Advanced Computing – 2023/2024

MsC in Informatics – ESTiG/IPB

Practical Work 1:
Raytracer Acceleration with Pthreads

Goal: To accelerate a simple raytracer, using the Pthreads programming model.

Preliminary Note:
This work is accompanied by a ZIP archive with a serial implementation (raytracing-serial-1.zip).

Introduction:
"Ray tracing is a rendering technique for generating an image by tracing the path of light as pixels in an image plane and simulating the effects of its encounters with virtual objects. The technique is capable of producing a high degree of visual realism [1]", but it is computationally demanding.

"Ray Tracing in One Weekend" (RTOW) [2] teaches the basics of coding a simple raytracer. It explains the main underlying concepts (consider reading it at least once) and provides source code (it was originally implemented in C++, but there are implementations in many other languages).

Details and Methodology:
In this work you are given a serial implementation of RTOW in C and you are required to develop a parallel version based on the Pthreads programming model, faster than the serial version.

The work involves several stages; some may be executed on your personal computer, others must be executed in the cluster used in the classes, and others may be executed on either systems; make sure you read the specific instructions for each stage regarding the required execution environment.

Note: each group will be told about the specific cluster node(s) it must use (do not use any other node(s)); if your group does not have a working access to the cluster, report the situation ASAP.

a) Compiling and Testing the Serial Code
Note: this stage may be done in your computer or in the cluster frontend.

To compile the serial implementation given, a recent version of `cmake` is needed. If you chose to do the development in your personal computer, make sure `cmake` >= 3.19 is installed. If you opt to develop directly in the cluster frontend, `cmake` is already available in `/share/apps/cmake-3.19.3`; to use it, add `source /share/apps/env_cmake-3.19.3` to the end of your `~/.bashrc` file, then logout and login (needed only once; afterwards, the new `cmake` version will be used by default).

Extract the ZIP file supplied, in the system you choose to develop the work. To compile, do (*):

```
cd raytracing-serial-1
mkdir build
cd build
cmake ..
make
```

You may then experiment running the raytracer with different options:

- see the usage

```
./ray_tracing_one_week -h
```

- render a sample

```
./ray_tracing_one_week --scene random_spheres random_spheres.ppm
```

There are 9 scenes to choose from and each takes a different amount of time to render. This time is determined by the number of samples (rays to cast for each pixel), 1000 by default. You may change this value using the **`-s`** parameter (for instance, `./ray_tracing_one_week -s 10 --scene …`).

To visualize the sample rendered, you may double click on the PPM image or execute:

```
xdg-open random_spheres.ppm
```

If you execute `xdg-open` on your computer and fails because it is not found, you must install the corresponding package (in Debian/Ubuntu, execute `sudo apt install xdg-utils`). In the cluster frontend, `xdg-open` is already available, but in order for the window with the image to be displayed in you personal computer, you must access the frontend with MobaXterm (if connecting from Windows) or with `ssh -X user@172.31.254.5` (if connecting from a Linux terminal).

b) Measuring the Performance of the Serial Version

Note: this stage must be done in the cluster, in the node assigned to your group

Before producing a parallel version, it is necessary to measure the execution time of the serial version with a selected scene, to have a comparison baseline. In this work, the reference scene will be `random_spheres` and the number of rays to cast will be the default (1000), unless otherwise stated. Also, make sure the profiling options are commented and the benchmark options are uncommented (this is the default) in the `CMakeLists.txt` file in the top folder of the project:

```
#SET(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-pthread -O0 -g")
SET(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-pthread -O2")
```

Note: if `CMakeLists.txt` is changed, remove the `build` folder and recompile the code (see (*)).

In the `build` folder, execute the next command three times (waiting 5s between each execution) :

```
time ./ray_tracing_one_week --scene random_spheres random_spheres.ppm
```

In the report present the smallest time (converted to a single number, in seconds) of the 3 runs.

Note: before benchmarking, confirm there is no other CPU-intensive applications running (you can do this with the `top` command); this will ensure the three times measured will all be very similar.

c) Profiling the Serial Version with Valgrind

Note: this stage may be done in your personal computer, or in the cluster.

Modify the `CMakeLists.txt` file to set the correct options for profiling through valgrind:

```
SET(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-pthread -O0 -g")
#SET(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-pthread -O2")
```

Then remove the `build` folder and recompile the code again (see (*)).

Use the Callgrind tool of Valgrind [3] to capture the profiling data, this time using only **10 samples**. Right after, make a backup of the `random_spheres.ppm` file generated, outside the `build` folder. This backup will later be used to control the correctness of the scenes produced during stage e.1).

Next, use kcachegrind [4] to visualize the call graph and identify the serial code hot-spots.

In the report: i) provide the command used to run ray_tracing_one_week.exe through valgrind; ii) provide one or more screenshots of the kcachegrind windows that shows the hot-spots (left panel) and the call graph of `main`; iii) identify the biggest hot-spots and their relative importance (%).

Also, analyze the code of the biggest hot-spots and identify those that seem to be easily parallelizable; in your report, state those functions and explain why they are easily parallelizable.

d) Applying Amdahl's Law

Considering the aggregated weight (%) of the code that you decided to parallelize, apply Amdahl's Law to calculate the theoretical speedup S_T of the parallelization for a number of threads N=1,...,8. Explain and justify, in your report, the value given to the different factors of the Amdahl's Law formula and provide a table like the following with the missing values (?) of the S_T speedups.

N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
S_T	?	?	?	?	?	?	?	?

Table 1 - Theoretical Speedups

Based on the Table 1 values, provide also a table with the corresponding theoretical efficiency E_T :

N	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
E_T *(%)*	?	?	?	?	?	?	?	?

Table 2 - Theoretical Efficiency (in percentage)

Finally, determine the theoretical speedup limit and also present that calculation in your report.

e) Parallelization

e.1) Development

Note: this stage may be done in your personal computer, or in the cluster.

Start by making a copy of the raytracing-serial-1 folder and name it raytracing-pthreads; this is the folder were you will make changes to the original code, and which must be provided for evaluation.

Decide how to divide the work (the hot-spot functions chosen to parallelize) by threads and how to synchronize them. Explain your parallelization strategy in the report. Try to minimize the synchronization points, as they will harm performance. Balance the work by the threads as equally as possible, and allow execution with any number of threads (N), irregardless of the amount of work units available (meaning, if you have more threads than work units, some threads will be idle).

To make development and testing faster, in this stage you may use again only **10 samples**. Also, start with 2 threads and compare the scene produced by that first parallel version with the one produced during profiling (use the `cmp` command for a byte-level comparison); if they match, increment the threads, up to 8, always ensuring the profiling and the parallel scene are the same.

e.2) Benchmarking

Note: this stage must be done in the cluster, in the node assigned to your group.

When ready for benchmarking, set the correct options for that purpose in `CMakeLists.txt`:

```
#SET(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-pthread -O0 -g")
SET(CMAKE_C_FLAGS ${CMAKE_C_FLAGS} "-pthread -O2")
```

Then remove the `build` folder and recompile the code again (see (*)).

In the `build` folder, execute three times (waiting 5s between each execution) the command

```
time ./ray_tracing_one_week --scene random_spheres random_spheres.ppm
```

for each different number of threads N=1,...,8. For each N, register the smallest real time T_R (converted to a single number, in seconds) and provide the following table filled in your report:

N	1	2	3	4	5	6	7	8
T_R								

Table 3 - Real Execution Times of the Parallel Version (in seconds)

Then, calculate the real speedups S_R and present the following table filled in your report:

N	1	2	3	4	5	6	7	8
S_R	--							

Table 4 - Real Speedups of the Parallel Version

Provide also in the report a table with the real efficiency E_R achieved by the parallel version, and the ratio E_R/E_T (this ratio tells how close is the parallel implementation to the Amdahl's Law prediction):

N	1	2	3	4	5	6	7	8
E_R (%)	--	?	?	?	?	?	?	?
E_R/E_t (%)	--	?	?	?	?	?	?	?

Table 5 - Efficiency of the Parallel Version (in %) and closeness of the real to ideal efficiency (in %)

For each table, provide also a companion graph, to make the data easier to understand and interpret.

f) Discussion

Discuss the results achieved. Do they correspond to the predictions ? Are they behind expectations (and if so, what could be the motive(s)) ? Are there other optimizations that could still be done ?

-- / --

Groups:

The work is to be solved by groups of 2 students (no more/no less, unless authorized by a teacher).

Deadline and Submission:

The work must be submitted for evaluation no later than midnight of December 17th, through the Assignments module of the virtual.ipb.pt site, as a single ZIP file containing the report (in PDF - please do not submit the report in any other format) and source code (raytracing-pthreads folder).

Plagiarism:

Plagiarism will not be tolerated and will be firmly handled by the current regulations of ESTiG/IPB.

References:

[1] https://en.wikipedia.org/wiki/Ray_tracing_(graphics)
[2] https://raytracing.github.io/books/RayTracingInOneWeekend.html
[3] https://valgrind.org/
[4] https://kcachegrind.github.io/