RUNNING APLLICATIONS ON A HYBRID CLUSTER[1]

A.V. Bogdanov[1], I.G. Gankevich[1], V.Yu. Gayduchok[2], N.V. Yuzhanin[1]

[1]Saint Petersburg State University, Russia,
[2]Saint Petersburg Electrotechnical University "LETI", Russia,
bogdanov@csa.ru

A hybrid cluster implies the use of computational devices with radically different architectures. Usually, these are conventional CPU architecture (e.g. x86_64) and GPU architecture (e.g. NVIDIA CUDA). Creating and exploiting such a cluster requires some experience: in order to harness all computational power of the described system and get substantial speedup for computational tasks many factors should be taken into account. These factors consist of hardware characteristics (e.g. network infrastructure, a type of data storage, GPU architecture) as well as software stack (e.g. MPI implementation, GPGPU libraries). So, in order to run scientific applications GPU capabilities, software features, task size and other factors should be considered.

This report discusses opportunities and problems of hybrid computations. Some statistics from tests programs and applications runs will be demonstrated. The main focus of interest is open source applications (e.g. OpenFOAM) that support GPGPU (with some parts rewritten to use GPGPU directly or by replacing libraries).

There are several approaches to organize heterogeneous computations for different GPU architectures out of which CUDA library and OpenCL framework are compared. CUDA library is becoming quite typical for hybrid systems with NVIDIA cards, but OpenCL offers portability opportunities which can be a determinant factor when choosing framework for development. We also put emphasis on multi-GPU systems that are often used to build hybrid clusters. Calculations were performed on a hybrid cluster of SPbU computing center.

Introduction

Recent years have shown growing interest to hybrid computations. It became clear that conventional architectures have limited performance, which is in many cases inferior to performance of hybrid ones, let alone energy consumption and heat generation. Today one can face many different hybrid architectures, the vast majority of them are usually employ SIMD-accelerator (GPU, Cell, MIC, etc.) and a conventional CPU. This report concerns GPGPU as one of the earliest accelerator implementation.

One should clearly understand that such systems are not a panacea: while there are many tasks that can be smoothly mapped on GPGPU there are some classes of algorithms that can not benefit from implementing them for GPGPU. This report concerns aspects of running applications on a hybrid cluster that contains several GPGPUs on each node.

One can look at TOP-500 list and find out that the most powerful supercomputers are hybrid clusters: today (June 2014 TOP-500 list) about 35% of overall performance of TOP-500 list systems is given by some accelerator extension cards which is almost four times more than in 2010 (hybrid systems provided only 9% of total performance at that time).

It can be explained with a simple fact: hybrid systems are constantly evolving that leads to performance growth while preserving and improving GFLOPS/Watt ratio. Software for hybrid systems is improving too. Software companies develop new libraries for GPGPU and applications that use such libraries, there are already several standards for GPGPU (e.g. OpenCL, OpenACC).

Such evolution can be seen on the example of GPGPU and other accelerators. Manufacturers try to ease programming of such systems. Some compilers can automatically split tasks between CPU

[1] The research was carried out using computational resources of Resource Center Computational Center of Saint Petersburg State University (T-EDGE96 HPC-0011828-001) and partially supported by Russian Foundation for Basic Research (project No. 13-07-00747) and Saint Petersburg State University (project No. 9.38.674.2013).

and GPU. In case of MIC there are two basic approaches: native compilation and offload when MKL automatically offloads parts of a program to accelerator.

But one should clearly understand that not all tasks benefit from GPGPU usage (some tasks can show speedup while others show even slowdown when running on GPGPU) and remember that this area is still developing, so it will probably take time for such systems to become quite common and quite simple for programming.

1 GPU use cases

There are several ways one can harness GPU.

*** Conventional usage.** The obvious case. GPU is used for graphics computations, relatively simple. One or multiple GPUs per one node. Probably is be no need for building a cluster in that case, since compute-intensive calculations are a part of the third case.

*** GPUs for virtual machines.** The next approach is to use some GPUs within one or several virtual machines. Virtualization technologies are wide-spread due to advantages they offer to organizations and end users. Virtual CPU and network devices become ubiquitous while virtualization of powerful GPUs is an actively developed area which becomes a point of interest for major manufacturers. Such techniques are implemented, for example, in VMWare products and XenServer. There are two basic approaches in this area.

- Dedicated GPU (GPU pass-through). This variant is similar to conventional GPU usage: hypervisor just gives a virtual machine unrestricted access to the whole GPU. So, such virtual machine can use the whole GPU, while other VMs have no access to this GPU.
- Virtual GPU. Fully virtualized GPU is used in this case. Such approach can be exemplified by NVIDIA GRID K2 GPU managing by XenServer. Such GPU can be divided into several virtual GPUs with different characteristics. Each virtual GPU can be assigned to a separate virtual machine.

A scheme in Figure 1 depicts this two approaches.

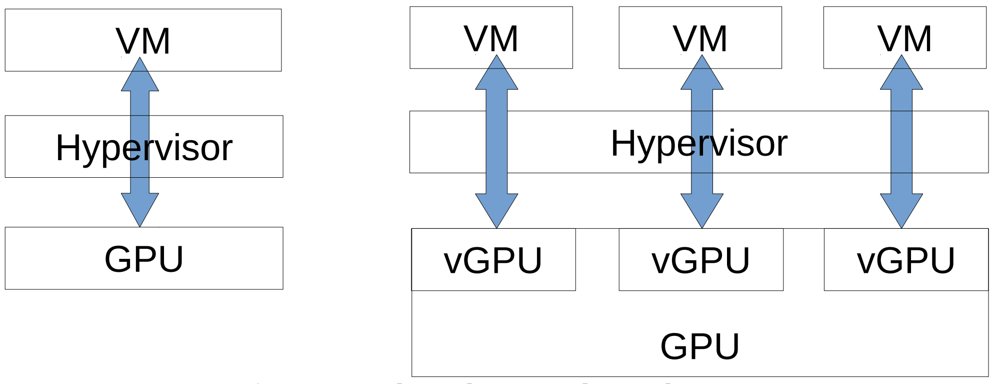

Figure 1. Dedicated GPU and virtual GPUs.

*** GPUs within computational clusters.** Finally, one can create cluster which nodes will contain one or more GPGPUs. Such clusters are usually referred as hybrid clusters. While the second case (especially virtual GPUs) is still not common this case is the most frequently used approach for scientific computations. The main issues that arise in this case are listed below.

- How many GPUs should be installed into one node? The answer to this question is determined by several factors: node interconnect (that can be a “bottleneck” when the number GPUs is growing), the number of CPU cores (CPU will assign tasks to GPUs), etc.
- How to share cluster resources between users? One can solve this problem (it can be a real challenge in situation with limited resources that should be accessible for many users with radically different tasks) via Portable Batch System (PBS). There are several implementations of PBS that varies in some parameters, but the main idea is still the same: administrator

creates several queues with different limitations. So, he can decide which users should have access to a particular queue.

- How to restrict user from using GPUs (or some other cluster resources) that assigned to another one? This question is solved by choosing appropriate managing system (for example, some reliable PBS implementations that can manage GPU access and so forth) and proper configuration as well.

2 Platform specifications

All tests for this report were performed using hardware of resource center of Saint Petersburg State University. This center has hybrid cluster with the following characteristics:

- 24 nodes;
- 2 CPU Intel Xeon X5650 (6 cores per CPU, total 12 cores);
- 96 RAM;
- 3 (16 nodes) or 8 (8 nodes) GPUs NVIDIA M2050 per node;
- Ethernet 10G network;
- Infiniband 4x QDR (40 Gbit/sec) network.

Peak performance of this complex is 59.6 GFLOPS. GPU peak performance is 0.5 TFLOPS (double precision), while CPU performance is 0.075 TFLOPS. Each node is running CentOS 6.4. Cluster management system is PBS.

3 Basic tests

For assessing the GPU performance and scalability of different tasks on GPU it's quite convenient to start from LINPACK. It is de facto standard benchmark for HPC systems (TOP-500 uses the best run of LINPACK test for creating the list). This test solves a dense system of linear equations using LU factorization. Figure 2 depicts the LINPACK test results (performance in GFLOPS for one node with 3 GPUs depending on the matrix size).

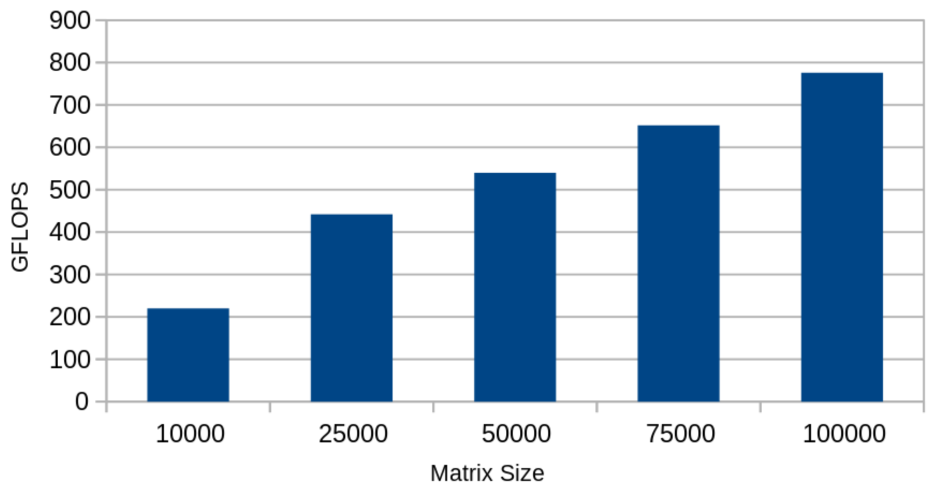

Figure 2. LINPACK test results for nodes with 3 GPUs.

While LINPACK test uses double precision (TOP-500 takes into account only double precision LINPACK) such precision is not necessary for many tasks. Next two figures depict the results of test that includes numerical integration for single and double precision.

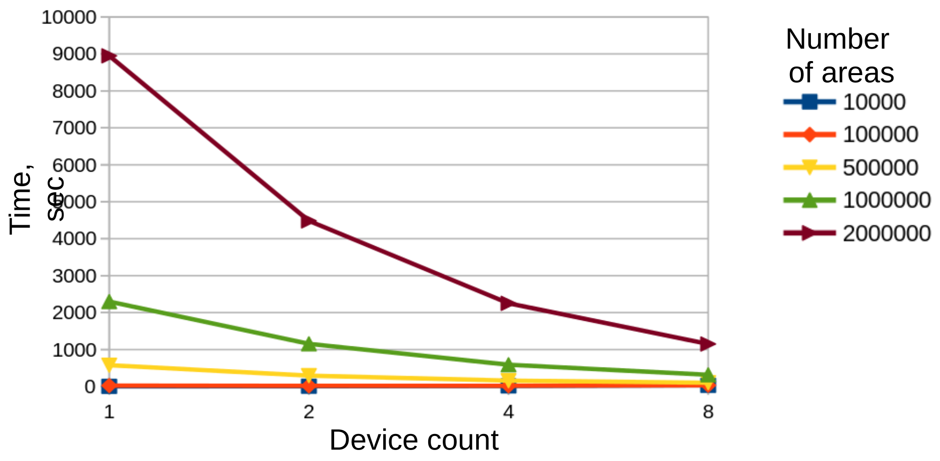

Figure 3. Numerical integration test, single precision.

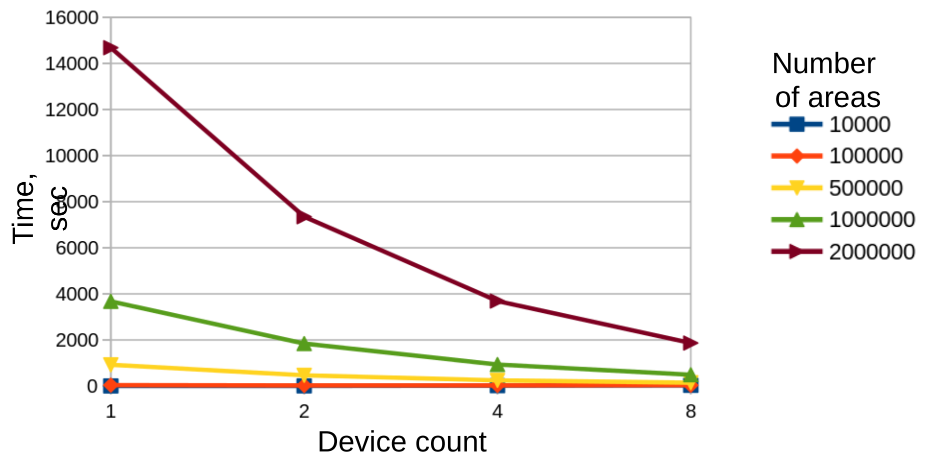

Figure 4. Numerical integration test, double precision.

Both these tests show similar behavior: performance gain increases with task size. The last test (for single and double precision) shows good scalability when runs on multi-GPU systems. GPUs shows about twice as much performance for single precision in comparison to double precision test. It is suits quite good to performance declared by manufacturer (0.5 TFLOPS for double and 1 TFLOPS for single precision). But these tests are synthetic, they both can be efficiently mapped to GPGPU. Users don't need such tests (except for initial hardware testing) because they want to run real applications that solves their scientific problems. Some real world applications that uses GPGPU are discussed in the next section.

4 Applications

There are many applications that support GPGPU, open source and free, as well as commercial: Abinit, ANSYS, GROMACS, Matlab, OpenFOAM (with Ofgpu library or some other library for GPGPU), QuantumEspresso, etc. The list of such applications is constantly growing. We are interested in open source applications, but we will start from two well-known commercial applications because different scientists work with different applications (tools, libraries) and get used to different instruments, commercial and non-commercial, while our aim is to provide users with information about general rules for running GPGPU applications, point to a possible “bottlenecks”, assess performance which can be gained on hybrid systems, we look at clusters from administrator perspective.

Figure 5 depicts the MATLAB 2011b BLAS level 3 test for single and double precision that was run on CPU and GPU.

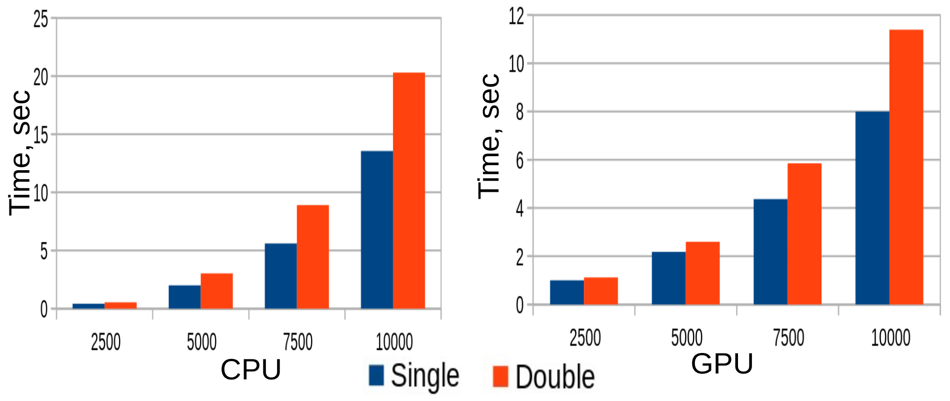

Figure 5. MATLAB R2011b BLAS 3 level test.

This diagrams once again show how task size influences the performance. Tasks that can be vectorized (matrix operations in this example) can be mapped to GPGPU and benefit from GPGPU usage.

The next example is ANSYS Fluent case. We decide to run small case on different number CPU cores and GPUs. ANSYS Fluent version 15 allows user to use both CPU and GPU within a run. The only rule is CPU cores number should be divisible by GPUs count (that's why for some configurations there is no time value). The results are depicted in Figure 6.

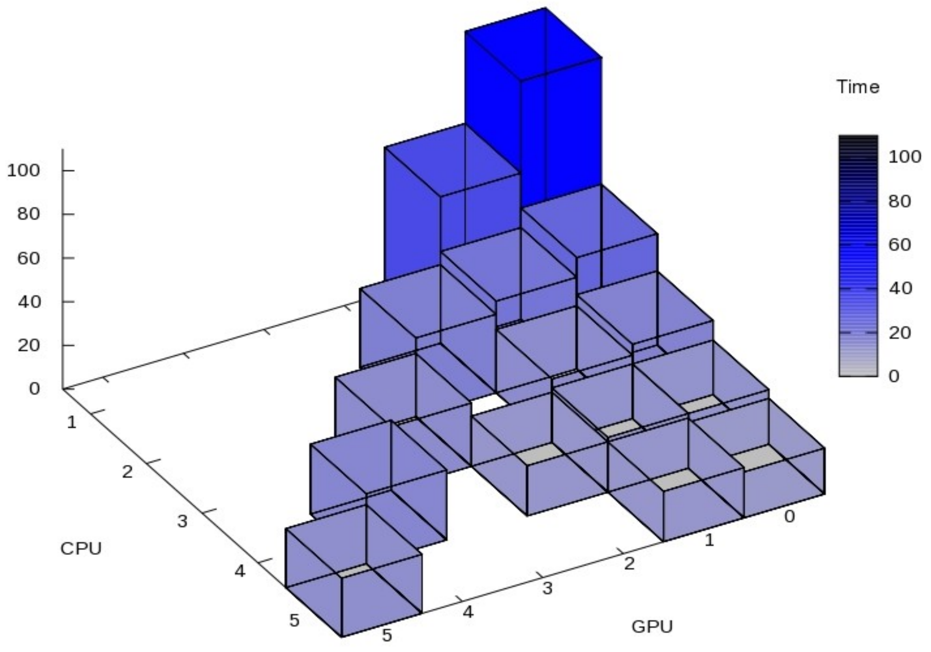

Figure 6. ANSYS Fluent test results (small case).

As one can see small case has not so good scalability even if user runs certified large commercial product.

As an open source alternative to MATLAB we used ViennaCL in our tests. It has CUDA library and OpenCL versions and also version for MIC. The results of LU decomposition test is depicted in Figure 7 (performance means here performance ratio, as compared with run time of the test with 500 matrix size).

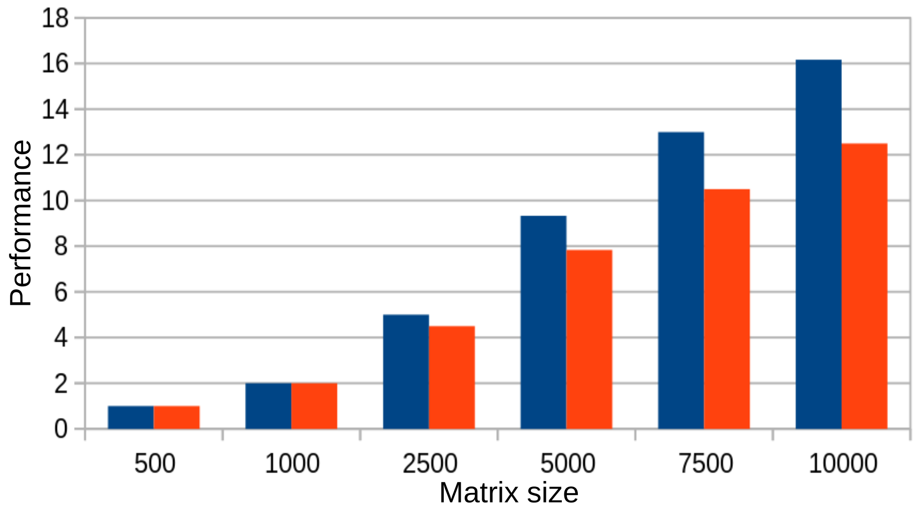

Figure 7. ViennaCL LU decomposition test results.

And finally OpenFOAM. OpenFOAM is an open source platform for solving CFD problems. There are several libraries for running OpenFOAM on GPU. Ofgpu is an open source library for GPU computations. It provides users with GPU linear system solvers and uses CUSP to work with matrices.

There also another libraries for running OpenFOAM on GPGPU, for example, SpeedIT. SpeedIT is a commercial library for GPGPU. It contains accelerated linear system solvers. It has a free version, but that version has restrictions. So, we decided to go for ofgpu library.

OpenFOAM CFD calculations involve solving systems of linear equations. This task is usually offloaded onto GPUs.

Ofgpu provides users with 2 linear system solvers: PCGgpu (preconditioned conjugate gradient solver for symmetric matrices for GPGPU) and (PBiCGgpu - preconditioned biconjugate gradient solver for asymmetric matrices for GPGPU). These solvers should be specified in the file: <case>/sysem/fvSolution.

Ofgpu is built as a separate library and it should be loaded when OF solver use "*gpu" linear system solvers. User can specify GPU device to use in the file: <case>/sysem/controlDict ("cudaDevice" parameter).

Figure 8 and 9 depicts test results for CPU and GPGPU runs for different mesh sizes. Test case corresponds to a steady turbulent flow in a tube (OpenFOAM 2.2.2 single precision with OFGPU 1.1, compiled using Intel compilers (Intel Cluster Studio 2013; ICC 13.0.1), for CPU runs IMPI 4.1.0.024 from Intel Cluster Studio 2013 was used, CUDA Toolkit 5.5 was used for GPGPU version).

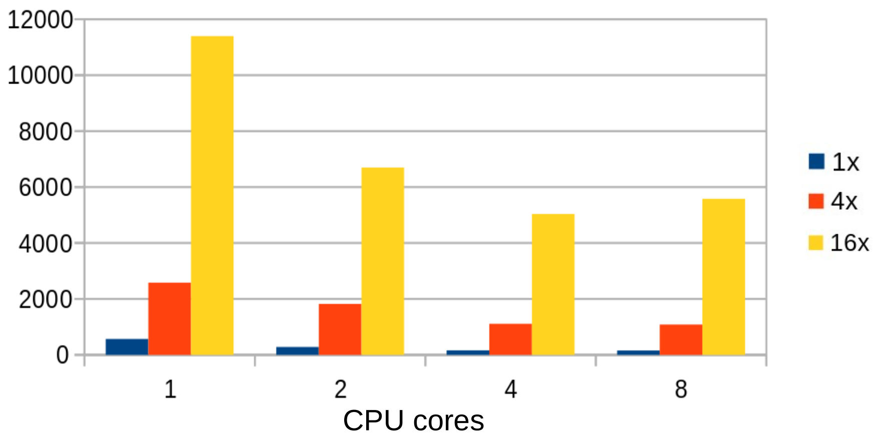

Figure 8. OpenFOAM CPU test results.

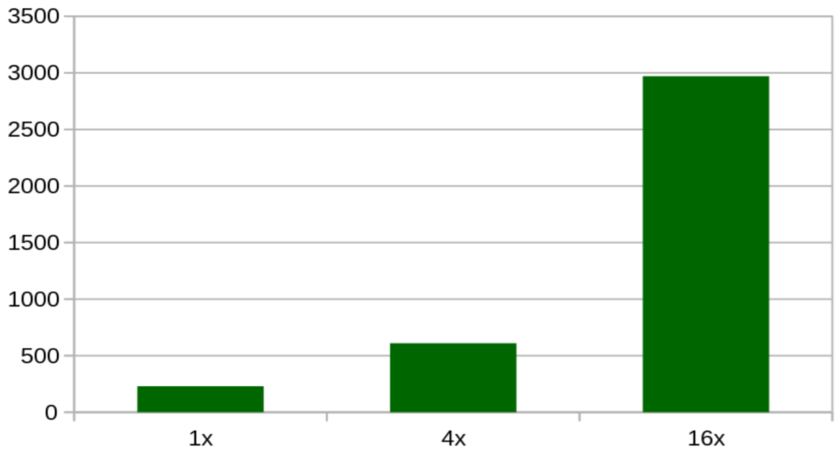

Figure 9. OpenFOAM GPGPU test results.

As one can see the finer mesh is used the more speedup is achieved.

But Ofgpu allows users to run OpenFOAM computations only on one GPU. Multi-GPU systems are not supported. One can run different tasks using different GPUs (by specifying GPU number), but one can not use several GPU within one task. We are currently working on multi-GPU version of ofgpu that will be able to harness multiple GPUs on different nodes on a hybrid cluster. The results will be published in future works.

Conclusions

Hybrid clusters offer great peak performance. But real performance depends on task. It is advisable to run coarse grained algorithms on such systems in order to decrease data transfer because data transfer can be a bottleneck. Data transfer from CPU to GPU (GPU to CPU) as well as data transfer between nodes via network since cluster architecture introduce new possible "bottleneck" - network. Software implementations can increase or decrease performance (for example, one should think about MPI implementation in case of MPI program). Performance gained is task size dependent. For example, one can get substantial speedup on GPU for CFD task when using finer mesh (but anyway there is a question whether or not one needs such fine mesh). Another possible problem is limited memory on GPU. When writing an application memory size should be taken into account. Programmer should choose the way he will use accelerators. He could write code for the accelerator using special API or delegate this work to compiler or special library that will offload the accelerator. The first method is advisable since compilers (or special libraries) are still can not split code for CPU and accelerators in a manner that will lead to great speedup. But programming such systems requires more accuracy from a programmer. Multi-GPU systems introduce new questions. And finally, one can get slightly different results on CPU and GPU working with floating point numbers. It concerns rounding and accordance to a floating point standard (e.g. IEEE 754) and it should be taken into account too.

Cluster management system can ease maintenance of such complexes. Sharing resources between users (especially GPUs) can be a challenge, but good management system can ease this task.

So, hybrid clusters are an actively developing area, promising systems that will probably be actively discussed theme in the future. Despite the fact that such systems introduce several issues that are not specific to traditional architectures they offer advantages that have never been specific to traditional architectures too.

References

[1]