Alejandro Gallego Rodríguez par1203 03/10/2019

Oriol Catalán Fernandez

Deliverable Lab 1

Node architecture and memory

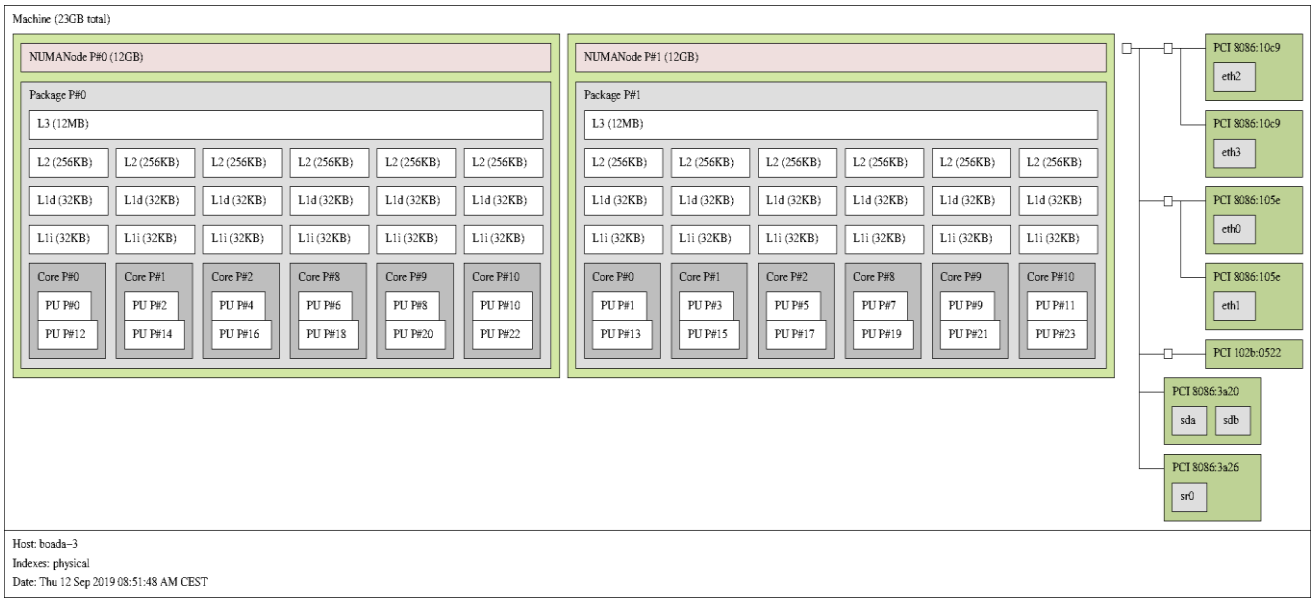
Describe the architecture of the boada server. To accompany your description, you should refer to the following table summarizing the relevant architectural characteristics of the different node types available:

Boada is divided into 8 different nodes, **boada-1** to **boada-8** equipped with 3 processor generations. **boada-1** to **boada-4** has **Intel Xeon E5645**, **boada-5** has **Intel Xeon E5-2620 v2 + Nvidia K40c** and **boada-6** to **boada-8** has **Intel Xeon E5-2609 v4.**

	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	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)	12288 KB	15360 KB	20480 KB
Main memory size (per socket)	23 GB	63 GB	31 GB
Main memory size (per node)	12 GB	31 GB	16 GB

Also include in the description the architectural diagram for one of the nodes boada-1 to boada-4:

Showing down below an image of the architectural design of **boada-3**, which is the exact same of **boada-1, boada-2** and **boada-4:**



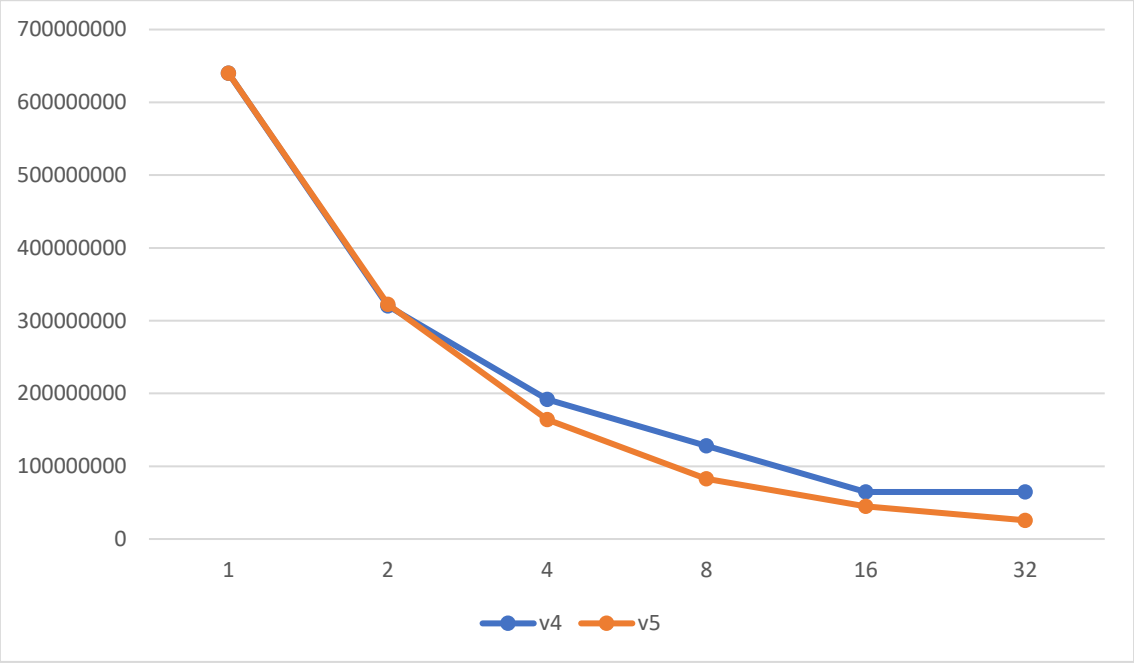
The main memory size of **boada-3** is 23 GB, divided into 2 nodes called: **NUMANode P#0** and **NUMANode P#1** both sized 12 GB each. Inside each node, there's a low-level cache named **L3** with 12 MB and 6 stacks. Each stack has caches named **L2, L1-D, L1-I** with 256, 32 and 32 KB, respectively, and a core.

Analysis of task decompositions for 3DFFT

In this part of the report you should summarize the main conclusions from the analysis of task decompositions for the 3DFFT program. Backup your conclusions with the following table properly filled in with the information obtained in the laboratory session for the initial and different versions generated for 3dfft_tar.c, briefly commenting the evolution of the metrics:

Version	T_1	T_∞	Parallelism
Tseq	639780001	639780001	1
v1	639780001	639780001	1
v2	639780001	361439001	1.77
v3	639780001	154939001	4.13
v4	639780001	64614001	9.9
v5	639780001	13781001	46.42

At the time we were improving granularity in our program the execution time was decreasing. That is because we were adding different parts to be executed in parallel in every version of the program. At one point, in v5, in order of accomplish the greater granularity we made every i loop parallel, figuring out that the excessive parallelism was not improving the execution time, furthermore, we saw that making the j loops parallel was more efficient that the i loops.

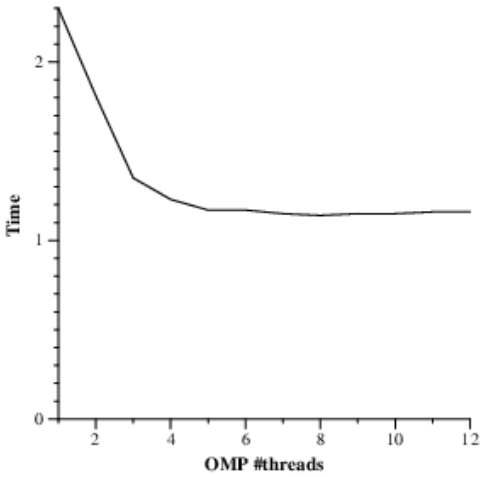


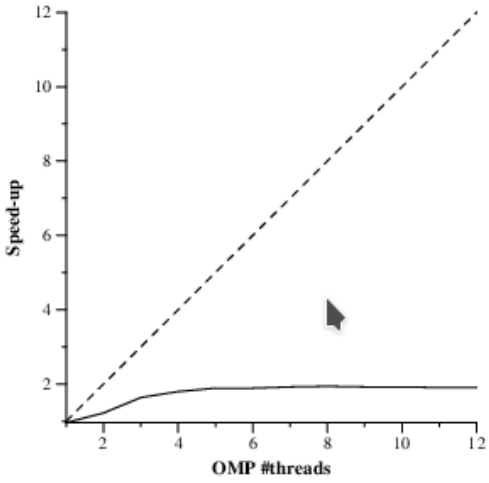
Being x the number of processors and y the execution time in ms.

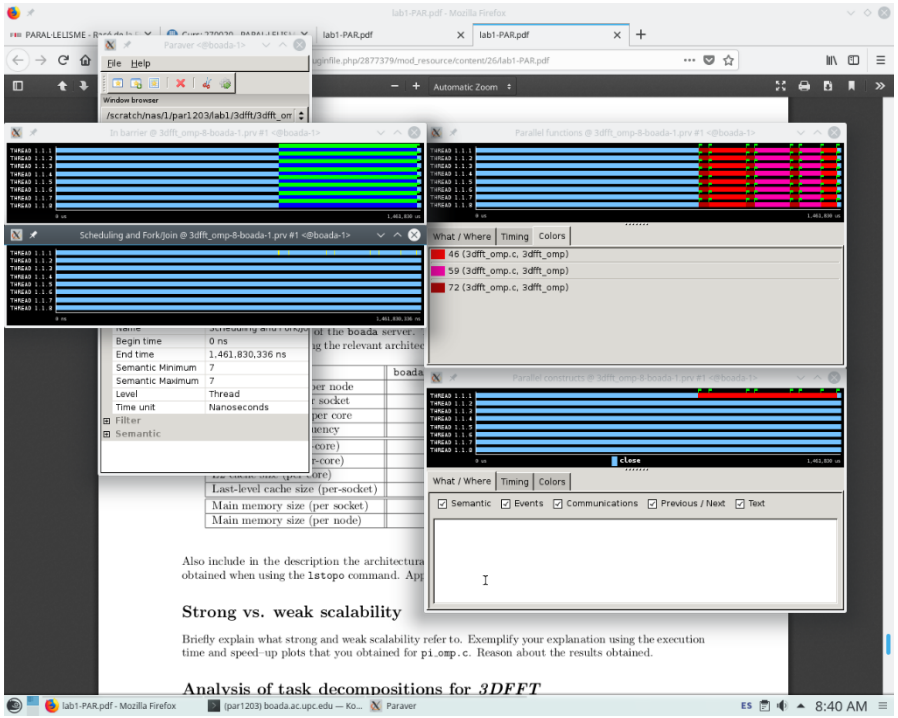
Understanding the parallel execution of 3DFFT

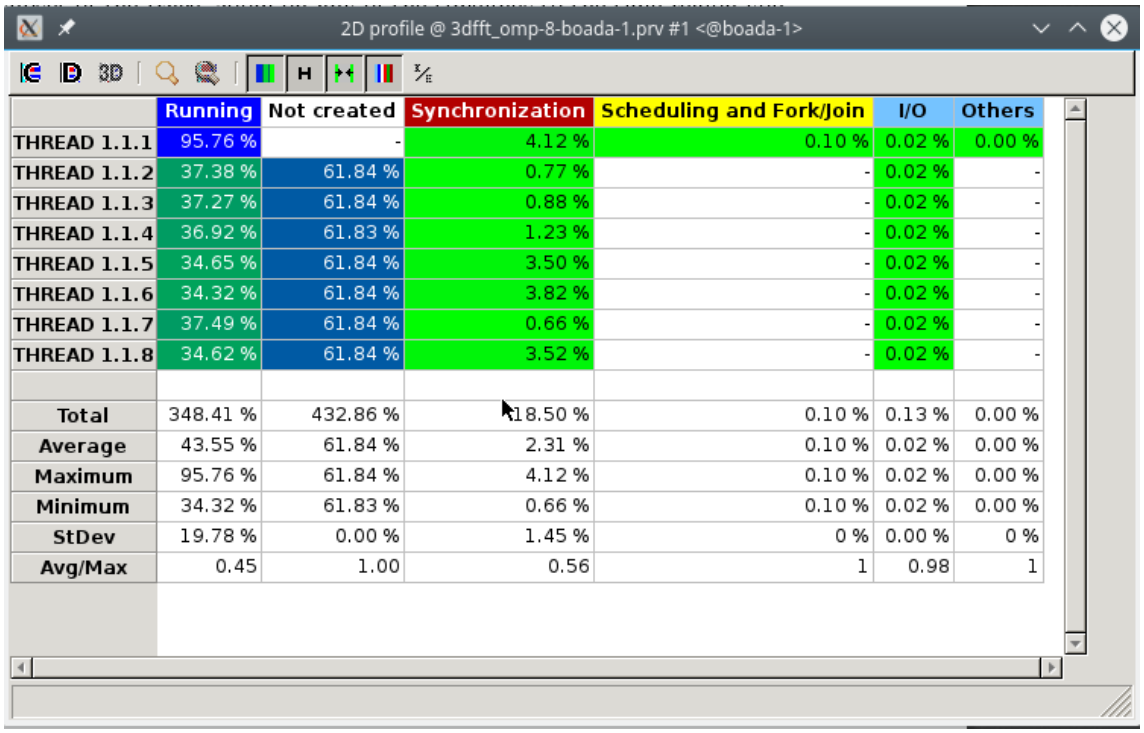
In this final section of your report you should comment about how did you observed with Paraver the parallel performance evolution for the OpenMP parallel versions of3DFFT. Support your explanations with the results reported in the following table which you obtained during the laboratory session. It is very important that you include the relevant Paraver captures (timelines and profiles of the % of time spent in the different OpenMP states) to support your explanations too.

Version	ø	S_∞	T_1 (ns)	T_8 (ns)	S_8
Initial version in 3dfft_omp.c	0.8328	1.91	54.485.445.383	1.461.830.336	37.27
New version with improved ø	0.90561	3.65	6.620.697.567	1.374.396.899	4.81
Final version with reduced parallelization overheads	1.01704	5.54	3.414.796.861	618.878.974	5.52









	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	95.76 %	-	4.12 %	0.10 %	0.02 %	0.00 %
THREAD 1.1.2	37.38 %	61.84 %	0.77 %	-	0.02 %	-
THREAD 1.1.3	37.27 %	61.84 %	0.88 %	-	0.02 %	-
THREAD 1.1.4	36.92 %	61.83 %	1.23 %	-	0.02 %	-
THREAD 1.1.5	34.65 %	61.84 %	3.50 %	-	0.02 %	-
THREAD 1.1.6	34.32 %	61.84 %	3.82 %	-	0.02 %	-
THREAD 1.1.7	37.49 %	61.84 %	0.66 %	-	0.02 %	-
THREAD 1.1.8	34.62 %	61.84 %	3.52 %	-	0.02 %	-
Total	348.41 %	432.86 %	18.50 %	0.10 %	0.13 %	0.00 %
Average	43.55 %	61.84 %	2.31 %	0.10 %	0.02 %	0.00 %
Maximum	95.76 %	61.84 %	4.12 %	0.10 %	0.02 %	0.00 %
Minimum	34.32 %	61.83 %	0.66 %	0.10 %	0.02 %	0.00 %
StDev	19.78 %	0.00 %	1.45 %	0 %	0.00 %	0 %
Avg/Max	0.45	1.00	0.56	1	0.98	1

As we can see in the table, the execution times are decreasing considerably as the granularity increases, the reason is by parallelizing the program, it increasingly uses the 8 threads it has much better. In particular, it decreases a lot more time with 1 thread than with 8. Also the speedup is affected by those changes. The graphic material is, in particular, for the first example; we can see that the relationship with the ideal speedup is quite bad and that the time graph stagnates very quickly at a fixed time. We can also observe in the Paraver, that the parallelizable part is quite short with the timelines; in fact, in the time table we see that thread 1 executes most of the program.