Adel Danandeh

Homework #4

Q1.

(PLEASE NOTE: Matheus said in his office hours (YouTube video on Wednesday 3/11/15) that filling out the table is essentially showing our work. Based on that assumption, I didn't find it necessary to type all my thought process for this question (Figuring out how to type up our thought process in English is just a challenging task on this quesiton) PLEASE don't deduct points for not showing my thought process.)

	Core	Instr	Addr	L1 Access	Current L1 State	L1-L2 Bus	Current L2 State	L2 Access	L2 State	L1 State
1	0	Ld	0x100	Miss	I	RRd	I	Miss	E	E
2	0	Ld	0x150	Miss	I	RRd	I	Miss	E	E
3	1	St	0x100	Miss	I	RRx	E	Hit	M	M
4	1	St	0x104	Hit	M	- *	M	- *	M	M
5	1	Ld	0x150	Miss	I	RRd	E	Hit	E	S
6	2	Ld	0x154	Miss	I	RRd	I	Miss	S	S
7	3	St	0x100	Miss	I	RRx/Bwb	I	Miss	M	M
8	3	St	0x150	Miss	I	RRx	S	Hit	M	M
9	1	Ld	0xA00100	Miss	I	RRd	I	Miss	E	E
10	1	Ld	0xB00100	Miss	I	RRd	I	Miss	E	E
11	1	Ld	0x100	Miss	I	RRd	I	Miss	S	E

*NOTE: Since we have a hit at L1, no requests will be send out to L1-L2 Bus.

Q2A.

		This is for one cacheline
Addr :	a[0][0]	miss
Addr+0x4:	a[0][1]	hit
Addr+0x8:	a[0][2]	hit
.		hit
.		
		This is for the second cachline
		miss
		hit
		hit
Addr+…	a[0][7]	hit

total of 2 cachelines => 2 misses for each

Sine first access at the beginning of each cache considers being a cold miss, so we would have a miss per each cacheline. Therefore we have 2 misses for every 2 cachelines. We would have the following situation: (assumption: I'm computing number of misses at L1)

I = 0 2.3(is miss for a,b,c. Miss =1 for each one of them) = 6 misses

I = 1 2.3(is miss for a,b,c. Miss =1 for each one of them) = 6 misses

I = 2 2.3(is miss for a,b,c. Miss =1 for each one of them) = 6 misses

I = 3 2.3(is miss for a,b,c. Miss =1 for each one of them) = 6 misses

Therefore, total number of misses is: (at L1) 24 misses

Since we fetch one cacheline for each a,b,c so therefore we have 24 cachelines fetched in the inside loop, and

I = 4 in the outer loop, so we have total number of 96 cachelines fetched

Therefore miss rate is = 24 / 96 = ¼

We will have the same miss rate in multithreaded on L1. But memory access time could be longer.

Let's say we have the following situation:

a[0][0]	a[1][0]	a[2][0]	a[3][0]
Core0	Core1	Core2	Core3

There will be a miss for first access let's say in core0 in L1 and a miss in L2, however, we would have a miss at core1 at L1, but we will a hit on L2 (Assume a[0][0] has address 0x0, so that being said the miss at L2 by core0 with bring in a cacheline 0x0 – 0x3C, so when there is miss at L1 in core1, there will be a hit at L2 since the cacheline was brought in to L2 by core0). So, there is going to be 1 miss for each two cores. In other words, average data memory access gets improved.

What if we have the situation below?!

a[0][0]	a[2][0]	a[1][0]	a[3][0]
Core0	Core1	Core2	Core3

There is going to be a miss for the first access at core0 (cold miss) at L1 as well as L2. But, this time we have a miss at L1 and L2 in core1, because the cacheline brought in to L2 by core0 won't satisfy the range of the address that core1 is dealing with. We will have a miss at L1 and L2 in core2 (cold miss), and then we will have a miss at L1 and L2 in core3. (since the cacheline brought in to L2 by core2 does not satisfy the address range for core3).

As we could see it does matter how threads are mapped to each processor.

Q2B.

Hit rate will decrease, because based on the mapping given there is less spatial locality. We have the situation below:

Addr+0x0: a[0][0]

Addr+0x10: a[0][1]

Addr+0x20: a[0][2]

Addr+0x30: a[0][3]

Addr+0x40: a[0][4]

Addr+0x50: a[0][5]

Addr+0x60: a[0][6]

Addr+0x70: a[0][7]

.

.

.

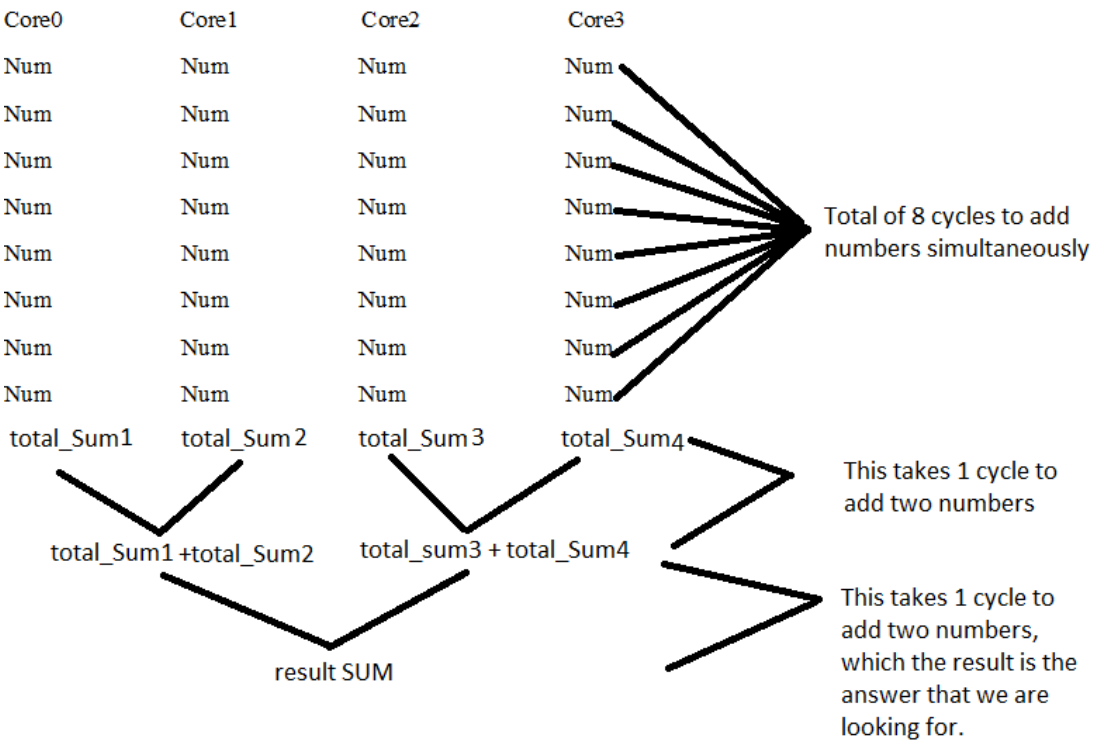
There is going to be gaps between addresses, which it requires more time to be found (or to be accessed).

Q2.C

In single thread program the number of cache hits don't change. But, if we are dealing with multi-threaded program, it really depends on how we arrange the threads (like the case we had in previous part, the number of misses was depended on how we mapped the threads to cores) and they overlap, so the order of accesses to cache hierarchy and memory changes could be arranged differently, which we could end up getting different hit rate (or miss rate).

Q2.D

In order to parallelize the operation of sum, we could do the operation of sum using 4 cores, which they add 8 number each. We have the situation below:



Therefore we will have the final sum by 8(cycles) + 1(cycle) + 1(cycle) = 10 cycles in total

In order to convert the single threaded code to multi-threaded code, we do the following:

```
int reduction ( int **c , int tid) {
  int sum = 0;
  for (int j = 0; j < 8; j++){
       sum += c[tid][j];
   }
   return sum;
}
void main(){
  .
  .
  .
   // we will have total of 8 add (assumption: we only care about add instruction) this logic sort of follows the
   // diagram shown above. We have 4 threads in which each does 8 additions.
   reduction_sum[0] = reduction(c , 0);
   reduction_sum[1] = reduction(c , 1);
   reduction_sum[2] = reduction(c , 2);
   reduction_sum[3] = reduction(c , 3);
```

```
    // This is where we add the 4 result computed by each thread.
    reduction_Sum0 = reduction_sum[0] + reduction_sum[1]; // This is adding the value of sum found by core0 and core1
    reduction_Sum1 = reduction_sum[2] + reduction_sum[3]; // This is adding the value of sum found by core2 and core3

    // At the end we add up the two sum values computed by each two cores:
    result_Sum = reduction_Sum0 + reduction_Sum1;          // computing the value of total sum.
     .
     .
     .
     return 0;
}
```