PAR Laboratory Assignment
Lab 3: Iterative task decomposition with OpenMP: the computation of the Mandelbrot set

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

Add delivery date here

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

1

Laboratory 3 notebook

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**). We have also included a cross-reference to the practice document. **The length of the document is expected to be approximately 8-10 pages (including figures and tables) with a font size of at least 11pt**. 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. DON'T include code in the document except for the fragment modified with respect to the original one: just provide a code except. 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.

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 *R*ubric that will be used.

1.1 Task decomposition analysis with *Tareador*

1.1.1 First analysis with *Tareador*

This part refers to point 2 of subsection 2.2 of the practice document.

Include the TDG of the first *Tareador* analysis (without `-d` nor `-h` option).
Comments/Observations
Which are the two most important characteristics for the task graph that is generated?

1.1.2 Second analysis with *Tareador*

This part refers to point 3 of subsection 2.2 of the practice document.

Include the TDG of the second *Tareador* analisis (with `-d` option).
Comments/Observations
Which are the two most important characteristics for the task graph that is generated? Which part of the code is making the big difference with the previous case? How will you protect this section of code in the parallel *OpenMP* code?

1.1.3 Third analysis with *Tareador*

This part refers to point 4 of subsection 2.2 of the practice document.

Include the TDG of the third *Tareador* analysis (with `-h` option).

Comments/Observations

What does each chain of tasks in the task graph represents? Which part of the code is making the big difference with the two previous cases? How will you protect this section of code in the parallel *OpenMP* code?

1.1.4 Point Strategy analysis with *Tareador*

This part refers to the last question of subsection 2.2 of the practice document.

Include the part of `mandel-tar.c` you have modified to analyse the `Point` strategy and the TDG of the *Tareador* analysis.

Comments/Observations

Which is the main change that you observe with respect to the *Row* strategy?

1.2 *Point* decomposition strategy

1.2.1 Explicit tasks without taskloops

This part refers to the overall analysis with *modelfactors* and detailed analysis with *Paraver* of subsection 3.1 of the practice document.

Include here the tables generated by *modelfactors* after executing the `submit-strong-extrae.sh` script and the *PostScript* file generated by `submit-strong-omp.sh` script for the *OpenMP* version with explicit tasks and with no taskloops. Also, include some captions of the *Paraver* analysis to visualise when the *explicit tasks* are created and executed.

Comments/Observations

Is the speed–up appropriate? Is the scalability appropriate? Which threads are creating and executing the explicit tasks? Is one of the threads devoted to create them, and therefore executing much less? If yes, this should be a clear contributor to load unbalance, right? Do you think the granularity of the tasks is appropriate for this parallelization strategy?

1.2.2 Taskloops without using neither `num_tasks` nor `grainsize`

This part refers to the optimization: granularity control using `taskloop`, overall analysis part of subsection 3.1 of the practice document.

Include here the *modelfactors* tables generated by the `submit-strong-extrae.sh` script and the *PostScript* file generated by `submit-strong-omp.sh` script for the *OpenMP* version with taskloops without using neither `num_tasks` nor `grainsize`.

Comments/Observations

Do you think the granularity of the tasks is appropriate for this parallelization strategy? Is the version with `taskloop` performing better than the last version based on the use of `task`? Do you notice a relevant change in the two metrics that contribute to the parallelization strategy efficiency? Should you blame to "load balancing" or to "in execution efficiency"? Remember to scan the information provided in the third table that is included in the `modelfactor-tables.pdf` file. What can you tell now from the total number of tasks generated and the new average execution time for explicit tasks and the synchronization and scheduling overheads percentage? How many tasks are executed per `taskloop`? Is the task granularity better? But notice that task synchronization still takes a big %. Why there are now task synchronizations in the execution? Where are these task synchronisations happening?

1.2.3 Taskloops without `nogroup`

This part refers to the optimization: granularity control using `taskloop`, detailed analysis and possible optimization part of subsection 3.1 of the practice document.

Include here the *modelfactors* tables generated by the `submit-strong-extrae.sh` script for the *OpenMP* version with taskloops with `nogroup` clause.

Comments/Observations

Has the scalability improved? What about task granularity and overheads?

1.3 *Row* decomposition strategy

This part refers to subsection 3.2 of the practice document.

Include here the *modelfactors* tables generated by the `submit-strong-extrae.sh` script.

Comments/Observations

Which are the main differences that you observe when comparing the *Row* and *Point* strategies? What is the number of tasks created for *Row* and *Point* strategies? Is there any load unbalance? What is the task synchronisation cost now?