Low-Level Parallel Programming

Project Information and Lab Assignment 1
Lab 1 Deadline: February 5, 23:59

1 Project Outline

This system simulates an area of people[1] walking around. Each person, or *agent*, is following a circular sequence of *waypoints* in the 2D area. The starting locations and waypoints are specified in a *scenario configuration file*[2], which is loaded at the beginning of the program run. At the beginning of the project the agents are ghosts - they may walk straight through each other without taking care of any collisions. With each update of the world, a *tick*, two things happen.

1. The *model* is updated.
2. The graphical visualization is updated.

The tick interval thus decides the frame rate.

This system has a problem. For really large simulations the model consumes more time than the available tick interval, and thus slows down the simulation. Your objective in this project is not extreme optimization, but rather to explore parallel tools and constructs, in an effort to make the system "fast enough". Of course, parallelization and optimization goes hand in hand, but remember the objective, to learn and compare different tools and approaches for code parallelization.

The project is divided into five labs. With each lab, we introduce new functionality to the simulation. We provide the sequential solutions. Your objective is to patch in the functionality to your codebase, to parallelize it and to measure the performance gains, if any.

[1]Even though they look like little balls, they are really people, we assure you.
[2]such as the given "scenario.xml"

Submission	Date
Lab 1	2015-02-05 23:59
Lab 2	2015-02-17 23:59
Lab 3	2015-02-25 23:59
Lab 4	2015-03-04 23:59
Lab 5 / Final Submission	2015-03-10 23:59

Table 1: Deadlines

2 Deadlines, Demonstrations, and Submissions

Each lab has to be submitted by its corresponding deadline (see Table 1). During the following lab session, demonstrate your working solution to a lab assistant. Be prepared to answer questions.

3 Setting Up The Project

Download the project from Studentportalen. You may choose to either work on the university windows work stations, or to use your own computers.

You may choose to work on your own computer. However, remember that the upcoming labs will include GPU programming. We only support CUDA which requires a computer with an NVIDIA graphics card. If you don't have an NVIDIA graphics card but still want to use your own computer, you're welcome to use OpenCL instead.

3.1 Building on Windows

Please use the separate Windows solution available[3] for download at studentportalen. Unpack and open with Microsoft Visual Studio 2010.

3.2 Building on Linux

We officially support Ubuntu 14.04 and onwards, but other versions and flavors should have little or no problem building the project.

1. Install Qt4 development libraries (ubuntu package qt4-dev-tools).
2. **If** you have an NVIDIA graphics card on your machine, install the CUDA development toolkit (ubuntu package nvidia-cuda-toolkit).

[3]Archive not uploaded yet at the time this is written

3.3 Building on OSX

Make sure that you have Qt4 installed. The compiler Clang doesn't support OpenMP yet, therefore we will be working with gcc. Here's a quick guide on how to make Qt work with GNU's gcc 4.9.

1. Install gcc 4.9 (brew install gcc, this can take a while!)
2. Download the *macx-g++4.9.zip* file from Studentportalen and unzip to `/usr/local/Cellar/qt/your-qt-version/mkspecs/`
3. (You do not need to do this step until Lab 2) **If** you have an NVIDIA graphics card on your machine, install the CUDA development toolkit by following instructions on NVIDIA - CUDA getting started guide for Mac OSx.

The make spec is written for gcc 4.9. If you already have another version of gcc installed, you will need to make the following adaptations:

1. Change the name of the directory to `macx-g++-your-version`
2. Adapt the environment variables `QMAKE_CXX` and `QMAKE_CC` in `macx-g++-your-version/qmake.conf`
3. Change `Makefile` to use your macx-g++ version
4. Change `libpedsim/Makefile` to use your g++ version

4 Lab Instructions

The objective of this lab is to become familiar with the project code and simple constructs from OpenMP and PThreads.

4.1 Finding Bottlenecks and Parallelizing with PThreads and OpenMP

Familiarize yourself with the code. Take a look at the `README.txt` in order to build and run the project. Parallelize the serial version of the bottleneck. Here you need to implement two versions, one that uses **OpenMP** and one that uses **Pthreads**.

A. Implement the serial version of the tick function in ped_model.cpp, such that in each time step each agent moves.

B. Identify sources of parallelism and implement one version using OpenMP, and one using Pthreads. The three versions (serial, OpenMP and Pthreads) must be interchangeable. Make it possible to easily choose between implementations, either by recompilation or command-line argument.

C. Vary the number of threads employed and generate a plot showing the speedup against the number of threads, using the serial version as baseline. *Hint:* Use the –timing-mode flag in order to run the simulation without GUI.

4.2 Questions

A. What kind of parallelism is exposed in the identified method?

B. How is the workload distributed across the threads?

C. Which number of thread gives you the best results? Why?

D. Which version (OpenMP, Pthreads) gives you better results? Why?

5 Submission Instructions

Submit the code and a **short** report including all answers in **PDF** format to student-portalen.

1. Your code **has** to compile.
2. Document how to choose different versions (serial, Pthreads, and OpenMP implementation).
3. Include a specification of the machine you were running your experiments on.
4. State the question and task number.
5. Include all generated plots.
6. Put everything into a directory and create a zip file and name it team-n-lastname$_1$-lastname$_2$-lastname$_3$.zip, where n is your team number.
7. Upload the zip file on Studentportalen by the corresponding deadline.

6 Acknowledgment

This project includes software from the PEDSIM[4] simulator, a microscopic pedestrian crowd simulation system, that was adapted for the Low-Level Parallel Programming course 2015 at Uppsala University.

[4]http://pedsim.silmaril.org/