IJCSN - International Journal of Computer Science and Network, Volume 7, Issue 4, August 2018
ISSN (Online) : 2277-5420
www.IJCSN.org
Impact Factor: 1.5

Comparative Analysis of Dataflow Engines and Conventional CPUs in Data-intensive Applications

[1]Abdulkadir Dauda; [2]Haruna Umar Adoga; [3]John Francis Ogbonoko

[1]Department of Computer Science, Federal University Lafia,
Nasarawa, Nigeria

[2]Department of Computer Science, Federal University Lafia,
Nasarawa, Nigeria

[3]Department of Computer Science, Federal University Lafia,
Nasarawa, Nigeria

Abstract - High-performance systems are a vital tool for supporting the rapid developments being recorded in software technologies. The recent innovations in software systems have changed the way we see and deal with our physical world. Many applications today, as part of their functions, implement highly data-intensive algorithms such as machine learning, graphics processing, and scientific calculations which require high processing power to deliver acceptable performance. The Central Processing Unit (CPU)-based architectures, which have been used over the years, are not coping well with these classes of applications. This has led to the emergence of a new set of architectures such as Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). These architectures are based on the computing paradigm referred to as dataflow computing in contrast to the popular control-flow computing. In this research, we used the dataflow engines developed by Maxeler Technologies, to compare performance with conventional CPU-based parallel systems by writing a program each for the two platforms, which solves a typical vector operation, and run it on the two platforms. The results of the experiment show that even though the Dataflow Engines (DFEs) we used in our experiments run at the clock frequency of 100MHz, its performance is at par with a quad core CPU which runs at 1.86GHz per core.

Keywords - *HPC, Parallel Systems, Control-flow Computing, Dataflow Computing, FPGAs, DFEs, Maxeler Technologies.*

1. Introduction

Every new generation of computing applications comes with the demand for more compute power the previous ones could offer. As a result, the quest for ever-faster processor chips had always been the objective. Before now, it was generally believed that the number of transistors built into microprocessor chips would double after every eighteen months and that the processing speed would also increase significantly, which is based on the assertion attributed to Gordon Moore, one of the founders of Intel Corporation [1]. This assertion which would later be referred to as Moore's law held for many years and had been the basis for processor chip manufacturers' design philosophy. Over the past decade, however, there has not been a significant performance improvement on a single silicon processor chip [2]. To improve computational performance, multiple processing units referred to as cores are built into a single chip to allow for parallel processing of program segments. The same technique is extended to connecting multiple processor chips and multiple compute nodes. This way, large-scale high speed supercomputers are built. Currently, most of the top speed supercomputers were built based on this technique [3].

Furthermore, the increase in processing power is achieved through various forms of parallelisms i.e. instruction level parallelism, data level parallelism, etc. The conventional CPU-based computing which is based on Von Neumann architecture is referred to as control-flow computing, and it is the type of computing that exploits instruction level parallelism i.e. bit level or thread level parallelism. The compute units in this computing paradigm are designed to fetch instructions from the memory and apply them on some sets of data. This is also known as the fetch-decode-execute cycle. High performance is achieved by building multiple compute units together so that large instructions are split into small segments and executed in parallel.

Control-flow computing performs well in applications involving large and complex instructions. However, it fails to provide the same level of performance on data intensive applications like those involving extensive graphics processing and scientific calculations. This is because most of the processor's time is spent on memory access and

other processing overheads like subroutine call, interrupt management, etc. rather than on the data.

Continuous scaling up of the compute units to achieve high level parallelism will no doubt increase the speed of processing, but will also increase transistor density which will in turn increase power consumption and heat generation and thereby resulting to increased costs of running large-scale high performance computing facility. For example, the Japanese Earth Simulator which was ranked as the top speed supercomputer on the TOP500 list between 2002 and 2004 consumed about 12MW of power for its operations and cooling which was estimated to cost around US$10 million per year [4]. Memory organization of this set of computer systems has also proven to be one of the major constraints to achieving high performance because of the read/write overhead involved in the fetch-decode-execute cycle, with the speed gaps between processors and dynamic random access memory (DRAM) coupled with the access bandwidth on the bus subsystems being the main reason [5].

Alternative to control-flow computing is the dataflow computing which is recently gaining popularity in the high performance computing domain. In this computing paradigm, computations are performed over reconfigurable hardware gates (computing in space) rather than the fetch-decode-execute cycle (computing in time). The approach allows the hardware elements to be laid as dataflow graph which serves as datapaths configured to reflect the specific operation to be executed. In this paradigm, though the programmer has to write a program, but instead of the program to control the movement of data, it configures the hardware, therefore as data comes to the dataflow node, it gets processed and passed to the next node or moved as output.

The concept of this approach was discussed decades ago but the lack of enabling technology hindered its realization until recently. This approach requires hardware elements which are reconfigurable at runtime. Recently, field programmable gate arrays (FPGAs) have been shown to support this great concept, with Maxeler technologies one of the leading company in this drive.

The dataflow architectures are not only a good alternative to control flow architectures in terms of performance but also in terms of energy efficiency and area per unit square a typical high performance system occupies, which are some of the major issues with high performance computing facility. Research has shown that as the performance of supercomputers increases, so do power consumption and the sizes of buildings required to host them [6].

2. Literature Review

The debate between Gene Amdahl and Dan Slotnick back in 1967 marks an important event in the historical evolution of parallel processing. Amdahl, considering the need for proper data management housekeeping inherent in serial processing and the fact that so much time and efforts are required to organize a well working parallel program argued that serial processing should always be the option. Slotnick in the other hand believed that performance improvement could be gained in parallel processing if new approaches to problem solving could be devised [1].

Ever since, the two arguments have continued to be relevant in the parallel processing literature. It is widely acknowledged that designing algorithms for parallel processing is extremely challenging and often only parts of most problems can run in parallel. This limitation has proven to be one of the major bottlenecks of high performance computing today which corresponds to Amdahl's argument which later became known as Amdahl's law that it is immaterial how many compute resources are used, the serial part of the problem will always limit performance gain. According to [7], Amdahl's law can be summarized as: Fraction of time on serial code, S, limits speed up to

$$SP = 1/ (S+ P/N)$$

where P stands for the fraction of time on parallel code and N the number of processors.

Since the emergence of microprocessors in the late 1970s, electronic computers have witnessed a steady performance improvement of about 25% per year [1]. This is owing to the tremendous technological advancements in the area of electronic engineering and computer architecture. However, despite these developments, the computational performance on a single semiconductor processor has not significantly improved for over a decade now [2].

Many researchers believe that the limit for which frequency can be scaled up in a uniprocessor chip has been reached and this is largely due to the inability to further improve in the areas such as bit level parallelism mechanism and lower heat generation that is associated with high transistor density. Multi-core and multi-processor systems have been used over the years to provide the needed speedup but the computational paradigm used in this model does not reflect the working principles of many application areas. As a result, manufacturers have begun to augment CPU-based systems with special purpose hardware accelerators such as GPUs

and FPGAs as a way to improve performance over CPU-based systems [8].

2.1 Computing Paradigms

Over the years, computations have been structured on control-flow paradigm which is based on Von Neumann architecture, and it has been used to implement most of the mathematical models that are used in computations today, which is why it is seen by many as the ideal model for computations. Control-flow architecture is suitable for the classes of applications which are generally referred to as multiple instructions multiple data (MIMD) applications which need to fetch and execute for every data operation one instruction. However, its measure drawback is the need for frequent memory access which significantly slows down the entire system. Reading and writing operations involving multiple types of memories with different architectures and speed requires the mapping of one address location to another which is very time consuming. [9] argue that no matter how fast and parallel today's control-flow computers can be their mode of operation is basically slow. This is especially due to frequent read/write of data through memory hierarchy and synchronization amongst cooperating threads. In contrast to Von Neumann architecture, dataflow architecture does not have program counter and the execution instruction is based on the availability of input arguments. This type of approach is best used with applications which are characterized as single instruction multiple data (SIMD) and have significant data-level parallelism like the matrix-oriented computations such as scientific computing and graphic processing [1]. These two classifications are part of what is generally referred to as Flynn's taxonomy where computer architectures were classified based on the instruction stream and data stream they can process at a given time [10]. Figure 1 shows the complete classifications.

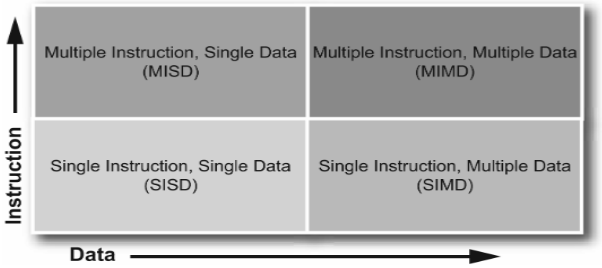

Fig. 1 Flynn's taxonomy [10].

2.2 Programming Models for CPU-based Parallel Systems

Parallel systems are systems which generally have more than one processing units. These systems can be categorized into two, namely, shared memory systems and distributed memory systems. For each of these classes, there are a number of models and techniques developed to effectively program them. The following are some of the common approaches.

- **Shared Memory Systems**: As noted earlier, most parallel architectures such as multicore and multiprocessor systems have a central memory that is accessible by all the processing units. This, therefore, allows for the use of global variables which can be accessed and updated directly by the processors. Many programming languages that are used today include libraries for programming shared memory parallel systems. This allows programmers to initiate as many threads as their applications require, which can run on the multiple processors in parallel. These languages include Java, C++, and Python. There is also a standard called Open Multi-Processing (OpenMP) which is an application programming interface that is used for programming large shared memory machines with programming languages like C, Fortran, and C++ to develop parallel applications [11].

- **Distributed Memory Systems**: Computer architectures like clusters and supercomputers which are also referred to as multicomputer have multiple processing units with independent memories which are connected together by an interconnection network. Since each processor has access to only its memory, accessing remote memories happens by sending and receiving messages across the network. This technique is called message passing and programming them requires a completely different approach from the shared memory systems. The most common standard for programming distributed memory systems is the Message Passing Interface (MPI) which can be used with many programming languages like C, C++, and Fortran etc. MPI has two major advantages which are portability and performance and an MPI program can run both on distributed and shared memory systems [12]. However, memory management remains a major challenge from the programmers' point of view because of the complex nature of communications going on within the system. Nowadays, a number of middleware have been developed which are used to make the communications between the processing nodes transparent to the programmers, thereby reducing the burden of communications management on them.

2.3 Programming Models for Dataflow Systems

The use of FPGAs as computing elements for dataflow architectures has for long been recognized. This is due to its power efficiency which has been shown to be better than GPUs which have been used to accelerate many applications involving graphics processing for over a decade now [13]. In dataflow programming model, program logic is mapped onto the hardware elements which are configured spatially as dataflow graphs. The operation at a dataflow node is performed as soon as the required number of inputs is met [14], and the result is sent to the next node or moved as the final output depending on the configuration.

Over the years, hardware description languages like VHDL and Verilog have been used to program dataflow devices. Although, these programming languages are like most human readable source code languages, they require special skills in dealing with hardware elements and configurations which make them very tedious to use and programs developed using them are sometimes error prone.

In order to improve programmability and platforms integration, OpenCL which is a programming framework based on C programming language that many programmers are familiar with was introduced to provide support for a wide range of FPGA devices. Recently, Altera Corporation, one of the leading FPGA chip manufacturers, introduced OpenCL SDK for their FPGA devices to ease programming difficulties associated with hardware description languages [15].

In an attempt to push the use of heterogeneous architectures to achieve maximum performance even further, Maxeler Technologies, the manufacturers of Maxeler dataflow engines created a platform which enables programmers to use their SLiC (Simple Live CPU) API to integrate CPU-based programs and dataflow engines based on FPGAs.

The SLiC interface allows programs written in languages like C, C++, Python, and MATLAB to communicate with the MaxJ program which is an extension of Java programming language used to program the data flow engines [16]. Figure 2 shows the Maxeler Maximum Performance Computing architecture.

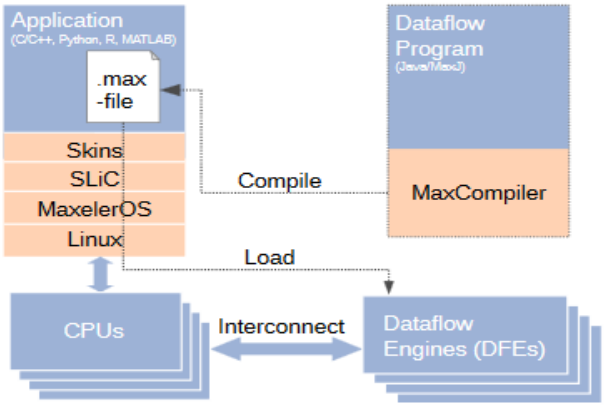

Fig. 2 Maxeler MPC system architecture [16].

2.4 Maxeler's Dataflow Computing Architecture

The Maxeler Technology's solution uses FPGAs to construct dataflow engines (DFEs) with dedicated local memories, which are attached to a CPU through a fast interconnection medium. Figure 3 shows the Maxeler dataflow engine architecture.

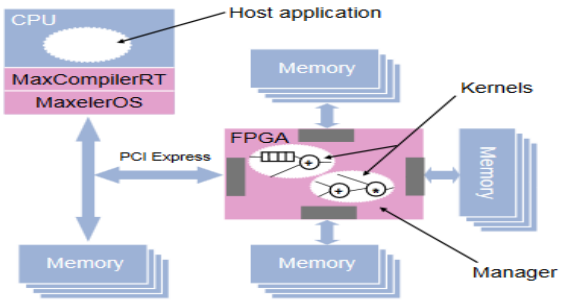

Fig. 3 Maxeler dataflow system [17].

The Maxeler's MPC-X series dataflow nodes are heterogeneous systems combining CPUs and DFEs which are designed to be used as part of a high performance computing facility. The nodes can be used to provide dataflow computing capability in a multi-user and multi-application environment such as the cloud or HPC cluster. CPU clients can access the dataflow nodes through the extremely fast Infiniband connectivity and the multiple DFEs are interconnected together using MaxRing interconnect. Dynamic allocation of DFEs to CPU threads is managed by the Maxeler software [18]. Figure 4 shows the internal structure of the MPC-X series node.

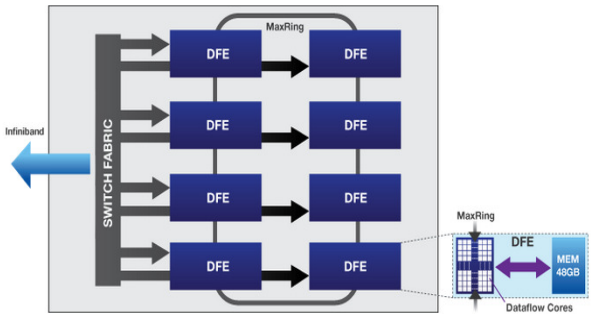

Fig. 4 Maxeler's MPC-X series node architecture [18].

3. Design and Implementation of Vector Operation on CPU Architecture

A vector, in this regard, refers to a one-dimensional matrix, which is also called a row vector and it can be represented as $[A] = [a_1\ a_2\ a_3\ a_4 \ldots a_n]$ where n = Dimension of the row vector.

Operation of the form $(a_i * 2 + a_{i-1})$ on row vectors means that for every element in the vector, the operators * and + will be applied and when implemented on a computer system and large size of n is used, will mean applying this simple instruction to multiple data which is the case with most SIMD applications.

3.1 Algorithm Design

Algorithm 1

```
1     START
2     CREATE VECTOR A
3     CREATE VECTOR B
4     INPUT DATA TO VECTOR A
5     SET i = 0
6     REPEAT STEPS 7 TO 11 WHILE i < n
7     IF i > 0 THEN GOTO STEP 10
8     SET B[i] = 0 + A[i] * 2
9     IF i = = 0 THEN GOTO STEP 11
10    SET B[i] = A [i-1] + A[i] * 2
11    SET i = i + 1
12    PRINT A & B
13    STOP
```

The algorithm above represents the simple and serial steps required to implement the vector operation in our application, however, for it to run and exploit the processing power of parallel systems, multiple threads would be created and used to work on different sections of the vector.
In order to achieve that, the vector A will be partitioned according to the following algorithm.

Assuming:

Size of A = n
Number of Threads = number_of_threads
Portion of A = portion_of_A

Algorithm 2

```
1     portion_of_A = n / number_of_threads

2     Thread1 start_point = 0 * portion_of_A
3     Thread1 end_point = (0 + 1) * portion_of_A

4     Thread2 start_point = 1 * portion_of_A
5     Thread2 end_point = (1 + 1) * portion_of_A

6     Thread3 start_point = 2 * portion_of_A
7     Thread3 end_point = (2 + 1) * portion_of_A
.
.
m     Threadn start_point  =  Threadn-1 end_point
m+1   Threadn end_point = n
```

As soon as the partitioning is done, each thread will then perform its task according to the following algorithm.

Algorithm 3

```
1     START
2     SET i = start_point
3     REPEAT STEPS 4 TO 8 WHILE i < end_point
4     IF i > 0 THEN GOTO 7
5     SET B[i] = 0 + A[i] * 2
6     IF i = = 0 THEN GOTO 8
7     SET B[i] = A[i-1] + A[i] * 2
8     SET i = i + 1
9     STOP
```

3.2 Design and Implementation of Vector Operation on Dataflow Engine

Dataflow architectures, as mentioned in the previous sections, are suitable for applications which are characterized as single instruction multiple data (SIMD). In order to implement our application on the dataflow architecture, we identified the part of the program where there is a substantial amount of data parallelism which can be seen from the portion of algorithm 1 as shown below.

```
5     SET i = 0
6     REPEAT STEPS 7 TO 11 WHILE i < n
7     IF i > 0 THEN GOTO STEP 10
8     SET B[i] = 0 + A[i] * 2
9     IF i = = 0 THEN GOTO STEP 11
10    SET B[i] = A [i-1] + A[i] * 2
11    SET i = i + 1
```

This part of the algorithm shows where the operations + and * are repeated for n number of data which can be translated into the dataflow graph as shown in figure 5.

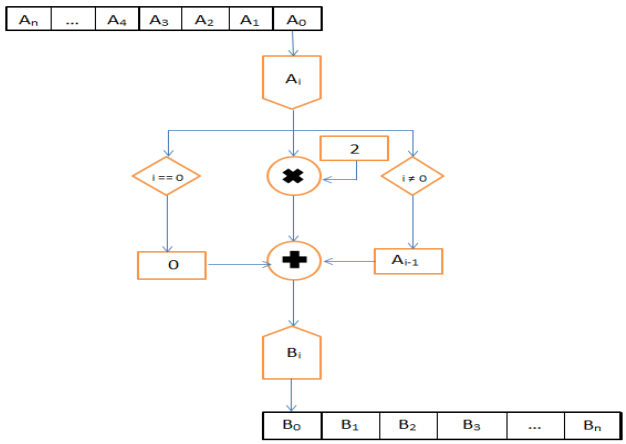

Fig. 5 Dataflow diagram for the vector operation.

The data values A_0, A_1, A_2, ... A_n are streamed to the dataflow engines which are processed according to the specified configurations. The advantage of this is that, the looping that is inherent in the algorithm is eliminated and all that is needed is the data values to be pushed to the dataflow nodes which happen in almost every clock cycle. As the inputs are streamed in, the outputs B_0, B_1, B_2, … B_n are passed out at the other end which makes dataflow architectures more efficient in terms of handling data intensive applications.

3.3 Dataflow Program in MaxJ

The Maxeler high performance platform allows programmers to develop dataflow applications using its general purpose MaxCompiler programming tool. In MaxCompiler environment, every dataflow application is split into three parts i.e. the kernel, the manager, and the CPU codes. The kernel defines the computational component and provides the implementation for the application in hardware. An application can have more than one kernel depending on its complexity. The manager provides connectivity between the kernels, CPU, and engine RAM, and lastly, the CPU code interacts with the dataflow engine to read and write data between them.

The kernel and manager codes are written in MaxJ programming language which is an extension of Java developed as part of the Maxeler's dataflow computing solution and the CPU code is written in C programming language. All the three parts of the Maxeler application can be written within an integrated development environment called the Maxeler Integrated Development Environment (MaxIDE). The MaxIDE integrates all the three parts of the application and makes it easy for the developers to build and test their applications.

In the CPU code, we created two arrays, one for input data and the other for output data. The size of the array is the only input needed as the inputs to the dataIn array are generated automatically using random number generator in C programming language.

Once the data are generated, the control is passed to the dataflow engine. The execution time is measured by taking the time difference between the time the control was passed to the dataflow engine and the time it is returned to the CPU. The kernel code contains the configuration statements where the vector operation A[n-1] +A[n] * 2 is mapped onto the hardware which corresponds to the dataflow diagram in figure 5.

The manager code manages the interaction between the CPU code and the kernel code. The data dataflow engine's clock speed is set to 100MHz.

4. Experiment, Results, and Analysis

4.1 System Resources

It is important to start with the breakdown of the resources available on the two computing platforms in order to have a clearer understanding of the outcome of the experiment because no matter how fast a computing architecture or model is, system resources like the memory, the bus subsystem, and the internetwork facilities largely affect its overall performance. As such, we explain the composition of the system resources available on the two platforms below.

- **CPU-based system:** The CPU-based parallel system used for this experiment is a multicore system consisting of an Intel Celeron N2920 processor with four processing units (cores) running at 1.86GHz each and a memory of 4GB. The system has a 64-bit Windows 10 operating system installed which is designed for x64 processors architecture.
- **Maxeler platform:** The Maxeler 1U MPC-X blade hybrid system which is a complete platform consisting of all the hardware and software resources required to run the dataflow program was used. The system consists of 4 Maia dataflow engines (DFEs) with 48GB of on-board RAM each. The control node has MaxOS installed which manages the resources on the platform and is connected to the MPC-X via a 40Gbit Infiniband connection with two 10-core Xeon

processors and 64 GB of RAM. The FPGA hardware core is set to run at the clock frequency of 100MHz which is much less than the clock frequency obtainable on a typical CPU but still delivers the remarkable performance that can be seen in the results we recorded. Table 1 gives the summaries of the resources available.

Table 1:

	CPU-Based Platform	**Maxeler Platform**
System	CPU-Based Quad Core System	1U MPC-X Blade
Number of Cores/DFE	4	4 Maia DFE
Clock Rate of each Core/DFE	1.86 GHz per core	100 MHz per DFE

4.2 Experiment and Results

The experiment was setup using the resources summarized in table 1 for the two platforms. The size of the data was varied between 300,000 and 1,500,000 and each time, the execution time of the process was recorded. The performance was calculated by dividing the size of the data (N) by the time (T) it took the process to complete. Table 2 shows the results recorded for experiment.

Table 2:

	Size of Data (N)	Maxeler DFE Exe. Time (T) in Seconds	DFEs Performance (N/T)	CPU-Based Sys Exe. Time (T) in Seconds	CPU-Based Performance (N/T)
1.	300,000	0.030000	10000000.00	0.035907	8354916.869
2	400,000	0.030000	13333333.33	0.044483	8992199.267
3	500,000	0.030000	16666666.67	0.056821	8799563.542
4	600,000	0.040000	15000000.00	0.056543	10611393.10
5	700,000	0.040000	17500000.00	0.058890	11886568.18
6	800,000	0.050000	16000000.00	0.056790	14086987.15
7	1,000,000	0.040000	25000000.00	0.058548	17080002.73
8	1,200,000	0.050000	24000000.00	0.058619	20471178.29
9	1,300,000	0.060000	21666666.67	0.060969	21322311.34
10	1,500,000	0.060000	25000000.00	0.073361	20446831.42

4.2 Comparison of Results

In the results from the experiments, the DFE has performed relatively better even though it runs at the clock frequency of 100MHz compared to the quad core CPU which runs at 1.86GHz per core. This is despite the network latency, which we expected to slightly affect the results as we included the communication time between the CPU and the DFE on the Maxeler system in the execution time which occurs over the fast Infiniband interconnection medium. Figure 6 shows the chart which represents the performance of the two platforms from the experiment. On the chart, the vertical points represent the performances, which were obtained by dividing the sizes of the data (N) by the time (T) it takes to process them, and the horizontal points are the number of data tested in the experiment.

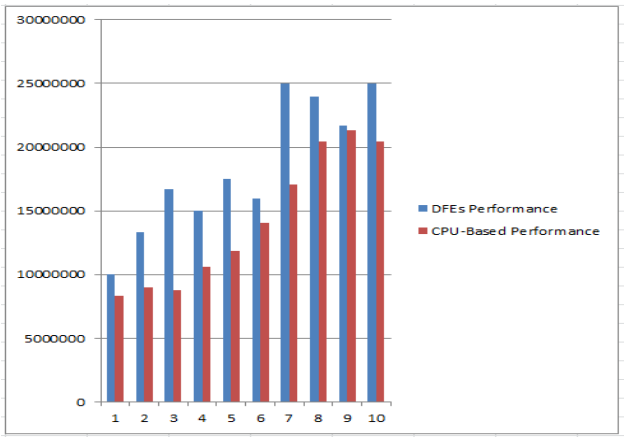

Fig. 6 Performance of DFEs versus CPU from experiment

4.3 Analysis of Performance

The results obtained from the experiment show that the dataflow engines perform better in terms of processing the vector operation. Even though the CPU-based system has more powerful resources, its mode of operation makes it difficult for it to achieve better performance because of the frequent memory access and the associated overhead. In the dataflow system, the instructions are split into simple operations and executed at the compute elements in parallel. So, instead of repeatedly fetching the instructions and the data, which happen after many clock cycles on a CPU system, the data are pushed to the computing components at approximately every clock cycle.

The results also show that, as the size of the data increases, the execution time on the dataflow system is not significantly affected, unlike on the CPU-based system which tends to increase proportionally to the size of the data.

5. Conclusions

In this work, we highlighted some of the important advancements in the area of high performance computing. We discussed the different architectures of the CPU-based systems and the incredible performance that is gained over single core processors by joining multiple cores together. We also pointed out some of the disadvantages, such as heat generation and power consumption and the associated costs, of the continued increase in the number of processing units in that manner. As an alternative to control flow computing, we discussed dataflow computing and how it can be used to accelerate certain classes of applications which are characterized as SIMD. As the main aim of our research is to compare performance, we implemented a vector operation on both CPU-based parallel systems and Maxeler dataflow engine and used it for the experiment. The outcome of our experiments shows that the DFEs perform better even though they run at a lower clock frequency than the CPUs.

References

[1] Hennessy, J.L. and Patterson, D.A., 2011. *Computer architecture: a quantitative approach*. Elsevier.

[2] Pell, O. and Mencer, O., 2011. Surviving the end of frequency scaling with reconfigurable dataflow computing. *ACM SIGARCH Computer Architecture News*, *39*(4), pp.60-65.

[3] Flynn, M.J., Pell, O. and Mencer, O., 2012, August. Dataflow supercomputing. In *22nd International Conference on Field Programmable Logic and Applications (FPL)* (pp. 1-3). IEEE.

[4] Feng, W.C., Feng, X. and Ge, R., 2008. Green supercomputing comes of age. *IT professional*, *10*(1), pp.17-23.

[5] Mahapatra, N.R. and Venkatrao, B., 1999. The processor-memory bottleneck: problems and solutions. *Crossroads*, *5*(3es), p.2.

[6] Feng, W.C. and Cameron, K., 2007. The green500 list: Encouraging sustainable supercomputing. *Computer*, *40*(12), pp.50-55.

[7] Gustafson, J.L., 1988. Reevaluating Amdahl's law. *Communications of the ACM*, *31*(5), pp.532-533.

[8] Sundararajan, P., 2010. High performance computing using FPGAs. *Xilinx White* Paper: FPGAs, pp.1-15.

[9] Milutinovic, V., Salom, J., Trifunovic, N. and Giorgi, R., 2015. *Guide to DataFlow Supercomputing: Basic Concepts, Case Studies, and a Detailed Example*. Springer.

[10] Akhter, S. and Roberts, J., 2006. *Multi-core programming* (Vol. 33). Hillsboro: Intel press.

[11] Terboven, C., Schmidl, D., Jin, H. and Reichstein, T., 2008, May. Data and thread affinity in openmp programs. In *Proceedings of the 2008 workshop on Memory access on future processors: a solved problem?* (pp. 377-384). ACM.

[12] Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L. and Chapman, B., 2011. High performance computing using MPI and OpenMP on multi-core parallel systems. *Parallel Computing*, *37*(9), pp.562-575.

[13] Chalamalasetti, S., Margala, M., Vanderbauwhede, W., Wright, M. and Ranganathan, P., 2012, April. Evaluating FPGA-acceleration for real-time unstructured search. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on* (pp. 200-209). IEEE.

[14] Lee, B. and Hurson, A.R., 1993. Issues in dataflow computing. *Advances in computers*, *37*, pp.285-333.

[15] Gao, S. and Chritz, J., 2014, December. Characterization of OpenCL on a scalable FPGA architecture. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)* (pp. 1-6). IEEE.

[16] Maxeler Technologies, 2013. "MaxCompiler white paper." [Online]. Available from: https://www.maxeler.com/media/documents/MaxelerWhitePaperProgramming.pdf [Accessed: 25th October, 2017].

[17] Maxeler Technologies, 2011. "MaxCompiler white paper." [Online]. Available from: http://www.maxeler.com/media/documents/MaxelerWhitePaperMaxCompiler.pdf [Accessed: 24th October, 2017].

[18] Maxeler Technologies, 2016. *MPC-X Series.* [Online]. Available from: https://www.maxeler.com/products/mpc-xseries/ [Accessed: 28th November, 2017].

First Author

Abdulkadir Dauda was born in Lafia, Nasarawa State of Nigeria on the 25th of October 1982. He obtained the Bachelor of Science degree in Computer Science from the Usmanu Danfodiyo University Sokoto, Nigeria in 2006. He worked with the Nigerian Judiciary as a Programme Analyst from February 2009 to April 2014 when he joined the Federal University Lafia as a Graduate Assistant. In 2015, he proceeded to the University of Bedfordshire, United Kingdom for his Masters of Science Degree which he completed in January 2017. He currently works as an Assistant Lecturer in the department of Computer Science, Federal University Lafia, Nigeria. His research interests are in the area of High-Performance Computing and Distributed Systems.

Second Author

Adoga, H. U. holds a Bachelor of Engineering (B.Eng.) in Electrical & Electronics Engineering from the University of Maiduguri, Nigeria, with specialization in data communications and networks. He also holds a Master of Science (MSc.) degree in Computer Science, from the University of Hertfordshire, England. He is currently a lecturer with the Department of Computer Science, Federal University Lafia, Nigeria. His research interests are in the areas of Software Defined Networking (SDN), IOT and Distributed Systems. He is a registered member of the Institute of Electrical and Electronics Engineers (IEEE), the Nigeria Computer Society (NCS), and the Nigeria Society of Engineers (NSE). As a CCNP professional, Haruna is also fascinated by design and configuration of computer networks.

Third Author
Ogbonoko, J. F. holds a Bachelor of Science (BSc.) degree in Computer Science from the Benue State University, Makurdi, Nigeria. He also holds a Master of Science (MSc.) degree in Software Systems and Internet Technology from the University of Sheffield, United Kingdom. He currently lectures in the Department of Computer Science, Federal University Lafia, Nasarawa State, Nigeria. His research interests are in the area of Software Engineering, Internet of Things, and Big Data.