PAR LABORATORY 4 REPORT

Q1 2025/2026

Arnau Hernández Navarro and Felipe Arturo Núñez Gómez

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

FIB

INDEX

1.Brief Introduction

In this fourth laboratory, we continue learning and adapting the codes of the different strategies developed in the previous laboratory session. This session will be divided in two halves where we will maximize parallelism following our analysis in the previous session:

The Iterative Task Decomposition part:

- Tile Strategy: Corresponds to the original tile strategy seen in session 3.
- Fine Grain Strategy: Corresponds to the iterative fine grain parallel strategy seen in session 3.

The Recursive Task Decomposition part:

- Leaf Strategy: Corresponds to the leaf strategy seen in session 3. It includes an improved version with a speed-up larger than 2 for more than 2 threads.
- Tree Strategy: Corresponds to the recursive tree task decomposition parallel strategy seen in session 3.

2.Iterative Task Decomposition

2.1. Tile Strategy

The sequential code was parallelized using OpenMP tasking with the following modifications:

- A #pragma omp parallel region creates the thread team and a #pragma omp single directive ensures only the master thread generates tasks.
- A #pragma omp task directive encapsulates the processing of each block, allowing dynamic assignment to threads.
- To ensure safety when accessing shared resources, two constructs were added:
    - #pragma omp atomic when updating the global histogram array to prevent race conditions during write operations.
    - #pragma omp critical used to protect the XDrawPoint function, ensuring that only one thread accesses the X11 display at a time.

Modelfactor Analysis

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	3.48	1.96	1.03	0.74	0.73	0.73	0.73	0.73	0.73	0.73
Speedup	1.00	1.78	3.40	4.68	4.78	4.77	4.78	4.78	4.78	4.78
Efficiency	1.00	0.89	0.85	0.59	0.40	0.30	0.24	0.20	0.17	0.15

Table 1: Analysis done on Tue Nov 11 11:23:37 AM CET 2025, par3113

Overview of the Efficiency metrics in parallel fraction, ϕ=99.99%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	100.00%	88.78%	84.95%	58.51%	39.87%	29.80%	23.89%	19.94%	17.09%	14.94%
Parallelization strategy efficiency	100.00%	88.88%	85.05%	58.75%	40.13%	30.16%	24.19%	20.23%	17.43%	15.21%
Load balancing	100.00%	88.90%	85.10%	58.82%	40.16%	30.22%	24.24%	20.27%	17.45%	15.25%
In execution efficiency	100.00%	99.97%	99.95%	99.87%	99.92%	99.80%	99.79%	99.81%	99.84%	99.75%
Scalability for computation tasks	100.00%	99.90%	99.88%	99.60%	99.35%	98.82%	98.75%	98.57%	98.07%	98.20%
IPC scalability	100.00%	100.00%	99.98%	99.97%	99.97%	99.96%	99.96%	99.96%	99.96%	99.97%
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%	99.90%	99.89%	99.63%	99.38%	98.86%	98.79%	98.60%	98.11%	98.23%

Table 2: Analysis done on Tue Nov 18 10:49:29 AM CET 2025, par3113

Statistics about explicit tasks in parallel fraction										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of explicit tasks executed (total)	64.0	64.0	64.0	64.0	64.0	64.0	64.0	64.0	64.0	64.0
LB (number of explicit tasks executed)	1.0	0.94	0.64	0.44	0.31	0.24	0.19	0.16	0.13	0.12
LB (time executing explicit tasks)	1.0	0.89	0.85	0.59	0.4	0.3	0.24	0.2	0.17	0.15
Time per explicit task (average us)	54420.42	54471.11	54475.69	54614.25	54712.63	54951.27	54934.87	54925.71	55033.94	54816.65
Overhead per explicit task (synch %)	0.0	12.49	17.52	70.04	149.18	231.41	313.59	395.44	476.9	561.9
Overhead per explicit task (sched %)	0.0	0.01	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Number of taskwait/taskgroup (total)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

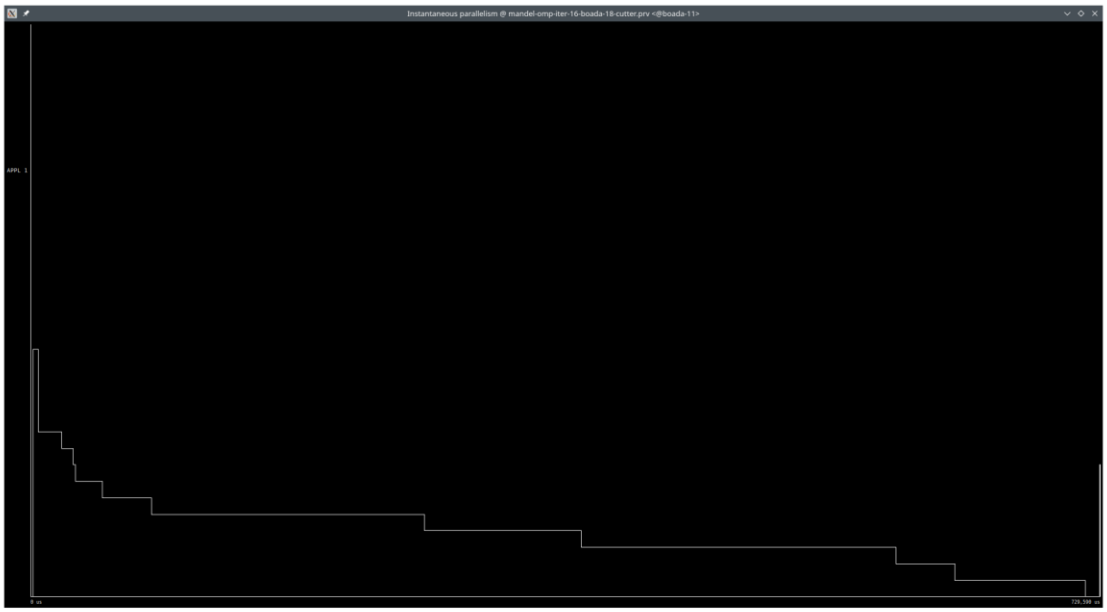
Table 3: Analysis done on Tue Nov 18 10:49:29 AM CET 2025, par3113

The primary bottleneck detected is severe Load Imbalance.

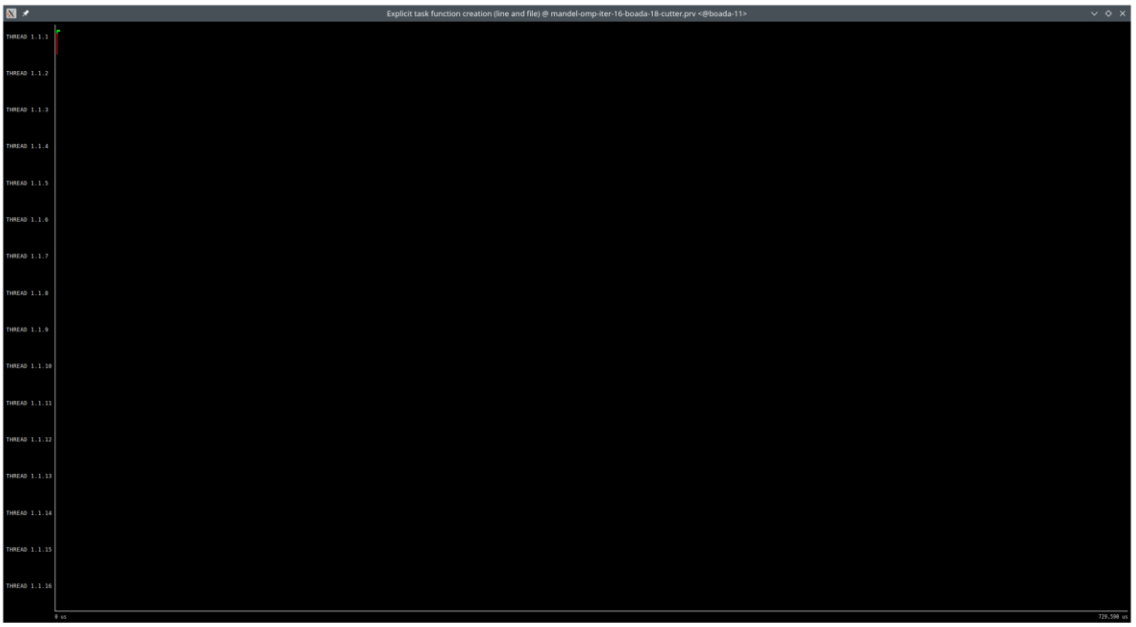
- The efficiency metrics row shows that Load Balancing efficiency collapses as the thread count rises dropping from 89% to 15%. This can occur because the tiles inside the Mandelbrot set require significantly more computation time. Consequently, threads assigned to complex blocks work for a long time, while other threads finish their light tasks quickly and remain idle (indicated by low LB time executing explicit tasks).

Paraver analysis

The previous image corresponds to explicit task function execution. The timeline visually demonstrates the load imbalance. Some specific threads show very long red bars (complex tiles), while others remain black (idle) most of the time.

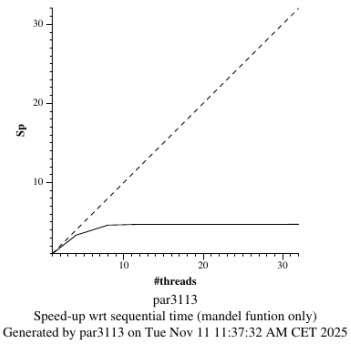


The instantaneous parallelism graph exhibits a staircase effect. Parallelism starts high but decreases rapidly as more complex tasks are added, leaving only a small set of threads working with expensive tasks.



The explicit task function creation shows that the creation is instantaneous and performed entirely by the master thread.

Scalability



The results show an early saturation of Speedup, stopping effectively after 4 threads with a value of 4.78x approximately. The limitation must be caused by the granularity of the decomposition. The total execution time is derived from the most complex tile. Adding more threads cannot add more speed up and the overheads of atomic or critical are negligible compared to the massive loss of efficiency caused by the uneven distribution of work.

2.2. Fine Grain Strategy

The sequential code was parallelized using OpenMP tasking with the following modifications:

- A #pragma omp parallel combined with #pragma omp single directive ensures that only one thread generates task
- A task for each inner loop divided with a task for the vertical borders, horizontal borders and if-else block.
- EqualH and EqualV temporal variables used with dependency directive to enforce correct order in order to assign the value for equal variable.
- Atomic and critical directives to protect updates to shared data and ensure that only one thread accesses the X11 display at a time.

Modelfactor Analysis

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	3.49	1.76	0.88	0.45	0.31	0.24	0.19	0.17	0.15	0.13
Speedup	1.00	1.99	3.95	7.79	11.42	14.83	18.13	21.14	23.62	26.21
Efficiency	1.00	0.99	0.99	0.97	0.95	0.93	0.91	0.88	0.84	0.82

Table 1: Analysis done on Tue Nov 25 11:11:17 AM CET 2025, par3113

Overview of the Efficiency metrics in parallel fraction, ϕ=99.99%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	99.94%	99.30%	98.76%	97.39%	95.26%	92.78%	90.72%	88.20%	84.48%	82.07%
Parallelization strategy efficiency	99.94%	99.43%	98.91%	97.69%	95.83%	93.37%	91.68%	89.26%	85.59%	83.40%
Load balancing	100.00%	99.97%	99.78%	99.20%	99.09%	96.79%	96.09%	93.46%	88.99%	87.65%
In execution efficiency	99.94%	99.46%	99.12%	98.49%	96.71%	96.47%	95.41%	95.51%	96.18%	95.15%
Scalability for computation tasks	100.00%	99.87%	99.85%	99.69%	99.41%	99.37%	98.95%	98.81%	98.71%	98.41%
IPC scalability	100.00%	99.88%	99.86%	99.84%	99.82%	99.79%	99.79%	99.79%	99.80%	99.80%
Instruction scalability	100.00%	100.09%	100.09%	100.09%	100.09%	100.09%	100.09%	100.09%	100.09%	100.09%
Frequency scalability	100.00%	99.89%	99.89%	99.75%	99.49%	99.49%	99.07%	98.93%	98.82%	98.52%

Table 2: Analysis done on Tue Nov 25 11:11:17 AM CET 2025, par3113

Statistics about explicit tasks in parallel fraction										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of explicit tasks executed (total)	8448.0	8448.0	8448.0	8448.0	8448.0	8448.0	8448.0	8448.0	8448.0	8448.0
LB (number of explicit tasks executed)	1.0	0.95	0.64	0.43	0.32	0.49	0.38	0.41	0.46	0.37
LB (time executing explicit tasks)	1.0	1.0	1.0	0.99	0.99	0.98	0.97	0.95	0.93	0.93
Time per explicit task (average us)	412.62	414.31	415.12	417.82	421.58	425.91	429.82	436.45	445.64	451.97
Overhead per explicit task (synch %)	0.0	0.38	0.65	1.37	2.53	3.74	5.1	6.05	8.01	8.52
Overhead per explicit task (sched %)	0.06	0.18	0.38	0.88	1.56	2.67	3.22	4.8	6.86	8.17
Number of taskwait/taskgroup (total)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 3: Analysis done on Tue Nov 25 11:11:17 AM CET 2025, par3113

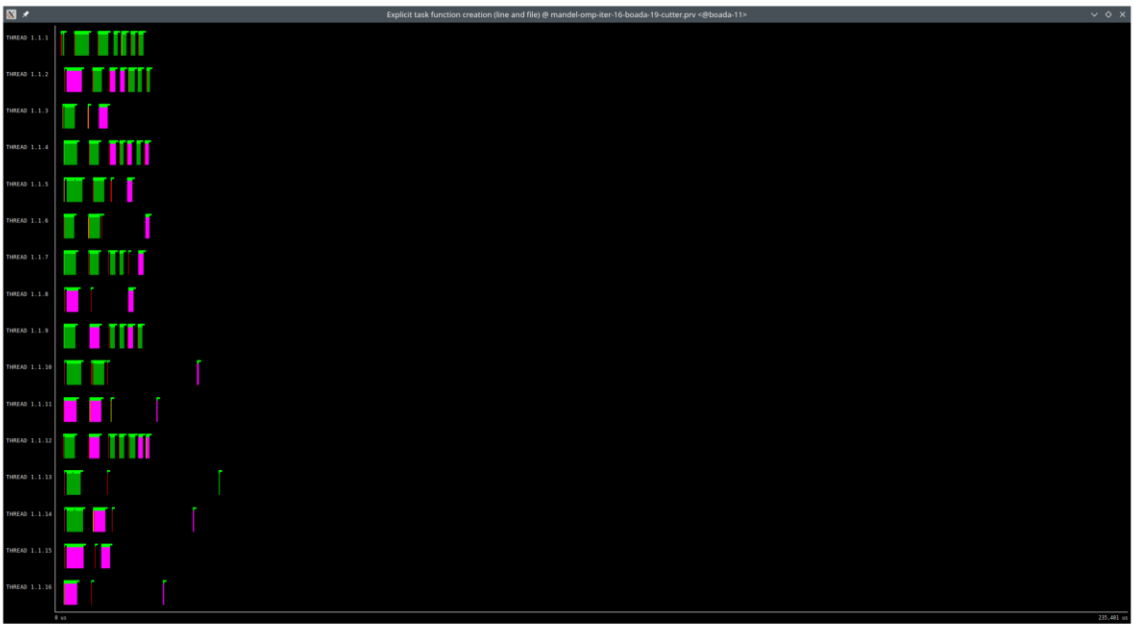
This strategy clearly improves over the tile version in the following terms:

- Efficiency remains stable at 82%, whereas it collapses to 15% in the Tile version and 13% in the improved leaf due to imbalances.
- Unlike the Improved Leaf strategy, which suffers from poor distribution 18.25% LB, this strategy maintains excellent balancing 87% LB, ensuring work is evenly divided among all threads.

Paraver analysis

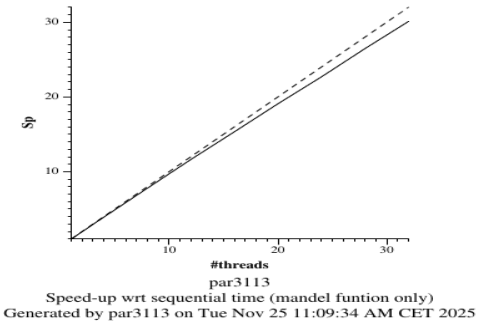
The explicit task function execution demonstrates that the load balance is really good with all the threads doing almost the same amount of work, unlike the tile version where it was the opposite.

The instantaneous parallelism graph shows an early high parallelism achieved, as the tile version, however, remains stable during all the execution. Only at the end does parallelism drop.



The explicit task function creation shows a longer task-creation phase in which all the threads participate unlike the tile version.

Scalability



The scalability exposes how much the speedup improves from the tile version.
The fine grain strategy more than 5 times the speedup of the tile one (4.78 to 26.21)
The reason is that this strategy generates many small tasks, ensuring that all the threads have work to do maintaining excellent load balance which results in a great scalability.

3.Recursive Task Decomposition

3.1. Leaf Strategy

The sequential code was parallelized using OpenMP tasking with the following modifications:

- Unimproved version: The image is recursively splitted into quadrants. When the block size reaches NCols <= Tile, a #pragma omp task is generated to compute the bloc. This results in tasks being created for every single tile in the grid, leading to a massive number of fine-grained tasks.
- Improved version: A cutoff mechanism was implemented to control granularity tasks. The cutoff macro was set to 3, giving us the best results. Also added a depth parameter to the function.
    - If depth <= CUTOFF, tasks are created with #pragma omp task.
    - Once depth exceeds the CUTOFF, the function executes the remaining recursion or computation serially without generating new tasks.
    - #pragma omp atomic is used for histogram updates and #pragma omp critical protects X11 calls.

Modelfactor Analysis Unimproved Version

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	1.83	1.42	1.42	1.42	1.42	1.42	1.42	1.42	1.43	1.43
Speedup	1.00	1.29	1.29	1.29	1.29	1.29	1.29	1.29	1.29	1.29
Efficiency	1.00	0.64	0.32	0.16	0.11	0.08	0.06	0.05	0.05	0.04

Table 1: Analysis done on Tue Nov 25 11:40:16 AM CET 2025, par3113

Overview of the Efficiency metrics in parallel fraction, ϕ=99.98%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	99.96%	64.44%	32.22%	16.10%	10.72%	8.04%	6.43%	5.36%	4.59%	4.02%
Parallelization strategy efficiency	99.96%	64.56%	32.30%	16.16%	10.78%	8.10%	6.50%	5.43%	4.67%	4.12%
Load balancing	100.00%	64.76%	32.39%	16.22%	10.82%	8.13%	6.52%	5.45%	4.69%	4.14%
In execution efficiency	99.96%	99.69%	99.72%	99.66%	99.64%	99.61%	99.64%	99.62%	99.61%	99.55%
Scalability for computation tasks	100.00%	99.82%	99.74%	99.60%	99.46%	99.28%	98.94%	98.74%	98.23%	97.53%
IPC scalability	100.00%	99.84%	99.78%	99.77%	99.72%	99.72%	99.66%	99.75%	99.71%	99.71%
Instruction scalability	100.00%	100.07%	100.07%	100.06%	100.06%	100.06%	100.06%	100.06%	100.06%	100.06%
Frequency scalability	100.00%	99.91%	99.89%	99.77%	99.67%	99.50%	99.22%	98.92%	98.46%	97.76%

Table 2: Analysis done on Tue Nov 25 11:40:16 AM CET 2025, par3113

Statistics about explicit tasks in parallel fraction										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of explicit tasks executed (total)	3031.0	3031.0	3031.0	3031.0	3031.0	3031.0	3031.0	3031.0	3031.0	3031.0
LB (number of explicit tasks executed)	1.0	0.54	0.57	0.87	0.84	0.91	0.79	0.8	0.79	0.74
LB (time executing explicit tasks)	1.0	0.5	0.75	0.77	0.63	0.6	0.58	0.58	0.5	0.53
Time per explicit task (average us)	137.35	137.98	138.11	138.19	138.27	138.23	138.24	138.16	138.3	138.36
Overhead per explicit task (synch %)	0.0	239.97	918.25	2275.08	3632.79	4990.88	6351.58	7705.39	9062.28	10409.86
Overhead per explicit task (sched %)	0.18	0.71	0.7	0.83	0.85	0.86	0.81	0.81	0.84	0.83
Number of taskwait/taskgroup (total)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 3: Analysis done on Tue Nov 25 11:40:16 AM CET 2025, par3113

The primary issue with the unimproved version is excessive overhead.

- The tables set above show that Efficiency drops catastrophically to 4% with 32 threads. The cause is visible in the Statistics About Explicit Tasks table where we can see that the overhead per explicit task reaches an absurd 10409.86% at 32 threads. The runtime spends more time managing the synchronization of thousands of tiny tasks than performing useful computation.

Modelfactor Analysis Improved Version

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	1.82	1.05	0.62	0.45	0.45	0.45	0.45	0.45	0.45	0.45
Speedup	1.00	1.74	2.96	4.06	4.06	4.05	4.06	4.05	4.06	4.05
Efficiency	1.00	0.87	0.74	0.51	0.34	0.25	0.20	0.17	0.14	0.13

Table 1: Analysis done on Tue Nov 25 10:16:19 AM CET 2025, par3113

Overview of the Efficiency metrics in parallel fraction, ϕ=99.97%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	99.98%	86.97%	73.90%	50.80%	33.84%	25.35%	20.28%	16.89%	14.49%	12.67%
Parallelization strategy efficiency	99.98%	87.08%	74.08%	50.94%	33.98%	25.51%	20.47%	17.13%	14.73%	12.97%
Load balancing	100.00%	91.29%	81.90%	71.41%	47.85%	35.95%	28.82%	24.11%	20.72%	18.25%
In execution efficiency	99.98%	95.39%	90.45%	71.34%	71.01%	70.94%	71.02%	71.05%	71.11%	71.10%
Scalability for computation tasks	100.00%	99.87%	99.75%	99.72%	99.59%	99.38%	99.07%	98.62%	98.38%	97.65%
IPC scalability	100.00%	99.96%	99.93%	99.87%	99.85%	99.84%	99.83%	99.81%	99.83%	99.80%
Instruction scalability	100.00%	100.03%	100.03%	100.03%	100.03%	100.02%	100.02%	100.02%	100.02%	100.02%
Frequency scalability	100.00%	99.88%	99.79%	99.82%	99.71%	99.51%	99.21%	98.78%	98.52%	97.82%

Table 2: Analysis done on Tue Nov 25 10:16:19 AM CET 2025, par3113

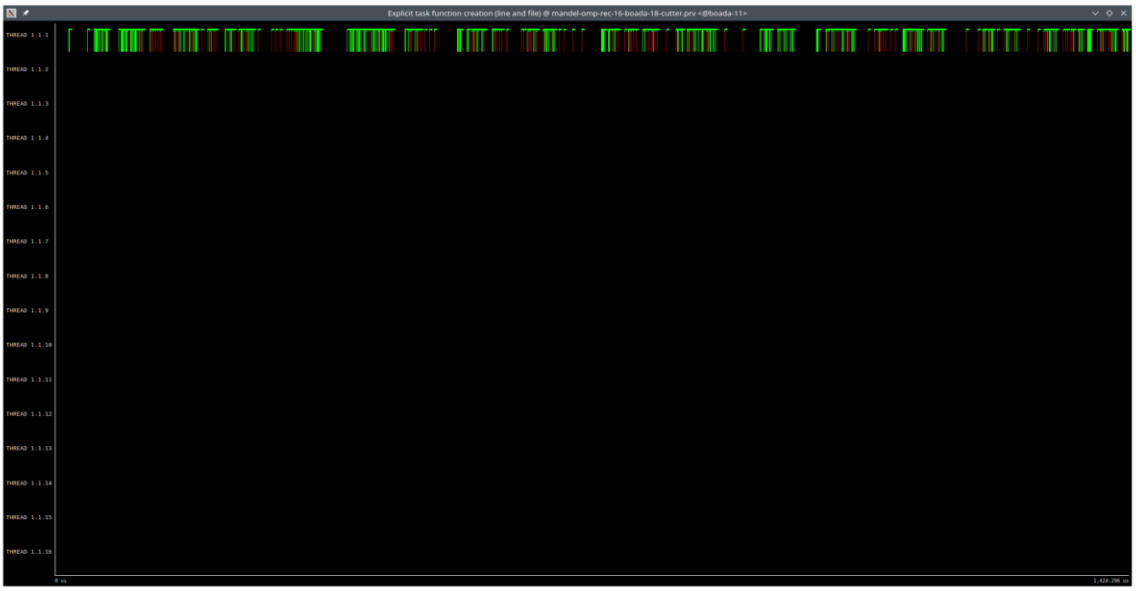
Statistics about explicit tasks in parallel fraction										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of explicit tasks executed (total)	1250.0	1250.0	1250.0	1250.0	1250.0	1250.0	1250.0	1250.0	1250.0	1250.0
LB (number of explicit tasks executed)	1.0	0.9	0.47	0.46	0.53	0.66	0.57	0.73	0.68	0.66
LB (time executing explicit tasks)	1.0	0.8	0.72	0.62	0.42	0.31	0.25	0.21	0.18	0.16
Time per explicit task (average us)	73.27	1273.87	1274.65	1275.59	1276.13	1276.3	1276.62	1277.29	1276.83	1277.63
Overhead per explicit task (synch %)	0.0	16.9	39.87	109.9	221.77	333.52	445.25	556.99	668.51	779.82
Overhead per explicit task (sched %)	0.35	0.04	0.05	0.1	0.12	0.13	0.14	0.14	0.13	0.15
Number of taskwait/taskgroup (total)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 3: Analysis done on Tue Nov 25 10:16:19 AM CET 2025, par3113

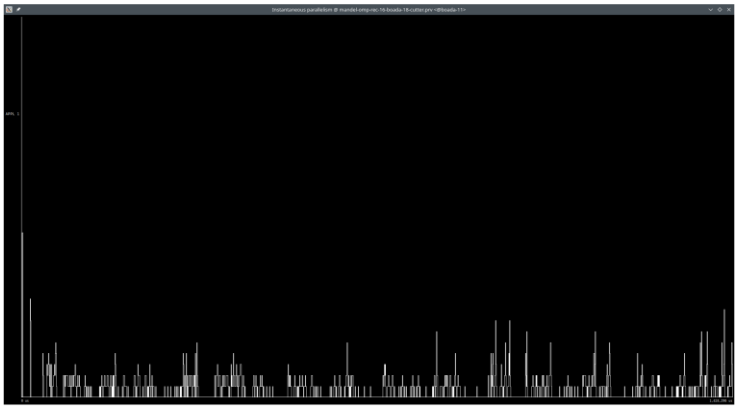
With the improved version we could observe the following:

- The cutoff successfully eliminates the overhead bottleneck. The synchronization overhead drops significantly, allowing better speed-ups.
- Still, there is a problem we didn't get rid of, Load Balancing. Its results are still poor, dropping to 18.25% at 32 threads. While better than the unimproved version, the static assignment of recursive blocks still leaves threads idle waiting for complex regions to finish.

Paraver analysis Unimproved Version

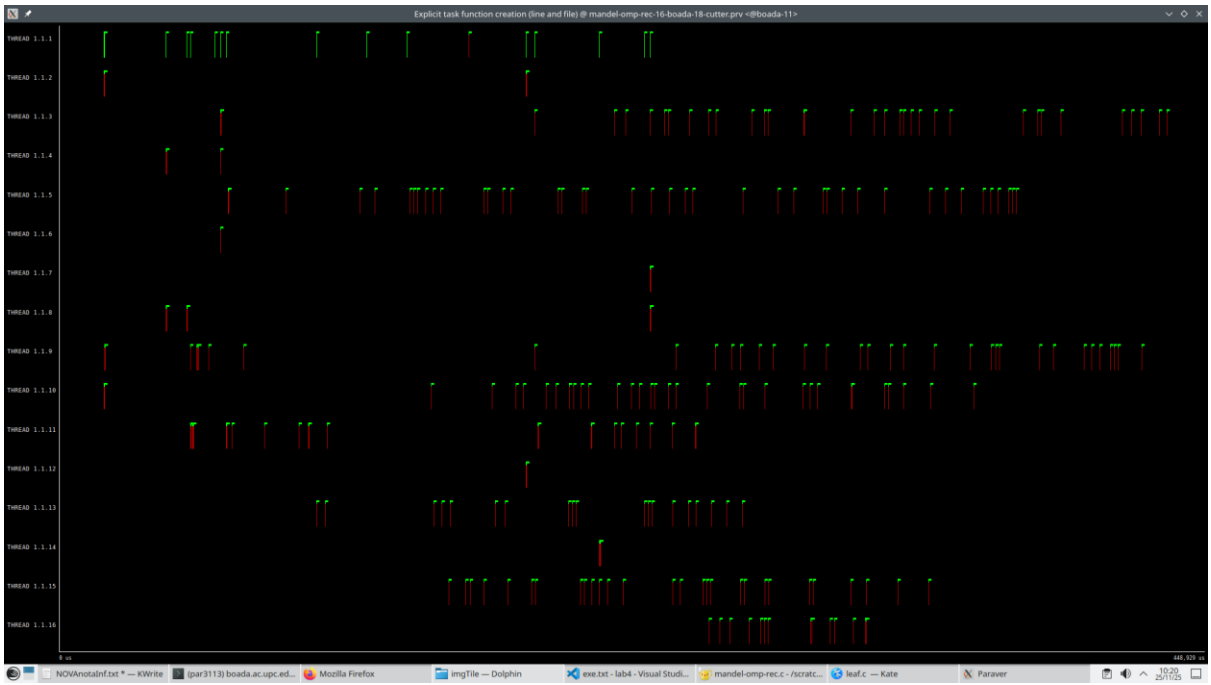


The Explicit Task Function Creation shows a dense and continuous wall of green/red across the entire timeline, indicating that the master thread is constantly overwhelmed by generating tasks throughout the execution.

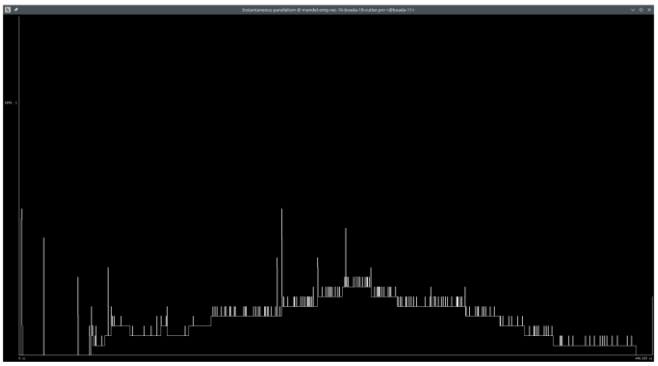


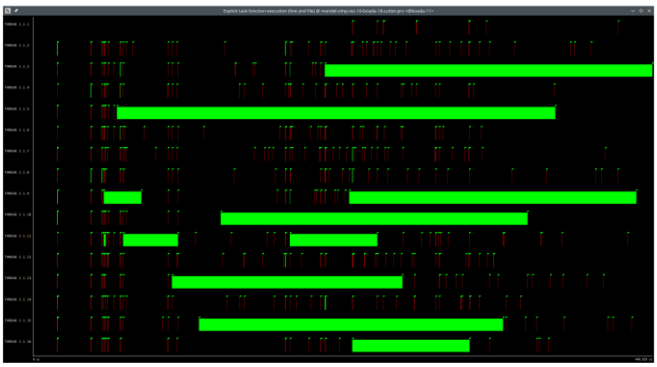
The instantaneous parallelism is erratic and low, never sustaining high values. The function execution is fragmented, with threads unable to settle into useful work due to constant scheduling interference.

Paraver analysis Improved Version



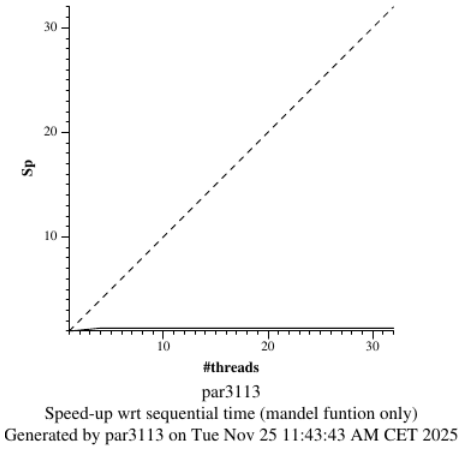
The Explicit Task Function Creation is now very sparse. The master thread generates a limited number of high-level tasks and the stops as intended by the cutoff logic.





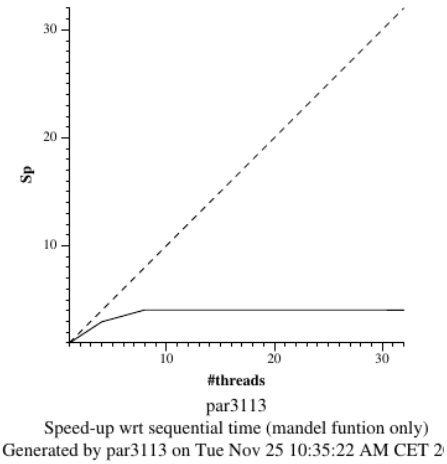
The Explicit Task Function Execution clearly shows large blocks of green bars (work) and visually confirms the Load Imbalance as some threads work for long periods, while others finish early. The instantaneous parallelism reflects this, showing a high peak that decays as threads run out of work.

Scalability Unimproved Version



The execution trace comparison highlights a complete failure in scalability. The speed-up line is nearly flat, maintaining a value of 1.29x approximately regardless of the thread count. The overhead of creating millions of tasks negates any benefit that parallelism could have.

Scalability Improved Version



The improved version shows a significant recovery in performance. The speed-up curve rises approximately to 4.06x. However, similarly to the Tile Strategy, the curve saturates early and flattens out. This confirms that while the cutoff fixed the overhead, Load Imbalance remains the limiting factor for further scalability.

3.2. Tree Strategy

The sequential code was parallelized using OpenMP tasking with the following modifications:

- A #pragma omp parallel combined with #pragma omp single directive ensures that only one thread generates task
- A taskgroup to exploit the parallelism in the horizontal and vertical borders, which waits because of the implicit task barrier before assigning a value to the equal variable.
- A task to the if-then block.
- A task with firstprivate(y, x) for the compute loop in the else block
- A task for each recursive that follows the tree strategy.
- Atomic and critical directives to protect updates to shared data and ensure that only one thread accesses the X11 display at a time.

Modelfactor Analysis

Overview of whole program execution metrics										
Number of threads	1	2	4	8	12	16	20	24	28	32
Elapsed time (sec)	1.85	0.94	0.48	0.26	0.19	0.15	0.13	0.12	0.11	0.10
Speedup	1.00	1.97	3.84	7.02	9.73	11.94	13.82	15.16	16.69	17.76
Efficiency	1.00	0.99	0.96	0.88	0.81	0.75	0.69	0.63	0.60	0.56

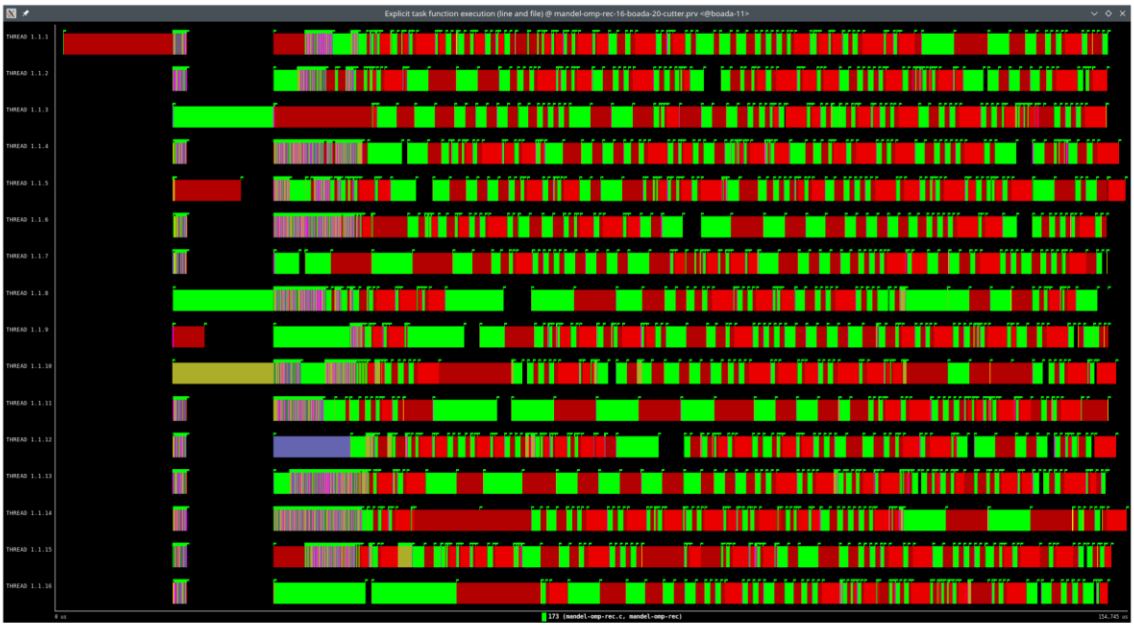
Table 1: Analysis done on Tue Dec 2 11:04:34 AM CET 2025, par3113

Overview of the Efficiency metrics in parallel fraction, ϕ=99.98%										
Number of threads	1	2	4	8	12	16	20	24	28	32
Global efficiency	99.69%	98.25%	95.76%	87.62%	80.97%	74.53%	69.01%	63.11%	59.56%	55.47%
Parallelization strategy efficiency	99.69%	98.20%	95.75%	87.83%	81.41%	75.24%	69.90%	64.26%	60.83%	57.12%
Load balancing	100.00%	99.17%	97.40%	95.87%	91.55%	86.64%	83.70%	78.31%	81.50%	80.41%
In execution efficiency	99.69%	99.03%	98.30%	91.62%	88.92%	86.84%	83.52%	82.05%	74.65%	71.03%
Scalability for computation tasks	100.00%	100.04%	100.01%	99.75%	99.47%	99.07%	98.72%	98.21%	97.90%	97.11%
IPC scalability	100.00%	99.77%	99.75%	99.58%	99.42%	99.28%	99.23%	99.15%	99.22%	99.13%
Instruction scalability	100.00%	100.33%	100.33%	100.33%	100.33%	100.33%	100.33%	100.33%	100.33%	100.33%
Frequency scalability	100.00%	99.94%	99.93%	99.84%	99.71%	99.46%	99.16%	98.73%	98.35%	97.64%

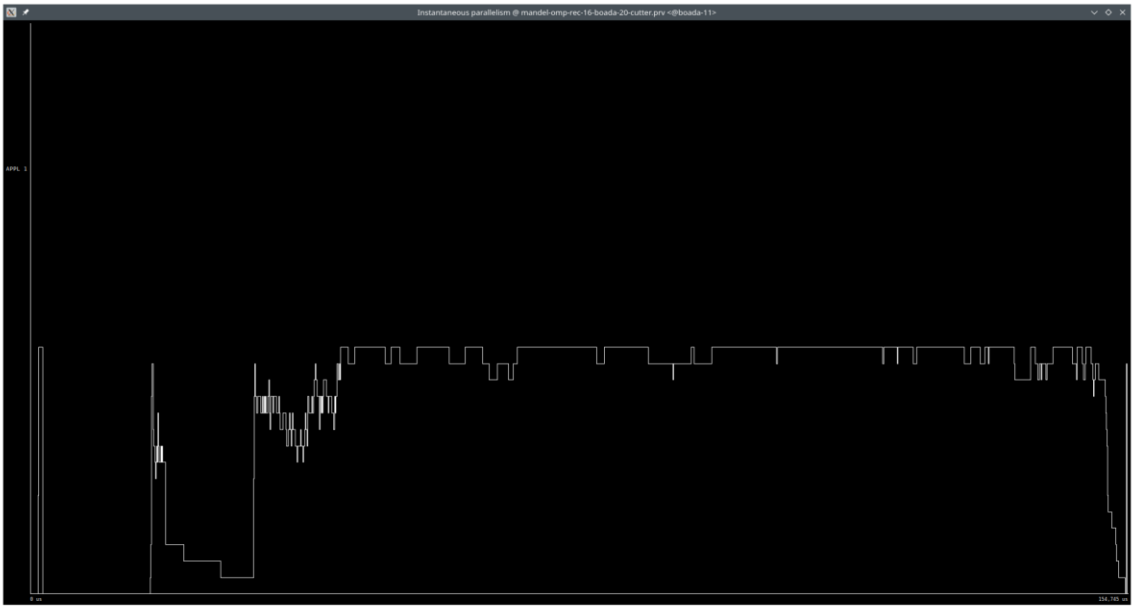
Table 2: Analysis done on Tue Dec 2 11:04:34 AM CET 2025, par3113

Statistics about explicit tasks in parallel fraction										
Number of threads	1	2	4	8	12	16	20	24	28	32
Number of explicit tasks executed (total)	15153.0	15153.0	15153.0	15153.0	15153.0	15153.0	15153.0	15153.0	15153.0	15153.0
LB (number of explicit tasks executed)	1.0	0.96	0.56	0.51	0.53	0.53	0.57	0.51	0.53	0.42
LB (time executing explicit tasks)	1.0	0.99	0.97	0.96	0.92	0.89	0.87	0.84	0.86	0.83
Time per explicit task (average us)	121.19	121.59	121.66	122.38	124.96	126.18	127.88	131.1	131.53	133.7
Overhead per explicit task (synch %)	0.14	1.58	4.09	12.82	20.29	28.61	36.54	44.62	52.23	60.26
Overhead per explicit task (sched %)	0.18	0.23	0.3	0.62	1.36	2.55	3.94	5.66	6.8	7.89
Number of taskwait/taskgroup (total)	4041.0	4041.0	4041.0	4041.0	4041.0	4041.0	4041.0	4041.0	4041.0	4041.0

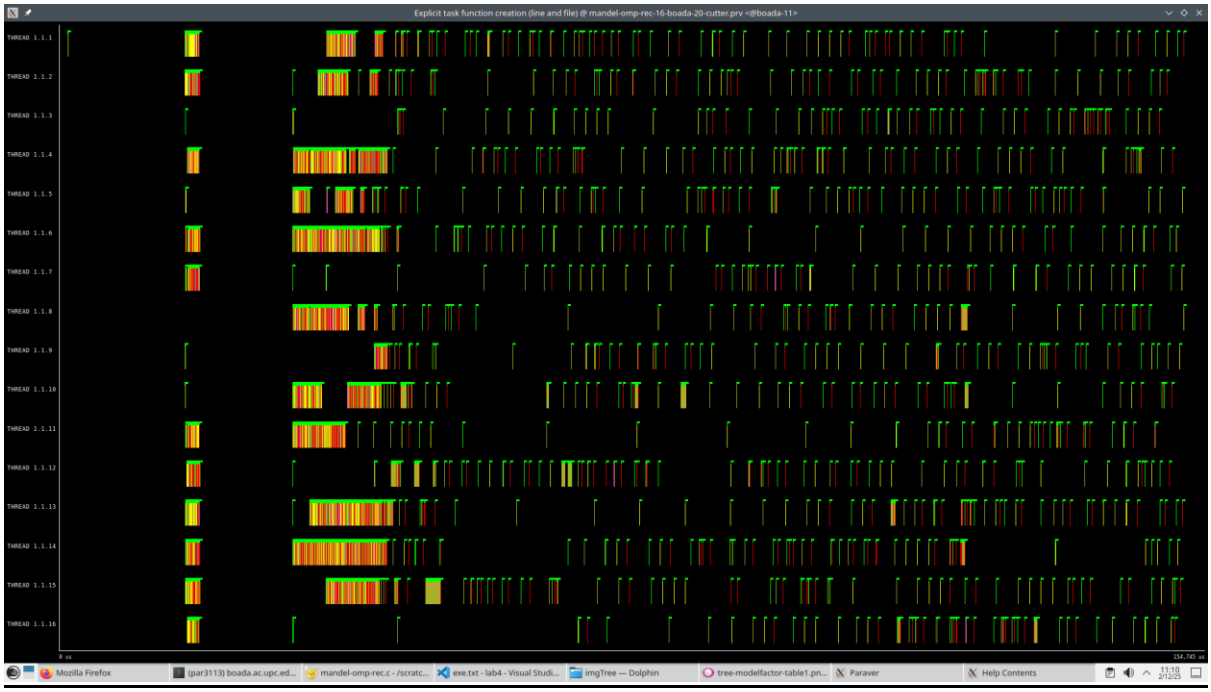
Paraver analysis



The explicit task function execution confirms a better load balancing with all the threads executing almost the same amount of work unlike the leaf version.

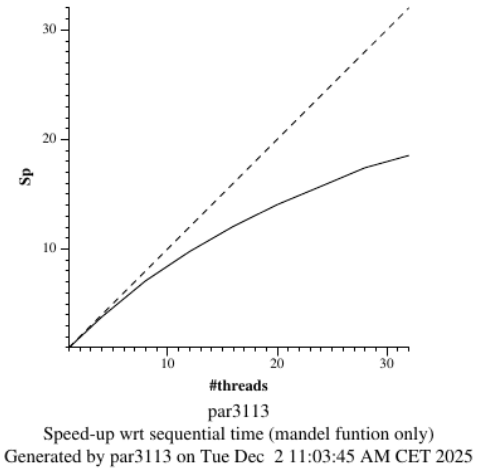


The instantaneous parallelism shows that at the beginning of the execution, parallelism is very low because the tree decomposition has not generated enough tasks. Once the recursive tree expands the parallelism rises and remains stable for most of the execution. In the leaf version we observed a similar initial behaviour, but it was worse and less stable as it was constantly dropping.



The explicit task function creation shows that all the threads create about the same amount of tasks and that the master thread does not stop since there is no cutoff logic.

Scalability



The scalability is better than the leaf version because the load imbalance is resolved. The speed increases from 4.05 to 17.76 at 32 threads and instead of flattening abruptly, it decreases gradually.

4.Conclusions

Version	**1**	**4**	**8**	**12**	**16**	**32**
Iterative: Tile	3.48	1.03	0.74	0.73	0.73	0.73
Iterative: Fine grain	3.49	0.88	0.45	0.31	0.24	0.32
Recursive: Leaf (Unimproved)	1.83	1.42	1.42	1.42	1.42	1.43
Recursive: Improved Leaf	1.82	0.62	0.45	0.45	0.45	0.45
Recursive: Tree	1.85	0.48	0.26	0.19	0.15	0.10
Best parallel strategy	Recursive Tree Strategy					
It has the quickest task decomposition among all versions. Even though global efficiency drops to 56%, it achieves the lowest elapsed time of all versions (0.10s), successfully solving the Load Balancing issue ensuring threads are kept busy throughout the execution.						