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

Your names here, group, date, and include boada usernames

Add delivery date here

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

How to:

Document contents

The length of the document is expected to be maximum 8 pages (including figures and tables) with a font size of at least 11pt.
   You have to add comments and observations following the questions that you have found along the document indicating **For the deliverable:** in each section (in this laboratory we have included those **For the deliverable** questions in the notebook with setences in gray under each **Comments/Observations**). DON'T include code in the document except for the fragment modified with respect to the original one: just provide a code except if you consider necessary.
   Finally, as you know, this course contributes to the **transversal competence "Tercera llengua"**. Deliver your material in English if you want this competence to be evaluated. Please refer to the "Rubrics for the third language competence evaluation" document to know the Rubric that will be used.

Submission

Only PDF format for this document will be accepted. Deliverable assignment will be opened in *Atenea* and set the appropriate dates for the delivery. You also have to deliver the complete C source codes for Tareador instrumentation and all the OpenMP parallelisation strategies that you have done. Your lab professor should be able to re-execute the parallel codes based on the files you deliver. You will have to **deliver TWO files**, one with the document in PDF format and one compressed file (`tgz`, `.gz` or `.zip`) with the requested C source codes.

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.

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?

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.

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, ...

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?

1.2.3 Leaf vs Tree strategy

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

Comments/Observations

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.

Comments/Observations

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

1.3.2 Analysis with *Paraver*

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

Comments/Observations

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

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?

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

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.

2

Shared-memory parallelisation with *OpenMP* task using dependencies

Include:

- Tables of modelfactors.

- Strong Scalability plot.

- The *Paraver* traces analyzed.

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?