Cache based throughput improvement techniques for Speculative SMT processors

Presenters
Ashay Rane
Sugan Vinayagam
Date - 11-29-2007

Project Goal:

- Throughput improvement for cache-based speculative SMT.
- Gap in performance between the proof-of-concept paper and the simulations in speculative SMT.
- Speculation support using LSQ, Cache based architectures.
- Combined benefits of LSQ and cache-based implementations.
Our Effort:

- Analyzed differences in architectures that supported speculation in SMT.
- Studied various factors that affected throughput.
- Thread squashing frequency is the dominant factor.
- Studied various architecture simulators and zeroed in on SSMT simulator and understood its design.
- Modified SSMT simulator to include cache-based support for speculation with our improved design.

2-thread scheme (Existing Scheme):

- 2 - extra states to each cache line
  - Speculative Valid (SV) state
  - Speculative Dirty (SD) state
- Extra bits to each cache line
  - Speculative Load (SL) bit
  - Speculative Modified (SMi) bit per word in the cache line.
- All speculative data are stored only in shared L1 cache.
- All data stored in L2 cache are non-speculative.
- At most one speculative thread.
**Dependence Checking:**

Non-Speculative thread Write

Cache line in SD State?

- Yes

- Write directly to L2 cache

Is SL bit set?

- Yes

- Squash & restart speculative thread

- No

Write data to shared L1 cache

---

**False Dependency:**

- On a non-speculative write, if SL is set, then thread squash.
Additional Bits (Our Architecture):

- SLi bit for every word in the L1 cache line.

- If a 32-bit word is assumed, cache size increased by 1/32% = 3.125%

- SMi bit for every word in the L1 cache line.

- Total increase in cache size is 6.25%.

Modified Speculative Load:

1. Speculative thread - load
2. Is SMi bit set?
   - Yes
   - No
3. Set SLi bit
4. Cache Line transits to SV state
5. Load data from the cache line
Modified Non-Speculative Write:

- Non-Speculative thread Write
  - Cache line in SD State?
    - Yes
      - Write directly to L2 cache
    - No
      - Is SLi bit set?
        - Yes
          - Squash & restart speculative thread
        - No
          - Write data to shared L1 cache

Replacement Policy:

- Cache Line with SLi or SMi bit set cannot be evicted from cache.
- Replace a line which has none of SLi or SMi bits selected.
- If a clean line is not found for non-speculative thread, squash the speculative thread and relinquish its resources to avoid deadlock.
- If a clean line is not found for speculative thread execution, it is suspended till it becomes non-speculative.
Commit & Squash:

- SLi and SMi bits are cleared, when a thread commits.
- Easier method to implement, unlike other schemes.
- SLi and SMi are cleared using gang-clearing which can be done in just one-cycle.
- SLi bit is cleared in all cache lines when a thread is squashed.
- If the SMi bit is set, then the valid bit of the corresponding cache line is cleared when a thread is squashed.

Simulator:

- SSMT simulator is used.
- LSQ based support for speculation in this simulator is modified to cache-based support.
- Understanding the simulator architecture.
- Code Changes:
  - L1 Cache architecture.
  - Cache access in L1.
  - State machine modifications in cache.
  - Dependence detection.
Expected Results:

- Reasons for thread squash -
  - Dependence Violation
  - Branch mis-prediction

- Two-level branch prediction accuracy is greater than 90%.

- Hence eliminating false dependencies is likely to yield significant improvement in performance.

- 75% of the dependence violations are false in a 4-worded cache line (assuming accessing all the words in a cache line are equi-probable).

- Therefore, eliminating false dependence violations is expected to decrease 75% of the thread squashing frequency.

Insight & Future Work:

- Gained a good understanding of throughput improvement techniques.

- Different hardware simulators, architecture and working of SSMT.

- Scaling up the design for supporting multiple speculative threads.

- Resource aware thread scheduling.
Questions?