CS570 Advance Computer Architecture

Homework 5

1. **What is the major contribution of the Layer Performance Matching?**

   - The rationale of LPM is that the performance of each layer of a memory hierarchy should and can be optimized to closely match the request of the layer directly above it. The LPM model simultaneously considers both data access concurrency and locality.
   - As data access concurrency becomes increasingly prevalent and applications become increasingly data intensive, the balance and optimization between locality and concurrency becomes a vital performance factor; therefore, the layered performance matching (LPM) becomes a key factor influencing performance.
   - **Major Contribution:** The LPM model is presented to quantify the performance match degree between the layers in a memory hierarchy

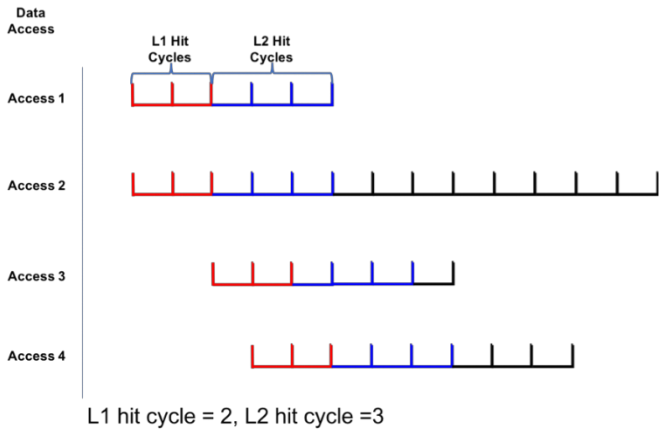
2. **We have given five case studies in Lecture 9. How these case studies related to I/O applications? Can you give one or two good examples for data transfer matching?**

   - The five case studies provided in the lecture are related to Layer Performance Matching and are essentially used to make Hermes related to I/O applications.
   - Uber can be taken into consideration as an example of Data transfer matching. For example let us consider uber provides car for every driver joining the company. Number of uber cars company providing should match the number of drivers available. Possible we can predict the number of cars required for particular season in the year which can make profits for the company.

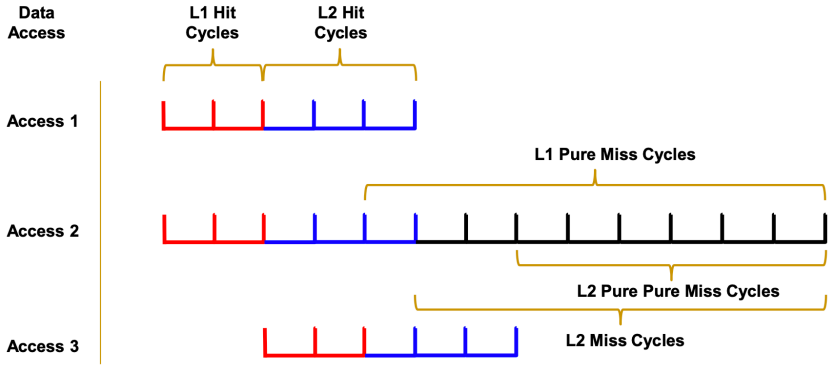
3. **What is memory stall time? What is the advantage of the Layered Performance Matching method? Why LPM can improve memory performance (in terms of memory stall time)by more than one hundred times?**

   - Having a stall in the pipeline means that the whole processor wastes one or more cycles, causing execution to slow down temporarily. The time that is consumed in stalling is memory stall time.
   - The rationale of LPM is that the performance of each layer of a memory hierarchy should and can be optimized to closely match the request of the layer directly above it. The LPM model simultaneously considers both data access concurrency and locality. It reveals the fact that increasing the effective overlapping between hits and misses of the higher layer will alleviate the performance impact of the lower layer.
   - The terms pure miss and pure miss penalty are introduced to measure the effectiveness of such hit-miss overlapping. By distinguishing between (general) miss and pure miss, we have made LPM optimization practical and feasible.

4. **The following figure shows a cycle-accurate pure[2] (pure-pure) miss example. In this example, there are 4 data accesses. Base on this figure, please calculate**



- Pure Miss Access: the miss access contains pure miss cycles
- Pure Miss Cycles: miss cycles without any hit
- Pure-pure miss, as its name says, is the pure miss of a pure miss
- The overlapping of L1 pure miss and L2 pure miss are the pure- pure miss (for a given core)



- The difference between lower level access time and cache access time is called the **miss penalty.**
- Average Miss Penalty (AMP) = $T_{MemPmiss}$ / $C_{MemPmiss}$
  $T_{MemPmiss}$ → Sum of total pure miss cycles
  $C_{MemPmiss}$ → Total number of pure misses.

(i) What is the average miss penalty of L1 misses?

{ 3+11+4+6 } / { 4 } = 6

(ii) What is the average pure miss penalty of L1 pure misses?

{ 0+8+3+6 } / { 4 } = 4.25

(iii) What is the average miss penalty of L2 misses?

{ 0+8+1+3 } / { 4 } = 3

(iv) What is the average pure miss penalty of L2 pure misses?

{ 0+5+0+3 } / { 4 } = 2

(v) How many pure^2 misses are there in this figure? How many pure^2 miss cycles does each pure^2 miss have?

It is having 2 pure^2 misses at Access 2 and Access 4 . There are 5 and 3 pure^2 miss cycles at access at 2 and 4 respectively.

5. Suppose there is no sequential portion, the program can be perfectly parallelized. The performance bottleneck is data access delay, and the data access delay is u_c. Please calculate the speedup of the three models of parallel speedup assuming the data access delay cannot be parallelized, and the number processors is p = 20.

a) Amdahl's law
b) Gustafson's law
c) Sun/Ni's law, assuming G(p)=p/2

Given:

a=0, p=20

a) **Amdahl's law**

1 / ((x) + (1-x/N)) = 1 / ((0) + (1/20))

= 20

b) Gustafson's law

Speedup $= f + P(1-f)$
$= 0 + 20 (1-0) = 20$

c) Sun/Ni's law, assuming G(20)=10.

$= [\alpha + (1 - \alpha) G (p)] / [\alpha + (1 - \alpha) G(p) / p]$
$= [0 + (1) * 10] / [0 + 1 * 10 / 20]$
$= [10] / [0.5]$
$= 20$

6. Please give a short answer to the following questions.

a) Why victim cache can improve cache performance?

- A victim cache is a small, usually fully associative cache placed in the refill path of a CPU cache that stores all the blocks evicted from that level of cache.
- Victim Caching is an improvement to miss caching that loads the small fully-associative cache with victim of a miss and not the requested cache line. A victim cache is a hardware cache designed to decrease conflict misses and improve hit latency for direct-mapped caches

b) What is the motivation and idea behind the skewed associative caches?

- Using a two-way skewed associative cache rather than a direct-mapped cache improves the performance of the system even when it slightly reduces the clock frequency

7. Please give a multicore false sharing example.

In symmetric multiprocessor (SMP) systems, each processor has a local cache. The memory system must guarantee cache coherence. False sharing occurs when threads on different processors modify variables that reside on the same cache line. This invalidates the cache line and forces an update, which hurts performance.

Example:

False sharing is a well-known performance issue on SMP systems, where each processor has a local cache. It occurs when threads on different processors modify variables that reside on the same cache line, as illustrated in Figure 1. This circumstance is called false sharing because each thread is not actually sharing access to the same variable. Access to the same variable, or true sharing, would require programmatic synchronization constructs to ensure ordered data access.

Following example code causes false sharing:

```
double sum=0.0, sum_local[NUM_THREADS];

#pragma omp parallel num_threads(NUM_THREADS)

{

 int me = omp_get_thread_num();

 sum_local[me] = 0.0;

 #pragma omp for

 for (i = 0; i < N; i++)
```

```
    sum_local[me] += x[i] * y[i];

    #pragma omp atomic

    sum += sum_local[me];

  }
```

There is a potential for false sharing on array sum_local. This array is dimensioned according to the number of threads and is small enough to fit in a single cache line. When executed in parallel, the threads modify different, but adjacent, elements of sum_local (the source line shown in red), which invalidates the cache line for all processors.

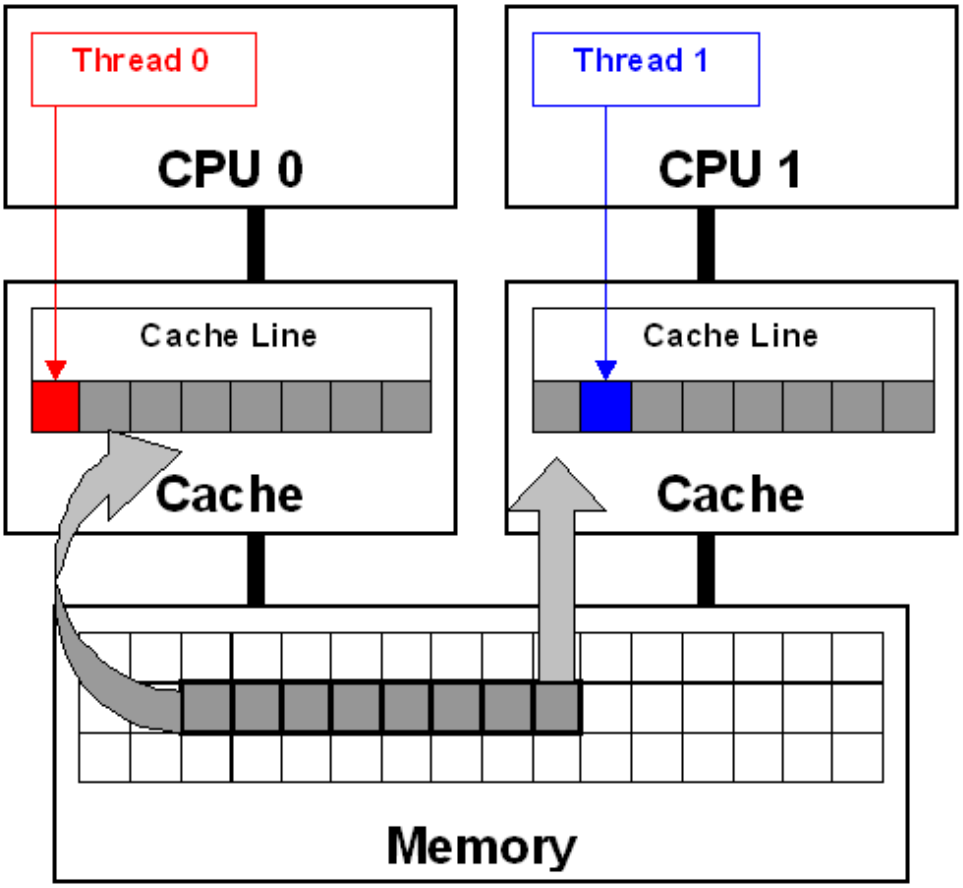


Figure 1. False sharing occurs when threads on different processors modify variables that reside on the same cache line. This invalidates the cache line and forces a memory update to maintain cache coherency.

8. **Each miss is allocated on an MSHR (Miss Status Holding Register) entry before a request to service that miss is sent to memory at each level of a memory hierarchy. So, the miss number can be calculated via MSHR allocation. Based on your knowledge of pure miss, please describe how to use MSHR to detect pure miss of L2.**

Miss Status Holding Registers (MSHRs):

- Holds enough information on all outstanding misses to complete the load. Primary and secondary misses can use the same MSHR.
- *Primary miss* is the first to a main memory block, *Secondary misses* are the subsequent misses to a block.
- The miss concurrency usually determined by the number of MSHR (Miss Status Holding Register) entries. The maximum miss concurrency is equal to the number of outstanding cache misses that MSHR can support.
- MSHR is a structured table. It records cache miss information such as access type (load/store), access address, and return register, etc. When the MSHR table is empty, there are no outstanding cache misses. When the MSHR is attached to LLC (Last Level Cache) and empty, designates there is no outstanding main memory access. When the MSHR table is full, then the cache cannot afford more cache accesses, and the CPU's memory accesses or nextlevel memory accesses are blocked due to the lack of MSHR entry. Therefore the number of MSHR entries can directly determine the miss
- when the number of MSHR entry is increased, the average hit concurrency approximately maintains the same, whereas the average pure miss concurrency constantly increases. But the larger the MSHR table is, the smaller the miss concurrency gain is.