PAR Laboratory Assignment
Lab 3: Embarrassingly parallelism with OpenMP: Mandelbrot set

Genís Bosch
Oriol Fonollà
Curs 2017/2018 Q2
Par2202

Introduction

In this session, we are going to observe and manipulate different parallelization methods or strategies on a matrix, using Rows and Points strategy. We'll also be testing different ways to parallelize a loop, using Open-MP clauses as task, taskloop and for.
To prove what's better and the advantages and the objections of each clause, we'll be testing them on the Mandelbrot algorithm to create fractals.

Parallelization strategies

We'll be using two strategies, one for rows and the other one for points:

The first one, Rows decomposition, leads us to create one task for each row, which makes the tasks bigger and can also mean that we are not having enough parallelization, but, as we don't have much tasks, the fork and join overhead is smaller.

The second one, Points decomposition, is about creating one task for each element of the matrix, so, as a result, there is no much difference between thread's execution time. Even though this stragety takes us to have a lot of tasks and therefore, a more efficient parallelism, there will be a lot of overhead scheduling those tasks.

Performance evaluation

To prove and avaluate both task and taskloop strategies, we used the submit-strong-omp script. This one executes the code three times sequentally, and then executes 12 times the code changing the number of threads, from 1 to 12, obtaining the elapsed time of all executions. Then, it creates two plots, one with the execution time over the number of threads, and another one with the speed-up over the number of threads.

For the for clause, instead of using the submit-strong-omp, we'll be using the submit-schedule-omp script, which executes the code with different schedule options, such as static, static with chunks of 10 iterations, dynamic witch chunks of 10 iterations and finally, guided with an initial chunk of 10 iterations. Then, it obtains the elapsed time of all executions, and creates two plots, this time with the execution time over the type of schedule and another one with the speed-up over the type of schedule.

Conclusions

As we have seen in this session, the best way to execute this code in parallel with 8 threads is using the task or taskloop clause and with a row strategy. Even though we've seen that, it's just in this code, so we can't believe that that's the best strategy in all codes. So, every time we have to parallelize some code, we can find different ways to do it, and we have to think about some points, such as where do we create the tasks, if we want a tree or leaf strategy, the schedule that we'll use if we use a for clause, or the strategy (row or point) that we'll take.

6.1 Task granularity analysis

1. Which are the two most important common characteristics of the task graphs generated for the two task granularities (Row and Point) for the non-graphical version of mandel-tareador? Obtain the task graphs that are generated in both cases for -w 8.

Between both mandel versions without display (Rows and points), we can see that we can reach a good parallelization of both rows and points, even though it's finer with the points case. They share the characteristic that their non-display version can be parallelized while their display version can't. They also have the same number of task regions that are bigger, because the row granularity version has 5 bigger tasks and the point granularity version has 5 groups of bigger tasks.

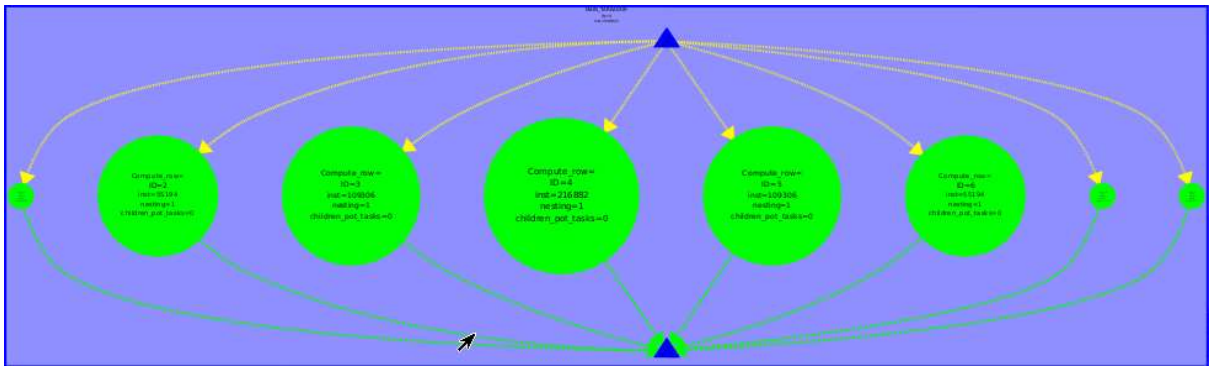


Figure 1. Mandel-tareador, non-graphical version with row granularity

Figure 2. Mandel-tareador, non-graphical version with point granularity

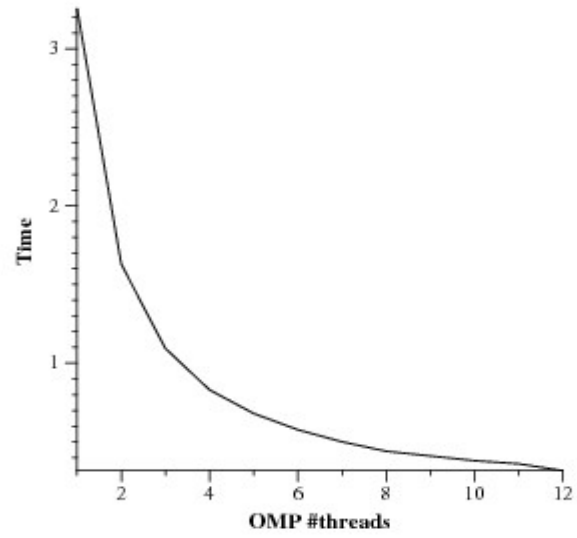
2. Which section of the code is causing the serialization of all tasks in mandeld-tareador? How do you plan to protect this section of code in the parallel OpenMP code?

As this is the display version, the section that is causing the serialization of all tasks is the #if Display section, so we should create a critical OpenMP region there.

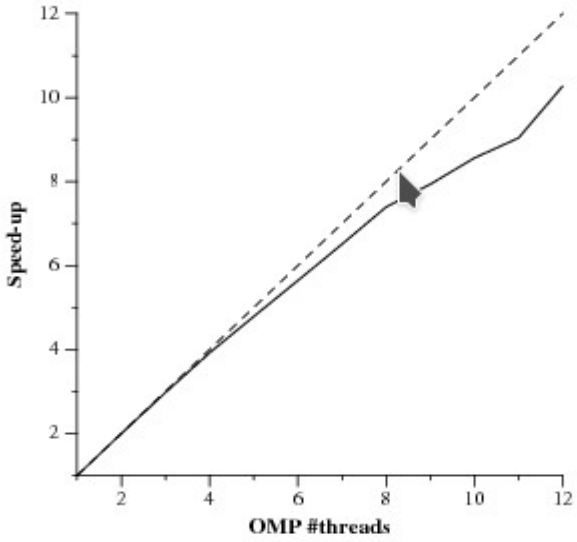
6.2 OpenMP task–based parallelization

1. For the Row and Point decompositions of the non-graphical version, include the execution time and speed–up plots obtained in the strong scalability analysis (with -i 10000). Reason about the causes of good or bad performance in each case.

Because of the critical region, as we get more threads(point version), there are more threads waiting to enter to the region, so the overhead increases. That's why we can perceive a better scalability on the row version of the program, because it has less tasks than the point version.

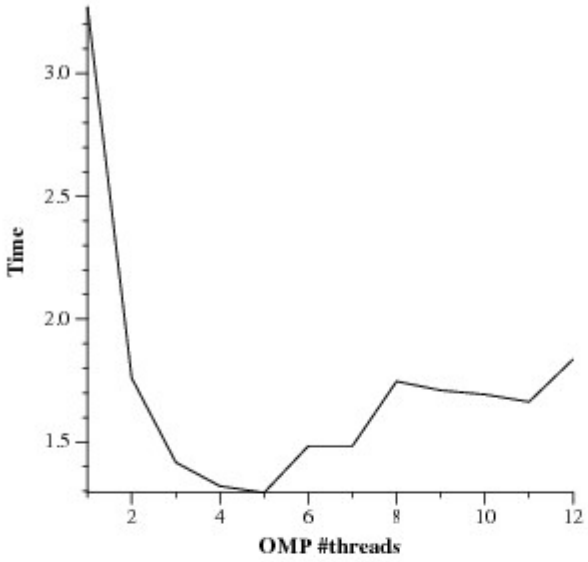


par2202
Average elapsed execution time
Wed Apr 18 11:32:03 CEST 2018

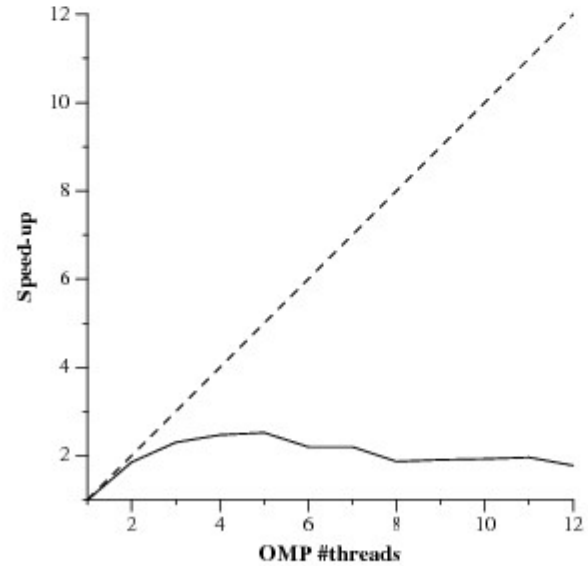


par2202
Speed-up wrt sequential time
Wed Apr 18 11:32:03 CEST 2018

Figures 3 and 4. Execution time and speed-up for the task-based row



par2202
Average elapsed execution time
Wed Apr 18 11:47:34 CEST 2018



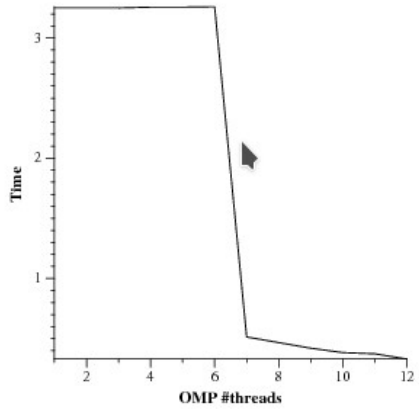
par2202
Speed-up wrt sequential time
Wed Apr 18 11:47:34 CEST 2018

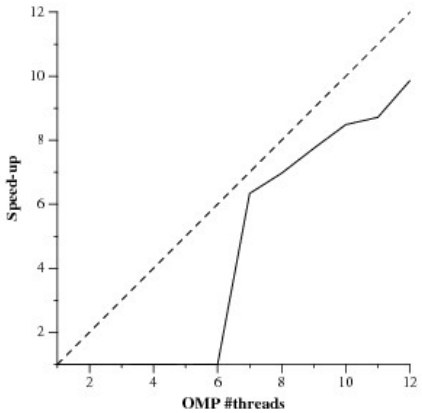
Figures 5 and 6. Execution time and speed-up for the task-based point

6.3 OpenMP taskloop–based parallelization

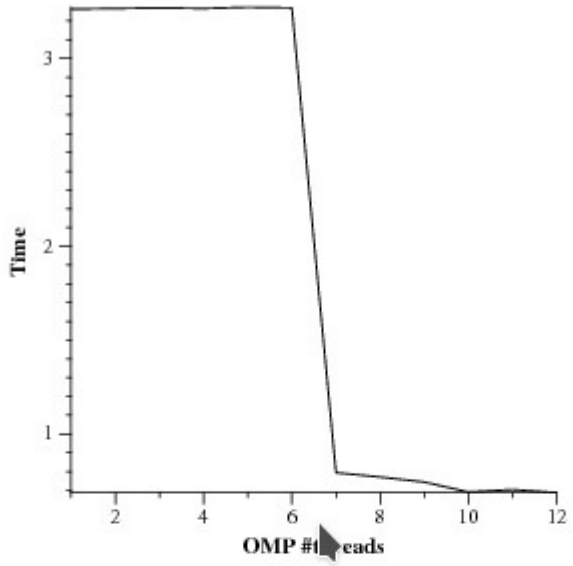
1. For the Row and Point decompositions of the non-graphical version, include the execution time and speed–up plots obtained in the strong scalability analysis (with -i 10000). Reason about the causes of good or bad performance in each case.

In light of the fact that the Point decomposition has a lot more tasks to synchronize, it produces more overheads, which leads to a worst speed-up on the occasion of using more and more threads.

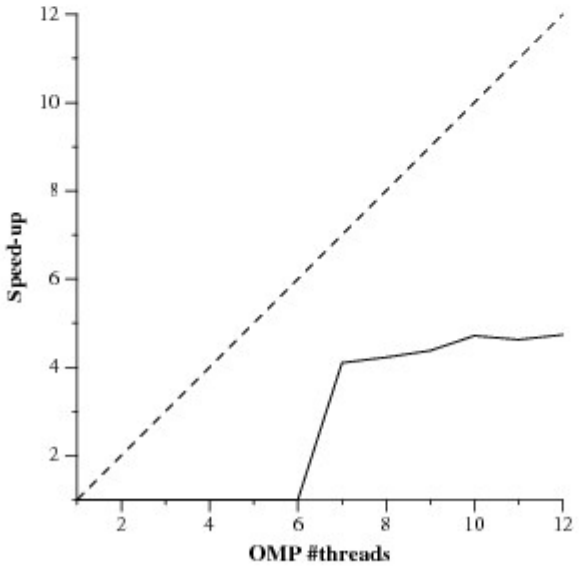




Figures 7 and 8. Execution time and speed-up for the taskloop-based row



par2202
Average elapsed execution time
Wed Apr 25 10:39:29 CEST 2018



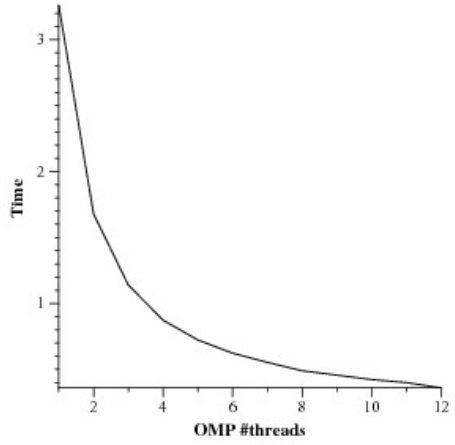
par2202
Speed-up wrt sequential time
Wed Apr 25 10:39:29 CEST 2018

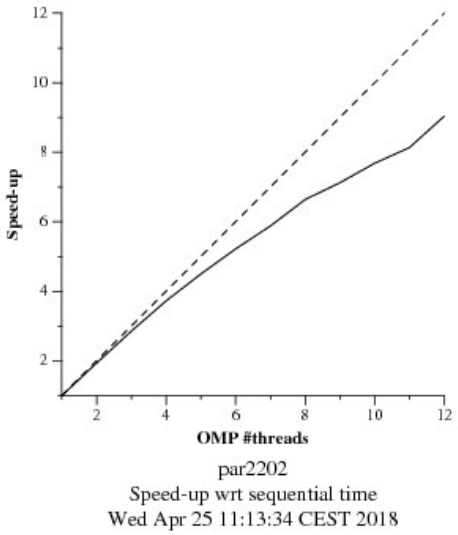
Figures 9 and 10. Execution time and speed-up for the taskloop-based point

6.4 OpenMP for–based parallelization

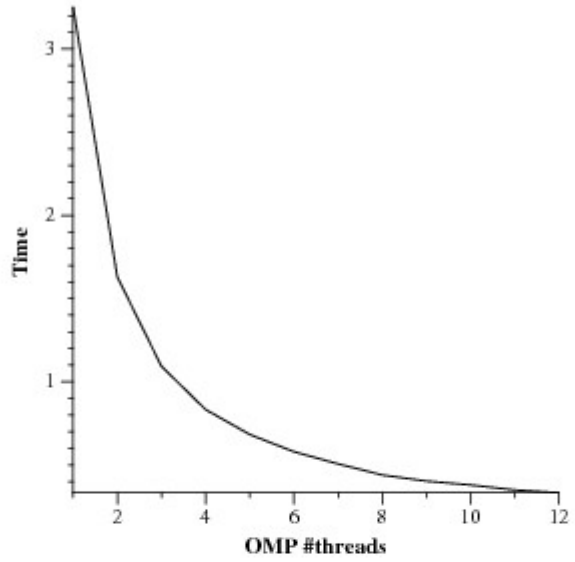
1. For the the Row and Point decompositions of the non-graphical version, include the execution time and speed–up plots that have been obtained for the 4 different loop schedules when using 8 threads (with -i 10000). Reason about the performance that is observed.

In this case, we can't almost see the difference between Rows and Points decomposition, as, even though they use a different schedule, the execution time is notablebly close. Dynamic and Guided schedulings may involve more overhead, but they solve the imbalance problems, so in the end, the overhead time is not a problem at all.

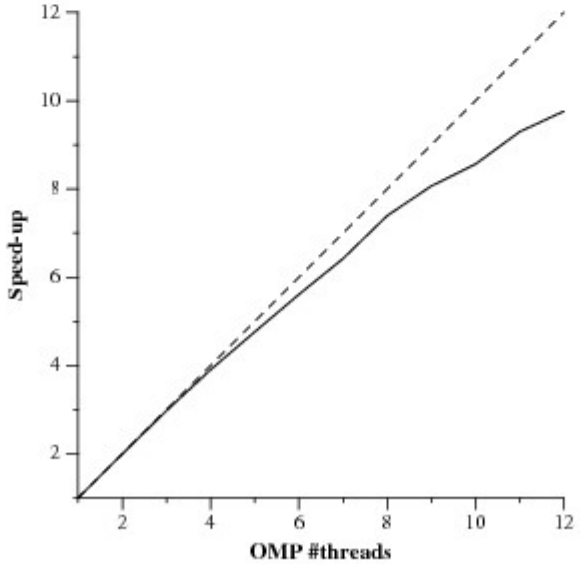




Figures 11 and 12. Execution time and speed-up for the for-based point with static schedule

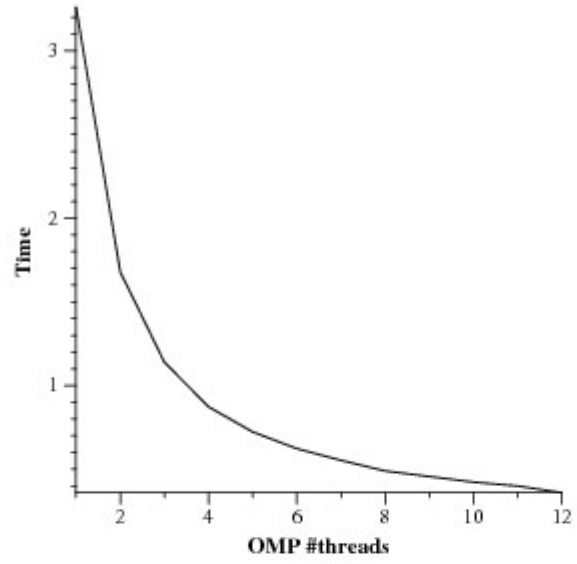


par2202
Average elapsed execution time
Wed Apr 25 11:00:18 CEST 2018

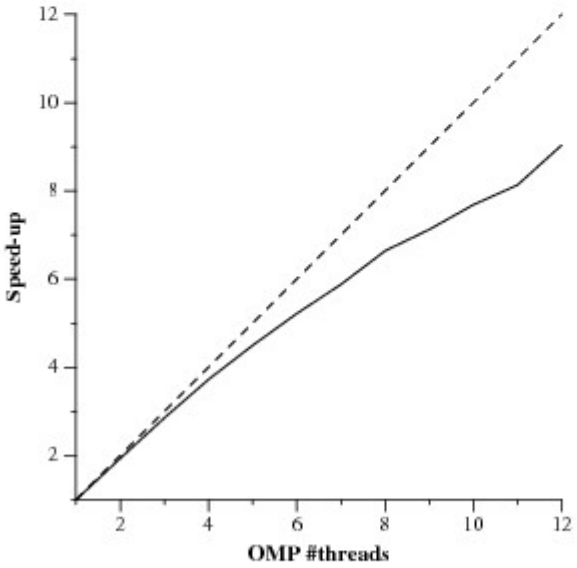


par2202
Speed-up wrt sequential time
Wed Apr 25 11:00:18 CEST 2018

Figures 13 and 14. Execution time and speed-up for the for-based row with static schedule

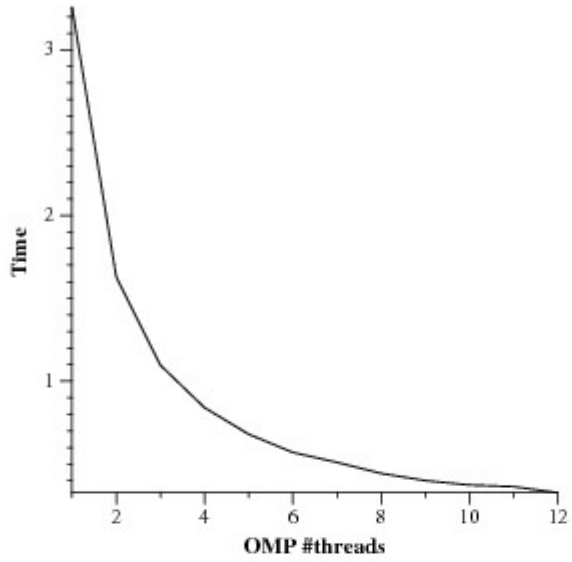


par2202
Average elapsed execution time
Wed Apr 25 11:13:34 CEST 2018

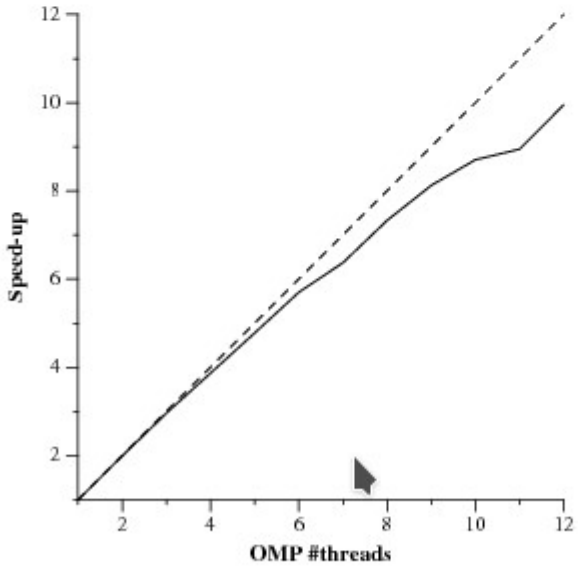


par2202
Speed-up wrt sequential time
Wed Apr 25 11:13:34 CEST 2018

Figures 15 and 16. Execution time and speed-up for the for-based point with static, 10 schedule

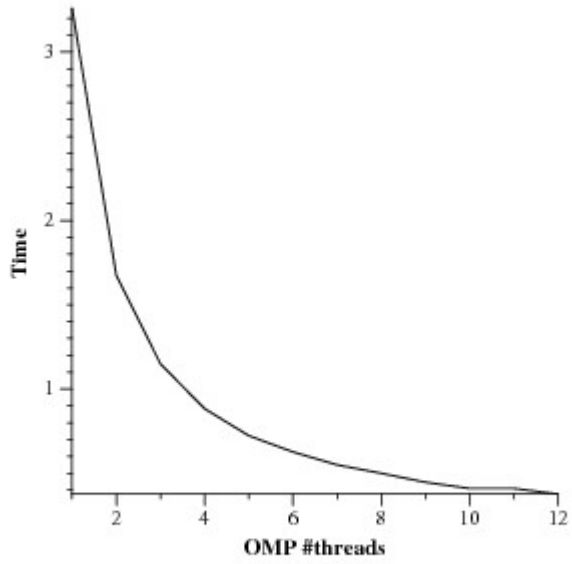


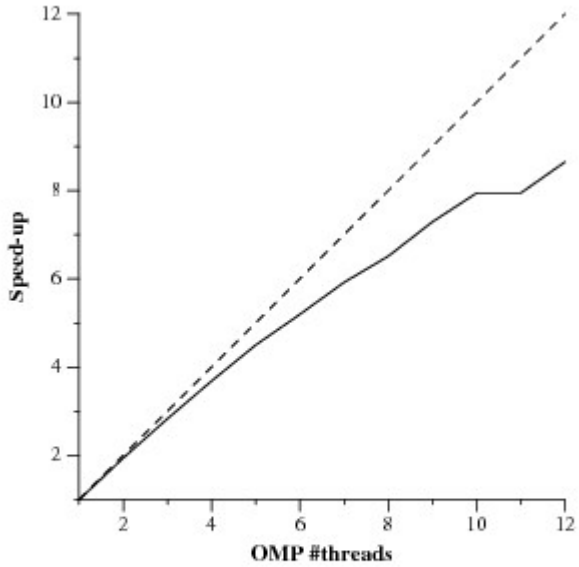
par2202
Average elapsed execution time
Wed Apr 25 11:04:02 CEST 2018



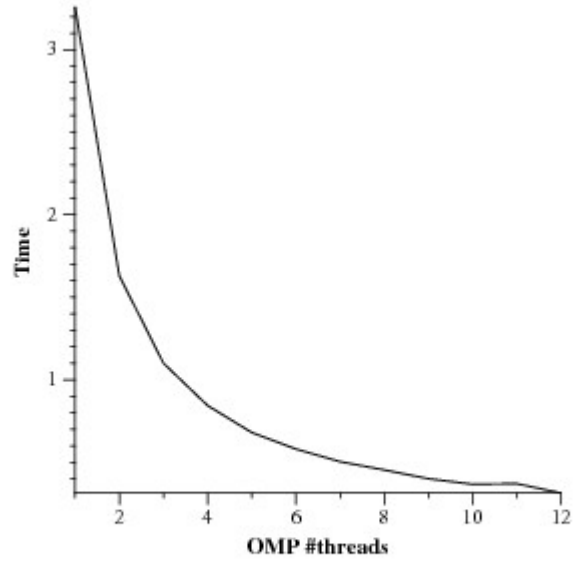
par2202
Speed-up wrt sequential time
Wed Apr 25 11:04:02 CEST 2018

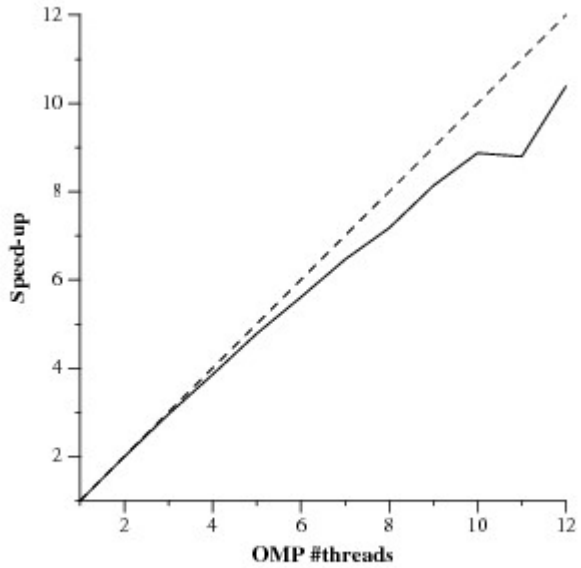
Figures 17 and 18. Execution time and speed-up for the for-based row with static schedule



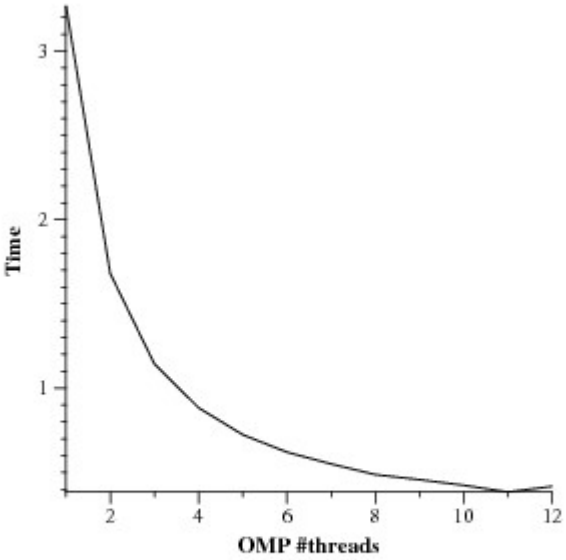


Figures 19 and 20. Execution time and speed-up for the for-based point with dynamic,10 schedule

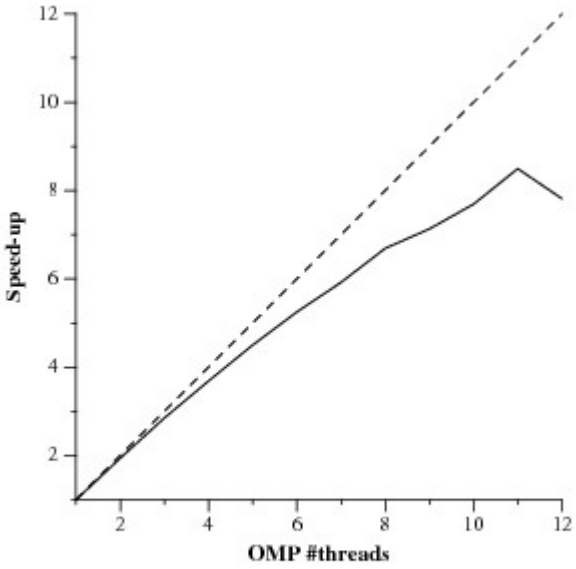




Figures 21 and 22. Execution time and speed-up for the for-based row with dynamic,10 schedule

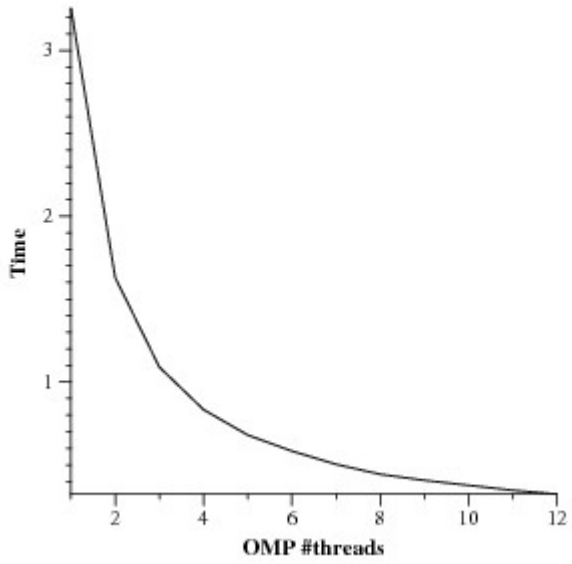


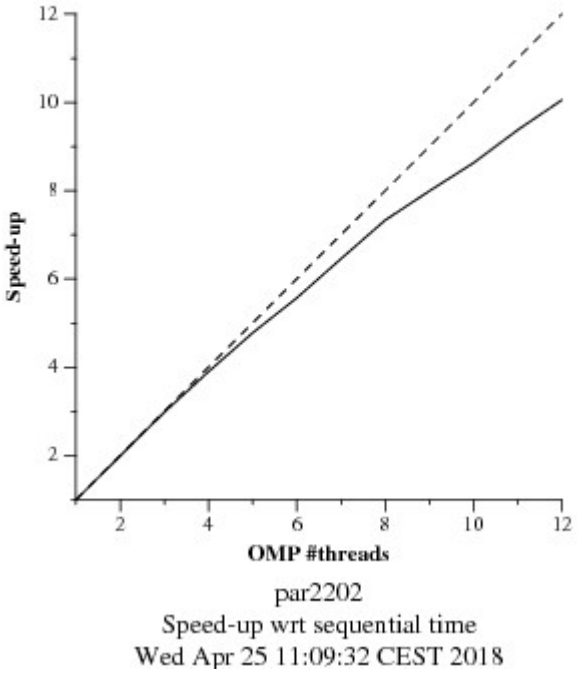
par2202
Average elapsed execution time
Wed Apr 25 11:19:09 CEST 2018



par2202
Speed-up wrt sequential time
Wed Apr 25 11:19:09 CEST 2018

Figures 23 and 24. Execution time and speed-up for the for-based point with guided,10 schedule

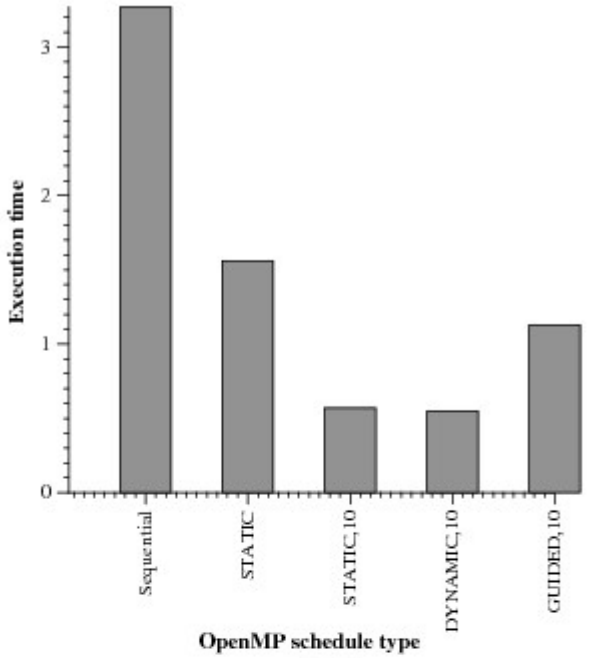




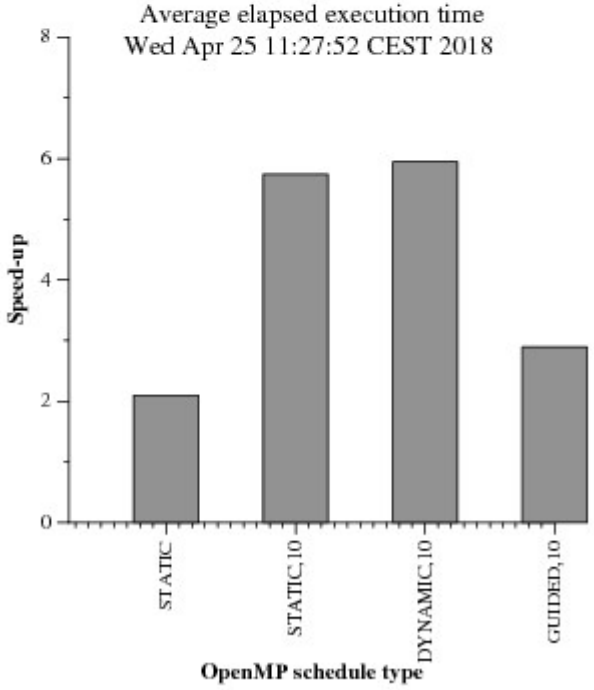
Figures 25 and 26. Execution time and speed-up for the for-based row with guided,10 schedule

2. For the Row parallelization strategy, complete the following table with the information extracted from the Extrae instrumented executions (with 8 threads and -i 10000) and analysis with Paraver, reasoning about the results that are obtained.

	Static	Static 10	Dynamic 10	Guided 10
Running average time per thread	437,414ms	468,726ms	490,881ms	454,208ms
Execution unbalance	0,32	0,68	0,70	0,40
SchedForkJoin	974,377ms	37,676ms	27,762ms	505,612ms

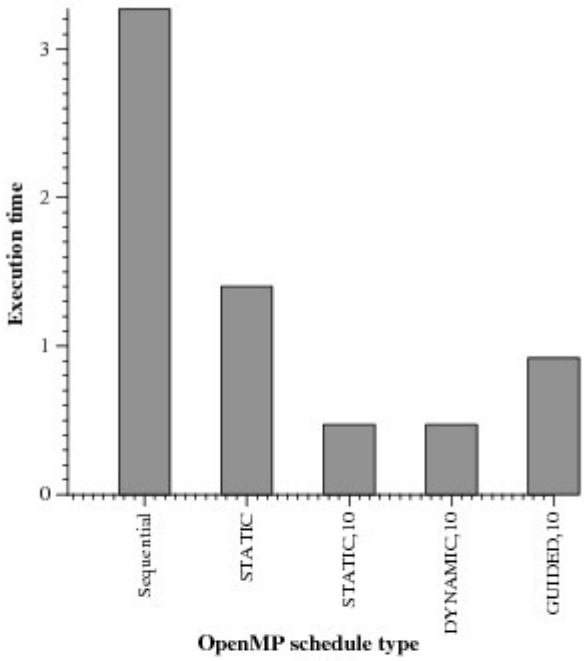


par2202
Average elapsed execution time
Wed Apr 25 11:27:52 CEST 2018

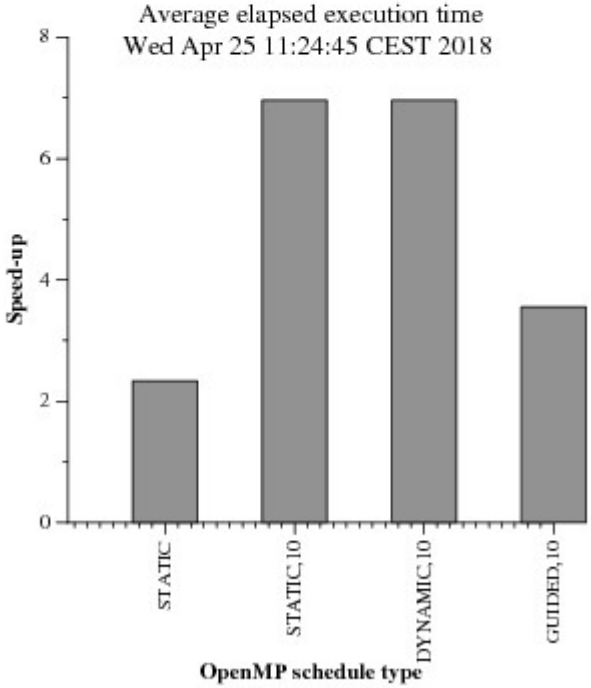


par2202
Speed-up wrt sequential execution time
Wed Apr 25 11:27:52 CEST 2018

Figures 27 and 28. Differences between schedule types in point decomposition



par2202
Average elapsed execution time
Wed Apr 25 11:24:45 CEST 2018



par2202
Speed-up wrt sequential execution time
Wed Apr 25 11:24:45 CEST 2018

Figures 29 and 30. Differences between schedule types in row decomposition