RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

GPU-centric communication for improved efficiency

Benjamin Klenk[*], Lena Oden[†], Holger Fröning[*]
[*]Heidelberg University, Germany
[†]Fraunhofer Institute for Industrial Mathematics, Germany

GPCDP Workshop @ Green Computing Conference 2014
November 3, 2014, Dallas, TX, USA

Green500 (June 2014)

Rank	GFLOPS/W	TFLOPS	System	Power (kW)	TOP500 Rank
1	4.39	151,79	TSUBAME-KFC 1U-4GPU Cluster Intel CPU & **NVIDIA K20x, IB**	34.58	437
2	3.63	191.10	Wilkes (Dell) Intel CPU & **NVIDIA K20, IB**	52.62	201
3	3.52	277.10	HA-PACS (Cray) Intel CPU & **NVIDIA K20x, IB**	78.77	165
4	3.46	253.60	Cartesius Accelerator Island Intel CPU & **NVIDIA K40m, IB**	44.40	421
5	1.75	5,587.00	Piz Daint (Cray) Intel CPU & **NVIDIA K20x, Aries**	1,753.66	6

Motivation

- There is a tradeoff between performance and **energy efficiency** (top500 != green500)
- Most energy efficient systems use **NVIDIA GPUs** as accelerators
- Today's HPC systems are **cluster systems**!
- According to Exascale Computing Study: Most energy is spent for **communication**!

- ***Top500***, *www.top500.org, June 2014*
- ***Green500***, *www.green500.org, June 2014*
- ***Exascale Computing Study**: Technology Challenges in Achieving Exascale Systems, September 28, 2008*

Motivation

- Accelerators (e.g. GPUS) are widely used, but
  - can only excel in performance when data can be held in **on-chip memory** (scarce resource)
  - are deployed in cluster systems, **requiring communication** to work on large amounts of data
  - **cannot control communication** and need return control flow to the CPU
  - are limited in performance due to **PCIe** data copies

- What we need is
  - a **direct communication path** between distributed GPUs
  - getting rid of PCIe copies and context switches

Outline

- Background
- GPU-centric communication
- Energy analysis
- Conclusion
- Future Work

Background

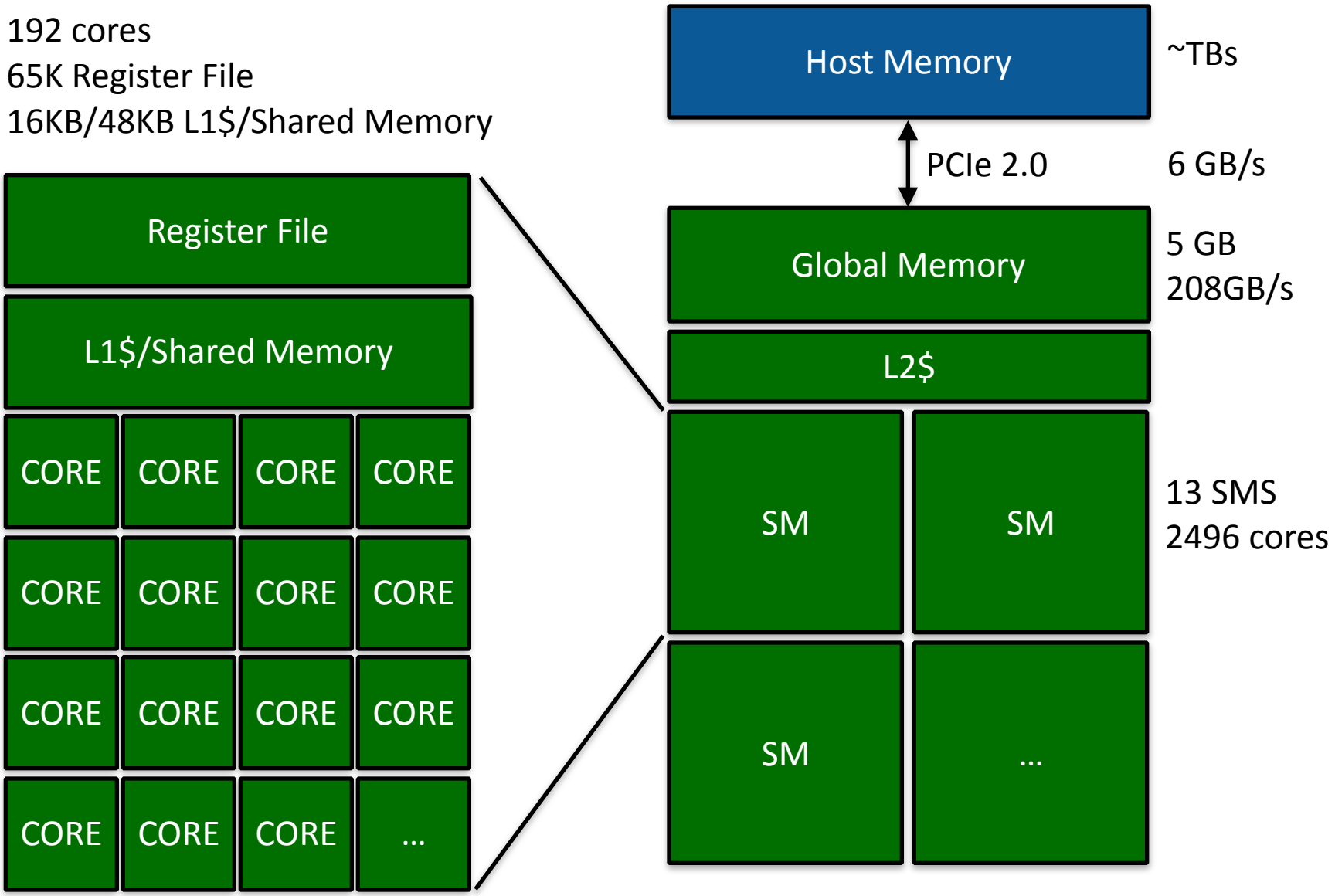
- Why GPUs?
  - Massively parallel (demanded by many applications)
  - Power efficient
    - Intel Xeon E5-2687W: 1.44 GFLOPs/W
    - **NVIDIA K20 GPU:** 14.08 GFLOPs/W
    - NVIDIA Tegra K1: 32.60 GFLOPs/W (only GPU)
- Drawbacks
  - Memory is a scarce resource
  - PCIe copies
  - Communication is still CPU controlled

GPU Computing: architecture

192 cores

K20: 65K Register File

16KB/48KB L1$/Shared Memory



GPU Computing: execution model

- CUDA (Compute Unified Device Architecture)
- Kernels define instructions that are executed by every thread (**SIMT** = Single Instruction Multiple Threads)
- Threads are organized in **blocks**, forming a grid
- Blocks are tightly bound to SMs (about 2 per SM concurrently)
- 32 Threads (**warp**) are executed in parallel
- Memory access latencies are tolerated by scheduling thousands of threads (hardware multi-threading)
- Synchronization only within blocks

How do we measure power/energy?

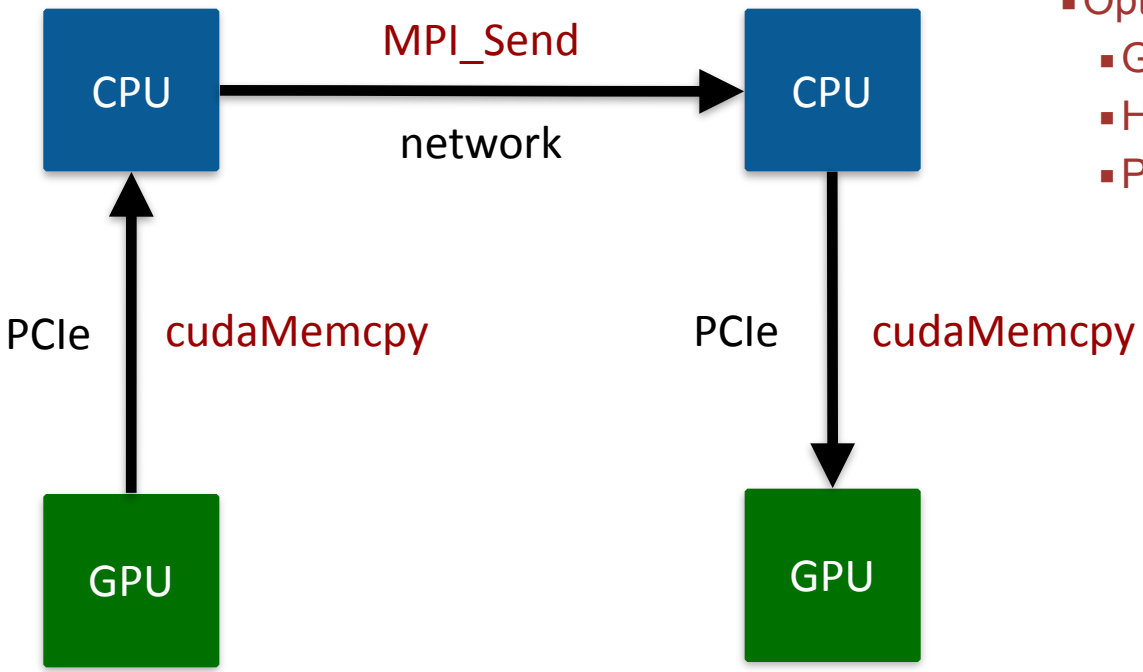
- Vendors provide software APIs to obtain power information
  - Intel: CPU, DRAM power provided by **RAPL**
  - NVIDIA: **NVML** library provide power information
  - Network power can be assumed to be static (no dynamic link on and off switching and embedded clock)

- We implemented a python framework
  - start application
  - poll RAPL and NVML to obtain power information during run time of application
  - write results to disk and create graphs

GPU-centric communication

State-of-the-art

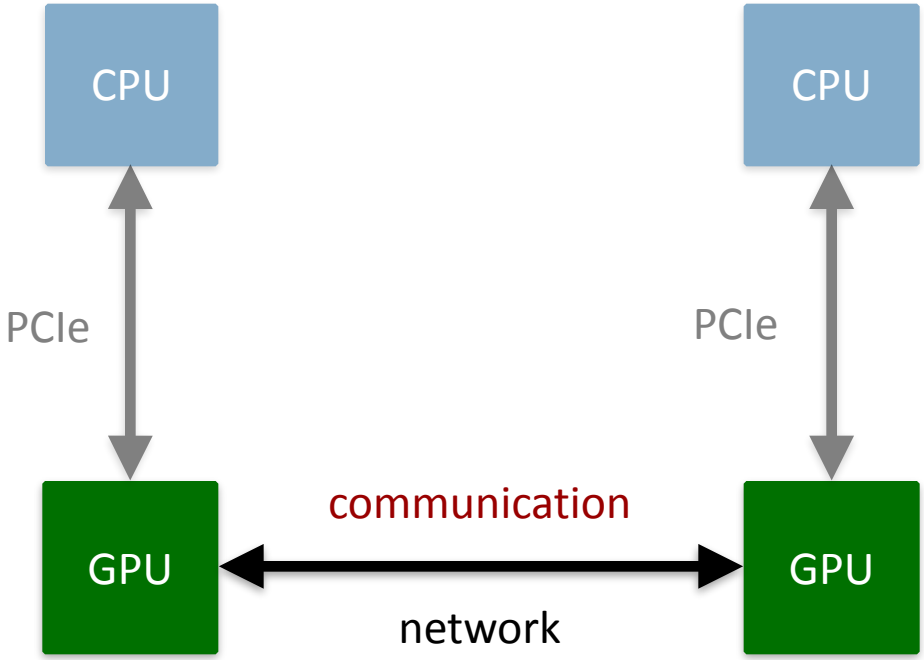
- Communication is tailored for CPUs
- Message Passing Interface (MPI) for cluster systems
- But: there is **no MPI on GPUs**



- Optimizations:
  - GPUDirect RDMA (in theory)
  - Host memory staging copies
  - Pipelining (cudaMemcpy/MPI_Send)

Direct GPU communication path

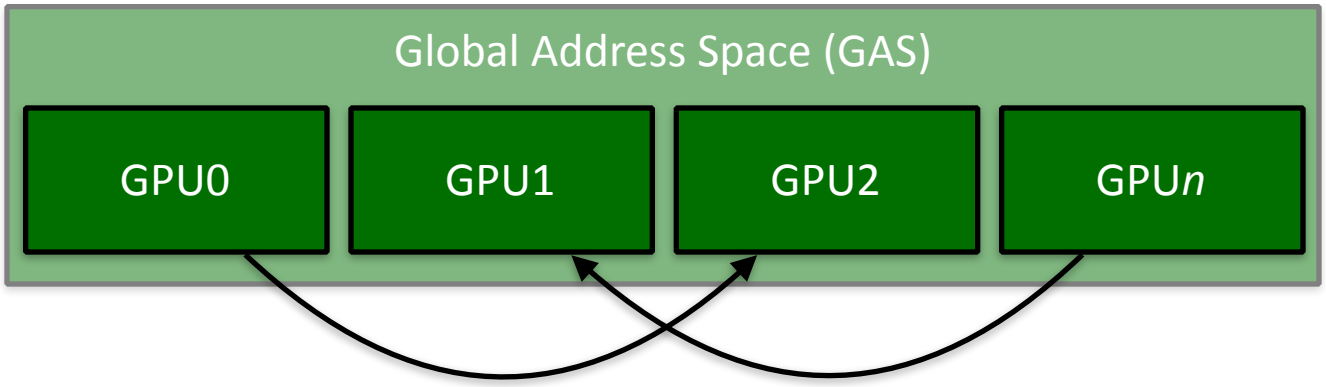
- Bypass the CPU and communicate directly
- **GPU controls network hardware**
- **No context switches** are necessary, reducing latency and improving energy efficiency



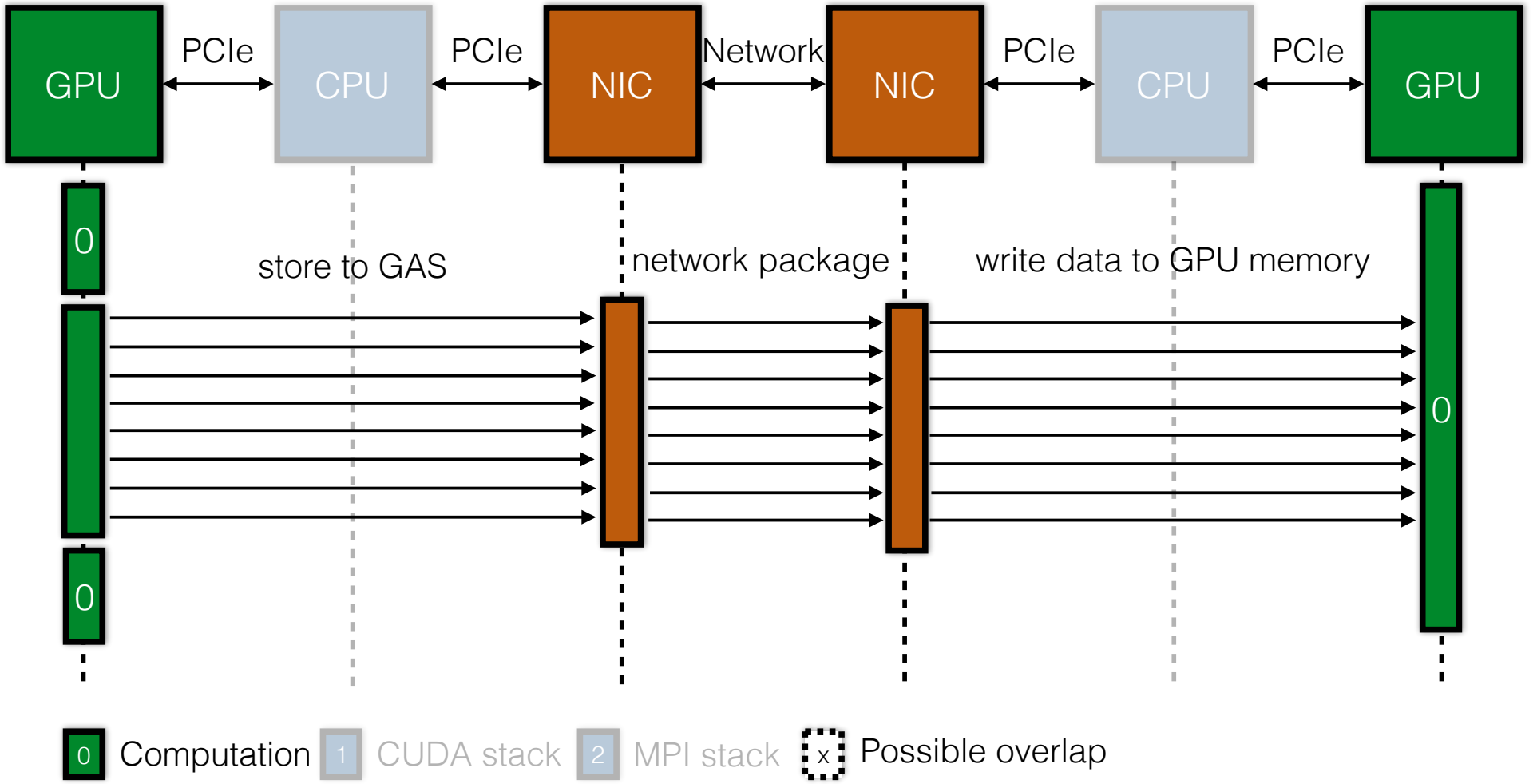
Examples	*CPU-controlled*	*GPU-controlled*
GGAS		*X*
GPU RMA		*X*
DCGN	*X*	
MVAPICH-GPU	*X*	

GGAS

- Idea: global address space for GPUs
- Memory can be accessed by every other GPU
- Communication inline with execution model
  - thread-collaborative
  - control flow remains on the GPU
- Global barrier enables synchronization



GGAS


GPU
PCIe
CPU
PCIe
NIC
Network
NIC
PCIe
CPU
PCIe
GPU
0
store to GAS
network package
write data to GPU memory
0
0
0 Computation
1 CUDA stack
2 MPI stack
x Possible overlap

GGAS

- **EXTOLL** interconnect is used
  - research project at Heidelberg University
  - company has been founded
  - still **FPGA** based (157MHz), **ASIC** (~800MHz) under test
- Functional unit to span **global address spaces** (SMFU)
  - PCIe BAR is mapped to user space
  - incoming memory transactions are forwarded and completed on target side

Energy analysis

Benchmarks

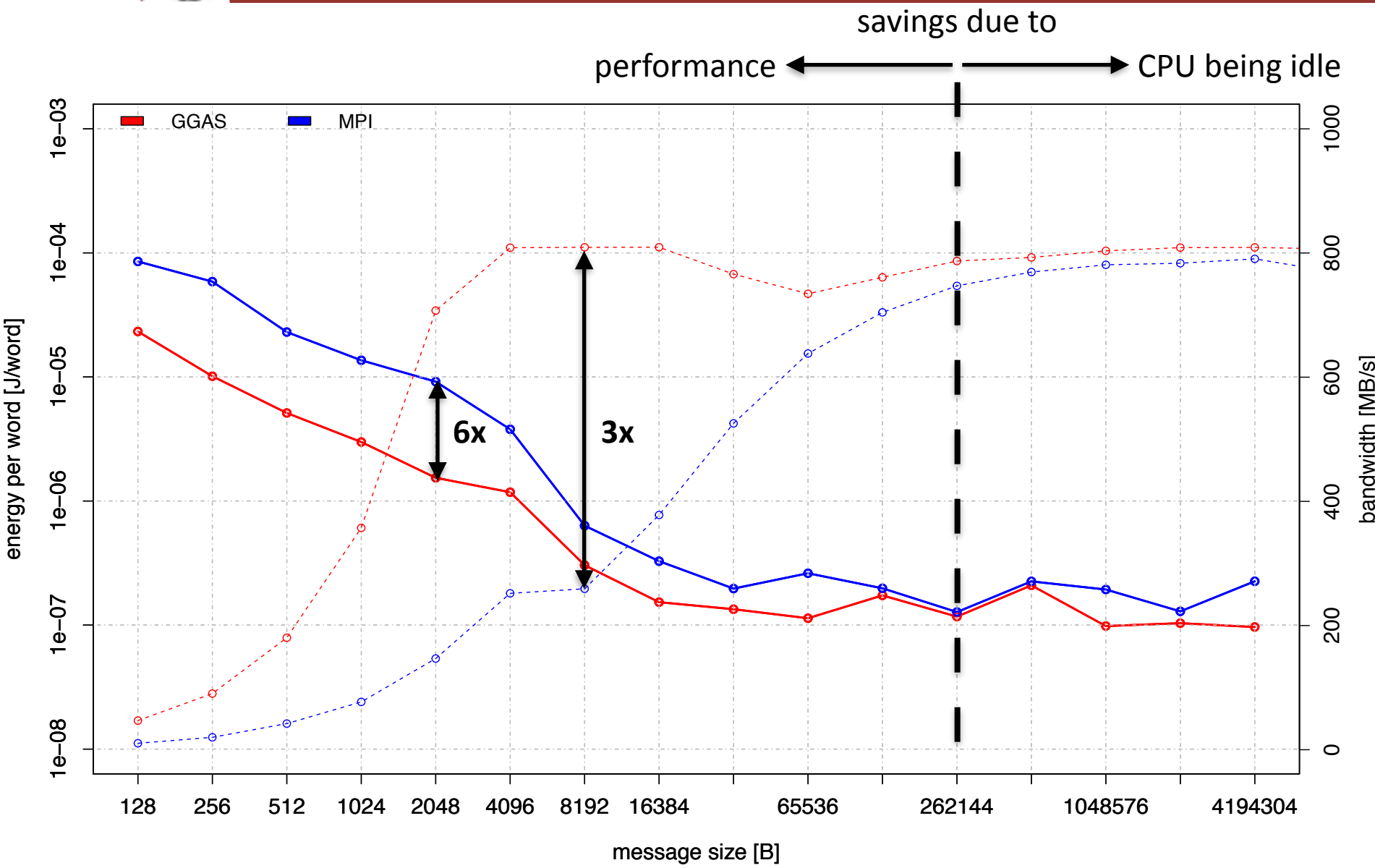
- We implemented and measured basic **microbenchmarks**
  - bandwidth
  - latency
  - barrier (synchronization costs)
- Benchmarks are implemented with
  - CPU-tailored communication: **MPI+CUDA**
  - Direct GPU communication: **GGAS**

- We introduce new metric: **Joule / word** of data transfers
- The whole system is considered and not only the network link energy consumption

Bandwidth

- Simple streaming bandwidth test
- GGAS:
  - CUDA kernel launch
  - within kernel: write data to remote GPU
  - finish
- CUDA+MPI:
  - memory copy from GPU to CPU memory
  - MPI send (respective receive on target side)
  - On target side: memory copy from CPU to GPU memory

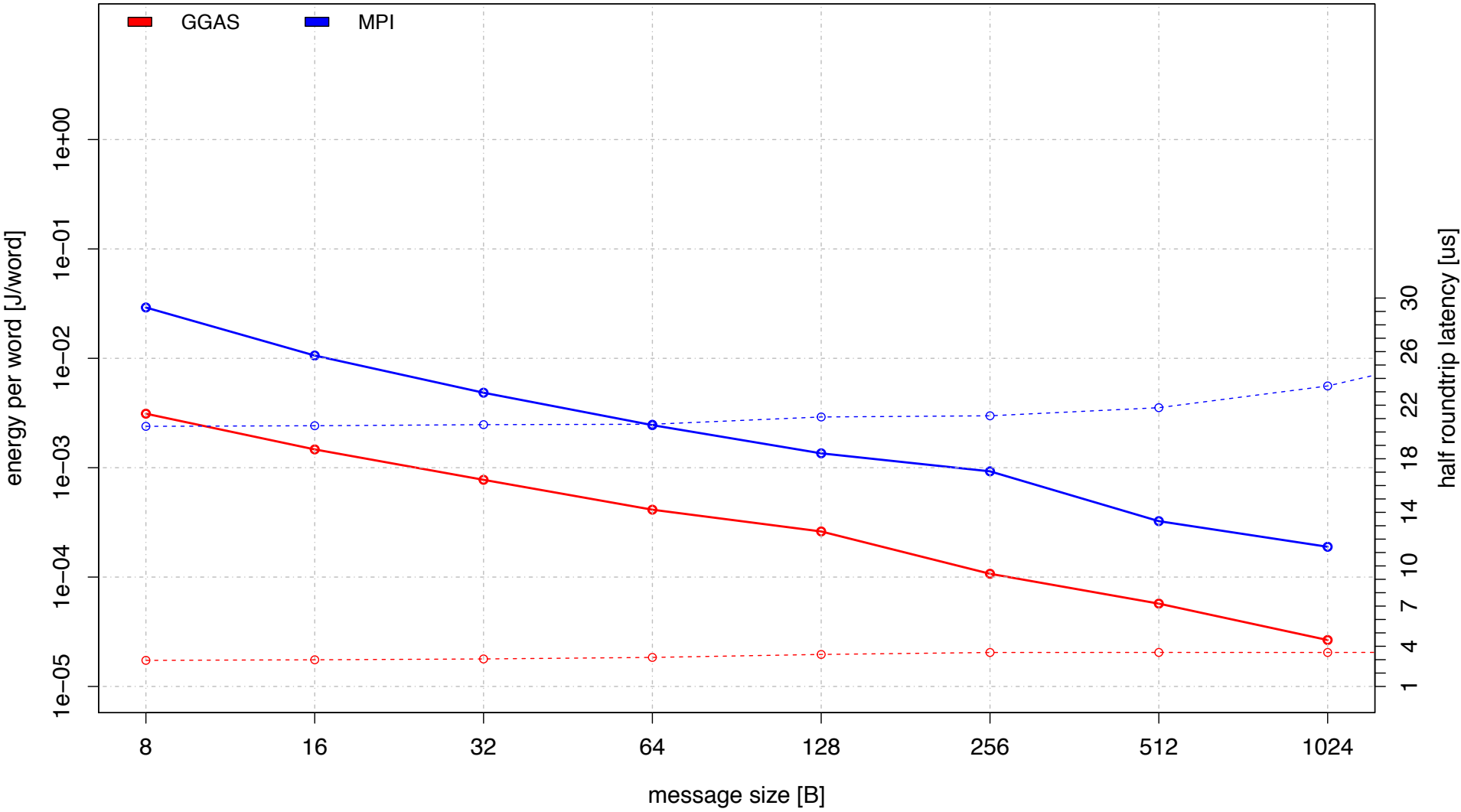
Bandwidth


savings due to
performance
CPU being idle
GGAS
MPI
6x
3x
energy per word [J/word]
bandwidth [MB/s]
1e−03
1e−04
1e−05
1e−06
1e−07
1e−08
1000
800
600
400
200
0
128
256
512
1024
2048
4096
8192
16384
65536
262144
1048576
4194304
message size [B]

Latency

- ping-pong test to determine half-round trip latency
- GGAS:
  - CUDA kernel launch
  - ping: write data to remote GPU and wait for response
  - pong: wait for data and write response to remote GPU
  - finish
- CUDA+MPI:
  - ping: memory copy to CPU, then MPI Send, followed by a Receive and memory copy to the GPU
  - pong: MPI Receive, memory copy to the GPU and back, MPI send for the response

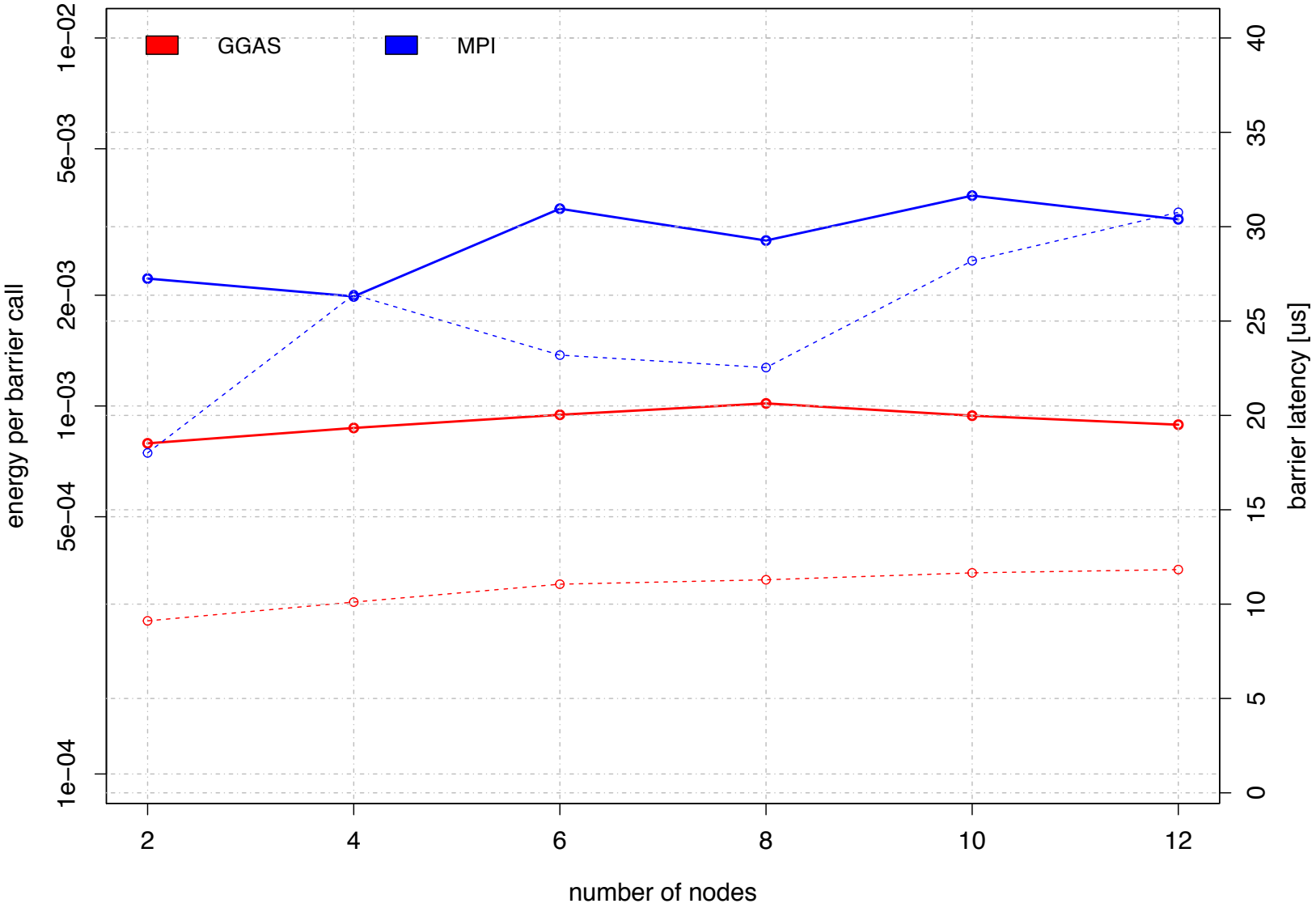
Latency


GGAS
MPI
energy per word [J/word]
1e+00
1e−01
1e−02
1e−03
1e−04
1e−05
half roundtrip latency [us]
1
4
7
10
14
18
22
26
30
8
16
32
64
128
256
512
1024
message size [B]

Barrier

- The barrier is called several times
- GGAS:
  - CUDA kernel launch
  - call barrier plenty of times
  - finish
- CUDA+MPI:
  - dummy kernel launch (to consider context switches)
  - MPI barrier call
  - repeat

Barrier


GGAS
MPI
energy per barrier call
1e−02
5e−03
2e−03
1e−03
5e−04
1e−04
barrier latency [us]
40
35
30
25
20
15
10
5
0
2
4
6
8
10
12
number of nodes

Power consumption

		CPU	GPU	DRAM	Total
Bandwidth	GGAS	26.86	47.05	3.99	77.9
	MPI	43.36	48.86	10.11	102.33
Ping-pong	GGAS	29.95	52.58	5.26	87.79
	MPI	43.25	48.45	8.07	99.77
Barrier	GGAS	18.76	56.68	4.33	79.77
	MPI	39.97	51.18	8.48	99.63

- Numbers refer to **average power consumption** over all measured message sizes
- CPU power is always lower for GGAS
- GPU power about the same
- Total power **savings from 10W to 15W**

Conclusion & Future Work

Conclusion

- We implemented and analyzed three microbenchmarks regarding energy per word of data transfers.
- GPU-centric communication shows **improved energy efficiency**
  - bandwidth: 50.67% energy savings ; 2.42x performance
  - latency: 85.01% energy savings ; 6.96x performance
  - barrier: 67.31% energy savings ; 2.28x performance
- This is due to two aspects:
  - **performance**: context switches increase latency
  - **power consumption**: context switches prevent the CPU from entering power saving states

Insights

- GPU-centric communication is **superior in performance and energy efficiency**
- It is more energy efficient to communicate larger messages instead of very small ones (**aggregating messages** can be an option to improve energy efficiency)
- Synchronization is more energy efficient when it does not require **context switches**
- **energy/word** metric characterizes overall communication efficiency (including source and sink processors), and not only the physical link implementation (energy/bit metric)

Future Work

- Analysis of the impact of communication methods on **application** performance / energy consumption
- **Communication library** for GPUs including
  - thread-collaborative communication
  - one-sided communication (offloading to NIC)
  - collective operations (reduce, all-to-all, scatter, gather)
- Exploration of further **optimizations** including simulation
- **Power and performance models** including CPU, GPU and network

Thank you for your attention !

Q&A

References

[1]Top500, www.top500.org, June 2014

[2] Green500, www.green500.org, June 2014

[3] Exascale Computing Study: Technology Challenges in Achieving Exascale Systems, September 28, 2008

[4] Oden, L.; Froning, H., "GGAS: Global GPU address spaces for efficient communication in heterogeneous clusters," Cluster Computing (CLUSTER), 2013 IEEE International Conference

[5] Stuart, J.A; Owens, J.D., "Message passing on data-parallel architectures," Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on , vol., no., pp.1,12, 23-29 May 2009

[6] S. Potluri, K. Hamidouche, A. Venkatesh, D. Bureddy, and D. K. Panda, Efficient Inter-node MPI Communication using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs Int'l Conference on Parallel Processing (ICPP '13), October 2013. , October 2013

[7] Holger Fröning and Heiner Litz, Efficient Hardware Support for the Partitioned Global Address Space, 10th Workshop on Communication Architecture for Clusters (CAC2010), co-located with 24th International Parallel and Distributed Processing Symposium (IPDPS 2010), April 19, 2010, Atlanta, Georgia.

[8] Benjamin Klenk, Lena Oden, Holger Fröning, Analyzing Put/Get APIs for Thread-Collaborative Processors, Workshop on Heterogeneous and Unconventional Cluster Architectures and Applications (HUCAA) co-located with Internation Conference on Parallel Processing (ICPP2014), September 12, 2014, Minneapolis, Minnesota