Problem 1

first, assume n is evenly divisible by P,

we have P cores in general.

∴ the sudo code is :

```
quo = n// P;
remain = n % P;
if ( my_ ind < remain)
{    my_first_i = my_ind * (quo +1);
}
else
{
     my_first_i = my_ind * (quo) + remainder;
}
  my_last_i = my_first_i + quo.
```

Problem 2.

from the question, we know each element follows the arithmetic series. Thus, one way of assigning is pairing the first element and the last into the core and then one by one assigned.

the psudo code is :

```
q = n // P ;
r = n % P :
  my_core_i = arr[i] for (i in range (0, r ))
  // assign the remainder to r cores
    iteratively e.g: if r= 2 , assign 0, 1 elements to core_0, core_1.
```

```
arr = arr[r::] // decrease the size of array.
if (q % 2 == 0): // q is even
    i=0
    while (len(arr) != 0):          // iterative assign elements to cores.
        my_core_i <- arr[0: q/2]    first q/2 and last q/2.
        my_core_i <- arr[len(arr) - q/2, len(arr)]
        arr = arr[q/2: len(arr) - q/2]
        i += 1.
else:                               // when q is odd
    i=0                             // iterative assign first q/2 and last q/2
    while (len(arr) != 0):          to every core and assign the middle element
        my_core_i <- arr[0: q//2]   to cores.
        my_core_i <- arr[len(arr) - q//2, len(arr)]
        my_core_i <- arr[len(arr)/2]
        arr = arr[q/2: len(arr) - q/2]
        arr = arr.pop(len(arr)/2)
        i += 1.
```

this psudo code uses the property that the array has the visiting time of arithmetic series. Thus, if we pick the first q/2 and last q/2 elements iteratively for different cores, all of the cores will have an averaged time of instruction.

Thus, every cores will have the tasks assigned almost evenly.

Problem 3.

Based on the hints given from the problem, we can have the psudo code below like this:

```
divisor =2,

core_difference =1

total= my_value

while ( divisor <=P)

{       if ( my_ind % divisor == 0 )
        {
          partner = my_ind + core_difference
           (receive from partner)
           total += received _value

         }
        else
        {
          partner = my_ind - core_difference
          (send  to  partner).
         }
        divisor *=2
        core_difference *=2

  }
```

Problem 4.

(a). $\because$ it has P cores

$\therefore$ it has $P-1$ receives and $P-1$ additions.

(b). $\because$ it is the tree-structured global sum

$\therefore$ # add = $\log_2 P$ where P is the number of cores.

$\therefore$ it will have $\log_2 P$ receives and $\log_2 P$ additions.

Problem 5.

(1). $\because$ there are 10^{12} instructions and a single processor can solve the problem in 10^6 seconds, which means 10^6 instruction per second

Also, we now have p processors and each executes $10^{12}/p$ instruction

$$\therefore \quad T_{ins\text{-}one} = \frac{10^{12}/p}{10^6} = \frac{10^6}{p}$$

which is the instruction execution time for one processor

$\because$ each processor must send $10^9(p-1)$ messages.

$\therefore$ $T_{send} = 10^9(p-1) \times t_s$

(where t_s is the time for send one message)

$$\therefore \quad T_{run} = T_{ins\text{-}one} + T_{send}$$

$$= \frac{10^6}{p} + 10^9(p-1) \cdot t_s.$$

Here, $p = 1000$ and $t_s = 10^{-9}$

$$\therefore \quad T_{run} = \frac{10^6}{10^3} + 10^9(10^3-1) \cdot 10^{-9}$$

$$= 10^3 + 10^3 - 1 = 2 \cdot 10^3 - 1 = 1999 \text{ s}$$

(2). $\because$ $T_{run} = T_{ins} + T_{send}$

$$= \frac{10^6}{p} + 10^9 (p-1) t_s$$

and $p = 10^3$, $t_s = 10^{-3}$

$$\therefore \; T_{run} = \frac{10^6}{10^3} + 10^9 (10^3 - 1) \cdot 10^{-3}$$

$$= 10^3 + 10^6 (10^3 - 1)$$

$$= 10^3 + 10^9 - 10^6$$

$$= 10^9 - 10^6 + 10^3 \text{ s.}$$

$$\approx 9.99001 \times 10^8 \text{ s.}$$

Problem 6.

(A). $\because$ there are p processors.

for a ring structure :

$\#$ links = $\boxed{p}$

Max_dis = $\boxed{\lfloor \frac{p}{2} \rfloor}$

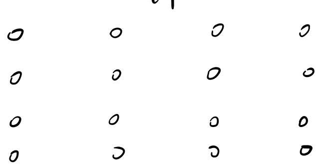
(B) for a 2-D Torus, assume we have p processors

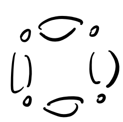
and we assume there are n rows and m columns

where $n^2 = p$. Then the structure is like this :

$\sqrt{p}$

$\sqrt{p}$





Since for each row, $\#$ links $= \sqrt{p}$

$\therefore$ $\#$ row_link $= n^2 = p$

Also, for each col, $\#$ links $= \sqrt{p}$

$\therefore$ $\#$ col_link $= n^2 = p$

$\therefore$ # total_link = #row_link + # col_link = $\boxed{2P}$

Also, max_dis = row_max_dis + col_max_dis

$$= \lfloor \frac{\sqrt{P}}{2} \rfloor + \lfloor \frac{\sqrt{P}}{2} \rfloor = \boxed{2 \cdot \lfloor \frac{\sqrt{P}}{2} \rfloor}$$

(C). Then, for a hypercube, if we have total processors of P,

we can have relationship:

dimension	processors	links
1	2	1
2	4	4
3	8	12
4	16	32

Based on the table, we can find the formula of edges based on the dimension and the processors:

$$\# \text{ links} = \frac{P}{2} \cdot \log_2 P = \boxed{\frac{P \log P}{2}}$$

Then, we can find the max distance is the dimension:

$$\text{max\_dis} = \boxed{\log P}$$

(D). Since it is the fully Connected Network,

each processor will have links to P-1 processors.

$$\therefore \# \text{ links} = P \cdot (P-1)/2 = \boxed{\frac{P^2 - P}{2}}$$

Since all the processors can have direct connection with any other processor,

$$\therefore \text{max\_dis} = \boxed{1}$$

Problem 7.

(a). Since it is a planar mesh with p processors,

the bisection width is only half of the toroidal mesh,

$\therefore$ bisec_width $= \sqrt{p} = p^{\frac{1}{2}}$

(b). $\because$ it is a three-dimensional mesh with p processors

$\therefore$ the bisec width is the area of its bisection plane,

$\therefore$ bisec_width $= (\sqrt[3]{p})^2 = p^{\frac{2}{3}}$

Problem 8.

$\because$ $T_{serial} = n$ and $T_{parallel} = \frac{n}{p} + \log_2(p)$

$\therefore$ $Eff = \frac{Speedup}{p} = \frac{T_{serial}}{p \cdot T_{parallel}} = \frac{n}{p \cdot (\frac{n}{p} + \log_2(p))} = \frac{n}{n + p\log p}$

$\because$ now we increase p by a factor of k.

$\therefore$ assume we increase n by a factor of a to maintain constant efficiency.

$\because$ $Eff^* = Eff$

$\therefore$ $\frac{an}{an + kp \cdot \log_2(kp)} = \frac{n}{n + p\log_2 p}$

$\therefore$ $a\cancel{n}(n + p\log_2 p) = [an + kp \cdot \log_2(kp)] \cdot \cancel{n}$

$\therefore$ $\cancel{an} + ap\log_2 p = \cancel{an} + kp\log_2(kp)$

$a\cancel{p}\log_2 p = k\cancel{p}\log_2(kp)$

$\therefore$ $a \log_2(p) = k\log_2(kp) = k[\log_2(k) + \log_2(p)]$

$a = k\left[\frac{\log_2(k)}{\log_2(p)} + 1\right]$

$\therefore$ We need to increase n by a factor of $k \cdot \left[\frac{\log_2(k)}{\log_2(p)} + 1\right]$.

Thus, Since we doubled the number of P from 8 to 16

$\therefore$ $k = 2$ and $p = 8$

$\therefore$ $a = 2\left[\frac{\log_2(2)}{\log_2(8)} + 1\right] = 2\left[\frac{1}{3} + 1\right] = \frac{8}{3}$

$\therefore$ we need to increase n by a factor of $\frac{8}{3}$.

Thus, the parallel program is scalable