COMP 1680
Clouds, Grids and Virtualisation.
Open MP Coursework

Student ID - 001002629

11$^{\text{th}}$ December 2020

Contents

1 Introduction

Parallel computing is a process of using more than a single Central Processing Unit (CPU) to complete a task, by implementing this process we are able to gain a significant speed up in computation. To see what effect that parallel programming has, we have been given Gauss-Seidel and Jacobi methods that are both iterative in nature as a serial program. Modifying these algorithms to take advantage of the modern CPU's ability to process data in parallel and describing the effect of any speed up that has been gained.

2 Task One

Task one is to modify the serial code to have boundary conditions set at top 30°C, bottom 60°C, left 110°C and right 140°C and optimise the code for performance. By running a complied unmodified version of the program to find the correct integers to change, as seen in Fig 1, finding these variables had been made simpler.

```
2 2 0.001000
iter = 15  difmax = 0.00068664551
30.00000 30.00000 30.00000 30.00000
40.00000 43.74931 56.24931 90.00000
40.00000 48.74931 61.24931 90.00000
30.00000 50.00000 50.00000 30.00000
```

Figure 1: Finding boundry condtions

To increase the performance and optimisation of the program,there has been implementation of the *register* keyword to variables that are used as counters. This allows the variables to be stored in CPU registers rather than in Random Access Memory (RAM) granting quicker access to these variable and therefore a faster run time. Assigning variables with short int rather then int data types to save on memory usage and using pointers for fast access to the value stored in memory.

Compilation optimisations have been applied to, with the -O1, -O2, -O3, -Ofast and -Os for both the algorithms with naming conventions to reflect the optimisation. To get the run time of the application, timing has been added to the program before the temperature array initialises and after the main loop has been completed, this then prints out the run time in microseconds with the number of iterations that have been completed.

2.1 Jacobi Results

Running the application with a range of results multiple times each then calculating the mean of the results will give a better idea of what the performance truly is, this is due to background processes that could interfere with the timing.

To reduce this factor, each test has been conducted five times with each result tabulated and the mean used as the result.

Jacobi Serial Standard || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	819254.2	2586054.4	4359586.2	5771460.8	7658377.4
200	2095438	9853599.6	-	-	-
300	3219229.8	-	41411162.6	-	-
400	4181682.2	-	-	107499281.6	-
500	5206133	-	-	-	227535684.8

Table 1: Jacobi Serial Results

Jacobi Serial Ofast || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	209628.2	633174.6	1066373.8	1446542.8	1879709.8
200	528646.8	2406211.2	-	-	-
300	776471.6	-	10053296.6	-	-
400	1021457.6	-	-	26156386.2	-
500	1240309.8	-	-	-	56795662.4

Table 2: Jacobi OFast Results

Jacobi Serial O1 || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	395844.4	1243974.8	2080688.8	2786062.8	3661934.4
200	1043890	4628138	-	-	-
300	1541510.4	-	20023377.4	-	-
400	2078744.4	-	-	51905553.8	-
500	2502911.6	-	-	-	111911494

Table 3: Jacobi O1 Results

Jacobi Serial O2 || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	207450.8	639286.6	1097610.6	1510054.8	1887337
200	529800.4	2358110.8	-	-	-
300	821621.2	-	10220483	-	-
400	1034016.4	-	-	26828475.8	-
500	1265661	-	-	-	56910394.2

Table 4: Jacobi O2 Results

Jacobi Serial O3 || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	226251.2	699155.4	1097197.6	1498684.8	1852724.4
200	567062.2	2362751.8	-	-	-
300	806064.4	-	10122013	-	-
400	1077603	-	-	27172252.8	-
500	1309601	-	-	-	55983513.6

Table 5: Jacobi O2 Results

Jacobi Serial OSmall || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	239500.4	737921.2	1231846.2	2011424	2174108.4
200	608914.4	2724475.4	-	-	-
300	998023.2	-	11688344.2	-	-
400	1512679.2	-	-	45022576	-
500	1931912	-	-	-	83599093.2

Table 6: Jacobi OSmall Results

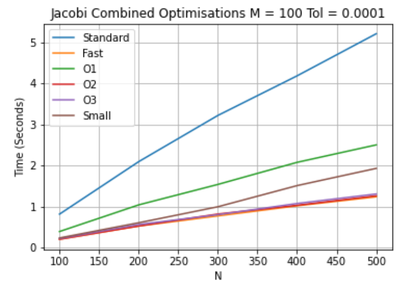

Figure 2: Jacobi Combined results. M = 100.

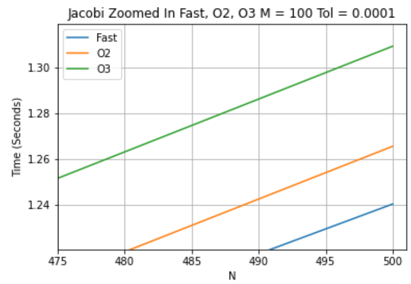

Figure 3: Jacobi Zoomed into fastest results M = 100.

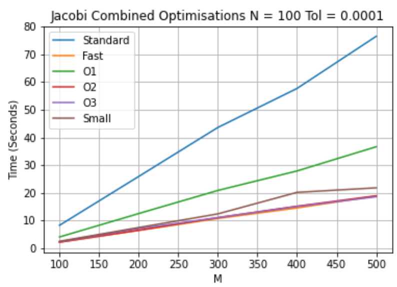

Figure 4: Jacobi Combined results. N = 100.

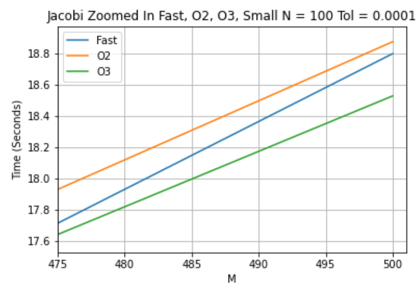

Figure 5: Jacobi Zoomed into fastest results N = 100.

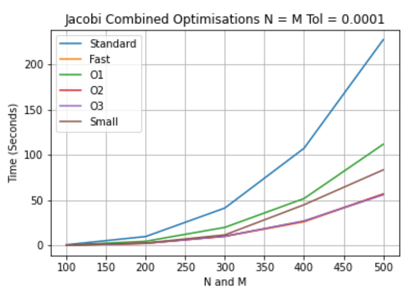

Figure 6: Jacobi Combined results. M = N.

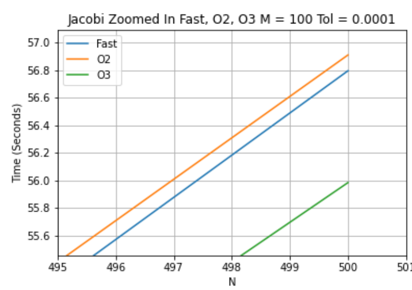

Figure 7: Jacobi Zoomed into fastest results M = N.

Observations gathered through testing different combinations of n and m values with different compile optimisations, there is a clear linear increase in time to solve the matrix using the Jacobi method. However, when n is increased in size and m is set at 100, the matrix is solved faster as seen in Figures 2 and

4. When observing the effect on increasing the matrix n m equally, an almost exponential increase in time can be inferred. Using the $O3$ optimisation gave the best results in two of the three results seen in Figures 3, 7 and 5,

2.2 Gauss Results

To test the Gauss-Seidel solution, the same operations have been run with the same optimisations as the Jacobi code. From these observations there is a clear difference between the two iterative solutions and the Jacobi method able to solve the problems used faster than Gauss-Seidel.

During testing there were no outliers to report or remove that could have skewed timing results, with results showing a linear increase in run time while keeping either n or m at a specific value and increasing both these values shows again an almost exponential increase in time to solve the matrix.

Gauss Serial Standard || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	562760.2	1764608.2	2999563	4188380.6	5226690.4
200	1506858.6	7108179.4	-	-	-
300	2343810.8	-	30700754.8	-	-
400	3087578.8	-	-	82931638.6	-
500	3853158.6	-	-	-	180133442.8

Table 7: Gauss Standard Results

Gauss Serial OFast || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	260633.2	818793.4	1357629.2	1882242.6	2539437.6
200	707242.2	3288469.2	-	-	-
300	1112775.8	-	14348157.4	-	-
400	1471445	-	-	39359960.2	-
500	1849969.4	-	-	-	86468644

Table 8: Gauss OFast Results

Gauss Serial O1 || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	276412.4	846352	1426522.8	1974325.6	2491293.4
200	743549	3569052.2	-	-	-
300	1175023	-	14972812.4	-	-
400	1542691.4	-	-	41133939.2	-
500	1912427	-	-	-	89338883.8

Table 9: Gauss O1 Results

Gauss Serial O2 || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	275048.8	856176	1477771.6	1980880.8	2608889
200	747956.4	3566378.8	-	-	-
300	1226086	-	15060915.8	-	-
400	1581262.4	-	-	41635607	-
500	1964414.8	-	-	-	90913539.6

Table 10: Gauss O2 Results

Gauss Serial O3 || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	276070.4	849378.6	1474842	1985579.2	2529182.2
200	750765.6	3574574.6	-	-	-
300	1190271.8	-	15047798.4	-	-
400	1563007.6	-	-	41725677.4	-
500	1964979.4	-	-	-	89979007.6

Table 11: Gauss O3 Results

Gauss Serial Os || Tol = 0.0001 | Time in microseconds					
n \ m	100	200	300	400	500
100	277041.4	856963.4	1431171.2	1990735.4	2535499.6
200	751560	3548913	-	-	-
300	1176688	-	15010688.6	-	-
400	1561810.4	-	-	41540306.4	-
500	1960538.6	-	-	-	89783697

Table 12: Gauss Os Results

To better visualise the results of the optimisations that have been applied, a compilation of the results into graphs has been plotted, with the values of the matrix as an axis and the time as the y axis.

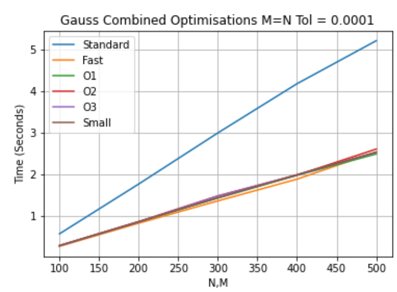

Figure 8: Gauss Combined results. M = 100.

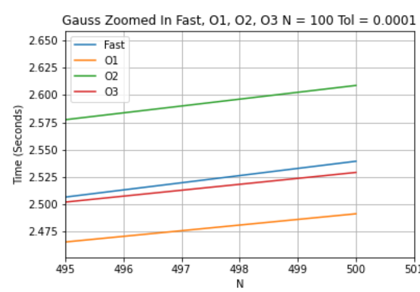

Figure 9: Gauss Zoomed into fastest results M = 100..

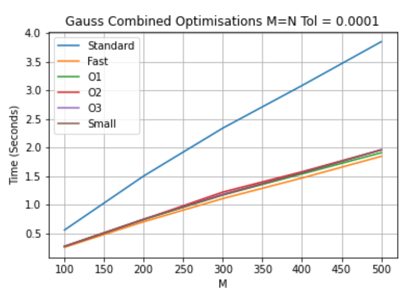

Figure 10: Gauss Combined results. N = 100.

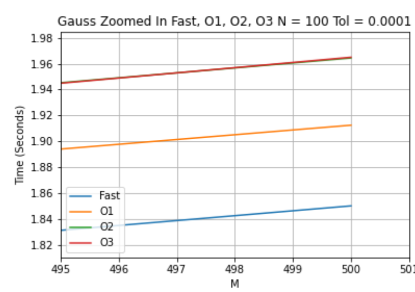

Figure 11: Gauss Zoomed into fastest results N = 100.

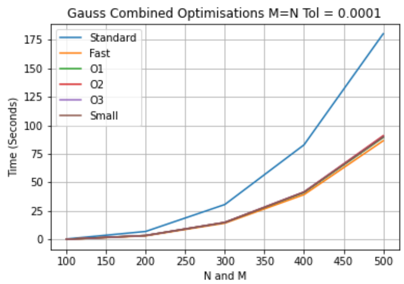

Figure 12: Gauss Combined results. M = N.

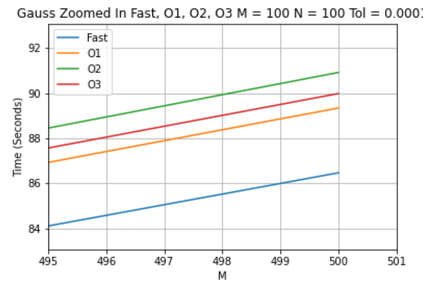

Figure 13: Gauss Zoomed into fastest results M = N..

2.3 Conclusion

With all the test performed, these observations give us an insight into how the algorithms behave with different problem sizes and how the compiler optimisations reduce the time it takes for the algorithm to resolve said problem sizes by a remarkably large margin. While optimisations Ofast, O3 and O1 have the best performance with regards to compute time reduction compared to no compiler level arguments being used.

3 Task Two

After modifying the Gauss-Seidel algorithm to make use of parallel directives, a simple test on a 20x20 problem size has been carried out over a single iteration to observe differences between single core and parallel operations. To implement parallel directives, code where the main computation is carried out is encapsulated within a parallel region. Within this parallel region the variables $diff$, $priv\_difmax$, i and j have been made private to each thread that is running. The addition of $priv\_difmax$ is critical as this allows each thread to have a private value for its computed dif max, which is compared in an additional omp critical region. With a private $priv\_difmax$ and global $difmax$ being compared in the critical region, in a single threaded manner which makes sure that each threads value is checked at this point, and if the global diff is greater than the private difference, the global variable takes the new value computed in the thread. This process repeats in a loop until the global difmax is less or equal to the tolerance value entered in the command line argument, which is 0.0001 for all the tests carried out in this report.

To force the computation to stop after a single iteration, the while loop argument has been modified so that

Threads	Difmax	Time taken(s)
1	46.66666668956	0.000022
2	46.66666668956	0.000199

Table 13: Gauss-Seidel single iteration results

From these results from Table 13 we can see that difmax is identical when calling for one or two threads, however, time to complete a single iteration is surprisingly 9.0455 times faster using a single thread. However, this can be attributed to cost overhead in spawning parallel threads. This makes the case that adding parallel loops or regions needs to be implemented in a manner which will benefit the code and achieve a desired outcome of speeding up computation rather then detracting from performance due to poor implementation.

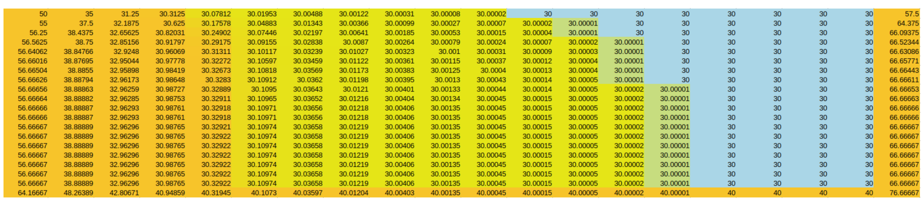

Figure 14: Gauss Single Core Output

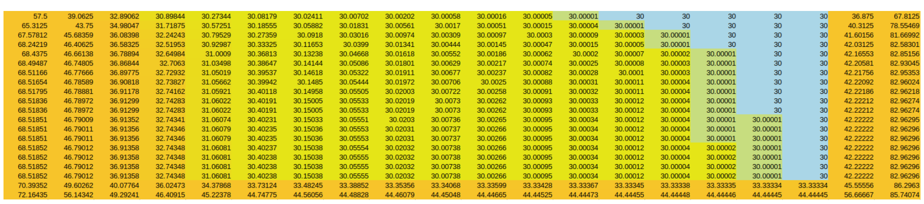

Figure 15: Gauss Dual Core Output

The resulting output from a single iteration for single and dual core can be seen in Fig 14 Fig 15 respectively. By adding a colour maps to values with a 70% cut off for maximum values for both outputs there is a clear difference between column one and five, showing faster convergence by using two threads. Faster convergence has also been observed, with two columns on the right and bottom two rows exhibiting similar behaviour. This is also further amplified by the large area on seen in Fig 14 that has had no change in its values when compered to the , shown by blue colour scales.

With the fact that running a single iteration will cost more due to overheads slowing down the computed time caused by spawning multiple threads, as it has also converged closer to tolerance value over the single iteration. This should then translate to better performance when allowing the algorithm to run its course solving for tolerance values on matrices that are larger then $20x20$ that has been tested here.

4 Task Three

With implementation of parallel regions to each algorithm complete, observing any decrease in run time using multiple cores, using 2, 4 and 8 cores testing any increase in speed up on computation. With information gathered that compile optimisations $Ofast$, $O1$ and $O3$ from serial test results have greater performance, these optimisations will be tested for performance gains with the parallel implementation that had been tested in section two with a reversion of the main computation loop back to testing $diff$ value against the tol value.

Instead of the linear problem sizes used in task one, four different problem sizes will use tested, $m = 100, n = 100$ as a small problem space to observe multi threaded overhead cost, $m = 700, n = 700$ as a larger problem size which should mitigate the overhead of spawning multiple threads. Also $m = 1000, n = 500$ and $m = 500, n = 1000$ which will have the same number of "cells" to solve but in different dimensional makeup.

4.1 Jacobi Parallel Results

Observations from parallel Jacobi code shows that this code has been implemented poorly over every optimisation level. Figure 16 shows a huge decrease in performance, especially using O3 and Ofast optimisations, which is the exact opposite behaviour that was expected from implementing parallel code. Only Fig 17 shows an increase in performance using more cores and a standard optimisation with all testing in Appendix B.1 showing no huge outlier results that could have skewed timings.

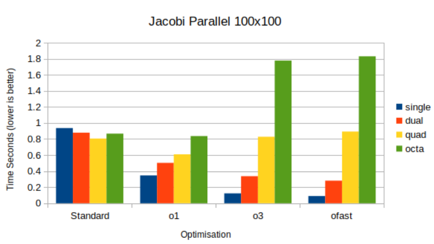

Figure 16: Jacobi Parallel 100x100

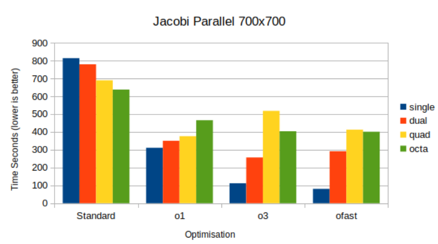

Figure 17: Jacobi Parallel 700x700

This trend continues for all problem sizes that have been tested, with single core implementation showing best performance in all but four tests which used standard optimisation.

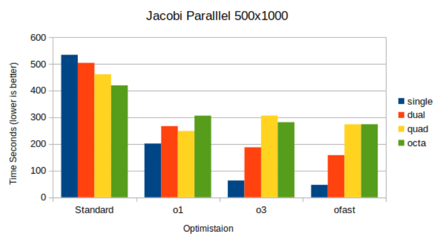

Figure 18: Jacobi Parallel 500x1000

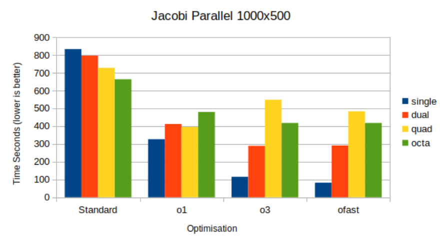

Figure 19: Jacobi Parallel 1000x500

Best Jacobi Parallel Performance			
Problem Size	Optimisation	Cores	Time
100x100	Ofast	Single Core	0.346693
700x700	Ofast	Single Core	80.734946
500x1000	Ofast	Single Core	82.035211
1000x500	Ofast	Single Core	46.70290

Table 14: Best Gauss Parallel Performance

Although while testing using more cores and optimisations resolved the problem spaces correctly, these observations show that more thought needs to be put into how and where parallel code is implemented. As a result of poor performance when using more that a single core has not been achieved with Table 14 confirming the observations that single core had out performed multiple cores.

4.2 Gauss-Seidel Results

The observations gathered, with all the results tabulated in Appendix B.2, show that with no command line optimisation, performance increase in almost linear with eight core workloads performing better than single core work loads as expected. However Fig 20 shows that when a small problem space uses more than two cores, the overheads of spawning multiple threads gives a performance decrease across all optimisations used.

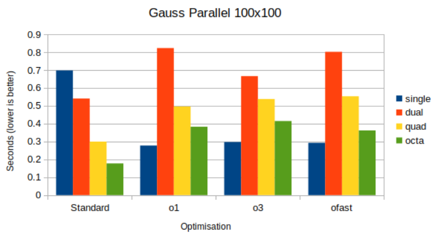

Figure 20: Gauss Parallel 100x100

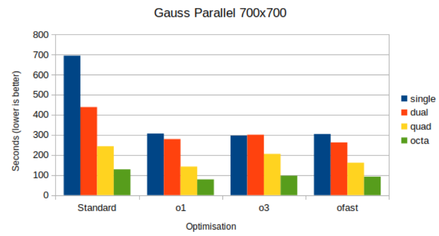

Figure 21: Gauss Parallel 700x700

With a larger problem like Fig 21 we observe a larger performance increase using four and eight cores, there is still little increase in computation performance using single or two cores, except using ofast optimisation where there is a increase in performance for every extra core used. This behavior is repeated again in Figures 22 and 23, except with worse dual core performance using O1 optimisations for both problem sizes with an outlier with the dual core ofast optimisation in the 500x1000 problem size..

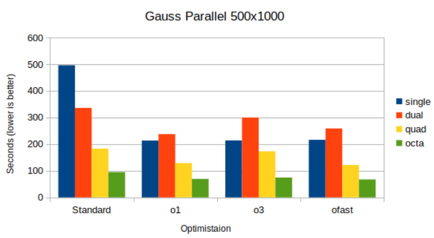

Figure 22: Gauss Parallel 500x1000

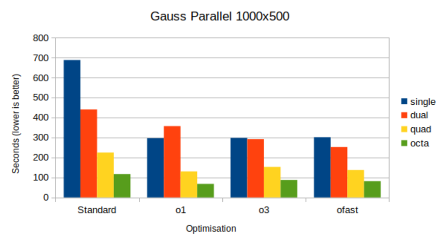

Figure 23: Gauss Parallel 1000x500

It is clear from these observations that using eight cores provides a large performance increase providing the problem size is bigger than a 100x100 matrix, using a single core approach to smaller problem sizes would give better performance as seen in Fig 20.

Best Gauss Parallel Performance

Problem Size	Optimisation	Cores	Time
100x100	Standard	8	0.177528
700x700	O1	8	78.374961
500x1000	O1	8	67.671063
1000x500	Ofast	8	67.477469

Table 15: Best Gauss Parallel Performance

Table 15 shows the best performing combinations for each of the problem sizes that have been tested. The most intriguing is the performance for both 500x1000 and 1000x500 finishing within 0.193594 seconds of each other but using different optimisations. This observation shows that optimisations are not a one-size fits all answer, but needs to be carefully considered when selecting any optimisation.

4.3 Conclusion

Although parallel code can have a big performance increase when used properly, the observations taken here for Jacobi parallel code shows that the opposite and what can happen when parallel code is not implemented properly. Decreasing rather than improving performance. With lessons learnt from these these tests, optimising the code to run faster is the next goal.

5 Task Four

With basic a parallel framework tested and completed in task three, further optimisations can be made to further improve performance. As the goal of this task is to find performance gains and these gains are found by tuning the parallel directives, no single core performance results have been noted.

5.1 Jacobi Parallel Optimised Results

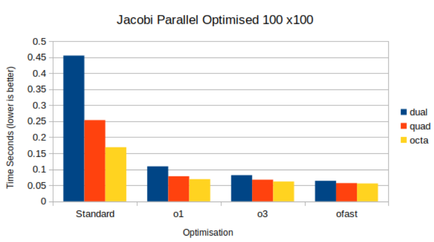

Figure 24: Jacobi Parallel Optimised 100x100

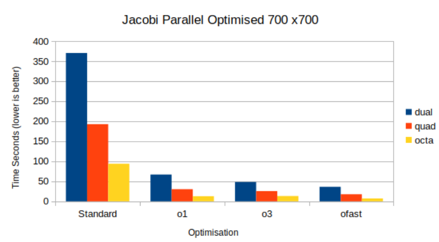

Figure 25: Jacobi Parallel Optimised 700x700

Optimising the parallel code to gain greater performance has been achieved by adding parallel dynamic for loops to code that initialises temperature arrays and loops that fix the boundary conditions. As the for loop that is used for the temperature array is a nested for loop, the collapse directive has been applied which removes the nesting and speeds up computation time. Changes to the main code block that computes differences has also been made, with a nowait directive being applied to a for loop that updates the temperature for the next iteration. This is able to be done as there are no other loops that require variables from this nested loop. If there were variables then it would be an illegal use of the nowait directive. Where the code works out the difference between new and old temperatures, a reduction directive has been applied with the max and difmax used as variables. Replacing the critical directive from task three.

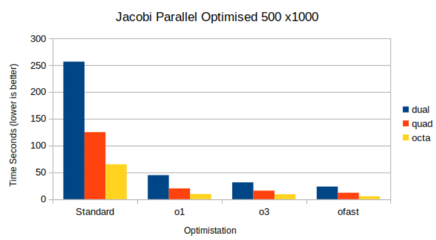

Figure 26: Jacobi Parallel Optimised 500x1000

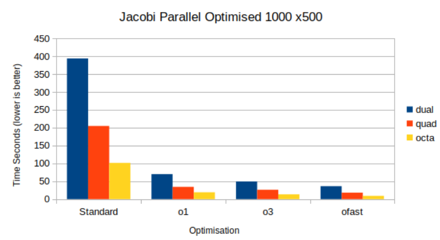

Figure 27: Jacobi Parallel Optimised 1000x500

All these changes have worked and observations show large gains in performance from task three, and as expected more cores providing better performance for every problem size, no as before where more processing cores detracted from performance, with all the results tabulated in Appendix C.1.

Best Jacobi Parallel Performance			
Problem Size	Optimisation	Cores	Time
100x100	Ofast	Eight Core	0.055803
700x700	Ofast	Eight Core	7.137993
500x1000	Ofast	Eight Core	9.197244
1000x500	Ofast	Eight Core	5.31049

Table 16: Best Jacobi Parallel Performance

From Table 16 the observations show that the improvements have been extremely successful when compared to the same results seen in Table 14. All performance points to using eight cores and Ofast optimisation with properly implemented parallel code that adds to the performance rather than detracting from it.

5.2 Gauss Parallel Optimised Results

6 Conclusion

Learning how to apply parallel programming techniques has been very interesting and eye opening, task three was a failure in regards's to performance gains with Jacobi code. Task four however showed how much performance improvement can be gained from correct application of parallel directives. Interestingly

Jacobi would solve the problem 1000x500 faster than 500x1000 even though the volume of these matrices are the same, so dimensional factors also have impacts on the speed at which Jacobi solves for any given tolerance value.

List of Figures

List of Tables

A Appendix A

A.1 Jacobi Serial Results

M = 100 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
826093	207325	394247	210712	225482	241421
820486	208854	408388	212367	226027	239952
814984	212318	401503	203528	225170	244099
815634	210051	393723	201373	228484	231145
819074	209593	381361	209274	226093	240885
Mean result					
819254.2	209628.2	395844.4	207450.8	226251.2	239500.4
Iterations	**12540**				

Table 17: Jacobi M = 100 || N =100 || Tol = 0.0001

M = 200 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
2621756	636738	1259333	635912	697020	735581
2597794	635670	1234181	637268	712622	715177
2621033	618640	1250129	637197	690747	747710
2534388	658602	1233198	644777	703109	743388
2555301	616223	1243033	641279	692279	747750
Mean result					
2586054.4	633174.6	1243974.8	639286.6	699155.4	737921.2
Iterations	**19670**				

Table 18: Jacobi M = 200 || N =100 || Tol = 0.0001

M = 300 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
4366501	1058347	2088637	1081268	1086462	1232413
4384317	1058888	2056661	1096033	1083002	1233484
4267628	1068761	2091461	1089151	1084539	1194669
4268323	1081560	2074924	1107558	1143209	1254543
4511162	1064313	2091761	1114043	1088776	1244122
Mean result					
4359586.2	1066373.8	2080688.8	1097610.6	1097197.6	1231846.2
Iterations	**22052**				

Table 19: Jacobi M = 300 || N =100 || Tol = 0.0001

M = 400 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
5697137	1450991	2768830	1520582	1494374	1720483
5774105	1445855	2796545	1527111	1488931	1729731
5723729	1441108	2794008	1496439	1500634	2439983
5760463	1445044	2772695	1506913	1495046	1732337
5901870	1449716	2798236	1499229	1514439	2434586
Mean result					
5771460.8	1446542.8	2786062.8	1510054.8	1498684.8	2011424
Iterations	**22970**				

Table 20: Jacobi M = 400 || N =100 || Tol = 0.0001

M = 500 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
7600253	1872971	3593376	1889643	1864400	2145590
7726563	1869570	3609815	1895762	1861228	2176343
7687483	1871838	3726003	1882056	1855699	2142917
7571725	1880403	3664621	1887892	1843343	2184027
7705863	1903767	3715857	1881332	1838952	2221665
Mean result					
7658377.4	1879709.8	3661934.4	1887337	1852724.4	2174108.4
Iterations	**23321**				

Table 21: Jacobi M = 500 || N =100 || Tol = 0.0001

M = 100 || N =200 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
2121363	523824	1062477	522265	556561	604829
2094582	527219	1021331	528389	571559	614175
2086992	527628	1033043	532139	569406	612246
2094800	527508	1063528	534116	567602	607369
2079453	537055	1039071	532093	570183	605953
Mean result					
2095438	528646.8	1043890	529800.4	567062.2	608914.4
Iterations	**16600**				

Table 22: Jacobi M = 100 || N =200 || Tol = 0.0001

M = 100 || N =300 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
3159402	776667	1547276	800493	817425	956668
3163379	775181	1542935	819720	780406	910912
3160124	776266	1547880	798987	824672	913665
3249236	771310	1529739	832368	821821	1298843
3364008	782934	1539722	856538	785998	910028
Mean result					
3219229.8	776471.6	1541510.4	821621.2	806064.4	998023.2
Iterations	**17018**				

Table 23: Jacobi M = 100 || N =300 || Tol = 0.0001

M = 100 || N =400 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
4237430	1000646	2106444	1044676	1115003	1726227
4125950	1018748	2095510	1012741	1073615	1719219
4121529	1016107	2090856	1011032	1062215	1209951
4200719	1042031	2080495	1039116	1050595	1724376
4222783	1029756	2020417	1062517	1086587	1183623
Mean result					
4181682.2	1021457.6	2078744.4	1034016.4	1077603	1512679.2
Iterations	**16730**				

Table 24: Jacobi M = 100 || N =400 || Tol = 0.0001

M = 100 || N =500 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
5138252	1236532	2567564	1252384	1304555	2206573
5160614	1237414	2491772	1248577	1305807	1518787
5286721	1252208	2478818	1248513	1305691	2208375
5283796	1235024	2482171	1264138	1315542	2203181
5161282	1240371	2494233	1314693	1316410	1522644
Mean result					
5206133	1240309.8	2502911.6	1265661	1309601	1931912
Iterations	**16672**				

Table 25: Jacobi M = 100 || N =500 || Tol = 0.0001

M = 200 || N =200 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
10049556	2396690	4619200	2373661	2342141	2704349
9910814	2397231	4607105	2345611	2330707	2697646
9856116	2394679	4622672	2338669	2328095	2711627
9693862	2399462	4652025	2384469	2356404	2715632
9757650	2442994	4639688	2348144	2456412	2793123
Mean result					
9853599.6	2406211.2	4628138	2358110.8	2362751.8	2724475.4
Iterations	**38400**				

Table 26: Jacobi M = 200 || N =200 || Tol = 0.0001

M = 300 || N =300 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
40735672	10108995	20169914	9935731	10148790	11904190
41289088	9951105	19890504	10369047	10207903	11640327
41721226	9998323	20143617	10345330	10028353	11680066
41578858	10038582	19805173	10180614	10226355	11377403
41730969	10169478	20107679	10271693	9998664	11839735
Mean result					
41411162.6	10053296.6	20023377.4	10220483	10122013	11688344.2
Iterations	**71288**				

Table 27: Jacobi M = 300 || N =300 || Tol = 0.0001

M = 400 || N =400 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
104433170	26024328	51887660	27250393	27215814	44158192
108541458	25786485	52172969	26655594	27195326	44036898
108925027	26711722	52058432	26317331	27291623	45230602
107474978	26153751	52169376	26310302	27103593	45311267
108121775	26105645	51239332	27608759	27054908	46375921
Mean result					
107499281.6	26156386.2	51905553.8	26828475.8	27172252.8	45022576
Iterations	**107831**				

Table 28: Jacobi M = 400 || N =400 || Tol = 0.0001

M = 500 || N =500 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
223655242	56450978	112045567	57207824	56998578	95198696
224723719	57431059	110884875	57163665	54538751	95319167
225058815	57445539	112034199	57242523	56646724	95848609
234242216	56071786	111677503	57201112	57247771	65888922
229998432	56578950	112915326	55736847	54485744	65740072
Mean result					
227535684.8	56795662.4	111911494	56910394.2	55983513.6	83599093.2
Iterations	**145669**				

Table 29: Jacobi M = 500 || N =500 || Tol = 0.0001

A.2 Gauss Serial Results

M = 100 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
561228	261789	288018	273493	276236	276798
563807	257647	273916	274816	274679	275121
560688	260950	273424	275316	278758	278231
563054	261332	272818	274640	273640	277801
565024	261448	273886	276979	277039	277256
Mean result					
562760.2	260633.2	276412.4	275048.8	276070.4	277041.4
Iterations	**6994**				

Table 30: Gauss M = 100 || N =100 || Tol = 0.0001

M = 200 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
1774024	813331	841897	856146	848167	857523
1769470	845362	841398	856100	855554	855864
1755774	812495	849712	856367	848344	856564
1761828	811183	848637	855878	847235	857289
1761945	811596	850116	856389	847593	857577
Mean result					
1764608.2	818793.4	846352	856176	849378.6	856963.4
Iterations	**10986**				

Table 31: Gauss M = 200 || N =100 || Tol = 0.0001

M = 300 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
2973592	1358547	1424668	1479738	1432510	1431211
2977345	1357352	1423476	1477347	1494423	1425500
2967328	1357925	1425582	1477917	1480187	1422463
3089290	1357008	1426777	1475785	1477339	1437883
2990260	1357314	1432111	1478071	1489751	1438799
Mean result					
2999563	1357629.2	1426522.8	1477771.6	1474842	1431171.2
Iterations	**12323**				

Table 32: Gauss M = 300 || N =100 || Tol = 0.0001

M = 400 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
4246518	1884116	1959453	1973905	1985819	1991183
4264374	1887660	1983122	1993218	1984340	1994161
4163640	1886994	1983579	1974029	1987717	1995604
4147721	1886631	1980862	1972856	1985723	1970825
4119650	1865812	1964612	1990396	1984297	2001904
Mean result					
4188380.6	1882242.6	1974325.6	1980880.8	1985579.2	1990735.4
Iterations	**12851**				

Table 33: Gauss M = 400 || N =100 || Tol = 0.0001

M = 500 || N =100 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
5192529	2532360	2495656	2605996	2530493	2531376
5232481	2603185	2481504	2620262	2535414	2535077
5207563	2531555	2487654	2603161	2535109	2515881
5231037	2510065	2494516	2602933	2507445	2532561
5269842	2520023	2497137	2612093	2537450	2562603
Mean result					
5226690.4	2539437.6	2491293.4	2608889	2529182.2	2535499.6
Iterations	**13065**				

Table 34: Gauss M = 500 || N =100 || Tol = 0.0001

M = 100 || N =200 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
1510528	704146	738696	749529	750614	750543
1511094	710542	748114	745975	750327	753645
1503271	703731	745476	744587	752640	751259
1506063	703262	736694	746100	746244	750013
1503337	714530	748765	753591	754003	752340
Mean result					
1506858.6	707242.2	743549	747956.4	750765.6	751560
Iterations	**9460**				

Table 35: Gauss M = 100 || N =200 || Tol = 0.0001

M = 100 || N =300 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
2333228	1112211	1154735	1217549	1179849	1176289
2332009	1103575	1168817	1225327	1179621	1175887
2349147	1116850	1214402	1223662	1181154	1175312
2347760	1114770	1165071	1230111	1180359	1176646
2356910	1116473	1172090	1233781	1230376	1179306
Mean result					
2343810.8	1112775.8	1175023	1226086	1190271.8	1176688
Iterations	**9845**				

Table 36: Gauss M = 100 || N =300 || Tol = 0.0001

M = 100 || N =400 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
3079778	1465834	1546841	1623137	1552217	1560651
3081901	1481344	1552792	1622471	1568509	1560263
3080143	1481713	1530762	1553551	1566469	1562147
3080505	1464019	1549433	1552552	1552410	1560084
3115567	1464315	1533629	1554601	1575433	1565907
Mean result					
3087578.8	1471445	1542691.4	1581262.4	1563007.6	1561810.4
Iterations	**9775**				

Table 37: Gauss M = 100 || N =400 || Tol = 0.0001

M = 100 || N =500 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
3849834	1846938	1905107	2029207	2027464	2017898
3856184	1849469	1927193	1919547	1931386	1919421
3857682	1849617	1926856	1919941	1930317	1920229
3861769	1847990	1902057	1925205	1928689	1920102
3840324	1855833	1900922	2028174	2007041	2025043
Mean result					
3853158.6	1849969.4	1912427	1964414.8	1964979.4	1960538.6
Iterations	**9719**				

Table 38: Gauss M = 100 || N =500 || Tol = 0.0001

M = 200 || N =200 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
6974249	3250759	3569124	3630824	3630408	3617414
7075407	3249712	3565130	3438952	3456386	3432201
7001908	3249284	3568236	3628959	3615554	3602179
7132964	3258207	3570304	3628120	3507501	3622853
7356369	3434384	3572467	3505039	3663024	3469918
Mean result					
7108179.4	3288469.2	3569052.2	3566378.8	3574574.6	3548913
Iterations	**22052**				

Table 39: Gauss M = 200 || N =200 || Tol = 0.0001

M = 300 || N =300 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
29888628	14268278	14911054	15115857	15107880	14913497
30153200	14138596	14906833	14959775	14944075	15012529
31533016	14295259	15151523	15126901	15127346	15030082
30557075	14740955	14963390	15148738	15135385	14993605
31371855	14297699	14931262	14953308	14924306	15103730
Mean result					
30700754.8	14348157.4	14972812.4	15060915.8	15047798.4	15010688.6
Iterations	**42029**				

Table 40: Gauss M = 300 || N =300 || Tol = 0.0001

M = 400 || N =400 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
83004920	39593677	40853504	41828301	41621830	41599768
82466470	39140087	41259778	41319492	41740607	41379045
83509203	39496494	41294847	41885353	41316634	41607545
82914167	39485615	40976787	41844714	42110380	41703435
82763433	39083928	41284780	41300175	41838936	41411739
Mean result					
82931638.6	39359960.2	41133939.2	41635607	41725677.4	41540306.4
Iterations	**65236**				

Table 41: Gauss M = 400 || N =400 || Tol = 0.0001

M = 500 || N =500 || Tol = 0.0001 | Time in microseconds					
Standard	Fast	O1	O2	O3	Os
181013605	85532692	89587633	93146871	90145975	89182028
178527060	88065418	89550250	89596940	89613585	90073116
180750807	84598824	88576018	89979738	90292201	90113731
179648681	84688902	89479791	91270011	90300912	89327411
180727061	89457384	89500727	90574138	89542365	90222199
Mean result					
180133442.8	86468644	89338883.8	90913539.6	89979007.6	89783697
Iterations	**90497**				

Table 42: Gauss M = 500 || N =500 || Tol = 0.0001

B Appendix B

B.1 Jacobi Parallel Results

B.1.1 Standard Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.700061	0.543066	0.301108	0.184552
0.696721	0.53811	0.297876	0.173636
0.697031	0.5408	0.298714	0.174397
Mean result			
0.697938	0.540659	0.299233	0.177528

Table 43: Jacobi Parallel Standard M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
792.72648	763.777951	688.283799	639.235987
813.708995	776.737934	708.897295	635.64298
833.972559	796.335182	670.117602	637.366962
Mean result			
813.469345	778.950356	689.099566	637.415310

Table 44: Jacobi Parallel Standard M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
830.071947	788.830731	722.185271	664.603937
850.329391	779.868192	742.309751	663.681091
818.48831	818.449524	717.247159	661.93748
Mean result			
832.963216	795.716149	727.247394	663.407503

Table 45: Jacobi Parallel Standard M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
549.088679	510.29925	471.754073	419.873201
525.712874	500.461054	465.392431	422.046098
526.837045	500.370019	445.206194	416.592525
Mean result			
533.879533	503.710108	460.784233	419.503941

Table 46: Jacobi Parallel Standard M = 500 || N =1000 || Tol = 0.0001

B.1.2 O1 Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.34918	0.481635	0.621733	0.844489
0.3443	0.511487	0.608845	0.857409
0.346599	0.517156	0.597323	0.807652
Mean result			
0.346693	0.503426	0.609300	0.836517

Table 47: Jacobi Parallel O1 M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
308.529549	252.247264	376.869001	469.021656
312.191529	394.927425	376.62571	462.815007
311.574968	402.81792	373.55808	464.901131
Mean result			
310.765349	349.997536	375.684264	465.579265

Table 48: Jacobi Parallel O1 M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
325.801566	408.215974	397.306876	471.553867
326.957171	413.26774	394.317444	484.491763
326.352219	414.546087	393.615828	482.960133
Mean result			
326.370319	412.009934	395.080049	479.668588

Table 49: Jacobi Parallel O1 M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
201.348227	282.347914	250.995169	305.324853
201.595241	251.38185	245.377747	306.747683
201.673013	266.066459	249.001251	306.163881
Mean result			
201.538827	266.598741	248.458056	306.078806

Table 50: Jacobi Parallel O1 M = 500 || N =1000 || Tol = 0.0001

B.1.3 O3 Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.125803	0.32297	0.816756	1.619051
0.120764	0.34237	0.882937	1.939399
0.123229	0.345274	0.783963	1.772101
Mean result			
0.12326	0.336871	0.827885	1.776850

Table 51: Jacobi Parallel O3 M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
108.007484	256.931666	519.300555	395.459747
113.566633	257.040399	518.059874	399.186685
114.141428	256.04902	516.593992	416.095083
Mean result			
111.905182	256.673695	517.984807	403.580505

Table 52: Jacobi Parallel O3 M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
115.728734	274.04147	548.01037	419.410935
119.326043	314.199428	556.835133	427.116057
110.742762	279.07467	540.11755	406.568234
Mean result			
115.265846	289.105189	548.321018	417.698409

Table 53: Jacobi Parallel O3 M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
62.969391	221.748811	347.446241	273.132634
62.327769	170.307993	337.126292	282.092528
63.409943	170.300713	233.833492	288.976119
Mean result			
62.902368	187.452506	306.135342	281.400427

Table 54: Jacobi Parallel O3 M = 500 || N =1000 || Tol = 0.0001

B.1.4 Ofast Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.091702	0.280337	0.920983	1.849277
0.086687	0.289223	0.804955	1.770892
0.090118	0.273997	0.952211	1.869249
Mean result			
0.089502	0.281186	0.892716	1.829806

Table 55: Jacobi Parallel Ofast M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
79.962215	297.325261	453.480819	402.795765
77.376488	294.899356	422.449726	406.462035
84.866135	281.352534	361.364109	393.027869
Mean result			
80.734946	291.192384	412.431551	400.76189

Table 56: Jacobi Parallel Ofast M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
83.608515	294.791963	501.931055	411.726568
81.199795	289.392198	472.892462	423.680979
81.297323	288.675483	474.982015	418.000207
Mean result			
82.035211	290.953215	483.268511	417.802585

Table 57: Jacobi Parallel Ofast M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
46.281273	172.273596	250.801621	277.722833
46.695678	127.666026	245.394251	265.852967
47.132018	174.106086	324.670584	278.007124
Mean result			
46.70299	158.015236	273.622152	273.860975

Table 58: Jacobi Parallel Ofast M = 500 || N =1000 || Tol = 0.0001

B.2 Gauss Parallel Results

B.2.1 Standard Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.700061	0.543066	0.301108	0.184552
0.696721	0.53811	0.297876	0.173636
0.697031	0.5408	0.298714	0.174397
Mean result			
0.697938	0.540659	0.299233	0.177528

Table 59: Gauss Parallel Standard M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
693.578522	435.884029	234.985973	133.994749
692.201347	437.075325	272.720373	127.573709
695.809181	441.893981	221.389599	122.574146
Mean result			
693.863017	438.284445	243.03198	128.047535

Table 60: Gauss Parallel Standard M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
686.375689	432.384969	222.239076	117.136736
684.978019	440.644899	229.376448	114.559556
691.075175	446.629971	221.450104	119.088092
Mean result			
687.476294	439.886613	224.355209	116.928128

Table 61: Gauss Parallel Standard M = 1000 || N =500 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
495.60809	326.972278	184.447611	96.177068
495.765491	335.997232	186.596061	96.177068
496.813669	345.100622	177.992395	92.594865
Mean result			
496.062417	336.023377	183.012022	94.983000

Table 62: Gauss Parallel Standard M = 500 || N =1000 || Tol = 0.0001

B.2.2 O1 Optimisation

M = 100 || N =100|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.282733	0.81712	0.495706	0.355325
0.274772	0.839539	0.496016	0.401246
0.274722	0.812391	0.496499	0.391074
Mean result			
0.277409	0.823017	0.496074	0.382548

Table 63: Gauss Parallel O1 M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
306.285933	283.803914	137.865674	84.236206
307.318142	300.739678	135.354263	91.549175
305.806419	252.156396	153.431253	89.227862
Mean result			
306.470165	278.899996	142.217063	88.337748

Table 64: Gauss Parallel O1 M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
295.986061	366.12512	131.392679	78.374961
296.187046	365.421404	129.082271	80.887983
296.365166	339.120497	130.640753	80.135171
Mean result			
296.179424	356.889007	130.371901	79.799372

Table 65: Gauss Parallel O1 M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
214.869212	237.418585	126.844778	71.262344
214.685202	237.659427	128.602206	67.671063
210.462551	237.291975	131.290287	68.866421
Mean result			
213.338988	237.456662	128.912424	69.266609

Table 66: Gauss Parallel O1 M = 500 || N =1000 || Tol = 0.0001

B.2.3 O3 Optimisation

M = 100 || N =100|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.296002	0.806658	0.519743	0.410733
0.298082	0.572638	0.536195	0.427121
0.297202	0.617644	0.55779	0.405588
Mean result			
0.297095	0.665647	0.537909	0.414481

Table 67: Gauss Parallel O3 M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
295.990964	248.503026	203.375436	96.280448
295.948017	309.093005	206.19109	97.599407
296.026867	344.080976	205.397095	96.26928
Mean result			
295.988616	300.559002	204.987874	96.716378

Table 68: Gauss Parallel O3 M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
297.94658	275.46004	155.678218	86.607405
297.915732	297.014345	149.368859	87.988035
297.551637	301.450785	152.33992	88.016546
Mean result			
297.80465	291.30839	152.462332	87.537329

Table 69: Gauss Parallel O3 M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
214.112967	331.886181	181.25442	73.825091
213.775623	298.34009	186.211446	75.10757
213.72886	269.533614	152.121926	75.155497
Mean result			
213.872483	299.919962	173.195931	74.696053

Table 70: Gauss Parallel O3 M = 500 || N =1000 || Tol = 0.0001

B.2.4 Ofast Optimisation

M = 100 || N =100|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
0.29543	0.801421	0.516352	0.367821
0.289967	0.804971	0.560008	0.284771
0.292115	0.799101	0.583325	0.43366
Mean result			
0.292504	0.801831	0.553228	0.362084

Table 71: Gauss Parallel Ofast M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
303.569703	240.274137	162.417238	92.938334
303.571103	264.215859	161.47041	90.793343
304.30212	281.767329	160.07522	91.32157
Mean result			
303.814309	262.085775	161.320956	91.684416

Table 72: Gauss Parallel Ofast M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
301.800979	222.521478	134.885286	81.813777
301.516988	264.625938	135.613541	80.280035
301.381606	268.419846	140.171734	82.200118
Mean result			
301.566524	251.855754	136.890187	81.43131

Table 73: Gauss Parallel Ofast M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000|| Tol = 0.0001 | Seconds			
Single	Dual	Quad	Octa
216.165618	231.067038	124.44607	69.607918
216.061475	328.91237	119.479609	66.163611
216.34457	216.054534	120.947952	66.660878
Mean result			
216.190554	258.677981	121.624544	67.477469

Table 74: Gauss Parallel Ofast M = 500 || N =1000 || Tol = 0.0001

C Appendix C

C.1 Jacobi Parallel Results

C.1.1 Standard Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
0.454909	0.254656	0.169365
0.454381	0.252672	0.169201
0.454997	0.253356	0.167974
Mean result		
0.454762	0.253561	0.168847

Table 75: Jacobi Parallel Optimised Standard M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
364.306007	191.521926	87.358815
373.958938	193.98375	94.144199
372.586544	192.658496	99.63786
Mean result		
370.28383	192.721391	93.713625

Table 76: Jacobi Parallel Optimised Standard M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
392.699681	203.470706	103.302508
395.220956	206.192493	94.047642
392.729389	204.589699	106.580168
Mean result		
393.550009	204.750966	101.310106

Table 77: Jacobi Parallel Optimised Standard M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
259.888116	125.963835	61.317549
255.314594	125.12729	66.625703
254.114873	123.569967	66.804673
Mean result		
256.439194	124.887031	64.915975

Table 78: Jacobi Parallel Optimised Standard M = 500 || N =1000 || Tol = 0.0001

C.1.2 O1 Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
0.11932	0.077201	0.068282
0.102404	0.079117	0.071598
0.105572	0.078982	0.067934
Mean result		
0.109099	0.078433	0.069271

Table 79: Jacobi Parallel Optimised O1 M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
66.711758	30.22612	12.329131
67.039507	30.796762	14.334538
66.832271	29.819945	11.779372
Mean result		
66.861179	30.280942	12.814347

Table 80: Jacobi Parallel Optimised O1 M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
69.833147	33.709566	18.833587
69.924769	35.588938	19.634445
70.430594	33.654949	18.926808
Mean result		
70.062837	34.317818	19.131613

Table 81: Jacobi Parallel Optimised O1 M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
44.787274	19.494254	7.246144
44.625067	20.530739	10.899818
44.892763	20.071906	10.609342
Mean result		
44.768368	20.03221	9.585101

Table 82: Jacobi Parallel Optimised O1 M = 500 || N =1000 || Tol = 0.0001

C.1.3 O3 Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
0.080004	0.068785	0.062763
0.082585	0.067934	0.060956
0.08234	0.065877	0.061679
Mean result		
0.081643	0.067532	0.061799

Table 83: Jacobi Parallel Optimised O3 M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
49.229271	26.100203	13.256604
47.60652	25.599069	13.524329
48.011862	24.945881	13.039959
Mean result		
48.282551	25.548384	13.273631

Table 84: Jacobi Parallel Optimised O3 M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
49.605841	26.346551	13.86822
49.050415	26.238202	13.783829
49.817442	26.256237	12.655231
Mean result		
49.491233	26.28033	13.43576

Table 85: Jacobi Parallel Optimised O3 M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
31.465431	16.725083	9.047407
31.838585	14.430413	9.361603
30.732019	16.42531	9.258386
Mean result		
31.345345	15.860269	9.222465

Table 86: Jacobi Parallel Optimised O3 M = 500 || N =1000 || Tol = 0.0001

C.1.4 Ofast Optimisation

M = 100 || N =100 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
0.063981	0.057427	0.054711
0.063932	0.057136	0.055073
0.064441	0.056021	0.057625
Mean result		
0.064118	0.056861	0.055803

Table 87: Jacobi Parallel Optimised Ofast M = 100 || N =100 || Tol = 0.0001

M = 700 || N =700 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
36.728068	18.292333	7.123252
35.781395	17.683761	6.742016
36.155252	16.894013	7.548711
Mean result		
36.221572	17.623369	7.137993

Table 88: Jacobi Parallel Optimised Ofast M = 700 || N =700 || Tol = 0.0001

M = 1000 || N =500 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
36.68644	18.616526	10.607768
36.652569	17.826834	8.995443
35.574137	17.964505	7.988521
Mean result		
36.304382	18.135955	9.197244

Table 89: Jacobi Parallel Optimised Ofast M = 1000 || N =500 || Tol = 0.0001

M = 500 || N =1000 || Tol = 0.0001 | Seconds		
Dual	Quad	Octa
23.771607	11.772715	6.98462
23.749009	11.476722	4.908689
23.05738	12.620669	4.038162
Mean result		
23.525999	11.956702	5.310490

Table 90: Jacobi Parallel Optimised Ofast M = 500 || N =1000 || Tol = 0.0001

C.2 Gauss Parallel Results