Paral·lelisme

Lab 1

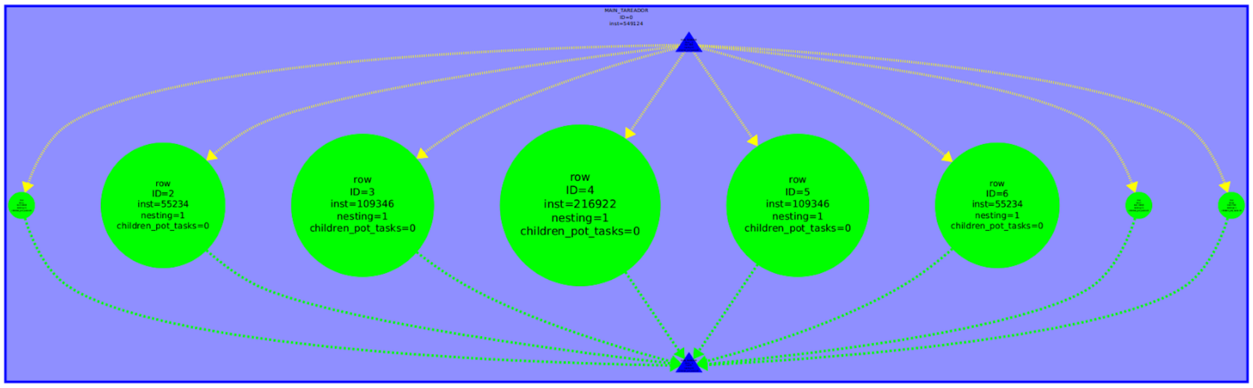
Gerard Bayego
Martín Dans

Grup: par3101
Curs: 2014-15
Fall semester

Question 1.1:

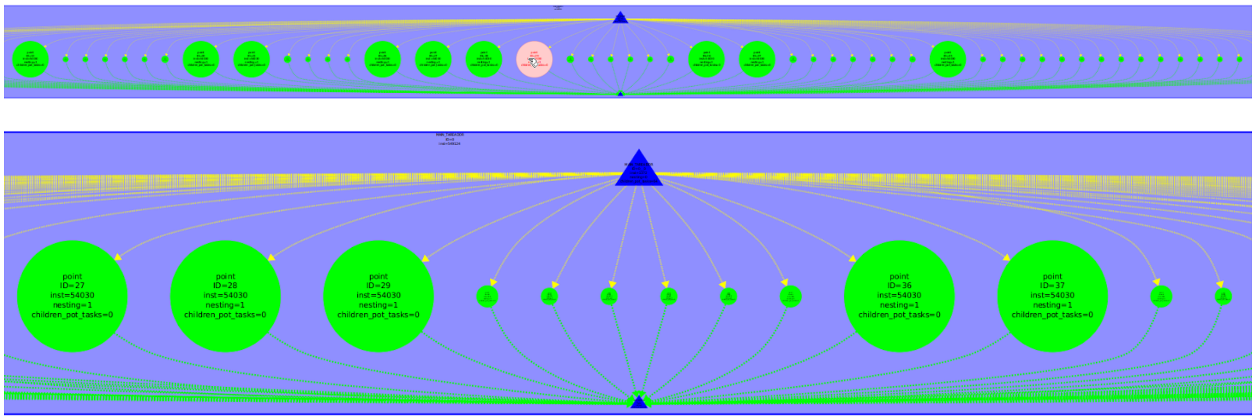
Both of them can be parallelized and they will have more or less the same execution time with 1 and 2 processors. Both of them also have big tasks that will limit the parallelization and a lot of small tasks.

Row graph:



In the case of row parallelization, we have one big task, 2 medium tasks that joined are like first one, 2 medium-small tasks that joined are like one of the second type and 3 small tasks.

Point graph:



In the case of point parallelization, we have 10 equal big tasks and a lot of small tasks.

Question 1.2:

In the mandeld-tareador, the execution is sequential because display, gc and win variables are dependences. To solve it we can protect this region with a critical pragma OMP.

Question 1.3:

	Row		Point	
Num. processors	Exec. time	Speed-up	Exec. time	Speed-up
1	549124	-	549124	-
2	275029	1.997	275691	1.992
4	217030	2.530	163725	3.354
8	216980	2.531	108816	5.046
16	216984	2.531	55026	9.979

(all the times in microseconds)

With the point tasks parallelization, once we have reached the number of big tasks (tasks with more or less 54000 instructions) the parallelization decays a lot since the small tasks practically do nothing.
In the case of row tasks there are 5 big tasks, having in the middle the biggest that is the double of the tasks next to them and this tasks are respectively the double of their adjacent task. So this means, we need 3 processors to cover all of them because we need to prevent the processor that does the big task from doing other big tasks.

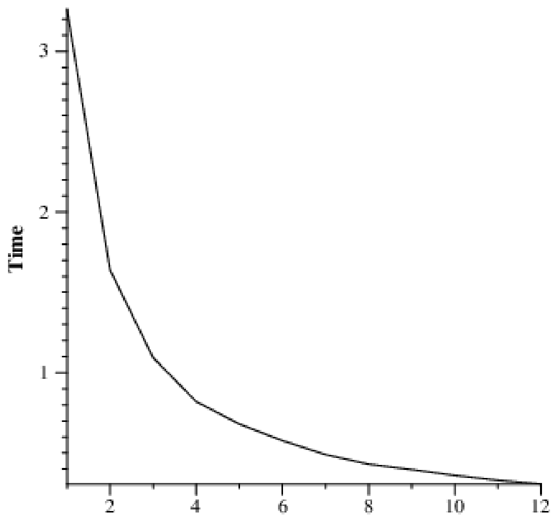
Question 2.1:

1. **mandel-omp-task-row.c**: We have added "#pragma omp parallel; #pragma omp single" to mandel-omp.c at the beginning of the function because we want to parallelize this function but only one processor has to create the tasks. Also we added " #pragma omp task private(col) firstprivate(row)" after the first for so what we get is a task for each row.

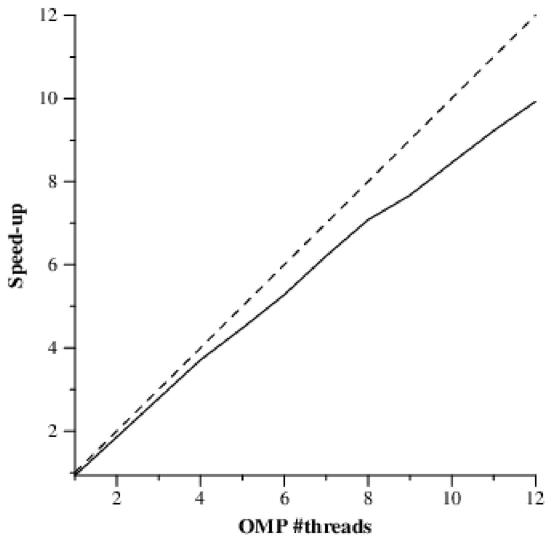
2. **mandel-omp-task-point.c:** Like the previous exercise we have added "#pragma omp parallel; #pragma omp single" to mandel-omp.c at the beginning of the function. But instead of setting the pragma, in this case the pragma is "#pragma omp task firstprivate(row,col)", after the first for we have to add it after the second and set the variable col this time also firstprivate instead of private.

Question 2.2:

1. mandel-omp-task-row.c:

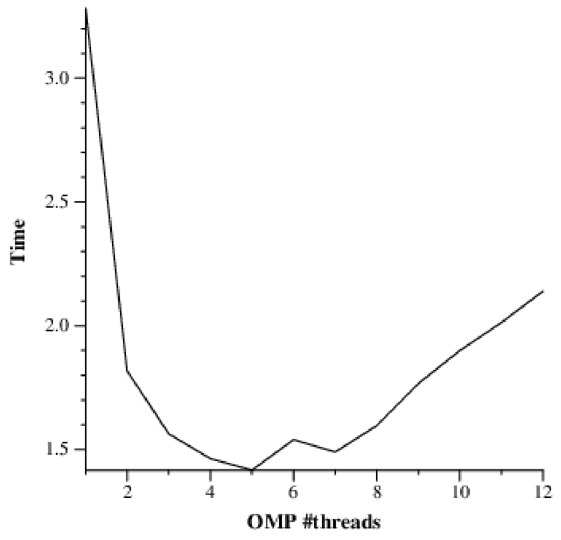


Average elapsed execution time

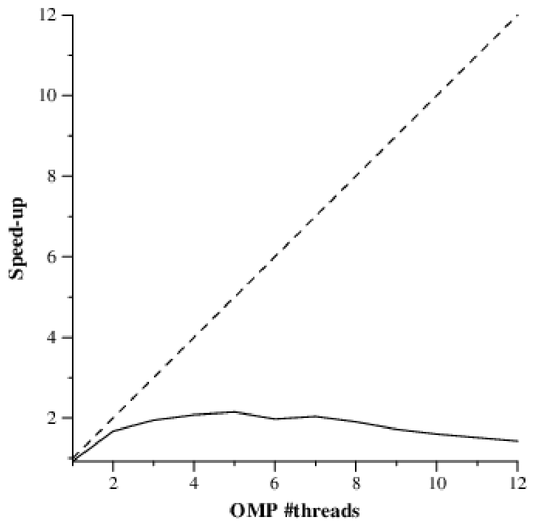


Speed-up wrt sequential time

2. mandel-omp-task-point:



Average elapsed execution time



Speed-up wrt sequential time

We don't understand with this four plots how the speed-up can be almost linear in row task, because as we mentioned before with the row task after the third processor we wouldn't expect a big improve of it and why the speed-up in point task does not increase more, since we don't think that the overhead is that much. In fact we were expecting the plots swapped.

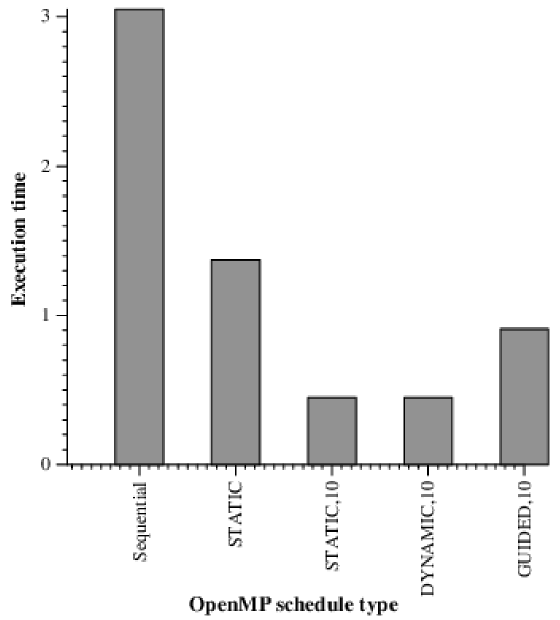
Question 3.1:

1. **mandel-omp-for-row.c:** We have added before the first for the pragma "#pragma omp for schedule(runtime)" apart from parallel "#pragma omp parallel private(row,col)" at the beginning of the function.

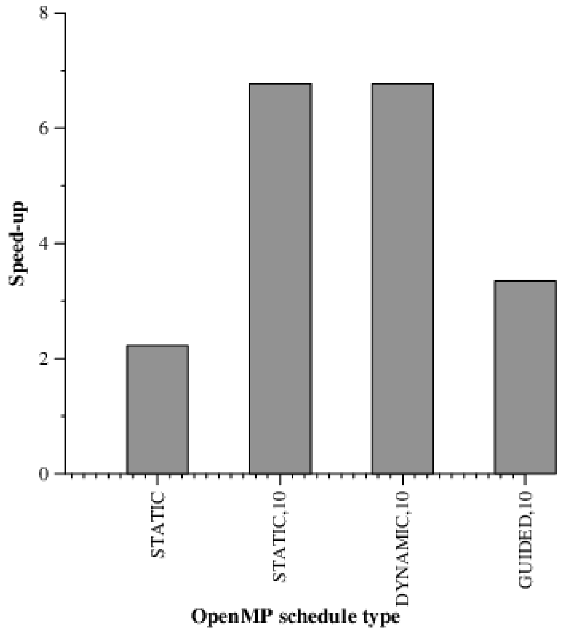
2. **mandel-omp-for-point.c:** We have added before the second for the pragma "#pragma omp for schedule(runtime) nowait" apart from parallel "#pragma omp parallel private(row, col)" at the beginning of the function.

Question 3.2:

1. mandel-omp-for-row.c



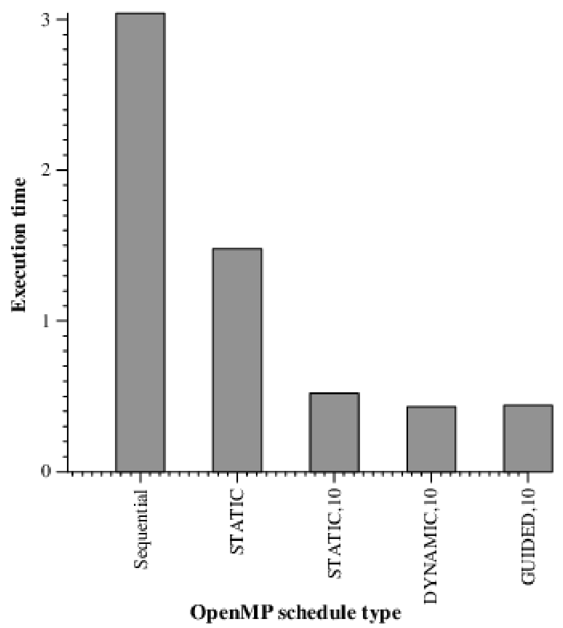
Average elapsed execution time



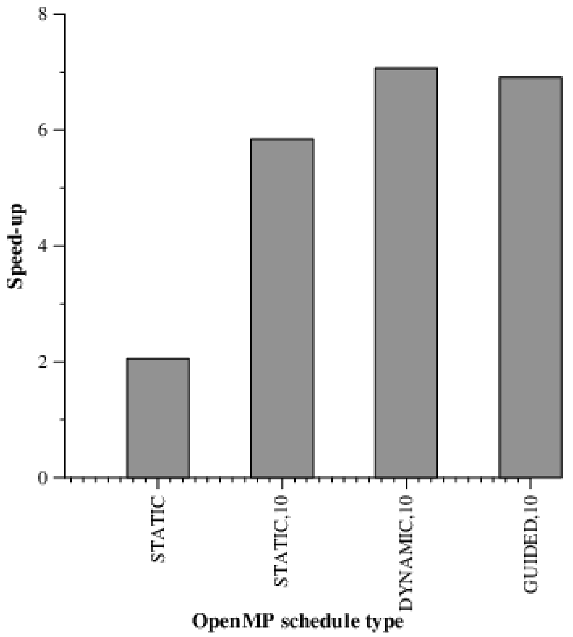
Speed-up wrt sequential execution time

Comparing the STATIC we can see that with ceiling 10 it works better. That’s because it has less overhead and with this program the big tasks go to different threads. DYNAMIC works fine but GUIDED does not go really well. This must be because the GUIDED is giving a lot of smalls task to one processor and then when it tries to gives smaller tasks to finish it still has one big task.

2. mandel-omp-for-point.c



Average elapsed execution time



Speed-up wrt sequential execution time

As the previous exercise the STATIC 10 is faster than the STATIC. DYNAMIC stills performing very well. The difference is that GUIDED this time also executes the program almost as fast as DYNAMIC.

Question 3.3:

	static	static,10	dynamic,10	guided, 10
Average time/thread	417293	437237	440551	429281
Execution unbalance	0,3	0,94	0,93	0,47
SchedForkJoin	173074	1025	185	112447

(all times in microseconds)

Like before the STATIC 10 is better that the simple STATIC. Another time GUIDED has grouped smalls tasks and has given this tasks at beginning to one thread so it’s performance decays a lot. Between STATIC 10 and

DYNAMIC,we can see that the average time per thread are similar and also the execution unbalance. The different is SchedForkJoin that takes more time in static than dynamic. This is an strange case because normally DYNAMIC has more overhead, but as we are doing row parallelism we have few chunks and the STATIC 10 has to divide all in the beginning of the execution while the DYNAMIC does it in runtime.