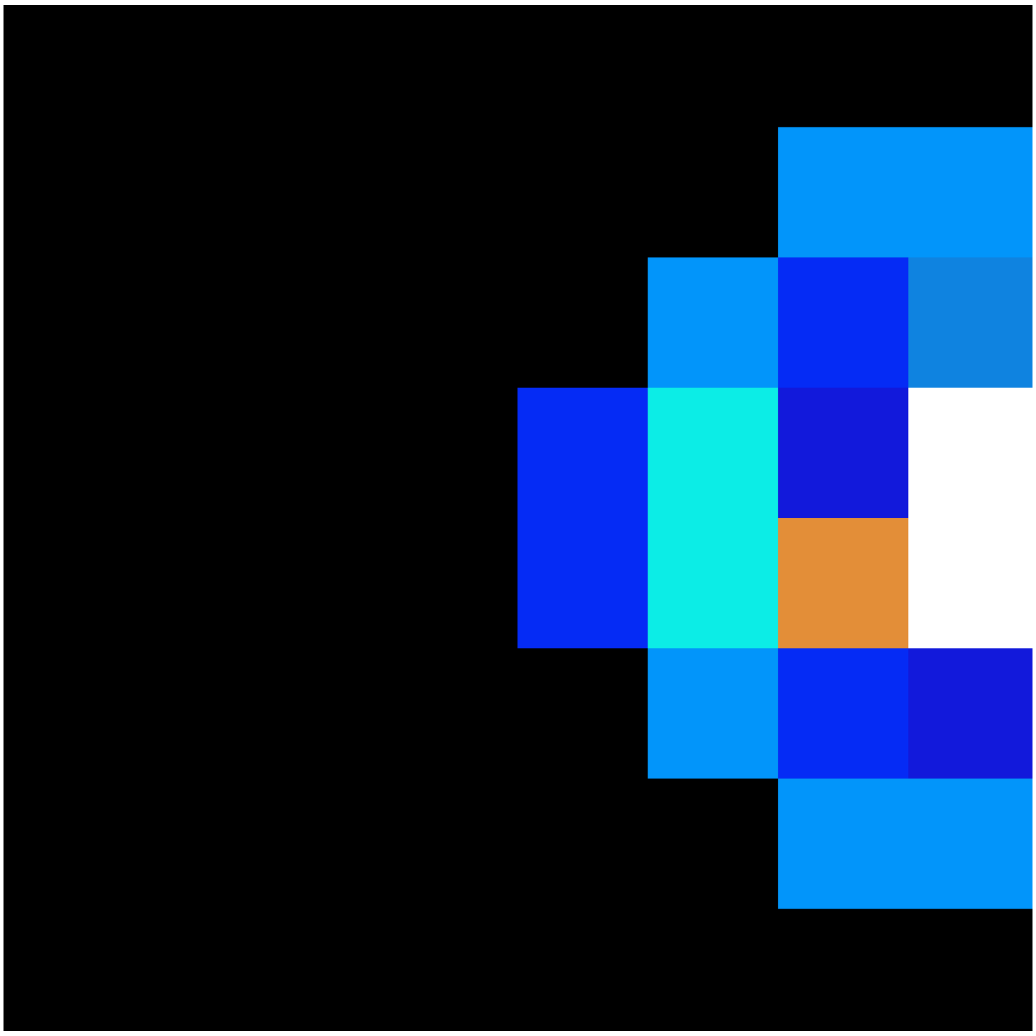
# PAR Laboratory 4

Font i Cabarrocas, Marc — Jara Palos, Adrià

November - December 2024



## Contents

## 1 Introduction

In this laboratory assignment, we continue our investigation of task decomposition strategies for the Mandelbrot set computation that we began in the previous session. The workload is divided into two main studies: iterative and recursive task decomposition (TD).

The iterative study consists of two parts. First, we will complete an unfinished parallel implementation of the Mandelbrot set computation, building upon our previous analysis of dependencies and data sharing to avoid concurrency issues. This implementation follows what we call the*Tile* strategy. Second, we will develop a Fine Grain strategy that maximizes parallelism by separately handling border checks and tile computations.

The recursive study similarly comprises two implementations. First, we will enhance a provided recursive code to exploit parallelism while maintaining proper synchronization, implementing the Leaf Strategy. Second, we will develop an OpenMP implementation of the Tree Strategy, designed to maximize parallel execution while carefully managing task creation and synchronization overhead.

# 2 Iterative Task Decomposition

## 2.1 Tile Analysis

In the first part of this lab assignment, we trace the work done in the previous lab report. The first thing we were asked to do in LAB3, was to fix the task decomposition in a code that computes the Mandelbrot Set so that it achieved parallelism using Tareador. This time, we were tasked with completing the original version, which will be referred to as *tile* hereafter, using OpenMP.

Revising the submission on LAB3, we were able to convert the work done with Tareador into a fully-functioning OMP version.

We added *#pragma omp atomic* to the updates of the histogram and a *#pragma omp critical* region on the X11's function calls to avoid data races that potentially changed the final results of the program.

As we can see in Figure 1,it starts with a lot of parallelism, and, as the tasks finish, the parallelism also dies out.

In Figure 2, we see the execution time of each of the tasks, this explains the behavior seen in Figure 1, as a lot of tasks are active in the beginning, but as the execution continues, more and more tasks end.

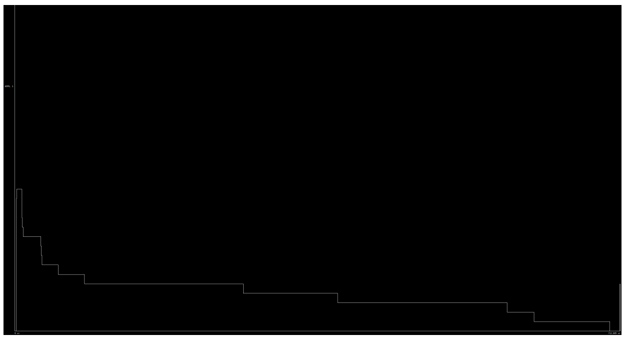


Figure 1: Instantaneous Parallelism for Tile Strategy

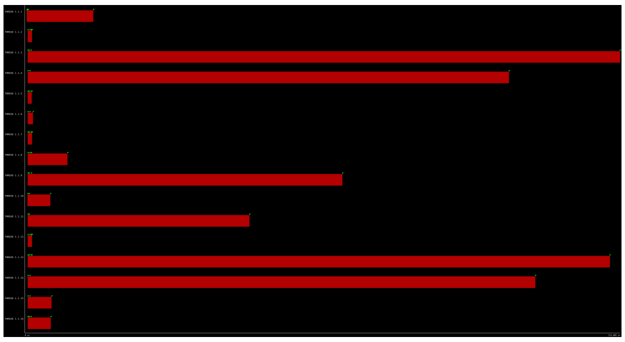


Figure 2: Explicit Task Function Execution for Tile Strategy

As per the modelfactors analysis, in Tables 2, 3 & 4 (provided in the Annexes Chapter) we can find that the global efficiency decreases rapidly as it reaches a minimum value of 27.10 with 16 threads. A similar case happens with the parallelization strategy efficiency, which at 16 threads has a value of 30.66. The only thing positive in this modelfactor analysis is the Scalability for Tasks, as it maintains a strong 88.37 at 16 threads.

Despite the third measure's apparent success, The first two parameters show concerning results, meaning that the parallelization strategy is far from perfect.

## 2.2 Fine Grain Analysis

The second part of the iterative section of the lab, was to implement a fully-functioning Fine Grain strategy on the tiled version of the code from scratch.

For this version, we created a task for each vertical and horizontal checks, ensuring they can work properly by splitting the *equals* variable into two. Which led us to a funny looking version of the mandelbrot set (see cover of the document). For simplicity and readability, we decided on the *depends* keywords on those two tasks, since the *mandel_IF* task also depended on the results given by the two previous checks. A final task is created on the compute and fill regions of the code.

The instantaneous parallelism graph (view Figure 3) shows a quick ramp-up followed by a sustained high level of parallelism for most of the execution time, with minimal fluctuations. This pattern suggests that Fine Grain achieves good parallelism with minimal overhead once the initial task distribution is complete

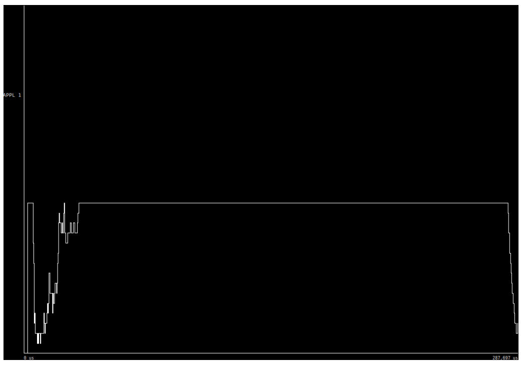


Figure 3: Instantaneous Parallelism for Fine Grain Strategy

The thread activity visualization (view Figures 4 & 5) reveals a brief initial setup period with some task creation and synchronization (red and yellow segments), followed by long continuous execution periods (green) across all 16 threads.

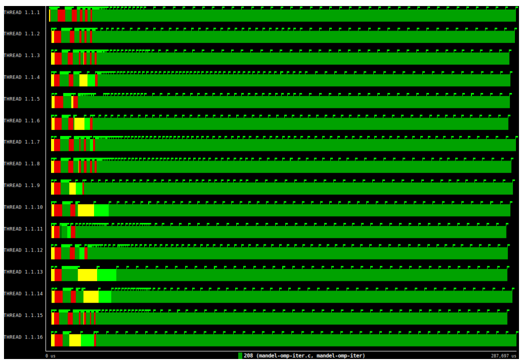


Figure 4: Explicit Task Function Execution for Fine Grain Strategy

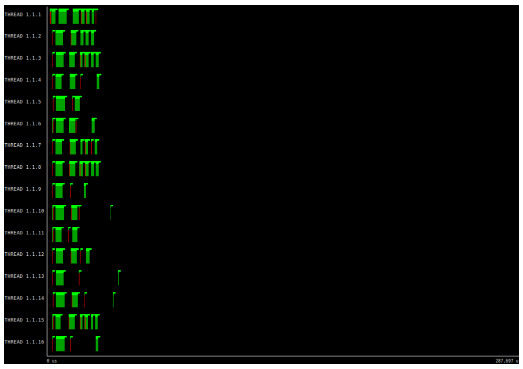


Figure 5: Explicit Task Function Creation for Fine Grain Strategy

After analyzing the modelfactor tables: 5, 6 & 7. The values for global efficiency, parallelization strategy efficiency and scalability for tasks showed great improvement when compared to the tile version, particularly the first 2. As global efficiency reached 82.56%, parallelization strategy efficiency a whopping 92.72% whilst maintaining the scalability for tasks at 89.04%.

This is thanks to the nature of the strategy which works great in the computation of the mandelbrot set, also helped by cleverly handling the vertical and horizontal checks.

## 3 Recursive Task Decomposition

### 3.1 Leaf Analysis

As we reach the recursive part of the work, we are to complete a recursive version of a program that computes the Mandelbrot set with the Leaf Strategy. To briefly sum up what the Leaf Strategy is, it consists of creating the tasks at the base cases of the recursion to achieve parallelism.

For the leaf pattern, we just added the atomic and critical regions to ensure no data race affects the outcome of the program like in Tile version, and a task that takes on the compute region of the code.

The instantaneous parallelism image (view Figure 6) shows highly variable and generally low parallelism levels with frequent spikes and drops rather than sustained parallel execution, demonstrating that the strategy fails to maintain consistent resource utilization throughout its execution time.

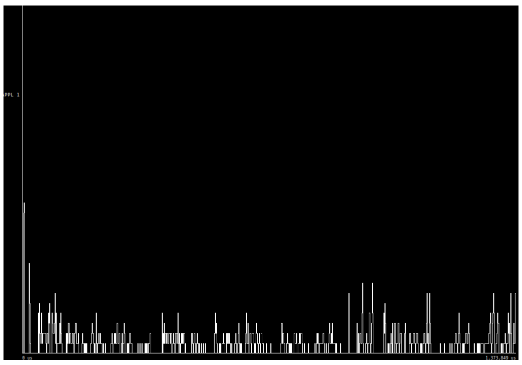


Figure 6: Instantaneous Parallelism for Leaf Strategy

Figure 7 shows thread activity with an unfavorable pattern of highly fragmented execution (green segments) and frequent synchronization points (red segments) across all 16 threads, with particularly concerning fragmentation in most threads. Figure 8 emphasizes how thread 1.1.1 becomes the primary executor while other threads show minimal activity, indicating poor load distribution.

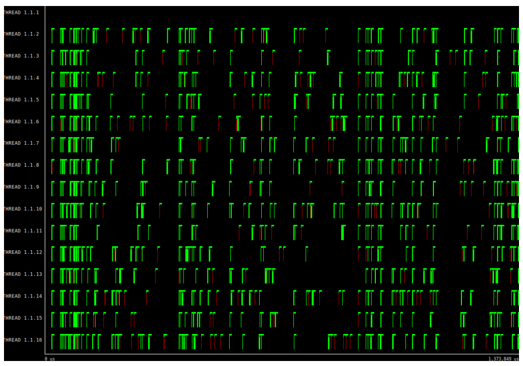


Figure 7: Explicit Task Function Execution for Leaf Strategy

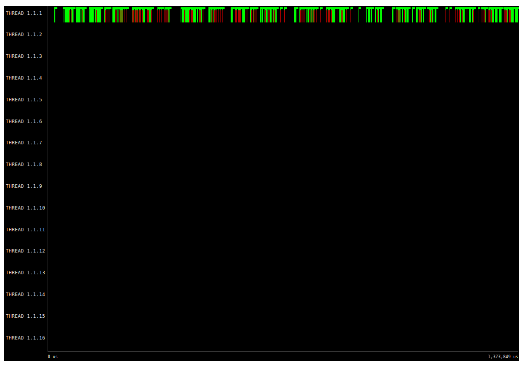


Figure 8: Explicit Task Function Creation for Leaf Strategy

The modelfactors analysis (view tables 8, 9 & 10) of this strategy served to prove it's by far the worst one. With a global efficiency of just 7.35% and a strategy efficiency of 8.25% at 16 threads, it shows that the Leaf strategy is pretty much worthless when used with mandelbrot. Nevertheless, it maintained a decent scalability at 89.20%.

## 3.2 Tree Analysis

The final part of this lab assignment was to implement the Tree Strategy on the recursive version by coding it entirely anew.

Like with Lab 3, we created omp tasks for the vertical and horizontal checks, and on each recursive call of the function. While this already got good results, we improved the overhead that each task created by implementing a cutoff value. This cutoff value, tied to *NCols* variable to avoid major changes on the code, was determined by manual testing and analysis of Modelfactors overhead and times. Ultimately, we set the cutoff value to 8.

After these tweaks to the code, we were able to generate the following images from *paraver* which lead us into the following conclusions.

The instantaneous parallelism graph (view Figure 9) reveals that, after an initial ramp-up, the strategy maintains a good level of parallel execution throughout most of its runtime, though with some expected drops at synchronization points.

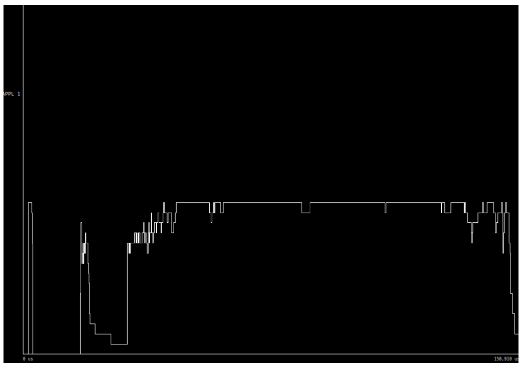


Figure 9: Instantaneous Parallelism for Tree Strategy

The burst patterns in task creation (view Figure 11) and the balanced distribution of work across threads (view Figure 10) confirm the tree-like decomposition approach, suggesting an efficient exploitation of available parallel resources despite the necessary overhead from task management.

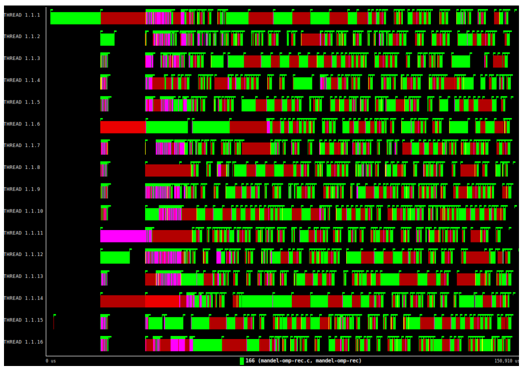


Figure 10: Explicit Task Function Execution for Tree Strategy

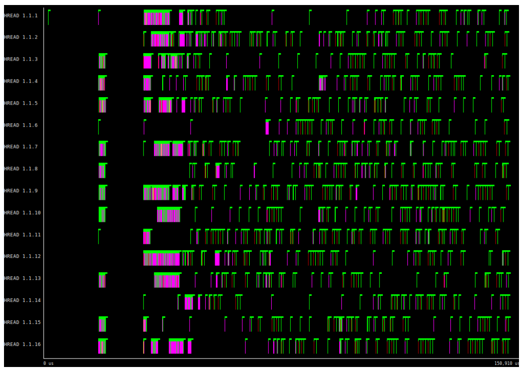


Figure 11: Explicit Task Function Creation for Tree Strategy

The modelfactors analysis (view tables 11, 12 & 13) showed mixed results, while it reached minimum elepsed time (0.15s, the best of all strategies) it offered less speedup than the 2nd best (Fine Grain), 10.71 to 13.20 respectively. The global efficiency stood above average at 66.88% and the strategy efficiency a notable 75.86%, this shows that there is space for improvement. The scalability for tasks computed was a respectable 88.16%.

## 4 Conclusion

| | **Number of Threads** | | | | | |
|---|---|---|---|---|---|---|
| **Version** | **1** | **4** | **8** | **12** | **16** | **20** |
| Iterative: Tile | 3.084 | 0.911 | 0.699 | 0.708 | 0.709 | 0.709 |
| Iterative: Finer grain | 3.777 | 0.951 | 0.510 | 0.353 | 0.267 | 0.215 |
| Recursive: Leaf | 1.611 | 1.241 | 1.325 | 1.364 | 1.369 | 1.370 |
| Recursive: Tree | 1.606 | 0.424 | 0.244 | 0.181 | 0.145 | 0.125 |

Table 1: Performance results for different versions and thread counts. (s)

As we can clearly see in table 1, the Tree Strategy offers the quickest task decomposition strategy among all versions. This contradicts the conclusion obtained in our previous lab report, in which we stated that Fine Grain TD was best, however, the results for the Tree Strategy speak for themselves, nearly halving the elapsed time computed for Fine Grain on all number of threads. Nevertheless, Fine Grain is overall more efficient, due to it creating a lot of tasks and executing them all at the same time, meanwhile, the Tree Strategy keeps creating tasks until the execution is almost finished. To prove this, we have but to compare Figures 4 and 10.

Leaf Strategy keeps on being the worst one, as per the last report, with the effect of parallelism hardly reflecting on the elapsed time with a notoriously bad efficiency too.

## 5 Annexes

**Modelfactors of Tile version:**

Table 2: Overview of whole program execution metrics (Tile)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| Elapsed time (sec) | 3.09 | 1.74 | 0.91 | 0.75 | 0.70 | 0.72 | 0.71 | 0.71 | 0.71 |
| Speedup | 1.00 | 1.77 | 3.39 | 4.12 | 4.41 | 4.28 | 4.35 | 4.34 | 4.33 |
| Efficiency | 1.00 | 0.89 | 0.85 | 0.69 | 0.55 | 0.43 | 0.36 | 0.31 | 0.27 |

Table 3: Overview of the Efficiency metrics in parallel fraction (Tile)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| Global efficiency | 100.00 | 88.68 | 84.83 | 68.72 | 55.11 | 42.83 | 36.25 | 31.02 | 27.10 |
| – Parallelization strategy efficiency | 100.00 | 88.95 | 85.25 | 73.76 | 59.40 | 47.68 | 40.49 | 34.85 | 30.66 |
| – Load balancing | 100.00 | 88.99 | 85.30 | 73.83 | 59.49 | 47.74 | 40.57 | 34.94 | 30.73 |
| – In execution efficiency | 100.00 | 99.96 | 99.94 | 99.90 | 99.85 | 99.88 | 99.80 | 99.76 | 99.78 |
| – Scalability for computation tasks | 100.00 | 99.70 | 99.52 | 93.17 | 92.78 | 89.82 | 89.52 | 88.99 | 88.37 |
| – IPC scalability | 100.00 | 99.96 | 99.92 | 99.94 | 99.91 | 99.90 | 99.92 | 99.91 | 99.91 |
| – Instruction scalability | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| – Frequency scalability | 100.00 | 99.73 | 99.60 | 93.23 | 92.86 | 89.91 | 89.60 | 89.07 | 88.46 |

Table 4: Statistics about explicit tasks in parallel fraction (Tile)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| Number of explicit tasks executed (total) | 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 | 0.43 | 0.47 | 0.38 | 0.31 | 0.27 | 0.24 |
| LB (time executing explicit tasks) | 1.0 | 0.89 | 0.85 | 0.74 | 0.59 | 0.48 | 0.40 | 0.35 | 0.31 |
| Time per explicit task (average) | 48285.48 | 48427.47 | 48504.25 | 51786.46 | 51972.14 | 53649.71 | 53770.9 | 54042.56 | 54345.45 |
| Overhead per explicit task (synch %) | 0.0 | 12.39 | 17.24 | 35.49 | 68.19 | 109.7 | 146.95 | 186.96 | 226.64 |
| Overhead per explicit task (sched %) | 0.0 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| Number of taskwait/taskgroup (total) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

**Modelfactors of Fine Grain version:**

Table 5: Overview of whole program execution metrics (FG)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| Elapsed time (sec) | 3.79 | 1.91 | 0.99 | 0.69 | 0.53 | 0.44 | 0.37 | 0.32 | 0.29 |
| Speedup | 1.00 | 1.98 | 3.81 | 5.49 | 7.15 | 8.68 | 10.31 | 11.94 | 13.20 |
| Efficiency | 1.00 | 0.99 | 0.95 | 0.91 | 0.89 | 0.87 | 0.86 | 0.85 | 0.83 |

Table 6: Overview of the Efficiency metrics in parallel fraction (FG)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| Parallel fraction (%) | 99.99 | | | | | | | | |
| Global efficiency (%) | 99.94 | 99.17 | 95.29 | 91.46 | 89.45 | 86.78 | 85.97 | 85.36 | 82.56 |
| – Parallelization strategy efficiency (%) | 99.94 | 99.25 | 98.56 | 97.85 | 97.33 | 96.19 | 95.49 | 94.83 | 92.72 |
| – Load balancing (%) | 100.00 | 99.90 | 99.79 | 99.50 | 99.41 | 99.17 | 98.68 | 97.48 | 96.54 |
| – In execution efficiency (%) | 99.94 | 99.36 | 98.76 | 98.34 | 97.91 | 96.99 | 96.77 | 97.28 | 96.05 |
| – Scalability for computation tasks (%) | 100.00 | 99.91 | 96.68 | 93.47 | 91.90 | 90.22 | 90.03 | 90.01 | 89.04 |
| – IPC scalability (%) | 100.00 | 99.85 | 99.80 | 99.75 | 99.80 | 99.76 | 99.76 | 99.77 | 99.70 |
| – Instruction scalability (%) | 100.00 | 100.08 | 100.08 | 100.08 | 100.08 | 100.08 | 100.08 | 100.08 | 100.08 |
| – Frequency scalability (%) | 100.00 | 99.98 | 96.80 | 93.62 | 92.02 | 90.37 | 90.17 | 90.15 | 89.24 |

Table 7: Statistics about explicit tasks in parallel fraction (FG)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| Number of explicit tasks executed (total) | 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 | 0.96 | 0.63 | 0.43 | 0.32 | 0.37 | 0.32 | 0.28 | 0.26 |
| LB (time executing explicit tasks) | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.99 | 0.99 | 0.98 | 0.98 |
| Time per explicit task (average) | 447.74 | 449.78 | 466.14 | 483.42 | 492.88 | 503.46 | 506.20 | 506.20 | 516.60 |
| Overhead per explicit task (synch %) | 0.0 | 0.46 | 0.82 | 1.2 | 1.42 | 2.2 | 2.5 | 2.97 | 4.01 |
| Overhead per explicit task (sched %) | 0.06 | 0.27 | 0.57 | 0.86 | 1.14 | 1.49 | 1.86 | 2.02 | 3.03 |
| Number of taskwait/taskgroup (total) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

**Modelfactors of Leaf version:**

Table 8: Overview of whole program execution metrics (Leaf)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| **Elapsed time (sec)** | 1.62 | 1.24 | 1.25 | 1.33 | 1.33 | 1.37 | 1.37 | 1.37 | 1.37 |
| **Speedup** | 1.00 | 1.30 | 1.30 | 1.22 | 1.22 | 1.18 | 1.18 | 1.18 | 1.18 |
| **Efficiency** | 1.00 | 0.65 | 0.32 | 0.20 | 0.15 | 0.12 | 0.10 | 0.08 | 0.07 |

Table 9: Overview of efficiency metrics in the parallel fraction (Leaf)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| **Parallel fraction (%)** | 99.98 | - | - | - | - | - | - | - | - |
| **Global efficiency (%)** | 99.94 | 65.01 | 32.40 | 20.30 | 15.21 | 11.78 | 9.82 | 8.43 | 7.36 |
| **Strategy efficiency (%)** | 99.94 | 65.22 | 32.64 | 21.79 | 16.37 | 13.11 | 10.94 | 9.40 | 8.25 |
| **Load balancing (%)** | 100.00 | 65.44 | 32.76 | 21.87 | 16.43 | 13.16 | 10.98 | 9.43 | 8.28 |
| **Execution efficiency (%)** | 99.94 | 99.67 | 99.64 | 99.61 | 99.61 | 99.60 | 99.63 | 99.62 | 99.58 |
| **Scalability for tasks (%)** | 100.00 | 99.68 | 99.24 | 93.17 | 92.91 | 89.88 | 89.73 | 89.69 | 89.20 |
| **IPC scalability (%)** | 100.00 | 99.66 | 99.56 | 99.57 | 99.49 | 99.51 | 99.52 | 99.68 | 99.45 |
| **Instruction scalability (%)** | 100.00 | 100.06 | 100.07 | 100.07 | 100.06 | 100.06 | 100.06 | 100.06 | 100.06 |
| **Frequency scalability (%)** | 100.00 | 99.95 | 99.62 | 93.51 | 93.32 | 90.27 | 90.11 | 89.92 | 89.63 |

Table 10: Statistics about explicit tasks in parallel fraction (Leaf)

| **Number of processors** | **1** | **2** | **4** | **6** | **8** | **10** | **12** | **14** | **16** |
|---|---|---|---|---|---|---|---|---|---|
| **Tasks executed (total)** | 3013.0 | 3013.0 | 3013.0 | 3013.0 | 3013.0 | 3013.0 | 3013.0 | 3013.0 | 3013.0 |
| **LB (tasks executed)** | 1.0 | 0.56 | 0.73 | 0.88 | 0.88 | 0.90 | 0.80 | 0.86 | 0.75 |
| **LB (execution time)** | 1.0 | 0.50 | 0.71 | 0.75 | 0.80 | 0.76 | 0.72 | 0.73 | 0.67 |
| **Time per task (avg)** | 126.04 | 127.24 | 127.72 | 136.19 | 136.46 | 140.72 | 140.63 | 140.29 | 140.84 |
| **Overhead synch (%)** | 0.0 | 224.53 | 871.70 | 1515.48 | 2159.46 | 2807.67 | 3456.08 | 4104.98 | 4744.16 |
| **Overhead sched (%)** | 0.24 | 0.78 | 0.99 | 1.08 | 1.00 | 1.00 | 0.93 | 0.88 | 0.94 |
| **Taskwait/taskgroup (total)** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

**Modelfactors of Tree version:**

Table 11: Overview of Whole Program Execution Metrics (Tree)

| Number of Processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 |
|---|---|---|---|---|---|---|---|---|---|
| Elapsed Time (sec) | 1.63 | 0.82 | 0.43 | 0.32 | 0.25 | 0.21 | 0.19 | 0.17 | 0.15 |
| Speedup | 1.00 | 1.97 | 3.78 | 5.14 | 6.51 | 7.60 | 8.78 | 9.73 | 10.71 |
| Efficiency | 1.00 | 0.99 | 0.95 | 0.86 | 0.81 | 0.76 | 0.73 | 0.69 | 0.67 |

Table 12: Overview of Efficiency Metrics in Parallel Fraction (Tree)

| Number of Processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 |
|---|---|---|---|---|---|---|---|---|---|
| Parallel Fraction | 99.98% | | | | | | | | |
| Global Efficiency | 99.69% | 98.25% | 94.31% | 85.42% | 81.25% | 75.83% | 73.05% | 69.39% | 66.88% |
| Strategy Efficiency | 99.69% | 98.31% | 94.82% | 91.42% | 87.35% | 84.47% | 81.83% | 78.11% | 75.86% |
| Load Balancing | 100.00% | 99.41% | 98.90% | 97.66% | 91.29% | 96.45% | 91.88% | 92.64% | 91.84% |
| Execution Efficiency | 99.69% | 98.89% | 95.87% | 93.61% | 95.69% | 87.57% | 89.06% | 84.31% | 82.59% |
| Scalability for Tasks | 100.00% | 99.94% | 99.46% | 93.44% | 93.02% | 89.78% | 89.28% | 88.85% | 88.16% |
| IPC Scalability | 100.00% | 99.88% | 99.76% | 99.70% | 99.54% | 99.47% | 99.22% | 99.11% | 99.05% |
| Instruction Scalability | 100.00% | 100.15% | 100.15% | 100.14% | 100.14% | 100.14% | 100.14% | 100.14% | 100.14% |
| Frequency Scalability | 100.00% | 99.92% | 99.55% | 93.59% | 93.31% | 90.12% | 89.85% | 89.51% | 88.89% |

Table 13: Statistics About Explicit Tasks in Parallel Fraction (Tree)

| Number of Processors | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 |
|---|---|---|---|---|---|---|---|---|---|
| Tasks Executed (total) | 12050.0 | 12050.0 | 12050.0 | 12050.0 | 12050.0 | 12050.0 | 12050.0 | 12050.0 | 12050.0 |
| LB (Tasks Executed) | 1.0 | 0.78 | 0.64 | 0.54 | 0.52 | 0.60 | 0.62 | 0.55 | 0.58 |
| LB (Time Executing Tasks) | 1.0 | 0.99 | 0.96 | 0.90 | 0.87 | 0.89 | 0.81 | 0.83 | 0.88 |
| Time per Task (avg) | 103.0 | 103.26 | 103.67 | 111.49 | 110.84 | 115.67 | 118.99 | 119.63 | 120.97 |
| Overhead per Task (sync %) | 0.15 | 1.86 | 6.57 | 11.24 | 17.56 | 21.95 | 25.47 | 31.82 | 35.15 |
| Overhead per Task (sched %) | 0.26 | 0.31 | 0.36 | 0.47 | 0.67 | 0.99 | 1.67 | 2.27 | 3.29 |
| Taskwait/Taskgroup (total) | 4017.0 | 4017.0 | 4017.0 | 4017.0 | 4017.0 | 4017.0 | 4017.0 | 4017.0 | 4017.0 |