Computer Science • 13 (2) 2012 http://dx.doi.org/10.7494/csci.2012.13.2.63

Michał Marks
Jarosław Jantura
Ewa Niewiadomska-Szynkiewicz
Przemysław Strzelczyk
Krzysztof Góźdź

HETEROGENEOUS GPU&CPU CLUSTER FOR HIGH PERFORMANCE COMPUTING IN CRYPTOGRAPHY

Abstract *This paper addresses issues associated with distributed computing systems and the application of mixed GPU&CPU technology to data encryption and decryption algorithms. We describe a heterogenous cluster HGCC formed by two types of nodes: Intel processor with NVIDIA graphics processing unit and AMD processor with AMD graphics processing unit (formerly ATI), and a novel software framework that hides the heterogeneity of our cluster and provides tools for solving complex scientific and engineering problems. Finally, we present the results of numerical experiments. The considered case study is concerned with parallel implementations of selected cryptanalysis algorithms. The main goal of the paper is to show the wide applicability of the GPU&CPU technology to large scale computation and data processing.*

Keywords parallel computing, HPC, clusters, GPU computing, OpenCL, cryptography, cryptanalysis

1. Introduction

One of the biggest advantages of distributed systems over standalone computers is an ability to share the workload between computers, processors, and cores. Clusters, grids, and cloud computing are one of the most progressive branches in a field of parallel computing and data processing nowadays, and have been identified as important new technologies that may be used to solve complex scientific and engineering problems as well as to tackle many projects in commerce and industry [6, 16, 28]. A broad spectrum of current parallel computing activities and scientific projects are carried out. A new model for parallel computing that relies on usage of CPU and GPU units to solve general purpose scientific and engineering problems revolutionized data computation last years [18, 19, 31]. The tasks that can be divided up into large numbers of independent parts are good candidates. GPU-enabled calculations seem to be very promising in data analysis, optimization, simulation, etc. Using CUDA or OpenCL, and graphical processing units many real-world applications can be easily implemented and run significantly faster than in multi-processor or multi-core systems [9].

We have designed and developed a hybrid cluster system HGCC (*Heterogenous GPU&CPU Cluster*) – a novel computing architecture with multi-core CPUs working together with many-core GPUs. Our cluster integrates two types of CPUs, i.e., Intel and AMD processors equipped adequately with NVIDIA and AMD graphical units. The novelty of our solution is not only the proposed hybrid architecture of the cluster but a new software environment that can support a potential user in his task execution. The goal of this software is to divide data into separate domains, allocate the calculation processes to cluster nodes, manage calculations, and communication. Therefore, from the user's perspective, the cluster system serves as one server – its heterogeneity due to various CPU and GPU architectures is hidden. The HGCC cluster is dedicated to perform complex calculations and the processing of large amounts of data. Data encryption and decryption algorithms are natural candidates. We have designed and developed parallel versions of some cryptography techniques, and implemented them employing the functionalities and facilities of our cluster. Numerical tests were performed to show the efficiency and scalability of our implementations, and possible applications of the HGCC system. It should be pointed out here that in our case study we focused only on the efficient implementation of commonly used encryption and decryption algorithms. We applied the SPMD (Single Program Multiple Data) parallelization technique with domain decomposition. The goal was to speed up calculations.

In this paper we show how to build a heterogenous cluster system that is composed of CPU and GPU units with different architecture. The remainder of this paper is organized as follows. Selected software tools for CPU and GPU clusters are discussed in section 2. In section 3, we provide description of our heterogenous system. In section 4, we describe the HGCC software framework that manages calculations in our cluster. Finally, we present a survey of the implementation of cryptography on

GPU, and briefly summarize the results of tests for selected types of cryptanalysis algorithms.

2. CPU and GPU clusters

For HPC enthusiasts there are two important months in the year: June and November, when the TOP500 list is published. The announcement of the list is not only a chance to observe what are the most powerful supercomputers but also a great opportunity to observe how the the HPC trends are changing. For the first time since the list began to be published in 1993 the Top 10 supercomputers on the latest list remain unchanged. However, we can observe a rising significance of GPU accelerators. In the November 2011 ranking, there are 39 machines that have GPU accelerators, up from 17 only six months ago. What is interesting 35 of them are using NVIDIA Tesla GPU coprocessors or Quadro graphics cards and only two IBM's Cell coprocessors and two use AMD's Radeon cards. There is no solutions utilizing GPU accelerators from different vendors.

The most common operating systems used for building clusters are UNIX and Linux. Clusters should provide the following features: scalability, transparency, reconfigurability, availability, reliability, and high performance. There are many software tools for supporting cluster computing. In the beginning of XXI century the common idea was to provide a view of one supercomputer for a cluster built from a group of independent workstations. SSI (Single System Image) clusters were designed and developed. In this approach all servers' resources such as disks, memory, processors are seen by a user as one unique machine. The whole cluster is identified from outside by one IP address. The popular systems that implement the idea of SSI are Mosix (www.mosix.org) that does not cover all SSI features, and two comprehensive clustering solutions offering full SSI environments: OpenSSI (openssi.org) and Kerrighed (www.kerrighed.org). A brief overview and comparative study of stability, performance and the efficiency of Mosix, OpenSSI, and Kerrighed systems is presented in the literature [23, 28].

Other commonly used systems that can be applied to high performance data processing and calculations in cluster systems are software frameworks that perform job scheduling. A commonly used Portable Batch System PBS (www.pbsworks.com) provides mechanisms for allocating computational tasks to available computing resources. Various versions of the system are available (open source and commercial): OpenPBS, Torque, PBS Professional.

Most of the presented cluster systems are mature solutions. However, they have some limitations. Mosix, OpenSSI and Kerrighed systems focuss on load balancing. The idea is to implement an efficient load balancing algorithm, which is triggered when loads of nodes are not balanced or local resources are limited. In general, processes are moved from higher to less loaded nodes. Unfortunately, the migration of processes involves extra time for load calculation and overhead in communication. Moreover, Mosix, OpenSSI, Kerrighed systems were designed for CPU clusters.

Currently, users are provided with software environments that allow us to perform calculations on a single GPU device. There are only a few software tools for running applications on GPU clusters. Virtual OpenCL VCL (`www.mosix.org/txt_vcl.html`) is a software platform for GPU clusters. It can run unmodified OpenCL applications on Linux clusters, with or without the Mosix system. VCL provides a view of one superserver for a cluster built from a group of GPU units. The components of VCL, its performance and applications are presented in [5].

Our goal is to develop a software framework that allows for unmodified OpenCL applications to transparently and concurrently run on multiple CPU and GPU devices in a cluster. In the case of our application we need a simple functionality, i.e., a calculation speed up, resistance, and ease of use. We perform a static decomposition of the problem in calculating startup, hence dynamic load balancing is superfluous. Our software framework is quite similar to VCL platform [5], however in our solution it is possible to utilize both CPUs and GPUs on computational nodes.

3. HGCC hardware architecture

Beginning our work on utilizing a cluster composed of CPUs and GPUs in cryptography and complex data analysis we focused on three objectives from the hardware perspective:

- validation of cryptographic and cryptanalysis algorithms efficiency on different CPUs and GPUs from many vendors,
- evaluation of different interconnects for the proposed software environment,
- determination of a cost-effective solution.

Due to the above objectives we had to design a relevant architecture of hardware components working together. The HGCC system is a heterogenous cluster system with multi-core CPUs working together with many-core GPUs. It consists of 24 nodes and integrates two types of CPUs: 12 servers with Intel Xeon X5650 processors and 12 servers with AMD Opteron 6172 processors. The system architecture is depicted in Fig. 1. All servers are equipped with advanced GPUs, adequately, NVIDIA Tesla M2050 and AMD FirePro V7800 units.

The choice of GPU in the case of Intel+NVIDIA nodes is quite easy, as there are many HPC solutions based on Intel processors and NVIDIA Tesla GPUs. The situation is more complicated if a user wants to build a cluster of AMD CPUs and GPUs. In our case we had additional requirements on providing different interconnects, hence we had to equip AMD nodes with single width GPUs. Therefore, the set of admissible solutions was quite limited. We investigated four FireStream and FirePro devices (see Table 1 – NVIDIA Tesla included as a reference solution), and finally decided on FirePro V7800 card with the best single precision floating point performance. It is worth to note that FirePro V7800 has a peak performance in the single precision almost two times better than NVIDIA Tesla M2050 and a peak performance in the double precision equal th 0.8 of Tesla's performance. Moreover, AMD GPU is approximately four times cheaper than NVIDIA GPU.

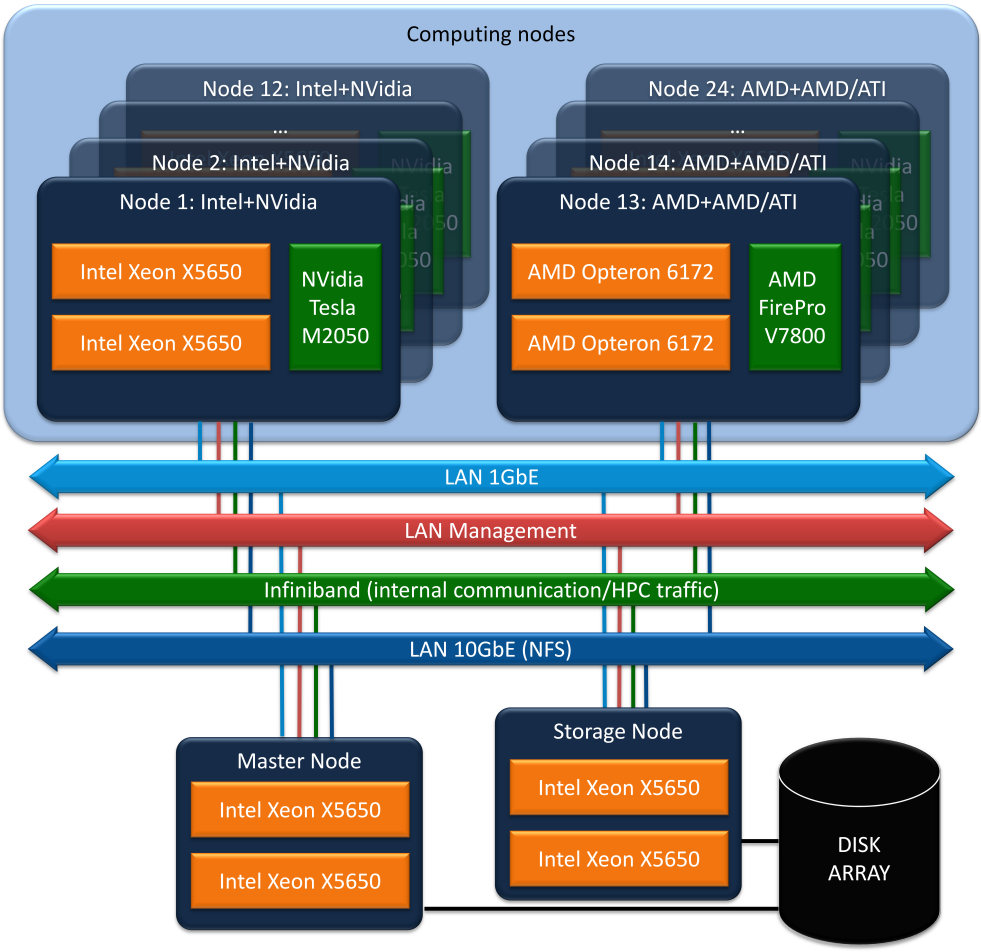

Figure 1. Hybrid system architecture with Intel+NVIDIA and AMD+ATI/AMD nodes

Table 1

List of considered AMD GPUs, where SP* and DP* denotes, adequately, the single and double precision floating point performance in TFlops

Model	SP*	DP*	RAM (size and bandwidth)
AMD FireStream 9350	2.00	0.40	2 GB (128 GB/s)
AMD FirePro V5800	1.10	0.22	1 GB (64 GB/s)
AMD FirePro V7800	**2.02**	**0.40**	**2GB (128GB/s)**
AMD FirePro V7900	1.86	0.46	2 GB (160 GB/s)
NVIDIA Tesla M2050	1.03	0.51	3 GB (148 GB/s)

The computing nodes are supported by dedicated master and storage nodes providing access to disk arrays and management capabilities. Communication between nodes is organized using different interconnects: InfiniBand 4× QDR, 10GbE and 1GbE. Such an excess network configuration allows us to verify the impact of selected interconnects on computation efficiency. Moreover, it is possible to separate communication connected with IO operations from computational traffic. The current configuration assumes utilizing the 10 GbE network for providing access to data storage. InfiniBand and the 1 GbE Ethernet are used for computational purposes.

4. HGCC software framework

The effective utilization of heterogeneous cluster requires developing an integrated software platform to manage calculations and provide interface between the application program and the system kernel. The main goal of our HGCC software framework is to minimize the user's effort during the design, implementation, and execution of the application. It allows the user to focus only on the numerical part of his application. The main idea was to allow user applications to transparently utilize many CPU and GPU devices in a cluster, as if all the devices were on the local computer. All servers' resources such as CPU, GPU, disks, and memory are seen by the user as one unique machine. Hence, applications written for HGCC benefit from the reduced programming complexity of a single computer, the availability of shared memory and multi threads, as in OpenMP (openmp.org/wp), and a concurrent access to cluster nodes and their devices, as in MPI (mpi-forum.org).

In order to take advantage of GPU accelerators from different vendors we decided to use OpenCL, which is a low level GPU programming toolkit, where developers write GPU kernels entirely by themselves with no automatic code generation [18]. OpenCL is an industry standard computing library developed in 2009 that targets not only GPUs but also CPUs and potentially other types of accelerator hardware. In OpenCL efficient implementation requires the preparation of slightly different codes for different devices, however, it is much less complicated than writing code in many native toolkits for NVIDIA and AMD devices.

4.1. Framework architecture

The main goal of the HGCC framework is to provide an environment for parallel calculations that are performed in a cluster formed by heterogenous CPU and GPU devices. Hence, the system facilities are provided in the form of four groups of services. These are:

- *User interface services*, which provide a consistent user interface. The most important tasks of the user interface are as follows: supporting the process of defining a considered application, processing of the calculation results, and providing communication with the user.
- *Calculation management services*, which allocate the calculation processes into cluster nodes and manage the execution of the user's application.

- *Communication services*, which manage communications between running processes and system kernel.
- *Data repository services*, which provide a store for all data objects.

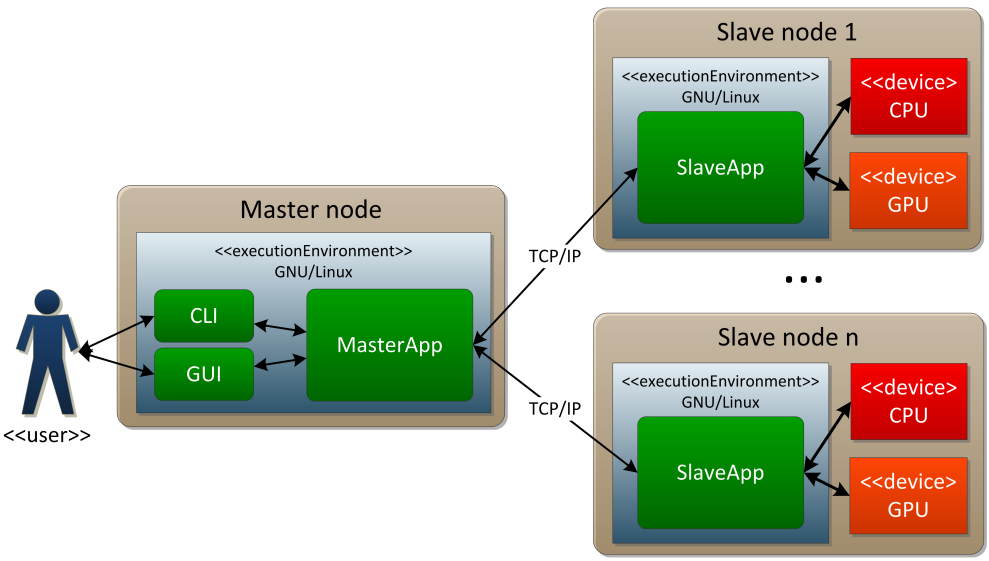

Figure 2. Core components of the cluster framework

The cluster framework consists of several components. The most important are: *MasterApp* – master node application (broker operating in the master node) and *SlaveApp* – the computational node application (a daemon operating in computational nodes) – see Fig. 2.

MasterApp is the main component that is responsible for the user-system communication and calculation management. It provides consistent user interface. Two mechanisms for interacting with our system will be provided: a command-line interface CLI (currently available) and a graphical user interface GUI (in progress). The main functionalities of the interface are: approving tasks assigned by the users, displaying results of calculations, and presenting the current status of each task and cluster node. The *MasterApp* component manages execution of all users' tasks. It is responsible for tasks creation, splitting data to be processed into separate domains, load-balancing, tasks termination, and suspension.

Moreover, the *MasterApp* component is responsible for computational resources management. From the user's point of view all resources are visible as the resources of a local machine. Hence, the master node creates a list of all available resources, monitors and assigns a current occupancy state to each resource. The important functionality of the *MasterApp* component is managing the communication between the master node and the slave (computational) nodes. The considered tasks are: forwar-

ding user tasks, and allocating them to the cluster nodes, approving results, and handling problems with computational nodes when they occur.

The second main component is *SlaveApp*. This component is responsible for calculations that are performed by the assigned server. The computations are managed using dedicated plugins. Plugins are dynamically loaded libraries. The following methods provide the interface to these libraries:

`init()` – responsible for plugin initialization (precomputation, memory allocation, etc.),

`run()` – responsible for running calculations,

`terminate()` – responsible for task termination, memory cleanup, etc.

Each computational node contains some number of resources. In our framework we distinguish and collect information about two types of such resources:

- CPUs – central processing units,
- GPUs – graphics processing units.

The computational resource can be in one of the following states:

- WAITING – ready for loading a new task to execution,
- WORKING – occupied, calculations are executed,
- LOST – lost because of the node failure.

The resource description is an important requirement for providing master node management capabilities. The *resource descriptor* file contains a list of available resources. Similar to the Mosix system it provides information about node's IP, the number of CPU cores and the number of GPU units in a given cluster. The sampled resource descriptor is as follows:

```
slave 192.168.1.1
cpu 4 100
gpu 1 500

slave 192.168.1.2
cpu 12 100
gpu 2 500
```

4.2. Inter-process communication

The current version of the system implements the master-slave communication scheme. It is the natural protocol for applications with data decomposition into blocks and iterative calculations. An XML-based communication protocol is proposed to perform communication between master and slave nodes. It is based on the TCP/IP protocol and BSD sockets. Our goal was to apply a very simple mechanism that fulfills the following requirements:

- flexibility – the protocol should be easy to modify and extend with new messages,
- failure resistance – the protocol should be robust as much as possible,
- text format – not so fast as binary but easy to maintain and debug.

We are aware of the limitations of the presented protocol. Hence, we plan to implement other communication schemes.

4.3. User task

A user's task is to implement the computational task in an object oriented way. Next, the user has to define his problem in the *task descriptor*. The XML Schema specification for building XML files with task description is provided in HGCC. The task descriptor contains: a type of the task, an algorithm, a destination platform, and device. All these parameters are mandatory and parsed inside the *MasterApp* component. The rest of this file is filled by parameters specific to a given task. A sampled task descriptor is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<task xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="schema/descriptor.xsd">
    <cipher algo="des" device="cpu" platform="native">
       <other direction="encrypt" key="5a8dd3ad0756a93ded72b823b19dd877"
         mode="ECB" />
       <file input="/tmp/data_in" output="/tmp/data_out" size="464058292" />
       <split type="maxEquals"/>
    </cipher>
</task>
```

In the descriptor presented above the problem to solve is to cipher data using DES algorithm. The calculations will be performed on CPU and native platform (any native executable form but not based on a virtual machine or interpreted ones). The specific parameters of the algorithms are as follows: a decision about encryption or decryption, a key and a mode of the execution. Next, input and output files are pointed. Finally, the parameter defining the type of the task partitioning is fixed.

4.4. HGCC platform operation

The cluster framework can handle any calculation task, which inherits from any of the core task classes. A committed task is sent to the *MasterApp*. Next, it is divided into smaller tasks. The *MasterApp* creates list of such tasks. They are allocated to the slave nodes, which contain any free resources. Hence, each task can be in one of the following states:

- PENDING – ready for execution, waiting for allocation to a slave node,
- RUNNING – currently running,
- LOST – lost because of a node failure,
- SUSPENDED – terminated with persistent state (may be recovered),
- DONE – task that has been already done.

The state diagram is presented in Fig. 3.

Two operations are performed after the *SlaveApp* initialization: a plugin list is loaded from a plugin descriptor file, and a socket is opened and waiting for *MasteApp*'s

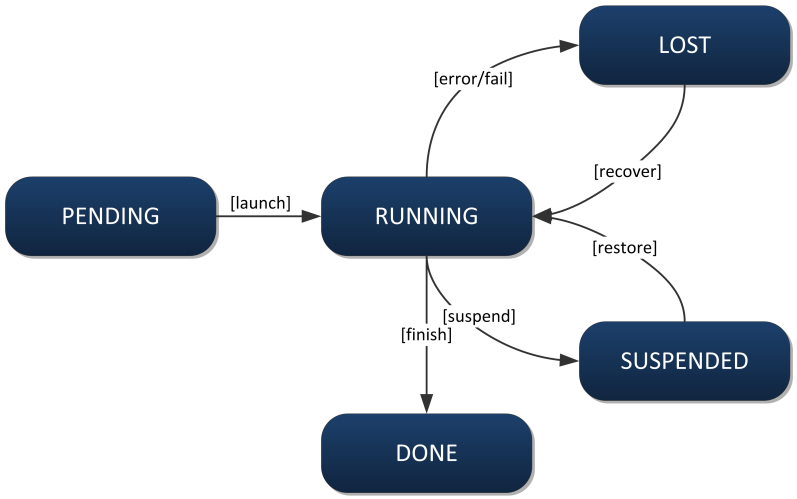

Figure 3. Subtask state machine

connection. The plugin descriptor file contains information about all plugins currently available in the system. Whenever a slave node gets a new set of tasks to execute it looks for available valid plugin, and loads it to the memory. Next, the control flow inside *SlaveApp* splits, and the newly spawned thread launches calculations stored in the loaded plugin.

5. Parallel implementation of cryptography and cryptanalysis

5.1. Cryptography and cryptanalysis on GPU

Graphic processing units allow us to perform massive parallel computations, which can be especially exploited in the field of cryptography and cryptanalysis. The first and the most common application of GPU in cryptanalysis is the password recovery from hashes (usually based on MD4, MD5, SHA-1 or SHA-2). The hashes are easy to compute but extremely hard to inverse. Therefore, the only general technique for recovering a password from hash, which does not take into account algorithm weaknesses, is to scan all potential passwords, compute their hash, and test the coincidence. Cryptographic hash functions are based on integer and binary operations such as: the addition modulo power of two, bit shift and rotation, bitwise xor, bitwise or, bit negation and words permutation. Those operations are natively supported by GPU computing units, allowing for maximum instruction throughput and the full use of their computational power.

Three main approaches to password strength validation that are usually considered are: brute-force, rainbow-table, and dictionary test. The first two of these were found very suitable for porting to GPU. Brute-force, thus not very sophisticated,

is the most common approach used together with GPUs. There are many brute-force GPU implementations, among which the fastest are: whitepixel [4], ighashgpu [11], BarsWF [30], ++oclHashcat plus[2], and password recovery tools offered by Elcomsoft[1]. When only one GPU is used, they are capable of checking up to several billion passwords per second (depending on the GPU type and version, whereas AMD chips tend to perform better). All of the applications mentioned are capable of using multiple GPUs installed on one host system, and in that case performance usually scales linearly. To the authors knowledge only the solution offered by Elcomsoft is capable of distributing the workload on many hosts [1].

Another method which benefits from GPU capabilities is the rainbow-table technique, which is based on the work of Philippe Oechslin [26, 27]. This algorithm is an example of time-memory trade-off. At the beginning it pre-computes hashes for all potential passwords, which is the most time consuming operation, and stores them in special compressed form called a rainbow table. The actual password search is reduced to several million hash calculations and rainbow table look-ups. In contrary to the brute-force approach the rainbow-table attack does not guarantee success, but speeds up the search significantly. There are several rainbow table implementations, among which Rainbowcrack is the most popular[3]. The creators of this application appreciated the capabilities of the GPUs both for table preparation and actual password testing, and now they offer a version with NVIDIA CUDA support. Other developers followed this approach but used the AMD chips instead.

Symmetric ciphers were also found to give significant speed when ported to GPU, especially when used to encrypt or decrypt huge data volumes or attack secured communication protocols. Due to its structure and popularity the AES algorithm is most often implemented [8, 7, 20, 24, 12], but other methods were also successfully optimized, e.g. RC4, 3DES, DES and BlowFish [21, 32, 22].

When asymmetric cryptography is concerned (RSA and ECC) with conjunction with GPUs, the main research goal is to design a parallel multi-precision arithmetic routines and Montgomery reduction algorithms, which are the basic building blocks. There were several efforts, which indicate that in some applications, GPU outperforms the best available CPU-based implementations [10, 17, 13, 25], but further research is still required.

Last but not least, the GPU are also used to increase the performance of the asymmetric cryptography methods which are alternative to RSA and ECC, eg. NTRU and Goldreich-Goldwasser-Halevi(GGH) [15, 14], which are examples of a lattice-based cryptography.

5.2. Numerical results in HGCC

Our HGCC cluster is dedicated to performing complex calculations and processing large amounts of data. The focus is on the parallel implementation of cryptography and cryptanalysis algorithms. We have performed multiple tests to show the efficiency

of parallel implementation of cryptanalysis on GPU, and utility and scalability of our cluster system. The selected results are presented in Tables 2–4.

The goal of the first series of experiments was to compare the efficiency of parallel implementations of rainbow table generation on CPU and GPU devices. A rainbow table is a precomputed table used for reversing one-way functions, especially cryptographic hash functions. This technique is placed between a CPU-intensive exhaustive search and storage-intensive full precomputation (often infeasible). Rainbow tables enable us to make time-memory tradeoffs: we can sacrifice more CPU time to reverse such a function keeping storage requirements under strong constrains or minimize computational effort using larger storage capacity.

Our experiments were performed on devices provided by different vendors:

CPU Intel : Intel Xeon X5650, 2.66GHz/3.06GHz turbo, 6 cores/12 threads, 6×256 L2, 12MB L3 cache.

CPU AMD : AMD Opteron 6172, 2.1GHz, 12 cores/12 threads, 12×512KB L2, 12MB L3 cache.

GPU NVIDIA : NVIDIA Tesla M2050, 448 CUDA cores, 384-bit memory bus.

GPU AMD : AMD FirePro V7800, 1440 stream processors (equivalent of 288 CUDA cores), 256-bit memory bus.

The calculation times for three series of experiments performed for various sizes of rainbow tables $RT[n, l]$, where n denotes number of rows (chains), l number of columns (chain length) are presented in Table 2. The following rainbow tables with assumed success rate equal to 43.766% per table (99% for 8 tables) were generated:

- RT_1: $n = 491080457$, $l = 1000$, size 7.4 GB,
- RT_2: $n = 49041409$, $l = 10000$, size 750 MB,
- RT_3: $n = 19614714$, $l = 25000$. size 300 MB.

Table 2

The calculation times (in seconds) for rainbow tables generation

Device	RT_1	RT_2	RT_3
GPU			
NVIDIA Tesla M2050	1358.5	1363.0	1374.0
AMD Firepro V7800	1145.3	1127.2	1142.5
CPU – calculations done using single thread			
Intel Xeon X5650 @2.67 GHz	74558.7	74781.3	74554.2
AMD Opteron 6172 @2.1 GHz	184758.7	184100.6	183593.2
CPU – calculations done using number of threads equal number of processors' cores			
Intel Xeon X5650 @2.67 GHz	6211.4	6217.8	6213.1
AMD Opteron 6172 @2.1 GHz	7656.3	7658.6	7660.3

It should be noted that the number of calculations C needed to create a table is equal to $C = n \cdot l$. Our experiments satisfied the theoretical considerations, as differences in time are minor and can be accounted by thread scheduling. Since

table generation can be done in parallel using domain decomposition the acceleration factor on CPU units is equal the number of utilized cores. Even though, GPU implementations were much more efficient that CPU ones.

The results presented in Table 3 show the performance of different CPU/GPU devices in the MD5 hashes generation [29]. The table collects the number of hashes generated per second using a multi-threaded version of the MD5 algorithm for CPUs and the same number of hashes generated using a single-threaded code for CPU with appropriate kernels for GPU (OpenCL). The best results were obtained for the OpenCL version of the MD5 hash function executed on AMD FirePro V7800 – almost 1900 million hashes. The efficiency of NVIDIA Tesla M2050 was worse, but still over two times better than results obtained using only CPU units. It should be pointed that in case of CPUs, two Opteron processors with 12 cores each, generated more hashes than two Intel Xeons with higher frequency but only 6 cores each plus Hyper-Threading technology.

Table 3

Number of generated MD5 hashes per second (in millions) – no hash inversions, no precomputations, full MD5 rounds

Device	Number of MD5 hashes
Intel Xeon X5650	342.8
AMD Opteron 6172	443.6
NVIDIA Tesla M2050	935.1
AMD FirePro V7800	1896.2

Two hash functions (MD5 and SHA1 [29]) were also used in order to verify the scalability of the HGCC platform. We tested both implementations working on CPUs and GPUs. The numerical experiments were carried out on a group of 8 Intel+NVIDIA nodes. As can be seen in Table 4 and Figure 4 the algorithms scales up very well – the speed up value for 4 nodes is between 3.46 and 3.81 and for 8 nodes is up to 7.81.

Table 4

Scalability of hashes generation using MD5 and SHA1 algorithms – hashes per second (in millions)

Nodes	MD5(CPU)	MD5(GPU)	SHA1(CPU)	SHA1(GPU)
1 node	342.8	935.1	156.0	181.0
4 nodes	1 330.8	3 236.9	580.1	707.0
8 nodes	2 513.8	5 259.9	1 131.2	1 414.0

6. Summary and conclusion

In this paper we presented the components of the heterogenous cluster system integrating CPU and GPU devices of various types. We described both the hardware

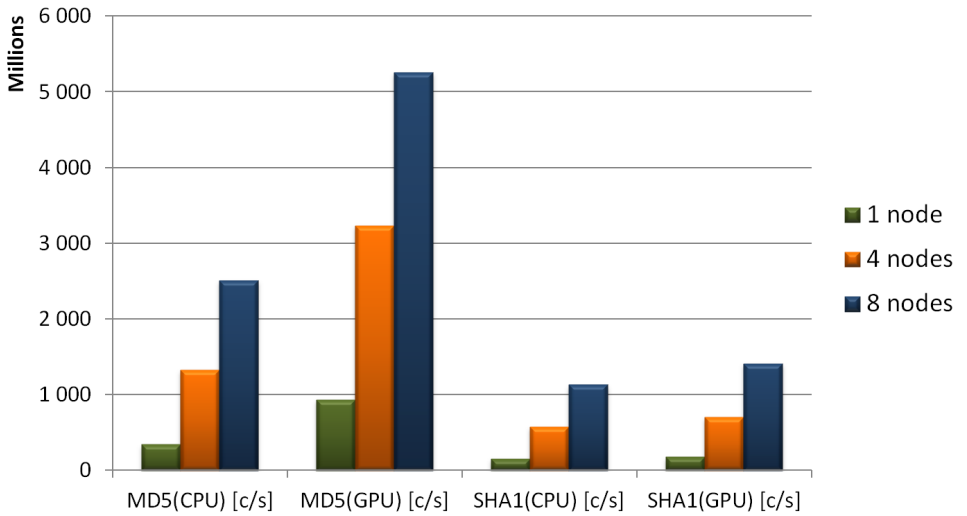

Figure 4. Scalability of hashes generation using MD5 and SHA1 algorithms

architecture and the software platform that provides a single system image in our cluster. The cluster system was designed to be powerful, effective, scalable, flexible, and easy to use. The cluster is especially useful in complex calculations and parallel processing of large amounts of data, in which speed of calculation and data decomposition are of essence, such as cryptography and cryptanalysis algorithms. Our experimental results presented in this paper demonstrate the effectiveness and scalability of the cluster system. As a final observation we can say that heterogeneous computing systems offer a new opportunity to increase the performance of parallel HPC applications on clusters, by combining traditional CPU and general purpose GPU devices.

Acknowledgements

The work was supported by the National Centre for Research and Development (NCBiR) under grant number O R00 0091 11.

References

[1] *Elcomsoft – privately owned company offering premium password recovery software.* `http://www.elcomsoft.com/`.

[2] *Hashcat – advance password recovery project website.* `http://hashcat.net/oclhashcat-plus/`.

[3] *Rainbowcrack project website.* `http://project-rainbowcrack.com/`.

[4] *Whitepixel project website with brute-force crackes comparision.* `http://whitepixel.zorinaq.com/`.

[5] Barak A., Shiloh A.: *The mosix virtual opencl (vcl) cluster platform.* [in:] *Proc. Intel European Research and Innovation Conf.*, page 196. Leixlip, 2011.

[6] Berman F., Fox G., Hey A. J. G.: *Grid Computing: Making the Global Infrastructure a Reality.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

[7] Bos J., Osvik D., Stefan D.: *Fast implementations of aes on various platforms.* Technical report, Cryptology ePrint Archive, Report 2009/501, 2009. `http://eprint.iacr.org`, 2009.

[8] Di Biagio A., Barenghi A., Agosta G., Pelosi G.: *Design of a parallel aes for graphics hardware using the cuda framework.* [in:] *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–8. IEEE, 2009.

[9] (ed.) W.-M. W. H., editor. *GPU Computing Gems Emerald Edition.* Morgan Kaufman.

[10] Fleissner S.: *Gpu-accelerated montgomery exponentiation. Computational Science–ICCS 2007*, pp. 213–220, 2007.

[11] Golubev I.: *Ighashgpu project website.* `http://www.golubev.com/hashgpu.htm`.

[12] Harrison O., Waldron J.: *Aes encryption implementation and analysis on commodity graphics processing units. Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 209–226, 2007.

[13] Harrison O., Waldron J.: *Efficient acceleration of asymmetric cryptography on graphics hardware. Progress in Cryptology–AFRICACRYPT 2009*, pp. 350–367, 2009.

[14] Hermans J., Vercauteren F., Preneel B.: *Implementing ntru on a gpu.* 2009.

[15] Hermans J., Vercauteren F., Preneel B.: *Speed records for ntru. Topics in Cryptology-CT-RSA 2010*, pp. 73–88, 2010.

[16] Karbowski A., Niewiadomska-Szynkiewicz E.: *Parallel and distributed computing (in Polish).* WUT Publishing House, 2009.

[17] Kaya Koc C., Acar T., Kaliski Jr B.: *Analyzing and comparing montgomery multiplication algorithms. Micro, IEEE*, 16(3):26–33, 1996.

[18] Kindratenko V., Enos J., Shi G., Showerman M., Arnold G., Stone J., Phillips J., Hwu W.: *Gpu clusters for high-performance computing.* [in:] *Proc. PPAC'09 Workshop*, 2009.

[19] Kunzman D. M., Kalé L. V.: *Programming heterogeneous clusters with accelerators using object-based programming. Sci. Program.*, 19:47–62, January 2011.

[20] Le D., Chang J., Gou X., Zhang A., Lu C.: *Parallel aes algorithm for fast data encryption on gpu.* [in:] *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 6, pp. V6–1. IEEE, 2010.

[21] Li C., Wu H., Chen S., Li X., Guo D.: *Efficient implementation for md5-rc4 encryption using gpu with cuda.* [in:] *Anti-counterfeiting, Security, and Identification in Communication, 2009. ASID 2009. 3rd International Conference on*, pages 167–170. IEEE, 2009.

[22] Liu G., An H., Han W., Xu G., Yao P., Xu M., Hao X., Wang Y.: *A program*

behavior study of block cryptography algorithms on gpgpu. [in:] *Frontier of Computer Science and Technology, 2009. FCST'09. Fourth International Conference on*, pp. 33–39. IEEE, 2009.

[23] Lottiaux R., Boissinot B., Gallard P., Vallee G., Morin C.: *Openmosix, openssi and kerrighed: A comparative study.* [in:] *Proceeding of IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, vol. 2, pp. 1016–1023, 2005.

[24] Mei C., Jiang H., Jenness J.: *Cuda-based aes parallelization with fine-tuned gpu memory utilization.* [in:] *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–7. Ieee, 2010.

[25] Moss A., Page D., Smart N.: *Toward acceleration of rsa using 3d graphics hardware.* [in:] *Proceedings of the 11th IMA international conference on Cryptography and coding*, pp. 364–383. Springer-Verlag, 2007.

[26] Oechslin P.: *Making a faster cryptanalytic time-memory trade-off. Advances in Cryptology-CRYPTO 2003*, pp. 617–630, 2003.

[27] Oechslin P.: *Password cracking: Rainbow tables explained. Constituent Contributions*, 14, 2005.

[28] Osiński P., Niewiadomska-Szynkiewicz E.: *Comparative study of single system image clusters.* [in:] *Evolutionary Computation and Global Optimization*, vol. 169, pp. 145–154, 2009.

[29] Paar C., Pelzl J., Preneel B.: *Understanding Cryptography: A Textbook for Students and Practitioners.* Springer, 2010.

[30] Svarychevski M. A.: *Barswf project website.* `http://3.14.by/en/md5`.

[31] Tsoi K., Luk W.: *Axel: a heterogeneous cluster with fpgas and gpus.* [in:] *Proc. of the 18th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '10)*, pp. 115–124, 2010.

[32] Wang Z., Graham J., Ajam N., Jiang H.: *Design and optimization of hybrid md5-blowfish encryption on gpus.*

Affiliations

Michał Marks
Institute of Control and Computation Engineering, Warsaw University of Technology, Nowowiejska 15/19, 06-665 Warsaw, Poland, `mmarks@ia.pw.edu.pl`;
Research and Academic Computer Network (NASK), Wawozowa 18, 02-796 Warsaw, Poland, `mmarks@nask.pl`

Jarosław Jantura
Research and Academic Computer Network (NASK), Wawozowa 18, 02-796 Warsaw, Poland, `jaroslaw.jantura@nask.pl`

Ewa Niewiadomska-Szynkiewicz
Institute of Control and Computation Engineering, Warsaw University of Technology, Nowowiejska 15/19, 06-665 Warsaw, Poland, `ens@ia.pw.edu.pl`;
Research and Academic Computer Network (NASK), Wawozowa 18, 02-796 Warsaw, Poland, `ewan@nask.pl`

Przemysław Strzelczyk
Research and Academic Computer Network (NASK), Wawozowa 18, 02-796 Warsaw, Poland, przemyslaw.strzelczyk@nask.pl

Krzysztof Góźdź
Hewlett-Packard Poland, krzysztof.gozdz@hp.com

Received: 16.12.2011
Revised: 27.02.2012
Accepted: 23.04.2012