HW5

1. TASK 1: PING PONG LATENCY

Summary:

I have used 1000 iterations of ping-pong loop to get accurate timings. Finally ran the code for 16 data points with data following message_length(L) in bytes: `0`, `100, 1000, 10000, 100000, 500000, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000, 10000000.` The code calculates total timing Tmsg for all the above mentioned data points.

To calculate T_s, I simply used the value of Tmsg for message_length(L) =0.
To calculate T_w, I am using mean of T_w value for all the 16 data points.

T_s = 0.007997999 sec
T_w = 0.00005823312 sec

Loop Iterations used:

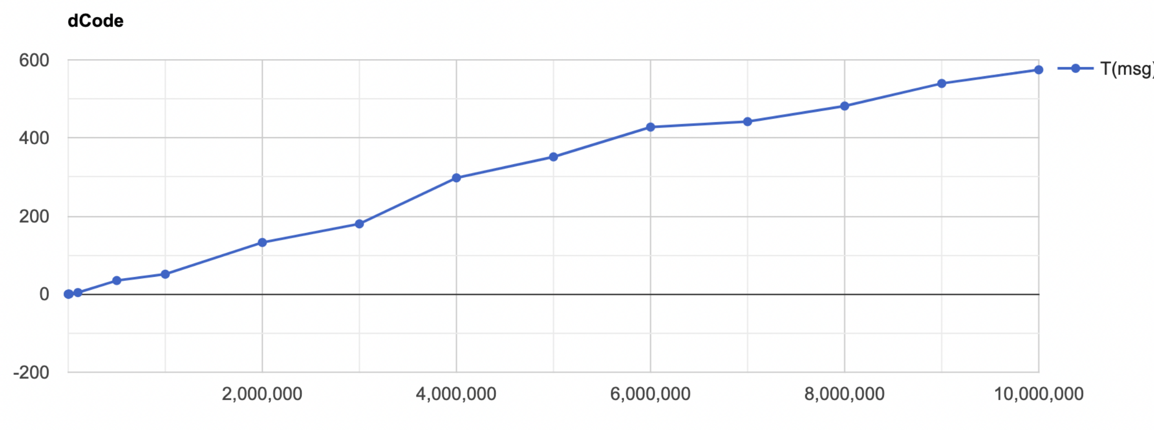
1000

Data Points:

Data Points	X Axis -> Message Length(L)	Y Axis -> T_{msg}
1	0	0.007997999
2	100	0.013313002
3	1000	0.048486501
4	10000	0.363008
5	100000	4.0599218
6	500000	34.8613816
7	1000000	51.0772118

8	2000000	132.3538463
9	3000000	179.9873296
10	4000000	297.8985994
11	5000000	351.6244534
12	6000000	427.94704218
13	7000000	441.8691686
14	8000000	481.82289031
15	9000000	539.689148131
16	10000000	574.63355542

Graph:



NOTE: X axis denotes Message length(L) in bytes, and Y axis denotes T_{msg} in seconds

Calculation for T_s:

- For L=0, $T_{msg} = T_s$
- From above data points, T_{msg} for L=0 is 0.007997999 sec
- Therefore, $T_s = 0.007997999$ sec

Calculation for T_w:

- For any data point (L, T_{msg})

$$T_{msg} = T_s + T_w * L$$

$$\Rightarrow T_w = (T_{msg} - T_s)/L$$

Data Points	**X Axis -> Message Length(L)**	**Y Axis -> T_{msg}**	**$(T_{msg} - T_s)/L$**
1	0	0.007997999	
2	100	0.013313002	0.0000531500200000000710
3	1000	0.048486501	0.0000404885009999998279
4	10000	0.363008	0.0000355009999999999597
5	100000	4.0599218	0.0000405192379999998407
6	500000	34.8613816	0.0000697067672000000078
7	1000000	51.0772118	0.0000510692137999999689
8	2000000	132.3538463	0.0000661729241500000206
9	3000000	179.9873296	0.0000599931105333333151
10	4000000	297.8985994	0.0000744726503499999889
11	5000000	351.6244534	0.00007032329
12	6000000	427.94704218	0.00007132317
13	7000000	441.8691686	0.00006312302437142862 4
14	8000000	481.82289031	0.00006022686
15	9000000	539.689148131	0.0000599645722367777787 9
16	10000000	574.63355542	0.0000574625557420000090
SUM -> $\sum_{n=1}^{15} (T_{msg} - T_s)/L$			0.00087349689

- Now, we will take mean of all the values for all data points m to get most accurate value for T_w, i,e,

$$T_w = \left(\sum_{n=1}^{m} (T_{msg} - T_s)/L\right) \div m$$

$$T_w = \left(\sum_{n=1}^{15} (T_{msg} - T_s)/L\right) \div m$$

$$T_w = \left(\sum_{n=1}^{15} (T_{msg} - T_s)/L\right) \div 15$$

$$T_w = \left(\sum_{n=1}^{m} (T_{msg} - T_s)/L\right) \div 15$$

$$T_w = 0.00087349689 \div 15$$

T_w = 0.00005823312 sec

2. TASK 2: FINITE-DIFFERENCE FOR 2D DATA DECOMPOSITION

Assume

- Dimentions = $Nx \times Ny \times Nz$**,** where Nx **=** Ny **=** N for simplicity

- 9-point stencil

- **Number of processes** = *P*

- 2-D domain decomposition

Given the dimensions:

$$Total\ grid\ points = N \times N \times N_Z = N^2 N_Z \qquad - (1)$$

Calculations:

A. Computation time

Assuming No replicated computation:

$$Tcomp = t_c \times N^2 \times N_Z \qquad - (2)$$

where tc = average computation time per grid point (slightly different at edges from interior etc)

B. Communication time

Given that its a 2-D domain decomposition, we will partition the grid points among P processors in 2 directions, namely, X and Y.

Assuming that the partitioning is equal in both the directions, **the dimension of task per processor** would then be:

X direction: $\frac{N}{\sqrt{P}}$, Y direction: $\frac{N}{\sqrt{P}}$, Z direction: N_Z

$$Total\ grid\ points\ per\ processor = \frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} \times N_Z \quad - (3)$$

Given that its a 9 point stencil and 2d decomposition, each task exchanges 2 planes with each of 4 neighbours.

Size of walls with these 4 neigbours is:

$$\frac{N}{\sqrt{P}} \times N_Z$$

Therefore:

$$T_{comm} = 4P * (T_S + 2 * T_W * \frac{N}{\sqrt{P}} * N_Z) \quad - (4)$$

C. 2D Finite difference time

Assuming that P divides N exactly, then assume load-balanced and no idle time:

$$T_{2D\_finite\_diff} = (T_{comp} + T_{comm})/P$$

$$\Rightarrow T_{2D\_finite\_diff} = \frac{t_c N^2 N_Z}{P} + 4P * (\frac{T_S + 2 * T_W * \frac{N}{\sqrt{P}} * N_Z}{P})$$

$$\Rightarrow T_{2D\_finite\_diff} = \frac{t_c N^2 N_Z}{P} + 4 * (T_S + 2 * T_W * \frac{N}{\sqrt{P}} * N_Z)$$

$$\Rightarrow T_{2D\_finite\_diff} = \frac{t_c N^2 N_Z}{P} + 4T_S + 8T_W * \frac{N}{\sqrt{P}} * N_Z \quad - (5)$$

D. 2D Finite difference Efficiency

We know, relative efficiency is given by:

$$E_{relative} = \frac{T_1}{PT_P} \quad - (6)$$

Where, T_1 is time to run algorithm by a single processor, and T_p is time to run algorithm on P processors

For our algorithm,

$T_1 = t_c N^2 N_Z$ And $T_p = T_{2D\_finite\_diff} = \frac{t_c N^2 Nz}{P} + 4T_S + 8T_W * \frac{N}{\sqrt{P}} * N_Z)$

$$\Rightarrow E \;=\frac{T_1}{PT_P}\;=\frac{t_c N^2 N_Z}{P*(\frac{t_c \times N^2 \times N_Z}{P} + 4T_S + 8T_W * \frac{N}{\sqrt{P}} * N_Z)}$$

$$\Rightarrow E \;=\frac{t_c N^2 N_Z}{t_c N^2 N_Z + 4PT_S + 8\sqrt{P}\,T_W N\,N_Z)} \qquad - (7)$$

E. 2D Finite difference Iso-Efficiency

For 2D decomposition, efficiency we calculated in equation 7 is:

$$\Rightarrow E \;=\frac{t_c N^2 N_Z}{t_c N^2 N_Z + 4PT_S + 8\sqrt{P}\,T_W N\,N_Z)}$$

$$\Rightarrow t_c N^2 N_Z \;= E * (t_c N^2 N_Z \;+\; 4PT_S \;+\; 8\sqrt{P}\,T_W N\,N_Z)$$

Here, dominant terms are:

$$\sim N^2 N_Z \;\approx\; \sim N^2 N_Z \qquad \sim \sqrt{P}\,N\,N_Z$$

For E to be constant,

$N^2 N_Z = \sqrt{P}\,N\,N_Z$

$$\Rightarrow N \;= \sqrt{P} \qquad - (8)$$

By substituting $N \;= \sqrt{P}$**,** E is constant given by equation

$$t_c N_Z \;= E * (t_c N_Z \;+\; 4T_S \;+\; 8T_W N_Z) \qquad - (9)$$

Total computation $Tcomp$, given by equation 2 is $Tcomp \;=\; t_c N^2 N_Z$

Therefore, to keep E constant: $Tcomp \propto N^2$

$$\Rightarrow Tcomp \propto P \;(Using\;Eq\;8:\; N \;= \sqrt{P}) \qquad - (10)$$

Number of grid points, given by equation 1 is $Grid\;points \;=\; N^2 N_Z$

Therefore,to keep E constant: $Grid\;points \propto N^2$

$$\Rightarrow Grid\ points \propto P\ (Using\ Eq\ 8{:}\ N\ = \sqrt{P}) \qquad - (11)$$

From equations 10 and 11, we can conclude:
the isoefficiency of this algorithm is O(P)

$$\Rightarrow Isoefficiency\ = O(P) \qquad - (12)$$

F. Analysis

Comparative Analysis of 2D_Decomposition and 1D_Decomposition

- 2D_Decomposition with $Isoefficiency$ of O(P) is way more scalable than 1D_Decomposition which has $Isoefficiency$ of $O(P^2)$.
- This is because the amount of computation needed to keep E constant in 2D_Decomposition must increase linearly with the number of processors to keep the efficiency constant. This is way better than 1D_Decomposition, where the amount of computation needed to keep E constant must increase as the square of the number of processors.

Reasoning:

- The number of neighbors in 2D_Decomposition taking part in the communication is 4 per processor, 2 more than the number of neighbors in 1D_Decomposition per processor.
- Although the amount of communication increases from 2P to 4P, the number of messages/grid_points shared per communication reduces.
- This is because the wall area is lesser for 2D_Decomposition($\frac{N}{\sqrt{P}} * N_Z$) in comparison to 1D_Decomposition's wall area($N * N_Z$).

Conclusion:

- Decomposition that increases the number of walls, will end in better efficiency. This is because, although the amount of communication may increase, the number of messages/grid_points shared per communication reduces.