Deliverable Lab1

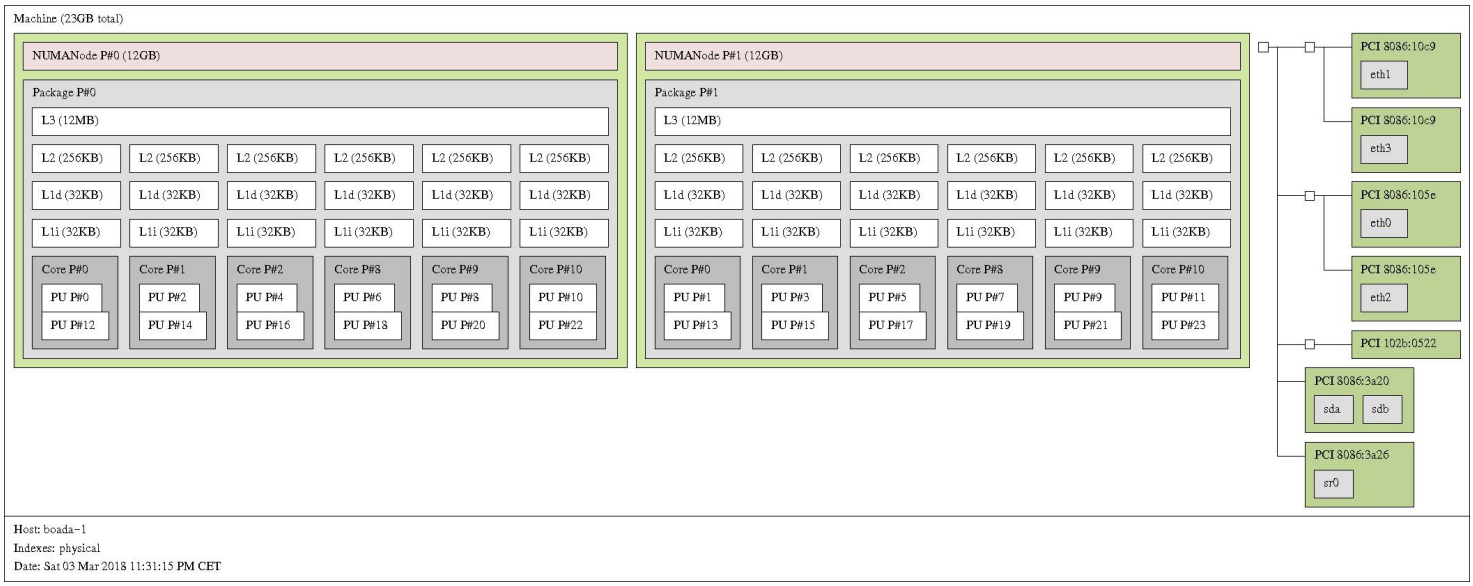
PAR 4113: Jordi Bosch, Dean Zhu

2018 Primavera

1 Node arquitecture and memory

	boada(1-4)	boada-5	boada(6-8)
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)	32K	32K	32K
L1-d cache size(per core)	32K	32K	32K
L2 cache size	256K	256K	256K
Last Level Cache size	12,288K	15360K	20480K
memory size(per socket)	12Gb	31Gb	16Gb
Main memory size(per node)	23Gb	64Gb	31Gb

Arquitecture diagram of node boada1



2 Timing sequential and parallel executions

2.1 Strong scalability

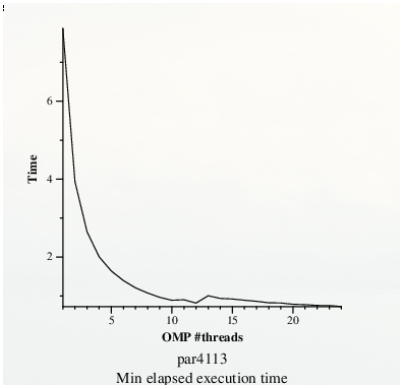


Figure 1: Execution: Temps d'execució, N = 10^9

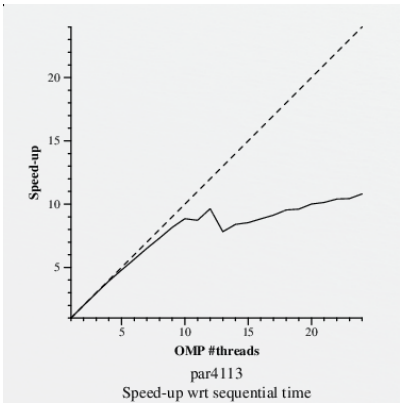


Figure 2: Execution: Speedup, N = 10^9

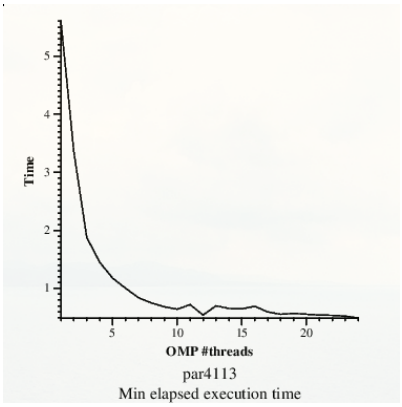


Figure 3: Cuda: Temps d'execució, N = 10^9

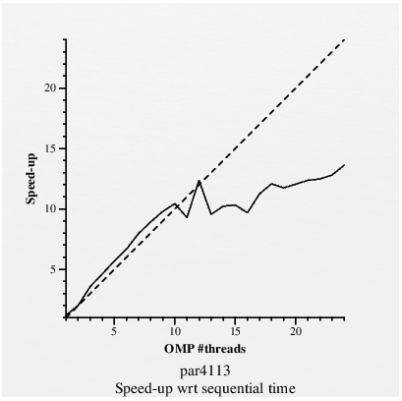


Figure 4: Cuda:Speedup, N = 10^9

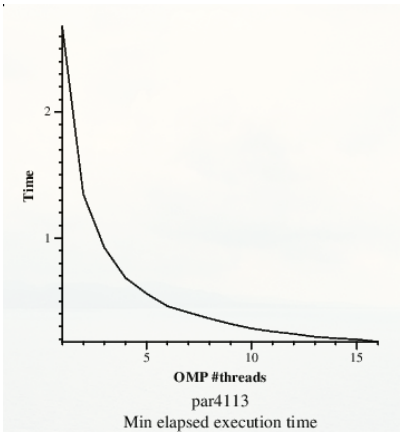


Figure 5: Execution2: Temps d'execució, N = 10^9

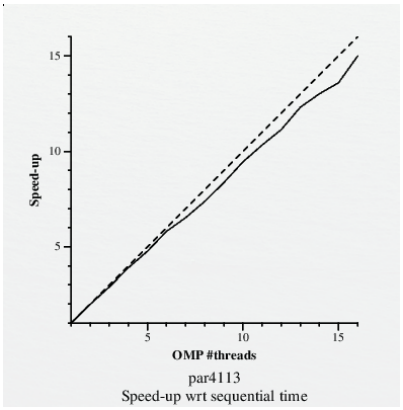


Figure 6: Execution2: Speedup, N = 10^9

Observant els 6 grafics anteriors amb el execution time i el speedup dels tres nodes, ens fixem que només Execution2 es comporta com esperariem d'un programa amb escalabilitat forta, és a dir, que el speedup sigui lineal en funció dels threads i el temps d'execució sigui inversament proporcional.
En canvi, pels nodes Execution i Cuda veiem que té escalabilitat forta fins els 11-12 threads, però quan utilitzem més threads el speedup deixa de ser lineal, i tot i que si utilitzem tots els CPUs disponibles, 24 en ambdos casos, el speedup no arriba al valor esperat senyalat per la linia discontínua. Una de les possibles raons per les quals passa això és que arribat a un cert punt el node necessita més temps per sincronitzar i planificar les tasques que la própia execució de les tasques i en conseqüencia ja no consegueix millorar la seva eficiència de la forma esperada. Això ho podem verificar aumentant la N al doble i veient que això permet que el speedup aumenti linealment fins a uns quants threads més.

2.2 Weak scalability

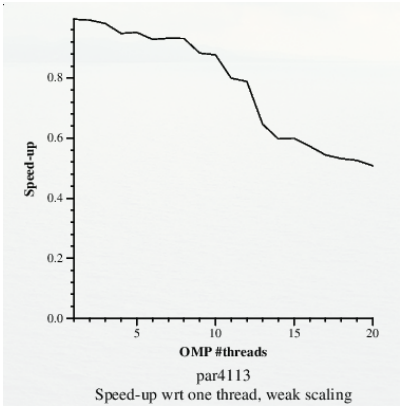


Figure 7: Execution: Speedup, N = $\#CPUs\cdot$

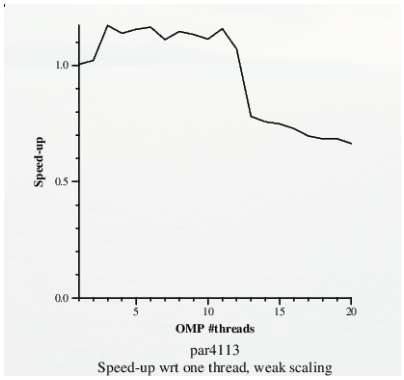


Figure 8: Cuda: Speedup, N = $\#CPUs \cdot 10^8$

10^8

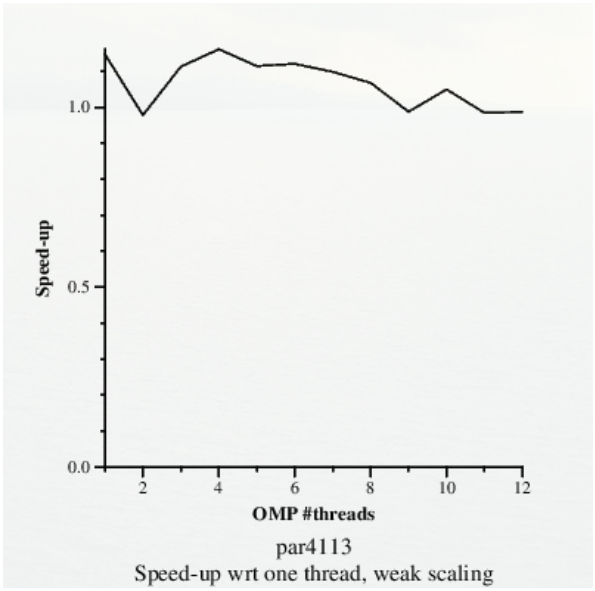


Figure 9: Execution2: Speedup, N = $\#CPUs \cdot 10^8$

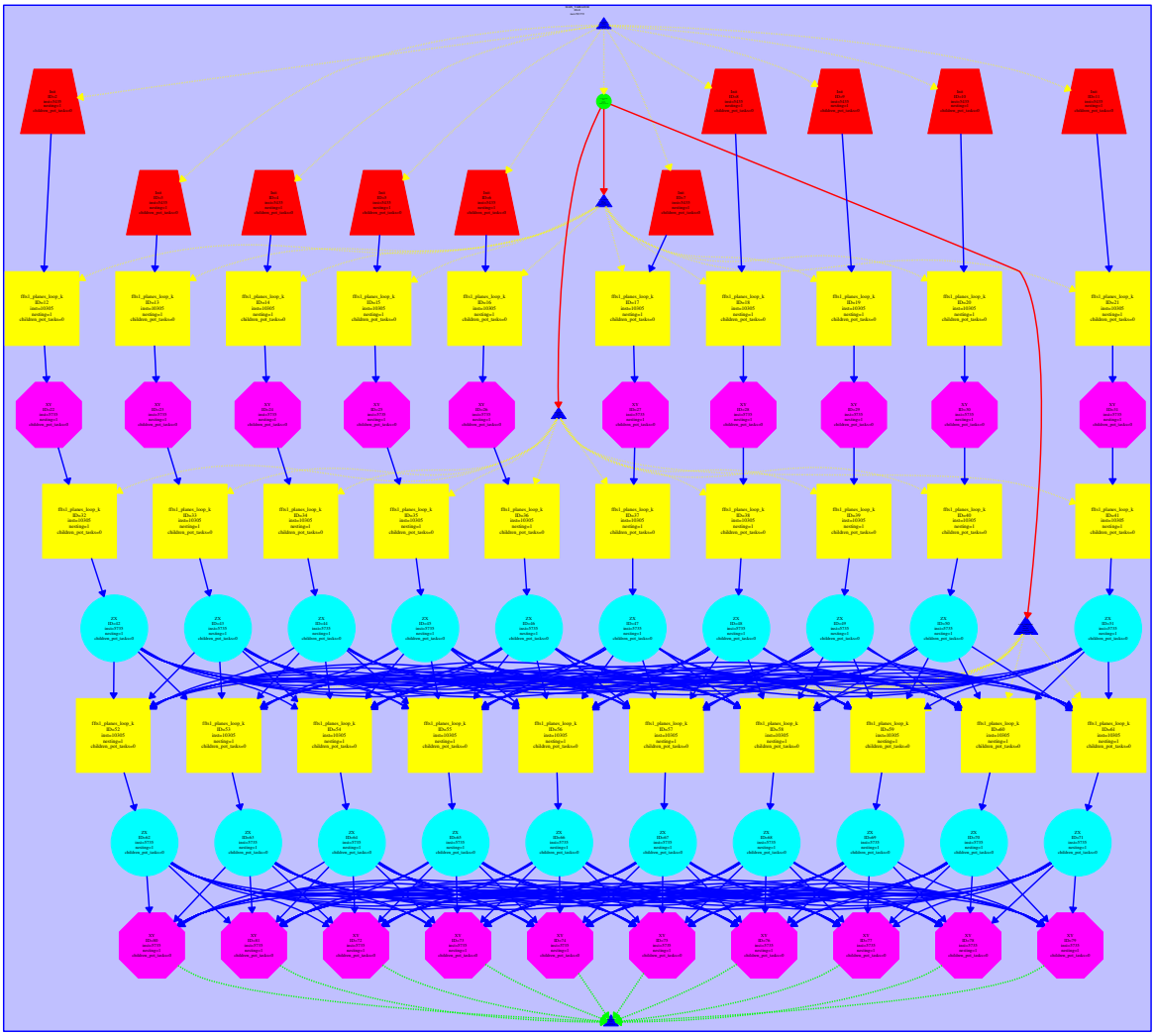
En les tres figures anteriors podem observar que un altre cop només Execution2 es comporta d'acord amb l'escabilitat feble. És a dir, quan la mida del problema creix proporcionalment amb el nombre de threads el speedup i per tant el temps d'execució es manté constant.
Si ens fixem en Execution i Cuda tornem a veure que a partir de 11 threads el temps requerit pel problema es major a la millora obtinguda al afegir un thread més. Un altre cop podem suposar que degut el *Overhead* de sincronitzar i planificar el nostre programa és massa gran i per tant el speedup respecte el nombre de threads no es manté constant.

3 Analysis of task decompositions with Tareador

Després de paral.lelitzar la majoria de funcions del programa, l'única subrutina que queda per tractar es *init_complex_grid*. Com en els altres casos farem que el loop-k es pugui executar en paral.lel.

```
1  void init_complex_grid(fftwf_complex in_fftw[][N][N]) {
2      int k,j,i;
3      for (k = 0; k < N; k++) {
4          tareador_start_task("Init");
5          for (j = 0; j < N; j++)
6              for (i = 0; i < N; i++) {
7                      in_fftw[k][j][i][0] = (float) (sin(M_PI*((float)i)/64.0)
8                      +sin(M_PI*((float)i)/32.0)+sin(M_PI*((float)i/16.0)));
9                      in_fftw[k][j][i][1] = 0;
10 #if TEST
11                     out_fftw[k][j][i][0]= in_fftw[k][j][i][0];
12                     out_fftw[k][j][i][1]= in_fftw[k][j][i][1];
13 #endif
14             }
15         tareador_end_task("Init");
16     }
17 }
```

I aquí tenim el graf de dependéncies final.



En la taula següent observem les millores del nostre programa a mesura que realitzem els canvis proposats. Ens fixem que el primer canvi no canvia el temps del execució del programa. Això pot voler dir que estem paral.lelitzant tasques dependents entre elles i per tant no poguem realitzar-les alhora. I de fet si mirem el graf de dependències generat podem verificar l'hipótesis També podem veure que T_1 es manté constant per tots els canvis que hem fet, és normal ja que amb una sola CPU només podem executar el programa seqüencialment.

Version	T_1 (ns)	T_∞ (ns)	Parallelism
seq	593772001	593705001	1.0001129
v1	593705001	593705001	1.0001129
v2	593705001	315188001	1.8838661
v3	593705001	108352001	5.4800280
v4	593705001	59415001	9.9936378

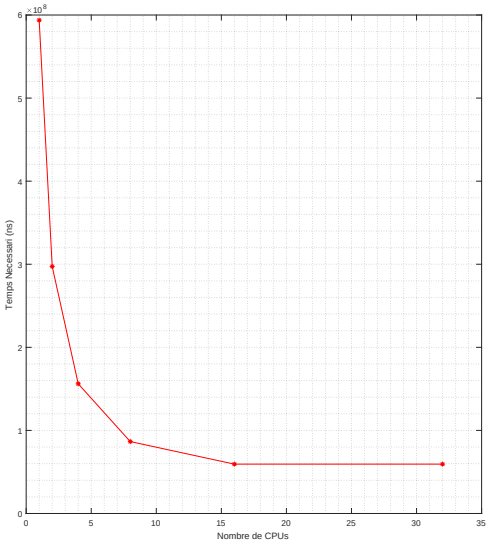


Figure 10: Temps necessari

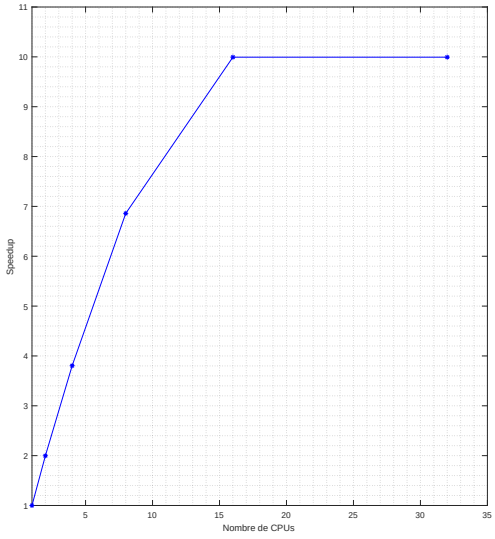


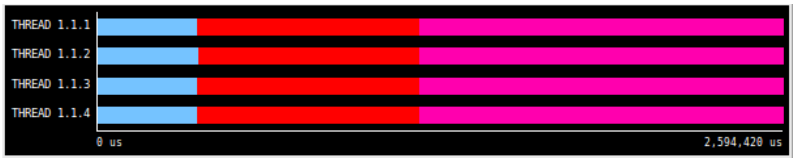
Figure 11: Speedup

Ens fixem que estem executant un programa amb un nombre de threads diferents però la mateixa mida, per tant només podem parlar de la escalabilitat forta. El speedup del nostre programa creix prou linealment excepte per quan passem de 16 threads a 32. Podem suposar que amb 16 threads ja paral.lelitzem el máxim, i la majoria del temps d'execució es degut a la part seqüencial, a més a més sabem que T_∞ serà un valor < 16. Per tant el programa te escalabilitat forta fins els 16 threads que és quan arribem a un màxim teóric.

4 Tracing the execution of parallel programs

Sabem que la fracció paral·lela es calcula com $\frac{T_{par}}{T_{total}} = \frac{T_{total} - T_{sequencial}}{T_{Total}}$

Representació gràfica de l'execució amb 4 nodes de 3dfftomp:



Repartiment dels temps entre els threads:

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	O
THREAD 1.1.1	2,566,200,421 ns	-	26,890,505 ns	1,291,721 ns	35,463 ns	2,
THREAD 1.1.2	1,976,537,032 ns	383,179,601 ns	211,802,035 ns	22,888,431 ns	13,473 ns	
THREAD 1.1.3	2,160,438,553 ns	379,253,699 ns	31,196,286 ns	23,522,216 ns	9,818 ns	
THREAD 1.1.4	2,104,836,693 ns	379,253,776 ns	87,344,933 ns	22,973,237 ns	11,933 ns	
Total	8,808,012,699 ns	1,141,687,076 ns	357,233,759 ns	70,675,605 ns	70,687 ns	2,
Average	2,202,003,174.75 ns	380,562,358.67 ns	89,308,439.75 ns	17,668,901.25 ns	17,671.75 ns	2,
Maximum	2,566,200,421 ns	383,179,601 ns	211,802,035 ns	23,522,216 ns	35,463 ns	2,
Minimum	1,976,537,032 ns	379,253,699 ns	26,890,505 ns	1,291,721 ns	9,818 ns	2,
StDev	220,592,134.19 ns	1,850,669.80 ns	74,635,040.74 ns	9,458,498.76 ns	10,353.41 ns	
Avg/Max	0.86	0.99	0.42	0.75	0.50	

Veiem que la part sequencial(la part blau clara de la primera figura) tarda 379,253,776ns. Ara volem saber quan tarda en total el programa. Per això necessitem executar-lo tot en un node, ja que si multipliquem el temps que tarda en aquest cas per 4(el número de nodes que tenim), estem sumant temps extra com ara sincronitzacions i forks/joins que no tenen perquè comportar-se exactament lineal. A la figura de baix tenim el resultat en el cas que només ho executem en un node.

	Running	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	7,459,210,483 ns	39,596 ns	91,813 ns	38,122 ns	2,362 ns
Total	7,459,210,483 ns	39,596 ns	91,813 ns	38,122 ns	2,362 ns
Average	7,459,210,483 ns	39,596 ns	91,813 ns	38,122 ns	2,362 ns
Maximum	7,459,210,483 ns	39,596 ns	91,813 ns	38,122 ns	2,362 ns
Minimum	7,459,210,483 ns	39,596 ns	91,813 ns	38,122 ns	2,362 ns
StDev	0 ns	0 ns	0 ns	0 ns	0 ns
Avg/Max	1	1	1	1	1

Podem veure que l'execució del programa en total tarda 7,459,210,483 ns.

$$\phi = \frac{T_{total} - T_{sequencial}}{T_{total}} = \frac{7459,219483 - 379,353776}{7459,219489} = 0.949 = 95\%$$

OpenMP Statistics @ 3dfft_omp-4-boada-2.prv

	Running	Not created	Synchronization	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	98.79 %	-	1.07 %	0.14 %	0.00 %	0.00 %
THREAD 1.1.2	79.46 %	18.11 %	1.63 %	0.79 %	0.00 %	-
THREAD 1.1.3	80.33 %	18.11 %	0.75 %	0.81 %	0.00 %	-
THREAD 1.1.4	80.22 %	18.11 %	0.85 %	0.81 %	0.00 %	-
Total	338.81 %	54.33 %	4.30 %	2.55 %	0.00 %	0.00 %
Average	84.70 %	18.11 %	1.08 %	0.64 %	0.00 %	0.00 %
Maximum	98.79 %	18.11 %	1.63 %	0.81 %	0.00 %	0.00 %
Minimum	79.46 %	18.11 %	0.75 %	0.14 %	0.00 %	0.00 %
StDev	8.14 %	0.00 %	0.34 %	0.29 %	0.00 %	0 %
Avg/Max	0.86	1.00	0.66	0.79	0.52	1

Aquests són els resultats en tant per cent. Veiem que el primer thread és el que executa el programa sequencialment. Mentres els altres tres threads esperen un 18% del rato a ser creats. En aquest punt és on comença el programa paral·lel. Veiem que durant una part del temps, els threads s'esperen per sincronitzar-se(1.08%) o bé s'ajunten(0.64%), com en el cas d'unificar variabes per exemple. En aquest programa no requerim d'entrada ni de sortida per tant no s'hi dedica temps de CPU.

5 Annex

5.1 Escalabilitat forta amb N = $2 \cdot 10^9$

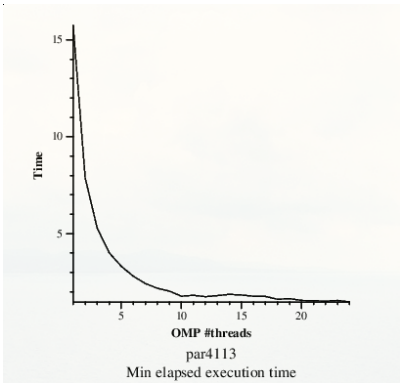


Figure 12: Execution: Temps d'execució, N = $2 \cdot 10^9$

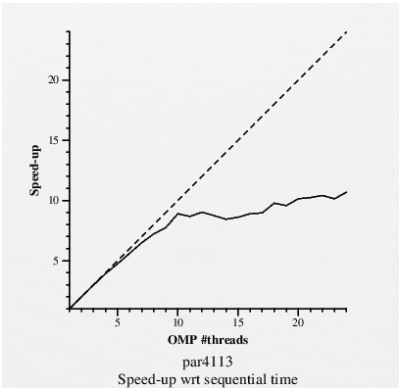


Figure 13: Execution: Speedup, N = $2 \cdot 10^9$

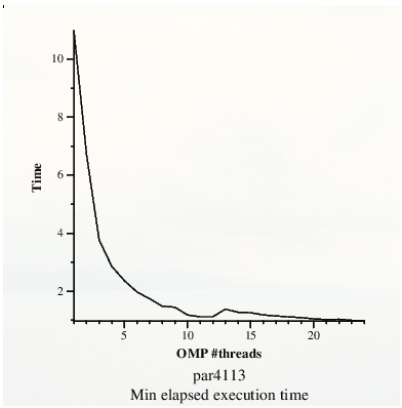


Figure 14: Cuda:Temps d'execució, N = $2 \cdot 10^9$

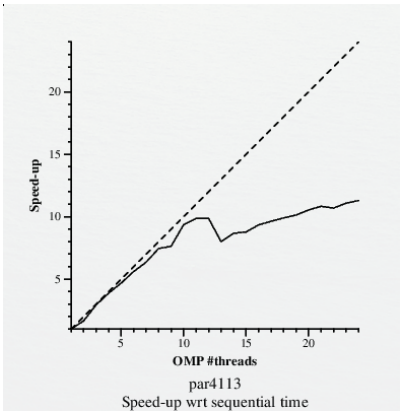


Figure 15: Cuda:Speedup, N = $2 \cdot 10^9$

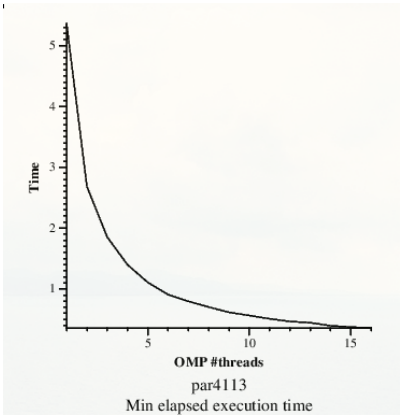


Figure 16: Execution2: Temps d'execuciò, N $= 2 \cdot 10^9$

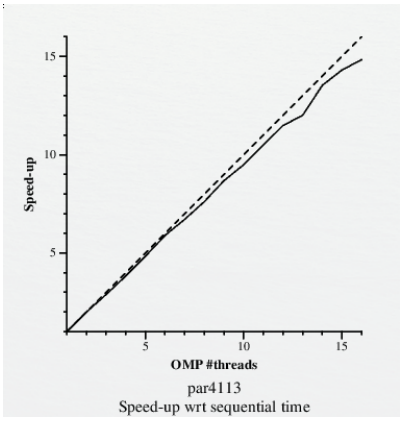


Figure 17: Execution2: Speedup, N $= 2 \cdot 10^9$