GPUnet: networking abstractions for GPU programs

Mark Silberstein
Technion – Israel Institute of Technology

Sangman Kim, Seonggu Huh, Xinya Zhang
Yige Hu, Emmett Witchel
University of Texas at Austin

Amir Wated
Technion
What
A socket API for programs running on GPU

Why
GPU-accelerated servers are hard to build

Results
GPU vs. CPU
50% \text{↑} \text{throughput}, 60% \text{↓} \text{latency}, \frac{1}{2} \text{LOC}
Motivation: GPU-accelerated networking applications

Data processing server

MapReduce
Recent GPU-accelerated networking applications
SSLShader (Jang 2011), GPU MapReduce (Stuart 2011), Deep Neural Networks (Coates 2013), Dandelion (Rossbach 2013), Rhythm (Agrawal 2014) ...
Recent GPU-accelerated networking applications
SSLShader (Jang 2011), GPU MapReduce (Stuart 2011), Deep Neural Networks (Coates 2013), Dandelion (Rossbach 2013), Rhythm (Agrawal 2014) ...

required heroic efforts
GPU-accelerated networking apps: Recurring themes

- Pipelining and buffer management
- NIC-GPU interaction
- Request batching
GPU-accelerated networking apps: Recurring themes

We will sidestep these problems
The **real** problem: CPU is the **only** boss

- GPU
- NIC
- Storage
- CPU

Mark Silberstein - EE, Technion
Example: CPU server

NIC

CPU

Memory

recv()
compute()
send()
Inside a GPU-accelerated server

Theory

recv()
GPU_compute()
send()
Inside a GPU-accelerated server

```
recv();
CPU
Memory
recv();
GPU
Memory
recv();
batch();
```

**Theory**

```
recv()
GPU_compute()
send()
```
Inside a GPU-accelerated server

```
transfer();

CPU

Memory

GPU

Memory

recv();
batch();
optimize();
transfer();

recv();

GPU_compute()

send()

Theory
```
Inside a GPU-accelerated server

Theory

recv()
GPU_compute()
send()

recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
Inside a GPU-accelerated server

```
transfer();

recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
transfer();
cleanup();
```

**Theory**

```
recv();
GPU_compute();
```

Mark Silberstein - EE, Technion
Inside a GPU-accelerated server

Theory
recv()
GPU_compute()
send()

recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
transfer();
cleanup();
dispatch();
send();
Aggressive pipelining
Double buffering, asynchrony, multithreading

recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
transfer();
cleanup();
dispatch();
send();
recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
transfer();
cleanup();
dispatch();
send();
recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
transfer();
cleanup();
dispatch();
send();
recv();
batch();
optimize();
transfer();
balance();
GPU_compute();
transfer();
cleanup();
dispatch();
send();
This code is for a CPU to manage a GPU
GPUs are not co-processors
GPUs are peer-processors
They need I/O abstractions

File system I/O – [GPUfs ASPLOS13]
Network I/O – this work
GPUnet: socket API for GPUs

Application view

```
socket(AF_INET,SOCK_STREAM);
connect("node0:2340")
```

```
socket(AF_INET,SOCK_STREAM);
listen(:2340)
```

```
socket(AF_INET,SOCK_STREAM);
connect("node0:2340")
```

```
socket(AF_INET,SOCK_STREAM);
connect("node0:2340")
```
GPU-accelerated server with GPUnet

CPU not involved

recv()
GPU_compute()
send()
GPU-accelerated server with GPUUnet

recv()

GPU_compute()
send()
GPU-accelerated server with GPUnet

No request batching

recv() → send()

NIC

Memory

recv()

GPU_compute()

send()
GPU-accelerated server with GPUnet

Automatic request pipelining

recv()  →  send()

Automatic buffer management

recv()

GPU_compute()

send()
Building a socket abstraction for GPUs
Goals

Simplicity
Reliable streaming abstraction for GPUs

Performance
NIC → GPU
data path optimizations
Design option 1: Transport layer processing on CPU

GPU controls the flow of data
Design option 1: Transport layer processing on CPU

- NIC
- Network buffers
- CPU: Transport processing
- Memory
- GPU: recv()
- Extra CPU-GPU memory transfers
Design option 2: Transport layer processing on GPU

- CPU
- Memory
- GPU
  - `recv()`
  - Transport processing
- Network buffers
- P2P DMA
- NIC
Design option 2: Transport layer processing on GPU

CPU
recv()

GPU
Transport processing

Network buffers

TCP/IP on GPU?
P2P DMA

CPU applications access network through GPU?
Not CPU, Not GPU

We need help from NIC hardware
RDMA: offloading transport layer processing to NIC

- CPU
  - Streaming
  - Message buffers
- GPU
  - Streaming
  - Message buffers
- Reliable RDMA NIC
GPUnet layers

- GPU Socket API
- Reliable in-order streaming
- Reliable channel

RDMA Transports:
- Infiniband

Non-RDMA Transports:
- UNIX Domain Socket, TCP/IP
GPUnet layers

- GPU Socket API
- Reliable in-order streaming
- Reliable channel

RDMA Transports: Infiniband
Non-RDMA Transports: UNIX Domain Socket, TCP/IP

Simplicity
Performance
See the paper for

- Coalesced API calls
- Latency-optimized GPU-CPU flow control
- Memory management
- Bounce buffers
- Non-RDMA support
- GPU performance optimizations
Implementation

- Standard API calls, blocking/nonblocking
- **libGPUnet.a**: AF_INET, Streaming over Infiniband RDMA
  - Fully compatible with CPU **rsocket** library
- **libUNIXnet.a**: AF_LOCAL: Unix Domain Sockets support for inter GPU/CPU-GPU
Implementation

**CPU**
- Bounce buffers
- GPUnet proxy

**GPU**
- GPU application
- GPUnet socket library
- Flow control
- Network buffers

**Network**
- NIC

**Memory**
- CPU memory fallback
- GPU memory

Mark Silberstein - EE, Technion
Evaluation

- Analysis of GPU-native server design
  - Matrix product server
- In-GPU-memory MapReduce
- Face verification server

2x6 Intel E5-2620, NVIDIA Tesla K20Xm GPU, Mellanox Connect-IB HCA, Switch-X bridge
In-GPU-memory MapReduce
# In-GPU-memory MapReduce: Scalability

<table>
<thead>
<tr>
<th></th>
<th>1 GPU (no network)</th>
<th>4 GPUs (GPUnet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-means</td>
<td>5.6 sec</td>
<td>1.6 sec (3.5x)</td>
</tr>
<tr>
<td>Word-count</td>
<td>29.6 sec</td>
<td>10 sec (2.9x)</td>
</tr>
</tbody>
</table>

GPUnet enables scale-out for GPU – accelerated systems
Face verification server

CPU client (unmodified) via rsocket

GPU server (GPUnet)

memcached (unmodified) via rsocket

Infiniband

recv()

GPU_features()

query_DB()

GPU_compare()

send()
Face verification: Different implementations

Throughput (KReq/sec)

Latency (μsec)

Median

25th-75th%

99th %

1 GPU (no GPUUnet)

CPU
6 cores

1 GPU
GPUUnet

23
34
54
Face verification: Different implementations

- 1 GPU (no GPUnet)
- GPUnet
- CPU 6 cores

<table>
<thead>
<tr>
<th>Throughput (KReq/sec)</th>
<th>Median</th>
<th>25th-75th</th>
<th>99th %</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>34</td>
<td>54</td>
<td>500</td>
</tr>
</tbody>
</table>

1.9x throughput
1/3x latency
1/2 LOC
Face verification: Different implementations

Throughput (KReq/sec)

Latency (μsec)

1 GPU (no GPUnet)

CPU
6 cores

Large variability in latency

GPU GPUnet

Median
25th-75th%
99th %

Mark Silberstein - EE, Technion
Face verification on all processors
2xGPU + 10xCPU

Latency (μsec)

Throughput (KReq/sec)

Similar latency
4.5x throughput

Cpu
6 cores

1 GPU
GPUnet

164
186

2xGPUnet + 10xCPU
Set GPUs free!

GPUUnet

GPUUnet is a library providing networking abstractions for GPUs

https://github.com/ut-osa/gpunet

mark@ee.technion.ac.il