CS7290 in-class Quiz-I

Name: Jipeng Wu

Submission guide: you submit a written version at the class and then submit an individual version and a group discussion version all together by next week's class time at T-square (only soft copy is accepted). Please indicate which version is yours and which version is after group discussion and also include your discussion members' names. In class no open book. After class, you can use books/papers/etc. and include citations if that's necessary. Do not exceed more than 1 page (total 4 questions) for each version.

Grading algorithm: in-class submission (10%) + individual submission (30%) + group discussion submission: 60% (no group size limit but it will affect the criteria for grading)

1 [Trace cache]

Trace cache can provide benefits in various aspects but it can also increase power consumption and complexity. Let's discuss pros and cons of trace cache.

(a) Discuss pros and cons of instruction trace cache in terms of power. Can trace cache increase or decrease power consumption? Do we need I-cache with a trace cache?

It depends on the general processor design. P4 as example, if you simply replace P4's TC as IC, TC does reduce power consumption. But modern processors have many supplement for IC design, such as a uop cache, which helps the IC design more power efficient.

- Pros:
  - TC can store decoded instructions(micro-ops). Therefore upon a TC hit, refetching and decoding power cost is saved. It saves power in cases where fetching and decoding is the bottleneck.
- Cons:
  - Misprediction is more costly than processors without TC.
  - The decoded micro-ops are much longer than instructions.[2] Therefore TC's size is smaller than IC, implying more I/O latency and power cost.
  - TC miss rate is significantly higher than IC's miss rate[1]. A TC miss(without IC in processor) will cost L2 latency.
  - Decoding cannot be paralleled if traces are filled at fetch. Only one instruction decoded and fed to TC in one cycle. Considering its low hit rate, the TC processor behaves like a single-issued antique 40-50% of the time.[3]

In terms of functionality, no, we don't need IC if general-purpose TC is already in the design. A TC that stores decoded ops can fully replace IC. But in terms of performance, a IC can improve performance of one-by-one fetch when there is TC miss.

(b) Discuss pros and cons of trace caches over compiler optimizations. Please show examples of code optimization effects in trace cache.

TC can work with and easily benefit from its corresponding compiler techniques.

- Pros:
  - Renaming: marks instructions which move a value from one register to another register as explicit move instructions.
  - Reassociation: unnecessary add dependency chain in a trace can be broken.
  - Scaled add instructions: add ops can be converted to add+shilft in one cycle.
  - Branch promotion: highly biased branches can be promoted to static prediction.
  - Can treat traces as an atomic operation, which enables higher level parallelization of traces. (Better concurrency programming support.)
  - Decoded ops can be reordered since the e dependencies between instructions within a trace cache line are stored explicitly.
- Cons:
  - All of these techniques are completely useless if TC's hit rate is as low as 60% while IC is usually above 90%.

(c) Discuss pros and cons of trace caches and branch predictors. Is it a good idea to turn off branch predictor?

Branch predictor should be used when TC misses, i.e., TC is being built. But TC predictor is definitely better when TC hits.

- Pros(turn off brach predictor, use TC predictor):
  - TC predictor has lower power cost and smaller prediciton latency.
  - TC predictor underlies micro-ops multi-fetch, the basis of TC design,
- Cons:
  - Branch predictors can improve processor's performance when TC is unusable. Considering TC's high miss rate, it's reasonable to use branch prediction.
  - If a TC predictor mispredicts, unlike branch predictors, all the ops that TC loads are not discarded instantly, which is extremely costly especially when some ops are time-consuming.

(d) Discuss the benefits of trace caches in terms of pipeline depth and width.

- The pipeline can have more stages since instructions are broken into micro-ops. For example, a stage that only transports data from one place to another. So it is definitely wider.
- The traces can be optimized for pipelining, renaming and reassociation for examle, can reduce dependencies and underlie wider pipelines.
- But in terms of depth, it's only deeper when TC hits. If TC misses, the processor behaves like it's single-issued single-fetched, which will not supply enough micro-ops per cycle to form a deeper pipeline. In this case the width of pipeline becomes a major disadvantage due to not only the inefficiency but also the higher penalty for misprediction.

[1] Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching
[2] The microarchitecture of Intel, AMD and VIA CPUs
[3] Inside Nehalem: Intel's Future Processor and System
[4] Putting the Fill Unit to Work: Dynamic Optimizations for Trace Cache Microprocessors

CS7290 in-class Quiz1 Group Discussion

Name: Jipeng, Jingyuan, Xin

1 [Trace cache]

Trace cache can provide benefits in various aspects but it can also increase power consumption and complexity. Let's discuss pros and cons of trace cache.

(a) Discuss pros and cons of instruction trace cache in terms of power. Can trace cache increase or decrease power consumption? Do we need I-cache with a trace cache?

Energy efficiency of different TC architectures vary:

1. CTC(concurrent trace cache): I-cache accesses occur in parallel with the trace cache, in which case TC always increases power consumption.
2. STC(sequential trace cache, e.g., Nehalem's LSD) & TC-only(e.g., Pentium4):
    a. pros: TC can store decoded instructions(micro-ops). Therefore upon a TC hit, refetching and decoding power cost is saved, especially when fetching and decoding is the bottleneck. (STC is more energy efficient than I-cache only engines according to research[5].)
    b. cons:
        i. Misprediction is more costly than processors without TC.
        ii. The decoded micro-ops are much longer than instructions.[2] Therefore TC's size is smaller than IC, implying more I/O latency and power cost.
        iii. Decoding cannot be paralleled if traces are filled at fetch. Only one instruction decoded and fed to TC in one cycle. Considering its low hit rate, the TC processor behaves like a single-issued antique 40-50% of the time.[3]

Pentium4's TC-only design pays unaffordable L2-level penalty. Its modern variant, Nehalem's LSD for instance, has better performance and energy efficiency. (I would consider LSD as a TC variant although it's less general-purposed) So we agree IC is needed with a TC.

(b) Discuss pros and cons of trace caches over compiler optimizations. Please show examples of code optimization effects in trace cache.

Based on paper[6] the effect of program optimization on trace cache efficiency, trace cache provides a significant boost in performance regardless of optimization level. However, the relative benefit of trace cache declines noticeably with increasing optimization level.

For example, in the case of using procedure inlining, since the instruction footprint is increased and the branch predictor' pattern history table is worse utilized due to the reduced number of taken branches, the decline of relative benefit of trace cache is quite dramatic with increasing optimization level.

(c) Discuss pros and cons of trace caches and branch predictors. Is it a good idea to turn off branch predictor?

Extra power might need to be expended on branch predictors. However, a good performed and highly accurate branch predictor will significantly improve overall architecture performance (IPC) and therefore save energy as well.

For example, If a TC-level predictor mis-predicted the next trace, without branch predictors, all the micro-ops that TC loaded will not be discarded until an access misses, which is extremely costly especially when some ops are time-consuming. In this case, a branch predictor is a fail-safe.

I would say that it is necessary keep branch predictor with trace cache.

(d) Discuss the benefits of trace caches in terms of pipeline depth and width.

In terms of depth, TC stores decoded micro-ops, such that the circuits can do less work per pipeline stage, for example, simply transport data from one part of chip to another. Therefore TC underlies deeper pipeline design.

In terms of issue width, The processor multi-fetches instructions whenever a new trace is being built. Therefore TC can increase issue width with a reasonable trace predictor plus accurate branch prediction.

[1] Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching
[2] The microarchitecture of Intel, AMD and VIA CPUs
[3] Inside Nehalem: Intel's Future Processor and System
[4] Putting the Fill Unit to Work: Dynamic Optimizations for Trace Cache Microprocessors
[5] [6] see Jingyuan's individual solution