2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines

An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using OpenCL

Yuliang Pu, Jun Peng, Letian Huang
School of Communication and Information Engineering
University of Electronic Science and Technology of China，UESTC
Chengdu, China
proust315@126.com,pj_for@yeah.net,huanglt@uestc.edu.cn

John Chen
China University Program
Altera
Chengdu, China
jochen@altera.com

***Abstract*—Accurate and efficient data classification techniques are of vital importance to many problems, and are rapidly developing in recent decades. K-Nearest Neighbor algorithm (KNN), as one of the most important algorithms, is widely used in text categorization, predictive analysis, data mining and image recognition, etc. To accelerate the algorithm and to optimize the parallel implementation solution are two key issues of KNN. In this paper, we propose a new solution to speed up KNN algorithm on FPGA based heterogeneous computing system using OpenCL. Based on FPGA's parallel pipeline structure, a specific bubble sort algorithm is designed to optimize KNN algorithm. The results have been shown that the efficiency of the solution in our paper is much higher than conventional GPU based KNN algorithm implementation.**

Keywords- KNN; FPGA; Heterogeneous Computing; OpenCL; Bubble Sort

I. INTRODUCTION

Owing to the rapid rise of resources integration, classification algorithms can be parallelizedly mapped on the Field Programmable Gate Arrays (FPGAs). Heterogeneous computing system architecture, which is constituted by FPGAs and CPUs, would become a meaningful architecture for its energy efficiency. However, the complexity of programmability is the major problem for designers to use this kind of architecture. Compared to traditional HDL-based design (High Description Language: Verilog, VHDL, etc.), some new approaches [1] have been established to reduce development time by giving access to a high-level framework. Open Computing Language (OpenCL) provides support to FPGA targets through Altera's implementation of an OpenCL compiler. By using Altera's OpenCL compiler, we are able to map the algorithms on parallel-pipeline structures implemented by internal resources of FPGAs.

K-Nearest Neighbor algorithm (KNN) is one of the most important algorithms used in text categorization, predictive analysis, data mining, image recognition etc [2]. The classical KNN method has gained its place as one of the 10 most important data mining algorithms developed in the 20th century [3]. Unfortunately, the computational cost of KNN is extremely intensive and the distance calculation and rank could be time consuming.

In this paper, we present an efficient parallel implementation of KNN algorithm using FPGA based heterogeneous computing system architecture, aimed at optimizing classical KNN algorithm. Traditional approach of distance rank needs to be done by full permutation before the *k* minimum distances are found [4]. However, due to the fact that in KNN only *k* minimum distances are necessary, we determine to implement a partial sort method which will merely sort out the k minimum distances of the query object. By introducing the idea of bubble sort into the distance rank process, we efficiently achieved the goal. For each query object, *k* work-items are used instead of the original n. Thus, the cost of hardware resources is reduced.

The rest of this paper is organized as follows. Section II presents other works related to the subject. Section II describes the KNN algorithm as well as the OpenCL program architecture. In Section III we propose the bubble sort enhanced KNN algorithm based on FPGA based heterogeneous computing platform and Section V details the performance results and comparisons.

II. RELATED WORK

Many works to accelerate the KNN algorithm have been done, but most of them focused on the optimization of algorithm strategy instead of implementation architecture. Besides, the energy efficient has become a crucial criterion, along computing speed and resources occupation.

Garcia et al. [5] proposed the first GPU-based KNN algorithm which is at least 10 times faster than traditional CPU implementation. In this paper comb sort and insertion sort were chosen to demonstrate their algorithm. Later, Nolan [4] came up with a method aiming at improving the performance of the algorithm by using a ranking strategy and bitonic sort. Quansheng et al. [6] proposed a GPU-based implementation of brute-force KNN using the CUDA-based radix sort [7] whose computing speed is 12 to 13 times faster than the sequential CPU program. In 2009, Liang et al. [8] proposed a new CUDA-based parallel implementation of KNN algorithm, namely CUKNN. In the work they use streaming and coalesced data access to improve the performance. Then Sun et al. [9] proposed a distributed approach for KNN on large instances. They introduced two layers of parallelization. In the first layer, they distribute the data to several GPU enabled nodes. In the second layer, which is a CUDA layer, they compute the k-nearest neighbours for each query point. Finally, all the results are combined in the merging step.

III. CONTEXT

A. K Nearest Neighbor algorithm

KNN algorithm is used to classify objects by a majority vote of the k nearest reference objects. Thus, KNN consists of two time consuming processes: distance computing and distance ranking. Our work focuses on these two parts.

The calculation of distances can be fully parallelized because each distance computation is independent. This property makes KNN perfectly suitable for an FPGA based heterogeneous architecture. In our work, after transferring the data from CPU to FPGA, each work-item performs the distance calculation between the query object and a reference object.

Through the distance computing process, the distances matrix between all the query objects and reference objects is ready. Then distance rank process is performed to find the *k* nearest neighbors for each query object.

Bubble sort is a basic algorithm in the field of computer science, and the most popular used quicksort is based on it. We choose to use bubble sort in our KNN algorithm for the following reasons: Each bubble will eventually pick out the smallest element in the current queue. Since every bubble compares only two elements at one time and then moves on to the next pair, this process can be easily parallelized. In KNN algorithm only the *k* smallest distances are necessary for vote, so only k bubbles would be used to handle the sort, which makes this approach more suitable than quicksort in KNN classification. The process is illustrated in Figure1.

Finally voting process is performed to classify the query objects. Considering this process is not the main time-consuming part of KNN algorithm, our works mainly focus on the former two processes.

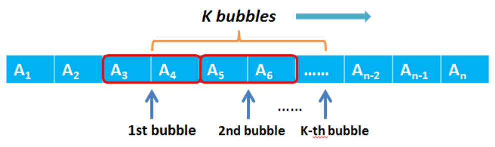

Figure 1. Bubble sort

B. OpenCL Architecture

OpenCL [10] is a framework for parallel programming on heterogeneous platforms, and it is based on a runtime host library and C99 extensions for device programming, adapted to support vectorized data types, synchronization points, and other functionalities. OpenCL is a standard implemented on several hardware architectures by manufacturers. An OpenCL program can be executed on any of those devices with only a handful of modifications, allowing portability. As shown on Figure 2, an OpenCL device is divided into compute units, which execute multiple copies (work-items) of a piece of code (kernel). However, performances can vary according to the compatibility between the program and the architecture of the targeted device.

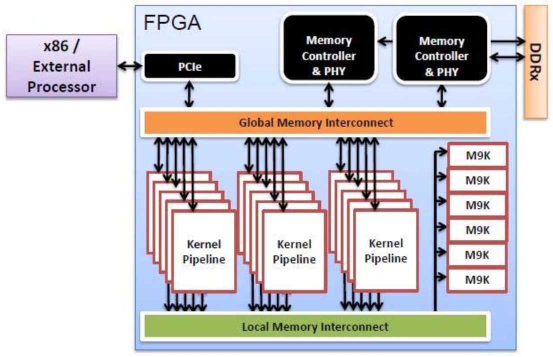

Figure 2. OpenCL platform

The master of an OpenCL architecture (the host) handles the application's data-flow to the connected devices through queues of orders. Data movement is thus explicitly specified by the programmer. This data management relies on a relaxed memory consistency model, with three memory levels: global, local and private. The work-items on a device are organized into work-groups which share a common memory region (local memory) and synchronization points. This memory level plays a role as a cache-coherent memory for the programmer, which eases any data partitioning problem that may arise: every work-item within a work-group can access any data stored in the address space of the local memory. The global memory can be accessed by any work-group and by the host. Access to global memory can be coalesced to reduce latency, provided that accessed pieces of data are adjacent. Nevertheless, each work-item has access to a single private memory region and access to global data is always slower than access to local data.

Two different devices have been used as implementation targets: an FPGA board as the final target, and a GPU as an initial target. The use of a GPU board reduced development time through faster compilation and thus shorter debug and development iterations. It has also served as a reference to compare the results accuracy and computation times of the proposed acceleration solutions.

IV. BUBBLE SORT ENHANCED KNN: AN OPENCL-BASED PARALLEL IMPLEMENTATION OF KNN

A. OpenCL Implementation

Compared with traditional HDL design methodology, work scheduling on hardware resources is delegated to device in OpenCL. The developers only need to consider about the necessary number of work-items in the host program and distribute the workload to the device's resources optimally. There is no need to program a layer of control over the massive cores. Enqueueing far more work than what can be processed at once on the device can then help it optimize its use of hardware resources, and is actually necessary to reach an acceleration factor that balances the communication overhead between the host and the device.

B. Distance Calculation Kernel

This kernel is proposed to maximize the concurrency of the distance calculation handled by each work-item. For the reason that the latency of global memory access is high, it's necessary to take advantage of local memory access.

Distance calculation can be fully parallelized since each distance between a query object and a reference object is independent. This feature contributes to KNN's parallel implementation on FPGA. The distance calculation kernel is parallelized in a data-parallel fashion. The reference dataset is loaded into the local memory so that each work-item can get access to the reference objects. The process is illustrated in Figure 3.

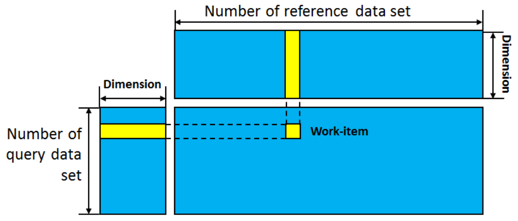

Figure 3. Distance calculation kernel

C. Distance Sorting Kernel

Once the distances matrix between the query objects and the reference objects are gained, distance sort kernel is launched to find the *k* smallest distances in each row of the matrix.

For each query object, we use *k* work-items to find the *k* nearest neighbors by using a partial bubble sort. In bubble sort, each bubble carries the smallest distance out from the head of one row to the end. For example, when the first bubble comes to compare the 3rd and 4th distances in a row, the second bubble can be launched to compare the 1st and 2nd distances, so on and so forth. Thus, there will be 3 stages: bubble increase, bubble saturated and bubble decrease periods. Once the *k*-th bubble reaches the end of each row, the k nearest neighbors are chosen out. All the 3 processes are shown in Figure 4.

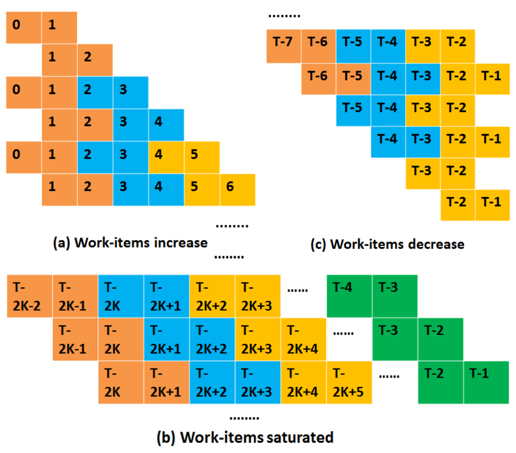

Figure 4. Distance sort kernel

V. Results

In order to verify the approach, both GPU based and FPGA based architectures have been implemented. In this section, the results of these implementations are presented.

A. Test Environment

Three target technologies are considered: a CPU on which the reference software runs, a GPU for development and comparison purposes, and an FPGA. The CPU is an Intel Core i7-3770K running at 3.5GHz, with the reference software written in Matlab. Four cores of the i7 are used during tests. The operating system running on the CPU is a Windows 7 kernel with 64 bits support. The targeted GPU is an AMD Radeon HD7950 with 28 compute units and maximum working frequency 900MHz. The board's global memory consists of a 3 GB GDDR5 memory with 240 GB/s of bandwidth and an access from the host through a PCIe 3.0 connection with x16 lanes. Local memory on the GPU is 32 kB per compute unit.

The FPGA board is a Terasic DE4 with a Stratix IV 4SGX530. Global memory is stored in two DDR2 memory banks, with a maximum bandwidth of 12.75 GB/s to and from the FPGA(at a 400MHz clock rate), and is accessible from the host through a PCIe gen2 4x connection. The PCIe connection has a maximum bandwidth of 500 MB/s per lane, meaning the DE4 board's maximum bandwidth is 2 GB/s. The local memory is implemented through an interconnect structure that gives access to on-chip RAM blocks as simple dual port RAMs, running at 600 MHz. Specifically, M9K RAM blocks are used(RAM blocks of 256x36 bits). Private memory is implemented as flip-flops within the data flow and thus runs at the kernel's frequency. Each kernel implemented on the FPGA board was compiled with Quartus II 64 bits.

B. Hardware Resources Management and Utilization

We use KDD-CUP 2004 quantum physics data set [11] to test the performance of our KNN algorithm. This data set is used to predict the classification of the particles in high energy collider experiments of quantum physics. It stores the physical features and the class label of each particle. The particles are converted into textual records, where unique IDs are assigned. To take full advantage of the hardware resources, we use 20480 records out of 50000 records. The number of dimensions of each record is 64. Since K is usually not large compared to the number of reference objects, we set it to 20 without loss of generality. For the fact that query objects are transferred and processed in batches, I/O time for query objects is included in the comparison. This provides a good compromise between computational speed and hardware constraints (in terms of memory resources). Further gain in efficiency could be achieved by manual fine tuning, as seen in classical FPGA designs. We chose not to do so as it would not yield significant enough benefits compared with the necessary development time but would defeat the purpose of using the OpenCL standard.

Distance computation kernel and distance sort kernel described in Section IV are detailed below. Table I shows resources' usage condition of the KNN system when implemented on the DE4 board.

The kernels were parallelized using several options of Altera's OpenCL Compiler. First, compiler directives can be used to either replicate entire hardware pipelines or to vectorised the kernel execution. When replicating the pipeline, computations can be done independently from one another, while vectorization corresponds to an SIMD work division (Single Instruction, Multiple Data). From empiric observations, vectorization is usually a less resource-consuming optimization than replication. It also eases memory coalescing optimization. However, it is more constraining: vectorization can only be done by powers of two, and be a divider of the total work group size. Besides, it is also possible to unroll any loop included in the kernel through #pragma directives. Loop unrolling uses less memory than a full replication, while giving another way to increase throughput and optimize resource consumption. Loop unrolling, replication and vectorization are 3 parameters that help reach the best compromise between resource utilization, latency and throughput. In our case, the distance computation kernel contains an internal loop, which has been unrolled 8 times. The distance sort kernel has been vectorised twice to make full use of possible resources on the FPGA. Since the bandwidth of global memory access is limited, no more vectorization or pipeline replication can be implemented for fear of performance degradation.

Table I. RESOURCE USAGE

Stratix IV EP4SGX530	
Logic utilization	64%
Combinational ALUTs	135 K/415 K(32%)
Dedicated logic registers	207 K/415 K(50%)
Memory bits	5452 K/20736 K(26%)
DSP block(18-bit)	80/1024(8%)
Clock Frequency	131.42 MHz

C. Comparison

Table II illustrates the performances for each kernel on GPU and FPGA, along with the software reference results. We chose to compute 20 query objects in our tests due to the fact that all the results would be averaged to each query object at last. The GPU accelerated our KNN algorithm by 410 times the speed of the 4-threads CPU implementation, while FPGA achieved 148 times.

When the power consumption is taken into consideration, it is interesting to see that the CPU implementation could merely classify 0.015 query objects per Joule and GPU achieved 4.024, while FPGA 12.056.

From table II we see that although GPU performs better in terms of computation speed, if the performances are averaged to Joule, FPGA becomes superior. The energy efficiency ratio (EER) of FPGA based heterogeneous computing system is 3 times that of GPU based heterogeneous computing system.

Table IVI. PERFORMANCES

Platform	**CPU**	**GPU**	**FPGA**
Feature size/nm	22	28	40
Runtime/ms	10211.05	24.85	69.12
Objects/s	1.96	804.96	289.34
Speedup	/	410	148
Power/w	130	200	24
Objects/J	0.015	4.024	12.056
EER	/	268	804

VI. CONCLUSION

This paper has presented a bubble sort enhanced KNN algorithm using the FPGA based heterogeneous computing system. In order to optimize KNN algorithms respectively on GPU based and FPGA based heterogeneous computing, we verified the new approach's high versatility and portability. Finally we showed that by optimizing KNN algorithm in accordance with the structures and features of the FPGA device, we achieved better performances than the traditional GPU device. The EER of FPGA based heterogeneous computing system achieved 3 times that of GPU based heterogeneous computing system.

REFERENCES

[1] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang,"High-level synthesis for FPGAs: from prototyping to deployment,"vol. 30, no. 4, pp. 473–491, Apr. 2011.

[2] Peng Y, Kou G, Shi Y, Chen ZX (2008) A descriptive framework for the field of data mining and knowledge discovery. Int J Inf Technol Decis Mak 7(4):639–682

[3] Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng,A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., Steinberg, D., 2007. Top10 algorithms in data mining. Knowl. Inf. Syst. 14, 1–37.

[4] Graham N (2009) Improving the k-nearest neighbour algorithm with CUDA. Technical report, School of CSSE, The University of Western Australia.

[5] Garcia V, Debreuve E, Barlaud M (2008) Fast k nearest neighbor search using GPU. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '08). 1–6.

[6] Quansheng K, Lei Z (2009) A practical GPU based kNN algorithm. In: Proc. of the Second Symposium International Computer Science and Computational Technology(ISCSCT'09). Academy Publisher, 151–155.

[7] Satish N, Harris M, GarlandM(2009) Designing e_cient sorting algorithms for manycore GPUs. In: Proc. of the 2009 IEEE International Symposium on Parallel&Distributed Processing. Washington, DC, USA: IEEE Computer Society, IPDPS '09: 1–10.

[8] Liang S,Wang C, Liu Y, Jian L (2009) CUKNN: A parallel implementation of knearest neighbor on cuda-enabled gpu. In: IEEE Youth Conference on Information, Computing and Telecommunication (YC-ICT '09). 415–418.

[9] Sun L, Stoller C, Newhall T (2010). Hybrid MPI and GPU approach to e_ciently solving large kNN problems. Tera Grid Poster. URL http://www.isgtw.org/pdfs/kNNposter.pdf. Accessed 2012 Aug 4.

[10] K. O. W. Group, "The opencl specification," 2011. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[11] KDD Cup 2004 Data (2011) http://kodiak.cs.cornell.edu/kddcup/datasets.htm