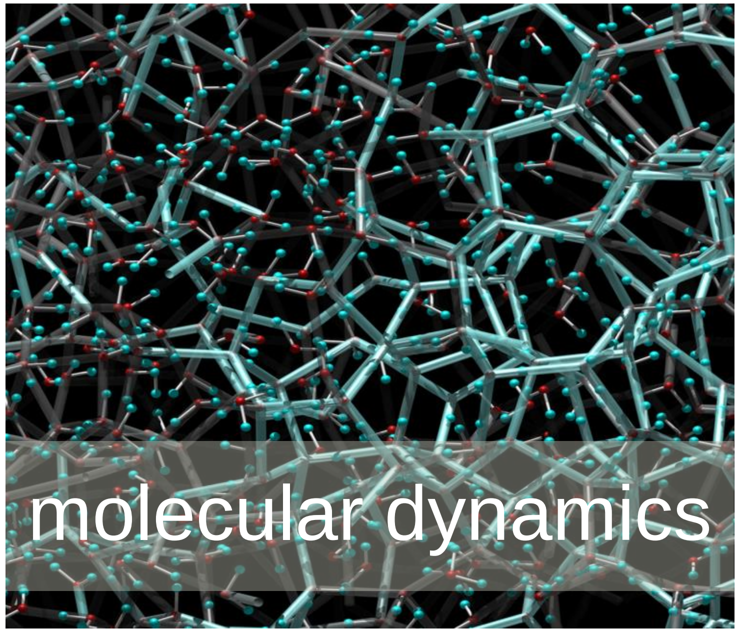
dCUDA: Hardware Supported Overlap of Computation and Communication

Tobias Gysi, Jeremia Bär, and Torsten Hoefler

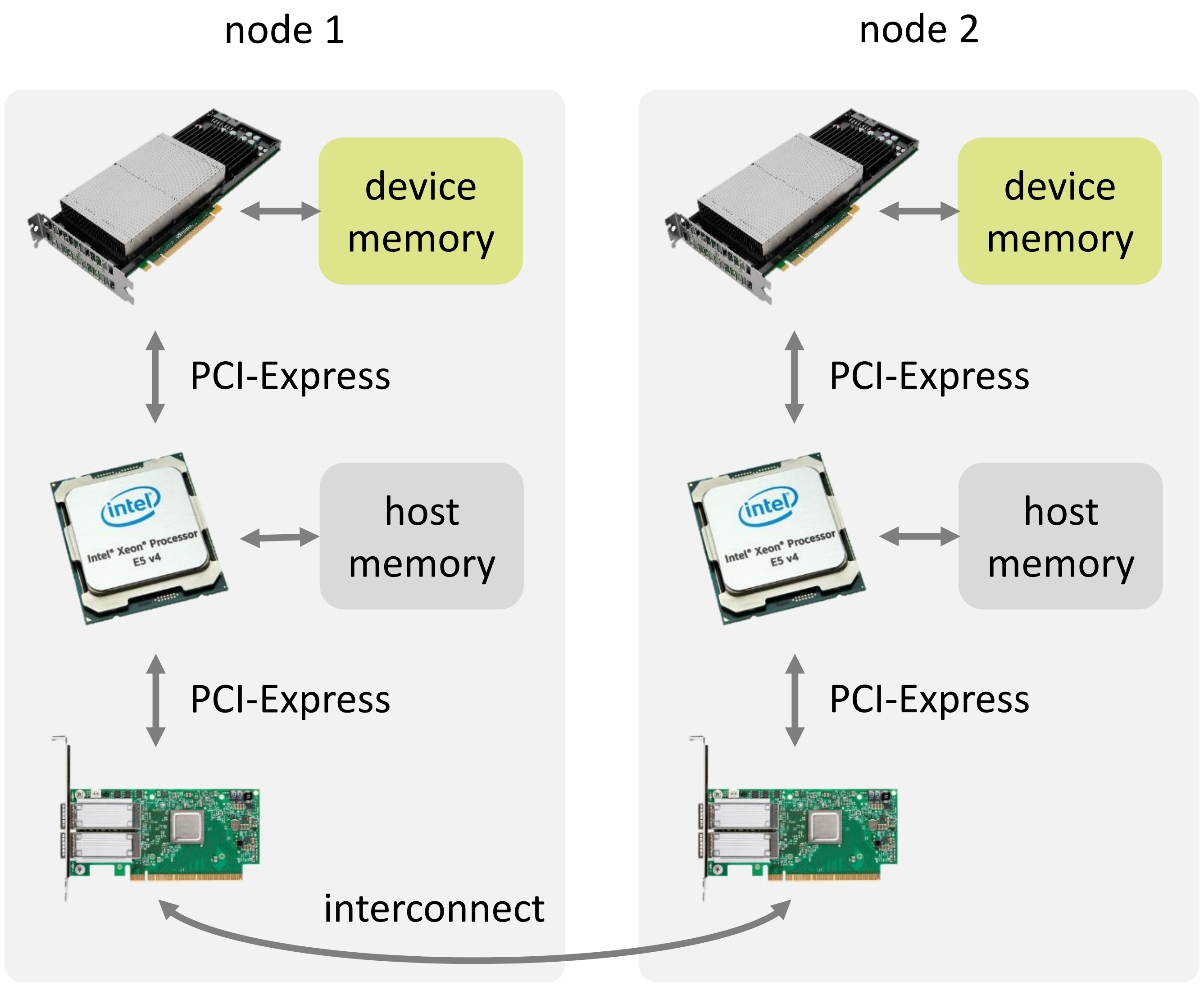
Platform for Advanced Scientific Computing

Swiss university conference

GPU computing gained a lot of popularity in various application domains



GPU cluster programming using MPI and CUDA



code

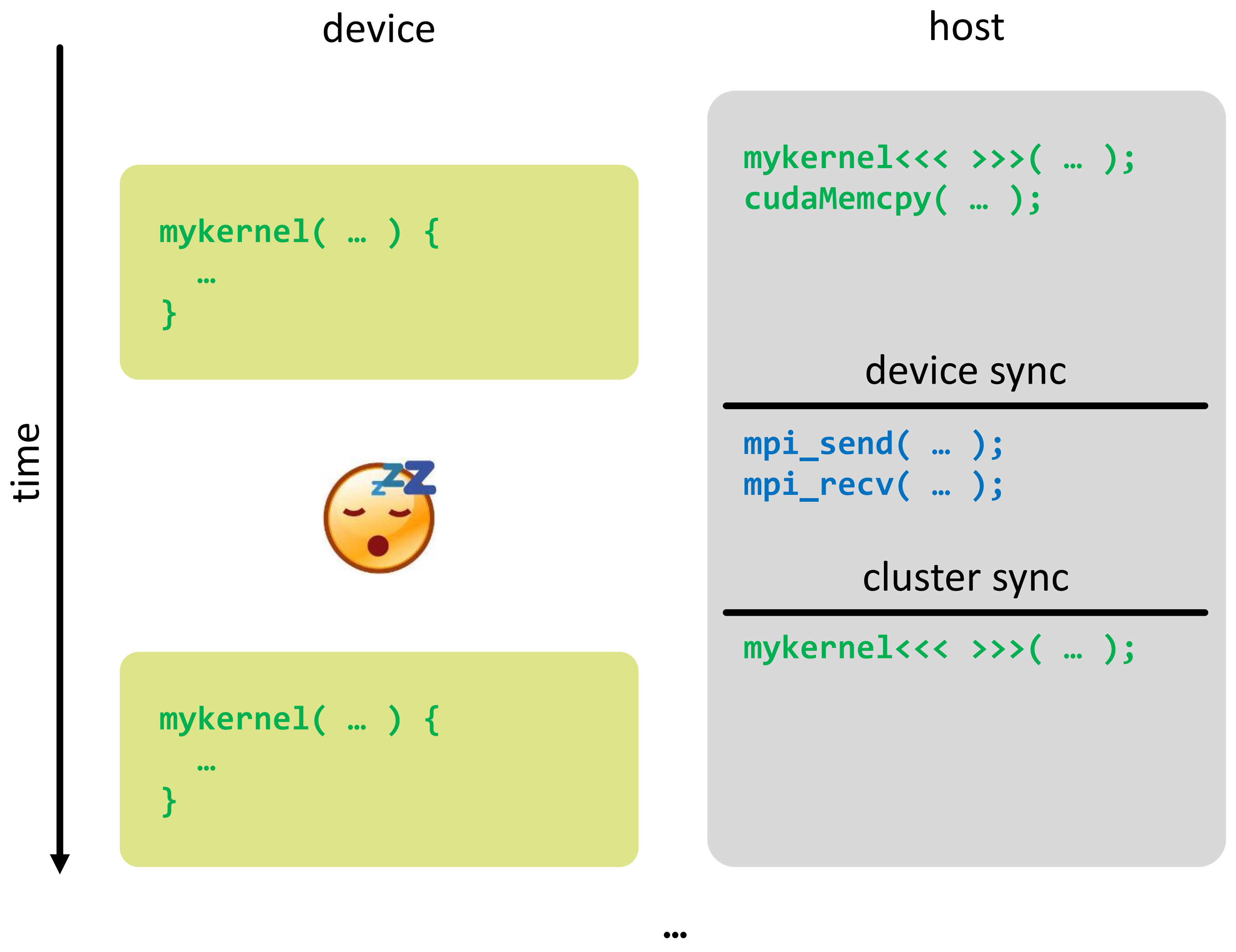
```
// run compute kernel
__global__
void mykernel( … ) { }
```

```
// launch compute kernel
mykernel<<<64,128>>>( … );

// on-node data movement
cudaMemcpy(
  psize, &size,
  sizeof(int),
  cudaMemcpyDeviceToHost);

// inter-node data movement
mpi_send(
  pdata, size,
  MPI_FLOAT, … );
mpi_recv(
  pdata, size,
  MPI_FLOAT, … );
```

Disadvantages of the MPI-CUDA approach



```
mykernel( … ) {
  …
}
```

```
mykernel<<< >>>( … );
cudaMemcpy( … );
```

```
mpi_send( … );
mpi_recv( … );
```

```
mykernel<<< >>>( … );
```

```
mykernel( … ) {
  …
}
```

complexity

- two programming models
- duplicated functionality

copy

sync

performance

- encourages sequential execution
- low utilization of the costly hardware

Achieve high resource utilization using oversubscription & hardware threads

code

```
ld  %r0,%r1
mul %r0,%r0,3
st  %r0,%r1
```

GPU cores use “parallel slack” to hide instruction pipeline latencies

time

thread 1	thread 2	thread 3	instruction pipeline	
ld %r0,%r1	ready	ready	ld	
stall	ld %r0,%r1	ready	ld	ld
ready	stall	ld %r0,%r1	ld	ld
mul %r0,%r0,3	ready	stall	mul	ld
stall	mul %r0,%r0,3	ready	mul	mul
ready	stall	mul %r0,%r0,3	mul	mul
st %r0,%r1	ready	stall	st	mul
stall	st %r0,%r1	ready	st	st
ready	stall	st %r0,%r1	st	st

…

Use oversubscription & hardware threads to hide remote memory latencies

code

```
get …
mul %r0,%r0,3
put …
```

introduce put & get operations to access distributed memory

time

thread 1	thread 2	thread 3	instruction pipeline	
get …	ready	ready	get	
stall	get …	ready	get	get
stall	stall	get …	get	get
stall	stall	stall	!	get
ready	stall	stall	!	!
mul %r0,%r0,3	ready	stall	mul	!
stall	mul %r0,%r0,3	ready	mul	mul
ready	stall	mul %r0,%r0,3	mul	mul
put …	ready	stall	put	mul

…

How much “parallel slack” is necessary to fully utilize the interconnect?

Little’s law

$$concurrency = latency \ * throughput$$

	device memory		interconnect
latency	1µs		19µs
bandwidth	200GB/s		6GB/s
concurrency	200kB		114kB
#threads	~12000	>>	~7000

dCUDA (distributed CUDA) extends CUDA with MPI-3 RMA and notifications

```
for (int i = 0; i < steps; ++i) {
  for (int idx = from; idx < to; idx += jstride)
    out[idx] = -4.0 * in[idx] +
      in[idx + 1] + in[idx - 1] +
      in[idx + jstride] + in[idx - jstride];

  if (lsend)
    dcuda_put_notify(ctx, wout, rank - 1,
      len + jstride, jstride, &out[jstride], tag);
  if (rsend)
    dcuda_put_notify(ctx, wout, rank + 1,
      0, jstride, &out[len], tag);

  dcuda_wait_notifications(ctx, wout,
    DCUDA_ANY_SOURCE, tag, lsend + rsend);

  swap(in, out);
  swap(win, wout);
}
```

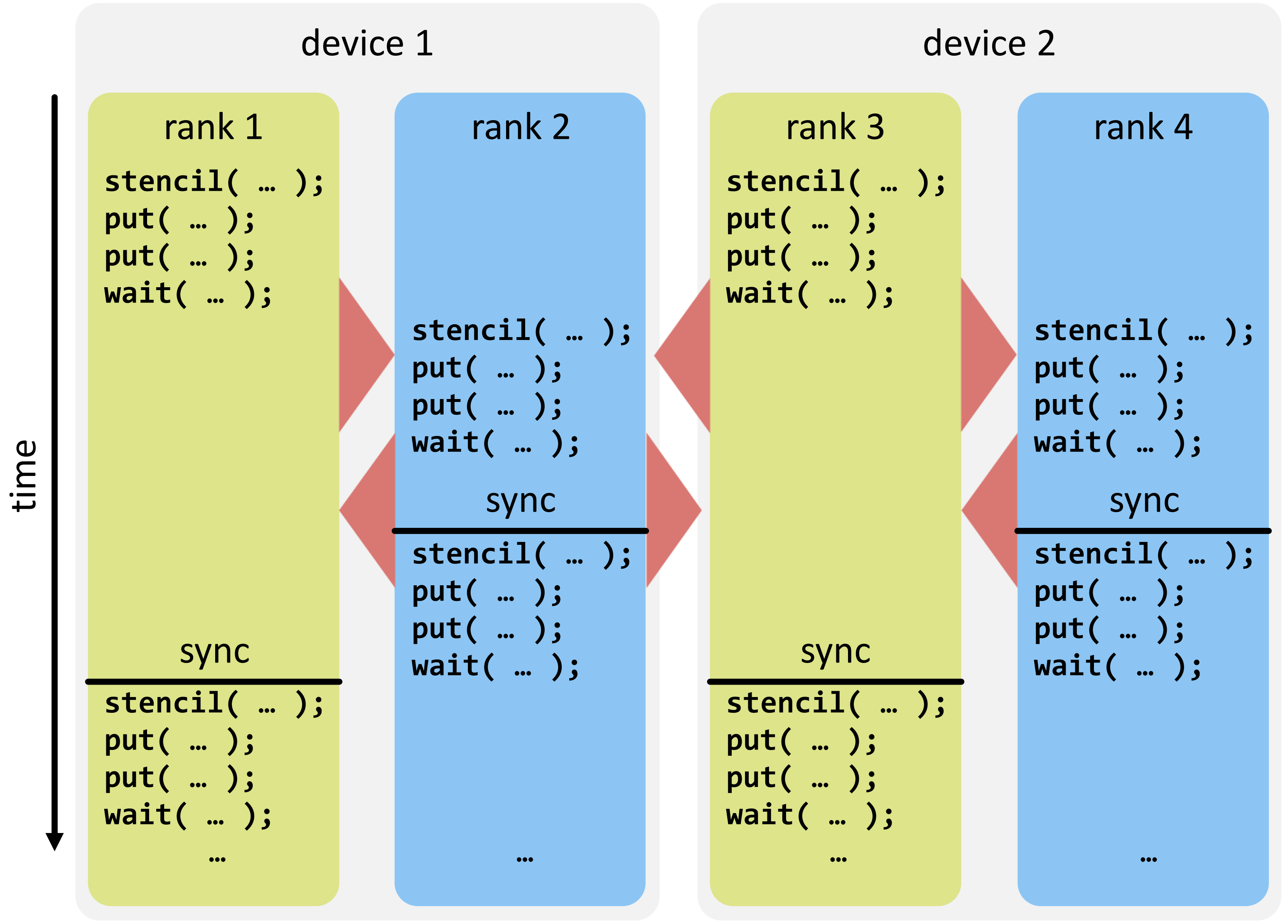
computation

communication

- iterative stencil kernel
- thread specific idx

- map ranks to blocks
- device-side put/get operations
- notifications for synchronization
- shared and distributed memory

Advantages of the dCUDA approach

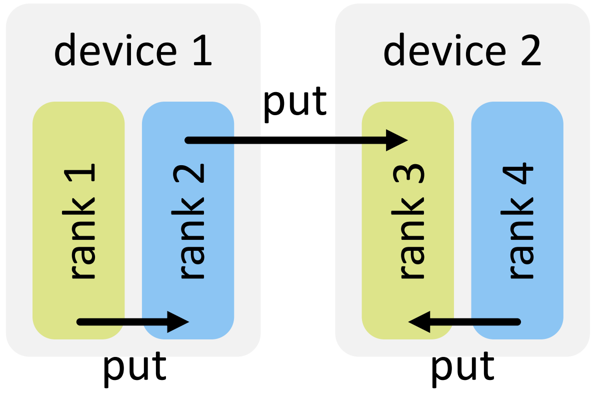


performance

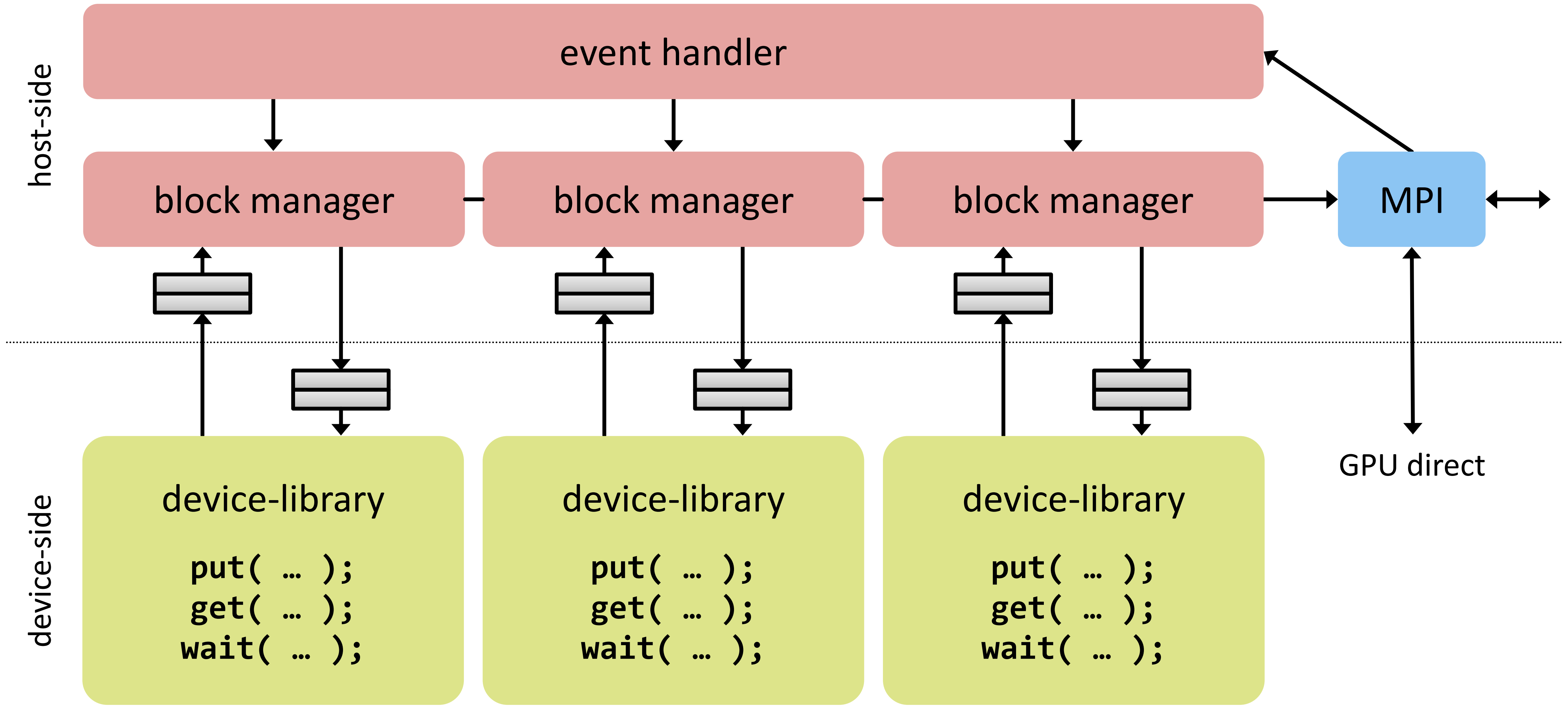
- avoid device synchronization
- latency hiding at cluster scale

complexity

- unified programming model
- one communication mechanism

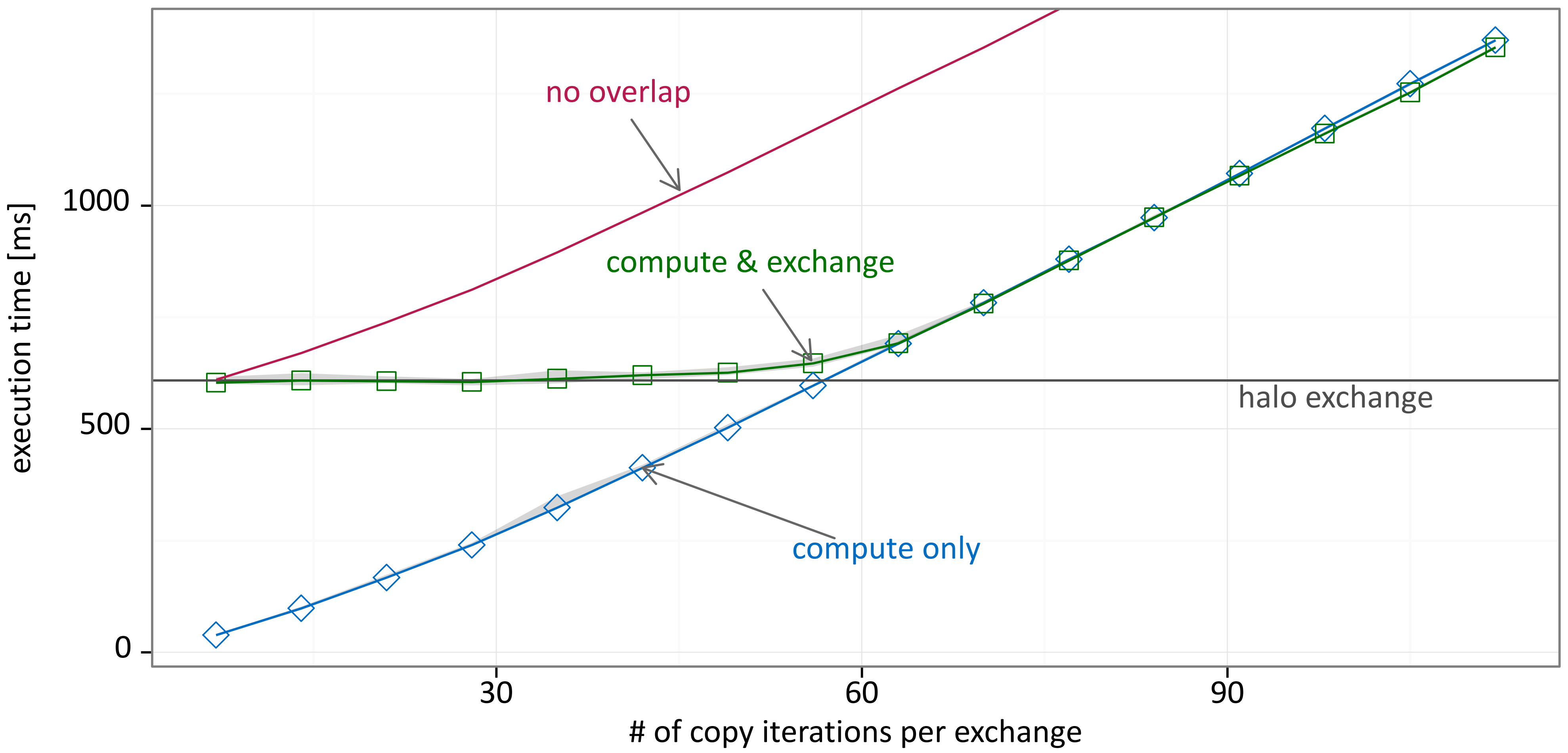


Implementation of the dCUDA runtime system



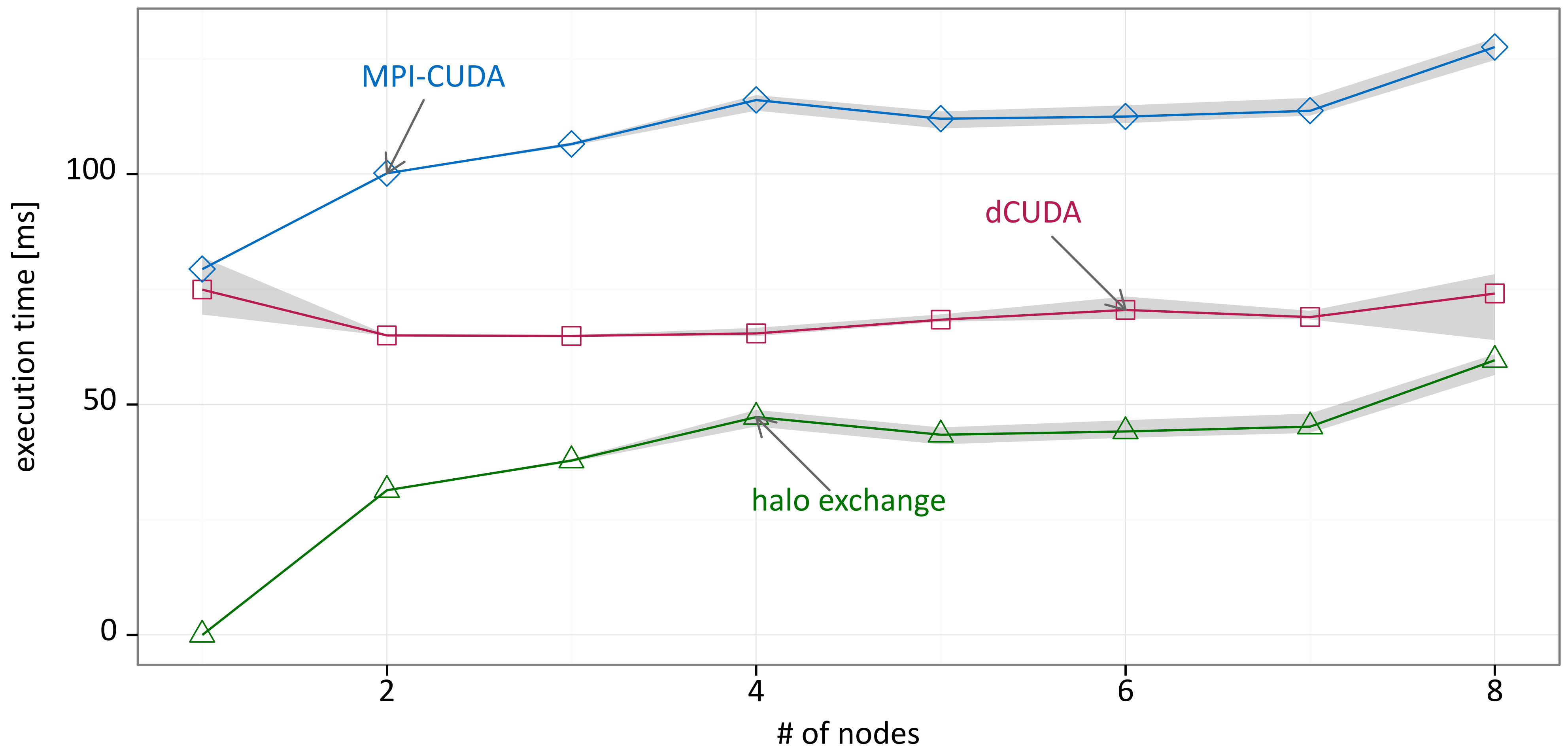
Overlap of a copy kernel with halo exchange communication

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)



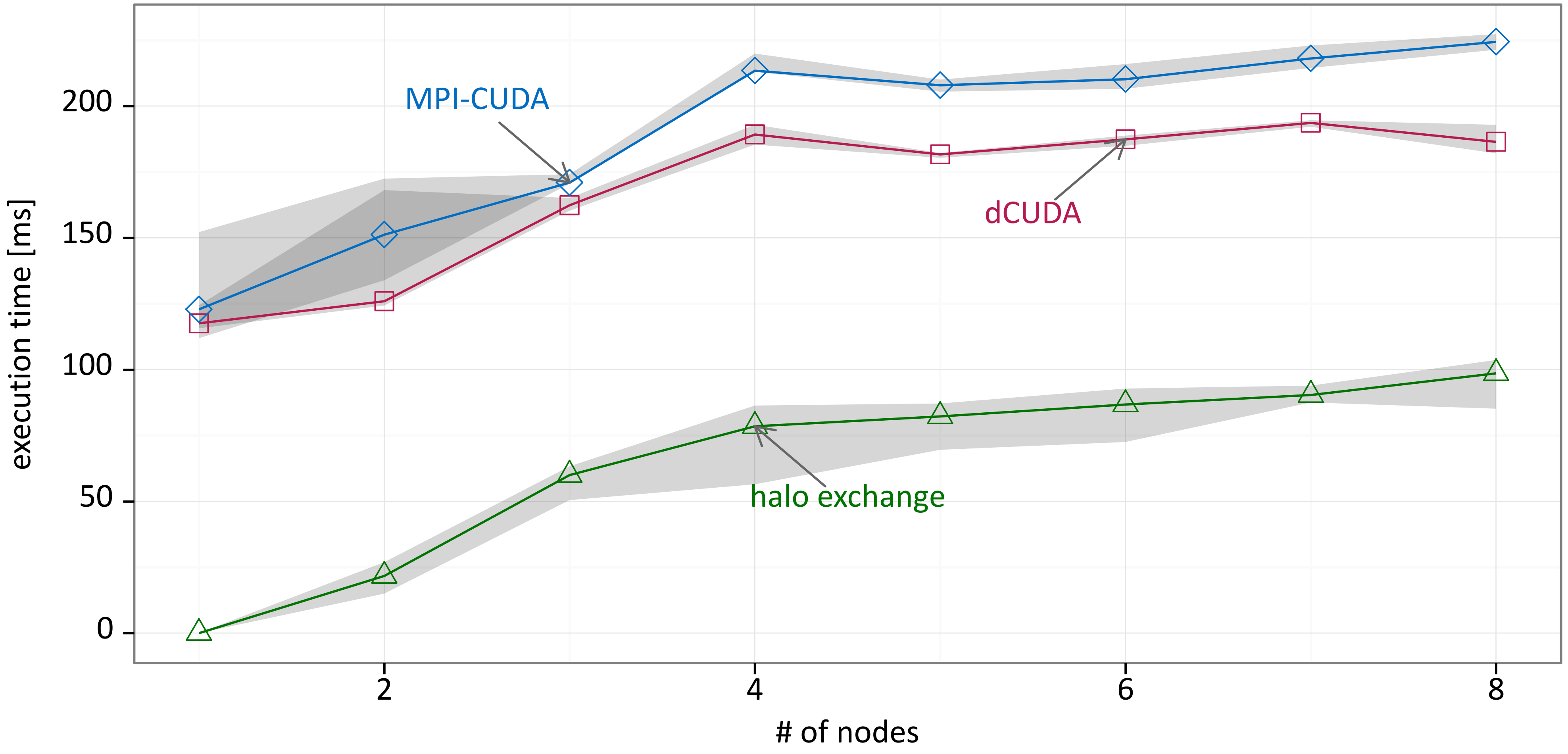
Weak scaling of MPI-CUDA and dCUDA for a stencil program

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)



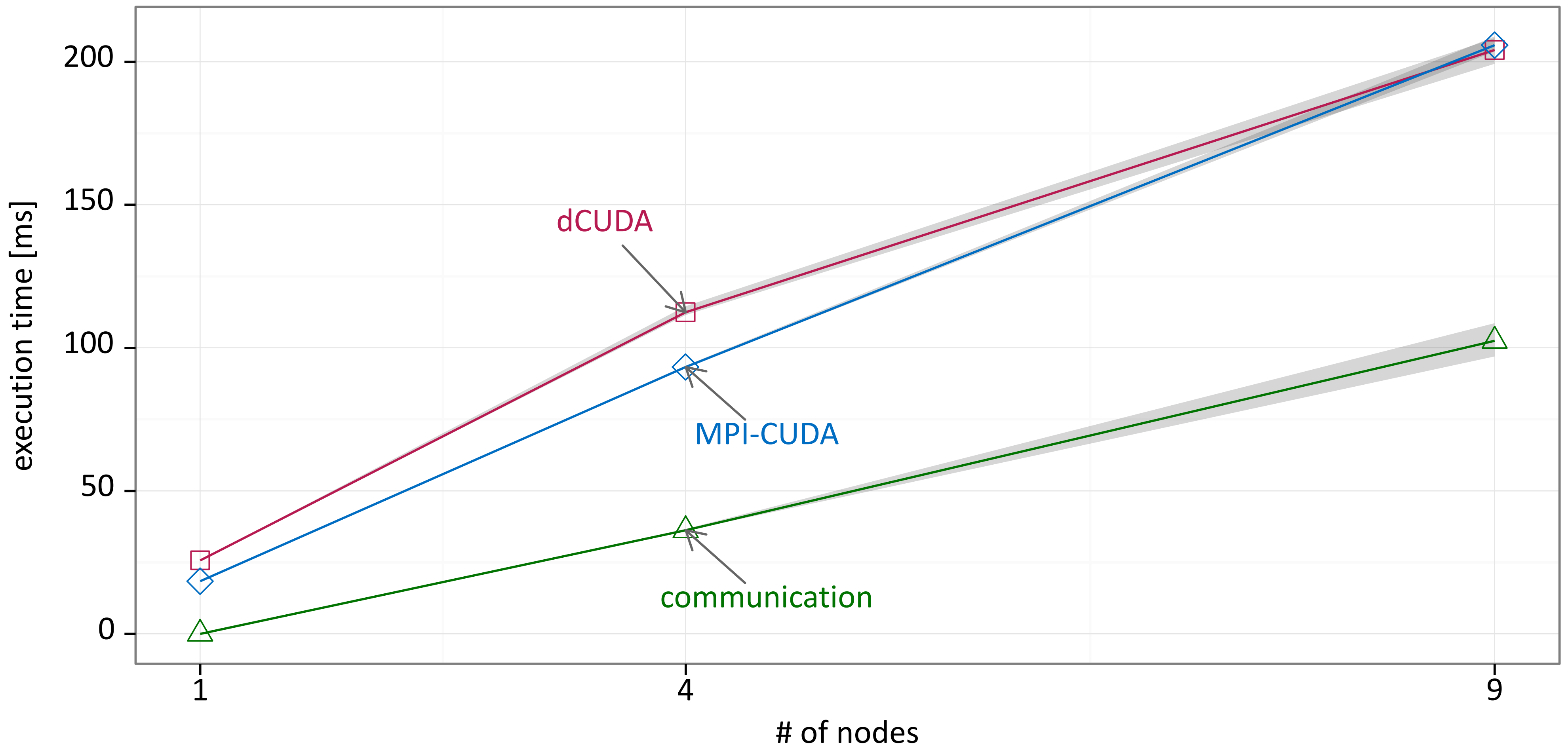
Weak scaling of MPI-CUDA and dCUDA for a particle simulation

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)



Weak scaling of MPI-CUDA and dCUDA for sparse-matrix vector multiplication

benchmarked on Greina (8 Haswell nodes with 1x Tesla K80 per node)



Conclusions

- unified programming model for GPU clusters
  - device-side remote memory access operations with notifications
  - transparent support of shared and distributed memory
- extend the latency hiding technique of CUDA to the full cluster
  - inter-node communication without device synchronization
  - use oversubscription & hardware threads to hide remote memory latencies
- automatic overlap of computation and communication
  - synthetic benchmarks demonstrate perfect overlap
  - example applications demonstrate the applicability to real codes
- https://spcl.inf.ethz.ch/Research/Parallel_Programming/dCUDA/

Platform for Advanced Scientific Computing
Swiss university conference
ETH-RAT
CSCS