# Micro-architecture Performance Estimation by Formula

Lucanus J. Simonson[1] and Lei He[2]

[1] Intel Corporation, Santa Clara CA 95052, USA
[2] University of California, Los Angeles CA 90095, USA

**Abstract.** An analytical performance model for out of order issue superscalar micro-processors is presented. This model quantifies the performance impacts of micro-architecture design options including memory hierarchy, branch prediction, issue width and changes in pipeline depth at all pipeline stages. The model requires a minimal number of cycle accurate and trace driven simulations to calibrate and once calibrated estimates performance by formula. The model estimates the performance of arbitrary micro-architecture configurations with an average error of 6.4%. During early design stages when cycle accurate simulation is prohibitive an analytical model can provide guidance to designers to increase design quality and reduce design effort. This allows the design of an embedded processor to be rapidly tuned to its application by reducing the cost of exploring the design space.

## 1 Introduction

During early planning stages of micro-processor design a clear understanding of the impact various design decisions will have on performance is critical. Cycle accurate simulation is often used to collect performance information, but is too time consuming to be well suited to exploration of a large number of competing design options and does not lead directly to an understanding of why a change in the micro-architecture impacts performance. Because performing a large number of cycle accurate simulations is prohibitively a faster method of producing performance information is needed. We present a super-scalar, out of order issue microprocessor performance model that estimates performance by formula and requires a minimal number of cycle accurate and trace driven simulations to calibrate. This model allows greater freedom to explore micro-architecture options and pipeline strategies during early design, and provides the capability to more easily tune a processor to its application.

Our model is capable of estimating performance for different combinations of micro-architecture options and pipeline depth at all stages of instruction execution. Micro-architecture choices include issue width, resource contention, memory hierarchy, branch predictor strategy and instruction prefetch. Combining these design elements into a single performance model enables designers and design tools to optimize all of these parameters simultaneously with minimal simulation time. The model is intuitive and reasonably accurate. It requires constant runtime once built up and provides insight into the causes of performance loss and the ways in which they interact in a realistic processor. This work also establishes the methodology for generating a similar model for other base micro-architectures such as those used in embedded processors or under development in industry. Up front simulation costs are minimized by decoupling all of the various factors that contribute to performance loss and recombining them analytically. Empirical cycle-accurate sensitivity analysis of pipeline depth and trace driven simulation of the behavior of each micro-architecture design variable in isolation are used to provide the inputs to the model.

Related to this paper, many approaches have been proposed to estimate performance. A theoretical method for analyzing in-order pipelines is presented in [1], however the work does not apply to out-of-order issue architectures. Both in-order and out-of-order front-end pipeline depth is analyzed in [2], though the backend was not. While exploring the impact of increasing pipeline depth on processor performance in the Pentium processor [3], an empirical performance model was developed to estimate performance as a function of pipeline depth, but no micro-architecture changes were considered.

A first-order superscalar processor model [4] provides a formula based estimate of performance. It assumes an idealized micro-architecture free of resource contention and capable of sustained throughput equal to issue width and estimates the performance penalties due to branch mispredictions and instruction and data cache misses from a trace-driven simulation. Building upon [4], we consider a more realistic micro-architecture that includes resource contention and instruction prefetch. We also decouple the behavior of the various caches and branch predictor so that a trace-driven simulation is not required for each unique combination of cache and branch predictor settings. In addition, we consider the performance impact of varying the backend pipeline depth and combine that with the performance impact of the frontend analytically by employing a novel probabilistic performance loss event overlap model.

The rest of the paper is organized as follows. In Section 2 we present background information on architecture design options considered, simulation engine used and our methodology for deriving the model. In Section 3 we present our analytical performance model. We experimentally verify the correctness and accuracy of our model in Section 4 and conclude the paper in Section 5.

## 2 Background

### 2.1 Micro-architecture Design Space

The Micro-Architecture considered in this work is a super-scalar out of order issue CPU that includes the seven modules shown in Table 1 with the options considered for each. Pipeline depth is simulated by clock cycle latency between the micro-architecture modules defined in Table 1. These interconnects are listed in Table 2. The symbol used to represent the latency of each interconnect is listed as well as a qualitative description of the type of performance degradation caused by latency on each.

### 2.2 Methodology

All simulations were performed using modified version of SimpleScalar 2.0 with PISA instruction set architecture in truncated runs with a fastforward period of forty million instructions and a sample period of twenty million instructions. Six benchmarks

**Table 1.** Micro-architecture Design Freedoms and Options

| Design Freedom | Options |
|---|---|
| Issue Width | 2, 4, 8 |
| Integer ALU Number | Equal to issue width, 3/4 of issue width |
| Other Arithmetic Unit Number | 1/4 Integer ALU, 1/2 Integer ALU |
| Branch Predictor Size/Strategy | bimod 1K, BTB 128; bimod 2K, BTB 256; combined bimod 2K, 2-level 1K, BTB 512; combined bimod 4K, 2-level 2K, BTB 1K |
| Instruction Level 1 Cache | 8KB Direct Mapped, 16KB Direct Mapped, 32KB 2 Way Associative, 64KB 4 Way Associative |
| Data Level 1 Cache | 8KB Direct Mapped, 16KB Direct Mapped, 32KB 2 Way Associative, 64KB 4 Way Associative |
| Unified L2 Cache | 128KB 2 Way Associative, 256KB 4 Way Associative, 512KB 4 Way Associative, 1MB 8 Way Associative |

**Table 2.** Interconnect Pipeline Design Freedoms

| Symbol | Interconnect | Performance Impact |
|---|---|---|
| $L_{IL1/L2}$ | IL1 Cache to L2 Cache | Increased IL1 Cache Miss Penalty |
| $L_{DL1/L2}$ | DL1 Cache to L2 Cache | Increased DL1 Cache Miss Penalty |
| $L_{fetch}$ | Fetch Unit to IL1 Cache | Increased Branch Misprediction Penalty Prefetch Penalty |
| $L_{dispatch}$ | Fetch Unit to Dispatch Unit | Increased Branch Misprediction Penalty |
| $L_{issue}$ | Dispatch Unit to Issue Unit | Increased Branch Misprediction Penalty |
| $L_{DL1}$ | Issue Unit to DL1 Cache | Stalls on data load dependencies |
| $L_{IALU}$ | Issue Unit to each I-ALU | Stalls on integer dependencies Increased Branch Misprediction Penalty |
| $L_{IMult}$ | Issue Unit to each I-Multiplier | Stalls on multiply dependencies |
| $L_{FALU}$ | Issue Unit to each FP-ALU | Stalls on floating point dependencies |
| $L_{FMult}$ | Issue Unit to each FP-Multiplier | Stalls on floating point dependencies |

(mcf, equake, art, mesa, parser and bzip2) from the SPEC2000 suite were evaluated to produce all experimental results presented. These include a mix of floating point and integer benchmarks and were chosen to represent a range of real world application behaviors. Performance of an architecture is summarized as the arithmetic mean of CPI for the six benchmarks.

The model was developed by performing a study of the performance impact of each design variable in isolation while holding all other design variables constant. The performance impact of the design variable was graphed and, if possible, a mathematical formula derived to fit the observed behavior. The formulas and empirical constants obtained by studying each design variable in isolation are combined incrementally, grouping variables by category, developing mathematical expressions for the interaction between the behavior of each variable with other variables in its category and finally between categories. This led to the division into front and backend in the model and the grouping of terms.

Deriving the expressions to describe the interaction between design variables in terms of their performance impact relied upon insight into the anatomy of a performance loss event. From [4] we know that some performance loss events are not inter-related. We verified these findings and proceeded to the new factors considered by our model. For each interaction we devised a hypothetical expression based upon insight into the micro-architecture. We tested each hypothesis by performing an experiment to isolate the interaction and measure by cycle accurate simulation.

## 3 Analytic CPI Model

We measure performance in terms of cycles per instruction (CPI) which we define as the average number of clock cycles per instruction issued by the processor on the correct execution path. CPI is proportional to execution time and is convenient for our purposes because the delay caused by some performance loss event directly adds to execution time, can simply be averaged over the number of instructions and added to the CPI. Our approach to performance modeling is to count the number of performance loss events and quantify their delay penalties. Should two delay penalties be incurred at the same time the performance overhead should not be double counted. We correct for overlapping performance loss events to obtain an accurate performance estimation formula.

### 3.1 Model with Interconnect Pipeline

The CPI of a micro-architecture is a combination of performance loss due to miss events, data dependency stalls and the baseline performance of the micro-architecture with no extra pipelining in the absence of miss events. Baseline performance is a function of the issue width and resource constraints of the micro architecture combined with the amount of instruction level parallelism in the benchmark. The miss events we consider are branch mispredictions, level one instruction cache misses and level two instruction and data cache misses. Level one data cache misses are modeled the same as the latencies of arithmetic units that result in data dependency stalls.

We quantify the performance impact of latency in each of the interconnects from Table 2 in terms of contribution to miss penalty and average duration of data dependency stall, then combine their performance impacts to estimate system performance with consideration of pipelined interconnect. Arithmetic units of the same type are grouped together into a single module and have identical interconnect latency. We divide the interconnects into frontend and backend interconnects. Frontend interconnects are those that contribute latency to the pipeline prior to the issue stage. The equation for the micro-architecture CPI model with consideration of pipelined interconnect is given in (1).

$$CPI = CPI_{ideal} + CPI_{IL1} + CPI_{L2} + CPI_{Front} + CPI_{Back} \tag{1}$$

### 3.2 Cache CPI Overhead

Miss rates for one type of level one cache are clearly independent of the other level one cache configuration as well as the level two cache configuration. Given that the level two cache is sufficiently larger than the level one caches, our experiments show its miss rates are roughly independent of level one cache configuration. For this reason the miss rates for each cache option are measured independent of the configuration options chosen for the other caches reducing the number of trace driven simulations required to build up the model.

In equation (1) performance loos due to instruction level one cache misses, $CPI_{IL1}$, is defined as the access latency of the L2 cache, $L_{L2(access)}$, plus the interconnect latency between the IL1 and L2 cache, $L_{IL1/L2}$, multiplied by the miss rate of the instruction level one cache. Instruction cache miss penalty is equal to the latency of the next higher level of memory hierarchy [4].

$$CPI_{IL1} = MissRate_{IL1}(L_{L2(access)} + L_{IL1/L2}) \tag{2}$$

$CPI_{L2}$ is the performance loss due to level two cache misses, broken down into instruction and data cache miss rates, multiplied by the latency to main memory, $L_{MM}$. The $F_{OverlapL2(data)}$ term in (3) is the expected value for the size of a group of level two data cache misses that all occur within the issue window size number of instructions of the previous level two data cache miss. This factor accounts for the overlapping of L2 data cache miss performance loss as described in [4]. We assume the miss penalty for level two cache to be the latency of a main memory access. Unlike [4] we do not calculate or use an overlap factor between level two data cache misses and other miss events because we assume that fetch is blocked by the time the L2 data miss penalty is incurred, preventing such overlap from occurring.

$$CPI_{L2} = (MissRate_{L2(inst)} + \frac{MissRate_{L2(data)}}{F_{overlapL2(data)}})L_{MM} \tag{3}$$

## 3.3 Frontend CPI Overhead

Two sources of performance loss contribute to frontend CPI, branch misprediction and prefetch overhead. When latency is added between the level one instruction cache and the fetch logic the branch predictor does not have the opportunity to decide whether a branch is taken until several clock cycles after it has been read from the cache. Prefetch proceeds to fetch contiguous blocks in memory until a branch predicted as taken reaches the fetch unit. The prefetch pipeline must then be flushed and fetching resumes at the target address. The formula for $CPI_{Front}$ is given in (4).

$$CPI_{Front} = \alpha(CPI_{BPred(pipe)} + CPI_{Prefetch}) + CPI_{BPred} \tag{4}$$

$CPI_{BPred}$ is the performance loss due to branch misprediction

$$CPI_{BPred} = MissRate_{BPred}Penalty_{Intrinsic} \tag{5}$$

where the intrinsic branch misprediction penalty, $Penalty_{Intrinsic}$, is measured for a given benchmark by cycle accurate simulation for a single micro-architecture by dividing the difference between $CPI_{ideal}$ with and without perfect branch prediction by the branch misprediction rate, $MissRate_{BPred}$. In our experiments we observed that the branch misprediction penalty had minimal dependence upon the branch prediction option chosen or the issue width of the microprocessor and is instead a function of pipeline depth and performance loss overlap. For this reason we use a single measurement of intrinsic, baseline branch misprediction penalty for all micro-architecture configurations. The branch misprediction rate is measured by trace driven simulation of each of the branch predictor options in Table 1. The $CPI_{BPred(pipe)}$ term in (4) is defined as

$$CPI_{BPred(pipe)} = MissRate_{BPred} Penalty_{BPred(pipe)} \tag{6}$$

where $Penalty_{BPred(pipe)}$ is defined as

$$Penalty_{Bpred(pipe)} = 2(L_{fetch} + L_{dispatch} + L_{issue} + L_{IALU}) \tag{7}$$

such that pipeline stages added anywhere in the integer pipeline contribute two cycles of branch misprediction penalty. The justification for this is that when a branch is being issued the instruction it depends upon (which we assume to be an integer instruction) has often not yet committed, so adding one cycle of latency anywhere in the integer pipeline will add one cycle of branch misprediction penalty due to data dependency delay of branch issue. If there is no data dependency then adding one cycle of latency to the frontend will add one cycle of branch misprediction penalty due to delayed resolution time of the branch. In both cases a cycle of latency in the backend will add one cycle of branch misprediction penalty due to delayed resolution time of the branch. Finally, adding one cycle of latency specifically to the frontend pipeline will add an additional cycle of branch misprediction penalty because the frontend pipeline takes longer to refill after it has been flushed due to a branch misprediction.

The $CPI_{Prefetch}$ quantity in (4) is defined as

$$CPI_{Prefetch} = \frac{HitRate_{BPred} Branches_{Taken} L_{fetch}}{Instructions} \tag{8}$$

where the ratio of $Branches_{Taken}$ to $Instructions$ is the probability that a given instruction is a taken branch. Multiplying by the branch prediction hit rate, $HitRate_{BPred}$, factors out the prefetch flushes that overlap performance loss due to branch mispredictions for those same branches.

## 3.4 Frontend Overlap Correction

Because performance loss in the frontend may overlap other performance loss events we introduce in (4) a correction factor, α, based upon the approximation that the timing of all potentially overlapping performance loss events are independent and random. The equation for this correction factor is

$$\alpha = \frac{CPI_{unit}}{CPI_{ideal} + CPI_{Bpred} + CPI_{BPred(pipe)} + CPI_{back}} \tag{9}$$

where $CPI_{unit}$ is similar to $CPI_{ideal}$ but with an initial latency of one on all arithmetic operations including the typically long latency floating point divide and root operations. The $CPI_{unit}$ quantity can be thought of as the time the processor spends performing useful work. The α overlap factor is the probability that performance loss events due to

**Table 3.** Fitness of α overlap factor

| IL1 Fetch Latency | 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| Error | 2.4% | 1.7% | 4.0% | 11.6% |

added frontend pipeline stages do not overlap some other source of performance loss, which is the probability that they happen during the time that the processor is performing useful work. Branch misprediction can overlap other branch misprediction events as well as data dependency stalls. It cannot overlap prefetch performance loss or instruction cache miss performance loss because during these periods no branches can be fetched. When fetch becomes blocked due to extremely long latency data dependency stalls due to level two data cache misses no overlap can occur.

To demonstrate the fitness of modeling the probability of overlap between performance loss events as independent random variables with the α overlap factor we measure cycle accurate CPI for our set of benchmarks while letting the interconnect latency, $L_{fetch}$ vary from one to eight while other micro-architecture freedoms are held constant and compare to cycle accurate simulation. As presented in Table 3 the error for this experiment is very low at small latencies and only 11.6% at extreme fetch latency.

## 3.5 Backend CPI Overhead

Performance loss for the interconnects in the back end is due to data dependency stalls. The stall incurred by a dependent instruction is determined by the maximum commit time of the instructions it is dependent upon. It does not matter which of the backend interconnects contribute latency to the maximum commit time because the effect is the same. Backend interconnects are the same way and can be modeled as a group by summing their individual models.

$$CPI_{Back} = \sum_{x \varepsilon \{IALU, IMult, FALU, FMult, DL1\}} CPI_x \tag{10}$$

Because performance loss in the back end is linear CPI overhead of individual interconnects can be accurately modelled as in (11) where $C_x$ is an empirical constant obtained in a similar way to $Penalty_{BPred}$ by calculating the slope of a line between two cycle accurate CPI measurements while varying $L_x$.

$$CPI_x = C_x L_x \tag{11}$$

In the case of data loads the latency is increased by the latency to go to the level two cache with probability equal to the level one data cache miss rate.

$$CPI_{DL1} = C_{DL1}(L_{DL1} + MissRate_{DL1}(L_{DL1/L2} + L_{L2(access)})) \tag{12}$$

We verify our backend pipeline model by checking it against cycle accurate simulation. In this experiment we use the baseline architecture used to calculate $CPI_{ideal}$ but let the latencies to the arithmetic units vary uniformly by factors of two from one to sixteen. Maximum estimation error for this experiment was around 2% showing that the backend model based upon linear performance penalty as a function of latency is highly accurate.

## 4 Experiments

We verify the correctness and accuracy of our performance model in an experiment where thirty-two random configurations chosen from the options in Table 1 and latencies ranging from zero to nine inclusive from the interconnects in Table 2 are measured by cycle accurate simulation.

The cycle accurate CPI for each of the thirty two configurations is plotted as the independent variable while the model estimate is plotted as the dependent variable. As we can see the data points line up nicely along the $y = x$ line. The model shows good fidelity, with average error for this experiment of 6.4% and maximum error of 18.4%. We compare this to the error reported in [4] of 4.4% on average and 12% in the worst case in which performance is estimated for a subset of five out of the sixteen design variables we consider here. We succeed in decoupling the behavior of the various caches and the pipeline depth as well as adding in consideration of pipeline depth in the back end without paying too high a price in accuracy over the state of the art.

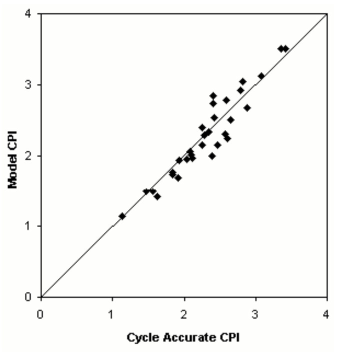

**Fig. 1.** Lumped Arithmetic Module Model Fidelity

**Table 4.** Average absolute error of individual benchmarks

| art | bzip2 | equake | mcf | mesa | parser | average |
|---|---|---|---|---|---|---|
| 7.3% | 10.1% | 8.8% | 8.8% | 7.2% | 9.5% | 8.6% |

In Table 4 the average absolute error is broken down by benchmark. One of the goals of the model is to abstract away the contribution of instruction stream to performance. Based upon the assumption that this was possible, cycle accurate simulation of only a single benchmark, equake, from the set of six mentioned in section 2.2 was used to derive the form of the model as discussed in section 2.2. This was necessary during the initial micro-architecture study because of the large number of cycle accurate simulations required. By comparing the accuracy of the full model against other benchmarks in Table 4 we validate the ability of the model to abstract away the contribution of instruction stream to performance.

The average absolute error reported for Fig. 1 differs from that in Table 4 because it is calculated by summing the CPI of the benchmarks before calculating absolute error, in effect treating the six benchmarks sampled as a single benchmark six times as long. Significantly, the difference is small because the model tends to underestimate CPI, as can be observed by comparing the number of data points below and above the $y = x$ line in Fig. 1. This tendency is a contributing factor to the model's good fidelity.

## 5 Conclusions and Discussions

We have developed an accurate analytical performance model for superscalar out of order issue microprocessors. It models changes in cache hierarchy, branch predictor size and strategy, issue width and pipeline depth at all stages of execution. The model provides performance estimates accurate to within 6.4% on average and 18.4% in the worst case for a random set of micro-architecture design options and pipeline depths. This compares to 4.4% on average and 12% in the worst case as reported in [4], which serves as the basis for our work but considers only a third of the micro-architecture design freedoms in our model. Additionally our model provides further insight into what the causes of performance loss are in modern micro-processors and how these sources of performance loss interact with each other to determine the overall performance of a micro-processor.

The model requires that two cycle accurate simulations be performed for each issue width under consideration, an additional cycle accurate simulation to obtain the intrinsic branch misprediction penalty of a benchmark and one cycle accurate simulation for each type of arithmetic unit employed by the micro-architecture. The model also requires a single trace driven simulation for each cache configuration option and each branch predictor configuration option under consideration. The relatively low cost of building up the model is the key feature that makes it practical. The entire range of possible combinations of micro-architecture options and pipeline depths can be explored with a minimal amount of simulation. This provides the basis for a speed/accuracy trade-off giving designers the option to choose between the analytical model and cycle accurate simulation to suite their needs.

Modular design of a processor targeting a specific application could benefit from rapid exploration of the micro-architecture design space, considering all of the discrete options in cache size, branch prediction, ALU type and issue width to minimize area while meeting performance constraints. This type of model provides the capability to perform rapid "what if" analysis of micro-architecture design choices, enabling designers to immediately see the impact a change can be expected to have on system performance or enabling automated design tools to execute interactive refinement optimization algorithms such as floorplanning with consideration of performance.

Future work includes the following objectives: developing an analytical power model based upon the analytical performance model, applying the analytical performance model to iterative automated micro-architecture and floorplanning co-optimization such as in [5], modeling the interaction between performance loss events in micro-architectures utilizing different branch misprediction recovery mechanisms, extending the analytical performance model to multi-core processors and eliminating the need for cycle accurate simulation in building up the analytical performance model by replacing empirical constants with formulas based upon instruction stream statistics.

## References

1. Emma, P.G., Davidson, E.S.: Characterization of branch and data dependencies on programs for evalutating pipeline performance. IEEE Trans. on Computers **36** (1987) 859–875
2. Hartstein, A., Puzak, T.R.: The optimum pipeline depth for a microprocessor. In: International Symposium on Computer Architecture. (2002)
3. Sprangle, E., Carmean, D.: Increasing processor performance by implementing deeper pipelines. In: International Symposium on Computer Architecture. (2002)
4. Karkhanis, T., Smith, J.E.: A first-order superscalar processor model. In: International Symposium on Computer Architecture. (2004)
5. C. Long, C., Simonson, L., Liao, W., He, L.: Floorplanning optimization with trajectory piecewise-linear model for ipelined interconnects. In: DAC. (2004)