Long Nguyen

OpenMP Gaussian Elimination

Data Use and Parallel Approach

I store the date using heap array instead of stack. The reason is that with heap array it will less likely to be overflow since calculate L^2 norm is creating an extra of 8000x8000 array
To approaching parallel work, I look at how Gaussian Elimination math look like, it turns out that you can't parallel while pivoting since you can't really choose next pivot until the operations using the previous one is applied. So, my approach is I parallelize after partial pivot when it is "zero" the column. When programs elimination is performed it has so many serial parts because of concurrent operations decreases. It leads into problems that parallel is capped at some certain point unless increasing array size

Compiler flag: icc -W -Werror -Wall pgauss.c -qopenmp -lm -O3 -o pgauss
For the best result of parallel I didn't really use any extra flag except for -O3. So, with this way I will see how much OpenMP can do and test out my parallel logic

Sudo code

Forward Elimination

```
Loop i = 0 upto n<-1
        Loop j = i+1 upto n (parallel this whole loop)
                Double mul= A[j][i] / A[i][i]
                Loop k = 0 upto n+1
                        A[j][k] = A[j][k] - mul* A[i][k]
                End loop
                b[j] = b[j] - b[i] * mul;
        End loop
End loop
```

Back Substitution

```
Loop i = n-1 downto 0
        double s = b[i];
        Loop   j=i+1 upto n (parallel this whole loop)
                s = s - A[i][j] *x[j];
        End loop
        x[i] = s/A[i][i];
End loop
```

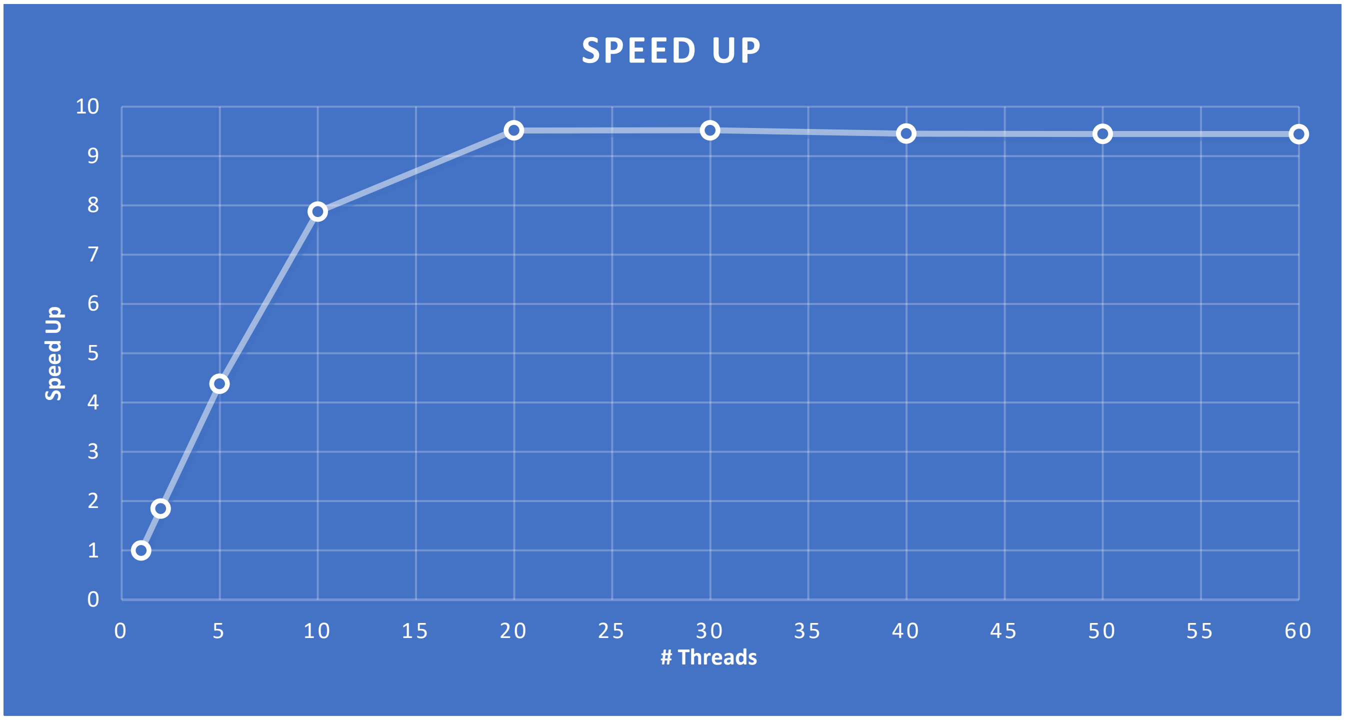
Table of 3 runtime

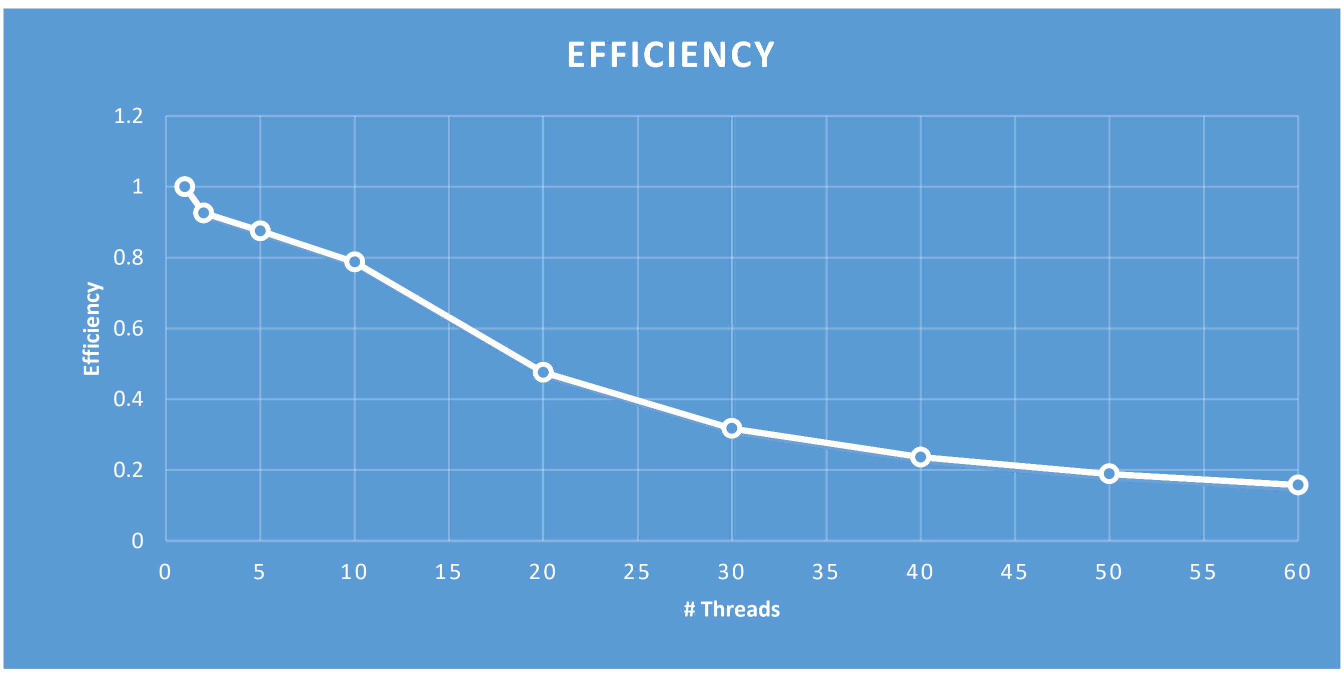
# Core	ONE		TWO		THREE	
	Run Time	Norm L^2	Run Time	Norm L^2	Run Time	Norm L^2
1	**521.89**	7.69E-06	553.2486	9.21E-06	563.255	5.97E-06
2	**281.77**	2.61E-07	291.9202	5.56E-07	289.1354	1.40E-05
5	**119.21**	2.35E-07	122.0504	5.67E-07	121.5096	2.35E-06
10	66.49	2.08E-05	**66.3047**	1.53E-06	66.70327	6.29E-08
20	54.89	1.17E-06	**54.82364**	2.25E-06	54.91751	5.29E-06
30	**54.80278**	1.27E-07	55.26304	3.95E-07	54.81608	8.53E-07
40	55.26989	8.03E-06	55.27983	1.30E-06	**55.19614**	4.50E-07
50	55.2631	9.07E-07	55.27983	1.99E-06	**55.24912**	2.68E-07
60	55.30139	1.27E-05	55.33527	3.57E-07	**55.25372**	1.53E-07

Best runtime speedup and efficiency

# Core	Run Time	Speed Up	Efficiency
1	521.89	1	1
2	281.77	1.8521844	0.926092
5	119.21	4.3779045	0.875581
10	66.3047	7.8710857	0.787109
20	54.82364	9.5194339	0.475972
30	54.80278	9.5230569	0.317435
40	55.19614	9.4551903	0.23638
50	55.24912	9.446124	0.188922
60	55.25372	9.4453371	0.157422

Graph of speedup and efficiency





Conclusion

Parallel is very much limited and depend on the size of the array at the small size array speedup is capped at certain point.

My prediction was very close to result. If I have more time, there still be a better solution for this approach by vectorize and data align and maybe we can speed it up 10 – 20 seconds more