Practica 1

Noel Carmona García

Node architecture and memory

1. Draw and briefly describe the architecture of the computer in which you are doing this lab session (number of sockets, cores per socket, threads per core, cache hierarchy size and sharing, and amount of main memory).

Número de sockets: 2 sockets

Cores por socket: 6

Hilos por core: 2

Jerarquía de cache: L1 → 32KB ; L2 → 256KB ; L3 → 12MB

Cantidad de memoria principal: 23 GB

*Fichero adjunto map.pdf

Timing sequential and parallel executions

2. Describe what do you need to add to your program to measure the elapsed execution time between a pair of points in the program, clearly indicating the library header file that needs to be included, the library functions that need to be invoked, the data structure and its fields.

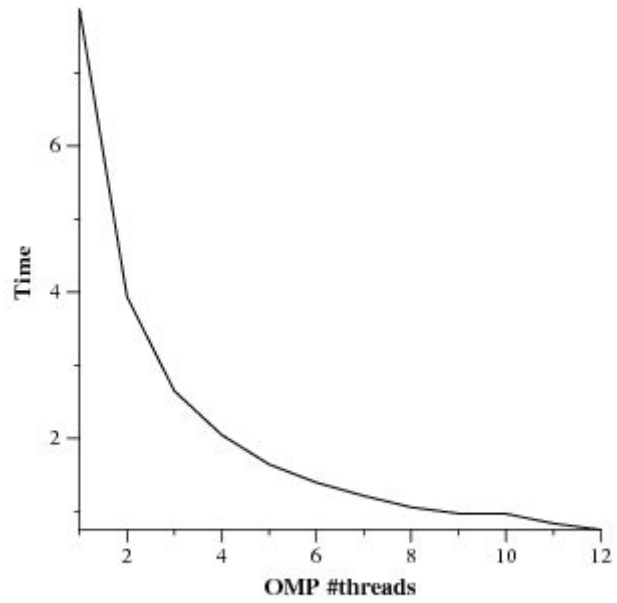
Necesitaremos la librería time.h, la función gettimeofday(timeval, timezone) y el struct timeval:

```
struct timeval {
        time_t      tv_sec;     /* seconds */
         suseconds_t tv_usec;    /* microseconds */
};
```

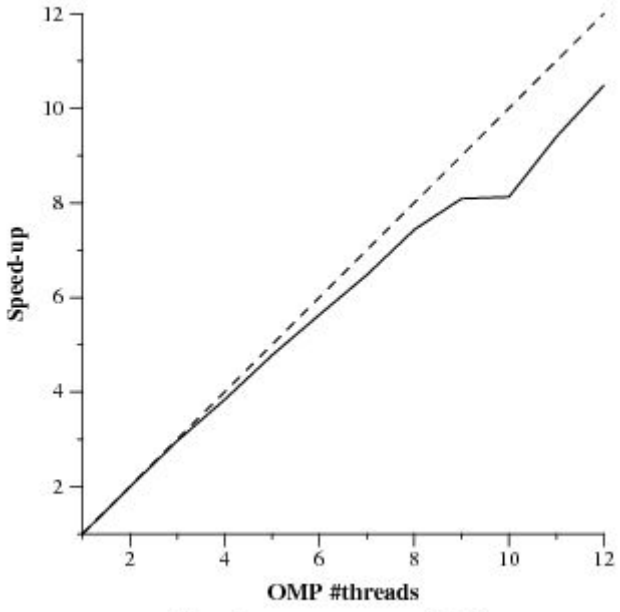
Utilizaremos gettimeofday() para obtener el tiemestamp al empezar a contar el tiempo y cuando queremos dejar de hacerlo. Para calcular el tiempo transcurrido calcularemos la diferencia entre los dos timestamps, de esta forma obtendremos cuando tiempo ha transcurrido entre dos puntos.

En el fichero pi_seq.c tenemos un ejemplo de uso.

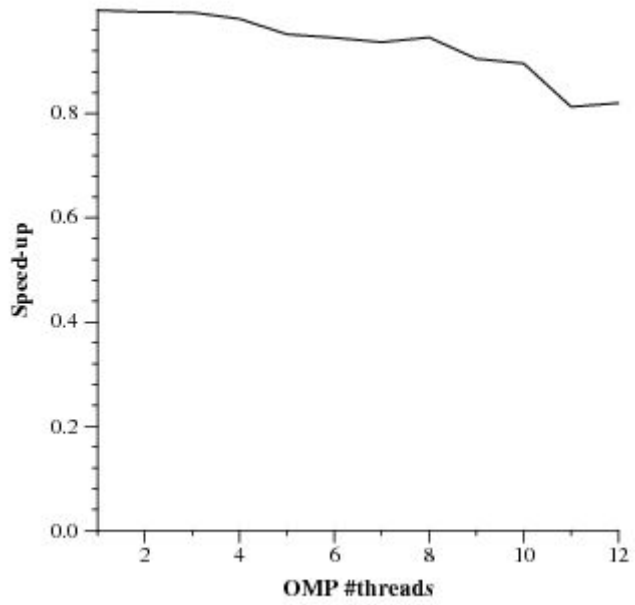
3. Plot the speed–up obtained when varying the number of threads (strong scalability) and problem size (weak scalability) for pi omp.c. Reason about how the scalability of the program.



Min elapsed execution time



Speed-up wrt sequential time



Speed-up wrt one thread, weak scaling

En el caso del gráfico 1 estamos ejecutando el código con weak parallelism, es decir estamos utilizando el paralelismo para incrementar el tamaño del problema, por lo tanto es normal que el speed-up no tenga variaciones a medida que vamos aumentando el número de threads. En cuanto a los dos últimos gráficos el speed-up debe de subir y el tiempo debe disminuir ya que al usar strong-parallelism estamos utilizando el paralelismo para reducir el tiempo de ejecución del problema.

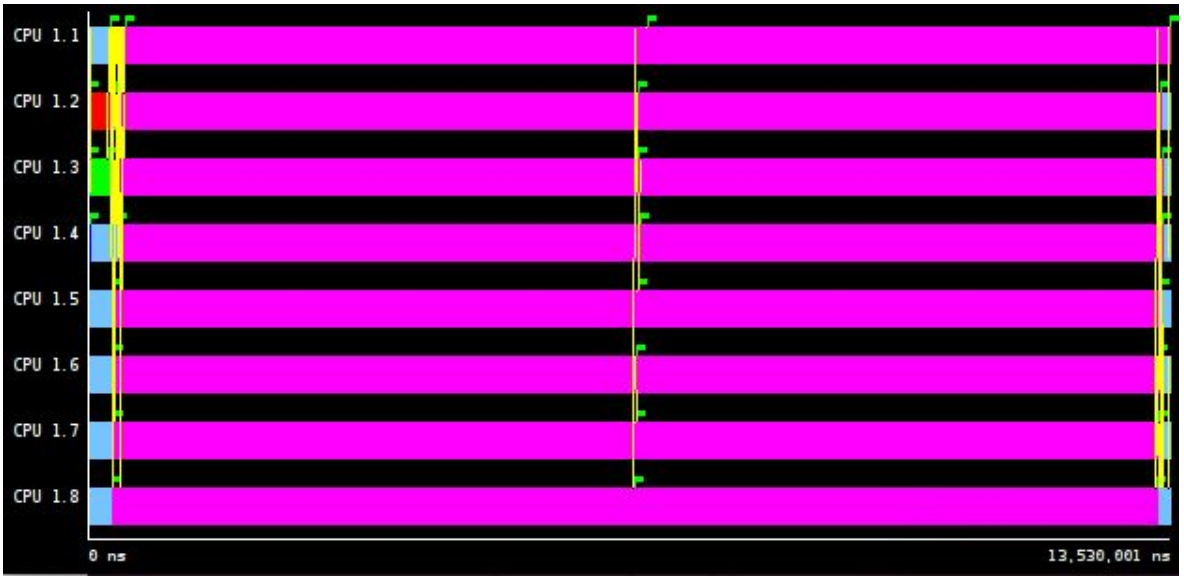
Visualizing the task graph and data dependences

4. Include the source code for function dot product in which you show the Tareador instrumentation that has been added to study the potential parallelism in the code. This instrumentation has to appropriately define tasks and filter the analysis of variable(s) that cause the dependence(s).

```
void dot_product (long N, double A[N], double B[N], double *acc){
        double prod;
        int i;

        *acc=0.0;
        for (i=0; i<N; i++) {
                tareador_start_task("bucle it");
                prod = my_func(A[i], B[i]);
                *acc += prod;
                tareador_end_task("bucle it");
        }
}

tareador_start_task("init_A");
for (i=0; i< size; i++) A[i]=i;
tareador_end_task("init_A");

tareador_start_task("init_B");

for (i=0; i< size; i++) B[i]=2*i;
tareador_end_task("init_B");


tareador_start_task("dot_product");
```

```
tareador_disable_object(&result);
dot_product (size, A, B, &result);
tareador_enable_object(&result);
tareador_end_task("dot_product");
```

5. Capture the task dependence graph for that task decomposition and the execution timelines (for 8 processors) that allow you to understand the potential parallelism attainable. Briefly comment the relevant information that is reported by the tools.



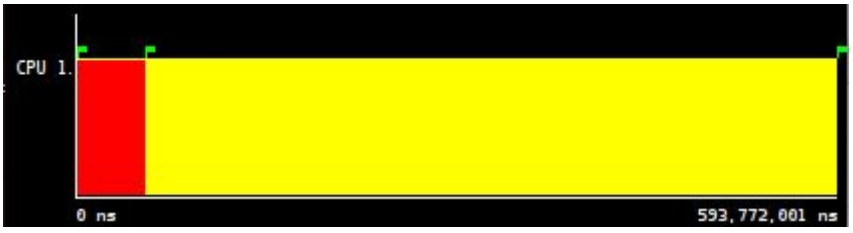
6. Complete the following table for the initial and different versions generated for 3dfft seq.c, briefly commenting the evolution of the metrics with the different versions.

Versión	T1	T_∞	Paralelismo
seq	593772	593758	1,00002
v1	593772	593758	1,00002
v2	593772	315523	1,88
v3	593772	109063	5,44
v4	593772	60148	9,87

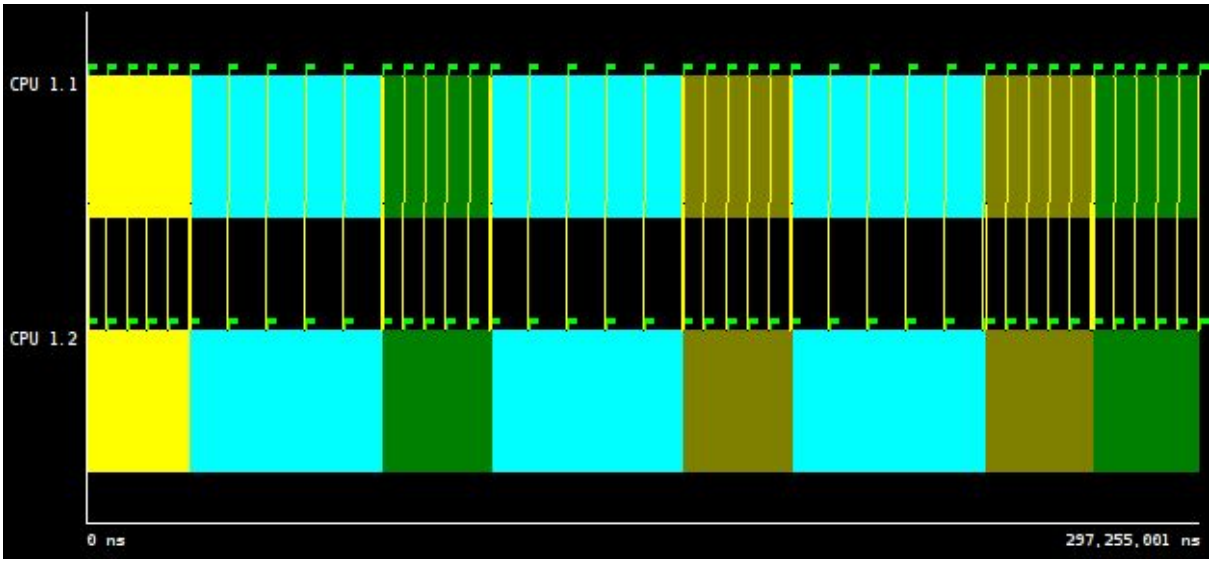
A medida que incrementamos de versión podemos notar a la hora de calcular T_∞ que el número de tareas es mayor, sin embargo el tiempo necesario para cada una de ellas es menor ya que el código es cada vez más paralelizable tal y como se indica en la columna “Parallelism”.

7. 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.

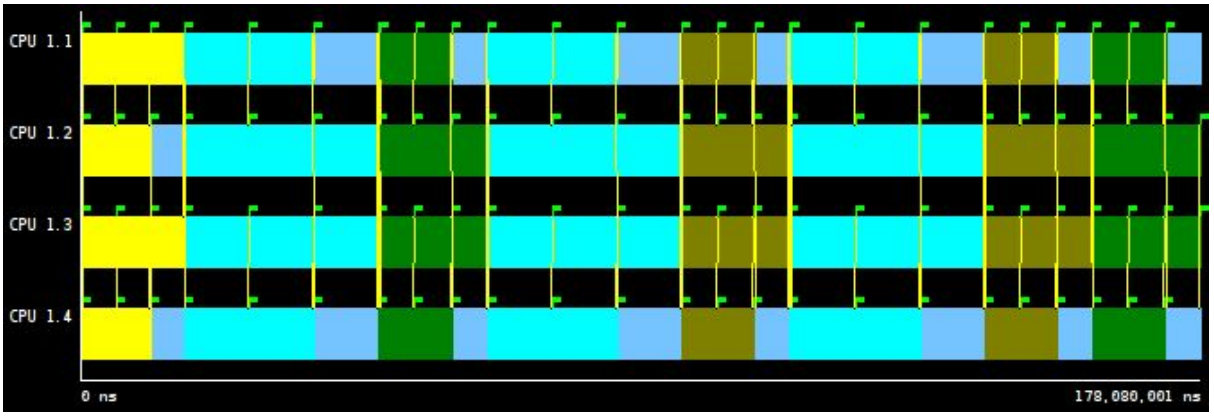
1 procesador (versión seq)



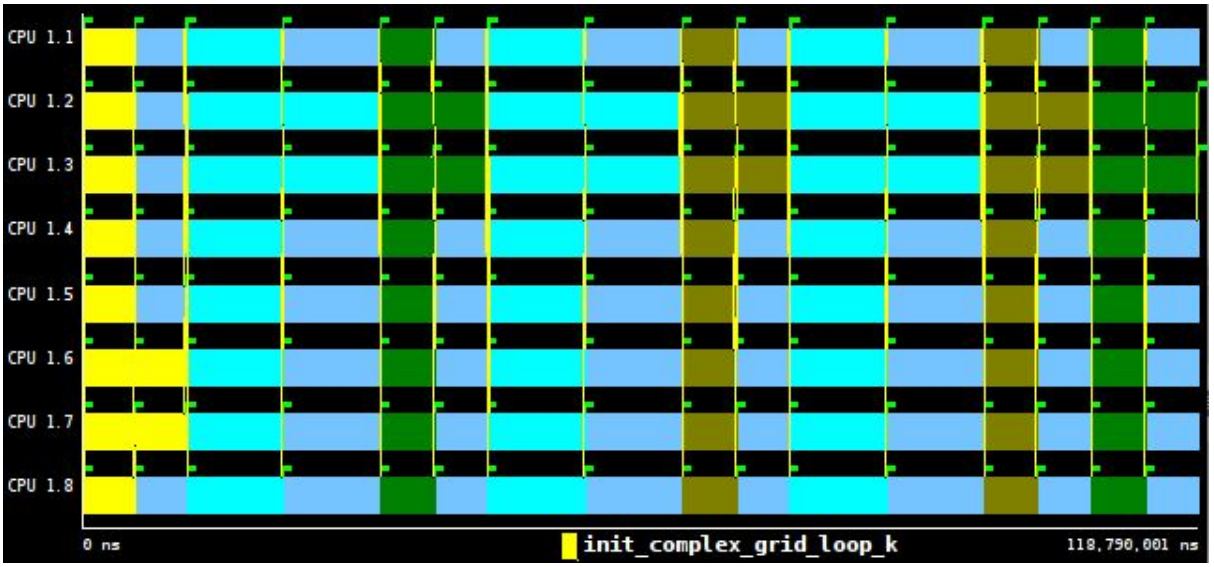
2 procesadores



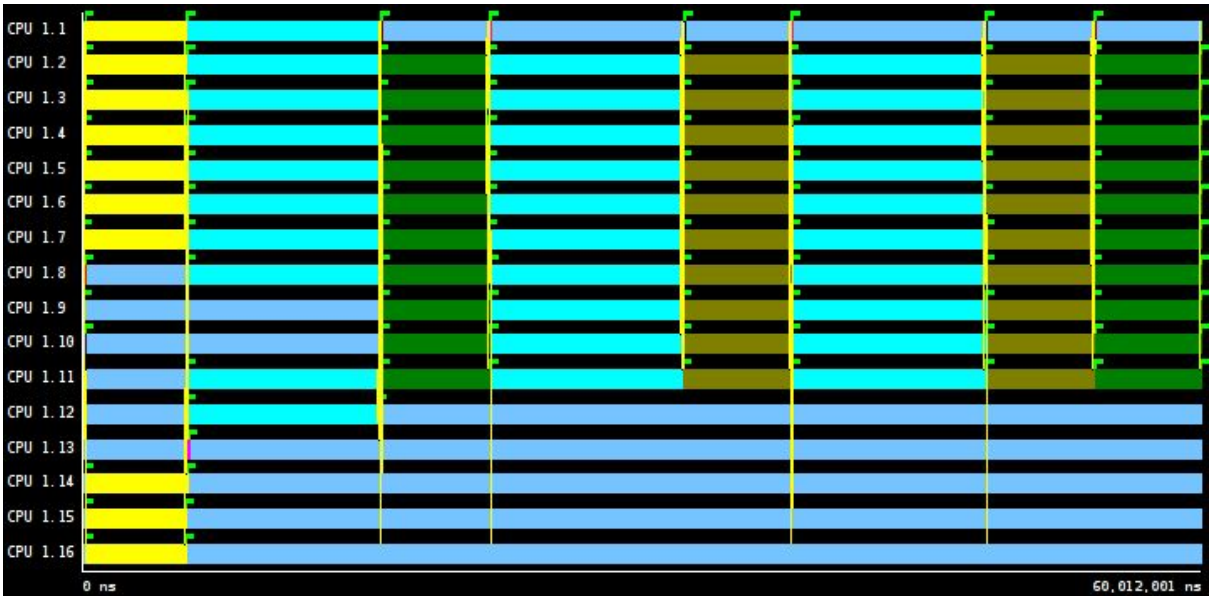
4 procesadores



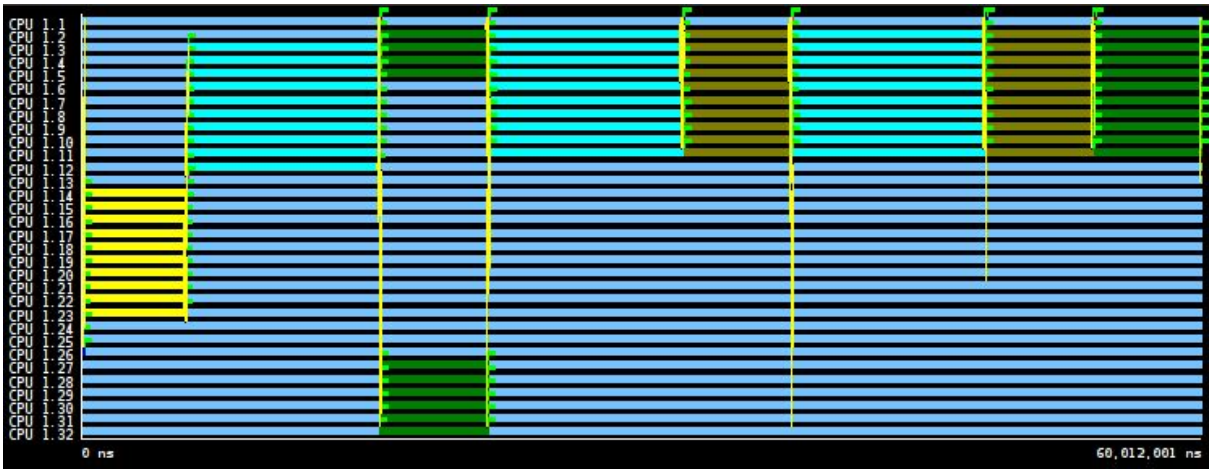
8 procesadores



16 procesadores



32 procesadores



Procesadores	Tiempo	Speedup
seq	593,772,001 ns	1
2	297,255,001 ns	1,99
4	178,080,001 ns	3,33
8	118,790,001 ns	4,99
16	60,012,001 ns	9,89
32	60,012,001 ns	9,89

Como podemos ver en las anteriores imagenes la version secuencial es muy lenta en comparación con las versiones de 2/4/8/16/32 procesadores. El speedup simulando 16 y 32 procesadores es el mismo, lo cual nos permite deducir que P_{min} = 16 en la versión 4.

Tracing sequential and parallel executions

8. From the instrumented version of pi_seq.c, and using the appropriate Paraver configuration file, obtain the value of the parallel fraction φ for this program when executed with 100.000.000 iterations, showing the steps you followed to obtain it. Clearly indicate which Paraver configuration file(s) did you use.

1. Ejecutamos el comando *./submit-seq-i.sh pi_seq.c* y nos generará un fichero .prv el cual debemos de abrir con paraver.

2. Abrimos paraver con el comando *wxparaver*

3. Cargamos el fichero .prv generado en el paso 1

4. Especificamos una configuración “cfgs/User/APP_userevents_profile.cfg” ya que nos mostrará el valor de φ.

5. Al hacerlo nos abrirá la siguiente ventana:

	End	SERIAL	PARALLEL
THREAD 1.1.1	26.32 %	0.00 %	73.68 %
Total	26.32 %	0.00 %	73.68 %
Average	26.32 %	0.00 %	73.68 %
Maximum	26.32 %	0.00 %	73.68 %
Minimum	26.32 %	0.00 %	73.68 %
StDev	0 %	0 %	0 %
Avg/Max	1	1	1

φ = 73.68%

9. From the instrumented version of pi_omp.c, and using the appropriate Paraver configuration file, show a profile of the % of time spent in the different OpenMP states when using 8 threads and for 100.000.000 iterations. Clearly indicate which Paraver configuration file(s) did you use and your own conclusions from that profile.

1. Ejecutamos el comando ./submit-omp-i.sh pi_omp.c y nos generará un fichero .prv el cual debemos de abrir con paraver.

2. Abrimos paraver con el comando *wxparaver*

3. Cargamos el fichero .prv

4. Especificamos una configuración “cfgs/OpenMP/OMP_state_profile.cfg”.

5. Al hacerlo nos abrirá la siguiente ventana:

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	378,327,388 ns	-	86,201,459 ns	3,437,110 ns	16,871 ns	2,737 ns
THREAD 1.1.2	139,719,395 ns	260,310,171 ns	49,139,272 ns	-	9,803 ns	-
THREAD 1.1.3	130,531,654 ns	260,319,829 ns	58,296,074 ns	-	6,525 ns	-
THREAD 1.1.4	188,681,678 ns	260,322,534 ns	28,750 ns	-	5,818 ns	-
THREAD 1.1.5	111,993,534 ns	260,393,627 ns	76,777,462 ns	-	5,940 ns	-
THREAD 1.1.6	126,559,612 ns	260,320,111 ns	62,271,837 ns	-	6,081 ns	-
THREAD 1.1.7	165,234,012 ns	260,326,817 ns	23,595,289 ns	-	7,575 ns	-
THREAD 1.1.8	128,295,982 ns	260,360,371 ns	60,504,237 ns	-	6,128 ns	-
Total	1,369,343,255 ns	1,822,353,460 ns	416,814,380 ns	3,437,110 ns	64,741 ns	2,737 ns
Average	171,167,906.88 ns	260,336,208.57 ns	52,101,797.50 ns	3,437,110 ns	8,092.62 ns	2,737 ns
Maximum	378,327,388 ns	260,393,627 ns	86,201,459 ns	3,437,110 ns	16,871 ns	2,737 ns
Minimum	111,993,534 ns	260,310,171 ns	28,750 ns	3,437,110 ns	5,818 ns	2,737 ns
StDev	81,568,082.13 ns	27,674.83 ns	26,260,890.47 ns	0 ns	3,543.03 ns	0 ns
Avg/Max	0.45	1.00	0.60	1	0.48	1