Lab 1: Experimental setup and tools

Username: par2110

Semester: Spring 2017-2018

Date: 8/3/2019

Authors:

- Maurici Abad Gutierrez
- Nikolaus Schrack

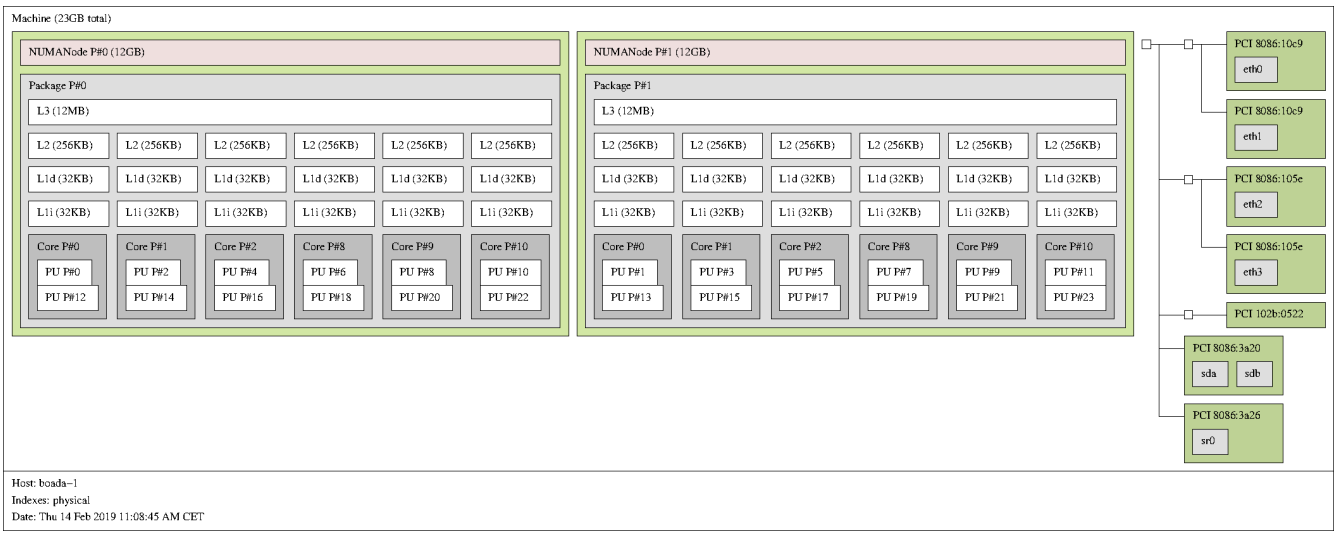
Node architecture and memory

> 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.



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.

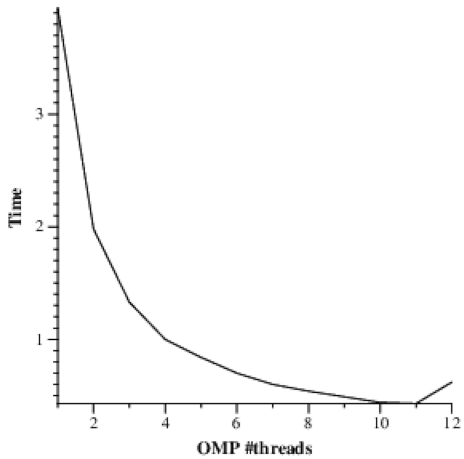
Weak scalability is when the number of processors are increased proportional to the problem size. This way it is possible to solve larger problems. It is used when strong scalability is not possible.

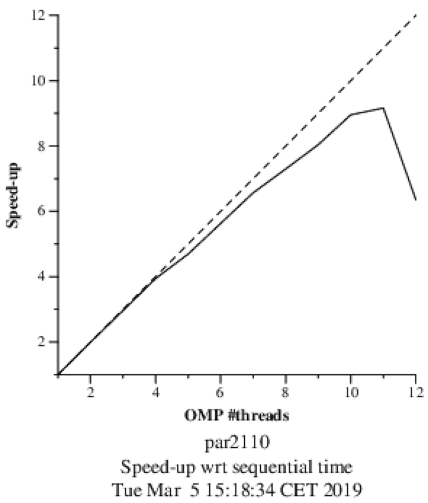
The strong scalability can be observed in the following plots.

Strong scalability

Boada 1-4

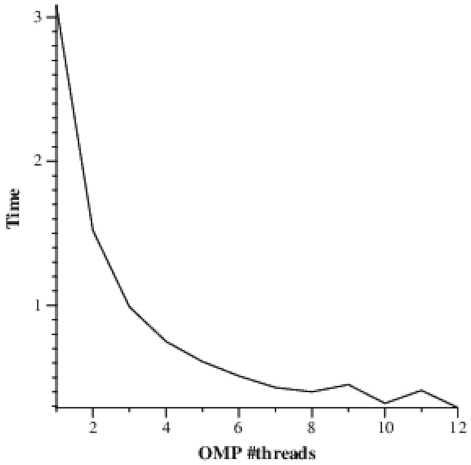
In the figure the time decreases with the number of threads in the beginning but after thread 11, it starts to increases again. This could be affiliated to the overhead of synchronization. The speedup is almost linear in the beginning but plums after a while.



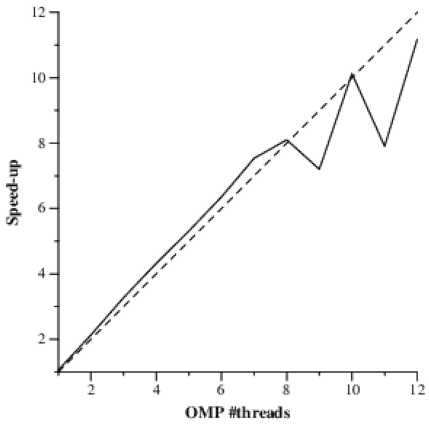


Boada 5

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.



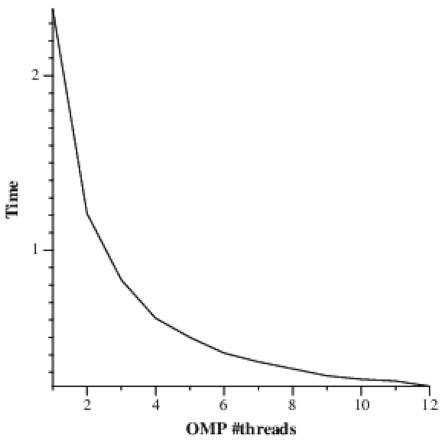
par2110
Min elapsed execution time
Tue Mar 5 15:57:01 CET 2019



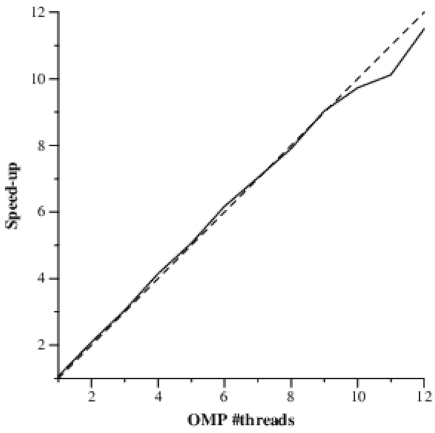
par2110
Speed-up wrt sequential time
Tue Mar 5 15:57:01 CET 2019

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.



par2110
Min elapsed execution time
Tue Mar 5 15:56:46 CET 2019



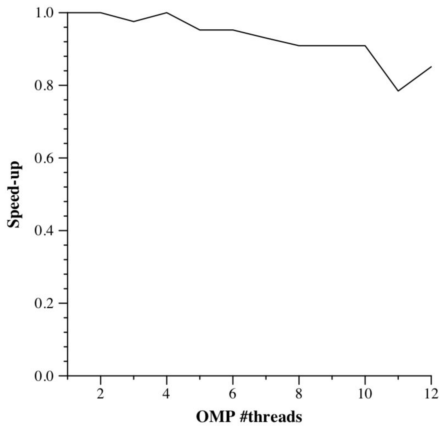
par2110
Speed-up wrt sequential time
Tue Mar 5 15:56:46 CET 2019

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

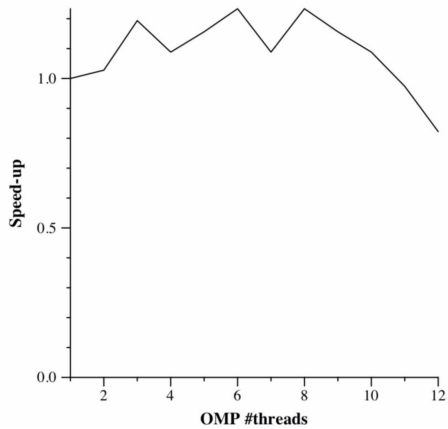
Boada 1-4

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.



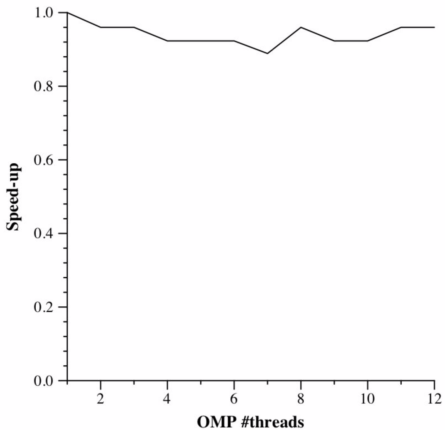
Boada 5

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



Boada 6-8

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



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_1	T_∞	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

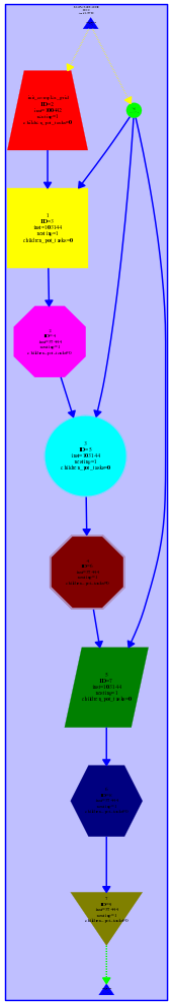
We replaced the task named ffts1 and transpositions with a sequence of finer grained tasks, one for each function invocation inside it.

Changed code from seq

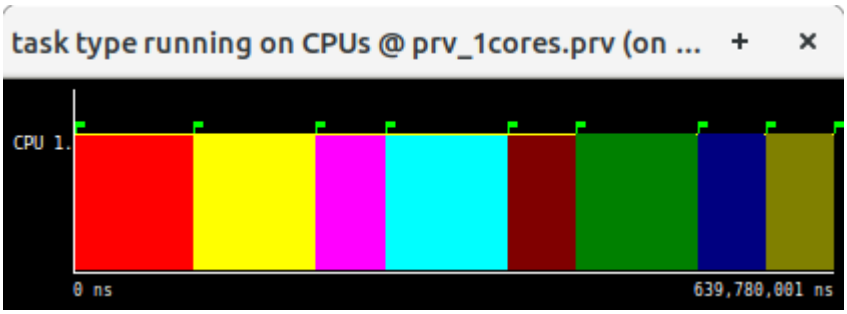
```
int main (int argc, char *argv[]) {
    ...

    /* 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 ();

    ...
}
```

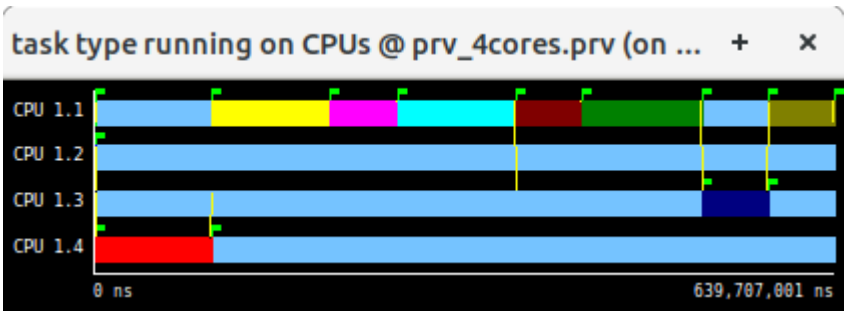
Dependency graph



Execution on 1 processor



Execution on 4 processors



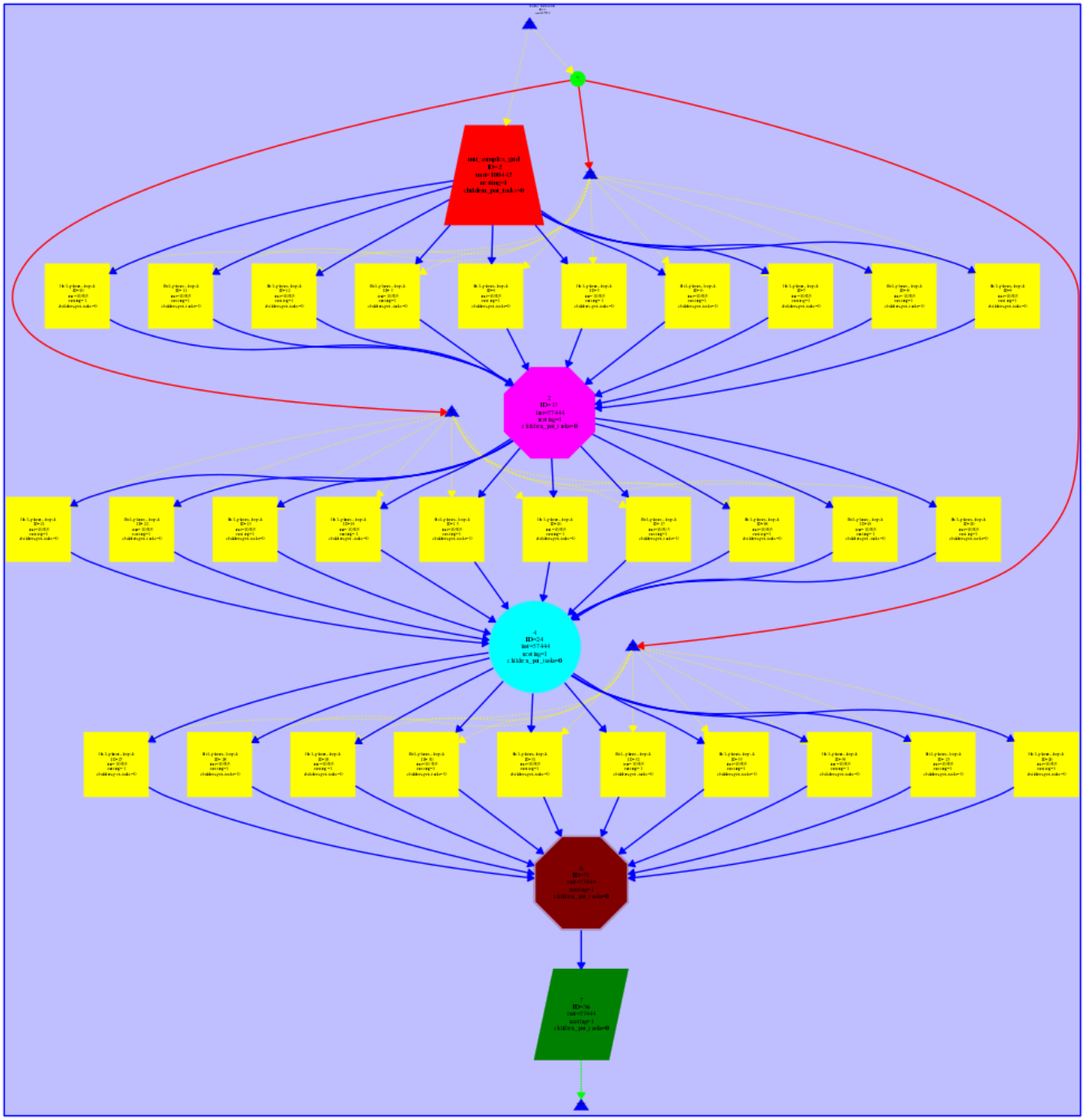
v2

Changed code from v1

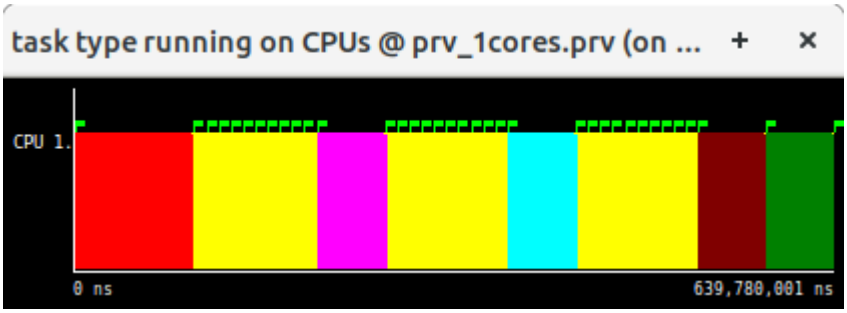
```
void ffts1_planes(fftwf_plan p1d, fftwf_complex in_fftw[][N][N]) {
  int k,j;
    for (k=0; k<N; k++)  {
      tareador_start_task("ffts1_planes_loop_k"); // <---- NEW
      for (j=0; j<N; j++)
        fftwf_execute_dft( p1d, (fftwf_complex *)in_fftw[k][j][0],(fftwf_complex *)in_fftw[k][j]
[0]);
      tareador_end_task("ffts1_planes_loop_k"); // <---- NEW
    }
}
```

```
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

    ...
}
```

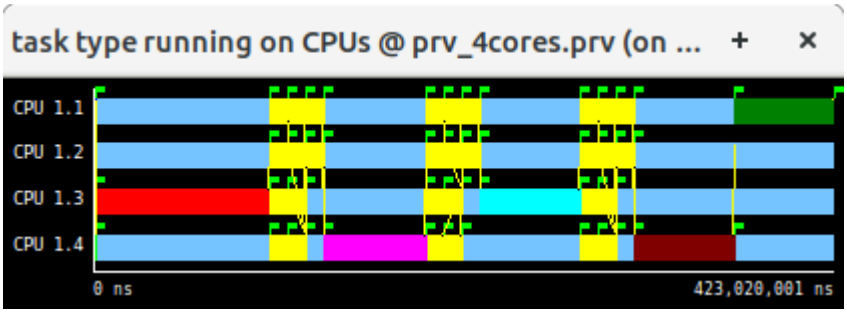
Dependency graph



Execution on 1 processor



Execution on 4 processors



v3

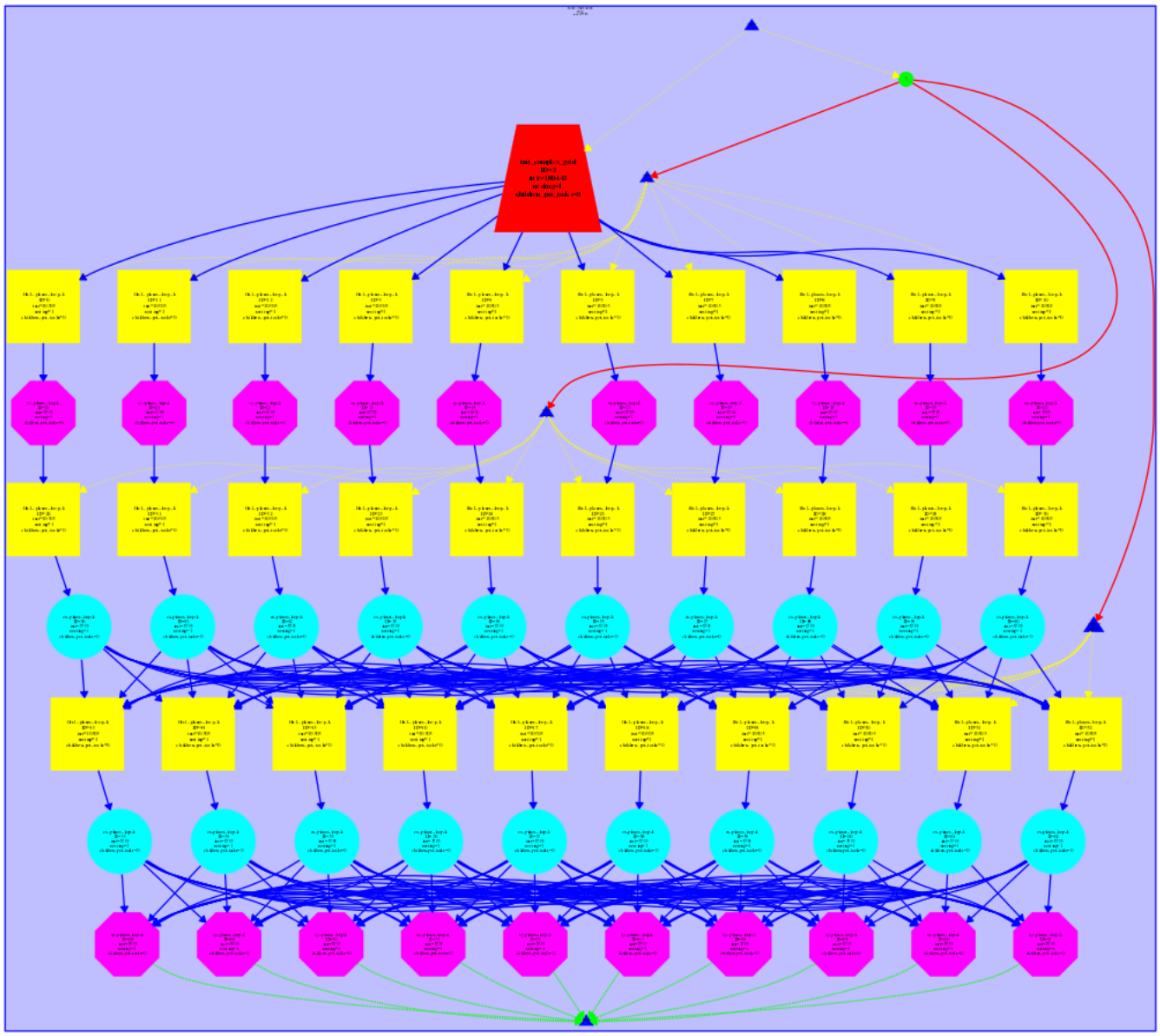
Changed code from v2

```
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
    }
}
```

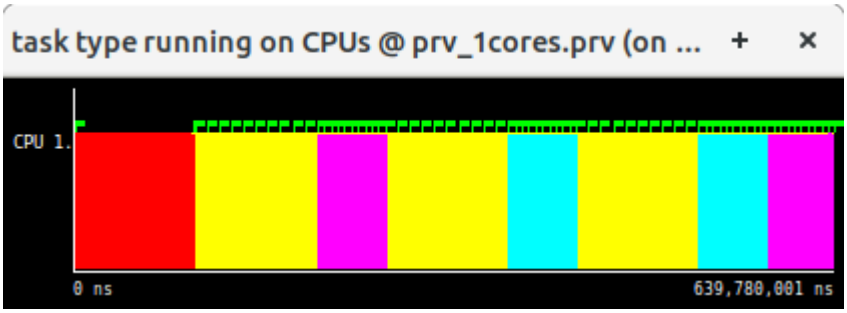
```
int main (int argc, char *argv[]) {
    ...

    // 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

    ...
}
```

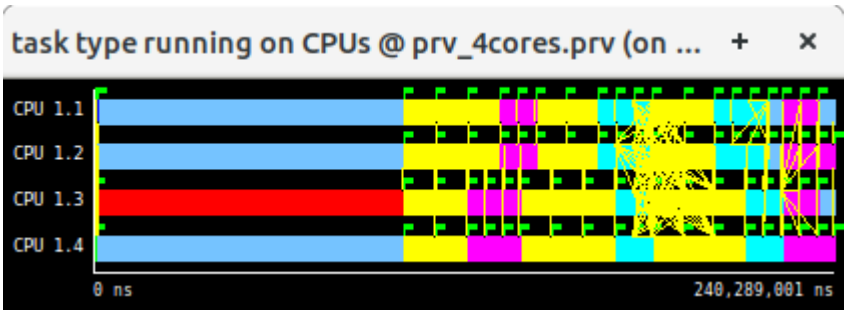
Dependency graph



Execution on 1 processor



Execution on 4 processors



v4

In this version we placed `tareador_start_task("init_complex_grid");` inside the `k` loop of `init_complex_grid(...)` .

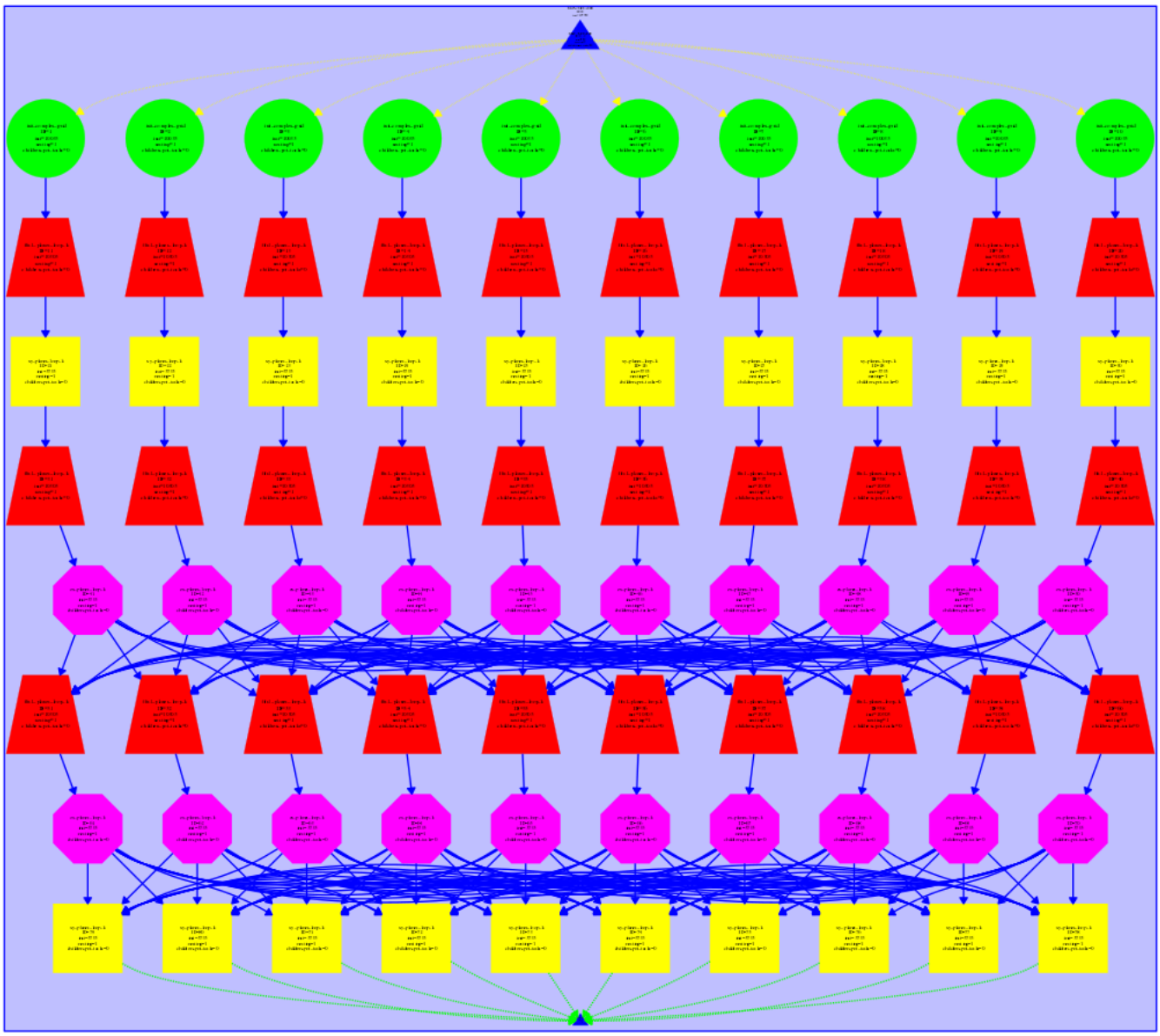
Changed code from v3

```
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

  }
}
```

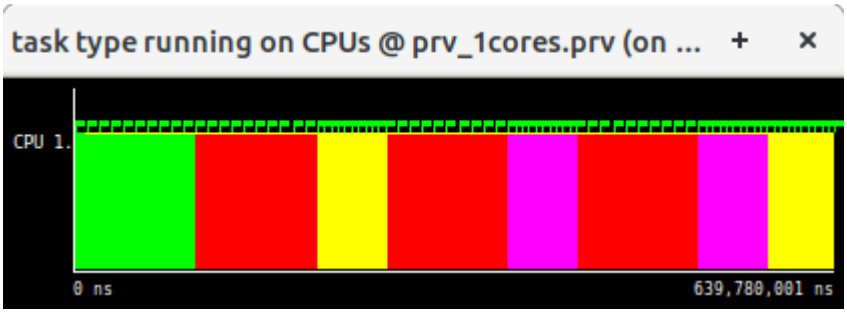
```
int main (int argc, char *argv[]) {
```

```
    ...

    // 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

    ...
}
```

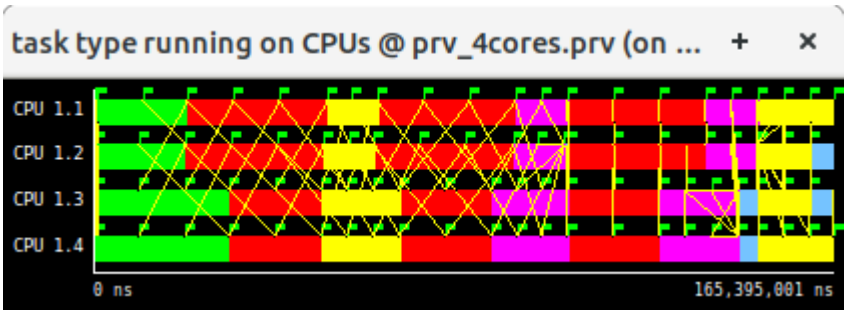
Dependency graph



Execution on 1 processor



Execution on 4 processors



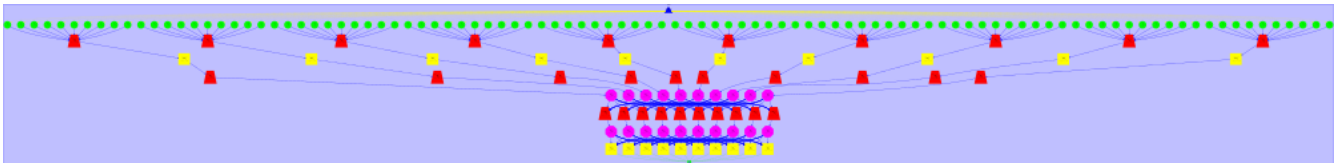
v5

In this version we moved `tareador_start_task("init_complex_grid");` inside the `j` loop of `init_complex_grid(...)` .

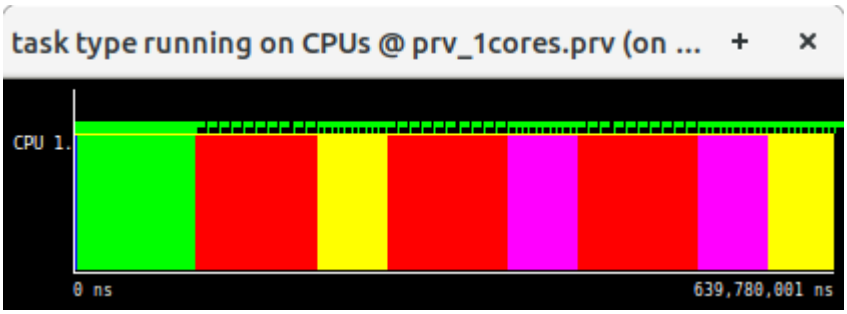
Changed code from v4

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

  for (k = 0; k < N; k++) {
    // tareador_start_task("init_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
  }
}
```

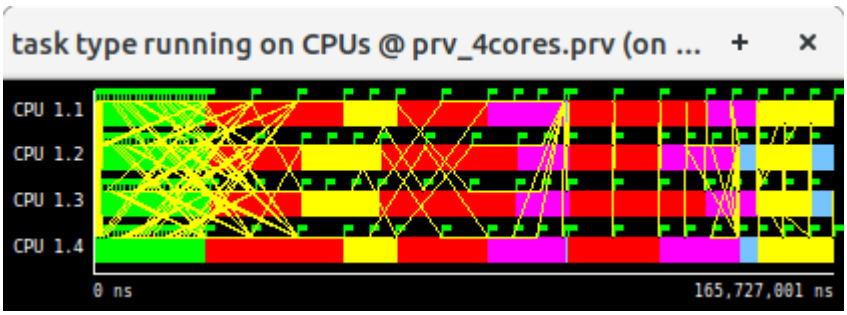
Dependency graph



Execution on 1 processor



Execution on 4 processors

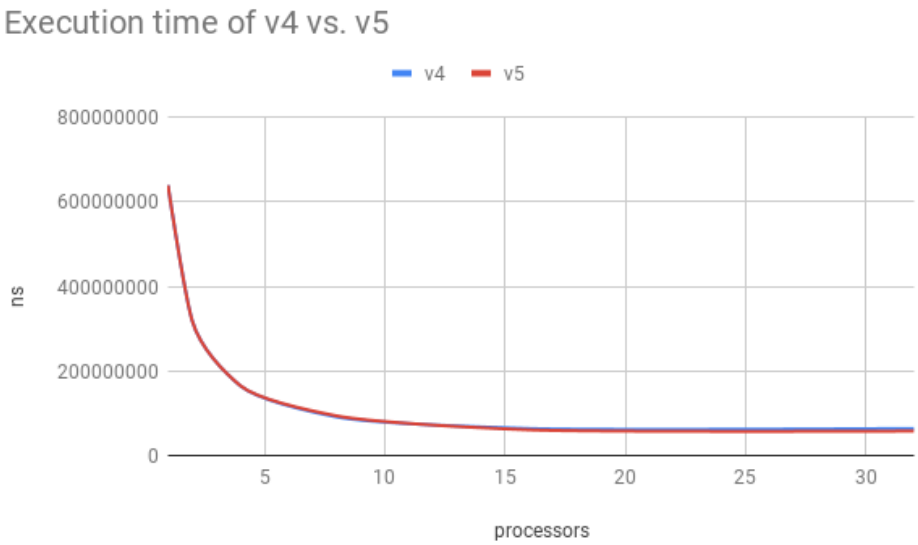


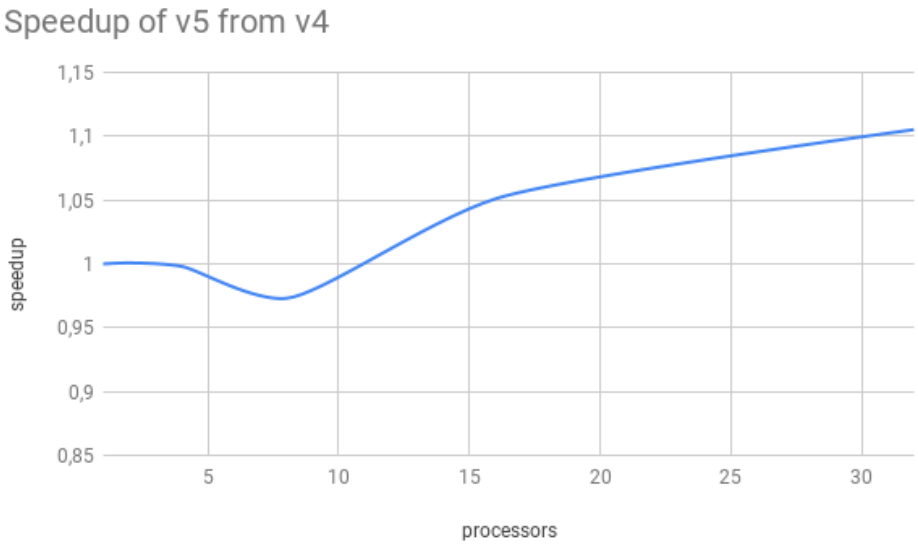
Comparison of v4 vs. v5

> 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





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_∞	T_1	T_8	S_8
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_{seq} we added the the time of each none parallelitzable block:

	initial	3dfft improved	3dfft final
T_{par}	1643 ms	2112	2309
T_{seq}	1421 ms	408	488

The calculations are based on the following functions:

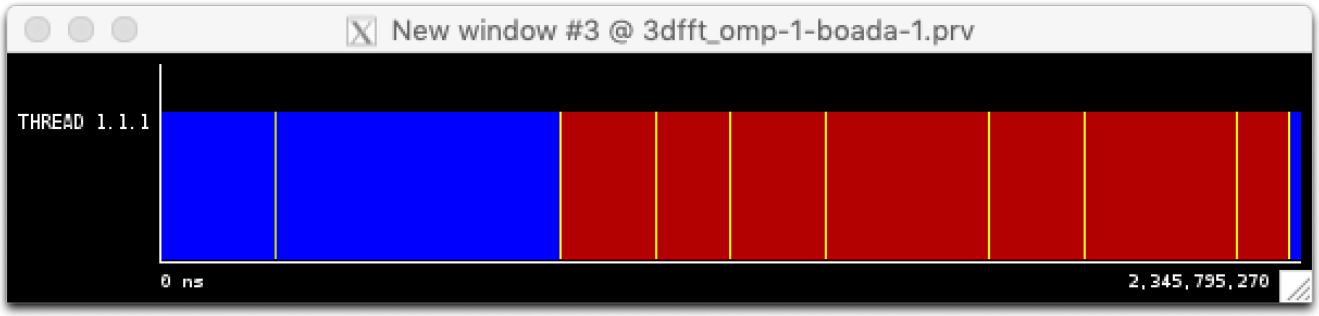
$\phi = T_{par} / (T_{seq} + T_{par})$

$S_{\infty} = 1 / (1 - \phi)$

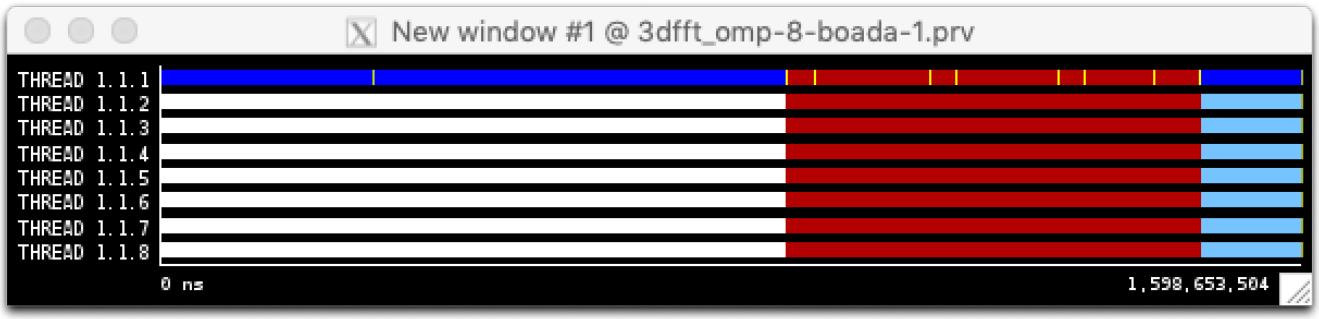
> Finally you should comment about the (strong) scalability plots (execution time and speed–up) that are obtained when varying the number of threads for the three parallel versions that you have analysed.

The data could be obtained using Paraver.

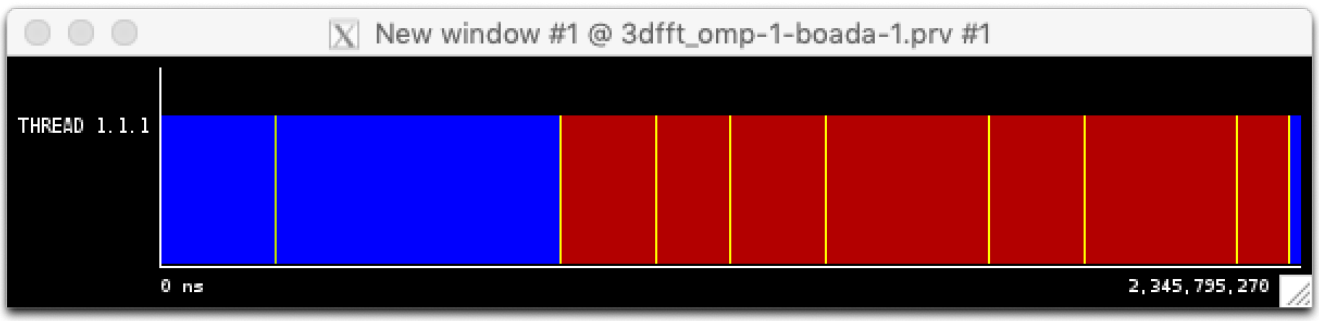
This is the initial version with one thread:



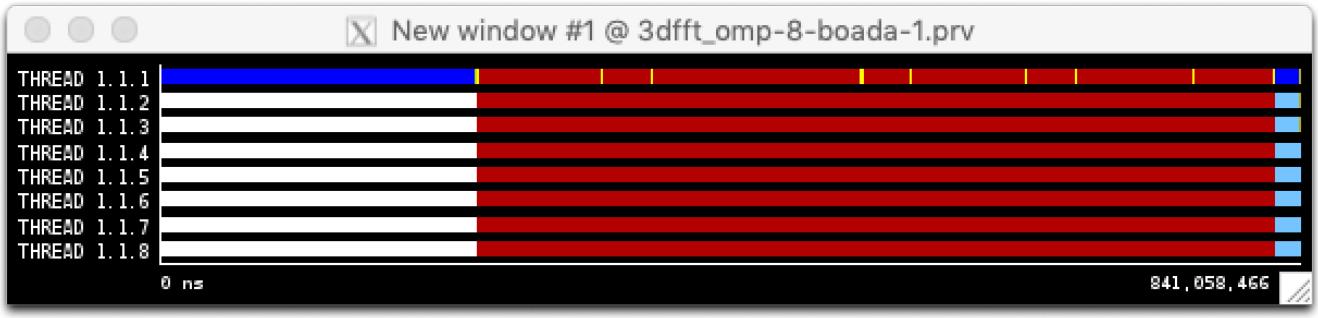
This is the initial version with 8 threads:



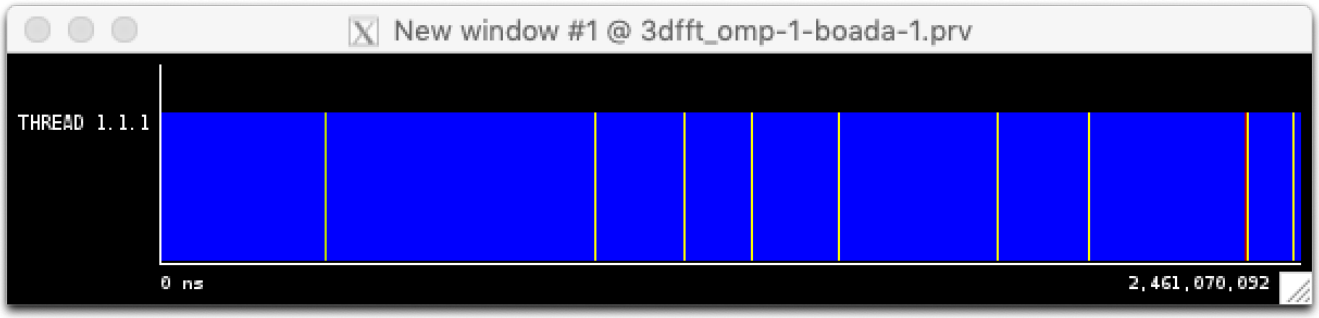
This is the improved version with one thread:



This is the improved version with 8 threads:



This is the final version with one thread:



This is the final version with 8 threads: