K-Nearest Neighbor Algorithm Implementation on FPGA Using High Level Synthesis

Zhe-Hao Li[1], Ji-Fang Jin[1], Xue-Gong Zhou[1], Zhi-Hua Feng[2]*
[1] State Key Laboratory of ASIC & System, Fudan University, Shanghai 200433, China
[2] Beijing Institute of Computer Technology and Application, Beijing 100854, China
* Email: zhihua_feng@126.com

Abstract

The K-Nearest Neighbor (K-NN) algorithm is one of the most common classification algorithms and widely used in pattern recognition and data mining. K-NN hardware acceleration is necessary for applications with massive high-dimensional data. High level synthesis (HLS) is an increasingly adopted technique in digital circuit design, which can help to raise the abstraction levels. In this paper, we exploit the parallelism and pipelining opportunities and apply the memory-mapped AXI4-Master Interface to implement K-NN on an FPGA using HLS. The evaluation result shows that our HLS-based solution is 35.1x faster than a general purpose processor (GPP) based implementation, and comparable to hardware description language (HDL) based implementations, while our HLS-based solution largely reduces the development complexity and cost.

1. Introduction

High level synthesis (HLS) is an increasingly adopted technique in digital system designs [1]. It allows users to describe their hardware systems in a higher abstraction level, using high level language such as C/C++. References [1][2] have summarized the benefits brought by HLS: (1) Improve productivity for hardware designers by allowing them to work at a higher level of abstraction and reducing their workload of hand coding the hardware architecture and timing. (2) Functional correctness of the design can be verified at C-level, which is much more efficient than the verifications with traditional HDLs. These benefits help to reduce the development cycle and render a fast time to market.

HLS and Field Programming Gate Array (FPGA) make a practicable combination for hardware acceleration. A series of HLS tools have been developed for commercial use by FPGA vendors, such as Xilinx Vivado HLS and Altera OpenCL SDK, and widely applied to computation-intense applications, such as image processing [3].

K-Nearest Neighbor (K-NN) algorithm is one of the most common supervised classification algorithms to classify an unknown query vector according to a set of training vectors with known class labels [4]. It has been widely used in pattern recognition and data mining.

The performance of the K-NN algorithm is sensitive to several parameters: number of training vectors (N), number of dimensions (M), number of class labels (C) and number of neighbors required for class label determination (K). As N and M increase, execution of the K-NN algorithm becomes a time-consuming task and requires high computational power, which makes hardware acceleration for K-NN necessary.

There have been some previous implementations on K-NN hardware acceleration. Reference [5] presents two hardware architectures described as soft IP cores, which are claimed to be the first flexible K-NN IP cores design. Reference [6] implements the hardware architectures proposed in [5] on FPGAs, and compares the performance with GPP and GPU implementations.

In [5][6], hardware architectures are implemented with HDLs, which is a time-consuming task for high performance. In this paper, we present a high performance HLS-based solution for K-NN. We exploit the parallelism and pipelining opportunities to enhance the performance of the algorithm execution and use the Xilinx standard AXI4-Master Interface [2] to ensure high efficiency for data transmission. When evaluating our design on FPGA, the performance is 35.1x faster than the GPP-based implementation and comparable to HDL-based implementations, while the development complexity and cost are largely reduced.

2. K-NN Implementation Using HLS

The implementation of K-NN consists of the hardware architecture part for K-NN algorithm execution, and the data transmission part for data transfer between the hardware architecture and the memory.

2.1 Hardware Architecture

The K-NN algorithm can be divided into three steps: (1) Distance Calculation: calculate the distances between the query vector and all the training vectors based on a particular distance metric such as Euclidian distance. (2) K-NN Selection: select the K nearest training vectors to the query vector from the training set as the K nearest neighbors. (3) Class Determination: use the class label that occurs most in the K nearest neighbors as the class label of the query vector. Based on these three steps, we divide our hardware architecture into three blocks—Distance Calculation (DC), K Nearest Neighbors Finder (KF) and Class Label Voter (CLV).

978-1-4673-9719-3/16/$31.00 ©2016 IEEE

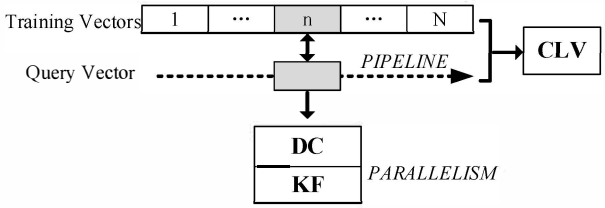

Figure 1. The K-NN hardware architecture

Figure 1 shows the K-NN hardware architecture. The DC block consists of several calculation units to calculate the distances between the query vector and all the training vectors. The KF block consists of *K* comparators, which are used to find the *K* nearest neighbors and store their correspondent distances and class labels. The CLV block is a majority voter that includes *C* counters and a comparator. The counters, each associated with one of the class labels, are used to record the frequencies of each class label in the *K* nearest neighbors. The comparator is used to find the class label with the highest frequency to be the class label of the query vector.

To achieve high performance, the DC block and the KF block are arranged to work in a parallel way, while each of them works in a pipelined manner individually, as shown in Figure 1. Thus the execution time of the first two steps of the algorithm can be reduced to approximately *N* clock cycles, much less than *(K+1)N* clock cycles if the two blocks work in a sequential way. Pipelining allows the DC block to output one distance result every clock cycle after the first distance result is generated. The KF block becomes active once the DC block outputs its first distance result. Then the distance results stream into the KF block and compare with every distance data in the *K* comparators sequentially. The comparisons will be executed in a pipelined manner. When the pipelining completes and the *K* nearest neighbors are found, the CLV block begins to work.

```
1.  Initialization
2.  for each training vector in the training set
3.  #pragma HLS PIPELINE
4.      Calculate the distance
5.      for each comparator in the KF block
6.      #pragma HLS ARRAY_PARTITION
7.          Compare and update the comparators
8.      end for
9.  end for
10. for each element in the K nearest neighbors
11. #pragma HLS UNROLL
12.     Update the counters
13. end for
14. Find the class label for the query vector
```

Figure 2. The pseudo-code for hardware architecture

Using HLS, the hardware architecture of K-NN can be described in less than 100 lines C code. The pseudo-code is shown in Figure 2.

The DC and KF blocks correspond to Lines 2-9 in Figure 2. For traditional designs using HDLs, the complexity of the timing required for parallel processing and pipelining makes the hardware design a time-consuming task. In contrast, HLS offers a much simpler way to achieve the same goal, as the HLS tool (Vivado HLS) provides a series of pragmas for users to optimize their design conveniently. Once applied, those pragmas can guide the tool to allocate the resource and schedule the timing automatically.

In our design, we use pragmas *PIPELINE* and *ARRAY_PARRTION* to optimize the performance of the DC and KF blocks, as shown in Figure 2. The pragma *PIPELINE* is used to ensure parallelism and pipelining. The pragma *ARRAY_PARRTION* is applied to remove the bottleneck for the pipelining caused by the feature of the block RAM (BRAM). A BRAM allows at most two accesses every clock cycle. In the KF block, data in the *K* comparators are stored as arrays, which are mapped to BRAMs by default, preventing the KF block from pipelining. The *ARRAY_PARTITION* pragma helps to remove the barrier for the pipelining by mapping arrays into registers instead of BRAMs.

The CLV block corresponds to Lines 10-14 in Figure 2. An *UNROLL* pragma is applied to ensure that the *C* counters work in parallel to reduce latency.

2.2 Data Transmission

The data transmission part is integral to a hardware acceleration system in addition to the algorithm execution part. Transmitting data with microprocessors may render low performance. Therefore in our design, we adopt the DMA controller driven Xilinx standard AXI4-Master Interface, which supports up to 4 Gbps bandwidth, to implement the data transmission. Figure 3 shows the data transmission architecture.

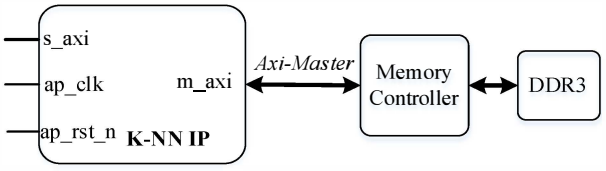

Figure 3. The K-NN data transmission architecture

```
1. function knn (volatile float *data)
2. #pragma HLS INTERFACE m_axi port=data
3. #pragma HLS INTERFACE s_axi port=return
4.     memcpy()
5.     …
6. end function
```

Figure 4. The pseudo-code for data transmission

Table 1. Performance Comparisons

Implementation	Throughput (MByte/s)
GPP*	0.35
Manolakos's solution[5]	12.50
Hussain's solution [6]	12.50
Our Solution	12.28

*Matlab2012b, Intel i5-2400 CPU @3.1GHZ, 12G RAM

Data are transmitted between the memory and the hardware computing circuit in the K-NN IP core through the AXI4-Master Interface and the data bus. The pseudo-code for the data transmission implementation using HLS is shown in Figure 4.

Using HLS, interfaces can be specified with pragmas conveniently as shown in Lines 2-3 of Figure 4. The AXI4-Master Interface is used for data transmission to ensure high performance, while the AXI4-Lite Interface is adopted for the microprocessor to control the behavior of the IP core via a control bus. The function *memcpy()* in Line 4 of Figure 4 is applied for data transfer in the burst mode. It ensures that the K-NN IP core transfers data with the memory using a single base address followed by multiple sequential data samples, which can guarantee high data throughput [2].

3. Experimental Result

Our proposed K-NN hardware accelerator is exported as an IP core, whose performance is evaluated on the Xilinx ZC706 FPGA board with Vivado 2015.2 Design Suite, with parameters shown below:

$$N = 1024, \quad M = 2, \quad C = 2, \quad K = 7$$

The evaluation architecture is based on Figure 3, and the system clock runs at 100MHz in the evaluation.

3.1 Evaluation of Algorithm Execution

According to the synthesis report provided by Vivado HLS, the maximum operation frequency for the K-NN IP core is 114MHz on the xc7z045ffg900-2 FPGA. The K-NN IP core needs 36528 clock cycles to classify 32 query vectors, among which 3184 clock cycles is consumed for data transmission, and 33344 clock cycles is used to execute the K-NN algorithm. By calculation, we know that 1042 clock cycles are needed to classify every query vector, while 1033 clock cycles are required in the implementations proposed in [5][6].

Table 1 shows the performance comparisons of the algorithm execution part among our HLS-based, the GPP-based and the HDL-based implementations proposed in [5][6]. As Table 1 shows, the performance of our design is 35.1x faster than the GPP-based implementation and close to the HDL-based implementations. Moreover, compared with using HDLs, applying HLS technique reduces the development complexity and cost of hardware design in a large scale.

Table 2. Percentage Distribution of Time

Data Transmission	Algorithm Execution
8.72%	91.28%

3.2 Evaluation of Data Path

Table 2 shows the percentage distribution between data transmission time and algorithm execution time when classifying 32 query vectors. As Table 2 shows, after applying the DMA controller driven Xilinx standard AXI4-Master Interface, the data transmission time only occupies 8.72% of the total runtime. Thus adopting the AXI4-Master Interface removes the bottleneck of data transmission and helps to achieve high performance for hardware acceleration.

4. Conclusion

In this work, an IP core for K-NN hardware acceleration is designed using HLS technique and evaluated on an FPGA. The performance of our design is 35.1x faster than a GPP-based implementation and the development complexity and cost are largely reduced when compared with HDL-based implementations.

Acknowledgments

This research is supported by National Natural Science Foundation of China 61131001.

References

[1] W. Meeus, K. V. Beeck, T. Goedemé and etc., "An overview of today's high-level synthesis tools," Design Automation for Embedded Systems, 16, p.31-51 (2012).

[2] Xilinx, "Vivado design suite user guide: High-Level Synthesis," UG902 (2015).

[3] A. Mishra, M. Agarwal and K. S. Raju, "Hardware and software performance of image processing applications on reconfigurable systems," 2015 Annual IEEE India Conference, p.1-5 (2015).

[4] H. Hussain, K. Benkrid and H. Seker, "An adaptive implementation of a dynamically reconfigurable K-nearest neighbor classifier on FPGA," 2010 NASA/ESA Conference on Adaptive Hardware and Systems, p.205-212 (2012).

[5] E. Manolakos and I. Stamoulias, "Flexible IP cores for the k-NN classification problem and their FPGA implementation," 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, p.1-4 (2010).

[6] H. Hussain, K. Benkrid, C. Hong and etc., "An adaptive FPGA implementation of multi-core K-nearest neighbour ensemble classifier using dynamic partial reconfiguration," 2012 International Conference on Field Programmable Logic and Applications, p.627-630 (2012).