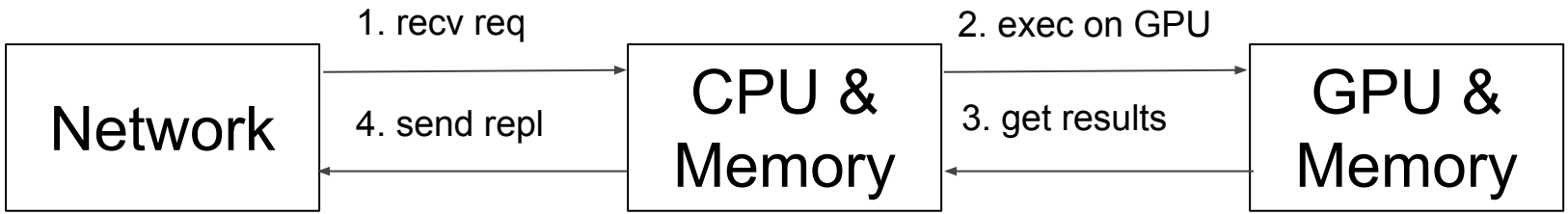
GPUnet: Networking Abstractions for GPU Programs

Author: Andrzej Jackowski

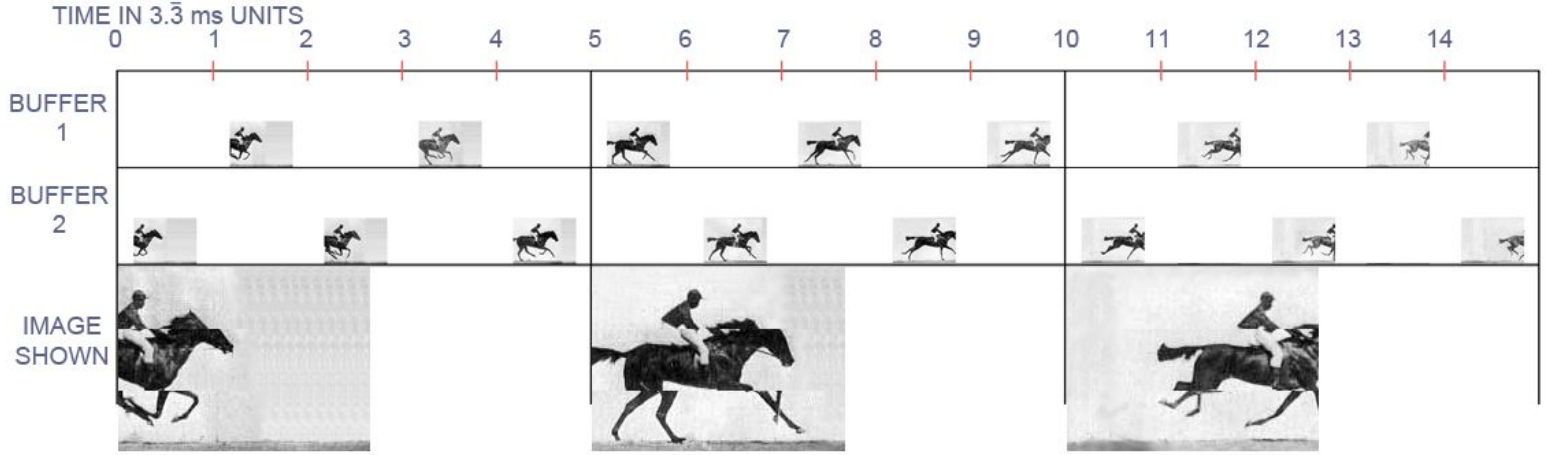
GPU programming problem

GPU distributed application flow

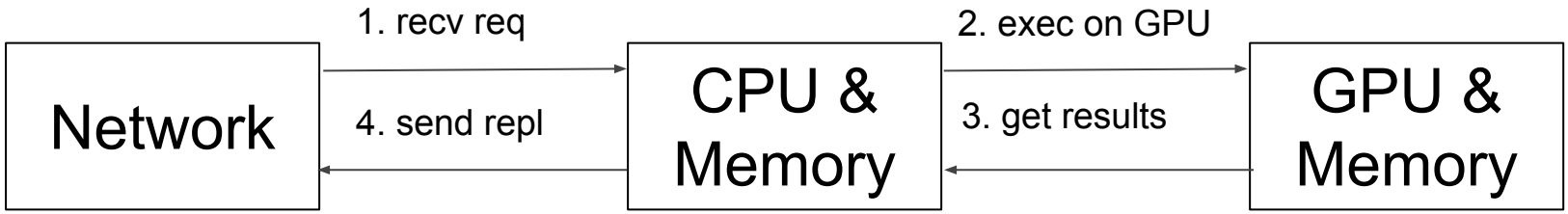


The tip of the iceberg: double buffering

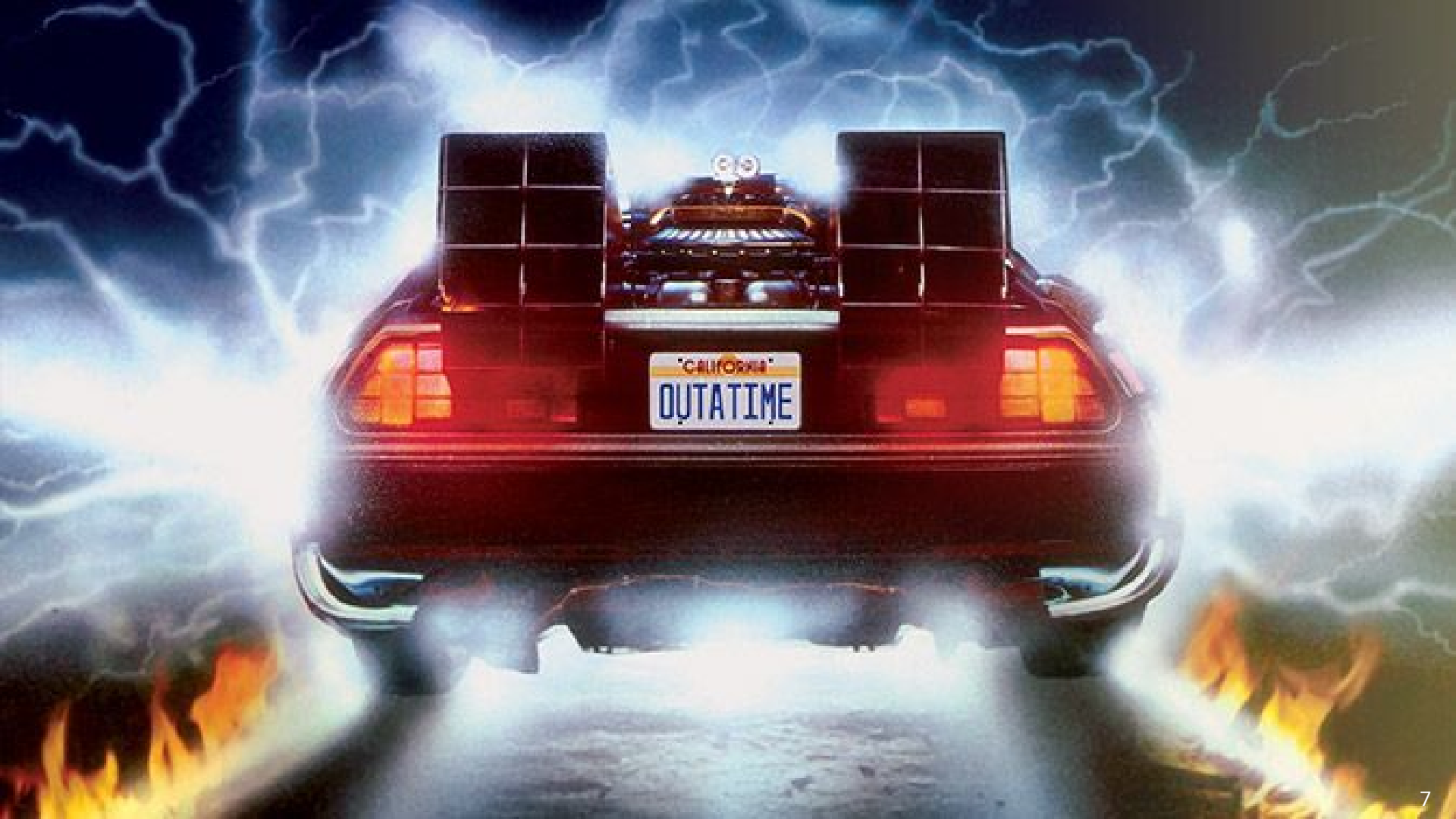
(not rocket science, still annoying)



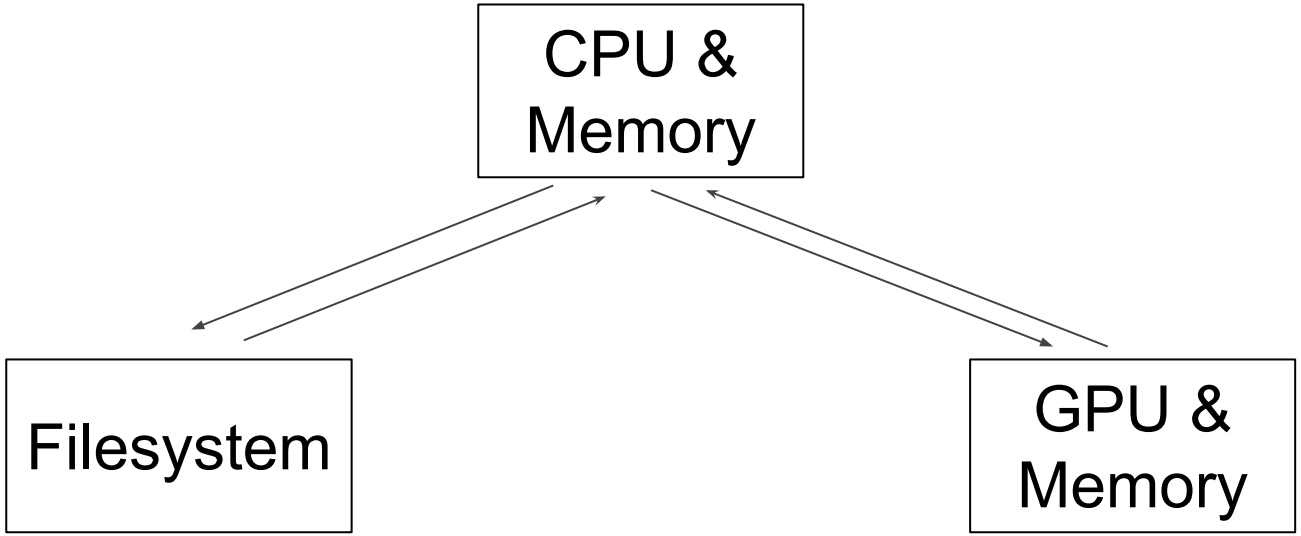
GPU distributed application flow



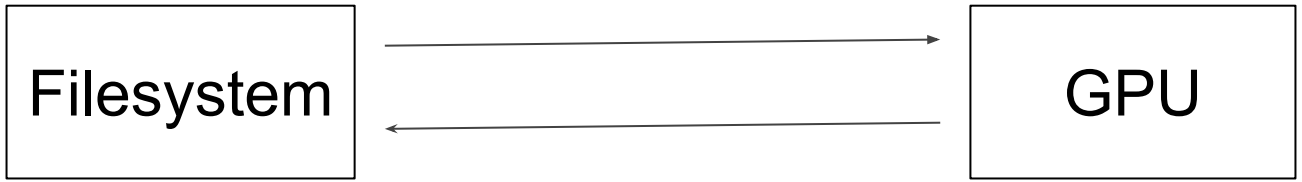
If we want to use GPU efficiently this design is more complicated - we have to batch requests, have multithreading or asynchrony etc.


CALIFORNIA
OUTATIME

GPUfs: Integrating a File System with GPUs [2013]



GPUfs: Integrating a File System with GPUs [2013]



GPUfs - main goals

- POSIX-like filesystem for GPU programs
  - Making common API calls efficient when executed in data parallel fashion
  - Consistency semantics that minimizes expensive CPU <-> GPU transfers
- Generic, software-only buffer cache with key features like read-ahead, data transfer scheduling, asynchronous writes
- Proof-of-concept running on NVIDIA Fermi cards (GF400, GF500 series etc.)
- Quantitative evaluation

GPUfs implementation

- **GPUfs API** - library provided for your GPU application, i.a. *gread/gwrite, gopen/gclose, gfstat, gfsync*
- **GPU File State** - open descriptors structure
- **GPU Buffer Cache** - one per application (instead of one per filesystem), but many kernels of one process can use the same cache!
- **CPU-GPU RPC/daemon** - layer responsible for CPU <-> GPU communication (CPU is reading files).
  There is a lot of interesting stuff about it in paper!
- GPUfs consistency module - ensures that for each file there is only one writer (it's assumption for GPUfs)

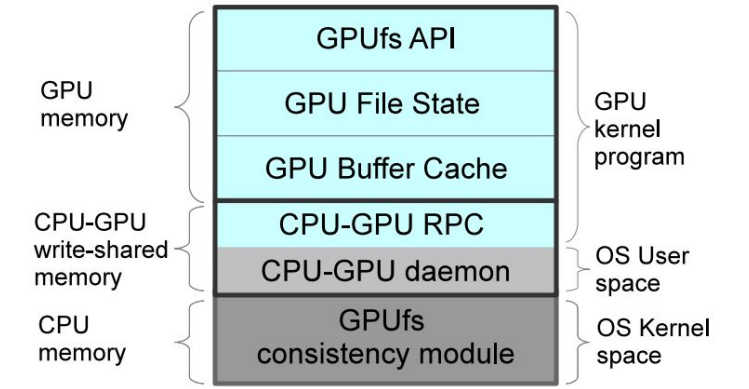


Figure 2. Main GPUfs software layers and their location in the software stack and physical memory.

Example: *gread()*

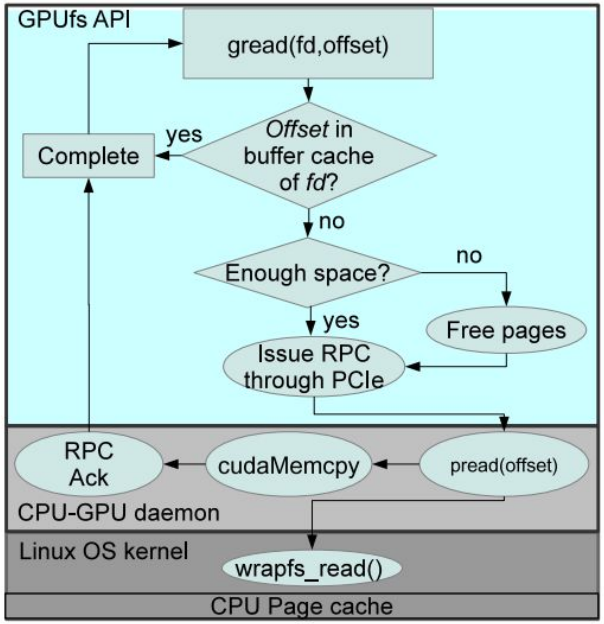


Figure 3. Functional diagram of a call to `gread`. Color scheme is the same as Figure 2.

- *gread/gwrite* are like *pread/pwrite* (offsets are given, no seek pointer)
- Calls has *per ThreadBlock* granularity, that's why you can't open file per thread, but you can open file per ThreadBlock

Performance results

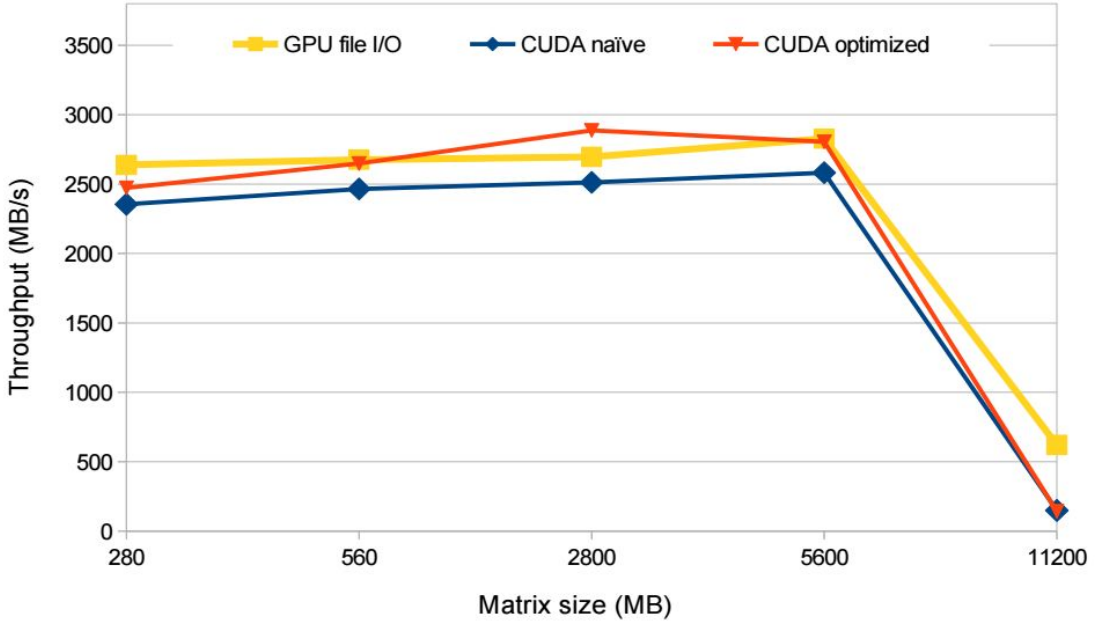


Figure 8. Matrix-vector product for large matrices

- GPUfs code can be faster than standard CUDA code
- In CUDA optimized version we read 1GB chunks into multiple buffers, which sometimes cause slowdowns due to buffer cache paging
- GPUfs reads are more granular (2MB for each page)
- Moreover GPUfs code is simpler

GPUfs code example (from GitHub)

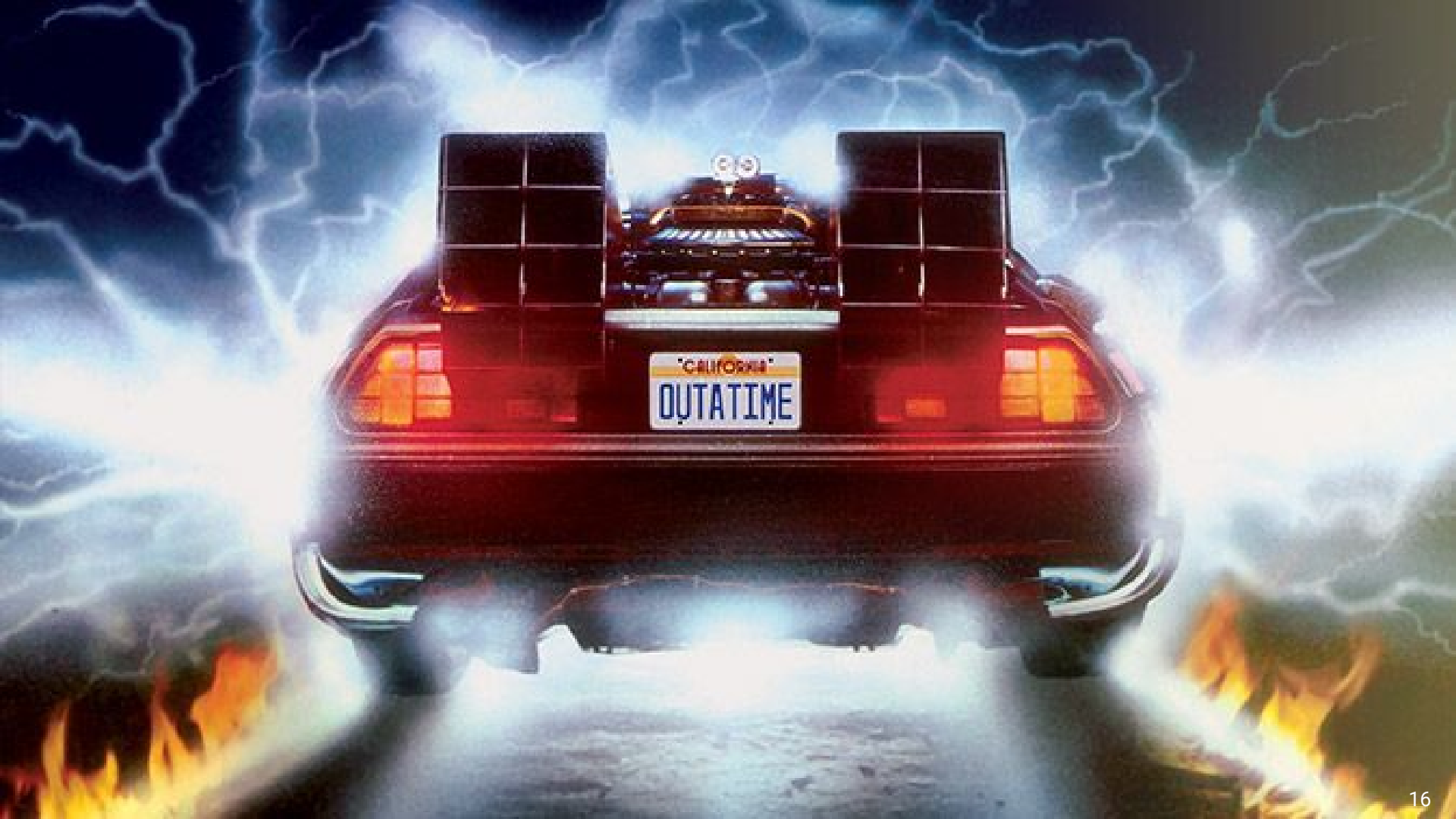
```
zfd = gopen(src, O_GRDONLY);
zfd1 = gopen(dst, O_GWRONCE);
filesize = fstat(zfd);

for (size_t me = blockIdx.x * FS_BLOCKSIZE; me < filesize; me += FS_BLOCKSIZE * gridDim.x)
{
        int toRead = min((unsigned int) FS_BLOCKSIZE, (unsigned int) (filesize - me));
        if (toRead != gread(zfd, me, toRead, scratch)) { assert(NULL); }
        if (toRead != gwrite(zfd1, me, toRead, scratch)){ assert(NULL);}
}

gclose(zfd);
gclose(zfd1);
```

Summary of GPUfs

- For now there are some hardware limitations (because GPU is designed as co-CPU)
- Code written with GPUfs can be **simple** and **efficient**
- You **can’t get rid of CPU**, but you **can get rid of CPU code** (in the GPUfs test CPU code had only 1 line - exec GPU program)


CALIFORNIA
OUTATIME

GPUnet design: server

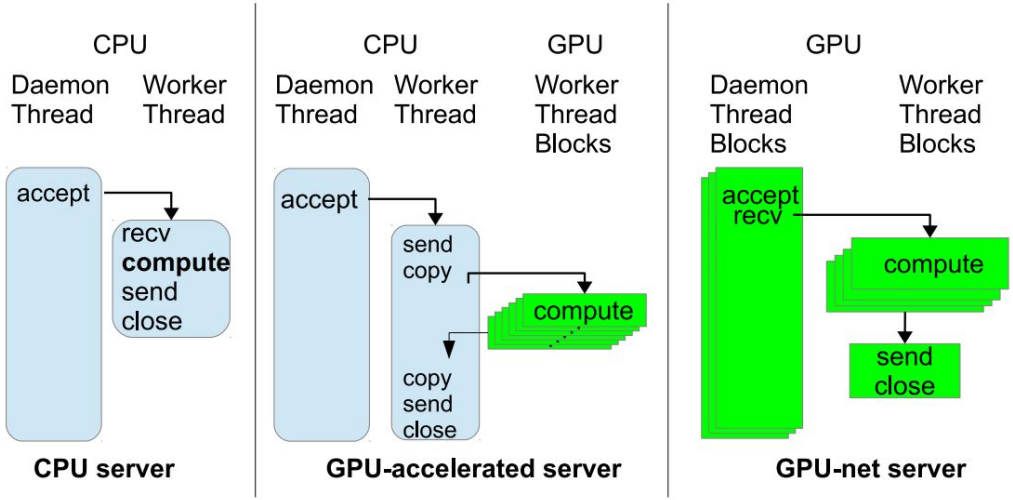


Figure 2: The architecture of a network server on a CPU, using a GPU as a co-processor, and with GPUnet (daemon architecture).

- In *GPU-accelerated* large batches are needed to amortize CPU-GPU communication
- In *GPU-net* we don't have to batch requests, so it's lightweight and simple to use
- Moreover we don't have to mix CPU and GPU code for handling a single request

GPUnet design: performance benefits

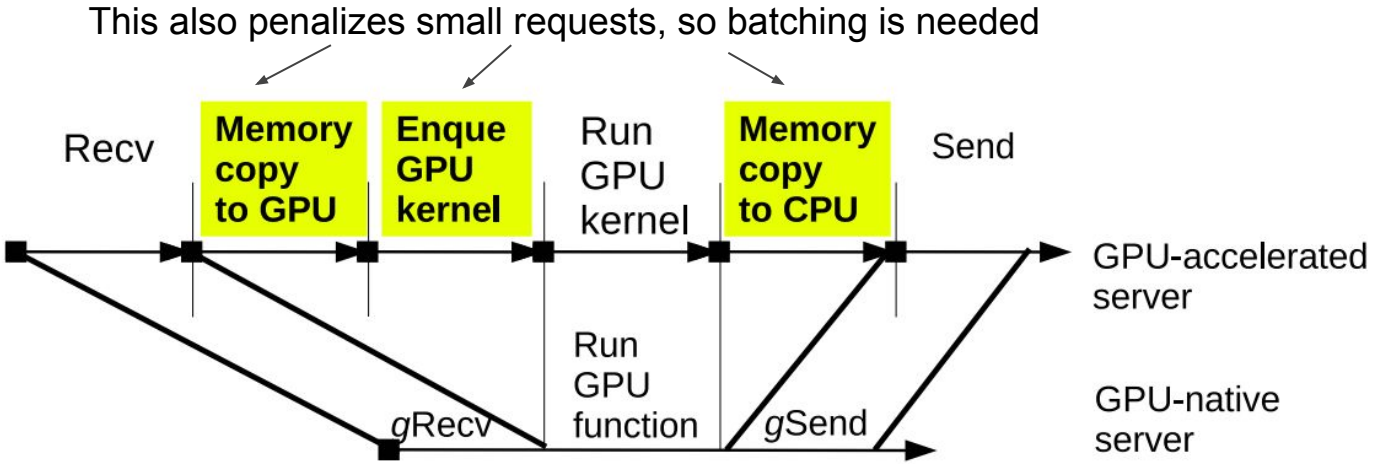


Figure 3: The logical stages for a task processed on a GPU-accelerated CPU server (top) and GPU-native network server(bottom). Highlighted stages are eliminated by the GPU networking support.

GPUnet design: library API

- Sockets are commonly used and versatile
- Functions similar to standard socket libraries: *gconnect, gbind, gsend, grcv…*
- *“[..] All threads in a threadblock must invoke the same GPUnet call together in a coalesced manner: with the same arguments, at the same point in application code [...]”*

```
__shared__ int sock;
__shared__ uchar buf[BUF_SIZE];
int ret, i;
while ((sock = gconnect_in(addr)) < 0) {};
assert(sock >= 0);
for (i = 0; i < NR_MSG; i++) {
  int sent = 0;
  do {
    ret = gsend(sock, buf + sent, BUF_SIZE - sent);
    if (ret < 0) {
      goto out;
    } else {
      sent += ret;
    }
  } while (sent < BUF_SIZE);
```

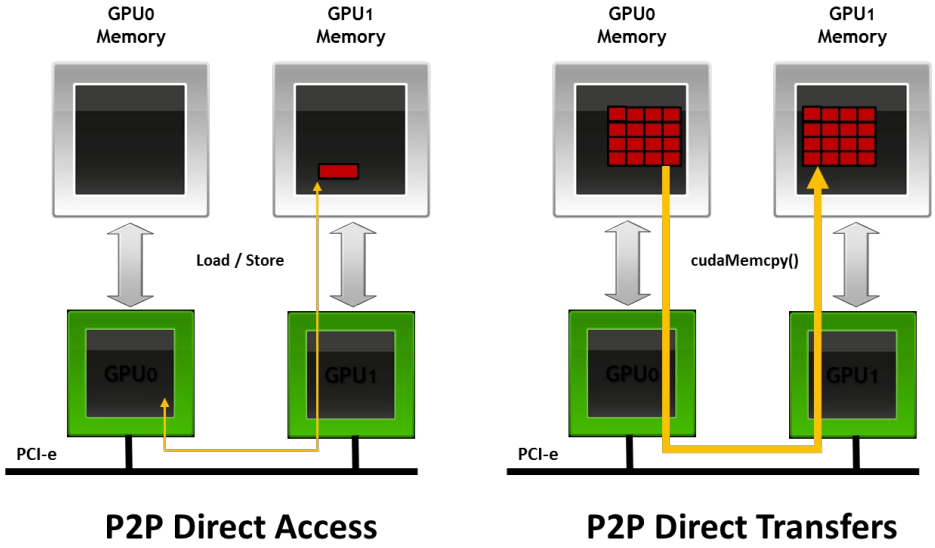
GPUnet design: library assumptions

- **Simplicity** - provide sophisticated, effective solution as easy to use socket abstraction
- **Compatibility with GPU programming** - support common GPU programming idioms
- 

Compatibility with CPU endpoints - CPU can talk with remote GPU using sockets
- **Network Interface Controller sharing** - both CPU and GPU can use NIC at the same time
- **Namespace sharing** - single network namespace (IP, ports, socket names) among CPUs and GPUs

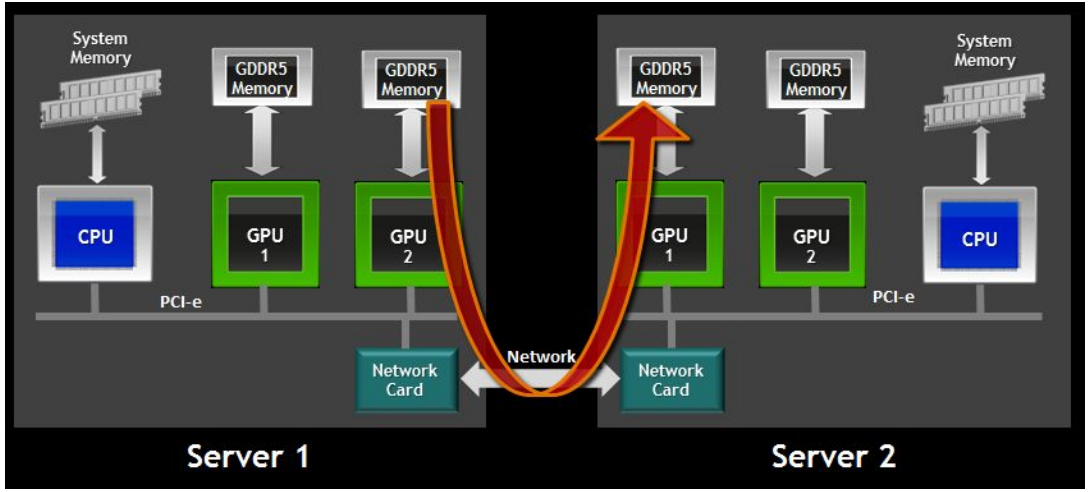
Peer-to-peer Direct Memory Access

- P2P DMA allows graphic cards to exchange data without using CPU memory
- In the same way GPU can work with other PCI-e devices e.g. network interface controller
- Improves both throughput and latency
- Moreover, relieve CPU&memory



Remote Direct Memory Access

- DMA over the network
- Low latency
- High bandwidth
- We don’t need to switch context
- CPU won’t be involved (so CPU cache is not touched)



GPUnet design: Rsockets compatibility

- Rsockets is library that provides socket-level API over RDMA
- GPUnet is extending rsockets and giving similar API for GPU code
- Rsockets and GPUnet are compatible

GPUnet design: for discrete GPU

- Discrete means not integrated (pic related)
- Outperforms hybrid (cpu-gpu) configuration for order of magnitude
- Authors believe that discrete GPUs will be popular for years [it’s 2016 they’re still right!]

Design vs Implementation

We can’t get rid of CPU again!

- Flow control mechanism - allows the sender to block if receiver buffer is full
- Cannot be implemented using only GPU
  - GPU can’t access HCA’s (host channel adapter, NIC in InfiniBand) door-bell registers in order to trigger a send operation
  - GPU can’t map completion queue - the structure that delivers completion notification e.g. when new data arrives
  - CPU is necessary to assist every GPU send and receive operation
- Implemented efficiently using GPU memory mapping into CPU’s address space

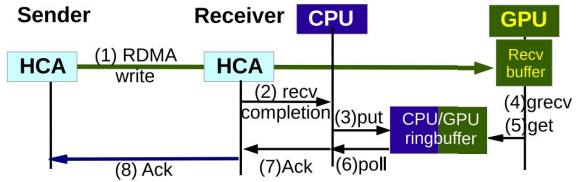


Figure 7: Ring buffer updates for GPU flow control mechanism in `grecv()` call.

Non-RDMA transport

- GPUnet is also working on hardware without RDMA, since RDMA is not yet popular
- Has higher latency and needs bigger buffer than RDMA solution

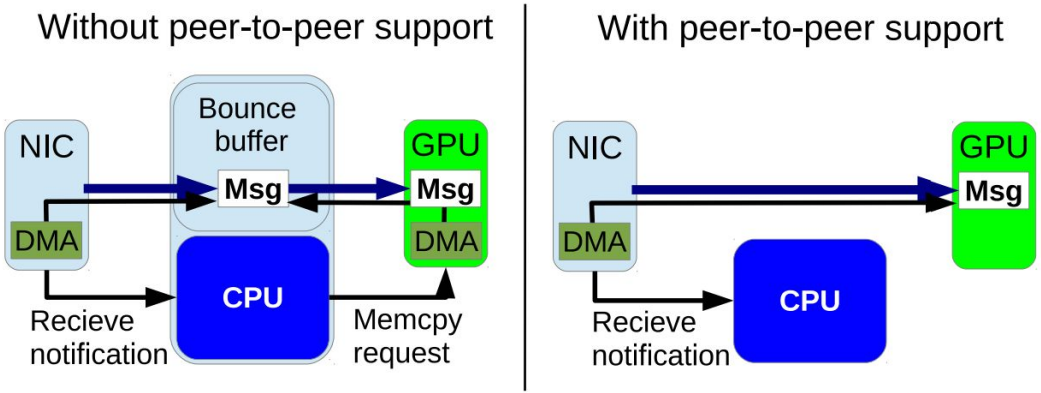
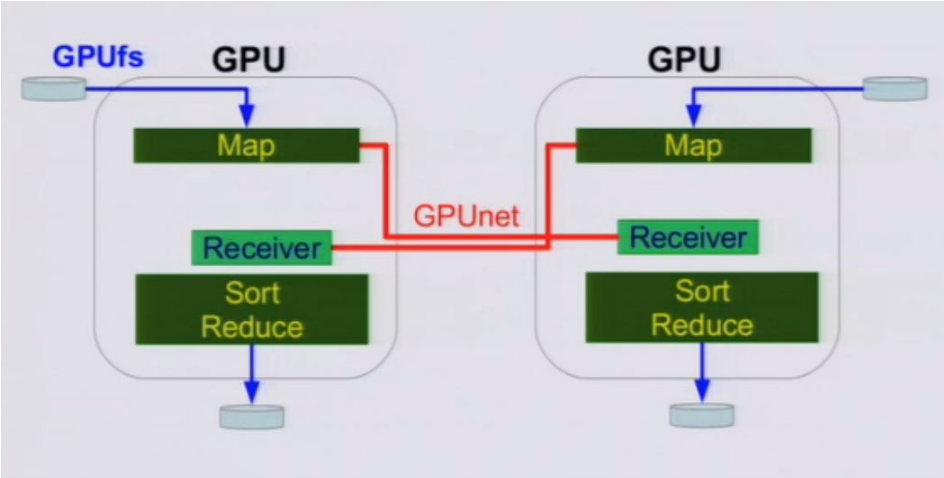


Figure 1: Receiving network messages into a GPU. Without P2P DMA, the CPU must use a GPU DMA engine to transfer data from the CPU bounce buffer.

Example: GPUnet Map Reduce

- (Almost) 0 lines of CPU code
  - GPUfs for file access
  - GPUnet for network usage



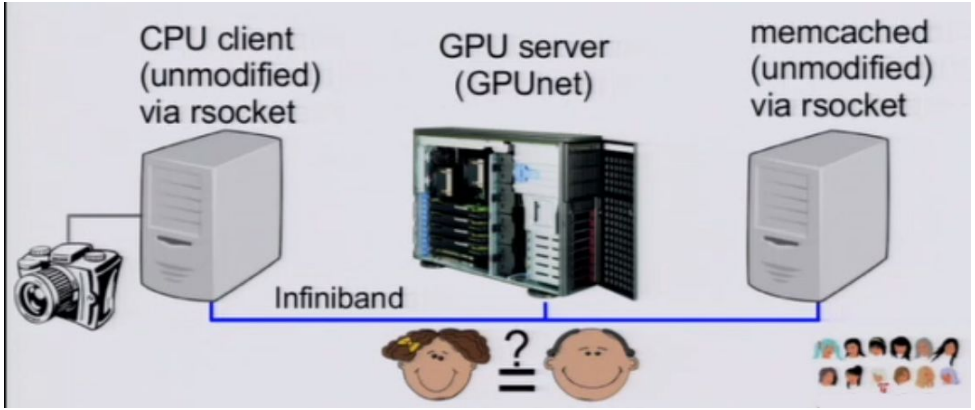
Example: GPUnet Map Reduce

- What a pity that scalability tests were so small (max 4 nodes)
- Only two machines were using GPU-NIC RDMA (two machines were using bounce buffers)

	phoenix++ 8-cores (1 node)	Hadoop 8-cores (1 node)	GPUnet 1 GPU (1 node)	GPUnet 4 GPU (4 nodes)
K-means	12.2s	71.0s	5.6s	1.6s
Word-count	6.23s	211.0s	29.6s	10.0s

Example: Face Verification Server

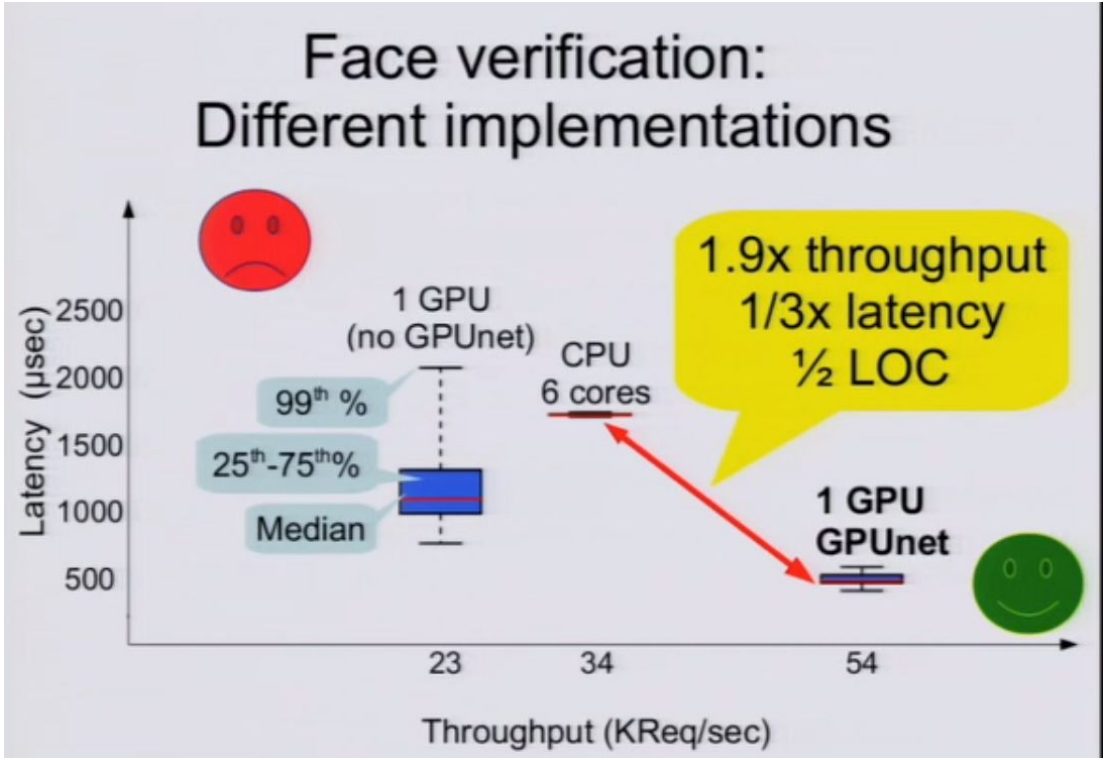
1. CPU client send image with label
2. GPU server gets image label histogram from memcached
3. GPU server creates image histogram
4. GPU server compares histograms
5. GPU server send reply with result to CPU client



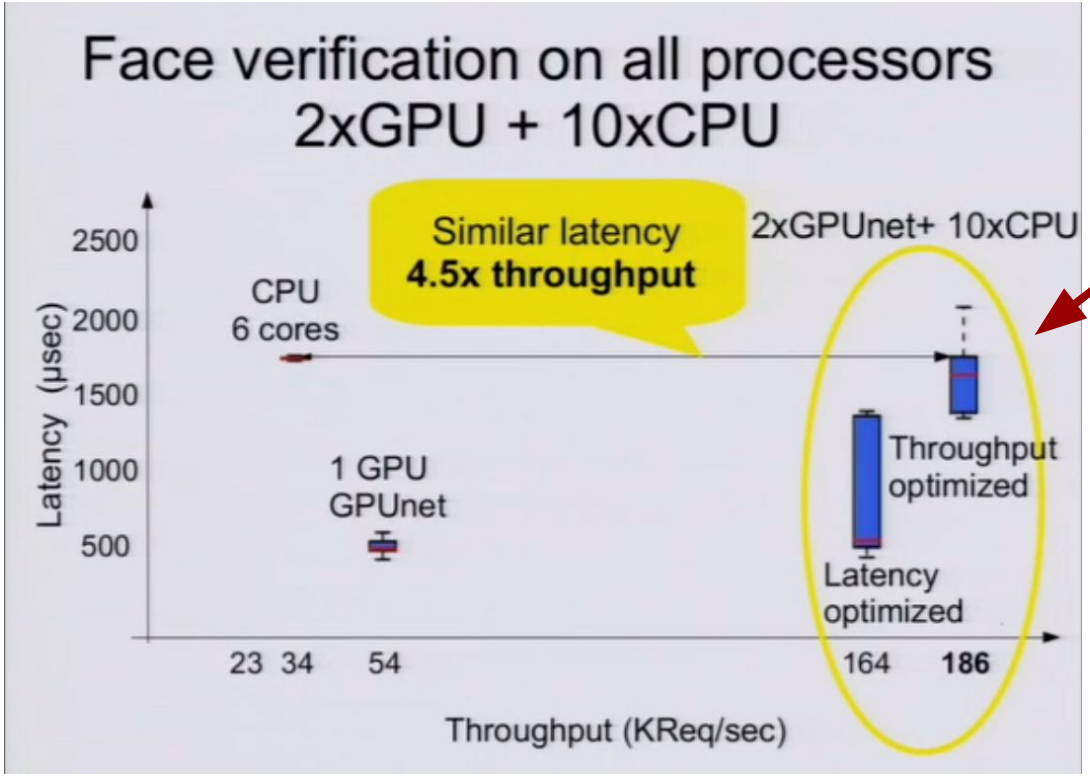
Three different GPU server implementations

- **CPU version** - running the whole *GPU server* as multi threaded CPU process *{506 lines of code}*
- **GPU (no GPUnet) version** - CPU version with algorithm executed as CUDA code *{596 lines of code}*
- **GPUnet version** - written as native GPU application with GPUnet {245 lines of code}

Performance results



Scaling results



CPU & GPU servers working together!

Server type	CPU	2× CPU	CUDA	GPUnet BB	GPUnet	2×GPUnet	2×GPUnet + CPU
Thpt (Req/s)	35K	69K	23K	17K	67K	136K	188K

Table 7: Face verification throughput for different servers.

Limitations

- GPUnet doesn’t provide mechanism for socket migration between CPU and GPU (it may be convenient to use it e.g. in load balancing)
- GPUnet relies on consistent reads to GPU memory when it’s used by both kernel and NIC RDMA. Anyway tests with CRC codes “proves” no violations, despite the lack of guarantee from NVIDIA
- GPUnet, as well as GPUfs is available on GitHub, but it has *prototype quality*

Summary

- GPUnet shows new approach for GPU programming
- It’s not the fastest possible solution, because you can always write dedicated code using P2P DMA & RDMA…
- … but it’s definitely fast and simply to use!
- There are a lot of interesting low-level investigations in the paper
- You can learn even more by studying the GPUfs/GPUnet source code

Questions, thoughts?

https://github.com/gpufs/gpufs

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