A Parallel Computing System with Specialized Coprocessors for Cryptanalytic Algorithms

Wolfgang Kastl
wolfgang.kastl@students.fh-hagenberg.at

Thomas Loimayr
thomas.loimayr@students.fh-hagenberg.at

Abstract: In this paper we present a scalable, parallel computing system consisting of specialized processors primarily designed for the implementation of cryptanalytic algorithms. Even though the system was developed in regard to solve cryptanalytic problems, it is suitable for many other tasks which can benefit from the enormous computing power of the system (e.g. malware analysis). In addition to the use of multi-core CPUs, the computing system takes advantage of graphic cards (GPUs) and FPGAs as specialized coprocessors. Thus, it gains an edge over other conventional parallel computing systems.

1 Introduction

In the last decade, the clock frequency of traditional processors (CPUs) increased significantly. Due to physical laws, such as thermal density, the increase of clock speed recently hit a wall. Thus, the CPU vendors were forced to accelerate the computation power of CPUs by integrating multiple cores onto a single die.

In recent years, GPUs and FPGAs became very popular as coprocessors in high performance computing systems. These two types of specialized coprocessors can very often achieve much better performance than multi-core CPUs for certain types of computations. Modern GPUs provide a huge amount of massive parallel processing units and are therefore well-suited for high performance computing. Furthermore, they contain up to 4 GB onboard memory and are capable to exceed 100 GB/sec of internal memory bandwidth. In contrast to GPUs, the high performance capability of FPGAs for certain types of applications has been well-known for a long time. Since multi-core CPUs, GPUs and FPGAs are different technologies, they achieve widely different performance on certain tasks. A comparison on different applications between the three technologies is presented in [Shu08].

Nowadays, a fair amount of high performance computing systems already take advantage of coprocessors such as GPUs and FPGAs. Most of them use either GPUs or FPGAs but rarely benefit from both types of coprocessors. In this paper we introduce a scalable, parallel computing system, which takes advantage of GPUs as well as FPGAs as coprocessors. Furthermore, a lot of today's most popular parallel computing systems with focus

on cryptanalysis such as the COPACOBANA [Ruh06] or the PS3-Cluster [Lab] consist of dedicated hardware especially built to solve a small range of specific problems. In contrast to these systems, our cluster exclusively consists of standardized interfaces and components and is freely scalable and configurable. Hence, it is not limited to a small number of problems.

In the following section, an overview of works related to our cluster presented in this paper is given. In Section 3, we present the hardware architecture of the cluster. Section 4 describes the software framework which is necessary to communicate between nodes and to provide the capability of using GPUs and FPGAs as coprocessors. Section 5 explains how the communication between different cluster nodes is handled. Section 6 covers suitable implementations of cryptographic and cryptanalytic algorithms on the cluster. The last section concludes with the current status of the project and gives an outlook on future tasks and improvements of the cluster.

2 Related Work

Obviously, a highly specialized system like COPACOBANA cannot be directly compared to our cluster, as both are completely different systems. However, in recent years, various COTS cluster systems were introduced, which take advantage of coprocessors like GPUs and FPGAs. In 2009, a first prototype of the Quadro Plex (QP) cluster [Mic09] was introduced by NCSA[1]. Each node of the prototype system is equipped with two AMD Opteron CPUs, four NVIDIA G80GL GPUs and one Xilinx Virtex-4 LX100 FPGA. In 2010 the heterogeneous cluster Axel [TL10] was introduced. The first prototype uses 16 nodes, equipped with one AMD Phenom Quad-Core CPU, one NVIDIA Tesla C1060 and one Xilinx Virtex-5 LX330 FPGA hosted on an ADM-XRC-5T2 card.
While these two systems provide a strong basis for developing any new applications, the goal of our cluster is to additionally provide a library of cryptographic modules that can be loaded into the cluster's coprocessors. Hence, the system allows to freely develop new applications or to use existing modules for e.g. cryptanalysis, which makes it easy to configure and easy to use.

3 Physical Architecture

The cluster is composed of a head node that performs management tasks; input nodes that distribute the data to be processed; compute nodes that execute the cryptanalytic algorithms; and output nodes that collect the output from the compute nodes to aggregate and concentrate it. The physical architecture of the cluster is depicted in Figure 1. Each compute node as well as the head node can additionally act as input or output node or both. It is also possible to define separate nodes as input or output nodes which do not

[1] National Center for Supercomputing Applications at the University of Illinois

act as compute nodes. The task of assigning different roles (input, output or compute) to the nodes has to be done on the head node of the cluster. Thus we can see that the cluster is freely configurable on the head node and does not have a predefined and fixed structure. As a result, the structure of the cluster can be configured to best fit to the underlying cryptanalytic algorithm.

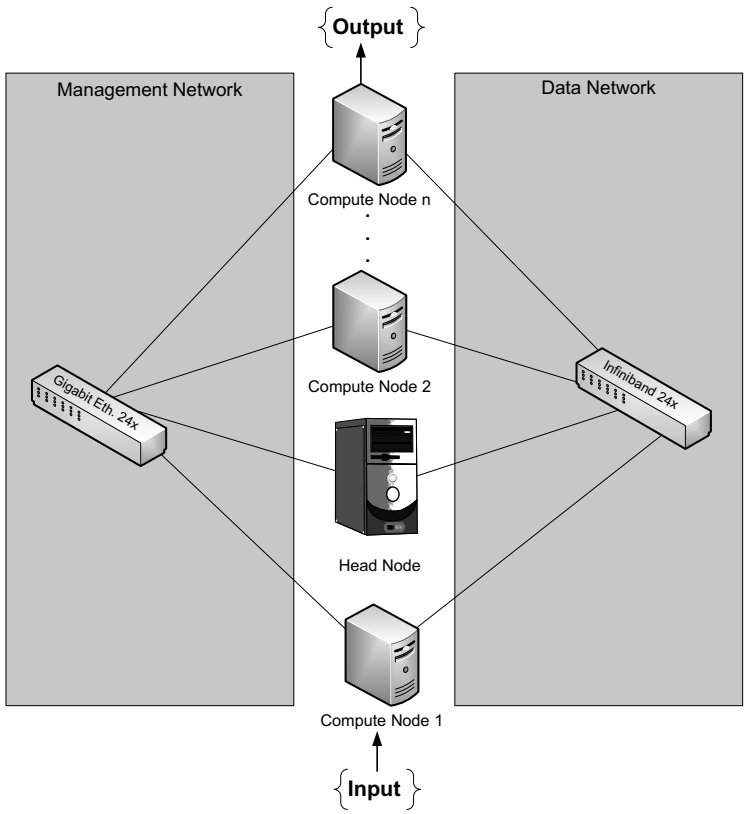

Figure 1: Physical Architecture of the Cluster

3.1 Networks

The architecture of the cluster consists of two different networks - the data network and the management network. The use of two separated networks allows performing management tasks (e.g. monitoring the cluster nodes) over the management network while the actual data throughput over the data network is not affected in any way. Each node in the cluster is consistently connected to both networks. A simplified architecture of the networks is depicted in Figure 1.

Data Network. The data network represents the high-speed Infiniband network. A 24-port Infiniband switch is used to enable a high-speed interconnection between the nodes. The data network transmits data to input nodes, distribute data to compute nodes, eventually exchange data between compute nodes and finally hand over the result to the output nodes. Depending on the type of computation an enormous amount of data might be transmitted over the data network.

Management Network. The management network represents the Gigabit Ethernet network where the nodes are connected through a 24-port Gigabit Ethernet switch. Data that is transmitted over the management network contains status messages, error codes, debug information and any other kind of information necessary for the management of the cluster. Moreover, the management network is responsible for node development and node updates.

3.2 Node Configuration

Each single compute node consists at least of the following standard hardware components:

- Motherboard with at least four PCIe slots which was a prerequisite for our cluster architecture.
- Multi-core CPU and huge main memory: Each node is equipped with a quad-core CPU and 16 GB of main memory.
- Gigabit Ethernet network card: Used to connect to the management network.
- Infiniband Host Channel Adapter (HCA): Used to connect to the data network.

Furthermore, each compute node consists of one or more of the following specialized coprocessors:

- NVIDIA GPUs: Nodes are (at the moment) equipped with NVIDIA graphic cards which come with CUDA[2] support.
- FPGA boards: For current testing a Xilinx ML605 Evaluation Board with one single Virtex-6 FPGA is used. In later versions of the cluster the evaluation board will be replaced by an FPGA board equipped with several FPGAs working in parallel.

3.3 Architecture Constraints

At the moment, the cluster uses a 24-port Infiniband switch for the data network and a 24-port Ethernet switch for the management network allowing to connect a maximum number

[2]Compute Unified Device Architecture

of 24 nodes. This means, the current architecture of the cluster is capable of operating 23 compute nodes and one head node. Since the nodes are equipped with motherboards consisting of 4 PCIe slots, each node can be equipped with a maximum number of three coprocessor boards (one PCIe slot is already used for the Infiniband HCA). Thus, the current cluster architecture is restricted to 69 (23 nodes each with 3 coprocessor boards) coprocessor boards in total.

4 Software Framework

In this section we introduce the software framework which is necessary for the computation of algorithms on the cluster. The software framework provides functionality which allow us to perform the following tasks:

- Configuring the cluster nodes including FPGAs and GPUs in a unitary way,
- assigning the data flow through the cluster and
- operating and monitoring the cluster.

4.1 Management Software

The management software which is running on the head node is responsible for managing the entire cluster. It configures the nodes for the appropriate tasks, sets the data flow within the cluster, initializes the nodes and requests status information regularly to provide a detailed status overview over the whole cluster. Furthermore, the management software is capable of collecting debugging information which is especially useful for the cluster development.

Microsoft HPC Server. We decided to use Microsoft Windows Server 2008 with the Windows High Performance Cluster (HPC) Extension [Mic08] as cluster management software. It provides utile cluster management functions and fully integrates a Microsoft release of the Message Passing Interface (MS-MPI). Due to the fact that the Microsoft HPC Server does not support GPUs and FPGAs as coprocessors, we need additional management software to fully integrate the coprocessors into the cluster environment. The additional management software is part of the MPI program and runs exclusively on the head node.

4.2 Grid Software

The grid software is running on each compute node. It is a main element of the cluster and receives commands from the management software of the head node to configure the

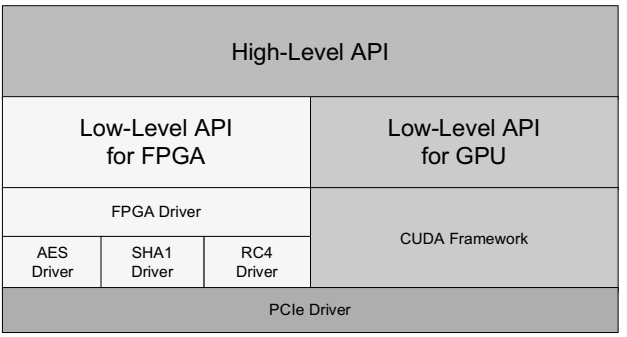

Figure 2: High- and Low-Level API

underlying specialized coprocessors.

The communication between the nodes is accomplished through MS-MPI which allows, in combination with Infiniband, high speed data transfer using Remote Direct Memory Access (RDMA) [LWP08]. Since the Infiniband interface cannot be directly accessed by the coprocessors, the communication has to be initialized by the grid software which has full access to the operating system's Infiniband interfaces.

4.3 Communication Layer

Figure 2 shows the different layers needed to communicate with the underlying coprocessors. As illustrated in Figure 2, the high-level API is situated on the top. The next layer consists of the low-level API which is actually splitted into the FPGA part and the GPU part. Underneath the low-level API, specific drivers required for FPGA development as well as the CUDA framework used for GPU programming can be found. The following two sections describe the high-level API and the low-level API in detail.

High-level API. The high-level API is implemented for abstraction purposes. It provides unified functions for accessing both coprocessor types and is responsible for the following tasks:

- Get the node's hardware information,
- configure the coprocessors related to the given task,
- initiate the coprocessors,
- get status information from the coprocessors and
- get debug information from the coprocessors.

Low-level API. The functions of the low-level API are invoked by the high-level API functions and are directly accessing the drivers of the coprocessors. Basically, the low-level API is divided into two parts to access either FPGAs or GPUs. The FPGA-part and the GPU-part of the low-level API differ in several aspects.
The low-level API for FPGAs is smaller, as any algorithms are directly implemented onto the FPGAs and can be loaded when they are used. The functions of the API are only used for accessing them. Contrary, on the GPU side a library of usable algorithms exists where the algorithms are fully implemented.

Driver Architecture. As illustrated in Figure 2, the low-level API on the side of the GPUs is built on the general CUDA driver for Nvidia graphic cards, which does not need to be modified for a proper use within the cluster.

In contrast, each configuration (actually each algorithm) on an FPGA differs in design, input and output parameters. Thus, a core FPGA driver underneath the low-level API for FPGAs is responsible for loading the specific algorithm driver (e.g. driver for AES, SHA1 or RC4) corresponding to the algorithm. Only if the algorithm driver was initiated successfully, the low-level API is able to communicate with the underlying FPGA. The flexible driver design (the core FPGA driver in combination with the algorithm drivers) enables to run different algorithms or multiple instances of the same algorithm in parallel on a single FPGA board.

5 Inter-Node Communication

This section explains in detail how the communication between two nodes is handled. No matter between which types of nodes (input, output, compute or head node) data is transmitted, any communication is handled the same way via MPI calls. Thereby, MPI is responsible for initiating nodes to send data and allowing nodes to receive data.

While management tasks are handled by the nodes' CPUs, the actual calculations for the cryptographic tasks are executed by the coprocessors. Due to the fact that the coprocessors are not capable of disposing MPI calls, the communication between nodes over the data network has to be triggered by the nodes' CPUs.

The following steps illustrate the principle of node communication between two nodes according to Figure 3.

Node 0. This node plays the role of the sender. It is assumed that the input buffer already contains data and the output buffer is empty. Both memory addresses are known by the coprocessor of the node. The following steps are executed:

1. The coprocessor fetches the input data and processes it.
2. After saving the result into the output buffer an interrupt is provoked.

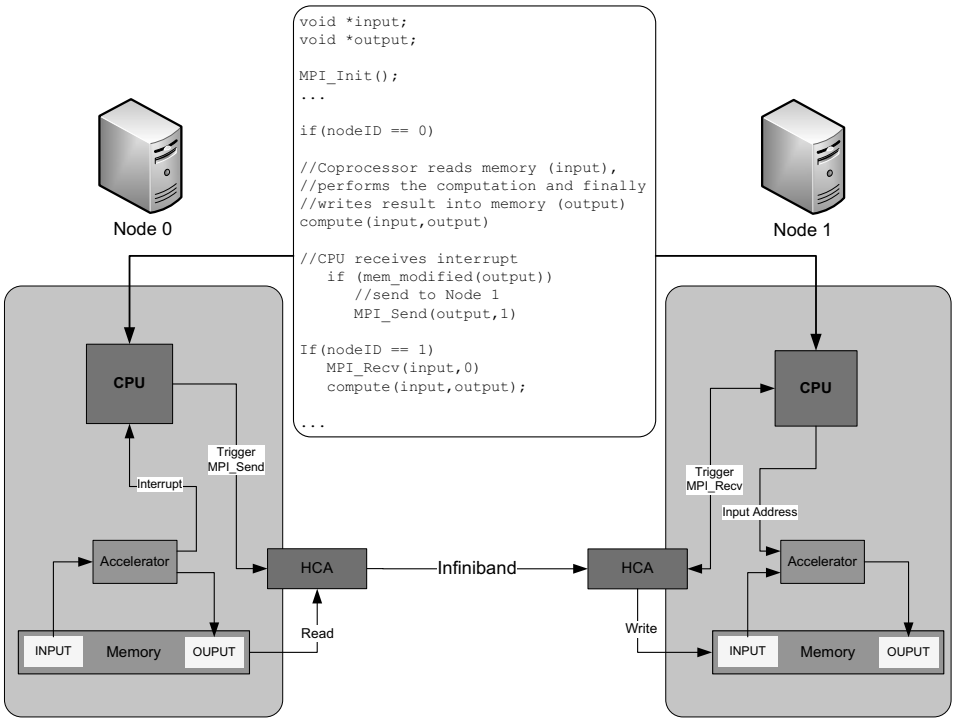

Figure 3: Inter-Node Communication

3. The CPU triggers an MPI call (MPI_Send()), which incites the HCAs to write the content of the first node's output buffer into the second node's input buffer.

Node 1. This node receives data from Node 0. The following are executed:

1. The CPU has to call MPI_Recv() to make the node's HCA ready to receive data.
2. The HCA stores the received data into the input buffer.
3. MPI_Recv() finishes, which indicates a successful data transmission.
4. The CPU sends the memory address of the filled input buffer to the coprocessor.
5. The coprocessor is now capable of processing the input data.

The white box between the nodes in Figure 3 shows a simplified pseudo-code example of how MPI is being used to control the communication. The two basic MPI functions *MPI_Send* and *MPI_Receive* are used to transmit data between the nodes depending on their node IDs. Although the above example shows just one coprocessor per node, the

Pwd: 3rr0r6		Pwd: xavier9	
No. of Nodes	Time [s]	No. of Nodes	Time [s]
1	11	1	367
2	6	2	183
4	1	4	87

Table 1: Results from testing the distributed SHA-1 cracker.

communication principle is very similar for two or more coprocessors included in a single node.

6 Implementations of Algorithms

The presented scalable and parallel computing system is capable of executing various types of algorithms. As this system is developed with focus on cryptographic and cryptanalytic issues, it will mainly deal with tasks of the following sections.

6.1 Brute Force Algorithms

Currently, a distributed SHA-1 cracking algorithm can be executed on the cluster, which uses brute forcing to get the original plaintext from SHA-1 hashes. The algorithm is implemented in CUDA for NVIDIA GPUs and runs in parallel on multiple compute nodes. It generates plaintext strings by using a given character-set. These strings are hashed and compared with the input hash. If both hashes are equal, the plaintext is found. The algorithm requires a SHA-1 hash as input, the length of the plaintext, a character-set and an input-offset for each cluster node. The input-offset defines the partitioning of the input data, which are the generated plaintext strings. The partitioning is carried out by dividing the generated strings by their beginning letter. If a character-set *a-z* would be used for two nodes, the first node would generate strings with the beginning letters *a-m* and the second node with the beginning letters *m-z*. According to that, the input-offset would be *1* for the first node and *2* for the second node, which would divide the input strings into two halfs.

The algorithm was tested for selected passwords on multiple cluster nodes. Two examples are illustrated in Table 1. The character-set *a-z0-9* is used. As the required time shows, the algorithm scales quite well with the amount of nodes. A doubling of the number of nodes, halves the duration on average until the algorithm finishes. In case of the password *3rr0r6* the jump from 2 to 4 nodes decreases the time even to a sixth part of the required time. The reason, why this may happen is illustrated in figure 4, which represents the used character-set. The beginning letter of *3rr0r6* is located in the last half of the specified set. This half pictures the beginning letters that the second node uses for string generation. The node has to generate strings with 11 different beginning letters, before it

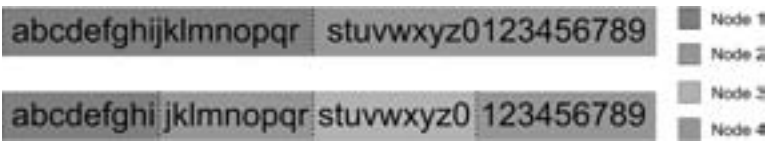

Figure 4: Partitioning of the beginning letters for the cluster nodes.

reaches the ”3”. Using four nodes, the fourth node only has to generate strings with two different beginning letters until it reaches the ”3”. Hence, using four equal nodes, cracking the hashed password *3rr0r6* causes a relatively high decrease of required cracking time, which is a stroke of luck. In average, the speed-up is linear meaning that the calculation time halves, when the amount of nodes is doubled.

6.2 Cryptanalytic Algorithms

Cryptanalytic tasks can be very complex and vary extremely in structure, execution time, required resources, and their capability of parallelization. Only a multi-purpose system that allows free configuration is capable of executing various cryptanalytic algorithms in an efficient way.

7 Future Work

The cluster is still in an early state of development. Recently, we encountered problems concerning the FPGA configuration. At the moment it is not possible to load images onto an FPGA board and use them for calculations within the cluster. The current sample application allows to bruteforce SHA-1 hashes by using only NVIDIA GPUs. Until now, only one algorithm module can be loaded onto the GPUs. In future, multiple crypto-modules shall be developed, which may be used with GPUs and/or FPGAs. Additionally, a graphical user interface shall provide a more intuitive handling of the cluster. The goal of our cluster is to provide a scalable and comprehensive cluster system, which allows easy handling, by the use of prepared and configurable modules.

References

[Lab] Laboratory for cryptologic algorithms. *PlayStation 3 computing breaks 2^{60} barrier*. URL, http://lacal.epfl.ch/page81774.html.

[LWP08] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K. Panda. *High Performance RDMA-Based MPI Implementation over InfiniBand*. Computer and Information Science, The Ohio State University, June 2008. URL, http://nowlab.cse.ohio-state.edu/publications/journal-papers/2004/liuj-ijpp04.pdf.

[Mic08] Microsoft. *Windows HPC Server 2008: Technical Overview of Windows HPC Server*, June 2008. URL, www.clustervision.com/Windows_HPC_Server_2008_Technical_Overview.pdf.

[Mic09] Micheal Showerman et al. *QP: A Heterogeneous Multi-Accelerator Cluster*. University of Illinois at Urbana-Champaign, Urbana, March 2009. URL, web-test.ncsa.illinois.edu/~kindr/papers/lci09_paper.pdf.

[Ruh06] Ruhr University of Bochum and University of Kiel. *COPACOBANA: A Codebreaker for DES and other Ciphers*, October 2006. URL, http://www.copacobana.org/paper/copacobana_CHES2006.pdf.

[Shu08] Shuai Che, Jie Li et al. *Accelerating Compute-Intensive Applications with GPUs and FPGAs*. Departments of Electrical and Computer Engineering and Computer Science, University of Virginia, May 2008. URL, http://www.nvidia.com/docs/IO/67189/che_sasp08.pdf.

[TL10] Kuen Hung Tsoi and Wayne Luk. *Axel: A Heterogeneous Cluster with FPGAs and GPUs*. Department of Computing, Imperial College London, UK, February 2010. URL, www.doc.ic.ac.uk/~wl/papers/10/fpga10bt.pdf.