CMPE 110 Homework #3

John Allard
February 23nd, 2015
jhallard@ucsc.edu

1. **Question #1 - Cache Coherency** Assume we have the following quad core system : (picture shown in handout).

   The processor is 32-bit and supports 32-bit virtual address and 32-bit physical address. Each core has its own dedicated IL1 and DL1 for instruction and data cache. L1 cacheline size of all the caches are 4-words. L1 is 4-way set associative, with 4KB size. core0 and core 1 share the same L2 cache. core2 and core 3 share the same L2 cache. L2 line size is 16-word, it is a 8-way set associative cache with 64KB size each. Assume the processor implements MESI cache coherence protocol. Note that for each cache, the requests are divided into two categories.
   a) requests coming from above (from processor)
   b) requests coming from below (other processors).
   Let.s indicate the requests as follows:
   Prd: Read request from the processor.
   Pwr: Write request from the processor
   RRd: Read request from a remote processor
   RRx: Read request from a remote processor with the intention to modify the data (this could be triggered by a Pwr of another processor in the system).
   BWb: Bus Write back
   BDis: Displacement, which is the cache response to a Brd or Brx request. (We can ignore this for now).

   Note that a Pwr to a local cache will trigger a RRx to all the other caches connected to the bus. a Prd will trigger a RRd. Either RRx or RRd in occasions could also trigger a BWb. Run the following accesses and complete the table. Assume all the accessing belong to the same multithreaded program. For each step, specify the coherency messages that the request will trigger, and state of the coherency state for the cachelines that changes due to the access.

2. **Question #2 - Memory Mapping Impact**

   Assume we have the following single threaded code: (Given in the handout)

   (a) **Question 2.A - Single-threaded vs Multi-threaded**
   Assume data in each of the a, b, and c matrices are stored in the memory in the following way:

   Assume the same cache hierarchy as in Question 1. How many data cache misses is the single threaded version to have? How about the multithreaded version? Does the average data memory access change between running the single threaded and multithreaded? Does it matter which thread is mapped to which processor? Justify your answer.

   ```
   addr: a[0][0]
   addr+0x4: a[0][1]
   ...
   addr+0x1C: a[0][7]
   addr+0x20: a[1][0]
   ...
   addr+0x3C: a[1][7]
   ...
   ```

   (b) **Question 2.B - New Memory Mapping**

Table 1: Q1 Calculations

	Core	Instr.	Addr.	L1 Access	L1 C-State	L1-L2 B	L2 C-State	L2 Access	L2 State	L1 State
0	read	2	0	0	0	M[0x100]	0	0		
1	write	30	1	1	0	D[0xF00]	0	0		
2	read	0	1	0	96	M[0xC00000]	0	0		
3	write	0	1	1	0	D[0x000]	0	0		
4	read	30	1	0	16	D[0x100F00]	0	0		
5	read	30	1	0	16	D[0x100F00]	0	0		
6	read	30	1	0	16	D[0x100F00]	0	0		
7	read	30	1	0	16	D[0x100F00]	0	0		
8	read	30	1	0	16	D[0x100F00]	0	0		
9	read	30	1	0	16	D[0x100F00]	0	0		
10	read	30	1	0	16	D[0x100F00]	0	0		
11	read	30	1	0	16	D[0x100F00]	0	0		

If we change the memory mapping as follows, how would it change the cache hit rate? You dont need to compute the exact cache hit rate, but provide enough justification for your answer, particularly from a locality perspective (you can list the data addresses a sample thread touches to guide you). Which version would have a better performance?

```
addr: a[0][0]
addr+0x4: a[1][0]
...
addr+0x1C: a[7][0]
addr+0x20: a[0][1]
...
addr+0x3C: a[7][1]
...
```

Answer :

(c) **Question 2.C - Pipelined Write Hit Path**

Assume you run the program multiple times. Does the cache hit rate changes assuming running only the single threaded program across different runs? How about the multithreaded across different runs? What is the source of variation?

(d) **Question 2.D - Reduction Stage**

The provided code computes the absolute difference of two blocks of an image. This is used to find the difference between consecutive frames in video encoding. Eventually the difference of two blocks needs to be converted to a scalar data. The following code snippet shows the code:

```
int reduction ( int **c ) {
int sum = 0;
for( int i = 0; i < 4; i++ ) {
for( int j = 0; j < 8; j++ ) {
sum += c[i][j];
}
}
return sum;
}
void main () {
...
adiff( a, b, c );
int sad = reduction( c ); // Sum of Absolute difference
...
```

How do you suggest parallelizing the reduction function? Ignore all other instructions, and just consider the add operation and the data dependency. Assume the add operation takes 1 cycle. What is the minimum number of cycles to finish the reduction, in theory?