Comparative Architectures Supervision Answer (Set 2)

Wei Song 28/02/2017

This answer provides a summary of key points in each question. It is the understanding of the supervisor, which does not necessarily always correct and does not represent the view of lecturers. It is expected to expand the key points into detailed description when similar questions are asked in exams.

Lecture 5 & 6. Superscalar

Read the paper about the ALPHA 21264 superscalar processor and answer the following question.

Link: https://www.cis.upenn.edu/~milom/cis501-Fall09/papers/Alpha21264.pdf

Q1. For the Alpha 21264 processor, two architectural techniques boosted the instruction fetch efficiency respectively, one is line and way prediction, and the other one is branch prediction.

1. **The line and way predictor predicts the line-way of the instruction cache that will be accessed in the next cycle. Please read the paper and explain how it could hide the long delay of the branch predictor and why the line and way predictor could be trained by the results of the branch predictor?**
   - The line-way predictor is an inaccurate (compared to the branch predictor) but fast predictor working in a similar way as BTB. (A classic fast-inaccurate and slow-accurate combination scheme)
   - The line-way predictor is able to predict the next line-way in one cycle; therefore hides the long latency of the branch predictor.
   - The branch predictor verifies (in the slot cycle) the fast prediction from the line-way predictor. If the prediction is wrong, the fetched instructions are flushed. At the same time, the line-way predictor is trained by the prediction from the branch predictor.
   - The prediction accuracy of the line-way predictor is comparatively high (>85%) and the misprediction cost is small (1~2 cycles). (Most of the time the line predictor does not predict, just pointing to the next line)
   - Please also note that the line-way predictor does add extra area to the L1 I$ as every 4 instruction block now needs to be augmented with line prediction, way prediction and a 2-bit saturation counter.
2. **What patterns in branches are better predicted by global branch predictors and what patterns are preferred by local branch predictors?**
   - Only conditional branches need branch prediction!
   - Local branch predictors are good for self-correlated branches.
   - Global branch predictors are good for inter-correlated branches.
   - Extra reading: Scott McFarling, “Combining branch predictors”, WRL Technical Note TN-36, 1993.
   - Extra key points: Please understand the implementation of local/global predictors; the gshare predictor and the tournament scheme (how to select between local and global predictors).

Q2. Register renaming exposes instruction-level parallelisms since it eliminates unnecessary dependencies. Out-of-order execution then exploits the benefits and deploys them on multiple execution units.

1. **Which dependences are removed and how renaming could remove the hazards caused by these dependences?**
   - False dependences (WAW and WAR) are removed. False dependences are caused by name conflicts. Which means the dependence is caused by reusing the same register but not any actual data dependence. Therefore, renaming architecture registers into the more abundant physical registers avoids such register reusing, which as a result removes false dependences.
   - Renaming cannot and should not remove true dependences (RAW). True dependences are data dependences which are required by applications.
   - The reason that we have false dependence is due to limited number of architecture registers visible by compilers.
2. **Instructions could be issued out of order, why do we need to fetch and commit (retire) instructions in order? What problems occur when committing instructions out of order?**
   - In order fetch is required for the pipeline to know the program order and detect true dependences.
   - In order commit has two major reasons: one is the cost of out-of-order commit is high. It is difficult to roll back writes to the register file and it is nearly impossible to roll back writes to memory (cache).
   - The other reason for in order commit is precise exception.
3. **When a mispredicted branch or an exception is detected, how would the register map be reverted to the latest valid state? What additional hardware is needed?**
   - Simply put, if a roll back (revert) is ever needed, a history is needed.
   - A buffer is used to store the mappings of all the in-flight instructions. Therefore when a misprediction/exception happens, the register map is rolled back to the last committed instruction.

Q3. The out-of-order execution of ALU instructions in a superscalar processor is only constrained by the availability of functional units and true data dependencies. Why must the out-of-order execution of memory instructions be constrained further? (2009 Paper 7, Q 7)

- It is difficult to undo a memory write.
- In order memory write is required by precise exception.
- In order memory write or even enforcing memory operations in program order is required by certain memory consistence modules (eg. SC and PSO).

Lecture 7. VLIW

Q4. What components in typical superscalar processors are no longer needed in VLIW processors?

Since data dependences and execution order are handled by the compiler, all components related to handling data hazards and speculation (dynamic reorder) are no longer needed. These include register renaming, out-of-order issue, reorder buffer, etc.

Q5. Why might dynamic binary translation be particularly useful in the case of a VLIW machine? (Sup-work Q 7.2)

VLIW processors lack binary compatibility with existing ISAs and often even between different generations of VLIW architectures. Dynamic binary translation enables VLIW processors to execute the same binary as the superscalar processors and thus reduces software development cost.

Q6. Some VLIW processors contain additional hardware to permit memory reference speculation. What optimisations does memory reference speculation permit? Briefly describe the additional hardware required to support this type of speculation. (2008 paper 8, Q 5)

Memory reference speculation permits loads to be speculatively moved above stores. It hides load latency by speculatively executing load before potentially aliased store.

To support this speculation, the load must be redone when the speculation is wrong.

- A check instruction is added at the original load position to check the correctness of speculation.
- If the check turns out wrong, jump to a fix-up code.
- The fix-up code re-executes the load instruction and all instructions dependent on this load, and then jumps back to continue execution.

Q7. What advantages does dynamic scheduling offer when compared with static scheduling in a superscalar processor? (2010 paper 8, Q 5)

- Simplifies the scheduler in compilers.
- Allow code compiled against another processor to run efficiently.
- The scheduling optimization in compilers is static, which is lack of run-time information.
- Compilers are restricted by the small number of architecture registers.

Lecture 8. Thread level parallelism

Q8. What are the advantages of exploiting Thread-Level Parallelism (TLP) in addition to Instruction-Level Parallelism (ILP)? (Sup-work Q 8.5)

- ILP may not be able to fully utilize the available parallel execution units due to the hardware limits in its speculative scheduling or there is just not enough ILP parallelism to explore in the application.
- TLP and ILP are almost orthogonal techniques. It may be easier (less speculation and scheduling cost) to achieve the same level of parallel utilization (instruction throughput) compared with using ILP along.
- TLP can hide memory latency and resolve data hazards.

Q9. Briefly describe the differences between coarse-grained, fine-grained and simultaneous multithreading. (Sup-work Q 8.2)

- Coarse-grained TLP: switch to alternative thread when the current thread stalls. Hide memory latency. The pipeline is flushed when switching. Small hardware cost with heavy switching cost.
- Fine-grained TLP: threads are selected and scheduled on every clock cycle. Hide memory latency and no flushing penalty for switching. Need a thread scheduler and separate pipeline control.
- Simultaneous multithreading: allow instructions from multiple threads to be issued out-of-order. Achieve the best utilization of pipelines allowed by TLP. More hardware cost in scheduling and speculation than fine-grained TLP.

Q10. Describe techniques for reducing the thread switch penalty in a coarse-grained multithreaded processor. (Sup-work Q 8.4)

- Short pipeline can reduce the penalty.
- Buffer instructions of the alternative thread.

- Like fine-grained TLP, separate pipeline control can avoid flush.