Micro-architecture Pipelining Optimization with Throughput-Aware Floorplanning*

Yuchun Ma* Zhuoyuan Li* Jason Cong** Xianlong Hong* Glenn Reinman** Sheqin Dong* Qiang Zhou*
*Department of Computer Science & Technology, Tsinghua University, Beiijng 100084, P.R.China
Email: myc@mail.tsinghua.edu.cn hxl-dcs@tsinghua.edu.cn
**Department of Computer Science & Technology, UCLA, USA

Abstract - For modern processor designs in nanometer technologies, both block and interconnect pipelining are needed to achieve multi-gigahertz clock frequency, but previous approaches consider block pipelining and interconnect pipelining separately. For example, all recent works on wire pipelining assume pre-pipelined components and consider only inserting pipeline stages on point-to-point wire or bus connections. To the best of our knowledge, this paper is the first that considers block pipelining and interconnect pipelining simultaneously. We optimize multiple critical paths or loops in the micro-architecture and insert the pipelines stages optimally in the blocks and wires of these loops to meet the clock frequency requirement. We propose two approaches to this problem. The first approach is based on mixed integer linear programming (MILP) which is theoretically guaranteed to produce the optimal solution, and the second one is an efficient graph-based algorithm that produces near-optimal solutions. Experimental results show that simultaneous block and interconnect pipelining leads to more than 20% improvement over wire-pipeling alone on the overall processor performance. Moreover, the graph-based approach gives solutions very close to the MILP results (2% more than MILP results on average) but in a much shorter runtime.

1. Introduction

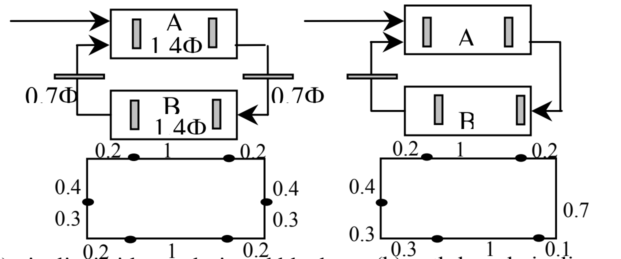
Industry trends indicate that the operating frequencies of leading-edge microprocessors have been doubling with every process generation [1], having broken the gigahertz barrier several years ago. The fraction of the interconnect delay in the shrinking system clock period has been increasing across process generations and become dominant in the deep submicron (DSM) regime. As global wire delays (e.g. register by pass wires) and RAM/CAM delays scale much slower than transistor delays, deeper superscalar pipelines experience increased latencies and a significant degradation in instruction throughput.

To ensure that the system operates at the right frequency, the delay of the global wire has to be distributed over several clock cycles by inserting flip-flops [2][3]. For instance, even with the optimal buffer insertion and wire-sizing, five clock cycles are still needed to go from corner-to-corner for the predicted die of 28.3*28.3 mm^2 in the 70-nm technology generation, assuming a 5.63-GHz clock frequency [12]. As a result of the projected wire delays, overall system performance could be reduced by a factor of up to 2 to 3, due to wire delays. Therefore, physical design should focus on keeping the throughput-critical paths as short as possible. The throughput-aware strategies [5][6], which identify and optimize throughput-critical paths using floorplanning algorithm to provide feedback to architects at very early design stages, are what is needed by the microarchitecture design.

There are some recent attempts on throughput-aware design at the floorplanning level. The MEVA system [24] was the first automated microarchitecture exploration system combined with physical planning that considers both IPC and clock frequency and optimizes the overall BIPS. It was enhanced in [6] with a sensitivity-based IPC model. In [7], a throughput lookup table (LUT), indexed by the set of bus latencies, is constructed using cycle-accurate simulations. For a given layout (and the corresponding bus latencies), the throughput is evaluated from the LUT using some distance metrics. The approach in [10] assigns weights to each of the system buses which are proportional to the amount of traffic seen on the buses, operating under the notion that the more often a bus is accessed, the more critical it is. The objective of the floorplanner then is to minimize a weighted sum of bus latencies. The authors in [11] propose an approach based on the methodology of a statistical design of experiments which identify the performance critical buses in a micro-architecture. The performance impact of each bus is quantified by assigning weights and the approach is applied at the floorplanning level. However, all these works [7,10,11] focus on wire-pipelining under the assumption that the blocks are separately designed subject to a clock frequency, and the wire pipelining is then carried out on the global wires of the circuits. As a result, they consider only wire pipelining on the buses or two-pin connections between blocks instead of the entire critical paths across multiple architecture components. Therefore, their pipelining designs are often sub-optimal due to the possible utilized slacks in block pipeline designs, resulting extra latencies along the paths which degrade the performance of the system.

As a popular technique for performance optimization of sequential circuits, retiming [9][22][23] can optimize the clock period by moving the flip-flops within a circuit while keeping its functionality, but the objective of retiming is to minimize the clock period while the latency numbers along the loops are remained unchanged. The pipeline designs inside blocks are still assumed fixed during the retiming process. In the retiming process, though the clock period is minimized with the given latency number, the throughput of the system is sub-optimal because of the limitation of the fixed pipeline design inside blocks.

Recent works [6][8] use throughput sensitivity models for selected critical paths, and these models guide the floorplanner to optimize the system throughput. Though it uses a path-based performance model, they assume all the paths can be optimized and pipelined independently, which is not true in general, as multiple paths may share a common block and the pipelining design of this block has to be *consistent* in all these paths. Hence, the performance sensitivity model used in [6][8] is not accurate. Since the throughput-aware floorplaner is intended for providing feedback to architects during very early design stages (even before the pipeline stages and HDL are generated), it is important to use accurate and realistic performance models. Therefore, it is important to consider both block and wire pipelining simultaneously. Fig.1 shows a simple example with a loop (A-B-A) which demonstrates the difference between wire pipelining and path pipelining. Suppose that the clock period is Φ and the two components are identical, each with the delay of 1.4Φ. If the components are pipelined independently without the knowledge of wire delay, it may end up with a pipeline solution where the maximum delay from the input pin to any flip-flop inside



(a) pipeline with pre-designed blocks (b) path-based pipeline

Fig.1 The wire pipelining and the path-based pipelining

* This work is supported by the NSFC 90407005 and 60606007, NSF CCR-0096383 and Hi-Tech Research & Development (863) Program of China 2004AA1Z1050

the blocks (T_{su}) and the maximum delay from any flip-flop inside the block to the output pin((T_{co}) are equally divided ($T_{su} = T_{co}= 0.2\Phi$). Since blocks are previously pipelined, there is less flexibility for us to pipeline the two paths. Now if it turns out that the delays for all wire segments to be 0.7Φ, the latency for this loop will be 6 (Fig.1(a)). However if we can perform pipeline design for blocks and wires together along the loop, the blocks can be pipelined with the consideration of wire delay. A feasible pipeline design can reduce the loop latency to 5 cycles as shown in Fig.1(b). Hence, the latency number can be optimized with the path-based pipeline design compared to the wire pipelining since the latency number is really sensitive to the distribution of the flip-flops.

In this paper, we propose a novel optimization methodology of architecture pipelining with physical design. Unlike previous works, we provide the architects the accurate pipeline design for both blocks and wires. To the best of our knowledge, this paper is the first that considers block pipelining and interconnect pipelining simultaneously. Our approach optimizes multiple critical paths or loops in microarchitecture by inserting and positioning the pipeline stages optimally in blocks and wires of the loops to meet the clock frequency requirement. We propose two approaches to this problem.

The first approach formulates the problem as a Mixed Integer Linear Programming (MILP) problem. Given a floorplanning design of a micro-architecture design, we can determine the distribution of flip-flops for both blocks and wires by solving the MILP problem. Though this approach can theoretically give the optimal solution for pipeline design, it is too time consuming to be embedded in the process of floorplanning. The second one is an efficient graph-based dynamic scanning heuristic that produces near-optimal solutions. It can be integrated with the throughput-aware floorplanner which feeds the floorplan engine with the actual latency information. Our approach enables the automated design for initial architecture design with consideration of both the architectural side and the physical design. Experimental results show that simultaneous block and interconnect pipelining leads to more than 20% improvement over wire-pipelining on performance. Moreover, the dynamic scanning heuristic gives solutions very close to MILP results (only 2% more than MILP results on average) but in a much shorter time.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of the superscalar processor and the problem description of the path-based pipelining; in Section 3, we first formulate the path-based pipelining into a MILP problem and then present a near-optimal approach which can pipeline all the critical paths by dynamically traversing the path graph. Section 4 gives the framework of the throughput-driven floorplanner which integrates the pipelining design. The experimental results with an Alpha 21264 processor design are shown in Section 5. We conclude the paper and discuss the future research directions in Section 6.

2. Problem Formulation

Superscalar processing is the ability of a microprocessor to initiate multiple instructions into multiple pipelines so that the computations of many instructions can be done in parallel if they are not dependent on each other. Fig.2 shows the important elements of a typical high-performance superscalar processor. Each of these blocks spans one or more pipeline stages, and instructions typically flow through these stages in more or less the order in which they have been listed. Since in micro-architecture design, there are many different sub-systems which have common components to each other - integer units, floating point units, main memory interface, and so on; extra delay in any of these sub-systems will have different effects on system performance. Usually the designer will have preferences, based on system performance. Therefore the path-based pipelining is to plan the latency number along each path by inserting flip-flop inside blocks or along the wires while the delay constraints are satisfied. The objective is to optimize the performance of the whole system.

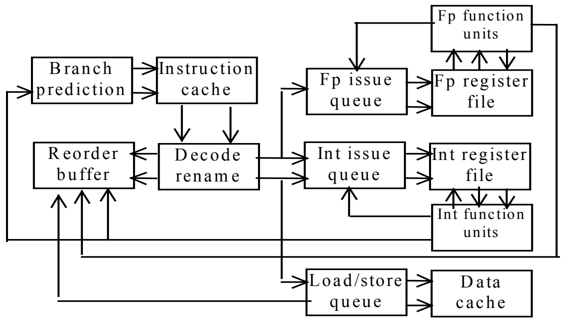


Fig. 2 Elements of a modern out-of-order, superscalar processor.

Since there are complex interactions between loops in superscalar processors, unlike the previous wire pipelining approaches which inserts flip-flops on pin-to-pin wire segments or buses, the path-based pipelining design should optimize the distribution of flip-flops along multiple paths. We define path-based pipelining as the *Simultaneous Block and Interconnect Pipelining (SBIP)* Problem. In the SBIP problem, we may insert flip-flops not only on the wires between blocks, but also inside blocks. Therefore we treat blocks and wires equally as the components with corresponding delays. To formulate the SBIP problem clearly, we represent the micro-architecture design by a *path graph* $G(V,E)$, where each directed edge e_i represents a wire or a block, d_e is the delay for this wire or block and each node v is the joint between a block and a wire. Each critical path in micro-architecture corresponds to a path between two nodes in graph G. With the directed graph G, the SBIP problem can be transformed into a problem of labeling the edge with the number of flip-flops on each edge. Let n_{ei} be the number of flip-flops of edge e_i. A solution can be viewed as a labeling of the edges n: $E \rightarrow Z$, where Z is the set of non-negative integers. We assume the inputs and outputs of the design are registers which can be treated as flip-flops. To meet the target clock period Φ, the delay between any two flip-flops along the same path is less than clock period Φ. The performance of the architecture can be evaluated by the weighted sum of n_{ei} along the paths Therefore the objective is to find a feasible solution with the optimal performance. Fig.3 gives an example with three critical loops (A-B-D-A, B-C-B, A-E-A). To distinguish the block delays with the wire delays, we use solid lines to represent blocks and different dashed lines for wires along different loops. In graph G, there may be multiple paths passing the same edge, therefore it is not a trivial problem to optimize the performance by minimizing the weighted sum of latencies along the critical paths. To solve the SBIP problem, it depends on not only the delay along paths, but also the topology of the path graph.

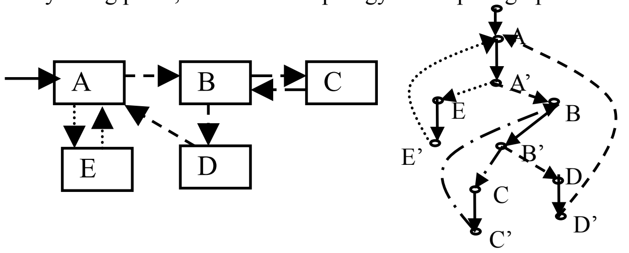


Fig.3 The path graph

3. Simultaneous Block and Interconnect Pipelining

Based on the SBIP formulation, we propose two approaches to handle path-based pipelining. We first formulate the problem into a MILP problem and a dynamic scanning heuristic is proposed.

3.1 Optimal pipeline design problem

For each node, we define a term $a(v, P_i)$ that represents the maximum arrival time at node v along path P_i, which is the longest delay from a flip-flop or source node to v along P_i (as shown in Fig.4). With the given clock period Φ, the set of paths P with the weight w_{pi} for each path P_i, we can then formulate the problem as the following MILP.

Obj. $Min \sum_{P_i \in P} (w_{Pi} \times \sum_{e_i \in P_i} n_{ei})$

s.t. $a(v,P_i) \le \Phi \quad \forall\, v \in V,\ P_i \in P$ (1)

$a(v,P_i) \ge 0 \quad \forall\, v \in V,\ P_i \in P$ (2)

$n_{ei} \ge 0 \quad \forall\, e_i \in E$ (3)

$a(v,P_i) \ge a(u,P_i) + d_{ei} - \Phi * n_{ei} \ \forall\ e_i \in E$ and e_i is a connection from node u to node v *along* P_i. (4)

The MILP formulation for path-based pipelining is a traditional mixed integer linear programming. Suppose the length of path Pi is $|P_i|$, then there are $\sum|P_i|$ real variables $a(v,P_i)$, $|E|$ integer variables n_{ei}, and $2\sum|P_i| + 2|V|$ constraints. If the above set of constraints is solvable, the values of $a(v,P_i)$ for all $v \in V$ and n_{ei} for all $e_i \in E$ are known. We can then find the exact position of each flip-flop one by one as follows. For each edge e_i from u to v, if there are flip-flops on this edge, the first flip-flop on this edge will be placed at a distance of delay $Min(\Phi - a(u,P_i))$ for all paths P_i passing through edge e_i. Other flip-flops can be placed as far from each other as possible, until reaching a previous flip-flop or the end of the path. Though MILP can be solved effectively, it is still not efficient enough to be embedded in floorplanning optimization iterations.

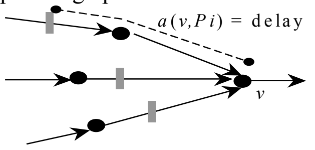


Fig.4 the meaning of $a(v)$

3.2 Graph-based heuristic algorithm

For the paths which compose a connected graph, we want to traverse the graph to decide the optimal insertion of flip-flops such that the weighted sum of cycle numbers of paths is minimized.

3.2.1 Slacks along paths

We find that for a single loop, it is easy to get the optimal insertion of flip-flops by scanning the path from the beginning and then back. The extra cycles are generated when the slacks distributed between flip-flops are larger than the total slack along the path. For each path P, if we ideally insert flip-flops, there will be some slacks distributed along the path and we define the total slack is $Slack(P)$. Suppose that we ideally insert k flip-flops $\{f_1, f_2, \ldots f_k\}$ along the path and the delay between f_i and f_{i+1} is d_i ($1 \le i \le k-1$). If d_i is less than Φ, then there is slack($Slack_i$) between f_i and f_{i+1} such that $Slack_i = \Phi - d_i$. Since we ideally inert flip-flops along the paths, there are no extra cycles generated and $Slack(P) = \sum_{i=1}^{k-1} Slack_i$.

We pick one flip-flop f_i and increase slack between f_i and f_{i+1} by decreasing the delay d_i. If $Slack_i > Slack(P)$ then one extra cycle is needed for the rest of the path to meet the clock period. As shown in Fig.5, the total delay for this path is 1.6Φ *so that* $Slack(P) = 0.4\Phi$. We assume there are flip-flops at the beginning and at the end of the path. Ideally we need at least 1 flip-flop in the middle of the path. Total, we have 3 flip-flops $\{f_1, f_2, f_3\}$. Suppose we divide the path by 1.0:0.6. $Slack_1$=0 and $Slack_2 = 0.4\Phi$. We move f_2 towards f_1 and decrease d_1, $Slack_i$ increases. If $Slack_1 \le 0.4\Phi$, there is no extra cycle generated, but if we move f_2 further ($Slack_i > 0.4\Phi$), then the delay between f_2 to f_3 exceeds the clock period. One extra cycle is necessary to meet the clock period and the total slack along P is changed. Therefore, we can optimize the pipeline design by controlling the extra delay slacks when inserting flip-flop along paths. Before we analyze the complicated graph with loops, we take wires in a combinational circuit to demonstrate the dynamic scanning heuristic. Then we analyze the path-based pipelining by constructing a directed acyclic graph (DAG) G'(V',E') based on the path information.

3.2.2 Dynamic scanning heuristic for combinational circuits

For a combinational circuit, there is no loop. Correspondingly, there is no cycle in the graph. A pair of source node s and target

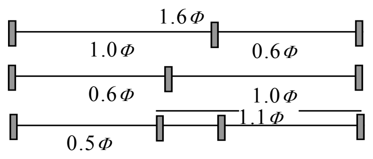


Fig.5 Relations between slacks and extra cycles

node t, which connect to the inputs and outputs respectively, is added to the graph. We assume there is a flip-flop at node s and node t respectively. Therefore, we can scan the graph in topology order and try to insert flip-flop to meet the clock period with the least extra cycles. We analyze the graph based on paths using the arrival time for each passing node along each path. Which is defined as $a(v, P_i)$ in previous section.

At the beginning of pipelining, the number of flip-flops on each edge is set to 0. With the dynamic pipelining, the arrival time for each node along the paths will be updated accordingly. Since we pipeline the scanned edges to meet the target clock period during the traversing, the paths are pipelined partially. We define the delay slacks on the partially pipelined paths.

Definition 1: For a path P_i with k edges$\{e_1, e_2 \ldots e_k\}$ and the first m-1 edges have been pipelined to meet the clock period Φ, the ideal slack for this path is the delay slack if the edges $\{e_m \ldots e_k\}$ are pipelined with no extra slack. Suppose the beginning node for e_m is v.

$$Ideal\_Slack(P_i) = \left(\left\lfloor (a(v,P_i) + \sum_{i=m}^{k} d_{ei})/\Phi \right\rfloor + 1\right) * \Phi - (a(v,P_i) + \sum_{i=m}^{k} d_{ei}) \quad (5)$$

Wherever we insert a flip-flop, it will change the timing distribution in the graph. And if the delay between a newly inserted flip-flop and the previous flip-flop (or source node) is less than one clock period, it will introduce extra slack.

Lemma 1: Suppose we insert a new flip-flop f to an edge e_i which connects two nodes from u to v, and the delay between u to f is d_{uf}. To meet the clock period, each path P_i passing e_i should make the following inequation satisfied:

$$a(u, P_i) + d_{uf} \le \Phi$$

If we insert a new flip-flop on edge e_i, the paths passing edge e_i will be influenced. The required time of the new inserted flip-flop is forced to be Φ. Therefore, the inserted flip-flop will introduce some more slacks

Definition 2: Suppose a new flip-flop f is inserted on edge E, which connects two nodes from u to v, and the delay between u to f is d_{uf}. The extra slack caused by f is

$$Extra\_Slack(P_i, f) = \Phi - (a(u, P_i) + d_{uf}) \quad (6)$$

With the inserted flip-flop and the extra slack, we can find whether the new inserted flip-flop will generate extra cycles and the ideal slacks for the paths should be updated.

Lemma 2: Suppose we insert a new flip-flop f on an edge e_i which connects two nodes from u to v and path P_i is one of the paths which pass edge e_i. The ideal slack of P before f is inserted is $Ideal\_Slack_{cur}(P_i)$, then:

- If $Extra\_Slack(P_i, f) > Ideal\_Slack(P_i)$ then this new inserted flip-flop will generate an extra cycle on path P_i and $Ideal\_Slack(P_i) = \Phi + Ideal\_Slack_{cur}(P_i) - Extra\_Slack(P_i, f)$
- If $Extra\_Slack(P_i, f) \le Ideal\_Slack(P_i)$ then this new inserted flip-flop will not generate an extra cycle on path P_i. $Ideal\_Slack(P_i) = Ideal\_Slack_{cur}(P_i) - Extra\_Slack(f)$

Because of the page limitation, we omit the proof here.

Based on these Lemmas and definitions, we can traverse the graph in topology order and dynamically find where to insert the flip-flops so that the weighted sum of the latencies are minimized.

The optimal positions for flip-flops will be obtained at the same time. For each node, we compute the arrival time for the passing path $a(v,P_i)$ using the longest path approach. If $a(v,P_i) > \Phi$, e_i is the edge along path P_i which connects nodes from u to v. According to lemma 1, we may need to add a flip-flop before node v or before node u. Therefore, we can enumerate the feasible positions for flip-flops and get the number of extra cycles generated by them.

Fig.6 is an example with 2 paths (P1: A-C-D and P2: B-C-E) and the corresponding path graph. The topology order in Fig.6 should be $\{S, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, T\}$. We scan the graph in topology order and figure out the arrival time for each node: $a(S,P1) = a(S,P2) = a(n1,P1) = a(n1,P2) = a(n2,P1) = a(n2,P1) = 0$; $a(n3,P1)=0.5$; $a(n4,P2)=0.7$. When we scan edge e_{35}, $a(n5,P1)= 1.1$, thus we need to add a flip-flop before node $n3$ or after node $n3$ and the inserted flip-flop will influence path $P1$.

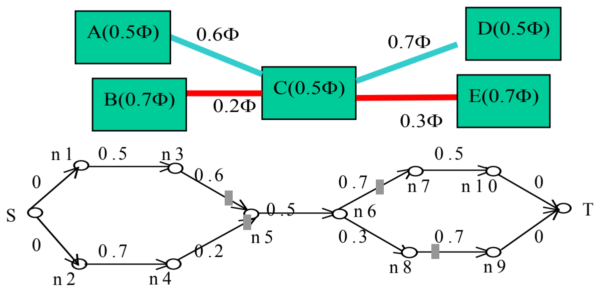


Fig.6 Two paths and the corresponding path graph

Before we insert the flip-flop, we have

$Ideal\_Slack_{cur}(P1) = ((\lfloor 0.5+0.6+0.5+0.7+0.5 \rfloor+1)-0.5+0.6+0.5+0.7+0.5)\Phi = 0.2\Phi$

✓ If we insert a flip-flop immediately before node $n3$ on the edge between $n1$ and $n3$, $d_{1f} = 0.5\Phi$ and $a(n1,P1) = 0$

$Extra\_Slack(P1, f) = \Phi - (a(n1,P1) + d_{uf}) = 0.5\Phi > Ideal\_Slack_{cur}(P1)$

According to lemma 2, there will be ***one extra cycle*** along path $P1$ generated because of this new inserted flip-flop.

✓ If we insert a flip-flop after $n3$ to meet the target clock period, $d_{3f} = \Phi - a(n3,P1) = 0.5\Phi$, and $a(n3,P1) = 0.5\Phi$.

$Extra\_Slack(P1, f) = \Phi - (a(n3,P1) + d_{3f}) = 0 < Ideal\_Slack_{cur}(P1)$

According to lemma 2, there will be ***no extra cycle*** along path $P1$ generated because of this new inserted flip-flop.

Therefore, we decide to insert flip-flop between $n3$ and $n5$ and the delay between $n3$ to f is 0.4Φ.

According to lemma 2, we can update the slack information:

$Ideal\_Slack(P1) = Ideal\_Slack_{cur}(P1) - Extra\_Slack(P1, f) = 0.2\Phi$.

Then we continue with the scanning until node t is met. The corresponding process is listed step by step in Table 1. Finally, we can get an optimal solution with no extra cycles for both paths as shown in Fig.6. Here we can see that in an optimal solution, the flip-flop is not inserted after node $n5$ to utilize the full period in path $P2$, but instead we insert a flip-flop immediately before node $n5$ on edge e_{45}. When we enumerate the positions for the flip-flop along $P2$ near node $n5$, we figure out the extra cycles needed for each possible position. If we insert a flip-flop on edge e_{56}, to meet the clock period, d_{5f} should be 0.1Φ. The extra slack for path $P1$ will be $Extra\_Slack(P1, f) = \Phi - (a(n5,P1) + d_{5f}) = 0.8\Phi > Ideal\_Slack_{cur}(P1)$. There will be ***one extra cycle*** generated. While if we insert flip-flop before $n5$ on edge e_{45}, though it will waste 0.1Φ before node $n5$ along path $P2$, there will be ***No extra cycle***. Therefore, we can find that the optimal position for this flip-flop should be on e_{45} instead of on e_{56}.

The overall algorithm *dynamic_scanning* () is summarized.

```
Algorithm Dynamic_Scanning()
Input: path graph with delay information: $G$
    Target clock period: $\Phi$
Output: The detail positions for flip-flops on each edge so that the target clock period is meet on the path graph
Compute Ideal_Slack for each path;
For each $P_i$: $a(S, P_i) = 0$;
For each node $v$ in $G$ in topology order:
    For each path $P_i$ passing node $v$:
        Compute $a(v, P_i)$;
        If $a(v, P_i) > \Phi$ then
            For each feasible position $f$ for a flip-flop:
                Compute extra_slack($f$, $Pi$);
                If extra_slack($f$, $Pi$)>Ideal_Slack$_{cur}$($Pi$) then
                  $f$ generate one extra cycle on $Pi$;
                Else
                  $f$ generate extra cycle on $Pi$;
                Endif;
            Endfor;
            Find an optimal position of $f$ and update slacks;
        Endif;
    Endfor;
Endfor;
END.
```

From Algorithm *dynamic_scanning*(), we can see that the complexity of this approach mainly depends on three loops. In the worst case, if all the paths pass through the same node and the arrival time exceeds Φ, the worst complexity should be $|Path|^2|V|$, where $|Path|$ is the number of critical paths in the architecture and $|V|$ is the number of nodes in the graph which is about the total number of blocks and wires. In practice, the number of paths is less than the number of nodes in graph G. Even in a simple design with a small quantity of components, the number of nodes in G is more than 10 times the number of critical paths in the architecture.

3.2.3 Near-optimal method for sequential circuits

In sequential circuits, there are several loops and especially in micro-architecture, almost every component is involved in one or more loops as shown in Fig.1. In combinational circuits, both the source and target nodes are treated as flip-flops so that all the edges between them can be pipelined accordingly. Therefore, to avoid the cycles inside the graph, we transform the graph $G(V,E)$ into a *directed acyclic graph* (*DAG*) $G'(V',E')$ by performing a depth-first traversal defining a tree in G. Since there are cycles in G, when we traverse the graph, if a cycle is detected with a node u pointing back to an ancestor v of u, we call the edge e_{uv} from u to v, the back edge. We directly change the direction of the back edge by pointing the edge to the target node t. Therefore, all the cycles will be broken and the total lengths of the paths are remained. Then we can proceed with the pipelining process proposed in the previous section so that we can get a feasible pipelining design, but since we break the cycle into a path from s to t, the information of the cycles will be lost. In some special cases, the optimality will be lost in our approach for sequential circuits because of the lack of the configuration of loops. However, from the analysis and the experimental results, we found the error to be pretty small which is tolerable, especially in a floorplanning design stage.

Table 1. The process of pipelining:
"X" means "does not matter", "-" means "does not change"

v	a (v,P1)	Ideal Slack (P1)	a (v,P2)	Ideal Slack (p2)	Extra slack	f	du f	Extra cycle
S	0	0.2	0	0.6	-	×	×	×
n1	0	-	×	×	-	×	×	×
n2	0	×	-		-	-	-	-
n3	0.5		×	×	-	-	-	-
n4	0.7	×	-		-	-	-	-
n5	~~1.1~~ 0.9	~~0.2~~ 0.2	-	-	0	f1 on e_{35}	0.6	0
n6	0.6	-	~~1.4~~ 0.5	~~0.6~~ 0.5	0.1	f2 on e_{45}	0.2	0
n7	~~1.3~~ 0.3	~~0.2~~ 0.2	×	×	0	f3 on e_{67}	0.4	0
n8	×	×	0.8	-	-	-	-	-
n9	0.8	-	×	×	-	-	-	-
n10	×	×	~~1.5~~ 0.5	0.5	0	f4 on e_{89}	0.2	0
T	-	-	-	-	-	-	-	-

4. Throughput aware floorplanning with pipelining

The objective of the throughput-aware floorplanner is to determine the positions of the blocks such that the performance of the architectural design, in addition to traditional objectives such as area and aspect ratio, is optimized. The performance of a micro architectural design depends on a weighted sum of latencies along the critical paths. Since our graph-based approach can pipeline the critical path efficiently and the detail pipeline results can be fed to a floorplanning engine, we can evaluate the performance accurately and provide the architectural designers with the pipelining information. The path-based pipelining design guides the block design to optimize the performance for the whole design.

The floorplanning problem that we investigate here considers several components in its objective function that are important tradeoffs in micro architectures. Specifically, we consider the die area (footprint), the performance of the micro architecture in *BIPS* and the wirelength. Formally, we define the problem as follows:

Given: (1) target clock period Φ

(2) clocking overhead $T_{overhead}$

(3) list of blocks in the micro architecture with their area, dimensions and total logic delay

(4)set of critical architectural paths with performance sensitivity models for the paths

Objective: Generate a floorplan which optimizes for the die area, wirelength and performance, based on the pipeline design for blocks and wires.

The floorplanner used in this work is based on a simulated annealing framework with CBL representation [14]. We integrate the graph based dynamic approach in floorplanning optimization. During floorplanning, we calculate the total latency based on our approach. Extra latency from the wires is used to compute the new IPC, and hence the performance of the processor for that floorplan. Our cost function uses a weighted combination of area, wirelength and performance, and can be represented by

$$cost = w1 * \frac{1}{BIPS} + w2 * Area + w3 * Wire$$

where BIPS corresponds to the performance of the micro architecture with that floorplan of the blocks; Area is the total area of the floorplan. The performance (BIPS) is calculated based on a pipeline design by a graph-based heuristic approach. The co-efficients of *w1*, *w2* and *w3* are used to control the different weight for each component. In our test evaluation, the performance component is given a high weight and will be optimized when the simulated annealing engine tries to minimize the cost function.

5. Design driver

In this section we present detailed evaluation results obtained for our design driver micro architecture. This architecture is an out-of-order micro-processor with detailed parameters shown in Table 2. We modified SimpleScalar [16] to model this architecture and to parameterize the different critical path latencies found through the floorplanning process. To perform our evaluation, results were collected for SPEC2000 [17] benchmarks. In order to consider the impact of pipelining based on interconnect delays, we use the critical paths in Fig.2 for this study. The area and delay of the blocks were derived based on [18, 19] for a 70*nm* process technology. Based on [20], we assume that the clock cycle overhead is 46*ps*, which corresponds to roughly 1.8FO4 (fan-out-of-four) for 70*nm* technology. The delay of interconnects is derived using the IPEM models [21] which consider several optimizations such as wire sizing, buffer insertion and sizing, etc.

To facilitate the insertion of repeaters, flip-flops etc., which are inevitable to achieve the required interconnect performance, we assume that 10% of each block's area is reserved around the block in the floorplan. Moreover, as the L2 cache occupies more than 50% of our die size, we allow the four L2 cache banks to be placed separately so that the floorplanner has more flexibility in packing the blocks. As we formulate the problem to a MILP, we use a software package GLPK (GNU Linear Programming Kit) to solve the MILP and get the corresponding results. We set the limited running time for the MILP to be in the 200s. Before we integrate our pipelining with floorplanning optimization, we try to demonstrate the efficiency of our graph-based approach. We compare the results with the wire-pipelining results (WP), and the solutions obtained from the MILP solver (MILP), the ideal upper bound used in [6][8](UB) and our graph-based heuristic approach (GH). In wire-pipelining, we assume the maximum delay from the input pin to any flip-flop inside the blocks (T_{su}) and the maximum delay from any flip-flop inside the blocks to the output pin (T_{co}) are equally divided. The delays between flip-flops inside blocks equal to clock period Φ.

Table 2. Baseline processor parameters

Instruction Cache	32KB, 32B/block, 2-way
Decode Width	8
ROB Size	128 entries
Issue Queue	32 entries
Issue Width	8
Register File	70 INT and 70 FP
Functional Units	Units 4 IntALU, 1 FPALU, 2 IntMult, 1 FPMult
Load/Store Queue	32 entries
L1Data Cache	16KB, 32B/block, 4-way, 2RW ports
Unified L2 cache	1MB, 64B/block, 8-way

5.1 Impact of frequencies

To evaluate the accuracy of our approach, we take a fixed packing result and run the different pipeline approaches under the frequency from 2GHz to 7GHz. The corresponding BIPS values are shown in Fig.7. Since with the approach used in [6][8], the flip-flops are assumed to be ideally inserted along each path and the confliction in the blocks which have multiple paths passing is not considered. Therefore, this approach just gives an ideal upper bound of the performance, which is far off from the real designs in most cases. Compared to the optimal results obtained from the MILP solver, the upper bound approach is on average 20% larger. Therefore, this upper bound is not able to give the correct guide for the performance optimization. Fig.7 shows that based on the fixed packing, the system reaches its highest performance under 6GHz, but drops a lot under 7GHz. It is reasonable since the extra latency number for 7GHz increases a lot, which may degrade much more than the increase of the frequency. However, the ideal upper bound does not correlate with this trend, so it is not possible to evaluate the system performance accordingly. While the wire pipelining loses a lot of the flexibility to get a better design, the difference between the optimal design and the wire-pipelining is pretty huge, which is on average 27%. Therefore, the path-based pipelining will give about a 27% performance improvement over wire pipelining.

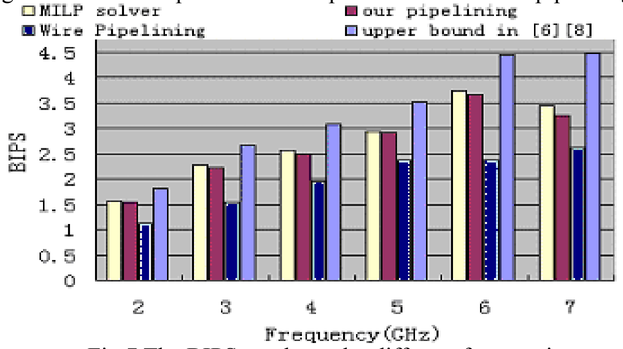


Fig.7 The BIPS results under different frequencies

Our near-optimal approach has an average of 2% error to the results of the MILP solver. Since our approach dynamically scans the path graph, our approach can achieve the solution in almost linear time. The running time for our approach is less than 1 second, while the MILP need about 200s to get a good result. Therefore, our approach is efficient in terms of running time and accuracy.

5.2 Different Packing results

To study the sensitivity of our approach to different packings, we randomly pick 5 packings and run the pipelining approaches under 3GHz. The detailed information is shown in Table 3. For different packing with various packing configuration, the difference between our approach and the MILP solution is about 3%. But the error for upper bound in [6][8] is about 16% and the error for wire-pipelining results is about 24%. Therefore, our approach has the stable performance with different frequencies and various packings.

Table 3 The results for different packings under 3GHz

packing	Packing dimension (mm*mm)	wire	BIPS			
			UB	WP	GH	MILP
1	4.1*7.76	98471	2.677	1.539	2.136	2.262
2	4.24*7.92	106594	2.417	1.731	2.139	2.16
3	4.9*6.35	102578	2.547	1.748	2.091	2.202
4	5.48*5.88	134051	2.654	1.563	2.319	2.346
5	6.99*4.64	135455	2.793	1.731	2.235	2.286
Average error			1.16	0.738	0.97	1

5.3 Integrated with floorplanning optimization

To cope with the pipeline design with floorplaning optimization, though the MILP approach will give an optimal result, it requires a lot of search which takes hundreds of seconds. We can use the MILP approach as a post process at the end of the floorplanning but it is not applicable to be integrated within the floorplanning searching stage. Though the upper bound used in [6][8] does not require too much extra effort, the large range of error may not allow for the correct guidance for optimization. Our near-optimal approach runs in linear time and it gives a pipeline design with about 2% error to the optimal solution. We integrate our approach with the thoughput-driven floorplannning. We also compare our approach to the approach using upper bound [6][8] as the evaluation of pipeline design and using the MILP approach as the post process to get the flip-flop design. Table 4 gives the detailed results. The results show our graph-based approach can result in better performance since it is an accurate evaluation of the pipelining design, which enables the optimization process to be guided correctly and converge well.

Table 4 The floorplanning with pipeline design integrated

Frequency GHz	UB+post_MILP			GH		
	Area (mm^2)	Wire (mm)	BIPS	Area (mm^2)	Wire (mm)	BIPS
2	32.	115.6	1.492	31.8	142	1.714
3	34.6	103.7	2.139	33.3	108.4	2.22
4	32.4	98.7	2.776	36.1	124.3	2.828
5	32.8	126.2	2.885	32.6	94.17	3.35
6	36.0	108.4	3.636	33.7	100.3	3.882
7	35.9	112.5	3.479	36.8	129.9	3.906
Comparison	1	1	1	1.003	1.05	1.091

6. Conclusions and Future Works

In this paper, we propose the optimization methodology of architecture pipelining with physical design. Since we simultaneously optimize the pipeline design and physical packing in terms of system throughput, the performance of the system can be improved a lot over the wire-pipelining. We first formulate the problem as a MILP formulation. Given a packing result of a micro-architecture design, we can optimally figure out the distribution of flip-flops for both blocks and wires. But to solve a MILP is time consuming, so it is not applicable to be embedded in the process of floorplanning iterations. Therefore, we devise a novel *dynamic scanning heuristic* to handle the pipeline design which can feed the floorplan engine with the actual latency informantion. Our algorithm enables the automated design for initial architectural design with the consideration of both the architectural side and the physical design. Our approach is a near-optimal approach which can pipeline all the critical paths by dynamically traversing the path graph. The experimental results show our heuristic gives solutions very close to the MILP results (2% more than MILP results on average) but in a much shorter runtime. Therefore, the dynamic scanning heuristic is stable and effective which is applicable in floorplanning optimization. We are currently working to refine the MILP formulation and attempting to handle it in a much more efficient way, such as a network flow approach.

References

[1] S. Borkar, “Obeying Moore’s law beyond 0.18 micron,” in *Proc. IEEE ASIC/SOC*, pp. 26–31, Sep. 2000.

[2] P. Cocchini, “Concurrent flip-flop and repeater insertion for high performance integrated circuits,” in *Proc. IEEE/ACM ICCAD*, pp. 268–273, Nov. 2002.

[3] S. Hassoun *et al.*, “Optimal buffered routing path constructions for single and multiple clock domain systems,” in *Proc. IEEE/ACMICCAD*, pp. 247–253, Nov. 2002.

[4] V. Nookala and S. S. Sapatnekar, “Correcting the functionality of a wirepipelined circuit,” in *Proc. ACM/IEEE DAC*, pp. 570–575, 2004.

[5] L. Scheffer, “Methodologies and tools for pipelined on-chip interconnect,” in *Proc. IEEE ICCD*, pp. 152–157, Oct. 2002.

[6] A. Jagannathan *et al.*, “Microarchitecture evaluation with floorplanning and interconnect pipelining,” in *Proc. ASPDAC*, pp. 32–35, 2005.

[7] C. Long *et al.*, “Floorplanning optimization with trajectory piecewise-linear model for pipelined interconnects,” in *Proc. ACM/IEEE DAC*, pp. 640–645, Jun. 2004.

[8] J. Cong, G. Reinman, Y. Ma, J. Wei, Y. Zhang, “An automated design flow for 3D microarchitecture evaluation,” in *Proc. ASPDAC* , 2006.

[9] C.E. Leiserson and J.B. Saxe, “Retiming synchronous circuitry,” *Algorithmica*, 6:5–35, 1991.

[10] M. Ekpanyapong *et al.*, “Profile-guided micro architectural floorplanning for deep submicron processor design,” in *Proc. ACM/IEEE DAC*, pp. 634–639, Jun. 2004.

[11] V. Nookala, Y. Chen, D. J. Lilja, S. S. Sapatnekar, “Micro architecture-aware floorplanning using a statistical design of experiments approach,” In *Proc. ACM/IEEE DAC*, 2005.

[12] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multi-cycle communication," *in Proc. IEEE Transactions on Computer-Aided Design of Integrate d Circuits and Systems*, pp.550 - 564, April 2004.

[13] R. McInerney, K. Leeper, T. Hill, H. Chan, B. Basaran and L. McQuiddy, “Methodology for repeater insertion management in the RTL, floorplan and fullchip timing databases of the ItaniumTM microprocessor,” In *Proc. International Symposium on Physical Design*, pages 99–104, 2000.

[14] X. Hong, G. Huang, Y. Cai, et al, "Corner block list: an efficient and effective topological representation of non-slicing floorplan", in *Proc. of International Conference on Computer Aided Design*, pp 8-12, 2000.

[15] E. Sprangle and D. Carmean, “Increasing processor performance by implementing deeper pipelines,” In *Proc. 29th Annual International Symposium on Computer Architecture (ISCA ’02)*, pp. 25–34, 2002.

[16] D. C. Burger et al., “The simplescalar tool set, Version 2.0,” *Technical Report CS-TR-97-1342*, University of Wisconsin, Madison, 1997.

[17] The Standard Performance Evaluation Corporation, 2000. http://www.spec.org.

[18] S. Palacharla, N. Jouppi and J. E. Smith, “Complexity effective superscalar processors,” In *Proc. International Symposium on Computer Architecture*, pp. 206–218, Jun. 1997.

[19] S. Palacharla, N. Jouppi, and J. E. Smith, “Complexity effective superscalar processors,” In *Proc. International Symposium on Computer Architecture*, pp. 206–218, Jun. 1997.

[20] M. S. Hrishikesh, K. Farkas, N. P. Jouppi, D. C. Burger, S. W. Keckler, and P. Sivakumar, “The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays,” In *Proc. of 29th International Symposium on Computer Architecture*, May 2002.

[21] J. Cong and D. Z. Pan, “Interconnect estimation and planning for deep submicron designs,” In *Proc. 36th ACM/IEEE Conference on Design Automation*, pp. 507–510, 1999.

[22] P. Pan, A. K. Karandikar, and C. L. Liu, “Optimal clock period clustering for sequential circuits with retiming,” In *Proc. IEEE TCAD*, Vol. 17 No. 6, pp. 489–498, 1998.

[23]C. Lin, H. Zhou, “Wire retiming as fixpoint computation,” In *Proc. IEEE Transactions on Computer-Aided Design of Integrate d Circuits and Systems*, Vol. 13, No. 12, 2005

[24] J. Cong, A. Jagannathan, G. Reinman, and M. Romesis "Microarchitecture Evaluation with Physical Planning ", Proc. of the Design Automation Conference, Anaheim, pp. 32 - 36, June 2003