First Deliverable

Èric Casanovas

par2110

13/03/2018

Node architecture and memory

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

	Boada 1-4	Boada 5	Boada 6-8
Sockets/node	1	1	1
Cores/socket	6	6	4
Threats/core	2	2	1
Max. Core frequency	2395MHz	2600MHz	1700MHz
L1-I cache size	32K	32K	32K
L1-D cache size	32K	32K	32K
L2 cache size	256K	256K	256K
Last-level cache size	12288K	15360K	20480K
Main memory size (socket)	12GB	768GB	1536GB
Main memory size (node)	12GB	768GB	1536GB

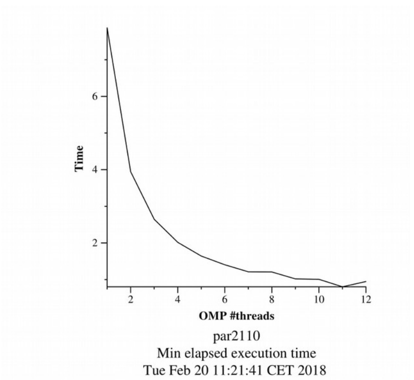
2. Include in the document the architectural diagram for one of the nodes boada-1 to boada-4 as obtained when using the lstopo command.

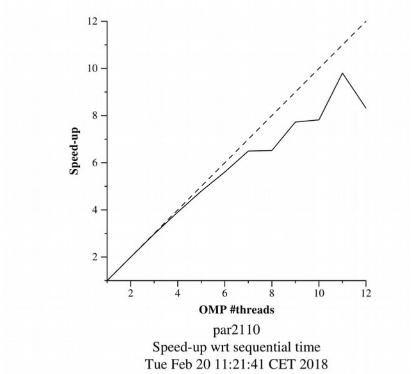
Timing sequential and parallel executions

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.

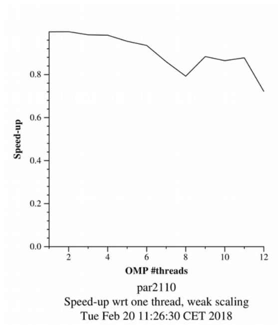
• In strong scalability the number of threads is changed with a fixed problem size. In this case parallelism is used to reduce the execution time of your program.

Strong scalability





Weak scalability



- In weak scalability the problem size is proportional to the number of threads. In this case parallelism is used to increase the problem size for which your program is executed.

Analysis of task decompositions with Tareador

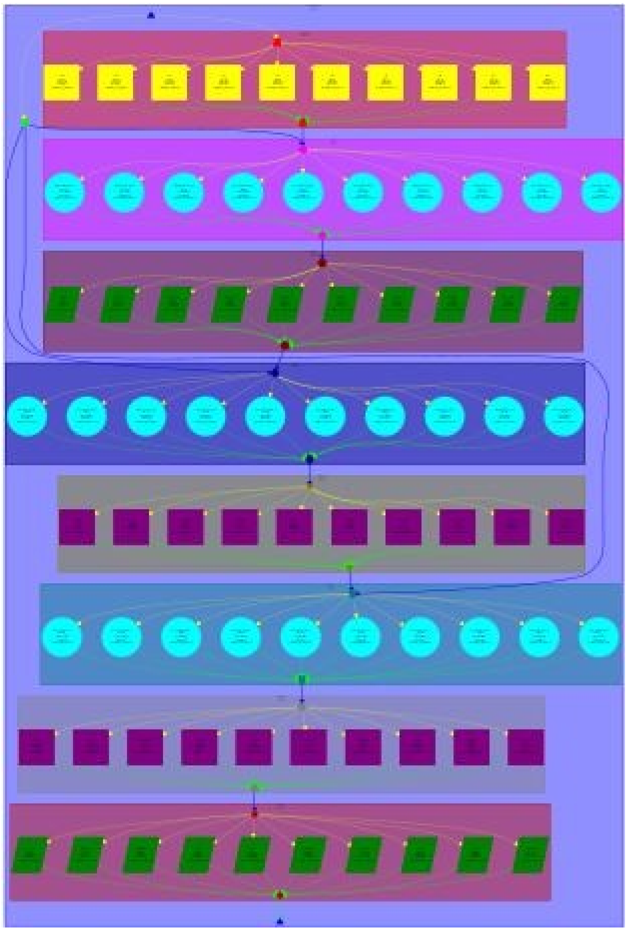
4. Include the relevant(s) part(s) of the code to show the new task definition(s) in v4 of 3dfft seq.c. Capture the final task dependence graph that has been obtained after version v4.

```
void init_complex_grid(fftwf_complex in_fftw[][N][N]) {
   int k,j,i;
   for (k = 0; k < N; k++) {
      tareador_start_task("init"); // Marked code is the relevant part
      for (j = 0; j < N; j++) {
         ...unmodified code
      }
      tareador_end_task("init");
   }
}

void transpose_xy_planes(fftwf_complex  tmp_fftw[][N][N], fftwf_complex in_fftw[][N][N]) {
   int k,j,i;
   for (k=0; k<N; k++) {
      tareador_start_task("xy_planes");
      for (j=0; j<N; j++) {
         ...unmodified code
      }
      tareador_end_task("xy_planes");
   }
}
```

```
void transpose_zx_planes(fftwf_complex in_fftw[][N][N], fftwf_complex tmp_fftw[][N][N]) {
    int k, j, i;
    for (k=0; k<N; k++) {
        tareador_start_task("zx_planes");
        for (j=0; j<N; j++) {
            ...unmodified code
        }
        tareador_end_task("zx_planes");
     }
}

void ffts1_planes(fftwf_plan p1d, fftwf_complex in_fftw[][N][N]) {
    int k,j;
    for (k=0; k<N; k++) {
        tareador_start_task("ffts1_planes_loop_k");
            for (j=0; j<N; j++) {
                ...unmodified code
             }
        tareador_end_task("ffts1_planes_loop_k");
    }
}
```

And in main:

```
    tareador_start_task("ffts1_1");
    ffts1_planes(p1d, in_fftw);
    tareador_end_task("ffts1_1");

    tareador_start_task("transpositions_1");
    transpose_xy_planes(tmp_fftw, in_fftw);
    tareador_end_task("transpositions_1");

    tareador_start_task("ffts1_2");
    ffts1_planes(p1d, tmp_fftw);
    tareador_end_task("ffts1_2");

    tareador_start_task("transpositions_2");
    transpose_zx_planes(in_fftw, tmp_fftw);
    tareador_end_task("transpositions_2");

    tareador_start_task("ffts1_3");
    ffts1_planes(p1d, in_fftw);
    tareador_end_task("ffts1_3");

    tareador_start_task("transpositions_3");
    transpose_zx_planes(tmp_fftw, in_fftw);
    tareador_end_task("transpositions_3");

    tareador_start_task("transpositions_4");
    transpose_xy_planes(in_fftw, tmp_fftw);
    tareador_end_task("transpositions_4");
```



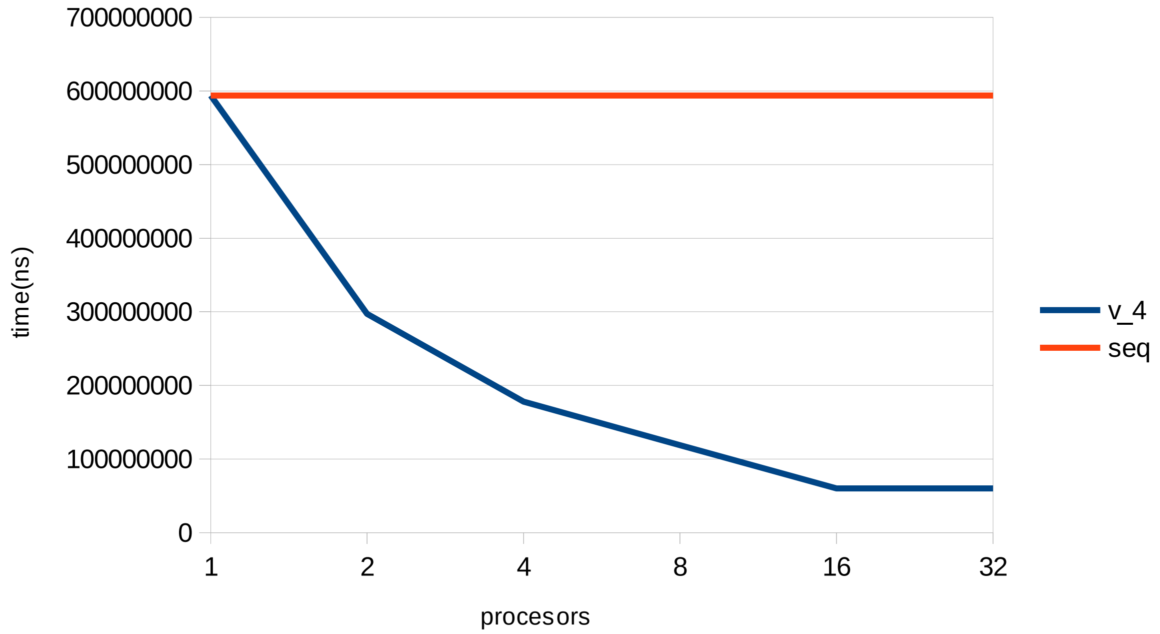
TDG 3dfft_v4

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	T1	T∞	Parallelism
seq	593.772.001	593.772.001	1
v1	593.772.001	593.705.001	1,0001
v2	593.772.001	315.437.001	1,8824
v3	593.772.001	108.937.001	5,4506
v4	593.772.001	60.012.001	9,8942

Time in ns

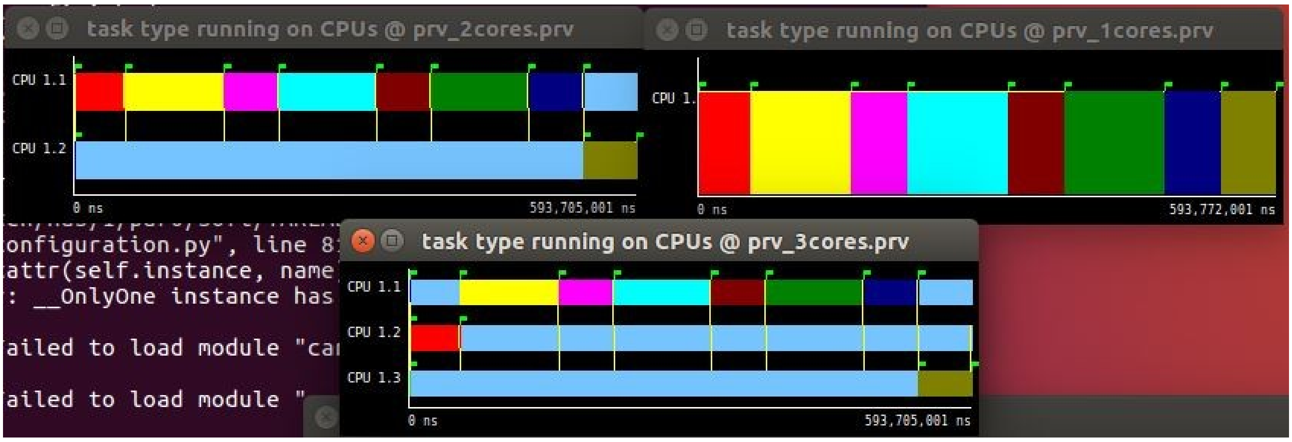
6. With the results from the parallel simulation with 2, 4, 8, 16 and 32 processors, draw the execution time and speedup plots for version v4 with respect to the sequential execution (that you can estimate from the simulation of the initial task decomposition that we provided in 3dfft seq.c, using just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



If we consider the weak scalability we observe that it is decreasing based on having more processors to the point that when we have 16 processors the improvement is insignificant. Regarding the strong scalability, we observe that more or less time will be maintained if we increase the size of the problem proportionally to the number of processors.

Tracing the execution of parallel programs

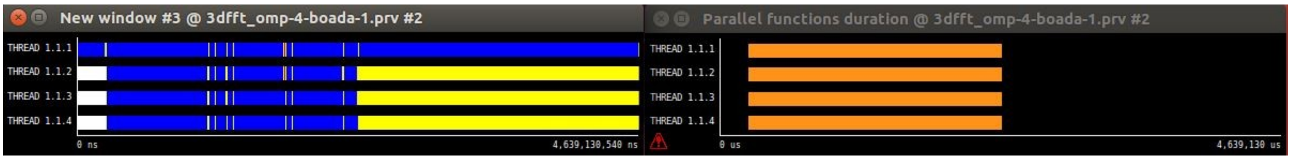
7. From the analysis with Paraver that you have done for the complete parallelization of 3dfft omp.c, explain how have you computed the value for φ, the parallel fraction of the application. Please, include any Paraver timeline that may help to understand how you have performed the computation of φ.



On the graph above on the right we have 1 core (sequential execution) and it took 593,772,001 ns, on the others we executed the program with 2 and 3 cores and it took 593,705,001 ns. This tells us that if we put more than 2 processors it will not improve the execution time.

So φ = (593.772.001-593.705.001)/ 593.772.001 = 1,13 x 10^{-4}

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.



	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	4,614,541,285 ns	-	20,913,706 ns	2,969,915 ns	703,034 ns	2,600 ns
THREAD 1.1.2	2,032,099,186 ns	235,514,419 ns	54,612,734 ns	2,316,214,342 ns	689,859 ns	-
THREAD 1.1.3	2,056,063,369 ns	231,537,554 ns	35,300,991 ns	2,315,549,133 ns	679,493 ns	-
THREAD 1.1.4	2,074,703,308 ns	231,532,054 ns	16,746,929 ns	2,315,520,330 ns	627,919 ns	-
Total	10,777,407,148 ns	698,584,027 ns	127,574,360 ns	6,950,253,720 ns	2,700,305 ns	2,600 ns
Average	2,694,351,787 ns	232,861,342.33 ns	31,893,590 ns	1,737,563,430 ns	675,076.25 ns	2,600 ns
Maximum	4,614,541,285 ns	235,514,419 ns	54,612,734 ns	2,316,214,342 ns	703,034 ns	2,600 ns
Minimum	2,032,099,186 ns	231,532,054 ns	16,746,929 ns	2,969,915 ns	627,919 ns	2,600 ns
StDev	1,108,724,780.75 ns	1,876,009.85 ns	14,813,415.18 ns	1,001,468,071.30 ns	28,475.78 ns	0 ns
Avg/Max	0.58	0.99	0.58	0.75	0.96	1

Running → 58,08%
Not created → 3,76%
Synchronization → 0,69%

Scheduling → 37,45%
I/O → 0,02%
Others → 0% (approx 0)