RC 21909 (98536) 8 December 2000 **Computer Science**

IBM Research Report

Abstraction via Separable Components: An Empirical Study of Absolute and Relative Accuracy in Processor Performance Modeling

David Brooks, Margaret Martonosi*, Pradip Bose
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

* Princeton University

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Abstraction via Separable Components: An Empirical Study of Absolute and Relative Accuracy in Processor Performance Modeling

Paper Number 291

Abstract

With each new chip generation, high-end microprocessors become more complex and therefore more difficult to design, analyze, model and simulate. Developing and debugging cycle-accurate simulators for modern microprocessors is increasingly time-consuming. Likewise, using these simulators to explore a design space thoroughly can take months or more of simulation time. It is therefore increasingly important that CPU designers employ abstraction methodologies that allow them to make good microarchitectural design decisions without relying solely on many runs of cycle-accurate simulators.

In this paper, we examine one popular modeling abstraction in detail. The separable components methodology separates program performance into an idealized base CPI (cycles per instruction) figure and other additive stall figures corresponding to key components of program performance. While this method is widely used by designers as a back-of-the-envelope trick, prior research has not thoroughly examined its accuracy characteristics. This paper studies the abstraction's accuracy in detail. Our primary finding is that separable components has excellent *relative* accuracy and is therefore highly effective at choosing appropriate design points from a design space, even in cases where its *absolute* accuracy is less impressive. Since the method provides upper and lower bounds on performance, it is also useful for early-stage reasoning about a design's sensitivity to different parameters and design choices.

1 Introduction

Designing high-end microprocessors requires more than a year of modeling efforts that begin with fairly high-level models intended to explore the broad design space, and that then move towards lower-level models with more details as the design matures. As microprocessors have increased in complexity, the corresponding simulation and design time overheads have ballooned [17]. In light of this, efforts to manage the overheads of CPU simulation times are as crucial as ever.

In industry, a typical modeling progression for designing a high-end processor involves a series of steps through simulators with different speed vs. accuracy tradeoffs. For example, architecting high-end CPUs at our company involves at least three different simulators as shown below:

- **RTL simulator in VHDL:** This simulator is a faithful register-transfer-level (RTL) simulator which includes enough detail that it will be passed off to circuit designers to help specify the architecture's

behavior. This model includes both all the functional logic as well as the pipeline stage information of interest to logic designers and architects. Simulation speeds at this level are typically on the order of tens of instructions per second, even on high-end workstations. Furthermore, this level of simulator is typically available only very late in the design process, once most of the design details have been set.

- **Latch-accurate simulator written in C:** This simulator is a hybrid between higher-level architectural simulators written in C, and more detailed RTL simulators written in VHDL. It achieves speeds of roughly 1.5K instructions per second.

- **High-level, general-purpose architectural simulator:** This level is similar to the SimpleScalar platform widely used in academia [2]. Such models are parameterized, cycle-by-cycle simulators that model the pipeline flow through a target machine, without actually simulating individual latch or register values in detail. Simulation speeds are roughly 100K instructions per second on a 500MHz simulation engine.

Although high-level architectural simulators are the fastest of the three models typically considered, today's processors are so complex that even these abstracted models are often still not fast enough to allow usefully large programs or workloads to be simulated to completion. Thus, a fourth level of abstracted modeling involves methods of sampling or simplifying the processing done by these high-level simulators. These methods include:

- Workload sampling: Portions of the program or workload are sampled rather than executing the programs to completion [16, 9, 6].

- Scaled microbenchmarks: Construct microbenchmarks that exhibit behavior that statistically resembles the full workload [15].

- Analytic models: Equation-based models that reflect key aspects of the underlying architecture [18, 13].

- Separable components: Consider performance to be the summation of a base performance level, plus several additive stall factors. Base performance and stall factors are computed from simpler simulators, rather than from fully-detailed, cycle-accurate simulators.

The first three of these abstraction methods have been published on in some detail. Our work primarily considers the fourth abstraction methodology: separable components. The method of separable components has several key benefits. Two of the major benefits of this method relate to simulation and validation time and effort. First, with the method of separable components, performance simulators can be built with a divide-and-conquer approach in much less time than building a traditional detailed simulator. For example, existing cache and branch prediction simulators can quickly be put together to quantify the performance effects of individual components. New components that need to be modeled can be developed separately, but with much less time and debugging effort than in building the integrated simulator. If the accuracy aspects are well understood, reasonably reliable projections, especially with upper and lower performance bounds,

are extremely valuable in the very early stages of a new project. These models will often be refined and eventually serve as a validation check for the eventual full-blown, detailed integrated model. The validation aspect, and the ability to understand the components of CPI degradation systematically is a real practical benefit.

The second major benefit from separable components involves actual simulation run-time. Pre-processing of traces with information from one-time cache and branch-predictor simulation runs can provide significant speedup. For prior early-stage cycle simulators developed in our company, the measured speedup factor was between 2x and 3x. With a more recent processor model (in which the processor being modeled was more complicated), the runtime savings were up to 46% with an average of around 28%. Obviously, if more components are separated out and inserted into pre-processed traces, additional speedup could be obtained. For example, in full system simulations, memory, bus, switch, and interconnect networks could be separated out. Finally, unlike other simulation abstraction methods, the method of separable components provides an upper and lower-bound on performance as will be explained in Section 2. Other methods, such as sampling, do not provide these bounds. The benefits that we have described do result in some loss of simulation accuracy. In this work we describe a large set of simulation-based experiments to quantify the loss of accuracy seen by this method, and we compare against a cycle-accurate simulation of a current-generation superscalar processor developed at our company.

This paper has two main contributions. First, we explicitly report on the accuracy tradeoffs of separable components when used as part of a modern family of CPU simulators. Second and more importantly, we demonstrate that while *absolute* accuracy often suffers in these more abstract simulators, the *relative* accuracy of these simulators can be quite acceptable. That is, while it may be difficult to use the abstracted simulator to predict a precise cycle count for a benchmark's execution, the *shape* of the curve it gives across a design tradeoff will allow the same design point to be chosen as in a slower, more-precise simulator.

Although the field of CPU performance evaluation is a mature one, we feel that these contributions are timely and important now. Design times and simulation overheads for current processors are becoming more and more burdensome; increasing efforts are needed to prune design spaces early, and to keep simulation times in check. The method we describe and evaluate here is an aid to designers in these efforts.

The remainder of this paper is structured as follows. In Section 2 we discuss the method of Separable Components in detail. In Section 3 we discuss the experimental methodology used to obtain our empirical results. In Sections 4 and 5 we discuss these results in detail. Finally, in Sections 6 and 7 we discuss related work and offer our conclusions.

2 The Method of Separable Components

The method of separable components has been used extensively, especially in older microprocessors with blocking caches and little instruction-level concurrency [4]. In these machines, the method has been known to provide quite good performance estimates. However, the accuracy implications of this method have not been widely studied, especially in modern superscalar processors which seek to increase performance by

exploiting instruction-level parallelism. In this section, we explain the details of this method, and how it can be used to estimate the average CPI performance of a system, for a given workload.

For a given processor, the CPI is formulated as:

$$CPI = CPI_{InfPrf} + \sum^{j} \Delta CPI_j \quad (1)$$

where CPI_{InfPrf} is the idealized ILP limit for the particular (machine, workload) pair. It measures the best-case CPI, assuming infinite buffer/cache resources and perfect prediction mechanisms, if applicable. Each "CPI adder" component, ΔCPI_j is contributed by pipeline disruption (hazard) events of type j that are not modeled in calculating the best-case CPI_{InfPrf} number.

Elaborating on ΔCPI_j, this expands to:

$$CPI = CPI_{InfPrf} + \sum^{j} (CyclesPerHazardEventofTypej) * (EventsofTypejPerInstruction) \quad (2)$$

Such a formalism is commonly used in summarizing cache effects:

$$CPI = CPI_{InfCache} + \Delta CPI_{FCE} \quad (3)$$

where the finite cache effect (FCE) on CPI is evaluated as:

$$\Delta CPI_{FCE} = (CacheMissPenalty) * (MissRate) = (CyclesPerMiss) * (MissesPerInstruction) \quad (4)$$

Other hazards that can be treated as "CPI adder" components, separate from a baseline CPI are: branch misprediction hazards, value misprediction hazards, store-load address conflicts, cache bank conflict hazards, data-dependent pipeline stalls, and other stalls due to finite queue/buffer sizes. The choice of the number and type of such "CPI adder components" depends on the pipeline structure and on how the "base, idealized CPI" is defined. The experiments reported in this paper (see Figure 2 and the results shown in Sections 4 and 5) use a baseline InfPrf model in which the caches are infinite and the branch prediction is perfect.

2.1 Example and Overview

Separable components, like any modeling abstraction, cannot be expected to be fully accurate when compared to a detailed, cycle-by-cycle simulation of the full machine. Here we use an example scenario to examine the possible sources of error in this model. Figure 1 shows the CPI plot for a processor, based on Equations 3 and 4 above. Along the x-axis, the cache miss rate (measured in misses per instruction) is plotted and the net CPI is plotted along the y-axis. The behavior predicted by Equation 4 results in a straight line, where the y-intercept is the base, infinite-cache CPI; this represents a lower bound on the true CPI the program will encounter. An upper bound on true CPI can be formed by conservatively estimating the CPI due to cache misses.

A key source of inaccuracy in separable components (SC) is overlap among different stall events. SC assumes that stall times are additive; when they are not, the estimated CPI is a loose upper bound, rather

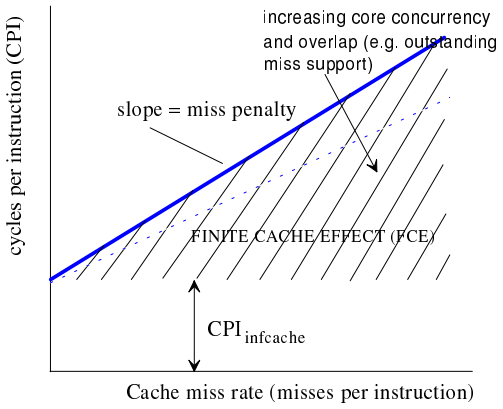

Figure 1: CPI variation with cache miss rate: separable components.

than a tight estimate. For example, if cache miss latencies can be overlapped with useful computation, then the linear relationship between misses and performance in Figure 1 will not be so precise.

In a processor model where the overlap can be varied by changing the number of cache ports, the number of outstanding cache misses or the load/store queue sizes, etc., the slope and shape of this curve would vary with increasing modes of concurrency and overlap, as shown by the dashed line in Figure 1.

In general, the cache misses over a given workload are often "clustered", i.e. they often tend to occur in bursts, separated by miss-free periods once the missing cache lines are brought in. Also, the "miss penalty" may not be a fixed number, even assuming a single level of cache between the processor and main memory. This is because:

- In a modern processor, the cache is usually non-blocking, capable of supporting multiple outstanding misses. So, the latency of a load miss partially overlaps that of a subsequent one. This causes the effective miss latency of a given load to vary, depending on context.

- A missed line, when in transit from the memory to the cache, may make the effective miss penalty for successive misses to that line appear to be a variable number, because of the "leading and trailing edge" effects: often, the critical word or word-multiple containing the missed reference is forwarded first, while the full cache line is loaded over multiple cycles. An immediate, subsequent reference to the same cache line may exhibit different latencies depending on whether it belongs to the initial "critical" datum that was forwarded or the exact spatial coordinates of the reference within the incoming cache line.

- Compared to load misses, store misses typically incur a much smaller cycle penalty on average in most superscalar processors of the current generation. This is because of the way stores are processed "on the side" using a separate store queue (feeding a dedicated cache write port). The store queue can hold store instructions that are "complete" from the viewpoint of architected state, while waiting for

the pending cache array write operation.

Thus, the questions that may arise in using this type of modeling abstraction are:

- What is the inaccuracy in the estimated CPI that results from ignoring the instantaneous variations and burstiness of stall frequencies and considering only a single average or worse-case value?
- Can the hazard's stall penalty be approximated as a single, average or worst-case number and if so, how should that value be computed for the given processor?
- How does overlap between events impact the estimate?

Regarding the first question, one can address the variable event frequencies by performing a detailed statistical analysis of the temporal distribution of events (such as cache misses) over detailed simulation runs. For example, the inter-miss distance histogram could be plotted. Suppose the distribution is unimodal, and approximately symmetric about the maximum value. Then, if the miss penalty is either constant or if its distribution correlates well with the miss rate distribution, the method of averages in computing the delta-CPI component can be expected to be quite accurate.

For most uses, however, average event frequencies (like cache miss rate and branch misprediction rate) can be evaluated efficiently using fast, stand alone simulators. These average event rates, which are computed just once per workload, can be reused for multiple tradeoff studies in tuning the core microarchitecture. If the accuracy is acceptable, the modeling abstraction afforded by the method of separable components can thus yield considerable speedup over full simulation-based experiments.

Regarding the second question, determining the "correct" event penalty value to use can be quite involved, requiring detailed consideration of the various scenarios in which a miss may be serviced. If the worst-case miss penalty is assumed, the method of separable components will always yield a CPI value which is an upper bound. Likewise, overlap between events will further cause the performance estimate to be an upper bound. Although separable components will over estimate the actual CPI in these cases, we show in later sections that this bounding still leads to reliable design choices.

2.2 Relative vs. Absolute Accuracy

Ultimately, a fundamental question about a modeling abstraction is: does such an abstraction allow the designer to make useful, reliable design decisions, even if the absolute performance estimates are not always fully accurate? For example, a model may systematically overestimate all program run-times and yet still reliably deduce that design option A is preferable to design option B.

For example, consider a scenario where the sizes of various queues and buffers are being chosen. The goal is to choose each resource size such that increasing it further will not help performance significantly, but decreasing it at all will tend to degrade performance. Rewriting Equation 4, we can include CPI adders as functions of these queue and buffer sizes:

$$CPI(x_1, x_2, ..., x_n) = CPI_{InfCache}(x_1, x_2, ..., x_n) + CacheMissRate(x_1, x_2, ..., x_n) * CacheMissPenalty(x_1, x_2, ..., x_n) \quad (5)$$

x_1 through x_n are queue and buffer length parameters of interest in our design trade-off study; we assume that cache miss rate and penalties are at least loosely related to them. (None of the parameters x_1 through x_n are allowed to represent cache size or geometry parameters, since in this particular case, we are considering the cache effect adder as a separable component).

Since miss rate and miss penalty depend on the queue/buffer sizes, it is interesting to consider the variation of CPI with each of the parameters x_1 through x_n. Each of the n sensitivity curves is expected to be of the form where the CPI starts from a higher value and diminishes monotonically towards an a minimum. The "knee" of this curve is typically a good design point for the parameter we are varying. This paper's key question regarding relative accuracy is: can these knee values and design points for the x_i parameters be accurately predicted by using the modeling abstraction considered in this paper? Sections 4 and 5 indicate that the answer is yes.

Note, from Equation 5 that if cache miss rate and penalty are constants that do not depend on a given parameter x_i, then taking the partial derivative with respect to x_i on both sides yields:

$$\partial/\partial x_i(CPI(x_1, x_2, ..., x_n) = \partial/\partial x_i(CPI_{InfCache}(x_1, x_2, ..., x_n) \quad (6)$$

That is, the shape of the sensitivity curve (and therefore the "knee") would be identical for CPI and $CPI_{InfCache}$ in this case. The average cache miss rate is essentially independent of the internal queue size resources, being largely dependent on the inherent characteristics of the workload. Thus, our expectation is that from the viewpoint of relative accuracy, the method of separable components should perform well when compared against a full-blown simulation experiment.

3 Experimental Methodology

Figure 2 depicts the basic experimental set-up used in measuring the error characteristics of the modeling abstraction described in Section 2.

The baseline InfPrf CPI component is obtained by running the workload through an idealized simulator. Here, the cache and branch predictor are modeled as perfect: each load or store results in a cache hit and each branch prediction is done with 100% accuracy and no misprediction penalty. For this paper, these are the only two additional CPI adders considered: namely the finite cache effect adder and the branch prediction adder for penalties associated with mispredictions. For the finite cache effect, we consider both the L1 instruction and L1 data caches. We have computed the miss penalty by running a trace with one memory instruction. We ran this with our infinite L2/infinite L1 cache model and our infinite L2/finite L1 cache model. The miss penalty is computed as the difference in the number of cycles between these two simulations.

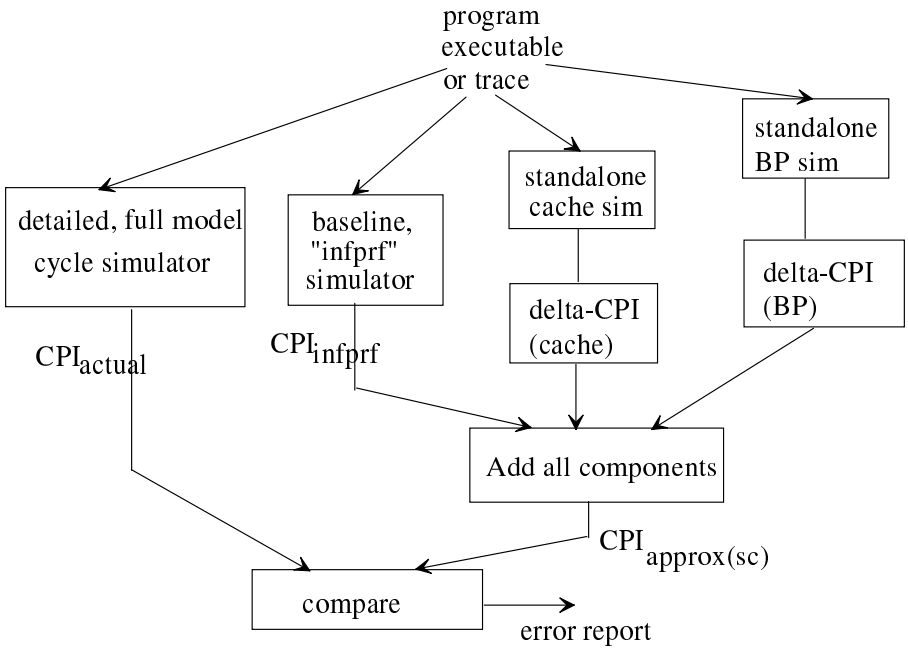

Figure 2: Experimental Setup.

3.1 Workloads Considered

In this paper, we report experimental results based on the SPECint95 and SPECfp benchmark suites. All workload traces are PowerPC-based. The SPEC95 traces were generated using the tracing facility called *Aria* within the MET toolkit [11]. The particular SPEC trace repository used in this study was created by using the full reference input set. However, sampling reduced the total trace length to 100 million instructions per benchmark program. The sampled traces have been validated against the full traces from which they are generated.

3.2 Superscalar Processor Model

Our simulation infrastructure is a PowerPC based toolkit [11] used to model future high-end, server-class PowerPC processors. The overall pipeline structure (as reported in [10]) is repeated here in Figure 3. This simulator has been validated against a more detailed, custom pre-RTL reference model to ensure accuracy [11]. The simulator is parameterized and the generic processor model is that of an out-of-order, superscalar processor. The modeled microarchitecture is similar in complexity to that of a current-generation microprocessor: e.g. the Compaq Alpha 21264.

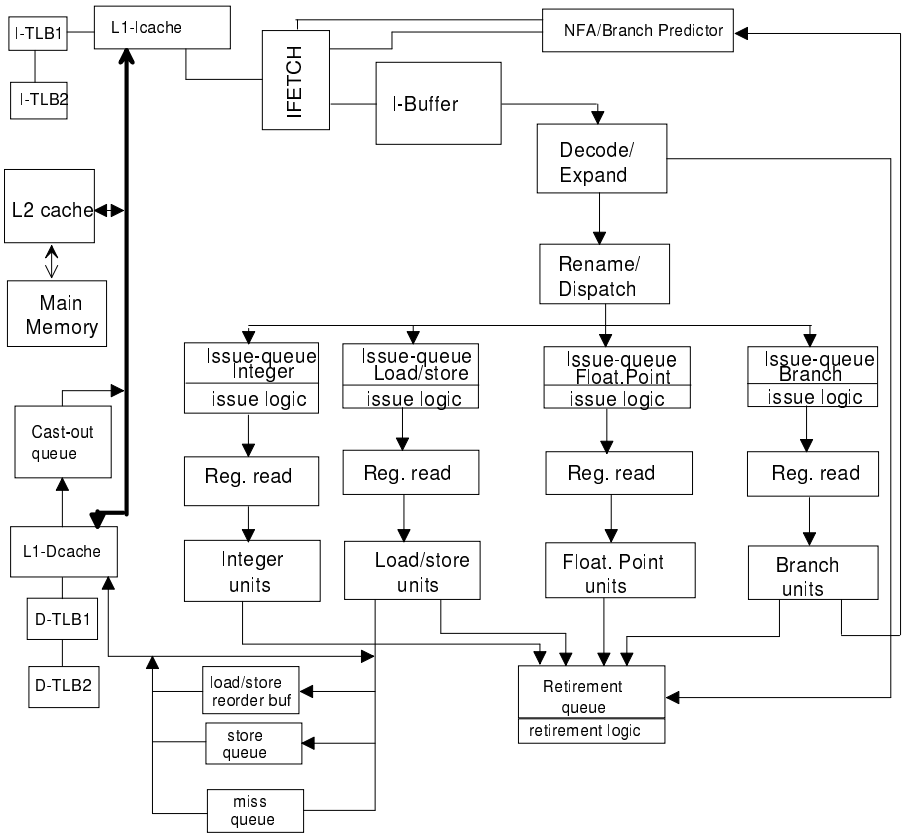

Figure 3: Processor Organization Modeled by the Turandot Simulator.

4 Separable Components Results: Overview

In this section we consider the absolute and relative accuracy of the separable components method for our two benchmark suites: SPECint95 and SPECfp95. To evaluate the separable components method, we consider two typical design constraint analysis problems that might occur in the design of superscalar microprocessors. First, we consider the impact of varying the number of rename registers available to the machine. Second, we consider varying the size of the instruction issue queues in the machine. The purpose of these examples is to give a concrete look at the separable components method and to present a qualitative overview of results. Section 5 will then go into more details about the data for particular applications and design choices.

4.1 Rename Registers Analysis

Figure 4 shows the performance impact of varying the number of rename registers in our superscalar processor simulator for the benchmark set described in Section 3. In this chart, the x-axis shows the number of general purpose rename registers varying from 48 to 112 in increments of 16. We have also simultaneously varied the number of floating point rename registers from 40 to 104 in increments of 16.

For each design point, we show three CPI values. First the true-CPI, which we arrive at with our detailed

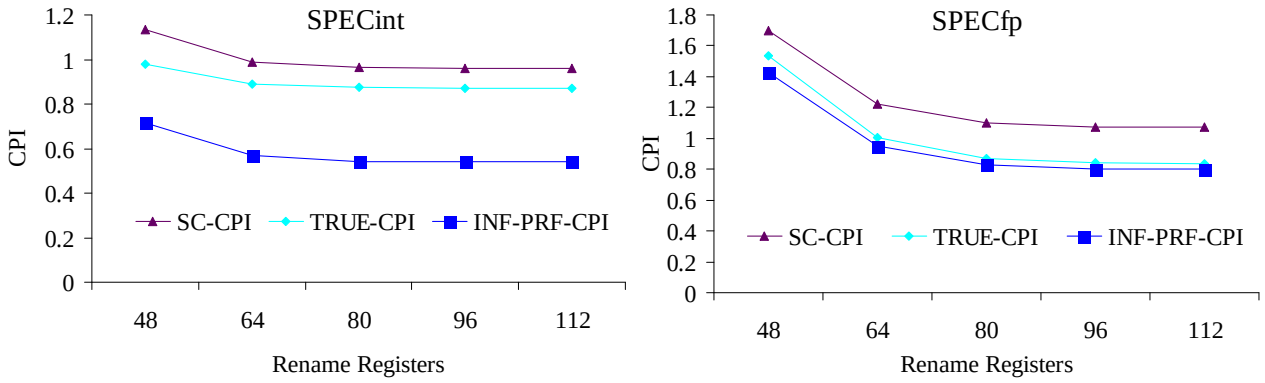

Figure 4: Detailed True-CPI (TRUE-CPI), Separable Components-CPI (SC-CPI), and Infinite Cache/Branch Predictor-CPI (INF-PRF-CPI) vs. Number of Rename Registers for SPECint and SPECfp.

performance simulator. Second, we show the separable-components-CPI (SC-CPI) which we derive using the method described in Section 2. This estimate provides an upper-bound on the true-CPI as explained previously. Finally, we show the Infinite Cache/Branch Predictor-CPI (INF-PRF-CPI) which provides a lower-bound to the true-CPI.

Based on the data in Figure 4, the absolute error introduced by the separable components method for SPECint ranged from 9% at 112 GPRs to 15% at 48 GPRs. For SPECfp the absolute error ranged from 10% at 48 GPRs to 27% at 112 GPRs. *Tomcatv* had the largest absolute error at nearly 75%. In Section 5.3 we will discuss in detail the cause of the large absolute error for this benchmark.

Despite the sometimes large absolute errors, one can look at the curves in a strictly qualitative manner and sense that the estimator curves will typically lead to the "right" data point being chosen. For example, for both the SPECint and SPECfp data shown in Figure 4, the knees of all three curves lie at 80 GPRs, even though the absolute accuracy of the bounding curves is not outstanding. Section 5.1 verifies the accuracy of these design point choices in a more quantitative way. In Section 5 we also report results of individual applications to discuss the effect of slope and sensitivity, and application characteristics that lead themselves to the separable components methodology.

4.2 Instruction Issue Queue Analysis

Figure 5 shows similar data as we explore the effects of varying the instruction queue size. We consider varying the fixed-point issue queue from 2 entries to 18 entries in increments of 4. We simultaneously vary the floating-point issue queue size from 2 entries to 10 entries in increments of 2.

We note that varying the instruction queue size introduced slightly less absolute error than varying the number of rename registers. The absolute error introduced by the separable components method for SPECint ranged from 9% with 18 queue entries to 13% with 2 queue entries. For SPECfp the absolute error ranged from 17% with 2 queue entries to 25% with 18 queue entries. Again, *tomcatv* had the most error, ranging

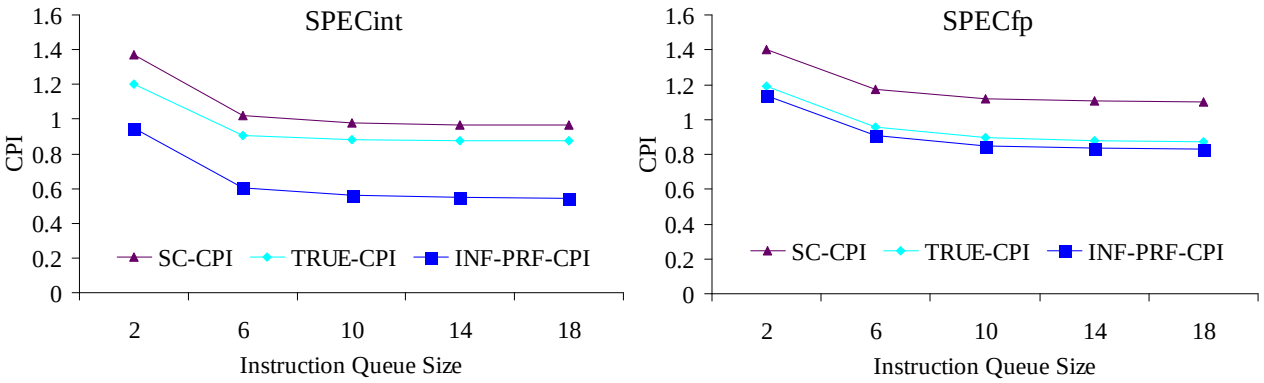

Figure 5: Detailed True-CPI (TRUE-CPI), Separable Components-CPI (SC-CPI), and Infinite Cache/Branch Predictor-CPI (INF-PRF-CPI) vs. Instruction Queue Size for SPECint and SPECfp.

from 45% to 70%.

5 Separable Components Results: Details

5.1 A Precise Methodology for Choosing Design Points

A fundamental premise of our separable components methodology is that even if its CPI estimates vary in their absolute accuracy, the estimates typically lead designers to choose the same design points they would choose with fully accurate data.

In Section 4 we showed that separable components gives good accuracy at the qualitative level despite sometimes large absolute errors. Here we elaborate on this qualitative assessment, by quantifying the accuracy of the separable components method in leading to correct design choices. We adopted the following methodology:

1. For each of the three curves, identify the minimum CPI point.
2. Assume that any design point within 5% of this minimum CPI point is a valid "correct" choice.
3. Identify the correct design point(s) based on the True CPI data.
4. Compare the set of design points chosen based on the estimated CPI to the correct design point(s) chosen when using the true CPI data.

Based on these steps, we have found that the separable components method has excellent accuracy in identifying correct design points, even when absolute accuracy in estimating CPI is poor. Across the 18 programs in the benchmark suite, 13 of them had exact agreement in what is the "correct" design point regarding the number of general purpose registers. Only five of the benchmarks had disagreement in the correct number of registers, and in these cases, the disagreement was only by a small amount, typically a

single 16-register increment in register set size. The results were similar when we considered the optimal size of the instruction queue. At 5% tolerance, 14 out of the 18 applications match true detailed simulation in their choice of design points. Disagreement typically occurred in cases where the CPI curve is fairly flat, and thus several options can end up looking fairly equivalent.

The amount of absolute error did not necessarily correlate with the applications which had agreement on design point choices. *Tomcatv* makes an accurate design point choice despite having 70% absolute error at the design point chosen. Many of the applications have correct design point choices despite absolute errors greater than 15%.

Interested in testing the limits of this methodology, we also repeated the same steps above, except using a more stringent definition of correct design choice: a point within 1% of the minimum CPI point. Using this very stringent definition, 13 of the 18 programs still matched in their choice of register set design points. 12 out of 18 programs precisely match in their choice of instruction queue sizes.

Overall, these results reinforce our hypothesis: a design methodology based on separable components can lead to excellent accuracy in choice of design points. This is true even in cases where *absolute* accuracy in CPI estimates is less than impressive.

5.2 Slope and Sensitivity

In addition to choosing specific design points, the bounded performance graphs also hold further information in the slope of the CPI curves. These slopes provide information on the sensitivity of the design around the chosen operating point. Where the CPI curve has a steep slope, it indicates that choosing a different design point will lead to markedly different performance. Where the CPI curve is shallow, choosing a different design point will not affect performance as much. The slope of the bounding curves, as well as the CPI difference between them, give indications of the slope of the true CPI curve.

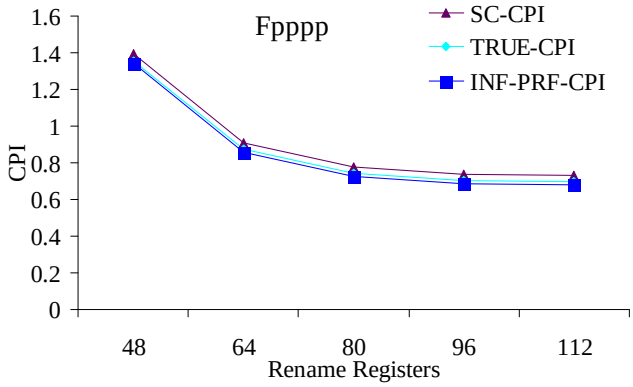

Figure 6: Detailed True-CPI (TRUE-CPI), Separable Components-CPI (SC-CPI), and Infinite Cache/Branch Predictor-CPI (INF-PRF-CPI) vs. Number of Rename Registers for *fpppp*.

Figure 6 shows a case (SPECfp benchmark *fpppp*) in which the lower bound and upper bound curves are packed quite closely together. In this case, it is clear that the slope of the True CPI curve must closely match that of the upper and lower bounds; where else could it go? In such situations, the user of the separable components methodology learns not only that GPR=80 is probably the correct design point, but also has a very good idea of what the design's sensitivity to other choices will be. If forced to choose fewer registers (eg. due to cycle time pressure) the designer can argue fairly forcefully that the degradation of performance with a choice of GPR=48 will be quite large.

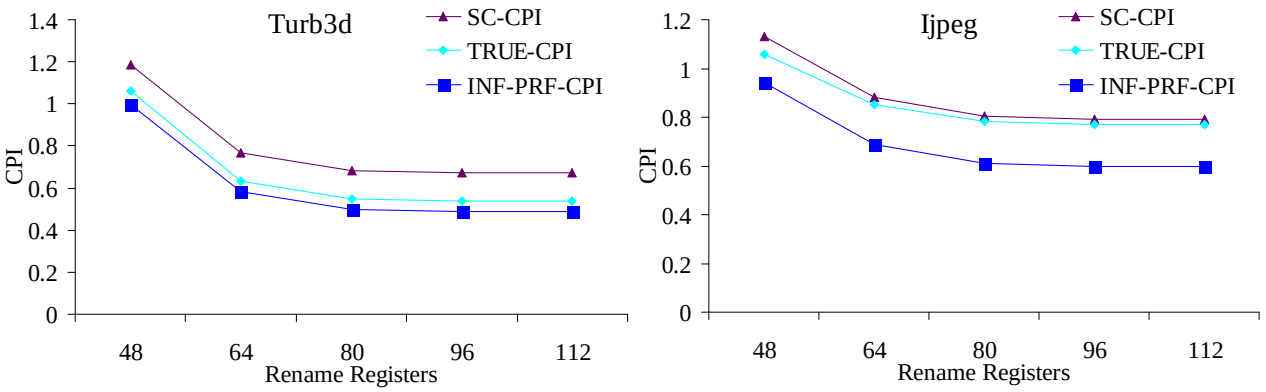

Figure 7: Detailed True-CPI (TRUE-CPI), Separable Components-CPI (SC-CPI), and Infinite Cache/Branch Predictor-CPI (INF-PRF-CPI) vs. Number of Rename Registers for *turb3d* and *ijpeg*.

Figure 7 shows two slightly different cases. In both of these graphs, the lower and upper bound curves are spaced more widely apart than in the previous *fpppp* example. This means that the bounding of slope cannot be done as tightly as in the *fpppp* case. The behavior in *turb3d* is a well-behaved case: although the slope cannot be bounded tightly, it happens that the slope of the true CPI curve matches well against the slope of the lower and upper bound curves. This means that the design sensitivities from the estimator curves will match those of the true CPI curves, and design choices can be made with good accuracy. For widely-spaced bounds, we have found this behavior to be the common case in our benchmarks.

The graph of *ijpeg* in Figure 7, however, is less well-behaved. Here, we see that the true CPI slope is somewhat different from that of the bounding curves. Particularly at lower values of GPRs, we see that the true CPI curve is shallower than either of the bounding curves.

Overall, the lesson from these examples is that the bounding curves give indications not only about likely design points, but also give information on the likely performance sensitivities around those design points. When bounding curves are close together, the designer has a good deal of confidence in the shape of the CPI curve. When bounding curves are more widely-spaced, the true curve's shape may deviate somewhat from the bounding curves, but is typically a fairly close match.

5.3 Application Characteristics that Lead to Separability

To first order, separable components' relative accuracy and its ability to bound true performance make it a useful methodology. In order to broaden its effectiveness, however, the concerns about absolute accuracy should also be addressed. To do so, it is important to understand the specific characteristics that make an application more or less amenable to "component separability". That is, what are the conditions under which we would expect this approximate model to yield good, average or poor absolute accuracy? And, can one envisage specific workload characteristics under which even relative accuracy may not be guaranteed?

Let us revisit equations 3, 4, 5 in Section 2. One may consider four different scenarios:

(a) A CPI adder's event frequency and event penalty are both very small; (b) Event frequency is small, but the penalty is quite large; (c) Event frequency is large, but the penalty is small; (d) Both the frequency and the penalty are quite large.

If the event frequencies or penalties are all small enough that the frequency*penalty products are negligible, then the CPI adder will be small, the upper and lower bound curves will be close together, and we should expect good absolute accuracy. In the application traces analyzed in our study, *vortex* and *fpppp* fall into this category. For example, with *vortex*, the measured data cache miss rate is only 7 misses per 1000 committed instructions and the branch misprediction rate is only 1 mispredict per 1000 committed instructions.

Condition (d) is clearly the case that is potentially most troublesome. If both frequency and penalty are large enough, then the CPI adder will be large, and thus there will be a sizable spacing between the lower bound, "InfPrf" curve, and the upper bound SC-CPI curve which includes the CPI adder for this event. When this happens, there are two main possible outcomes in terms of absolute accuracy.

The first possibility is that the program's average effective event penalty turns out to be quite close to the worst-case assumed event penalty that is used in the bounding analysis. If this is the case, then the upper bound curve will turn out to have fairly good absolute accuracy as an estimator of program performance. Among the SPEC benchmarks, *gcc* is a good example of this case. In this program, most of the data references are loads, and due to load-store dependences, a non-blocking cache does not reduce the effective cache miss penalty to much below the assumed worst-case value. Furthermore, *gcc* also has substantial branch misprediction rates and so the variance in context-dependent misprediction penalty is minimized. Both of these effects mean that *gcc* sees performance that is quite close to its estimated upper bound from the separable CPI adders; as a result, the SC-CPI curve offers a good estimate of performance.

A second possibility is that the wide gap between upper and lower bound performance curves will be due to worst-case modeling assumptions that are not borne out in real life. For example, consider *tomcatv*. The measured cache and branch misprediction rates for this trace are 70 data cache misses and 0.6 branch mispredicts per 1000 committed instructions. Furthermore, this trace is one with significant store misses. The load and store miss rates are: 33 load misses and 37 store misses per 1000 instructions. As mentioned previously in Section 2.1, dominating store misses have the effect of drastically reducing the effective cache miss penalty, because they can typically be handled by the store buffer in much less than the worst-case

assumed penalty. Thus, in this case the assumed event penalty for cache misses (9 cycles) turns out to be a gross over-estimate. In fact, analysis of the simulation run shows that the effective cache MP, due to the phenomena of overlapped stores in this case, was much lower: 2 cycles. This dramatic difference between the actual application behavior and the worst-case behavior assumed in the upper-bound leads to the 70+% absolute errors seen by *tomcatv* using this method. (We reiterate, however, that the good relative accuracy seen by *tomcatv* still leads to fairly good design point choices in general.)

5.3.1 Improving the Model

One way to improve the model to handle cases like that exemplified by *tomcatv*, is to consider only the load miss rate, while retaining the nominal MP value of 9 cycles. That is, in workloads where stores dominate and the store addresses tend to be independent of the load addresses, the cache miss rate should only consider the load access stream. This may result in a slight under-estimation of the corresponding cache miss CPI adder, but the absolute accuracy of net CPI projections would be quite good. If the store misses do not completely dominate over load misses, neglecting the store misses may still result in an over-estimation of the net CPI; nonetheless, the accuracy gap is reduced significantly.

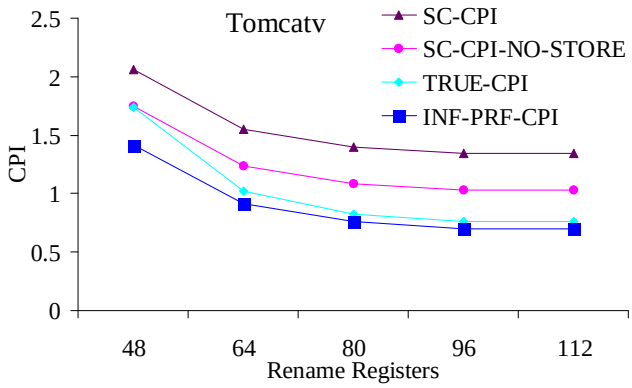

Figure 8: Detailed True-CPI (TRUE-CPI), Separable Components-CPI (SC-CPI), and Infinite Cache/Branch Predictor-CPI (INF-PRF-CPI) vs. Number of Rename Registers for *tomcatv*.

Applying this approach to *tomcatv*, Figure 8 shows the absolute accuracy comparisons with and without the store misses included in the cache MR value. The data shown is for the rename buffer sensitivity experiment. The absolute error dropped from 18% at 48 entries and 75% at 112 entries to 1% and 34% respectively.

In keeping with work done in prior bounds modeling research [1, 8], it is useful to understand the sources of absolute inaccuracy (between SC-CPI and True-CPI) in full detail. For *tomcatv*, store misses account for a large portion of the accuracy gap. Other factors that explain the remaining mismatch can also be described by analyzing the detailed run statistics. We will include more such analysis in the revised, final version of

this paper.

6 Related Work

The use of bounds models and limit-study simulators to understand the fundamental performance limits and tradeoffs is quite pervasive. In [8] the authors describe the research in static (i.e. source-code based) bounds modeling that has helped identify fundamental performance bottlenecks and application tuning opportunities. The work reported in [1] applies similar techniques to real-life super scalar machine. Wellman [20] uses dynamic (trace-based) bounding methods to characterize machine performance. Classical limit-study models (e.g. Wall [19]) have been used to quantify fundamental ILP limits in conventional superscalar processing. In terms of workload-level modeling abstractions, the classical trace sampling work of Laha et al. [6] has been followed by many others (e.g. [7, 12, 3]). The work by Iyengar et al. [5] proposes a trace regeneration methodology that results in achieving drastic reduction in simulation time, with acceptable loss of accuracy. The use of microbenchmarks to understand machine performance is also quite widespread. The work by Saavedra et al. [15] was a pioneering contribution in that regard.

In the work above, gaps in absolute accuracy were empirically quantified through experimentation. However, to our knowledge, a systematic analysis of absolute and relative accuracy was not addressed simultaneously in any of the related work above.

7 Conclusion

In this paper, we have examined the method of separable components in detail. Separable components provides a methodology for developing performance simulators far more quickly than conventional simulators as well as providing significant runtime speedup. This methodology has been used extensively in the modeling of simpler processors where each type of pipeline stall condition inhibits further processing until the stall condition is resolved. In this paper, we consider the accuracy implications of using this method on an aggressive, out-of-order processor with non-blocking caches.

We conducted a series of simulation-based experiments to measure the error characteristics of a commonly used modeling abstraction. The goal of this study was to measure the CPI errors across a suite of diverse workloads. For workloads where the error margins where found to be large, we attempted to understand why the gap was so pronounced by looking at the workload characteristics. Also, we studied the relative accuracy characteristics of the "error-prone workloads" to see if the SC abstraction allows us to make sound tradeoff decisions. In general, our results seem to indicated that in spite of absolute accuracy problems, most of the "error-prone" workloads exhibited acceptable relative accuracy characteristics. Furthermore, this method always provides a solid upper and lower-bound on performance. We have also explained how the shape of the curves can give hints about the sensitivity of performance to the particular design point under study.

We see two major applications of the separable components method. First, the results reported in this study point to a modeling methodology, where the traces are characterized once, to collect statistics related

to various types of pipeline stall conditions. Each trace can then be "marked up" by adding bit qualifiers to color various instruction classes that give rise to stalls. For example, each load/store instruction can be "attributed" to indicate whether it is a cache "hit" or a "miss." Similarly, conditional branches could be tagged to indicate if they will be mispredicted or not. The simulator itself, which is run many times to run tuning experiments, could then be simplified by not simulating the exact cache simulation piece or the branch prediction/misprediction scheme. Instead, when a load or store is encountered during simulation, the appropriate latency is added to the execution pipe, depending on whether it was tagged to be a hit or a miss. Similarly, depending on whether a particular branch was tagged to be a mispredicted one or not, the appropriate average processing delay could be added. Such trace preprocessing methods (see [14]) have the potential of significant simulation speedup without giving up a lot of accuracy.

Finally, we anticipate that tools used to rapidly explore the design space can be quickly built up using the method of separable components, allowing the design space to be pruned very early in the design process. Conventional cycle-accurate performance simulators will eventually be built in any design project, and the tool based on separable components will be a vital part of the validation methodology for the more detailed conventional simulator.

References

[1] P. Bose, S. Kim, F. O'Connell, and W. Ciarfella. Bounds modeling and compiler optimizations for superscalar performance tuning. *Journal of Systems Architecture*, 45:1111–1137, 1999.

[2] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. Tech. Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept., July 1996.

[3] P. Dubey and R. Nair. Profile-driven sampled trace generation. In *Proceedings of the International Conference on Computer Design*, October 1996.

[4] P. Emma. Understanding some simple processor performance limits. *IBM Journal of Research and Development*, 41(3):215–232, May 1997.

[5] V. Iyengar et al. Representative traces for processor models with infinite cache. In *Proc. of the 2nd Int'l Symp. on High-Performance Computer Architecture*, Feb. 1996.

[6] S. Laha, J. H. Patel, and R. K. Iyer. Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems. *IEEE Trans. on Computers*, pages 1325–1336, Nov. 1988.

[7] G. Lauterbach. Accelerating architectural simulation by parallel execution of trace samples. In *Proceedings of the 27th Hawaii Int'l Conference on System Science*, 1994.

[8] W. Mangione-Smith, T. Shieh, S. Abraham, and E. Davidson. Approaching a machine application-bound in delivered performance on scientific code. In *Proc. IEEE*, volume 81, pages 1166–1178, Aug. 1993.

[9] M. Martonosi, A. Gupta, and T. Anderson. Effectiveness of Trace Sampling for Performance Debugging Tools. In *Proc. ACM SIGMETRICS Conf. on Meas. and Modeling of Computer Systems*, May 1993.

[10] M. Moudgill, P. Bose, and J. Moreno. Validation of Turandot, a fast processor model for microarchitecture exploration. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 451–457, Feb. 1999.

[11] M. Moudgill, J. Wellman, and J. Moreno. Environment for PowerPC microarchitecture exploration. In *IEEE Micro*, volume 19, pages 9–14, May/June 1999.

[12] A.-T. Nguyen et al. Accuracy and speedup of parallel trace-driven architectural simulation. In *Proceediungs of the IEEE Int'l. Parallel Processing Symposium (IPPS)*, 1997.

[13] M. Oskin, F. Chong, and M. Farrens. Hls: Combining statistical and symbolic simulation to guide microprocessor designs. In *Proc. of the 27th Int'l Symp. on Computer Architecture*, June 2000.

[14] P. Bose. Performance evaluation of processor operation using trace pre-processing. US Patent 6,059,835, May 2000.

[15] R. Saavedra, R. Gaines, and M. Carlton. Micro benchmark analysis of the KSR1. In *Proc. Supercomputing '93*, Nov. 1993.

[16] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Transactions on Computers*, 48(11):1260–810, Nov. 1999.

[17] K. Skadron, M. Martonosi, and D. Clark. Speculative updates of local and global branch history: A quantitative analysis. *Journal of Instruction-Level Parallelism*, 2, June 2000. http://www.jilp.org/vol2.

[18] D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood. Analytic evaluation of shared-memory systems with ILP processors. In *Proc. of the 25th Int'l Symp. on Computer Architecture*, June 1998.

[19] D. W. Wall. Limits of Instruction-Level Parallelism. In *Proc. ASPLOS-IV*, pages 176–88, Apr. 1991.

[20] J. Wellman. *Processor modeling and evaluation techniques for early design stage performance comparison.* PhD thesis, University of Michigan, 1996.