

# **Lab 1: Experimental setup and tools**

**Pablo Cebollada Hernández  
Iván Díaz Ortega  
Par1104  
7/10/19**

## Node architecture and memory

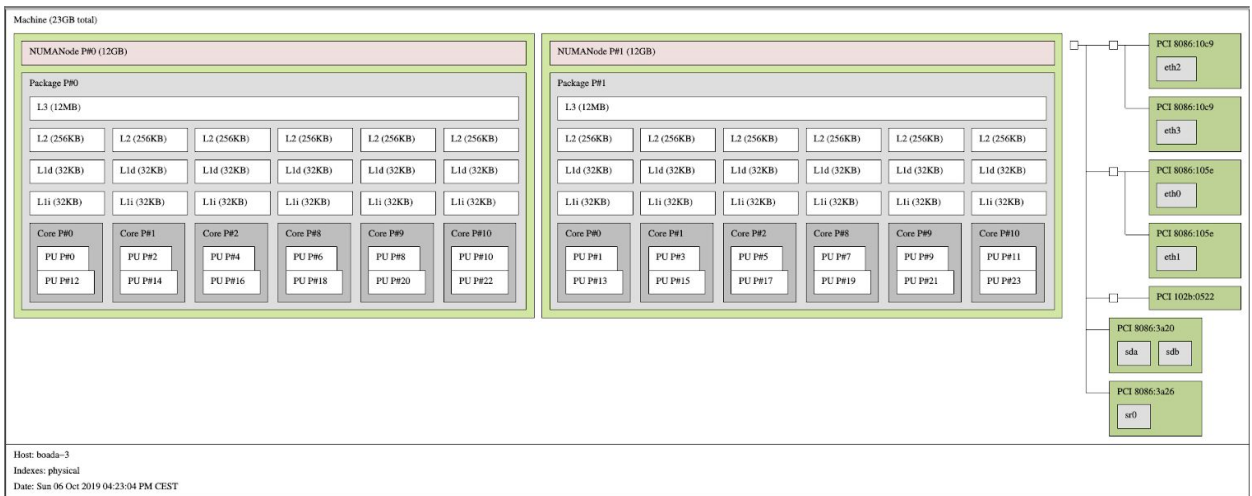
El servidor boada está compuesto por 8 nodos, desde boada-1 hasta boada-8, separados en tres grandes grupos. El primero está compuesto por los boada-1 hasta el boada-4. Cada nodo contiene dos sockets con procesadores Intel Xeon E5645 de 6 núcleos con una frecuencia máxima de 2395 MHz. Cada procesador dispone de 12GB de memoria principal, con un total de 23GB de memoria por nodo.

El segundo grupo está compuesto únicamente por boada-5, el cual tiene, igual que el anterior grupo, 2 sockets, pero estos llevan un Intel Xeon E5-2620 v2 de 6 núcleos con una frecuencia máxima de 2600 MHz. Cada procesador dispone de 31GB de memoria principal, haciendo un total de 63GB de memoria en total para el nodo. Este también contiene una tarjeta gráfica Nvidia K40C.

El tercer y último grupo está compuesto por los nodos boada-6 hasta el boada-8 y, como todos los anteriores, tiene dos sockets por nodo que contiene un Intel Xeon E2609 v4 de 8 núcleos a una frecuencia máxima de 1700 MHz. Cada procesador dispone de 16GB de memoria principal, haciendo un total de 32GB de memoria principal para el nodo.

	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)	12 MB	15 MB	20 MB
Main memory size (per socket)	12 GB	31 GB	16 GB
Main memory size (per node)	23 GB	63 GB	32 GB

1.1. Tabla características boada

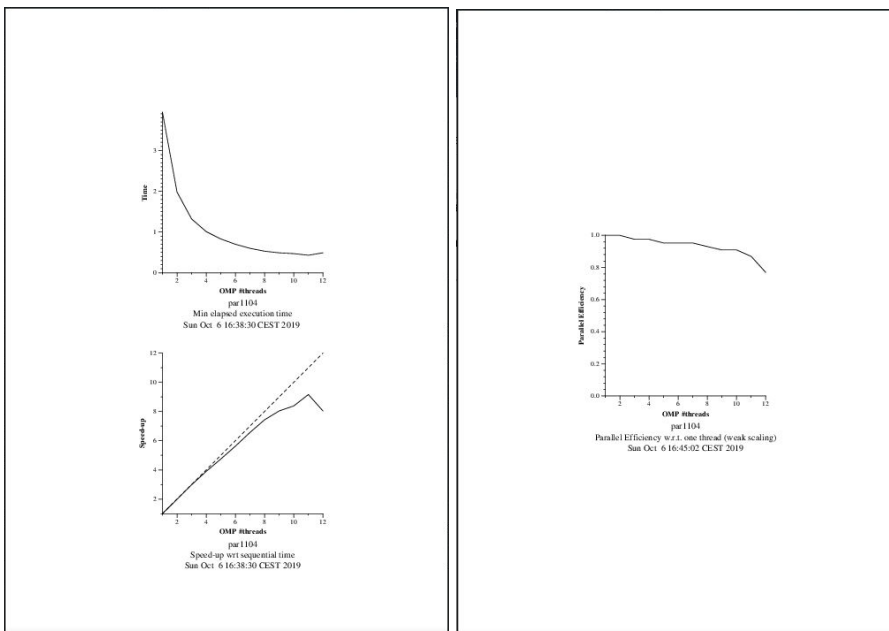


1.2. Estructura de boada-3

## Strong vs weak scalability

La escalabilidad fuerte consiste en reducir el tiempo de ejecución dado un mismo número de entradas, tal y como se puede observar en la gráfica 1.3. con el tiempo de ejecución, el cual se va reduciendo según aumentamos el número de hilos.

Por otra parte, la escalabilidad débil consiste en mantener el mismo tiempo de ejecución según se aumenta el número de entradas. Si observamos la segunda gráfica (1.4.) se puede ver como la eficiencia se mantiene (hasta cierto punto), lo que significa que el tiempo de ejecución según se mantiene según se aumenta el número de hilos. La bajada de eficiencia al final se puede achacar a los overheads por la creación de nuevos threads.



1.3. Strong Scalability

1.4. Weak Scalability

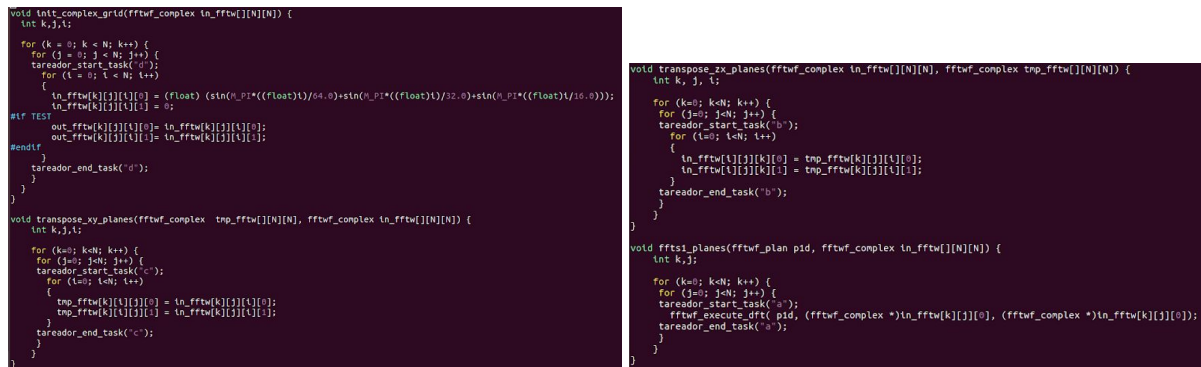
## Analysis of task decomposition for 3DFFT

Teniendo en cuenta los valores obtenidos y representados por la tabla 1.5., se puede observar como según se aumenta la granularidad de la descomposición de las tareas, el nivel de paralelismo aumenta, sobretodo en la versión 5, en la cual el nivel de granularidad es tal que se puede conseguir un paralelismo de casi 80, es decir, que el programa paralelizado es, aproximadamente, 80 veces más rápido, dado un número de procesadores ideal, que la versión secuencial.

Version	$T_1$	$T_\infty$	Parallelism
seq	0.639 s	0.639 s	1
v1	0.639 s	0.639 s	1
v2	0.639 s	0.361 s	1.77
v3	0.639 s	0.154 s	4.15
v4	0.639 s	0.064 s	9.99
v5	0.639 s	0.008 s	78.45

1.5. Tabla de paralelismo

La escalabilidad potencial de las versiones 4 y 5 es muy superior a la de las otras versiones, dado que sus grafos de dependencias muestran una granularidad mucho mayor, sobretodo de la v5, como se puede observar en las imágenes 1.7. y 1.8., sin embargo la diferencia en este potencial es sumamente enorme entre v4 y v5. Simplemente observando los grafos se puede apreciar este hecho en la granularidad de la descomposición de las tareas y si nos fijamos en la gráfica 1.6 se puede ver este hecho a partir de 16 hilos, donde v4 se mantiene constante hasta 32 mientras que v5 sigue disminuyendo su tiempo de ejecución. Este mayor potencial, como ya he comentado antes, se debe a la mayor nivel de granularidad, el cual es causado al paralelizar los bucles de ciertas tareas a un nivel más interno, lo que permite afinar más este nivel y, por lo tanto, ser capaz de paralelizar a una cantidad mayor de hilos. Este hecho se puede apreciar en la imagen 1.9. en que se ve como la declaración que genera la ejecución en paralelo de las diferentes iteraciones del bucle está en el bucle j.



```

void init_complex_grid(fftw_complex in_fftw[N][N]) {
    int k,j,l;
    for (k=0; k<N; k++) {
        for (j=0; j<N; j++) {
            taredor_start_task("d");
            for (l=0; l<N; l++) {
                in_fftw[k][j][l] = (float) (sin(N_PI*((float)l)/64.0)+sin(N_PI*((float)l)/32.0)+sin(N_PI*((float)l)/16.0));
            }
            out_fftw[k][j][l] = in_fftw[k][j][l];
            taredor_end_task("d");
        }
    }
}

void transpose_xy_planes(fftw_complex tnp_fftw[N][N], fftw_complex in_fftw[N][N]) {
    int k,j,l;
    for (k=0; k<N; k++) {
        for (j=0; j<N; j++) {
            taredor_start_task("c");
            for (l=0; l<N; l++) {
                tnp_fftw[k][j][l] = in_fftw[k][j][l];
            }
            taredor_end_task("c");
        }
    }
}

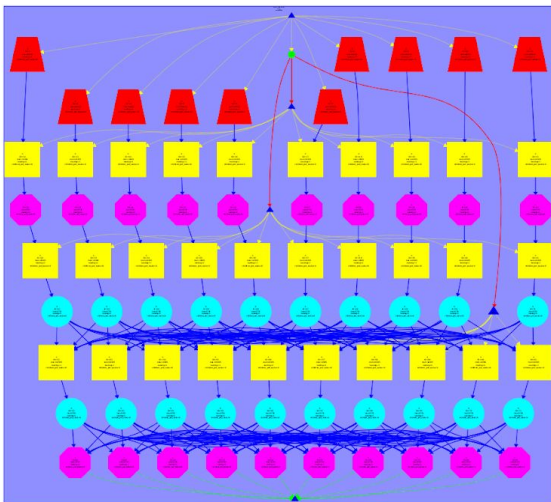
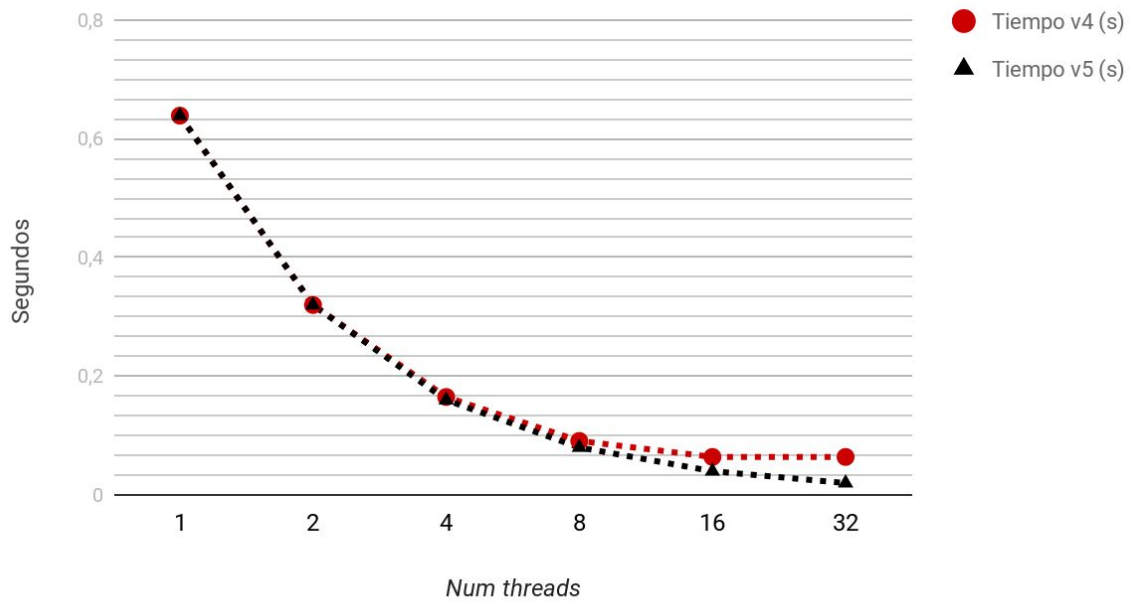
void transpose_zx_planes(fftw_complex in_fftw[N][N], fftw_complex tnp_fftw[N][N]) {
    int k,j,l;
    for (k=0; k<N; k++) {
        for (j=0; j<N; j++) {
            taredor_start_task("b");
            for (l=0; l<N; l++) {
                in_fftw[k][j][l] = tnp_fftw[k][j][l];
            }
            taredor_end_task("b");
        }
    }
}

void ffts1_planes(fftw_plan pid, fftw_complex in_fftw[N][N]) {
    int k,j;
    for (k=0; k<N; k++) {
        for (j=0; j<N; j++) {
            taredor_start_task("a");
            fftw_execute_dft(pid, (fftw_complex *)in_fftw[k][j], (fftw_complex *)in_fftw[k][j]);
            taredor_end_task("a");
        }
    }
}

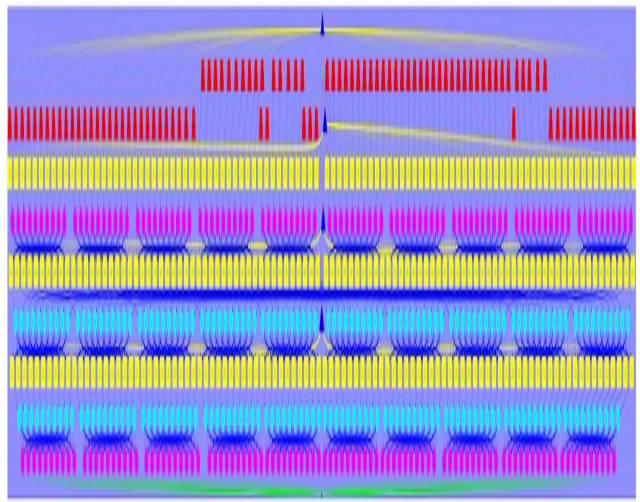
```

1.6 El principal cambio es hacer la ejecución en paralelo dentro del siguiente bucle al que estaba previamente y así aumentar su granularidad.

### 1.7. Tiempo de ejecución v4 y v5



1.8. Grafo de dependencias v4



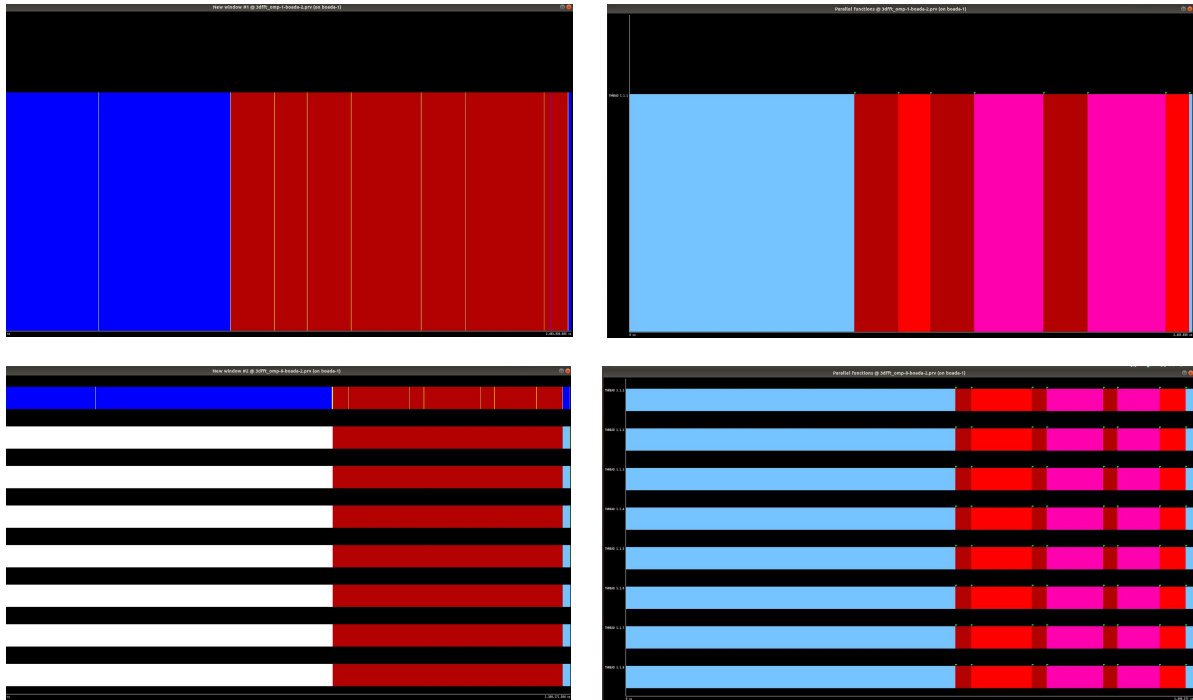
1.9. Grafo de dependencias v5

## Understanding the parallel execution of 3DFFT

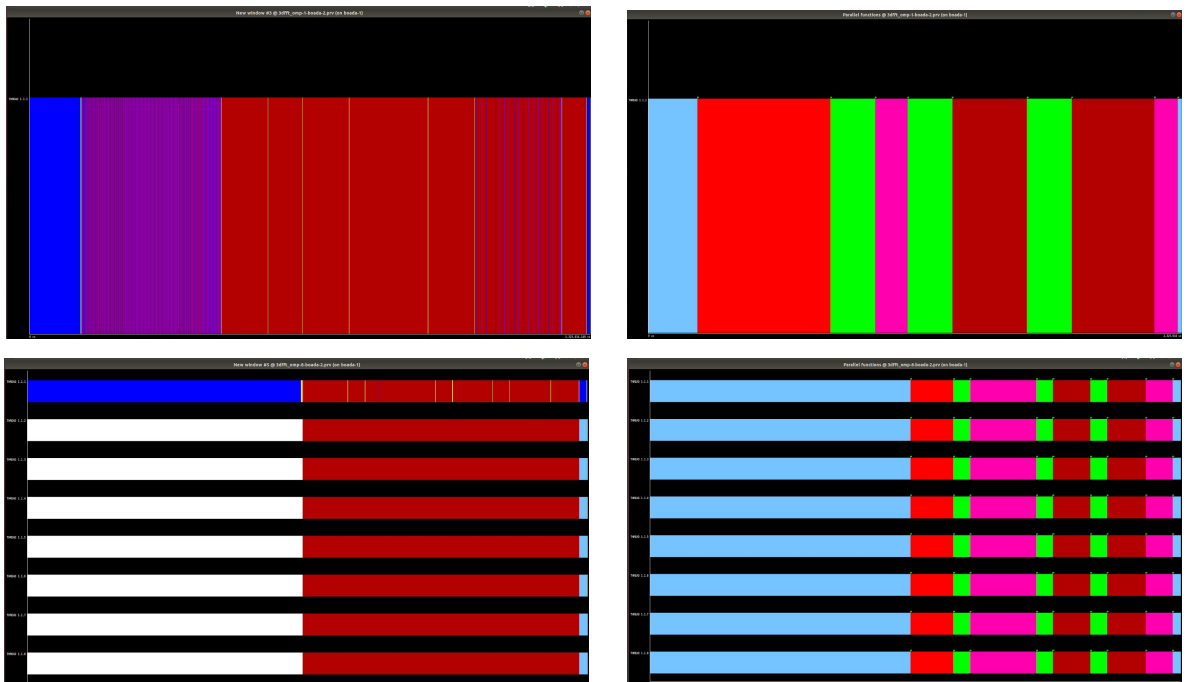
En la primera versión del programa se observa un paralelismo de 2.44 mientras que el paralelismo real observado con 8 hilos es menor. Esta reducción se debe a los overheads creados por la sincronización y creación de procesos. En la siguiente versión se ve que el paralelismo ideal es mucho mayor, como podemos ver en la imagen 1.11 y la 1.12 la porción del programa no paralelizada (zona en azul y azul claro) es mucho mayor en la primera versión (1.11) que en la segunda (1.12), pero cuando comprobamos esto utilizando 8 hilos se puede observar que dicha mejora no contrasta con la ideal sino que ni se le llega a acercar. Este resultado se debe a la acción de los overheads (en esta versión se crean más procesos), los cuales provocan que haya más tiempo de sincronización y creación de procesos. Entonces si redujeramos estos overheads mejoraríamos este paralelismo, que es justo lo que hacemos en la tercera versión. Al reducir estos tiempos se puede observar una mejora en el paralelismo con 8 hilos, aún sin renunciar a la gran parte paralelizada que teníamos en la versión anterior.

Versión	$\phi$	$S^\infty$	$T_1$	$T_8$	$S_8$
initial version in 3dfft_omp.c	0.59	2.44	2485850 $\mu$ s	1380171504 ns	1.8
new version with improved $\phi$	0.9	10	2325634140 ns	1172614139 ns	1.98
final version with reduced parallelisation overheads	0.9	10	2453121 $\mu$ s	588936 $\mu$ s	4.165

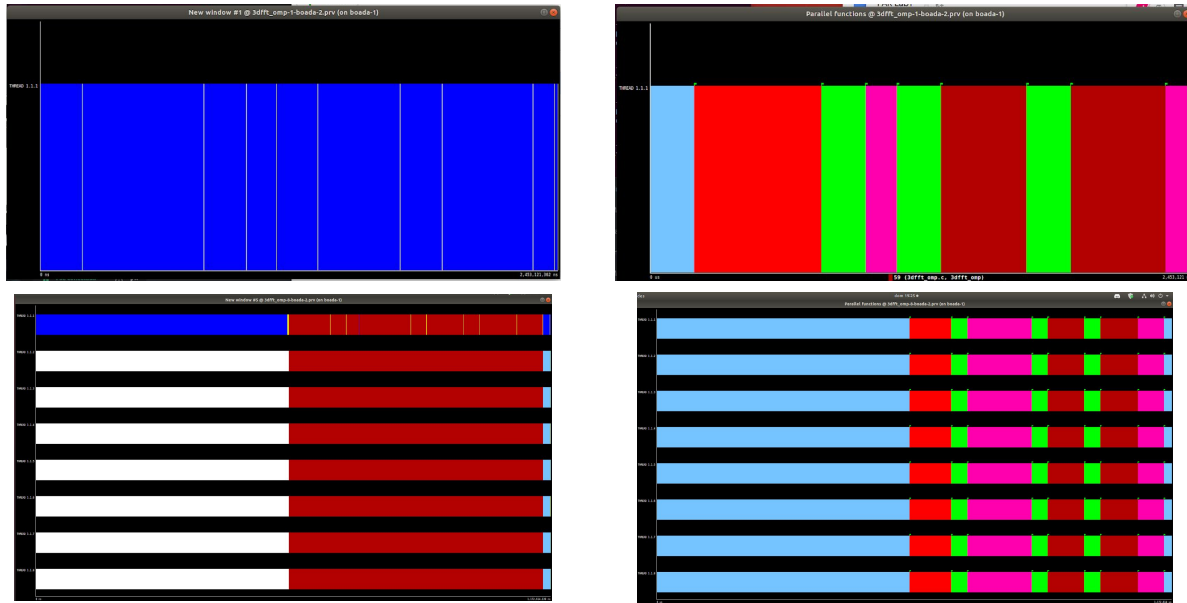
1.10. Tabla de tiempo, porcentaje de tiempo paralelo y paralelismo



1.11 Paraver versión inicial



1.12. Paraver versión  $\phi$  mejorada



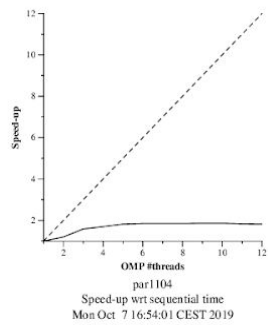
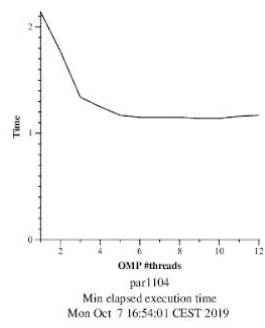
1.13. Paraver versión mejora de overhead

Teniendo en cuenta las conclusiones anteriores podemos ver que estas se cumplen en las gráficas 1.14., 1.15. y 1.16.. En la segunda gráfica podemos ver que, aún teniendo una granularidad mayor que en la primera versión, la mejora en el speed-up según aumentamos el número de hilos de ejecución no mejora mucho respecto a esta y, de la misma forma, este se mantiene constante a partir de un número bastante bajo de hilos. Sin embargo, la tercera versión sí que se mantiene más cercana a la relación lineal entre el speed-up y el número de hilos hasta un número mayor, aún teniendo una granularidad más gruesa que la segunda versión.

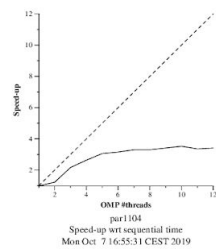
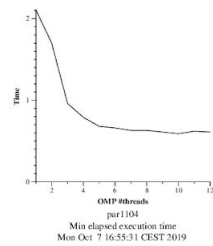
Con respecto al tiempo se puede ver esta misma situación, como en la segunda versión la reducción de tiempo con cada aumento en el número de hilos aumenta hasta cierto punto respecto a la primera versión, y de igual manera la tercera con la segunda.

Esto, como ya dijimos antes, se debe a los overheads creados por la creación y sincronización de los procesos, por eso en la tercera versión, en la cual el número de procesos creados es menor que en la segunda, se va mejorando el tiempo de ejecución conforme vamos añadiendo hilos hasta un número de hilos mayor.

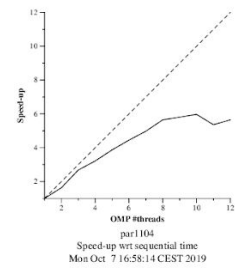
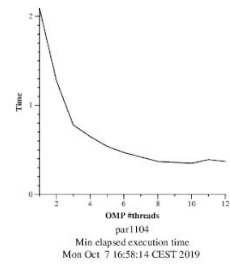




1.14. Gráfica de tiempo y speedup de la versión inicial.



1.15. Gráfica de tiempo y speedup de la segunda versión



1.16. Gráfica de tiempo y speedup de la tercera versión.