PAR LABORATORY 5 REPORT

Q1 2025/2026

Arnau Hernández Navarro and Felipe Arturo Núñez Gómez

INDEX

1.Brief Introduction

In this fifth laboratory, we focus on parallel data decomposition strategies, and as in the previous two sessions, we continue using the Mandelbrot Set. This session is divided into three different strategies:

- **Block Geometric Data Decomposition by columns**: Made to minimize load unbalance due to the data decomposition and exploit data locality accessing matrix M as much as possible.
- **Bloc-Cyclic Geometric Data decomposition by columns**: Made to reduce cache coherence protocol overheads and exploit data locality accessing matrix M as much as possible.
- **Cyclic Geometric Data Decomposition by rows**: Made to minimize load unbalance, reduce cache coherence protocol overheads and exploit data locality accessing matrix M as much as possible.

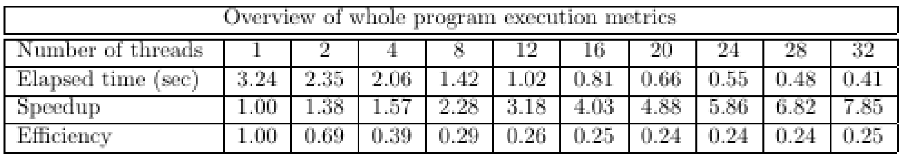
For each strategy we will be conducting an analysis that includes the following:

- **Modelfactor analysis**: Evaluates the code performance.
- **Paraver analysis**: Execution observations.
- **Memory analysis**: Evaluates L2 cache performance.
- **Strong scalability**: Thread scalability.

In each analysis, we will make a comparison between the different strategies and at the end we will include a table containing a brief summary of the explained analysis.

2.Block Geometric Data Decomposition by Columns

The modelfactor analysis reveals the fundamental efficiency issues inherent in the first strategy.

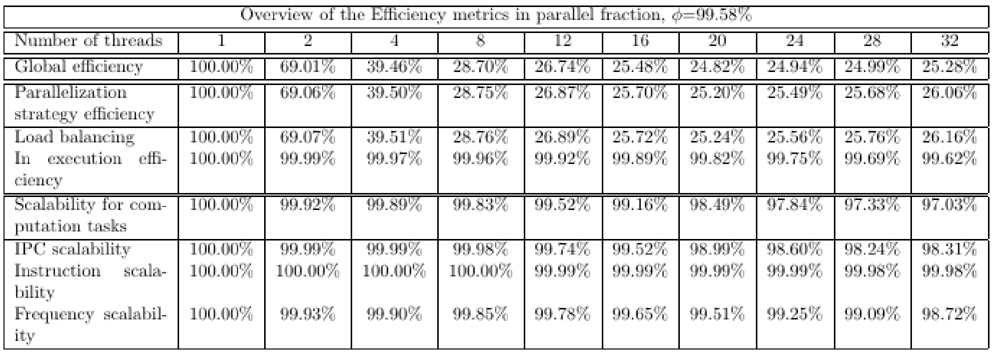


Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	3.24	2.35	2.06	1.42	1.02	0.81	0.66	0.55	0.48	0.41
Speedup	1.00	1.38	1.57	2.28	3.18	4.03	4.88	5.86	6.82	7.85
Efficiency	1.00	0.69	0.39	0.29	0.26	0.25	0.24	0.24	0.24	0.25

Table 1: Analysis done on Tue Dec 9 11:41:38 AM CET 2025, par3113

In this first strategy, we observe a severe degradation in Efficiency as the number of threads increases. With 32 threads, the efficiency drops to 25%, and the speedup is only 7.82 far below the ideal 32.

Compared with the other strategies that will follow this one, Block-Cyclic Columns maintains an efficiency of 73%-80% at 32 threads, while the Cyclic Rows achieves the best performance with an efficiency of 85%-95% at 32 threads.



Overview of the Efficiency metrics in parallel fraction, ϕ=99.58%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	100.00%	69.01%	39.46%	28.70%	26.74%	25.48%	24.82%	24.94%	24.99%	25.28%
Parallelization strategy efficiency	100.00%	69.06%	39.50%	28.75%	26.87%	25.70%	25.20%	25.49%	25.68%	26.06%
Load balancing	100.00%	69.07%	39.51%	28.76%	26.89%	25.72%	25.24%	25.56%	25.76%	26.16%
In execution efficiency	100.00%	99.99%	99.97%	99.96%	99.92%	99.89%	99.82%	99.75%	99.69%	99.62%
Scalability for computation tasks	100.00%	99.92%	99.89%	99.83%	99.52%	99.16%	98.49%	97.84%	97.33%	97.03%
IPC scalability	100.00%	99.99%	99.99%	99.98%	99.74%	99.52%	98.99%	98.60%	98.24%	98.31%
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%	99.90%	99.85%	99.78%	99.65%	99.51%	99.25%	99.09%	98.72%

Table 2: Analysis done on Tue Dec 9 11:41:38 AM CET 2025, par3113

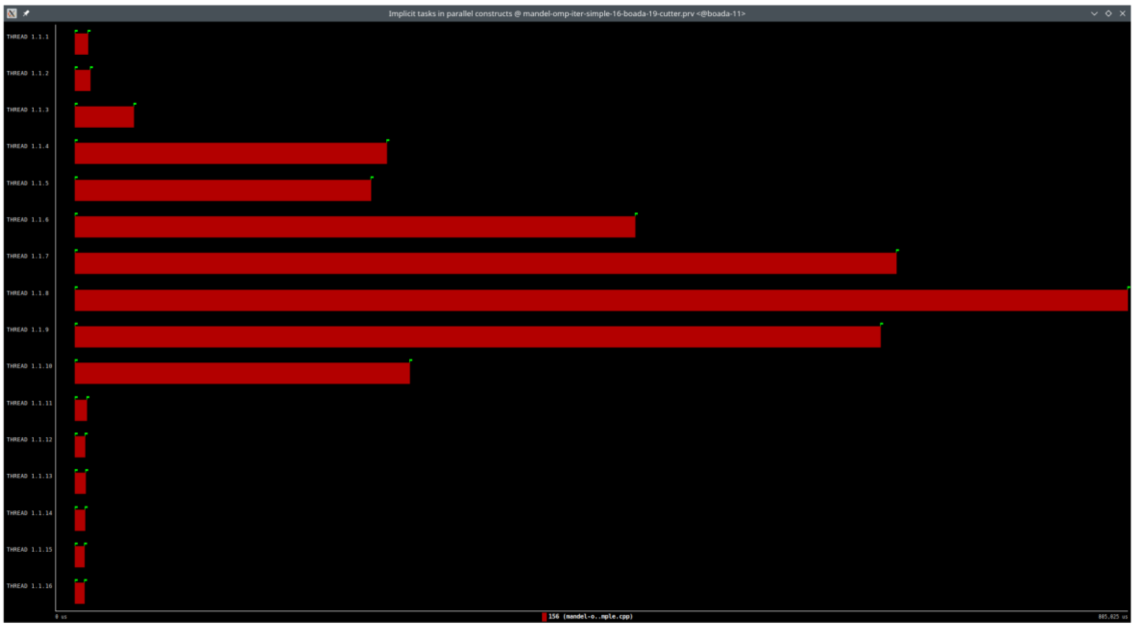
The root cause is identified in the metric “Load balancing in execution efficiency”. This metric plummets to 26.16% at 32 threads. This indicates that work is not evenly distributed (some threads are overloaded while others are idle).

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)	3227729.93	1615095.41	807838.82	404170.63	270277.76	203432.8	163867.76	137452.07	118440.76	103955.46
Load balancing for implicit tasks	1.0	0.69	0.4	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)	116.8	148.62	1749678.5	1386694.2	992858.67	781766.75	642652.76	532385.81	454634.15	393499.1

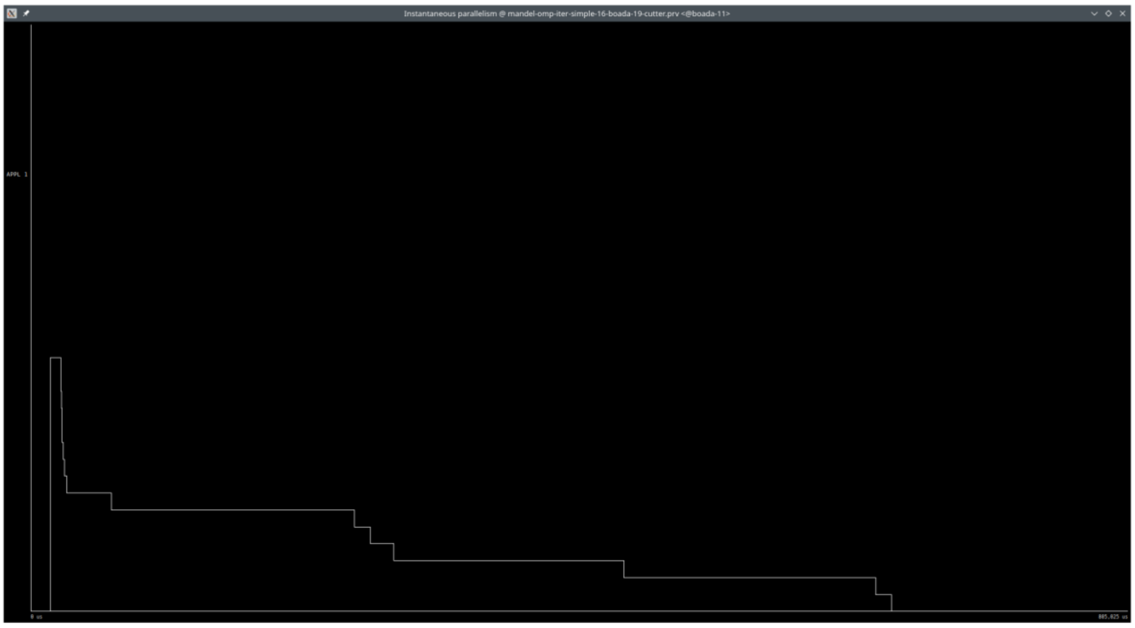
Table 3: Analysis done on Tue Dec 9 11:41:38 AM CET 2025, par3113

In contrast, Cyclic Rows maintains a load balancing efficiency of nearly 99%. This confirms that dividing the image like in this first strategy fails to account for the concentration of computational work whereas other strategies distribute heavy regions across all threads.

The paraver traces provide a visual confirmation of the Load Imbalance derived from the tables.

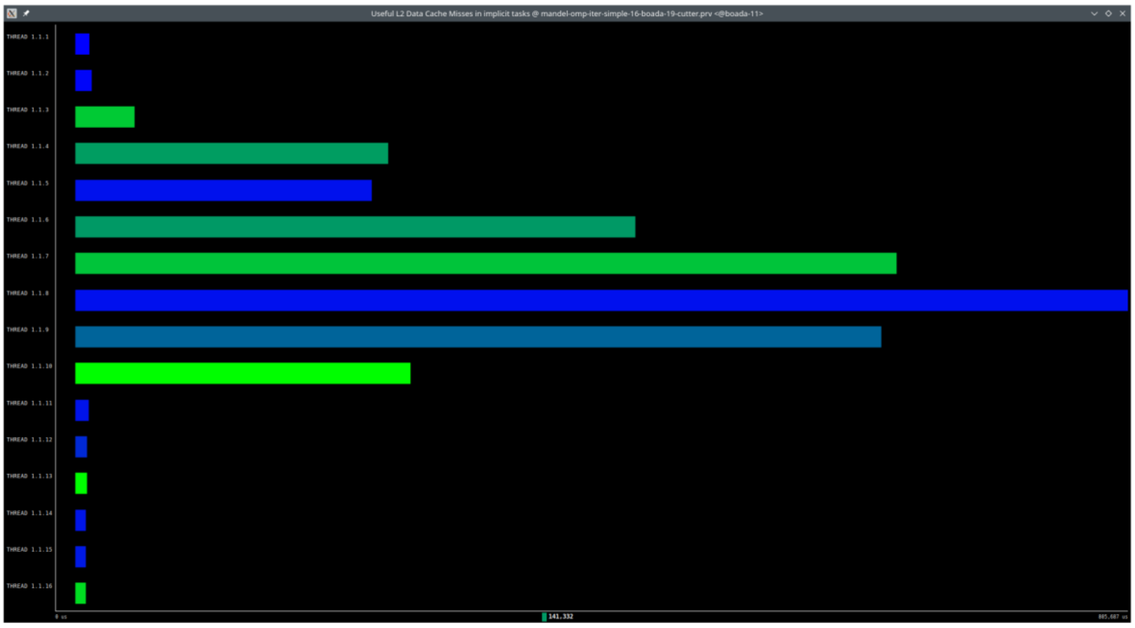


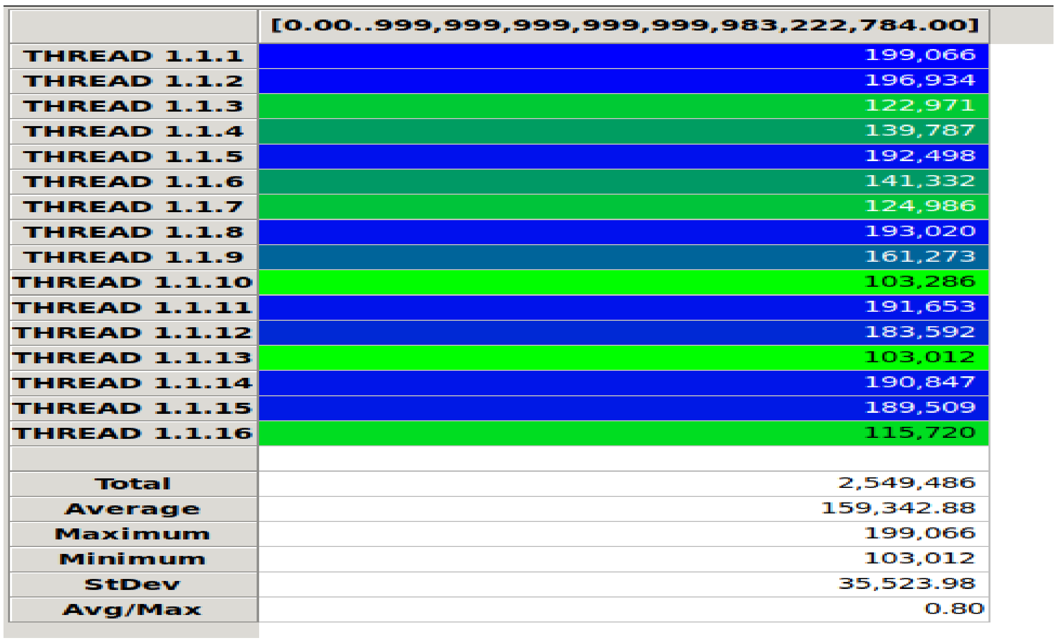
The threads assigned to the edges of the Mandelbrot Set finish almost instantly (this is indicated by short bars) whereas the threads assigned to center columns take a very long time to complete. The traces of Block-Cyclic and Cyclic-Rows show solid blocks in comparison where all threads start and finish approximately at the same time.



The instantaneous parallelism graph for this strategy shows a rapid decline. Starting at maximum number of threads but quickly declining as the threads assigned to the edges finish their work, leaving only a few threads running. This basically destroys speed-up.

The memory analysis analyzes specifically L2 cache misses.



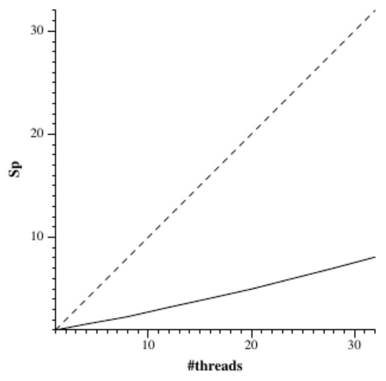


	[0.00..999,999,999,999,999,983,222,784.00]
THREAD 1.1.1	199,066
THREAD 1.1.2	196,934
THREAD 1.1.3	122,971
THREAD 1.1.4	139,787
THREAD 1.1.5	192,498
THREAD 1.1.6	141,332
THREAD 1.1.7	124,986
THREAD 1.1.8	193,020
THREAD 1.1.9	161,273
THREAD 1.1.10	103,286
THREAD 1.1.11	191,653
THREAD 1.1.12	183,592
THREAD 1.1.13	103,012
THREAD 1.1.14	190,847
THREAD 1.1.15	189,509
THREAD 1.1.16	115,720
Total	2,549,486
Average	159,342.88
Maximum	199,066
Minimum	103,012
StDev	35,523.98
Avg/Max	0.80

We can observe high variance in cache misses between threads. Basically, threads that do more computational work tend to have more cache misses simply because they are active longer than the rest and keep writing to the matrix often.

The other two strategies remain stable across all threads. So while L2 cache misses are a problem, memory is not the primary bottleneck as the load imbalance is significantly more impactful.

Finally the scalability graph summarizes the success of the parallelization.

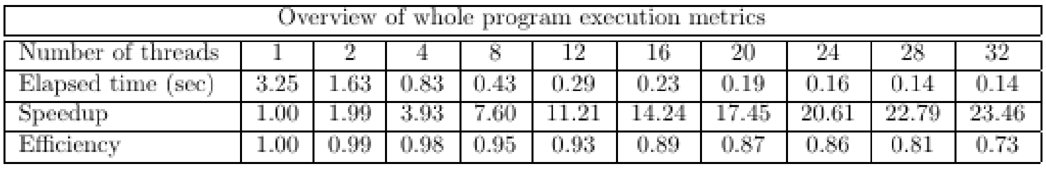


par3113
Speed-up wrt sequential time (mandel funtion only)
Generated by par3113 on Tue Dec 9 11:35:30 AM CET 2025

The curve separates early from the ideal speed-up, being severely flattened when it reaches 32 threads. Both other strategies adapt much better to the ideal speed-up, making this first strategy the worst.

3.Block-Cyclic Geometric Data Decomposition by Columns

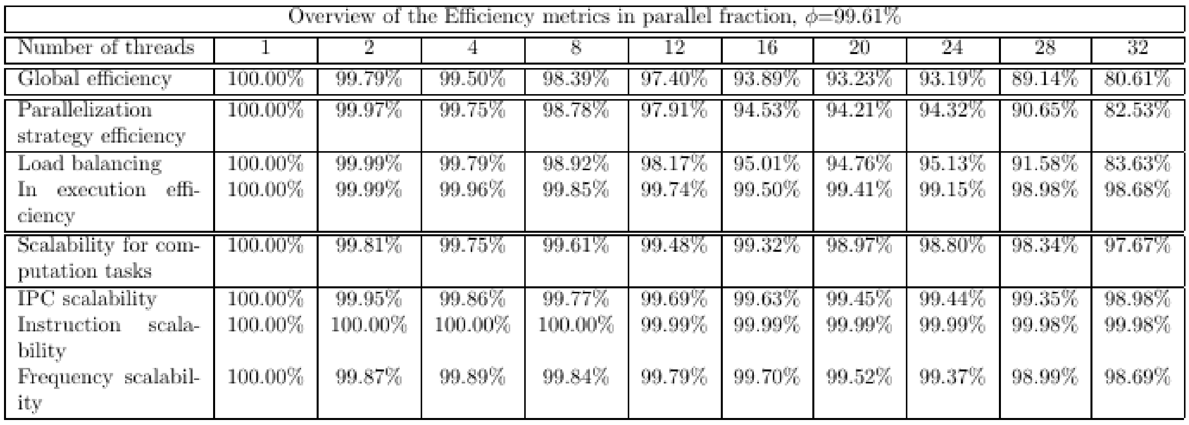
Shifting to a block-cyclic distribution results in incredible improvements in the modelfactor analysis compared to the first strategy.



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	3.93	7.60	11.21	14.24	17.45	20.61	22.79	23.46
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 Tue Dec 9 11:56:06 PM CET 2025, par3107

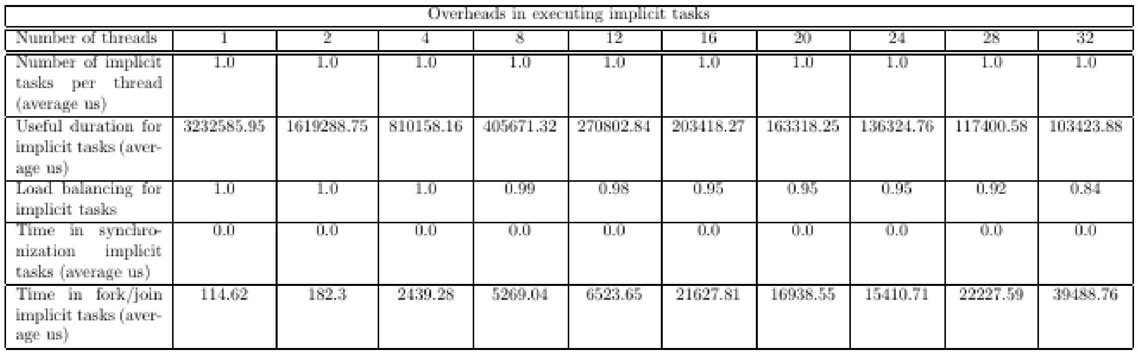
In this second strategy, efficiency gets to 73% at 32 threads clearly indicating a massive improvement over the first strategy. Speed-up reaches 23.46, which is great but still not perfect.
While the first strategy collapsed under load imbalance, the second strategy distributes the center columns among all threads preventing bottlenecks.
Cyclic-Geometric achieves slightly higher efficiency, indicating that the second strategy is good but granularity can still be improved.



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%	99.79%	99.50%	98.39%	97.40%	93.89%	93.23%	93.19%	89.14%	80.61%
Parallelization strategy efficiency	100.00%	99.97%	99.75%	98.78%	97.91%	94.53%	94.21%	94.32%	90.65%	82.53%
Load balancing	100.00%	99.99%	99.79%	98.92%	98.17%	95.01%	94.76%	95.13%	91.58%	83.63%
In execution efficiency	100.00%	99.99%	99.96%	99.85%	99.74%	99.50%	99.41%	99.15%	98.98%	98.68%
Scalability for computation tasks	100.00%	99.81%	99.75%	99.61%	99.48%	99.32%	98.97%	98.80%	98.34%	97.67%
IPC scalability	100.00%	99.95%	99.86%	99.77%	99.69%	99.63%	99.45%	99.44%	99.35%	98.98%
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%	99.87%	99.89%	99.84%	99.79%	99.70%	99.52%	99.37%	98.99%	98.69%

Table 2: Analysis done on Tue Dec 9 11:56:06 PM CET 2025, par3107

The “Load balancing in execution efficiency” confirms the success of this strategy, it stays above 83.63% even at 32 threads. This proves that computational work is now evenly distributed. The drop in Global Efficiency (80%) at 32 threads is likely due to the overhead of managing more numerous and smaller tasks rather than idle threads.

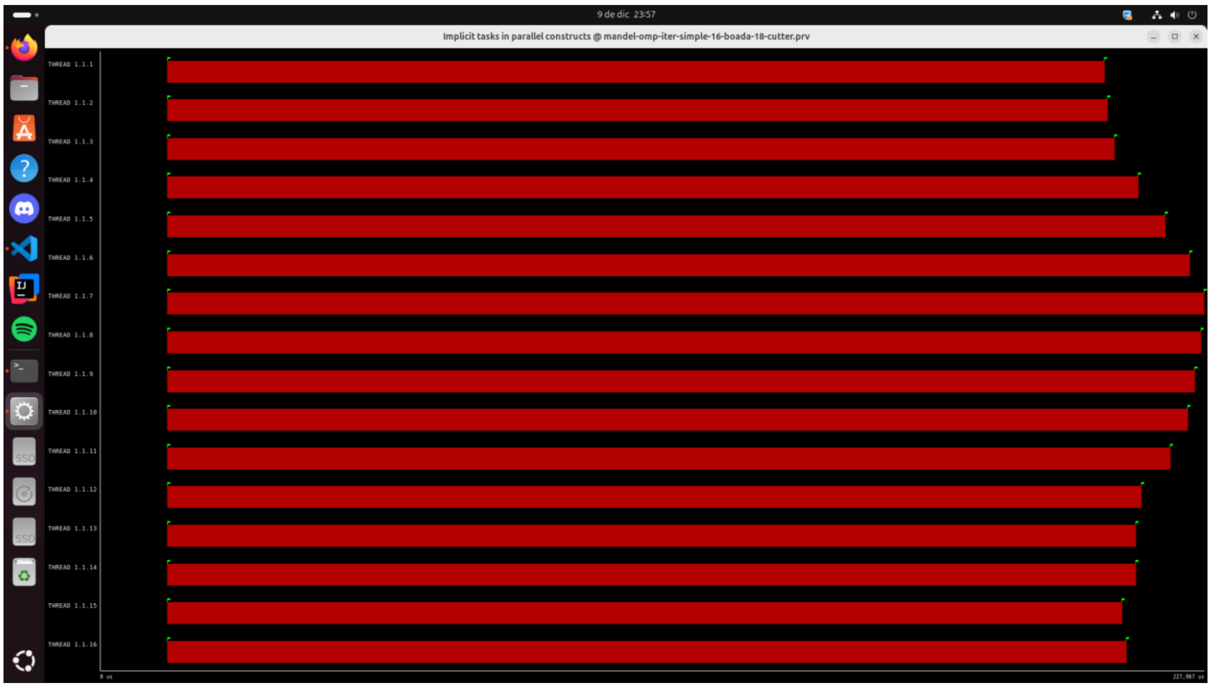


Overheads in executing implicit tasks										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of implicit tasks per thread (average us)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Useful duration for implicit tasks (average us)	3232585.95	1619288.75	810158.16	405671.32	270802.84	203418.27	163318.25	136324.76	117400.58	103423.88
Load balancing for implicit tasks	1.0	1.0	1.0	0.99	0.98	0.95	0.95	0.95	0.92	0.84
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)	114.62	182.3	2439.28	5269.04	6523.65	21627.81	16938.55	15410.71	22227.59	39488.76

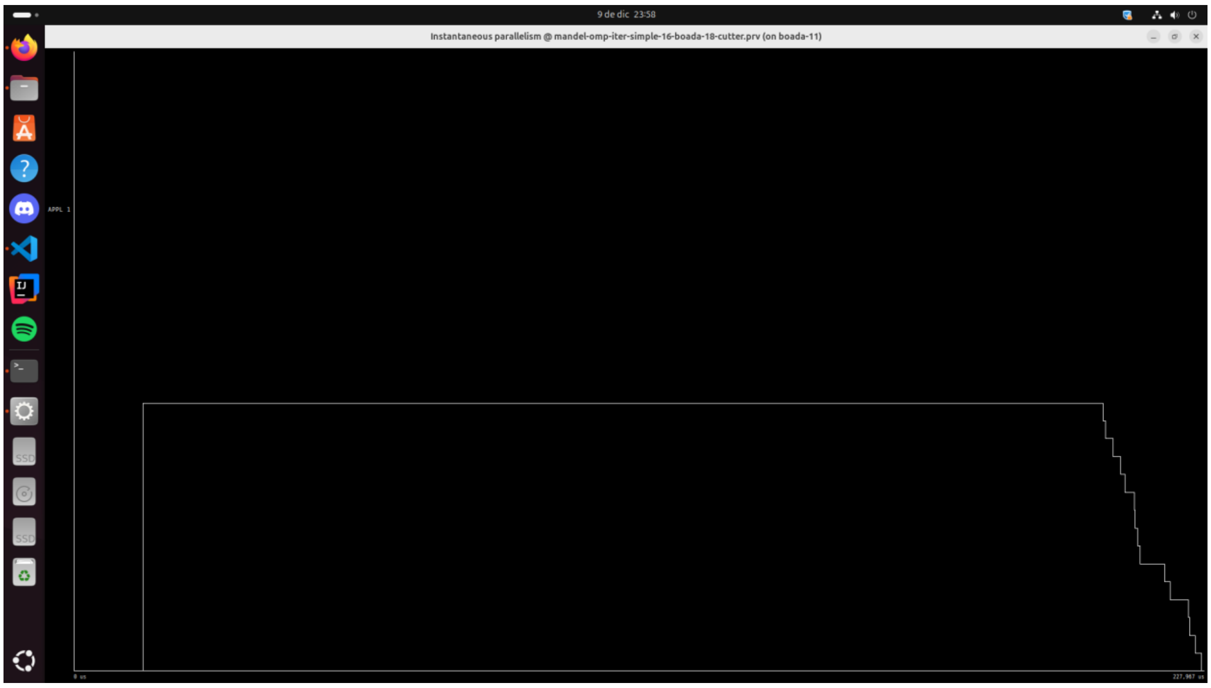
Table 3: Analysis done on Tue Dec 9 11:56:06 PM CET 2025, par3107

Results are also visible in the metric “Time in fork/join” as it is almost ten times lower at 32 threads than the first strategy due to the good amortization over the computation.

The paraver analysis also shows great improvements over the first strategy.

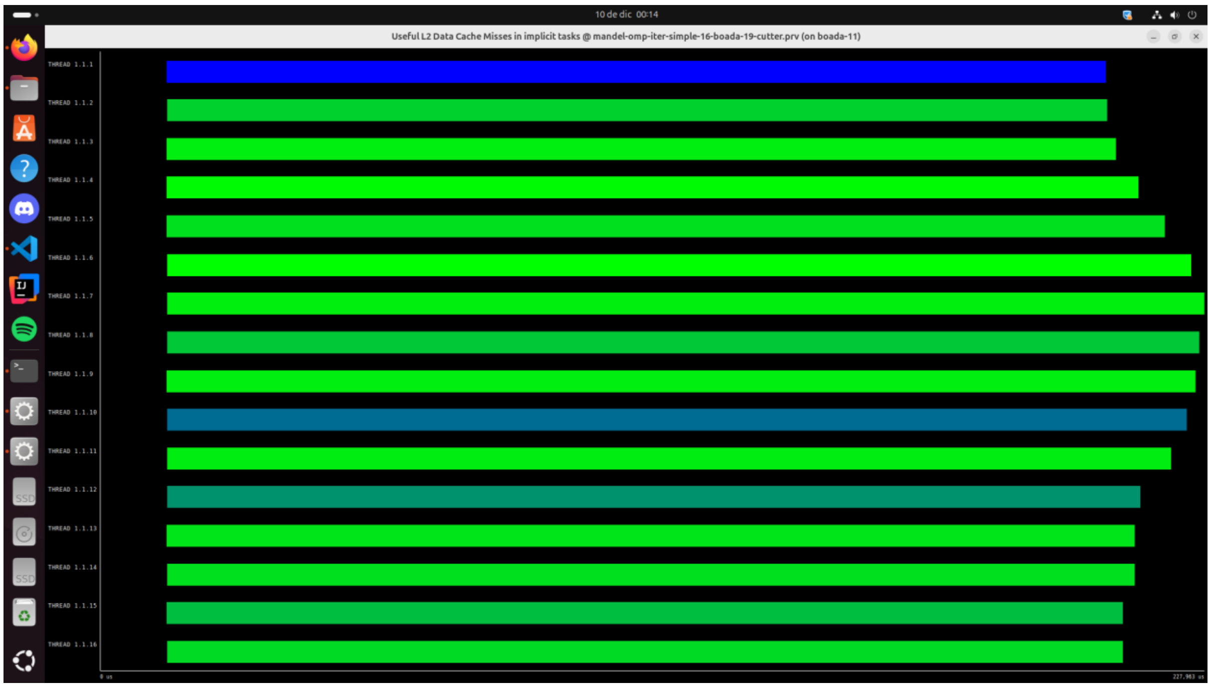


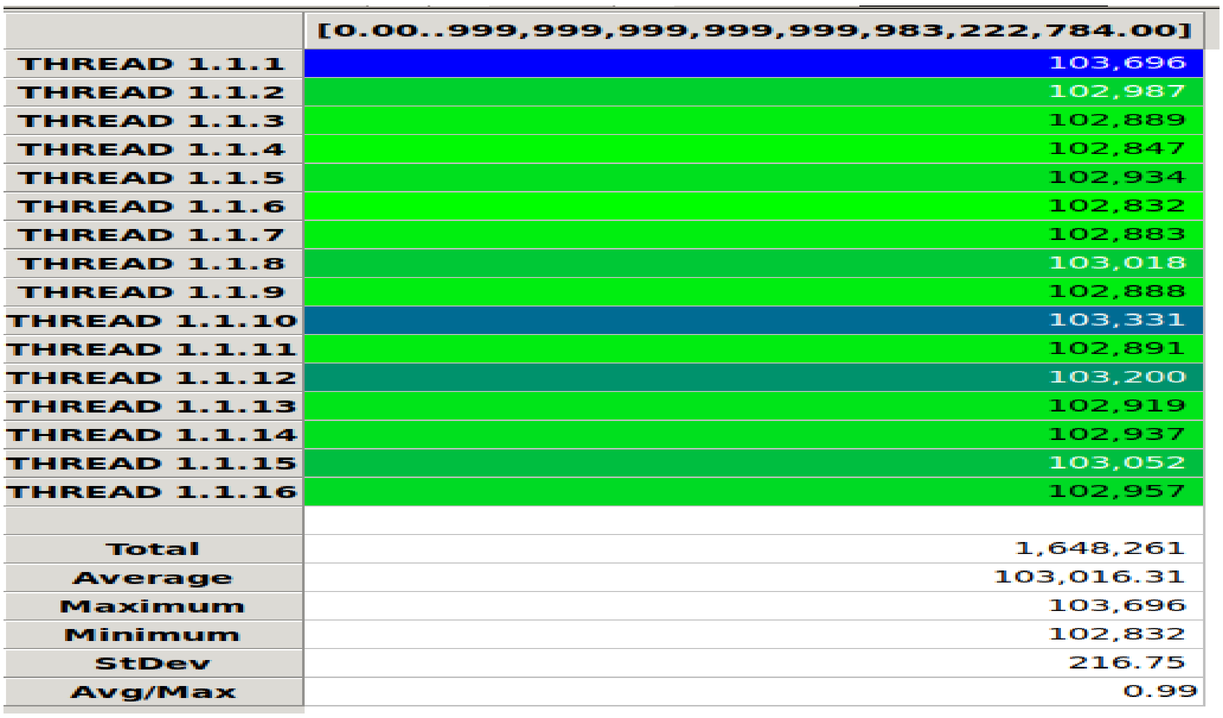
We can see solid red bars for all threads starting and ending almost at the same time. Unlike the first strategy where the threads finished at different times, here the work is balanced. The Block-Cyclic distribution ensures that every thread is assigned the same quantity of work.


9 de dic 23:58
Instantaneous parallelism @ mandel-omp-iter-simple-16-boada-18-cutter.prv (on boada-11)
APPL 1
0 us
227,967 us

The graph almost shows a perfect rectangle. The strategy makes use of all available threads for the entire duration of the parallel region, only dropping at the very end. An outstanding improvement over the first strategy.

While load balancing is solved, we must inspect if the Block-Cyclic strategy hurts memory performance.

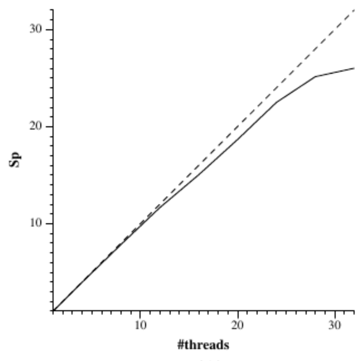




	[0.00..999,999,999,999,999,983,222,784.00]
THREAD 1.1.1	103,696
THREAD 1.1.2	102,987
THREAD 1.1.3	102,889
THREAD 1.1.4	102,847
THREAD 1.1.5	102,934
THREAD 1.1.6	102,832
THREAD 1.1.7	102,883
THREAD 1.1.8	103,018
THREAD 1.1.9	102,888
THREAD 1.1.10	103,331
THREAD 1.1.11	102,891
THREAD 1.1.12	103,200
THREAD 1.1.13	102,919
THREAD 1.1.14	102,937
THREAD 1.1.15	103,052
THREAD 1.1.16	102,957
Total	1,648,261
Average	103,016.31
Maximum	103,696
Minimum	102,832
StDev	216.75
Avg/Max	0.99

The misses are extremely uniform across all threads. This confirms load balancing as everyone is doing the same amount of work.

It is very stable and the block size of 16 pixels (int block = 16) that we deduced through available mechanics at the lab and implemented in the code, appears to be good enough for the memory.



par3107

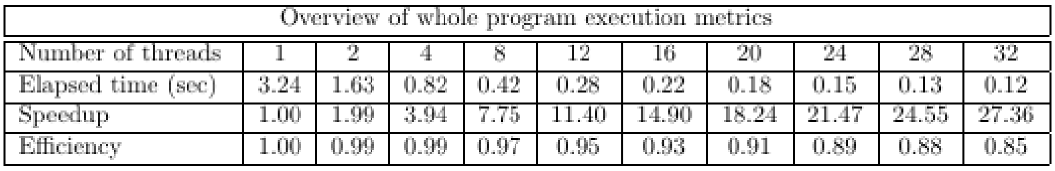
Speed-up wrt sequential time (mandel funtion only)

Generated by par3107 on Wed Dec 10 12:23:33 AM CET 2025

The scalability graph shows an excellent curve, remaining almost close to the ideal speed-up. It reaches its maximum at 23. Vastly superior compared to the first strategy while it is very close to the Cyclic Rows that we will analyze on the next page.

4.Cyclic Geometric Data Decomposition by Rows

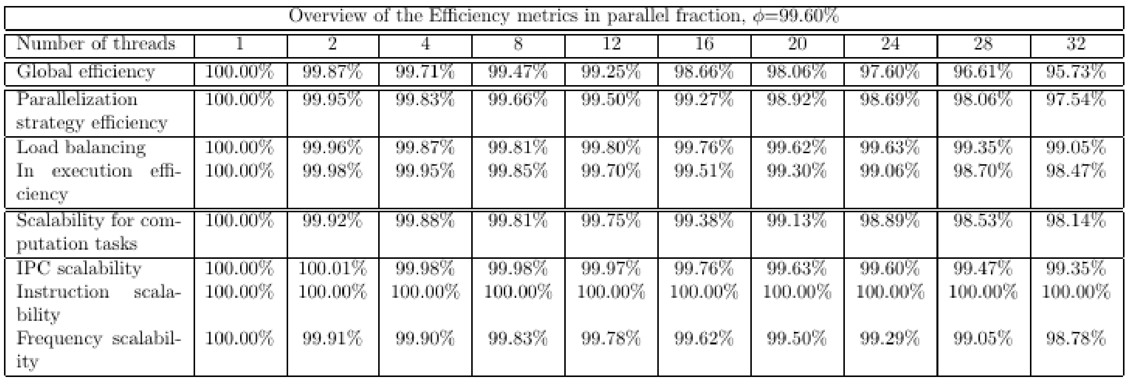
Modelfactor analysis for this last strategy reveals the highest efficiency and performance metrics among all tested strategies.



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	11.40	14.90	18.24	21.47	24.55	27.36
Efficiency	1.00	0.99	0.99	0.97	0.95	0.93	0.91	0.89	0.88	0.85

Table 1: Analysis done on Tue Dec 16 10:32:39 AM CET 2025, par3107

At 32 threads, Cyclic Geometric achieves a global efficiency of 85% and a speed-up of 27.36. This is the closest to the ideal speed-up of 32. Compared to the first strategy, this one is almost three times more efficient, while the difference with the second strategy is only 4 speed-up points and a little more efficiency, still providing a noticeable boost.



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%	99.87%	99.71%	99.47%	99.25%	98.66%	98.06%	97.60%	96.61%	95.73%
Parallelization strategy efficiency	100.00%	99.95%	99.83%	99.66%	99.50%	99.27%	98.92%	98.69%	98.06%	97.54%
Load balancing	100.00%	99.96%	99.87%	99.81%	99.80%	99.76%	99.62%	99.63%	99.35%	99.05%
In execution efficiency	100.00%	99.98%	99.95%	99.85%	99.70%	99.51%	99.30%	99.06%	98.70%	98.47%
Scalability for computation tasks	100.00%	99.92%	99.88%	99.81%	99.75%	99.38%	99.13%	98.89%	98.53%	98.14%
IPC scalability	100.00%	100.01%	99.98%	99.98%	99.97%	99.76%	99.63%	99.60%	99.47%	99.35%
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%	99.91%	99.90%	99.83%	99.78%	99.62%	99.50%	99.29%	99.05%	98.78%

Table 2: Analysis done on Tue Dec 16 10:32:39 AM CET 2025, par3107

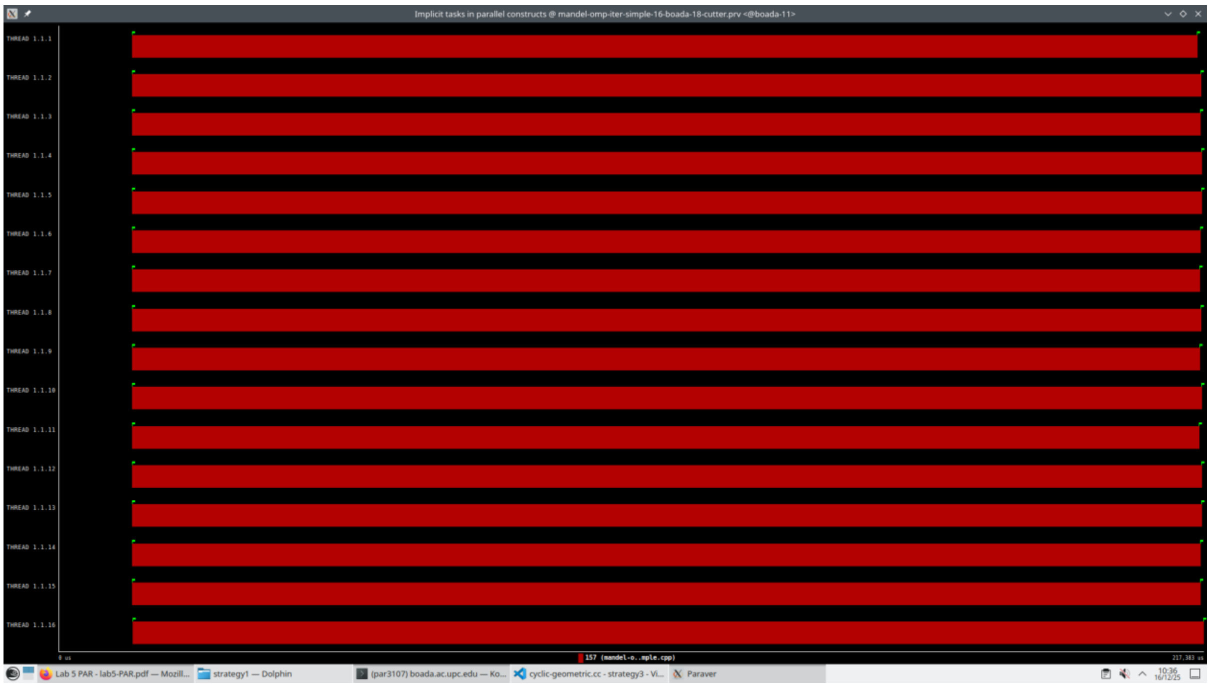
The “Load balancing in execution efficiency” is outstanding, remaining above 98%. By slicing the Mandelbrot Set by rows and distributing them cyclically, every thread processes a fair share of work.

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)	3225985.79	1614249.63	807459.94	404013.35	269508.34	202878.66	162711.29	135921.37	116936.37	102724.32
Load balancing for implicit tasks	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	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)	109.99	1479.77	2199.6	1877.62	1757.9	2131.24	2630.3	2798.53	3584.63	4251.45

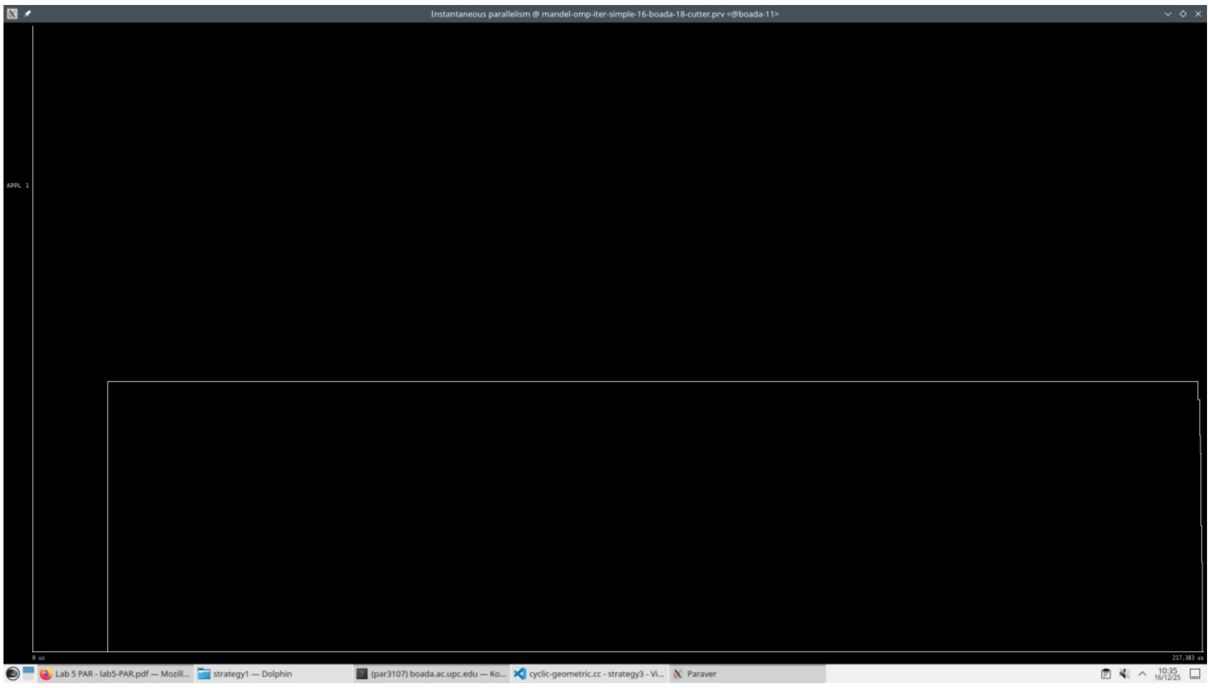
Table 3: Analysis done on Tue Dec 16 10:32:39 AM CET 2025, par3107

The "Time in fork/join" metric also shows improvement as it is ten times faster than the second strategy and a hundred times faster than the first one.

The traces from the paraver analysis are nearly identical to those of the second strategy, but slightly improved (like an "optimal solution).

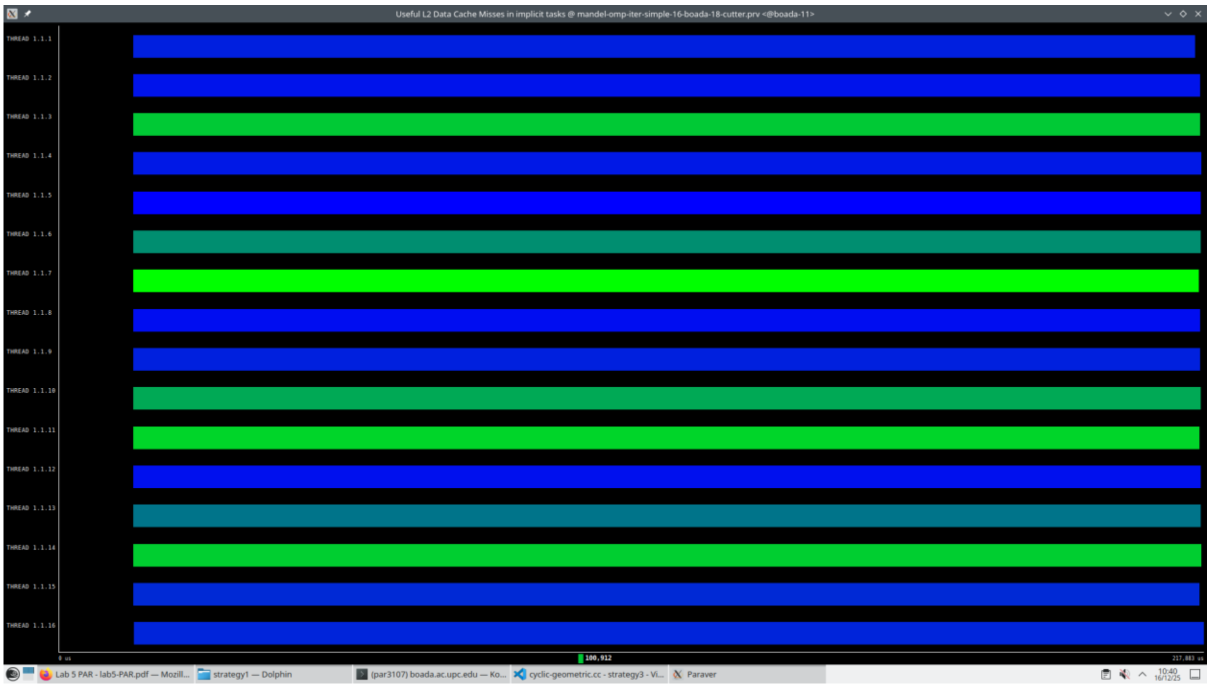


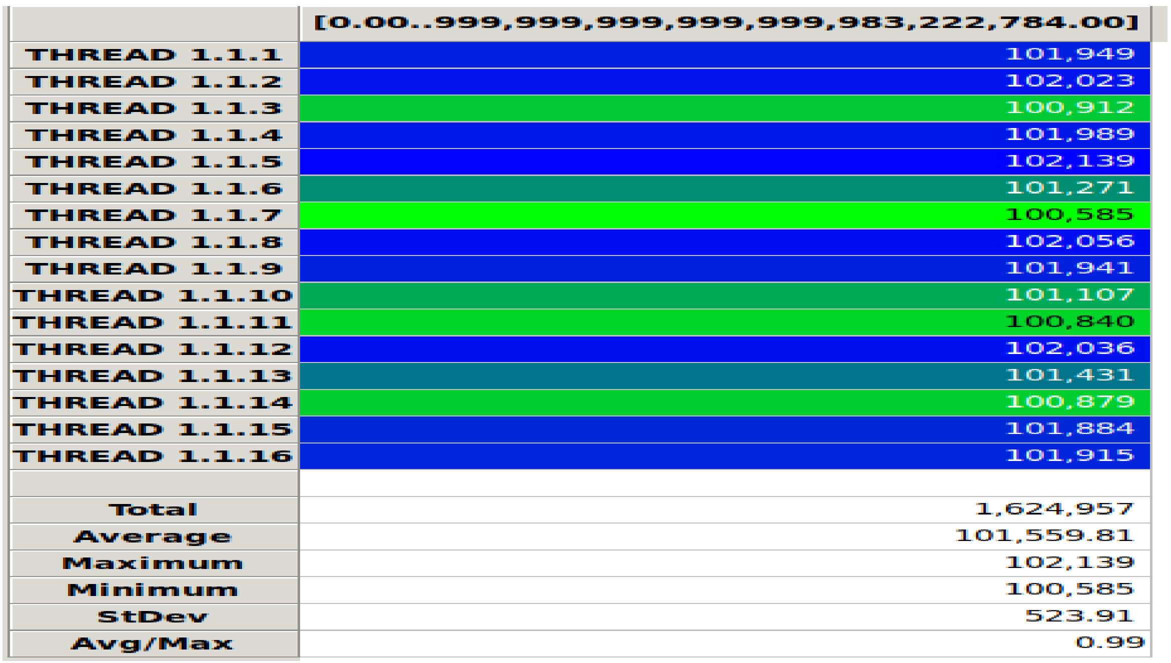
We can observe a solid block where all threads start and finish at the same time. There are no gaps or rare effects.


Instantaneous parallelism @ mandel-omp-iter-simple-16-boada-18-cutter.prv <@boada-11>
APPL 1
0 us
217.383 us

The instantaneous parallelism graph also shows a perfect rectangle, making use of all the available threads throughout the entire computational phase of the program.

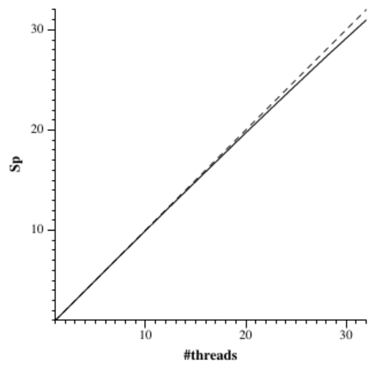
Even though the memory analysis remains very similar to the one of the second strategy, we can deduce why this strategy performs better in this aspect.





	[0.00..999,999,999,999,999,983,222,784.00]
THREAD 1.1.1	101,949
THREAD 1.1.2	102,023
THREAD 1.1.3	100,912
THREAD 1.1.4	101,989
THREAD 1.1.5	102,139
THREAD 1.1.6	101,271
THREAD 1.1.7	100,585
THREAD 1.1.8	102,056
THREAD 1.1.9	101,941
THREAD 1.1.10	101,107
THREAD 1.1.11	100,840
THREAD 1.1.12	102,036
THREAD 1.1.13	101,431
THREAD 1.1.14	100,879
THREAD 1.1.15	101,884
THREAD 1.1.16	101,915
Total	1,624,957
Average	101,559.81
Maximum	102,139
Minimum	100,585
StDev	523.91
Avg/Max	0.99

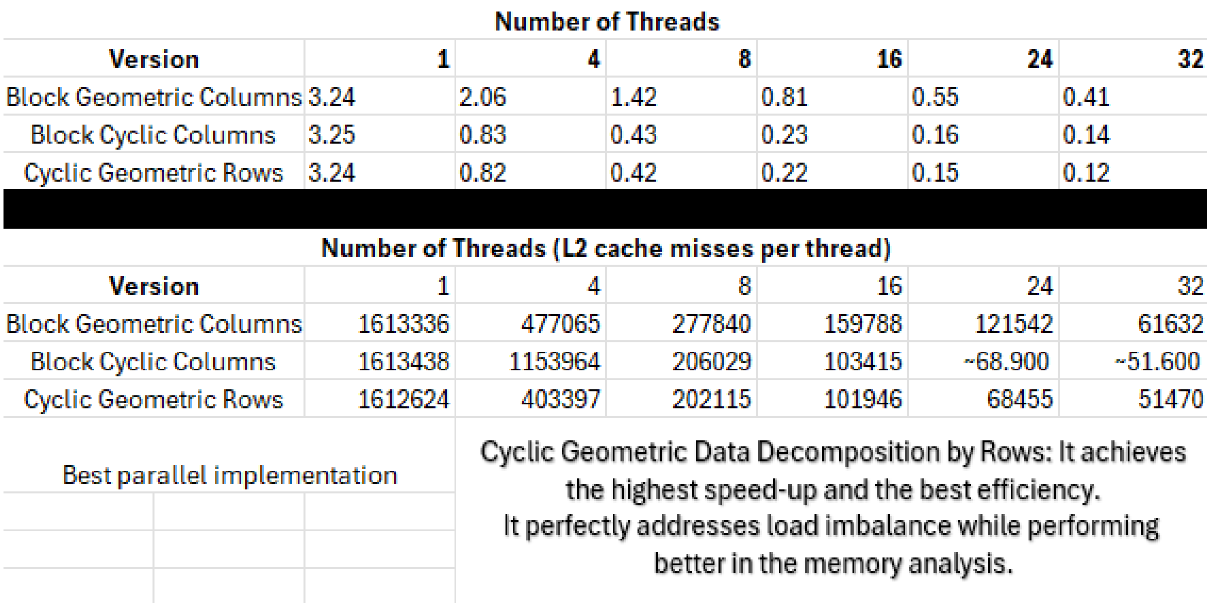
While the second strategy processed a small block of columns, this strategy processes full rows. This can be the main reason that helps this last strategy to get 12% extra efficiency over the second strategy.



par3107
Speed-up wrt sequential time (mandel funtion only)
Generated by par3107 on Tue Dec 16 10:42:08 AM CET 2025

The scalability graph shows an even better curve than the second strategy, extremely close to the ideal speed-up even at 32 threads. This makes this last strategy the best one.

5.Final Comparison



	Number of Threads					
Version	**1**	**4**	**8**	**16**	**24**	**32**
Block Geometric Columns	3.24	2.06	1.42	0.81	0.55	0.41
Block Cyclic Columns	3.25	0.83	0.43	0.23	0.16	0.14
Cyclic Geometric Rows	3.24	0.82	0.42	0.22	0.15	0.12

	Number of Threads (L2 cache misses per thread)					
Version	1	4	8	16	24	32
Block Geometric Columns	1613336	477065	277840	159788	121542	61632
Block Cyclic Columns	1613438	1153964	206029	103415	~68.900	~51.600
Cyclic Geometric Rows	1612624	403397	202115	101946	68455	51470

Best parallel implementation

Cyclic Geometric Data Decomposition by Rows: It achieves the highest speed-up and the best efficiency.
It perfectly addresses load imbalance while performing better in the memory analysis.