# PAR Laboratory 5

Font i Cabarrocas, Marc — Jara Palos, Adrià

28th December 2024

# Contents

## 1 Introduction

Here is the introduction of a C2 English level for the provided file *lab5-PAR.pdf*.

In this lab section.....

In case you haven't noticed you've fallen right into our trap, making you believe this was generated by AI, well sorry to disappoint but the introduction comes from our heads as this joke comes from our hearts.

This fifth laboratory assignment focuses on parallel data decomposition implementation and analysis, building upon a simplified sequential Mandelbrot set computation. The sequential code, while streamlined compared to previous labs, maintains the same data dependency patterns analyzed in Lab 3, providing a foundation for our parallel implementations.

Our task encompasses implementing and analyzing three distinct parallel decomposition strategies:

- A column-based 1D Block Geometric Data Decomposition
- A column-based 1D Block-Cyclic Geometric Data Decomposition
- A row-based 1D Cyclic Geometric Data Decomposition

For each implementation, we will conduct a comprehensive analysis including:

- Correctness verification against the sequential implementation
- Performance evaluation through Modelfactors analysis
- Parallel execution behavior visualization using Paraver
- Memory hierarchy analysis, particularly focusing on L2 cache performance
- Strong scalability assessment with varying thread counts

The analysis will utilize various tools including submit-strong-extrae.sh for Modelfactors analysis, submit-strong-memory.sh for cache performance evaluation, and submit-strong-omp.sh for scalability testing. Our findings and implementations will be documented in a detailed report, accompanied by the corresponding OpenMP source code implementations.

This laboratory synthesizes concepts from previous assignments while introducing new perspectives on geometric data decomposition strategies and their impact on parallel performance.

## 2 Block Geometric Data Decomposition

Block Geometric Data Decomposition is a strategy for dividing data amongst parallel processes/threads in a contiguous ”block” pattern. In this approach, each thread is tasked with a chunk of the matrix, divided in columns, meaning that for a 1000 column matrix and 4 threads, each thread takes 250 continuous columns each, working independently on their respective block.

For this strategy, we modified the inner loop to process a contiguous block of columns. Each block is assigned to a thread by their id and the length is determined by

$$block\_size = \frac{COLS}{num\_threads}$$

Each thread computes its assigned columns without overlapping with each other.

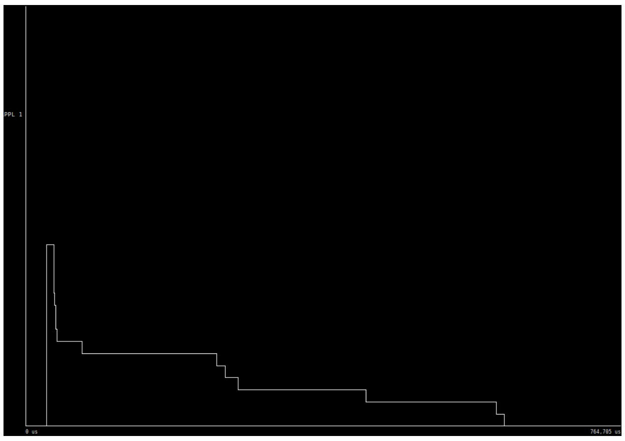


Figure 1: Instantaneous Parallelism for Block

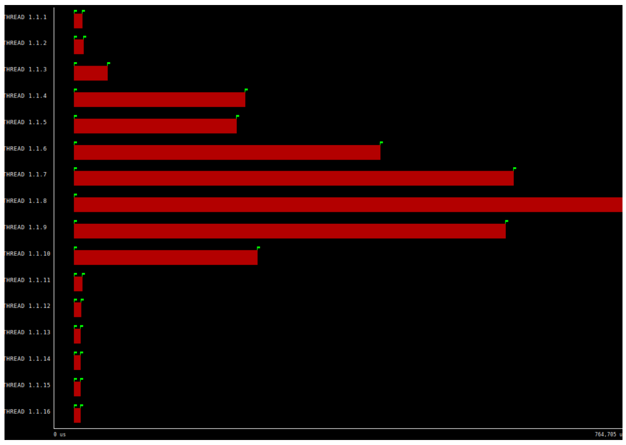


Figure 2: Implicit Tasks in Parallel Constructs for Block

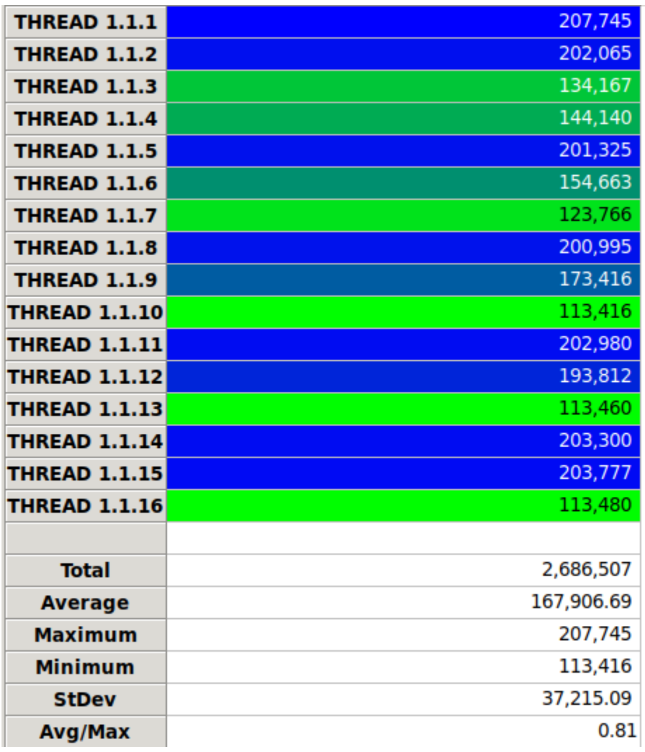
The Block decomposition by columns strategy, as visualized through Paraver traces across 16 threads, reveals significant load imbalance issues. Figure 1 illustrates this through its step-like pattern of parallel activity, indicating a sequential-like behaviour where threads work in an almost cascading fashion rather than concurrently. This observation is reinforced by Figure 2, which shows the individual thread timelines where the work is highly uneven. This pattern strongly suggests that the column-based block decomposition results in poor work distribution.

The modelfactors analysis (view tables 3, 4 & 5) showed a terrible global efficiency, reaching a value of 23.26% at 20 threads. The parallelization strategy efficiency had a similar value of 25.73%, meaning that as far as strategies go, this one is not efficient at all. However, the scalability for computation tasks stood above 90%.

The L2 cache analysis showed the highest initial cache misses at 1 thread (1,642,906) and exhibits poor scaling with increased thread count, culminating in 149,934 misses at 20 threads, suggesting inefficient memory access patterns and potential cache line conflicts due to the column-wise data access strategy.

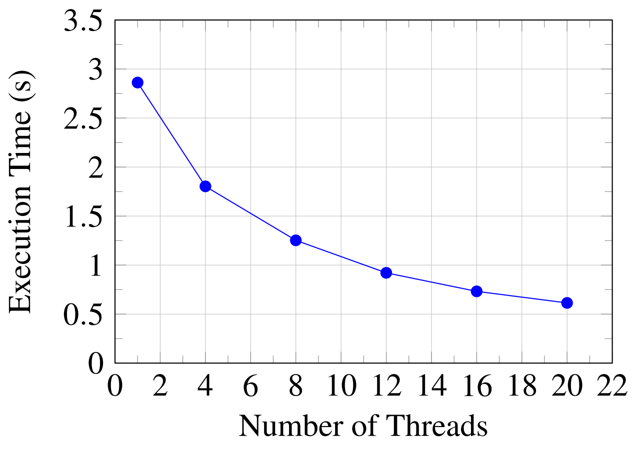
As per the amount of misses in each thread (view Figure 3) it showed an unbalanced amount of misses. This could be due to the fact that there was a lot of load unbalance to begin with so it’s only logical that the threads that finish the work earlier have the least amount of misses.

Finally, in figure 4 we can observe that the strategy exhibits poor scalability characteristics, achieving a maximum *speedup* of only 4.65x with 20 threads and showing significant performance degradation as the number of threads increases. The execution time reduction becomes minimal after 8 threads, demonstrating that this strategy fails to effectively utilize the additional computational resources.

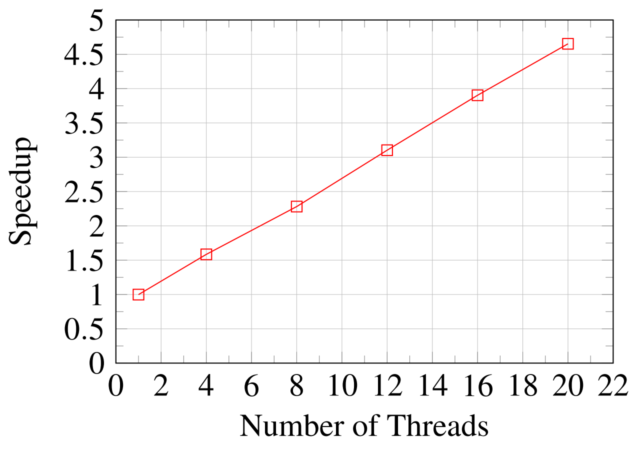


| | |
|---|---|
| THREAD 1.1.1 | 207,745 |
| THREAD 1.1.2 | 202,065 |
| THREAD 1.1.3 | 134,167 |
| THREAD 1.1.4 | 144,140 |
| THREAD 1.1.5 | 201,325 |
| THREAD 1.1.6 | 154,663 |
| THREAD 1.1.7 | 123,766 |
| THREAD 1.1.8 | 200,995 |
| THREAD 1.1.9 | 173,416 |
| THREAD 1.1.10 | 113,416 |
| THREAD 1.1.11 | 202,980 |
| THREAD 1.1.12 | 193,812 |
| THREAD 1.1.13 | 113,460 |
| THREAD 1.1.14 | 203,300 |
| THREAD 1.1.15 | 203,777 |
| THREAD 1.1.16 | 113,480 |
| | |
| Total | 2,686,507 |
| Average | 167,906.69 |
| Maximum | 207,745 |
| Minimum | 113,416 |
| StDev | 37,215.09 |
| Avg/Max | 0.81 |

Figure 3: Misses for each thread in Block



(a) Execution Time



(b) Speedup

Figure 4: Scalability analysis of Block Geometric Data Decomposition

## 3 Block-Cyclic Geometric Data Decomposition

Block-Cyclic Geometric Data Decomposition is a strategy similar to Block Geometric Data Decomposition. The key difference in this approach is that instead of dividing the data into single large blocks, the data is divided into multiple smaller blocks and distribute them cyclically among threads. This approach theoretically achieves a better load balancing and would work better for applications where work varies across the data space, like the Mandelbrot Set.

To implement this strategy we added an outer loop that iterates over indices assigned to each thread. The inner loop computes all columns before moving to the next blockk, ensuring cyclic distribution.

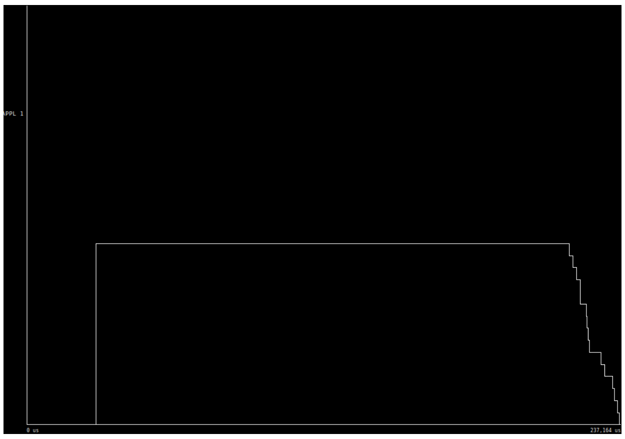


Figure 5: Instantaneous Parallelism for Block Cyclic

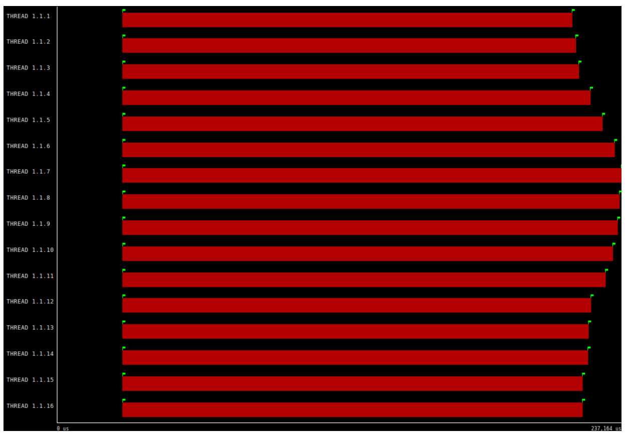


Figure 6: Implicit Tasks in Parallel Constructs for Block Cyclic

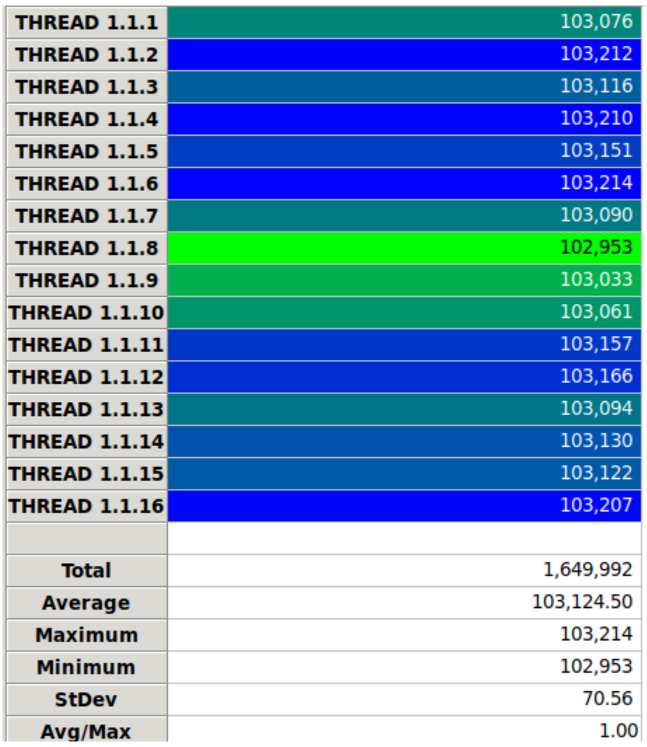
The Block Cyclic Geometric Data decomposition by columns, as shown in the Paraver traces over 16 threads, exhibits nearly ideal parallel execution. Figure 5 shows uniform parallel activity throughout most of the execution, followed by a brief synchronized completion. Similarly, Figure 6 highlights balanced thread timelines with nearly identical execution durations. This distribution strategy effectively resolves load balancing issues from simple block decomposition while preserving data locality, ensuring efficient parallel performance.

The modelfactors analysis (view tables 6 7 & 8) showed that the strategy has both great global and parallelization strategy efficiency, reaching values of 82.74% and 91.61% respectively. The scalability for computation of tasks also displayed a high value of 90.32%.

The 1D Block-Cyclic Geometric Data Decomposition by columns demonstrates improved cache behaviour compared to the block strategy, especially at higher thread counts. Starting with 1,642,968 misses at 1 thread and reducing to 132,414 at 20 threads, this approach benefits from better data distribution that helps mitigate some of the cache-related issues inherent in column-based access patterns.

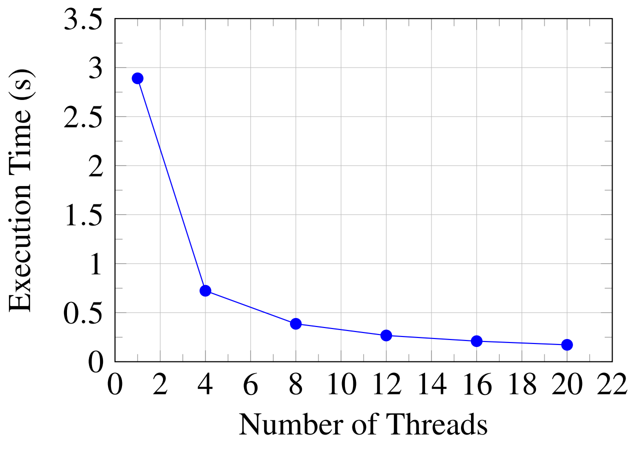
When accounting for the amount of misses of each thread (view Figure 7), it showed great miss distribution as the difference between the thread with the most misses and the thread with the least was no more than 261.

As per the scalability analysis (view Figure 8), it presents substantially improved scalability compared to the simple block approach, reaching a speedup of 16.59x with 20 threads and maintaining near-linear speedup up to 8 threads (7.39x). Although efficiency begins to decrease at higher thread counts, it maintains consistent performance improvements throughout, reducing execution time from 2.89s to 0.17s with 20 threads, indicating effective work distribution and better resource utilization.

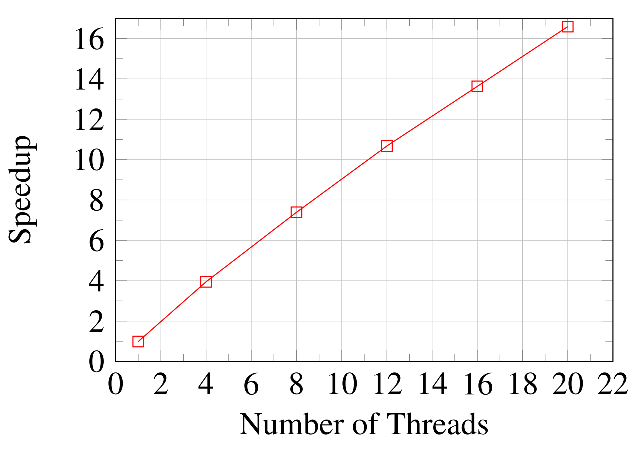


| | |
|---|---|
| THREAD 1.1.1 | 103,076 |
| THREAD 1.1.2 | 103,212 |
| THREAD 1.1.3 | 103,116 |
| THREAD 1.1.4 | 103,210 |
| THREAD 1.1.5 | 103,151 |
| THREAD 1.1.6 | 103,214 |
| THREAD 1.1.7 | 103,090 |
| THREAD 1.1.8 | 102,953 |
| THREAD 1.1.9 | 103,033 |
| THREAD 1.1.10 | 103,061 |
| THREAD 1.1.11 | 103,157 |
| THREAD 1.1.12 | 103,166 |
| THREAD 1.1.13 | 103,094 |
| THREAD 1.1.14 | 103,130 |
| THREAD 1.1.15 | 103,122 |
| THREAD 1.1.16 | 103,207 |
| | |
| Total | 1,649,992 |
| Average | 103,124.50 |
| Maximum | 103,214 |
| Minimum | 102,953 |
| StDev | 70.56 |
| Avg/Max | 1.00 |

Figure 7: Misses for each thread in Block



(a) Execution Time



(b) Speedup

Figure 8: Scalability analysis of Block-Cyclic Geometric Data Decomposition

## 4 Cyclic Geometric Data Decomposition

In Cyclic Geometric Data Decomposition individual rows or columns are assigned to threads in a round-robin fashion. This approach achieves excellent load balancing for irregular workloads since the work is evenly distributed at a very fine granularity, but it comes at the cost of potentially poor cache utilization due to non-contiguous memory access patterns.

To implement this strategy we modified the outer loop to distribute rows cyclically among all threads. Each thread starts on their given id and computes rows by increasing the index by the total number of threads for each iteration. For example, thread 0 will compute rows 0, 4, 8, 12, etc. if the number of threads is 4.

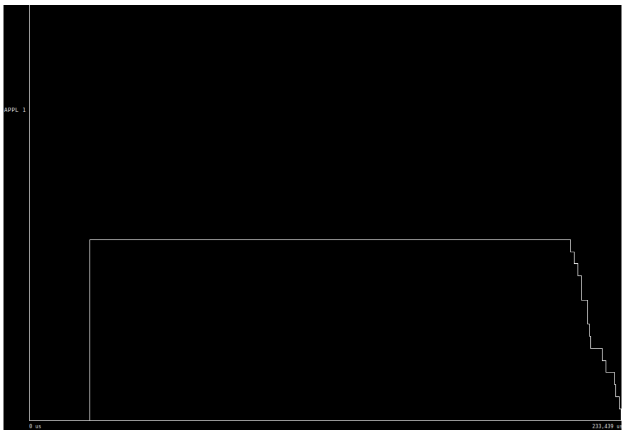


Figure 9: Instantaneous Parallelism for Cyclic

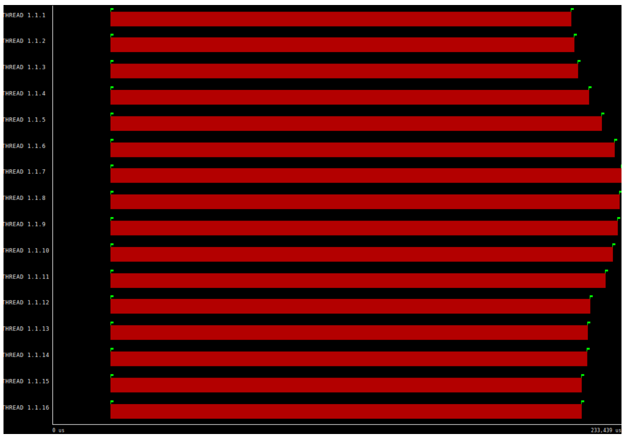


Figure 10: Implicit Tasks in Parallel Constructs for Cyclic

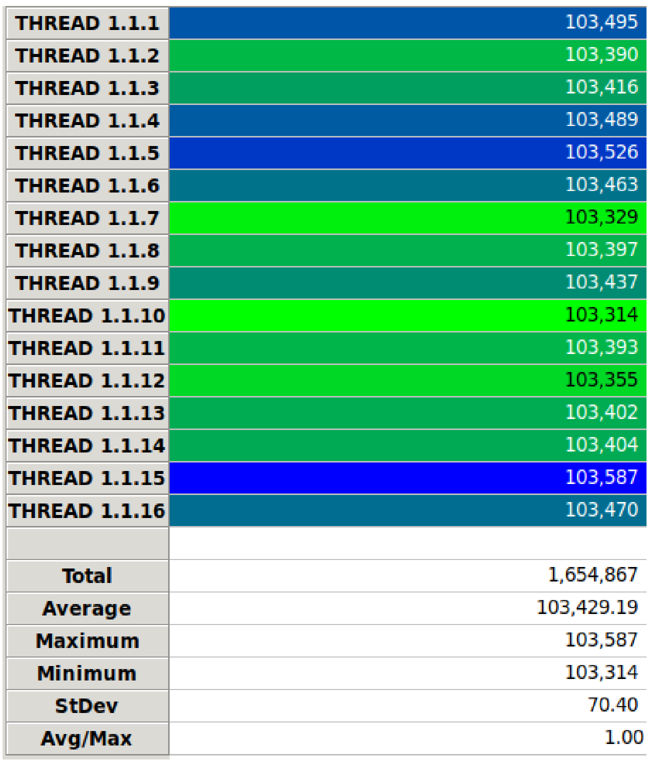
The paraver analysis demonstrates generally effective parallel performance across 16 threads. As shown in Figure 9, the level of parallelism visualization reveals a characteristic triangular pattern towards the execution's end, indicating a gradual decrease in parallel activity as threads complete their assigned work at slightly different times. This observation is further supported by Figure 10, which displays the individual thread timelines through red bars, showing predominantly uniform work distribution across all threads but with minor variations in completion times.

The modelfactors analysis on this strategy (view tables 9, 10 & 11) showed the best results of the entire assignment. At 20 threads, it achieved a global efficiency of 88.05% and a parallelization strategy efficiency of 98.32%. The scalability for computation of tasks, although slightly lower than its peers, stood at 89.55%.

This strategy exhibited the best cache behaviour among all strategies. Beginning with 1,642,452 cache misses at one thread, it achieved the lowest number of cache misses (83,811) with 20 threads. This superior cache performance can be attributed to its row-wise access pattern and cyclic distribution, which provides optimal data locality and minimizes cache conflicts. The reduced cache misses directly correlate with its excellent scalability, as fewer cache misses mean less time spent waiting for data from memory, allowing for more efficient parallel execution.

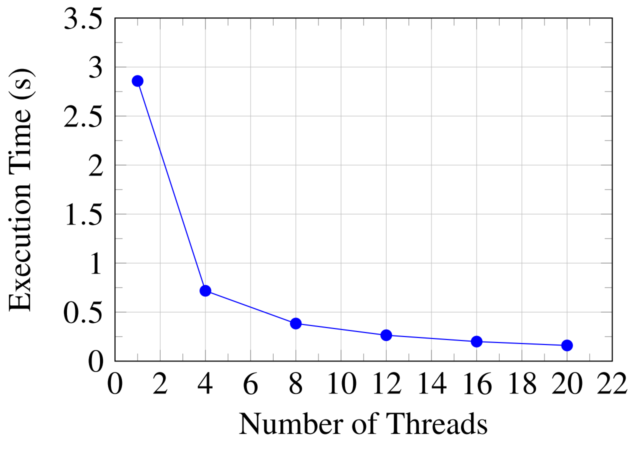
As per the misses for each individual thread, it's safe to say it obtained a very well-balanced result, with a difference between the thread with most misses and the one with the least of just 273 misses. Truly a remarkable feat and second only to the Block-Cyclic strategy

The Cyclic Geometric Data Decomposition emerged as the most efficient strategy (view Figure 12), demonstrating superior scalability with a maximum speedup of 17.83x using 20 threads and maintaining the most consistent near-linear scaling across all thread counts. This strategy achieves the best

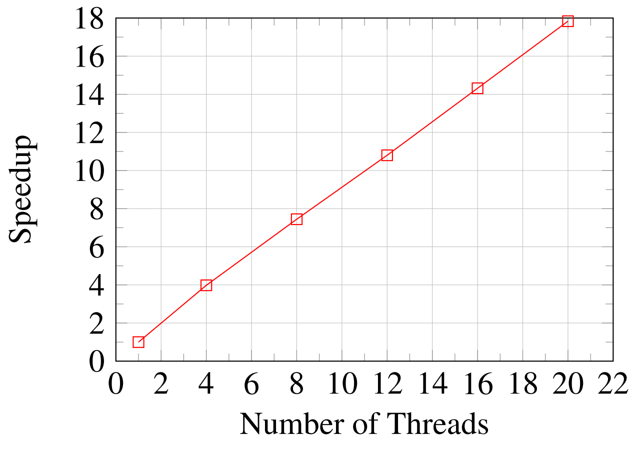


| | |
|---|---|
| THREAD 1.1.1 | 103,495 |
| THREAD 1.1.2 | 103,390 |
| THREAD 1.1.3 | 103,416 |
| THREAD 1.1.4 | 103,489 |
| THREAD 1.1.5 | 103,526 |
| THREAD 1.1.6 | 103,463 |
| THREAD 1.1.7 | 103,329 |
| THREAD 1.1.8 | 103,397 |
| THREAD 1.1.9 | 103,437 |
| THREAD 1.1.10 | 103,314 |
| THREAD 1.1.11 | 103,393 |
| THREAD 1.1.12 | 103,355 |
| THREAD 1.1.13 | 103,402 |
| THREAD 1.1.14 | 103,404 |
| THREAD 1.1.15 | 103,587 |
| THREAD 1.1.16 | 103,470 |
| | |
| Total | 1,654,867 |
| Average | 103,429.19 |
| Maximum | 103,587 |
| Minimum | 103,314 |
| StDev | 70.40 |
| Avg/Max | 1.00 |

Figure 11: Misses for each thread in Cyclic



(a) Execution Time



(b) Speedup

Figure 12: Scalability analysis of Cyclic Geometric Data Decomposition

execution time reduction (from 2.86s to 0.16s) and exhibits excellent efficiency even at higher thread counts, suggesting optimal load balancing and cache utilization patterns that effectively leverage the increased parallelism.

## 5 Conclusions

| **Version** | **1** | **4** | **8** | **12** | **16** | **20** |
|---|---|---|---|---|---|---|
| 1D Block Geometric Data Decomposition by columns | 2.862 | 1.803 | 1.252 | 0.921 | 0.732 | 0.614 |
| 1D Block-Cyclic Geometric Data Decomposition by columns | 2.890 | 0.724 | 0.386 | 0.268 | 0.210 | 0.172 |
| 1D Cyclic Geometric Data Decomposition by rows | 2.857 | 0.718 | 0.384 | 0.264 | 0.199 | 0.160 |

Table 1: Elapsed time (seconds) for different numbers of threads.

| **Version** | **1** | **4** | **8** | **12** | **16** | **20** |
|---|---|---|---|---|---|---|
| 1D Block Geometric Data Decomposition by columns | 1642006 | 492987 | 290553 | 225959 | 168482 | 149934 |
| 1D Block-Cyclic Geometric Data Decomposition by columns | 1642683 | 1783991 | 1315216 | 565457 | 103638 | 132414 |
| 1D Cyclic Geometric Data Decomposition by rows | 1642452 | 412610 | 206819 | 138398 | 103925 | 83811 |

Table 2: L2 Cache misses (per thread) for different numbers of threads.

As we can see in tables 1, the Cyclic strategy provides the best elapsed time for computing the mandeobrot set, narrowly beating the Block Cyclic for some hundredths of a second at each thread. The Block strategy, in contrast, did the computations almost 4 times slower than the other two strategies.

As per the cache misses, in table 2 we can see that once again, the Cyclic strategy gave the least amount of misses throughout the execution. The Block strategy gave the highest number of misses per thread, whilst the Block Cyclic, started very poorly at a low number of threads but gradually improved as the number of threads increased, almost catching up to the Cyclic strategy.

To summarise, the best strategy for computing the Mandelbrot set is 1D Cyclic Geometric Data Decomposition by rows.

# 6 Anexes

## 6.1 Tables for Block

| Overview of whole program execution metrics | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Elapsed time (sec) | 2.90 | 2.10 | 1.83 | 1.57 | 1.28 | 1.10 | 0.95 | 0.86 | 0.77 | 0.72 | 0.64 |
| Speedup | 1.00 | 1.38 | 1.58 | 1.84 | 2.26 | 2.62 | 3.05 | 3.37 | 3.78 | 4.03 | 4.51 |
| Efficiency | 1.00 | 0.69 | 0.40 | 0.31 | 0.28 | 0.26 | 0.25 | 0.24 | 0.24 | 0.22 | 0.23 |

Table 3: Analysis done on Mon Dec 23 05:42:33 PM CET 2024, par1208

| Overview of the Efficiency metrics in parallel fraction, $\phi$=99.10% | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Global efficiency | 100.00% | 69.18% | 39.70% | 30.97% | 28.54% | 26.60% | 25.89% | 24.58% | 24.21% | 23.03% | 23.26% |
| Parallelization strategy efficiency | 100.00% | 69.22% | 39.78% | 31.54% | 29.82% | 28.27% | 27.84% | 26.61% | 26.45% | 25.36% | 25.73% |
| Load balancing | 100.00% | 69.23% | 39.80% | 31.56% | 29.84% | 28.30% | 27.88% | 26.65% | 26.50% | 25.43% | 25.81% |
| In execution efficiency | 100.00% | 99.99% | 99.95% | 99.95% | 99.93% | 99.89% | 99.86% | 99.85% | 99.81% | 99.73% | 99.71% |
| Scalability for computation tasks | 100.00% | 99.95% | 99.79% | 98.17% | 95.73% | 94.10% | 92.99% | 92.39% | 91.53% | 90.81% | 90.41% |
| IPC scalability | 100.00% | 100.00% | 100.01% | 99.99% | 100.00% | 100.00% | 100.00% | 99.95% | 99.91% | 99.86% | 99.95% |
| Instruction scalability | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 99.99% | 100.01% | 100.01% | 99.99% | 100.00% | 99.99% |
| Frequency scalability | 100.00% | 99.94% | 99.78% | 98.18% | 95.73% | 94.10% | 92.99% | 92.43% | 91.62% | 90.93% | 90.47% |

Table 4: Analysis done on Mon Dec 23 05:42:33 PM CET 2024, par1208

| Statistics about explicit tasks in parallel fraction | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 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 | 1.0 |
| Useful duration for implicit tasks (average us) | 2870287.79 | 1435914.7 | 719097.86 | 487281.49 | 374806.41 | 305039.18 | 257222.42 | 221905.99 | 195996.2 | 175600.95 | 158743.95 |
| Load balancing for implicit tasks | 1.0 | 0.69 | 0.4 | 0.32 | 0.3 | 0.28 | 0.28 | 0.27 | 0.27 | 0.25 | 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 | 0.0 |
| Time in fork/join implicit tasks (average us) | 31.08 | 311.97 | 1538880.04 | 1504642.4 | 1235201.95 | 1061287.28 | 909537.57 | 821627.93 | 730163.17 | 683912.71 | 609251.85 |

Table 5: Analysis done on Mon Dec 23 05:42:33 PM CET 2024, par1208

## 6.2 Tables for Block Cyclic

| Overview of whole program execution metrics | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Elapsed time (sec) | 2.92 | 1.47 | 0.79 | 0.54 | 0.42 | 0.35 | 0.29 | 0.26 | 0.23 | 0.21 | 0.20 |
| Speedup | 1.00 | 1.99 | 3.67 | 5.39 | 6.89 | 8.40 | 9.97 | 11.23 | 12.50 | 13.67 | 14.73 |
| Efficiency | 1.00 | 0.99 | 0.92 | 0.90 | 0.86 | 0.84 | 0.83 | 0.80 | 0.78 | 0.76 | 0.74 |

Table 6: Analysis done on Mon Dec 23 06:41:40 PM CET 2024, par1208

| Overview of the Efficiency metrics in parallel fraction, $\phi$=99.11% | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Global efficiency | 100.00% | 100.29% | 94.07% | 93.42% | 90.64% | 89.42% | 89.58% | 88.43% | 85.64% | 85.04% | 82.74% |
| Parallelization strategy efficiency | 100.00% | 99.94% | 97.32% | 99.65% | 98.05% | 98.72% | 98.89% | 97.60% | 94.34% | 94.22% | 91.61% |
| Load balancing | 100.00% | 99.98% | 97.40% | 99.77% | 98.28% | 99.04% | 99.29% | 98.16% | 94.98% | 95.04% | 92.73% |
| In execution efficiency | 100.00% | 99.96% | 99.92% | 99.88% | 99.77% | 99.68% | 99.60% | 99.44% | 99.33% | 99.14% | 98.80% |
| Scalability for computation tasks | 100.00% | 100.35% | 96.66% | 93.75% | 92.44% | 90.58% | 90.58% | 90.61% | 90.77% | 90.25% | 90.32% |
| IPC scalability | 100.00% | 99.01% | 97.82% | 97.74% | 97.93% | 97.55% | 97.61% | 97.77% | 98.04% | 97.69% | 97.91% |
| Instruction scalability | 100.00% | 101.49% | 102.25% | 102.50% | 102.63% | 102.71% | 102.76% | 102.79% | 102.82% | 102.84% | 102.86% |
| Frequency scalability | 100.00% | 99.87% | 96.64% | 93.57% | 91.98% | 90.40% | 90.30% | 90.16% | 90.05% | 89.83% | 89.69% |

Table 7: Analysis done on Mon Dec 23 06:41:40 PM CET 2024, par1208

| Statistics about explicit tasks in parallel fraction | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 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 | 1.0 |
| Useful duration for implicit tasks (average us) | 2892625.7 | 1441280.86 | 748126.67 | 514232.12 | 391154.92 | 319353.93 | 266130.35 | 228039.17 | 199165.05 | 178059.6 | 160136.67 |
| Load balancing for implicit tasks | 1.0 | 1.0 | 0.97 | 1.0 | 0.98 | 0.99 | 0.99 | 0.98 | 0.95 | 0.95 | 0.93 |
| 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 | 0.0 |
| Time in fork/join implicit tasks (average us) | 29.59 | 1249.93 | 1131.86 | 1821.02 | 4685.71 | 4220.23 | 3690.12 | 9377.92 | 22075.61 | 11524.44 | 19279.19 |

Table 8: Analysis done on Mon Dec 23 06:41:40 PM CET 2024, par1208

## 6.3 Tables for Cyclic

| Overview of whole program execution metrics | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Elapsed time (sec) | 2.89 | 1.46 | 0.79 | 0.54 | 0.42 | 0.34 | 0.29 | 0.25 | 0.23 | 0.21 | 0.19 |
| Speedup | 1.00 | 1.98 | 3.67 | 5.39 | 6.86 | 8.38 | 9.94 | 11.34 | 12.74 | 14.01 | 15.34 |
| Efficiency | 1.00 | 0.99 | 0.92 | 0.90 | 0.86 | 0.84 | 0.83 | 0.81 | 0.80 | 0.78 | 0.77 |

Table 9: Analysis done on Mon Dec 23 05:29:27 PM CET 2024, par1208

| Overview of the Efficiency metrics in parallel fraction, $\phi$=99.07% | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| Global efficiency | 100.00% | 99.86% | 93.79% | 93.46% | 90.34% | 89.83% | 89.69% | 89.25% | 89.06% | 88.46% | 88.05% |
| Parallelization strategy efficiency | 100.00% | 99.93% | 97.09% | 99.82% | 98.27% | 99.48% | 99.34% | 99.01% | 98.93% | 98.55% | 98.32% |
| Load balancing | 100.00% | 99.97% | 97.16% | 99.95% | 98.47% | 99.87% | 99.79% | 99.55% | 99.67% | 99.52% | 99.50% |
| In execution efficiency | 100.00% | 99.96% | 99.93% | 99.88% | 99.80% | 99.60% | 99.56% | 99.46% | 99.25% | 99.03% | 98.81% |
| Scalability for computation tasks | 100.00% | 99.93% | 96.60% | 93.63% | 91.93% | 90.30% | 90.28% | 90.14% | 90.03% | 89.76% | 89.55% |
| IPC scalability | 100.00% | 99.99% | 99.98% | 99.98% | 99.98% | 99.94% | 99.98% | 99.98% | 99.97% | 99.97% | 99.96% |
| Instruction scalability | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |
| Frequency scalability | 100.00% | 99.94% | 96.62% | 93.64% | 91.95% | 90.36% | 90.30% | 90.16% | 90.06% | 89.79% | 89.58% |

Table 10: Analysis done on Mon Dec 23 05:29:27 PM CET 2024, par1208

| Statistics about explicit tasks in parallel fraction | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Number of processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 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 | 1.0 |
| Useful duration for implicit tasks (average us) | 2860333.69 | 1431126.74 | 740254.24 | 509174.06 | 388912.57 | 316760.69 | 264017.48 | 226667.1 | 198565.8 | 177032.67 | 159706.05 |
| Load balancing for implicit tasks | 1.0 | 1.0 | 0.97 | 1.0 | 0.98 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 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 | 0.0 |
| Time in fork/join implicit tasks (average us) | 26.73 | 1484.56 | 520.13 | 857.0 | 1281.37 | 2295.58 | 2466.95 | 2846.85 | 3075.64 | 3923.25 | 4279.81 |

Table 11: Analysis done on Mon Dec 23 05:29:27 PM CET 2024, par1208