Comparative Architectures Supervision Answer (Set 3)

Wei Song 07/03/2016

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 9 & 10. Cache

Q1. What does it mean if a cache memory hierarchy adopts a multi-level inclusion policy? What might influence a decision on whether to adopt a multi-level inclusion or exclusion policy? (2008 Paper 7 Q 5)

- Multi-level inclusion means the content of inner caches (such as L1 to L2) always presents in the outer caches (such as L2 to L1).
- Multi-level inclusion leads to duplicated cache lines in outer caches.
- In a single-core environment, a multi-level exclusion policy can be used to avoid duplication and cache more data.
- In coherent multi-core, multi-level inclusion is normally adopted to avoid probing all cores for every outer cache request.

Q2. A naive programmer writes the following code for performing the matrix multiply-add function C=AB+C on square matrices:

```
for (i=0; i<N; ++i) {
   for (j=0; j<N; ++j) {
      for (k=0; k<N; ++k) {
         C[k][i] = C[k][i] + ( A[k][j] * B[j][i] );
      }
   }
}
```

(Where X[v][u] refers to the element in row v, column u. Arrays are stored in memory row by row, i.e. X[0][0], X[0][1], X[0][2], ...X[0][N], X[1][0], ... etc.)

1. **When used to multiply very large matrices, performance of the programmer's algorithm is very poor. Explain what is happening.**

   The data access pattern lacks of spatial locality, which causes frequent cache misses.

2. **The algorithm can be improved simply by changing the order of the loops. Demonstrate how and why.**

   Move the inner loop (k <- 0:N) to the outermost loop. Then the data access in the innermost loop shows spatial locality, which significantly reduces cache misses.

3. **Show how further improvement can be obtained through a technique known as cache blocking.**

   According the size of data cache, break the loop i and j into sub-blocks that fit in the cache. See lecture note 10, page 26 for more details.

4. **Could the algorithm be successfully parallelised to run a on a microprocessor supporting Simultaneous Multithreading (SMT)? Briefly justify your answer.**

Yes. Using the cache blocking method, every sub-block can be parallelised into a thread. Since there is no data sharing between threads, threads can run perfectly in parallel.

Q3. [Open] Describe at least 3 different ways to improve the performance of a directly mapped cache.

- Increase the cache size by either increasing the number of sets or the size of cache blocks.
- Add outer caches to reduce missing penalty.
- Use write-through with a write buffer to avoid caching write-only data.
- Use victim cache to reduce conflict misses.
- Use assist cache to avoid thrashing.
- Use non-blocking cache to reduce missing penalty.
- Use stream buffer to prefetch cache misses.

Q4. [Open] Describe the three difference reasons of cache misses and the potential software/hardware methods to reduce the cache miss overhead caused by each reason.

- Compulsory misses
    - Increase block size to reduce control overhead when fetching data.
    - Use hardware/software prefetch to reduce load time.
    - Use stream buffer to prefetch cache misses.
- Capacity misses
    - Increase cache size.
    - Use assist cache to avoid thrashing.
    - Use write-through to avoid caching write-only data.
    - Use loop fusion and cache blocking in programming.
- Conflict misses
    - Increase the number of ways (associativity).
    - Use victim cache.
    - Use least recently used (LRU) replace scheme rather than FIFO or random.

Q5. A multicore processor has 8 scalar cores, each of which has an 8KB private write-through L1. A shared L2 of 2MB keeps all L1s in coherent state. Instead of maintaining a directory for each L2 cache line, this L2 chooses to maintain a directory by shadowing L1 tags. Can you figure out the reason? What if each L1 is 128KB, or if the L1 is write-back, would you design differently?

Sorry I have missed to provide one key information that the connection between L1 and L2 is a crossbar rather than a snoopy bus. Normally the use of directory indicates non-snoopy coherence protocol though.

The smaller directory size is the major reason of doing so.

If write-back is used, the benefit reduces for two reasons: 1. Each cache line now has at least three states: invalid, shared, exclusive/modified. The size of the directory is doubled (1 -> 2 bits). 2. Accessing to the shadowed L1 tags is slow and power consuming. With write-back L1 caches, read/write might need to probe L1s for updated data, which means extra enquires to the shadowed tags.

If the size of each L1 cache is increased to 128K, the size of state bits increased to 50% of the size if using per L2 line directory. However, the shadow tags need to be addressed individually (they are not tables but caches as well). The overall hardware cost can easily eat up the area benefit (actually there is no area benefit in this case).

Lecture 11. Vector Machine

Q6. [Open] Both VLIW and Vector (SIMD) improve ILP. Describe the pros and cons of VLIW and Vector machines.

There is no definite answer for this question. VLIW and Vector (SIMD) can be seen as orthogonal techniques and they can be utilized in the same processor (eg. ADI's TigerSHARC).

- VLIW explores instruction parallelism while Vector explores data parallelism.
- A VLIW machine can run all applications while a Vector machine normally works with a scalar processor or works as an extension to a superscalar ISA.
- A Vector machine needs scatter/gather instructions to communicate between vector and scalar registers.
- Vector machines have much regular (simple) switches between vectors and function units, which leads to lower energy per instruction compared with VLIW machines.
- Vector machines have much higher instruction density compared with VLIW machines.

Q7. What does support for vector chaining and tailgating allow in a vector processor?

Vector chaining resolves RAW hazards and tailgating resolves WAR hazards in vector processors.

Q8. [Open] Why does a vector processor offers a particularly energy efficient solution to execute some type of program?

With enough data parallelism to be exploited, the number of vector instructions is much less than the number of scalar instructions. Also loop fusion and unrolling reduce the number of branches.

The connection between vectors (registers) and function unit has a more regular pattern compared with superscalar or VLIW processors. Therefore, much simpler switches can be used to save switching energy.

Q9. [Open] In which situations might a vector processor perform worse than a simple pipelined processor?

Since a Vector processor needs scatter/gather instructions to load/store data from scalar register file and the available data parallelism may suffer from conditional execution, a Vector processor may perform worse even than a scalar processor when there is no enough data parallelism to compensate the scatter/gather and conditional execution overhead.