ETHzürich

High Performance Computing for Science and Engineering II

Spring semester 2019

P. Koumoutsakos, S. Martin
ETH Zentrum, CLT E 13
CH-8092 Zürich

HW 4: High-Throughput Computing with UPC++

Issued: April 1, 2019
Due Date: April 22, 2019 10:00am

Warm-Up & Information

This homework requires the use of *Korali*, our new Uncertainty Quantification and Optimization framework developed at ETH Zürich. Along with this pdf, you should download the provided code which contains the following files/folders:

- *korali/* contains a ready-to-use, pre-compiled version of Korali. The library and some of the task models will only work on Euler (or compatible ETH systems). The only step required to make it work is to decompress the tar ball: $tar -xzvf korali/lib/libkorali.tar.gz
- *plotting/* Contains Python and MATLAB codes required to plot the distribution plots based on the output of Korali's TMCMC engine.
- *modules.sh* Execute it to load the modules required to run this homework on Euler.
- *task1/* Contains the skeleton code for Task 1.
- *task2/* Contains the skeleton code for Task 2.
- *task3/* Contains the skeleton code for Task 3.
- *task1/conduits/upcxx.cpp* Is the skeleton code for Task 4.

Guidelines for Moodle submission:

- For HW4: at least 4 solution codes: task1.cpp task2a.cpp task2b.cpp task3.cpp upcxx.cpp. You can also include codes for optional items.
- Your report PDF. Do not forget to answer every question in the HWs.

Grading

- CSE Students: Max 170 points to be converted linearly to a 1-to-6 scale.
- Engineering Students: Max 100 points to be converted linearly to a 1-to-6 scale.
- Bonus: Every correct optional item submitted will grant 0.25 bonus points to improve the grade of this or another homework, up to 6 points.

Installing and Using UPC++/Korali Please, check this guide to learn to install, compile, and install a UPC++ application. To select Korali's conduit, run:
$export KORALI_CONDUIT=**single**, for the single-core conduit, and:
$export KORALI_CONDUIT=**upcxx**, for the UPC++ conduit (to be developed by you).

Task 1: n-Candle Problem. (40 Points)

Here we are revisiting the n-Candle problem from Homework 3b. The company is expanding, thanks to the consistently good quality of the produced chassis. The company decides on investing a large amount of capital on the automation of the production and heat treatment of the metal sheets on the way to mass production of the cars' chassis. The company decides on investing on 4 robotic heat treatment units that will work in parallel on the 2D metal plates. The robotic torches work with a beam of a certain width and intensity and can be placed quite close to each other.

As the head of operations in the factory, your task is to manage the operation of the four torches in order for the metal sheets to pass the structural failure tests. Again, you are armed with the automatic machine that registers the temperature at multiple points of each sheet, just after they leave the treatment facility. The structural engineers of the company managed to create an optimal temperature profile of a metal sheet that would according to their simulations pass all integrity and failure tests. Your task will be to define the position and the technical characteristics of each torch beam (width and intensity) to replicate the optimal temperature profile you obtained by your engineers.

You decide to use your previous knowledge of Korali to solve this optimization problem. The metal sheets are still square and 1m long. The specifications for the robotic torches the company acquired are: "The intensity of the beams for metal treatment should be within $0.4-0.6$ (a.u.), whereas the width of the beams should be kept below 0.06 ($\times 10^{-2}$) m. The minimum beam width is 0.04 m. Due to spatial constraints in the factory, you decide on splitting the torches along the two sides of the heat treatment facility, i.e. two of the torches will be placed in the lower half of the x-axis ($0.0-0.5$ m) and the other two torches will be placed in the upper half of the x-axis ($0.5-1.0$ m).

Because the positioning of the robotic torches is much more accurate than the human hands, the grid size of the solver has increased to 2^7+1 elements per side (heat2d.cpp), which increases the run time of the heat-2d solver.

- Describe the model and report the number of parameters you will need to represent this problem in Korali.
- What is the most likely (x,y) position and characteristics (beam width, intensity) of each of the robotic torches?
- How much time did Korali take to solve this problem? What could we possible do to improve this, or which parts would you parallelize first in CMA-ES? (for

reference see Algorithm 1: simplified CMA-ES Pseudo Code).

Algorithm 1 CMA-ES Pseudo Code

```
procedure OPTIMIZE
    Initialize CMA-ES parameters:
    $N \leftarrow$ number dimensions
    $\lambda \leftarrow$ number offspring population size
    $u \leftarrow$ parent population size for next generation
    $\sigma^{(0)} \leftarrow$ initial standard deviation, respectively step size
    $C^{(0)} \leftarrow I$ (initial covariance matrix)
    $\mu^{(0)} \leftarrow$ initial mean
    while stopping criterion is not met do
        $A \leftarrow chol(C^{(g+1)})$
        for $k = 1 : \lambda$ do
            $x_k \sim N(\mu^{(g+1)}, A)$ (generate new Sample Population)
        end for
        for $k = 1 : \lambda$ do
            evaluate $f(x_k)$
        end for
        sort results, find best $u$ samples
        update best ever solution
        $C^{(g+1)} \leftarrow$ updated Covariance Matrix (based on $u$ best samples)
        $\mu^{(g+1)} \leftarrow$ updated Mean (based on $u$ best samples)
        $\sigma^{(g+1)} \leftarrow$ updated Step Size
    end while
end procedure
```

Task 2: Parallel Tasking (60 Points)

Your recent success as the Factory's head of operations and Korali expert has made you a prestigious figure in the manufacturing world. Soon, you are approached by the company's CEO, who tells you:

"You have proven that Korali is an excellent tool to determine the best treatment for car's metal plates. I am now planning to expand our operations to compete in the airplane industry. Our initial research indicates that we will need to treat many different types and sizes of metal plates. However, your tool still takes too long to provide results for a single plate. We need to find a way to produce results in a much shorter time. If you can show that you can solve our current problem **10 times faster**, we will be able to make the move. For this, you will earn a generous raise".

Determined to prove you are up to the task, you embark yourself in a quest to parallelize Korali, making it much more efficient.

a) (30 Points) We will start by building a simple parallel tasking engine for a problem in which all samples are well-known at the beginning of the generation. Such is the case of CMA-ES in which all the new samples are taken from the previous generation's best fitting samples. To solve this, we will apply a **divide-and-conquer** strategy using UPC++, distributing the samples among ranks and having each rank evaluate it's set of samples in parallel. First, take a look, compile, and execute the **single** tasking engine. This engine will only run in a single-core, just like the current Korali. Build your parallel version of the engine on task2a.cpp and run it using 24 ranks on an Euler compute node. Answer:

   - How much faster did your parallel version run, compared to the single-core version? What can you say about the efficiency of your implementation?
   - Compare the total time taken by your engine to the average time taken by each rank. What can you say about load balance in this case?
   - **<Optional>** Build a similar parallel engine using MPI, and answer: Did you take a similar approach? Was it easier or more difficult to implement? Provide the code as task2a_mpi.cpp and justify your answers.

   Note: Your submission(s) need to successfully run *checkResults()* to be considered correct.

b) (30 Points) Use task2b.cpp to build an alternative parallel engine that solves the load imbalance problem using the **producer-consumer** strategy, and answer:

- What can you say about load imbalance in the producer-consumer strategy.
- How much did your new engine improve over the divide-and-conquer strategy?
- **<Optional>** Build a similar parallel engine using MPI, and answer: Did you take a similar approach? Was it easier or more difficult to implement? Provide the code as task2b_mpi.cpp and justify your answers.
  Note: Your submission(s) need to successfully run *checkResults()* to be considered correct.

Task 3: Real-Time Sampling (30 Points)

<Mandatory for CSE Students> <Optional for Engineering Students>

Not every sampling/optimization engine produces all the samples at the beginning of the generation. Instead, other methods like Transitional MCMC, will only evaluate an initial set of samples and use their evaluations to produce a set of new candidate samples to evaluate. Therefore, it is not possible to employ a static distribution of work at the beginning.

In this exercise, we provide a sampler that only produces 24 initial samples, and requires to be updated with their evaluations before being able to produce new samples.

You will have to design an engine that constantly receives the latest results to produce new samples to verify. Check **single** to gain an idea of what functions are used to update evaluations and obtain new samples to be evaluated. Answer:

- What challenges did you face during this exercise? What UPC++ tools did you use to overcome them?
- **<Optional>** Build a similar parallel engine using MPI, and answer: Did you take a similar approach? Was it easier or more difficult to implement? Provide the code as task3a_mpi.cpp and justify your answers.

Note: Your submission(s) need to successfully run *checkResults()* to be considered correct.

Task 4: Parallel Korali (40 Points)

<Mandatory for CSE Students> <Optional for Engineering Students>

By now you have become familiar with UPC++ and are ready to put your new skills to use in parallelizing Korali and obtain your well-deserved raise.

Use the principles you applied in **Task 3** to implement the UPC++ conduit for Korali. First, analyze the implementation of the *Single* conduit in task1/conduits/single.cpp. Then, modify task1/conduits/upcxx.cpp to create Korali's UPC++ conduit.

- Briefly mention the challenges you faced during this point and what UPC++ tools did you use to overcome them.
- Run your new conduit on 24 cores of an Euler computing node. How much faster did your parallel version run, compared to the single-core version for Task 1? What can you say about the efficiency of your implementation? Did the CEO of the company give you a raise?
- **<Optional>** Build a similar parallel engine using MPI. Work it out on top of the upcxx.cpp file, but provide the code as mpi_conduit.cpp. Answer: Did you take a similar approach? Was it easier or more difficult to implement? Justify your answers.

Note: Your submission(s) need(s) to achieve a similar result to that of *./single*, and at least 10x faster to be considered correct.