Scientific and Technical Computing
Homework: Hardware and Code Optimization

Part 2: Calculating pi

General Description

In this homework you will write, compile and execute code on Frontera's Cascade Lake nodes. You will report execution times and inspect optimization reports. Based on the data collected you will provide logic about optimization strategies.

General Instruction

Everything in blue will require a response in your report.

Posting date: 10/26/20

Return date: 11/09/20, afternoon (2 weeks after posting)

How to submit: Submit your pdf file through Canvas

What to submit: Short document, maybe 2 pages.

Purpose and scope

In this home work you will setup and run an experiment by yourself. The instructions will mostly show what to measure and what the output is supposed to look like. All the programming details are up to you.

Instructions: Setup

- Compile and execute your experiments on a Frontera Caskade Lake node. Refer to the instructions in the other home work for details, if you are unsure about this.
- Create a directory where you will conduct the experiment --- somewhere on home, work, or scratch. Choose a name and include the location (full path) in your report. You may use the command **`pwd`** to get the full path.
- Generally the code is compiled with one of these commands:
  ```
  icc   -xcore-avx512 -O2 <source-file>
  ifort -xcore-avx512 -O2 <source-file>
  ```

Calculating pi

As discussed in class the following python instructions can be used to calculate pi

```
In [1]: import numpy as np

In [2]: points     = np.random.random((10000, 2))

In [3]: points2    = np.square(points)

In [4]: norm2      = np.sum(points2, axis=1)

In [5]: num_inside = np.sum(norm2 <= 1.0)

In [6]: 4.0 * num_inside / 10000.0
Out[6]: 3.1256
```

Explain how the algorithm works (2 points)
What size and shape are the arrays `points`, `points2`, `norm2`? (1 point each)

Programming: Part 1

Write code in your favorite language, i.e. C/C++ or Fortran, that is equivalent to the python code. Specifically use individual loops for each step of the code.

- Use a constant (C/C++) or parameter (Fortran) to set the number of points. For testing you may use a small number.
- If your code does not finish in a reasonable time period for the sample size, then reduce the sample size.
- Create the same arrays as in the python code
- Use single or double precision arrays in your code
- Important: Use a 'double precision' float or real number for the variable `'num_inside'`
- Add a timer to your code. Start the timer after 'Step 2', i.e. after the array points has been filled, and stop the timer after 'Step 6'

Assignment: Part 1

Add statements to your code so that the output is generated that shows what type of precision is used, how large the sample size was, how much memory was used in the arrays (in MB and GB), the value of pi, and how much time the calculation took. Here is an example. The '…' represent the values.

```
Precision                       =    …
Sample size                     =    …
Total memory footprint (MB)     =    …
Total memory footprint (GB)     =    …

Pi = 3.14…
Timing :: … (seconds)
```

Assignment: Part 2

Explain how often data is moved from memory into the CPU (and back!) for the individual steps. Does the cache (or the caches) provide any help to boost performance? (3 points)
Why would the calculation give an incorrect result for a large sample size (like the one we are using here, see 'Part 4'), if the variable `'num_inside'` were a single precision number? (2 points)

Assignment: Part 3

Copy your code from part 1 into a new file. Perform the following modifications

- Change the part between the calls to the timer
- Do not change how the random numbers are created, nor the array `'points'`
- Change the calculation of `'num_inside'`, i.e. steps 3 to 5, such that the calculation is done in one loop. This should eliminate using the 2 other arrays `'points2'` and `'norm2'`.
- Add output with this information:

```
Pi = 3.14…
Timing (single loop) :: … (seconds)
```

Assignment: Part 4

- For your report, set the number of the sample size to 1000000000. Use a smaller number if your two codes do not finish in a reasonable time (minutes).
- Run the code on a Frontera Cascade Lake node
- Use `numactl -N 0 -m 0 <your-executable>` to run your code
- Add the ouput from 'part 1' (10 points) and 'part 3' (10 points) to your report. Manually add a line that shows how much faster or slower the second code is compared to the first code
- Include the source code (for both codes) to your report. Do not show the 2 complete codes, but only the parts between the calls to the timer.

Assignment: Part 5

Explain/speculate why the execution speed is different. (5 points)

Best of luck!