LU decomposition using OpenMP

Akash Desai
akashpra@buffalo.edu
13th March, 2015

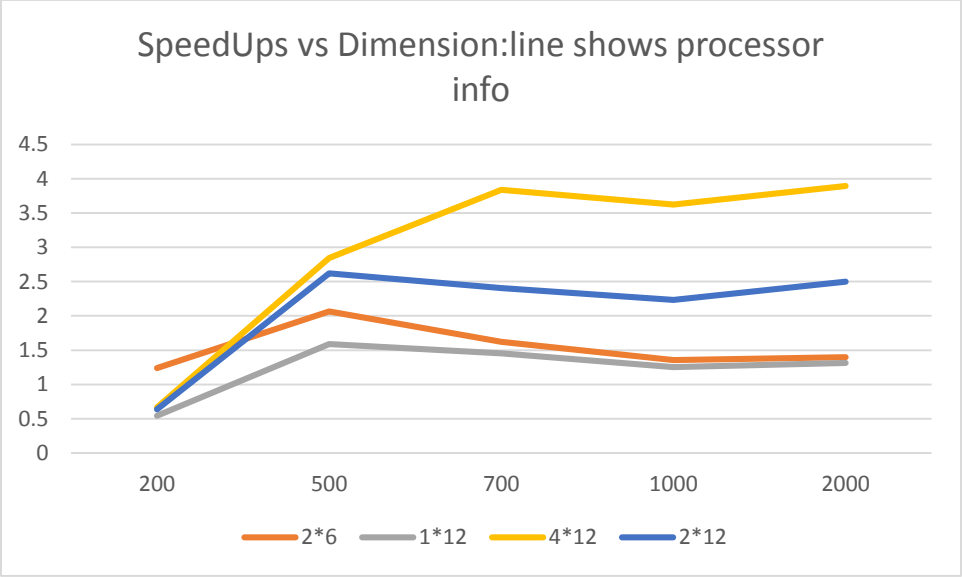
1. Run

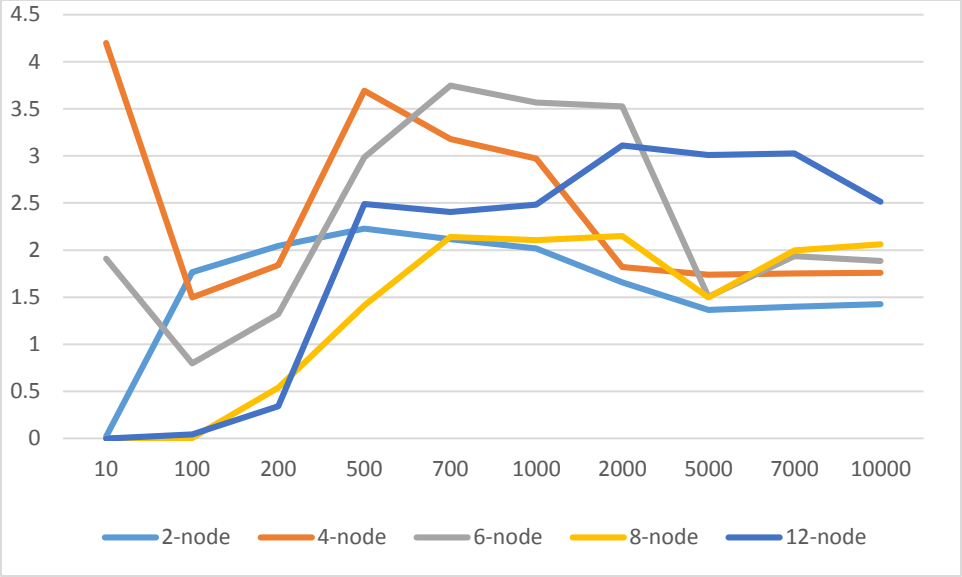
- In given zip folder, there is a folder called OpenMpi.
- This time I have provided the shell script which we can run using sbatch command and passing 5 arguments as size of dimension.
- Basically this program will decompose for each of those cases.
- Since I have verified my results on local system and the method of doing that has been mentioned in previous assignments, I chose to ignor those method in main function for tester's convenience so he/she doesn't have to wait till it cross checks matrix multiplication.
- Example of shell file run: sbatch SLURM_MPI_LU.sh 1000 2000 5000 7000 10000
- SLURM_MPI_LU.sh file is there in the OpenMPI folder. One can manually go there and change the number of cores and processor, if he/she wants to.
- All the result of given arguments will be stored in a '.csv' file, from which I have collected the data and plotted the graphs.

2. Code Implementation and Results

- http://www.cs.berkeley.edu/~demmel/MS_Intel_ParLab_Apr08/Li_SparseLU_multicore.pdf. I have used only one 2D array to store both L and U component's value and the implementation is based on this paper, presented by X.Sherry Li under the UCB publication.
- All the logical implementation of this algorithm is same as performed in OpenMP and sequential code.
- Result sets that I have presented here are not huge , though I have kept a special case where I went till 10000. Since server's nodes unviability, I restricted data sizes to 2000.
- File containing all the results, is saved as, sample_results.csv in the folder given.
- I have tested the code on 2*6, 4*6, 2*12, 1*12 and 2*6, an results of those are stored in csv file.

- **a. 1. Increase in size of the problem:** Here with the increasing size the difference in speed is not drastic that is mainly to communication bandwidth. Since processer across different nodes take longer time to communicate and in our case we need to transfer huge data on each participating processor, Still once you cross 200 dimension size, speed varies between 2 to 3. In case of 4*6 processors, I got speedups near 4 for 20000 data size. So for extremely large size data MPI would perform better due to not lacking memory space, else OpenMP perform much better compare to MPI tools.
- **2. Increase in Number of nodes:** Here, in this case having more number of nodes doesn't give you super linear speed ups but in comparison to rise in OpenMP, here number of node do have a say in determining the system's speed.
- In general you can say with larger data sizes, more nodes give you better speedups but at the cost of very large resources.
- Here is the graph indicating speedup values with different size of dimension and nodes.
- Below that I have shown OpenMP graph to compare the performances of both systems.
- By the looks of the graph it is very evident that OpenMP I much cost efficient than OpenMPI.





OpenMP

- **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.
- Whereas In OpenMPI, we know that it can cope with the large data size, though It has to face the communication lag, but still for high dimension LU decomposition, OpenMPI Is better suited to OpenMP.
- **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 constraints.

- 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.

- Whereas in OpenMPI you can see that the speedups are getting better with the larger data since now it has more workspace to work on. In openMp you are not supposed to change the node so the memory access is very limited but in OpenMPI, you have large exposure to memory so comparatively, OpenMPI's code is more scalable.

- **I have not implemented cyclic ditrisbution of data in OpenMPI though It has been implemented in OpenMP.(assignment:1.)**

- **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.

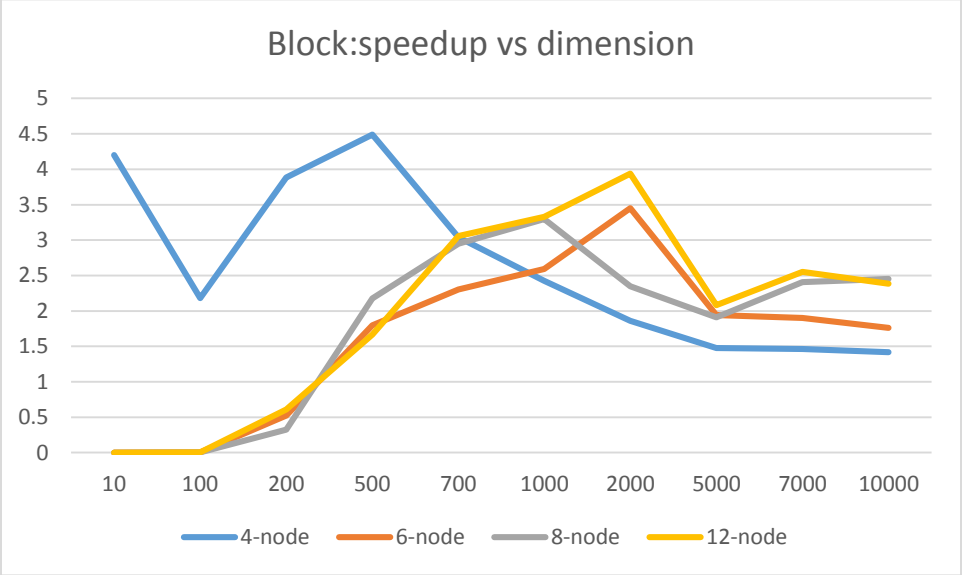


Figure 2.

Extra credit question has been discussed during the 3.a 3.b and 3.c, Other data and code related to that is available on assignment-1.