB size or adding more function units when CPI is close to CPI_{ss}. On the other hand, too small cache/TLB size or number of function units becomes the bottleneck of pipeline and hurts performance significantly. Therefore, with a fixed issue width, other parameters should stay in a certain range.

To determine a suitable range for each parameter we conduct simulations. Fixing other parameters at typical values, we determine the sensitive range for each parameter one by one. A sensitive range means that the change of this parameter in this range affects IPC significantly. We show these ranges for each parameter in Table 1. Note that each parameter can be changed independently. The total number of configurations in Table 1 is over 1,000,000. As suggested by [27], the RUU and LSQ size are deterministic for given issue width.

C. TPWL model for configurations and exploration of configurations

To extend TPWL model to cover micro-architecture configurations, we expand the bus latency vector to include both global interconnect latencies and micro-architecture configurations, and this new vector is employed to characterize floorplanning as well as the micro-architecture configurations during SA. This change does affect the general flow of building model presented in Section III.

In traditional SA based floorplanning, the solution space is explored by moves that change the shape, position, or orientation of a module in the floorplan. In addition to moves for floorplanning, we employ new moves explore micro-architecture configurations. These new moves include resizing branch predictor, resizing cache/TLB, and changing the number of function units. At first glance, changing issue width can also be a new move. However, to achieve high performance, most parameters should stay in a suitable range with respect to issue width. As observed in our experiment, in most cases a change of issue width in SA brings in inconsistency among parameters and degrades the quality of the solution. To explore the solution space more efficiently, we propose to employ multiple starts for issue width. In each start, issue width is fixed and the best case is selected among all starts.

VI. Experiment results

A. Experiment results for microprocessor floorplanning

We have implemented CPI-aware floorplanning based on the *parquet* package [22]. Below we assume that the interconnect distance between adjacent flip-flops is $2000\mu m$ under $3GHz$ clock in the ITRS $100nm$ generation and $6GHz$ clock in the ITRS $70nm$ generation based on the algorithm from [3].

In this section, we first verify the TPWL model, and then compare the floorplan with CPI minimization and without CPI minimization to demonstrate the necessity of considering CPI during floorplanning. Finally we compare the floorplans obtained by different approaches. This experiment is based on designated micro-architecture configuration shown in Table I.

1) Validation of TPWL models: During the construction of TPWL model, we solve a TPC problem to select points to cover the trajectory and then simulate these points to obtain a CPI table. Because cycle accurate simulation is time-consuming[6], we empirically decide the radius, r, of "balls" to control the number of selected points. In general, small r leads to large $|C|$ and large number of cycle-accurate simulations to construct the CPI table, and a larger CPI table achieves better accuracy. Therefore, we choose r as small as possible while the number of cycle-accurate simulations is acceptable in our experiment. Specifically, we set r equal to 5, and 8 latencies in our experiment for the ITRS 100nm and 70nm generations, respectively. Also, bus latencies are weighted by normalized sensitivity when computing the distance between two bus latency vectors. This is due to the fact that the latency in a more sensitive bus can cause larger CPI difference and therefore the two bus latency vectors should be treated more distant.

Overall, the procedure to build the TPWL model is as follows. We conduct SA to minimize AL and build up the CPI table of the TPWL model by sampling the SA process and running cycle accurate simulations. We repeat the SA process two more times with objective of ALC to build the iTPWL model. The total number of cycle accurate simulation that is needed to build the TPWL model is around 100.

The purpose of the CPI model developed in this paper is to estimate CPI accurately during a SA optimization process with consideration of CPI minimization. To verify the accuracy of the proposed model, we compare it with the CPI obtained by

[6]Several hours may be needed to simulate one point.

cycle accurate simulation for floorplanning solutions selected from a SA optimization process and show the results in Fig. 4. As shown in Fig. 4 for the processor in the 70nm generation, the average error of TPWL is 1.8%. We have also observed similar results for the 100nm generation.

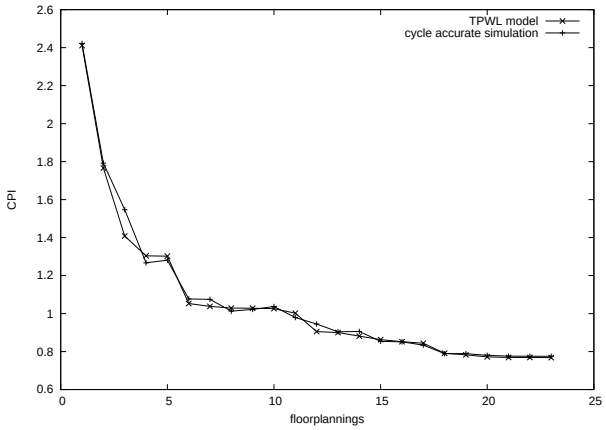

Fig. 4. TPWL model estimation results for $70nm$ (Average error is around 1.8%).

	μp in 100nm generation				μp in 70nm generation			
	best		avg		best		avg	
Metrics	AL	ALC	AL	ALC	AL	ALC	AL	ALC
White Space(%)	3.62	7.12	9.87	13.2	4.52	6.82	7.80	8.56
CPI	1.79	0.78 (-56.4%)	1.73	0.81 (-46.8%)	1.67	0.79 (-52.7%)	2.97	0.82 (-72.4%)
TWL(10^2 mm)	4.61	5.35 (+16.1%)	4.84	5.19 (+7.2%)	5.60	5.60 (+0.0%)	5.52	6.15 (+11.4%)
Total/Max. #FF	53/6	42/6	54/8	40/7	168/20	96/13	145/21	98/16
Area(10^2 mm^2)	1.18	1.23 (+4.2%)	1.27	1.32 (+3.9%)	2.18	2.23 (+2.3%)	2.25	2.27 (+0.8%)

TABLE V

COMPARISON BETWEEN FLOORPLANS OBTAINED BY DIFFERENT OBJECTIVES.

	μp in 100nm generation				μp in 70nm generation			
	best		avg		best		avg	
Metrics	AR	$TPWL$	AR	$TPWL$	AR	$TPWL$	AR	$TPWL$
White Space(%)	15.13	7.12	15.21	13.2	7.65	6.82	11.40	8.56
CPI	0.78	0.78 (-0.0%)	0.90	0.81 (-10.0%)	0.80	0.79 (-1.3%)	0.84	0.82 (-2.4%)
TWL(10^2 mm)	4.95	5.35 (+8.0%)	5.83	5.19 (-11.0%)	5.51	5.60 (+1.6%)	6.49	6.15 (-5.2%)
Total/Max. #FF	25/5	42/6	54/8	26/6	81/10	96/13	145/21	77/12
Area(10^2 mm^2)	1.35	1.23 (-8.9%)	1.35	1.32 (-2.2%)	2.25	2.23 (-0.9%)	2.35	2.27 (-3.4%)
Objective	3.00	2.88 (-4.0%)	3.33	3.07(-7.8%)	3.27	3.26 (-0.3%)	3.57	3.43 (-3.9%)

TABLE VI

COMPARISON BETWEEN TPWL AND ACCESS RATIO BASED APPROACH.

2) Comparison of floorplanning with different objectives: We compare the floorplans obtained by SA subject to the objectives of AL and ALC, respectively. Note that the AL objective is used by the traditional floorplanning and the area for each block in the floorplan is shown in Table II. The size of blocks are obtained from Alpha 21264 and scaled to the 100nm and 70nm generations. We summarize the results of floorplanning using these objectives in Table V. For each objective we ran ten cases of floorplanning as the floorplanning algorithm in [22] is not deterministic. We show both best and average results for comparison.

We first show the white space rate of these floorplans in the first row. The low rate indicates the good quality of these floorplans. We show the CPI of floorplans in the second row. The ALC and AC solutions can reduce CPI over AL by around 50.0%, which is corresponding to around 100.0% increase in throughput. The results on total wire length show that there

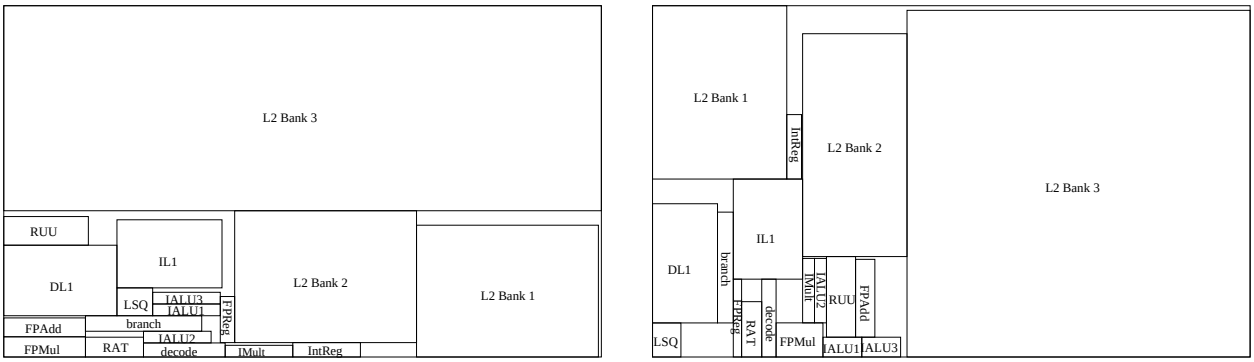

Fig. 5. Comparison between floorplanning without (a) and with (b) CPI minimization.

is no strong correlation between CPI and total wire length. This indicates that the objective of minimizing total wire length in traditional floorplanning does not maximize performance. Instead, latency of pipelined interconnects in CPI-critical paths should be considered to maximize performance. Similarly, the results in the fourth row show that minimizing total wire length does not necessarily reduce the total number of flip-flops and the maximum number of flip-flops in a single bus. We show the area in the fifth row. The ALC results in the lowest CPI with only a small area overhead of smaller than 5.0% over AL.

To demonstrate how CPI minimization affects the final floorplan, we compare the floorplan obtained without CPI minimization and with CPI minimization under the 100nm generation in Fig. 5. In our experiment, the aspect ratio of the die is fixed but those of blocks are flexible. As shown in Fig. 5 (b), with consideration of CPI minimization during floorplanning, the length of critical buses that have high access ratios and intuitively significant effect on CPI is generally shorter than that in (a). For example, in (b) the "RUU" module is placed close to the modules of "IALU", "FPAdd" and "FPMul". Note that the buses between these modules are critical buses based on the access ratio in Table III.

3) Comparison of floorplans under different CPI models: In this Subsection, we compare the floorplans obtained by TPWL model and access ratio based approach to show the quality of the approach. Specifically, for both methods, we choose the best and average results among ten optimization results for comparison and show it in Table VI.

We compare the floorplans with an ALC objective function, white space ratio, CPI, total wire length (TWL), total/max number of flip-flops and area. The objective function is the weighted sum of normalized area, total wire length and CPI. As shown in the table, although TPWL model always finds a floorplanning better than that of access ratio based approach, the total objective function of the floorplanning obtained by access ratio is less than 10.0% away from that of the TPWL model.

This result indicates that the access ratio based approach is effective in considering the impact of global interconnects especially when taking cost into consideration. In our experiment, the TPWL model is built with one-time cost of around 100 cycle accurate simulations. Since each simulation takes several hours to finish, it takes around several hundred hours to build the model[7]. Although these simulations can be done in parallel, it is still time-consuming. In contrast, because it only needs to simulate once over all benchmarks, the access ratio based approach has a much smaller cost.

However, in this paper we prefer TPWL model due to the generality of the model. As shown in this paper, it is easily expanded to consider micro-architecture reconfigurations, which can never be done by heuristics such as the access ratio based approach. We believe the TPWL model can still be expanded to consider other design freedoms and objectives while maintaining a high quality. Also, TPWL model is completely compatible with simulation methods that estimate CPI quickly, such as trace-driven simulations. Therefore, one can employ fast CPI estimation methods such as [32] instead of cycle accurate simulation to obtain "accurate" CPI value to construct the TPWL model. This can significantly reduce cost and may broaden the usage of the model.

B. Experiment results for micro-architecture and floorplanning co-optimization

1) IPC versus area: We have integrated the TPWL model and the methodology to explore micro-architecture configurations into *Parquet*. The physical sizes of modules, such as RUU, decoder, LSQ, and etc, are obtained from Alpha 21264 [33] and

[7]Because the SA finishes in seconds, we omit its runtime here.

scaled to the 100nm and 70nm generations. The physical size of caches are computed by Cacti[31] according to configuration values. In this section, we always employ SA optimization with objective of AC due to the fact that total wire length has no strong correlation with throughput as shown in Section VI.

We show the curve of IPC versus area in Figure 6 for the purpose of validating the TPWL model and showing the relation between IPC and area. The data points are obtained by executing multiple SA runs with different weights assigned to area and IPC in the objective function. We also choose the best IPC value for solutions with close area values.

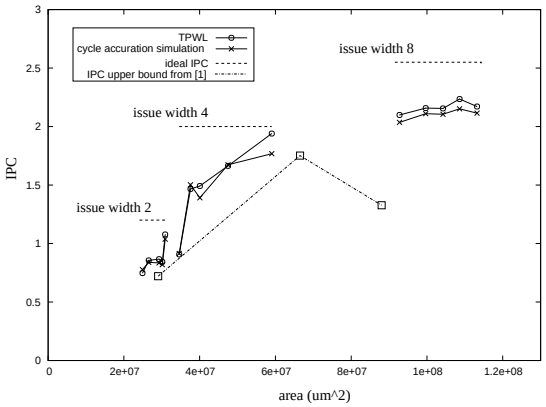

Fig. 6. IPC versus area (under TPWL model and cycle accurate simulation).

We first validate the TPWL model by comparing the IPC value obtained by the model and by cycle accurate SimpleScalar simulation in Figure 6. The data points in the figure indicate that the average error of TPWL model is around 3.3%. We then show the trend of IPC with respect to increasing area. The dotted lines in the figure represent the ideal IPC value with no performance loss caused by insufficient of instruction parallelism, missing events and lack of function units. We approximate the ideal IPC value for each issue width by providing large enough branch predictors, caches, TLBs and enough number of function units in the micro-architecture configurations and conduct SimpleScalar simulations. As shown in the figure, the optimization results are less than 20% away from ideal cases. We also compare our IPC value with the IPC value of the first three configurations enumerated in [11], which is also shown in Table VII as the last value. These three configurations are for issue width 2, 4 and 8, respectively. We show the IPC value of these three configurations but without considering interconnect latency since the floorplanning is unavailable. Therefore, shown in the figure is actually the upper bound IPC value from [11]. Also, we obtain the area of each configuration by mapping configuration parameters to corresponding physical size used in this paper. As we can see from the figure, the co-optimization methodology proposed in this paper leads to much better designs (26.9% higher IPC) than configurations common used and studied in [11]. Note that the case of issue width 8 in [11] experiences a major performance degradation due to limited size of caches.

As discussed in Section V-A, reducing CPI encounters a diminishing return in terms of area efficiency when performance approaches the optimal value. This phenomenon is observed in Figure 6. Within the same issue width, it is difficult to improve performance by increasing the chip area when IPC is close to the ideal. A similar trend is also observed among different issue widths. Compared to issue width 4, issue width 8 is less effective to improve IPC by the same amount of area increase. The primary reason is that it becomes more difficult to find enough ILP when issue width increases. The increase in global interconnect latency with the increase in area also contributes to this trend.

2) Configuration details: To reveal the micro-architecture configuration details, we show all micro-architecture configuration parameters for average and best cases of multiple SA runs in Table VII. Note that all the IPC values in the table are obtained from cycle accurate simulation with global interconnect latencies extracted from corresponding floorplannings. As shown in the Table, average values are fairly close to the best case value, indicating the convergence of SA optimization. Also, the obtained configuration by the proposed method has a significant performance advantage over the upper bound of designated configurations used in [11] without considering the impact of global interconnects.

μ-arch modules	issue width 2	issue width 4	issue width 8
Bpred	4.1KB/8KB/128	4.2KB/8KB/512	3.2K/8K/512
L1 Icache (KB)	12.8/16/8	27.2/64/64	129.9/256/8
L1 Dcache (KB)	12.8/4/8	31.5/8/64	97.0/32/8
L2 Ucache (KB)	60.4/32/64	60.8/64/512	209.5/128/128
ITLB (Entry)	32.0/32/32	46.9/32/128	62.1/32/128
DTLB (Entry)	26.3/32/32	42.1/128/128	68.9/32/128
IALU	1.7/2/1	1.9/4/3	5.8/7/3
FPALU	1.5/1/1	1.7/2/1	1.9/4/1
IMult	1.5/1/1	1.8/1/1	2.3/4/1
FPMult	1.5/2/1	1.7/2/1	2.2/2/1
Area(mm^2)	29.8/32.9/29.4	51.2/59.0/71.2	100.7/108.6/91.5
IPC	0.95/1.05/0.72	1.50/1.77/1.75	2.11/2.15/1.25

TABLE VII

CONFIGURATION DETAILS (AVG/BEST/[11]).

VII. CONCLUSIONS AND DISCUSSIONS

Considering interconnect pipelining, we have developed a floorplanning formulation minimizing CPI (cycles per instruction) for given microarchitecture. Compared to conventional floorplanning minimizing area and wire length, the new floorplanning formulation obtains a floorplan with around 100.0% throughput increase and a small area overhead of less than 5.0%. CPI optimization during floorplanning is achieved by shortening the lengths of CPI-critical buses. At first glance, the set of CPI-critical buses is a subset of all global interconnects and the traditional floorplanning objective of minimizing total wire length should lead to a floorplan with optimized system performance. However, we have shown in the experiment that minimizing total wire length does not necessarily lead to minimization of CPI. Yet, using the bus access ratio as weight and minimizing the weighted interconnect length is shown to be a good heuristic for CPI-aware floorplanning.

We have developed a TPWL model for CPI to consider micro-architecture configurations and interconnect pipelining, and further developed co-optimization of micro-architecture and floorplanning based on it. We explore over 1,000,000 configurations candidates and find micro-architecture configurations and corresponding floorplannings with an IPC (instructions per cycle) less than 20.0% away from the ideal case, which counts no performance loss caused by insufficient parallelism, branch and cache missing penalty, and lack of function units. Our solutions are 26.9% better than [11] that was able to consider only a limited number of micro-architecture configurations.

There is a tradeoff between quality and simulation runtime to build the TPWL model. Tracing SA reduces the simulation runtime without sacrificing the accuracy around the SA trajectory. In addition, cycle-accurate simulation may be replaced by traced based simulation [27], or incremental simulation to reduce model building time. Note that once models are built, the runtime is virtually same for TPWL model and access ratio based approach. The TPWL model is a general model and can be expanded to consider more design freedoms and objectives, such as power, and still maintain high quality. This will be our future work.

Also, the approach and experiment results reported in this paper are based on an out-of-order SuperScalar architecture. Our future work will consider techniques such as Hyper-Threading and Globally-Asynchronous Locally-Synchronous (GALS). In our work, blocks are treated as soft modules, this might be acceptable for certain elements, such as control logic, but not for some others, such as caches. We will consider this and more physical constraints on blocks in the future.

REFERENCES

[1] D. Matzke, "Will physical scalability sabotage performance gains?," *Computer*, vol. 30, pp. 37–39, 1997.

[2] P. Cocchini, "Concurrent flip-flop and repeater insertion for high performance integrated circuits," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 268–273, Nov 2002.

[3] W. Liao and L. He, "Full-chip interconnect power estimation and simulation considering concurrent repearter and flip-flop insertion," in *ICCAD*, pp. 574–580, 2003.

[4] J. Smith and G. Sohi, "The microarchitecture of superscalar processors," *Proceedings of the IEEE*, vol. 83, pp. 1609 – 1624, Dec. 1995.

[5] B. A. Gieseke and et al, "A 600mhz superscalar risc microprocessor with out-of-order execution," in *Proc. IEEE Int. Solid-State Circuits Conf.*, pp. 176–177, 1997.

[6] J. Clabes, J. Friedrich, M. Sweet, J. Dilullo, S. Chu, D. Plass, J. Dawson, P. Muench, L. Powell, M. Floyd, B. Sinharoy, M. Lee, M. Goulet, J. Wagoner, N. Schwartz, S. Runyon, G. Gorman, P. Restle, R. Kalla, J. McGill, and S. Dodson, "Design and implementation of the power5/spl trade/ microprocessor," in *Proc. IEEE Int. Solid-State Circuits Conf.*, pp. 56 – 57, 2004.

[7] S. Nussbaum and J. Smith, "Modeling superscalar processors via statistical simulation," in *International Conference on Parallel Architectures and Compilation Techniques*, pp. 15–24, 2001.

[8] M. Oskin, F. Chong, and M. Farrens, "Hls: combining statistical and symbolic simulation to guide microprocessor designs," in *27th Int. Symp. on Computer Architecture*, pp. 71–82, June 2000.

[9] International Technology Roadmap for Semiconductors (ITRS). 2003.

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

[11] M. Ekpanyapong, J. R. Minz, T. Watewai, H.-H. S. Lee, and S. K. Lim, "Profile-guided microarchitectural floorplanning for deep submicron processor design," in *Proc. Design Automation Conf*, 2004.

[12] J. Cong, T. Kong, and D. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. Int. Conf. on Computer Aided Design*, pp. 358–363, Nov. 1999.

[13] I.-R. Jiang, Y.-W. Chang, J.-Y. Jou, and K.-Y. Chao, "Simultaneous floor plan and buffer-block optimization," pp. 694–703, May 2004.

[14] K. Wong and E. Young, "Fast buffer planning and congestion optimization in interconnect-driven floorplanning," in *Proc. Design Automation Conf*, pp. 411–416, 2003.

[15] C. wing Sham, E. Young, and H. Zhou, "Interconnect-driven floorplanning by searching alternative packings," in *Proc. Asia South Pacifi c Design Automation Conf.*, pp. 417–422, 2003.

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

[17] F. Rafiq, M. Chrzanowska-Jeske, H. Yang, M. Jeske, and N. Sherwani, "Integrated floorplanning with buffer/channel insertion for bus-based designs," pp. 730–741, June 2003.

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

[19] D. Burger and T. Austin, *The simplescalar tool set version 2.0*. University of Wisconsin-Madison, 1997.

[20] N. Sherwani, *Algorithms For VLSI Design Automation*. Kluwer, 3rd ed., 1999.

[21] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence pair," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1518–1524, 1996.

[22] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. IEEE Int. Conf. on Computer Design*, pp. 328–334, 2001.

[23] M. Rewienski and J. White, "A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pp. 155–170, 2003.

[24] S. Rajagopalan and V. Vazirani, "Primal-dual rnc approximation algorithms for (multi)-set (multi)-cover and covering integer programs," in *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pp. 322–331, 1993.

[25] D. S. Johnson, "Approximation algorithms for combinatorial problems," *J. Comput. Sys. Sci.*, vol. 9, pp. 256–278.

[26] L. Lovasz, "On the ratio of optimal integral and fractional covers," *Discrete Math.*, vol. 13, pp. 383–390.

[27] T. Karkhanis and J. Smith, "A first-order superscalar processor model," in *omputer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pp. 338 – 349, June 2004.

[28] E. Riseman and C. Foster, "The inhibition of potential parallism by conditional jumps," *IEEE Trans. on Computers*, vol. C-21, pp. 1405–1411, 1972.

[29] G. Sohi, "Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers," *IEEE Trans. on Computers*, vol. C-39, pp. 349–359, Mar. 1990.

[30] McFarling, "Combining branch predictors," Tech. Rep. TN-36, DEC WRL, June 1993.

[31] S. Wilton and N. Jouppi, "Cacti: an enhanced cache access and cycle time model," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 677 – 688, 1996.

[32] T. Karkhanis and J. Smith, "A first-order superscalar processor model," in *ISCA*, 2004.

[33] L. Gwennap, "Digital 21264 sets new standard," *Microprocessor Report*, vol. 10, October 28 1996.