Review of Approaches in High Performance Computing

Nenad Anchev, Blagoj Atanasovski, Sasko Ristov and Marjan Gusev
University Sts Cyril and Methodius
Faculty of Computer Science and Engineering
RugjerBoshkovikj 16
Skopje, Macedonia
anchev.nenad@students.finki.ukim.mk} {atanasovski.blagoj@students.finki.ukim.mk}
{sashko.ristov@finki.ukim.mk} {marjan.gushev@finki.ukim.mk}

ABSTRACT: *High performance computing is a preferred area of study in computing which has potential impact in the future computing and hence using many approaches, the studies have been carried out. In the first approach a cluster of systems linked in LAN is used to bring a single system. In the second one in the grid system, a union of many computer resources are linked in multiple locations. When using the data flow approach, the issues are solved, while the von Nuemann principles is found to be less effective. Easily accessible dataflow engines have given ideas to advocate the developments. We in this work analysed the existing many approaches on high performance computing with the special observation about the dataflow use in the HPC.*

Keywords: HPC, Data flow, FPGA, Grid

Received: 4 March 2022, Revised 30 May 2022, Accepted 14 June 2022

DOI: 10.6025/pca/2022/11/2/44-49

1. Introduction

There is an age old philosophical question, does technology drive the development of society or is it the other way around. In a similar way in the computing world we can pose the same question, does the increase in computing power and innovation of how computing is done leads to processing and storing more data and work with more information or does the need to process and derive more information from the available data push the limits of innovation and results in more and more powerful computers. Either way all can agree that the amount of data processed every day is on a steep rise. The limits on the data setssize that are feasible to process in a reasonable amount of time were on the order of Exabyte [1].

Common knowledge taught as early as undergraduate students in their first courses of Computer Architecture is the rate at which computing speed and memory speed have increased through time is very unbalanced. This has resulted in creating solutions ranging from changing the inside CPUarchitecture to use prefetching and caching, through optimizing compilers for optimal reordering up to developing entire paradigms for programming. To handle the amounts of big data the possibilities of dataflow computing, which has the data at its focus, should be taken in consideration.

In this paper we give an overview of dataflow computing, compare it to other established HPC approaches and a way to make it feasible.

2. Comparison of HPC Platforms

The mentioned platforms in this section are not different in their computational architecture (except the GPUs) but more or less in their organization and connection, on how the data is transferred to the computing elements and the results gathered after the computing is done. We cover the characteristics of 4 established HPC approaches: cluster, grid, cloud and general purpose GPU computing.

2.1. Computer Clusters

A form of distributed system consisting of a set of interconnected working and available computing nodes (computers) connected with a local network. The activities of the nodes are controlled by a special software middleware layer that is present on all the nodes allowing the system to be perceived as one cohesive computing unit. Computer clusters rely on a centralized management approach in contrast to grid computing. Typically clusters use the same or similar types of machines, they are tightly coupled and use dedicated network connections, share resources as a common home directory and use an MPI implementation for passing messages between nodes. [2]

The benefits are the low cost, complexity for configuring and operating them because of the off-the-shelf components that can be added as needed, which helps with the elasticity required to add or remove resources proportional to the workload. Tightly coupled clustersconnected with high speed networks are optimized to create supercomputers. The benefits of low cost and elasticity are those compared to buying monolith supercomputers where many processors are connected on an ultra-speed bus. Figure 1 depicts a Beowulf cluster, a model created by identical nodes from commoditygrade hardware networked in a LAN.

The programming model relies on the use of MPI for using parallelism, sending and gathering messages. Programs created for a single processor must be rewritten to include the MPI directives, but are simpler than creating programs for a custom supercomputer operating system.

2.2. Grid Computing

The federation of computer resources from multiple locations to reach a common goal. The name is an analogy of the electrical power grid. It can be thought of as a distributed system similar to cluster computing whose nodes are more loosely coupled, heterogeneous and dispersed on distant locations. The idea is to create parallel computing based on complete computers connected to a private or public network via standard interfaces instead of the approach of traditional supercomputers.

A characteristic of grid computers is that they can be formed from computing resources belonging to multiple administrative domains, allowing them to share the costs and computing power. A disadvantage is the lack of central control over the hardware, so there is no uptime guarantee and problem with trustworthiness.

The programming model is the sameas the model for cluster computing with more thought given to decreasing inter-node communication.

2.3. Cloud computing

The disadvantages of having multiple administrative domains and unguaranteed uptime can be avoided if cloud computing is used. It represents the use of computing resources delivered as a service over a network, usually the Internet. Characteristics of cloud computing are the elasticity and scalability of resources via dynamic provisioning, multi tenancy for sharing resources and costs, and in public clouds the management of the platform is taken care of. The Cloud is rarely used for HPC reasons. It can be used when a lot of computing power is needed for short periods, otherwise the constant price of sending the huge data needed by HPC applications keeping it in the cloud and shifting the results back would accumulate over time to match the price required to buy and implement cluster.

2.4. General Purpose computing with GPUs (GPGPU)

GPGPU has been defined as the use of Graphical Processing Units to handle computing traditionally handled by the CPU. This kind of computing can be done only on newer generation GPUs that offer a complete set of instructions for doing operations on

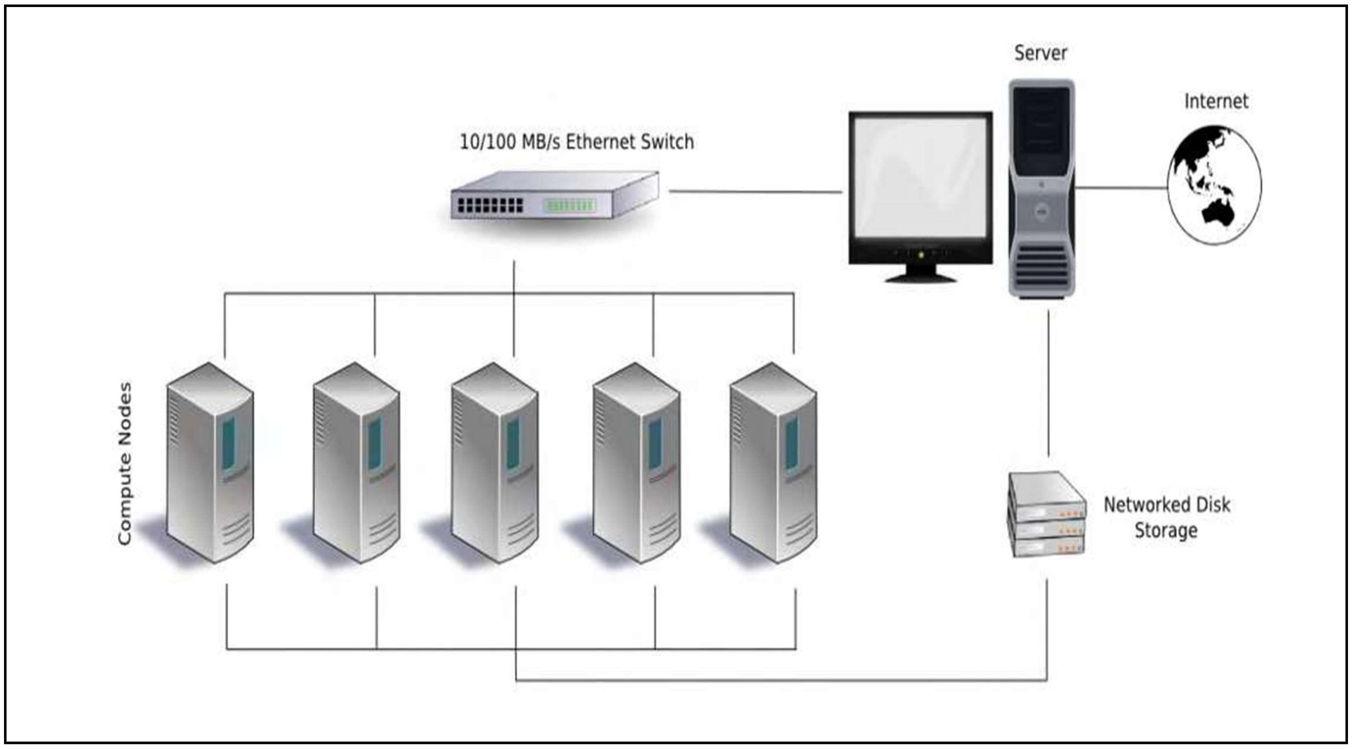

Figure 1. Beowulf cluster

arbitrary bits. GPUs are designed to work with streams of records that require similar computation. GP Us process data independently so there is no shared or static data. Multiple inputs and outputs can be defined, but a piece of memory cannot be both readable and writeable. GP GPU applications require large data sets, high parallelism and minimal dependency between elements to avoid memory access latency and achieve speedup. A popular parallel programming platform and programming model created by NVIDIA that is implemented in their GPU products is CUDA [5]. Figure 2 shows the processing flow in a typical CUDA application where the numerous cores of the GPU would be used to run an application in parallel. The flow begins by sending the data from the main memory to the graphics card memory and instructing the GPU cores what kind of processing to do. Each of the cores runs in parallel and in the end the result is copied back to the main memory.

3. What is Dataflow Computing

Dataflow computing can be seen both as a different architecture and as a completely different programming model needed for that architecture. It is a direct contrast to the traditional control flow architecture. The execution of instructions is determined based on the availability of input arguments of the instructions. Dataflow architectures were a major research topic in 1970s. The two types of dataflow machines that have been researched were static and dynamic ones. Static designs use conventional memory addresses to tag the dependencies. Dynamic designs use content-addressable memory, where they use tags to facilitate parallelism. These designs were supposed to execute programs by first loading them into CAM, when all of the operands tagged for an instruction are available the instruction is marked as ready. There were several problems with these architectures, such as the inability to build a large enough CAM to contain all the dependencies of an executing program.

Dataflow can be also viewed from a programming model perspective, a type of software architecture. The increasing demand for processing of larger data quantities requires a model that has been designed to handle enormous flows of data through high-speed computations.

The idea of dataflow computing has been hindered for decades by the success of the supercomputers. The drawbacks of dataflow computing, namely the specialized hardware needed for every different program, instead of the programmable nature of the control-flow computers, meant that dataflow computing was unfeasible compared to control flow computing.

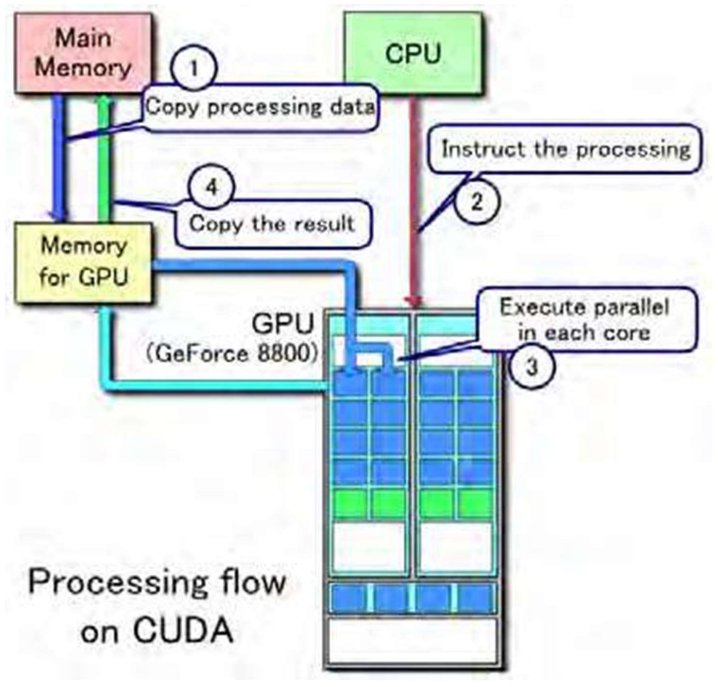

Figure 2. Processing flow on CUDA[3]

With the appearance of field programmable gate arrays, the idea for dataflow computing was given a rebirth. The FPGAs property of "reprogrammable hardware" was crucial for the rebirth of this idea.

So, let's compare the two paradigms. On one side we have the well-known control flow architecture, CPU and main memory, connected with a bus. Memory is filled with instructions from a compiled program written in some programming language (control logic), and application input data. During the execution of a program, instructions and data are being transferred to the processor and being executed. The output data is then returned back to the memory. The dataflow architecture with FPGAs (Maxeler) works as follows: First, a data-flow code is being written. The code is compiled to a configuration file, which describes the way in which FPGAs are configured. Then the FPGAs are configured, and they are ready to do the computing, as soon as data arrives in them. In the execution part, input data is streamed into the dataflow engine, the engine does all the computation according to the configuration, and the output is streamed back to the memory.

As we can notice, in dataflow computing there is no instruction stream (program code) in the stage of execution. Instead, instructions are "written" on the FPGA at the compilation stage. This is the main advantage of the data-flow computing, as it gets rid of all the problems associated with the "unpredictable" instruction stream, so all the techniques for resolving these problems in modern processors become obsolete. This is one of the main contributors for the achieved speedup, compared to control flow architectures.

The other obvious advantage of this technology is the high level of optimization and fine tuning that an FPGA allows. Here we are not limited by the bottlenecks of modern computers, and we have a relatively greater degree of freedom in "programming" our own hardware, allowing us to boost performance and speedup algorithms many times. However, this can be also viewed as a drawback. Dataflow programming and FPGA configuration is a relatively new paradigm, which requires a different way of thinking and coding, and very few people are able to successfully program dataflow logic.

The only major bottleneck in dataflow computing is the transfer of data streams onto, and from the dataflow engine. As Maxeler dataflow engines have to be attached to a regular computer via PCI Express bus, data transfer rates are limited. This bottleneck reduces the usability of dataflow engines only for compute intensive algorithms, with small I/O.

Another bottleneck of the dataflow computing, which comes from the immaturity of the FPGA technology, is the low working frequency of the FPGAs, which currently is in the range of 200MHz. This is 10 times lower compared to modern processors, which slightly lowers the potential of speedup at dataflow engines. However, this may be also seen as an advantage, as power consumption is much lower at these frequencies than at the GHz order at the modern processors. Maxeler states that power consumption per computation is 30 times lower at their technologies compared to standard control flow multiprocessor. With the advance of the FPGA technology, working frequencies may be increased, but the power consumption is still predicted to be lower than conventional computers.

4. Dataflow Programming

The dataflow computing platform, as a computing platform, was already presented in the previous section. Here we are going to present the dataflow platform as a programming paradigm. As we mentioned in the previous section, dataflow computing lacks the existence of instructions as defined in the well-known computer architectures. Instead, we are configuring FPGAs to manipulate the input data streams, and produce the output stream. The code is compiled similarly, but the lowest level of code here describes the configuration of the FPGAs, which are then being “programed” to solve the particular problem. After the configuration is finished, the dataflow engine may be running, when the input data streams are provided. Here, we are going to describe the programming paradigm of a particular dataflow implementation, Maxeler’s dataflow engines.

Maxeler offers a specialized Java-like programming language for dataflow engine programming. It offers a modified Eclipse IDE, MaxCompiler which compiles the high-level programming code down to FPGA configuration files, and MaxelerOS, which has the task to deal with the FPGA configuration, and communication of the dataflow engine with the host computer. Except for the Java-like dataflow code, a C code is required for the host part. The C code has the task to transfer data to, and from the dataflow engine, and possibly do some minimal computation to avoid being idle while the dataflow engine does the bulk of the computation.

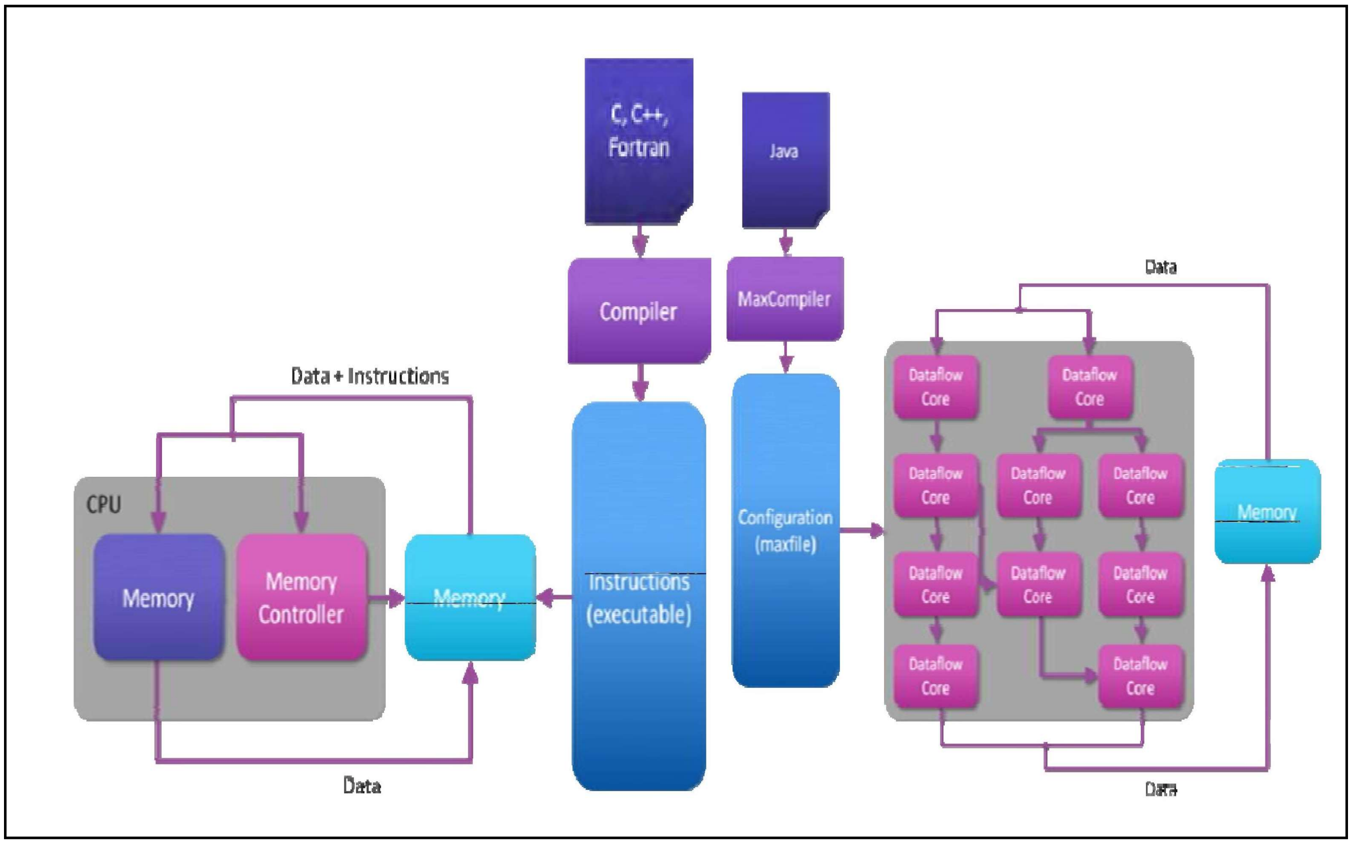

Figure 3. Comparison of control flow and data flow architectures [4]

The Java-like dataflow programming code consists of two main objects: kernels, and managers. Kernels are objects that describe what kind of hardware building blocks are going to be put in the dataflow engine, how and which computations are going to be done on the data streams, and when and on which part of the data streams will these blocks operate.

When a kernel programmed, the most difficult part is the translation of nested loops and conditional statements to dataflow logic. Nested loops are unrolled, whenever it is possible, or programmed with counters and stream offsets, which temporary memorize part of the streams in the local memory, and then stream then again them to the kernel. Conditional statements pose less challenge to be translated, as kernels allow elements that conditionally select or process certain elements of the stream, while ignoring others.

Coding a kernel in dataflow programming is somewhat similar to coding a function or routine in control flow programming. W hile on the other hand, the manager is similar to the main function in a code that usually dispatches data to particular functions, and then collects the results. More specifically, the manager here deals only with the task of connecting and synchronizing input and output streams from/to particular kernels, host machine, local dataflow engine memory, and other additional dataflow engines if such.

When considering the data streams, it is important to note that the working frequencies of multiple kernels, dataflow memory, and inter-dataflow communication buses are similar, so there are no wasted cycles or synchronization problems. However, communication with the host machine is usually slower, which the dataflow engine resolves by adding empty computation cycles, similar to filling the processor-memory gap in the conventional computers.

Except for reducing the communication with the host, other points that should be considered while programming and optimizing a Maxeler dataflow engine are the follows:

- Finding a convenient way of transforming nested cycles. This usually implies that the whole logic of a nested cycle should be rethinked again in order to comply with the dataflow paradigm. This is the hardest part of the code translation in dataflow logic. Unsuitable or non-optimal conversion may even produce performances that are worse than conventional computers.

- Using as much as parallelism and pipelining as possible. The number of used adders, multipliers, and comparators on an FPGA is limited, and finding an implementation that uses most of the available ones implies much greater efficiency.

The way of rethinking and reprogramming the application in dataflow logic is the most important (or probably, the only) factor that defines the achieved performance and speedup. The experience and the higher level of understanding of dataflow logic allow a programmer to write an optimal dataflow code.

5. Conclusion

Different approaches for today's hunger of computation exist. All of them offer huge computing power. However, the customers should select the computing platforms according to their price, availability, performance, adaptability, interoperability, portability, cost etc. In this paper we present the most common platforms used today for high performance computing, their architecture, use, advantages and disadvantages. We also present the dataflow architecture and programming model as a new platform that can be used for the same goal. Dataflow computing emerges as one of the ideal platform. It is cost and performance effective.

References

[1] Francis, Matthew (2012-04-02). "Future telescope array drives development of exabyte processing". Retrieved 2012-10-24.

[2] Baker, Mark, et al. "Cluster computing and applications." Encyclopedia of Computer Science and Technology 45.Supplement 30 (2002): 87-125.

[3] The image is published under the Creative Commons licence, taken from en.wikipedia.org/wiki/File:CUDA_processing_flow_(En).PNG as seen on 6.4.2013.

[4] The image is taken from www.maxeler.com/technology/dataflow-computing/ as seen on 10.4.2013

[5] NVIDIA CUDA Home Pagewww.nvidia.com/object/cuda_home_new.html