Non-Speculative Enhancements for a Pipelined Scheduling Logic

Rubén Gran, Enric Morancho, Àngel Olivé, José María Llaberà

Computer Architecture Department, Universitat Politècnica de Catalunya, Jordi Girona 1-3, 08034 Barcelona, Spain

ABSTRACT

Pipelining the scheduling logic over two cycles degrades performance with respect to an unpipelined logic (about 11% in a 4-way machine). We present two non-speculative enhancements for a pipelined scheduling logic. Our results show that both enhancements perform, in average, slightly better than two previously proposed speculative schedulers. The performance of our proposals is within a 2.6% and 2% of an unpipelined scheduler.

KEYWORDS: Dynamic Scheduling, Pipelined Scheduling Logic

1 Introduction

The dynamic scheduling logic allows both exposing and exploiting the instruction-level parallelism (ILP). The scheduling task is divided in two phases: wakeup and select. They form a hardware loop, the scheduling loop, because an instruction cannot be scheduled until its producer instructions have been scheduled. Assuming that the producer-instruction latency is one cycle then, in order to execute dependent instructions in consecutive cycles, the scheduling task must be performed in one cycle.

Enlarging the issue queue to expose more ILP may increase the latency needed to wakeup and select instructions, which may require reducing clock frequency. An approach to either maintaining or increasing clock frequency is pipelining the scheduling logic over several cycles. Techniques that allow pipelining the scheduling logic without sacrificing the back-to-back execution of dependent instructions are an option to design high-frequency processors. However, proposed techniques ([1], [2]) are speculative.

In this paper, we enhance a scheduling logic pipelined over two cycles to increase its performance. The proposed enhancement is non-speculative and able to execute dependent instructions in consecutive cycles when not enough ILP is available. Our results show that our two proposed enhancements outperform, in average, two previously proposed speculative schedulers, in SPEC-2000 integer benchmarks. The performance of our proposals is within a 2.6% and 2% of an ideal (unpipelined) scheduler.

2 Baseline processor model

We assume a dynamically scheduled processor with a conventional pipeline (stages: fetch, decode, rename, issue queue, read payload, read register file, execution; write register file, commit). Each stage can take one or more cycles.

E-mail: {rgran, enricm, angel, llaberia}@ac.upc.edu
This work has been supported by the Ministry of Science and Technology of Spain under contract TIC-2004-07739-C02-01, grant AP2003-4621 (Rubén Gran) and the HiPEAC European Network on Excellence.
Wakeup logic (W). We use a wired-OR style wakeup logic array. Dependencies are indicated using an instructions-instructions wakeup matrix. Bit vectors (rows) perform dependence tracking. Each bit in the vector represents the dependence on a parent instruction. When an instruction is issued, it sets the wakeup line (column) corresponding to its own IQ entry. Each instruction monitors the readiness of its source operands every cycle by checking if all wakeup lines of matching dependence bits are set. Each issue-queue entry corresponding to a ready instruction activates a request signal in order to notify its readiness.

Select logic (S). The input of the select logic are request signals from the wakeup logic plus priority information. The select logic picks the oldest ready instructions taking into account available resources. Instructions selected by the select logic are the input of the wakeup logic on next clock cycle in order to wakeup instructions dependent on the selected ones.

Figure 1 shows diagrams of one-cycle latency (unpipelined) and two-cycle latency scheduling loops. Back-to-back execution is possible only if the execution latency of the producer instruction is greater than or equal to the scheduling-loop latency. As a reference, 44.30% of committed instructions in SPEC-2000 integer benchmarks are one-cycle-latency, register-writing instructions; in SPEC-2000 FP benchmarks this percentage drops to 23.62%. In this paper, the scheduling-loop latency of the baseline processor is two cycles.

3 Enhanced scheduling logic

3.1. Base enhancement (E)

The idea is computing in advance which instructions will be woken up by all instructions currently competing for selection. Once all of them have been selected, the pre-computed group of instructions can compete for selection in next cycle. This idea is applied only to instructions dependent on one-cycle execution-latency instructions.

Figure 2 shows the scheme of a two-cycle latency scheduling logic with our enhancement. The base scheduling logic is composed by the wakeup matrix A and the select logic. The wakeup matrix B computes in advance which dependent instructions will be woken up by the one-cycle execution-latency instructions in the input of the selection logic.

In rename stage, each instruction is classified considering the latency of the instructions that wake it up (parent instructions). At dispatch time, all instructions are stored in wakeup matrix A. In wakeup matrix B are stored instructions that may be woken up by one-cycle execution-latency instructions.

Inputs of the wakeup matrix B are: a) one-cycle execution-latency instructions in the input of the select logic and b) selected instructions with an execution latency greater than one cycle. These inputs are respectively calculated by filters D and C.

The logic circuit M merges the request signals generated by both wakeup matrices. These request signals are merged if all one-cycle execution-latency instructions pending for selection in the input of the selection logic have been scheduled (zero-detection logic).
The request signal of an instruction that is woken up by an one-cycle execution-latency instruction is activated in both wakeup matrices at different cycles. First, when the parent instruction is in the input of the select logic, the request signal is activated in wakeup matrix B. Later, when the parent is selected, the request signal is activated in wakeup matrix A. Then, for every instruction inserted in both wakeup matrices, the logic circuit M filters out the latest request signal to arrive related to the same entry. Therefore, only one request signal is observed by the selection logic. Another case happens when the latest arriving operand of an instruction stored in wakeup matrix B is produced by an instruction whose execution latency is greater than or equal to the scheduler-logic latency. In this case both request signals are activated in the same cycle.

3.2. Adding instruction fusing (E-F)

The proposed enhancement can be improved by taking advantage of a program characteristic: at dispatch time, a large number of instructions has only one source operand (avg: 78.6% of committed instructions in SPEC-2000 int). Then, we fuse instructions (a producer instruction and its dependent one) in order to favour back-to-back execution of dependent instructions. Two instructions are fused when producer instruction is an one-cycle execution-latency instruction and the consumer instruction only depends on this instruction.

In our model E-F, the advantages of fusing instructions are twofold. First, the consumer instruction can compete for selection once the producer instruction have been selected. Therefore, back-to-back execution is possible. Second, it is not necessary to store the consumer instruction in the wakeup matrix B for waking it up.

The possibility of fusion is detected in rename stage. The fused instructions must belong to the same dynamic basic block and the producer instruction must be in the issue queue. In our evaluations, two issue-queue entries are allocated to fused instructions in wakeup matrix A and two issue cycles are needed to schedule them.

4 Simulation Environment

We have modified SimpleScalar 3.0c in order to model separate IQs (integer IQ and floating-point IQ). We assume fifteen stages from Fetch to IQ and two stages between IQ and Execution. Other processor parameters are listed in Table 1. Our workload is composed by the SPEC2000 integer benchmarks compiled on an Alpha machine. We simulate a representative contiguous run of 100M-instruction from after a 100M-instruction warming-up.

<table>
<thead>
<tr>
<th>Table 1 Processor and memory parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch predictor: hybrid (bimodal, gshare)</td>
</tr>
<tr>
<td>ROB entries / LSQ entries</td>
</tr>
<tr>
<td>Issue-queue entries INT / FP</td>
</tr>
<tr>
<td>Integer ALU’s / FP ALU’s / Memory ports</td>
</tr>
<tr>
<td>INT (ALU 1, * 10) FP (+* 4, / 15, others 1)</td>
</tr>
</tbody>
</table>

5 Results

We have simulated several models with a two-cycle latency scheduling loop: A baseline model (B); also, we model instruction fusing (B-F). Our proposed models (E and E-F). B-Double model doubles the number of integer-IQ entries of the B model. The Speculative Wakeup (SW, [2]) and the Select-Free (SF, [1]) models; both are speculative mechanisms designed to tolerate the scheduling-logic
latency. Moreover, we simulate an ideal model (ID) with unpipelined scheduling-loop; its pipeline depth is kept consistent by adding an extra stage in the front-end. Table 2 shows the IPC of each benchmark in B model.

Table 2 IPC of the baseline model (B).

<table>
<thead>
<tr>
<th>bzip2</th>
<th>crafty</th>
<th>eon</th>
<th>gap</th>
<th>gcc</th>
<th>gzip</th>
<th>mcf</th>
<th>parser</th>
<th>perl</th>
<th>twolf</th>
<th>vortex</th>
<th>vpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.34</td>
<td>1.86</td>
<td>2.12</td>
<td>1.94</td>
<td>1.38</td>
<td>1.60</td>
<td>0.13</td>
<td>1.01</td>
<td>1.38</td>
<td>0.88</td>
<td>2.38</td>
<td>0.77</td>
</tr>
</tbody>
</table>

Figure 3 shows the speed-up of all models with respect to the B model. We present individual results and two average values: for all benchmarks (HM) and for all benchmarks but mcf (HM-mcf). E and E-F models, in average, outperform both the B-Double and the B-F models. B-Double outperforms our proposed models in benchmarks mcf and vortex. However, in the other benchmarks, doubling the number of issue-queue entries to expose more parallelism is not cost-effective. Our proposed models also outperform B-F model in all benchmarks. B-F model outperforms, in average, about 2.8% the B model. Performance of E and E-F models are, in average, within 2.6% and 2% of the ID model, respectively. In average, a 19.2% of dispatched instructions are fused with their producer instruction.

We observe that our proposals, in average, outperform the speculative models (SW and SF). The SW model outperforms our proposals only in vpr; the SF model outperforms E model in bzip2, gzip, perl and vpr. However, E-F model is outperformed by SF model only in vpr. While both SF and SW models are speculative, our proposed models are not. The SF model must re-schedule some instructions, that have been speculatively woken-up. This involves activity, which wastes energy, in both the select logic and the register file. In SF model the re-schedulings affect to 3.4% of committed instructions and 3.0% of selections. The SW model may select instructions whose selection will be later nullified because parent instructions have not been issued. These false selections affect, in average, to a 7.6% of the committed instructions and a 4.6% of selections by the select logic.

6 Conclusions

We propose two non-speculative enhancements (E and E-F) for a scheduling logic pipelined over two cycles. They try to tolerate the scheduling-loop latency. Our evaluations show that E and E-F perform, in average, within 2% and 2.6% of an ideal (unpipelined) scheduler, respectively. Compared to baseline model, E and E-F increase performance, in average, 7.9% and 8.6%. Also, E and E-F perform, in average, slightly better than two previously-proposed speculative schedulers.

References