# Lab 1: Experimental setup and tools

\*\*Username\*\*: par2110

\*\*Semester\*\*: Spring 2017-2018

\*\*Date\*\*: 8/3/2019

\*\*Authors\*\*:

- Maurici Abad Gutierrez

- Nikolaus Schrack

## Node architecture and memory

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

The following architecture will be used to run parallel programs and evaluate the different outcomes on the hardware. Baoda is a cluster with multiple types of nodes. The baoda-1 node is available to the users and the others can be accessed through a queuing system. The table describes the varying specifications of the nodes.

| | 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 | 2395MHz | 2600MHz | 1700MHz |

| L1-I cache size (per-core) | 32KB | 32KB | 32KB |

| L1-D cache size (per-core) | 32KB | 32KB | 32KB |

| L2 cache size (per-core) | 256KB | 256KB | 256KB |

| Last-level cache size (per-socket) | 12MB | 15MB | 20MB |

| Main memory size (per socket) | 12GB | 31GB | 16GB |

| Main memory size (per node) | 23GB | 63GB | 31GB |

> Also include in the description the architectural diagram for one of the nodes boada-1 to boada-4 as obtained when using the lstopo command, appropriately comment whatever you consider appropriate.

![](img/map-1.png)

## Strong vs. weak scalability

> Briefly explain what strong and weak scalability refer to. Exemplify your explanation using the execution time and speed–up plots that you obtained for pi omp.c on the different node types available in boada. Reason about the results that are obtained.

Strong scalability is when the numbers of processors are increased and the run time of the program is reduced while the problem size stays constant. This type of scalability is desirable because in theory it should be possible to decrease the run time of the parallel part of a program to 0. Of course this isn’t possible because parallelization has overhead.

Weak scalability is when the number of processors are increased proportional to the problem size. This way it is possible to solve larger problems. It is used when strong scalability is not possible.

The strong scalability can be observed in the following plots.

### Strong scalability

#### Boada 1-4

In the figure the time decreases with the number of threads in the beginning but after thread 11, it starts to increases again. This could be affiliated to the overhead of synchronization. The speedup is almost linear in the beginning but plums after a while.

<img src=”img/boada1\_1.png” style=”width:250px;”>

<img src=”img/boada1\_2.png” style=”width:250px;”>

#### Boada 5

In the figures of Boada 5 the speedup is better than in Boada 1-3, however, it has a zickzack movement in the end. The time decreases logarithmically with the number of threads.

<img src=”img/boada2\_1.png” style=”width:250px;”>

<img src=”img/boada2\_2.png” style=”width:250px;”>

#### Boada 6-8

The figures of Boada 6-8 are similar to Boada 5 but the speedup is linear until the end. The time decreases with the number of threads logarithmically as well.

<img src=”img/boada3\_1.png” style=”width:250px;”>

<img src=”img/boada3\_2.png” style=”width:250px;”>

As expected the strong scalability decreases the run time with the number of threads while the problem size stays the same. Depending on the hardware the speedup stops sooner or later.

### Weak scalability

#### Boada 1-4

As expected the speedup stays at around one with a slight decrease to the end. This means that a bigger problem size can be calculated with a slight increase in time through the use of more threads.

<img src=”img/weak\_1.png” style=”width:250px;”>

#### Boada 5

The figure of Boada 5 shows strange zickzack movement when the threads increase. In the end there is a big decrease in performance.

<img src=”img/weak\_2.png” style=”width:250px;”>

#### Boada 6-8

This figure shows a very stable speedup with the increase of threads. This is a great example of weak scaling.

<img src=”img/weak\_3.png” style=”width:250px;”>

## Analysis of task decompositions for 3DFFT

> In this part of the report you should summarise 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<sub>1</sub> | T<sub>∞</sub> | Parallelism |

|---------|---------------|---------------|-------------|

| seq | 639780001ns | 639780001ns | 1.000000000 |

| v1 | 639780001ns | 639707001ns | 1.000114115 |

| v2 | 639780001ns | 361190001ns | 1.771311496 |

| v3 | 639780001ns | 154354001ns | 4.144887705 |

| v4 | 639780001ns | 64024001ns | 9.992815054 |

| v5 | 639780001ns | 55826001ns | 11.46025131 |

### v1

We replaced the task named ffts1 and transpositions with a sequence of finer grained tasks, one for each function invocation inside it.

#### Changed code from seq

```c

int main (int argc, char \*argv[]) {

...

/\* Initialize Tareador analysis \*/

tareador\_ON (); // <---- NEW

START\_COUNT\_TIME;

tareador\_start\_task("start\_plan\_forward"); // <---- NEW

start\_plan\_forward(in\_fftw, &p1d);

tareador\_end\_task("start\_plan\_forward"); // <---- NEW

STOP\_COUNT\_TIME("3D FFT Plan Generation");

START\_COUNT\_TIME;

tareador\_start\_task("init\_complex\_grid"); // <---- NEW

init\_complex\_grid(in\_fftw);

tareador\_end\_task("init\_complex\_grid"); // <---- NEW

STOP\_COUNT\_TIME("Init Complex Grid FFT3D");

START\_COUNT\_TIME;

tareador\_start\_task("1"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

tareador\_end\_task("1"); // <---- NEW

tareador\_start\_task("2"); // <---- NEW

transpose\_xy\_planes(tmp\_fftw, in\_fftw);

tareador\_end\_task("2"); // <---- NEW

tareador\_start\_task("3"); // <---- NEW

ffts1\_planes(p1d, tmp\_fftw);

tareador\_end\_task("3"); // <---- NEW

tareador\_start\_task("4"); // <---- NEW

transpose\_zx\_planes(in\_fftw, tmp\_fftw);

tareador\_end\_task("4"); // <---- NEW

tareador\_start\_task("5"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

tareador\_end\_task("5"); // <---- NEW

tareador\_start\_task("6"); // <---- NEW

transpose\_zx\_planes(tmp\_fftw, in\_fftw);

tareador\_end\_task("6"); // <---- NEW

tareador\_start\_task("7"); // <---- NEW

transpose\_xy\_planes(in\_fftw, tmp\_fftw);

tareador\_end\_task("7"); // <---- NEW

STOP\_COUNT\_TIME("Execution FFT3D");

/\* Finalize Tareador analysis \*/

tareador\_OFF ();

...

}

```

#### Dependency graph

<img src=”img/dependency\_graph\_v1.png” style=”height:500px;”>

#### Execution on 1 processor

![](img/timeline\_v1\_1p.png)

#### Execution on 4 processors

![](img/timeline\_v1\_4p.png)

### v2

#### Changed code from v1

```c

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"); // <---- NEW

for (j=0; j<N; j++)

fftwf\_execute\_dft( p1d, (fftwf\_complex \*)in\_fftw[k][j][0],(fftwf\_complex \*)in\_fftw[k][j][0]);

tareador\_end\_task("ffts1\_planes\_loop\_k"); // <---- NEW

}

}

```

```c

int main (int argc, char \*argv[]) {

...

// tareador\_start\_task("1"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

// tareador\_end\_task("1"); // <---- NEW

...

// tareador\_start\_task("3"); // <---- NEW

ffts1\_planes(p1d, tmp\_fftw);

// tareador\_end\_task("3"); // <---- NEW

...

// tareador\_start\_task("5"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

// tareador\_end\_task("5"); // <---- NEW

...

}

```

#### Dependency graph

![](img/dependency\_graph\_v2.png)

#### Execution on 1 processor

![](img/timeline\_v2\_1p.png)

#### Execution on 4 processors

![](img/timeline\_v2\_4p.png)

### v3

#### Changed code from v2

```c

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\_loop\_k"); // <---- NEW

for (j=0; j<N; j++) {

for (i=0; i<N; i++)

{

tmp\_fftw[k][i][j][0] = in\_fftw[k][j][i][0];

tmp\_fftw[k][i][j][1] = in\_fftw[k][j][i][1];

}

}

tareador\_end\_task("xy\_planes\_loop\_k"); // <---- NEW

}

}

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\_loop\_k"); // <---- NEW

for (j=0; j<N; j++) {

for (i=0; i<N; i++)

{

in\_fftw[i][j][k][0] = tmp\_fftw[k][j][i][0];

in\_fftw[i][j][k][1] = tmp\_fftw[k][j][i][1];

}

}

tareador\_end\_task("zx\_planes\_loop\_k"); // <---- NEW

}

}

```

```c

int main (int argc, char \*argv[]) {

...

// tareador\_start\_task("2"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

// tareador\_end\_task("2"); // <---- NEW

...

// tareador\_start\_task("4"); // <---- NEW

ffts1\_planes(p1d, tmp\_fftw);

// tareador\_end\_task("4"); // <---- NEW

...

// tareador\_start\_task("6"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

// tareador\_end\_task("6"); // <---- NEW

// tareador\_start\_task("7"); // <---- NEW

ffts1\_planes(p1d, in\_fftw);

// tareador\_end\_task("7"); // <---- NEW

...

}

```

#### Dependency graph

![](img/dependency\_graph\_v3.png)

#### Execution on 1 processor

![](img/timeline\_v3\_1p.png)

#### Execution on 4 processors

![](img/timeline\_v3\_4p.png)

### v4

In this version we placed `tareador\_start\_task("init\_complex\_grid");` inside the `k` loop of `init\_complex\_grid(...)`.

### Changed code from v3

```c

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"); // <---- NEW

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"); // <---- NEW

}

}

```

```c

int main (int argc, char \*argv[]) {

...

// tareador\_start\_task("start\_plan\_forward"); // <---- NEW

start\_plan\_forward(in\_fftw, &p1d);

// tareador\_end\_task("start\_plan\_forward"); // <---- NEW

...

// tareador\_start\_task("init\_complex\_grid"); // <---- NEW

init\_complex\_grid(in\_fftw);

// tareador\_end\_task("init\_complex\_grid"); // <---- NEW

...

}

```

#### Dependency graph

![](img/dependency\_graph\_v4.png)

#### Execution on 1 processor

![](img/timeline\_v4\_1p.png)

#### Execution on 4 processors

![](img/timeline\_v4\_4p.png)

### v5

In this version we moved `tareador\_start\_task("init\_complex\_grid");` inside the `j` loop of `init\_complex\_grid(...)`.

#### Changed code from v4

```c

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"); // <---- NEW

for (j = 0; j < N; j++) {

tareador\_start\_task("init\_complex\_grid"); // <---- NEW

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"); // <---- NEW

}

// tareador\_end\_task("init\_complex\_grid"); // <---- NEW

}

}

```

#### Dependency graph

![](img/dependency\_graph\_v5.png)

#### Execution on 1 processor

![](img/timeline\_v5\_1p.png)

#### Execution on 4 processors

![](img/timeline\_v5\_4p.png)

### Comparison of v4 vs. v5

> For versions v4 and v5 of 3DFFT tar.c perform an analysis of the potential strong scalability that

is expected. For that include a plot with the execution time and/or speedup when using 1, 2, 4, 8, 16 and 32 processors, as reported by the simulation module inside Tareador. You should also include the relevant(s) part(s) of the code that help the reader to understand why v5 is able to scale to a higher number of processors compared to v4, capturing the task dependence graphs that are obtained with Tareador.

Because v5 generates more parallel tasks than v4 it’s able to scale to a higher number of processors.

V5 generates more tasks because `tareador\_start\_task("init\_complex\_grid");` is inside a deeper nested loop.

| | 1 processor | 2 processors | 4 processors | 8 processors | 16 processors | 32 processors |

|---------|--------------|--------------|--------------|--------------|---------------|---------------|

| v4 | 639780001ns | 320330001ns | 165395001ns | 91502001ns | 64024001ns | 64024001ns |

| v5 | 639780001ns | 320087001ns | 165727001ns | 94040001ns | 60919001ns | 57934001ns |

| speedup | 1 | 1,0007591690 | 0,9979967054 | 0,9730114848 | 1,050969319 | 1,105119617 |

<img src="img/comparison1.png" style="height: 300px;">

<img src="img/comparison2.png" style="height: 300px;">

## 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 of 3DFFT. 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 | φ | S<sub>∞</sub>| T<sub>1</sub> | T<sub>8</sub> | S<sub>8</sub> |

|------------------------------------------------------|--------------|--------------|---------------|---------------|---------------|

| initial version in 3DFFT omp.c | 0.53 | 2.13 | 3064 ms | 1598 ms | 1,92 |

| new version with improved φ | 0.84 | 6.25 | 2520 ms | 841 ms | 2,99 |

| final version with reduced parallelisation overheads | 0.82 | 5.55 | 2797 ms | 685 ms | 4,08 |

To calculate T<sub>seq</sub> we added the the time of each none parallelitzable block:

| | initial | 3dfft improved | 3dfft final |

|-----------------|-------------|----------------|-------------|

| T<sub>par</sub> | 1643 ms | 2112 | 2309 |

| T<sub>seq</sub> | 1421 ms | 408 | 488 |

The calculations are based on the following functions:

φ = T<sub>par</sub> / (T<sub>seq</sub> + T<sub>par</sub>)

S<sub>∞</sub> = 1 / (1 - φ)

> Finally you should comment about the (strong) scalability plots (execution time and speed–up) that are obtained when varying the number of threads for the three parallel versions that you have analysed.

The data could be obtained using Paraver.

This is the initial version with one thread:

![](img/V1\_1.png)

This is the initial version with 8 threads:

![](img/V1\_8.png)

This is the improved version with one thread:

![](img/V2\_1.png)

This is the improved version with 8 threads:

![](img/V2\_8.png)

This is the final version with one thread:

![](img/V3\_1.png)

This is the final version with 8 threads:

![](img/V3\_8.png)