Microarchitecture Evaluation With Physical Planning

Jason Cong, Ashok Jagannathan, Glenn Reinman, Michail Romesis

{cong,ashokj,reinman,romesis}@cs.ucla.edu

Computer Science Department,

University of California, Los Angeles, Technical Report 030022

1 Abstract

Conventional microprocessor designs are guided mainly by the maximum throughput (measured as IPC), but fail to evaluate the impact of microarchitectural decisions on the physical design, and in particular, the impact on the interconnects. In this paper, we propose MEVA, a system to consider both IPC and cycle time in the design space search for a given microarchitectural design. MEVA can consider a variety of user-specified architectural alternatives that trade IPC and cycle time in the design, and performs accurate floorplanning and simulation to fully evaluate each alternative. The resulting solution will maximize the benefit from both IPC and cycle time to provide a better solution than a design space exploration based simply on IPC or cycle time alone. For a sample architectural design, we are able search a space of 32 architectural configurations with physical planning in less than 2 hours to find a processor configuration that, in terms of BIPS, outperforms the configuration with the best IPC performance by 14% and the configuration with the fastest clock by 27%. This initial exploration only considers the boundary cases of the large design space, but still features substantial IPC and cycle time variation.

2 Introduction

There are a number of hardware challenges that future architects will need to face in the design of the next generation of microprocessors. One of the primary means of increasing the speed of microprocessors has been to scale the feature size of the current technology. As feature sizes continue to shrink, it has become evident that interconnect delay is not scaling at the same rate as transistor delays. There have been a significant amount of analytical and empirical analyses of the interconnect scaling bottleneck in the process technology literature. Recently, these analyses have carried over into the computer architecture literature [1, 2].

Architects have been adding a variety of components to the chip to improve processor performance. These components have caused the chip area to increase with successive technology generations [1], complicating layout and routing. As the clock speed of the processor continues to increase, correspondingly dropping the cycle time of the processor, this problem becomes even more severe, as the processor may be unable to communicate across the chip in a single cycle [1, 3]. Agarwal et al. [1] predict that current processor designs will improve by at best, 12.5% per year in terms of performance over the next fourteen years due to hardware scaling concerns.

Prior work [1, 2, 4] has demonstrated the need to consider both cycle time and throughput (IPC) when measuring overall processor performance. However, architects often have little or incomplete physical design information about the architectural space they are considering. Accurate area and delay information for a given logic block can be difficult to derive without an actual implementation of the architecture. Moreover, without accurate physical planning, designers cannot measure the considerable impact interconnect can have on a given architecture.

In general, the execution time for a given program is defined as

$$T_{exec} = num\_instructions * CPI * cycle\_time$$

where *num_instructions* is the dynamic instruction count of the program, *CPI* or cycles/instruction is the average number of cycles required to execute a given instruction, and *cycle_time* is the inverse of the processor frequency (measured in seconds). Much research has been dedicated to exploring the interaction between compilers and architectures to better improve both the CPI and dynamic instruction count of a given application space. Hardware/software co-design has enabled architects to more intelligently focus silicon area on processor bottlenecks. However, it is equally important to consider the interaction between architectural design and physical design (the latter two variables in the execution time equation). While the throughput or IPC[1] of a processor is determined by the architectural design, the cycle time (clock rate) of the processor is determined by the physical design (layout and routing). This requires designers to carefully explore both spaces together, making architectural decisions that maximize benefits to both throughput and cycle time.

The design space explored by architects during processor design can grow quite large when considering different algorithms (i.e. different branch predictors), component sizes and characteristics (i.e. cache size or associativity), or pipeline depth (i.e. cache access latency). To better manage and explore this space in the face of future hardware scaling challenges, we propose a Microarchitectural EVAluation (MEVA) system to provide a set of flexible and customizable tools to explore an architectural design space with both accurate physical planning information and cycle-accurate architectural simulation. Through this joint exploration of physical and architectural design, architects can better study what designs are able to tolerate poor wire scaling and still achieve high performance.

The rest of the paper is organized as follows: In Section 3, we present an overview of the MEVA system. Section 4 discusses the physical planning engine, while Section 5 and Section 6 deal with the architecture simulation methodology and the design driver used to evaluate our system. Experimental results are presented in Section 7 and the paper is concluded with some suggestions for future work in Section 8.

[1]Architects often use IPC or instructions/cycle than CPI to measure the effectiveness of a processor architecture.

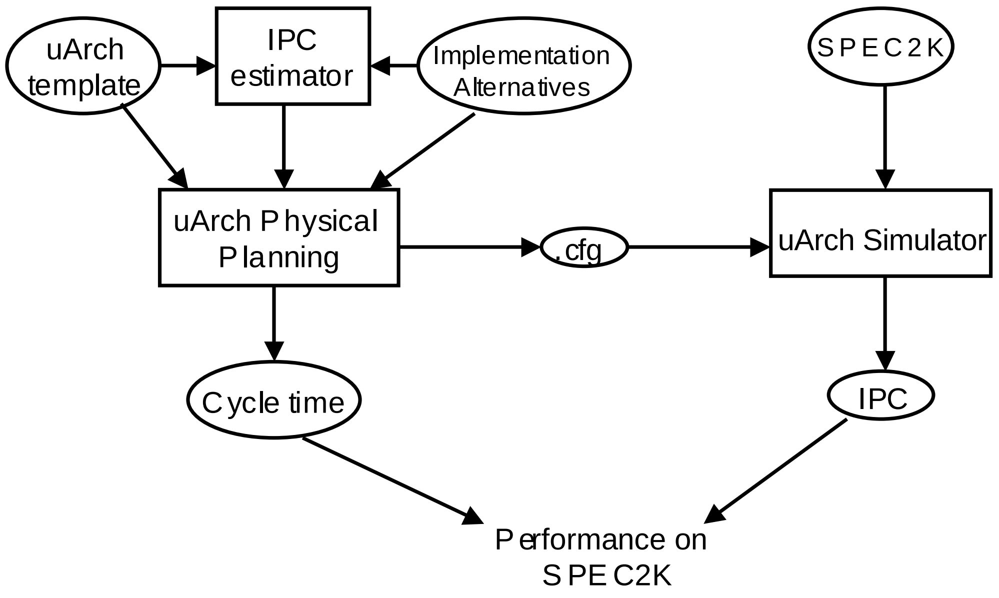

Figure 1: Overview of the MEVA system

3 Overview of MEVA

Figure 1 shows the overall picture of the MEVA system. In the following section, we briefly discuss the inputs and outputs of this system and the various components which build the entire system.

Inputs to MEVA

The MEVA system takes two inputs – (a) an *architectural template*, which is essentially a block-level netlist that captures connectivity of the major functional blocks and (b) a *library* of architectural alternatives for the different blocks in the template. Each block in the template can be implemented in a variety of ways, and architects usually have a set of alternatives they would like to explore for each block. Together, the architectural template and the library of alternative block implementations capture a class of microarchitectures where the underlying connectivity is fixed, while the individual block properties such as area, timing, latency can vary significantly. Such alternative architectures for the blocks affect the IPC through varying latency properties, and also affect the cycle time as they have different area/timing characteristics. For example, one would like to explore different sizes such as 8K, 32K or 64K for the instruction and data caches in the template. Decision as to which cache size is appropriate can only be made after a careful understanding of their impact on both the architectural and physical design spaces. It is exactly this combined design space that the MEVA system is trying to explore. It is important to note that it is impossible to represent all classes of microarchitectures using a single template. Thus, one can come up with a variety of templates and corresponding library files to model any architectural space, and use the MEVA system to evaluate each class of microarchitectures with physical planning.

Since the choice of any architectural alternative on the physical design space is measured by its impact on the achievable cycle-time, the area and timing properties of these alternatives should be modeled in sufficient detail. Figure 3 shows how the timing properties of any block alternative is modeled in MEVA. Each alternative

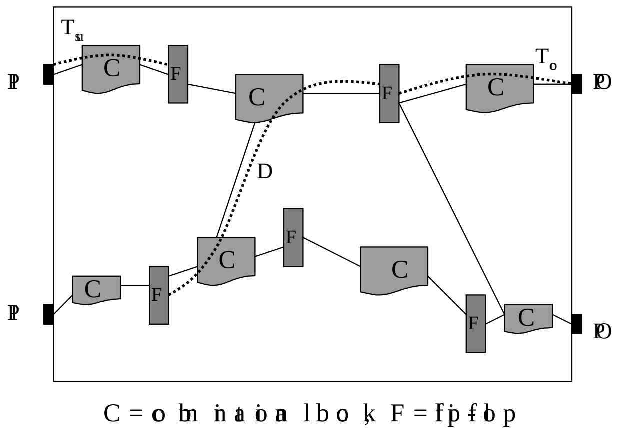

Figure 2: Timing model used in MEVA for each alternative block implementation.

is characterized with the following information:

(i) **Area** of the block.

(ii) **Longest delay** D of any combinational path inside the block as shown in Figure 3.

(iii) **Input-to-clock time (T_{su})**: For each input pin on the block, this value specifies the maximum delay from this pin to any flip-flop inside this block.

(iv) **Clock-to-output time (T_{co})**: For each output pin on the block, this value specifies the maximum delay to this pin from the output of any flip-flop inside this block.

The T_{su} and T_{co} values are very important in deciding the freedom available for the interconnects that are connecting this block to the rest of the design. We believe that the above mentioned information is good enough for the purpose of careful interconnect planning.

In MEVA, we use structural Verilog to represent a given architectural template and a Synopsys .lib like format for our library of architecture alternatives.

Components of MEVA

The MEVA system consists of three main components which are (a) physical planning engine, (b) IPC estimator and (c) cycle-accurate microarchitectural simulator.

The *physical planning engine* takes as input the architectural template and the library of architecture alternatives and performs a floorplanning of the design with interconnect planning to optimize a given cost function, which can include a combination of objectives from the architectural and physical design spaces such as IPC, cycle time, power etc. During this process, the planning engine also chooses different alternative implementations for each block from the library to achieve the best set of block implementations for the given objective. The planning engine also considers various layout related issues such as pin-assignment for the blocks, routability of the floorplan etc., when it attempts to find the best microarchitecture in terms of the given constraints. This planning engine is presented in Section 4.

The goal of the *IPC estimator* is to provide the planning engine with a quick and accurate IPC estimation for any configuration of block alternatives chosen by the planning engine at any point of time. Such estimates can be obtained either by a combination of statistical and analytical methods or a table-lookup method, if the number of different configurations is not too high.

Once the planning engine determines the best configuration for each block based on the IPC estimates and the results of the floorplan, this result can be fed to a microarchitecture simulator. The *simulator* performs a cycle-accurate simulation of the underlying architectural model for a given set of benchmark programs, and provides the IPC information characterizing the microarchitecture.

Outputs from MEVA

The output from MEVA includes the selected architectural alternative for each block in the template along with the best possible cycle time information that it can derive using its physical planning engine, subject to the cost function specified to the planning engine. The latency of the architectural alternative for each block can then be fed to a cycle-accurate architecture simulator for that template to generate an accurate IPC value. The cycle time information from the physical planning engine and the accurate IPC estimation from the simulator can then be used to get a good estimate of the performance of the microarchitecture on the given set of benchmarks.

In the following sections, we describe our implementation of the MEVA system and a base architecture template that we have developed to study the MEVA system.

4 Physical Planning Engine

The physical planning engine in MEVA has been developed to address the following goals:

- Optimize the performance of the system measured as the number of instructions executed per second – i.e., IPC/cycle_time, subject to other physical design constraints such as area.
- Consider different architectural alternatives for the blocks when searching for the architecture with the best performance.
- Consider interconnect planning during the floorplanning stage.

The inputs to the engine are (a) an architectural template, (b) a library file that contains area and timing information for different architecture alternatives of each block and (c) a list of allowed architectural combinations along with the IPC for that configuration. The output of the engine is a layout of the blocks with a selection of an architecture combination such that the performance of the system is maximized under given area constraints. Below, we explain how we address each of the three main goals:

Objective function: The objective function for our planning engine is as follows:

$$\frac{\sum_{i=1}^{n} w(i)wl(i)}{IPC(c)}$$

where $w(i)$ is the weight of a net i, $wl(i)$ is the wirelength of i, and $IPC(c)$ is the IPC of the current configuration c. The weights for the nets are computed according to the slacks of their pins. We use a traditional simulated annealing engine and at every temperature we perform static timing analysis. The delays for every pin-to-pin connection are computed by the IPEM estimator [5] for $100nm$ technology which considers optimal buffer insertion, sizing and wire sizing. Using static timing analysis, we can estimate the cycle time for the current layout and the timing slack for each pin. For a net n , suppose that the slack on its source pin is s picoseconds and if the current cycle time is c picoseconds, the weight of the net n is computed as $(1 - s/c)^a$. The exponential factor a is initialized to 1, and is gradually increased as we move to lower temperatures, so that timing optimization is concentrated on highly critical nets at lower temperatures.

Alternative Block Selection: We introduce an alternative block configuration selection move in the simulated annealing engine. When a new configuration is selected, the dimensions and the timing characteristics of one or many blocks change, as well as the IPC of the configuration in the architectural space. This is usually a significant change to the current solution we have. In order to evaluate the effect of such moves, we first perform a small number of low-temperature traditional floorplanning moves on the new configuration and we then decide whether to accept or reject this configuration.

Interconnect Planning: As mentioned earlier, we use the interconnect performance estimator IPEM [5] for estimating the wire delays under optimal buffer insertion, sizing and wire sizing. Currently, we adopt a simple pin assignment strategy. Ignoring pin spacing constraints, all pins are assigned initial locations along the boundary of the block depending on the locations of the other blocks which connect to these pins. The pins are then spread out until they satisfy the width and spacing requirements. For the immediate future, we plan to integrate pin assignment with global routing according to the algorithm described in [6]. For the global router, a good candidate is the L-Z router presented in [7], as it is very fast and can provide congestion information.

We have implemented a version of our physical planning engine extending the floorplanner PARQUET [8] to support timing optimization, alternative block selection and interconnect planning. Table 1 shows the quality of our physical planning engine compared to existing state-of-the-art floorplanners [9][10] on a set of MCNC benchmarks. It can be seen that our physical planning engine produces competitive results in terms of area and wirelength.

Circuit	Sim-Tempering		TCG		MEVA	
	Area	WL	Area	WL	Area	WL
ami33	1.29	48.6	1.24	50.3	1.28	47.7
ami49	39.24	715.8	38.20	663.1	38.47	646.8
hp	9.58	114.9	9.49	151.8	9.96	101.2
xerox	20.50	417.4	20.42	385.0	21.02	384.3
apte	48.50	223.8	48.48	378.0	49.58	276.6
Average	0.99	1.03	0.97	1.19	1	1

Table 1: Comparison of MEVA with existing floorplanners on MCNC benchmarks.

5 Simulation Methodology

The simulator used in this study was derived from the SimpleScalar/Alpha 3.0 tool set [11], a suite of functional and timing simulation tools for the Alpha AXP ISA. The timing simulator executes only user-level instructions, performing a detailed timing simulation of a dynamically scheduled microprocessor. Simulation is execution-driven, including execution down any speculative path until the detection of a fault, TLB miss, or branch misprediction. We have made extensive modifications to the simulator to handle the architecture proposed in Section 6, including creation of a finite scheduling window (and realistic release from the issue queue), memory dependence prediction, duplicated register file support for clustered functional units, and a decoupled front-end architecture [4]. The simulator is highly parameterizable, and a single configuration file can specify the issue width, predictor sizes, cache configuration, pipeline depth, and more, of a given processor.

To perform our evaluation, results were collected for 20 randomly selected SPEC2000 benchmarks. The programs were compiled on a DEC Alpha AXP-21164 processor using the DEC C and C++ compilers under an OSF/1 V4.0 operating system using full compiler optimization (`-O4 -ifo`). Each program has been simulated using the *ref* set for that application for 200 million instructions after fast forwarding past the initialization portion of the benchmark (as described in [12]).

6 Design Driver

We use the architectural template shown in Figure 3 to study the effectiveness of the MEVA system. It represents a 4-way out-of-order superscalar processor and the architecture has been carefully chosen to allow us to study many of the interesting architectural alternatives available for a microprocessor designer, and to understand the impact of such architectural decisions in the physical domain.

The template features a decoupled front-end [4], which provides latency tolerance between the branch prediction architecture and the instruction fetch unit, and provides a glimpse of the future fetch stream of the processor. Our branch prediction architecture includes a basic block target buffer [13] (BBTB) to provide accurate, high-bandwidth instruction supply – crucial for exploiting instruction level parallelism in future microprocessors. The design has a minimum branch prediction penalty of 11 cycles (based on our minimum latency estimates for each block along the pipeline). We have two levels of on-chip caches: non-blocking instruction and data caches, and a unified L2 cache. The architecture features two renaming units and two dynamic instruction schedulers, one each for integer and floating point instructions. As in a deeply pipelined processor like the Pentium 4 [14], our design features a large scheduling window due to scheduler, dispatch, and register file read latency. Our scheduler makes use of memory dependence prediction, and we verify these predictions through an in-order load/store structure. To recover from memory dependence mispredictions, we use the same squash recovery hardware as is used by the branch predictor for control path mis-speculations. Our execution core uses a duplicated register file architecture with two distinct clusters of functional units in the integer pipeline.

In this paper, we focus on scaling the sizes of the following structures in our architectural exploration:

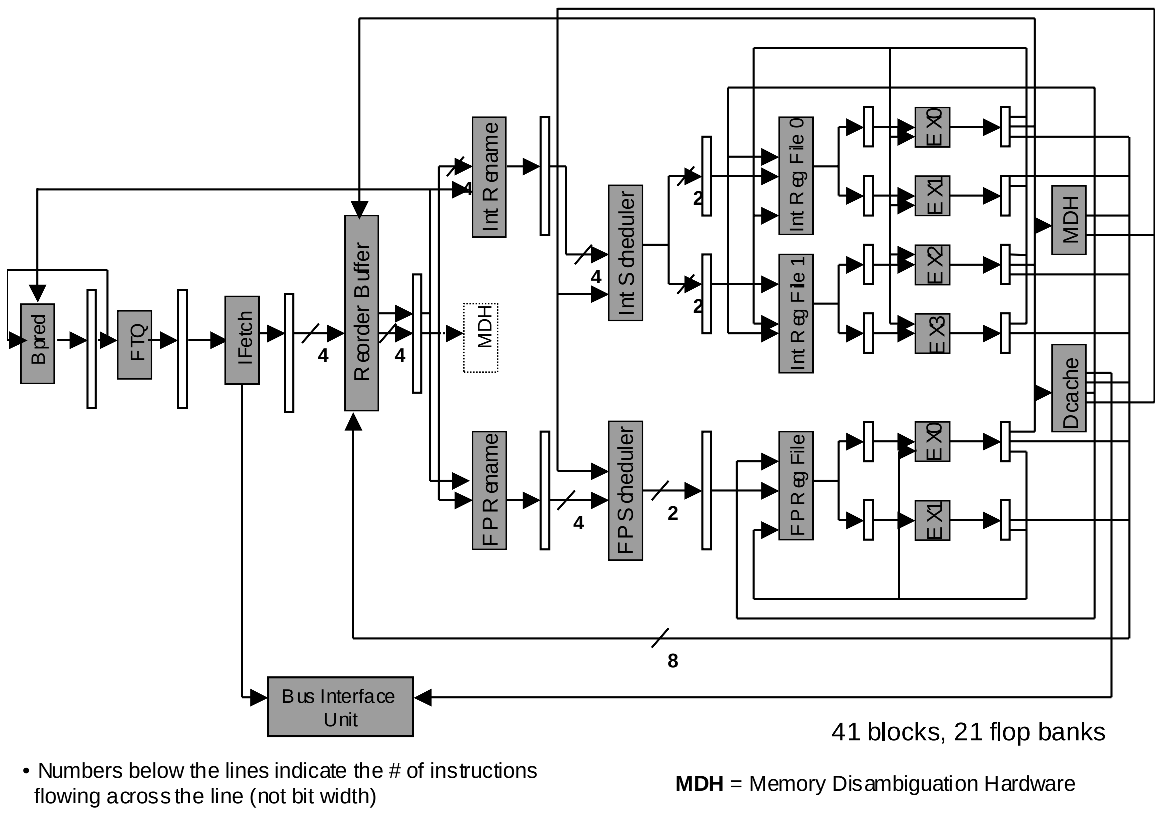

Figure 3: A sample microarchitectural template

BBTB - the BBTB holds the basic block address predictions for the branch prediction architecture. As this structure grows larger, more basic blocks will be able to be tracked simultaneously by the branch prediction architecture. For applications with large instruction footprints or with frequent branches, scaling the BBTB can provide a large benefit.

Instruction Window - the instruction window consists of the register file, the reorder buffer (ROB), and the load/store queue (LSQ) that is used to detect memory aliasing. It makes sense to scale these structures together, as most entries in the ROB will require storage space in the register file and/or LSQ. Scaling the instruction window can provide benefit for applications that are plagued by long latency instructions (such as loads that miss in the data cache) and have sufficient instruction level parallelism to continue executing instructions in the face of these long latency instructions.

Instruction Cache - the instruction cache stores the actual instructions to be executed by the processor (specified by the basic block address predictions of the BBTB). If there is a miss in the cache, the front-end must stall while the L2 is probed for the desired cache block. Thus, this structure can significantly impact fetch bandwidth, and applications with large instruction footprints may see a benefit from increasing the size of the instruction cache.

Data Cache - the data cache stores recently accessed blocks of data memory. If there is a miss in the data cache, the instructions dependent on the load that caused the miss will have to stall, potentially filling up the schedule-to-execute window and instruction window if there is not enough instruction level parallelism. Larger data caches can help programs with large data footprints.

For this paper, we assume that the instruction cache is always a 2-way set associative cache and that the data cache is always a 4-way set associative cache. The unified L2 cache is a 256KB 4-way set associative cache. Our `gshare` branch predictor has 16K entries, and the FTQ has 32 entries.

7 Experimental Results

In this section, we report evaluation results on the architecture design driver described in Section 6 using a preliminary implementation of the MEVA system to show the viability and impact of our proposed approach of combining architecture exploration with physical planning.

As mentioned in Section 6, we have chosen to vary the size and latency of the BBTB, Instruction Window and the instruction and data caches primarily due to the fact that variation in these block characteristics will result in significant change in the IPC of the architecture, leading to interesting design tradeoffs. As a secondary reason, these blocks have been chosen also because the area and delay of these blocks can be modelled reasonably well using CACTI [15], which is crucial for our physical planning. All our area/delay estimates using CACTI are based on the *100nm* technology parameters built inside the tool. Based on the delay values, we derive a minimum and maximum latency value for each of the blocks based on the cycle-time of existing state-of-the-art microprocessors. We assume that these blocks can be pipelined using a given number of flip-flops, such that the delay of each stage inside the block is the same. It is important to note that the minimum and maximum latency options will imply a different longest combinational delays and T_{su}/T_{co} requirements, which will significantly affect the freedom during physical planning. We generate 32 architectural combinations, 16 for the minimum latency values and 16 for the maximum latency values, as shown in Table 2.

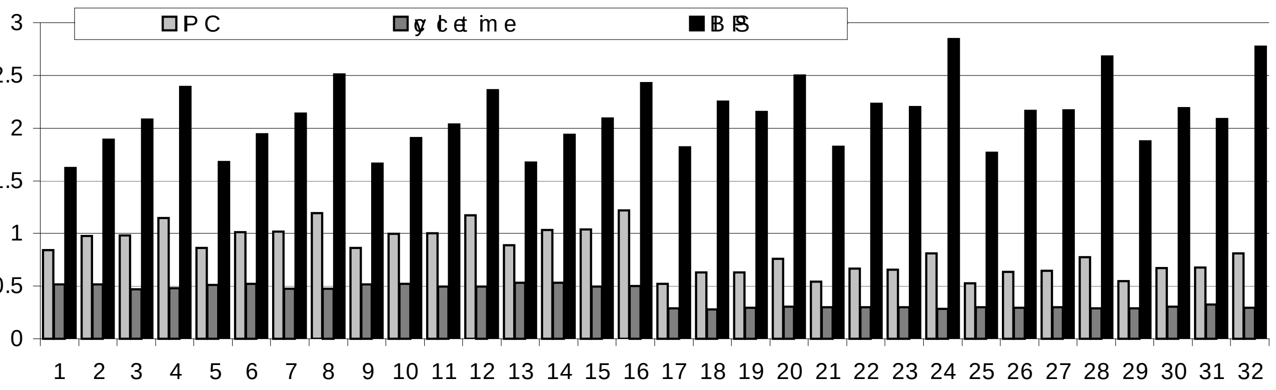

Figure 4: IPC, cycle time and BIPS results for the 32 configurations.

Figure 4 presents three data points for each of the 32 configurations listed in Table 2 – (a) the average IPC for each architectural configuration obtained using our cycle-accurate simulator on a set of 20 SPEC2000 benchmarks, (b) the cycle time of the best floorplan obtained using our physical planning engine and (c) the corresponding BIPS rating. It can be seen from Figure 4 that configuration #16 has the best IPC and configuration #18 has the best cycle time. However, configuration #24 has the best performance in terms of BIPS, underlining the fact that it is important to look at both the architectural and physical design spaces together to draw conclusions on the overall performance of any microarchitecture. Also, despite the relatively lower IPC of the maximum latency configurations (17-32), we see a dramatic increase in BIPS for these configurations due to the cycle time we are able to achieve with the deeper pipelining. The extra data cache latencies and larger branch misprediction depth are particularly hard to tolerate, as shown in [2], and have a large impact on IPC for these configurations. This again, demonstrates the importance of examining both IPC and cycle time when exploring a design space. Ultimately, these results emphasize the importance of taking interconnect effects into account when exploring an architectural design space. The latencies in the architectural alternatives impact the IPC of the configuration, and can be tuned for a desired cycle time, but the true cycle time is only known once the floorplanning is complete. This can have surprising results, as in the case of configurations #2 and #3. Despite a comparable IPC, the wirelength impact of the larger branch prediction unit has a significant impact on BIPS (nearly a 10% reduction in BIPS from configuration #3 to #2).

It should be noted that these results only represent the performance of the boundary cases (corresponding to the minimum and maximum latencies of the blocks) in our driver architecture design space. A configuration with latency between the boundary configurations should provide the most beneficial tradeoff between IPC and cycle time, as the minimum latency alternatives do not provide an aggressive enough clock speed and the maximum latency alternatives are too heavily pipelined (impacting IPC) relative to the impact of wirelength.

Finally, to validate the alternative architecture selection part of our physical planning engine, we tried to search the combined solution space using the method explained in Section 4. Since we did not have a fast and accurate IPC estimation technique that can provide quick IPC for architectural configurations during physical planning, we use a cycle-accurate simulator to obtain IPC for all the 32 configurations that our physical planning engine will explore and use a table-lookup method for obtaining IPC during architectural exploration. Since the tool is searching a very large solution space, we measure the effectiveness of the tool by the quality/runtime tradeoff – i.e., what is the quality of the best configuration identified by the tool, given a fixed amount of computing time? Over several runs, we observed that the tool finds the best configuration (#24 in our case) in about 1/4th the total running time required to generate the best possible floorplan for each of the 32 individual configurations. This, we believe, is a significant savings in time and makes this method suitable for searching even larger solution spaces.

8 Conclusion and Future Work

We have presented a microarchitectural evaluation system named MEVA that attempts to combine the architectural design space with the physical design space. MEVA provides early feedback to architects about the impact of architectural changes on physical design. A representative architectural template and corresponding block alternatives were presented which has a significant variation in terms of architectural and physical design objectives. Both IPC values and cycle times for these different architecture configurations were shown, along with the BIPS value, to demonstrate the need for a combined consideration of physical planning during microarchitecture evaluation. We are able to find a solution with the best BIPS rating, which is over 14% better than best IPC solution and over 27% better than solution with the fastest clock. This performance improvement is just based on the set of boundary alternatives in our initial exploration and further improvement may be expected. Finally, runtime vs. quality tradeoff of the MEVA system in exploring the combined solution space was presented. Experimental results show the viability of this approach and the need for better tools in to evaluate microarchitectures with careful physical planning. Future work will also involve a more efficient search of a much larger design space, beyond the boundary alternatives and more efficient ways to quickly estimate the IPC of microarchitectures.

References

[1] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger, "Clock rate versus ipc: The end of the road for conventional microarchitectures," in *27th Annual International Symposium on Computer Architecture*, 2000.

[2] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," in *29th Annual International Symposium on Computer Architecture*, 2002.

[3] J. Cong, "An interconnect-centric design flow for nanometer technologies," in *Proceedings of IEEE*, pp. 505–527, April 2001.

[4] G. Reinman, T. Austin, and B. Calder, "A scalable front-end architecture for fast instruction delivery," in *26th Annual International Symposium on Computer Architecture*, May 1999.

[5] J. Cong, and D. Pan, "Interconnect estimation and planning for deep submicron designs," in *Proc. Design Automation Conference*, pp. 507–510, 1999.

[6] J. Cong, "Pin assignment with global routing for general cell design," *IEEE Trans. on Computer Aided Design*, vol. 10, pp. 1401–1412, 1991.

[7] C. Chang, J. Cong, D. Pan, and X. Yuan, "Physical hierarchy generation with routing congestion control," in *Inter. Symposium on PHysical Design*, pp. 36–41, 2002.

[8] S. Adya, and I. Markov, "Fixed-outline floorplanning through better local search," in *Proc. International Conference on Computer Design*, pp. 328–334, 2001.

[9] J. Cong, T. Kong, D. Xu, F. Liang, J. Liu, and W. Wong, "Relaxed simulated tempering for vlsi floorplan design," in *Proc. Asia and South Pacific Design Automation Conference*, pp. 13–16, 1999.

[10] J. Lin, and Y. Chang, "Tcg: A transitive closure graph-based representation for non-slicing floorplans," in *Proc. Design Automation Conference*, pp. 764–769, 2001.

[11] D. C. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[12] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.

[13] T. Yeh and Y. Patt, "A comprehensive instruction fetch mechanism for a processor supporting speculative execution," in *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pp. 129–139, Dec. 1992.

[14] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The microarchitecture of the pentium 4 processor," *Intel Technology Journal Q1*, 2001.

[15] S. Wilton and N. Jouppi, "Cacti: An enhanced cache access and cycle time model," in *IEEE Journal of Solid-State Circuits*, May 1996.

BPRED	ROB	RF	LSQ	I$	D$
32/1	128/1	128/1	72/1	8K/2	8K/2
512/1	128/1	128/1	72/1	8K/2	8K/2
32/1	512/1	512/2	288/1	8K/2	8K/2
512/1	512/1	512/2	288/1	8K/2	8K/2
32/1	128/1	128/1	72/1	32K/2	8K/2
512/1	128/1	128/1	72/1	32K/2	8K/2
32/1	512/1	512/2	288/1	32K/2	8K/2
512/1	512/1	512/2	288/1	32K/2	8K/2
32/1	128/1	128/1	72/1	8K/2	32K/2
512/1	128/1	128/1	72/1	8K/2	32K/2
32/1	512/1	512/2	288/1	8K/2	32K/2
512/1	512/1	512/2	288/1	8K/2	32K/2
32/1	128/1	128/1	72/1	32K/2	32K/2
512/1	128/1	128/1	72/1	32K/2	32K/2
32/1	512/1	512/2	288/1	32K/2	32K/2
512/1	512/1	512/2	288/1	32K/2	32K/2
32/2	128/1	128/3	72/2	8K/4	8K/4
512/2	128/1	128/3	72/2	8K/4	8K/4
32/2	512/1	512/4	288/3	8K/4	8K/4
512/2	512/1	512/4	288/3	8K/4	8K/4
32/2	128/1	128/3	72/2	32K/5	8K/4
512/2	128/1	128/3	72/2	32K/5	8K/4
32/2	512/1	512/4	288/3	32K/5	8K/4
512/2	512/1	512/4	288/3	32K/5	8K/4
32/2	128/1	128/3	72/2	8K/4	32K/5
512/2	128/1	128/3	72/2	8K/4	32K/5
32/2	512/1	512/4	288/3	8K/4	32K/5
512/2	512/1	512/4	288/3	8K/4	32K/5
32/2	128/1	128/3	72/2	32K/5	32K/5
512/2	128/1	128/3	72/2	32K/5	32K/5
32/2	512/1	512/4	288/3	32K/5	32K/5
512/2	512/1	512/4	288/3	32K/5	32K/5

Table 2: List of 32 different architecture configurations explored. Each entry corresponds to the size/latency of that block.