CSCI-UA.0480-003

Parallel Computing

Homework Assignment 2

Name: Meisi LI

NetID: ml6095

<u>Question 1:</u> We discussed briefly how caches are designed. Among cache characteristics are whether a cache is write back (when a cache block is modified, it is written back to the lower level cache only when the block is replaced) or write through (whenever a cache block is updated, it updates also the lower level copy). Discuss the pros and cons of each.

Write-back:

- Pros:
  - Low latency and high throughput for write-intensive applications.
  - Only write their contents back when a line is evicted.
  - If a given line receives, multiple store requests while it is in the cache, waiting until the line is evicted can significantly reduce the number of writes sent to the next level of the cache hierarchy.
  - Improves performance.
  - Uses less memory bandwidth which is useful in multiprocessors.
- Cons:
  - There is data availability exposure risk because the only copy of the written data is in the cache before the data is persisted to the backing store. This result in the data being lost.

Write-through:

- Pros:
  - It simplifies the design of the computer system.
  - Ensures fast retrieval while making sure the data is in the backing store and is not lost in case the cache is disrupted.
  - It is not necessary to record which lines have been written.
  - Easy to implement.
  - Main memory always has the most current copy of the data(consistent).
- Cons:
  - Writing date will experience latency as you have to write to two places every time.
  - Slower.

- Uses more memory bandwidth.

Question 2: Suppose a processor has to wait 10 cycles for its memory system to provide a 64-bit word. What can we do to reduce this delay when we have several loads coming from the processor to the memory? Ignore the existence of caches for now

- Increase block size
- Increase associativity
- Increase memory bandwidth
- Eliminate memory operations
- Reduce the number of misses
- Reduce the miss penalty
- Decrease the cache/memory access times
- Hide memory latencies

Question 3: Suppose we have a system with three level of caches: L1 is close to the processor, level 2 is below it, and level 3 is the last level before accessing the main memory. We know that two main characteristics of a cache performance are: cache access latency (How long does the cache take before responding with hit or miss?) and cache hit rate (how many of the cache accesses are hits?). As we go from L1 to L2 to L3, which of the two characteristics become more important? and why

Level 1(L1) cache very small in comparison to others and instructions are first searched in this cache, thus making it faster than the rest two caches.

Level 2(L2) cache is often more capacious than L1; it may be located on the CPU or on a separate chip or coprocessor with a high-speed alternative system bus

interconnecting the cache to the CPU, so as not to be slowed by traffic on the main system bus. If the instructions are not present in the L1 cache then it looks in the L2 cache, which is a slightly larger pool of cache, thus accompanied by some latency.

Level 3 (L3) cache is typically specialized memory that works to improve the performance of L1 and L2. It can be significantly slower than L1 or L2, but is usually double the speed of RAM. In the case of multicore processors, each core may have its own dedicated L1 and L2 cache, but share a common L3 cache. When an instruction is referenced in the L3 cache, it is typically elevated to a higher tier cache.

Therefore, as we go from L1 to L2 to L3, cache access latency could become more important.

Question 4: A sequential application with a 20% part that must be executed sequentially, is required to be accelerated three-fold. How many CPUs are required for this task? How about five-fold speedup?

According to Amdahl's Law, 1/(F+(1-F)/P) = maximum speed-up, F = 20%, then:

Three-fold: 1/(0.2+(1-0.2)/P) = 3, P = 6.

Five-fold: 1/(0.2+(1-0.2)/P) = 5, P = infinity.

Question 5: How does coherence protocol affect performance? Why?

There are some protocols in coherence protocol, such as MI(Modified-Invalid) Protocol and MSI(Modified, Shared and Invalid) Protocol.

- Coherence protocol is solution for maintaining consistency between the shared cache block.
- It supports low latency cache-to-cache transfer.
- Avoidance of dependency on bus like architecture as it restricts integration of more cores on the system.
- Bandwidth efficiency.

More specific for some protocol:

- MI(Modified-Invalid) protocol:
  - Easy to implement
  - Less transient states
  - Fewer burdens on cache controller
- MSI(Modified, Shared and Invalid) protocol:
  - Distinction between modified and shared state
  - Multiple copies of block can be present at the same time
  - Shared to Modify transition can be made without reading data from cache
- MOESI(Modified, Owned, Exclusive, Shared and Invalid) protocol:
  - Avoid extra CPU stall during write-back to main memory
  - One time only one cache can be owner(modified), other processors holding same block are in shared state.

Question 6: In slide# 11 of the performance analysis lecture (lecture 6) we saw that as the number of processes increases, the speedup increases.

a. Why the curve of “double size” seems better than the other two?

From Amdahl’s Law, the maximum speedup S achieved by a parallel computer with P processors performing the computation is S <= 1/(F+(1-F)/P). Then as P is the same, S depends on F. F is the fraction of a calculation that is sequential, so

F in “double size” is smaller than the other two. Therefore, the curve of “double size” is better than the other two.

b. If we keep increasing the number of processes (i.e. the x-axis) what do you think the rest of the curve will look like?
Upper limit: as P is infinity, S <= 1/F. If we keep increasing the number of processes, the rest of the curve will reach the upper limit 1/F in speedup(the y-axis) and will not increase anymore.

Question 7: In slide# 12 of the performance analysis lecture (lecture 6), efficiency decreases as the number of processes increases, why is that?

In slide #9 of the performance analysis lecture(lecture 6), efficiency E = (T of serial/ T of parallel) / P. As the number of processes increases, 1/P is decreasing, which means the efficiency decreases.

Question 8: What is the relationship between synchronization points in a parallel program and load balancing? Explain why do we have such a relationship?

To handle data dependencies, communicate requires data at synchronization points.

Load balancing us important to parallel programs for performance. It distributes work among tasks so that all are kept busy all of the time and minimization of task idle time.

If all tasks are subject to a barrier synchronization point, the slowest task will determine the overall performance. This means load imbalance is more severe

as the number synchronization points increases. If one of those tasks is very slow, the rest of work will in idle for a long time and result in load imbalance.

Question 9: In slide 12 of lecture 5, we found that parallelism = 6.25, yet, we can see from the graph that there are 8 paths that can be executed in parallel. How can you explain this discrepancy?

Because the number of processes. Work Law is T(p) >= T1/P and Span Law is T(p) >= T(infinity). The ration of work to span is T(p)/T(p) >= T1/(T(infinity)*P). Therefore, T1/T(infinity)*P <= 1. Because of P, the parallelism could be 6.25.

Question 10: Suppose we have two threads doing the same operations but on different data. Also suppose these two threads execute on two different cores, and those cores are not executing anything else but the assigned threads. If the two threads start at the same time, can we assume that they will always finish at the same time? Justify.

Yes, they will. This is exactly what threads are for.

When multi-threaded program execution occurs on a multiple core system(multiple uP, or multiple multi-core uP) threads can run concurrently, or in parallel as different threads may be split off to separate cores to share the workload.