Report

Lab 4: Parallel Task Decomposition Implementation and Analysis

Josep Antoni Martínez García
Santiago Oliver Suriñach

Index

1. Iterative task decomposition.....3
1.1 Execution and Check the Correctness..... 3
1.2 Performance Analysis..... 4
1.2.1 First version.....4
1.2.2 Finer grain.....6
2.1 Execution and Check the Correctness..... 8
2.2 Performance Analysis..... 9
2.2.1 Leaf strategy..... 9
2.2.2 Tree strategy..... 11
3. Comparison.....13

1. Iterative task decomposition

1.1 Execution and Check the Correctness

Here is the output running the sequential code with 1000 iterations with:

```
Unset
sbatch submit-seq.sh mandel-seq-iter -h -o -i 10000
```

Then we need to run the parallel code and compare outputs in order to check the behavior.

The version **mandel-omp-iter_FIRST.c** has the necessary OMP instrumentation to perform the required parallelism in order to behave like the sequential code.

We can check that by comparing the output files with the ***diff*** command line tool.

```
Unset
diff mandel_histogram.out mandel_histogram_FIRST.out

diff mandel_image.jpg mandel_image_FIRST.jpg
```

Then, we repeat the same process with the **finer grain (mandel-omp-iter_FINER.c)** task decomposition strategy. See the performance results in the next section.

1.2 Performance Analysis

1.2.1 First version

Modelfactors

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	2.91	1.47	0.85	0.81	0.64	0.57	0.56	0.55	0.55
Speedup	1.00	1.97	3.41	3.59	4.52	5.12	5.23	5.34	5.32
Efficiency	1.00	0.99	0.85	0.60	0.56	0.51	0.44	0.38	0.33

Table 1: Analysis done on Fri May 3 09:06:27 AM CEST 2024, par1314

Overview of the Efficiency metrics in parallel fraction, ϕ=99.99%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	100.00%	98.67%	85.40%	59.88%	56.48%	51.25%	43.59%	38.14%	33.28%
Parallelization strategy efficiency	100.00%	98.88%	85.79%	64.05%	60.79%	56.80%	48.50%	42.46%	37.30%
Load balancing	100.00%	98.92%	85.84%	64.11%	60.88%	56.96%	48.60%	42.56%	37.41%
In execution efficiency	100.00%	99.95%	99.94%	99.92%	99.85%	99.72%	99.78%	99.76%	99.70%
Scalability for computation tasks	100.00%	99.79%	99.54%	93.48%	92.91%	90.22%	89.89%	89.83%	89.23%
IPC scalability	100.00%	99.99%	99.96%	99.97%	99.94%	99.97%	99.96%	99.96%	99.96%
Instruction scalability	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Frequency scalability	100.00%	99.80%	99.58%	93.51%	92.96%	90.24%	89.93%	89.86%	89.27%

Table 2: Analysis done on Fri May 3 09:06:27 AM CEST 2024, par1314

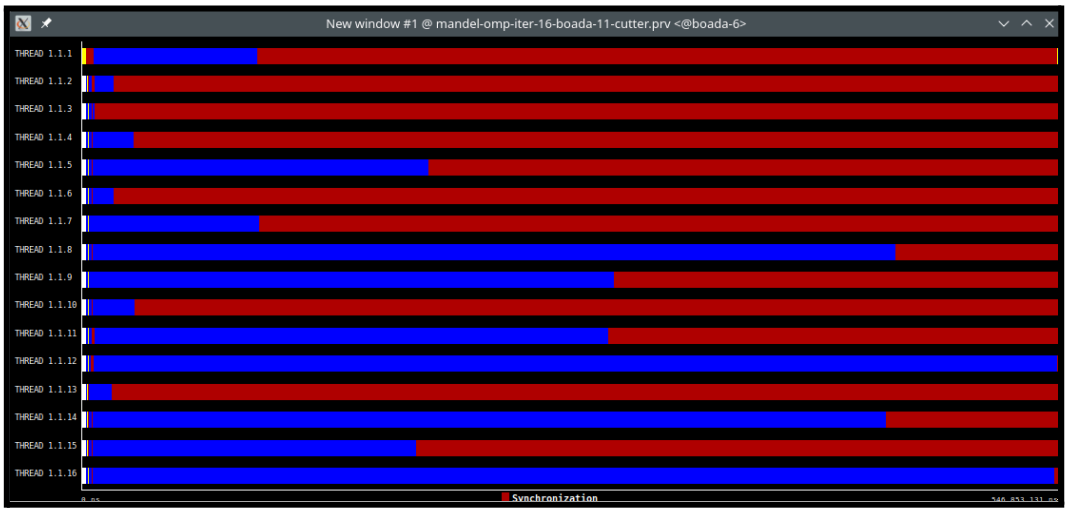
Statistics about explicit tasks in parallel fraction									
Number of processors	1	2	4	6	8	10	12	14	16
Number of explicit tasks executed (total)	64.0	64.0	64.0	64.0	64.0	64.0	64.0	64.0	64.0
LB (number of explicit tasks executed)	1.0	0.91	0.57	0.51	0.38	0.32	0.27	0.23	0.2
LB (time executing explicit tasks)	1.0	0.99	0.86	0.64	0.61	0.57	0.49	0.42	0.37
Time per explicit task (average us)	45459.26	45546.0	45653.2	48596.38	48853.68	50264.06	50459.49	50379.81	50592.97
Overhead per explicit task (synch %)	0.0	1.1	16.51	56.03	64.36	75.76	105.88	135.59	168.46
Overhead per explicit task (sched %)	0.0	0.01	0.01	0.0	0.01	0.01	0.03	0.01	0.01
Number of taskwait/taskgroup (total)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table 3: Analysis done on Fri May 3 09:06:27 AM CEST 2024, par1314

As the table shows, the **execution time** improves as the number of threads increases. We do not obtain a significant improvement in **Speedup** terms from 10 threads onwards. Besides, the overall **Speedup** is around 5 and therefore, a great improvement in performance terms. Moreover, the **efficiency** could be better when we use a large number of threads, since we start obtaining less than 50% when using more than 10 processors.

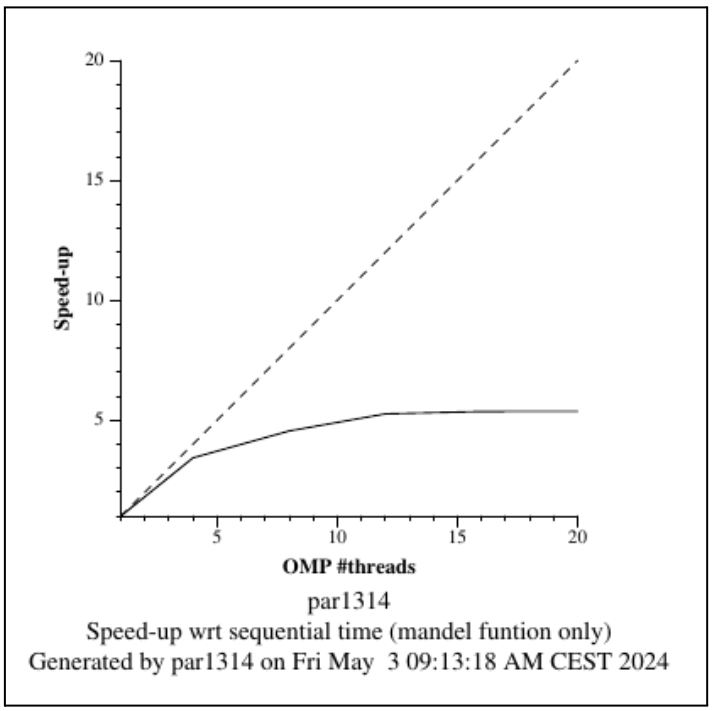
Referring to the second table, the most useful metric to comment on is the **load balancing.** As we can see, when the number of threads increases the Load balancing suffers a high penalty, affecting the efficiency of parallelization.

Paraver



As we said before, the load balancing between CPUs is so low and the Paraver analysis confirms our interpretation.

Strong scalability analysis



As the plot shows, this first version has a very bad strong scalability due to the huge difference between the resulting line and the ideal scalability (discontinuous line).

1.2.2 Finer grain

Modelfactors

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	3.04	2.24	1.14	0.81	0.61	0.51	0.43	0.37	0.33
Speedup	1.00	1.36	2.66	3.77	4.99	6.00	7.10	8.18	9.25
Efficiency	1.00	0.68	0.67	0.63	0.62	0.60	0.59	0.58	0.58

Table 1: Analysis done on Fri May 17 09:55:58 AM CEST 2024, par1310

Overview of the Efficiency metrics in parallel fraction, ϕ=99.99%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	99.90%	67.87%	66.52%	62.82%	62.33%	60.00%	59.17%	58.39%	57.86%
Parallelization strategy efficiency	99.90%	99.67%	99.12%	98.57%	98.11%	97.51%	96.33%	95.04%	94.29%
Load balancing	100.00%	99.96%	99.86%	99.53%	99.61%	99.39%	98.98%	98.95%	98.86%
In execution efficiency	99.90%	99.71%	99.26%	99.04%	98.49%	98.10%	97.33%	96.05%	95.38%
Scalability for computation tasks	100.00%	68.10%	67.10%	63.74%	63.53%	61.53%	61.43%	61.44%	61.36%
IPC scalability	100.00%	99.83%	99.75%	99.71%	99.68%	99.71%	99.67%	99.70%	99.70%
Instruction scalability	100.00%	68.21%	68.20%	68.19%	68.17%	68.16%	68.15%	68.14%	68.14%
Frequency scalability	100.00%	100.00%	98.64%	93.74%	93.49%	90.54%	90.43%	90.43%	90.33%

Table 2: Analysis done on Fri May 17 09:55:58 AM CEST 2024, par1310

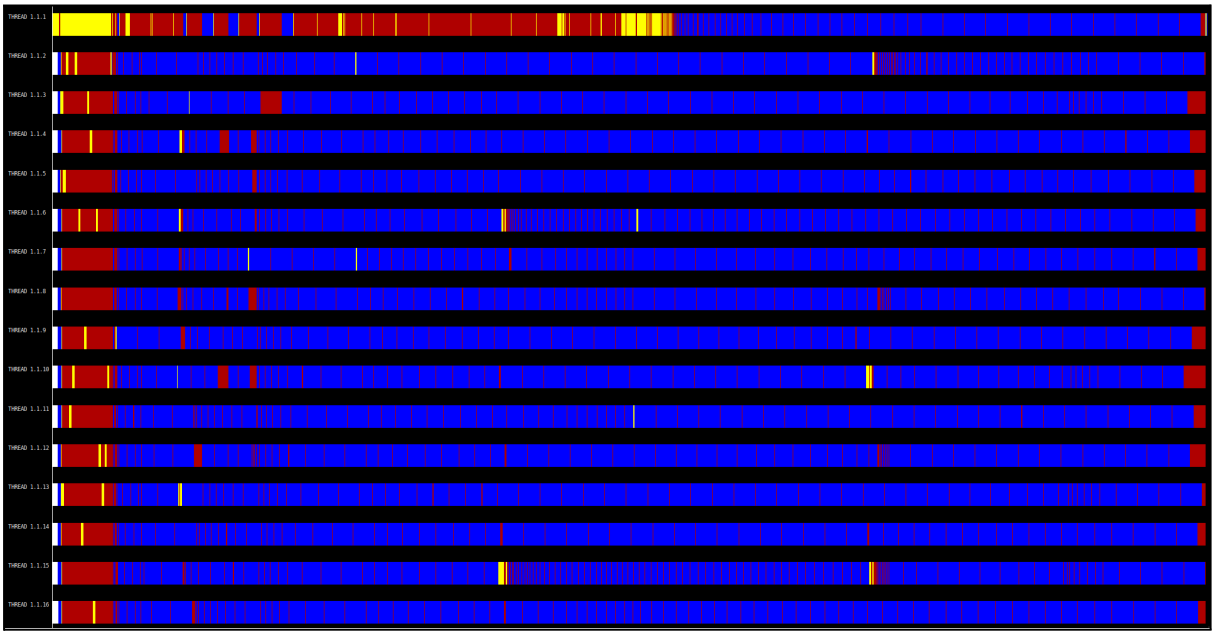
Statistics about explicit tasks in parallel fraction									
Number of processors	1	2	4	6	8	10	12	14	16
Number of explicit tasks executed (total)	9536.0	10816.0	13376.0	15936.0	18496.0	21056.0	23616.0	24640.0	24640.0
LB (number of explicit tasks executed)	1.0	0.83	0.39	0.26	0.21	0.18	0.14	0.13	0.11
LB (time executing explicit tasks)	1.0	1.0	1.0	0.99	1.0	0.99	0.99	0.99	0.99
Time per explicit task (average us)	318.17	412.82	339.3	299.84	259.24	234.71	209.52	200.66	200.77
Overhead per explicit task (synch %)	0.0	0.23	0.56	1.04	1.31	1.96	3.0	4.4	5.09
Overhead per explicit task (sched %)	0.09	0.09	0.29	0.32	0.41	0.41	0.51	0.53	0.56
Number of taskwait/taskgroup (total)	128.0	128.0	128.0	128.0	128.0	128.0	128.0	128.0	128.0

Table 3: Analysis done on Fri May 17 09:55:58 AM CEST 2024, par1310

Again, the **time to execute the whole program** decreases as the number of threads increases. The **SpeedUp** gets a significant boost, reaching a 9.25 factor when 16 threads are used. On the other hand, the **efficiency** receives a penalty when using multiple threads.

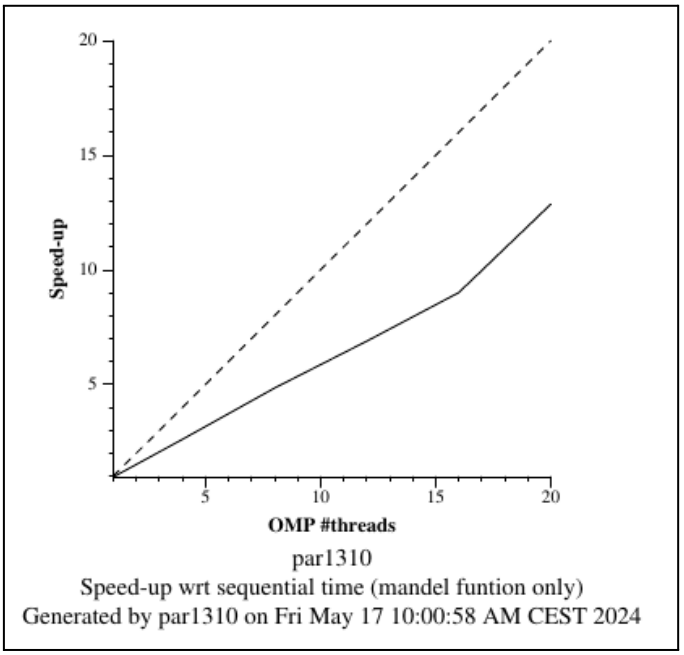
Taking a look at the second table, we can appreciate an excellent **load balancing** between threads, always getting a value close to 100%.

Paraver



As the **Load Balancing** metric showed at the previous table, the task assignment is equally distributed and is reflected in the above plot.

Strong scalability analysis



The strong scalability plot shows that our finer grain approach receives a performance little close to the ideal behavior.

2. Recursive task decomposition

2.1 Execution and Check the Correctness

Here is the output running the sequential code with 1000 iterations with:

```
Unset
sbatch submit-seq.sh mandel-seq-rec -h -o -i 10000
```

Then we need to run the parallel code and compare outputs in order to check the behavior.

The version **mandel-omp-rec_FIRST.c** has the necessary OMP instrumentation to perform the required parallelism in order to behave like the sequential code.

We can check that by comparing the output files with the ***diff*** command line tool.

```
Unset
diff mandel_histogram.out mandel_histogram_REC.out

diff mandel_image.jpg mandel_image_REC.jpg
```

We must check the two approaches (tree and leaf strategies) with this procedure.

We added to the code, apart from the parallel and single activation zone, the following protections:

- Protecting histogram[][] with atomic clauses.
- Protecting variables enabling with criticals

```
Unset

#pragma omp atomic
histogram[M[py][px]-1]++;

#pragma omp atomic
histogram[M[y][x]-1]+=(NRows*NCols);
```

```
#pragma omp critical
{
       XSetForeground (display, gc, color);
       XDrawPoint (display, win, gc, px, py);
}
```

2.2 Performance Analysis

2.2.1 Leaf strategy

Modelfactors

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	2.01	1.50	1.54	1.60	1.63	1.66	1.66	1.66	1.66
Speedup	1.00	1.34	1.31	1.25	1.23	1.21	1.21	1.21	1.21
Efficiency	1.00	0.67	0.33	0.21	0.15	0.12	0.10	0.09	0.08

Table 1: Analysis done on Thu May 23 05:45:15 PM CEST 2024, par1310

Overview of the Efficiency metrics in parallel fraction, ϕ=99.98%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	99.78%	66.96%	32.56%	20.86%	15.39%	12.08%	10.08%	8.62%	7.54%
Parallelization strategy efficiency	99.78%	67.32%	33.63%	22.55%	16.89%	13.55%	11.32%	9.72%	8.51%
Load balancing	100.00%	68.14%	34.13%	22.82%	17.07%	13.68%	11.44%	9.82%	8.61%
In execution efficiency	99.78%	98.80%	98.54%	98.82%	98.96%	99.05%	98.97%	99.02%	98.94%
Scalability for computation tasks	100.00%	99.46%	96.82%	92.52%	91.08%	89.10%	89.02%	88.67%	88.54%
IPC scalability	100.00%	99.37%	98.71%	98.59%	98.49%	98.37%	98.51%	98.32%	98.37%
Instruction scalability	100.00%	100.11%	100.24%	100.22%	100.20%	100.20%	100.20%	100.20%	100.20%
Frequency scalability	100.00%	99.98%	97.85%	93.64%	92.29%	90.40%	90.19%	90.01%	89.82%

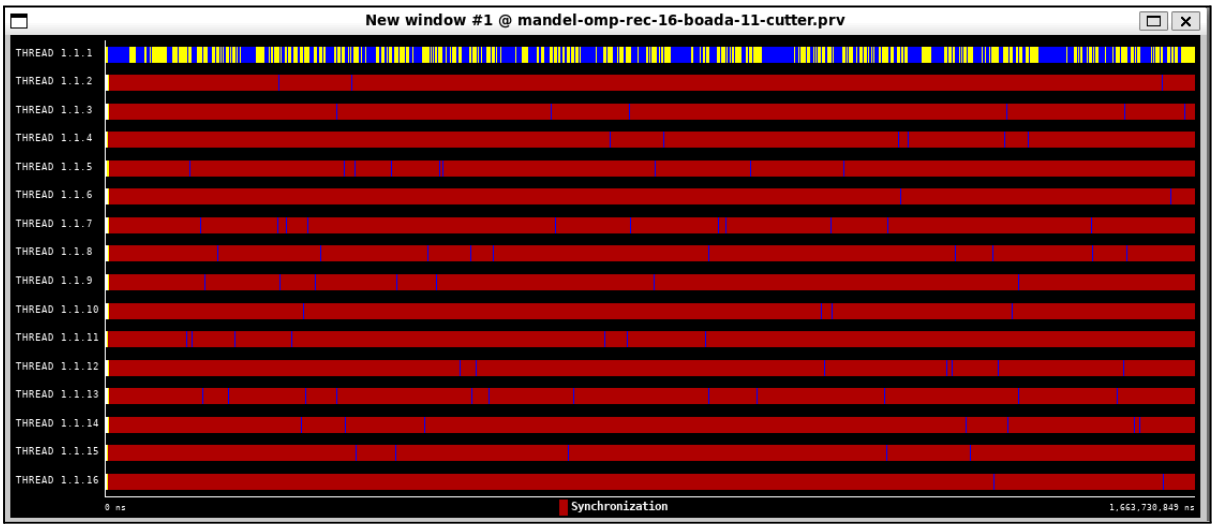
Table 2: Analysis done on Thu May 23 05:45:15 PM CEST 2024, par1310

Statistics about explicit tasks in parallel fraction									
Number of processors	1	2	4	6	8	10	12	14	16
Number of explicit tasks executed (total)	15244.0	15244.0	15244.0	15244.0	15244.0	15244.0	15244.0	15244.0	15244.0
LB (number of explicit tasks executed)	1.0	0.79	0.86	0.85	0.84	0.81	0.8	0.79	0.81
LB (time executing explicit tasks)	1.0	0.5	0.86	0.71	0.83	0.71	0.74	0.65	0.57
Time per explicit task (average us)	34.95	35.5	36.29	38.16	38.42	39.28	39.36	39.38	39.31
Overhead per explicit task (synch %)	0.0	177.58	734.88	1276.43	1845.65	2394.39	2936.85	3493.84	4055.72
Overhead per explicit task (sched %)	0.81	3.21	4.0	3.16	2.72	2.47	2.61	2.45	2.64
Number of taskwait/taskgroup (total)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 3: Analysis done on Thu May 23 05:45:15 PM CEST 2024, par1310

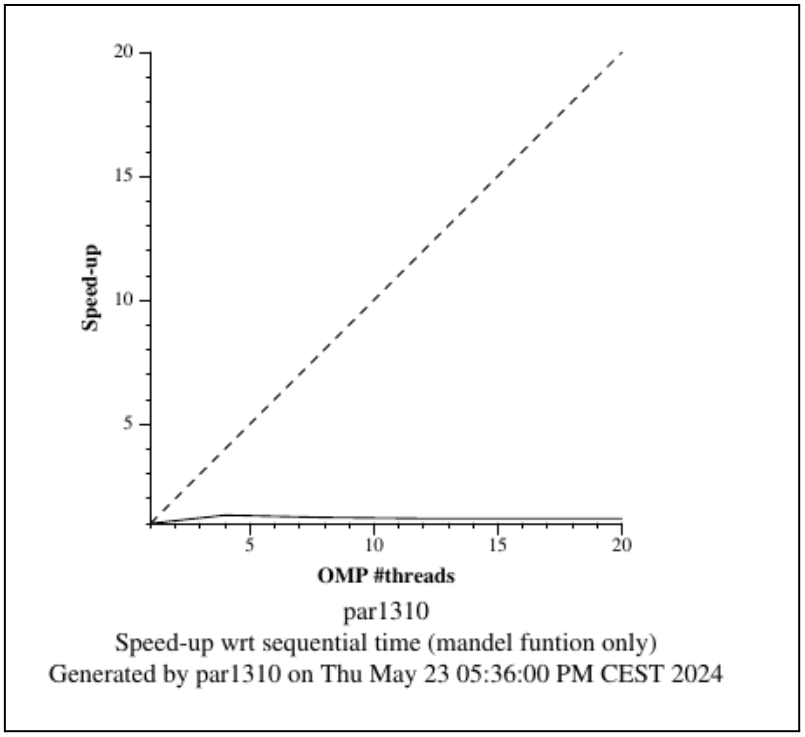
The execution time when increasing the number of threads does not significantly vary and the SpeedUp neither. We can also see a deplorable efficiency growth when adding more processors.

Paraver



As the **Load Balancing** metric showed at the previous table, the task assignment is awful.

Strong scalability analysis



As the plot reflects, the strong scalability of this approach is truly bad as the performance line differs a lot from the ideal behavior.

2.2.2 Tree strategy

Modelfactors

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	2.00	1.01	0.51	0.39	0.30	0.25	0.22	0.20	0.19
Speedup	1.00	1.97	3.88	5.17	6.74	7.84	9.01	10.17	10.67
Efficiency	1.00	0.99	0.97	0.86	0.84	0.78	0.75	0.73	0.67

Table 1: Analysis done on Thu May 23 06:32:29 PM CEST 2024, par1310

Overview of the Efficiency metrics in parallel fraction, ϕ=99.98%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	99.62%	98.26%	96.74%	85.86%	84.00%	78.21%	74.87%	72.43%	66.50%
Parallelization strategy efficiency	99.62%	98.47%	97.04%	92.21%	90.52%	87.11%	83.59%	81.27%	74.74%
Load balancing	100.00%	99.87%	97.94%	94.75%	92.49%	89.53%	85.57%	85.80%	79.16%
In execution efficiency	99.62%	98.60%	99.08%	97.32%	97.86%	97.30%	97.70%	94.72%	94.42%
Scalability for computation tasks	100.00%	99.78%	99.69%	93.11%	92.80%	89.79%	89.56%	89.13%	88.98%
IPC scalability	100.00%	99.73%	99.54%	99.24%	98.93%	99.00%	98.89%	98.73%	98.80%
Instruction scalability	100.00%	100.36%	100.36%	100.36%	100.36%	100.36%	100.36%	100.36%	100.36%
Frequency scalability	100.00%	99.69%	99.79%	93.48%	93.47%	90.37%	90.25%	89.96%	89.74%

Table 2: Analysis done on Thu May 23 06:32:29 PM CEST 2024, par1310

Statistics about explicit tasks in parallel fraction									
Number of processors	1	2	4	6	8	10	12	14	16
Number of explicit tasks executed (total)	20324.0	20324.0	20324.0	20324.0	20324.0	20324.0	20324.0	20324.0	20324.0
LB (number of explicit tasks executed)	1.0	0.77	0.6	0.45	0.52	0.55	0.5	0.46	0.64
LB (time executing explicit tasks)	1.0	1.0	0.98	0.96	0.95	0.92	0.91	0.92	0.87
Time per explicit task (average us)	97.32	98.38	98.72	106.95	108.12	112.12	115.26	117.0	120.45
Overhead per explicit task (synch %)	0.11	1.18	2.32	6.52	7.63	11.3	13.23	15.18	22.02
Overhead per explicit task (sched %)	0.27	0.34	0.62	1.54	2.29	2.62	4.62	5.63	7.69
Number of taskwait/taskgroup (total)	5081.0	5081.0	5081.0	5081.0	5081.0	5081.0	5081.0	5081.0	5081.0

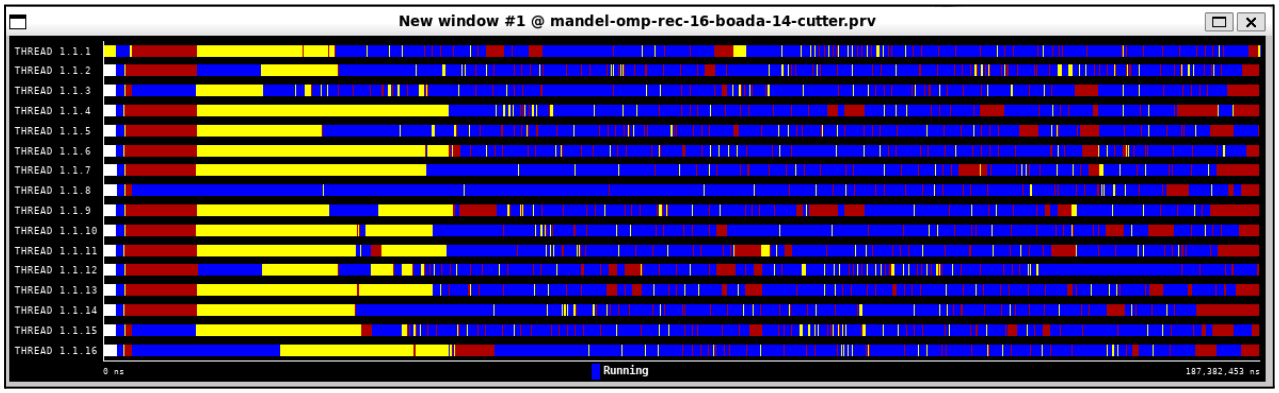
Table 3: Analysis done on Thu May 23 06:32:29 PM CEST 2024, par1310

Analyzing this data, we can see that the **elapsed time** decreases a lot when having more threads. Also, the SpeedUp is incredibly high when using more than 10 threads.

Besides, the **efficiency** receives a little penalty when using 12 or more CPUs, getting closer to 50%, a bad usage of the resources.

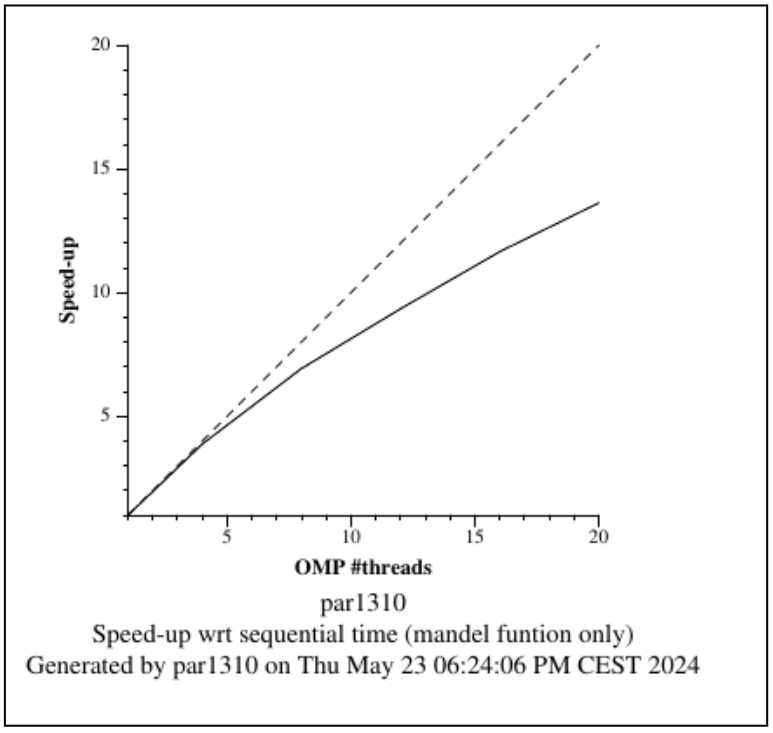
As always, we now analyze the **Load Balancing** metric, which allows us to understand how the task will be assigned and, later, confirm our hypothesis with the Paraver plots. This value is so good, even when we use 16 threads. Showing values close to 100%.

Paraver



This plot displays the load of each thread during the execution of the program. As we could see before in the *Modelfactors* tables, the load balancing metric was so good, and this graphic confirms that. Compared to the leaf strategy, this load per thread is much better. Also, it is reflected at the obtained performance time.

Strong scalability analysis



As we can see, our tree strategy task decomposition shows a really good strong scalability. Compared to the previous approach (leaf strategy), now we get a massive improvement.

3. Comparison

	Number of Threads					
Version	**1**	**4**	**8**	**12**	**16**	**20**
Iterative: Tile (provided, but should be completed)	2.914843	0.883940	0.650843	0.566101	0.558928	0.556911
Iterative: Finer grain	2.914347	0.985940	0.605845	0.426102	0.341923	0.243919
Recursive: Leaf (provided, but should be completed)	2.061126	2.241233	2.377812	2.379442	2.400214	2.432423
Recursive: Tree	2.030416	0.546223	0.314411	0.251128	0.201991	0.179156

Comparing the elapsed time, we can conclude that the finer grain (iterative) and tree strategy (recursive) are the best alternatives to its first approaches to exploit an optimal parallelism.

In other words, the finer grain task decomposition is better than the Tile and, in the Recursive case, Tree strategy is superior to Leaf.