GANDALF
A GPU-based Architecture For Network and Data Level Flows

Deliverable 1.1

The design document of GANDALF

European Union
European Social Fund

OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS, CULTURE & SPORTS
MANAGING AUTHORITY

NSRF 2007-2013
programme for development
EUROPEAN SOCIAL FUND

Co-financed by Greece and the European Union

1 Introduction

The emergence of commodity *many*-core architectures, such as multi-core CPUs and modern graphics processors (GPUs) has proven to be a good solution for accelerating many network applications, and has led to their successful deployment in high-speed environments [1–5]. Recent trends have shown that certain network packet processing operations can be implemented efficiently on GPU architectures. Typically, such operations are either computationally intensive (e.g., encryption [3]), memory-intensive (e.g., IP routing [2]), or both (e.g., intrusion detection and prevention [4–6]). Modern GPU architectures offer high computational throughput and hide excessive memory latencies.

Unfortunately, the lack of programming abstractions and GPU-based libraries for network streaming processing—even for simple tasks such as packet decoding and filtering—increases significantly the programming effort needed to build, extend, and maintain high-performance GPU-based network applications. More complex critical-path operations, such as flow tracking and TCP stream reassembly, currently still run on the CPU, negatively offsetting any performance gains by the offloaded GPU operations. The absence of adequate OS support also increases the cost of data transfers between the host and I/O devices. For example, packets have to be transferred from the network interface to the user-space context of the application, and from there to kernel space in order to be transferred to the GPU. While programmers can explicitly optimize data movements, this increases the design complexity and code size of even simple GPU-based packet processing programs.

As a step towards tackling the above inefficiencies, we design GANDALF, a network stream processing framework tailored to modern graphics processors. GANDALF will integrate into a purely GPU-powered implementation many of the most common operations used by different types of network traffic processing applications, including the first GPU-based implementation of network flow tracking and TCP stream reassembly. By hiding complicated network processing issues while providing a rich and expressive interface that exposes only the data that matters to applications, GANDALF will allow developers to build complex GPU-based network traffic processing applications in a flexible and efficient way.

2 Motivation

2.1 Eliminating Redundant Data Transfers

A major drawback of heterogeneous computing is the number of data transfers required between the host and I/O devices. The problem of data transfers between the CPU and the GPU is well-known in the GPGPU community, and several mechanisms have been introduced to hide the overhead of excessive data transfers. For example, page-locked memory buffers and asynchronous transfer mechanisms using DMA can decrease transfer latencies by overlapping communication with computation. However, applications may need to copy data from one page-locked memory area to another, which is actually the case in GPU-accelerated network traffic processing applications, which have to move packets from the NIC's memory to the GPU's memory. Addressing such redundant cross-device communication requires OS-level support and appropriate programming interfaces.

2.2 The Need for Modularity

Applications such as intrusion detection systems, VPN proxies, and redundancy elimination systems exhibit a similar structure and are built around a common set of domain-specific idioms and components. However, the majority of GPU-assisted implementations of such applications follow a monolithic design, lacking both modularity and flexibility. As a result, building, maintaining, and extending such systems eventually becomes a real burden.

The rise of general-purpose computing on GPUs (GPGPU) and related frameworks, such as CUDA [7] and OpenCL [8], has made the implementation of GPU-accelerated applications easier than ever. Still, the absence of libraries for network processing operations—even for simple tasks like packet decoding or filtering—increases development costs even further.

GANDALF will integrate a broad range of operations that different types of network applications rely on, with all the advantages of a GPU-powered implementation, into a single application development platform. This will allow developers to focus on core application logic, alleviating the low-level technical challenges of data transfer to and from the GPU, batching, asynchronous execution, synchronization issues, connection state management, and so on.

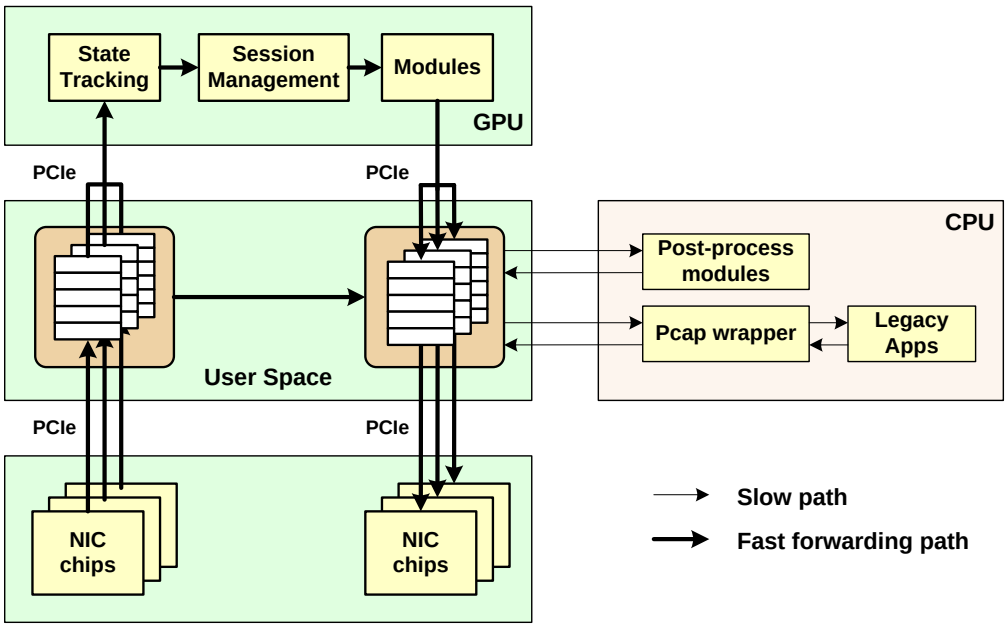

Figure 1: GANDALF architecture.

3 Design

The high-level design of GANDALF is shown in Figure 1. Packets will be transferred from the network interfaces to the memory space of the GPU in batches. The captured packets will then be classified according to their protocol and will be processed in parallel by the GPU. For stateful protocols, connection state management and TCP stream reconstruction will be supported for delivering a consistent application-layer byte stream.

GANDALF applications will consist of *modules* that control all aspects of the traffic processing flow. Modules will be represented as GPU device functions, which take as input a network packet or stream chunk. Internally, each module will be executed in parallel on a batch of packets or data chunks. After processing is completed, the packets will be transferred back to the memory space of the host, and depending on the application, to the appropriate output network interface. If needed, selected packets will be passed to the host CPU for further post-processing (i.e. fuctionality that is hard to fit well on the GPU or cannot benefit from GPU parallelization).

3.1 Processing Modules

A central concept of NVIDIA's CUDA [7] that has influenced the design of GANDALF is the organization of GPU programs into *kernels*, which in essence are functions that are executed by groups of threads. GANDALF will allow users to specify processing tasks on the incoming traffic by writing GANDALF *modules*, applicable on different protocol layers, which will then be mapped into GPU kernel functions. Modules will be implemented according to the following prototypes:

```
__device__ uint processEth(unsigned pktid,
    ethhdr *eth, uint cxtkey);
__device__ uint processIP(unsigned pktid,
    ethhdr *eth, iphdr *ip, uint cxtkey);
__device__ uint processUDP(unsigned pktid,
    ethhdr *eth, iphdr *ip, udphdr *udp, uint cxtkey);
__device__ uint processTCP(unsigned pktid,
    ethhdr *eth, iphdr *ip, tcphdr *tcp, uint cxtkey);
__device__ uint processStream(unsigned pktid,
    ethhdr *eth, iphdr *ip, tcphdr *tcp, uchar *chunk,
    unsigned chunklen, uint cxtkey);
```

The framework will be responsible for decoding incoming packets and executing all registered `process*()` modules by passing the appropriate parameters. Packet decoding and stream reassembly will be performed by the underlying system, eliminating any extra effort from the side of the developer. Each module will be executed at the corresponding layer, with pointer arguments to the encapsulated protocol headers. Arguments also include a unique identifier for each packet and a user-defined key that denotes the packet's class. Initially, GANDALF will support the most common network protocols, such as Ethernet, IP, TCP and UDP. Other protocols will easily be handled by explicitly parsing raw packets. Furthermore, packets may be marked for dropping, transfer to legacy applications, or dumping on disk. The callback functions will be registered at the initialization phase of the application. Modules will be executed per-packet in a data-parallel fashion. If more than one modules have been registered, they will be executed back-to-back in a packet processing pipeline, resulting in GPU module chains, as shown in Figure 2.

The `processStream()` modules will be executed whenever a new normalized TCP chunk of data is available. These modules will be responsible for keeping internally the state between consecutive chunks—or, alternatively, for storing chunks in global memory for future use—and continuing the processing from the last state of the previous chunk. For example, a pattern matching application will match the contents of the

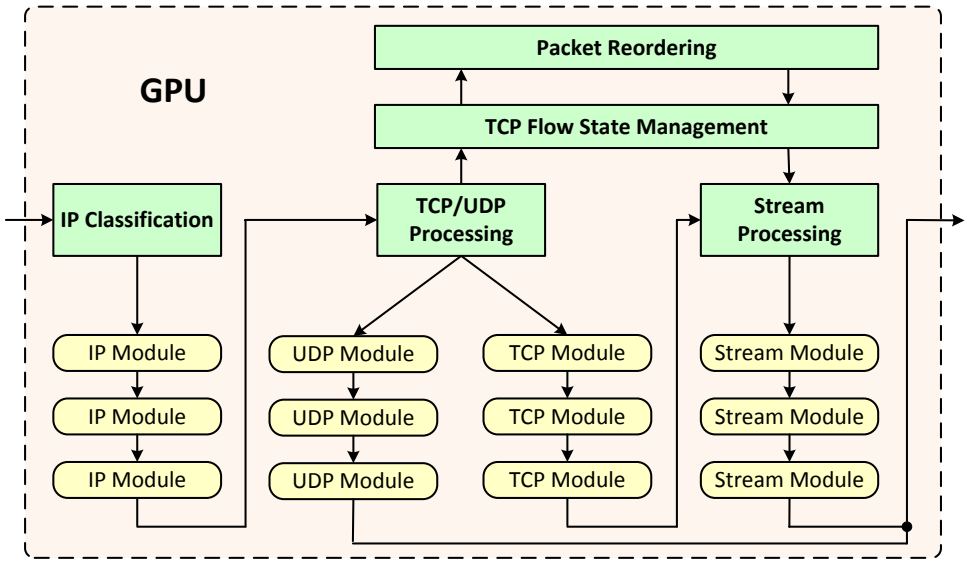

Figure 2: GPU packet processing pipeline. The pipeline is executed by a different thread for every incoming packet.

current chunk and keep the state of its matching algorithm to a global variable; on the arrival of the next chunk, the matching process will continue from the previously stored state.

As modules are simple to write, we expect that users will easily write new ones as needed using the function prototypes described above. In fact, we expect that the complete implementation of a module that simply passes packets from an input to an output interface will take only a few lines of code. More complex network applications, such as NIDS, L7 traffic classification, and packet encryption, may require a few dozen lines of code.

3.2 API

To cover the needs of a broad range of network traffic processing applications, GANDALF will offer a rich GPU API with data structures and algorithms for processing network packets.

The GANDALF API will be splitted in two parts: (i) the Host API, that will be used to initialize and configure GANDALF, and (ii) the GPU API, which will provide a rich set of building blocks for processing network streaming data, discarding them, or passing them to the OS network stack for further processing. To support the easy control of network applications, GANDALF will provide the functions shown in Table 1. These

Function	Description
`gandalf_new`	Create a new GANDALF instance
`gandalf_set_io`	Set a specific in/out interface binding
`gandalf_add_module`	Add a callback module
`gandalf_remove_module`	Remove a module
`gandalf_run`	Run a GANDALF instance
`gandalf_terminate`	Terminate a GANDALF instance

Table 1: GANDALF Host API

functions will allow the creation of GANDALF instances, the configuration of input and output network ports, and the registration of user-defined processing modules.

3.2.1 Shared Hash Table

GANDALF will enable applications to access the processed data through a global hash table. Data stored in an instance of the hash table will be persistent across GPU kernel invocations, and will be shared between the host and the device. Our hash table implementation will allow insertion, acquisition, deletion, and updates of data objects from both the host and the device, allowing applications to easily access any data that needs to be processed further or stored on disk. Internally, data objects will be hashed using a universal hashing algorithm [9], and then mapped to a given bucket. Universal hashing provides strong uniformity, however collisions may occur (i.e., more than one key maps to the same bucket). The simplest way to store all of the values that map to a given bucket is simply to maintain a list of values in the bucket. Therefore, in order to insert a new data object, we will compute the hash function on the input key to determine the new entry's bucket. If the bucket is already occupied, (e.g., another data object is already stored), the thread will allocate a new bucket node, using the on-GPU `malloc()`, and will insert the entry at the front of the bucket's list. To enable GPU threads to add or remove nodes from the table in parallel, our hash table will ensure that only one thread can safely make modifications to a bucket at a time. For this reason, we an atomic lock will be associated with each bucket, so that only a single thread may make changes to a given bucket at a time.

3.2.2 Pattern Matching

Pattern matching is one of the most common operations in network traffic processing applications. Our framework will provide a GPU-based

API for matching fixed strings and regular expressions. We will port a variant of the Aho-Corasick algorithm for string searching, and will use a DFA-based implementation for regular expression matching. Both implementations have linear complexity over the input data, independent of the number of patterns to be searched. i.e., a sequence of n bytes can be processed using $O(n)$ operations. Each GPU thread will scan a different data chunk independently, using either the same or a different automaton. Each thread will maintain its own state, eliminating any need for communication between them. To utilize efficiently the GPU memory subsystem, packet payloads will be accessed 16-bytes at a time, using an `int4` variable [10].

3.2.3 Cipher Operations

GANDALF will provide AES (128-bit to 512-bit key sizes) and RSA (1024-bit and 2048-bit key sizes) functions for encryption and decryption, and will support all modes of AES (ECB, CTR, CFB and OFB). Again, packet contents will be read and written 16-bytes at a time, as this substantially improves GPU performance. The encryption and decryption process will happen in-place and as packet lengths may be modified, the checksums for IP and TCP/UDP packets will be recomputed to be consistent. In cases where the NIC controller supports checksum computation offloading, GANDALF will simply forward the altered packets to the NIC.

3.2.4 Network Packet Manipulation Functions

GANDALF will provide special functions for dropping network packets (`Drop()`), ignoring any subsequent registered user-defined modules (`Ignore()`), passing packets to the host for further processing (`ToLinux()`), or writing their contents to a dump file (`ToDump()`). Each function will update accordingly the packet index array, which holds the offsets where each packet is stored in the packet buffer, and a separate "metadata" array.

4 Conclusion

In this report, the requirements and design of the GANDALF framework have been described. We discussed the interface that will be used by application developers in order to build their applications upon. Finally, we sketched a rich GPU API with data structures and algorithms for data processing, in order to cover the needs of a broad range of data stream processing applications.

References

[1] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting Parallelism to Scale Software Routers," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, 2009.

[2] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated Software Router," in *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, August 2010.

[3] K. Jang, S. Han, S. Han, K. Park, and S. Moon, "SSLShader: Cheap SSL Acceleration with Commodity Processors," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, March 2011.

[4] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture," in *Proceedings of the 18th ACM conference on Computer and Communications Security*, 2011.

[5] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, "Kargus: a highly-scalable software-based intrusion detection system," in *Proceedings of the 2012 ACM conference on Computer and Communications Security*, 2012.

[6] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, 2008.

[7] "CUDA Programming Guide," http://http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[8] "The OpenCL Framework," http://www.khronos.org/opencl/.

[9] L. Carter and M. N. Wegman, "Universal classes of hash functions," in *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, ser. STOC, 1977.

[10] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "Parallelization and characterization of pattern matching using GPUs," in *Proceedings*

of the 2011 IEEE International Symposium on Workload Characterization, 2011.