G-NetMon: A GPU-accelerated Network Performance Monitoring System

Wenji Wu, Phil DeMar, Don Holmgren, Amitoj Singh

Computing Division, Fermilab
Batavia, IL 60510, USA
E-mail: {wenji, demar, djholm, amitoj}@fnal.gov

Abstract—**At Fermilab, we have prototyped a GPU-accelerated network performance monitoring system, called G-NetMon, to support large-scale scientific collaborations. In this work, we explore new opportunities in network traffic monitoring and analysis with GPUs. Our system exploits the data parallelism that exists within network flow data to provide fast analysis of bulk data movement between Fermilab and collaboration sites. Experiments demonstrate that our G-NetMon can rapidly detect sub-optimal bulk data movements.**

Keywords: GPU, Flow Analysis, Network Performance Monitoring, High-speed networks.

I.   FERMILAB G-NETMON SYSTEM DESIGN

Fermilab is the Tier-1 Center for the Large Hadron Collider's (LHC) Compact Muon Solenoid (CMS) experiment, as well as the central data center for several other large-scale research collaborations. Scientific data (e.g., CMS) dominates off-site traffic volumes in both inbound and outbound directions. At Fermilab, we have prototyped a GPU-accelerated network performance monitoring system (G-NetMon) for our large-scale scientific collaborations. In this work, we explore new opportunities in network traffic monitoring and analysis with GPUs. G-NetMon exploits the inherent data parallelism that exists within network flow data and uses a GPU to rapidly calculate transfer rates between Fermilab and collaboration sites in near real time. G-NetMon aims to quickly detect sub-optimal data movement for large-scale scientific collaborations. A sub-optimal bulk data movement is detected if the associated transfer rates fall below some standard that is either predefined or provided by other network services.

Our GPU-accelerated network performance monitoring system is deployed as shown in Figure 1. It receives flow data from site border routers as well as internal LAN routers. The routers export NetFlow V5 records. The flow data is complete, not sampled.

A.   System Hardware Configuraton

Our flow-based analysis requires traffic scrutiny on a per-flow-record basis. Fermilab is the US-CMS Tier-1 Center and the main data center for a few other large-scale research collaborations. Every hour, millions of flow records are generated at Fermilab border routers. Considering the increasing volume of scientific data created every year, coupled with the evolution towards to 100 GigE network technologies, it is anticipated that our network flow data analysis requirements will be increasing accordingly. Therefore, our G-NetMon not only needs to handle current network conditions, but have the capability to accommodate the large growth of traffic expected in the near future. For now, Fermilab border routers generate less than 5,000,000 flow records every hour. Our target is to allow G-NetMon to handle 50,000,000 flow records per hour.
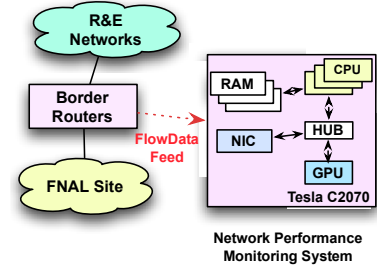

Figure 1.   G-NetMon – Deployment

G-NetMon is implemented in a system that consists of two 8-Core 2.4 GHz AMD Opteron 6136 processors, two 1Gbps Ethernet interfaces, 32 GB of system memory, and one Tesla C2070 GPU. The Tesla C2070 GPU features the Fermi GPU architecture [1].

B.   System Architecture

The G-NetMon architecture is as shown in Figure 2. The system consists of a few parts that are executed on either the host (CPU) or GPU. Based on the CUDA design principle [2], the parts that exhibit little or no data parallelism are implemented as sequential CPU threads; the parts that exhibit a rich amount of data parallelism are implemented as GPU kernels.
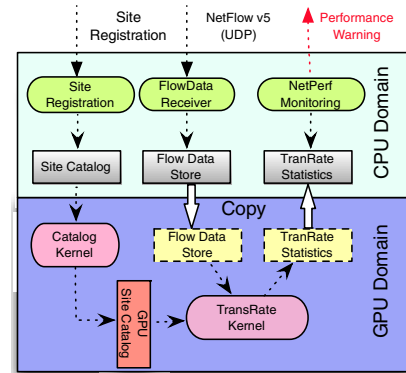

Figure 2.   A G-NetMon – Architecture

B.1 CPU Domain

Three CPU threads are implemented in the CPU domain.

Site Registration Thread: it registers scientific subnets to our network performance monitoring system. The registered

subnets are stored in the Site Catalog (a data buffer in host memory), which helps to identify scientific data transfer between Fermilab and collaboration sites. Large-scale research efforts like LHC CMS are built upon large, globally distributed collaborations. However, available computing and networking resources at different collaboration sites varies greatly. It is difficult to design machine-learning algorithms to automatically identify scientific data transfers in terms of traffic patterns or characteristics. Nevertheless, for a large-scale scientific application, the collaboration relationships between research institutions tend to be relatively static. In addition, the systems and networks assigned to a scientific application at a site are relatively fixed. Large-scale scientific data movement usually occurs between some specific subnets at each site. Therefore, by registering these subnets to our system, we can easily monitor data transfers between Fermilab and its collaboration sites through flow analysis of traffic between those subnets.

FlowData Receiver Thread: a UDP daemon, which receives NetFlow V5 packets from border routers. The received flow records are stored in Flow Data Store (a data buffer in host memory). In the current implementation, Flow Data Store is designed to hold 50,000,000 flow records. Since a NetFlow V5 flow record is less than 50 Bytes, these 50,000,000 flow records require approximately 2.5GB of memory. Processed flow records in Flow Data Store are periodically cleaned and stored to disk to create space for subsequent network flow data.

NetPerf Monitoring Thread: the main thread of our network performance monitoring system. Periodically (each hour), it copies Site Catalog and Flow Data Store to GPU memory and launches the corresponding GPU kernels to calculate the transfer rates between Fermilab and its collaboration sites. When GPU computation is completed, the NetPerf Monitoring Thread will synthesize the final results. A sub-optimal bulk data movement is detected if the associated transfer rates are below some predefined standard. Considering that TCP traffic is elastic, we use the statistics of transfer rate medians as our evaluation criteria. For a given site, network performance warnings would be issued if the associated median were less than 1Mbps for two consecutive hours.

B.2 GPU Domain

1) GPU Kernels

In the GPU domain, we have implemented two GPU kernels, Catalog Kernel and TransRate Kernel.

Catalog Kernel: it builds GPU Site Catalog, a hash table for registered scientific subnets in GPU memory, from Site Catalog. TransRate Kernel makes use of GPU Site Catalog to rapidly assign flow records to their respective subnets by examining their source or destination IP addresses. To make the hash table easy to implement and fast to search, all registered networks are transformed into /24 subnets and then entered in GPU Site Catalog. For the sake of scientific data transfer, a /24 subnet is large enough for most collaboration sites. Any network larger than /24 is divided into multiple entries in the hash table. Since GPU Site Catalog is mainly used for lookup operations and is rarely updated, there is no need to implement locks to protect unsynchronized write accesses. If any update is necessary, the table is rebuilt from scratch.

TransRate Kernel: it calculates the transfer rates between Fermilab and its collaboration sites. TransRate Kernel exploits the inherent data parallelism that exists within network flow data. When GPU instantiates TransRate Kernel on a grid of parallel threads, each thread handles a separate flow record. On a C2070 GPU, thousands of flow records can be processed simultaneously. To handle a flow record, a TransRate thread first attempts to assign the flow record to its respective site and then calculates the corresponding transfer rates. With a hash of the /24 subnet of the flow record's source or destination IP address, TransRate Kernel looks up the site to which the flow record belongs in GPU Site Catalog. Because each flow record includes data such as the number of packets and bytes in the flow and the timestamps of the first and last packet, calculation of transfer rate is simple. However, two additional factors must be considered. First, because a TCP connection is bidirectional, it will generate two flow records, one in each direction. In practice, a bulk data movement is usually unidirectional. Only the flow records in the forward direction reflect the true data transfer activities. The flow records in the other direction simply record the pure ACKs of the reverse path and should be excluded from transfer rate calculations. These flow records can be easily filtered out by calculating their average packet size, which is usually small. Second, a bulk data movement usually involves frequent administrative message exchanges between the two endpoints. A significant number of flow records are generated due to these activities. These records usually contain a small number of packets with short durations; their calculated transfer rates are generally of low accuracy and high variability. These flow records are also excluded from our transfer rate calculation.

We calculate transfer rates (maximum, minimum, average, median) for each registered site and for each host in a registered site. To calculate the median statistics, we create an array of buckets for each host to count transfer rate frequencies. Each bucket represents a 10kbps interval. To save space, all transfer rates greater than 100Mpbs are counted in the last bucket. Therefore, for each host, we maintain a bucket array of size 10001. A bucket n represents the frequency of flow rates that fall within the interval [n*10kbps (n+1)*10kbps]. From the resulting bucket counts we determine the host and site medians. We use atomic CUDA operations to calculate and store all transfer rates in order to prevent unsynchronized data accesses by the threads.

2). GPU Kernel Optimization

The Catalog Kernel is relatively simple, with few opportunities for optimization. In fact, its functionality could be included in TransRate Kernel. However, because the overhead to launch a kernel is negligible [3], we have chosen to implement it as an independent kernel to preserve a modular design.

Our TransRate kernel is optimized using various approaches:

- Register Spilling Optimization. Without this optimization, a TransRate thread will use 47 registers. These registers hold compiler-generated variables. Because registers are in-chip memories that can be accessed rapidly, a single thread's performance increases if registers are readily available. However, when we used the CUDA Occupancy

Calculator [1] to measure SM occupancy with varying block sizes, to our surprise, the occupancy rates were unacceptably low (Table 1). At such a low SM occupancy, the overall GPU performance would be greatly degraded. The improvement in each single thread cannot make up for the loss in overall thread parallelism. To raise GPU occupancy, we limit the maximum number of registers used by TransRate to 20 by compiling this kernel with the "-maxrregcount 20" option. As shown in Table 1, this register spilling optimization is effective, and the best GPU occupancy achieved as the number of threads per block is varied is now 100%.

- Shared memory. Shared memories are on-chip memories and can be accessed at very high speed in a highly parallel manner. The TransRate kernel makes use of shared memory as much as possible to accelerate flow data processing.

- Non-caching Load. Fermi architecture global memory has two types of loads, caching and non-caching. The caching load is the default mode. It first attempts to load from L1 cache, then from L2 cache, and finally from the global memory. The load granularity is 128 bytes. The non-caching load first attempts to hit in L2, and then the global memory. Its load granularity is 32 bytes. Our experiments show that non-caching load can boost the performance by at least 10%, and so the optimized TransRate kernel uses non-caching load to access Flow Data Store.

TABLE I.   SM Occupancy Rates at Different Kernel Block Sizes

Thread Size per Block	64	128	256	512
SM Occupancy Rates @ Register/Thread=47	33%	42%	33%	33%
SM Occupancy Rates @ Register/Thread=20	33%	67%	100%	100%

II.   Experimental Evaluation

A.   Performance Evaluation

At present, Fermilab border routers produce fewer than 5,000,000 flow records in an hour. However, our G-NetMon system is designed to handle a maximum load of 50,000,000 flow records per hour. To evaluate the capabilities and performance of our system at such a network load, we collected more than a day's flow records from the border routers and fed G-NetMon with 50,000,000 flow records. FlowData Receiver Thread receives these flow records and stores them in Flow Data Store. We also select the top 100 /24 scientific subnets that transfer to and from Fermilab in terms of traffic volume, and register them with Site Catalog.

To calculate transfer rates between Fermilab and collaboration sites, it is first necessary to assign flow records to their respective sites. G-NetMon implements a hash table to perform this function. We have also implemented a Non-Hash-Table method (i.e., sequential search) in which all of the registered scientific subnets are maintained in a sequential list. To categorize a flow record, the TransRate kernel searches the list one by one until a matching site, or none, is found. The G-NetMon variant with the Non-Hash-Table method is termed as NHT-G-NetMon.

A.1 GPU vs. CPU

In order to evaluate how GPU can accelerate network flow data processing in high-bandwidth network environments, we compare G-NetMon with its corresponding CPU implementations. We implemented two CPU variants, which are termed H-CPU and NHT-CPU, respectively. Like G-NetMon, H-CPU applies a hash table mechanism to rapidly assign flow records to their respective sites and then calculates the corresponding transfer rates. In contrast, NHT-CPU implements a similar Non-Hash-Table method (sequential search) as NHT-G-NetMon, in which all of the registered scientific subnets are maintained in a sequential list. To assign a flow record, CPU searches the list one by one until a matching site, or none, is found. We ran each of H-CPU and NHT-CPU on a single 2.4 GHz AMD Opteron 6136 core, with the same set of data as used above. We make the comparisons of G-NetMon vs. H-CPU and NHT-G-NetMon vs. NHT-CPU. The results are shown in Figure 3. It takes H-CPU 4916.67 ms to handle 50,000,000 flow records; in contrast, G-NetMon requires 900ms. For the non-hash-table variants, NHT-CPU and NHT-G-NetMon take 36336.67 ms and 1098.23 ms, respectively. The comparisons clearly show that GPU can significantly accelerate the flow data processing, by a factor of 5.38 (G-NetMon vs. H-CPU), or by a factor of 33.08 (NHT-G-NetMon vs. NHT-CPU). The reason that we present the comparison of GPU vs. CPU for the non-hash-table implementations is because many network applications feature a similar sequential search computation pattern as our non-hash-table implementations. For example, a network security application needs to examine each packet or flow with security rules one by one. The experiment results show GPU can significantly accelerate the data processing.
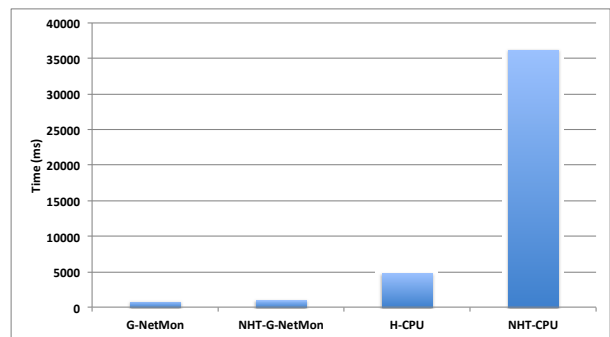
Figure 3.   G-NetMon vs. CPU

A.2 Receiving Flow Records

```
wenji@gputest2:/data/wenji/gpu-shared-memory — ssh — 215×18
Site Network: 144.16.111.0 /24; Data Transferred:   41983390959 Bytes,    28343460 Packets,   31448 Flows,   11911 Bulk Flows; Throughputs (Mbps): Aver  0.42, Max 39.46, Median  0.07,Min  0.03

        Host IP Address: 144.16.111.43, Data Transferred:    3571785056 Bytes,    2388696 Packets,    2145 Flows,    1094 Bulk Flows; Throughputs (Mbps) Aver 0.39, Max23.53, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.44, Data Transferred:    1421253338 Bytes,     957531 Packets,    1201 Flows,     537 Bulk Flows; Throughputs (Mbps) Aver 0.32, Max22.17, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.47, Data Transferred:     759688010 Bytes,     532476 Packets,    1409 Flows,     444 Bulk Flows; Throughputs (Mbps) Aver 0.20, Max 7.30, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.48, Data Transferred:    3383424495 Bytes,    2273552 Packets,    1644 Flows,     667 Bulk Flows; Throughputs (Mbps) Aver 0.61, Max38.06, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.49, Data Transferred:    2802190919 Bytes,    1897711 Packets,    1920 Flows,     583 Bulk Flows; Throughputs (Mbps) Aver 0.62, Max39.43, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.50, Data Transferred:    5378985617 Bytes,    3670408 Packets,    4712 Flows,    1463 Bulk Flows; Throughputs (Mbps) Aver 0.44, Max30.96, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.53, Data Transferred:    2807062504 Bytes,    1906604 Packets,    1818 Flows,     467 Bulk Flows; Throughputs (Mbps) Aver 0.71, Max31.25, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.54, Data Transferred:     640696974 Bytes,     455437 Packets,    1585 Flows,     506 Bulk Flows; Throughputs (Mbps) Aver 0.15, Max 0.51, Median 0.13, Min 0.03
        Host IP Address: 144.16.111.55, Data Transferred:    2072298576 Bytes,    1384526 Packets,    1425 Flows,     756 Bulk Flows; Throughputs (Mbps) Aver 0.35, Max16.00, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.56, Data Transferred:    2198479037 Bytes,    1478845 Packets,    1224 Flows,     484 Bulk Flows; Throughputs (Mbps) Aver 0.54, Max23.10, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.57, Data Transferred:    3257524266 Bytes,    2195328 Packets,    1960 Flows,     777 Bulk Flows; Throughputs (Mbps) Aver 0.49, Max26.23, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.58, Data Transferred:    3452357646 Bytes,    2305347 Packets,    2122 Flows,    1245 Bulk Flows; Throughputs (Mbps) Aver 0.32, Max24.88, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.59, Data Transferred:    4545265998 Bytes,    3050542 Packets,    2181 Flows,     948 Bulk Flows; Throughputs (Mbps) Aver 0.56, Max33.78, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.60, Data Transferred:     207812880 Bytes,     148475 Packets,    1713 Flows,     394 Bulk Flows; Throughputs (Mbps) Aver 0.05, Max 0.14, Median 0.04, Min 0.03
        Host IP Address: 144.16.111.61, Data Transferred:    3416444045 Bytes,    2298317 Packets,    2472 Flows,    1144 Bulk Flows; Throughputs (Mbps) Aver 0.35, Max39.46, Median 0.06, Min 0.03
        Host IP Address: 144.16.111.62, Data Transferred:    2062358312 Bytes,    1383854 Packets,     971 Flows,     402 Bulk Flows; Throughputs (Mbps) Aver 0.67, Max36.68, Median 0.06, Min 0.03
```
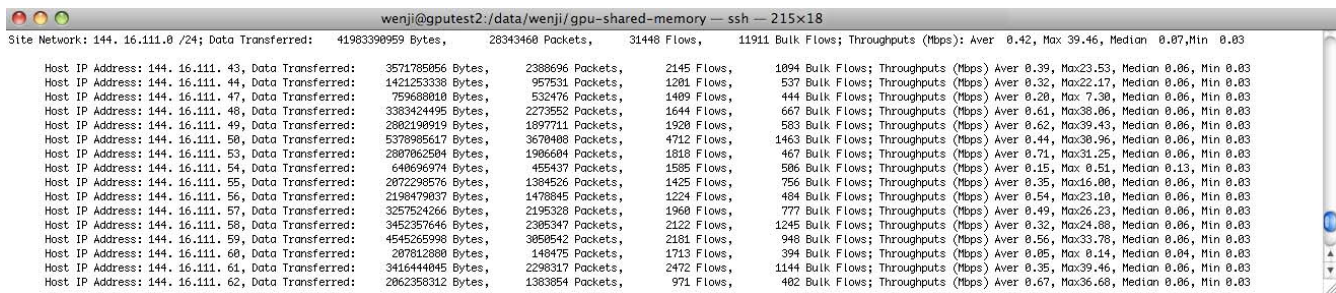
Figure 4.   Transfer rates between Fermilab and a Collaboration Site

G-NetMon receives NetFlow V5 packets from border routers via UDP. The received flow records are stored in Flow Data Store. A NetFlow V5 flow record is 48 bytes. A 1500-byte UDP packet, the largest allowed by standard Ethernet at the network, can transmit at most 30 flow records. Our G-NetMon system is designed to handle a maximum load of 50,000,000 flow records per hour. Therefore, the FlowData Receiver thread needs to handle at least 463 packets per second, which amounts to an average traffic load of 5.56Mbps. Our G-NetMon system can easily handle such a traffic load. However, because the flow records are transmitted via UDP, if CPU is busy with other tasks and the FlowData Receiver thread is not scheduled to handle the NetFlow traffic in time, the incoming packets can be dropped when the UDP receive buffer is full. We have run experiments to verify this scenario. In the experiments, the FlowData Receiver thread was assigned to share a core with a CPU-intensive application and the UDP receive buffer size was set to 4MB. We then sent it UDP traffic at varying rates, ranging from 100Mbps to 1Gbps, for 0.5 seconds. When the UDP traffic rates reached 500Mbps or above, serious packet loss would occur. We repeated the above experiments with the FlowData Receiver thread assigned a dedicated core. No packet loss was detected. Therefore, to avoid the situation of NetFlow packets being dropped, G-NetMon assigns a dedicated core for the FlowData Receiver thread to handle NetFlow traffic.

B. Network Performance Monitoring

We have registered 100 /24 scientific subnets that transfer to and from Fermilab in G-NetMon. G-NetMon monitors the bulk data movement status between Fermilab and these subnets by calculating the corresponding data transfer statistics every hour. G-NetMon calculates the transfer rates (maximum, minimum, average, median) for each registered site and for each host in a registered site. Figure 4 gives the data transfer rates in an hour between Fermilab and a collaboration site.

A sub-optimal bulk data movement is detected if the associated transfer rate falls below a predefined standard. Considering that TCP traffic is elastic and network conditions are volatile, we use the statistics of transfer rate medians as our evaluation criteria. For a given site, network performance warnings would be issued if the associated median were less than 1Mbps for two consecutive hours.

To evaluate the effectiveness of G-NetMon in detecting sub-optimal bulk data movements, we investigated the G-NetMon warnings for a period of two weeks. During this period, G-NetMon issued performance warnings for 7 sites in total (there were multiple warnings for the same sites). For those sites that G-NetMon issued warnings, we contacted their network administrators to conduct end-to-end performance analysis. Five sites responded to our requests. The end-to-end performance analysis indicated poor network conditions between these sites and Fermilab. To our surprise, one site in Greece is even connected to the outside world with a 100 Mbps link. The investigation of these warnings demonstrated that our G-NetMon can effectively detect sub-optimal bulk data movements in a timely manner. G-NetMon can detect a sub-optimal bulk data movement in two hours.

III.   CONCLUSION & DISCUSSION

At Fermilab, we have prototyped a GPU-accelerated network performance monitoring system for large-scale scientific collaborations, called G-NetMon. Our system exploits the inherent data parallelism that exists within network data flows and can rapidly analyze bulk data movements between Fermilab and its collaboration sites. Experiments demonstrate that our G-NetMon can detect sub-optimal bulk data movement in time.

The main purpose of this work is to explore new opportunities in network traffic monitoring and analysis with GPUs. The experiment results show that GPU can significantly accelerate the flow data processing, by a factor of 5.38 (G-NetMon vs. H-CPU), or by a factor of 33.08 (NHT-G-NetMon vs. NHT-CPU). At present, G-NetMon is designed to detect sub-optimal bulk data movements. In the future, we will enhance it with security features. To implement security features, G-NetMon needs to examine flow records with security rules one by one in real time or semi-real time, which require more computation capabilities. The computation pattern of examining flow records with security rules one by one is similar to that of the non-hash-table implementations discussed in the paper, in which GPU can significantly accelerate the flow data processing.

REFERENCES

[1] www.nvidia.com

[2] D. B. Kirk, Wen-mei W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach, Morgan Kaufmann Publishers Inc. ISBN: 0123814723 9780123814722

[3] S. Han, K. Jang, K. Park, S. Moon, "PacketShader, a GPU-Accelerated Software Router," In Proceedings of SIGCOMM'10, New Delhi, India.