COMP3001
Parallel Programming

20 CREDIT MODULE

ASSESSMENT: 100% Coursework **W1: 30% Set Exercises**
W2: 70% Report

MODULE LEADER: Dr. Vasilios Kelefouras

MODULE AIMS

This module aims on giving students the practical understanding and skills needed to parallelize and optimize real world programs on modern processors. Students will learn about state of the art high performance computing (HPC) technologies, hardware computer architectures and parallel programming models. This module will give students real world coding experience and hands on project work.

ASSESSED LEARNING OUTCOMES (ALO):

1. Understand state of the art high performance computing technologies and parallel programming models
2. Write performance efficient code to speedup programs on modern multi-core processors
3. Design and implement performance efficient programs on GPUs.

Overview

This document contains all the necessary information pertaining to the assessment of *COMP3001 Parallel Programming*. The module is assessed via **100% coursework**, across two elements: *30% Set Exercises* and *70% Report*.

The sections that follow will detail the assessment tasks that are to be undertaken. The submission and expected feedback dates are presented in Table 1. All assessments are to be submitted electronically via the respective DLE module pages before the stated deadlines.

	Submission Deadline	Feedback
Set Exercises (30%)	**11th of Nov. 2024 15.00**	Within 20 working days
Report (70%)	**7th of Jan. 2025 15.00**	Within 20 working days

Table 1: Assessment Deadlines

All assessments will be introduced in class to provide further clarity over what is expected and how you can access support and formative feedback prior to submission. Whilst the assessment information is provided at the start of the module, it is not necessarily expected you will start this immediately – as you will often not have sufficient understanding of the topic. The module leader will provide guidance in this respect.

Assessment 1: Set Exercises (30%)

The set of exercises consists of the following two questions. The source code is provided in this link https://github.com/kelefouras/comp3001/tree/23_24/comp3001-master/24_25_coursework .

Question 1 – Performance Profiling [40 Marks]

In this question you need to apply performance profiling to a C program. Please note that you will need to utilize several tools available in the Linux operating system. Download the '*Question1.c*' source code file from the module's GitHub repository and compile it with the command provided on the top of the file; if you get an error then you need to use '*Question1_alternative_file.c*' instead.

Do the following tasks:

1A. Use gprof profiler to identify the most computationally expensive routines; provide the percentages of the running time of these functions. [5 marks]

1B. Find the FLOPs values achieved by the most computationally expensive routines. You need to provide a) the FLOPs value for each routine separately, b) the lines of code that calculate the FLOPs values, c) the system's information (CPU and DDR specs and OS). [15 marks]

Tip. To get an accurate FLOPs value, you need an accurate execution time value; to do so, the routine must run for at least a couple of seconds. It is suggested to repeat the experiment several times and get the average execution time / FLOPs value.

Marks	**0-1 marks**	**2-6**	**7-15**
Marking Criteria	FLOPs values are not provided.	FLOPs values are provided but the code that calculates FLOPs is not right.	The FLOPs values are correctly calculated and provided for all the routines.

1C. Explain the results obtained in Step B. Additionally, provide a detailed theoretical discussion on the optimizations you would consider applying to this program to enhance its performance. Please note that you are not required to implement these optimizations; just elaborate on the strategies and their potential impact. The answer should be provided in a single paragraph (up to 250 words). Long answers will not get all the marks. [10 Marks]

Marks	**0-1 marks**	**2-6**	**7-10**

Marking Criteria	Little or no analysis. Little understanding of the subject. Almost no explanation is given	Answers are given at a good level of detail and are explained clearly. The student has explained why the routines provide high/low FLOPs. The theoretical peak FLOPs value on this PC is provided. The student has suggested the right optimizations.	An outstanding analysis is provided. The student has explained why the measured FLOPs values are lower than the theoretical peak ones. The student has explained why the suggested optimizations are efficient.

1D. Use Cachegrind tool (Valgrind) to measure the number of L3 cache misses of the program above; note that running valgrind will take about 3 minutes in this case. Theoretically justify the result. [10 Marks]

Marks	**0-1**	**2-5**	**6-10**
Marking Criteria	The student has not well measured the number of misses.	Little or no analysis. Little understanding of the subject.	An outstanding analysis is provided. The student justifies the number of misses measured, by analyzing the data access patterns and the cache size.

Question 2 – Speeding up a linear algebra application on your PC [60 Marks]

Drawing upon all the optimization techniques that you have learned so far in this module, you will speedup the '*slow_routine()*' located in '*gemver_default.c*' file. The main optimization needs to be applied is vectorization, by using ***x86-64 SSE/SSE2/SSE4/AVX/AVX2*** C/C++ intrinsics. All the C/C++ x86-64 intrinsics are provided in the following link: https://software.intel.com/sites/landingpage/IntrinsicsGuide/ . Provide the FLOPs value being achieved before and after optimization.

The marking scheme is as follows **[60 marks]**:

Marks	**0-6 marks**	**7-19 marks**	**20-25**	**26-45 marks**	**46-60 marks**

Marking criteria	The student has not provided appropriate vectorized code. The routine does not generate the right output. The implementation contains bad practice.	Not all the loop kernels are efficiently vectorized. The implementation does not work for any input size.	All the loop kernels are efficiently vectorized. The flops value being achieved before and after the optimization is provided.	Additional optimizations are well applied like register blocking. Padding code is provided.	An outstanding implementation is provided achieving improved performance. AVX is used and not SSE. Extra optimizations are applied, e.g., loop tiling, loop merge.

***Hints**: There are many different ways to implement this routine and each solution includes different intrinsics. However, a valid solution exists by using the following SSE instructions: _mm_load_ps, _mm_load_ps1, _mm_load_ss, _mm_add_ps, _mm_mul_ps, _mm_store_ps, _mm_store_ss, _mm_set1_ps, _mm_hadd_ps. Note, however, that the highest marks will be awarded to those who use AVX and not SSE.*

Submission Details

The submission will be done via the submission link on the COMP3001 DLE page. You should submit **two files**:

1. A **.pdf** file containing the solution to question 1.
2. The **.c or .cpp file** containing the solution to question 2.

Do **not** upload any .zip files.

Assessment 2: Report (70%)

The source code is provided in this link https://github.com/kelefouras/comp3001/tree/23_24/comp3001-master/24_25_coursework .

Question 1 – Parallelize a software application using OpenMP framework [30 Marks]

Your task is to parallelize a C software application by using OpenMP framework. You can find the code on the module's GitHub repository. The marking criteria are as follows.

Marks	0-6	7-14	15-22	23-30
Marking Criteria	The student has not used the OpenMP annotations appropriately. The code includes bad practice.	The student has used the OpenMP annotations appropriately, but just for a few loop kernels. He/she has not used the OpenMP annotations to all the loop kernels that can be parallelized*.	The student has used the OpenMP annotations appropriately to all the loop kernels that can be parallelized*. However, the code delivered includes either multi-threaded code only or vectorized code only, not both.	The student has used the OpenMP annotations appropriately for all the loop kernels that can be parallelized*. The code delivered contains both multi-threaded and vectorized code.

* *If a loop kernel cannot be parallelized or vectorized in its current form, then you do NOT have to take actions against this problem, e.g., the loop kernel in line 178 cannot be vectorized by using OpenMP (in its current form).*

***Hint**: The Openmp clauses that need to be used are the following: omp for, reduction, private, shared, omp simd, omp for simd, aligned. The aligned clause requires that the_mm_malloc should be used instead of malloc. Consider the fact that wrong parallel code might generate the right output sometimes.*

Question 2 – CPU vs GPU performance evaluation and analysis [20 Marks]

You are provided with four different implementations of the Matrix-Matrix Multiplication (MMM) algorithm: two for the CPU and two for the GPU (located in the "question 2" folder). Your task is to measure and evaluate

the performance of these routines. A complete code template, including routines for array initialization, correctness checking, and more, is available in the lab sessions' GitHub repository. As part of this task, you are expected to locate and reuse the relevant code provided during the lab sessions.

1. **Measure the FLOPs values of the four implementations. Make a single graph showing '*FLOPs vs N*'** for the four routines (the FLOPs values will be shown in the y-axis while the N values will be shown in the x-axis). Consider using a logarithmic scale in the y-axis. Use square matrices of size NxN, where *N=[64,128,256,512,1024,2048,4096]*. The two GPU implementations will run on the lab's GPUs. The two CPU implementations will run on any PC; however, it is strongly recommended to run them on a non-virtual machine environment, e.g., consider using the PCs in SMB201. If you use VS, make sure you run the codes under the 'release mode'. If you use Linux, compile the codes by using '-O3' option. When measuring the execution time on the GPUs, include the time needed to transfer the arrays from the CPU to the GPU and vice versa. Do not forget to submit your code. **[10 marks]**

Marks	**0-5**	**6-10**
Marking Criteria	The student has either provided an incomplete figure (where some values are missing) or some values are wrong. The experimental procedure is not well applied. The code submitted is not right.	The student has provided a figure where the FLOP values are shown clearly. All the FLOPs values are correctly measured. The CPU/GPU specs are provided. The code submitted is appropriate.

2. Explain the results you found in task1**. [10 marks]**

Marks	**0-2 marks**	**3-7**	**8-10**
Marking Criteria	Little or no analysis. Little understanding of the subject. Almost no explanation is given	Answers are given at an appropriate level of detail and are explained clearly. The student has explained how the FLOP values are affected by the input size and why, on both the CPU and the GPU	An outstanding analysis is provided. The text provided is well written.

Question 3 – Parallelize a software routine by using CUDA [20 Marks]

You are provided with a .cu file (module's GitHub repository). Your task is to parallelize the 'default()' routine using CUDA. Provide the FLOPS value achieved. The marking scheme is as follows: **[20 marks]**

Marks	0-2	3-5	6-9	10-20
Marking Criteria	The student has not provided a parallel implementation that generates the right output.	The student has provided a parallel implementation that generates the right output but contains bad practice.	The student has provided a parallel implementation that generates the right output. The code does not include bad practice. Performance is limited. Shared memory is not used. FLOPS value is provided.	The student has provided an outstanding implementation of high performance. The shared memory is used. FLOPS value is provided.

Question 4 – Optimize an image processing application [30 Marks]

Drawing upon the optimization techniques that you have learned in this module, you will speed up an image processing application. The code is provided on the module's GitHub repository. The routine to be optimized is the 'Sobel()', which is located in the canny.c file. Do not optimize any other routines or block of code. Provide the speedup value achieved. Please note that there is no single solution. **[30 marks]**
The marking criteria are as follows:

Marks	0-3	4-9	10-20	21-30
Marking Criteria	The speedup is low/marginal. The student has either not applied the appropriate optimizations or has not applied them efficiently.	The student has just applied some 'easy to implement' optimizations like parallelization and strength reduction.	The student has provided an efficient implementation. Optimizations include some (but not all) of the following: parallelization, SSE/AVX x86-64 intrinsics, strength reduction, register blocking, reducing the number of arithmetical and/or Load/Store instructions.	The student has applied all the optimizations mentioned in the second column, in an efficient way. The student has provided the right SSE/AVX intrinsics and has applied them the right way.

Tips:

1. *Try to understand how the algorithm works.*
2. *The mask elements (GxMask, GyMask) contain constant values.*
3. *Before you apply vectorization, fully unroll the two innermost loops.*

4. *There are many different ways to implement this routine using SSE/AVX intrinsics and each solution includes different intrinsics. However, a valid solution exists using the instructions hereafter.*

- *If using AVX intrinsics:*
  - *_mm256_loadu_si256()*
  - *_mm256_maddubs_epi16()*
  - *_mm256_add_epi16()*
  - *_mm256_hadd_epi16()*
  - *_mm256_extract_epi16()*
  - *_mm256_set_epi8()*
- *If using SSE instrinsics:*
  - *_mm_loadu_si128()*
  - *_mm_maddubs_epi16()*
  - *_mm_add_epi16()*
  - *_mm_hadd_epi16()*
  - *_mm_extract_epi16()*
  - *_mm_set_epi8()*

5. *Make sure that the load instructions do not exceed the array bounds. Remember that the '_mm256_loadu_si256((__m256i *) &A[i][j])' instruction reads 256bits of data starting from A[i][j], or equivalently 32 char elements.*
6. *The application of register blocking to vectorized code is the same as to non-vectorized code. The only difference is that instead of using 32bit registers, you are using 256bit registers.*

Submission Details

The submission will be done via the submission link on the COMP3001 DLE page. You will submit **four files**:

- q1**.c** or q1**.cpp** file for question 1
- q2**.pdf, q2.c and q2.cu** files containing the answers for question 2
- q3**.cu** file for question 3
- The modified **canny.c or canny.cpp** file containing the solution of question 4

Do **not** upload any .zip files. Do **not** upload the whole Visual Studio Project.

General Guidance

Extenuating Circumstances

There may be a time during this module where you experience a serious situation which has a significant impact on your ability to complete the assessments. The definition of these can be found in the University Policy on Extenuating Circumstances here:
https://www.plymouth.ac.uk/uploads/production/document/path/15/15317/Extenuating_Circumstances_Policy_and_Procedures.pdf

Plagiarism

All of your work must be of your own words. You must use references for your sources, however you acquire them. Where you wish to use quotations, these must be a very minor part of your overall work.

To copy another person's work is viewed as plagiarism and is not allowed. Any issues of plagiarism and any form of academic dishonesty are treated very seriously. All your work must be your own and other sources must be identified as being theirs, not yours. The copying of another persons' work could result in a penalty being invoked.

Further information on plagiarism policy can be found here:

Plagiarism: https://www.plymouth.ac.uk/student-life/your-studies/essential-information/regulations/plagiarism

Examination Offences: https://www.plymouth.ac.uk/student-life/your-studies/essential-information/exams/exam-rules-and-regulations/examination-offences

Turnitin (http://www.turnitinuk.com/) is an Internet-based 'originality checking tool' which allows documents to be compared with content on the Internet, in journals and in an archive of previously submitted works. It can help to detect unintentional or deliberate plagiarism.

It is a formative tool that makes it easy for students to review their citations and referencing as an aid to learning good academic practice. Turnitin produces an 'originality report' to help guide you. To learn more about Turnitin go to:
https://guides.turnitin.com/01_Manuals_and_Guides/Student/Student_User_Manual

Referencing

The University of Plymouth Library has produced an online support referencing guide which is available here: http://plymouth.libguides.com/referencing.

Another recommended referencing resource is Cite Them Right Online; this is an online resource which provides you with specific guidance about how to reference lots of different types of materials.

The Learn Higher Network has also provided a number of documents to support students with referencing:
References and Bibliographies Booklet:
http://www.learnhigher.ac.uk/writing-for-university/referencing/references-and-bibliographies-booklet/
Checking your assignments' references:
http://www.learnhigher.ac.uk/writing-for-university/academic-writing/checking-your-assigments-references/