PROJECT ON HETEROGENEOUS COMPUTING

COURSE: EFFICIENT HETEROGENEOUS COMPUTING, M.EIC033

MASTER IN INFORMATICS AND COMPUTING ENGINEERING (M.EIC)

FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO (FEUP)

November 2021

This document describes the project for the "Efficient Heterogeneous Computing" course of the Master in Informatics and Computing Engineering (M.EIC), of the Faculty of Engineering of the University of Porto (FEUP).

THE PROGRAM

The k-Nearest Neighbors (*kNN*) [1] is a widely used machine learning technique for classification and regression[1]. For classification, and using the basic version of *kNN*, the training corresponds to the storage of the *n* training instances, each one represented as a vector of d features (dimensions), with each vector labeled by the respective class. For each instance to classify, kNN determines the *k* nearest instances, i.e., the *k* training instances nearest the instance and then classifies based on the classes of those *k* instances. One possibility is to classify with the class most frequent in the k closest training instances.

The code of the project implements *kNN* in C targeting the classification of ***m*** instances. All instances consist of vectors of ***d*** features represented in single- or double-precision floating point numbers (float or double types in C). Training considers ***c*** classes, and the Euclidean distance is used for the distance between the instance to classify and each of the training instances.

The configuration of a *kNN* implementation is defined by the tuple (*m, k, n, d, c, t*), where ***m*** represents the number of instances to classify, ***k*** the number of neighbors to consider, ***n*** the number of training instances (stored in the *kNN* database), ***d*** is the number of features, ***c*** is the number of classes, and ***t*** the data type used for the data and distance calculations.

The computational complexity of the basic *kNN* is *O(knd)* or *O(nd + kn)*, depending on the code location for updating and/or calculating the *k* nearest neighbors.

GOALS

The goal is to accelerate the execution of the function *classifyinstance* in two environments: (1) a PC/desktop; (2) an embedded system with a hardware accelerator. For parallelization targeting a multicore CPU, the intention is to use the OpenMP directive-driven programming model. For optimizations regarding the generation of the hardware accelerator, the intention is to use the optimization directives provided by the specific FPGA compilers.

[1] A brief description of the kNN technique is presented in https://en.wikipedia.org/wiki/K-neare_st_neighbors_algorithm.

The implementation of this function and of its auxiliary functions are in the file `knn.c`. However, changes do not need to be limited to the function *classifyinstance*.

SCENARIO A: PC AND EMBEDDED WITH DATA EMBBEDED IN THE CODE

This is the scenario that shall be used for the experiments in the desktop and in the embedded device with the hardware accelerator. The scenario uses input data embedded on the code of the program and the program does not receive any argument from command line. The input data is from WISDM[2] v1.1 [2], a dataset of Human Activity Recognition (HAR), with data from sensors in mobile devices and collected by different users.

In this scenario, the *kNN* implementations to be improved are based on the following parameters:

Parameters	**m**	**k**	**n**	**d**	**c**	**t**
Identification in the code	num_new_points	K	num_points	num_features	num_classes	DATA_TYPE
Generic implementation (values are for testing)	Variable input: size for tests = 1082	variable (use 3 and 20 for testing)	variable (use 4336 for testing)	variable (use 43 for testing)	variable (use 6 for testing)	float double
Specialized implementation (fixed values)	NA	3	4336	43	6	float

The classes (human activities) of the WISDM dataset are represented here using the following numerical identifiers (IDs):
Class: Upstairs ID = 3
Class: Jogging ID = 0
Class: Sitting ID = 2
Class: Standing ID = 1
Class: Walking ID = 4
Class: Downstairs ID = 5

SCENARIO B: PC AND EMBEDDED WITH RANDOM DATA

In this scenario, data are randomly initialized and the program code provides three configurations ("d1", "d2, and "d3) according to the table below.

Parameters	**m**	**k**	**n**	**d**	**c**	**t**
Identification in the code	num_new_points	K	num_points	num_features	num_classes	DATA_TYPE
d1 specialized implementation (fixed values)	NA	15	10,000	100	8	float double
d2 specialized implementation (fixed values)	NA	15	250,000	100	8	float double
d3 specialized implementation (fixed values)	NA	15	1,000,000	100	8	float double

[2] This data has been released by the Wireless Sensor Data Mining (WISDM) Lab: http://www.cis.fordham.edu/wisdm/

SCENARIO C: PC + DATA LOADED FROM FILES

This scenario was provided by the SPeCS group for the POP19 competition. In this scenario, data are read from files (.dat), and the program provides a score at the end of execution. A score of 100 represents the execution time of the original code on a lab machine (ANTAREX[3]) when compiled without optimizations and using double-precision floating-point values. Higher scores represent faster implementations, and the objective is to submit a version that gets the best score when using the large input (i.e., the input file *knn_large_1.dat*).

The original version of *knn* for the large input takes around one minute to run on the ANTAREX machine.

Example

Below are the results of executing the code using the medium input.

```
> . / knn knn medium 1 . dat
Initializing data points ...
Reading data from . /input /knn_medium_1.dat .
1000-NN, 8 classes, 250000 instances, 100 features,
10 points to classify
Initialization done.

Executing kNN...
class id: 5
class id: 2
class id: 7
class id: 2
class id: 6
class id: 7
class id: 7
class id: 7
class id: 5
class id: 1
kNN done.

kNN: number of classes = 8
kNN: number of training instances = 250000
kNN: number of features = 100
kNN: k = 1000
kNN: number of classified instances = 10

Time:  7.9271 s
Score: 103
```

At the end of execution, the program reports whether the produced outputs are correct or not and the accuracy achieved in the classification of the input instances. It also prints the computation time to classify the points in the input data, as well as the corresponding score.

[3] CPU 2x INTEL XEON E5-2630V3 2.4 GHZ 20MB CACHE 8 cores per CPU, and 2 threads per core (total of 16 cores, and 32 threads); RAM 8x 16GB DDR4-2133MHZ REG ECC MODULE (128GB total, 64GB per NUMA node). The machine has Ubuntu 16.04 LTS installed. Intel® Turbo Boost is enabled, which means that the CPU can go above the frequency of 2.4GHz for short amounts of time (up to 3.2 GHz).

STEPS TO BE FOLLOWED AND REPORTED:

A. Analysis and improvements targeting Intel multicore CPUs and a PC/desktop/laptop

1) Analyze via profiling (gprof and gcover) and via Intel VTune and Intel Advisor the program given. For example, identify the hotpots, draw the call graph and the task graph of the program and decorate them with the information you find useful for helping to improve the execution time and the selection of the code regions to be possibly migrated to an hardware accelerator.
2) Considering its execution on a PC/desktop/laptop try to improve its overall performance and use Intel VTune and Intel Advisor to help you (e.g., verify the positions in the Roofline model of the subsequent versions).
3) Use the gcc compiler and selected flags and show the energy consumption, power dissipation, and execution time gains that you were able to obtain with the different versions over the original one;

B. Analysis and improvements targeting a SoC with a multicore ARM CPU and hardware accelerators using FPGAs.

NOTES REGARDING THE WORK AND/OR THE TECHNICAL REPORT:

The most important experiments, e.g., involving compiler optimizations and/or code transformations, and the respective results shall be reported even if they do not improve execution time, power dissipation, and/or energy consumption.

In addition, the report shall describe:

- The experimental setup (e.g., the machine used where the experiments were conducted) shall be described in detail;
- The measured results and an analysis of those results;
- Possible code transformations and optimizations that you think can have a significant impact but did not have the time to evaluate.

REFERENCES

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," in IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27, January 1967, doi: https://doi.org/10.1109/TIT.1967.1053964.

[2] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. 2011. Activity recognition using cell phone accelerometers. SIGKDD Explor. Newsl. 12, 2 (December 2010), 74–82. doi: https://doi.org/10.1145/1964897.1964918.