AM250 HW6: Latency

Owen Morehead

May 25 2022

1 Running the Programs

A `README.md` file is included which contains the commands necessary to compile and run the programs below. Brief instructions are also included here. To compile the program, the command used is:

```
mpif90 -o  program program.f90
```

where the name of the program, `program.f90`, is the name provided in each section title below, and the name of the executable, `program`, was chosen to be the exact same name as the program without the `.f90` extension. After the executables are made (they are also attached in the homework submission), they can be run interactively on `Grape` using the command:

```
mpirun -np 2 program
```

The number of threads used to run the program below was 2, since there are only two processors interacting during the ping-pong message sending. One can declare more processors to be used, but the code will only choose two of them to use for ping-pong.

2 Determining Latency Time – latency_pong_mpi.f90

This program wraps the mpi timer, `MPI_WTIME()` around the previously made ping-pong code (HW4), which sends a message back and forth between two processors N times. After one processor completes a send and receive, the total time is recorded and appended to an array. Once all communication times are recorded for every iteration of ping-pong, the average is taken to get more accurate timings. This ping-pong loop and average message time recording is recorded for a range of message size values, ranging from $L = 50$, to $L = 100,000$. The message length and average communication time is written out to a `.dat` file, which is then opened in `Matlab` to plot the data. A linear regression is performed in order to determine the line which best fits the data. The equation of the line can be written as,

$$T_{\text{msg}} = t_s + t_w L \tag{1}$$

where we have data for both T_{msg} and L, the message communication time, and size of the message. We see in Figure 1 best fit lines plotted against two sets of data. For both, we see a strong linear relationship in the data, and the best fit lines fit the data well. I will denote the data with the smaller slope as D_2, and the data with the larger slope as D_1. The two sets of data are further described:

- D_1: 10 average message time data points ranging from $L = 50$ to $L = 100,000$.
- D_2: 2,000 average message time data points ranging from $L = 50$ to $L = 100,000$.

For both sets of data, the message time for each message length was computed as the average of 5,000 recorded message times from the ping-pong loop. The message used was a double precision real value.

We see that the two sets of data differ, although both follow the linear model of data transfer fairly well. For both sets of data, there seem to be initial transients for smallish-sized messages. We can see that the slope of the data is slightly larger in this region, but then stabilizes to a constant value for the rest of the larger-sized messages. We can deduce the values for t_s and t_w, the message startup time (latency), and cost per word time (transfer time per 'word' of data, determined by bandwidth of channel) for both sets of data.

From linear regression analysis, we deduce the following values for these quantities for the D_1 and D_2 data sets:

$$D_1 : t_s = 5.23 \times 10^{-6} \qquad t_w = 2.24 \times 10^{-9} \tag{2}$$
$$D_2 := t_s = 7.94 \times 10^{-6} \qquad t_w = 1.70 \times 10^{-9}. \tag{3}$$

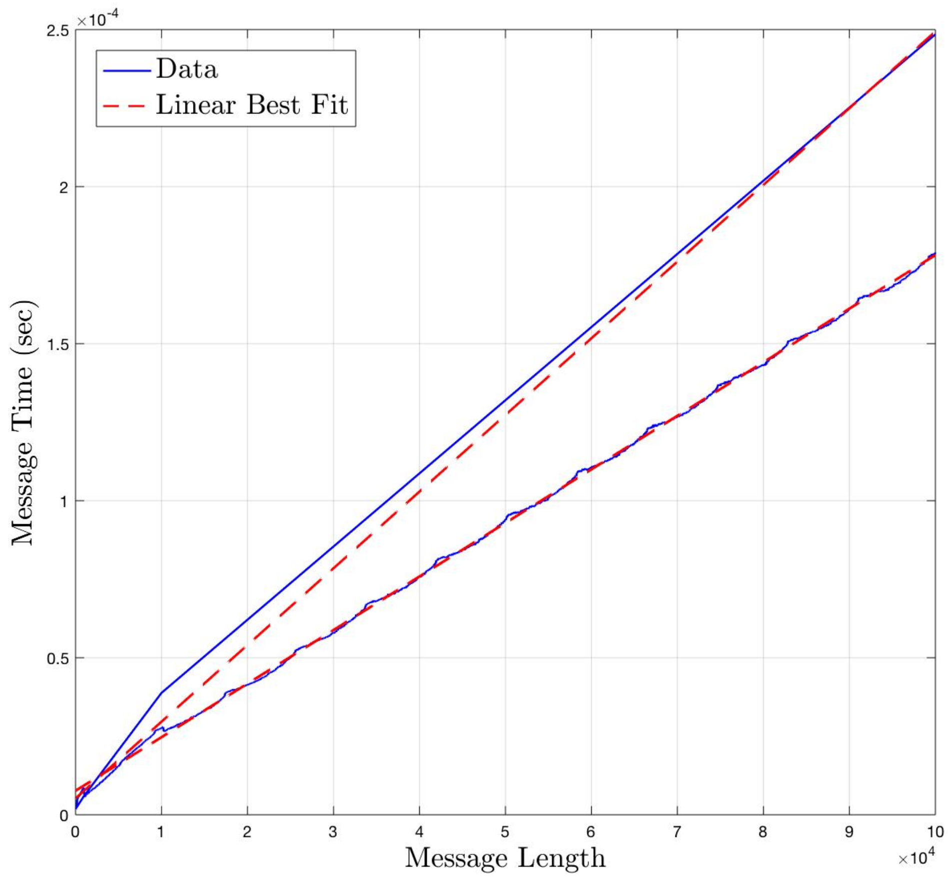

Figure 1: Message communication time vs. size of message plotted against a best fit line, defined by the equation $T_{\text{msg}} = t_s + t_w L$. From this we deduce the values for the coefficients t_s and t_w. The lower line of data contains 2,000 data points ranging from $L = 50$ to $L = 100,000$. The upper line of data corresponds to only 10 data points ranging from $L = 50$ to $L = 100,000$.

3 2D Data Decomposition - Finite Differences Performance Model

Here we derive an analytical model for the execution time, the efficiency, and the isoefficienty (i.e. performance model analysis) of a 9 point finite differences stencil, computed with a 2D data decomposition in the horizontal direction.

If there is no replicated communication, then the computation time for the entire 3D ($N \times N \times N$) grid is,

$$T_{comp} = t_c(N_x \times 1)N_yN_z = t_cN_xN_yN_z \tag{4}$$

where t_c is the average computation time per grid point.

Using a 9 point stencil, each task exchanges 2 rows to the right, left, above, and below. In other words, there are $2N$ points exchanged with each of 4 neighbors:

$$T_{comm} = 4P(t_s + 2t_wN_z). \tag{5}$$

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

$$T_{2D\_finite\_diff} = (T_{comp} + T_{comm})/P = \frac{t_cN_xN_yN_z}{P} + 4t_s + 8t_wN_z \tag{6}$$

and thus the efficiency is,

$$E = \frac{T_1}{PT_p} = \frac{t_cN_xN_yN_z}{t_cN_xN_yN_z + 4Pt_s + 8Pt_wN_z}. \tag{7}$$

If we consider how must the amount of computation performed scale with P to keep E constant, this result is a function of N, and is denoted as the isoefficienty function.

Noting the efficiency as Equation 6, we can write it as,

$$t_cN_xN_yN_z = E\bigg(t_cN_xN_yN_z + 4Pt_s + 8Pt_wN_z\bigg). \tag{8}$$

For large N and P, the two terms that scale as N^3 are the dominating terms.

To balance the efficiency equation and make E constant, we require that $N_xN_y \sim P$, which gives us,

$$t_cN_xN_yN_z = E\bigg(t_cN_xN_yN_z + 4N_xN_yt_s + 8N_xN_yt_wN_z\bigg), \tag{9}$$

$$t_cN_z = E\bigg(t_cN_z + 4t_s + 8t_wN_z\bigg). \tag{10}$$

Thus we see that in scaling $N^2 \sim P$, the number of grid points to keep E constant $\sim P$. If we look at Equation 6, for large P, we see that with $N^2 \sim P$, both the numerator and denominator of E scale as P. In other words, the amount of computation to keep E constant goes as $\sim P$, since $T_{comp} = t_cN_xN_yN_z \sim t_cPN_z$ (as N_z is chosen constant). Thus, the isoefficiency function $\sim \mathcal{O}(P)$, which means this algorithm is highly scalable, since the amount of computation needs to increase only linearly with P. In comparing to the 1D data decomposition for the same finite difference 9-pt stencil, we found that the isoefficiency function $\sim \mathcal{O}(P^2)$, which means this algorithm is not as scalable. Thus, we see that the 2D data decomposition is much more easily scalable compared to the 1D decomposition. In the case of a 1D data decomposition, the amount of computation must increase as the square of the number of processors to keep the efficiency constant. In the case of a 2D data decomposition, the amount of computation must increase linearly with the number of processors to keep the efficiency constant!