Home Assignment 2

Yuxuan Jing

February 2020

Question 2.1

Given:

- Each of the functional units took the same amount of time.
- Fetch and store each take 2 nanoseconds (ns).
- Remaining operations each take 1 nanosecond.

Remember that the floating point addition has 7 steps:

- Fetch operands
- Compare exponents
- Shift one operand
- Add
- Normalize result
- Round result
- Store result

2.1.a

A floating point addition take:

$$t_{add} = t_{fetch} + t_{compare} + t_{shift} + t_{add} + t_{normalize} + t_{round} + t_{store}. \quad (1)$$

$$t_{add} = 2 + 1 + 1 + 1 + 1 + 1 + 2 = 9[ns]. \quad (2)$$

2.1.b

An unpipelined addition of 1000 pairs of floats take:

$$t_{1000un} = 1000 \times t_{add} = 9000[ns]. \quad (3)$$

2.1.c

Since one fetch or store operation takes 2 ns. After the first result is produced, each next result takes 2 ns. So the total time takes:

$$t_{1000pip} = t_{add} + t_{store} \times (1000 - 1) = 9 + 2 \times 999 = 2007[ns]. \tag{4}$$

2.1.d

When a level 1 miss cache occurs, we need to search for level 2 cache one time and level 1 cache one time:

$$t_{search} = t_{lvl1} + t_{lvl2} = 2 + 5 = 7[ns]. \tag{5}$$

When a level 2 miss cache occurs, we need to search for level 3 cache once, level 2 cache one time and level 1 cache one time:

$$t_{search} = t_{lvl1} + t_{lvl2} + t_{lvl3} = 2 + 5 + 50 = 57[ns]. \tag{6}$$

Question 2.3

Given:

$$MAX = 8 \tag{7}$$

$$\text{number of cache lines} = 4 \tag{8}$$

Assume the each line can store 4 doubles.

For row major matrix, every 4 read come with a cache miss. $2 \times 8 = 16$ misses occur. For column major matrix, every read come with a cache miss. $8 \times 8 = 64$ misses occur.

Question 2.4

Given:

```
byte offset = 12 bits
virtual page number = 20 bits
```

Answer:

```
Maximum pages a program can have: 2^20 [pages]
with:
page size: 2^12 = 4 [KB]
```

Question 2.6

It takes 10 cycles to load a single 64-bit word from memory. So **10** banks are needed so that a stream of loads can, on average, require only one cycle per load.

$$n_{banks} = t_{tot}/t_{ave} = 10/1 = 10 \tag{9}$$

Question 2.10

Using p to represent the number of processors.

$$p = 1000. \tag{10}$$

Number of instructions executed by every processor is $n = 10^{12}/p = 10^9$. The time to execute instructions is $t_{exe} = 1000 \times 10^6/10^{12} = 1000[sec]$. The number of message is $n_{message} = 10^9(p-1) = 10^9 \times 999$.

2.10.a

$$t_{tot.a} = t_{exe} + n_{message} \times t_{ave.message} = 1000 + 10^9 \times 999 \times 10^{-9} = 1999[sec]. \tag{11}$$

2.10.b

$$t_{tot.a} = t_{exe} + n_{message} \times t_{ave.message} = 1000 + 10^9 \times 999 \times 10^{-3} = 999001000[sec]. \tag{12}$$

Question 2.11

In distributed system (DS) for a p node DS:

- A ring DS has p links.
- A toroidal mesh DS has $2p$ links.
- A fully connected network DS has $\frac{p \times (p-1)}{2}$ links.
- A n-dimension hypercube DS has $\frac{2^p \times p}{2}$ links.
- A crossbar interconnect DS has p^2 switches, and $8 \times p^2$ links.
- An omega network DS has $2 \times plog_2(p)$ switches, and $16 \times plog_2(p)$ links.

Question 2.12

Assume n is the number of nodes in a one-dimension row. (2D cube has n^2 nodes and 3D cube has n^3 nodes. Using q to present bisection width and p as number of processors.

2.12.a

The square planar mesh's bisection width is equal to the number of nodes in one row:

$$q = n = p^{\frac{1}{2}}. \tag{13}$$

2.12.a

The three-dimensional mesh's bisection width is equal to the number of nodes in one plain:

$$q = n \times n = n^2 = p^{\frac{2}{3}}. \tag{14}$$

Question 2.15

2.15.a

The value of y will be 5, since the following steps happen in order:

- a clean cache line with x
- a dirty cache line with $x = 5$.
- shared memory update $x = 5$, be clean
- core 1 get the value of x, and assign $y = x = 5$

2.15.b

The value of y will be the none or any former x's value in it's own distributed cache.

2.15.c

The problem for the second part is that the update of the distributed memory cannot be record in the shared memory. One potential solution could be build a table of cache line status to mark which line is accessible and which one is not in the shared memory.

Question 2.16

2.16.a

Given:

$$T_{serial} = n^2, \tag{15}$$

and

$$T_{parallel} = n^2/p + log_2(p). \tag{16}$$

We can derive the formula for **speedup** and **efficiency**:

$$S = \frac{T_{serial}}{T_{parallel}} = \frac{n^2}{n^2/p + log_2(p)}, \tag{17}$$

and

$$E = \frac{T_{serial}}{p \cdot T_{parallel}} = \frac{n^2}{n^2 + p \cdot log_2(p)} = \frac{1}{1 + \frac{p \cdot log_2(p)}{n^2}}. \tag{18}$$

speedup	10	20	40	80	160	320
1	1	1	1	1	1	1
2	1.960784	1.99005	1.997503	1.999375	1.999844	1.999961
4	3.703704	3.921569	3.9801	3.995006	3.99875	3.999688
8	6.451613	7.54717	7.881773	7.970112	7.992507	7.998125
16	9.756098	13.7931	15.38462	15.84158	15.9601	15.99001
32	12.30769	22.85714	29.09091	31.21951	31.80124	31.95008
64	13.22314	32.65306	51.6129	60.37736	63.05419	63.7609
128	12.85141	39.50617	82.05128	112.2807	123.6715	126.8897

Figure 1: Speedup

efficiency	10	20	40	80	160	320
1	1	1	1	1	1	1
2	0.980392	0.995025	0.998752	0.999688	0.999922	0.99998
4	0.925926	0.980392	0.995025	0.998752	0.999688	0.999922
8	0.806452	0.943396	0.985222	0.996264	0.999063	0.999766
16	0.609756	0.862069	0.961538	0.990099	0.997506	0.999375
32	0.384615	0.714286	0.909091	0.97561	0.993789	0.99844
64	0.206612	0.510204	0.806452	0.943396	0.985222	0.996264
128	0.100402	0.308642	0.641026	0.877193	0.966184	0.991326

Figure 2: Efficiency

We can get the result for $n = 10, 20, 40, ..., 320$ and $p = 1, 2, 4, ..., 128$, in figure 1 and 2.

- As p is increased and n is held fixed, speendup increased and efficiency decreased.
- As p is fixed and n is increased, speendup increased and efficiency increased.

2.16.b

Suppose:

$$T_{parallel} = T_{serial}/p + T_{overhead}. \tag{19}$$

and we fix p and increase the problem size.

$$E = \frac{T_{serial}}{p \cdot T_{parallel}} = \frac{T_{serial}}{p \cdot (T_{serial}/p + T_{overhead})} = \frac{T_{serial}}{T_{serial} + p \cdot T_{overhead}}. \tag{20}$$

Simplify:

$$E = \frac{1}{1 + p \cdot \frac{T_{overhead}}{T_{serial}}}. \tag{21}$$

Remember we are fixing the value of p, We can draw the conclusion that:

- if $T_{overhead}$ grows more slowly than T_{serial}, the parallel efficiency will increase as we increase the problem size.
- if $T_{overhead}$ grows faster than T_{serial}, the parallel efficiency will decrease as we increase the problem size.

Question 2.19

Given:

$$T_{serial} = n, \tag{22}$$

$$T_{parallel} = n/p + log_2(p). \tag{23}$$

Efficiency:

$$E = \frac{T_{serial}}{p \times T_{parallel}} = \frac{n}{p \times (n/p + log_2(p))} = \frac{n}{n + p \times log_2(p)}. \tag{24}$$

Simplify:

$$E = \frac{1}{1 + \frac{p \times log_2(p)}{n}}. \tag{25}$$

We increase p by a factor of k, $p_{new} = k \times p$:

$$E_{new} = \frac{1}{1 + \frac{p_{new} \times log_2(p_{new})}{n_{new}}} = \frac{1}{1 + \frac{kp \times log_2(kp)}{n_{new}}}. \tag{26}$$

If we want to keep the efficiency constant, $E = E_{new}$:

$$E = \frac{1}{1 + \frac{p \times log_2(p)}{n}} = E_{new} = \frac{1}{1 + \frac{kp \times log_2(kp)}{n_{new}}}. \tag{27}$$

Solve:

$$\frac{p \times log_2(p)}{n} = \frac{kp \times log_2(kp)}{n_{new}}. \tag{28}$$

$$n_{new} = k \times log_2(k) \times n. \tag{29}$$

If we double the number of processes from 8 to 16, which means $k = 16/8 = 2$. We should increase n by $2 \times log_2(2) = 2 \times log_2(2) = 2$ times to keep the efficiency.

Since we can find a corresponding rate of increase in problem size to number of processes, so that the program always has efficiency E, This program is **scalable**.

Question 2.22

Given:

```
double utime(void);// user time
double stime(void);// system time
double rtime(void);// total time
```

2.22.a

Given:

```
u = double utime(void);
s = double stime(void);
r = double rtime(void);
```

Relation:

$$r = u + s + t_{idle} \tag{30}$$

So:

$$r > u + s \tag{31}$$

2.22.a

The time waiting for messages is included in rtime, but excluded in utime or stime. We can assume, $t_{message}$ is included in idle time:

$$t_{idle} = t_{message} + t_{other} \tag{32}$$

and

$$t_{idle} = r - (u + s) \tag{33}$$

If the different between r and $u + s$ is large, we can guess that the MPI process is spending too much time waiting for messages. But the cause of increasing idle time could be other factors.

2.22.c

If the time waiting for message is counted as user time, the above equation cannot be used to determine whether an MPI process is spending too much time waiting for message.

We can guess the waiting time by checking whether the user time is abnormal large or not.