Parallel Programming (GIA) Practical project - laboratory

Anna Sikora
Eduardo Cesar
Carles Carrillo
Computer Architecture and Operating Systems
Universitat Autònoma de Barcelona
Cerdanyola, Spain
{Anna.Sikora, Eduardo.Cesar, Carles.Carrillo@uab}@uab.cat

***Abstract*—The document presents the lab work we will carry out in the Parallel Programming subject during this semester. With the dual objective of putting into practice the knowledge acquired in this subject and of acquiring habits of autonomous work, the laboratory practices will focus on the development of a single project with different approaches.**

We propose a set of possible milestones (implementation of 2 parallel functional versions of the program we provide you) and deadlines (deliverables) in which you will have to deliver parts of the work done. In this paper we introduce a use case of the k-means algorithm, in particular, pixel clustering to reduce the number of different colors in an image.

***Index Terms*—High Performance Computing, parallel applications, OpenMP, MPI CUDA, machine learning, supervised clustering.**

I. Introduction

With the dual objective of putting into practice the knowledge acquired in this subject and of acquiring habits of autonomous work, the laboratory practices will focus on the development of an open project for the parallelization of a real program. In this project, we propose a set of possible goals (implementation of functional parallel versions of the program provided) and deadlines (deliveries) in which you will have to submit parts of the work done.

The necessary planning to achieve these goals and even deciding which goals you want to achieve is your responsibility.

In Section II of this document, we provide an introduction to the machine learning algorithm we will use, highlighting the most important features of the original code provided to you that you will need to parallelize. In Section III, we describe the project goals, that is, the parallelizations we ask of you, and provide some general guidance and recommendations. Finally, in Section IV, the evaluation criteria are indicated, as well as how and when to make the deliveries.

II. Description of the problem

The code we provide corresponds to the implementation in the C language of the k-means algorithm adapted for images. The k-means algorithm is one possible implementation of clustering algorithms that belongs to the category of unsupervised machine learning algorithms. Therefore, the data we use as input will be grouped according to certain patterns that we, as programmers, will define. A classic and widely used example in data science is the clustering of a company's customers to design different marketing actions for each group.

A. *K-means*

The K-means algorithm is iterative and is designed to partition a dataset into a user-defined number of clusters. This number of clusters is defined as K. For each cluster, we define the centroid as the representative point of the cluster. This point is recalculated at each iteration with the mean of the data assigned to the cluster in that iteration (mean computation). Hence, we get the expression K-means.

The first step to perform K-means is to define the number of clusters K. Subsequently, the centroids of each cluster must be initialized. Random initializations, uniform distributions, or initializations with fixed values can be used. Depending on the initialization, different results can be obtained since it is a sensitive algorithm to initialization, meaning it does not always converge to the same state.

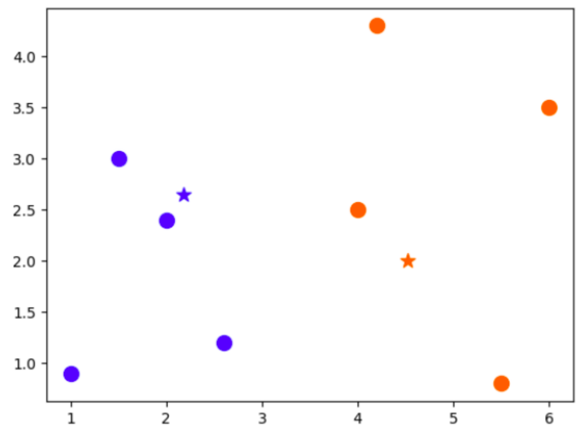

Fig. 1. Assignment of points to clusters according to centroids.

Next, for each data point in the dataset, a proximity function defined by the user is computed, and a relationship to a group is established based on a minimum value criterion. In other words, each data point is assigned to the nearest cluster according to a function that determines distance calculations (Figure 1).

When all data points have been assigned, each cluster computes the new centroid by calculating the mean of all the data points assigned to that cluster in this iteration.

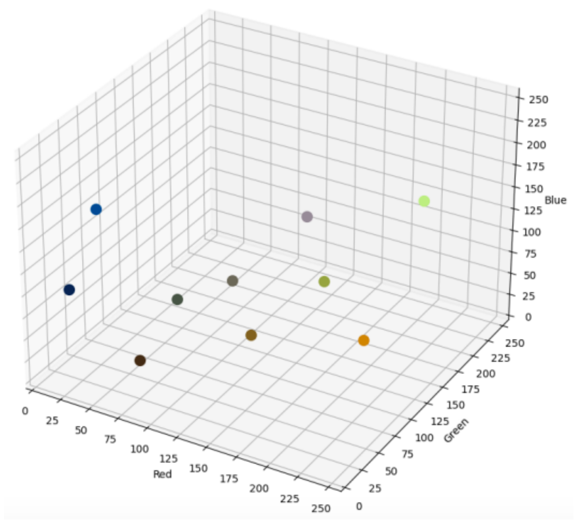

Fig. 2. Final centroids calculated by the K-means algorithm.

This process of assigning and updating centroids is repeated until no centroid changes its value in one iteration. When no centroid changes its value, it means that we have obtained the representative values of each group and a unique group assignment for each data point in the dataset (Figure 2

B. Description of the code provided

The functionality of the code we provide consists of classifying the pixels of an input image into a user-defined number of clusters K. Once all pixels are classified, the program sets the color of each pixel according to the color of the centroid of the cluster to which it belongs. Therefore, from a user perspective, we are applying a filter to reduce the number of different colors in an image while preserving its most representative colors.

The code requires an image with a bitmap color (bmp) extension and a color depth of 24 bits. These 24 bits store the information of the 3 values red, green, and blue (RGB), 1 byte for each of these colors (and therefore with values from 0 to 255). From this image, two outputs are generated: a file with the values of the found centroids and the image with the new color assignment.

C. Quantification of the Problem

The image provided for the problem execution[1] is 3840×2160 pixels; since each pixel occupies 24 bits, we have an image of about 24 MB. The centroids occupy 19 bytes each, and we won't have more than 255 of them, so we are talking about the order of a few KB.

[1]https://s2.best-wallpaper.net/wallpaper/3840x2160/1807/League-of-Legends-game-characters_3840x2160.jpg

You should consider that some of the strategies you implement may be more or less efficient depending on the size of the image and the number of centroids.

III. Parallelization

he main objective of the project is to parallelize the provided implementation of the algorithm using different approaches: shared memory and accelerator usage (we will use a different problem for message passing). This way, you will experiment with the differences between the implemented versions and gain knowledge to decide in which contexts you would implement solutions in the business and/or scientific world.

The second objective is to perform a performance analysis of the application. An analysis of the behavior of the implemented versions will be carried out, they will be executed using different configurations, measurements will be taken, and one or more performance issues will be identified (indicating their potential importance). We will use available analysis tools (`likwid`, `perf`, etc.).

It is important to note that you should not confuse optimizing serial code for a single processor with preparing the code to be executed on high-performance platforms. Good serial code can be very complex and may also hinder its parallelization.

A. Shared Memory

We will use OpenMP for this implementation.

In this case, you need to decide first whether you will use a task-based or loop-based parallelization.

If using task-based parallelization, you will need to define what a task is in this program and possibly make considerable changes to the code.

If using loop-based parallelization, you will need to identify which loops can be parallelized, what data dependencies exist, and find solutions that introduce the least overhead possible

B. Accelerators

We will use OPENACC (and optionally CUDA) for this implementation.

In this case, it is most important to identify the code sections that can benefit from execution on the accelerator and therefore justify offloading them to the accelerator or writing a CUDA kernel. Additionally, you need to clearly identify the operation performed on the data with the aim of using known (and possibly well-optimized) algorithms for accelerators.

C. Development Recommendations

Here are some recommendations on how to develop the practice:

- Make use of documents about the operation of the two department clusters available (lab cluster and Wilma cluster) and sending work to the resource manager. Use them! And if you don't have enough, go to the original manuals.
- Some tests may take a long time, and furthermore, in the case of performance analysis, they can generate very large amounts of information. Consequently, it may be a

TABLE I
WEIGHT OF EACH SUBMISSION IN THE FINAL GRADE OF THE SUBJECT.

OpenMP	**10%**
MPI	**10%**
OPENACC (CUDA)	**10%**

good idea to generate reduced versions of your program (doing only a few iterations, for example).

- Organize your project properly in your user account and use version control tools (there are several with free options):
  - **GitHub** https://github.com/
  - **GitLab** https://about.gitlab.com/

  This way, you will avoid surprises due to information loss, catastrophic decisions, etc.
- Use make, that is, write Makefiles that allow you to generate different versions of the code (e.g., complete, reduced) and clean the working directory of executable files and object code efficiently.
- Comment your code with useful comments (especially for yourselves). For a comment to be useful, it is very important that it is accompanied by the date and the author.
- The functional tests of your OpenMP, MPI, and CUDA programs will be done remotely. Remember to use the available execution queues, i.e., the `test.q, aolin.q, and cuda.q` queues of the laboratory or the `nodo.q` queue of the Wilma cluster.
- Performance analysis tests of your OpenMP and MPI programs can be done on both the Wilma cluster (queue `nodo.q`) and the laboratory architecture cluster (`aolin.q`). Regarding the CUDA practice, you will need to use the `cuda.q` queue of the laboratory architecture.

IV. DELIVERY AND ASSESSMENT

For each parallelization approach, you have 2 laboratory sessions. In the last session of each part, you will have to orally explain the implementations made. At the end of the last session, you will have a short deadline to submit an explanatory document of the practice. This means that by the last session, you should already have all the documentation and only check with the professors for any minor details or observations you may have in your last executions.

The total grade for practices represents 30% of the final grade of the subject. It is essential to have a grade of 5 or higher in the final grade of the practices to pass the subject. The contribution of each version is shown in Table I.

During the practices, you will have to make 3 submissions corresponding to each parallelization version: OpenMP, OPENACC (CUDA)[2] and MPI[3]. A submission consists of:

- A functional parallel implementation of the proposed code that will be shown to the practice professor before each submission (OpenMP, OPENACC (CUDA)).
- A PDF report describing:
  - Detailed explanation of how to compile and run your code.
  - The parallelization strategy.
  - The changes made to the code.
  - Performance metrics: the speedup and efficiency obtained compared to the original using different resource numbers.
  - The performance analysis performed.
  - The problem(s) found, their importance, and possible solutions.

The submission of the report in PDF format and the source code (`*.h, *.c, *.cu`) will be done on the Virtual Campus of the subject packaged in a `.zip` or `.tar` file by the date indicated on the Virtual Campus.

The following criteria will be used to calculate the grades for the submissions:

- [1.5 points] Accurate and reasoned explanation of the parallelization strategy.
- [1 point] The parallelization strategy used is suitable for the proposed code.
- [2 points] The adaptations made to the code and its 'correct' functioning.
- [0.5 points] Documentation and code style.
- [1.5 points] Obtaining metrics with different tools, problem sizes, and resource numbers.
- [1.5 points] Performance analysis for the executions performed.
- [0.5 points] Description of the problems encountered and their solutions found in the development of the practice.
- [0.5 points] Continuous evaluation of attendance and participation in class [INDIVIDUAL GRADE].
- [1 point] Oral responses given in the last session of each version [INDIVIDUAL GRADE].

Please note that for each submission, you must specify the compilation commands of your code and the compiler you used

V. CONCLUSIONS

We have presented the project you will undertake during this semester. Any doubts or questions you have, please contact us at `Anna.Sikora@uab.cat`, `Eduardo.Cesar@uab.cat` and `Carles.Carrillo@uab.cat`. Don't hesitate to ask for tutorials!

[2]Those who, in addition to the OPENACC version, perform a functional test with CUDA will have an extra grade in the practices.

[3]The statement for the MPI practice will be published later.