PAR Laboratory Assignment

Lab 1: Experimental setup and tools

Aleix Boné
Alex Herrero
par2109

2020-03-06

Contents

1 Node architecture and memory

The boada server consists of 8 different nodes with different processors. As we can see in the table 1 there are 3 different architectures with slight variations `boada-1 to boada-4` have the same number of sockets(2), cores(6) and threads per core (2), which adds up to a total of 24 threads.

`boada-5` has the same number of sockets, threads and cores as the previous `boadas` but with higher Maximum core frequency and a much larger shared cache size and Main memory. The output of `lstopo` also shows us that it has 4 `GPUs`.

`boada-6 to boada-8` have 8 cores per socket but only one thread per core (amounting to 16 threads instead of the 24 of the other nodes) and a much lower clock frequency. However it has the highest last-level cache size.

	`boada 1 to 4`	`boada 5`	`boada 6 to 8`
Number of sockets per node	2	2	2
Number of cores per socket	6	6	8
Number of threads per core	2	2	1
Maximum core frequency	2395 MHz	2600 MHz	1700 MHz
L1-I cache size (per-core)	32K	32K	32K
L1-D cache size (per-core)	32K	32K	32K
L2 cache size (per-core)	256K	256K	256K
Last-level cache size (per-socket)	12288K	15360K	20480K
Main memory size (per socket)	12 Gb	31 Gb	16 Gb
Main memory size (per node)	23 Gb	63 Gb	31 Gb

Table 1: Architecture of boada nodes

In figure 1 we can see the data from table 1 corresponding to `boada-1`. The cache L3 is shared between the cores of each socket. while the others are local to each core. There is no shared memory between the sockets.

Figure 1: Architectural diagram for `boada-1`

2 Strong vs. weak scalability

In strong scalability the number of threads is changed with a fixed problem size. In this case parallelism is used to reduce the execution time of the program.

In weak scalability the problem size is proportional to the number of threads. In this case parallelism is used to increase the problem size for which the program is executed.

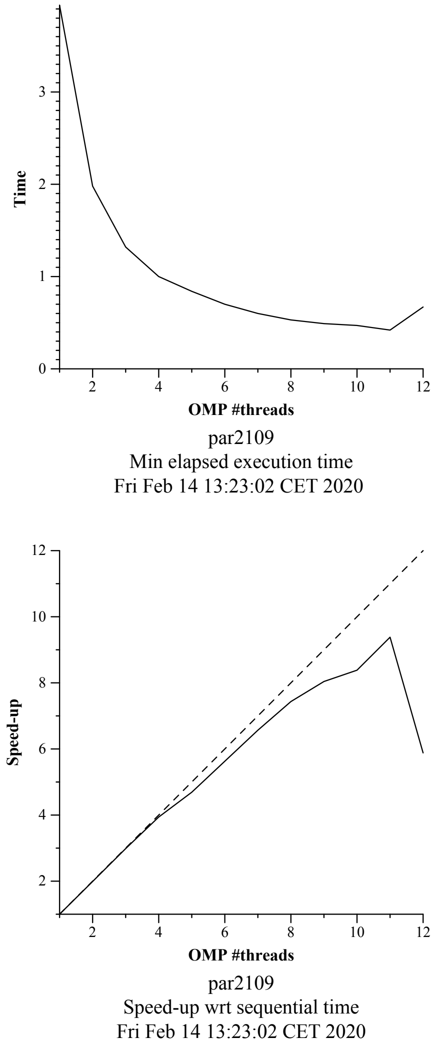


Figure 2: Strong scalability

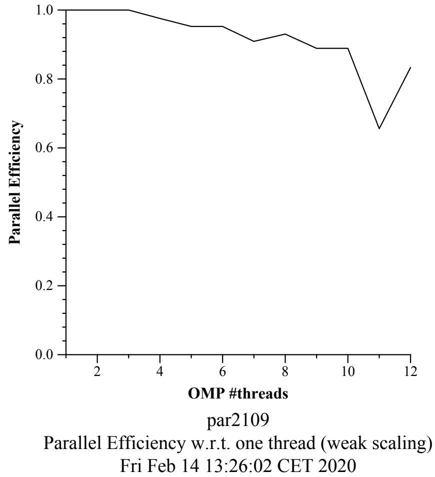


Figure 3: Weak scalability

In strong scalability, as we can observe in figure 2, time decreases as we increase the number of threads and therefore the speedup almost increases linearly. Except in the case we use 12 threads that time and speedup increase because there are threads without doing useful work or the overhead of creating the threads starts to out-weight its benefits.

Similarly, in weak scalability (figure 3) we can see that the parallel efficiency remains close to the ideal vale of 1 slowly decreasing and dips at 11 threads. Although the parallel efficiency increases again at 12 threads, its still significantly lower that the value of 10 threads.

3 Analysis of task decompositions for *3DFFT*

Version	T_1	T_∞	Parallelism
seq	639 780 001	639 780 001	1
v1	639 780 001	639 707 001	1.000 114 115
v2	639 780 001	361 190 001	1.771 311 496
v3	639 780 001	154 438 001	4.142 633 269
v4	639 780 001	64 102 001	9.980 655 69
v5	639 780 001	8 155 001	78.452 473 642

Table 2: Parallelism for different `3dfft` versions

Table 2 shows the different values of parallelism for each modification. We can see that the first changes don't affect much but the more finer ones can *theoretically* gives us really high values of paralellization.

Figures 4 and 5 show the change in execution time with the number of processors. Since its difficult to appreciate the differences with a linear scale, we have included figures 6 and 7 with a logarithmic scale to show how the execution time of *v5* decreases from 16 to 32 processors and *v4* does not, meaning that *v5* can be further parallelized while we have reached a limit with *v4*.

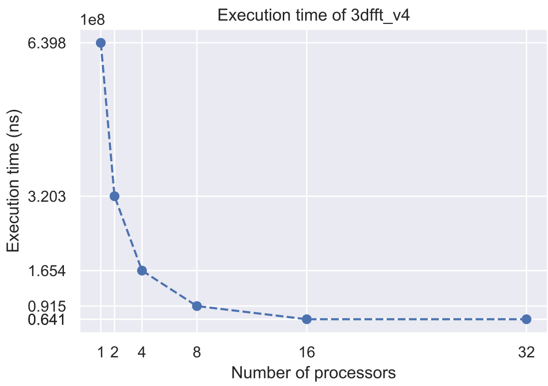


Figure 4: Execution time of v4

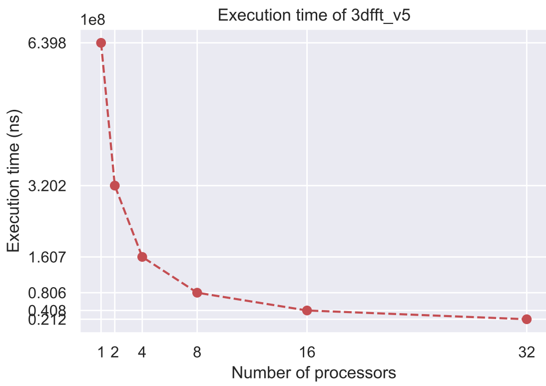


Figure 5: Execution time of v5

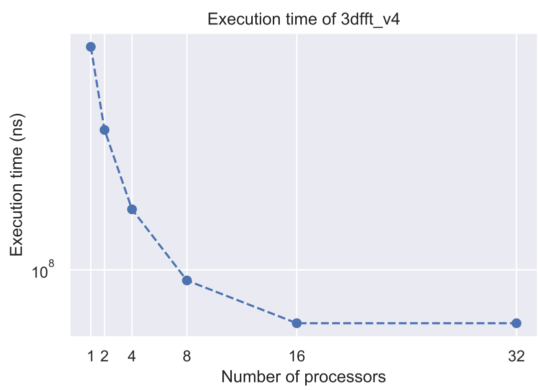


Figure 6: Execution time of v4 (log scale)

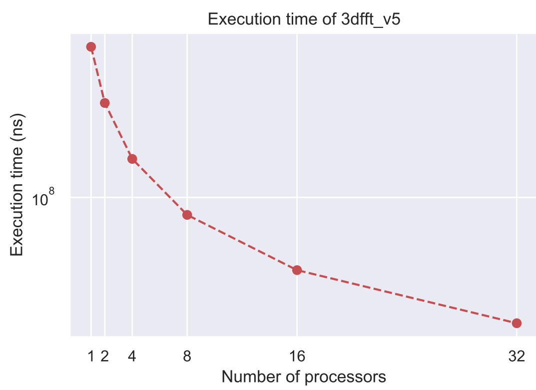


Figure 7: Execution time of v5 (log scale)

The task dependency graphs for *v4* (figure 8) and *v5* (figure 9) illustrate that *v5* has much more fine grained tasks that can be parallelized as opposed to *v4*.

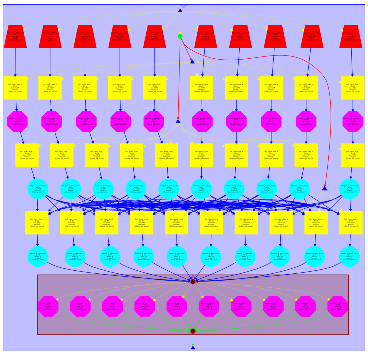


Figure 8: Dependency graph v4

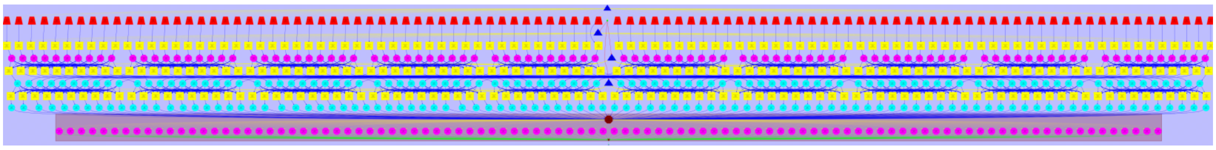


Figure 9: Dependency graph v5

Figure 10 shows a portion of the differences between *v4* and *v5*. As we can see the tasks of *v5* are in a more inner part of the nested for loops (although not the deepest, there is 1 extra level we cold have tried), this means that *v5* can have N times more tasks than *v4* for those methods, allowing for much higher parallelization as we have seen in figures 4-7.

```
@@ -28,8 +28,8 @@ void init_complex_grid(fftwf_complex in_fftw[][N][N]) {
   int k,j,i;

   for (k = 0; k < N; k++) {
-       tareador_start_task("init_complex_grid_loop_k");
     for (j = 0; j < N; j++) {
+      tareador_start_task("init_complex_grid_loop_j");
       for (i = 0; i < N; i++)
       {
         in_fftw[k][j][i][0] = (float) (sin(M_PI*((float)i)/64.0)+sin(M_PI*((float)i)/32.0)+sin(M_PI*((float)i/16.0)));
@@ -39,8 +39,8 @@ void init_complex_grid(fftwf_complex in_fftw[][N][N]) {
         out_fftw[k][j][i][1]= in_fftw[k][j][i][1];
 #endif
       }
+      tareador_end_task("init_complex_grid_loop_j");
     }
-       tareador_end_task("init_complex_grid_loop_k");
   }
 }
```

Figure 10: Differences between v4 and v5

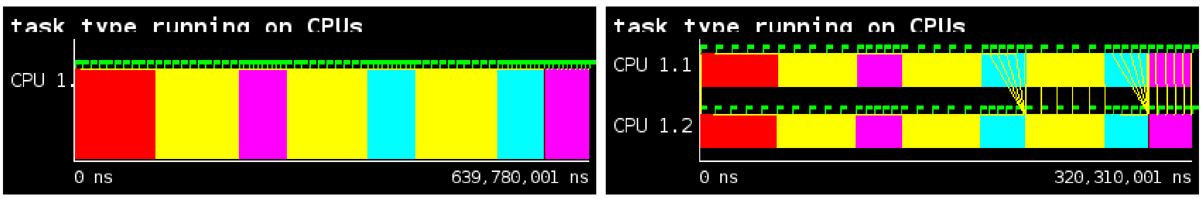


Figure 11: Execution time of v4 with 1 and 2 processors

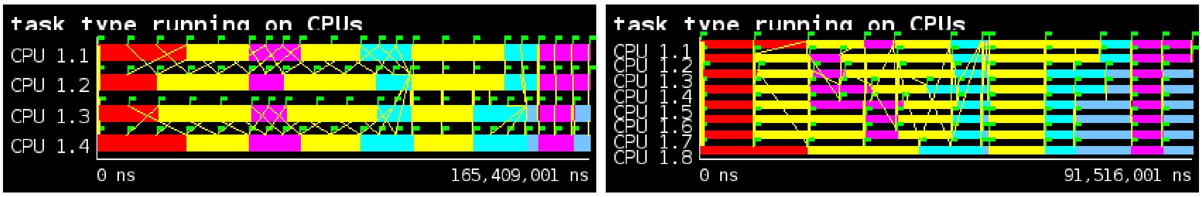


Figure 12: Execution time of v4 with 4 and 8 processors

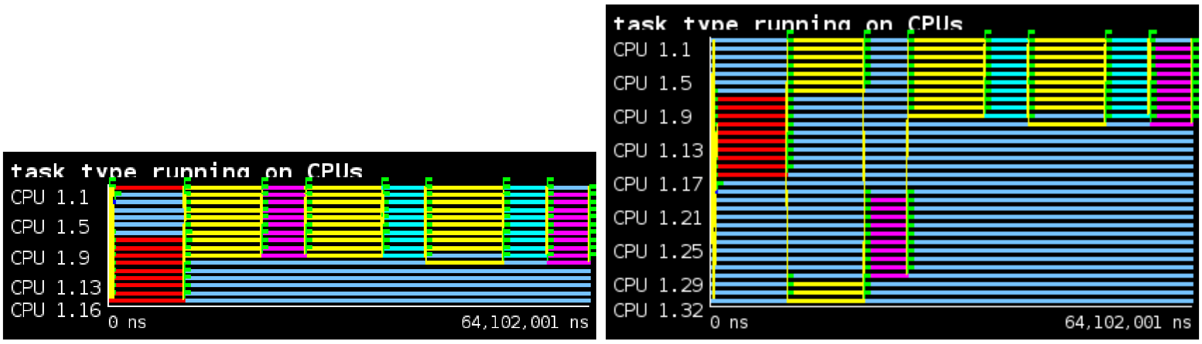


Figure 13: Execution time of v4 with 16 and 32 processors

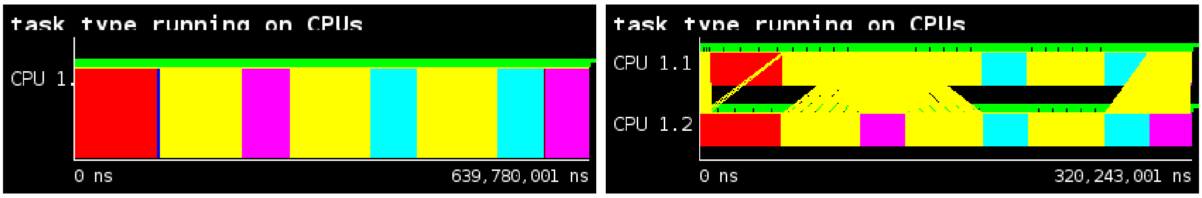


Figure 14: Execution time of v5 with 1 and 2 processors

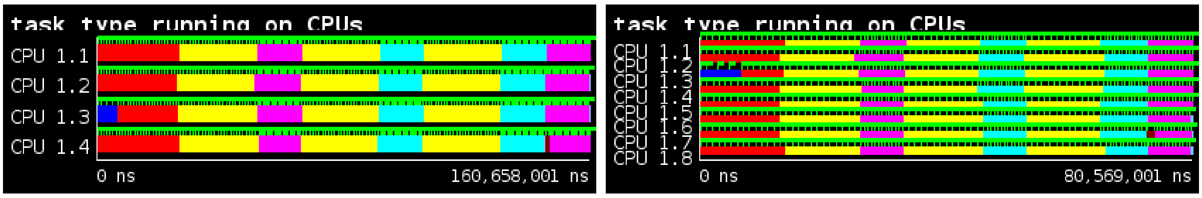


Figure 15: Execution time of v5 with 4 and 8 processors

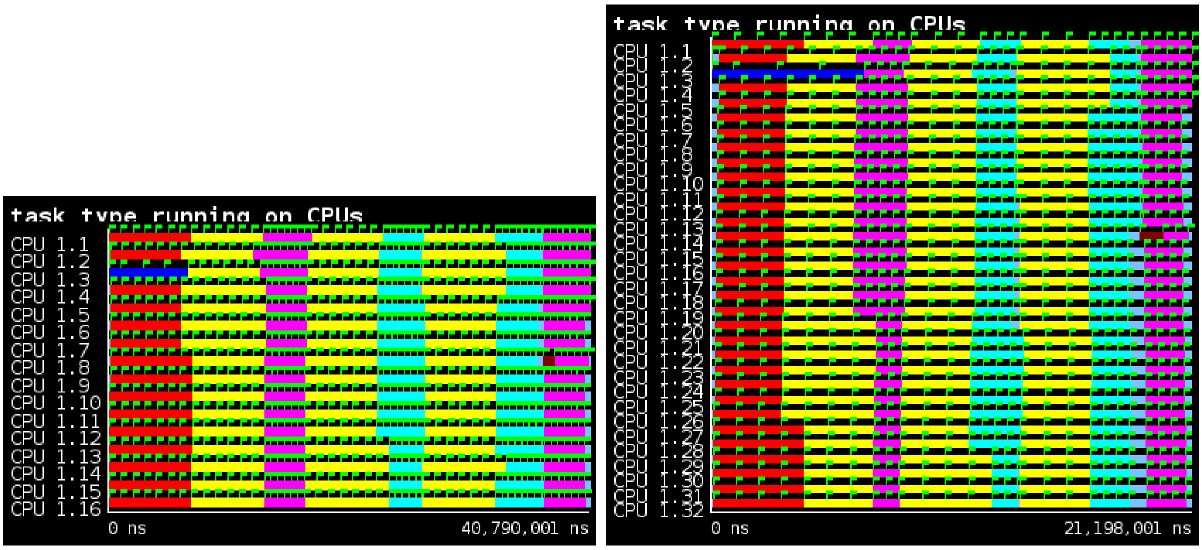
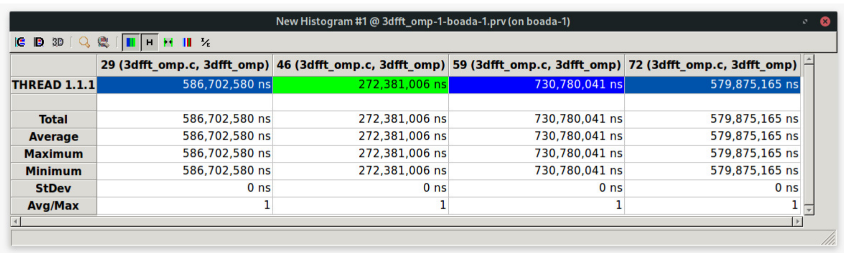


Figure 16: Execution time of v5 with 16 and 32 processors

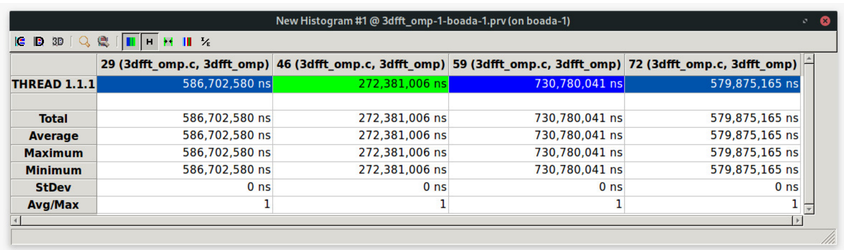
4 Understanding the parallel execution of *3DFFT*

Version	ϕ	S_∞	T_1 (ns)	T_8 (ns)	S_8
initial version in `3dfft_omp.c`	0.619	2.623	2 519 853 052	1 710 690 423	1.473
new version with improved ϕ	0.891	9.160	2 365 124 299	955 055 104	2.474
final version[1]	0.903	10.346	2 401 885 497	626 806 635	3.832


New Histogram #1 @ 3dfft_omp-1-boada-1.prv (on boada-1)

	29 (3dfft_omp.c, 3dfft_omp)	46 (3dfft_omp.c, 3dfft_omp)	59 (3dfft_omp.c, 3dfft_omp)	72 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Total	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Average	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Maximum	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Minimum	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
StDev	0 ns	0 ns	0 ns	0 ns
Avg/Max	1	1	1	1

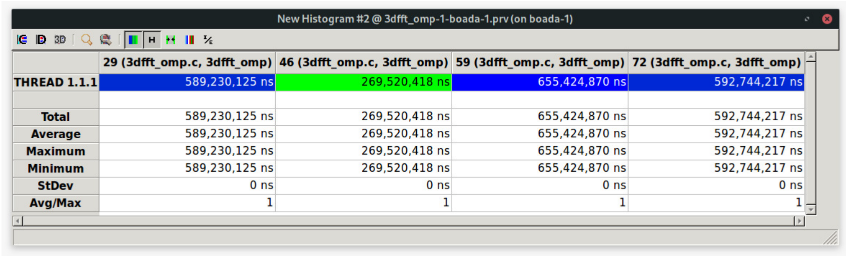
Figure 17: 2DAnalyzer window of the initial version with the parallel functions cfg


New Histogram #1 @ 3dfft_omp-1-boada-1.prv (on boada-1)

	29 (3dfft_omp.c, 3dfft_omp)	46 (3dfft_omp.c, 3dfft_omp)	59 (3dfft_omp.c, 3dfft_omp)	72 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Total	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Average	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Maximum	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
Minimum	586,702,580 ns	272,381,006 ns	730,780,041 ns	579,875,165 ns
StDev	0 ns	0 ns	0 ns	0 ns
Avg/Max	1	1	1	1

Figure 18: 2DAnalyzer window of the improved version with the parallel functions cfg

[1] with reduced parallelization overheads



New Histogram #2 @ 3dfft_omp-1-boada-1.prv (on boada-1)

	29 (3dfft_omp.c, 3dfft_omp)	46 (3dfft_omp.c, 3dfft_omp)	59 (3dfft_omp.c, 3dfft_omp)	72 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	589,230,125 ns	269,520,418 ns	655,424,870 ns	592,744,217 ns
Total	589,230,125 ns	269,520,418 ns	655,424,870 ns	592,744,217 ns
Average	589,230,125 ns	269,520,418 ns	655,424,870 ns	592,744,217 ns
Maximum	589,230,125 ns	269,520,418 ns	655,424,870 ns	592,744,217 ns
Minimum	589,230,125 ns	269,520,418 ns	655,424,870 ns	592,744,217 ns
StDev	0 ns	0 ns	0 ns	0 ns
Avg/Max	1	1	1	1

Figure 19: 2DAnalyzer window of the final version with the parallel functions cfg

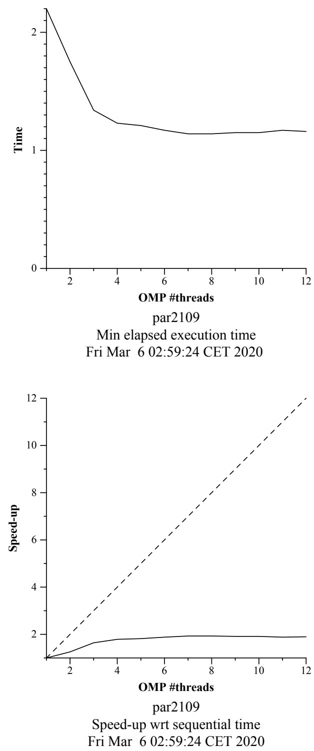


Figure 20: Scalability of the initial version

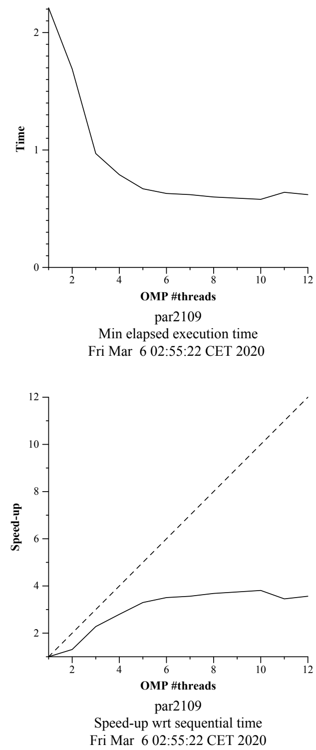


Figure 21: Scalability of the improved version

We can see that for the initial version in figure 20 there is no difference in execution time after 4 threads and the speed-up is really low and far from the ideal value.

The second version (figure 21) has lower execution time than the first one but again, there is no noticeable difference from 6 threads on-wards, in the speed-up plot we can really see an improvement, we can therefore conclude that allowing the parallelization of the function `init_complex_grid` function had a great impact on ϕ.

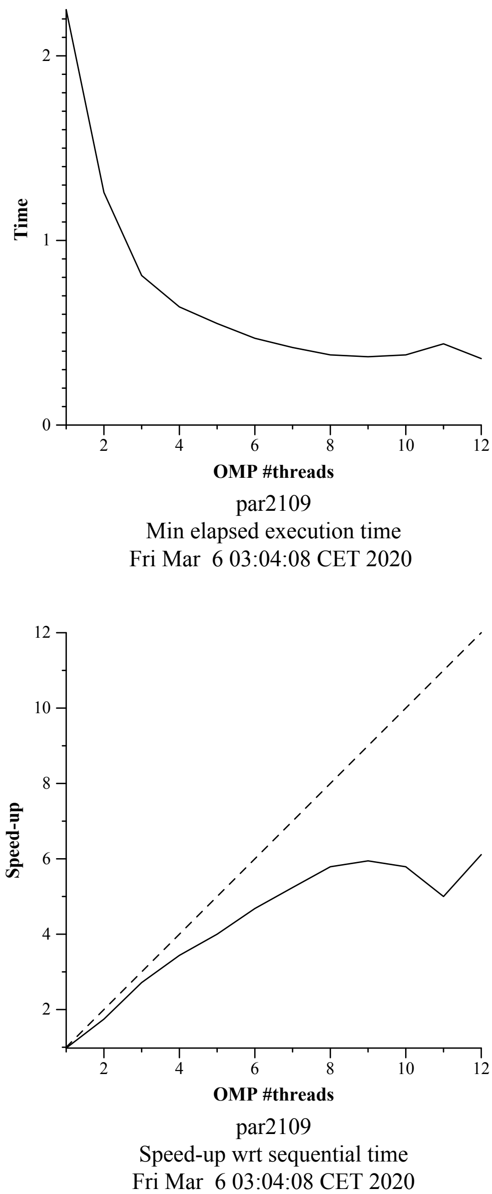


Figure 22: Scalability of the final version

In the final version shown in figure 22 we can see a much better speed-up without much overhead although it starts to fall off at 8 threads with a big dip in 11 threads.

Although in out analysis of the task decomposition for *3DFFT* we concluded that we could theoretically get parallelism values as high as 78 (table 2) the results of figures 20-22 show that in practice, the parallelization overhead is too high and its better to use less fine tasks to reduce the overhead.