Michał KARWATOWSKI, Kazimierz WIATR
AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY, 30 Mickiewicza Ave., 30-059 Krakow, Poland
ACC CYFRONET AGH, 11 Nawojki St., 30-950 Krakow, Poland

The versatile hardware accelerator framework for sparse vector calculations

Abstract

In this paper, we present the advantage of the ability of FPGAs to perform various computationally complex calculations using deep pipelining and parallelism. We propose an architecture that consists of many small stream processing blocks. The designed framework maintains proper data movement and synchronization. The architecture can be easily adapted to be implemented in FPGA devices of a various size and cost - from small SoC devices to high-end PCIe accelerator cards. It is capable to perform a selected operation on a sparse data that are loaded as the stream of vectors. As an example application, we have implemented the cosine similarity measure for the text similarity calculations that uses the TF-IDF weighting scheme. The presented example application calculates the similarity of texts from the set of input documents to documents from the large database. The scheme is used to find the most similar documents. The proposed design can decrease the service time of search queries in computer centers while reducing power consumption.

Keywords: FPGA, sparse vectors, cosine similarity, Zynq, hardware accelerator.

1. Introduction

In modern supercomputer centers, we observe a constant effort to increase the speed of calculations. Also, energy efficiency becomes a significant factor. Both issues can be addressed by using hardware accelerators. They can enhance typical CPU-based processing nodes in most computationally demanding tasks. The most commonly used hardware accelerators for heterogeneous platforms are General-Purpose Graphics Processing Units (GPGPU) and Field-Programmable Gate Arrays (FPGA). While both technologies achieve remarkable runtime speedup, GPGPUs consume significantly more power than FPGAs [1]. The workload of CPU-based servers includes tasks that are suitable for hardware acceleration. An example of such an operation is a computation that is performed on the sparse data. Representing sparse data in the same way as dense data results in storing many redundant zero values. For that reason, the special data formats are better for sparse data representation [2] but, unfortunately, these are cumbersome to perform computing. An FPGA architecture allows overcoming that dilemma. FPGAs are capable to execute simultaneously multiple, not necessarily identical, operations. Profits from such an architecture are visible especially when there are hundreds of parallel operations to be performed. This unique architecture is suitable to build long pipelines, which allows implementation of very complicated algorithms that return results in each clock cycle after an adequate pipeline delay. Moreover, FPGA architecture is not bounded to a specific data size like, for example, 32 bit registers in CPU. The data can be processed with any required width, which allows for optimal fitting and easy manipulations on single bits. Note, however, that many algorithms are not suitable for parallel execution and require sequential processing. Therefore, FPGAs often work as a part of a bigger heterogeneous processing system performing most demanding parallel operations.

2. Sparse data calculations

The sparse data contains many zero values that can be often ignored in calculations, because they do not affect the results. Moreover, performing calculations on the whole data set would require substantial amounts of internal memory. Much research has been done on how to store the sparse data in the most efficient way (e.g. in the work described in [3]). However, even a simple ID – value coding shows satisfactory properties. For example a sparse vector $\vec{A}$ (1) can be represented as a set of pairs $\vec{B}$ (2), where each pair consists of nonzero value and its address.

$$\vec{A} = (0,3,0,0,0,0,5,0,4,0,0,0,0,0,0,2,0,0,0,0,), \tag{1}$$

$$\vec{B} = \{(1,3),(6,5),(8,4),(15,2)\}. \tag{2}$$

The solution described in [4] concentrates on acceleration operations on a single pair of very large sparse matrices, with up to 28,726 nonzero values. Work [5] also uses FPGAs with good results. However, the system that is presented in this paper is more specialized for the specific type of calculations. It performs calculations on smaller sets of data with around 1,500 nonzero values. The architecture proposed in this paper loads a group of vectors into internal memory and processes them with a very long series of vectors from a database. The database size may reach a few hundred thousand of vectors, or more. The design itself does not limit this size. The proposed architecture was used to solve a specific problem of calculating the cosine similarity measure [6] using a term frequency - inverse document frequency (TF-IDF) weighting scheme [6]. The measure was calculated for two text documents represented with use of the Vector Space Model (VSM) [6]. The VSM projects the text into the multidimensional space. As a result, the text is represented as a vector of pairs, consisting of a feature ID and the coefficient of its importance. This method was used to search for the most similar document, in a database of documents, to the inquiry text. Therefore, the calculation scheme assumes that one reference vector is compared with many database vectors. Also, to speed up the query service, a few reference vectors can be processed at the same time.

3. Hardware architecture

The proposed system architecture is designed to use a simple native FIFO interface to transmit data between blocks. Additional signals for status and control of all modules are controlled using register space. The processing system can be easily adapted to work on the heterogeneous server with an FPGA accelerator card connected through PCIe or inside a small SoC device [7]. Connection to software can be maintained using any tool that is capable of sending streamed data and access to internal memory. In our project, we used Xillybus [8] for that purpose. Xillybus contains the Linux operating system based on Ubuntu 12.04 LTS. It provides a simple file I/O interface to Xillybus IP core in programmable logic, to which FIFO queues are connected. We wrote software application in C++ using C++11 standard. Figure 1 presents basic connections between the top modules. There are FIFO queues between the software interface and FPGA – one FIFO for reference vectors data stream, one FIFO for database vectors data stream, and one for each result data stream.

Control over blocks implemented in programmable logic is maintained using register space inside the Command and Status block. From a software point of view, it is accessed using the file I/O functions with the addition of lseek function to select an address. From FPGA point of view, it is a block of distributed memory. Every register in that memory handles control over different part of logic. Each bit in every register is connected to the related function like reset, control and information signals. Besides registers, additional logic performs simple operations like choosing which and how many channels will be used for processing, or when to reload internal memory with a new reference vector. This approach provides easy to use and robust control mechanism.

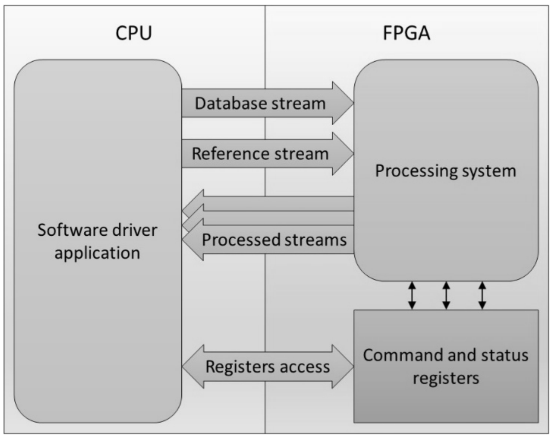

Fig. 1. Control of the processing system and connection to the software application

The processing system consists of many identical processing channels (Figure 2) working in parallel. According to Figure 4, a single processing channel requires two streams of data to operate on. Those streams are fed by the software.

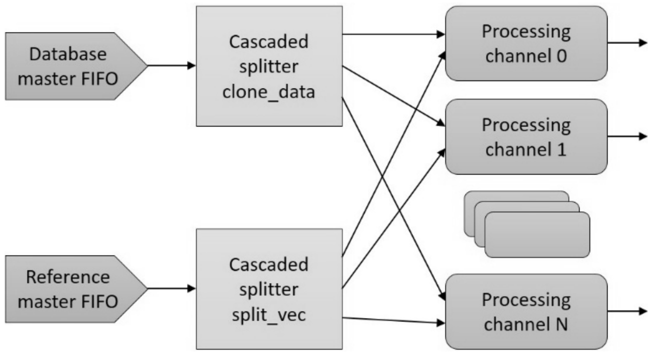

Fig. 2. Top level scheme of the processing system

For the purpose of calculation the similarity measure of two documents, we have only one database. Therefore, there is only one stream of database vectors with which we have to compare a group of reference vectors from the second stream. So, each processing channel needs to receive a copy of the database vectors stream and unique reference vector. For that purpose, we designed a splitter module. It reads the data from the input FIFO and sends to multiple output FIFOs, with two different modes of operation. In the first mode, it clones the input data and writes the same values to every output FIFO, which is suitable for the database data stream. In the second mode, it sends the data to the first output FIFO only, and when the vector ends, it switches to the next channel. Therefore each output FIFO receives a different vector. This mode is suitable for sending reference vectors. Additionally, the splitter has "enabled_channels" input signal, which is fed by control registers, to select where to send the data. This approach allows for multiplying overall bandwidth, which is initially limited by the CPU – FPGA connection interface. However, the number of output channels for a single splitter is limited by timing requirements of a specific device. In our design the single splitter is capable of handling eight channels. To overcome this issue, we created a cascaded version (Figure 3), which consisted of a few single splitters. That allows for creation many more parallel processing channels. The only cost is a slight prolongation of latency that is negligible comparing to the overall processing time.

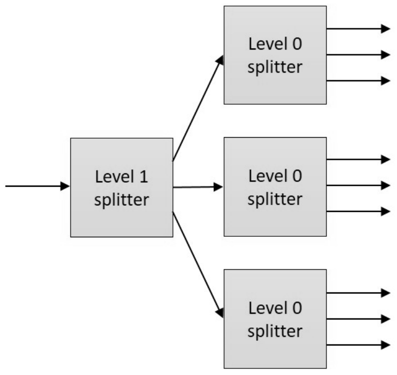

Fig. 3. Scheme of the cascaded stream splitter

The single channel is presented in Figure 4. It consists of two input FIFO queues for loading the data into the process block. The queue for reference vectors is significantly smaller because it is only needed for moving a single reference vector to the internal memory, while the database queue is constantly streaming data. The memory module stores the user data vector so it would not have to be sent each time from the outside of FPGA, saving software – hardware interface throughput. The interface for both read and write operations is compatible with the native FIFO interface. Even though there is no queue inside, the FIFO interface is used to read from the memory. This solution allows more versatile and reusable construction of the process block. When a read pointer reaches the end of the memory, it automatically moves to the first element, so for the output interface its behavior looks like a continuous stream of data. When the processing of one vector is over, another can be sent to the FIFO queue. Then an adequate signal is applied, the read pointer is cleared and another vector is loaded into the memory. The processing speed of a single channel may vary depending on the data, which may cause disproportion among processing channels. To maintain proper data buffering preventing the processing stagnancy, we use FIFO queues. Parameters of a single processing channel can be altered to fit best into available resources of the device the system is implemented in. We can change the length of the FIFOs, the size of internal memory, or switch between using block RAM or distributed RAM for each.

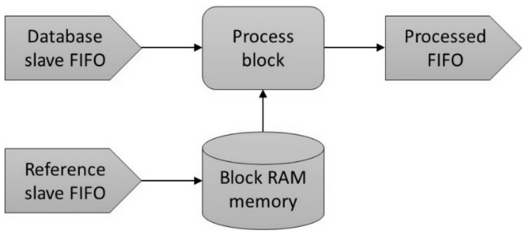

Fig. 4. Scheme of the processing channel

The key part of the whole system is the process block, which reads two streams of data and performs required calculations. For the cosine similarity measure calculation, the algorithm is as follows: through both streams two vectors are continually loaded into internal logic, each data words are compared and if the identical pair is found, their coefficients are used for similarity calculation. When any of the vectors ends, the outcome measure is sent through the output queue, next the vectors are loaded, and the calculation starts all over again. Both reference and database vectors can have headers of fixed but selectable size that are sent before the measurement result in chosen order. Headers mechanism provides additional control information, typically required by the software. The concept of a processing block is generic. Any calculation that can be done on two streams of data may be implemented. Additionally, through the use of pipelining calculation, the outcome is generated in every clock cycle.

4. Experiments and results

The proposed architecture was implemented in two significantly different platforms. The first one was Zadboard, with Zynq-7020 [7], which is a small device for embedded systems. The Advanced eXtensible Interface Accelerator Coherency Port (AXI ACP) was used as an interface between software and hardware. The internal FPGA logic clock frequency was 100 MHz. Another platform was VC707 Evaluation Kit, with Virtex-7 VX485T [9], which is a high-end device. The PCIe gen2x8 was used as an interface between software and hardware. The internal FPGA logic clock frequency was 250 MHz. The system worked faultlessly on both platforms. A migration of the design from one to another does not require any changes in the code. Only the regeneration of the whole project for a different device is required. All additional modifications can be done from the top level module by adjusting a few parameters. For the experiments all parameters were identical for both platforms. The design contained 8 internal processing channels. The performed experiments showed that the calculations on the VC707 were about 2.5 times faster than those on the ZedBoard. That difference directly corresponded to 2.5 higher operating frequency. However, the Virtex-7 device has significantly more logic available and it is possible to implement a higher number of parallel processing channels inside, which would further amplify the throughput. The best results are achieved when all the internal channels are utilized.

5. Conclusion and future work

The proposed architecture can be implemented in devices of a various size and cost. It can accelerate a wide range of operations - starting from calculations on sparse data, through vectors of pairs, to the simple streamed data. The process block is the only part that needs to be customized for the required purpose. The design is very reusable. It can be moved between various devices without code changes. The possibility to easily change the number of channels, and the properties of FIFO queues allow fitting the design in a chosen device, optimizing resource utilization. The performance increase achieved by a single channel was not big. However, thanks to the usage of many parallel channels, the overall performance surpassed software only solutions. The architecture still requires some improvements. In bigger devices, like Virtex-7 VX485T, the number of processing channels may exceed a few hundred. With the current approach, each processing channel has its own output queue. Handling so many channels may be problematic for a software application. Headers mechanism can reduce this problem, but it complicates software application and may lower overall performance. Therefore, there is a need for a less complex solution. It is also possible to store the database in an external RAM memory connected directly to an FPGA to speed up the transmission and release the software from maintaining it. However, this would make the design more platform specific because on ZedBoard, RAM is not connected directly to the FPGA.

This research is supported by the European Regional Development Program no. POIG.02.03.00-12-137/13 PL-Grid Core.

6. References

[1] Lange H., Stock F., Koch A. & Hildenbrand D. (2009, April): Acceleration and energy efficiency of a geometric algebra computation using reconfigurable computers and GPUs. In Field-Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on (pp. 255-258). IEEE.

[2] Jamro E., Pabiś T., Russek P. & Wiatr K. (2014): The algorithms for FPGA implementation of sparse matrices multiplication. Computing & Informatics, 33(3).

[3] Smailbegovic F. S., Gaydadjiev G. N. & Vassiliadis S. (2005, November): Sparse matrix storage format. In Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005 (pp. 445-448).

[4] Sun S., Monga M., Jones P. H. & Zambreno J. (2012): An I/O Bandwidth-Sensitive Sparse Matrix-Vector Multiplication Engine on FPGAs. Circuits and Systems I: Regular Papers, IEEE Transactions on, 59(1), 113-123.

[5] Zou D., Dou Y., Guo S. & Ni S. (2013): High performance sparse matrix-vector multiplication on FPGA. IEICE Electronics Express, 10(17), 20130529-20130529.

[6] Kiela D. & Clark S. (2014, April): A systematic study of semantic vector space model parameters. In Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC) at EACL (pp. 21-30).

[7] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

[8] http://xillybus.com/doc

[9] http://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html

Received: 07.04.2015 *Paper reviewed* *Accepted: 02.06.2015*

Michal KARWATOWSKI, MSc, eng.

He received the BSc Eng. and MSc degrees in electronic engineering in 2013 and 2014 respectively from the AGH University of Science and Technology, Kraków, Poland. Currently PhD student. His research interests include usage of hardware accelerators in complex computations, control and energy efficient systems of small and big scale, mainly using Field-Programmable Gate Arrays.

e-mail: mkarwat@agh.edu.pl

Prof. Kazimierz WIATR, DSc, eng.

He received the MSc and PhD. degrees in electrical engineering from the AGH University of Science and Technology, Kraków, Poland, in 1980 and 1987, respectively, and the DSc degree in electronics from the University of Technology of Łódź in 1999. Received the professor title in 2002. His research interests include design and performance of dedicated hardware structures and reconfigurable processors employing FPGAs for acceleration computing. He currently is a director of Academic Computing Centre CYFORNET AGH.

e-mail: wiatr@agh.edu.pl