Name: Hasanat Jahan
CS381-16 Assignment 3

Answer to 2.10

(a) A single processor has 10^6 instructions per second
Each processor takes (10^12)/p instructions

Each processor sends 10^9*(p-1) messages &
Each message takes 10^-9 seconds

Here p = 1000
Each processor performs = 10^12/1000 = 10^12/10^13 = 10^9 instructions

10^9 instructions take (10^9)/(10^3) = 10^3s = 1000s
Time for instruction = 1000s
Time for message = (10^9)*(1000-1)*(10^-9) = 999s

Total time to run program = time for instruction + time for message
= 1000 + 999
= 1999s

(b) Time to send message = (10^9) * (1000-1) * (10^-3)
= 999 * 10^6
= 9.99 * 10^8
Time to run program = 0.0001 * 10^8 + 9.99 * 10^8
= 9.9901 * 10^8 s
This is a much longer time than a single processor

Answer to 2.14

As removing 12 unidirectional links makes it so that two parts are not connected, there need to be 6 links removed which is <= 8

Bisection Width = 6

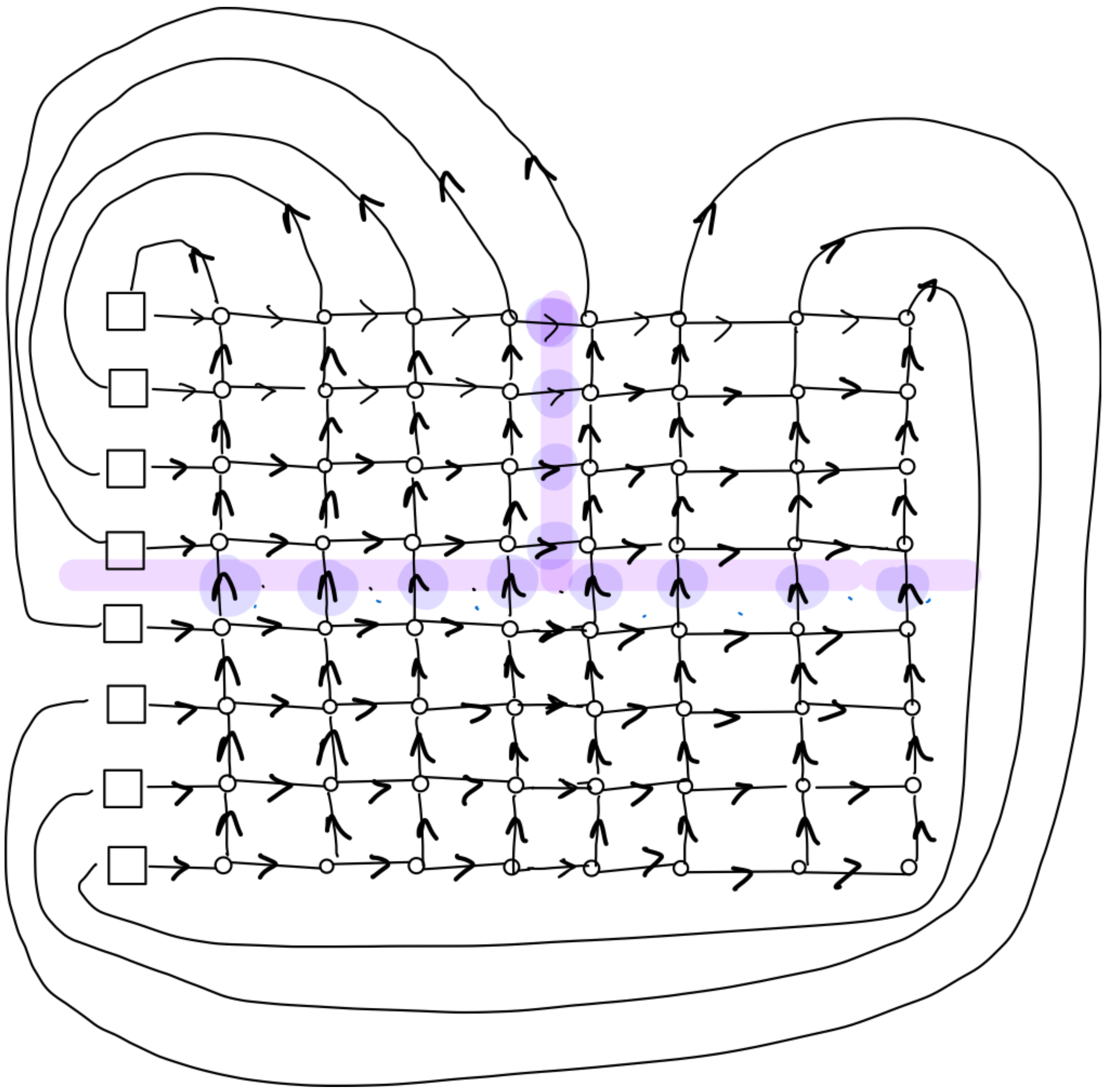

Figure 1: Crossbar Interconnect for distributed memory.

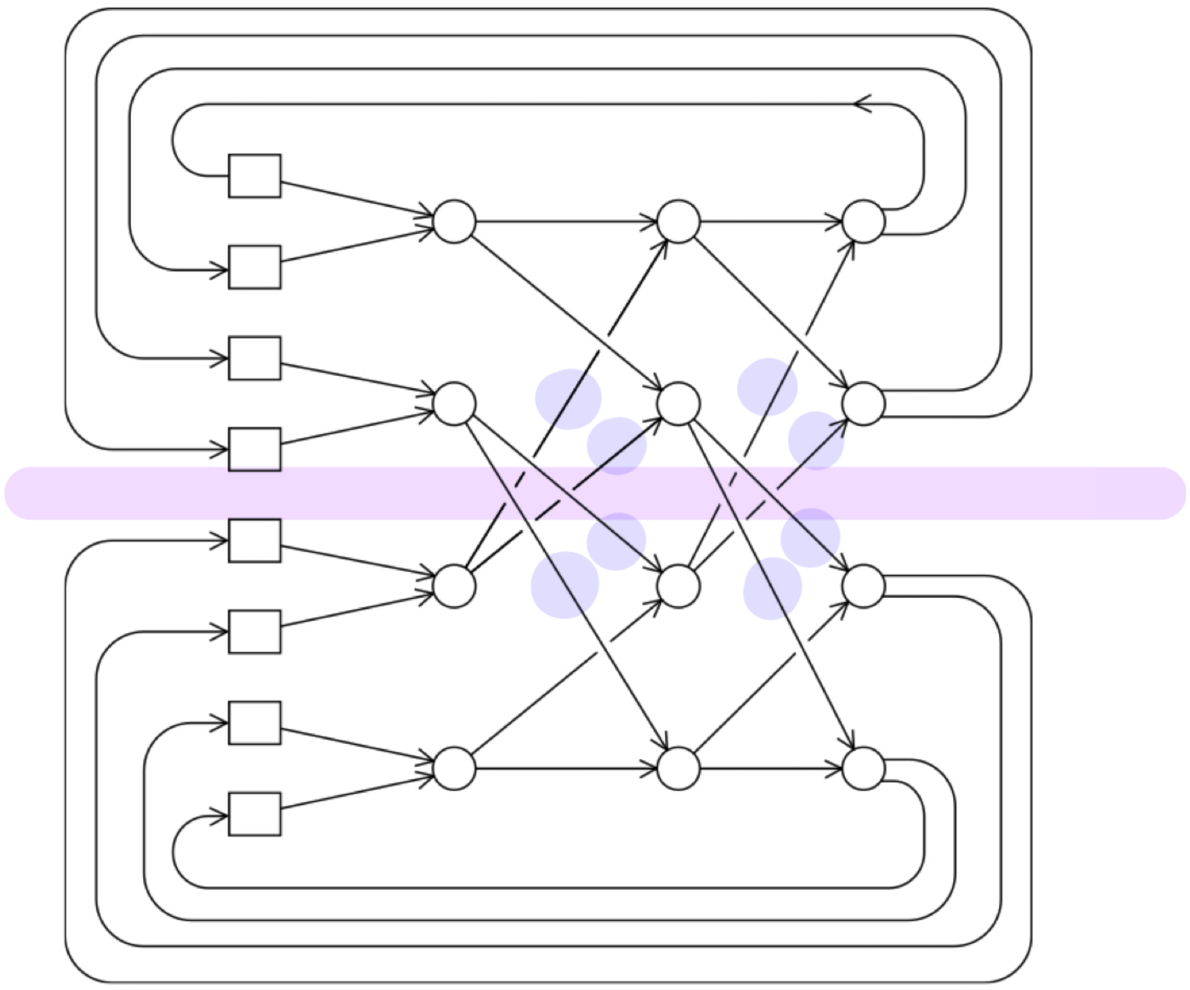

Figure 2: Omega network for distributed memory
As we need to remove 8 links to completely separate the two parts
The bisection width is therefore = 8/3 = 4

<u>Answer to 2.15</u>
(a) Core 0: x = 5
Core 1: y = x
Core 1 does not have x in its cache.
Core 0 uses snooping cache coherence but it uses write back cache which means that extra communication is necessary since the cache does not get immediately sent to memory. If the cache for x = 5 is sent to memory by the time Core 1 executes y = x, then y receives the value 5 otherwise it receives an old value of x which was already in memory.

(b) If it uses a directory based protocol, so when x = 5 is read into Core 0's cache, the directory entry corresponding to that line is updated to show that Core 0 has a copy of the line. However when a cache variable is updated only the cores storing that variable is updated. As Core 1 does not already have the value in its cache. Similarly from before if x is not updated in main memory, Core 1 will have an old copy of x, else if it is updated Core 1 will receive the updated value for x.

(c) In the last two problems, there is guaranteed cache coherence only if both cores already have the variable in cache. It should be the case that a core which does not have a value in it’s cache should have the updated value. We can do this by employing a critical section, so that a core can only get the value of x once it’s been updated in main memory.

Answer to 2.19

$m =$ new n

$$E = \frac{m}{kp\left(\frac{m}{kp} + \log_2 kp\right)}$$

$$E = \frac{m}{m + kp\log_2 kp}$$

$$\frac{m}{m + kp\log_2 kp} = \frac{n}{n + p\log_2 p}$$

$$\Rightarrow m\left(n + p\log_2 p\right) = n\left(m + kp\log_2 kp\right)$$

$$\Rightarrow mn + mp\log_2 p = nm + nkp\log_2 kp$$

$$\Rightarrow mn + mp\log_2 p - nm = nkp\log_2 kp$$

$$\Rightarrow m(\quad p\log_2 p \quad) = nkp\log_2 kp$$

$$\Rightarrow m = \frac{nk\cancel{p}\log_2 kp}{\cancel{p}\log_2 p}$$

$$\Rightarrow m = \frac{nk\log_2 kp}{\log_2 p}$$

$$\Rightarrow m = nk\left(\frac{\log_2 k}{\log_2 p} + \frac{\log_2 p}{\log_2 p}\right)$$

$$\Rightarrow m = nk\left(1 + \frac{\log_2 k}{\log_2 p}\right)$$

$T_{SERIAL} = n$

$T_{PARALLEL} = n/p + \log_2(p)$

$$E = \frac{T_{serial}}{p * T_{parallel}}$$

$$E = \frac{n}{p\left(\frac{n}{p} + \log_2 p\right)}$$

$$E = \frac{n}{n + p\log_2 p}$$

$k = 2, p = 8$

$$m = 2n\left(1 + \frac{\log_2 2}{\log_2 8}\right)$$

$$m = 2n\left(1 + \frac{1}{3}\right)$$

$$m = 2n\left(\frac{4}{3}\right)$$

$$m = \frac{8n}{3}$$

$$m = nk\left(1 + \frac{\log_2 k}{\log_2 p}\right)$$

As $k \to \infty$, the program is scalable to a limit. The effeciency remains the same with increasing k upto a limit.

Answer to 2.20

A program is strongly scalable if we can increase the number of threads/processors but keep the problem size the same without losing efficiency.

Therefore a program that obtains linear speedup should be strongly scalable as we increase speed linearly with increased processors without increasing problem size.