LU decomposition using OpenMP

Akash Desai
akashpra@buffalo.edu
4th March, 2015

1. Run

- In given zip folder, there are folder such as Parallel, Parallel_block and seq alongside file called out.sh.
- Now './out.sh' will fetch you all the results for sample node and dimension size in '.csv' file format to plot necessary graphs.
- To get results for user given parameter, in home directory, there is file called "SLURM_MPI_Matrix_decomp.sh", now user can open the file and change the number of core and add argument at the end of the command " ./Matrix_decomp argv" to add dimension of their choice.
- To run this part, you need to run "sbatch SLURM_MPI_Matrix_decomp.sh" in your home directory.
- Here, Parallel_block gives you the results using block distribution where as Parallel folder is for cyclic distribution using 128 chunk size.
- Assignment of value to matrix has been done randomly within the range of (1,100)
- Margin of error has been given as 50. (since value of L and U were correct, though at times while verifying procedure for bigger matrix dimension (i.e 5000,7000) were carrying large error.

2. Code Implementation and Results

- For sequential code, LU decomposition is done using Gaussian elimination algorithm. This part is as it is as mentioned in the lecture slides.
- For parallelizing the code, I modified the for loops in search of getting correct and fast results. In process of doing that, I have combine 2 different for loops of sequential code into 1 loop and did parallelization on that modified part.

- Now results has been collected for dimension size 10, 100,200,500,700,1000,2000,5000, 7000, 10000. Results have been saved in different files according to number of nodes used.
- File containing all speedup values and time taken values is in the file under the name: **sample_results.csv.**
- **a. 1. Increase in size of the problem:** For initial dimension size 10, sequential code is performing better than parallel code, but as you move forward you get better speedups with increasing size. After crossing 2000 mark it decreases a bit and behave in stabilized manner.
- **2. Increase in Number of nodes:** As the number of nodes increase, I tend to get better speedups for 4 and 6 nodes, but once I crossed 2000 mark then results with 8 and 12 nodes are better. But still these speedups are not significantly better given the resources being used are almost twice a size.
- In general you can say with larger data sizes, more nodes give you better speedups but at the cost of very large resources.
- For smaller problems, I would still prefer sequential code since I am not getting superlinear speedups, so the **cost** of sequential code is much lesser than any parallel code.
- Here is the graph indicating speedup values with different size of dimension and nodes.

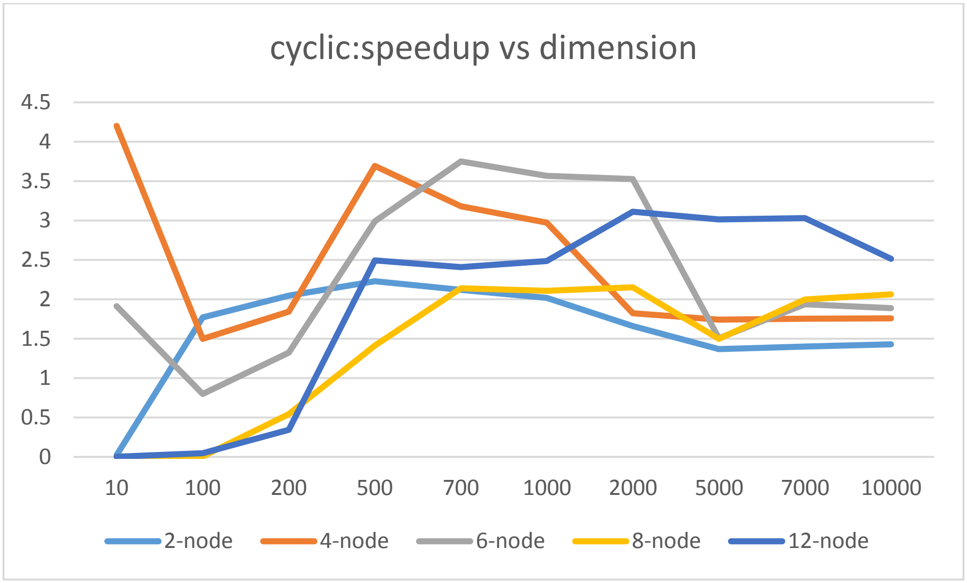

Figure 1

- **b. Impact of the memory on each node:** As I increased a data size, I came to know that for values 30000 and 40000, My job automatically gets cancelled due to lack of the memory on node. Now for each node size there is peak speed up as shown in graph above where it get perfect memory distribution. After that point it doesn't get back to that point. This shows that for every number of threads, there is a data size for which it works better than others, so basically we need to find out that number of threads value for different dimension.
- **c.** This shows that LU decomposition not highly scalable since for larger value you can allocate all the data on 1 Main node. You need more nodes to store that large data, which is not feasible using OpenMP given its communication contraints.
- Moreover to get better speedup for extremely large data, you need more than 12 nodes on 1 Main node, which requires communication, so it is not possible using OpenMP.

- **d.** We have 2 graphs, one representing cyclic distribution and other using block distributin.(Figure 1 and Figure 2)
- Analyzing 2 graphs, you can infer that cyclic distribution works better when there are lesser nodes to work with(i.e. 1,2,4,6) where as in case of node size =8 or 12 block distribution works better than cyclic.
- That is because when you are using 12 nodes of 1 Main node, everybody gets equal division of tasks, so there is hardly a case where one node is working and other is staying ideal. So in this case, block distribution works better than Cyclic distribution.

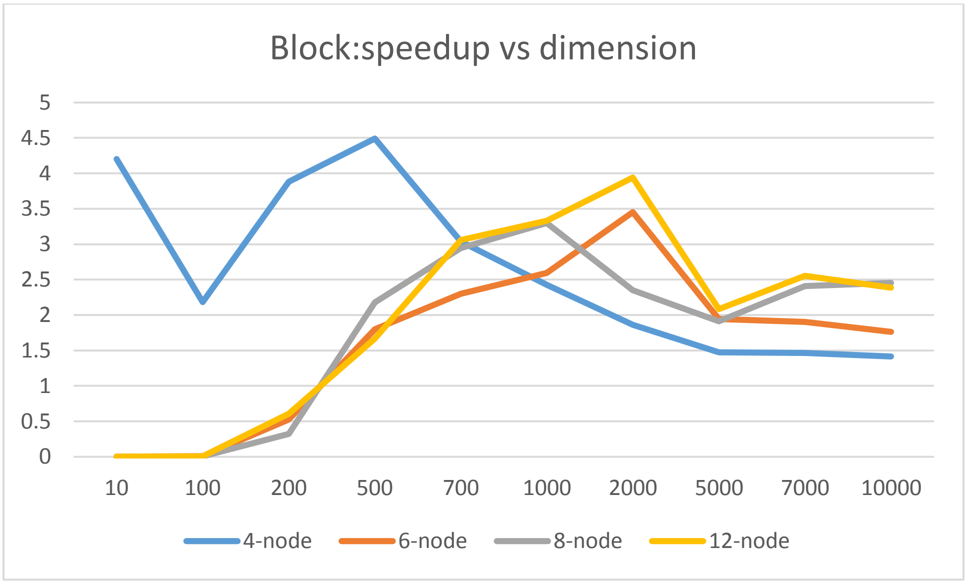

Figure 2.