PAR Laboratory Assignment

Lab 1:

Experimental setup and tools

Guillem Dubé Quintín par3103

Ricard Guixaró Trancho par3108

Fall 2021-2022

Date: 06/10/2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Node architecture and memory

Boada is a multiprocessor server, composed of 8 nodes (boada-1 - boada-8). To obtain data about the nodes' hardware architecture we use 2 commands:

- **$ *lscpu***

  It shows a list of aspects of the current node's hardware directly in the terminal. The obtained information is the same for nodes boada-1 to boada-4. We can see a summary of the important architectural characteristics obtained with ***lscpu*** in *Table 1*.

- **$ *lstopo --of fig map.fig***

  With this command we create a map (creating a new file, *map.fig*) showing the architecture of the node, visualizing the information obtained by the ***lscpu*** command, such as that there are, in fact, 2 sockets in this node (green background). We can also see the caches and their respective sizes, the number of cores in each socket (6), etc.

  The map allows us to complete *Table 1*, as now we can see the main memory size per socket and per node. We visualize it using the command ***xfig map.fig***.

	boada-1 **to** boada-4
Number of sockets per node	2
Number of cores per socket	6
Number of threads per core	2
Maximum core frequency	2395MHz
L1-I cache size (per-core)	32k
L1-D cache size (per-core)	32k
L2 cache size (per-core)	256k
Last-level cache size (per-socket)	12288K
Main memory size (per socket)	35GB
Main memory size (per node)	23GB + 12GB

Execution of the command lstopo --of fig map.fig, openned with *xfig*.

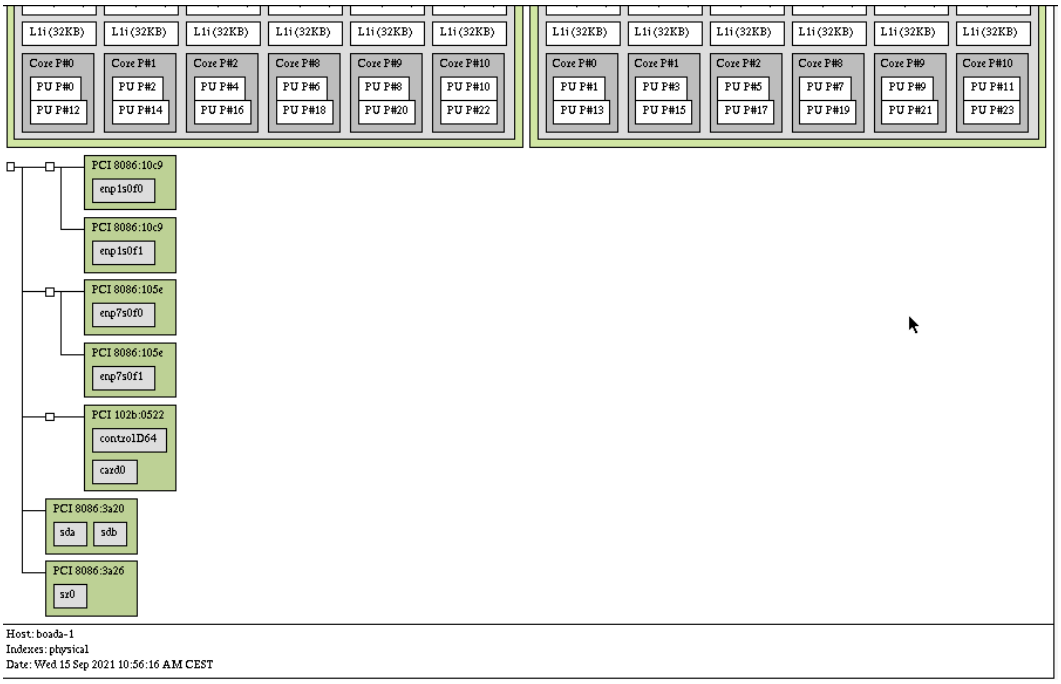


Figure 1: Boada-1 architecture

Strong vs. weak scalability

Now, we will be observing the scalability when varying the number of threads used in a parallel program. In our case, the parallel version we'll be studying is in the *pi_omp.c* file, located in the */scratch/nas/1/parXXXX/lab1/pi* directory.

The scalability shows how the speed-up evolves when the number of processors is increased. It is measured by calculating the ratio between the sequential (T1) and parallel (Tp for p threads) execution times (speed-up) in two different cases:

- **Strong scalability**: Parallelism is used to reduce the execution time of the program.
    - The problem size is fixed.
    - The number of threads is changed.
- **Weak scalability**: Parallelism is used to increase the problem size.
    - The problem size is proportional to the number of threads.

To analyze said speed-up, we use 2 scripts: *submit-strong-omp.sh* and *submit-weak-omp.sh*. First we submit *submit-strong-omp.sh,* using the command ***sbatch -p execution submit-strong-omp.sh*** (no

arguments needed). The problem size is 1.000.000.000 iterations. If we want to check the status we use ***squeue***.

The .sh file generates a Postscript file which we open with the ***gs*** command. Once we use the ***gs*** command we obtain the following plots:

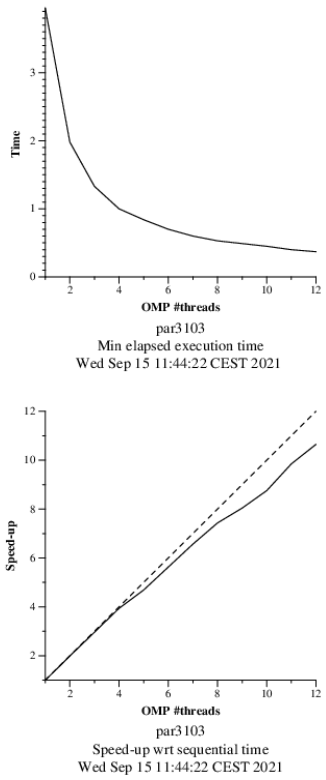


Figure 2: Plot of strong scalability showing both the minimum elapsed execution time and the speedup.

As we can see in the first graphic, the execution time of the program is reduced by the usage of more threads. In the second one we can observe that the speed-up approximates the perfect straight line and increases in a linear form.

If we change the value for np_NMAX in the *submit-strong-omp.sh* from 12 to 24, we obtain the following plots:

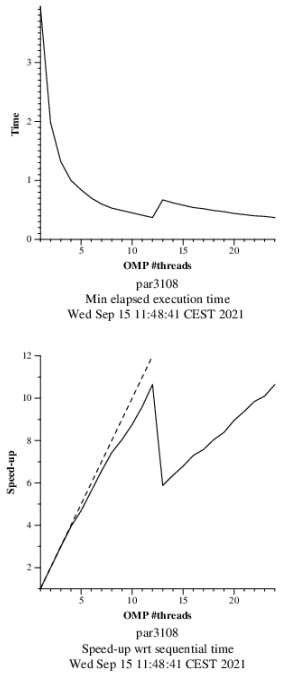


Figure 3: Strong scalability plot

The behaviour observed in the plots on the left is caused by the fact that using 12 threads is the optimum scenario.

As we can see, the lines change at 13 threads - that is because using 13 threads is not necessary.

Now we submit the weak version and obtain a plot showing us the ratio between the speed-up and the number of threads used:

After, we submitted the weak version and obtained a plot that showed us the ratio between the speed-up and the number of threads used:

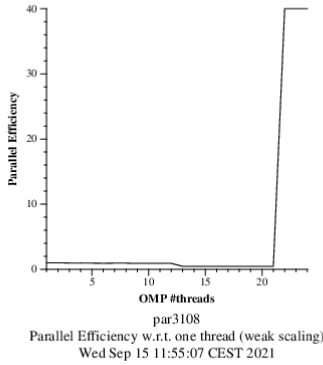


Figure 4: Weak Scaling Plot

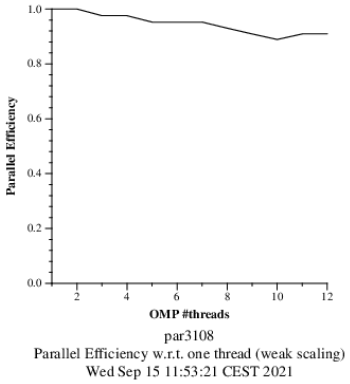


Figure 5: Weak Scaling Plot

In weak scalability the problem size is proportional to the number of threads and parallelism is used to increase the problem size and, therefore, solve larger problems.

We can observe that this line approaching the 1.0 parallel efficiency imaginary line is quite similar to the one we've observed in the strong scalability scenario (approaching the diagonal). This means that the speed-up obtained in both scenarios (weak and strong) is quite similar, even though the line obtained in the strong scalability case is more straight.

Analysis of task decompositions for 3DFFT

We use Tareador to analyze the task decompositions for the *3dfft* program. We can find the *3dfft_tar.c* file in the *lab1/3dfft* directory. We can identify the different calls to the *Tareador* API:

- **tareador_ON();** to initialize analysis with *Tareador* and **tareador_OFF();** to stop it.
- **tareador_start_task("NameOfTask");** and **tareador_end_task ("NameOfTask");** to specify the region of the code we want to consider a potential task.

Then we create an executable file (*3dfft_tar*) with the command ***make 3dfft_tar***. We execute the binary generated with ***./run-tareador.sh 3dfft_tar*** and it opens a *Tareador* window that shows a task dependence graph (*Photo 1*) that we can see on the right. We will refer to this task decomposition as *v0*.

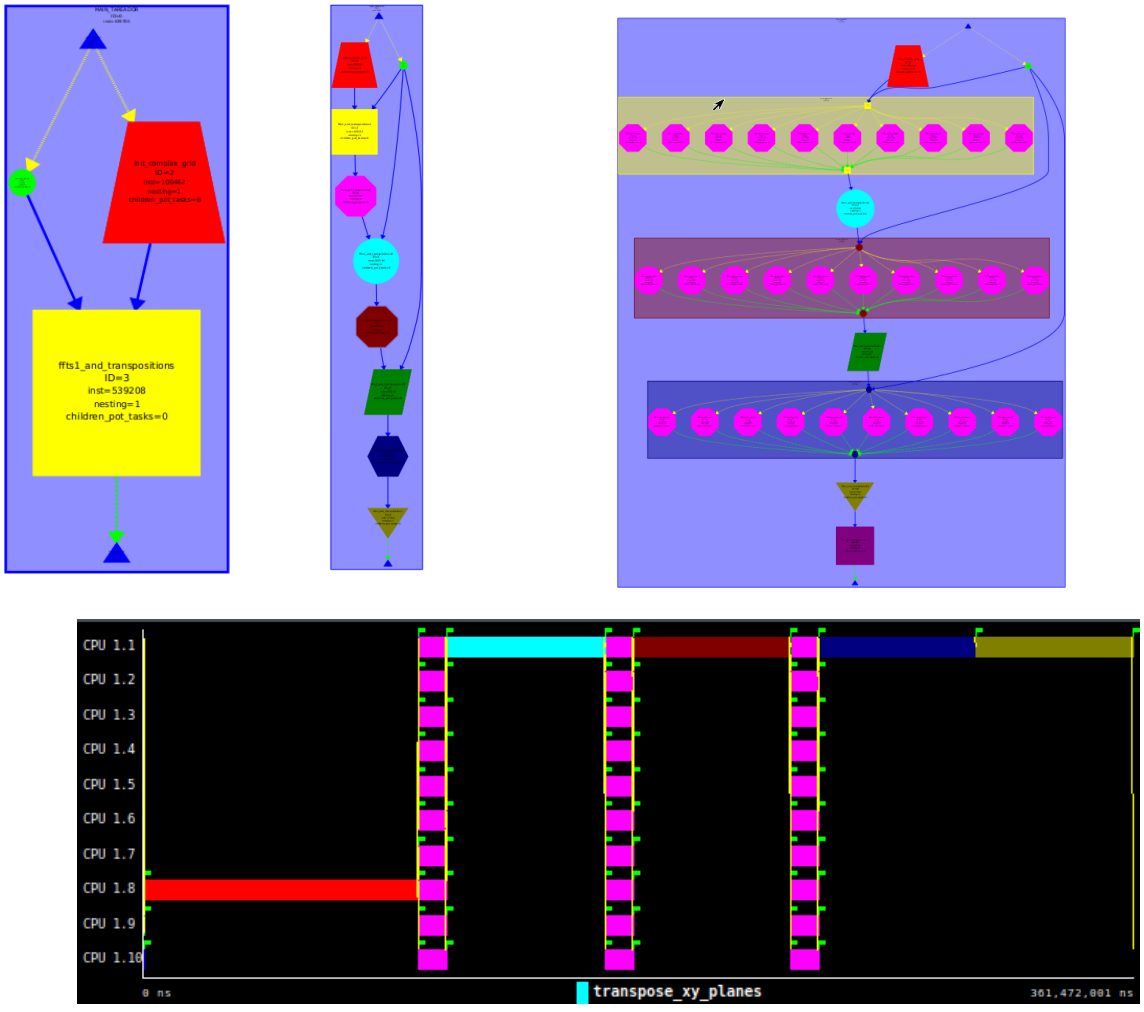
Each node of the graph represents a task and each one is labeled with a task instance number. Edges in the graph represent different dependencies between task instances with different colours

# threads	Interactive				Queued			
	user	system	elapsed	% of CPU	user	system	elapsed	% of CPU
1	3.94	0	03.94	99	3.94	0.00	03.97	99
2	7.94	0	3.98	199	3.95	0.01	2.00	198
4	8.00	0.03	4.02	199	3.99	0.00	1.02	390
8	7.97	0.05	04.01	199	4.18	0.00	00.54	770

Also explain what strong and weak scalability refer to, exemplifying your explanation with the execution time and speed–up plots that you obtained for pi omp.c.

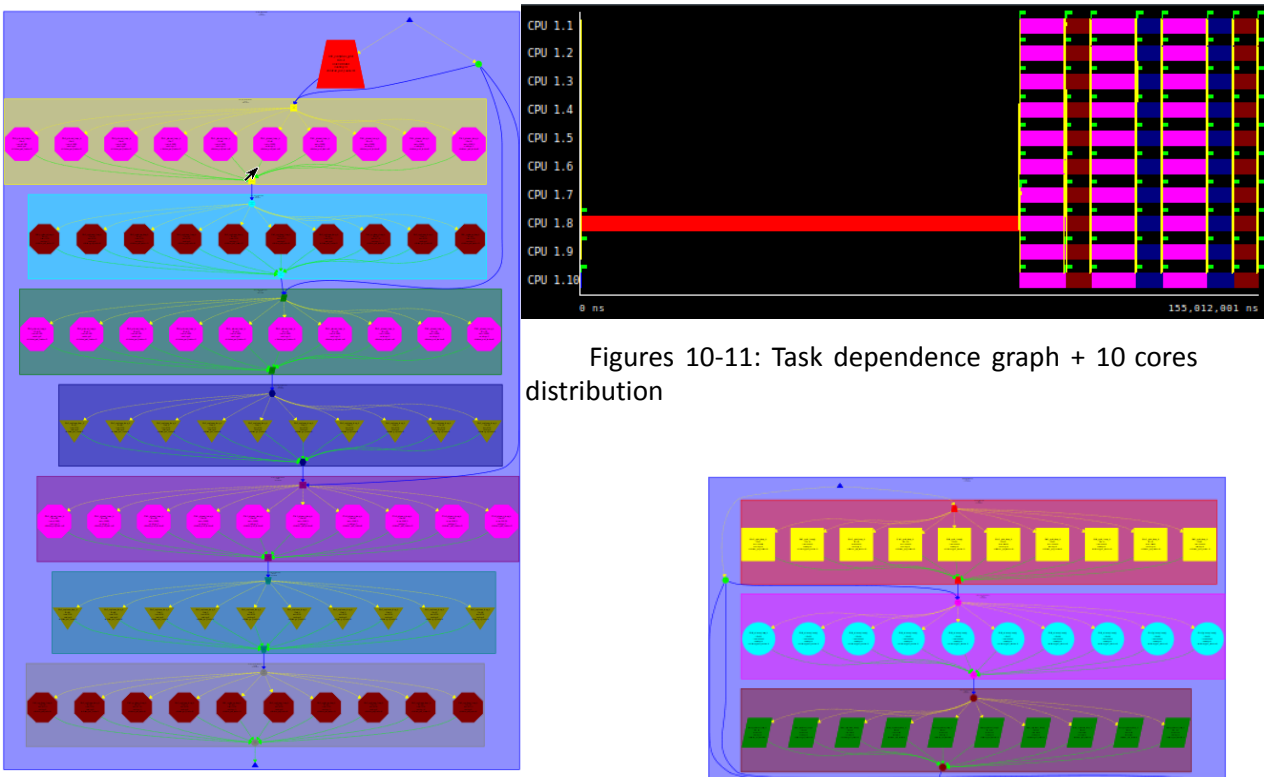
Analysis of task decompositions for *3DFFT*

Initial version: First version: Second version (TDG + 8 cores distribution):



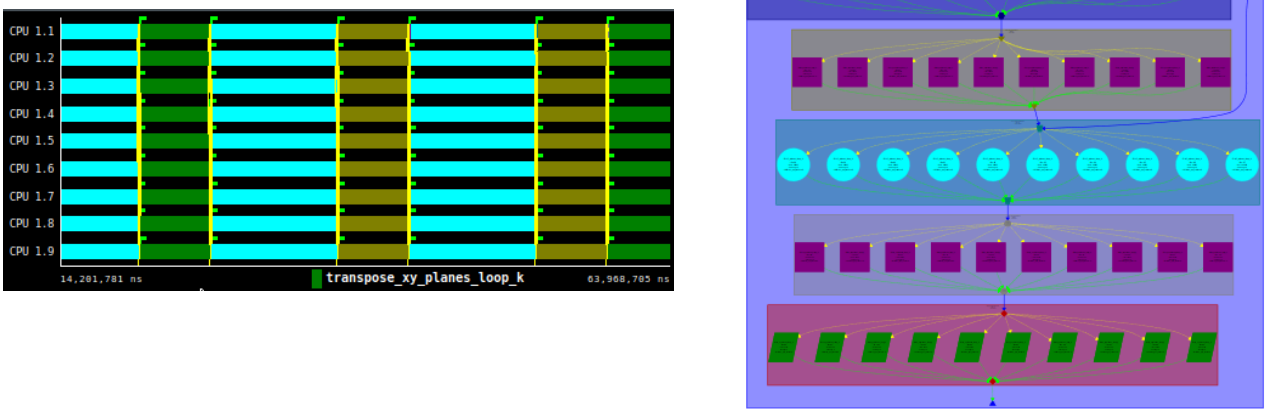
Figures 6-9: Task Dependence Graph for versions 0, 1 and 2.

Third version:



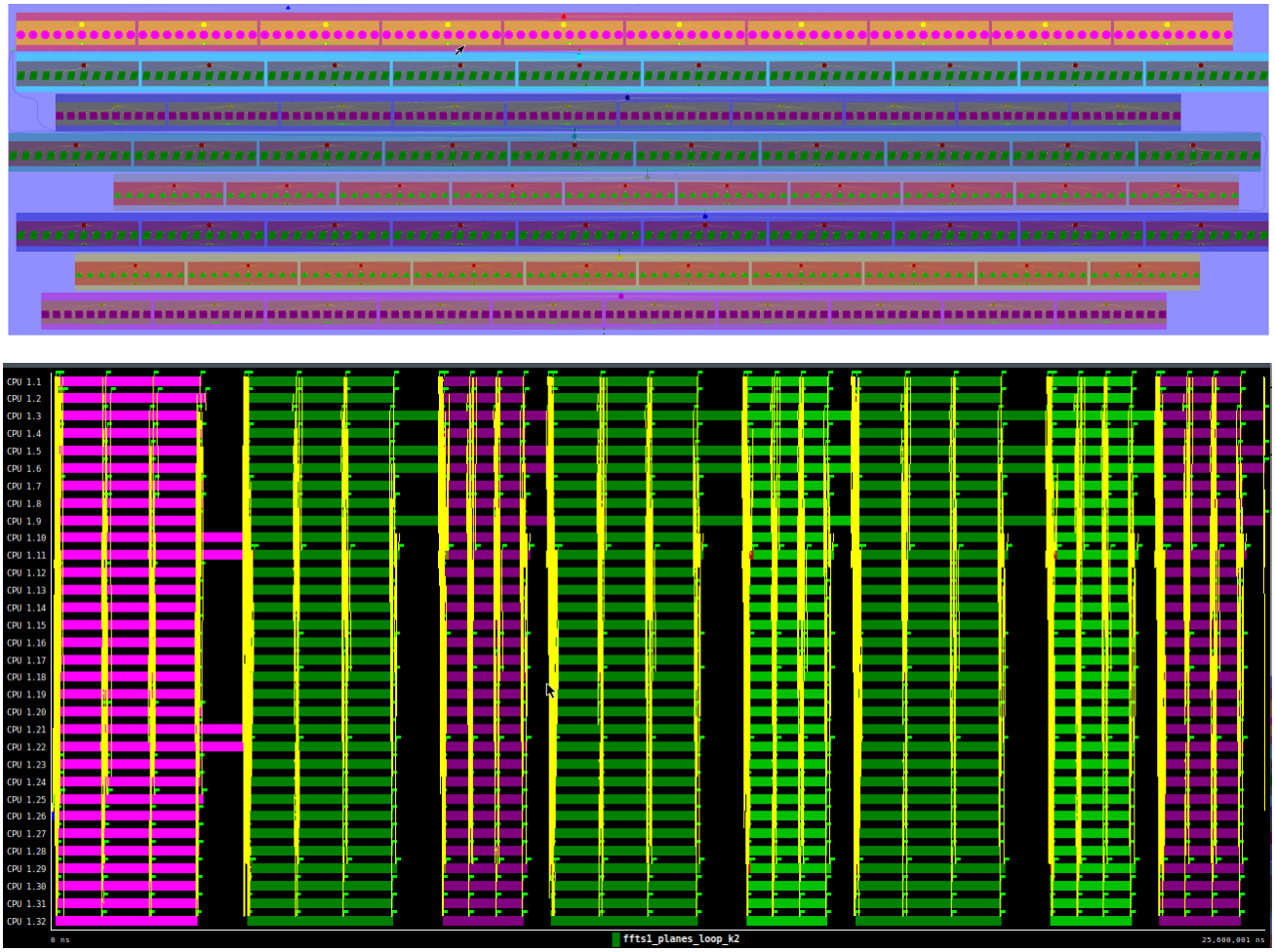
Figures 10-11: Task dependence graph + 10 cores distribution

Fourth version:



Figures 12-13: Task dependence graph + 10 cores distribution

Fifth version:



Figures 14-15: Task dependence graph + 32 cores distribution)**:**

From the task dependence graphs above we can see that as each version progressively decomposes the code, its execution time drops.

Therefore, we can deduce that the more we decompose and parallelize the code, the less it takes to execute it, although in order to truly improve the time, it is essential that we also increase the number of cores we use. Eventually though, we will get to a certain point where more cores won't help us to parallelize the code, since there won't be enough tasks for all the cores..

As mentioned above, in the last table (v4) we see that even if we increase the number of cores from 10 to 32, the execution time remains the same.

Version	T_1	T_∞	Parallelism
seq	639780	639780	1.0(no hi paral.)
v1	639780	639760	1.0000(no hi ha paral.)
v2	639780	361472	1.7699
v3	639780	155012	4.1272
v4	639780	64697	9.8888
v5	639780	7661	83.511

version/cores	1	2	4	8	16	32
v4	639780	320257	191882	127992	64614	64614
v5	639789	320321	161544	83105	44563	25600

For versions v4 and v5 of 3dfft tar.c perform an analysis of the potential strong scalability that is expected. For that include a plot with the execution time and speedup when using 1, 2, 4, 8, 16 and 32 processors, as reported by the simulation module inside *Tareador* . You should also include the relevant(s) part(s) of the code to understand why v5 is able to scale to a higher number of processors compared to v4.

Understanding the parallel execution of *3DFFT*

3.2.1 INITIAL VERSION

In this first version of the code, the *init_complex_grid* function is not entirely parallelized, which implies that the parallel fraction is low. The improvement from using 1 to 8 threads is considerable, though.

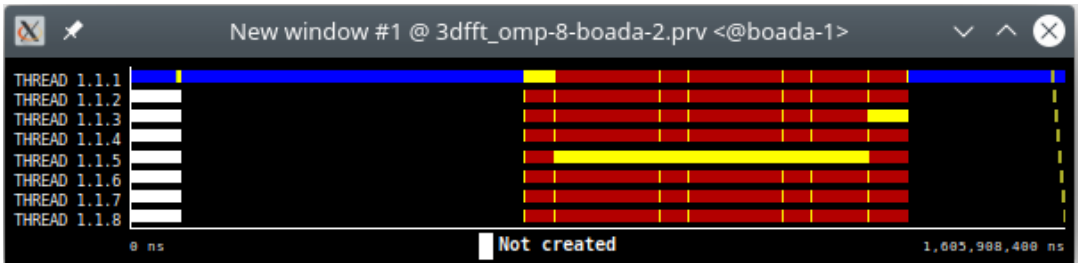


Figure 16: Execution of 3dfft_omp with 8 threads.

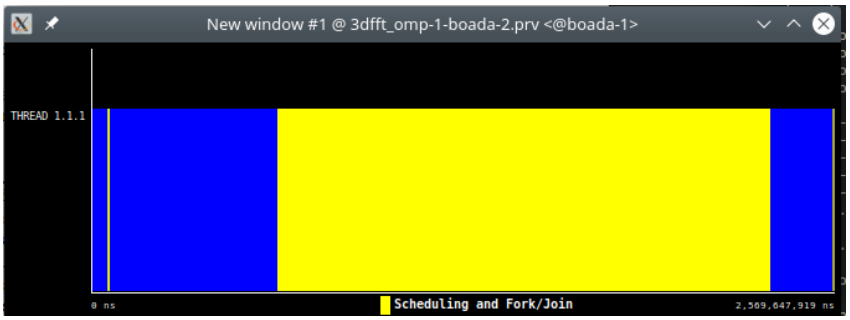


Figure 17: Execution of 3dfft_omp with 1 thread.

On this part we will use the parallel_constructs configuration file to see the parallel constructs.

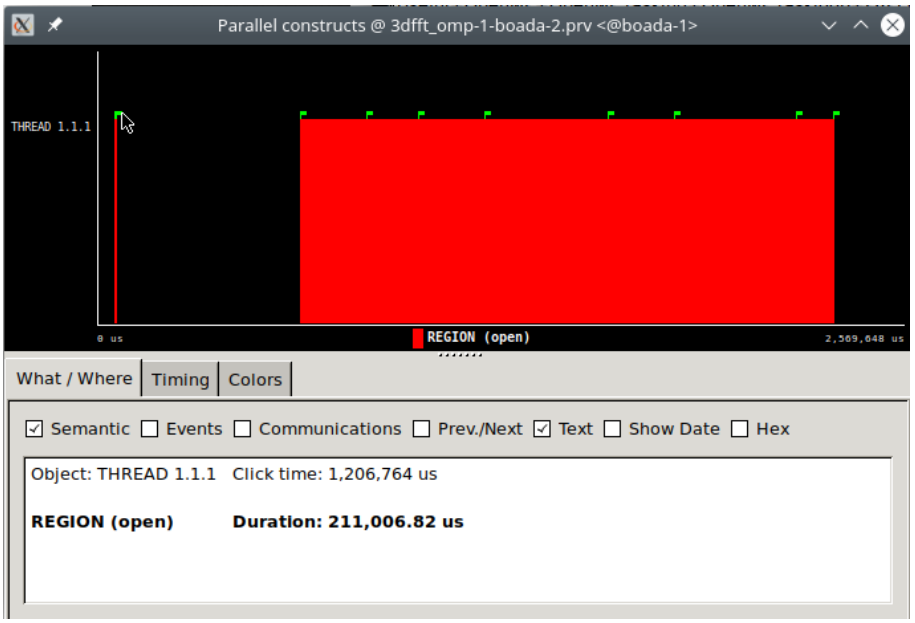


Figure 18: Parallel Constructs of the execution of 3dfft_omp with 1 thread.

We will also use the implicit tasks profiles to calculate the parameter phi, so we can calculate the other expressions asked.

2D Implicit tasks profile @ 3dfft_omp-1-boada-2.prv <@boada-1>

Default

	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	282,227.96 us	716,593.38 us	633,143.86 us	16.87 us	2.63 us
Total	282,227.96 us	716,593.38 us	633,143.86 us	16.87 us	2.63 us
Average	282,227.96 us	716,593.38 us	633,143.86 us	16.87 us	2.63 us
Maximum	282,227.96 us	716,593.38 us	633,143.86 us	16.87 us	2.63 us
Minimum	282,227.96 us	716,593.38 us	633,143.86 us	16.87 us	2.63 us
StDev	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1

Figure 19: Implicit tasks profile of the execution of 3dfft_omp with 1 thread.

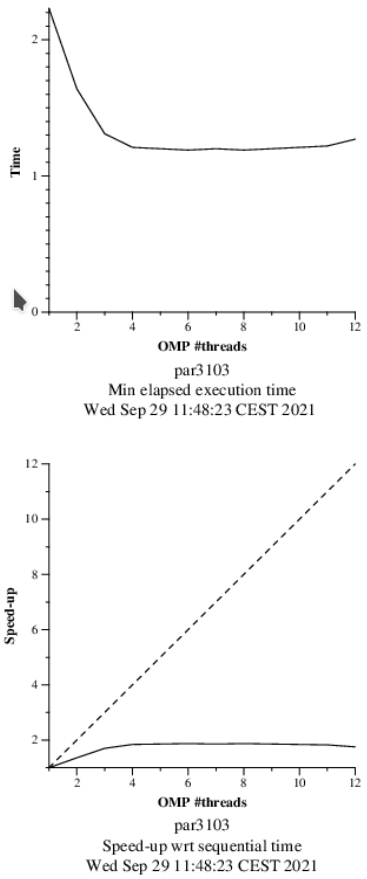


Figure 20: Strong scalability plot of the initial version of 3dfft_omp.c

3.2.2 IMPROVING φ

In this improved version we reduced the time that we spend creating and synchronizing all the threads of the execution. To do so, we started to make the function *init_complex_grid* more parallelizable by uncommenting a line inside the loop.

Doing so, we managed to increase the parallel fraction of the code from 0.64 to 0.87, which led to a rise of the speed up with 8 threads from 1.95 to 2.23. TNevertheless, that speed-up is still far below from the ideal (7.87).

Figure 21: Improved execution of 3dfft_omp with 1 thread.

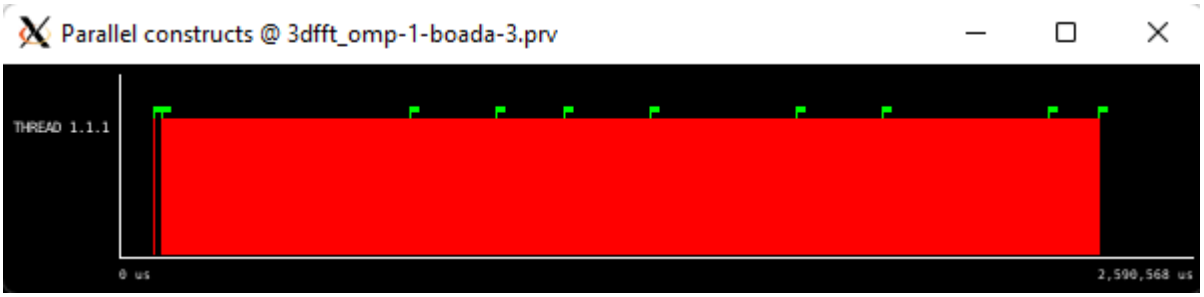
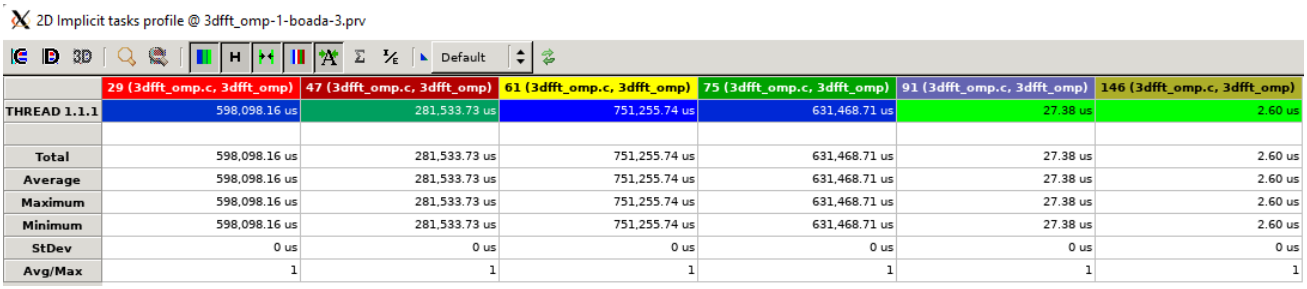


Figure 22: Parallel Constructs of the improved execution of 3dfft_omp with 1 thread.



	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	598,098.16 us	281,533.73 us	751,255.74 us	631,468.71 us	27.38 us	2.60 us
Total	598,098.16 us	281,533.73 us	751,255.74 us	631,468.71 us	27.38 us	2.60 us
Average	598,098.16 us	281,533.73 us	751,255.74 us	631,468.71 us	27.38 us	2.60 us
Maximum	598,098.16 us	281,533.73 us	751,255.74 us	631,468.71 us	27.38 us	2.60 us
Minimum	598,098.16 us	281,533.73 us	751,255.74 us	631,468.71 us	27.38 us	2.60 us
StDev	0 us	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1	1

Figure 23: Implicit tasks profile of the improved execution of 3dfft_omp with 1 thread.

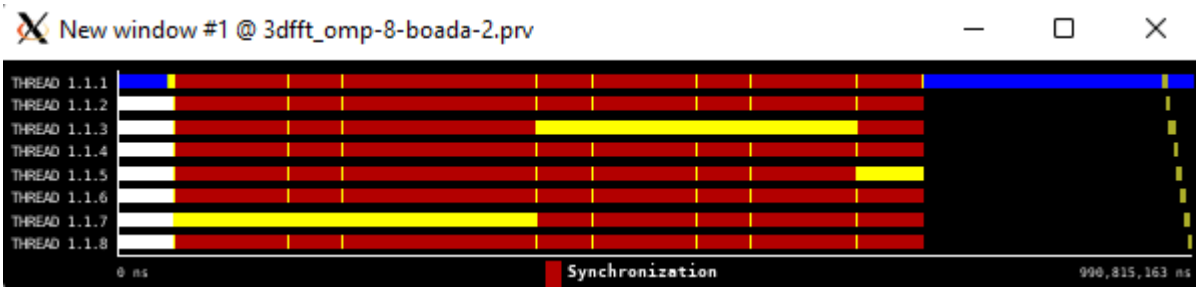


Figure 24: Improved execution of 3dfft_omp with 8 threads.

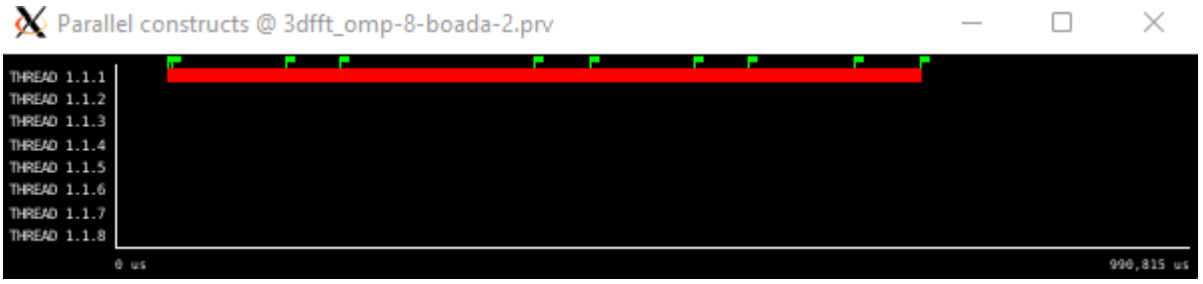


Figure 25: Improved execution of 3dfft_omp with 8 threads.

2D Implicit tasks profile @ 3dfft_omp-8-boada-2.prv

Default

	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	105,802.84 us	241,068.67 us	193,915.31 us	150,312.55 us	26.39 us	3.26 us
THREAD 1.1.2	105,802.30 us	241,071.75 us	193,918.26 us	150,315.55 us	205.49 us	2.80 us
THREAD 1.1.3	105,801.15 us	241,069.07 us	193,919.73 us	150,311.80 us	211.78 us	2.73 us
THREAD 1.1.4	105,803.32 us	241,071.58 us	193,918.47 us	150,315.70 us	421.62 us	2.27 us
THREAD 1.1.5	105,801.02 us	241,068.08 us	193,915.04 us	150,312.70 us	200.04 us	2.58 us
THREAD 1.1.6	105,801.99 us	241,071.52 us	193,918.35 us	150,315.73 us	173.95 us	2.63 us
THREAD 1.1.7	105,800.60 us	241,068.34 us	193,915.98 us	150,313.14 us	149.05 us	3.13 us
THREAD 1.1.8	105,803.43 us	241,071.62 us	193,918.35 us	150,315.39 us	140.07 us	2.62 us
Total	846,416.64 us	1,928,560.63 us	1,551,339.48 us	1,202,512.55 us	1,528.39 us	22.02 us
Average	105,802.08 us	241,070.08 us	193,917.43 us	150,314.07 us	191.05 us	2.75 us
Maximum	105,803.43 us	241,071.75 us	193,919.73 us	150,315.73 us	421.62 us	3.26 us
Minimum	105,800.60 us	241,068.08 us	193,915.04 us	150,311.80 us	26.39 us	2.27 us
StDev	1.01 us	1.56 us	1.62 us	1.56 us	103.47 us	0.30 us
Avg/Max	1.00	1.00	1.00	1.00	0.45	0.84

Figure 26: Implicit tasks profile of the improved execution of 3dfft_omp with 8 threads.

Here we can compare the initial timelines of both versions and see the evident change of the parallelism:

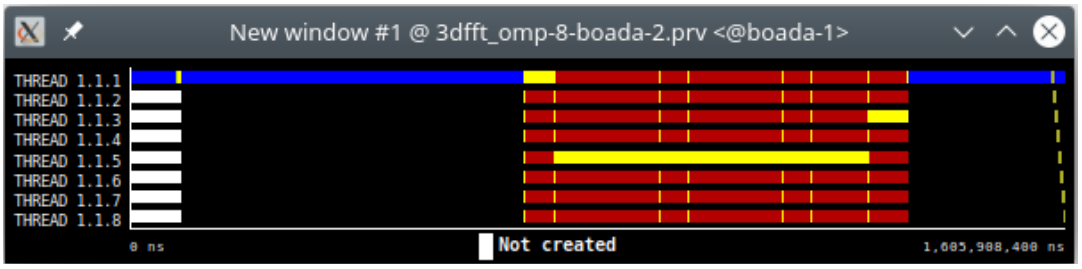


Figure 27: Timeline of the execution of 3dfft_omp with 8 threads.

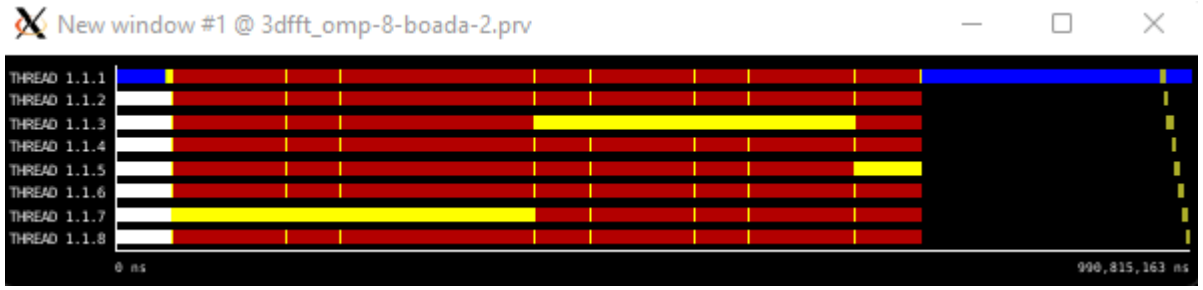


Figure 28: Timeline of the improved execution of 3dfft_omp with 8 threads.

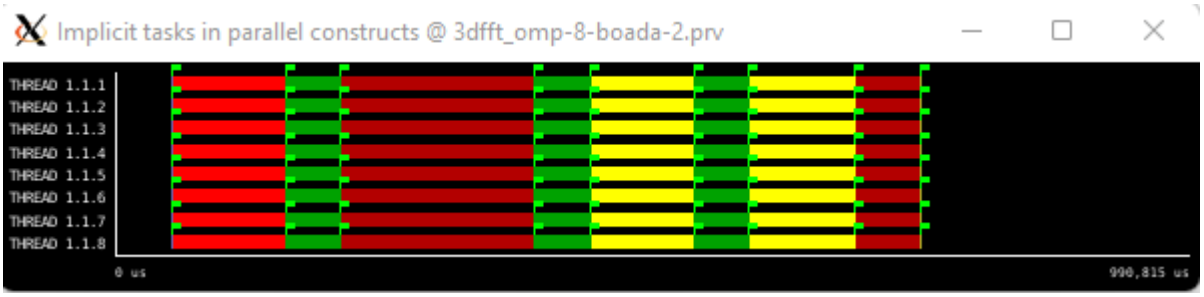
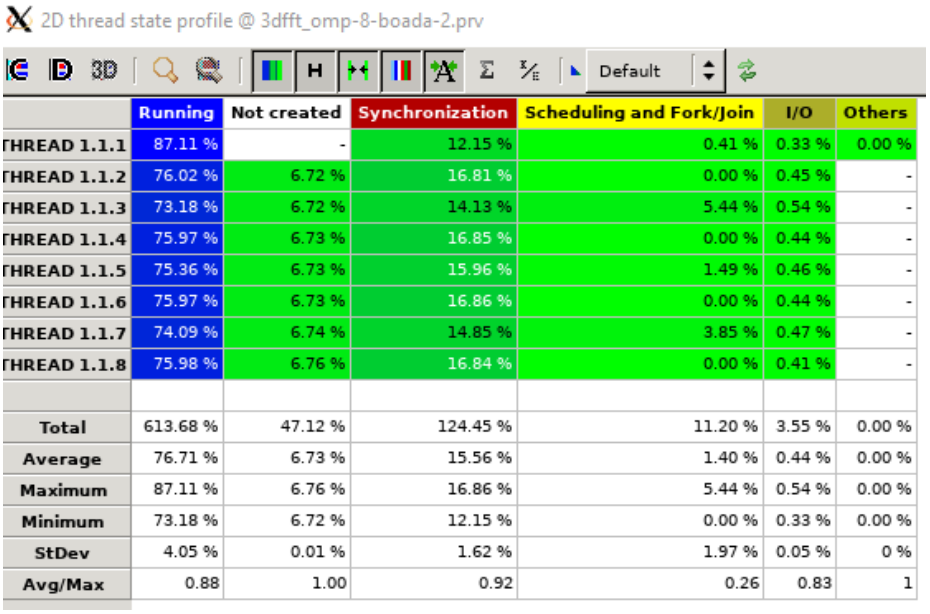
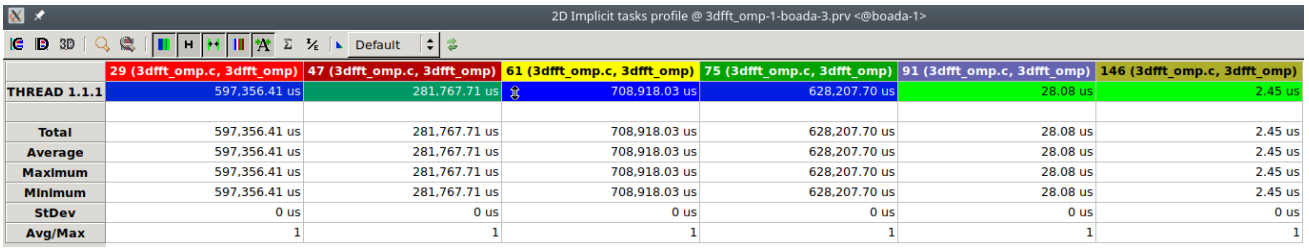


Figure 29: Histogram implicits tasks of the improved execution of 3dfft_omp with 8 threads.



	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	87.11 %	-	12.15 %	0.41 %	0.33 %	0.00 %
THREAD 1.1.2	76.02 %	6.72 %	16.81 %	0.00 %	0.45 %	-
THREAD 1.1.3	73.18 %	6.72 %	14.13 %	5.44 %	0.54 %	-
THREAD 1.1.4	75.97 %	6.73 %	16.85 %	0.00 %	0.44 %	-
THREAD 1.1.5	75.36 %	6.73 %	15.96 %	1.49 %	0.46 %	-
THREAD 1.1.6	75.97 %	6.73 %	16.86 %	0.00 %	0.44 %	-
THREAD 1.1.7	74.09 %	6.74 %	14.85 %	3.85 %	0.47 %	-
THREAD 1.1.8	75.98 %	6.76 %	16.84 %	0.00 %	0.41 %	-
Total	613.68 %	47.12 %	124.45 %	11.20 %	3.55 %	0.00 %
Average	76.71 %	6.73 %	15.56 %	1.40 %	0.44 %	0.00 %
Maximum	87.11 %	6.76 %	16.86 %	5.44 %	0.54 %	0.00 %
Minimum	73.18 %	6.72 %	12.15 %	0.00 %	0.33 %	0.00 %
StDev	4.05 %	0.01 %	1.62 %	1.97 %	0.05 %	0 %
Avg/Max	0.88	1.00	0.92	0.26	0.83	1

Figure 30: Thread state profile (% time) of the improved version.



	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	597,356.41 us	281,767.71 us	708,918.03 us	628,207.70 us	28.08 us	2.45 us
Total	597,356.41 us	281,767.71 us	708,918.03 us	628,207.70 us	28.08 us	2.45 us
Average	597,356.41 us	281,767.71 us	708,918.03 us	628,207.70 us	28.08 us	2.45 us
Maximum	597,356.41 us	281,767.71 us	708,918.03 us	628,207.70 us	28.08 us	2.45 us
Minimum	597,356.41 us	281,767.71 us	708,918.03 us	628,207.70 us	28.08 us	2.45 us
StDev	0 us	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1	1

Figure 31: Implicit tasks profile of the improved execution of 3dfft_omp with 1 thread.

In the next figure we can appreciate how once we reach 8 threads, the execution time won't improve significantly no matter how many threads we use.

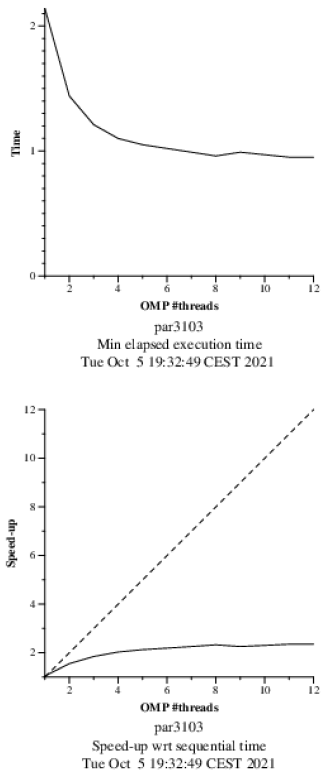


Figure 32: Strong scalability plot of the improved version of 3dfft_omp.c

3.2.3 REDUCING PARALLELISATION OVERHEADS

We will increase the granularity of tasks by moving all #pragma omp taskloop one line up in the four functions that are parallelised (*init_complex_grid()*, *transpose_xy_planes()*, *transpose_zx_planes()* and *ftts1_planes()*).

This way, we reduce the overhead caused by parallelisation. As shown in the table below, the speed-up for 8 threads has gone from 2.23 up to 5.8.

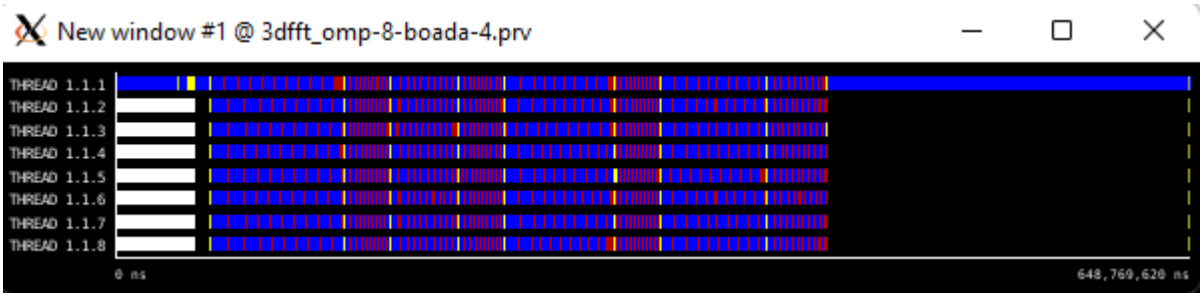


Figure 33: Timeline of the final execution of 3dfft_omp with 8 threads.

2D Implicit tasks profile @ 3dfft_omp-8-boada-4.prv

Default

	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	81,593.04 us	77,156.12 us	130,918.16 us	82,751.63 us	26.16 us	2.37 us
THREAD 1.1.2	81,594.56 us	77,159.27 us	130,920.82 us	82,752.82 us	169.15 us	2.28 us
THREAD 1.1.3	81,593.29 us	77,159.70 us	130,916.13 us	82,750.35 us	172.33 us	2.24 us
THREAD 1.1.4	81,595.77 us	77,156.43 us	130,918.93 us	82,753.79 us	152.69 us	2.22 us
THREAD 1.1.5	81,592.99 us	77,169.29 us	130,916.11 us	82,748.50 us	156.30 us	2.32 us
THREAD 1.1.6	81,594.78 us	77,156.09 us	130,920.50 us	82,751.69 us	139.12 us	2.27 us
THREAD 1.1.7	81,592.79 us	77,159.14 us	130,917.16 us	82,751.99 us	150.57 us	2.16 us
THREAD 1.1.8	81,594.72 us	77,159.57 us	130,920.32 us	82,752.55 us	118.79 us	2.26 us
Total	652,751.93 us	617,275.61 us	1,047,348.13 us	662,013.32 us	1,085.12 us	18.11 us
Average	81,593.99 us	77,159.45 us	130,918.52 us	82,751.67 us	135.64 us	2.26 us
Maximum	81,595.77 us	77,169.29 us	130,920.82 us	82,753.79 us	172.33 us	2.37 us
Minimum	81,592.79 us	77,156.09 us	130,916.11 us	82,748.50 us	26.16 us	2.16 us
StDev	1.03 us	4.01 us	1.81 us	1.52 us	44.29 us	0.06 us
Avg/Max	1.00	1.00	1.00	1.00	0.79	0.96

Figure 34: Implicit tasks profile of the final execution of 3dfft_omp with 8 threads.

1 thread:

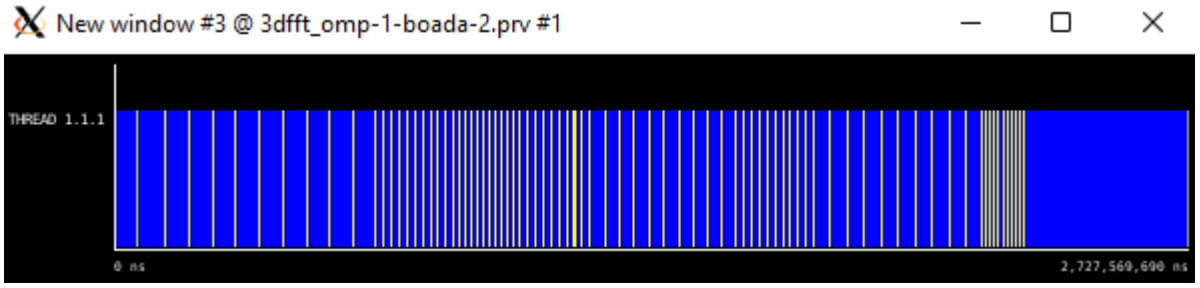


Figure 35: Timeline of the final execution of 3dfft_omp with 1 thread.

2D Implicit tasks profile @ 3dfft_omp-1-boada-2.prv #1

Default

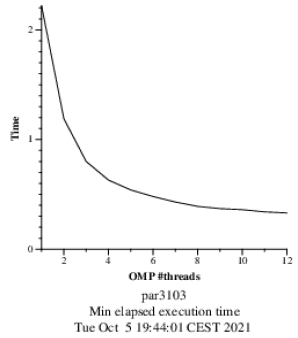
	29 (3dfft_omp.c, 3dfft_omp)	47 (3dfft_omp.c, 3dfft_omp)	61 (3dfft_omp.c, 3dfft_omp)	75 (3dfft_omp.c, 3dfft_omp)	91 (3dfft_omp.c, 3dfft_omp)	146 (3dfft_omp.c, 3dfft_omp)
THREAD 1.1.1	604,250.06 us	256,130.91 us	801,100.39 us	589,038.03 us	28.32 us	2.89 us
Total	604,250.06 us	256,130.91 us	801,100.39 us	589,038.03 us	28.32 us	2.89 us
Average	604,250.06 us	256,130.91 us	801,100.39 us	589,038.03 us	28.32 us	2.89 us
Maximum	604,250.06 us	256,130.91 us	801,100.39 us	589,038.03 us	28.32 us	2.89 us
Minimum	604,250.06 us	256,130.91 us	801,100.39 us	589,038.03 us	28.32 us	2.89 us
StDev	0 us	0 us	0 us	0 us	0 us	0 us
Avg/Max	1	1	1	1	1	1

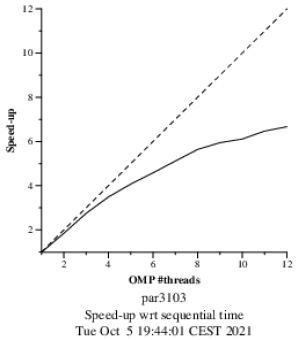
Figure 36: Implicit tasks profile of the final execution of 3dfft_omp with 1 thread.

2D thread state profile @ 3dfft_omp-8-boada-4.prv #1

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	96.79 %	-	2.55 %	0.65 %	0.01 %	0.00 %
THREAD 1.1.2	86.32 %	11.07 %	2.60 %	0.01 %	0.01 %	-
THREAD 1.1.3	85.73 %	11.08 %	3.10 %	0.09 %	0.01 %	-
THREAD 1.1.4	87.53 %	11.07 %	1.39 %	0.01 %	0.01 %	-
THREAD 1.1.5	86.22 %	11.08 %	2.61 %	0.07 %	0.01 %	-
THREAD 1.1.6	86.16 %	11.09 %	2.73 %	0.01 %	0.01 %	-
THREAD 1.1.7	87.27 %	11.08 %	1.63 %	0.01 %	0.01 %	-
THREAD 1.1.8	86.91 %	11.11 %	1.97 %	0.01 %	0.01 %	-
Total	702.93 %	77.59 %	18.59 %	0.84 %	0.05 %	0.00 %
Average	87.87 %	11.08 %	2.32 %	0.10 %	0.01 %	0.00 %
Maximum	96.79 %	11.11 %	3.10 %	0.65 %	0.01 %	0.00 %
Minimum	85.73 %	11.07 %	1.39 %	0.01 %	0.01 %	0.00 %
StDev	3.42 %	0.01 %	0.56 %	0.21 %	0.00 %	0 %
Avg/Max	0.91	1.00	0.75	0.16	0.84	1

Figure 37: Thread state profile (% time) of final version.





The speed-up obtained is a lot higher than the one in the previous version. We see that increasing the granularity, we've indeed reduced the parallelisation overheads, reducing the executional time, too.

Figure 38: Strong scalability plot of the final version.

Version	φ	**ideal** S_8	T_1	T_8	**real** S_8
initial version in 3dfft omp.c	0.6445	2.2932	2.286545s	1.170473s	1.9525
new version with improved φ	0.8733	7.8740	2.211555s	0.990815s	2.2320
final version with reduced parallelisation overheads	0.89	4.66	2.178 s	0.3749s	5.8