Table 1: Execution time over number of processes and number of variables

Time (sec) Processes\Problem Size	10	100	1000	10000	40000
1	0.419	0.476	0.901	54.400	1018.603
2	0.434	0.445	0.814	41.507	710.624
10	0.545	0.552	0.866	31.514	506.274
20	/	0.717	1.021	30.855	491.901
40	/	1.136	1.465	34.120	532.385

Table 2: Speedup over number of processes and number of variables

Speedup Processes\Problem Size	10	100	1000	10000	40000
1	1	1	1	1	1
2	0.965	1.069	1.106	1.311	1.433
10	0.769	0.861	1.041	1.726	2.012
20	/	0.664	0.883	1.763	2.071
40	/	0.419	0.615	1.594	1.913

Note that the program read in numbers from text file, which will take quite significant amount of time when the problem size is large e.g. 10000 variables. That file reading portion of time **is significant to the total execution time of the problem. To provide a clearer interpretation of the speedup results, I also provide a version of the 2 tables that excluding the file reading time.

Problem Size	10	100	1000	10000	40000
Time Taken Reading Input (sec)	0.000554	0.003958	0.299300	27.550360	467.810500

Table 1: Execution time over number of processes and number of variables (exclude file reading time)

Time (sec) Processes\Problem Size	10	100	1000	10000	40000
1	0.419	0.472	0.602	26.850	550.793
2	0.434	0.441	0.515	13.956	242.814
10	0.545	0.548	0.566	3.964	38.464
20	/	0.713	0.721	3.304	24.091
40	/	1.132	1.166	6.569	64.575

Table 2: Speedup over number of processes and number of variables (exclude file reading time)

Speedup **Processes\Problem Size**	10	100	1000	10000	40000
1	1	1	1	1	1
2	0.965	1.069	1.168	1.924	2.268
10	0.769	0.860	1.062	6.773	14.320
20	/	0.662	0.834	8.125	22.863
40	/	0.416	0.516	4.087	8.530

a. When don't you get speedup (i.e. at what number of processes and problem sizes)?

First, we don't get speedup when the problem size is very small e.g. 10 / 100 variables, increasing number of processes doesn't help.

Second, we don't get speedup when the number of processes greater than a certain number (~20 processes in our case). It is due to the parallel algorithm design, though the problem size is large, increasing the number of processes incur more overhead (the overhead is also relative to the problem size).

b. Why you don't get speedup in the case above?

In my parallel design, it first scatters the input data evenly over number of processes, then each process calculate assigned unknown variables. After each round, all the processes need to communicate to gather one another's new values for next round.

When the problem size is small e.g. 10 / 100 variables, the **overhead of creating extra processes** and communicate small amount of content outweighs the benefit of splitting the computation time amount processes. So, we get slow down instead of speedup.

When the problem size is large e.g. 10000 / 40000 variables, we don't get further speedup after a certain number of processes (~20 processes). In my parallel design, each process has to gather the amount of unknown variables (x) data which is relative to problem size. That means each iteration, there will be **a lot of communication, the number of communication and total size are proportional to number of processes.** Increase the number of processes will increase this amount of overhead proportionally. And the increase in communication time will be more than the decrease in computation time after a certain number of processes.

c. When do you get speedup, if at all?

We get more speedup when the problem size is large and the number of processes is striking a balance between communication overhead and computation gain. (~<20 processes for problem size of 10000 and 40000)

The speedup ratio is initially large (~equal to the ratio of #processes) and will decrease along the increase in number of processes.

d. Explain c above.

As I explained in b the parallel design,

We split the calculation of new value of unknown variables among processes; this will make each process to calculate less number of unknown. Thus it speeds up the computation time of new unknown variables each round.

But when we increase number of processes, as explain in b, the number of communication and total size both proportionally increases. When the ratio of this overhead increase to the computation time gain is small (<1), we get speedup.

Since this overhead is creasing with number of processes, that's why we see decreasing speedup ratio.