Zachary Humphries
COMP 605 HW 1

1. a)

A	[ ][0]	[ ][1]	[ ][2]	[ ][3]	[ ][4]	[ ][5]	[ ][6]	[ ][7]
[0][ ]	Miss →	Hit →	Hit →	Hit →	Miss →	Hit →	Hit →	Hit
[1][ ]	Miss	Hit	Hit	Hit	Miss	Hit	Hit	Hit
[2][ ]	Evict 0	Hit	Hit	Hit	Evict 1	Hit	Hit	Hit
[3][ ]	Evict 2	Hit	Hit	Hit	Evict 3	Hit	Hit	Hit
[4][ ]	Evict 0	Hit	Hit	Hit	Evict 1	Hit	Hit	Hit
[5][ ]	Evict 2	Hit	Hit	Hit	Evict 3	Hit	Hit	Hit
[6][ ]	Evict 0	Hit	Hit	Hit	Evict 1	Hit	Hit	Hit
[7][ ]	Evict 2	Hit	Hit	Hit	Evict 3	Hit	Hit	Hit

There are 16 total misses, including evictions.

b)

A	[ ][0]	[ ][1]	[ ][2]	[ ][3]	[ ][4]	[ ][5]	[ ][6]	[ ][7]
[0][ ]	Miss →	Evict 0	Evict 0	Evict 0	Evict 0	Evict 0	Evict 0	Evict 0
[1][ ]	Miss →	Evict 1	Evict 1	Evict 1	Evict 1	Evict 1	Evict 1	Evict 1
[2][ ]	Miss →	Evict 2	Evict 2	Evict 2	Evict 2	Evict 2	Evict 2	Evict 2
[3][ ]	Miss →	Evict 3	Evict 3	Evict 3	Evict 3	Evict 3	Evict 3	Evict 3
[4][ ]	Evict 0 →	Evict 0	Evict 0	Evict 0	Evict 0	Evict 0	Evict 0	Evict 0
[5][ ]	Evict 1 →	Evict 1	Evict 1	Evict 1	Evict 1	Evict 1	Evict 1	Evict 1
[6][ ]	Evict 2 →	Evict 2	Evict 2	Evict 2	Evict 2	Evict 2	Evict 2	Evict 2
[7][ ]	Evict 3 →	Evict 3	Evict 3	Evict 3	Evict 3	Evict 3	Evict 3	Evict 3

There are 64 total misses, including evictions.

Zachary Humphries

2. a) $\frac{10^6}{10^{12}} = 10^{-2}$ seconds per instruction

$$\overbrace{\left(10^{-2} \times \frac{10^{12}}{1000}\right)}^{\text{time for instructions}} + \overbrace{\left(10^9 (1000-1)\, 10^{-9}\right)}^{\text{time for messages}} = 10,000,999 \text{ seconds}$$

b)

$$\left(10^{-2} \times \frac{10^{12}}{1000}\right) + \left(10^9 (1000-1)\, 10^{-3}\right) = 1,009,990,000 \text{ seconds}$$

3. a) In a snooping cache coherence, when a core updates a variable (in this case, x), the core sends a broadcast to the other processors that the cache line containing x has been updated. However, in write-back caches, data is only updated in memory only when the cache line is ready to be replaced. Assuming that core 0's cache does not need to be replaced in the time that core 1 executes $y=x$, the old value of x will be assigned to y so $y \neq 5$.

b) In a directory-based protocal, a directory stores the status of each cache line. When a core updates the value of a variable (in this case x), the directory is consulted & the cache controllers of the cores that contain the cache line with x in their caches are invalidated. If x were in core 1's cache, the directory controller would have invalidated the cache line that contains the value of x. However since x is not in core 1's cache, it will read that the line of memory having x is invalid & will request core 0 for the updated cache line. Thus, $y=5$

Zachary Humphries

4. a) i.

p \ n	10	20	40	80	160	320
1	100	400	1600	6400	25600	102400
2	51	201	801	3201	12801	51201
4	27	102	402	1602	6402	25602
8	15.5	53	203	803	3203	12803
16	10.25	29	104	404	1604	6404
32	8.125	17.5	55	205	805	3205
64	7.5625	12.25	31	106	406	1606
128	7.78125	10.125	19.5	57	207	807

ii. As p increases & n stays fixed, the time taken to complete the parallelized program decreases at a marginally decreasing rate

iii. As n increases & p stays fixed, the time taken to complete the parallelized program increases by a factor of $(\Delta n)^2$

b) i. Efficiency is defined as...

$$E(p,n) = \frac{T_{serial}(n)}{pT_{parallel}(p,n)} \rightarrow \frac{T_{serial}}{p\left(\frac{T_{serial}}{p} + T_{overhead}\right)} \rightarrow$$

$$E(p,n) = \frac{1}{1 + \left(p\frac{T_{overhead}}{T_{serial}}\right)}$$

If $T_{overhead} < T_{serial}$ as n increases, then

$$\lim_{n\to\infty} \frac{T_{overhead}(p,n)}{T_{serial}(n)} = 0$$

so $E(p,n)$ approaches 1 as n increases.

ii. If $T_{overhead} > T_{serial}$ as n increases, then

$$\lim_{n\to\infty} \frac{T_{overhead}(p,n)}{T_{serial}(n)} = \infty$$

so $E(p,n)$ approaches 0 as n increases

Zachary Humphries

5. a) Efficiency is defined as...

$$E(p,n) = \frac{T_{serial}(n)}{pT_{parallel}(p,n)}$$

To maintain efficiency

$$\frac{T_{serial}(n)}{pT_{parallel}(p,n)} = \frac{T_{serial}(xn)}{(kp)T_{parallel}(kp,xn)} \longrightarrow$$

$$\frac{n}{p(\frac{n}{p} + \log_2(p))} = \frac{xn}{kp(\frac{xn}{kp} + \log_2(kp))} \rightarrow \frac{n}{n+p\log_2(p)} = \frac{xn}{xn+kp\log_2(kp)}$$

$$(xn)(n+p\log_2(p)) = n(xn+kp\log_2(kp))$$

$$xn^2 + xnp\log_2(p) = xn^2 + nkp\log_2(kp)$$

$$xnp\log_2(p) = nkp\log_2(kp)$$

$$x = \frac{nkp\log_2(kp)}{np\log_2(p)}$$

$$x = k\log_2(kp-p)$$

b) $k = \frac{16}{8} = 2 \quad x = 2\log_2(16-8) \quad x = 6$

n should increase by a factor of 6

c) The parallel program is weakly scalable since an increase in k must correspond with an increase in x.

Zachary Humphries

6. a) $rtime = utime + stime + time_{idle}$

$time_{idle}$: the time the program is idle (for example, waiting for messages)

b) For programmer A, can use the equation above to find...

$time_{idle} = rtime - utime - stime$

which allows the programmer to check if the MPI process is spending too much time waiting for messages since $time_{idle}$ includes time waiting for messages

c) Programmer B cannot use programmer A's functions to determine time spent waiting for messages since the equation in (a) becomes...

$rtime = utime + stime$

because utime now includes $time_{idle}$. There isn't a way now for $time_{idle}$ to be calculated by programmer B.