Comparing Simulation Techniques for Microarchitecture-Aware Floorplanning

Vidyasagar Nookala*, Ying Chen†, David J. Lilja* and Sachin S. Sapatnekar*

*ECE Department
University of Minnesota, Minneapolis
Email: {vidya,lilja,sachin}@ece.umn.edu

†School of Engineering
San Francisco State University, San Francisco
Email: yingchen@sfsu.edu

***Abstract*— Due to the long simulation times of the `reference` input sets, microarchitects resort to alternative techniques to speed up cycle-accurate simulations. However, the reduction in the runtimes comes with an associated loss of accuracy in replicating the characteristics of the reference sets. In addition, the effect of these inaccuracies on the overall performance can vary across different microarchitecture optimizations or enhancements. In this work, we study and compare two such techniques, reduced input sets and statistical sampling, in the context of microarchitecture-aware floorplanning, a physical design stage, where the objective is to find an IPC-optimal global placement of the blocks of a microprocessor. The variation in the IPC results due the insertion of additional flip-flops on some across-chip wires of the processor that have multicycle delays in nanometer technology nodes. The objective of IPC-aware floorplanning is to minimize the amount of pipelining required by the system buses that are critical in determining the system performance. Our results indicate that, although the two techniques exhibit contrasting behavior in quantifying the criticality of bus latencies, the ensuing floorplanning optimization process results in almost identical performance improvements for both reduced input sets and sampling. The reason behind this is that, for discrete optimization problems such as IPC-aware floorplanning, a reasonably accurate relative ordering of performance bottlenecks is sufficient, absolute accuracy is not necessary.**

I. Introduction

With the operating frequencies of high-performance microprocessors doubling every process generation [1], the distance traveled by a signal along a wire in a single clock period has been gradually decreasing, and in nanometer technologies, the delays of some of the global interconnects, even after aggressive optimization, exceed the system clock cycle. In such a scenario, a popular approach to ensure that the system operates at the desired frequency has been to pipeline those global wires whose delays exceed the clock period, i.e., the delay of each of those global wires is distributed over several clock cycles by inserting memory elements such as edge-triggered flip-flops [2], [3]. It can be noted that this approach is tantamount to inserting additional, dummy, stages in the pipeline of the processor.

However, as observed in [4], such a deeper pipelining strategy may not necessarily result in overall performance improvement. As the number of stages in a microprocessor pipeline increases, the average number of instructions executed per clock cycle, IPC, decreases. Specifically, some events such as a branch misprediction consume more clock cycles in a deeper pipeline, thereby decreasing the IPC. This is an important concern, since the decrease in the IPC may more than offset any gain achieved by the reduction in the clock period. In addition, the IPC penalty caused by pipelining the buses of a microprocessor depends on the locations at which the extra flip-flops are added: additional flip-flops on some buses of the microprocessor can impact the IPC more than on others. In particular, the number of flip-flops that must be inserted on a bus is proportional to the length of the bus, which in turn depends on the locations of the connecting functional units (end points) of the bus in the layout. These lengths are determined during the *physical design* step of the microprocessor circuit design cycle, which transforms a functional net-list into a circuit layout, through procedures that include floorplanning, placement, and routing.

Traditional physical design, where the typical objectives are minimizing the area and total wire length of the chip, can lead to IPC-suboptimal layouts in the wire pipelining regime. For improved performance, physical design must attempt to keep the IPC-critical buses as short as possible to minimize the amount of pipelining required by those buses. Such an IPC-aware strategy [5] is particularly useful at the early stages of the physical design flow, such as floorplanning, which have a major share in determining the system/global bus delays.

Microarchitecture optimizations and analyses typically involve cycle-accurate simulations on benchmark programs for performance evaluation. It has been widely accepted that the SPEC benchmark suite [6], along with the `reference` input sets, represents a realistic work-load that is executed on microprocessors, and therefore has become a popular choice in microarchitecture research. However, executing the reference input sets to completion, in most cases, is prohibitive, due to the inherent slow nature of the cycle-accurate simulations; simulating one cycle of the target microarchitecture consumes about 3000–5000 cycles of the host machine. To realistically explore the solution search space, microarchitects employ alternative techniques to speed up the simulations, such as reducing the size of the input sets and statistical sampling. This reduction in the simulation times, however, comes at the cost

This work was supported in part by a gift from Intel Corporation, by the NSF under award CCCR-0205227, by the Minnesota Supercomputing Institute, and by the University of Minnesota Digital Technology center.

of loss of accuracy associated with simulating only a fraction of the reference input sets. Such inaccuracies can potentially lead to incorrect conclusions and performance bottlenecks, and, therefore, can undermine an optimization process.

Due to the inaccuracies, it is necessary to understand the nature of the simulation speedup techniques, and, importantly, how these techniques affect the results of an intended microarchitecture optimization, i.e., whether different approaches lead to different conclusions and optimizations. This paper addresses this issue for the IPC-aware floorplanning problem. Specifically, we compare two simulation techniques, namely, reduced input sets and sampling, and study their impact on the overall performance speedup obtained.

A recent work [7] evaluates the accuracies of a number of simulation techniques, including reduced input sets and sampling. The comparison is based on three different characterizations, one each at the hardware (processor bottleneck), software (execution profile), and architecture levels. In addition, the work attempts to quantify the effect of the inaccuracies on the execution times of the benchmarks, for a couple of microarchitecture enhancements [8], [9]. The results of the comparison indicate that, in general, sampling techniques are more reliable than reduced input sets in tracking the actual performance speedups obtained, due to the enhancements, on the reference sets.

However, while these results hold for the enhancements considered, it is possible that the impact of the inaccuracies can vary across different optimizations. Specifically, for the hardware enhancements handled in [7], the decision making is directly based on the results obtained from the simulations, and therefore a high reliability is required. IPC-aware floorplanning, on the other hand, is a discrete optimization problem where the variables are bus latencies. The purpose of the simulations is to describe the IPC of a program as a function of the bus latencies, and the floorplanner uses this description to come up with a block-level placement that represents an IPC-optimal bus latency configuration.

For such optimization problems, a reasonably accurate characterization that does not significantly alter the relative ordering of the performance-criticality of the parameters is sufficient; "absolute" accuracy may not be necessary. We focus on this issue in this paper and the objective is to determine if there is any correlation between perceived inaccuracy of the reduced input sets and the corresponding optimization results for the IPC-aware floorplanning problem. Although our study specifically concentrates on floorplanning, the results are likely to be applicable for any microarchitecture optimization in the physical design context, or, in fact, any related discrete (microarchitecture) optimization problem. For this study, we employ the statistical design of experiments [10] based strategy that is first proposed in [11] for IPC-aware floorplanning.

The remainder of the paper is organized as follows. Section II presents an overview of the IPC-aware floorplanning flow, along with the baseline architecture and block configuration used in this work, while section III lists the benchmarks used, describes the simulation speedup techniques compared for IPC-aware floorplanning, and presents the results of the comparison. We conclude the paper in section V, after describing some related work in section IV.

II. IPC-aware floorplanning

Floorplanning is a physical design stage that determines the positions of the global blocks of a circuit on the layout. Traditionally, the objective of this global placement step has been to minimize a weighted combination of the total area of the layout, the total length of the connections between the blocks, and other topological features of the circuit. However, since wire pipelining has become a necessity to support higher clock frequencies in nanometer technologies, physical design and microarchitecture optimizations can no longer be performed independently. For better performance, floorplanning, thus, must also consider microarchitectural ramifications on the placement of the blocks.

The amount of pipelining required by each bus of a microprocessor is proportional to its length, which is typically true for buffered interconnects [12], and therefore, for every block-level placement, there is a corresponding bus-latency configuration. For each of these configurations, the IPC for a given program can be determined using a cycle-accurate simulation. The objective of floorplanning is to obtain a bus-latency configuration that maximizes the IPC for each benchmark program.

Using cycle-accurate simulations to evaluate the IPC of each and every latency configuration is not practical, given the large, exponential, number of possible configurations. Specifically, if each of n buses on a layout can have k possible latencies, then the cycle-accurate simulator may have to perform up to n^k simulations to fully explore the search space. Therefore, the clear bottleneck in the IPC-aware floorplanning flow is the estimation of IPC.

To reduce the number of simulations to a practical level, we use a statistical design of experiments [10] based simulation methodology for IPC-aware floorplanning. In this approach, the total number of simulations required to sample the search space is proportional to n, compared to the $O(n^k)$ possible combinations of bus latencies. The IPC-impact of each bus is quantified in the form of a weight, and these weights are supplied to the floorplanner. The floorplanner, then, attempts to minimize the weighted sum of bus latencies.

The following sections illustrate this approach, and we tie the description to the processor microarchitecture that is employed in this work. The microarchitecture, which is based on the DLX architecture [13], and the corresponding functional blocks are shown in Table I and Figure 1, respectively. The instruction fetch and decode blocks are shown as fet and dec, respectively, while $il1$ and $dl1$ are the level-1 instruction and data caches, respectively. The instruction and data translation look-aside buffers (TLB) are indicated as $itlb$ and $dtlb$, respectively, while $l2$ represents the unified level-2 cache. The block ruu is the register update unit, which contains the reservation stations and instruction issue logic, while the block lsq represents the load store queue. The system

Parameter	Value
Fetch width	8 instrs/cycle
Issue width	8 instrs/cycle
Commit width	8 instrs/cycle
RUU entries	128
LSQ entries	64
IFQ entries	16
Branch pred	comb, 4K table 2-lev 2K table, 11-bit 2K BHT
BTB	512 sets, 4-way
IL1	64K, 64B, 2-way LRU, latency: 1
DL1	32K, 32B, 2-way LRU, latency: 1
L2	2M, 128B, 4-way latency: 12
ITLB, DTLB	128 entries Miss latency: 200

TABLE I

BLOCK CONFIGURATION OF THE PROCESSOR.

Bus	Parameter	ID
fet_il1		
$il1\_bpred$	$extra\_fet$	1
fet_bpred		
fet_itlb	fet_itlb	2
$itlb\_l2$	$itlb\_l2$	3
ruu_reg	ruu_reg	4
ruu_lsq	ruu_lsq	5
ruu_iadd1	$iadd\_add1$	6
ruu_iadd2	ruu_iadd2	7
ruu_iadd3	ruu_iadd3	8
ruu_imult	ruu_imult	9
ruu_fadd	ruu_fadd	10
ruu_fmult	ruu_fmult	11
lsq_dl1	lsq_dl1	12
$dtlb\_l2$	$dtlb\_l2$	13
lsq_dtlb	lsq_dtlb	14
$il1\_l2$	$il1\_l2$	15
$dl1\_l2$	$dl1\_l2$	16
dec_reg	dec_reg	17
fet_dec	fet_dec	18
dec_ruu		
dec_lsq	max_lsq_ruu	19

TABLE II

BUSES AND FACTORS.

register file is represented by reg, whereas $bpred$ consists of the branch predictor and the target buffer (BTB), which predict the direction and target address for a branch instruction, respectively. The blocks $iadd1$, $iadd2$, $iadd3$, $imult$, $fadd$ and $fmult$ are the functional units that execute arithmetic and logic instructions. The figure also shows the 22 system buses that can impact the performance (IPC) of the processor, when pipelined.

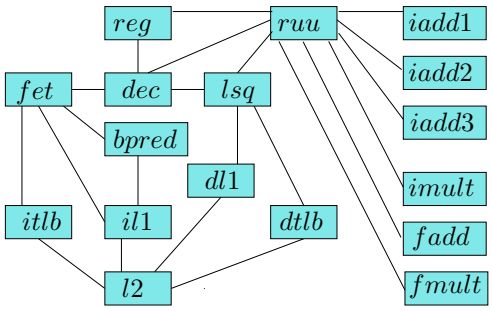

Fig. 1. Microarchitecture and buses.

A. Wire pipelining models

The additional latencies on each of the 22 buses of the microarchitecture are modeled as dummy stages in the processor pipeline. For instance, extra flip-flops inserted on, say, the bus between ruu and $fadd$ units shown in Figure 1 can be modeled as an increase in the latency of a floating-point add instruction. To achieve this, the 22 buses are grouped into 19 factors that can be made configurable in the chosen simulator. The buses, along with the factors they are grouped into, are shown in Table II. For example, the bus between the ruu and $fadd$ units of Figure 1 is indicated as ruu_fadd in the table. As can be seen in Table II, most of the factors directly model the buses with the same name. The first exception is the factor $extra\_fet$, which represents the sum of the latencies of three buses, as shown in Table II. This factor corresponds to the number of extra stages to be inserted in the *fetch* stage of the processor pipeline. The second exception is max_lsq_ruu, which models the maximum of the latencies of the buses dec_ruu and dec_lsq.

B. Simulation methodology

Statistical *design of experiments* is an approach that characterizes the response of a system in terms of changes in the factors which influence the system. The basic idea is to conduct a set of experiments, in which all factors are varied systematically over a specified range of acceptable values, such that the experiments provide an appropriate sampling of the entire search space. The subsequent analysis of the resulting experimental data will identify the critical factors, the presence of interactions between the factors, etc. In this work, the system is a microarchitecture, such as that shown in Figure 1, the response is the IPC, and the factors that influence the IPC of the microarchitecture are shown in Table II. Since it is impractical to fully explore the exponential search space, even when the number of factors is small ($N = 19$), we employ a *fractional factorial* design to reduce the number of

simulations. However, such designs are only valid when some or all of the interactions between the factors are negligible.

In general, it is not easy to identify potential significant interactions before hand in a complex system such as a microprocessor. However, in most cases, the interactions in a microarchitecture tend to be negligible. For instance, it is unlikely that, say, the L2 cache interacts with the instruction decoder, given the varied functionalities of the two units. We have identified a few potential significant interactions, which resulted from the nature of wire-pipelining models integrated into the simulator, as shown below:

- We have incorporated functional unit scheduling in the simulator. Specifically, the number of latencies inserted on the buses between the register update unit and the three integer adders can be different, and while issuing an integer add instruction, of all the available units, the one with the least latency is chosen. This indicates possible significant (two and three factor) interactions.
- In the decode stage, the number of extra pipeline stages to be inserted is modeled as a maximum function of three factors dec_ruu, max_lsq_ruu and ruu_reg (refer to Table II). Such a nonlinear function can result in significant interactions among these three factors.

All of the other interactions, other than the eight listed above, are assumed to be negligible. In addition, to further reduce the number of simulations, we restrict each factor to have two values: the minimum and the maximum possible values for the factor. The idea is that, by stimulating the system with inputs at their extreme values, we provoke the greatest response for each input. As is shown in [14], such a *two-level* approach can be effectively used to design simulation strategies for microarchitectural optimizations. Since the factor levels represent bus latencies, the extreme (high and low) values can be obtained by assuming worst-case and best-case scenarios for the corresponding wire lengths.

These assumptions allow us to utilize a 2-level resolution III fractional factorial design [10]. For N factors, the number of experiments required is equal to the nearest highest power of 2, which turns out to be 32 for our work, since $N = 19$. We refer the reader to [10], [11] for more details of the methodology.

C. Cost function of floorplanning

The output of the resolution III design is a set of factor and interaction weights, each weight quantifies the IPC-impact of the corresponding factor or interaction. These weights are used to construct the cost function of the floorplanner, and the objective is to determine the positions of the blocks such that a weighted sum of factor and interaction latencies, which are related to the bus latencies, besides the traditional objectives such as area and aspect ratio, is minimized.

III. Comparing simulation techniques

Several techniques have been proposed in the past to reduce simulation times to practical levels, while attempting to reproduce the behavior of the reference input sets. These techniques can be broadly categorized into three groups: (i) Reducing the input sets, (ii) Truncated execution, and (iii) Sampling. The work of [7] compares the accuracies of six such techniques, and the results of the work indicate that sampling techniques have much higher accuracies in tracking microarchitecture (reference) performance than the other two categories. However, as explained in section I, the findings are specific to the two enhancements considered, and it is not clear whether the inaccuracies can be generalized for all microarchitectural optimizations. In discrete optimization problems such as IPC-aware floorplanning, a moderate perturbation in the weight of a factor (or an interaction) may not be sufficient to shift the optimal value of that factor by an integer above or below.

Specifically, changing the latency of a bus in a particular placement involves a significant change in the locations of the connecting blocks in the layout, to increase/decrease the bus length by appropriate amount, and this can potentially lead to a massive realignment of the positions of other blocks, resulting in a drastically different placement with a significant change in the value of the cost function, which includes the weighted sum of factor/interaction latencies. It is unlikely that small or moderate perturbations in factor weights can result in such a scenario, which changes the cost function by a significant amount, during optimization. Therefore, any simulation technique with a reasonable accuracy (or moderate inaccuracy) may be sufficient in problems such as IPC-aware floorplanning.

In this section, we compare a few approaches that can be used to speed up the simulation methodology described in section II-B for the IPC-aware floorplanning problem. Due to the high number of simulations (32 per benchmark) required for each technique, we limit our comparison to two techniques, namely, sampling and reduced input sets, as shown below:

- *Reduced Input sets:* The idea behind the reduced input sets is to alter the reference input sets so that the simulation times are reduced when using these reduced input sets, while endeavoring to retaining the characteristics of the unaltered reference input sets. The `test` and `train` input sets from SPEC, and the MinneSPEC [15] reduced input sets are a few examples. For this work, we choose MinneSPEC reduced input sets for evaluation.
- *Sampling:* In statistical sampling, only selected portions of the instruction sequence of a benchmark are measured. The program segments between the selected portions are fast-forwarded using functional simulation. These samples must be chosen carefully such that they accurately reflect the behavior of the *population*, i.e, the whole program. The sampling technique proposed in [16], called SMARTS, simulates periodically selected subsets of the instruction sequence. The sampling frequency and the length of each sample are used to control the simulation time. The statistics measured for the simulated samples are generalized for the whole program. In addition, SMARTS uses statistical sampling theory to estimate the IPC error of the sampled simulation results, as compared to the complete simulation. On the other hand, the approach of SimPoint [17] selects a few representative

simulation points beforehand and then uses statistical based clustering to select a set that is representative of the whole program. At the end of the simulation, the results from each simulation point are weighed to compute the final statistics. The number of simulation points, and the length of each determines the simulation time.

We choose SMARTS as a representative of sampling techniques for our comparisons. As noted in [7], there is little difference in the accuracies of SMARTS and SimPoint. One reason we decided on SMARTS is that the simulator provided by the authors also includes a framework for power estimation, based on Wattch [18], and this facilitates our future plan of extending the methodology of this paper to optimize other critical objectives, such as power consumption.

In addition to the above mentioned techniques, we consider a third case, a hybrid approach that is obtained by combining the two techniques. Specifically, in this case, we apply SMARTS on the MinneSPEC reduced input sets, to further reduce the simulation times. Hence, we actually compare three techniques in this paper.

A. Benchmarks

For simulations, we use sim–smarts [16], which is based on the sim–outorder [13] simulator, developed by the authors of SMARTS. Wire pipelining models are inserted in the simulator, as per the description provided in section II-A, and each of the factors is made configurable. In addition, we use the default values that are listed in [16] for the sampling parameters (a sampling interval of 1000 and a sample size of 1000). We choose a set of eight SPEC 2000 benchmarks for evaluations in this work. The benchmarks, along with the corresponding reference and MinneSPEC input instruction counts are shown in Table III. The total simulation time limited the number of benchmarks that we could use. The benchmarks are chosen because of their distinct instruction mixes. For instance, 177.mesa has a high percentage of conditional branches, while 181.mcf has a very large number of memory operations, particularly "store" instructions. All benchmarks are compiled at optimization level O3 using the SimpleScalar version of the gcc compiler.

B. Results

For each of the three cases described in the previous section, we perform 32 simulations per benchmark as prescribed by the resolution III design of section II-B. Our implementation uses PARQUET [19], a simulated annealing (SA) based floorplanner available in the public domain. The advantage of this SA-based approach is that it allows easy integration of our weights into the cost function.

The areas of the individual blocks, showed in Figure 1, are based on [20]. The low value for a bus latency is chosen to be 0, depicting the best case placement of the connecting blocks. For the high value, we pick the corner-to-corner latency in the chip, for a frequency of 6.0GHz in 90nm technology, based on projections from [21].

Benchmark	Type	Instr. count (Billions)	
		MinneSPEC	reference
164.gzip	Integer	1.065	63
175.vpr	Integer	0.217	110
177.mesa	Floating-point	1.297	305
179.art	Floating-point	7.700	54
181.mcf	Integer	0.175	49
183.equake	Floating-point	0.716	175
197.parser	Integer	0.914	301
256.bzip2	Integer	3.800	94

TABLE III
BENCHMARKS FROM THE SPEC SUITE, ALONG WITH THE REFERENCE AND REDUCED INSTRUCTION COUNTS.

We present results for a number of clock frequencies, ranging from 3.0GHz to 6.0GHz. The weights can be used for each of these frequencies, since the bus latency ranges are valid for all of the frequencies less than or equal to 6.0GHz. In addition, we assume that the operating frequency of the chip is constrained only by the bus delays, and the maximum of the delays of the buses is the minimum possible clock period when wire-pipelining is not employed. The corresponding maximum frequency, obtained by minimizing the maximum of wire lengths of the global wires in the floorplanner, is determined to be about 1.6GHz, and this forms the baseline unpipelined design.

We compare four different floorplanning scenarios in this section. The first floorplanner does not use weights, and attempts to minimize traditional objectives, such as total area and wire length. The other floorplanners use factor/interaction weights generated using three simulation speedup techniques as prescribed by the resolution III design described in section II-B. The four scenarios are labeled as shown below:

- **minWL:** Traditional floorplanning, does not require simulations.
- **Minne:** MinneSPEC reduced input sets are simulated to completion.
- **SMARTS-R:** SMARTS is applied on the reference input sets.
- **SMARTS-M:** SMARTS is applied on the MinneSPEC input sets.

The comparison metric is the execution time, T_{exec}, which, as shown in (1), is the product of the number of instructions (N_{instr}) in the benchmark, the reciprocal of the IPC ($\frac{1}{IPC}$), and the corresponding clock cycle time evaluated as the reciprocal of the clock frequency ($\frac{1}{f}$).

$$T_{exec} = \frac{N_{instr}}{IPC \cdot f} \quad (1)$$

In addition, we use a common platform to compare the four cases: the evaluations are performed on the reference input sets, with SMARTS speeding the simulations. In doing so, we are biasing the evaluations towards the SMARTS-R technique. We note that, our objective is to examine how the reduced

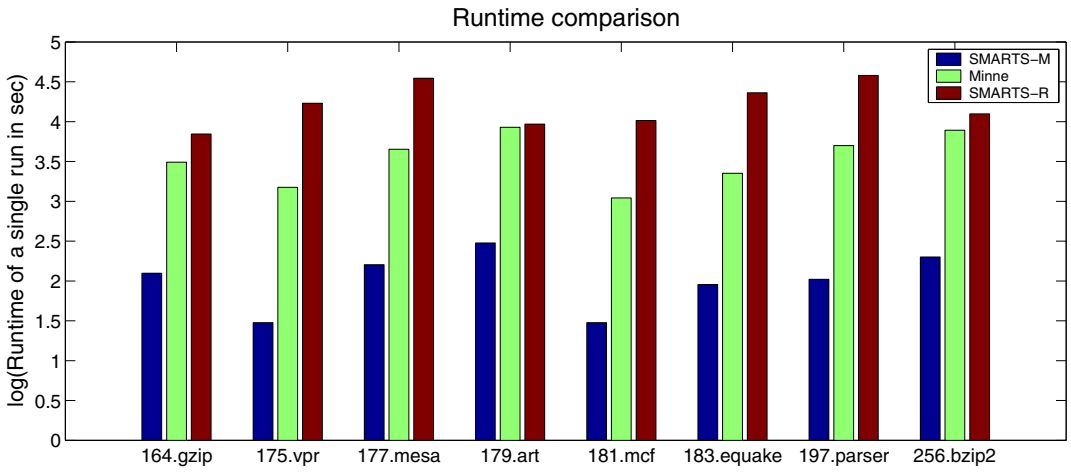

Fig. 2. Runtime comparison of a single simulation for the three techniques. The y-axis plots the runtimes in logarithmic scale.

input sets compare with the sampling techniques, not exactly to measure the accuracy of these techniques in tracking the reference performance. The reason behind choosing "sampling–on–reference" as the common framework is that sampling techniques replicate the reference behavior with very high accuracies, as indicated in [7], and the speedups evaluated on this platform are likely to represent those observed when the reference input sets are run to completion.

Figure 2 plots the runtimes of a single simulation for the three techniques in logarithmic scale. It can be seen from the figure that SMARTS-R has the longest simulation times among the three techniques. However, the SMARTS-M case has simulation times that are more than two orders less than the other two approaches, specifically SMARTS-R, while the simulation times in Minne are somewhere between those of the other two techniques. As an example, for the benchmark 181.mcf, the simulation times associated with SMARTS-M, Minne, and SMARTS-R are 30, 1100, and 10300 seconds, respectively.

Figure 3 presents the results obtained from floorplanning for the four scenarios described earlier in this section, for each of the eight benchmarks, and for four frequencies, 3-6GHz. For each benchmark, all execution times plotted in the graphs are normalized to that of the corresponding baseline case, where the frequency is 1.6GHz. It can be observed that the normalized execution times are less than one for all cases shown in Figure 3, suggesting that interconnect pipelining, indeed, results in performance improvement over the baseline processor for each of the benchmarks. For example, at 3GHz, we see that SMARTS-M produces a design that results in only 54% of the execution time that would be required for the baseline processor for 175.vpr. In addition, all of the three simulation speedup techniques outperform the traditional floorplanning, shown as minWL, and the improvements tend to get better at higher frequencies, which is not surprising since the amount of pipelining required, in general, gets higher as the frequency increases. For instance, on an average, the case SMARTS-R, where SMARTS is applied on reference input sets, results in an improvement of about 11% at 3GHz over the traditional floorplanner, shown as minWL in the figure, while it is about 40% at 6GHz.

In addition, for most benchmarks, there is not much difference in the execution times obtained for the three cases, Minne, SMARTS-R and SMARTS-M, and the differences in execution times are with in 1%. This indicates that the reduced input sets compare well with the sampling technique for the IPC-aware floorplanning problem. In fact, by employing sampling (SMARTS) on the reduced input sets (MinneSPEC), we can drastically reduce the simulation times without much loss in the performance. For instance, for the benchmark 175.vpr, on an average, each simulation run for SMARTS-R takes about 5 hours. However, almost the same performance improvements (as seen in SMARTS-R) can be obtained when MinneSPEC reduced inputs are used to generate the factor/interaction weights, and in this case (Minne), the time required for each simulation is about 30 minutes, a nearly $10\times$ speedup over SMARTS-R. The simulation times can further be decreased with a negligible reduction in the performance by sampling the reduced input sets, i.e, SMARTS-M, where each simulation runs for about 30 seconds, approximately $600\times$ faster than SMARTS-R.

We note that, due to the similarities in the results obtained, there are no accuracy-runtime tradeoffs that can be explored. From Figures 2 and 3, it is clear that, SMARTS-M, while achieving the same performance speedups as the other two techniques, represents the best approach with least simulation times, in the order of a few hundreds of seconds. Given the small runtimes of SMARTS-M, it may also be possible to employ a more accurate or a higher resolution design than that it is described in section II-B.

Table IV shows pairwise comparisons of the three techniques in terms of the magnitudes of the weights obtained from the resolution III design of section II-B for the three techniques, SMARTS-R, SMARTS-M, and Minne. For each pairwise combination of these three, the value shown for each benchmark is the average Euclidean distance between

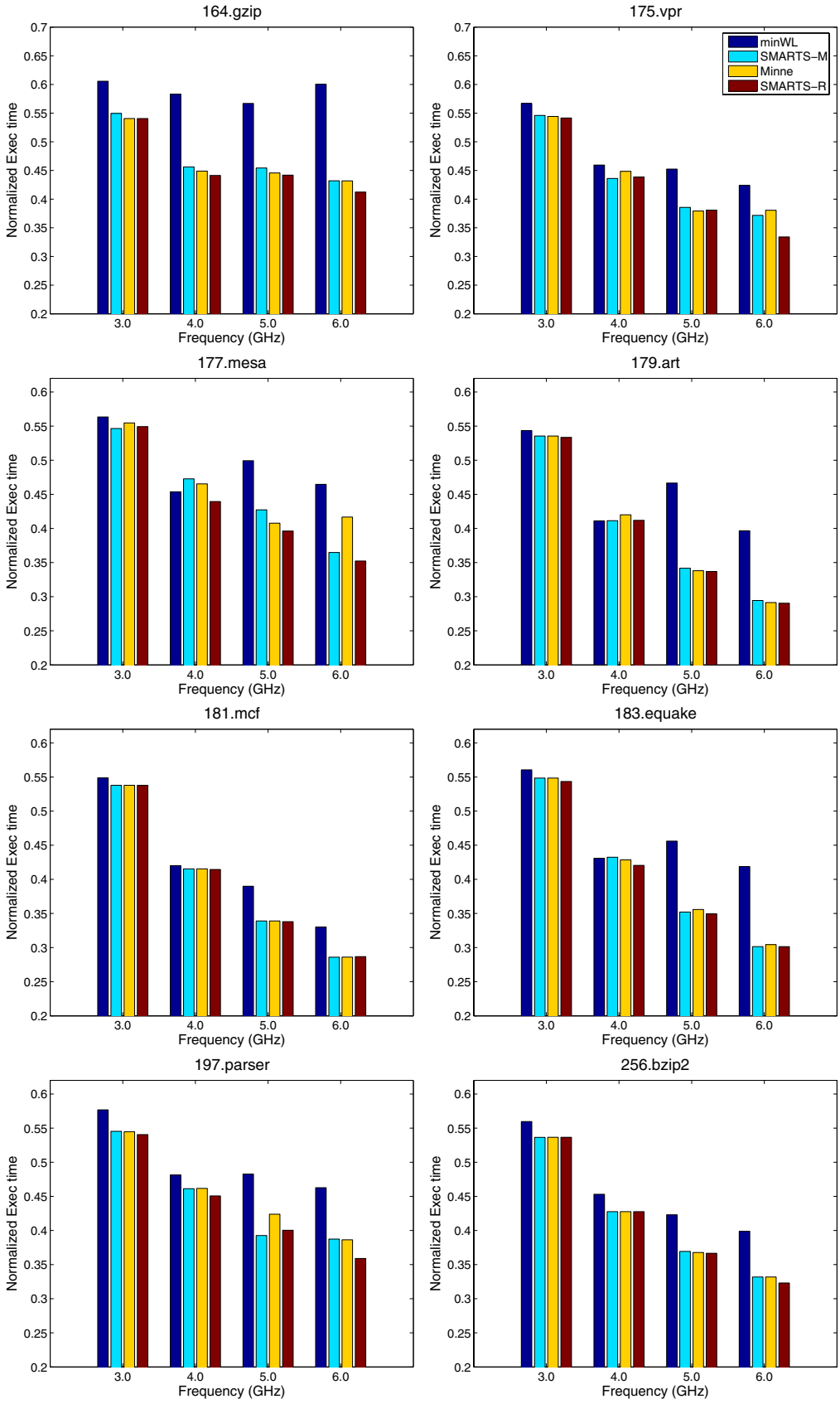

Fig. 3. Results for eight benchmarks for four different frequencies. The execution times are normalized to the baseline case, where wire-pipelining is employed and the frequency cannot exceed 1.6GHz.

the corresponding main and interaction weight vectors. If $X = \langle x_1, \cdots, x_n \rangle$ and $Y = \langle y_1, \cdots, y_n \rangle$ are two weight weights, then the average Euclidean distance between X and Y is determined as shown below:

$$E_{xy} = \sqrt{\frac{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}{n}} \quad (2)$$

Each weight vector is normalized to 100, i.e., maximum weight in each vector is 100. In such a case, the maximum bound on the average Euclidean distance is 100, with the minimum being 0. The idea of this distance metric is to observe how the factor/interaction weights generated in the three cases compare with each other. Each value in Table IV indicates how much the weight of a factor/interaction obtained using an approach differs, on an average, from the corresponding weight obtained using the compared technique. For instance, for the benchmark 175.vpr, a value of 8.18 shown in column labeled **SMARTS-R Vs Minne** indicates that, on an average, the weight of a factor/interaction in the case Minne differs by about 8 from that of the corresponding factor/interaction obtained using SMARTS on the reference input sets (SMARTS-R).

- **Minne Vs SMARTS-M:** The distance is negligible for most of the benchmarks, as shown in the table, which suggests that sampling with SMARTS on the reduced input sets tracks the behavior of the whole input sets with high accuracy. This is an interesting observation, since, it shows that, for applications which employ reduced input sets, the simulations can further be speeded up by applying sampling.

- **SMARTS-R Vs Minne:** The distances are relatively higher than those seen in **Minne Vs SMARTS-M** comparison, presumably because of change in the input sets, in tune with the conclusions of [7]. However, other than 177.mesa and 181.mcf, the distances are still moderate, and it is unlikely that such moderate changes in the factor and interaction weights shift the optimal operating points by significant amounts, given the discrete nature of the cost function that is minimized in floorplanning. For 177.mesa and 181.mcf, the reason behind the large differences is the contrasting instruction mixes of the corresponding MinneSPEC and reference input sets. For instance, MinneSPEC input set for 177.mesa has negligible floating point instruction count, while the reference input set has about 9% floating point instructions. Similarly, the reference input set of 181.mcf has about 35% load instructions, whereas it is about 23% in the MinneSPEC input set. This is also the reason behind the relatively higher differences in the execution times obtained for 177.mesa for the three techniques shown in Figure 3. However, the big differences in the weights for 181.mcf do not affect the pattern of the results, as shown in Figure 3. A possible reason behind this is that the relative ranking of the factors is maintained even if the weights differ by a significant magnitude.

- **SMARTS-R Vs SMARTS-M:** The distances follow the same trend as seen in the **SMARTS-R Vs Minne** comparison above, since both Minne and SMARTS-M use reduced input sets. The distances are slightly lower than those observed in **SMARTS-R Vs SMARTS**, however. This may be because both SMARTS-R and SMARTS-M employ SMARTS, the inaccuracies, however small they are, associated with the SMARTS technique creep into both of them, having identical effects.

IV. Related work

There has not been much published work that specifically compares simulation speedup techniques, other than [7] described in section I. To the best of our knowledge, our paper is the first contribution that handles the comparison in the floorplanning (or, in general, the physical design) context. However, there has been a good amount of research, in the last couple of years, towards introducing microarchitecture-awareness in the floorplanning domain.

IPC-aware floorplanning: Besides our statistical design of experiments based strategy of [11], there have been some other recent attempts [20], [22] towards IPC-aware design at the floorplanning level of physical design. In [20], a IPC look-up 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 IPC is evaluated from the LUT using some distance metrics. In contrast, the approach in [22] assigns weights to each of the system buses that 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, where the weights depend on the amount of traffic.

The work of [23] uses a "one-at-a-time" approach to build IPC sensitivity models for a few selected critical paths, and these models guide the floorplanner to optimize the system IPC. Another recent approach [24] explores the frequency-IPC tradeoff in floorplanning. A set of implementations varying in area and latency is specified for some or all of the blocks of the processor. The objective of the floorplanner is to find a configuration of blocks with a placement to reduce the product of clock period and the CPI. The lengths of the global wires, combined with given arrival times at the terminals, determine the clock period.

Simulation methodology: The work of [14] uses a Plackett and Burman, PB [25], design to rank the criticality of the microarchitectural configuration parameters. However, PB designs are useful only when the interactions among the parameters are negligible, and therefore are generally not used in analyses where interactions are significant. Additionally, this work attempts at identifying similarities in the benchmarks and groups them using a distance metric similar to that of (2).

Statistical simulation [26]–[28] is another way of reducing the simulation times, thereby allowing an efficient exploration of the design search space. For a benchmark program, using a single detailed simulation, statistical tables are constructed

Benchmark	SMARTS-R Vs Minne	SMARTS-R Vs SMARTS-M	Minne Vs SMARTS-M
164.gzip	10.46	10.66	0.74
175.vpr	8.18	7.50	2.18
177.mesa	27.14	23.93	8.38
179.art	11.13	9.88	1.86
181.mcf	20.79	20.62	0.93
183.equake	10.09	10.30	0.78
197.parser	7.30	6.61	1.50
256.bzip2	2.36	2.43	0.64

TABLE IV

PAIRWISE DISTANCE COMPARISON OF THE THREE TECHNIQUES. FOR EACH PAIR OF TECHNIQUES, THE VALUES SHOWN REPRESENT THE AVERAGE DIFFERENCES BETWEEN THE FACTOR/INTERACTION WEIGHTS, COMPUTED USING (2). THE MAXIMUM WEIGHT IN EACH TECHNIQUE IS NORMALIZED TO 100, I.E., THE MAXIMUM POSSIBLE DISTANCE IS 100.

for various program characteristics such as cache miss rates and register dependencies. This synthesis trace generated, essentially a *statistical image* of the benchmark, can be used to speed up the subsequent simulations of the benchmark.

V. CONCLUSION

Microarchitects resort to alternative techniques to speed up the simulation process due to the long simulation times of the reference input sets. However, these techniques suffer from a loss of accuracy since only a fraction of the reference instruction sequence is actually simulated. This paper compared two such techniques, MinneSPEC reduced input sets and SMARTS for the IPC-aware floorplanning problem, where the objective is to find an IPC-optimal block-level placement. The purpose of the simulation methodology is to quantify the IPC-criticality of each of the buses of a microprocessor with a weight.

We use a distance metric to compare the set of weights generated using the two techniques in the simulation methodology. This comparison suggests that the two techniques generate significantly different sets of weights. However, this variation in the weights did not affect that subsequent optimization, and the performance improvements seen in both cases are almost identical. Therefore, there is no correlation between the contrasting weights generated and the actual delivered performance. The best technique for this optimization is obtained by applying SMARTS on the MinneSPEC reduced input sets, and this case drastically quickens the simulation process by several orders, besides generating high quality designs.

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/ACM ICCAD*, pp. 247–253, Nov. 2002.

[4] V. Agarwal *et al.*, "Clock-rate vs IPC: the end of the road for conventional architectures," in *Proc. ACM ISCA*, pp. 248–259, May 2000.

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

[6] J. L. Henning, "SPEC CPU 2000: Measuring CPU performance in the new millennium," *IEEE Computers*, vol. 33, Jul. 2000.

[7] J. J. Yi *et al.*, "Characterizing and comparing prevailing simulation techniques," in *Proc. ACM HPCA*, pp. 266–277, Feb. 2005.

[8] J. J. Yi and D. J. Lilja, "Improving processor performance by simplifying and bypassing trivial computations," in *Proc. ACM ICCD*, pp. 462–467, Oct. 2002.

[9] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proc. ACM ISCA*, pp. 364–373, May 1990.

[10] D. C. Montgomery, *Design and analysis of experiments*. New York, NY: John Wiley, 1991.

[11] V. Nookala *et al.*, "Microarchitecture-aware floorplanning using a statistical design of experiments approach," in *Proc. ACM/IEEE DAC*, pp. 579–584, Jun. 2005.

[12] N. Sherwani, *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Boston, Massachussets, 2nd ed., 1995.

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

[14] J. J. Yi *et al.*, "A statistically rigorous approach for improving simulation methodology," in *Proc. ACM HPCA*, pp. 281–291, Feb. 2003.

[15] A. J. KleinOsowski and D. J. Lilja, "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research," *IEEE Computer Architecture Letters*, vol. 1, Jun. 2002.

[16] R. E. Wunderlich *et al.*, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proc. ACM ISCA*, pp. 84–97, Jun. 2003.

[17] T. Sherwood *et al.*, "Automatically characterizing large scale program behavior," in *Proc. ACM ASPLOS*, pp. 84–97, Oct. 2002.

[18] D. Brooks *et al.*, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. ACM ISCA*, pp. 83–94, June 2000.

[19] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *Proc. IEEE ICCD*, pp. 228–334, Oct. 2001.

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

[21] J. Cong, "An interconnect-centric design flow for nanometer technologies," *Proc. IEEE*, vol. 89, pp. 505–528, Apr. 2001.

[22] M. Ekpanyapong *et al.*, "Profile-guided microarchitectural floorplanning for deep submicron processor design," in *Proc. ACM/IEEE DAC*, pp. 634–639, Jun. 2004.

[23] A. Jagannathan *et al.*, "Microarchitecture evaluation with floorplanning and interconnect pipelining," in *Proc. ACM/IEEE ASPDAC*, vol. 1, pp. 8–15, Jan. 2005.

[24] J. Cong *et al.*, "Microarchitecture evaluation with physical planning," in *Proc. ACM/IEEE DAC*, pp. 32–35, Jun. 2003.

[25] R. Plackett and J. Burman, "The design of optimum multifactorial experiments," *Biometrika*, vol. 33, pp. 305–325, Jun. 1956.

[26] S. Nussbaum and J. E. Smith, "Modeling superscalar simulators via statistical simulation," in *Proc. ACM PACT*, pp. 15–24, Sep. 2001.

[27] L. Eekhout *et al.*, "Performance analysis through synthetic trace generation," in *Proc. IEEE ISPASS*, pp. 1–6, Apr. 2000.

[28] M. Oskin *et al.*, "HLS: Combining statistical and symbolic simulation to guide microprocessor designs," in *Proc. ACM ISCA*, pp. 71–82, May 2000.