Assignment 3: MPI Point-to-Point and One-Sided Communication

Mihai-Gabriel Robescu (03709201)
mihai.robescu@tum.de
Yi JU (03691953)
yi.ju@tum.de
Hsieh Yi-Han
kimihsieh.tw@tum.de
Koushik Karmakar (03704659)
koushik.karmakar@tum.de

Q3. Setting a baseline

Q3.1.1

Check the attached job.ll file.

Q3.1.2

Because the time of different kinds should be different among different processes in a run, the sum of each kind of time can show the information more clearly, we choose the sum of each kind of time, which different processes take in the same run, for the plot of Times with fixed process counts and varying size of input for Sandy Bridge node and Haswell node. But when it comes to the plot of Times with fixed input sets and varying process counts, the amount of processes would strongly influence the sum of time. Therefore, we choose to use the average instead of sum of time mentioned above. The average time also shows how fast each run is. The script named plot.py is used for plotting and some analysis.

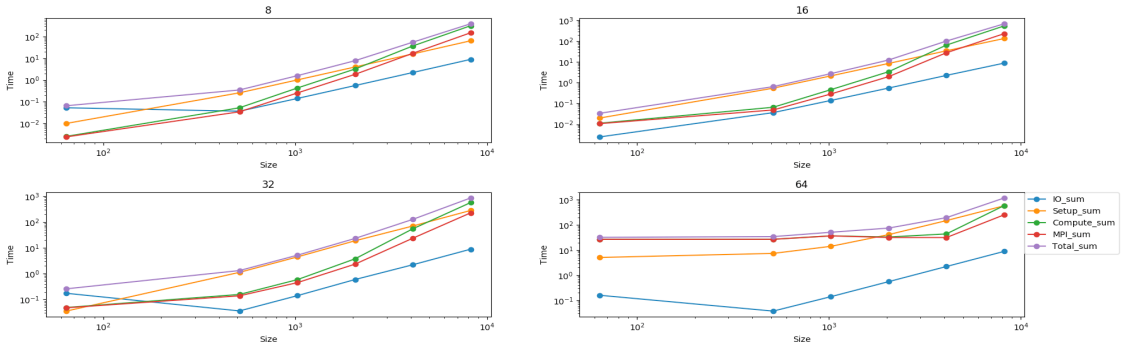

Fig 3.1 Times with fixed process counts and varying size of input for Sandy Bridge node

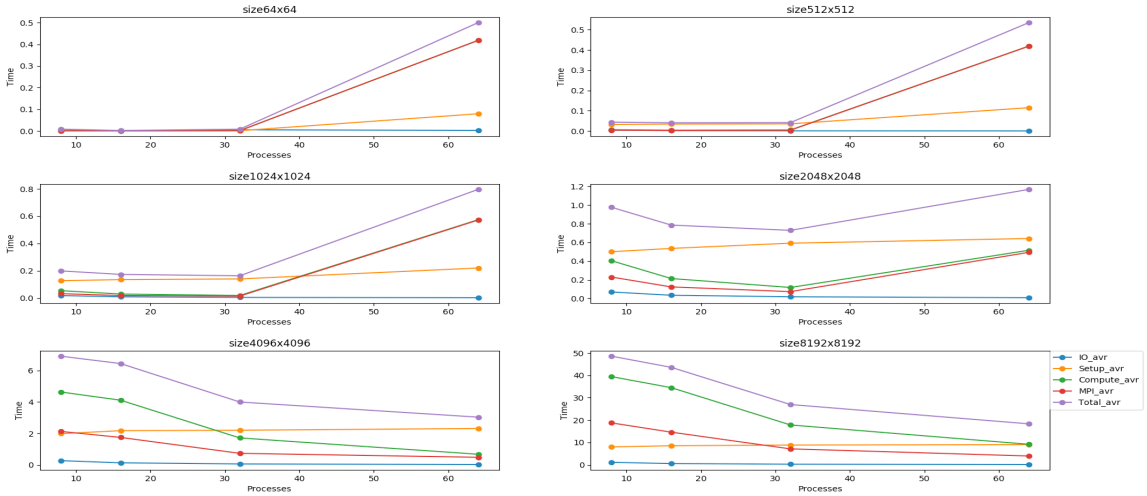

Fig. 3.2 Times with fixed input sets and varying process counts for Sandy Bridge node

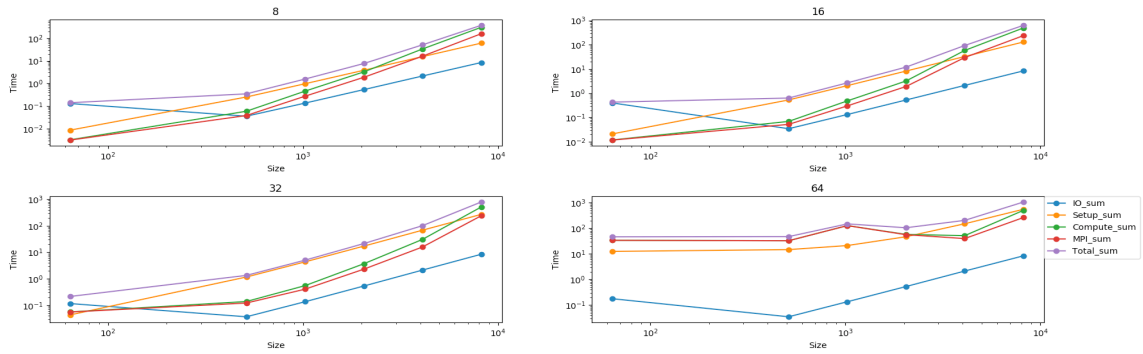

Fig. 3.3 The times with fixed process counts and varying size of input for Haswell node

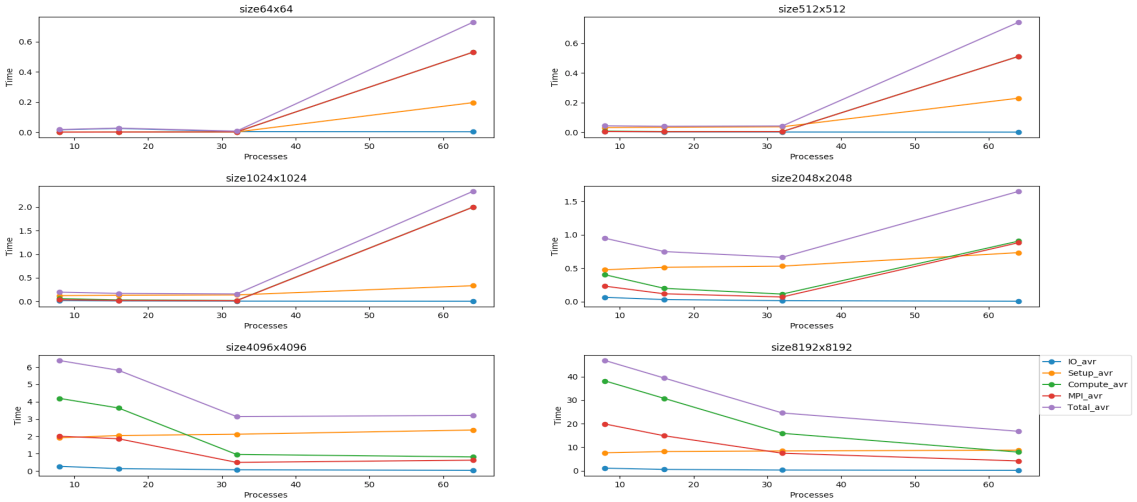

Fig 3.4 Times with fixed input sets and varying process counts for Haswell node

Q3.2.1

The Gaussian Elimination is a popular algorithm to find the solution of the linear system in form of $A\vec{x} = \vec{b}$. It involves two steps. First step is usually called forward elimination and the second step is named as back substitution. In the Forward elimination A and $\vec{b}$

are transformed synchronously. So that A becomes the upper triangular matrix U:

$$A=\begin{pmatrix} a_{11} & \cdots & \cdots & \cdots & a_{1n} \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{pmatrix} \rightarrow U := \begin{pmatrix} r_{11} & \cdots & \cdots & \cdots & r_{1n} \\ 0 & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & r_{nn} \end{pmatrix}, \vec{b} \rightarrow \vec{c}$$

The original system of linear equations is reduced to row echelon form: $U\vec{x} = \vec{c} \in \mathbf{R}^n$. In Back Substitution:

$$U\vec{x} = \vec{c} \rightarrow \begin{pmatrix} r_{11} & \cdots & \cdots & \cdots & r_{1n} \\ 0 & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & r_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ \vdots \\ c_n \end{pmatrix}$$

$$\rightarrow x_n = \frac{c_n}{r_{nn}}$$

$$\rightarrow x_{n-1} = \frac{c_n - r_{n-1n} x_n}{r_{n-1n-1}}$$

$$\rightarrow \cdots$$

In the provided implementation, we use the different number of process (8, 16, 32 and 64) to deal with different size (64x64, 512x512, 1024x1024, 2048x2048, 4096x4096 and 9182x9182) linear system. So that the As are separated into several parts to do the transformation in parallel. In order to get necessary data, some data communication is involved after some calculation. Detailed information could be seen below.

Q3.2.2

At beginning the data are separated uniformly into different groups. The number of the group is equal to the number of process. The detailed way to deal with data is shown as following.

$$A=\begin{pmatrix} a_{11} & \cdots & \cdots & \cdots & a_{1n} \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ \vdots & & & & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{pmatrix}\rightarrow$$

$$A_0=\begin{pmatrix} a_{11} & \cdots & \cdots & \cdots & a_{1n} \\ \vdots & & & & \vdots \\ a_{\frac{n}{p}1} & \cdots & \cdots & \cdots & a_{\frac{n}{p}n} \end{pmatrix}\cdots A_p=\begin{pmatrix} a_{\left(n+1-\frac{n}{p}\right)1} & \cdots & \cdots & \cdots & a_{\left(n+1-\frac{n}{p}\right)n} \\ \vdots & & & & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{pmatrix}\rightarrow$$

$$U_0=\begin{pmatrix} r_{11} & \cdots & \cdots & \cdots & \cdots & r_{1n} \\ 0 & \ddots & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & r_{\frac{n}{p}\frac{n}{p}} & \cdots & r_{\frac{n}{p}n} \end{pmatrix}\cdots A_p=\begin{pmatrix} a_{\left(n+1-\frac{n}{p}\right)1} & \cdots & \cdots & \cdots & a_{\left(n+1-\frac{n}{p}\right)n} \\ \vdots & & & & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nn} \end{pmatrix}\rightarrow$$

$$U_0=\begin{pmatrix} r_{11} & \cdots & \cdots & \cdots & \cdots & r_{1n} \\ 0 & \ddots & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & r_{\frac{n}{p}\frac{n}{p}} & \cdots & r_{\frac{n}{p}n} \end{pmatrix}U_1=\begin{pmatrix} 0 & \cdots & 0 & r_{\left(\frac{n}{p}+1\right)\left(\frac{n}{p}+1\right)} & \cdots & \cdots & \cdots & \cdots & r_{\left(\frac{n}{p}+1\right)n} \\ \vdots & & & 0 & \ddots & & & & \vdots \\ \vdots & & & & & \ddots & & & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & r_{\frac{2n}{p}\frac{2n}{p}} & \cdots & r_{\frac{2n}{p}n} \end{pmatrix}\cdots$$

$$A_p^{'}=\begin{pmatrix} 0 & \cdots & 0 & a^{'}_{\left(n+1-\frac{n}{p}\right)\left(\frac{n}{p}+1\right)} & \cdots & a^{'}_{\left(n+1-\frac{n}{p}\right)n} \\ \vdots & & & \vdots & & \vdots \\ \vdots & & & \vdots & & \vdots \\ 0 & \cdots & 0 & a^{'}_{n\left(\frac{n}{p}+1\right)} & \cdots & a^{'}_{nn} \end{pmatrix}\rightarrow\cdots\rightarrow$$

$$U_0=\begin{pmatrix} r_{11} & \cdots & \cdots & \cdots & \cdots & r_{1n} \\ 0 & \ddots & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & r_{\frac{n}{p}\frac{n}{p}} & \cdots & r_{\frac{n}{p}n} \end{pmatrix}\cdots U_0=\begin{pmatrix} 0 & \cdots & 0 & r_{\left(n+1-\frac{n}{p}\right)\left(n+1-\frac{n}{p}\right)} & \cdots & \cdots & r_{\left(n+1-\frac{n}{p}\right)n} \\ \vdots & & & & \ddots & & \vdots \\ \vdots & & & & & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & r_{nn} \end{pmatrix}$$

Q3.2.3

Because in the bash, it is inconvenient to do the calculation of float-type number. We choose the python to do the calculation. In the modified Load-Leveler batch, we have

used a python file named as time_calculater.py to do the calculation of the average and sum of IO time, Setup time, Compute time, MPI time and the Total time of different run with different process or different matrix size. Firstly, we write the time recorded by the gauss.c into the interfile.txt. Secondly, we call the python script named as time_calculater.py. In the time_calculater.py, we firstly read the different kinds of time of each process and store them in the matrix. After that we calculate the average and sum of them and then write them in the time.txt.
Because we noticed that there is relatively big difference in some kinds of time between each run, we repeat the test part in the original Load-Leveler batch for several times and calculate the average of these runs after checking the relative difference between runs.

Q3.2.4

The first challenge is the relatively big differences in some kinds of run time between each run, to conquer this we repeat the test part in the original Load-Leveler batch for several times and calculate the average of these runs after checking the relative difference between runs.
The second challenge is that because of the limited rum time of each job submitted to the supercomputer. At very beginning, we run more times to have an accurate baseline time. But before we got the time_avr.txt we needed, the job stopped. We have to decrease the times.

Q3.2.5

As the fig. 3.1 and 3.3 show, the compute and MPI times scalability with fixed process counts and varying size of input files for the Sandy Bridge and Haswell nodes are not strong. We choose the set *N*: [64, 512, 1024, 2048, 4096, 8192] as the x-axis and the time as the y-axis in the log-log plot. We do the linear regression to find out the relation between *N* and time. The relation between *log(N)* and *log(time)* is showed in the way: $\log(time) = a\log(N) + b$, in which $a < 3$. That means: $time \propto N^a (a < 3)$. The problem size should be: $problem\_size \propto N^3$. So, times scalability are not strong. In the plot of 64 processes we also observe that the MPI time and compute time decrease when the problem size increases.

Q3.2.6

As the fig. 3.2 and 3.4 show, the compute and MPI times scalability with fixed input sets and varying process counts for the Sandy Bridge and Haswell nodes are strong, because the time decreases when the number of processes increases.

Q4. MPI Point-to-Point Communication

Q4.1.1 Ans. See attached gauss_non_blocking.c file.

Q4.1.2 The new performance plots and the description in the report. (5 points)

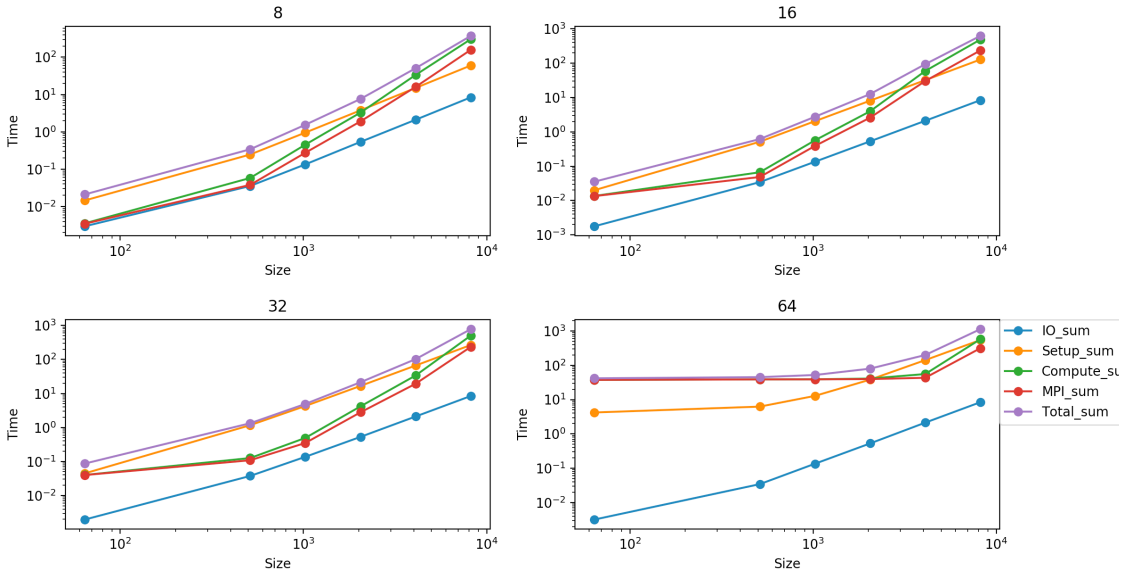

Fig. 4.1 The times with fixed process counts and varying size of input for Haswell node

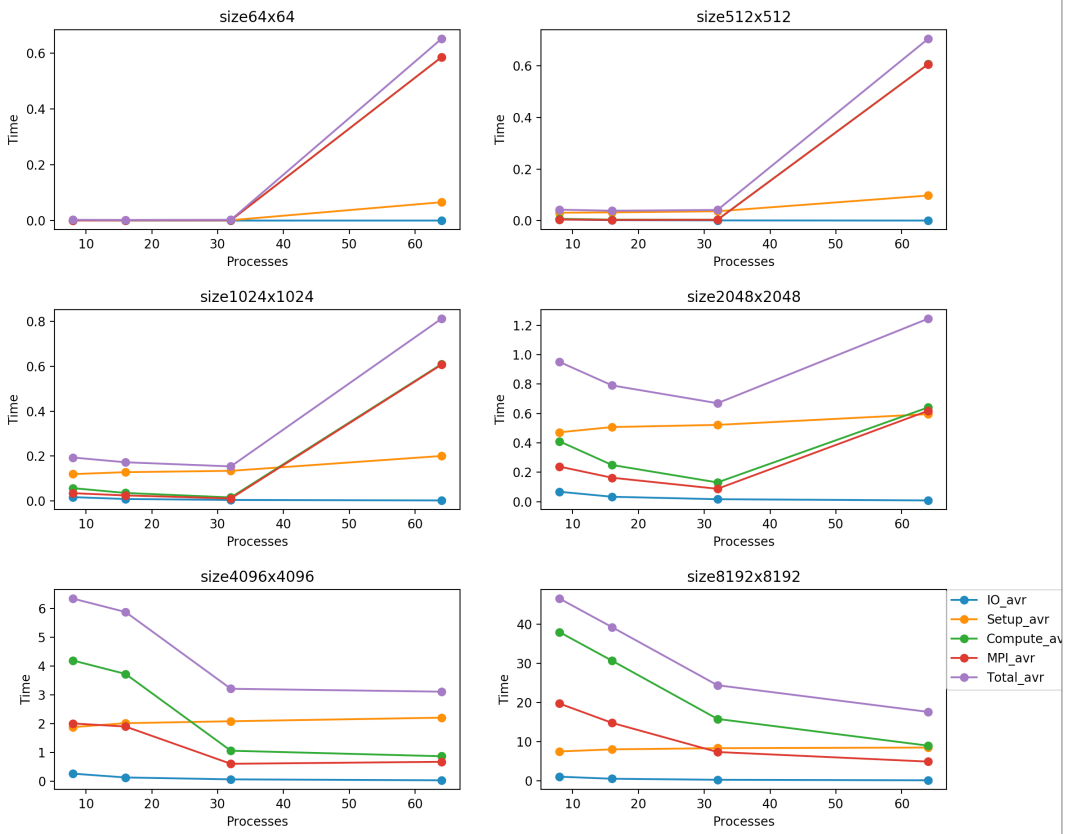

Fig 4.2 Times with fixed input sets and varying process counts for Haswell node

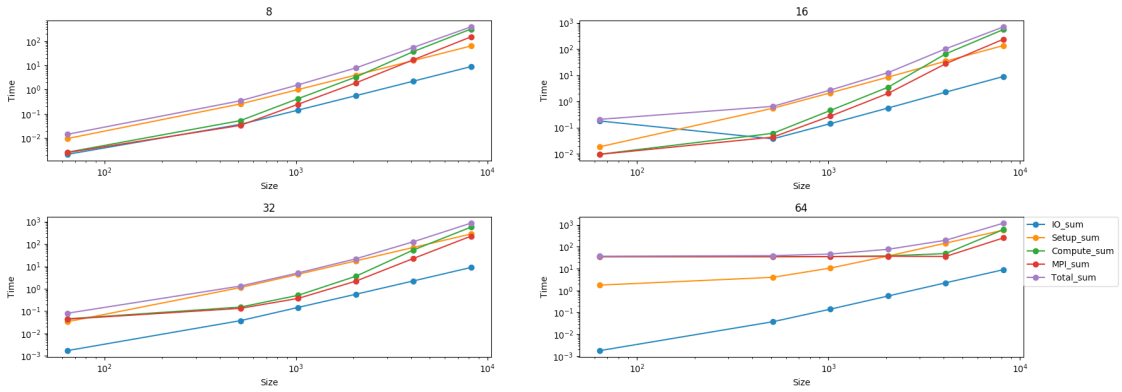

Fig 4.3 Times with fixed process counts and varying size of input for Sandy Bridge node

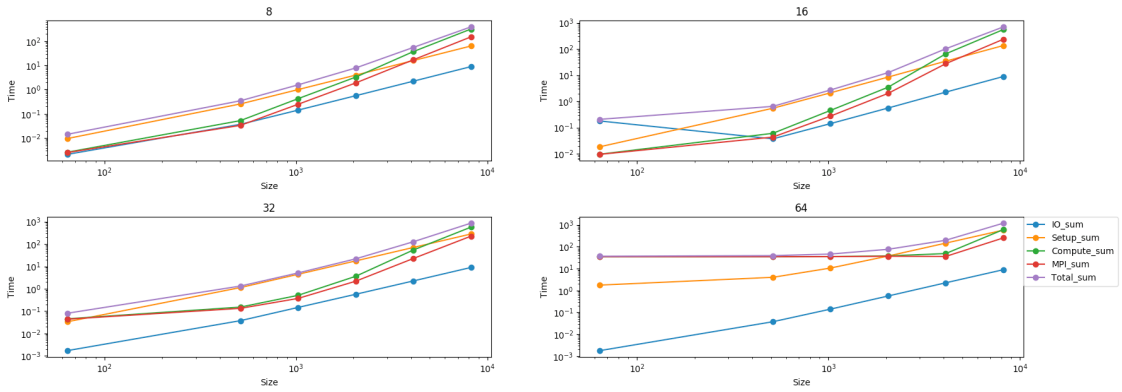

Fig. 4.4 Times with fixed input sets and varying process counts for Sandy Bridge node

4.2.1.

Ans. For the MPI point to point communication, we have used MPI_Isend, MPI_Irecv and MPI_Wait.
In our non-blocking code, we have replaced every blocking MPI_Send and MPI_Recv with non-blocking MPI_Isend and MPI_Irecv so that the MPI communication is non-blocking and we also have used MPI_Wait in appropriate places so that waits for an MPI request to complete.
For example, in the computation stage, we have used MPI_Wait to make sure that the all the process has the pivot elements before they start the computation. (gauss_non_blocking.c : line no. 148)

Again, in line no. 199 in the file gauss_non_blocking.c, we have used MPI_Wait to make sure the the accumulation_buffer is received by MPI_Irecv before the computation for the next stage begins.

4.2.2

No, from Fig 4.5 and Fig 4.6, I found that downstairs processes still wait, that's *MPI_Wait,* until the upstairs process *Isend* the pivots. When the downstairs processes get pivots, then they compute the new pivots. In my view, it's the drawback of gaussian elimination algorithm itself. The sequent processes need to get data firstly to compute otherwise do nothing, just like blocking. Or the way which I use to implement the code is more like the blocking communication.

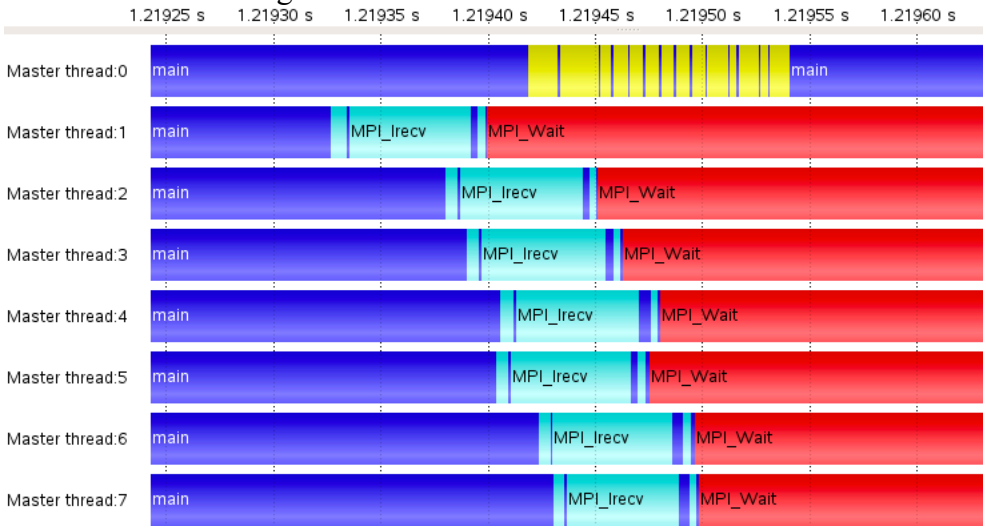

Fig. 4.5 Processes except process 0 execute *MPI_Irecv* and then execute *MPI_Wait* while Process 0 still compute the first pivots.

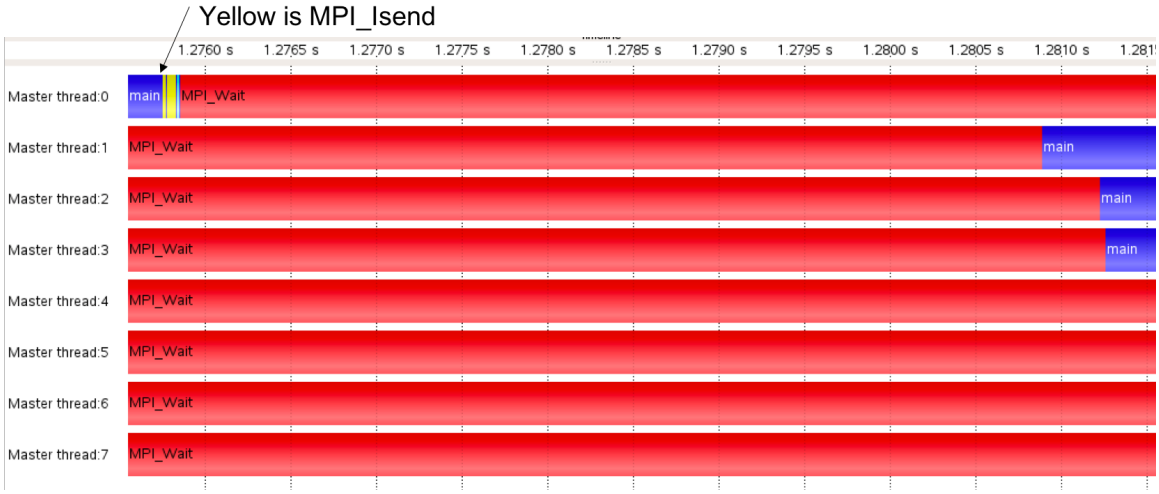

Fig. 4.6 Other processes start computing when process 0 sends the pivots.

4.2.3.

Table 4.1 the speedup observed versus the baseline for Haswell nodes.

Size	Num_process	IO_avr	Setup_avr	Compute_avr	MPI_avr	Total_avr	IO_sum	Setup_sum	Compute_sum	MPI_sum	Total_sum
size64x64	8	43.315	0.585	0.897	0.892	6.631	43.315	0.585	0.897	0.892	6.631
size64x64	16	224.933	1.067	0.882	0.880	12.233	224.933	1.067	0.882	0.880	12.233
size64x64	32	59.180	0.973	1.393	1.394	2.480	59.180	0.973	1.393	1.394	2.480
size64x64	64	54.813	2.973	0.905	0.905	1.118	54.813	2.973	0.905	0.905	1.118
size512x512	8	0.991	1.010	1.003	0.995	1.007	0.991	1.010	1.003	0.995	1.007
size512x512	16	1.009	1.031	1.044	1.082	1.032	1.009	1.031	1.044	1.082	1.032
size512x512	32	0.978	0.995	1.097	1.115	1.004	0.978	0.995	1.097	1.115	1.004
size512x512	64	1.002	2.354	0.844	0.844	1.054	1.002	2.354	0.844	0.844	1.054
size1024x1024	8	0.997	1.013	0.999	0.983	1.008	0.997	1.013	0.999	0.983	1.008
size1024x1024	16	0.990	1.008	0.857	0.777	0.976	0.990	1.008	0.857	0.777	0.976
size1024x1024	32	0.990	1.010	1.116	1.152	1.020	0.990	1.010	1.116	1.152	1.020
size1024x1024	64	0.988	1.654	3.278	3.286	2.871	0.988	1.654	3.278	3.286	2.871
size2048x2048	8	0.982	1.011	0.983	0.975	0.996	0.982	1.011	0.983	0.975	0.996
size2048x2048	16	0.997	1.014	0.805	0.733	0.947	0.997	1.014	0.805	0.733	0.947
size2048x2048	32	1.002	1.018	0.877	0.820	0.990	1.002	1.018	0.877	0.820	0.990
size2048x2048	64	0.998	1.232	1.411	1.426	1.323	0.998	1.232	1.411	1.426	1.323
size4096x4096	8	1.005	1.021	1.001	1.000	1.007	1.005	1.021	1.001	1.000	1.007
size4096x4096	16	1.002	1.014	0.977	0.980	0.991	1.002	1.014	0.977	0.980	0.991
size4096x4096	32	1.007	1.019	0.899	0.813	0.979	1.007	1.019	0.899	0.813	0.979
size4096x4096	64	0.999	1.072	0.934	0.921	1.032	0.999	1.072	0.934	0.921	1.032
size8192x8192	8	0.999	1.012	1.004	1.005	1.005	0.999	1.012	1.004	1.005	1.005
size8192x8192	16	0.998	1.010	1.002	1.003	1.003	0.998	1.010	1.002	1.003	1.003
size8192x8192	32	0.993	1.005	1.004	1.007	1.004	0.993	1.005	1.004	1.007	1.004
size8192x8192	64	0.998	1.025	0.878	0.841	0.950	0.998	1.025	0.878	0.841	0.950

Table 4.2 the speedup observed versus the baseline for Sandy bridge nodes.

Size	Num_process	IO_avr	Setup_avr	Compute_avr	MPI_avr	Total_avr	IO_sum	Setup_sum	Compute_sum	MPI_sum	Total_sum
size64x64	8	24.910	1.055	0.983	0.961	4.592	24.910	1.055	0.983	0.961	4.592
size64x64	16	0.014	1.083	1.169	1.133	0.164	0.014	1.083	1.169	1.133	0.164
size64x64	32	102.935	1.073	1.095	1.101	3.252	102.935	1.073	1.095	1.101	3.252
size64x64	64	88.371	2.884	0.769	0.769	0.875	88.371	2.884	0.769	0.769	0.875
size512x512	8	1.005	1.014	1.013	1.040	1.013	1.005	1.014	1.013	1.040	1.013
size512x512	16	0.983	1.005	1.095	1.129	1.012	0.983	1.005	1.095	1.129	1.012
size512x512	32	0.977	1.009	1.049	1.056	1.013	0.977	1.009	1.049	1.056	1.013
size512x512	64	1.001	1.845	0.769	0.768	0.879	1.001	1.845	0.769	0.768	0.879
size1024x1024	8	1.001	1.014	1.020	1.030	1.014	1.001	1.014	1.020	1.030	1.014
size1024x1024	16	0.997	1.016	1.035	1.056	1.018	0.997	1.016	1.035	1.056	1.018
size1024x1024	32	0.974	1.016	1.169	1.231	1.030	0.974	1.016	1.169	1.231	1.030
size1024x1024	64	0.998	1.330	1.040	1.040	1.107	0.998	1.330	1.040	1.040	1.107
size2048x2048	8	0.999	1.010	0.985	0.976	0.999	0.999	1.010	0.985	0.976	0.999
size2048x2048	16	1.001	1.013	0.984	0.982	1.004	1.001	1.013	0.984	0.982	1.004
size2048x2048	32	1.060	1.083	1.055	1.090	1.077	1.060	1.083	1.055	1.090	1.077
size2048x2048	64	0.994	1.099	0.874	0.867	0.986	0.994	1.099	0.874	0.867	0.986
size4096x4096	8	0.999	1.009	1.003	1.007	1.004	0.999	1.009	1.003	1.007	1.004
size4096x4096	16	1.009	1.018	1.008	1.022	1.012	1.009	1.018	1.008	1.022	1.012
size4096x4096	32	1.002	1.011	1.024	1.061	1.016	1.002	1.011	1.024	1.061	1.016
size4096x4096	64	0.988	1.021	0.905	0.878	0.992	0.988	1.021	0.905	0.878	0.992
size8192x8192	8	1.006	1.016	0.999	1.001	1.002	1.006	1.016	0.999	1.001	1.002
size8192x8192	16	0.999	1.008	1.000	0.992	1.002	0.999	1.008	1.000	0.992	1.002
size8192x8192	32	1.001	1.010	1.005	1.018	1.007	1.001	1.010	1.005	1.018	1.007
size8192x8192	64	0.998	1.013	0.974	1.006	0.993	0.998	1.013	0.974	1.006	0.993

From Table 4.1 and 4.2, we can see that point-to-point communication has significant speed up in most circumstances. Although, under certain conditions like problem size greater than 1024x1024, it has either no significant speed up or worse performance.
Moreover, under same size condition, with 64 process for the SandyBridge nodes, the performances are worse for some cases.

Q5. MPI One-Sided Communication

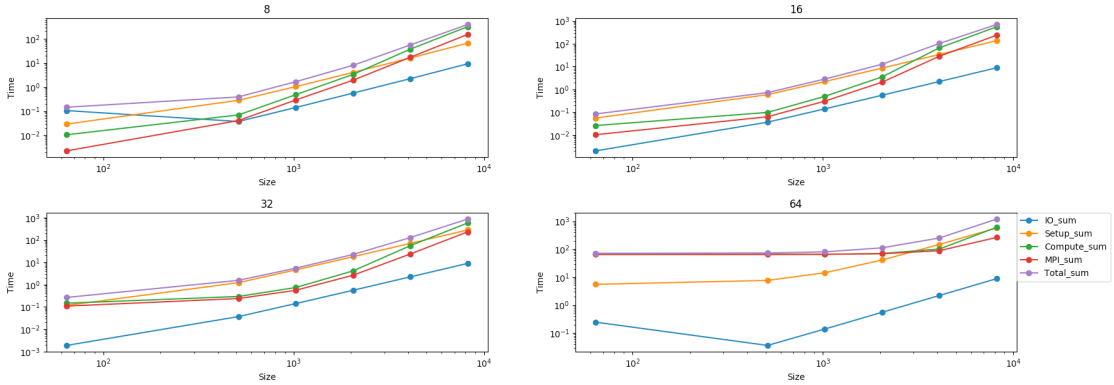

Fig 5.1 Times with fixed process counts and varying size of input for Sandy Bridge node

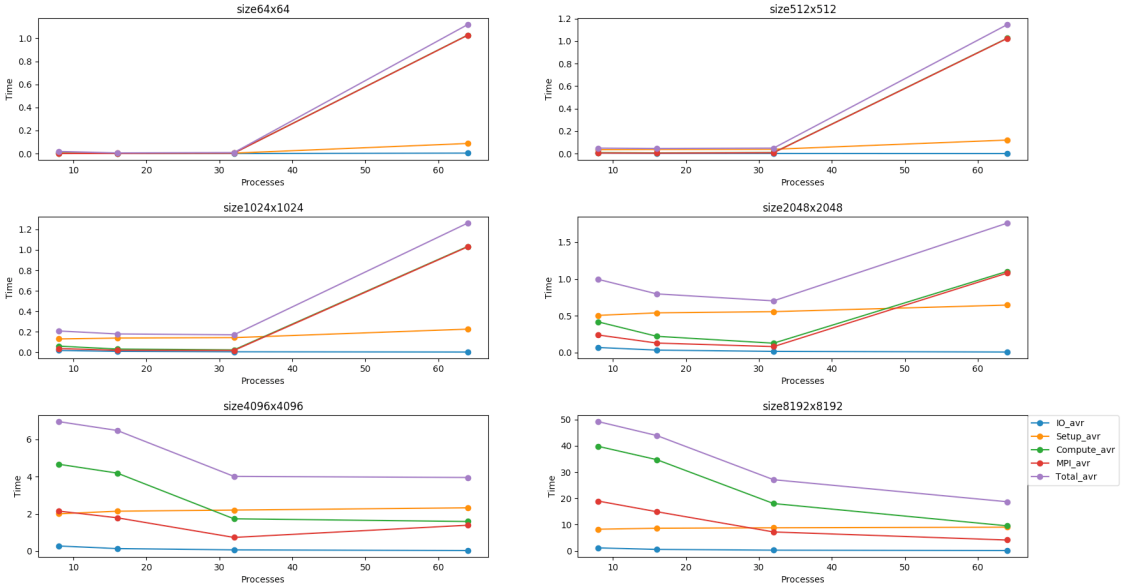

Fig. 5.2 Times with fixed input sets and varying process counts for Sandy Bridge node

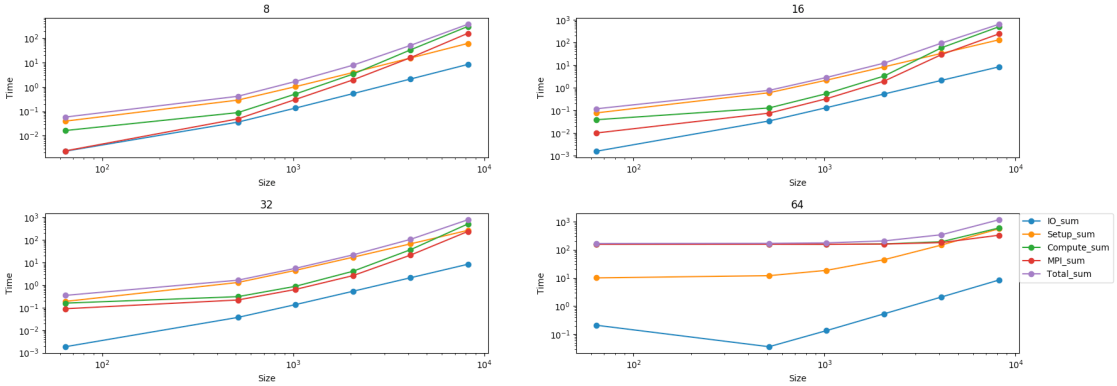

Fig. 5.3 The times with fixed process counts and varying size of input for Haswell node

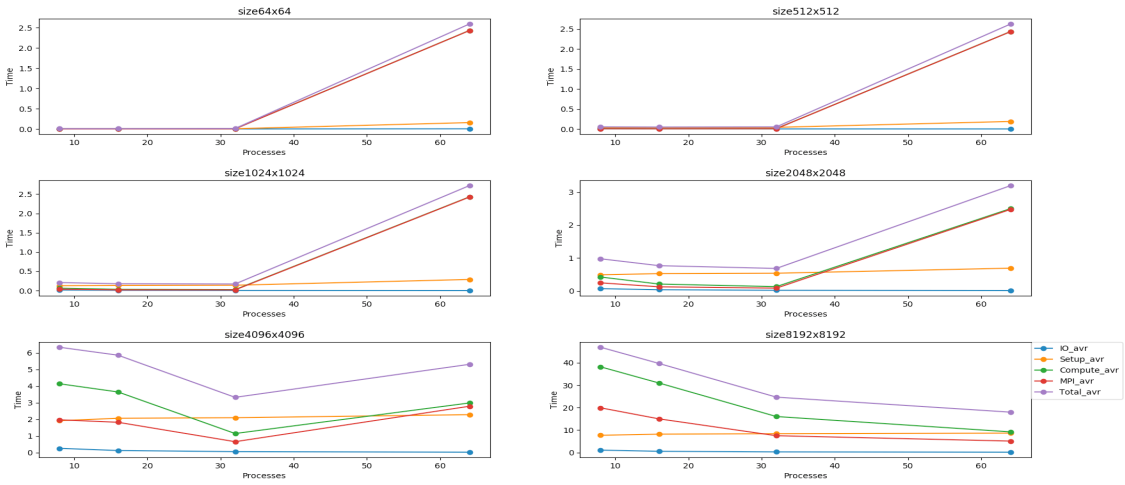

Fig 5.4 Times with fixed input sets and varying process counts for Haswell node

5.2 Questions

Q5.2.1.

I used both *MPI_Win_fence* and *Post-Start-Complete-Wait*. I choose one of both according to whether the processes need the same data or not. For example, in Setup stage, processes except process 0 need specific address in 1-D global matrix. Therefore, I call process 0 to put these addresses into other process by *MPI_Win_fence* and *MPI_Put*. In contrast, in Computation stage, because downstairs processes need the same pivots from the upstairs process, I used *Post-Start-Complete-Wait* to let downstairs process get the pivots by *MPI_Get* at the same period. I plan that *Post-Start-Complete-Wait* will cause overlap also our goal.

Q5.2.2.

No, from Fig 5.5 and Fig 5.6, the conclusion is the same with Q4.2.2. I found that downstairs processes still wait, that's *MPI_Win_complete,* until the upstairs process *posts* the pivots. When the downstairs processes get pivots, then they compute the new pivots. In my view, it's the drawback of gaussian elimination algorithm itself. The sequent processes need to get data firstly to compute otherwise do nothing, just like blocking. Or the way which I use to implement the code is more like the blocking communication.

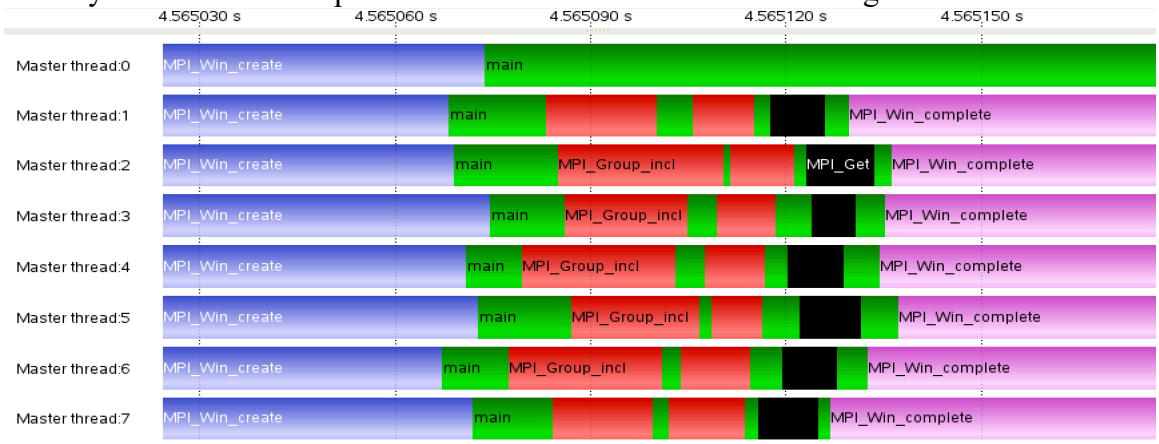

Fig. 5.5 Processes except process 0 execute *MPI_Get* and then execute *MPI_Win_complete.* Process 0 still compute the first pivots.

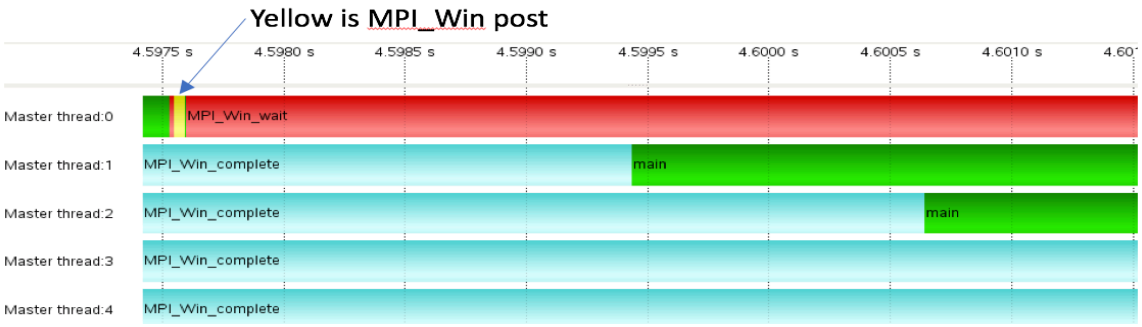

Fig. 5.6 Other processes start computing when process 0 post the pivots

Q5.2.3.

From Table 5.1 and 5.2, one-side communication has almost no speed up. And under certain conditions like problem size smaller than 2048, it has bad performances. Moreover, under same size condition, with 64 process, the performances are very bad.

Table 5.1 the speedup observed versus the baseline for Sandy Bridge nodes.

Size	Num_process	IO_avr	Setup_avr	Compute_avr	MPI_avr	Total_avr	IO_sum	Setup_sum	Compute_sum	MPI_sum	Total_sum
size64x64	8	0.504	0.348	0.246	1.079	0.454	0.504	0.348	0.246	1.079	0.454
size64x64	16	1.215	0.360	0.430	1.039	0.403	1.215	0.360	0.430	1.039	0.403
size64x64	32	92.905	0.305	0.334	0.437	0.976	92.905	0.305	0.334	0.437	0.976
size64x64	64	0.645	0.911	0.407	0.408	0.447	0.645	0.911	0.407	0.408	0.447
size512x512	8	0.996	0.922	0.766	0.854	0.901	0.996	0.922	0.766	0.854	0.901
size512x512	16	0.994	0.930	0.673	0.771	0.898	0.994	0.930	0.673	0.771	0.898
size512x512	32	0.985	0.924	0.535	0.586	0.852	0.985	0.924	0.535	0.586	0.852
size512x512	64	1.030	0.956	0.408	0.409	0.466	1.030	0.956	0.408	0.409	0.466
size1024x1024	8	1.006	0.976	0.899	0.879	0.956	1.006	0.976	0.899	0.879	0.956
size1024x1024	16	1.002	0.972	0.935	0.941	0.967	1.002	0.972	0.935	0.941	0.967
size1024x1024	32	0.998	0.980	0.795	0.810	0.956	0.998	0.980	0.795	0.810	0.956
size1024x1024	64	0.998	0.974	0.555	0.555	0.631	0.998	0.974	0.555	0.555	0.631
size2048x2048	8	0.999	0.990	0.969	0.958	0.982	0.999	0.990	0.969	0.958	0.982
size2048x2048	16	1.002	0.993	0.964	0.953	0.985	1.002	0.993	0.964	0.953	0.985
size2048x2048	32	1.068	1.064	0.919	0.895	1.038	1.068	1.064	0.919	0.895	1.038
size2048x2048	64	0.997	0.995	0.468	0.458	0.664	0.997	0.995	0.468	0.458	0.664
size4096x4096	8	0.999	0.994	0.995	0.993	0.995	0.999	0.994	0.995	0.993	0.995
size4096x4096	16	1.022	1.018	0.983	0.979	0.995	1.022	1.018	0.983	0.979	0.995
size4096x4096	32	1.002	1.003	0.992	1.009	0.998	1.002	1.003	0.992	1.009	0.998
size4096x4096	64	0.997	0.998	0.431	0.354	0.769	0.997	0.998	0.431	0.354	0.769
size8192x8192	8	0.975	0.970	0.991	0.988	0.987	0.975	0.970	0.991	0.988	0.987
size8192x8192	16	0.995	0.989	0.993	0.979	0.993	0.995	0.989	0.993	0.979	0.993
size8192x8192	32	1.003	1.005	0.990	0.989	0.995	1.003	1.005	0.990	0.989	0.995
size8192x8192	64	0.995	0.998	0.961	0.953	0.979	0.995	0.998	0.961	0.953	0.979

Table 5.2 the speedup observed versus the baseline for Haswell nodes.

	Num_process	IO_avr	Setup_avr	Compute_avr	MPI_avr	Total_avr	IO_sum	Setup_sum	Compute_sum	MPI_sum	Total_sum
size64x64	8	55.421	0.215	0.195	1.297	2.400	55.421	0.215	0.195	1.297	2.400
size64x64	16	248.770	0.271	0.305	1.157	3.623	248.770	0.271	0.305	1.157	3.623
size64x64	32	62.408	0.230	0.351	0.620	0.614	62.408	0.230	0.351	0.620	0.614
size64x64	64	0.839	1.251	0.218	0.218	0.281	0.839	1.251	0.218	0.218	0.281
size512x512	8	0.977	0.864	0.660	0.770	0.830	0.977	0.864	0.660	0.770	0.830
size512x512	16	1.005	0.877	0.537	0.687	0.826	1.005	0.877	0.537	0.687	0.826
size512x512	32	0.995	0.890	0.450	0.554	0.811	0.995	0.890	0.450	0.554	0.811
size512x512	64	0.950	1.224	0.210	0.210	0.283	0.950	1.224	0.210	0.210	0.283
size1024x1024	8	0.991	0.942	0.887	0.886	0.929	0.991	0.942	0.887	0.886	0.929
size1024x1024	16	1.000	0.950	0.879	0.913	0.939	1.000	0.950	0.879	0.913	0.939
size1024x1024	32	0.997	0.962	0.624	0.630	0.909	0.997	0.962	0.624	0.630	0.909
size1024x1024	64	0.989	1.145	0.823	0.824	0.857	0.989	1.145	0.823	0.824	0.857
size2048x2048	8	0.997	0.978	0.961	0.956	0.972	0.997	0.978	0.961	0.956	0.972
size2048x2048	16	0.992	0.981	0.973	0.971	0.979	0.992	0.981	0.973	0.971	0.979
size2048x2048	32	0.997	0.991	0.895	0.874	0.974	0.997	0.991	0.895	0.874	0.974
size2048x2048	64	0.991	1.063	0.362	0.356	0.514	0.991	1.063	0.362	0.356	0.514
size4096x4096	8	1.005	0.998	1.013	1.015	1.008	1.005	0.998	1.013	1.015	1.008
size4096x4096	16	0.991	0.988	0.997	1.018	0.994	0.991	0.988	0.997	1.018	0.994
size4096x4096	32	1.006	1.009	0.826	0.743	0.946	1.006	1.009	0.826	0.743	0.946
size4096x4096	64	1.010	1.035	0.271	0.223	0.605	1.010	1.035	0.271	0.223	0.605
size8192x8192	8	0.997	0.990	0.999	0.996	0.997	0.997	0.990	0.999	0.996	0.997
size8192x8192	16	0.996	0.992	0.992	0.990	0.992	0.996	0.992	0.992	0.990	0.992
size8192x8192	32	0.998	1.001	0.990	0.991	0.993	0.998	1.001	0.990	0.991	0.993
size8192x8192	64	0.996	1.007	0.857	0.810	0.930	0.996	1.007	0.857	0.810	0.930

Q5.2.4.

From Table 5.3, there is no speed up. From Table 5.4, one-side communication has almost no speed up. Like question5-3, under the same size condition with 64 processes, the performances are also bad like problem size 4096 and 2048.

Table 5.3 the speedup observed versus the non-blocking for Sandy Bridge nodes.

Size	Num_process	IO_avr	Setup_avr	Compute_avr	MPI_avr	Total_avr	IO_sum	Setup_sum	Compute_sum	MPI_sum	Total_sum
size64x64	8	0.020	0.330	0.250	1.122	0.099	0.020	0.330	0.250	1.122	0.099
size64x64	16	87.756	0.332	0.368	0.918	2.451	87.756	0.332	0.368	0.918	2.451
size64x64	32	0.903	0.284	0.305	0.397	0.300	0.903	0.284	0.305	0.397	0.300
size64x64	64	0.007	0.316	0.530	0.530	0.511	0.007	0.316	0.530	0.530	0.511
size512x512	8	0.991	0.909	0.756	0.821	0.889	0.991	0.909	0.756	0.821	0.889
size512x512	16	1.011	0.925	0.615	0.683	0.887	1.011	0.925	0.615	0.683	0.887
size512x512	32	1.008	0.916	0.510	0.555	0.841	1.008	0.916	0.510	0.555	0.841
size512x512	64	1.029	0.518	0.531	0.532	0.530	1.029	0.518	0.531	0.532	0.530
size1024x1024	8	1.005	0.962	0.881	0.854	0.942	1.005	0.962	0.881	0.854	0.942
size1024x1024	16	1.006	0.957	0.903	0.891	0.950	1.006	0.957	0.903	0.891	0.950
size1024x1024	32	1.026	0.965	0.680	0.658	0.928	1.026	0.965	0.680	0.658	0.928
size1024x1024	64	1.000	0.732	0.534	0.534	0.570	1.000	0.732	0.534	0.534	0.570
size2048x2048	8	1.001	0.980	0.984	0.982	0.983	1.001	0.980	0.984	0.982	0.983
size2048x2048	16	1.001	0.980	0.980	0.970	0.981	1.001	0.980	0.980	0.970	0.981
size2048x2048	32	1.008	0.983	0.871	0.821	0.963	1.008	0.983	0.871	0.821	0.963
size2048x2048	64	1.003	0.905	0.536	0.528	0.674	1.003	0.905	0.536	0.528	0.674
size4096x4096	8	1.000	0.985	0.992	0.986	0.990	1.000	0.985	0.992	0.986	0.990
size4096x4096	16	1.013	1.000	0.975	0.959	0.984	1.013	1.000	0.975	0.959	0.984
size4096x4096	32	1.000	0.992	0.969	0.951	0.982	1.000	0.992	0.969	0.951	0.982
size4096x4096	64	1.009	0.977	0.476	0.403	0.776	1.009	0.977	0.476	0.403	0.776
size8192x8192	8	0.968	0.955	0.992	0.988	0.985	0.968	0.955	0.992	0.988	0.985
size8192x8192	16	0.996	0.981	0.993	0.987	0.991	0.996	0.981	0.993	0.987	0.991
size8192x8192	32	1.002	0.995	0.984	0.971	0.988	1.002	0.995	0.984	0.971	0.988
size8192x8192	64	0.997	0.985	0.987	0.947	0.986	0.997	0.985	0.987	0.947	0.986

Table 5.4 the speedup observed versus the non-blocking for Haswell nodes.

Size	Num_process	IO_avr	Setup_avr	Compute_avr	MPI_avr	Total_avr	IO_sum	Setup_sum	Compute_sum	MPI_sum	Total_sum
size64x64	8	1.279	0.368	0.217	1.454	0.362	1.279	0.368	0.217	1.454	0.362
size64x64	16	1.106	0.254	0.346	1.316	0.296	1.106	0.254	0.346	1.316	0.296
size64x64	32	1.055	0.236	0.252	0.445	0.248	1.055	0.236	0.252	0.445	0.248
size64x64	64	0.015	0.421	0.241	0.241	0.251	0.015	0.421	0.241	0.241	0.251
size512x512	8	0.986	0.855	0.658	0.775	0.824	0.986	0.855	0.658	0.775	0.824
size512x512	16	0.995	0.851	0.514	0.635	0.801	0.995	0.851	0.514	0.635	0.801
size512x512	32	1.017	0.894	0.411	0.496	0.807	1.017	0.894	0.411	0.496	0.807
size512x512	64	0.949	0.520	0.249	0.249	0.268	0.949	0.520	0.249	0.249	0.268
size1024x1024	8	0.994	0.930	0.888	0.901	0.922	0.994	0.930	0.888	0.901	0.922
size1024x1024	16	1.009	0.943	1.026	1.176	0.962	1.009	0.943	1.026	1.176	0.962
size1024x1024	32	1.008	0.953	0.559	0.546	0.891	1.008	0.953	0.559	0.546	0.891
size1024x1024	64	1.000	0.692	0.251	0.251	0.299	1.000	0.692	0.251	0.251	0.299
size2048x2048	8	1.015	0.967	0.978	0.981	0.975	1.015	0.967	0.978	0.981	0.975
size2048x2048	16	0.995	0.967	1.209	1.324	1.034	0.995	0.967	1.209	1.324	1.034
size2048x2048	32	0.995	0.974	1.021	1.066	0.983	0.995	0.974	1.021	1.066	0.983
size2048x2048	64	0.993	0.863	0.256	0.250	0.389	0.993	0.863	0.256	0.250	0.389
size4096x4096	8	1.000	0.977	1.012	1.015	1.001	1.000	0.977	1.012	1.015	1.001
size4096x4096	16	0.989	0.974	1.021	1.039	1.003	0.989	0.974	1.021	1.039	1.003
size4096x4096	32	1.000	0.990	0.919	0.914	0.965	1.000	0.990	0.919	0.914	0.965
size4096x4096	64	1.011	0.966	0.290	0.242	0.586	1.011	0.966	0.290	0.242	0.586
size8192x8192	8	0.998	0.978	0.994	0.991	0.992	0.998	0.978	0.994	0.991	0.992
size8192x8192	16	0.998	0.982	0.991	0.987	0.989	0.998	0.982	0.991	0.987	0.989
size8192x8192	32	1.004	0.996	0.986	0.984	0.989	1.004	0.996	0.986	0.984	0.989
size8192x8192	64	0.998	0.982	0.976	0.963	0.979	0.998	0.982	0.976	0.963	0.979

Table. Assignment distribution

Task **Name**	3	4	5	script	Report
Mihai-Gabriel Robescu	P	R	P	A	P
Hsieh Yi-Han	P	P	R	A	P
Yi JU	R	P	P	R	P
Koushik Karmakar	P	R	P	A	R

P = **P**articipate **R** = be **R**esponsible to **A** = **A**bsent