Introduction to FastFlow programming

Massimo Torquati
<torquati@di.unipi.it>

SPM lecture 4 -- Class Work

Master Degree in Computer Science
Master Degree in Computer Science & Networking
University of Pisa

Class Work lecture

- Today you will use FastFlow to parallelize some simple computations using the parallel patterns that we have seen so far (Task-Farm, Master-Worker, ParallelFor)
- To install FastFlow in your local Linux machine (or Docker container):
  - *git clone* https://github.com/fastflow/fastflow.git
    - or update your existing version by going into the *fastflow* directory and then *git pull*
  - cd into the ff directory and run the **mapping_string.sh** script accepting the result proposed (the script modifies the ff/config.hpp file). Note that you have to re-run the script if you copy or download FastFlow on a different machine!
    - To run the script you need a modern bash shell and **hwloc** (e.g., sudo apt install hwloc)

Exercise 1 (π approximation)

- Approximate π by computing the definite integral between 0 and 1 of the function

$$f(x) = 4.0/(1+x*x)$$

- The sequential program as well as the OpenMP version that uses a parallel-for, have already been provided to you in Lesson 12 (*pi-seq.cpp* and *pi-parfor1.cpp*, respectively).
- You have to implement the algorithm by using the FastFlow library:
  - A first version based on the **ParallelForReduce** pattern mimicking the OpenMP version
    - NOTE: pay attention to correctly privitize the variables!
  - A second version based on the **ff_Farm** pattern in which the Emitter schedules the partitions of the iterations space to the Workers; each Worker computes a local sum of the values of the partition; the Collector computes the final sum.
- Test your implementation by using a range (num_steps) of 100M intervals

Exercise 2 (finding primes)

- Finding all prime numbers in a given range of values.
  - Example, all primes in [200,250] are: 211, 223, 227, 229, 233, 239, 241
- The sequential version is provided (*primes.cpp*). You have to implement two versions:
  - A **Master-Worker** version in which the Master schedules sub-ranges of the original range to the Workers (*keeping one range for itself,* in case); the Workers compute the primality test on the sub-ranges received and return the vector of primes they found; the Master will rearrange all the primes computed in a single (ordered) vector.
  - A **ParallelFor**-based version of the program. Try different scheduling policies (static and dynamic with different chuck sizes), and to disable the scheduler.
    - It might be useful to use the *parallel_for_idx method* of the ParallelFor class, which allows you to know the index of the thread running the lambdas.
- How many primes are there in the range [950M, 1000M]?

Comments

- For the farm-based implementations (i.e., the ones that require more coding), first try to implement a working solution, then try to optimize the code considering how to reduce the number of communications, the usage of dynamic memory, try some profiling, etc.
- We shall discuss a possible solution for each version at the beginning of the next lesson
    - I will provide you “a possible solution” beforehand so that you can compare with your solutions and to ask questions/provide your comments during the lesson.