The Thread Building Blocks Lab Time Hands-on

SPD Course

2017-18

Massimo Coppola

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

TBB Installation

- Download latest TBB either as binary package (for your OS) or as source package
  - unpack in your working directory
  - If starting from source, check out the readme and compile it
- Be sure to have a suitable tool chain (e.g. recent C++ compiler and linker version)
  - https://www.threadingbuildingblocks.org/system-requirements
- ***EITHER*** Set up your environment for command line usage
  - Look for the script …/bin/tbbvars.* (sh / csh versions)
  - Add your install directory inside the script by editing it
  - Call it from a shell every time you need TBB (or add to .profile)
- ***OR*** follow the instructions in TBB readme files to use GUI-based coding environments
  - Check your own : Eclipse, MS Visual Studio, Xcode
  - Note that many configurations work besides those officially listed - Eclipse with CDT plugins is known to support TBB on Linux and OS X

Details

- Set up the install dir of TBB inside the tbbvars script
- Call the script in your bash/csh profile passing arguments <architecture> <os>
  - On ottavinareale: intel64 linux
  - On the xeon phi: intel64

- Example:

```
source /opt/intel/tbb/bin/tbbvars.sh intel64
```

Exercise 1

- Write a parallel for with 1d and 2 ranges in steps
  - Without any actual computation (leave operator() empty)
  - Without an actual value passed, just the indexes
  - Passing an initialized array of given type: have the operator() perform some computation
  - Add another array to store the results
- Take the 2d version, add a Mandelbrot function and compute over a 2D range that spans a rectangle in the complex plane
  - result is an integer = number of iterations performed before detecting divergence

Exercise 1

- Some code snippets are found here
  http://didawiki.cli.di.unipi.it/doku.php/magistraleinformaticanetworking/spd/2018/mandel
  - Code for the Mandelbrot function
  - Code for saving array data as .ppm files you can view
- Other ideas:
  - compute any long, iterative function and write the result somewhere, so that the compiler does not optimize it away

Exercise 2

- Starting from the Mandelbrot example do the following
  - Set the investigated area to cover at least part of the mandelbrot set
  - Raise the number of pixels and the number of iterations of a couple of order of magnitude
    - Set TBB to only use **one** thread – see example there: https://software.intel.com/en-us/node/506296 where explicit task scheduler init allows to control the thread pool size
    - Experimentally find values that cause a sequential running time in the range in between 15 and 60 s
    - Allow TBB to create TN>1 threads, measure the difference in execution time with varying TN

Exercise 2

- Devise a mechanism for approximating the distribution of computation time over the tasks due to the function imbalance)
  - E.g. use an array of buckets where you total how many times a certain number of iterations shows up
  - This is a source of thread conflicts: evaluate the use of atomics, locking, or thread-local structures in order to gather the sums efficiently (reduce the performance impairment due to the monitoring)
  - Is it possible to model the program computation time using this information?

Exercise 3

- Starting up from the description of the K-means algorithm and the example code on the didawiki page, **implement a TBB version of the K-means code**
  - The provided code has a random data generator function you can use to test your own code
  - Code is in C, you will have to port it to C++
  - Code should be general, but stay with 2D data to be able to see the output
  - There is support for showing results of the algorithm using gnuplot over the dataset and the code output
- One useful level of **parallelism** is **among different rows of the dataset**
  - Store the data in a suitable container
  - Employ a parallel for, analyze the impact of different partitioners and grain size choices
  - For this test, adopt a fixed number of iteration as the stopping criteria
- If the space dimension is fixed to 2D, is may be worth to use vectorized types fro storing the dataset

Exercise 4

- Evolution of the K-means algorithm in TBB
- A second useful level of **parallelism** in these settings is **among several runs at the same time**
  - Modify the code to manage multiple sets of centroids at the same time
    - At least 3
    - Allow for one or more centroid sets to be inactive
  - Select a termination criteria to be applied to the search
    - e.g. magnitude of centroid movement, or number of points which change assignment at each iteration
  - Apply the criteria separately to each centroid set
    - Allow a terminated search to save its result and restart at random points
  - Evaluate performance/efficiency over a **(1)** fixed number of iterations (which may produce a varying number of solutions) and **(2)** a fixed amount of time in the search