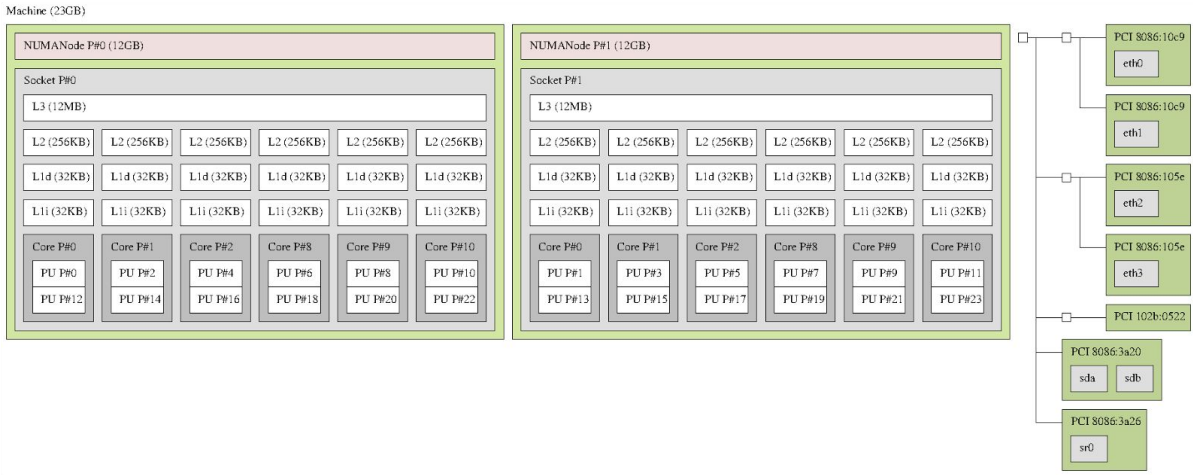
PAR Laboratory Assignment

Lab 1: Experimental setup and tools

PAR 2312
Fèlix Arribas Pardo
Ana Mestre Borges

Node architecture and memory

1. Draw and briefly describe the architecture of the computer in which you are doing this lab session (number of sockets, cores per socket, threads per core, cache hierarchy size and sharing, and amount of main memory).



2 sockets
6 cores per socket
2 threads per core
L1d cache (32KB), L1i (32KB)
L2 (256KB)
L3 (12288KB = 12MB)

Timing sequential and parallel executions

2. Describe what do you need to add to your program to measure the elapsed execution time between a pair of points in the program, clearly indicating the library header file that needs to be included, the library functions that need to be invoked, the data structure and its fields.

To get the elapsed time between two points of the code we can use GNU library: `/usr/bin/time` and `gettimeofday`.

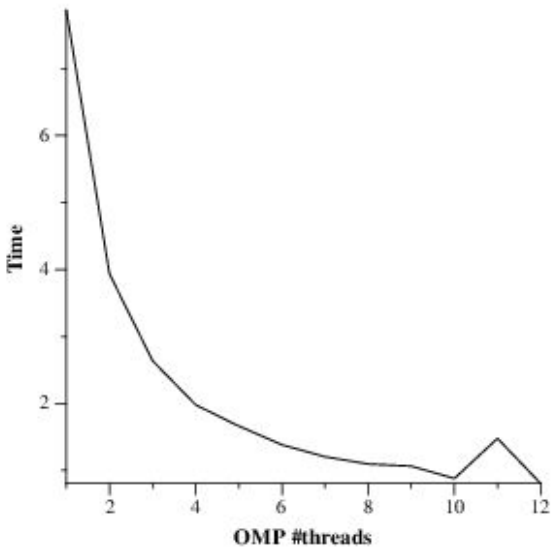
That operation returns the milliseconds since 1970. So we can subtract them and get the difference that is the program execution time.

```
#include <time.h>
#include <sys/time.h>
...
struct timeval start, end;
gettimeofday(&start);
...
gettimeofday(&end);
// we can get the elapsed time by doing: end.tv_sec - start.tv_sec
```

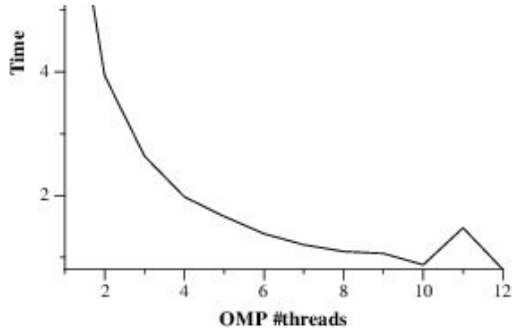
3. Plot the speed–up obtained when varying the number of threads (strong scalability) and problem size (weak scalability) for pi omp.c. Reason about how the scalability of the program.

Strong

As we can see in the execution time plot, the time decreases as more processors are used. This is due to the improvement of the program's parallelism.



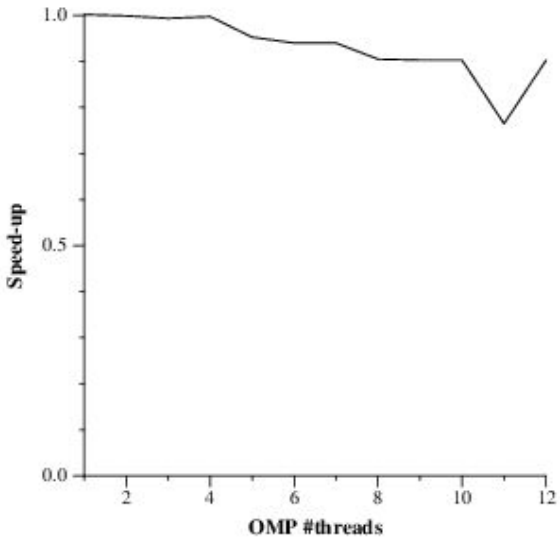
Min elapsed execution time



Min elapsed execution time

Weak

In this plot we are able to appreciate the dependencies between the program's size and the number of processors.



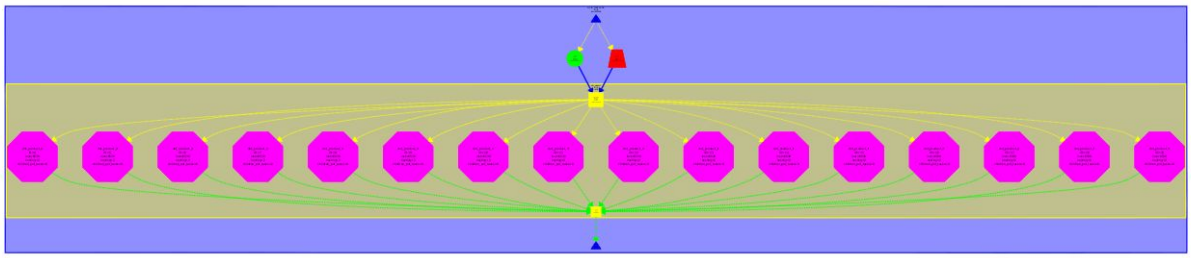
Speed-up wrt one thread, weak scaling

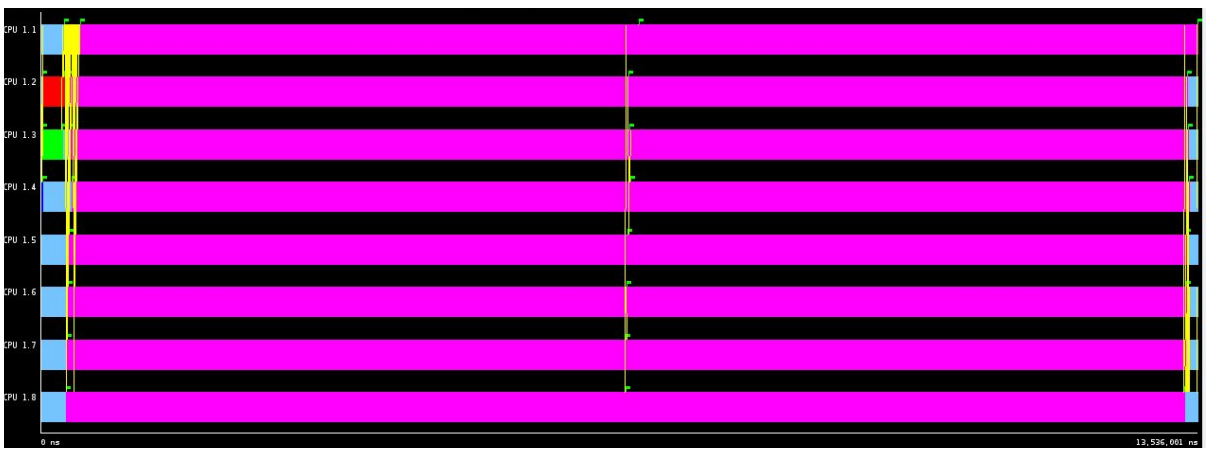
Visualizing the task graph and data dependences

4. Include the source code for function dot product in which you show the Tareador instrumentation that has been added to study the potential parallelism in the code. This instrumentation has to appropriately define tasks and filter the analysis of variable(s) that cause the dependence(s).

```
void dot_product(long N, double A[N], double B[N], double *acc) {
	double prod;
	int i;
	*acc=0.0;
	for (i = 0; i < N; i++) {
		tareador_start_task("dot_product_it");
		prod = my_func(A[i], B[i]);
		tareador_disable_object(acc);
		*acc += prod;
		tareador_enable_object(acc);
		tareador_end_task("dot_product_it");
	}
}
```

5. Capture the task dependence graph for that task decomposition and the execution timelines (for 8 processors) that allow you to understand the potential parallelism attainable. Briefly comment the relevant information that is reported by the tools.





On the sequential program every task depends on the previous one. When we disable the object *acc* we can do parallelism for each loop iteration (the pink task).

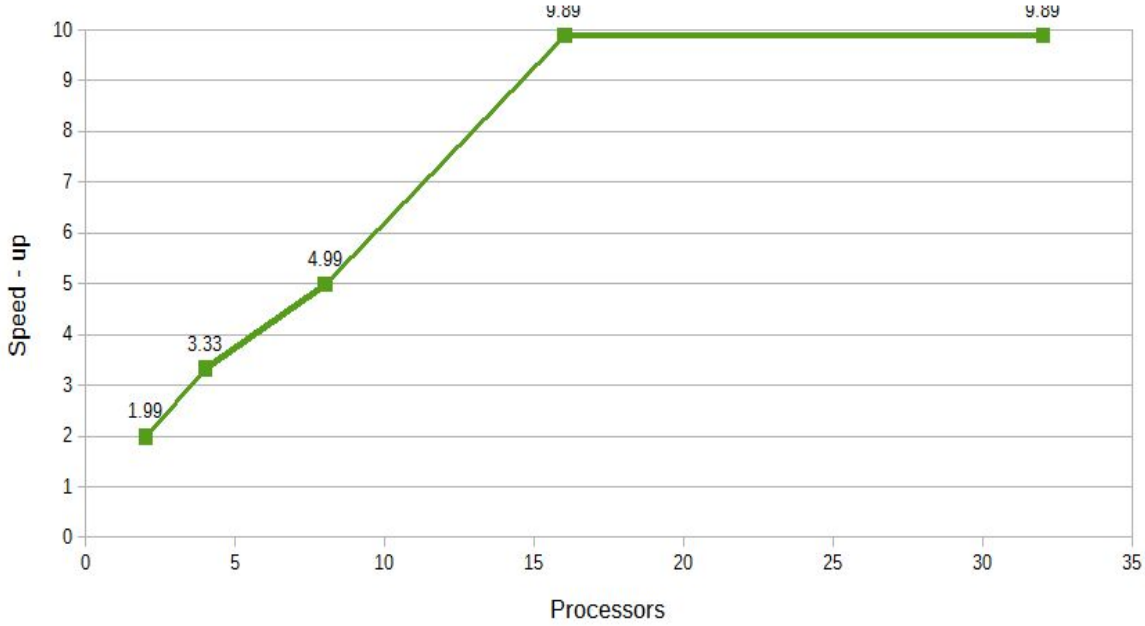
Analysis of task decompositions

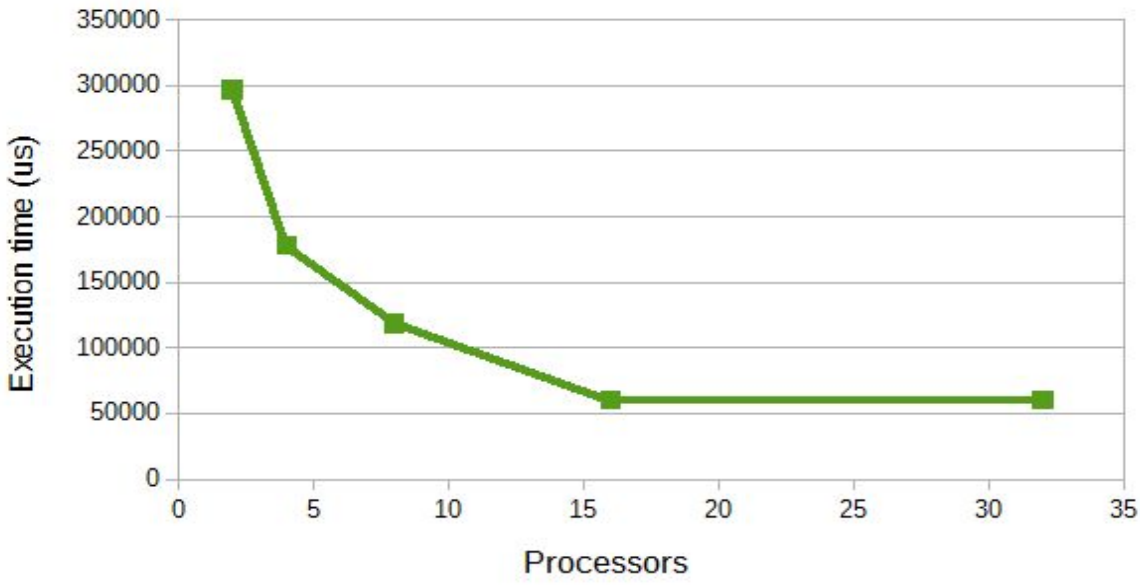
6. Complete the following table for the initial and different versions generated for 3dfft seq.c, briefly commenting the evolution of the metrics with the different versions

Version	T1	Tinf	Parallelism
seq	593,772.00 us	593,705.00 us	1.00
v1	593,772.00 us	593,705.00 us	1.00
v2	593,772.00 us	315,437.00 us	1.88
v3	593,772.00 us	108,937.00 us	5.45
v4	593,772.00 us	60,012.00 us	9.89

7. With the results from the parallel simulation with 2, 4, 8, 16 and 32 processors, draw the execution time and speedup plots for version v4 with respect to the sequential execution (that you can estimate from the simulation of the initial task decomposition that we provided in 3dfft seq.c, using just 1 processor). Briefly comment the scalability behaviour shown on these two plots.

Processors	3dfft_seq.c	3dfft_v4.c	Speed-up
2	593,705.00 us	297,255.00 us	1.99
4	593,705.00 us	178,080.00 us	3.33
8	593,705.00 us	118,790.00 us	4.99
16	593,705.00 us	60,012.00 us	9.89
32	593,705.00 us	60,012.00 us	9.89



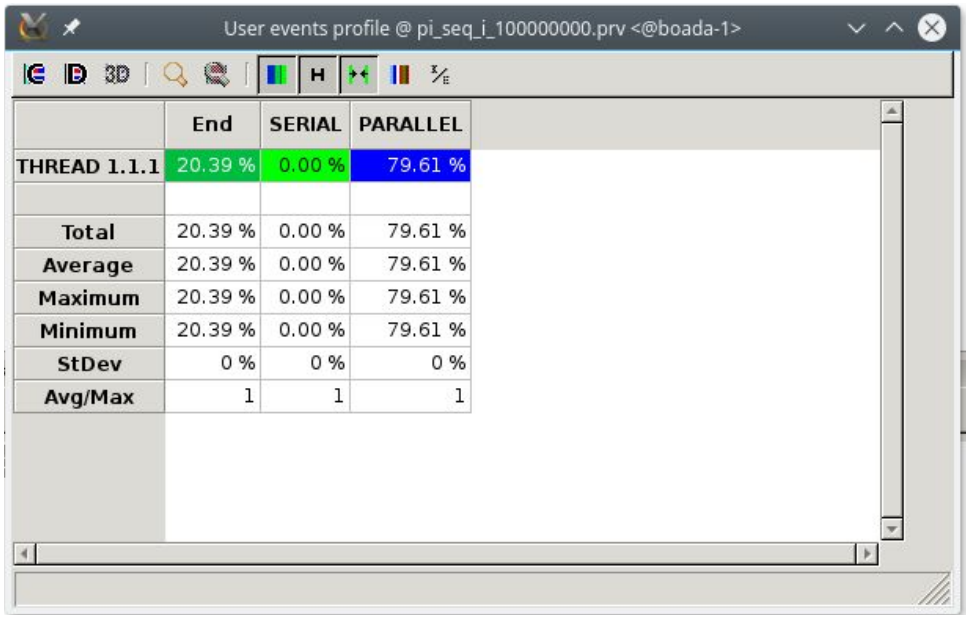


We can see that in the sequential execution the program's execution time does not change, that is because the program is always being executed in only one processor. Version 4 improves the execution time as more processors are provided.

Tracing sequential and parallel executions

8. From the instrumented version of pi seq.c, and using the appropriate Paraver configuration file, obtain the value of the parallel fraction for this program when executed with 100.000.000 iterations, showing the steps you followed to obtain it. Clearly indicate which Paraver configuration file(s) did you use.

Paraver configuration: cfgs / User / APP_user_events_profile

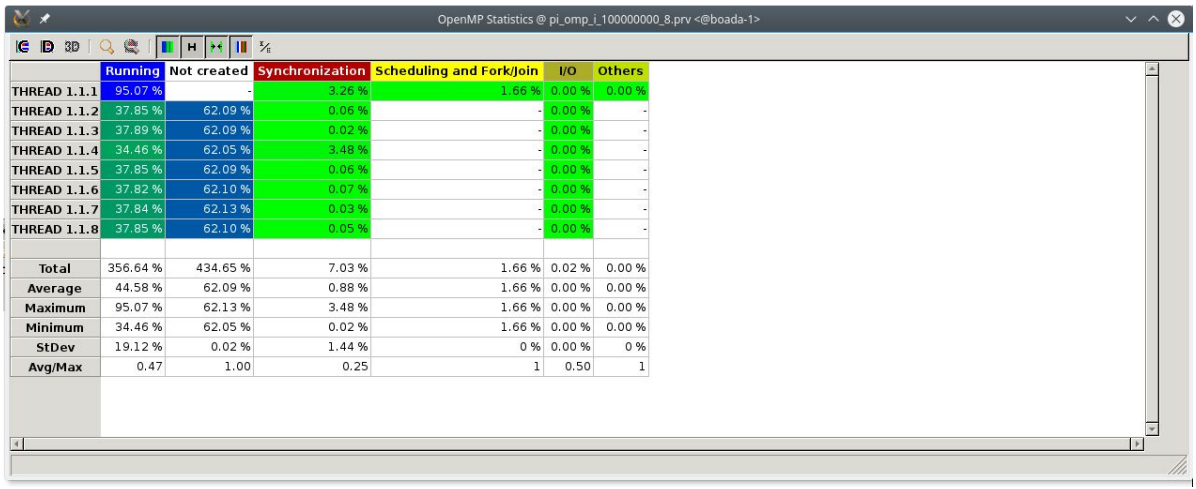


	Program	Parallelism
T_0	0	185 ms
T_{end}	994 ms	978 ms
T	994,87 ms	791,98 ms

$$\varphi = \frac{791,98}{994,87} = 0,796 = 79,60\ \%$$

9. From the instrumented version of pi_omp.c, and using the appropriate Paraver configuration file, show a profile of the % of time spent in the different OpenMP states when using 8 threads and for 100.000.000 iterations. Clearly indicate which Paraver configuration file(s) did you use and your own conclusions from that profile.

Paraver configuration: cfgs / OpenMP / state_profile



	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	95.07 %	-	3.26 %	1.66 %	0.00 %	0.00 %
THREAD 1.1.2	37.85 %	62.09 %	0.06 %	-	0.00 %	-
THREAD 1.1.3	37.89 %	62.09 %	0.02 %	-	0.00 %	-
THREAD 1.1.4	34.46 %	62.05 %	3.48 %	-	0.00 %	-
THREAD 1.1.5	37.85 %	62.09 %	0.06 %	-	0.00 %	-
THREAD 1.1.6	37.82 %	62.10 %	0.07 %	-	0.00 %	-
THREAD 1.1.7	37.84 %	62.13 %	0.03 %	-	0.00 %	-
THREAD 1.1.8	37.85 %	62.10 %	0.05 %	-	0.00 %	-
Total	356.64 %	434.65 %	7.03 %	1.66 %	0.02 %	0.00 %
Average	44.58 %	62.09 %	0.88 %	1.66 %	0.00 %	0.00 %
Maximum	95.07 %	62.13 %	3.48 %	1.66 %	0.00 %	0.00 %
Minimum	34.46 %	62.05 %	0.02 %	1.66 %	0.00 %	0.00 %
StDev	19.12 %	0.02 %	1.44 %	0 %	0.00 %	0 %
Avg/Max	0.47	1.00	0.25	1	0.50	1