Research Article

New Benchmarking Methodology and Programming Model for Big Data Processing

Anton Kos, 1 Sašo Tomažič, 1 Jakob Salom, 2 Nemanja Trifunovic, 3 Mateo Valero, 4 and Veljko Milutinovic 5

1 Faculty of Electrical Engineering, University of Ljubljana, Tržaška Cesta 25, 1000 Ljubljana, Slovenia
2 Mathematical Institute of the Serbian Academy of Sciences and Arts, Knez Mihailova 36, 11001 Belgrade, Serbia
3 Maxeler Technologies Ltd., 1 Down Place, London W6 9JH, UK
4 Barcelona Supercomputing Center, Carrer de Jordi Girona 29, 08034 Barcelona, Spain
5 School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia

Correspondence should be addressed to Anton Kos; anton.kos@fe.uni-lj.si

Received 18 December 2014; Accepted 9 March 2015

Copyright © 2015 Anton Kos et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Big data processing is becoming a reality in numerous real-world applications. With the emergence of new data intensive technologies and increasing amounts of data, new computing concepts are needed. The integration of big data producing technologies, such as wireless sensor networks, Internet of Things, and cloud computing, into cyber-physical systems is reducing the available time to find the appropriate solutions. This paper presents one possible solution for the coming exascale big data processing: a data flow computing concept. The performance of data flow systems that are processing big data should not be measured with the measures defined for the prevailing control flow systems. A new benchmarking methodology is proposed, which integrates the performance issues of speed, area, and power needed to execute the task. The computer ranking would look different if the new benchmarking methodologies were used; data flow systems would outperform control flow systems. This statement is backed by the recent results gained from implementations of specialized algorithms and applications in data flow systems. They show considerable factors of speedup, space savings, and power reductions regarding the implementations of the same in control flow computers. In our view, the next step of data flow computing development should be a move from specialized to more general algorithms and applications.

1. Introduction

Big data processing and big data applications are shifting the computing paradigms, computing concepts, and treatment of data. Big data processing is becoming increasingly important in cyber-physical systems (CPSs). CPS is a complex system integrating computation, communication, and physical processes. It can be seen as a sort of upgrade to its building blocks and elements, which enables coupling of cyber and physical worlds. Some of the technologies closely connected to the CPS are wireless sensor networks, Internet of things, and cloud computing. Wireless sensor networks, with their sensing capabilities, are considered to be a vital component of the emerging CPS [1]. Similarly, cloud computing provides computation capabilities, and Internet of Things (IoT) provides communication capabilities, and so forth. One common term that connects the abovementioned technologies and systems, including the CPS, is big data.

Managing big data is a many-sided problem. In addition to its volume, big data exhibits other unique characteristics that differentiate it from traditional data. For instance, big data analysis requires distinct processing; therefore, the design of scalable big-data systems may face a series of technical challenges [2]. Big data comes in various forms; from unstructured data to highly structured data streams. It is difficult to manage these volumes and forms of data and it is even more difficult to make sense of it by extracting some useful knowledge [3]. The majority of efficient big data
systems and applications require a problem-specific solution and in many times also a shift in the traditional computing paradigms and concepts.

Until recently big data was a reality only in highly specialized fields such as meteorology and geophysics. Now, big data applications are starting to penetrate more general research fields such as biology, medicine, and politics. Big data is becoming a part of our daily lives.

The data volume growth is exponential. Recent studies from 2012, summarized in [4], show that the amount of data doubles every two years. It is predicted that the global amount of data will grow by a factor of 300, from 130 exabytes in 2005 to 40,000 exabytes in 2020. In recent years, the processing power growth ratio has been lower than the data volume growth ratio. With the expected wide-spread usage of data-collection technologies such as IoT and WSN, the data growth ratio may increase even more.

How can we handle and process such vast amounts of data? A possible answer is the change of computing paradigm and implied change of programming model. Problems including big data imply that it might be more rational and reasonable to put the focus on data and not on the processes around it. A good match is the data flow computing paradigm and the programming model that can be implemented on the data flow computers. In this paper we present the data flow approach to big data processing. We include the examples of algorithms and applications, where we show that data flow computing outperforms the traditional control flow computing models for one or more orders of magnitude.

This paper is organised in the following way. In Section 2 we begin with the motivation for the study and continue with a brief presentation of the related work in Section 3. The strengths of data flow computing against control flow computing are explained in Section 4. The principal ideas of this paper are presented in Sections 5 and 6. In the former we propose a new benchmarking methodology that is big data oriented and data flow computers friendly; the new methodology treats data flow computers more fairly in comparison with control flow computers. Section 6 comprises the presentation and the demonstration of the data flow programming model. We used the model developed for Maxeler data flow systems, the current leader in this field. Section 7 is dedicated to a short list of recent performance data of various algorithms implemented on a data flow system. The current leader in this field. Section 8 discusses the need for more general data flow applications and gives one such example. We conclude with Section 8.

2. Motivation

The primary motivation of this paper is to open a general discussion about the data flow computing concept and programming model, with special focus on big data problems.

Another motivation is to present an alternative benchmarking methodology for big data applications and computers. In our opinion the new benchmarking methodology is fairer to data flow computers than the existing ones. We would also like to stress that with focusing on data, rather than on the process, the new data flow computing paradigm requires some minor and some major mind shifts. Perhaps the most notable and challenging of all is the shift in the programming model.

Besides the general facts and findings about the advantages of data flow computing mentioned, we are also motivated to present the achieved speedups of selected algorithms in various research fields.

3. Related Work

Data flow computing is not a new idea. Data flow concept has been proposed and proven in [5–7]. There were several reasons why a proven concept did not result in data flow computers, and the most important among them are as follows.

(i) The development stage of programmable hardware technology, such as today's FPGA, was not high enough.

(ii) The technology and tools for the system software were not yet ready.

(iii) Data flow computers show their full strength and capabilities with big data problems and applications, which did not exist at the time. Consequently, data flow computers could not demonstrate their superiority.

The work of Flynn et al. [8] suggests that each computing paradigm and its programming model can be characterized through its qualitative and quantitative measures. The given viewpoint suggests that big data problems should be measured not in petaflops but in petadata (the current area of supercomputing is referred to as petascale era [8]). In this work the Maxeler data flow programming model and computers are discussed in terms of their quantity and quality aspects.

There are other similar approaches to data flow computing [9], but to our knowledge they do not reach the quality and flexibility of Maxeler's solutions. For the above reasons we have decided to present the Maxeler data flow computing and programming model. Advantages of data flow computing paradigm are evident from the survey of the most recent performance data of various algorithms implemented on a data flow computers [10]. Some of the most interesting results are presented below.

For the Gaussian mixture models algorithm the authors of [11] managed to get the speedup of 517 times over a CPU computer and the speedup of 28 times over GPU computer. They were using a 150 MHz data flow computer. For the genetic sequence algorithm the authors of [12] managed to prove that a 150 MHz data flow computer performs 13 times faster than a 20 core CPU computer and four times faster than GPU computer. The speedup of 163 times over the quad core CPU was presented at running the Monte Carlo simulation algorithm on a data flow computer [13]. In addition to the speedup, the power reduction of the factor 170 was demonstrated. Even the 200 MHz PCIe card entry model of the data flow computer exhibits a speedup of up to 20 times
over a quad core CPU for the network implementation of the bitonic merge sort algorithm [14]. The authors of [15] show that two 150 MHz data flow nodes (2U) outperform 1,900 CPU cores at the calculation of the velocity stress form of the elastic wave equation with the 3D finite difference algorithm.

4. The Strength of Data Flow Computing

To indicate the strengths of data flow computing in comparison with control flow computing, we briefly illustrate the major differences between the control flow and the data flow concepts of data processing.

(i) Control flow focuses on the processes and operations that are required to complete them. Data enter and exit the process on an as-needed basis. For example, when the process requires some data, it is read from the memory. The process uses the data in the defined manner, possibly transforming it, and when needed the results are written back to the memory. The process flow can be significantly influenced by the intermediate results and the data being used.

(ii) Data flow focuses on data streams. Streams originate from the data source(s) and are passed to the destination(s) through the data flow computer using (predefined) data paths between the components that transform the passing data. The process can be modeled as a directed graph of the data that flows between operations.

The primary strength of data flow computers is their potential to accelerate data flows and the execution of application loops for one or many orders of magnitude. The exact order depends on the degree of data reusability within the flow or within the application loop. This acceleration is possible because the data flow program code is compiled down to the gate level, much below the machine code level used in control flow computers. Compiling the code to the gate level leads to several important and advantageous effects: lower execution time, less power dissipation, and smaller chip size (area).

One important question arises: can a big data application benefit from the abovementioned strengths of data flow computing? We illustrate some of the many possible scenarios that would give a positive answer to the above question.

(i) Confined space: a big data application is constrained to run in the confined space, for instance, on an airplane, on a ship, and at the remote research station. Given the same chip area, the data flow application performs in less time than a control flow application.

(ii) Periodicity: a (daily) periodic big data application performed on a data flow computer would perform within the time period, while the control flow application that is given the same chip area (equipment size) would not.

(iii) Limited power: a big data application has limited power resources, for instance, on an airplane, on a ship, and at the remote research station. Given the same level of power, the data flow application performs in less time than a control flow application.

(iv) Streaming: large data streams processed on a data flow computer could be processed in real time, while control flow computers would stall. Possible examples are vast amounts of data being generated by sensor networks, devices included in the IoT, and cyber-physical systems.

(v) Unbounded time: when execution time is not a prime concern, data flow computers can save both space and energy.

With the above presented strengths, data flow computers demonstrate a great potential to become the top ranking computers for processing of big data. Why do not we find them on the popular lists of the fastest supercomputers? The answer lies in the traditional benchmarking methodologies that favor control flow computers.

5. New Benchmarking Methodology

In a control flow computer world we are somehow used to the paradigm that more speed (flops) would make a computer faster. With the increasing number of big data applications this perception of computer speed should shift from the number of operations executed in the specified amount of time to the amount of data processed in the same time.

It is argued in [8] that speed is not the only and the most important issue for computer ranking; equally important are the issues of area and power. Therefore, the computer ranking methodologies should focus on more than one of the above issues of importance. Ideally, they should consider all three issues—speed, area, and power—together and at the same time.

Measurement data from real big data applications demonstrate that data flow computers rate better than control flow computers of the same size and power consumption [8]. Concrete figures show that a relatively data-intensive application (order of gigabytes) running on a data flow computer has a speedup of 30 times compared with a traditional control flow computer. A highly data-intensive application (order of terabytes) shows a speedup of 70 times and an extremely data-intensive application (order of petabytes) even greater speedup close to 200 times [8].

Considering the above results it is time to redefine the top 500 benchmarking methodology. A new benchmarking methodology for the maximum performance computing (MPC) should be based on the performance measure that integrates all issues of importance: speed, power, and size. We define the MPC performance measure $M(T_{BD}, N_U)$ as the number of unit size computers required to achieve the projected big data application execution time $T_{BD}$. One unit size computer represents the size of one standard rack unit (1U box) or equivalent; it is assumed that the size of the 1U box is always the same and that it always uses the same amount of power. The performance measure $M$, therefore, implicitly covers the issues of size and power.

The performance measure $M$ is conformant with the ATP (area, time, power) concept of the optimal computer design.
presented in [16]. The \( AT \) bound represents the tradeoffs between area (A) and time (T); if the computer design has more area available, it should be able to perform the computation in less time, which is defined by the expression \( AT = k_1 \). The designs with the area-time product greater than the constant are considered nonoptimal. Similar tradeoffs are possible for time and power (P). The expression \( PT^3 = k_2 \) defines the bounds for power-time tradeoffs. Both bounds, when put together, define a surface of an optimal ATP design. The problem is that it may take excessive design efforts to achieve such an optimal design.

We define the ranking \( R \) on the big data application execution time \( T_{BD} \) multiplied by the number of unit size computers \( N_U \) needed to achieve the time \( T_{BD} \); therefore, \( R = T_{BD} N_U \).

Putting the above into the context of data flow computing, we can note that the defined performance measure \( M \) represents an upper bound for control flow computers and lower bound for data flow computers. Since control flow machines are based on a complex Von Neumann logic, they are difficult to design. On the other hand, data flow computers are based on field programmable gate arrays (FPGA), which are simple to design (high degree of repetitiveness). This is favorable for numerous big data problems where data is repetitively processed by relatively simple operations carried out by the logic on the FPGA.

With the new performance measure \( M \), the petaflops unit should be replaced by petadata unit. Data flow computers would outperform control flow computers if the ranking of computer systems was based on measure \( M \) and ranking \( R \). We support this statement with results stated in [8] and further with the results presented in Sections 3 and 7.

6. Programming Model

A simple hardware design of data flow computers comes at a price: data flow computers are generally more difficult to program. Instead of writing a program for a control flow computer that dynamically controls the flow of data through the computer and operations performed on them, one must write a program that statically configures the data flow computer hardware. The data is then sent through the configured hardware, which performs the operations on flowing data to produce the desired results.

Not all applications are suitable for data flow computers. They perform best with large streaming data sets and algorithms with high degree of data reusability. Algorithms and operation that can be parallelized are also a good match for data flow computers [17]. Generally, the best results are achieved when we run a serial part of the application with dynamic events and controls on the control flow computer and large scale streaming operations and parallel part of the application on the data flow computer [18].

We present one possible approach to programming data flow computers: Maxeler’s data flow programming model [19]. Maxeler data flow computing systems, presented in Figure 1, consist of two parts: a data flow part and a control flow part. The data flow part runs on a data flow engine (DFE)
DFEType type = dfeFloat(8,24);
DFEVar x = io.input("x", type);
DFEVar x prev = stream.offset(x, -1);
DFEVar x next = stream.offset(x, +1);
DFEVar cnt = control.count.simpleCounter(32, N);
DFEVar valid = (cnt > 0) & (cnt < (N-1));
DFEVar y = valid ? (x prev + x + x next) / 3.0: 0.0;
io.output("y", y, type);

Algorithm 1: The moving average kernel code. DFEVar is the keyword used for defining the variables that internally flow through the configured hardware.

Manager m = new Manager(params);
Kernel k = new MovingAverageKernel(
    m.makeKernelParameters());
m.setKernel(k);
m.setIO(link("x", CPU),
     link("y", CPU));
m.createSLiCInterface();
m.build();

Algorithm 2: The moving average manager code. Listing shows only the core code that is managing the data flow between the CPU and DFE for the moving average algorithm.

and the control flow part runs on a CPU. For the data flow application running on a Maxeler data flow system, one must write \( n+2 \) programs, where \( n \) is the number of the application code chunks, named kernels, intended to run on the data flow part of the system. The other two programs are the control flow application code running on a control flow part of the system, named CPU code, and a \textit{manager} that orchestrates the data movement between the control flow and data flow parts of the system.

The development of the data flow part of the application within the described programming model is best explained through a simple example: the calculation of a moving average [19]. The Maxeler MPC system with the programming environment is shown in Figure 1.

There are two parts of an MPC application, the control flow part running on the CPU and the data flow part running on the DFE. The developer first creates the data flow programs: the kernel and the manager. They are written in MaxJ: a Maxeler extension of Java adding operator overloading. The MaxJ code of the moving average kernel is shown in Algorithm 1 and the code of the moving average manager is shown in Algorithm 2. By compiling them with MaxCompiler, the developer creates a “max” file containing the DFE configuration, SLiC (Simple Live CPU) functions, and meta-data. SLiC is the Maxeler’s API for the integration of CPU and DFE.

The control flow part of the application (CPU application) can be written in one of the supported languages including C/C++, R, Python, and MATLAB. The CPU application sits on the top of SLiC and Maxeler OS. It can be seen from the C code shown in Algorithm 3 how easily the data flow code running on the DFE can be integrated into a CPU application.

The CPU application can now be compiled and linked to the “max” file, Maxeler OS, and SLiC to create an application executable, which includes all the codes required for the utilization of the data flow engine. All necessary tools are included in the Maxide development environment, based on Eclipse open source platform.

Another advantage of the presented programming model is that in Maxeler’s MPC systems the Maxeler OS allows the CPU and DFE to run in parallel. This means that while the DFE is processing the data, the CPU performs the non-time-critical parts of the application [19].

7. Application Example

Section 3 listed some examples of algorithm performance on data flow computers. All show considerable performance gain in one or more performance measures: time, area, and power. The usability of the listed algorithms and their performance gains can be further proven when those algorithms are implemented in wide-spread commercial applications.

Pioneers of data flow computing are oil and gas companies with the application for seismic data interpretation in efforts to discover new oil and gas reserves, which require considerable computational effort [20]. Another example comes from the financial analytics field where financial institutions experience a massive increase in the need to perform large, complex computations extremely quickly [21].

While specialized algorithms have found their place in specialized data flow applications, more general algorithms used in different types of wide-spread applications still have to prove their usefulness. One candidate is a group of network sorting algorithms. Their execution on a data flow computer considerably reduces sorting times. Results from [22] indicate that speedups up to 400 times can be achieved.

Figure 2 shows one example of gained speedups for sorting 16-bit number arrays of different sizes. The speedup rises with the number of consecutive sorting episodes (one to a million). The detailed explanation of results is found in [22]. Sorting is inherent to many applications and tasks such as databases, searching, and indexing, to mention just a few. Hence, using data flow processing for number sorting could save much time inside more generally used applications.
#include "Maxfiles.h"
#include <MaxSLiCInterface.h>

const int N = 80;
float x[N], y[N];
for (int i = 0; i < N; ++i)
x[i] = 10.0 * rand() / RAND_MAX;

// This SLiC function is generated automatically
// by MaxCompiler.
MovingAverage(N, x, y);

Algorithm 3: The C program code running on the CPU. Listing shows only the excerpt from the C program code that is connected to the data flow engine.

Data flow computers show their superiority with applications that have high degree of operation repetitiveness and some degree of processed data reusability. Data reusability is particularly important and efficient in big data problems where huge amounts of data are repetitively processed using simple operations.

Because recent supercomputer performance measures and benchmarks are tailored to control flow computers, we present a new benchmarking methodology based on a measure $M$ defined as the number of unit size computers required to achieve the projected execution. Using the new methodology data flow computers would outrank the control flow computers on a top 500 list for a number of big data applications.

The presented results of algorithm and application performance tests show that data flow computers can save time, space, and power, which all cost money. Data centers, running big data applications, should pay attention to these facts, if not earlier, then at least at the next equipment refreshment.

One question still remains: how many big data applications can be divided into (sub)task and (sub)operations that are suitable for implementation on data flow computers? We argue that most of them.

Conflict of Interests
The authors declare that there is no conflict of interests regarding the publication of this paper.

References
[1] C.-Y. Lin, S. Zeadally, T.-S. Chen, and C.-Y. Chang, “Enabling cyber physical systems with wireless sensor networking technologies,” International Journal of Distributed Sensor Networks, vol. 2012, Article ID 489794, 21 pages, 2012.
[2] H. Hu, Y. Wen, T.-S. Chua, and X. Li, “Toward scalable systems for big data analytics: a technology tutorial,” IEEE Access, vol. 2, pp. 652–687, 2014.
[3] N. Mishra, C.-C. Lin, and H.-T. Chang, “A cognitive adopted framework for IoT big-data management and knowledge discovery prospective,” International Journal of Distributed Sensor Networks. In press.
[4] J. Gantz and D. Reinsel, “The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far East,” in Proceedings of the IDC iView, IDC: Analyze the Future, 2012.
[5] J. B. Dennis and D. P. Misunas, “A preliminary architecture for a basic data-flow processor,” in Proceedings of the 2nd Annual Symposium on Computer Architecture (ISCA ’75), pp. 126–132, ACM, New York, NY, USA, 1975.

[6] J. B. Dennis, “Data Flow Supercomputers,” Computer, vol. 13, no. 11, pp. 48–56, 1980.

[7] T. Agerwala and Arvind, “Data flow systems: guest editors’ introduction,” Computer, vol. 15, no. 2, pp. 10–13, 1982.

[8] M. J. Flynn, O. Mencer, V. Milutinovic et al., “Moving from petaflops to petadata,” Communications of the ACM, vol. 56, no. 5, pp. 39–42, 2013.

[9] Convey Computer, 2014, http://www.conveycomputer.com/technology/.

[10] J. Salom and H. Fujii, “Overview of acceleration results of maxeler FPGA machines,” IPSI Transactions on Internet Research, vol. 5, no. 1, pp. 1–4, 2013.

[11] C. Guo, H. Fu, and W. Luk, “A fully-pipelined expectation-maximization engine for Gaussian mixture models,” in Proceedings of the International Conference on Field-Programmable Technology (FPT ’12), pp. 182–189, Seoul, South Korea, December 2012.

[12] J. Arram, K. H. Tsoi, W. Luk, and P. Jiang, “Hardware acceleration of genetic sequence alignment,” in Proceedings of the 9th International Conference on Reconfigurable Computing: Architectures, Tools, and Applications (ARC ’13), pp. 13–24, Los Angeles, Calif, USA, March 2013.

[13] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. W. Leong, and D. B. Thomas, “A mixed precision Monte Carlo methodology for reconfigurable accelerator systems,” in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA ’12), pp. 57–66, Monterey, Calif, USA, February 2012.

[14] V. Ranković, A. Kos, and V. Milutinović, “Bitonic merge sort implementation on the Maxeler dataflow supercomputing system,” The IPSI R&D Transactions on Internet Research, vol. 9, no. 2, pp. 5–10, 2013.

[15] D. Oriato, O. Pell, C. Andreoletti, and N. Bienati, “FD Modeling Beyond 70Hz with FPGA Acceleration,” Maxeler Summary, Summary of a talk presented at the SEG HPC Workshop, Denver, Colo, USA, 2010, http://www.maxeler.com/media/documents/MaxelerSummaryFDModelingBeyond70Hz.pdf.

[16] M. J. Flynn, “Area—time—power and design effort: the basic tradeoffs in application specific systems,” in Proceedings of the IEEE 16th International Conference on Application-Specific Systems, Architectures, and Processors (ASAP ’05), pp. 3–6, Samos, Greece, July 2005.

[17] Y. N. Patt, “Future microprocessors: what must we do differently if we are to effectively utilize multi-core and many-core chips?” IPSI Transactions on Internet Research, vol. 5, no. 1, pp. 5–9, 2009.

[18] O. Pell and V. Averbukh, “Maximum performance computing with dataflow engines,” IEEE Computing in Science & Engineering, vol. 14, no. 4, pp. 98–103, 2012.

[19] Maxeler Technologies, “Programming MPC systems,” White Paper, Maxeler Technologies, London, UK, 2013, http://www.maxeler.com/media/documents/MaxelerWhitePaperProgramming.pdf.

[20] Oil & Gas Solutions Webpage, London, UK, 2014, https://www.maxeler.com/solutions/oil-gas/.

[21] Financial Analytics Solutions, Maxeler Technologies, London, UK, 2014, https://www.maxeler.com/solutions/financial-analytics/.

[22] A. Kos, V. Ranković, and S. Tomažič, “Sorting networks on the maxeler data flow super computing systems,” in Advances in Computers, vol. 96, Elsevier, 2015.