UPC: Facultad de Informática de Barcelona

Lab 1: Experimental setup and tools

PAR Laboratory assignment

Oriol Moyano Núñez
Martí Ferret Vivó

ID: par2108

12/03/2018

Index

Exercises

Note architecture and memory

1. Complete the following table with the relevant architectural characteristics of the different node types available in boada:

Architecture and memory of the different nodes in the boada machine			
	boada-1 to boada-4	**boada-5**	**boada-6 to boada-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	2.395 GHz	2.60 GHz	1.7 GHz
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)	12288 KB (12 MB)	15360 KB (15MB)	20480 KB (20 MB)
Main memory size (per socket)	23 GB	63 GB	31 GB
Main memory size (per node)	12 GB	31 GB	16 GB

2. Include in the document the architectural diagram for one of the nodes boada-1 to boada-4 as obtained when using the lstopo command.

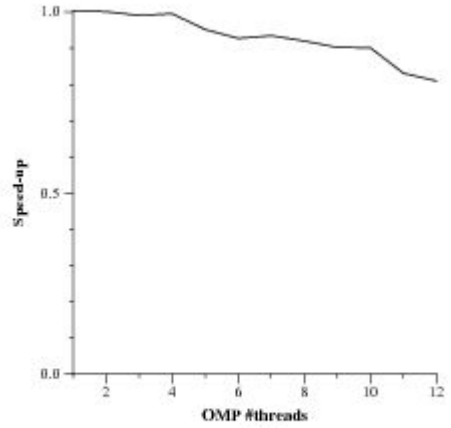
Diagram of boada-1 node (CPU-wise):

Timing sequential and parallel executions

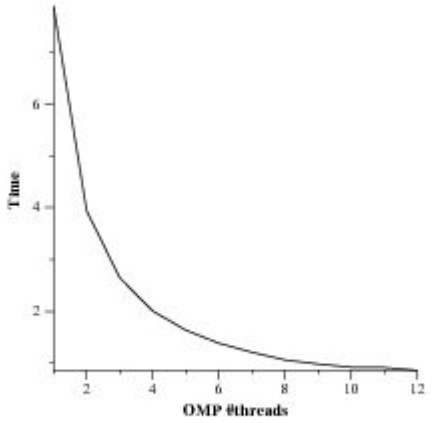
3. Plot the execution time and speed–up that is obtained when varying the number of threads (strong scalability) and problem size (weak scalability) for pi omp.c on the different node types available in boada. Reason about the results that are obtained.

boada-1 to boada-4:

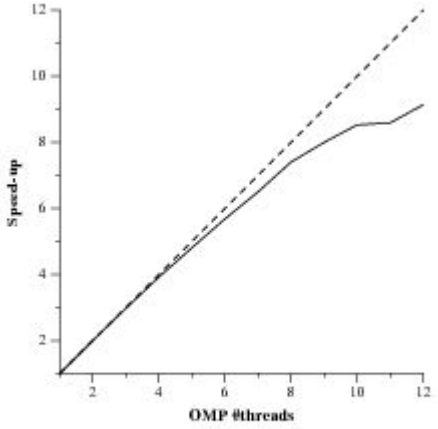
Weak	Strong



par2108
Speed-up wrt one thread, weak scaling
Fri Feb 23 12:23:47 CET 2018

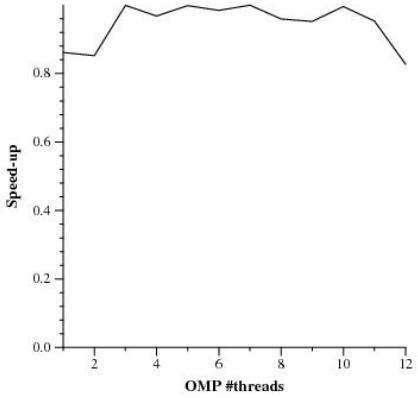
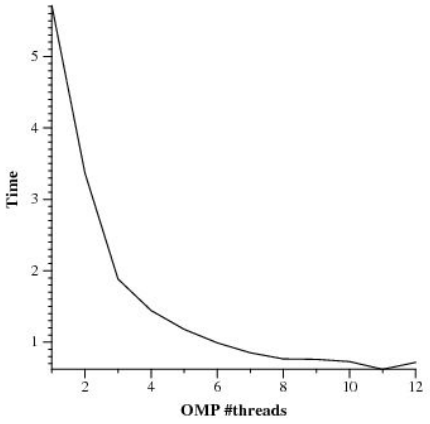
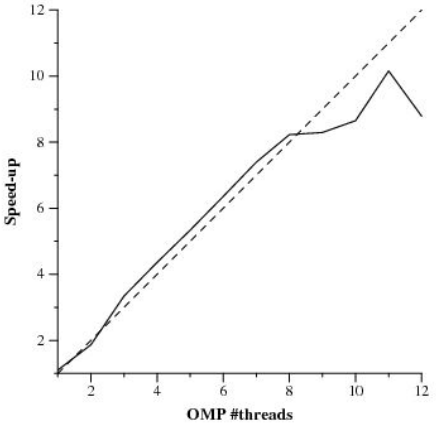


par2108
Min elapsed execution time
Fri Feb 23 12:24:38 CET 2018



par2108
Speed-up wrt sequential time
Fri Feb 23 12:24:38 CET 2018

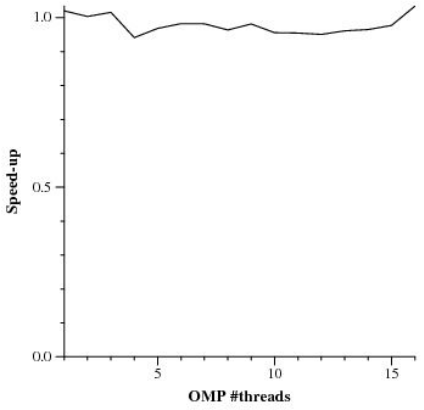
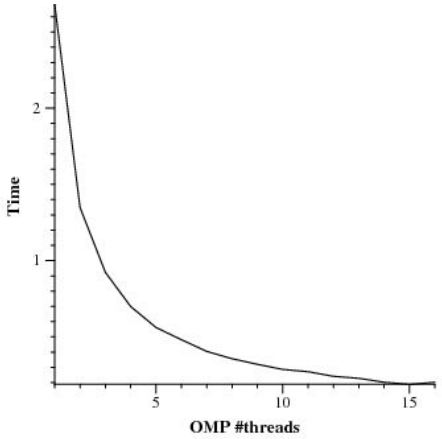
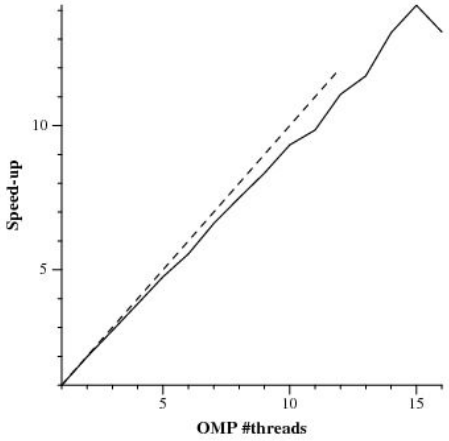
boada-5:

Weak	Strong
 par2108 Speed-up wrt one thread, weak scaling Sun Feb 25 19:46:04 CET 2018	 par2108 Min elapsed execution time Sun Feb 25 19:32:57 CET 2018  par2108 Speed-up wrt sequential time Sun Feb 25 19:32:57 CET 2018

In this note, a comment on the difference found on boada-5 node, more explanations related to all nodes below:

We can see in the strong speed-up chart that the speed-up from 3 to 8 threads is greater than the theoretical maximum (it shouldn't be possible to have a speedup of 3.x with only 3 processors) which means that we are in front of a superlinear speedup. This effect may be possible due to a wide variety of reasons such as cache usage or hits on memories.

boada-6 to boada-8:

Weak	Strong
 par2108 Speed-up wrt one thread, weak scaling Sun Feb 25 20:07:25 CET 2018	 par2108 Min elapsed execution time Sun Feb 25 20:07:32 CET 2018  par2108 Speed-up wrt sequential time Sun Feb 25 20:07:32 CET 2018

We'll do the reasoning of the 3 cases together because the differences between them are minimal (in terms of scalability). The best way to analyze our results is by dividing them in two sections:

- Strong scalability: the plots show what was expected, more threads mean a decrease in the execution time of the program. As for the speedup, we can see a little deviation from the ideal line (speedup/#threads) likely produced by overheads such as the ones from task creation or synchronization.
- Weak scalability: we can extract from the graphics that the speedup is close to 1 when the size of the problem and threads increases, which means that the problem is scalable (if that wasn't the case, even if we used more threads, the problem wouldn't have accepted more parallelism). It is also possible to see a gradual decrement of the speedup as we use more threads (thus increasing the size of the problem), this effect could be produced by a myriad of reasons: the different overheads, the increase in the sequential part of the problem, the access to the shared memory...

Analysis of task with Tareador

4. Include the relevant(s) part(s) of the code to show the new task definition(s) in v4 of 3dfft seq.c. Capture the final task dependence graph that has been obtained after version v4.

Relevant code changes:

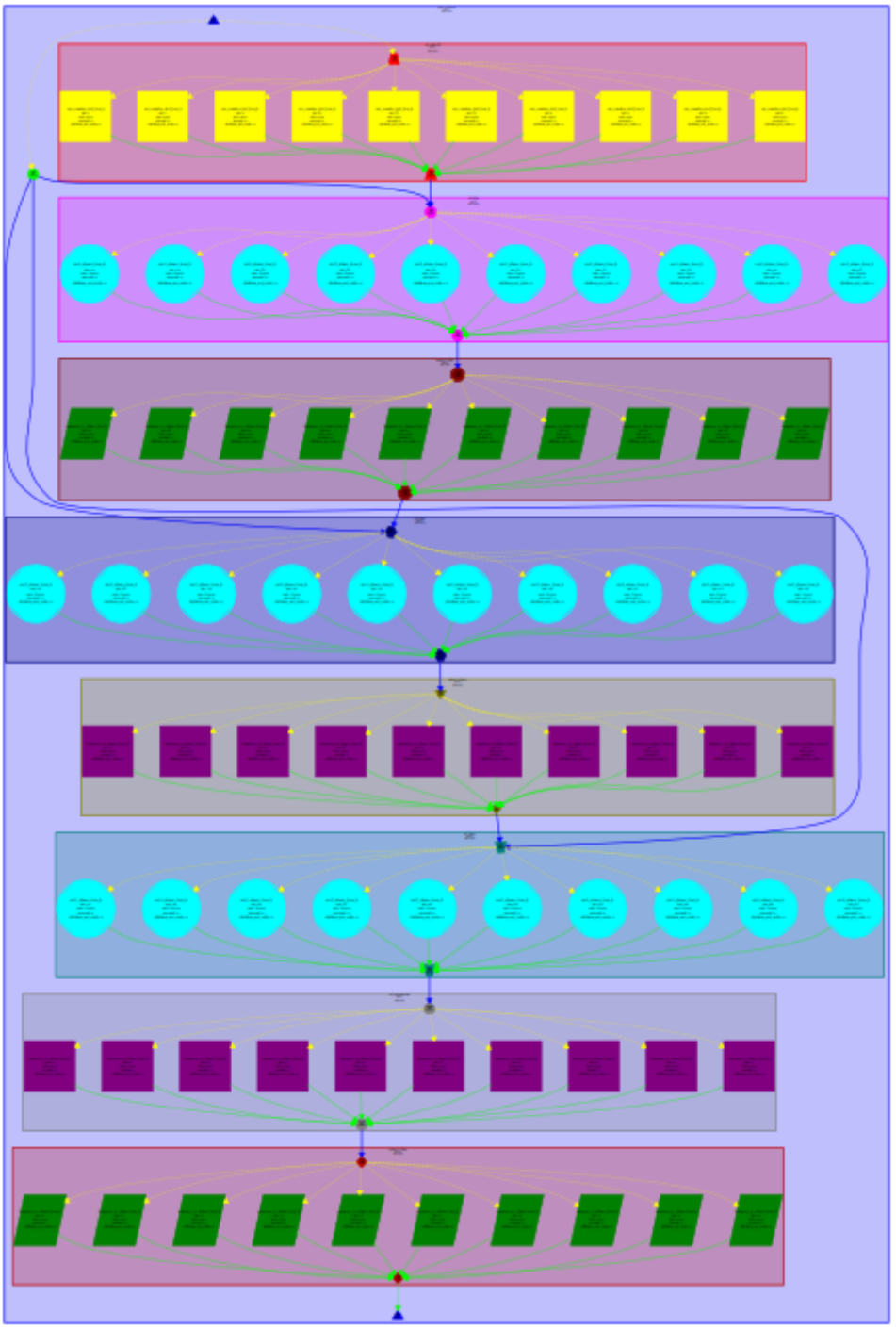
```
 void init_complex_grid(fftwf_complex in_fftw[][N][N])
{
  int k,j,i;

  for (k = 0; k < N; k++) {
    tareador_start_task("init_complex_grid_loop_k");
    for (j = 0; j < N; j++)
      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)));
        in_fftw[k][j][i][1] = 0;
#if TEST
        out_fftw[k][j][i][0]= in_fftw[k][j][i][0];
        out_fftw[k][j][i][1]= in_fftw[k][j][i][1];
#endif
      }
       tareador_end_task("init_complex_grid_loop_k");
  }
}
```

We have only modified the init function, because decreasing more the size of other tasks would lead to more time being used by task creation and synchronization (among other overheads). That would render useless further changes and decrease efficiency, which we assumed is not the goal in this exercise.

**We know, though, that we could augment the granularity creating separate tasks for each of the inner loops.*

The dependence graph obtained with our modifications is the one below:



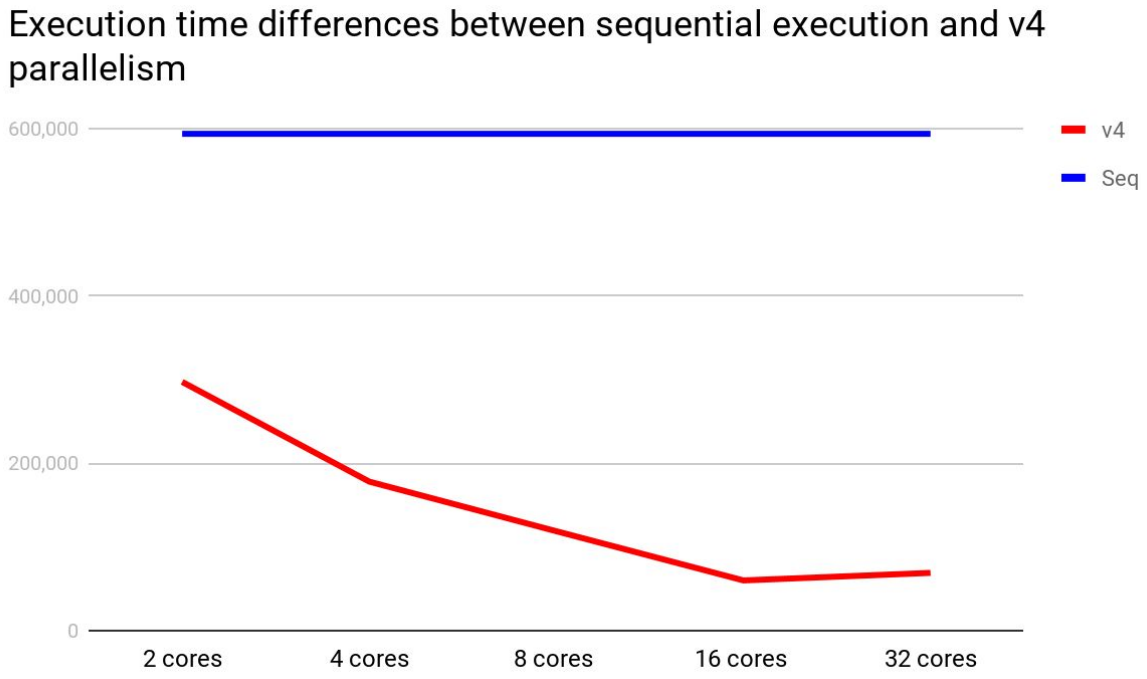
5. Complete the following table for the initial and different versions generated for 3dfft_seq.c, briefly commenting the evolution of the metrics with the different versions.

Version	T_1 /ms/	T_∞ /ms/	Parallelism
seq	**593,772**	**593,772**	**1**
v1	**593,772**	**592,726**	**1**
v2	**593,772**	**315,437**	**1.88**
v3	**593,772**	**108,937**	**5.45**
v4	**593,772**	**60,012**	**9.89**

There's not much to say, with each improvement the potential and real parallelism grows, thus enabling the program to run faster. The changes between versions was focused on increasing the parallel fraction of the program (making the sequential shorter) to get those results (it could also be said the the granularity of the tasks grew on each version).

6. 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.

Number of processors	Execution time	Speed-up
2	**297,255ms**	**1,99**
4	**178,080ms**	**3.33**
8	**118,790ms**	**4.99**
16	**60,012ms**	**9.89**
32	**69,012ms**	**8.60**



Those plots show that the program strong scalability is relatively close to the theoretical line (perfect speedup with a specific number of cores) with few cores. However, as the number of threads grows, the speedup deviates from this perfect line until past 16 cores. At this point, probably caused by parallelism overheads, synchronizations or other problems, the values for the speedup start to decrease. We are not able to discuss the weak scalability properties of the program because the size of the program is kept constant in our plots.

Tracing the execution of parallel programs

7. From the analysis with Paraver that you have done for the complete parallelization of 3dfft omp.c, explain how have you computed the value for φ, the parallel fraction of the application. Please, include any Paraver timeline that may help to understand how you have performed the computation of φ.

We've computed the parallel fraction of the program using the data given by paraver; sequential time can be found in Figure 1 (values may differ from our computation due to using screenshots on another computer), the same as the total execution time. From there we can easily calculate the parallel fraction:

- $Sequential\ time\ =\ 3.575.494.729\ ns$
- $Total\ execution\ time\ =\ 4.916.650.419\ ns$

$$100\%\ =\ Sequential\ fraction\ +\ Parallel\ fraction$$

$$Parallel\ fraction\ =\ 100\%\ -\ Sequential\ fraction$$

$$Parallel\ fraction\ =\ 100\%\ -\ (3,576/4,917) * 100\ = 27,28\%$$

$$S_{\infty} = \frac{1}{1-0,2728} = 1.375$$

Figure 1. Exercise 7 reference on parallel and sequential times of pi_omp.

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	4,834,065,075 ns	-	89,312,273 ns	2,789,645 ns	690,727 ns	2,586 ns
THREAD 1.1.2	1,201,050,661 ns	3,455,711,432 ns	147,179,279 ns	122,402,857 ns	516,077 ns	-
THREAD 1.1.3	1,230,295,350 ns	3,455,771,170 ns	117,236,289 ns	122,898,500 ns	658,997 ns	-
THREAD 1.1.4	1,278,710,621 ns	3,455,733,573 ns	69,519,540 ns	122,328,990 ns	567,582 ns	-
Total	8,544,121,707 ns	10,367,216,175 ns	423,247,381 ns	370,419,992 ns	2,433,383 ns	2,586 ns
Average	2,136,030,426.75 ns	3,455,738,725 ns	105,811,845.25 ns	92,604,998 ns	608,345.75 ns	2,586 ns
Maximum	4,834,065,075 ns	3,455,771,170 ns	147,179,279 ns	122,898,500 ns	690,727 ns	2,586 ns
Minimum	1,201,050,661 ns	3,455,711,432 ns	69,519,540 ns	2,789,645 ns	516,077 ns	2,586 ns
StDev	1,557,957,910.54 ns	24,658.53 ns	29,288,008.62 ns	51,855,380.63 ns	69,870.28 ns	0 ns
Avg/Max	0.44	1.00	0.72	0.75	0.88	1

Figure 2. Exercise 7 second reference.

8. Show and comment the profile of the % of time spent in the different OpenMP states for the complete parallelization of 3dfft omp.c on 4 threads.

- $Sequential\ time\ =\ 262.712.971\ ns$
- $Total\ execution\ time\ =\ 2.790.015.447\ ns$

$$100\%\ =\ Sequential\ fraction\ +\ Parallel\ fraction$$

$$Parallel\ fraction\ =\ 100\%\ -\ Sequential\ fraction$$

$$Parallel\ fraction\ =\ 100\%\ -\ (0.2627/2.790) * 100\ = 90.58\%$$

$$S_{\infty} = \frac{1}{1-0,2728} = 1.375$$

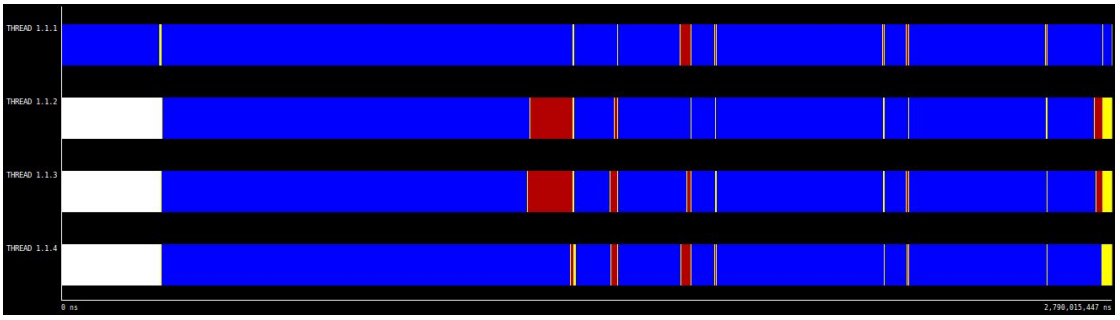


Figure 3. Exercise 8 reference on parallel and sequential times of 3dfft_omp.c.

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	98.12 %	-	1.81 %	0.06 %	0.01 %	0.00 %
THREAD 1.1.2	24.38 %	70.14 %	2.99 %	2.48 %	0.01 %	-
THREAD 1.1.3	24.97 %	70.14 %	2.38 %	2.49 %	0.01 %	-
THREAD 1.1.4	25.95 %	70.14 %	1.41 %	2.48 %	0.01 %	-
Total	173.42 %	210.42 %	8.59 %	7.52 %	0.05 %	0.00 %
Average	43.35 %	70.14 %	2.15 %	1.88 %	0.01 %	0.00 %
Maximum	98.12 %	70.14 %	2.99 %	2.49 %	0.01 %	0.00 %
Minimum	24.38 %	70.14 %	1.41 %	0.06 %	0.01 %	0.00 %
StDev	31.62 %	0.00 %	0.59 %	1.05 %	0.00 %	0 %
Avg/Max	0.44	1.00	0.72	0.75	0.88	1

Figure 4. Exercise 8 plot showing the time percentages of the different omp tasks on 3dfft_omp.c.

We can see in the table that In the first section of the program (while the parallel tasks are not created yet) the program only uses 1 core, then creates and executes 4 threads. That's why CPUs 1 to 3 runs less than CPU0 (which does the sequential portion of the program too). As for the time dedicated to overheads caused by the parallelization, the values are little in comparison to the total execution time (as a side note, synchronization and task creation times have similar percentages).