GPU Programming Assignment

Cristian Di Pietrantonio
cdipietrantonio@pawsey.org.au

Maciej Cytowski
Maciej.Cytowski@csiro.au

Ilkhom Abdurakhmanov
Ilkhom.Abdurakhmanov@csiro.au

1 Preliminaries

In this assignment you are asked to write programs that run on GPUs using HIP and OpenMP.

1.1 Using the GPUs of Setonix

Setonix is the latest Pawsey supercomputer whose computational power mostly comes from its GPUs. There are 192 GPU nodes, each with four AMD MI250X GPUs. A MI250X GPU is composed of two Graphics Compute Dies (GCDs). In practice, each node has 8 GPUs as seen by the software environment.

A Slurm *reservation* has been created, named `CurtinHPCcourse`, that grants course attendees exclusive access to 2 `gpu` nodes, for a total of 16 GPUs. In addition to those nodes you can still submit jobs to the `gpu` and `gpu-dev` partitions without reservation, using other nodes as they become available (which should not take much time). If you encounter an issue, contact one of the lecturers via email.

The login nodes of Setonix are available at `setonix.pawsey.org.au`; username and project code are the same as the ones used in previous lectures. Use any SSH client you prefer.

1.2 Assignment structure

The assignment archive, `gpu-assignment.zip`, is structured as follow:

- `assignment.pdf`: this file, containing the description of the assignment;
- `ex1-gol-hip/`: directory for the first exercise;
- `ex2-gol-gpu-directives/`: directory for the the second exercise;

- `report.pdf`, `report.tex`: sample assignment report.

Grades for this assignment will range from 0 to 1. A scaling factor will be applied to bring the grades in the desired range.

1.3 What to submit

Submit the two directories `ex1-gol-hip` and `ex2-gol-gpu-directives` containing all the modified files representing the solutions to the assignment. In addition you must a PDF report summarising your work, key findings and conclusions. A sample of PDF report and associated LaTeX source can be found in the assignment package.

2 Exercise 1: the game of life using HIP (0.5 points total)

This exercise will make you familiar with the process of adapting an existing serial CPU code to make it run on a GPU. This process is often called *porting*. The program, which now you should be familiar with, implements The Game of Life, described in Section 1 of the OpenMP assignment.

For this exercise, you will work in the `ex1-gol-hip` directory. It contains

- a `Makefile` file to assist you in compiling both the CPU code and the GPU one (that does not exist yet).
- a `src/` directory to collect source files for this exercise. Files implementing the CPU program in C are already there.
- a `LICENSE` file, containing the license which this material is distributed under.

The very first step is to compile the CPU code. Execute the `make` command form within the root directory to do so. It will generate the executable `bin/game_of_life`, but also will print an error regarding the absence of a source file. It happens because the `make` command tries to build the source code for the GPU version, but you have not created it yet.

To start with the assignment, create the missing file. It is easier if you start working with a copy of the serial version instead of an empty file:

```
cp src/game_of_life.c src/game_of_life_hip.cpp
```

You DO NOT port code in `common.*`.

The executable, both for the CPU and GPU versions, requires at least two positional arguments: n and m, the size of the grid. A third argument specifies the number of iterations to play; by default it is ten.

2.1 Task 1 - GPU porting (0.35 points)

Adapt the serial version of the code to run the most computational intensive part on a GPU using HIP. Evaluation criteria include:

- correctness: most of the computation runs on GPU and produces the correct result. To check this, you should use the function that already implements game of life on CPU.

  Add at the top of the `src/game_of_life_hip.cpp` file the following lines:

  ```
  #define INCLUDE_CPU_VERSION
  #include "game_of_life.c"
  ```

  Try different grid sizes and do not limit yourself to square grids only.

- code safety: the code reports if something went wrong so that the user knows the output is not valid.

2.2 Task 2 - performance evaluation (0.15 points)

The *speedup* measure compares the execution time of a parallel code against its serial implementation. It is defined as $S = \frac{T_s}{T_p}$, where T_p is the execution time of the parallel code and T_s is the execution time of the serial version. The greater S, the better. In this task you are asked to compute the speedup of your GPU code against the original one, for inputs of various size.

To do so you must measure execution times first. The serial code has already timers in place. On the GPU code you can measure the total execution time (actual GPU computation plus any other HIP API call and CPU code to support it) in a similar way. It is interesting and useful to compute what percentage of the total execution time represents the HIP kernel execution alone as the input size increases.

Fix the number of steps to 100. Then, run both the CPU and GPU implementations of The Game of Life for (at least) the following inputs:

1. $n = m = 10$
2. $n = m = 100$
3. $n = m = 1000$
4. $n = m = 10000$
5. $n = 1000$, $m = 10000$

For each of the above inputs, report:

1. T_s, the execution time of the CPU code,
2. T_p, the execution time of the GPU implementation,

3. T_s/T_p, the speedup,
4. t_k, the percentage of T_p spent executing only kernel code.

You should provide the information in a tabular format where each row presents the requested information for a given input. You are encouraged to also plot a curve for each measure as a function of the input size.

2.3 Further readings

The best source of information about GPU programming is the CUDA C programming guide, followed closely by the HIP documentation.

3 Exercise 2: GPU implementation of the game of life using OpenMP (0.5 points total)

This exercise will make you familiar with the process of adapting an existing serial CPU code to make it run on a GPU with directive based GPU programming model OpenMP. This process is often called *porting*. The program, which now you should be familiar with, implements The Game of Life, described in Section 1 of the OpenMP assignment.

For this exercise, you will work in the `ex2-gol-gpu-directives/` directory. It contains `openmp/` directory with the following structure:

- a `Makefile` file to assist you in compiling both the CPU code and the GPU one (that does not exist yet) both for C and Fortran;
- a `src/` directory to collect source files for this exercise.;
- a `LICENSE` file, containing the license which this material is distributed under.

For this exercise you will need to choose between the following options:

- C + OpenMP implementation
- Fortran + OpenMP implementation

Please focus on implementation and optimisation only for single chosen option from the above list.

To compile the project type `make` command. It will require an appropriate compiler module, i.e. `module load rocm/5.2.3; module load craype-accel-amd-gfx90a`.

The `make` command will compile and build CPU and GPU versions, although the GPU versions initially do not contain any directives.

To start the assignment edit appropriate GPU source file, e.g. if you have chosen to work on Fortran + OpenMP implementation edit

$$openmp/src/02\_gol\_gpu\_openmp\_fort.f90$$

file, compile, execute and compare performance.

3.1 Task 2.1 - GPU porting (0.35 points)

Adapt the serial version of the code to run the most computational intensive part on a GPU using OpenMP. You DON'T HAVE TO port code in `common.*`. Evaluation criteria include correctness and portability. The code changes should be minimal and the GPU code should also execute correctly on CPUs when compiled without directives support.

For data transfers, please use `omp target enter data`, `omp target exit data`. They have the similar clauses as regular `data` directives, but allow you to implement data movement for the remainder of the program, or until a matching `exit data` directive deallocates the data. The regular data directives work only for a structured block of the program. With enter, exit data directives you can schedule data transfers from various functions (not only the one containing the GPU kernel).

For parallel loops, please make sure that all arrays used in GPU kernels have appropriate data types, some of the arrays need to be marked as `private` for performance and correctness.

3.2 Task 2.2 - performance evaluation (0.15 points)

The *speedup* measure compares the execution time of a parallel code against its serial implementation. It is defined as $S = \frac{T_s}{T_p}$, where T_p is the execution time of the parallel code and T_s is the execution time of the serial version. The greater S, the better. In this task you are asked to compute the speedup of your GPU code against the original one, for inputs of various size.

To do so, you must measure execution times first. The serial code has already timers in place.

Fix the number of steps to 100. Then, run both the CPU and GPU implementations of The Game of Life for (at least) the following inputs:

1. $n = m = 10$
2. $n = m = 100$
3. $n = m = 1000$
4. $n = m = 10000$
5. $n = 1000$, $m = 10000$

For each of the above inputs, report:

1. T_s, the execution time of the CPU code,
2. T_p, the execution time of the GPU implementation,
3. T_s/T_p, the speedup.

You should provide the information in a tabular format where each row presents the requested information for a given input. You are encouraged to also plot a curve for each measure as a function of the input size.

3.3 Further readings

Two documents might be helpful for checking syntax and meaning of various OpenMP directives and clauses:

- OpenMP Reference Guide

Please note that above listed reference guides are for the most recent OpenMP standards version. Support for different standard versions might differ between compilers.