Optimum Instruction-level Parallelism (ILP) for Superscalar and VLIW Processors

Patrick Hung and Michael J. Flynn

Technical Report No. CSL-TR-99-783

July 1999

This work was supported by DARPA under Army contract DABT63-96-C-0106.

Optimum Instruction-level Parallelism (ILP) for Superscalar and VLIW Processors

by

Patrick Hung and Michael J. Flynn

Technical Report No. CSL-TR-99-783

July 1999

Computer Systems Laboratory

Stanford University

Gates Building 3A, Room 332

Stanford, California 94305-9030

pubs@shasta.stanford.edu

Abstract

Modern superscalar and VLIW processors fetch, decode, issue, execute, and retire multiple instructions per cycle. By taking advantage of instruction-level parallelism (ILP), processor performance can be improved substantially. However, increasing the level of ILP may eventually result in diminishing and negative returns due to control and data dependencies among subsequent instructions as well as resource conflicts within a processor. Moreover, the additional ILP complexity can have significant overhead in cycle time and latency.

This technical report uses a generic processor model to investigate the optimum level of ILP for superscalar and VLIW processors.

Key Words and Phrases: Instruction Level Parallelism, Superscalar processor, VLIW

Contents

List of Figures

List of Tables

1. Introduction

Computer architecture has evolved in the last fifty years in order to pursue higher performance, which usually comes at the expense of increased complexity. Server processors today are typically superpipelined superscalar or VLIW processors, implying that these processors can potentially execute many instructions in parallel.

Dubey and Flynn [DF90] used an analytical processor model and showed that superpipelining technique can result in diminishing and even negative returns due to pipeline overhead as well as unexpected processor behaviors such as branch disruptions and exceptions. This means that there is an optimum number of pipeline stages to maximize processor performance.

There is an analogous behavior in selecting the optimum level of instruction-level parallelism (ILP). Researchers have showed that increasing the level of ILP results in diminishing returns due to dependencies among the instructions in a program [JW89]. In this report, we use an analytical ILP-processor model and conclude that if we consider the effects of the ILP overhead in instruction latency and cycle time, increasing the level of ILP can eventually result in negative returns.

2. ILP-Processor Model

In this section, we develop a processor model to analyze superscalar and VLIW processors. In both superscalar and VLIW processors, multiple instructions are fetched, decoded, executed, and retired per cycle, but a superscalar processor schedules instructions dynamically whereas a VLIW processor schedules instructions statically.

The transfer of instructions from decode stage to execution stage is known as "instruction issue." Instructions can only be issued if there are available processor resources to service the new instructions, and these instructions do not depend on any previously issued instructions which have not yet been completed.

There are two different kinds of instruction dependencies: control and data dependencies. Control dependency refers to the case that an instruction may affect the execution of another instruction due to a conditional branch. Data dependency corresponds to the fact that subsequent instructions may be dependent on each other because of data; for instance, the input operand of an instruction is the same as the result operand of a previous instruction.

Assume that a processor can issue up to N instructions per cycle. Let P be the probability that a new instruction cannot be issued, and M be the number of issued instructions which have not yet completed. If M is zero, a new instruction can always be issued; on the other hand, if M is very large, it is likely that a new instruction has to be stalled because of control and data dependencies or resource conflicts. Therefore, P is zero if M is zero, and P approaches one if M is very large.

For simplicity, we assume that the control and data dependencies between any two pairs of instructions are statistically independent. The assumption simply means that if we select any two pairs of instructions in a program, the fact that there are dependencies between one pair of instructions does not provide any information on the dependencies between the other pair of instructions.

Let p be the probability that two randomly selected instructions have control or data dependencies, P and p are then related by:

$$1 - P = (1 - p)^M \tag{1}$$

The probability p is usually small, and we can use binomial expansion to expand $(1 - p)^M$. The above equation can be approximated by:

$$P = M \cdot p \tag{2}$$

Equations (1) and (2) assume that there is no resource conflict within the processor. However, the probability that there is resource conflict is also roughly proportional to the number of active instructions in the processor. We can therefore assume that P is proportional to M with a new proportional constant p' which considers instruction control and data dependencies as well as resource conflicts.

$$P = M \cdot p' \tag{3}$$

The number of active instructions M depends on both the instruction latencies and the processor throughput. On the average, M is equal to the product of the processor throughput in IPC (number of instructions per cycle) and the average instruction latency $L(N)$.

$$M = IPC \cdot L(N) \tag{4}$$

Here, we assume that the average latency $L(N)$ is a function of the ILP width N. The reason is that the ILP width N affects the processor complexity and can therefore affect the average instruction latency. Substituting (4) into (3), we get:

$$P = p' \cdot IPC \cdot L(N) \tag{5}$$

On the other hand, IPC also depends on the number of issues N and the issue probability $(1 - P)$.

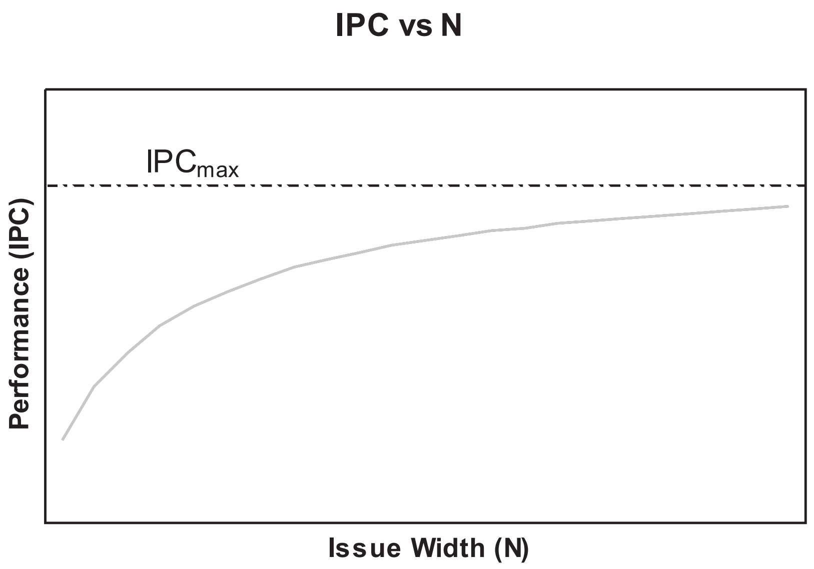

Figure 1: IPC versus N (Instruction Latency = L_0)

$$IPC = N \cdot (1 - P) \tag{6}$$

Substituting (6) into (5), we obtain a simple relationship among processor throughput in IPC, ILP width N, and average instruction latency $L(N)$.

$$IPC = \frac{N}{1 + N \cdot p' \cdot L(N)} \tag{7}$$

2.1 Fixed Cycle Time and Instruction Latency

In this section, we analyze the optimistic case that the level of ILP neither affects the average instruction latency nor the cycle time. In this case, the processor performance is proportional to the IPC and we assume $L(N)$ is equal to a constant L_0. Substituting $L(N)$ into (7), we get:

$$IPC = \frac{N}{1 + N \cdot p' \cdot L_0} \tag{8}$$

Figure 1 shows the relationship between IPC and N. Although the ILP complexity does not affect the average instruction latency or the cycle time, we still observe diminishing returns by increasing

N. When N approaches infinity, we can derive the maximum attainable IPC (IPC_{max}). The equation shows that IPC_{max} is inversely proportional to the product of L_0 and p'. For example, if L_0 is 10 and p' is 5%, IPC_{max} is 2.0.

$$IPC_{max} = \frac{1}{p' \cdot L_0} \tag{9}$$

2.2 ILP Impacts on Cycle Time and Instruction Latency

In this section, we consider the general case when the increased level of ILP does have negative impacts on instruction latency and cycle time. Let L_0 and T_C be the average instruction latency and the cycle time for a scalar processor, respectively. Table 1 shows nine possible modeling schemes to model ILP overhead in instruction latency and cycle time, where $\mathcal{O}$ represents a constant overhead factor.

In general, increasing the level of ILP increases both gate delay and interconnect delay. In deep submicron VLSI designs, interconnect delay is usually more dominant than gate delay. The exact overhead model depends on the actual processor implementation, which in turn depends on the circuit design and the fabrication technologies. In scheme A, the average instruction latency and the cycle time are assumed to be independent of the ILP width N; this case has already been discussed in Section 2.1.

Suppose the functional blocks are laid out in a linear array, the interconnect length is then proportional to N. If buffer insertion technique is employed, interconnect delay is proportional to the interconnect length; on the other hand, if buffer insertion technique is not used, interconnect delay is proportional to the square of interconnect length. Consequently, the interconnect delay is proportional to N with buffer insertion, and the delay is proportional to N^2 without buffer insertion. The overhead can be a penalty in the average instruction latency as shown in Schemes B and C, or a penalty in the cycle time as shown in Schemes F and G.

Table 1: Different Schemes to Model ILP Overhead

Scheme	Average Instruction Latency	Cycle Time
A	L_0	T_C
B	$L_0 \cdot (1 + \mathcal{O} \cdot N^2)$	T_C
C	$L_0 \cdot (1 + \mathcal{O} \cdot N)$	T_C
D	$L_0 \cdot \left(1 + \mathcal{O} \cdot \sqrt{N}\right)$	T_C
E	$L_0 \cdot (1 + \mathcal{O} \cdot log(N))$	T_C
F	L_0	$T_C \cdot (1 + \mathcal{O} \cdot N^2)$
G	L_0	$T_C \cdot (1 + \mathcal{O} \cdot N)$
H	L_0	$T_C \cdot \left(1 + \mathcal{O} \cdot \sqrt{N}\right)$
I	L_0	$T_C \cdot (1 + \mathcal{O} \cdot log(N))$

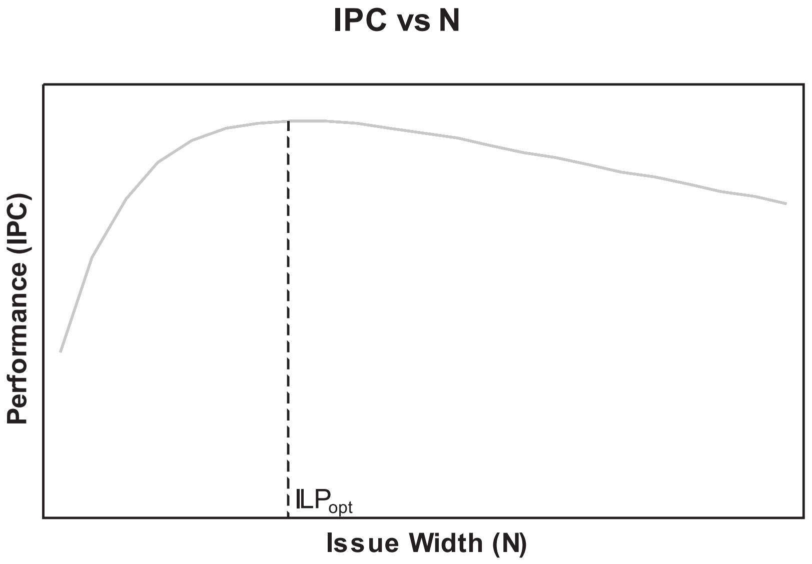

Figure 2: IPC versus N (Instruction Latency = $L_0 \cdot (1 + \mathcal{O} \cdot N)$)

On the other hand, suppose the functional blocks are laid out in a 2-D array, the interconnect length is then proportional to $\sqrt{N}$. In this case, the interconnect delay is proportional to $\sqrt{N}$ with buffer insertion, and the delay is proportional to N without buffer insertion. The overhead can be a penalty in the average instruction latency as shown in Schemes C and D, or a penalty in the cycle time as shown in Schemes G and H.

If the interconnect resistance is very small or if the interconnect delay is small compared with gate delay, increasing N has logarithmic effects on the total delay, which corresponds to Schemes E and I. For instance, assuming fanout-of-4 (FO4) for all logic gates, the decoding time of a memory array is only increased by one logic gate if the memory is quadrupled in size.

2.2.1 ILP Impacts on Average Instruction Latency

In Table 1, Schemes B, C, D, and E represent different ILP impacts on the average instruction latency. All these models exhibit similar behaviors, although in some cases the optimum N (N_{opt}) are more distinctive than in other schemes. We show the calculations for Scheme C, but the other schemes have similar results. In Scheme C, the average latency $L(N)$ increases linearly with the ILP width N. Substituting $L(N)$ into (7), we get Equation (10). Figure 2 shows the relationship between IPC and N.

$$IPC = \frac{N}{1 + N \cdot p' \cdot L_0 \cdot (1 + \mathcal{O} \cdot N)} \tag{10}$$

If we differentiate (10) with respect to N, we get (11).

$$\frac{\partial IPC}{\partial N} = \frac{1 - p' \cdot \mathcal{O} \cdot L_0 \cdot N^2}{(1 + N \cdot p' \cdot L_0 \cdot (1 + \mathcal{O} \cdot N))^2} \tag{11}$$

By setting $\frac{\partial IPC}{\partial N}$ to zero, and we can determine the optimum N (N_{opt}) and the maximum attainable IPC (IPC_{max}) (Equations (12) and (13)). When $\mathcal{O}$ is zero, N_{opt} becomes infinity and IPC_{max} becomes $\frac{1}{p' \cdot L_0}$, which is consistent with the results shown in Section 2.1.

$$N_{opt} = \frac{1}{\sqrt{\mathcal{O} \cdot p' \cdot L_0}} \tag{12}$$

$$IPC_{max} = \frac{1}{p' \cdot L_0 + 2\sqrt{\mathcal{O} \cdot p' \cdot L_0}} \tag{13}$$

2.2.2 ILP Impacts on Cycle Time

In this case, processor performance is proportional to $\frac{IPC}{CycleTime}$. We only show the calculations with Scheme G, but the calculations for Schemes F, H, and I are similar. In Scheme G, the cycle time T_C increases linearly with the ILP width N. Using (7), we can derive the performance equation as below.

$$Performance = \frac{N}{(1 + N \cdot p' \cdot L_0) \cdot (1 + \mathcal{O} \cdot N)} \tag{14}$$

We can differentiate (14) with respect to N to get the optimum ILP width N_{opt}, which is identical to N_{opt} in Scheme C (Equation 12).

$$N_{opt} = \frac{1}{\sqrt{\mathcal{O} \cdot p' \cdot L_0}} \tag{15}$$

3. Simulation Results

We use the MXS simulator [Ben98], the detailed simulator component of the SimOS simulation environment [RHWG95] to verify our model. MXS employs an execution based simulation method to accurately model a dynamically scheduled processor. The benchmarks are from the SPEC 92 benchmark suite.

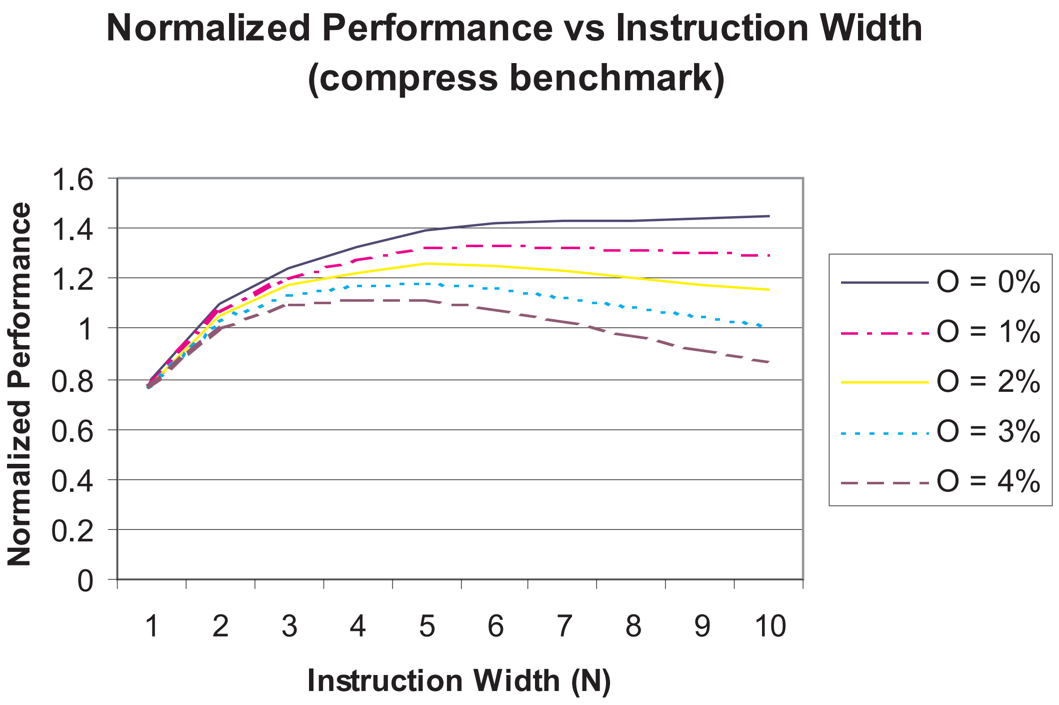

Figure 3: Performance versus ILP

The processor model was selected to represent a modern dynamically scheduled processor. It is a N-issue, dynamically scheduled processor with register renaming, branch prediction, speculative execution, and precise interrupts. Instructions issue out-of-order, and a reorder buffer is used to restore the precise state after an interrupt. The load/store buffer has 16 entries, the instruction window has 16 entries, and the reorder buffer is 32 entries.

For simplicity, only single-level 8-K instruction cache and data cache are modeled. They are four way set associative with an LRU replacement policy and a fixed line size of 32 bytes. The data cache is write back with write miss allocate. Both memory latency and memory bus traffic are modeled. The cache miss has a latency of 20 cycles and consumes 4 bus cycles.

We assume the ILP overhead in cycle time increases linearly with the ILP width N, which corresponds to Scheme G in Table 1. In Figure 3, we show the results of the "compress" program with $\mathcal{O}$ equals 0%, 1%, 2%, 3%, and 4%. The performance behaviors are consistent with our proposed analytical model. We take $\mathcal{O} = 4\%$ as an example. In this example, the optimum ILP width (N_{opt}) is around 4. We also ran MXS simulations for some other SPEC 92 benchmarks (Espresso, Sc, Spice, Wave, Lisp) and obtained similiar results.

4. Conclusions

We proposed a generic processor model to analyze the optimum level of ILP for superscalar and VLIW processors, and showed that increasing the level of ILP can eventually result in negative returns. There is an optimum ILP width (N_{opt}) which depends on three factors: average instruction latency (L_0), ILP overhead ($\mathcal{O}$), and the probability that two subsequent instructions have control and data dependencies and resource conflicts (p').

We showed a number of ways to model ILP overhead in cycle time and average instruction latency, but concluded that these schemes all exhibit similar behaviors. Our model has been verified with SPEC 92 benchmark suite using the MXS simulator.

5. Future Works

We are working on a more generalized model with the following IPC equation, where K is a constant. This equation is same as Equation (7) when K is 1.

$$IPC = \frac{N}{K + N \cdot p' \cdot L(N)} \tag{16}$$

References

[Ben98] James E. Bennett. *Latency Tolerant Architectures.* PhD thesis, Stanford University, 1998.

[DF90] Pradeep K. Dubey and Michael J. Flynn. Optimal pipelining. *Journal of Parallel and Distributed Computing*, 8:10–19, 1990.

[JW89] N. P. Jouppi and D. W. Wall. Available instruction-level parallelism for superscalar and superpipelined machines. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 290–302, 1989.

[RHWG95] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: the SIMOS approach. *IEEE Parallel & Distributed Technology: Systems & Applications*, 3:34–43, 1995.