Parallel implementations of LU decomposition

Parallelization strategy

The outer loop (`for k = 1 to n`) is not parallelizable. This is because each of the runs for a given k needs the values from the runs for the other values of k. For each of these values of k, there are further 3 loops that are parallelizable. Of which two are of size O(n-k) and the other is of size $O((n-k)^2)$. The gain from parallelizing the loops of order (n-k) is negligible as we are dealing with the values of n of order thousand. So we parallelize only the following loop which is of order $(n-k)^2$.

```
for i = k+1 to n
  for j = k+1 to n
    a(i,j) = a(i,j) - l(i,k)*u(k,j)
```

.

The following are the results of parallelization using p-threads. The first row has the n values and the first column has the number of threads.

	1000	2000	4000	8000
1	1.22433	9.8663	78.8312	635.48
2	0.801779	5.95288	47.2543	364.004
4	0.44014	3.3106	25.6815	201.851
8	0.68412	4.1509	26.9366	200.113
16	0.779398	4.13246	26.9589	207.342
32	0.858359	4.35256	27.545	207.833

On trying to parallelize the following order (n-k) loops along with the $(n-k)^2$ loops, the following were the results.

```
for i = k+1 to n
  l(i,k) = a(i,k)/u(k,k)
  u(k,i) = a(k,i)
```

```
for i = k to n
  if max < |a(i,k)|
    max = |a(i,k)|
    k' = i
```

	1000	2000	4000	8000
1	1.22433	9.8663	78.8312	635.48
2	0.851865	6.09036	46.9268	370.731
4	0.617773	3.72504	28.7656	219.044
8	1.13597	5.16066	32.2272	221.71
16	1.62388	6.19103	34.1242	225.167
32	2.514	8.03862	36.8921	227.779

While minimal gains were observed in a few cases as one would expect without considering the communication overhead, Mostly this extra parallelization only increased the total time. Hence we hereafter consider the parallelization of the order $(n-k)^2$ loop only.

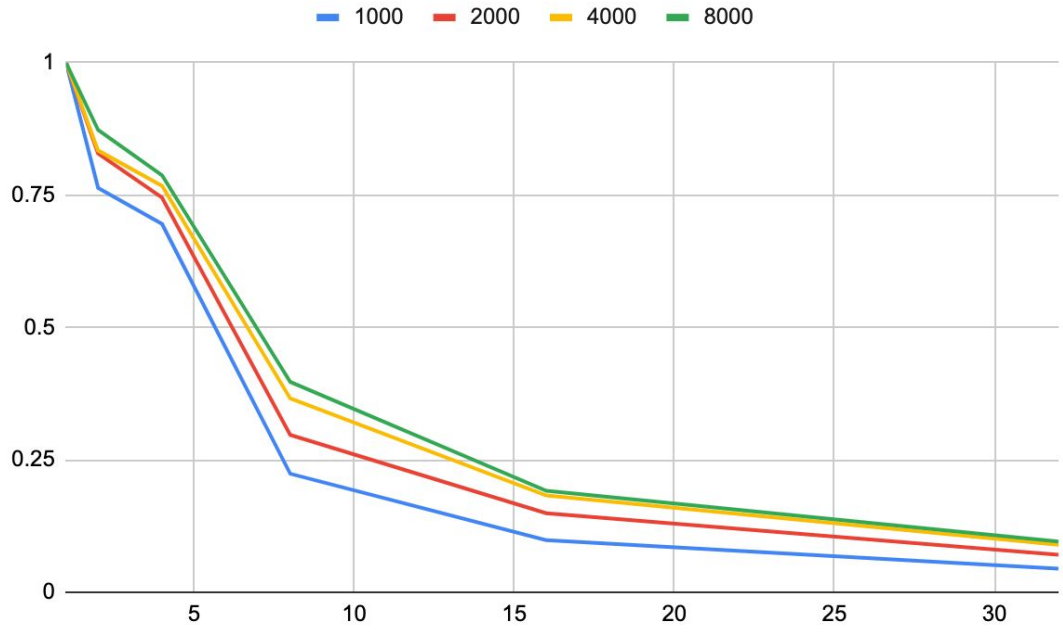
Parallel efficiency - Pthreads

The timings as mentioned above are:

	1000	2000	4000	8000
1	1.22433	9.8663	78.8312	635.48
2	0.801779	5.95288	47.2543	364.004
4	0.44014	3.3106	25.6815	201.851
8	0.68412	4.1509	26.9366	200.113
16	0.779398	4.13246	26.9589	207.342
32	0.858359	4.35256	27.545	207.833

The parallel efficiency of a program is the speed-up divided by the number of processors. Where speed-up is the time after parallelization divided by the minimum time a sequential program would take. The following are the parallel efficiencies for the above.

	1000	2000	4000	8000
1	1	1	1	1
2	0.7635083982	0.8286997218	0.8341166836	0.8729024956
4	0.6954207752	0.7450537667	0.7673928704	0.7870657069
8	0.223705271	0.2971132766	0.3658182547	0.3969507228
16	0.09817913954	0.1492195327	0.1827578277	0.1915554977
32	0.04457378847	0.07083690403	0.08943456163	0.09555147643



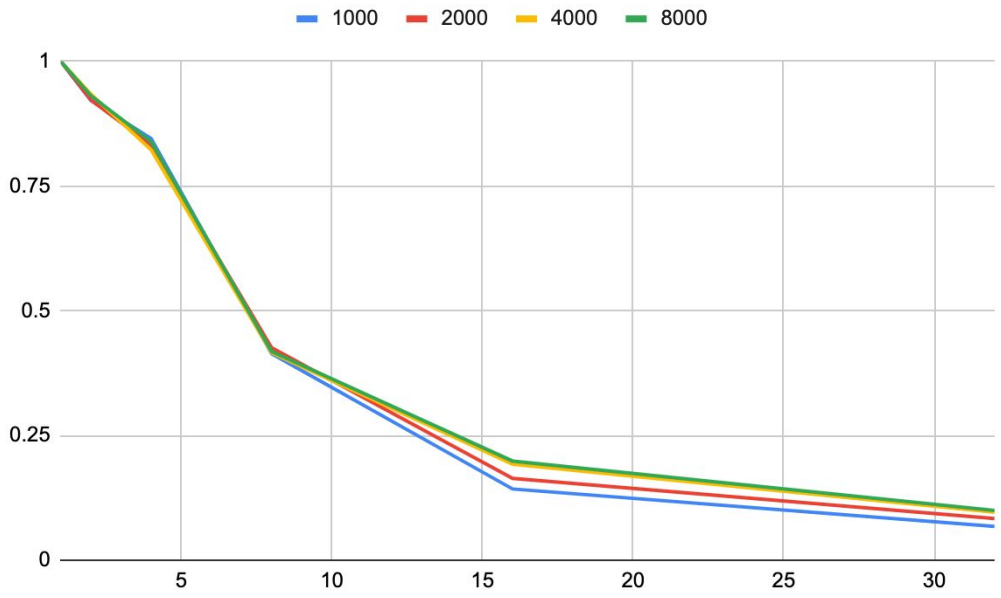
Parallel efficiency - OpenMP

By using OpenMP instead of p-threads, the following are the results. The first row has the n values and the first column has the number of threads.

	1000	2000	4000	8000
1	1.21633	9.8785	79.1213	634.505
2	0.65974	5.35339	42.3478	340.591
4	0.359648	2.96804	24.0533	189.106
8	0.367458	2.90077	23.7772	189.46
16	0.532586	3.76182	25.7344	199.586
32	0.56114	3.6937	25.7131	198.993

The parallel efficiency of a program is the speed-up divided by the number of processors. Where speed-up is the time after parallelization divided by the minimum time a sequential program would take. The following are the parallel efficiencies for the above.

	1000	2000	4000	8000
1	1	1	1	1
2	0.9218252645	0.9226396732	0.9341843024	0.931476463
4	0.8455003225	0.832072681	0.8223538974	0.8388218777
8	0.4137649745	0.4256843872	0.4159515208	0.4186272828
16	0.1427386845	0.1641243467	0.1921584047	0.1986941093
32	0.06773766351	0.0835755814	0.09615879163	0.09964310931



Deductions

This code was run on a processor with 8 cores. That is the reason that most of the times the time of the code run on 8 threads is the least out of all with minor deviations. Similarly we checked that the same code run on a processor with 4 cores gave best performance with 4 threads.

Sequential part of the program

On increasing the number of threads, the speed-up $\frac{(sequential\ part + parallel\ part)}{(sequential\ part + parallel\ part/P)}$ converges to a constant value which is $\frac{(sequential\ part + parallel\ part)}{(sequential\ part)}$ when p tends to infinity. This is the inverse of the sequential factor.