PAR Laboratory Assignment
Lab 4: Divide and Conquer parallelism with OpenMP: Sorting

Rubén Dabrio (boada: 4309)
Sergi Campuzano (boada: 4305)

April - May 2023

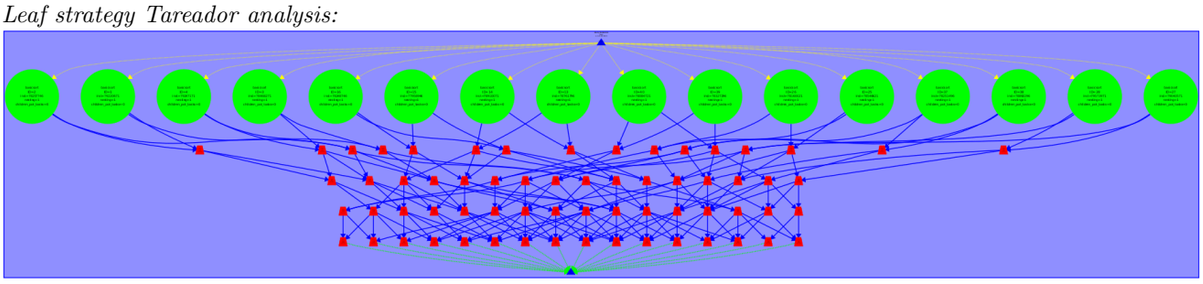
1 Laboratory 4 notebook

1.1 Task decomposition analysis with Tareador

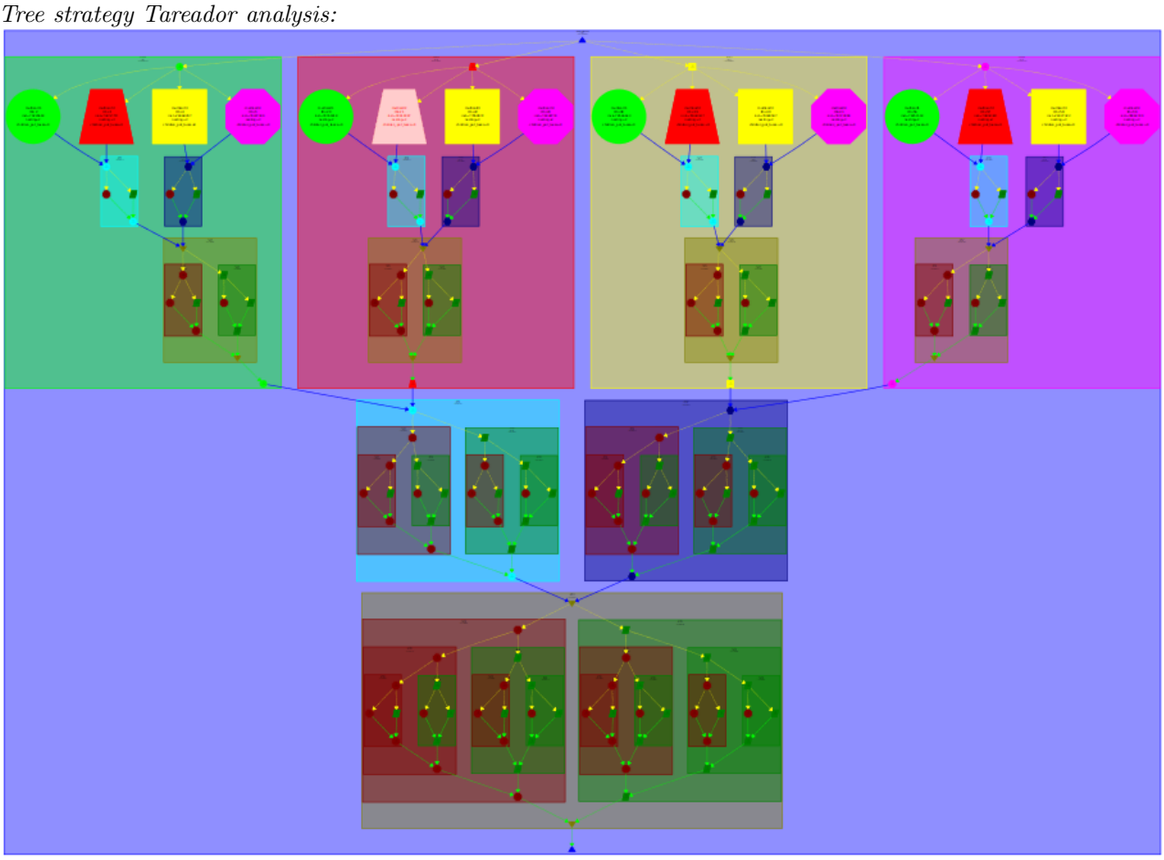
1.1.1 Leaf and Tree Analysis with Tareador

Include the TDG of both *leaf* and *tree* strategies *Tareador* analysis.

Leaf strategy Tareador analysis:



Tree strategy Tareador analysis:



Comments/Observations

Identify the points of the code where you should include synchronizations to guarantee the dependences fore each task decomposition strategy. Which directives/annotations/clauses will you use to guarantee those dependences in the OpenMP implementations?

When we use the Leaf Strategy we will have to put taskwaits. As we want to wait for tasks executed at the same recursion level(so this level is where we have the leafs), we will have

to put the taskwait after the call of the basicmerge and basicsort. Here we can see an exemple:
pragma omp task
basicmerge(n, left, right, result, start, length);
pragma omp taskwait

In the other case, when we have the Tree Strategy, we will need a taskgroup as each task have to wait for all of the internal tasks that has been created. The taskgroup will be put before the call of the merge and multisort functions as we want each of them to be part of the taskgroup. Here we can see an example:
pragma omp taskgroup
pragma omp task
merge(n/4L, data[0], data[n/4L], tmp[0], 0, n/2L);
pragma omp task
*merge(n/4L, data[n/2L], data[3L*n/4L], tmp[n/2L], 0, n/2L);*

1.2 Leaf and Tree strategies in OpenMP

1.2.1 Analysis with `modelfactors`

Include the three tables generated for the *leaf* and *tree* strategies.

Leaf strategy:

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	0.28	0.44	0.42	0.45	0.43	0.46	0.45	0.45	0.46
Speedup	1.00	0.65	0.67	0.63	0.66	0.62	0.64	0.63	0.62
Efficiency	1.00	0.32	0.17	0.11	0.08	0.06	0.05	0.04	0.04

Table 1: Analysis done on Thu May 4 11:19:32 AM CEST 2023, par4305

Overview of the Efficiency metrics in parallel fraction, ϕ=91.81%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	74.70%	23.46%	12.18%	7.65%	6.00%	4.52%	3.86%	3.26%	2.80%
Parallelization strategy efficiency	74.70%	30.82%	15.51%	10.32%	7.83%	6.28%	5.24%	4.50%	3.90%
Load balancing	100.00%	70.25%	62.11%	41.07%	28.59%	22.03%	18.62%	14.48%	12.65%
In execution efficiency	74.70%	43.87%	24.97%	25.14%	27.39%	28.48%	28.15%	31.07%	30.84%
Scalability for computation tasks	100.00%	76.13%	78.52%	74.10%	76.59%	71.97%	73.66%	72.50%	71.79%
IPC scalability	100.00%	68.22%	69.73%	69.71%	71.41%	69.19%	71.26%	70.11%	69.59%
Instruction scalability	100.00%	112.68%	112.54%	112.49%	112.88%	112.73%	112.71%	112.48%	112.75%
Frequency scalability	100.00%	99.03%	100.07%	94.50%	95.02%	92.27%	91.72%	91.94%	91.50%

Table 2: Analysis done on Thu May 4 11:19:32 AM CEST 2023, par4305

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)	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0
LB (number of explicit tasks executed)	1.0	0.5	0.9	0.77	0.44	0.35	0.29	0.21	0.19
LB (time executing explicit tasks)	1.0	0.5	0.87	0.83	0.5	0.4	0.32	0.23	0.21
Time per explicit task (average us)	2.76	3.45	3.43	3.64	3.59	3.72	3.71	3.71	3.7
Overhead per explicit task (synch %)	15.24	292.79	723.53	1163.26	1552.33	2027.82	2408.87	2881.2	3388.03
Overhead per explicit task (sched %)	9.04	20.54	17.28	16.22	15.55	15.57	16.02	15.12	14.92
Number of taskwait/taskgroup (total)	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0	53248.0

Table 3: Analysis done on Thu May 4 11:19:32 AM CEST 2023, par4305

Tree strategy:

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	0.26	0.30	0.25	0.23	0.22	0.22	0.22	0.22	0.22
Speedup	1.00	0.87	1.06	1.16	1.18	1.19	1.19	1.18	1.17
Efficiency	1.00	0.44	0.26	0.19	0.15	0.12	0.10	0.08	0.07

Table 1: Analysis done on Thu May 11 11:22:34 AM CEST 2023, par4305

Overview of the Efficiency metrics in parallel fraction, ϕ=91.38%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	88.85%	38.25%	23.67%	17.52%	13.49%	10.92%	9.02%	7.71%	6.70%
Parallelization strategy efficiency	88.85%	47.43%	34.46%	26.15%	20.22%	17.09%	14.27%	11.93%	10.20%
Load balancing	100.00%	93.63%	96.63%	94.46%	93.56%	91.51%	93.97%	89.84%	91.32%
In execution efficiency	88.85%	50.66%	35.67%	27.68%	21.61%	18.68%	15.18%	13.28%	11.16%
Scalability for computation tasks	100.00%	80.64%	68.69%	67.00%	66.73%	63.90%	63.21%	64.60%	65.67%
IPC scalability	100.00%	67.10%	57.39%	58.52%	58.41%	57.57%	57.49%	59.09%	60.17%
Instruction scalability	100.00%	121.95%	121.84%	121.97%	121.85%	122.06%	121.88%	121.90%	121.86%
Frequency scalability	100.00%	98.55%	98.23%	93.86%	93.75%	90.93%	90.20%	89.67%	89.56%

Table 2: Analysis done on Thu May 11 11:22:34 AM CEST 2023, par4305

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)	99669.0	99669.0	99669.0	99669.0	99669.0	99669.0	99669.0	99669.0	99669.0
LB (number of explicit tasks executed)	1.0	0.97	0.95	0.95	0.98	0.97	0.97	0.98	0.96
LB (time executing explicit tasks)	1.0	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.98
Time per explicit task (average us)	1.83	3.84	5.62	7.17	8.93	10.85	12.99	14.94	16.98
Overhead per explicit task (synch %)	1.0	44.39	59.08	68.61	75.71	78.95	80.83	84.04	86.43
Overhead per explicit task (sched %)	13.7	32.47	46.67	57.66	66.49	71.36	76.29	80.19	83.23
Number of taskwait/taskgroup (total)	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0

Table 3: Analysis done on Thu May 11 11:22:34 AM CEST 2023, par4305

Comments/Observations

Which of the factors do you think is making the parallelisation efficiency so low? Several options to think about: parallel fraction, in-execution efficiency (related with the overheads of sync and sched reported in the third table), number of tasks and their execution time, load balancing, ...

In the leaf case we think that what is making the efficiency so low is that load balancing which is too low when we do the execution with more than 4 processors. Also the overheads per explicit task of synch are too big and penalise a lot the execution time. In the tree strategy we can see that the number of explicit tasks executed is bigger than in the leaf strategy and it can penalise the execution time. For the part of parallel fraction we cannot see any problem as we consider that a parallel fraction of 91 percent is so good.

1.2.2 Analysis with `Paraver`

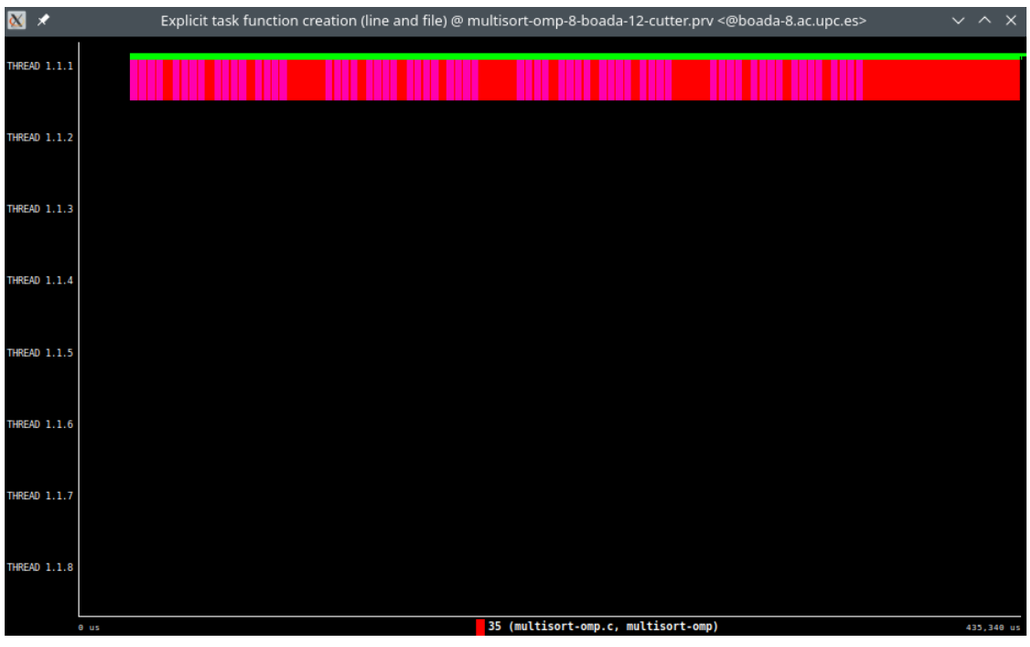
Include parts (zoom in) of the Paraver visualisations that help you explain the lack of scalability. In particular, we think it would be good to show those parts that show examples of: 1) the amount of task executed in parallel, 2) which threads are executing tasks and 3) which thread/s is/are creating tasks?.

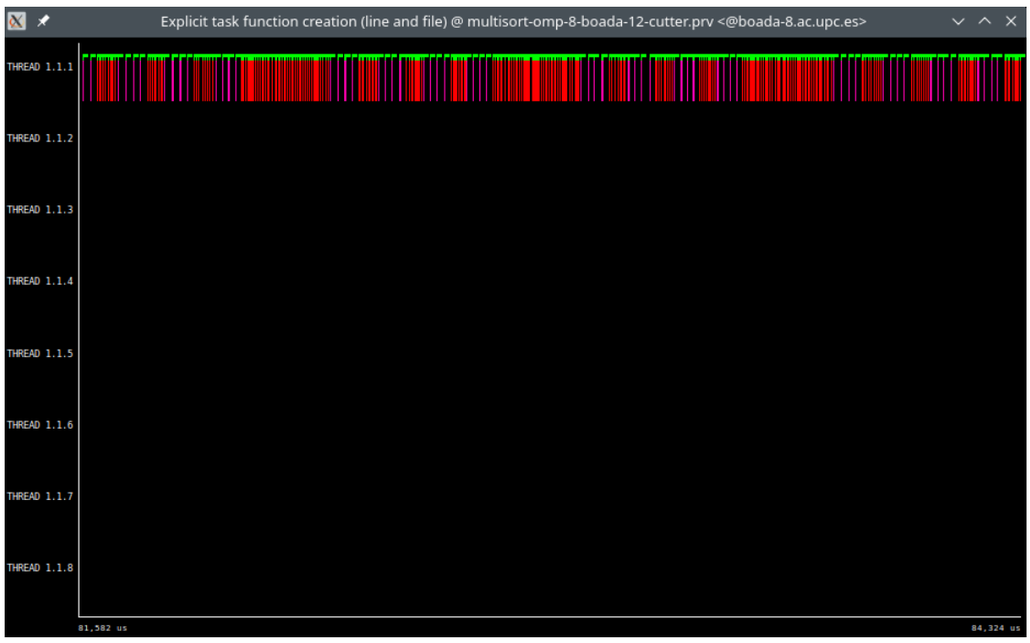
Comments/Observations

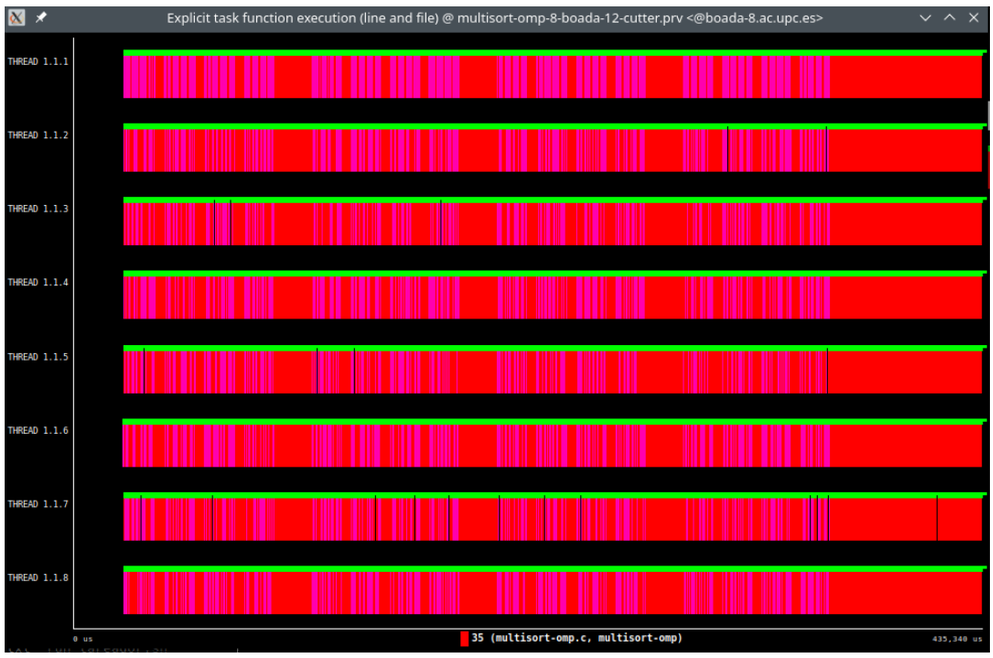
Is the program generating enough tasks to simultaneously feed all threads? How many? How many threads are creating tasks?

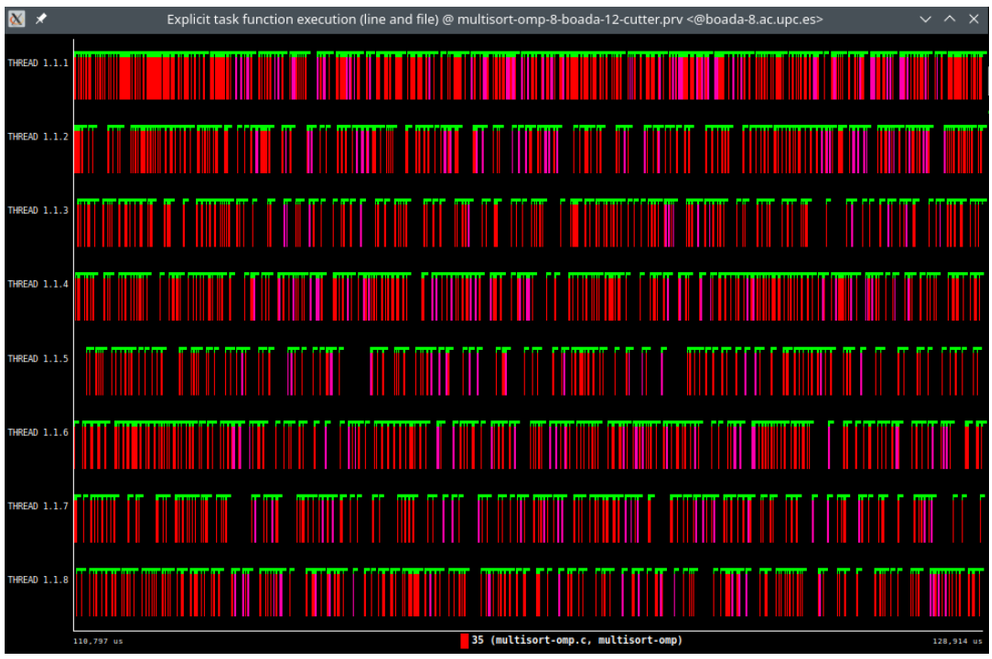
In the leaf strategy, in the 1st and 2nd(zoom) photographs we can see that the program is not genereating too many tasks during all the time so sometimes the execution is waiting without executing any tasks. For the other part, as we can see in the 5th and 6th(zoom) photographs, when there are tasks executing, all the threads are doing some execution task at the same time. We can finally see in the 3rd and 4th(zoom) photopraphs that thread1 is the only who is creating tasks.
In the tree strategy, in the 1st and 2nd(zoom) photographs we can see that the program is genereting more tasks that in the leaf strategy and all the 8 threads are busy all the time doing tasks executions. In the 3rd and 5th(zoom) photographs we can confirm that the threads are all executing tasks are the same time in this region and as we have said before, during all execution. We can finally see in the 4th and 6th(zoom) photopraphs that all the threads are the responsables and contribute with the work of the tasks creation.

Leaf strategy:

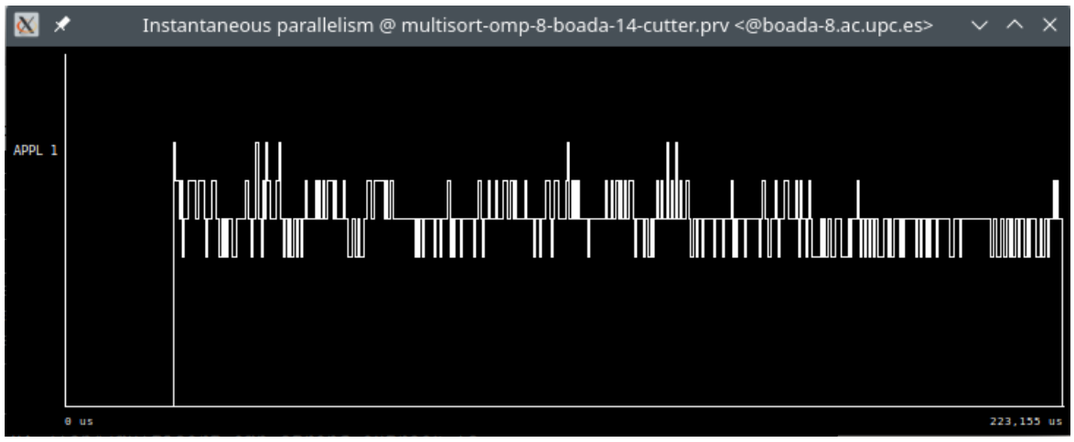


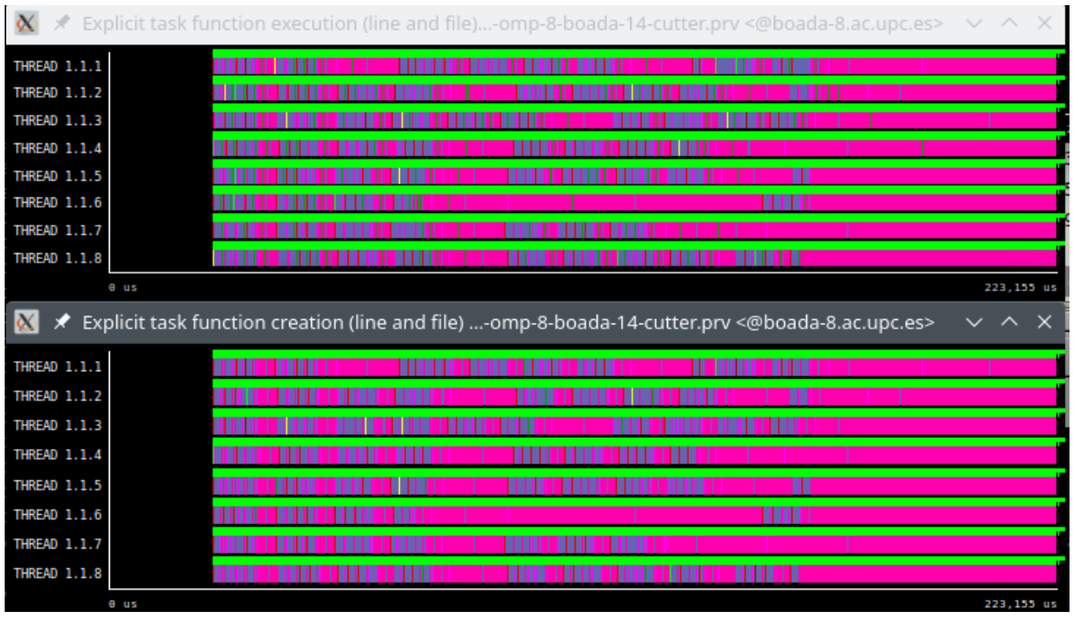

Explicit task function creation (line and file) @ multisort-omp-8-boada-12-cutter.prv <@boada-8.ac.upc.es>
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.1.5
THREAD 1.1.6
THREAD 1.1.7
THREAD 1.1.8
81,582 us
84,324 us

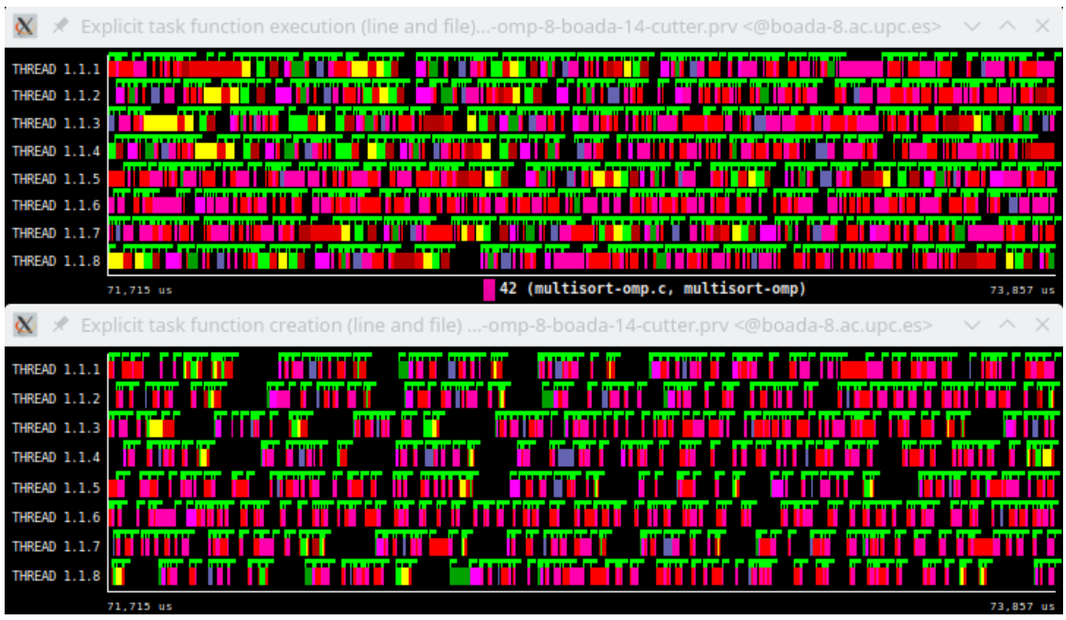

Explicit task function execution (line and file) @ multisort-omp-8-boada-12-cutter.prv <@boada-8.ac.upc.es>
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.1.5
THREAD 1.1.6
THREAD 1.1.7
THREAD 1.1.8
0 us
35 (multisort-omp.c, multisort-omp)
435,340 us


Explicit task function execution (line and file) @ multisort-omp-8-boada-12-cutter.prv <@boada-8.ac.upc.es>
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.1.5
THREAD 1.1.6
THREAD 1.1.7
THREAD 1.1.8
110,797 us
35 (multisort-omp.c, multisort-omp)
128,914 us

Tree strategy:







1.2.3 Leaf vs Tree strategy

Neither *leaf* nor *tree* strategies are showing a good parallel performance.

Comments/Observations

Is the granurality of both (*leaf* and *tree*) strategies influencing the parallel performance? What is the number of tasks created?

In the leaf strategy there is too little granularity for each tasks so it affect the execution time with the overhead that produces the generation of this tasks. The number of created tasks is 53248 tasks as we can see in the previous 3rt modelfactors leaf table. In the tree strategy the granularity is more adequate as the work is divided by the tasks but each tasks have the considerable work for that all the tasks can be executing all the time and do not produce too much creation overhead. The number of created tasks is 99669 tasks as we can see in the previous 3rt modelfactors tree table.

1.3 Task granularity control: the *cut–off* mechanism

1.3.1 Analysis with `modelfactors`

Include the tables of modelfactors for cut–off level equal to 4 and the information of the number of tasks generated for each of the cut–off levels used.

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	0.18	0.11	0.08	0.07	0.06	0.06	0.06	0.06	0.06
Speedup	1.00	1.57	2.30	2.70	2.89	3.00	3.02	3.07	3.15
Efficiency	1.00	0.78	0.57	0.45	0.36	0.30	0.25	0.22	0.20

Table 1: Analysis done on Thu May 11 01:48:18 PM CEST 2023, par4309

Overview of the Efficiency metrics in parallel fraction, ϕ=86.81%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	96.47%	82.30%	69.33%	59.87%	51.09%	43.46%	36.92%	32.48%	29.67%
Parallelization strategy efficiency	96.47%	84.05%	76.54%	69.48%	61.90%	53.41%	45.77%	40.57%	37.04%
Load balancing	100.00%	99.85%	98.87%	99.00%	96.49%	93.11%	89.35%	94.15%	87.55%
In execution efficiency	96.47%	84.18%	77.42%	70.18%	64.15%	57.37%	51.22%	43.09%	42.31%
Scalability for computation tasks	100.00%	97.92%	90.58%	86.17%	82.54%	81.38%	80.68%	80.07%	80.11%
IPC scalability	100.00%	92.42%	88.14%	86.85%	84.97%	84.65%	84.01%	83.78%	83.81%
Instruction scalability	100.00%	104.59%	104.57%	104.55%	104.54%	104.53%	104.57%	104.53%	104.54%
Frequency scalability	100.00%	101.30%	98.28%	94.90%	92.92%	91.97%	91.83%	91.44%	91.43%

Table 2: Analysis done on Thu May 11 01:48:18 PM CEST 2023, par4309

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)	13461.0	13461.0	13461.0	13461.0	13461.0	13461.0	13461.0	13461.0	13461.0
LB (number of explicit tasks executed)	1.0	1.0	1.0	0.94	0.97	0.95	0.95	0.93	0.94
LB (time executing explicit tasks)	1.0	0.99	0.98	0.98	0.98	0.96	0.94	0.96	0.92
Time per explicit task (average us)	10.43	11.43	12.71	14.01	15.47	17.01	18.82	20.39	21.75
Overhead per explicit task (synch %)	1.39	12.93	18.76	24.11	30.54	39.03	47.34	54.47	58.31
Overhead per explicit task (sched %)	2.44	5.59	10.31	15.63	22.1	29.78	37.88	43.56	48.24
Number of taskwait/taskgroup (total)	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0	2730.0

Table 3: Analysis done on Thu May 11 01:48:18 PM CEST 2023, par4309

Cut-off:0 NumberOfTasks: 0
Cut-off:1 NumberOfTasks: 189
Cut-off:2 NumberOfTasks: 805
Cut-off:4 NumberOfTasks: 13461
Cut-off:8 NumberOfTasks: 72021

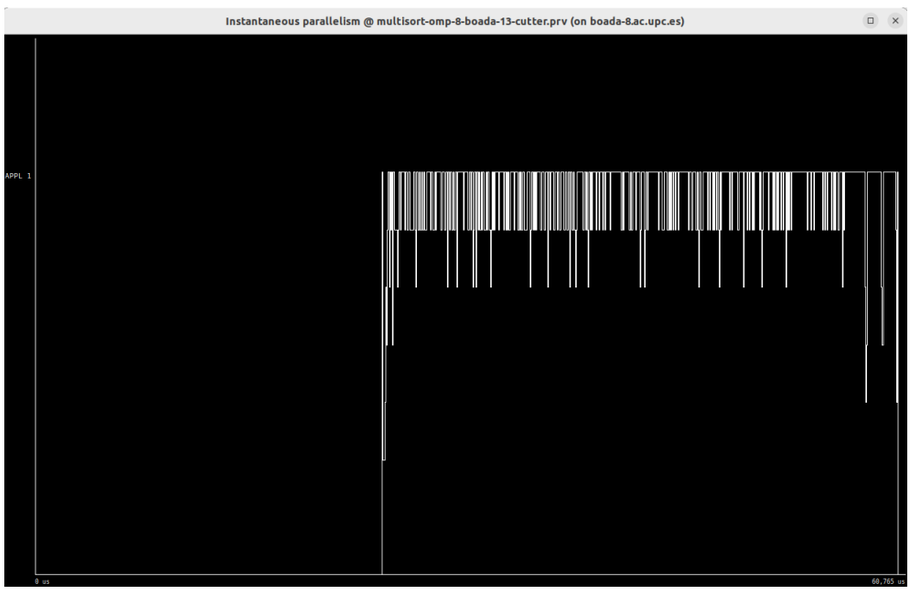
Comments/Observations

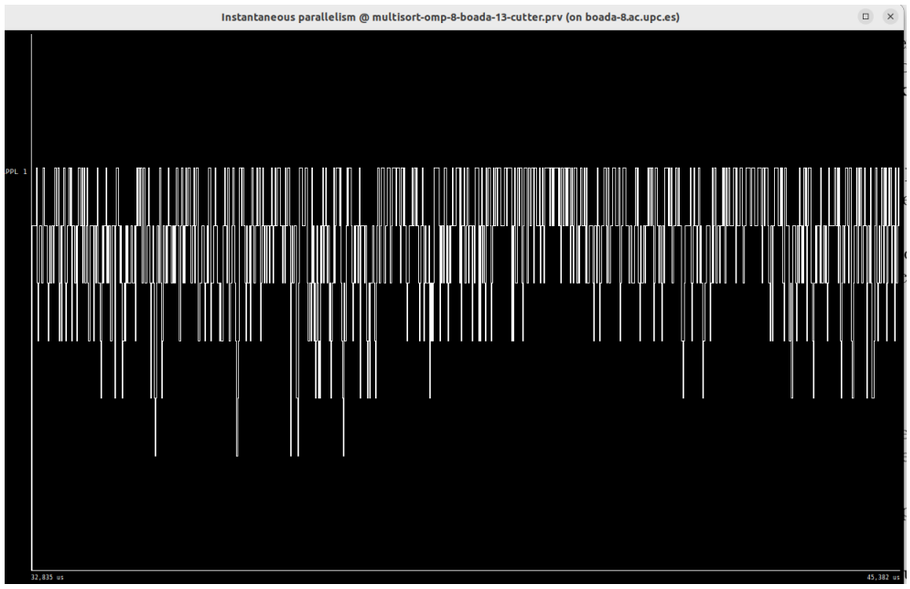
Which is the best value for cut-off? Does it significantly change with the number of threads used?

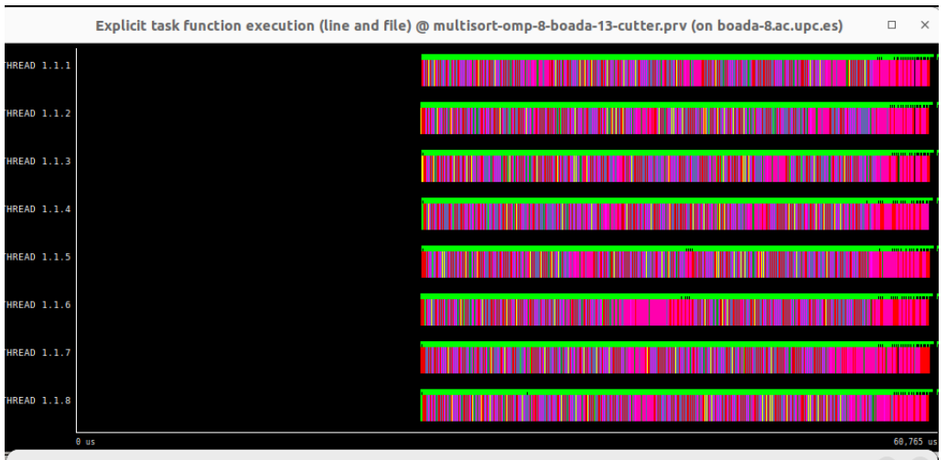
The best value of the cut-off depends on the number of threads we use as for example if we only use 1 thread it will be better to put a little cut-off number(for example cut-off = 0) because we will not have to pay for all the overhead creation tasks. For the other part if we use for example 16 threads it will be better to put a considerable high cut-off number(for example cut-off=8) because if don't put a high number it will be less tasks created and consequence there will be threads that will not execute tasks in large periods of time. So, we considerate that for a medium number of threads(8 threads) we can aplly a cut-off=4 that will limite the tasks creation overhead but there will not be threads waiting too much time.

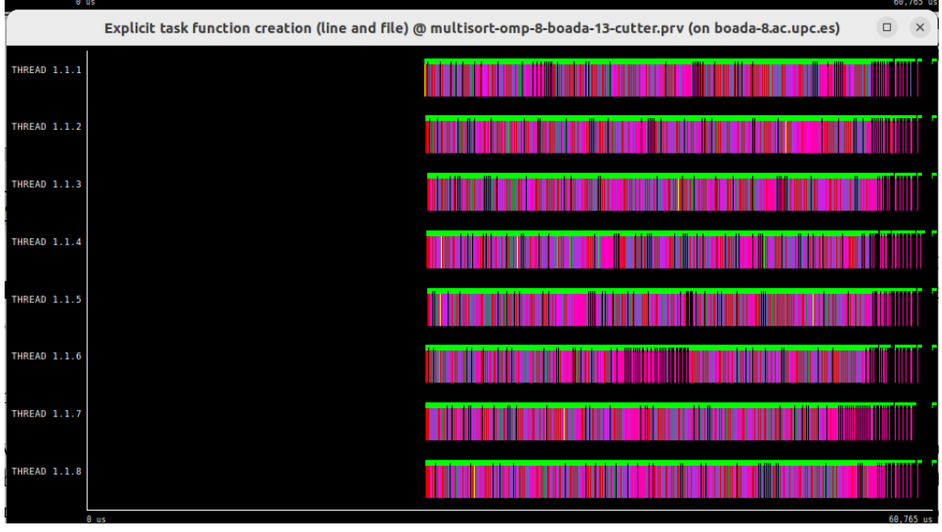
1.3.2 Analysis with Paraver

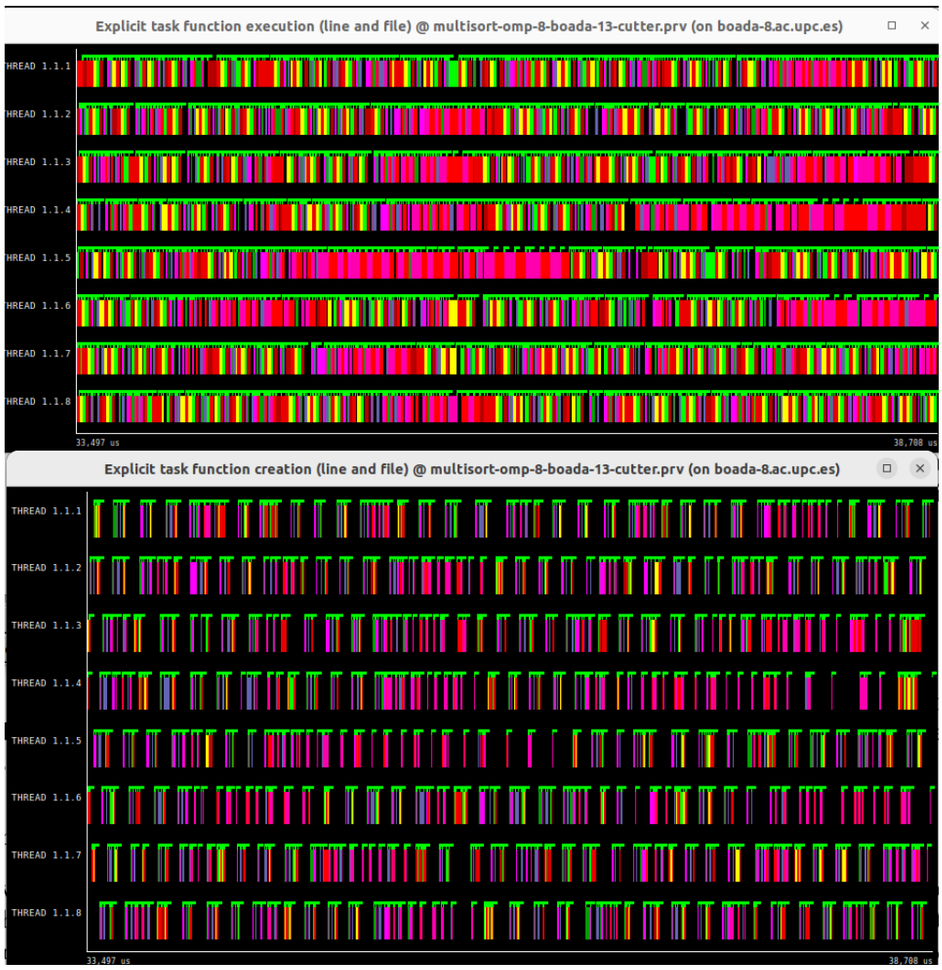
Include the Paraver trace visualizations you have analyzed for the execution with 8 threads.




Instantaneous parallelism @ multisort-omp-8-boada-13-cutter.prv (on boada-8.ac.upc.es)
PPL 1
32,835 us
45,382 us


Explicit task function execution (line and file) @ multisort-omp-8-boada-13-cutter.prv (on boada-8.ac.upc.es)
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.1.5
THREAD 1.1.6
THREAD 1.1.7
THREAD 1.1.8
0 us
60,765 us


Explicit task function creation (line and file) @ multisort-omp-8-boada-13-cutter.prv (on boada-8.ac.upc.es)
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.1.5
THREAD 1.1.6
THREAD 1.1.7
THREAD 1.1.8
0 us
60,765 us



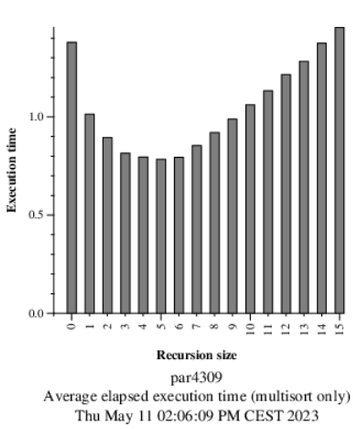
Comments/Observations

Is the instantaneous parallelism achieved larger than one along all the execution trace?

No, it is not achieved along all the trace as we can see in the photographs above. The tasks are not create nor executed after some period of time have passed. Then, in the middle of the time and when the tasks start to be created the execution remains contanstly until the final of the execution

1.3.3 Granularity Analysis

Include the postscript file generated by `submit-cutoff-omp.sh` to explore different cut–off levels for 8 threads.

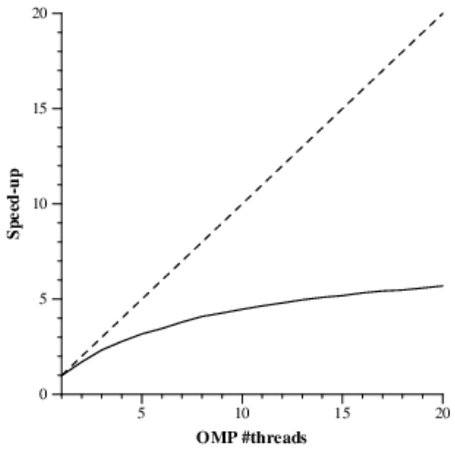


Comments/Observations

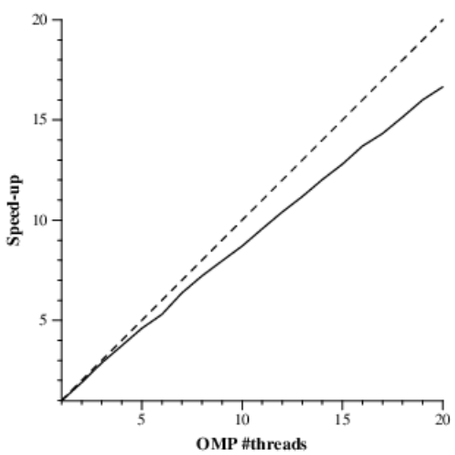
Which is the best value for cut-off for this problem size (look at the script to figure out the size of the problem) and 8 threads?

The best value for cut-off for this problem size (32768) and 8 threads is 5.

Include the strong scalability plot generated for the best cut–off level.



par4309
Speed-up wrt sequential time (complete application)
Generated by par4309 on Thu May 11 02:33:02 PM CEST 2023



par4309
Speed-up wrt sequential time (multisort funtion only)
Generated by par4309 on Thu May 11 02:33:02 PM CEST 2023

Comments/Observations

Are the speedup for the complete and `multisort` function close to the speed-up ideal? Reason the difference in speedup based on Paraver trace you have analyzed and the parallelization of your `multisort-omp.c` code.

For the complete application the real speed-up is not close to the ideal speed-up and for the multisort function the real speed-us is quite close to the ideal speed-up. If we see the instant parallelism plot generated by Paraver, we can see that in the beginning of the program the instant parallelism is very low but then during the most of the time the instant parallelism is in its maximum values, and finally near to the end of the program the instant parallelism decreases a lot.

2 Shared-memory parallelisation with OpenMP task using dependencies

Include:

- **Tables of modelfactors.**
- **Strong Scalability plot.**
- **The paraver traces analyzed.**

Overview of whole program execution metrics									
Number of processors	1	2	4	6	8	10	12	14	16
Elapsed time (sec)	0.29	0.19	0.15	0.12	0.11	0.11	0.10	0.10	0.10
Speedup	1.00	1.55	1.97	2.45	2.52	2.67	2.83	2.91	2.98
Efficiency	1.00	0.77	0.49	0.41	0.32	0.27	0.24	0.21	0.19

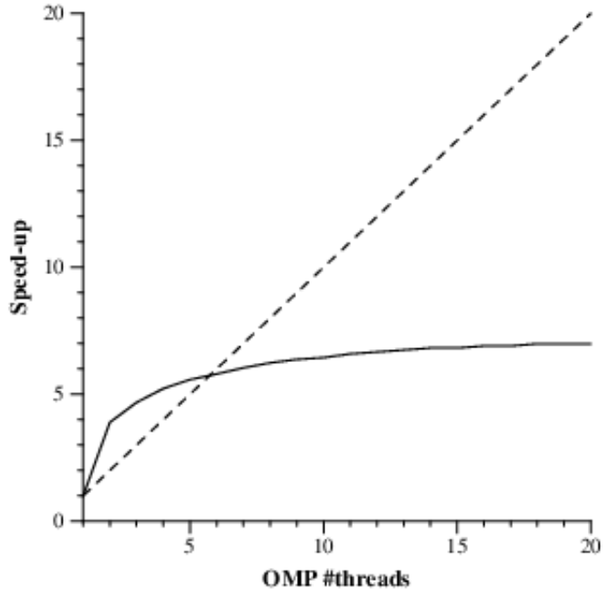
Table 1: Analysis done on Thu May 11 04:21:13 PM CEST 2023, par4309

Overview of the Efficiency metrics in parallel fraction, ϕ=91.91%									
Number of processors	1	2	4	6	8	10	12	14	16
Global efficiency	72.10%	58.43%	38.78%	34.04%	26.60%	22.91%	20.61%	18.37%	16.55%
Parallelization strategy efficiency	72.10%	51.81%	48.81%	45.68%	44.99%	42.23%	39.36%	37.08%	35.74%
Load balancing	100.00%	99.92%	98.52%	95.05%	96.79%	96.09%	88.57%	83.20%	82.97%
In execution efficiency	72.10%	51.85%	49.55%	48.05%	46.48%	43.95%	44.43%	44.57%	43.07%
Scalability for computation tasks	100.00%	112.78%	79.45%	74.51%	59.14%	54.26%	52.36%	49.54%	46.32%
IPC scalability	100.00%	97.66%	66.80%	63.65%	50.46%	47.08%	46.18%	44.29%	41.29%
Instruction scalability	100.00%	116.30%	116.27%	116.13%	116.12%	116.33%	116.31%	116.17%	116.26%
Frequency scalability	100.00%	99.29%	102.30%	100.81%	100.93%	99.09%	97.47%	96.30%	96.48%

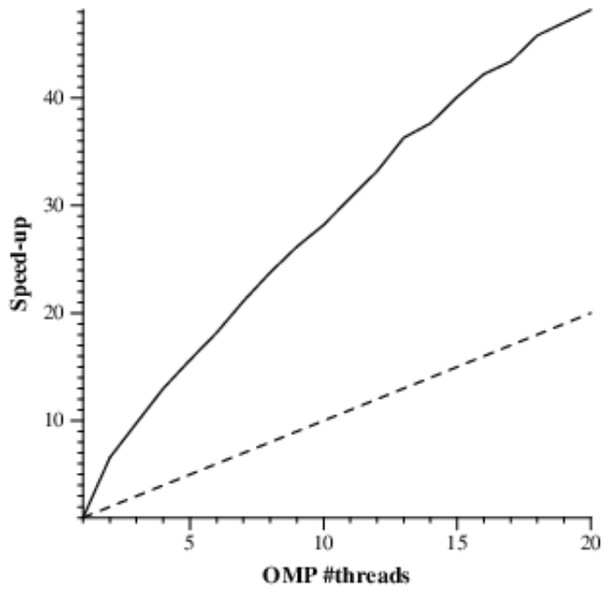
Table 2: Analysis done on Thu May 11 04:21:13 PM CEST 2023, par4309

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)	25557.0	25557.0	25557.0	25557.0	25557.0	25557.0	25557.0	25557.0	25557.0
LB (number of explicit tasks executed)	1.0	0.99	0.85	0.86	0.74	0.76	0.75	0.76	0.76
LB (time executing explicit tasks)	1.0	0.99	0.98	0.96	0.96	0.97	0.92	0.91	0.91
Time per explicit task (average us)	9.43	11.08	16.56	18.61	23.87	27.46	30.04	33.43	36.83
Overhead per explicit task (synch %)	15.58	45.72	48.0	50.33	51.14	52.11	53.83	54.96	56.24
Overhead per explicit task (sched %)	2.69	9.73	11.4	13.56	13.4	16.3	19.16	21.37	22.29
Number of taskwait/taskgroup (total)	99670.0	99670.0	99670.0	99670.0	99670.0	99670.0	99670.0	99670.0	99670.0

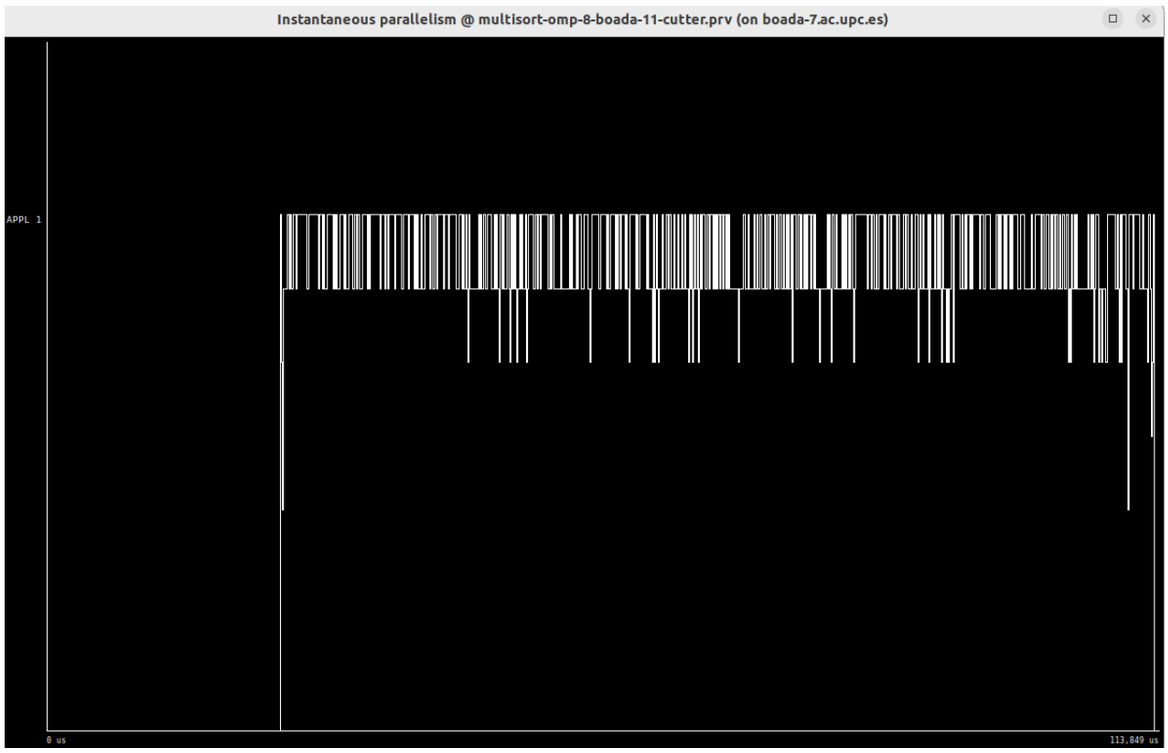
Table 3: Analysis done on Thu May 11 04:21:13 PM CEST 2023, par4309



par4309
Speed-up wrt sequential time (complete application)
Generated by par4309 on Thu May 11 04:42:22 PM CEST 2023



par4309
Speed-up wrt sequential time (multisort funtion only)
Generated by par4309 on Thu May 11 04:42:22 PM CEST 2023



Comments/Observations

Is this parallel implementation better or worse compare to OpenMP versions of previous chapter in terms of performance? In terms of programmability, was this new version simpler to code?

Is better because we are indicating the explicit dependencies in terms of variables, not by blocks (taskgroup). In terms of programmability this new version is harder than the previous one to code, because in the previous version with 2 taskgroups the synchronization was good and to do this new version it is essential to understand what dependencies (read and write) are needed to keep the synchronization.