Part 1, Theory

Problem 1

a)

- MPI_send preforms a blocking send. Upon returning you don't know if the message has successfully been sent, you only know that the buffer used for sending is available again.

- MPI_Ssend preforms a blocking synchronous send. Does not return until the message has been matched with a receive on the receving end. Thus it returns when the message has been successfully recieved or an error has occured.

- MPI_Isend preforms a nonblocking send. But you must make sure that you don't use the sending buffer before you know that MPI is done using it. I.e. you can not use the buffer before you've done a successful wait/test or you <u>know</u> that the message has been matched with a recieve on the recieving end.

All the functions return a int value which returns various error or success codes.

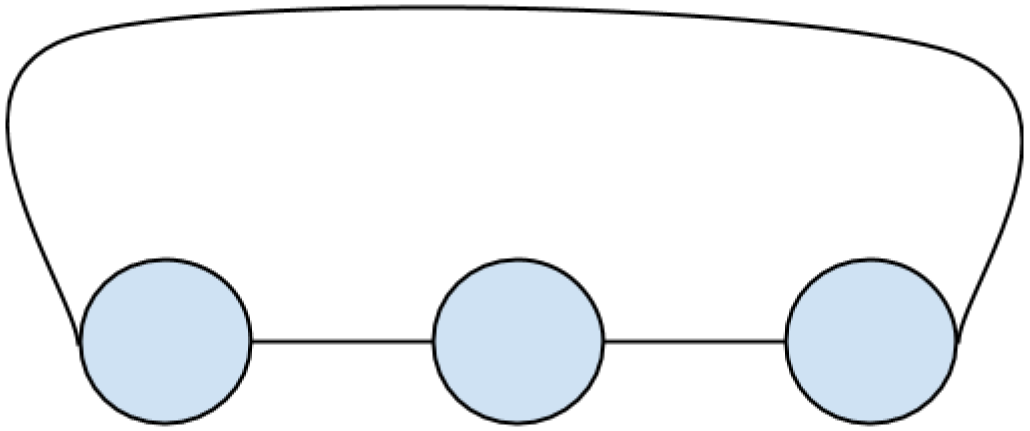
b) The book, Parallel computing by Peter S. Pacheco, defines a communicator as a collection of processes that can send messages to each other. It provides each process with an ID (rank) and orders the processes in a certain topology.

A cartesian communicator is when you structure the processes in a communicator in a software grid. The cartesian communicator provides all the processes with coordinates that defines where they're placed in the grid. MPI also proved several functions for managing the grid, e.g. MPI_Cart_Rank, MPI_Cart_Shift, MPI_Cart_coords etc.

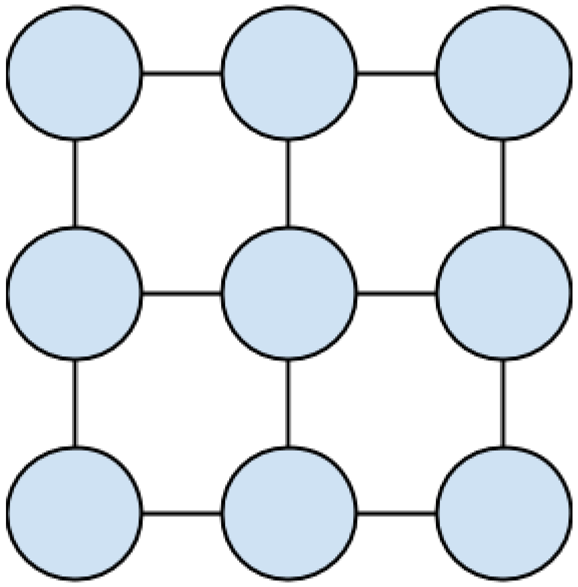
Problem 2

a)

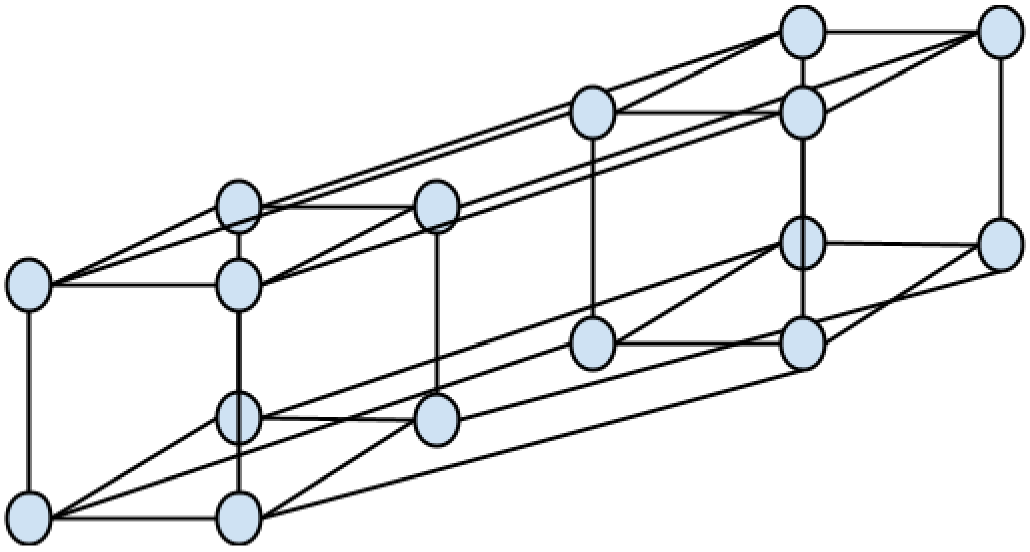
1. Ring



2. Grid



3. Hypercube(4-dimensional):



b) A hypercube consists of $p = 2^d$ nodes, where each node has a switch of size d. Every time we increase the dimension by 1 we double the number of processes. Thus $p_d = 2 * p_{d-1}$ $for\ d \geq 1$. Since the new hypercube p_d has a switch size of d, we see that the number of edges increases with p_{d-1} from p_{d-1} to p_d. Thus we easily see that p_d consists of two p_{d-1} hypercubes connected with p_{d-1} edges. This gives us that the bisection width *bw* of p_d is equal to p_{d-1}. Thus we get $p_d = 2 * bw$ => $bw = p_d/2$.

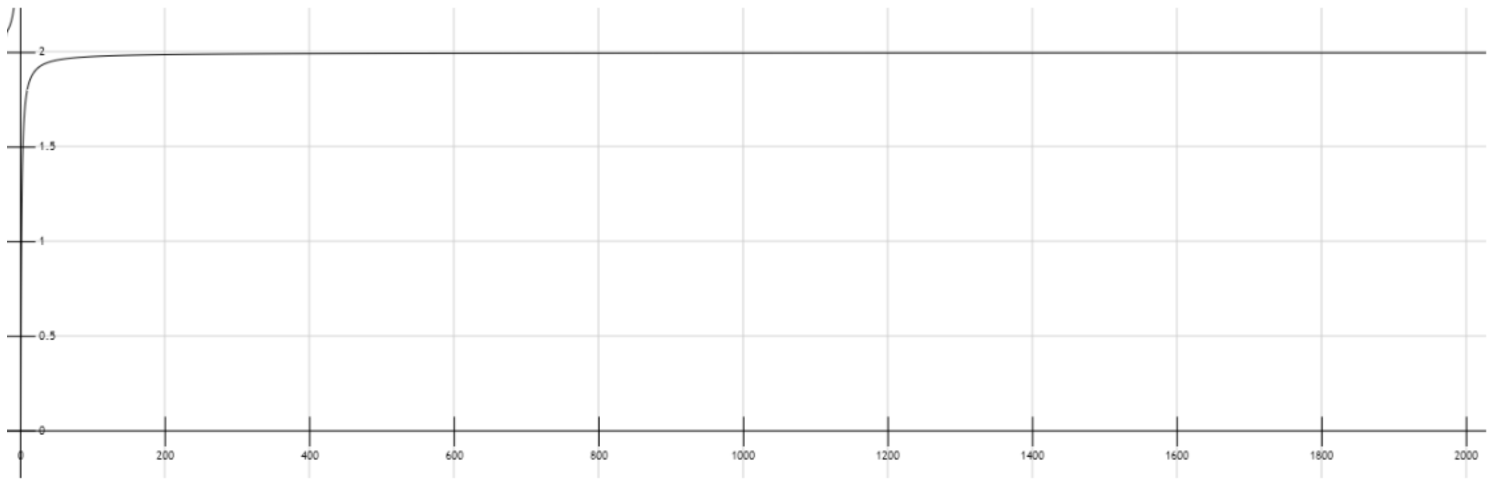
Problem 3

a) Amdahl's law gives:

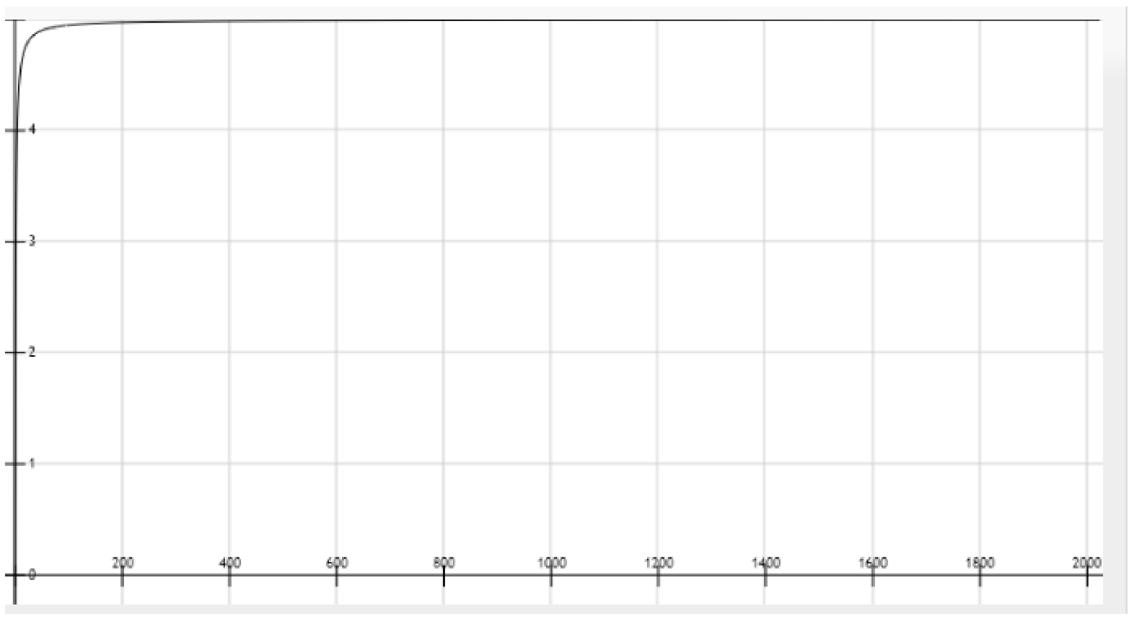
$$S(n) = \frac{1}{B + \frac{1}{n}(1-B)}$$

Where B is the fracation of the program that is strictly serial and n is the number of threads in execution. It gives the following graphs:

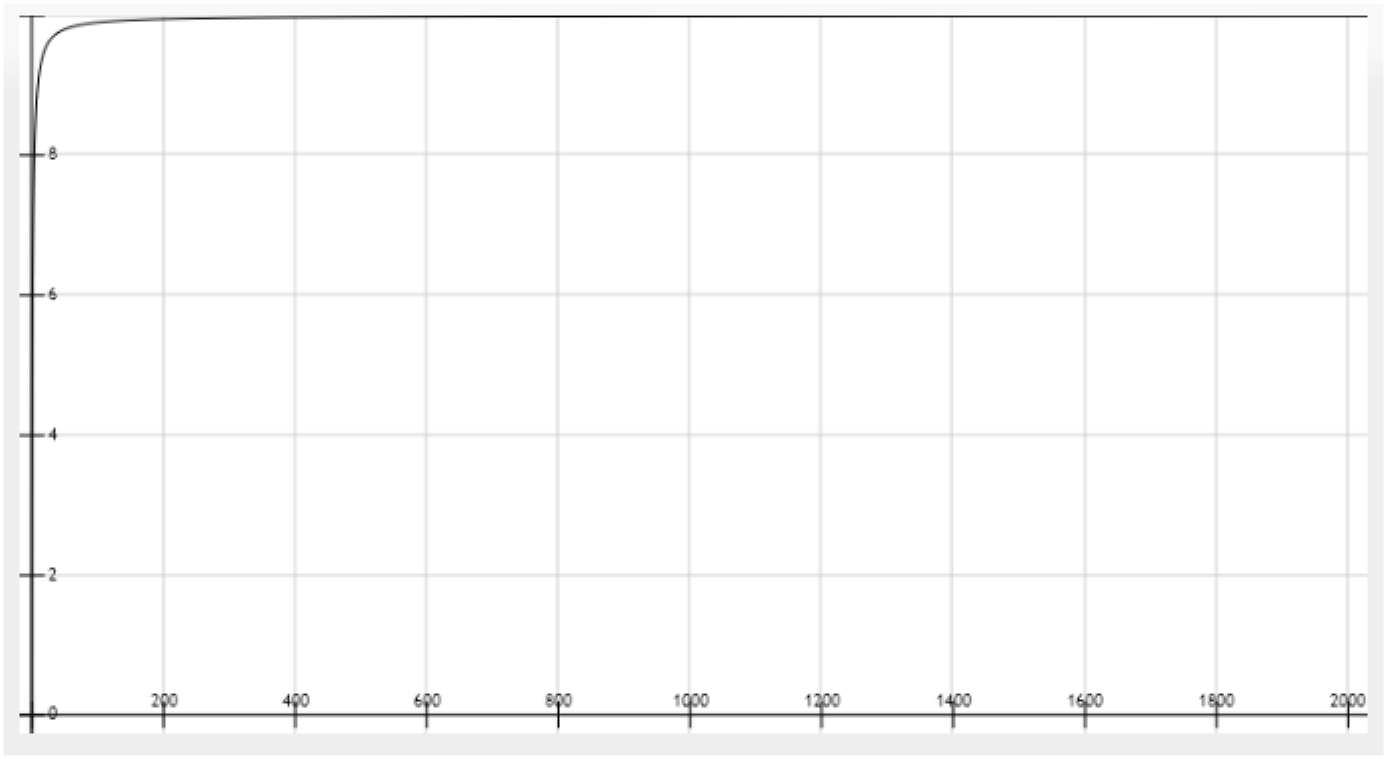
1) B = 0.5, n = [0, 2048]. It can be improved to be twice as fast.



2) B = 0.2, n = [0, 2048]. The maximum improvement is 5 times as fast.



B = 0.1. n = [0, 2048]

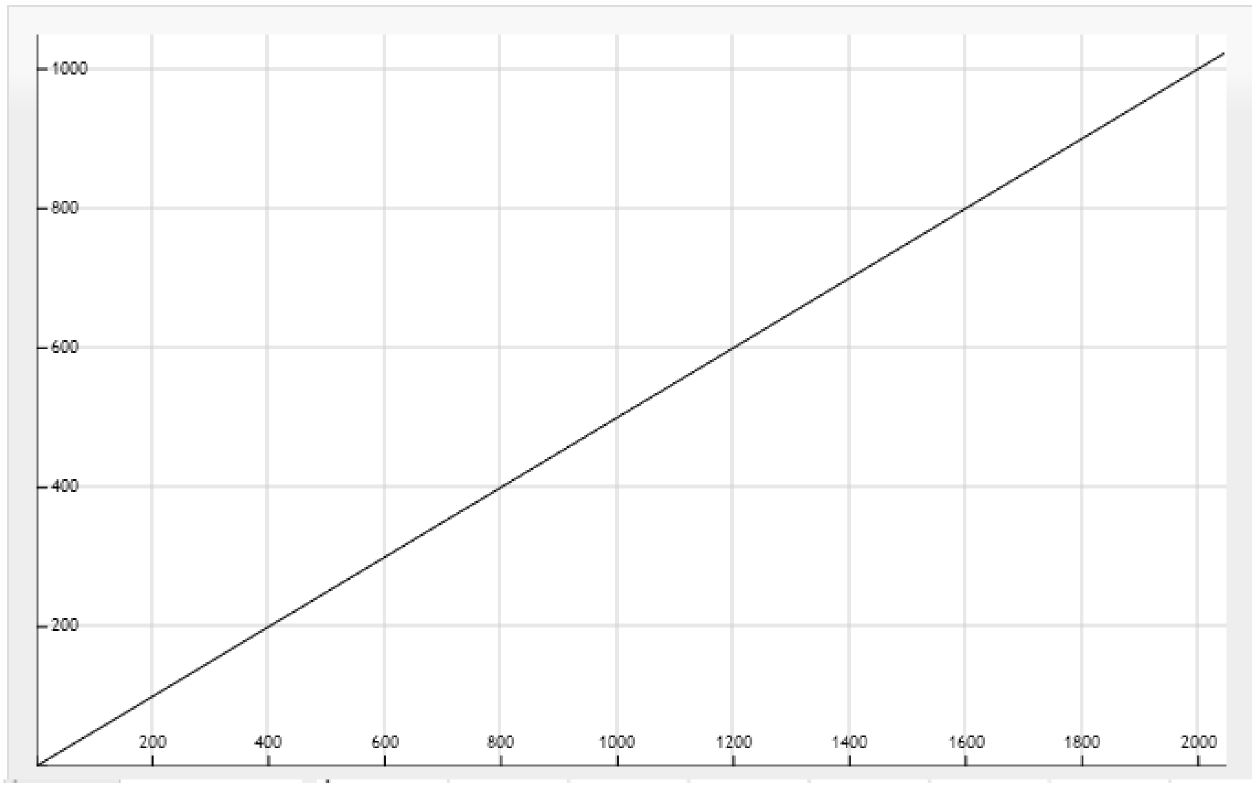


b) Gufstafson's law states that:

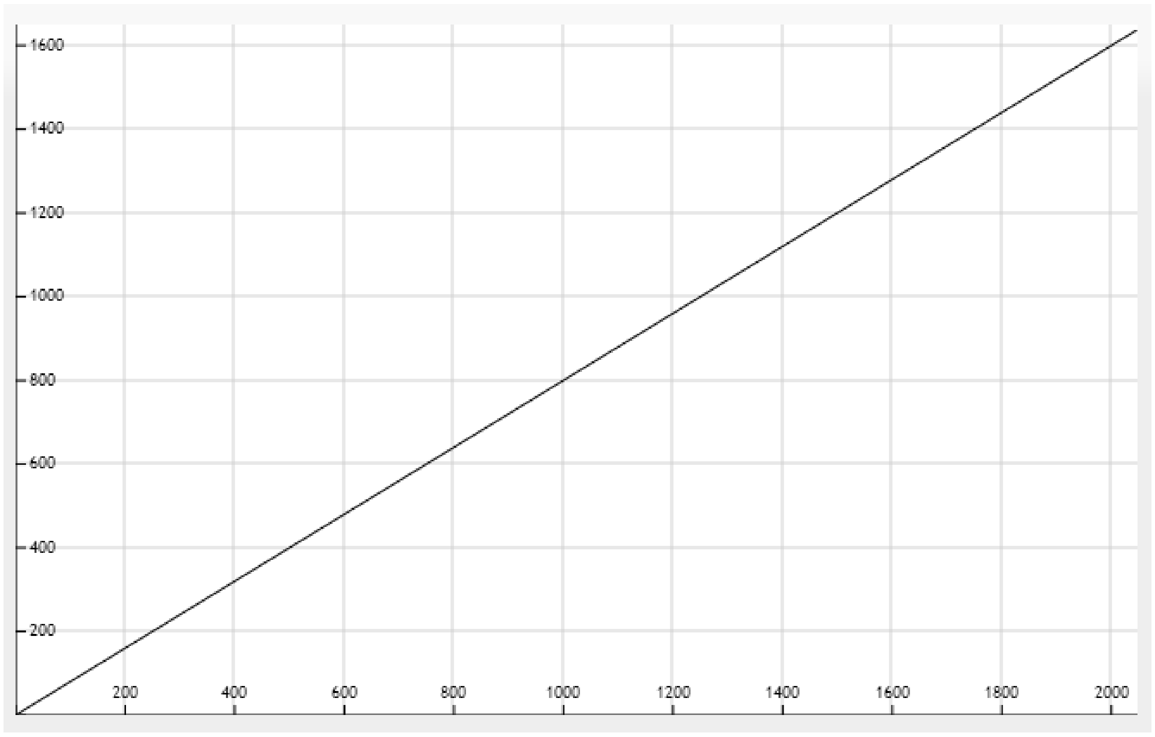
$$S(P) = P - B(P-1)$$

Where S is the speedup, P is the number of processes and B is fraction of the program that is non-parallelelizable faction of any parallel process. This gives the following graphs:

B = 0.5, P = [1, 2048]

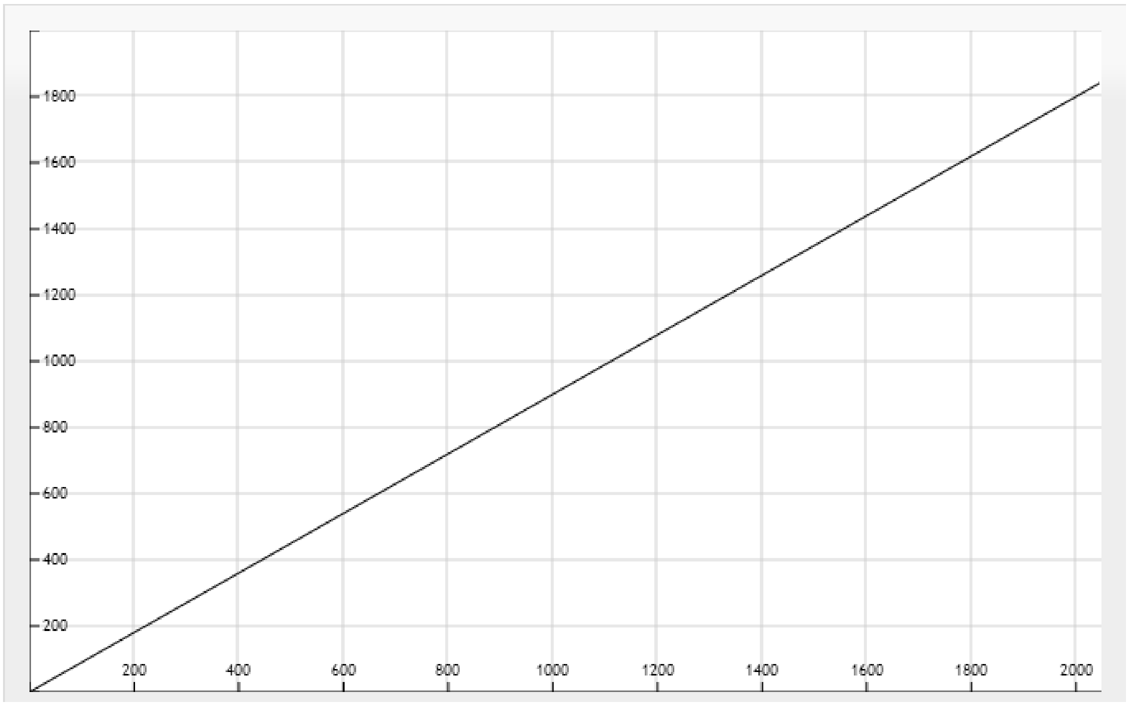


B = 0.2, P = [1, 2048]:



B = 0.1, P

= [1, 2048]:



b) Amdahl's law defines the serial fraction as $B_a = \frac{time_s}{time_s + time_p(1)}$ while Gustafon's law defines it as $B_g = \frac{time_s}{time_s + time_p(P)}$.

Problem 4

a)
Choice 1: **128x128**
Given the formula:
1024*1024=64(x*x)

x = $\sqrt{\frac{1024*1024}{64}}$ = 128

Choice 2: **64*1024**

Choice 3: **1024*64**

b)
For choice 1:

- There will be 4 grids (the corner grids) that needs to send two borders.
- 62*4 (the edge grids) needs to send three borders.
- 62*62 grids need to send 4 borders.

Thus we need to send 4*2*128 + 62*4*3*128 + 62*62*4*128 = 2064384 bytes.

For choice 2 and 3: There will be 1024/64 - 2 = 16 - 2 = 14 grids that need to exchange southern and northern borders. And 2 grids that only need to exchange one border. Thus the total number of bytes sent is 14*1024*2 + 2*1024 = 30720 bytes.