Matthew Riddell
CSCI 551
Assignment 4B

This is the professional write up for Assignment 4B. This program stores data as a one dimensional array of type double using Intel's type of memory allocation. The data is aligned with a 64 byte boundary. It exploits parallelism in four sections: it generates random numbers in the initial matrixes and to set initial array values; it sets multiplier arrays in the forward elimination step to preserve the multiplier values; it processes forward elimination in both row and column; it swaps rows when executing a partial pivot. In my implementation, only the sections which could be best parallelized were even parallelized in the first place, which avoids any needs to synchronize the parallelism.

The following is a bulleted formatted pseudocode to highlight key elements of my parallelization strategy (note: sections in bold are parallelized):

- **Generate random matrix values.**
- For Loop for rows zero to n-1.
    - Find row with max value.
    - **Perform partial pivot with max value row.**
    - **Perform forward elimination for all rows below current row.**
- Calculate solutions vector.
- Print data.
- Calculate the sum of squares.
    - Print L^2-norm.

The compiler/linker command used was: icc -O3 -qopenmp -fast -align *.c -Wall -W -Werror. The -O3 was chosen because it was the highest level of compiler optimization available. The -fast flag was chosen because it produces the quickest results, however the drawback is a loss in precision through the -no-prec-div flag it enables. This speed boost also includes the flags of -ipo, -static, and -xP. The -align flag was chosen to align the matrix and vector data for better access times.

The justification for the choice in compiler/linker flags was that the only emphasis placed in this assignment was in regards for speed, given that the sum of squares was significiantly less than one. Therefore, I elected for a loss in precision in order to maximize this run speed, and would have chosen differently if more emphasis had been placed on minimizing the sum of squares value. Given the chosen focus and comparison to timing runs of fellow students from the collective spreadsheet, this seems to have been a successful choice as the best timing runs were very close or better than the highlighted best run time values in each category of number of cores.

The following pages contain graphs for the requested information.

Table for all runs, time to execute, L^2-norm, minimum values highlighted.

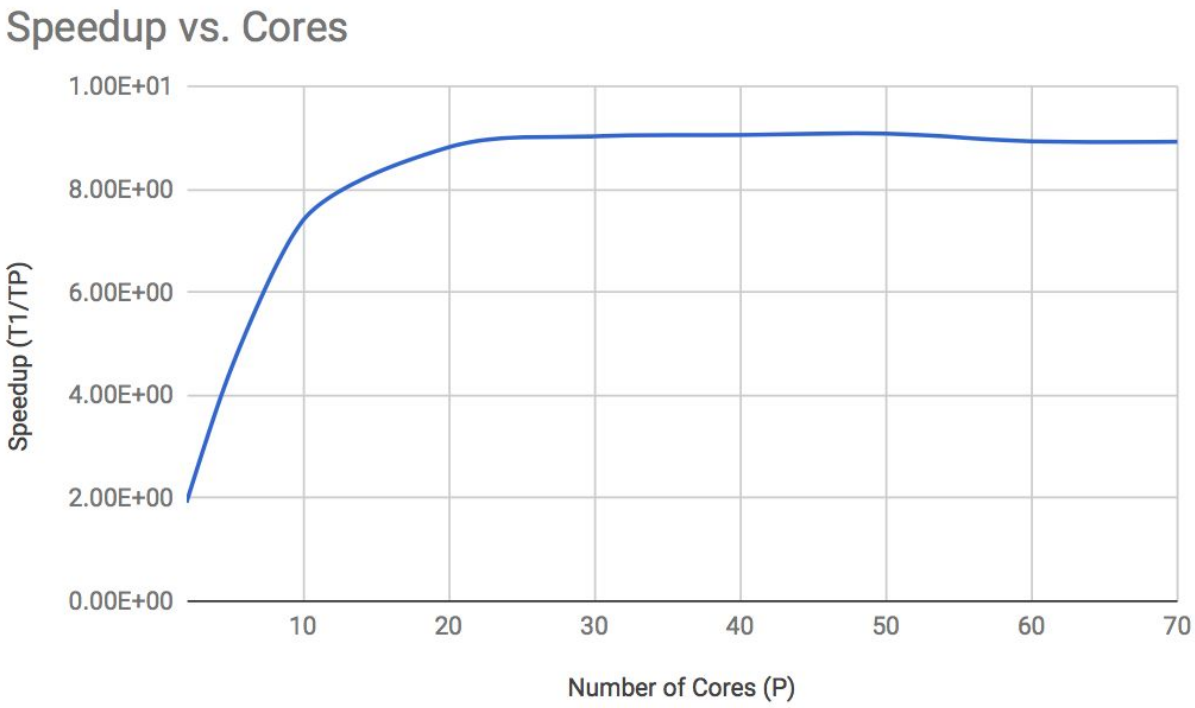
(Note: L is abbreviated for L^2-norm for visual reasons.)

Cores	T1	L1	T2	L2	T3	L3
1	**3.22E+02**	1.47E-03	3.33E+02	3.61E-04	3.27E+02	6.63E-04
2	**1.68E+02**	4.06E-04	1.73E+02	9.59E-04	1.76E+02	1.76E-03
5	**7.20E+01**	2.30E-03	7.64E+01	1.36E-03	7.54E+01	8.81E-04
10	**4.35E+01**	3.83E-03	4.36E+01	6.05E-04	4.35E+01	1.84E-03
20	**3.65E+01**	2.28E-03	3.65E+01	6.30E-04	3.65E+01	6.29E-04
30	**3.56E+01**	5.39E-03	3.58E+01	1.27E-03	3.58E+01	2.81E-03
40	**3.55E+01**	3.33E-03	3.55E+01	1.26E-02	3.56E+01	1.51E-03
50	**3.54E+01**	8.80E-04	3.55E+01	3.35E-03	3.55E+01	6.78E-04
60	**3.60E+01**	4.69E-03	3.61E+01	9.85E-04	3.60E+01	5.62E-04
70	**3.60E+01**	2.19E-03	3.61E+01	5.73E-04	3.60E+01	1.50E-03

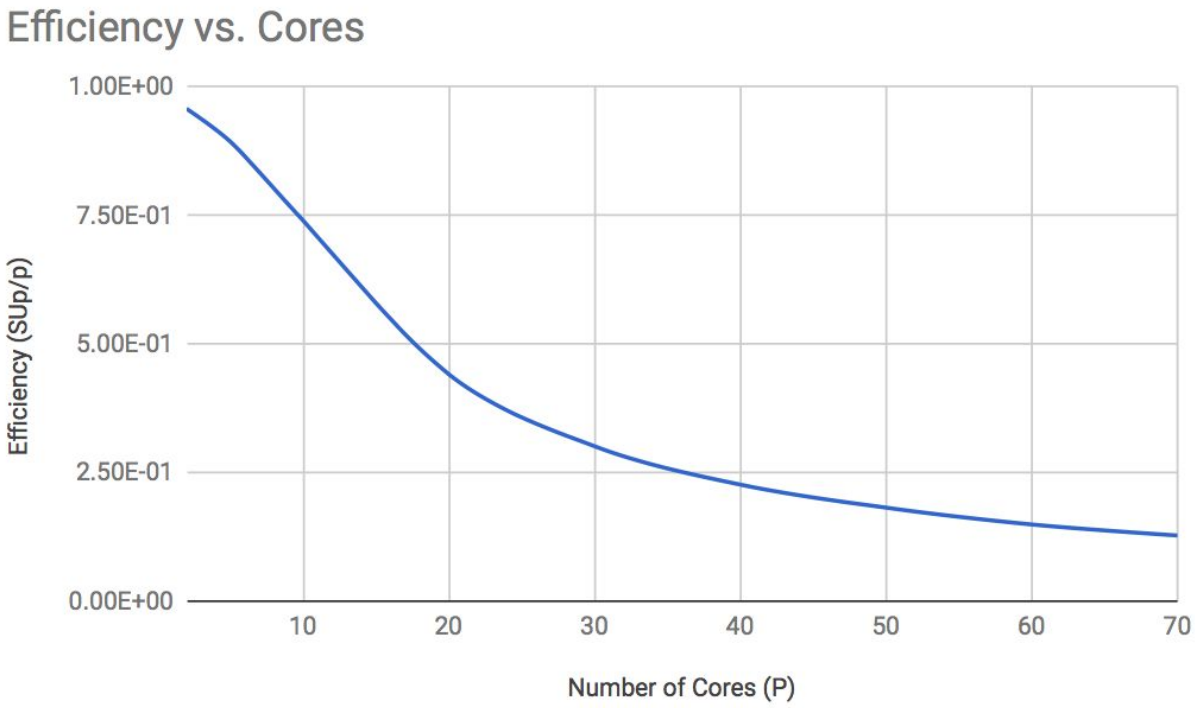
Table of minimum times, calculated speedup, efficiency, L^2-norm

Cores	Times	Speedup	Efficiency	L^2-norm
1	3.22E+02	x	x	1.47E-03
2	1.68E+02	1.92E+00	9.58E-01	4.06E-04
5	7.20E+01	4.47E+00	8.94E-01	2.30E-03
10	4.35E+01	7.41E+00	7.41E-01	3.83E-03
20	3.65E+01	8.83E+00	4.42E-01	2.28E-03
30	3.56E+01	9.04E+00	3.01E-01	5.39E-03
40	3.55E+01	9.07E+00	2.27E-01	3.33E-03
50	3.54E+01	9.10E+00	1.82E-01	8.80E-04
60	3.60E+01	8.95E+00	1.49E-01	4.69E-03
70	3.60E+01	8.94E+00	1.28E-01	2.19E-03

Graph of Speedup



Graph of efficiency



Sources:
https://software.intel.com/en-us/articles/tutorial-on-intel-xeon-phi-processor-optimization
https://computing.llnl.gov/tutorials/openMP/
https://software.intel.com/en-us/articles/more-work-sharing-with-openmp
https://software.intel.com/en-us/node/684329
http://scv.bu.edu/computation/bladecenter/manpages/icc.html