F20DP Distributed and Parallel Technologies.
Coursework 2: Sum Totient in SYCL and C+MPI.

Summary

In Coursework 1 you used and compared an *imperative* parallel programming model (OpenMP) with a *functional* parallel programming model (parallel Haskell) for programming CPUs. In Coursework 2 (CWK2) your task is to use lower level programming models that give you more explicit control of memory management and data transfer for GPUs (SYCL), and message passing for multiple multicore CPUs (C+MPI).

The parallel machines are nodes in the Robotarium high-performance cluster at Heriot-Watt.

Robotarium cluster website: `https://ecr-cluster.github.io`

Plagiarism policy

In your group pair you can share ideas for your parallel code implementation and work together on the report text. However you must not plagiarise text from the internet or from external publication sources. You must also not work with other coursework groups. **Your report will be checked for plagiarism, both against external sources and against other coursework submissions**. *If any is detected your submission will be treated as an academic misconduct case*. Please read carefully the university's plagiarism policy:

`https://www.hw.ac.uk/uk/students/studies/examinations/plagiarism.htm`
`https://www.hw.ac.uk/uk/students/doc/discguidelines.pdf`

Use of AI systems that generate text, e.g. **ChatGPT**, is **not permitted** for this coursework. The university's guidance *"Introduction to AI content creation tools and university study: Student Guide"* states:

> *"Submitting work which is not your own is plagiarism i.e. cheating. If you were to use an AI content tool to draft/write your assessment, then the assessment would not be your own work."*

`https://tinyurl.com/3vtnsk2k`

Submission instructions

There are two parts to the submission process.

1. **Your source code implementation.** Push your code to the Heriot-Watt GitLab Student server.
   - **One** student in each coursework pair should for this GitLab project for your submission:
     `https://gitlab-student.macs.hw.ac.uk/f20dp/f20dp-cwk2`
     Do not rename the project when forking it. The student who forked the project should add their coursework partner as a *Developer* member of this forked project (Settings → Members when looking at the project page on GitLab).
   - Include the URL of your coursework's GitLab project in your PDF report.
2. **A report**. A PDF should be submitted on Canvas.
3. **Statement of authorship**. Your PDF should include a *"Statement of Authorship"* section to declare the division of work in your coursework pair. It should state (1) who implemented and evaluated each technology, (2) who wrote the answers to each question and (3) who produced the performance graphs.
4. **Student authorship**. Submit a declaration of student authorship on Canvas.

Marks will be lost if your report does not follow the structure outlined below or does not contain all of the specified results. The deliverable is due 3:30pm UK time on 5^{th} April 2024.

Learning Objectives of coursework 2

The Learning Outcomes (LO) of this F20DP course are listed at the end of this document.
The learning objectives of coursework 2 is tied to many of these Learning Outcomes:

- Imperative (SYCL, C+MPI) parallel programming skills (LO2, LO4, LO6, L08).
- Ability to benchmark/measure parallel performance: *efficiency*, *scalability* and *speedup* (LO5).
- Compare the message passing and SYCL parallel programming models (LO1, L07, LO10).

The Algorithm to parallelise

The program that should be parallelised is the computation of the *sum of Euler totient computations* over a range of integer values.

The *Euler totient function* computes, for a given integer n, the number of positive integers smaller than n and relatively prime to n, i.e. $\Phi(n) \equiv \mid \{m \mid m \in \{1,\ldots,n-1\} \wedge m \perp n\} \mid$. Two numbers m and n are relatively prime, if the only integer number that divides both is 1. To test this, it is sufficient to establish that their greatest common divisor is 1, i.e. $m \perp n \equiv \gcd\ m\ n = 1$. Thus, the task for this program is: for a given integer n, compute $\Sigma_{i=1}^{n}\Phi(i)$.

A video has been created to describe the Sum Totient algorithm to be implemented. The video also gives suggestions on how to measure, plot and analyse your performance results:

`https://web.microsoftstream.com/video/39738d62-57ad-4669-8d97-5de387b81471`

Mathematical properties of the Euler's totient function $\Phi(n)$ are described in this video by Khan Academy:

`https://www.youtube.com/watch?v=qa_hksAzpSg`

Sequential C reference version

The following C code is a direct implementation of the above specification, as a starting point for your SYCL and C+MPI implementations. Your job is to adapt the reference C implementation into a SYCL program to execute on a GPU, and adapt the same C implementation into a C+MPI program for distributed-memory parallelism on multiple multicore CPUs.

`https://gitlab-student.macs.hw.ac.uk/f20dp/f20dp-totient-range`

Organisation

The **assessed coursework work is to be carried out in pairs**, and you should choose your own partner. If you do not have a partner, contact Rob Stewart (R.Stewart@hw.ac.uk) and he will pair people together.

Each member of the team chooses either SYCL or C+MPI, and implements a parallel version of *totient range* in this technology and evaluates the performance of this parallel version:

- SYCL for GPUs – develop two versions using the SYCL memory and execution models in different ways;
- C+MPI for distributed memory parallelism across a cluster of CPUs.

Together you can share ideas about how to parallelise the code and how to tune the parallel performance. However, it needs to be clearly stated who implemented the parallel version using which technology. *Together* you should prepare a comparative report using the structure below. It is relatively easy to produce a simple parallelisation of both programs, however *marks are available for thoughtful sequential and parallel performance tuning* (Question 6).

Tools that can help you, and have been discussed on these slides are `gprof` and `cachegrind` for profiling sequential C functions. You are welcome to use other tools for profiling your C+MPI and SYCL implementations, if you find them useful. In each case you should motivate your choice of tool and reflect how useful it was for tuning performance. For plotting parallel performance results `gnuplot`, `R`, `Excel` or one of many Python libraries are recommended.

Structure of the report

1. (4 points) Section 1: **Introduction**

   This should give a short summary of the task to implement and parallelise, describe the software and hardware environments it is performed in, the parallel technologies used, and the learning objectives of this coursework.

2. (4 points) Section 2: **Sequential Performance Measurements**

   Run the sequential C version of the programs and analyse the performance. For an extra mark use a profiling tool such as those mentioned above. Discuss the sequential performance of, and possible improvements to, these programs. Of interest are in particular code on the critical path (hotspots) in the program. *max 1 A4 page*

3. Section 3: **Comparative Parallel Performance Measurements**

   You should measure and record the following results in numbered sections of your report. The measurements are based on these inputs:

   - DS1: calculating the sum of totients between 1 and 15000.
   - DS2: calculating the sum of totients between 1 and 30000.
   - DS3: calculating the sum of totients between 1 and 100000.

   For each of these inputs, measure:

   - SEQ: the sequential runtime (C only) on one compute nodes (computers) on the Robotarium cluster.
   - MPI <n>: the runtime of the C+MPI implementation using <n> MPI processes, where n varies between 1, 2, 4, 8, ..., 192 running on **3 nodes** of the cluster. Each MPI node runs on a 64 core CPU, so each node should run up to 64 MPI processes ($64 \times 3 = 192$).
   - SYCL: the runtime on a Nvidia K20 GPU on one of the cluster nodes using different SYCL configurations e.g. workgroup sizes, dimensions in the NDRange and use of different memory regions.

   Runtime measurements should be repeated 5 times each.

   (a) (6 points) Section 3a: **Runtime Table**

   Present your measurements in a table containing the median averages of the runtimes for SYCL and C+MPI as well as difference between this median and the slowest and fastest runtimes. Use a suitable unit to ensure readability.

   (b) (2 points) Section 3b: **Runtime Graphs**

   Present your runtimes in **three** graphs, one for each problem size. For C+MPI, The x-axis should be the parallel degree (MPI processes), the y-axis runtime. For SYCL, you should plot the runtimes for the different SYCL configurations you have implemented – possibly as a bar graph, each running on the same number of cores on a GPU.

   (c) (2 points) Section 3c: **Speedups (C+MPI only)**

   Plot graphs for C+MPI, showing speedup graphs corresponding to the runtime results for DS1, DS2 and DS3 with an x-axis being MPI processes from 1 to 192. Include the ideal speedup as a line as well. Also show a table with the raw data for C+MPI: the sequential performance and the average parallel runtimes using different numbers of a MPI processes across 1-3 compute nodes.

   (d) (3 points) Section 3d: **Efficiency**

   Plot the corresponding parallelism efficiency graph for C+MPI. Analyse the efficiency performance of your C+MPI and SYCL implementations, speculate on the reasons for the efficiency performance.

   (e) (2 points) Section 3e: **Hardware Utilisation**

   Figure out how many operations on `long` values are being performed for the three different inputs for both the C+MPI and SYCL implementations. Then compute the number of operations per second that you achieved in your implementations for the different inputs and for the fixed number of GPU cores (SYCL) and on the cluster of CPUs (C+MPI). Plot these findings as graphs and explain how you obtained the number of operations.

(f) (2 points) Section 3f: **Compare SYCL Implementations**

Describe your two SYCL implementations. Explain how they differ in terms of how you have mapped the Totient Range algorithm to the SYCL memory and execution models in different ways.

(g) (5 points) Section 3g: **Discussion**

A discussion of the comparative performance of scalability, efficiency and speedups for your parallel implementations. Try to identify a suitable platform-independent measure of performance and derive those figures from your experiments. Discuss the impact of the distributed-memory model of C+MPI and the SYCL memory model respectively, and how these models might explain your results. Also discuss how Amdahl's law (see lecture 1) impacts parallel performance of the Totient Range algorithm. *max 1 A4 page*

4. (4 points) Section 4: **Programming Model Comparison**

   An evaluation of the C+MPI and SYCL parallel programming models, specifically for implementing the Totient Range application. *Your answer for this question should be written jointly by both group members.* You should indicate any challenges you encountered in constructing and parallelising your programs and discuss situations/algorithms where each technology may usefully be applied.

5. (20 points) **Appendix A and B**

   For each parallel implementation, the appendix should include a GitLab URL for your parallel Totient programs. Each implementation should clearly labelled with the single author's name in the report and in the GitLab source code as a comment. Include a paragraph in the appendix, and possibly diagram(s), identifying the *parallel paradigm* used, and *performance tuning approaches* used.

6. (4 points) Additional marks are available for thoughtful sequential and parallel performance profiling and tuning throughout the report.

7. (4 points) Additional marks are available if you can reflect on how to use *algorithmic* knowledge of the $\Sigma_{i=1}^{n}\Phi(i)$ computation, with the goal of creating parallel tasks of approximately equivalent computation size in your SYCL and C+MPI programs. Hint: try `runBenchmark()` in `TotientRange.c` and design a partitioning and agglomeration strategy to shorten runtimes further.

Your total coursework mark is calculated as follows:

Question:	1	2	3	4	5	6	7	Total
Points:	4	4	22	4	20	4	4	62

Notes

- Complete the SYCL and C+MPI labs before starting the coursework.
- Graphs and tables must have appropriate captions, and the axes must have appropriate labels.
- You should use your own individual Robotarium accounts to login to the head node and deploy jobs to the compute nodes.
- The Robotarium cluster uses a batch-job system to give you exclusive access to the cluster when you need it for measurements. Follow this link to familiarise yourself with the batch-job system.

Learning Outcomes of this Course

The course aims and learning outcomes of this course are available course descriptor online:
`https://curriculum.hw.ac.uk/coursedetails/F20DP?termcode=202324`

The Learning Outcomes of this F20DP course are:

LO1 Understanding of foundational concepts of distributed and parallel software.

LO2 Knowledge of contemporary techniques for constructing practical distributed and parallel systems using both declarative and imperative languages.

LO3 Parallel performance tuning using appropriate tools and methodologies.

LO4 Appreciation of relationship between imperative and declarative models of parallelism.

LO5 Critically analyse parallel and distributed problems.

LO6 Generate, interpret and evaluate parallel performance graphs.

LO7 Develop original and creative parallel problem solutions.

LO8 Demonstrate reflection on core concepts and technologies, e.g. understanding of applicability of, and limitations to, parallel and distributed systems.