PAR Laboratory Deliverable

Course 2019/20 (Fall semester)

Lab 1: Experimental setup and tools

Rafel-Albert Bros Esqueu, Joan Vinyals Ylla-Català

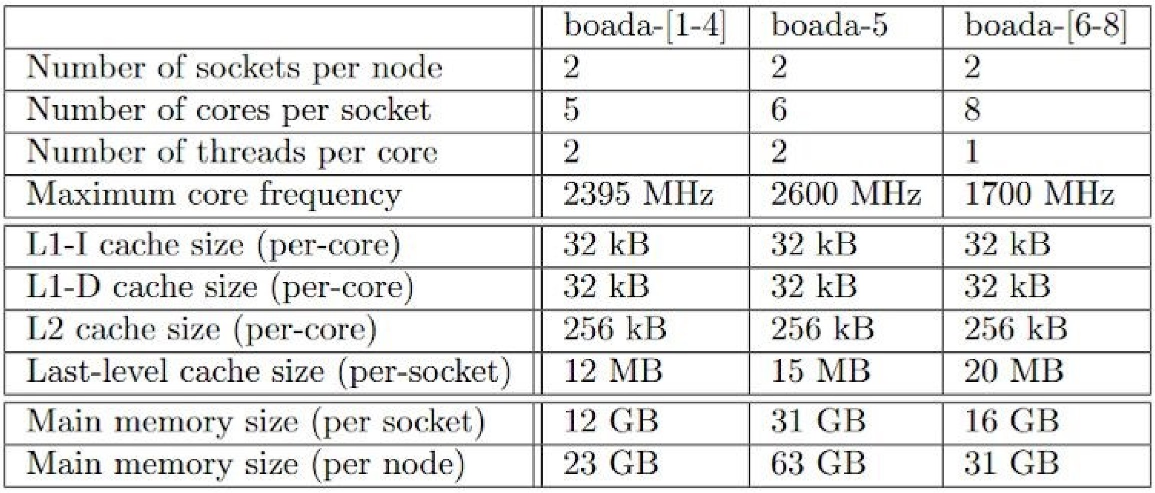
User: Par2201

October, 2019

1 Node architecture and memory

In order to explore the concepts explained in the theory lectures, we are using the **boada** server, which is an environment very similar to those found in the professional world.

Boada has three types of nodes, each one with its particular properties, as seen in the *Table 1*.



	boada-[1-4]	boada-5	boada-[6-8]
Number of sockets per node	2	2	2
Number of cores per socket	5	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)	32 kB	32 kB	32 kB
L1-D cache size (per-core)	32 kB	32 kB	32 kB
L2 cache size (per-core)	256 kB	256 kB	256 kB
Last-level cache size (per-socket)	12 MB	15 MB	20 MB
Main memory size (per socket)	12 GB	31 GB	16 GB
Main memory size (per node)	23 GB	63 GB	31 GB

Table 1: Relevant architectural characteristics of the different node types available in **boada** server.

For the sake of these practices, we are mainly using **boada** nodes from one to four, which have a node topology shown in *Figure 1*.

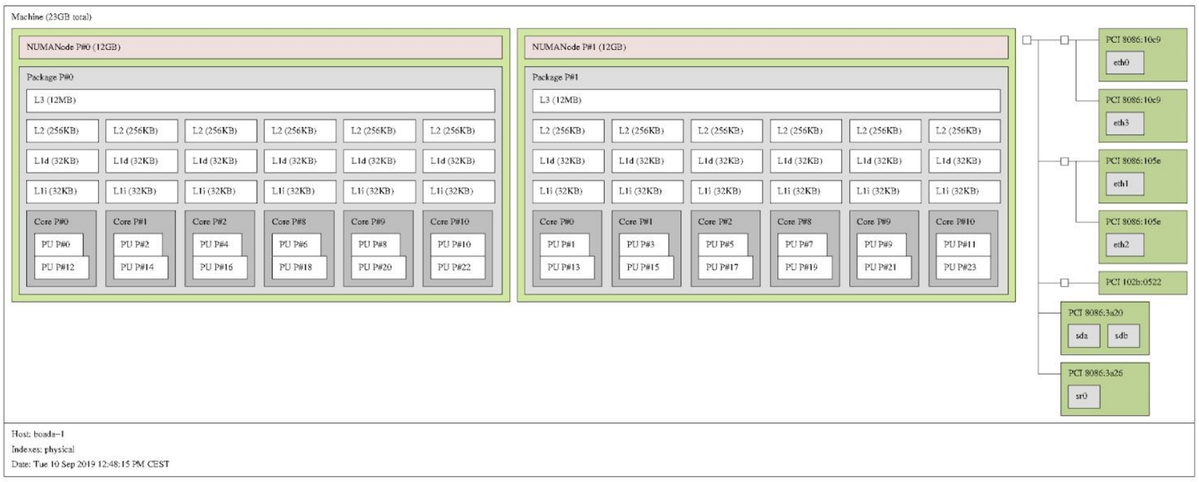


Figure 1: Architectural diagram of **boada-1**.

2 Strong vs. weak scalability

Being aware that scalability is the way a system reacts when increasing the amount of resources available in it, we acknowledge two different approaches to analyze this behaviour: one titled *strong*; the other one named *weak*.

On the one hand having a system handling a particular problem, adding more resources to the system could result in that problem getting finished faster proportionally to the added resources, being that an excellent example of ideal strong scalability.

On the other hand, besides expanding the number of resources, we also could be making our problem harder to solve, therefore ideally getting a constant execution time. That means the problem size increase is proportional to that of the added resources.

2.1 Scalability of pi_omp

One relevant metric to measure the performance of a program is the relationship between the execution time versus the numbers of threads used when executing a particular problem.

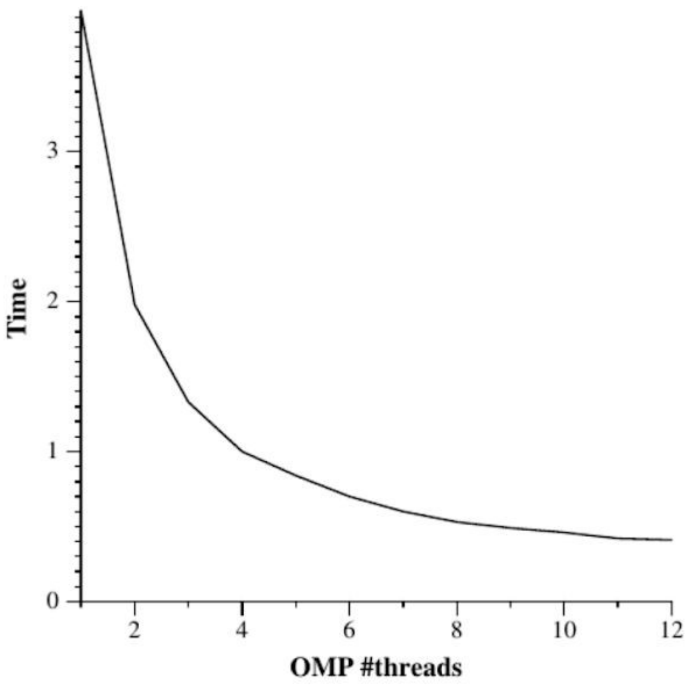


Figure 2: Execution time of pi_omp for different amounts of threads.

That data can be seen in *Figure 2*, where we observed that the execution time is apparently decreasing logarithmically as the number of threads are increased.

Notice how this means that theoretically there is a point where adding more threads do no longer imply a reduction of the execution time.

Another good method to analyze the scalability of a parallel program is to employ speedup altogether with the number of threads as reference.

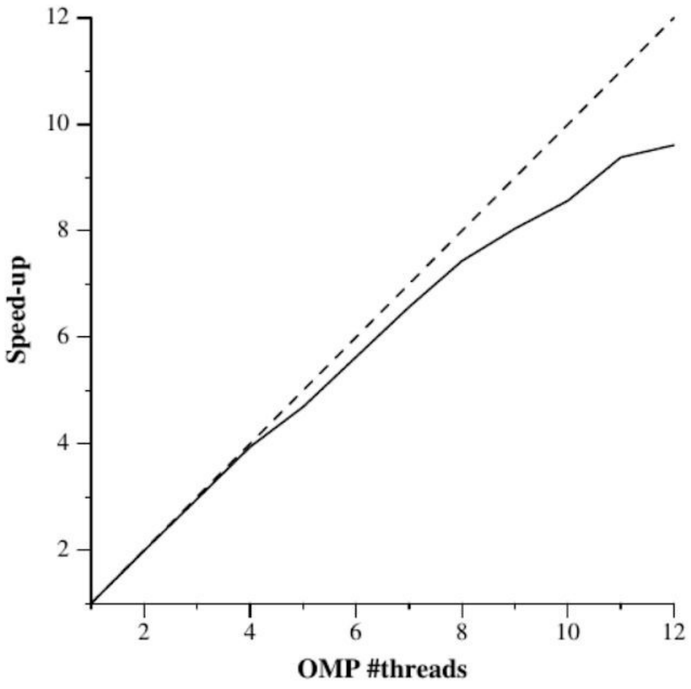


Figure 3: Speedup of pi_omp when increasing the number of threads.

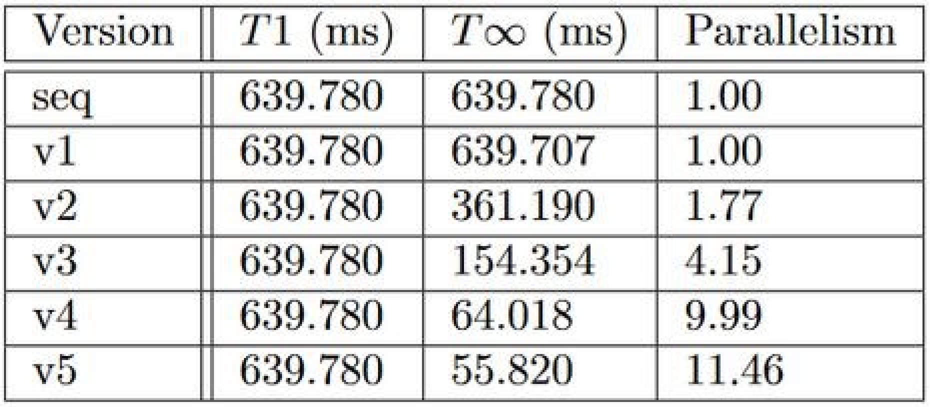
In *Figure 3* we observed how faster are different executions in parallel versus a sequential execution (ergo using only one thread). Pay special attention to the fact that the speedup does not rise linearly with the number of threads.

Furthermore, the bigger the amount of threads the slower the speedup grows. Therefore, the possibility of a situation where an increase in the amount of threads implies a fall in the speedup values can be considered.

3 Analysis of task decompositions for *3DFFT*

As seen in *Table 2,* for the execution of the first parallel version of *3DFFT* there is almost no parallelization, consequently the T∞ value is practically the same that in the sequential version of the program.

Nonetheless, with the second parallel version of *3DFFT* we already notice some difference, and up to the fifth version there is some noticeable increase in the calculated parallelism index.



Version	$T1$ (ms)	$T\infty$ (ms)	Parallelism
seq	639.780	639.780	1.00
v1	639.780	639.707	1.00
v2	639.780	361.190	1.77
v3	639.780	154.354	4.15
v4	639.780	64.018	9.99
v5	639.780	55.820	11.46

Table 2: Parallelism for each *3DFFT* version.

3.1 Difference between v4 and v5

The variation within *v4* versus *v5* can be explained by looking into the codes of both versions. Rather than having the structure seen in *Code 1* like v4 does, with the *tareador* task inside the first loop of the function, *v5* keeps the *tareador* task inside the second loop, hence the structure shown in *Code 2*.

```
for (k = 0; k < N; k++) {
      tareador_start_task("init_complex_grid_loop_k");
      for (j = 0; j < N; j++) {
      [...]
      }
      tareador_end_task("init_complex_grid_loop_k");
}
```
Code 1: Fragment of function *init_complex_grid* from file *3dfft_v4_tar.c*.

```
for (k = 0; k < N; k++) {
     for (j = 0; j < N; j++) {
     tareador_start_task("init_complex_grid_loop_j");
     [...]
     tareador_end_task("init_complex_grid_loop_j");
     }
}
```
Code 2: Fragment of function *init_complex_grid* from file *3dfft_v5_tar.c*.

Additionally, *Figure 4* and *Figure 5* show the dependence graphs of both versions. Note that as a result of the modification of the code in *v5* the dependency graph is more complex.

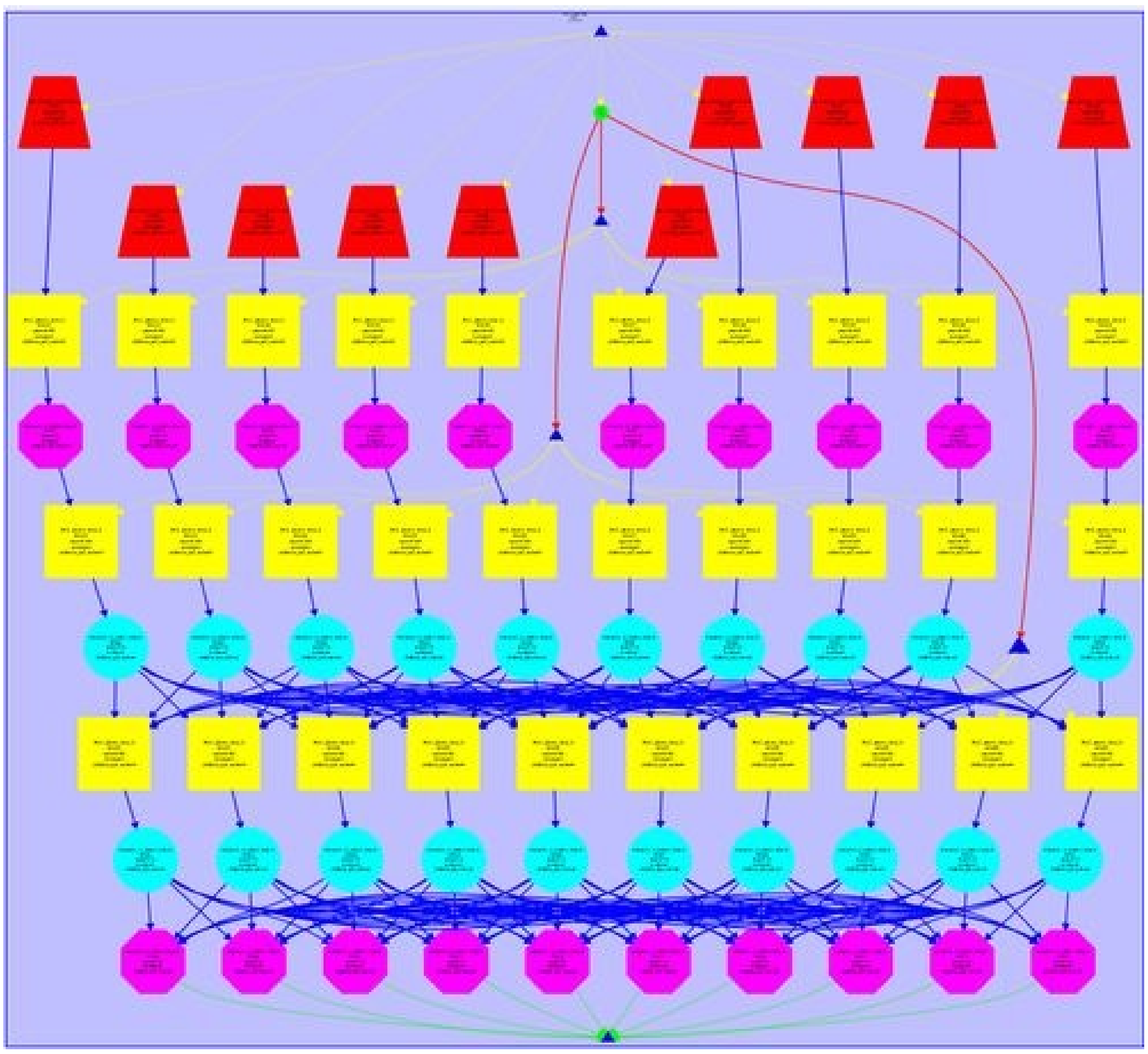


Figure 4: Dependence graph for *3dfft_v4_tar.c*.

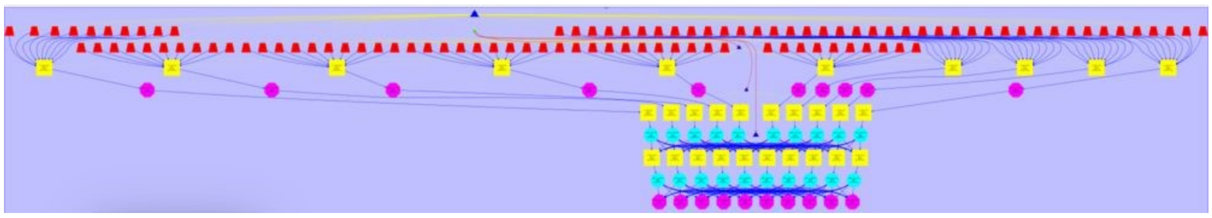


Figure 5: Dependence graph for *3dfft_v5_tar.c*.

4 Understanding the parallel execution of *3DFFT*

Working with the code of *3dfft_omp.c* we took three different approaches with all the data to be found in *Table 3*. In the first place we simply obtained metrics from the initial parallel version of the program.

Secondly, by uncommenting some pragmas already in the code we allowed the parallel execution of function *init_complex_grid*, therefore getting an improved ϕ.

Finally, we increased the granularity of tasks exclusively by reducing parallelisation overheads and in theory getting a better performance of the execution.

Version	ϕ	$S\infty$	T1 (ms)	T8 (ms)	S8
initial version in 3dfft_omp.c	0.65	2.89	2362.43	1415.68	1.67
new version with improved ϕ	0.90	10.40	2403.420	1066.320	2.25
final version with reduced parallelisation overheads	0.90	10.71	2308.086	836.888	2.76

Table 3: Results of the parallel performance evolution for the OpenMP parallel versions of 3DFFT.