Instituto Superior Técnico

HW/SW Co-Design

MEEC

k-NN HW Acceleration using a Zybo Board - P2

Authors:
André Sousa nº **65313**
João Borrego nº **78990**

Group:
15

June 11, 2017

1 Introduction

This report covers the second half of our course project with the objective of accelerating a k-NN classification algorithm using a Zybo[1] board.

Having implemented a functional solution involving custom hardware, we were mainly focused on improving the overall system, changing our custom IP as little as possible, investigating the benefits of deploying several instances of distance calculation units and evaluating the performance increase of using more than one CPU.

Our initial communications were performed using an AXI Stream FIFO which is certainly not very efficient for large streams, and requires the CPU to handle actual data transactions, instead of just performing control operations. In this iteration we resort to Direct Memory Access (DMA) Interfaces instead which is expected to result in an immediate improvement over the AXI FIFO.

Our hardware design allows for the use of several instances of floating point distance calculation units, thus allowing for the exploitation of data parallelism. Furthermore, we also improved the overall performance of our system by delegating some of the workload to the second CPU on the board. This involved dealing with synchronisation and workload balancing issues, which are explored in the body of the report.

Finally, we benchmarked our solution using two separate data sets, with fairly different size and characteristics, obtaining decent speedups.

2 Hardware Design

We tried to alter the hardware design as little as possible in the second part of this project.

The main change consists of replacing the AXI Stream FIFO by DMA (Direct Memory Access) interfaces to handle communication between the custom hardware and the ZYNQ® processing unit. It is substantially faster than the previous solution and reduces the overhead on the CPU side, as it is no longer required to handle actual data transaction. As it is compliant with the AXI master slave protocol, no major changes were required to be performed at the custom distance calculation IP level.

Our hardware effectively calculates a distance matrix from each training set object to a testing object given the respective feature vectors. For this task, it first expects to receive the testing object feature vector, and then all of the training objects. Since the number of training objects is independent from the characteristics of a single testing object, it had to somehow be communicated to the hardware, so that the state machine of our previous implementation could function properly. In order to solve this problem we introduced a terminator byte sequence that signalled when to expect the last training object. This was removed in the second iteration, as we can now rely on an end of stream signal (`T_LAST`) that is activated by the DMA interface.

[1] `https://reference.digilentinc.com/reference/programmable-logic/zybo/start`

2.1 Scalability

In order to maximise the resource utilisation of our board and overall speedup, our implementation allows for multiple IPs and respective DMA instances. Since inherent data parallelism is present in the problem, as the entries of the distance matrix are independent, one can effectively broadcast a different testing object to each IP instance and then obtain the respective distances vector, corresponding to multiple lines of the distance matrix.

One should mention that the removal of the terminator data sequence in favour of a simpler solution poses a problem as we increase the size of the training set, since the DMA write buffer's size may become insufficient, resulting in the data stream being split. We configured the DMA with 20-bit addresses, which allows for a total of over 260 000 single-precision floating point features, meaning the number of training objects multiplied by the dimension of the feature vector may not exceed this number. Should one need a bigger data set, one needs to break the training set down in chunks, and between sending each chunk first send the testing example. This step is meant to be implemented in software. Although some redundancy is introduced in our solution, it should be insignificant for very large data sets.

Nevertheless it should be mentioned that a far more efficient system could have been developed, using a single DMA instance and routing hardware between it and several distance calculation IPs. Each DMA instance consumes a fair amount of resources to be implemented in the FPGA. Furthermore, since each IP must have each training set object, the existing resources are effectively being wasted by communicating the same data using separate DMAs.

While aware of this limitation, it was also intended that in this second phase of the project no major changes were made to the custom IP, and instead we focused on improving the higher-level architecture and interaction with the CPU.

2.2 Specifications

Ultimately, we decided to include two IPs each with a respective DMA instance, although it should be possible to use at least 3 and possibly even 4 instances on the Zybo board. A representation of the high-level block diagram can be seen in Figure 1.

It should be mentioned that two separate GP ports were used for controlling the DMA, although a single one would have sufficed. This was done in an attempt to avoid connection errors between the hardware blocks and ill configured interconnect units. The DMA is connected to the ZYNQ processing unit by the HP (High-Performance) ports, in order to optimise data transfer rates.

Timing Report

A brief summary of the timing report can be seen in Table 1

Worst Negative Slack	Worst Hold Slack
0.443 ns	0.015 ns

Table 1: Timing report summary

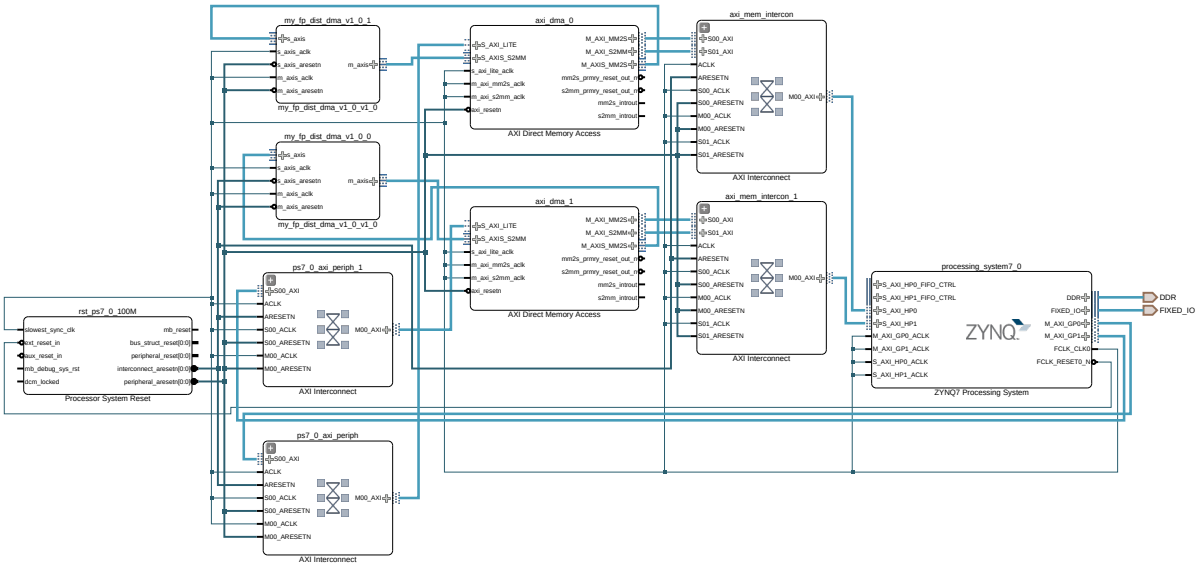

Figure 1: High-level representation of the hardware system

These results show that the system can operate at the desired frequency of 100 MHz.

Utilisation Report

A brief summary of the utilisation report is present in Table 2.

Site Type	**Used**	**Utilisation(%)**
Slice LUTs	7314	41.56
as Logic	6837	38.85
as Memory	477	7.95
Slice Registers	8627	24.51
as Flip Flop	8627	24.51
as Latch	0	0
Block RAM Tile	6	10.00
DSPs	12	15.00

Table 2: Utilisation report summary

It can be seen that less than half of the resources of the board were used, which supports our claim that at least 4 IPs with respective DMA could be implemented with our design.

3 Software

This version of our system involves more complex software, since it has to handle:

- Configuration and initialisation of the DMA interfaces;
- Sending data set objects to hardware and retrieving the calculated distance matrix;
- Coordination of 2 ARM CPUs;
- Classifying test objects given the respective distance matrix.

The data set is expected to be split into four binary files: two for the training and testing samples feature vectors, encoded as single-precision floats, and another two for the class labels, encoded as integers. All of the dataset properties (e.g. number of training examples, dimensionality of feature vectors, etc.) should be configured in the header file `data.h`.

The number of nearest neighbours k is configured to be 3 by default, but can easily be altered in the code by changing a macro definition. The classification process depends on the value of k linearly, as it needs to perform k selection sort iterations in each of the distance matrix rows.

In total, we implemented 4 different versions of the software:

1. Sequential version, that fully executes the algorithm in the ARM CPU and is used as reference;
2. 1 IP, that uses a single hardware instance of distance calculation and respective DMA for obtaining the distance matrix;
3. 2 IPs, that uses 2 instances of our IP and respective DMAs;
4. 2 IPs and 2$^{\text{nd}}$ CPU, that uses all the resources available in our design.

3.1 Code Structure

The versions that rely on hardware share the same structure. Firstly, one needs to configure the DMA interfaces and initialise the devices in poll mode. The calculation of the distance matrix is performed in a loop, each iteration respective to one or two testing objects, depending on whether it resorts to one or two IPs. Firstly the testing object needs to be copied to the hardware and then the whole training set. The CPU only issues commands to the DMA units, and does not need to handle actual data transfers. In fact, this happens asynchronously, and one needs to make sure that the communications have followed through in order to continue execution correctly. This was dealt by introducing brief active wait periods in the end of the body of the loop, thus ensuring that the output distance vector has been successfully received.

Once the distance matrix is available the actual classification takes place.

3.2 Memory

Our initial implementation used a custom data structure that stored distance-label pairs kept in heap memory. This was not only unnecessary but also inefficient. In this iteration all of the data, both input and output, is kept in known positions of DDR memory. The default address of every data object is shown in Table 3.

Object	Macro	Base Address Value
Training Set Data	`TRN_DATA_BASE_ADDR`	`0x010000000`
Testing Set Data	`TST_DATA_BASE_ADDR`	`0x013000000`
Training Set Labels	`TRN_LABEL_BASE_ADDR`	`0x016000000`
Testing Set Labels	`TST_LABEL_BASE_ADDR`	`0x017000000`
Output Labels	`OUT_LABELS_BASE_ADDR`	`0x018000000`
Distance Matrix	`OUT_DIST_BASE_ADDR`	`0x019000000`

Table 3: Memory default base addresses for each data object

Since the memory management is exclusively done by the programmer, one needs to be careful and avoid overlapping data segments. This configuration works for both the referenced data sets. The program itself is uploaded to faster Zybo's OCM. In fact, each CPU has their own program code, heap and stack stored in this OCM memory, in manually specified addresses so as to avoid conflicts (`0x00000` and `0x10000` respectively). Some other data is also required by each CPU and is stored in their own DDR section (`0x100000` and `0x200000`).

Since the distance matrix entries are changed directly by the hardware, the CPU must not rely on cached copies of the data, so the programmer has to invalidate the cache explicitly in the code. This guarantees that the next time it needs to access a variable the program retrieves its values directly from DDR memory.

3.3 Multi-Processor Synchronisation

Since a second CPU is available in the Zybo board we decided to use its capabilities for both distance calculation and classification. This adds synchronisation issues, which can be solved by introducing explicit synchronisation points, with a lock variable that is stored in OCM memory. Each of the CPUs must be configured to prevent caching the respective OCM section, thus ensuring they both see the same value at all times.

Regarding the workload distribution, assuming that using a single DMA exhibits a speedup s relative to the software version, the theoretical optimal fraction of work to be assigned to the second CPU is given by

$$f_{\text{CPU1}} = \frac{1}{s+1}. \tag{3.1}$$

The classification work load is split evenly, since it is done purely by software. In total, the final version of our system has 3 synchronisation points, one right at the start, another after completing distance matrix calculations and a final one after the actual classification task. The synchronisation effort is expected to add some overhead and limit speedups.

4 Results

4.1 Data set Description

Two data sets from the UCI Machine Learning Repository[2] were used for benchmarking our application.

Iris Data Set

Also known as the Fisher's Iris data set[3], it contains 4 flower measurements of 150 samples of 3 flower species: *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*. This data set was chosen since it is fairly common in literature and overall well-known in the Pattern-Recognition community. A training/testing set split of $\frac{2}{3}$ was used.

Wine Quality Data Set

This data set consists of 12 features from over 6000 samples of 2 different variants (red and white) of Portuguese wine known as "Vinho Verde"[4] provided by University of Minho[1]. It is fairly large data set and it allows us to evaluate the performance and roughly estimate its scalability. In this case, we chose a 50/50 training/testing set split.

4.2 Execution Times

We started by defining two time measurements:

1. Kernel execution, which only accounts for the time spent in the calculation of the distance matrix;
2. Global execution, which regards the execution as a whole, including variable initialisation and peripheral (such as DMA interfaces) configuration. For the final version, with 2 processors, the timer is only initiated after an explicit synchronisation point at the start of the program.

Our system's output was verified to be correct and the classification accuracy for the Iris and Wine Quality data sets for $k = 3$ were 96.00% and 96.47%.

The registered execution times for both Iris and Wine Quality data sets are present in Figures 2 and 3 respectively.

Notice in Figure 2 that introducing a second CPU effectively degrades the kernel performance, yet speeds up overall execution. This is due to the workload distribution, that was not fine-tuned to the data set and resulted in CPU1 bottle-necking the kernel. However, the overall execution was faster due to the classification part, that is split evenly among the processors and makes up for the slower kernel. In Figure 3 both the kernel and overall execution benefit from the

[2] `https://archive.ics.uci.edu/ml/datasets/`
[3] `https://archive.ics.uci.edu/ml/datasets/Iris`
[4] `https://archive.ics.uci.edu/ml/datasets/Wine+Quality`

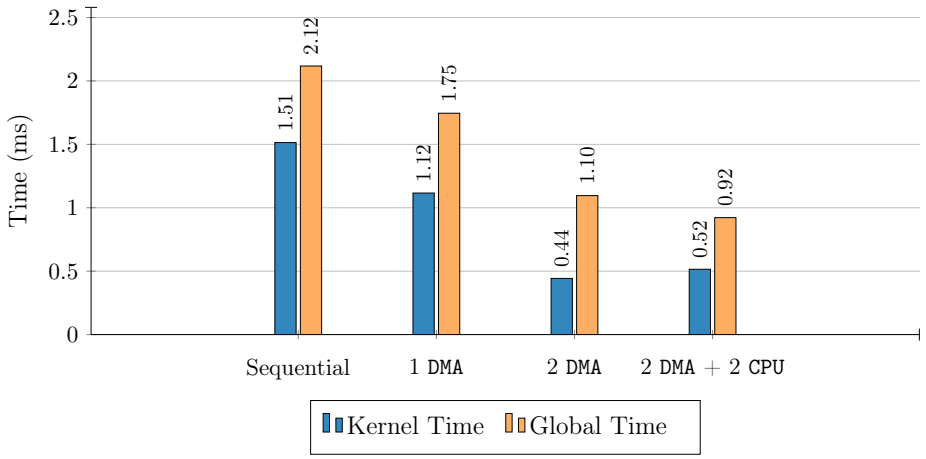

Figure 2: Kernel and global execution times for each implementation with Iris data set (Averaged 3 runs)

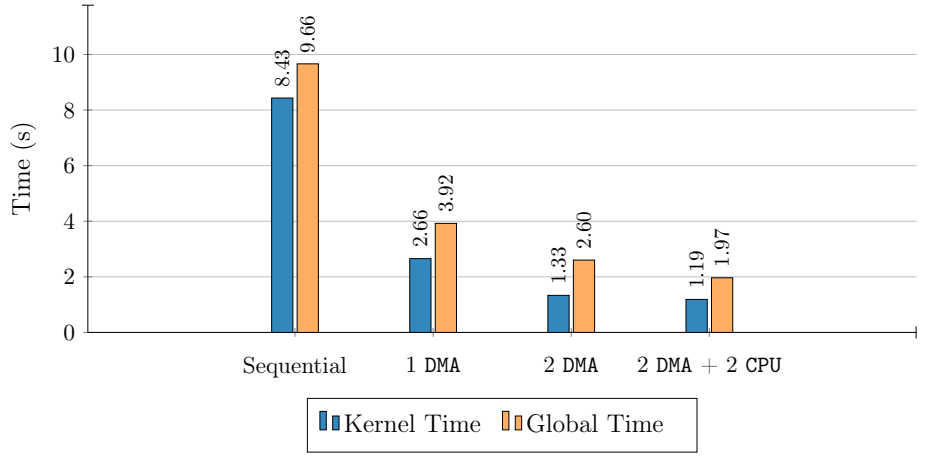

Figure 3: Kernel and global execution times for each implementation with Wine Quality data set (Averaged 3 runs)

added processor, as the workload distribution was properly calculated. This is further detailed in the next subsections.

4.3 Speedup

The average speedup relative to the pure software version were calculated and are shown in Figures 4 and 5.

First of all, it should be mentioned that by simply replacing the AXI FIFO with the DMA instance, our program was able to outperform the software baseline (which had not been achieved by the previous version).

When adding the second DMA and distance IP the speedup scales perfectly as the kernel performance effectively doubles. This was to be expected since the overhead is minimal and there

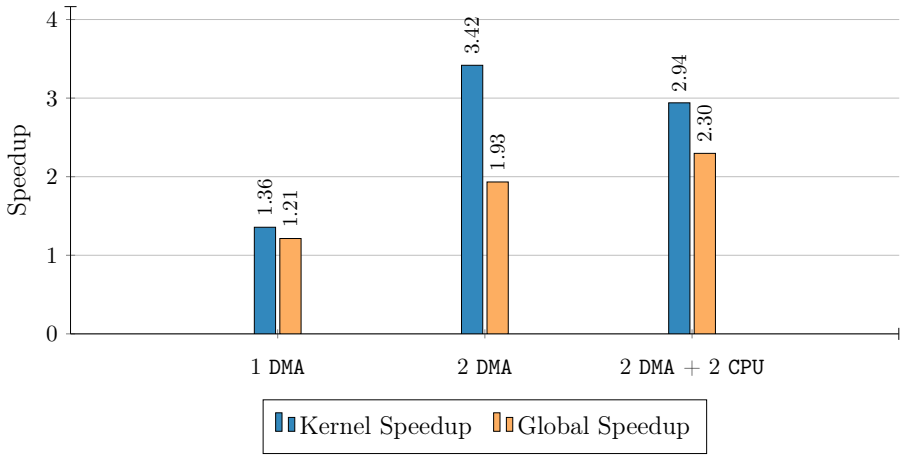

Figure 4: Kernel and global speedups for each implementation for Iris dataset

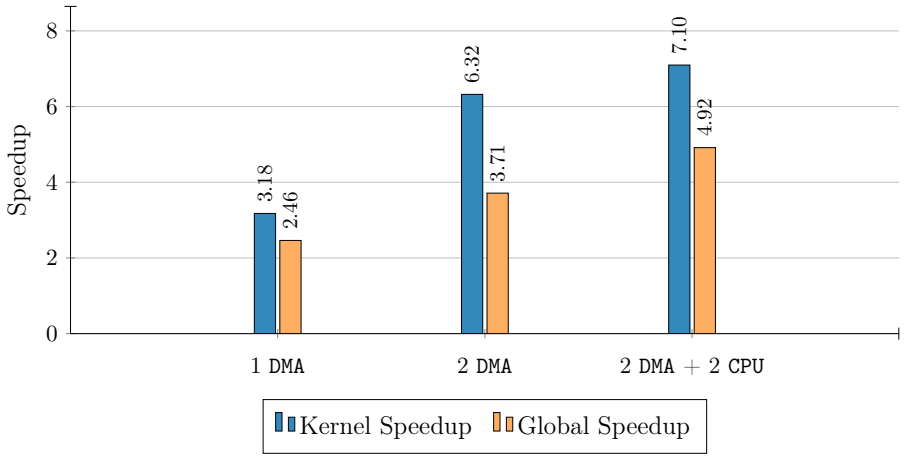

Figure 5: Kernel and global speedups for each implementation for Wine Quality dataset

is no communication between the hardware instances, which results in efficient parallelisation. Given the previous hardware utilisation report, one can estimate that the kernel speedup could have been increased to upwards of tenfold by adding 2 DMA instances and respective distance calculation IPs and assigning them a quarter of the total workload each, even without using the second CPU.

The workload distribution between IPs and the second CPU is very important and an optimal solution would result in each of these units finishing their work at the same time. The ideal fraction of work to be assigned to the second CPU has been calculated in Equation 3.1. However, in practice this value has to be fine-tuned as the communication overhead may force us to assign a smaller work load. Figure 6 shows the impact of changing the CPU workload.

Even though the theoretical fraction is approximately $\frac{1}{8}$ the experimental results show a higher speedup for $\frac{1}{9}$ of the total workload. Nevertheless, the expected value is an excellent starting point for testing out different values. The results confirm that this fraction plays an important role in the overall speedup and thus it must be fine-tuned for optimal performance.

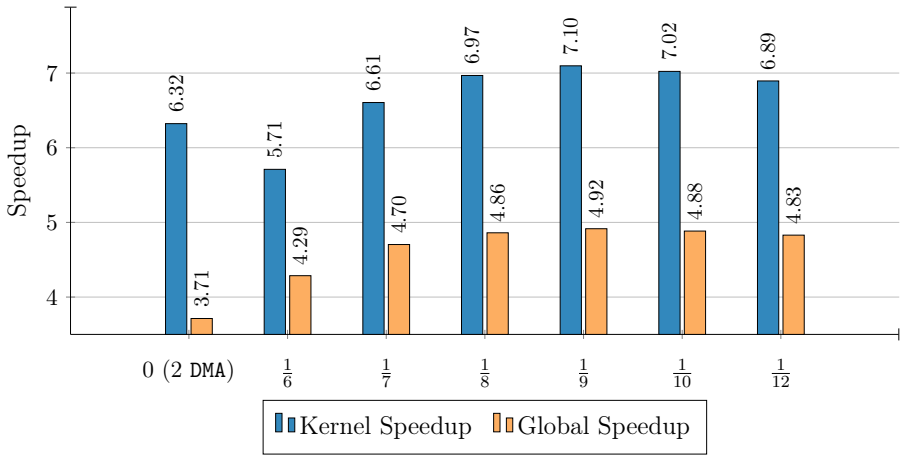

Figure 6: Kernel and global speedups for each implementation for Wine Quality dataset with varying CPU1 workload distribution

5 Conclusion

We successfully designed a system that sped up the execution of k-NN classification algorithm using the ZYBO board as initially proposed. This was achieved by optimising squared euclidean distance calculations using dedicated hardware units that can be deployed and function in parallel. Our solution takes advantage of both ARM CPUs in the board, one of which is handles the communication with hardware. The second CPU also handles a small portion of the determination of the distance matrix. The final classification task is split evenly among processors.

Our system could have been further improved modularly by adding more hardware instances and respective memory management units, although altering the hardware to incorporate the classification task, i.e., the storing the smallest k distances, seems a much more profitable approach regarding speedup potential.

This project gave us proper insight into the process of designing systems that exploit software and hardware symbiosis in order to achieve optimal performance, understanding the major bottlenecks in communication.

References

[1] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems 47*, 4 (2009), 547–553.