Lab5 — *Data Decomposition*

Jordi Navarrete – Edgar Aguadé

	Number of threads					
Number of threads	**1**	**4**	**8**	**16**	**24**	**32**
1D Block geometric data decomposition by columns	3.227187	2.044298	1.404713	0.790876	0.538073	0.399483
1D Block-cyclic geometric data decomposition by columns	3.231608	0.811812	0.410160	0.214676	0.143246	0.124042
1D Cyclic geometric data decomposition by rows	3.225763	0.807867	0.404614	0.203721	0.137063	0.104298
Version	**Number of threads (L2 cache misses per thread)**					
1D Block geometric data decomposition by columns	1613037	474797	279594	159414	119436	61602
1D Block-cyclic geometric data decomposition by columns	1613085	1148348	205970	103352	69167	52020
1D Cyclic geometric data decomposition by rows	1612496	403448	202062	101950	68396	51472
Best parallel Implementation	**Version**	**Reason why**				
	Cyclic	Com s'evidencia a l'anàlisi, resulta ser l'estratègia que obté millors temps, eficiència i escalabilitat. En ser la repartició per files senceres, resulta molt satisfactòria per aquest problema en concret, no obstant això, en cas d'haver repartit per elements "solts" resultaria un desastre d'estratègia. *A l'apartat de l'anàlisi de l'estratègia s'explica amb més profunditat el perquè és la millor opció.*				

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Index

1D Block geometric data decomposition by columns

mandel-omp-iter-simple-block.cpp

Modelfactor analysis

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	3.24	2.35	2.06	1.42	1.02	0.81	0.66	0.55	0.48	0.41
Speedup	1.00	1.38	1.57	2.28	3.18	4.02	4.88	5.87	6.81	7.83
Efficiency	1.00	0.69	0.39	0.29	0.26	0.25	0.24	0.24	0.24	0.24

Table 1: Analysis done on Thu Dec 18 01:01:21 PM CET 2025, par1301

Veiem com el temps d'execució va baixant segons el nombre de threads començant per 3.24 s, fins a aconseguir uns 0.41 s als 32 *threads*. També veiem com l'speedup escala conseqüentment. Així i tot, veiem una baixada de l'eficiència segons augmentem el threads, indicant que la feina feta total no escala de la mateixa manera segons anem afegint threads.

Overview of the Efficiency metrics in parallel fraction, ϕ=99.60%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	100.00%	68.99%	39.45%	28.70%	26.73%	25.48%	24.82%	24.93%	24.94%	25.21%
Parallelization strategy efficiency	100.00%	69.04%	39.49%	28.74%	26.80%	25.69%	25.21%	25.49%	25.78%	25.88%
Load balancing	100.00%	69.04%	39.50%	28.76%	26.83%	25.72%	25.26%	25.54%	25.85%	25.98%
In execution efficiency	100.00%	99.99%	99.98%	99.95%	99.89%	99.89%	99.82%	99.77%	99.72%	99.63%
Scalability for computation tasks	100.00%	99.93%	99.89%	99.85%	99.76%	99.16%	98.42%	97.81%	96.77%	97.40%
IPC scalability	100.00%	100.00%	100.00%	100.00%	99.98%	99.49%	98.94%	98.54%	97.77%	98.64%
Instruction scalability	100.00%	100.00%	100.00%	100.00%	99.99%	99.99%	99.99%	99.99%	99.98%	99.98%
Frequency scalability	100.00%	99.93%	99.89%	99.85%	99.79%	99.68%	99.49%	99.27%	99.00%	98.77%

Table 2: Analysis done on Thu Dec 18 01:01:21 PM CET 2025, par1301

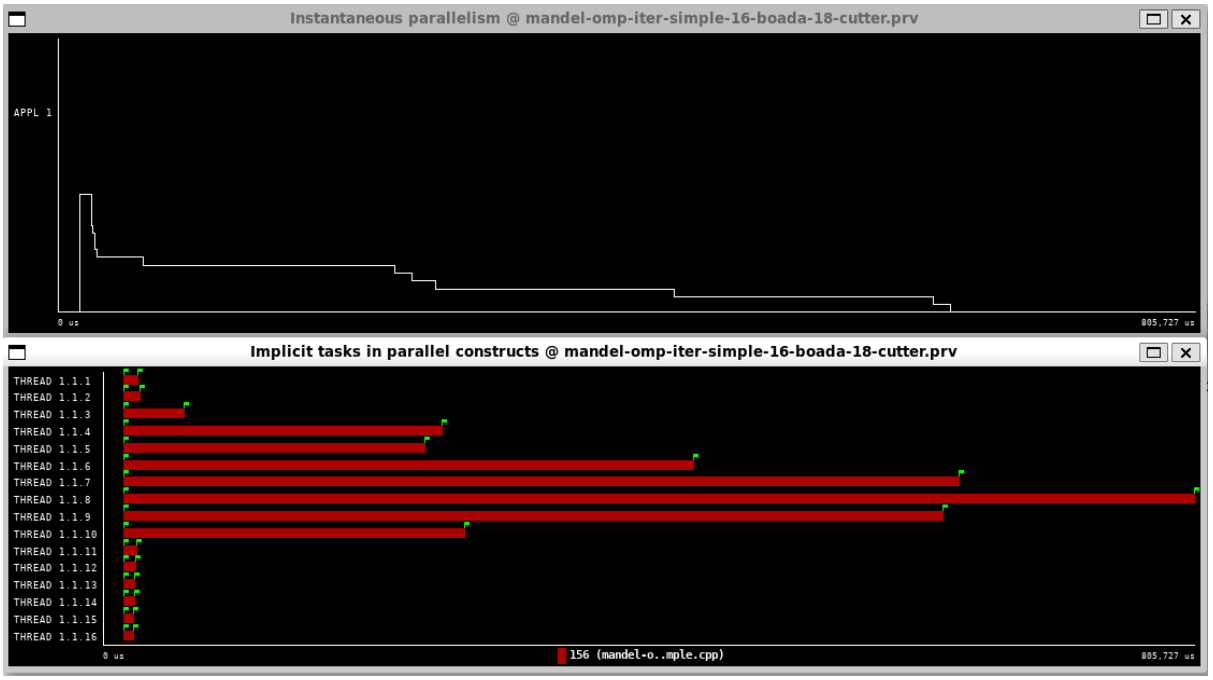
Pel que veiem en aquesta taula, deduïm un balanceig de càrrega bastant pobre, que es deu a com es fa el Mandelbrot Set i ens indica que alguns blocs carregaran amb més feina que d'altres. Això és fàcilment observable a simple vista amb la imatge del Mandelbrot. Aquest fet produeix que l'eficiència global es vegi molt afectada.

Overheads in executing implicit tasks										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of implicit tasks per thread (average us)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Useful duration for implicit tasks (average us)	3226845.68	1614487.04	807610.56	403980.69	269541.45	203381.16	163931.42	137458.31	119087.01	103530.26
Load balancing for implicit tasks	1.0	0.69	0.39	0.29	0.27	0.26	0.25	0.26	0.26	0.26
Time in synchronization implicit tasks (average us)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Time in fork/join implicit tasks (average us)	111.61	156.27	1749961.01	1386325.47	993404.31	781632.19	642287.58	532649.24	455469.43	394833.37

Table 3: Analysis done on Thu Dec 18 01:01:21 PM CET 2025, par1301

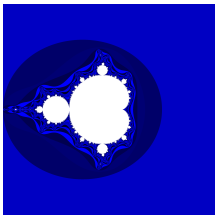
Aquí veiem com el "*useful duration for implicit tasks*" baixa gràcies a la repartició de feina, encara així tornem a veure com baixa el load balancing per culpa del desbalanceig de càrrega de feina per cada tasca.

Paraver analysis



Paraver de 16 threads

Aquí podem observar com aconseguim el màxim paral·lelisme al principi i segons els *threads* amb menor feina van acabant les seves feines, el paral·lelisme total va baixant, fet que ens indica que hi ha tasques que finalitzen molt abans que d'altres, donant això més èmfasi en el desbalanceig de càrrega.



Això és el que podem observar en el segon diagrama, que s'observen les tasques que fan més feina tot coincidint amb la imatge del Mandelbrot si observem les columnes.

Memory analysis

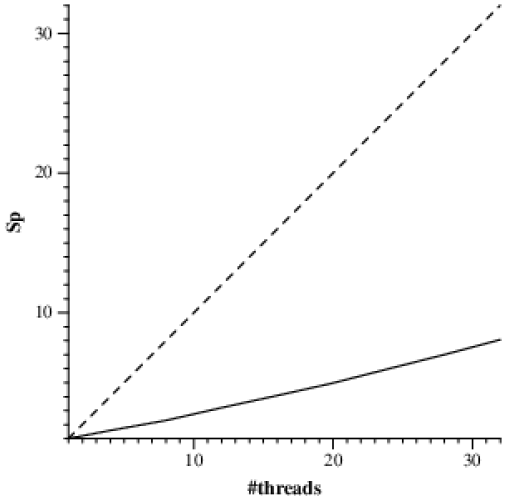
	[0.00..999,999,999,999,999,983,222,784.00]
THREAD 1.1.1	201,321
THREAD 1.1.2	198,150
THREAD 1.1.3	121,809
THREAD 1.1.4	141,333
THREAD 1.1.5	185,411
THREAD 1.1.6	141,568
THREAD 1.1.7	124,412
THREAD 1.1.8	196,824
THREAD 1.1.9	162,444
THREAD 1.1.10	103,154
THREAD 1.1.11	189,704
THREAD 1.1.12	183,636
THREAD 1.1.13	102,974
THREAD 1.1.14	192,927
THREAD 1.1.15	193,280
THREAD 1.1.16	112,625
Total	2,551,572
Average	159,473.25
Maximum	201,321
Minimum	102,974
StDev	36,176.58
Avg/Max	0.79

Thread	Misses Totals	Misses per thread
1	1613037	1613037
2	1722429	861214
4	1899190	474797
8	2236758	279594
12	2618980	218248
16	2550637	159414
20	2850799	142539
24	2866479	119436
28	3007952	107426
32	1971272	61602

En dividir les files amb més tasques també es divideixen les línies de cache entre els processadors, el que fa que els *miss* totals vagin augmentant segons els *threads* ho fan. Això per culpa del *false sharing* i les constants invalidacions provocades per mantenir la coherència.

Així i tot, la relació amb els *miss/thread* es veu com evoluciona amb més threads, ja que en fer l'execució per blocs s'aprofita un xic la localitat física de les dades.

Strong scalability



par1301
Speed-up wrt sequential time (mandel funtion only)
Generated by par1301 on Thu Dec 11 01:50:48 PM CET 2025

En la gràfica de *strong scalability* podem observar com l'speedup en relació amb el número de threads, quan el comparem amb l'ideal, és força baix. Que com hem pogut observar amb anterioritat es pot deure al fet que no s'optimitza la càrrega de feina de cada thread i, per tant, el desbalanceig de càrrega provoca que tinguem aquesta baixa escalabilitat.

1D Block-cyclic geometric data decomposition by columns

mandel-omp-iter-simple-blockcyclic.cpp

Modelfactor analysis

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	3.25	1.63	0.83	0.43	0.29	0.23	0.19	0.16	0.14	0.14
Speedup	1.00	1.99	3.92	7.61	11.21	14.20	17.44	20.59	22.80	23.36
Efficiency	1.00	0.99	0.98	0.95	0.93	0.89	0.87	0.86	0.81	0.73

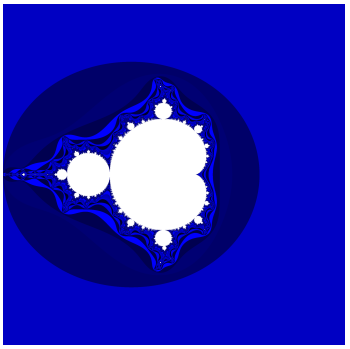
Table 1: Analysis done on Sat Dec 20 01:46:53 AM CET 2025, par1316

A simple vista és observable com aquesta estratègia aconsegueix un speed-up molt superior a l'estratègia anterior, *block geometric data decomposition*. Això és degut a quelcom força simple: Cada thread tracta tants elements com li caben a una línia de *cache*. D'aquesta manera, un thread no invalida línies de la cache d'un altre thread.

Overview of the Efficiency metrics in parallel fraction, ϕ=99.60%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	100.00%	99.79%	99.49%	98.42%	97.32%	93.90%	93.32%	93.15%	89.15%	80.48%
Parallelization strategy efficiency	100.00%	99.96%	99.80%	98.74%	97.83%	94.63%	94.33%	94.37%	90.57%	82.30%
Load balancing	100.00%	99.98%	99.83%	98.90%	98.13%	95.00%	95.04%	95.30%	91.56%	83.29%
In execution efficiency	100.00%	99.99%	99.97%	99.85%	99.70%	99.61%	99.25%	99.02%	98.91%	98.82%
Scalability for computation tasks	100.00%	99.83%	99.69%	99.68%	99.48%	99.23%	98.94%	98.71%	98.44%	97.79%
IPC scalability	100.00%	99.94%	99.83%	99.81%	99.69%	99.58%	99.49%	99.37%	99.43%	99.10%
Instruction scalability	100.00%	100.00%	100.00%	100.00%	99.99%	99.99%	99.99%	99.99%	99.98%	99.98%
Frequency scalability	100.00%	99.89%	99.87%	99.87%	99.79%	99.66%	99.45%	99.35%	99.02%	98.70%

Table 2: Analysis done on Sat Dec 20 01:46:53 AM CET 2025, par1316

Pel que fa al balanceig de càrrega, podem veure que està força ben repartit. D'igual manera l'eficiència ha millorat per igual. El motiu és que les iteracions es reparteixen amb una granularitat més fina, reduint la diferència de càrrega entre threads, ja que, a diferència del comentat anteriorment, ara no hi ha threads que únicament computin "part blava" de la imatge, la qual és la "zona" de menor cost.



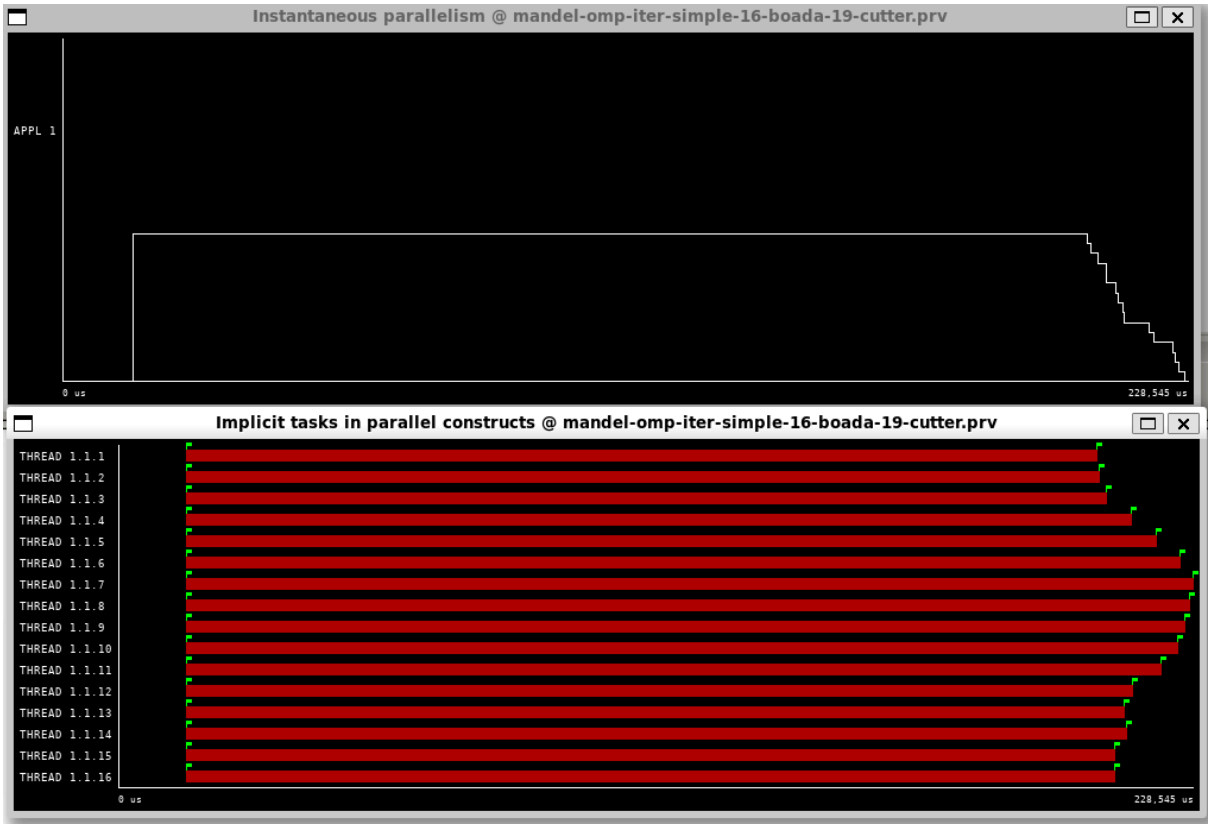
Overheads in executing implicit tasks										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of implicit tasks per thread (average us)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Useful duration for implicit tasks (average us)	3231902.68	1618756.37	810488.01	405303.0	270744.31	203558.93	163330.35	136425.98	117259.46	103281.86
Load balancing for implicit tasks	1.0	1.0	1.0	0.99	0.98	0.95	0.95	0.95	0.92	0.83
Time in synchronization implicit tasks (average us)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Time in fork/join implicit tasks (average us)	105.22	958.65	2025.72	5764.88	6764.4	21450.99	16694.54	15526.67	22376.89	39689.27

Table 3: Analysis done on Sat Dec 20 01:46:53 AM CET 2025, par1316

Com és d'esperar, observem que a mesura que augmenten els threads disminueix el *"useful duration for implicit tasks"* ja que, en ser el mateix nombre d'iteracions a repartir entre més *workers*, a *worker* li correponen menys iteracions. Alhora, observem que el temps de

sincronització augmenta, tot i que en ser poques tasques i no haver-hi gaire data-sharing és força petit.

Paraver analysis



Paraver de 16 threads

Com bé podem veure a l'*instantaneous parallelism*, gràcies a la bona distribució de la càrrega aconseguim explotar molt el paral·lelisme. Així i tot, veiem una petita davallada al final, això causa d'aquells threads que computen les parts més costoses del *conjunt de Mandelbrot*, fent que aquells qui acaben abans esperin (com podem veure a l'*implicit task in parallel constructs*). Tot i això, el desbalanceig és força baix.

Memory analysis

Thread	Misses Totals	Misses per thread
1	1613085	1613085
2	3163718	1581859
4	4593394	1148348
8	1647765	205970
12	1652405	137700

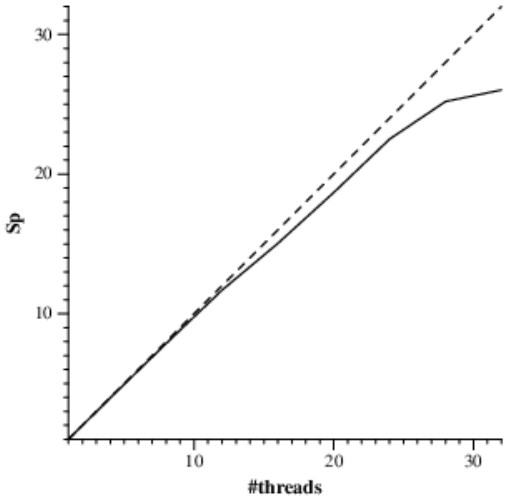
16	1653643	103352
20	1657952	82897
24	1660017	69167
28	1663021	59393
32	1664665	52020

Com podem observar, el nombre de *miss/thread* disminueix a mesura que augmentem els threads. Com és d'esperar per dos motius: cada thread fa menys accessos a memòria en ser més *workers* pels mateixos elements i, el més important, el nombre d'iteracions consecutives que realitza cada thread coincideixen a la perfecció amb la mida de línia de cache, evitant el *false-sharing*.

	[0.00..999,999,999,999,999,983,222,784.00]
THREAD 1.1.1	103,235
THREAD 1.1.2	102,975
THREAD 1.1.3	102,850
THREAD 1.1.4	102,811
THREAD 1.1.5	102,869
THREAD 1.1.6	103,101
THREAD 1.1.7	102,867
THREAD 1.1.8	102,953
THREAD 1.1.9	102,935
THREAD 1.1.10	102,930
THREAD 1.1.11	102,864
THREAD 1.1.12	102,892
THREAD 1.1.13	102,934
THREAD 1.1.14	102,998
THREAD 1.1.15	102,926
THREAD 1.1.16	102,988
Total	1,647,128
Average	102,945.50
Maximum	103,235
Minimum	102,811
StDev	100.96
Avg/Max	1.00

A la figura podem observar com a diferència de l'anterior estratègia, el nombre de *miss* es manté força uniforme en tots els threads. Podem observar-ho també veient que el màxim son 103.235, el mínim son 102.811 i hi ha una desviació estandar de 101 aproximadament. Això evidència novament com el fet de fer coincidir la mida de línia amb la mida del bloc a tractar evita que uns threads estiguin invalidant les línies dels altres.

Strong scalability



par1316
Speed-up wrt sequential time (mandel funtion only)
Generated by par1316 on Sat Dec 20 01:39:55 AM CET 2025

En comparativa amb l'estratègia anterior, podem observar una gran millora en l'speed-up, essent aquest gairebé l'ideal. Això, com bé ja s'ha comentat, és gràcies al fet que s'hi aconsegueix un bon balanceig de càrrega.

1D Cyclic geometric data decomposition by rows

mandel-omp-iter-simple-cyclic.cpp

Modelfactor analysis

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	3.24	1.63	0.82	0.42	0.28	0.22	0.18	0.15	0.13	0.12
Speedup	1.00	1.99	3.94	7.75	11.41	14.89	18.26	21.47	24.45	27.42
Efficiency	1.00	0.99	0.98	0.97	0.95	0.93	0.91	0.89	0.87	0.86

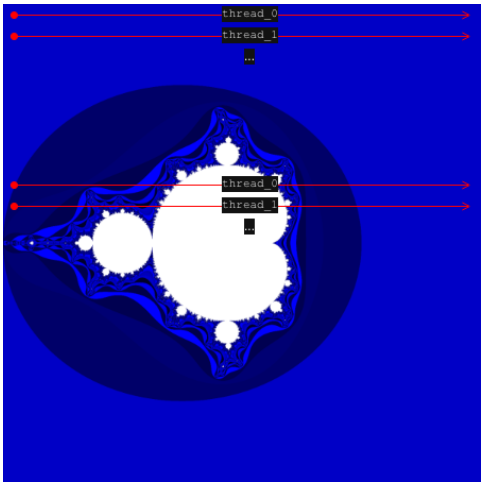
Table 1: Analysis done on Thu Dec 18 01:55:22 PM CET 2025, par1316

Com és ben observable, en aquesta estratègia el temps també decreix a mesura que el nombre de *threads* augmenta, arribant a aconseguir el millor (més baix) de totes tres estratègies quan és executada amb 32 *threads*. Per altra banda, podem observar també com l'eficiència és força millor que en ambdues estratègies anteriors, aconseguint una eficiència del 86% en l'execució de 32 *threads*. Això últim ens fa pensar que per algun motiu la càrrega de treball s'està distribuint d'una manera més uniforme en aquest cas.

Overview of the Efficiency metrics in parallel fraction, ϕ=99.61%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	100.00%	99.65%	99.72%	99.56%	99.25%	98.55%	98.14%	97.45%	96.61%	95.81%
Parallelization strategy efficiency	100.00%	99.85%	99.86%	99.70%	99.52%	99.14%	98.92%	98.51%	98.00%	98.03%
Load balancing	100.00%	99.87%	99.91%	99.88%	99.79%	99.73%	99.61%	99.43%	99.25%	99.37%
In execution efficiency	100.00%	99.98%	99.95%	99.82%	99.73%	99.40%	99.30%	99.07%	98.74%	98.65%
Scalability for computation tasks	100.00%	99.80%	99.86%	99.85%	99.73%	99.41%	99.21%	98.93%	98.58%	97.73%
IPC scalability	100.00%	99.92%	99.97%	99.98%	99.95%	99.76%	99.65%	99.61%	99.52%	99.02%
Instruction scalability	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Frequency scalability	100.00%	99.87%	99.89%	99.88%	99.79%	99.66%	99.56%	99.32%	99.05%	98.70%

Table 2: Analysis done on Thu Dec 18 01:55:22 PM CET 2025, par1316

Pel que fa a l'eficiència podem veure com hi ha hagut una gran millora en tots els aspectes, obtenint en tots casos valors propers a una eficiència del 100%. Quant al *Load Balancing*, observem que és l'estratègia que millors resultats té amb un 99.37 % en l'execució de 32 *threads*. Si ens fixem en la imatge que s'està computant és evident veure el perquè:



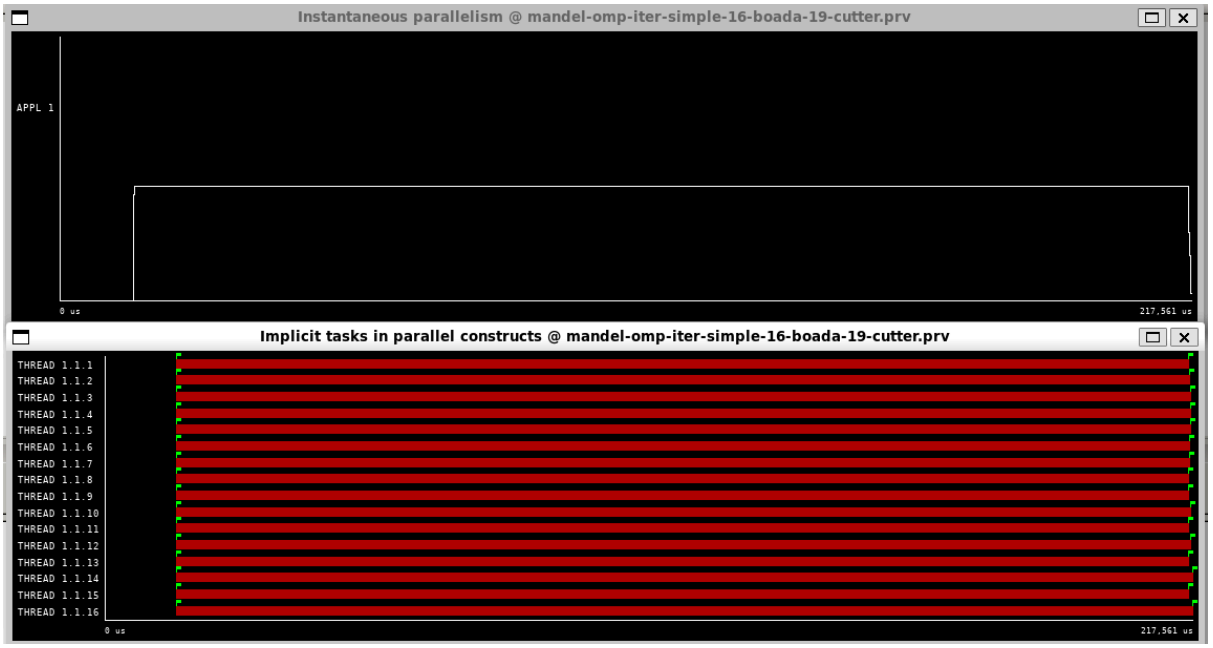
Com podem observar, la complexitat de còmput en files consecutives és molt similar. Mentre que en repartir en "grups de columnes" la banda dreta és força més simple que la banda esquerra, en repartir en files consecutives en una mateixa iteració tots els threads estan fent una quantitat de feina similar. És a dir, a l'inici, tots estan computant files blaves, al centre, tots estan computant files del centre (que tenen una dificultat similar) i al final, tots estan computant files blaves.

Overheads in executing implicit tasks										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of implicit tasks per thread (average us)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Useful duration for implicit tasks (average us)	3226204.86	1616359.68	807665.57	403864.5	269568.42	202830.01	162592.85	135879.72	116883.12	103159.83
Load balancing for implicit tasks	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	0.99	0.99
Time in synchronization implicit tasks (average us)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Time in fork/join implicit tasks (average us)	105.42	4391.01	868.2	1089.49	1703.96	1953.66	2584.36	3071.86	3202.5	4128.23

Table 3: Analysis done on Thu Dec 18 01:55:22 PM CET 2025, par1316

Pel que fa als *overheads*, novament veiem que, tot i créixer amb el nombre de *threads*, ja que hi ha tantes tasques com *threads*, no causen un gran impacte.

Paraver analysis



Paraver de 16 threads

Com bé s'ha comentat, la càrrega està molt ben distribuïda, cosa que podem evidenciar en observar l'*instantaneous parallelism*. Podem veure com tots els threads comencen alhora, estan "fent feina" tota l'estona i acaben tots pràcticament alhora.

De l'*implicit task in parallel constructs* podem observar com el petit desbalanceig que apareixia en l'estratègia anterior en els *threads* 4...11 ha desaparegut, això a gràcies al que s'ha explicat al punt anterior sobre la repartició de les iteracions.

Memory analysis

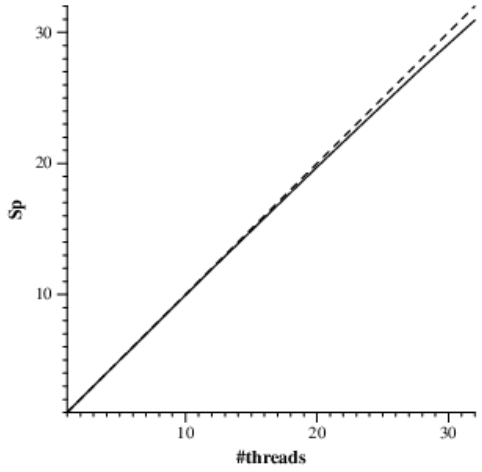
	[0.00..999,999,999,999,999,983,222,784.00]
THREAD 1.1.1	102,097
THREAD 1.1.2	102,115
THREAD 1.1.3	100,887
THREAD 1.1.4	101,911
THREAD 1.1.5	102,082
THREAD 1.1.6	101,207
THREAD 1.1.7	100,541
THREAD 1.1.8	102,229
THREAD 1.1.9	101,946
THREAD 1.1.10	101,101
THREAD 1.1.11	100,881
THREAD 1.1.12	102,006
THREAD 1.1.13	101,489
THREAD 1.1.14	100,999
THREAD 1.1.15	101,983
THREAD 1.1.16	101,980
Total	1,625,454
Average	101,590.88
Maximum	102,229
Minimum	100,541
StDev	544.00
Avg/Max	0.99

Thread	Misses Totals	Misses per thread
1	1612496	1612496
2	1618335	809167
4	1613795	403448
8	1616503	202062
12	1624823	135401
16	1631211	101950
20	1640505	82025
24	1641527	68396
28	1652789	59028
32	1647129	51472

Pel que fa a la memòria, és important destacar que tot i que el nombre de *miss/thread* sigui similar a l'estratègia anterior, hem aconseguit una gran millora en temps d'execució, *speedup* i eficiència. Això pensem que, a banda del bon *Load Balancing*, pot ser degut al fet que cada *thread* executi una fila completa de la matriu. Això permet que la memòria exploti

satisfactòriament la localitat espacial, ja que un mateix thread estarà accedint blocs consecutius a memòria.

Strong scalability



Finalment, podem observar com obtenim una *strong scalability* molt bona, obtenint pràcticament l'speed-up ideal. Això, juntament amb tot l'anteriorment comentat, evidencia que bona que resulta aquesta última estratègia per a resoldre aquest problema en concret.