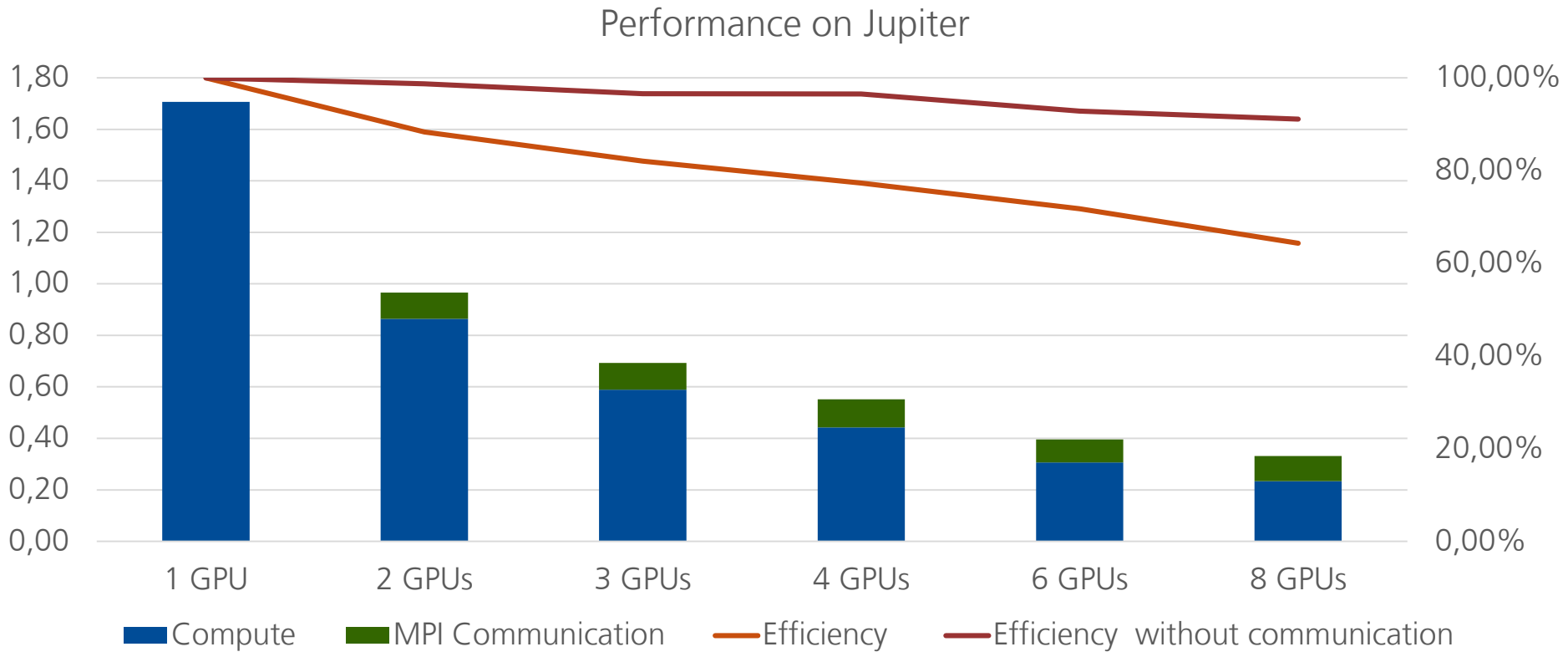
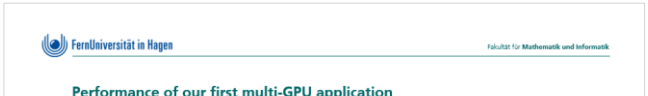
Fakultät für
Mathematik und Informatik

Optimizing Strategies for Multi-GPU Applications

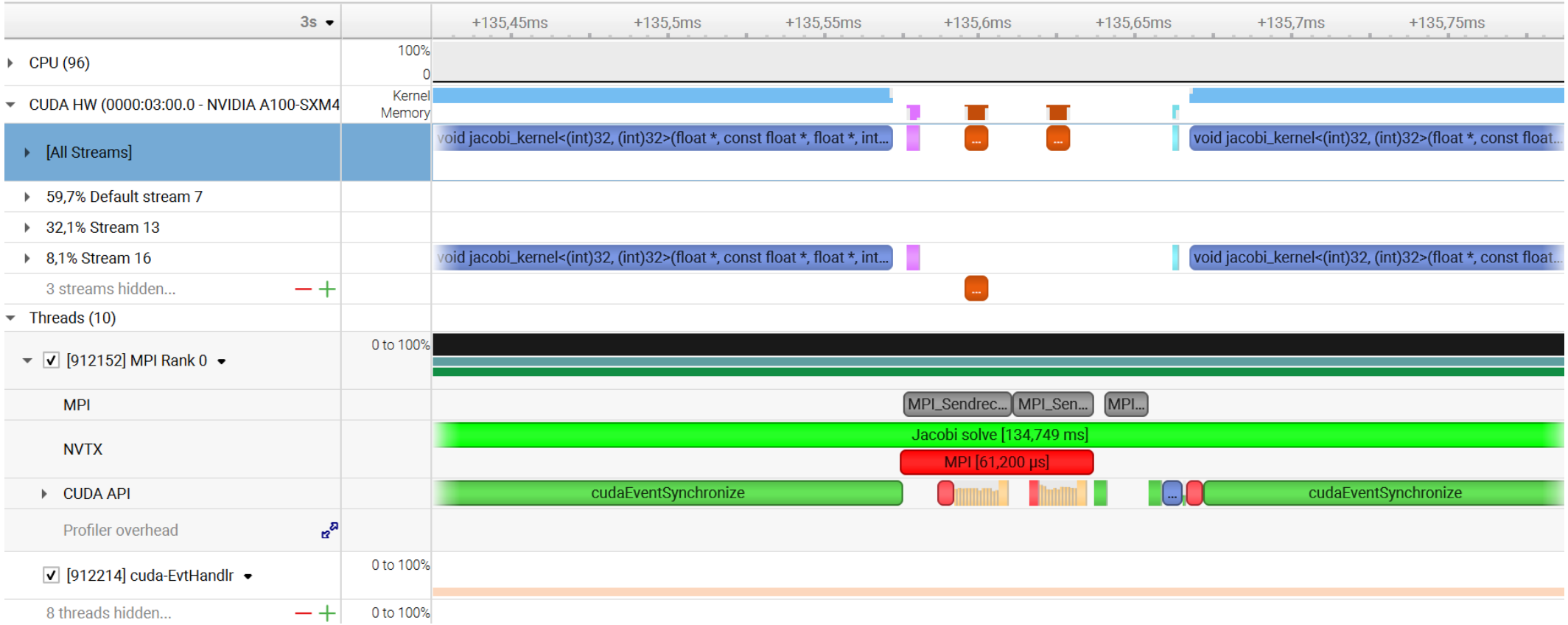
FernUniversität in Hagen

Performance of our first multi-GPU application

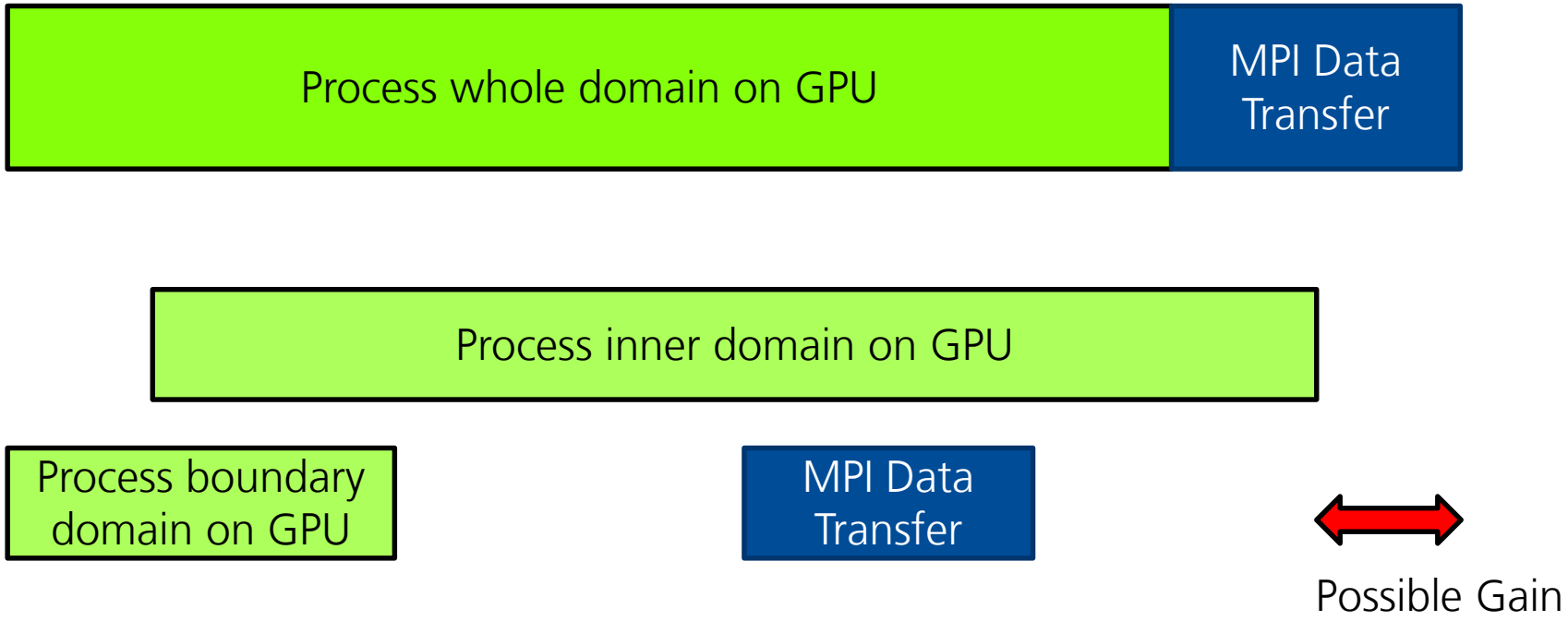




Analsysis with Nsight System



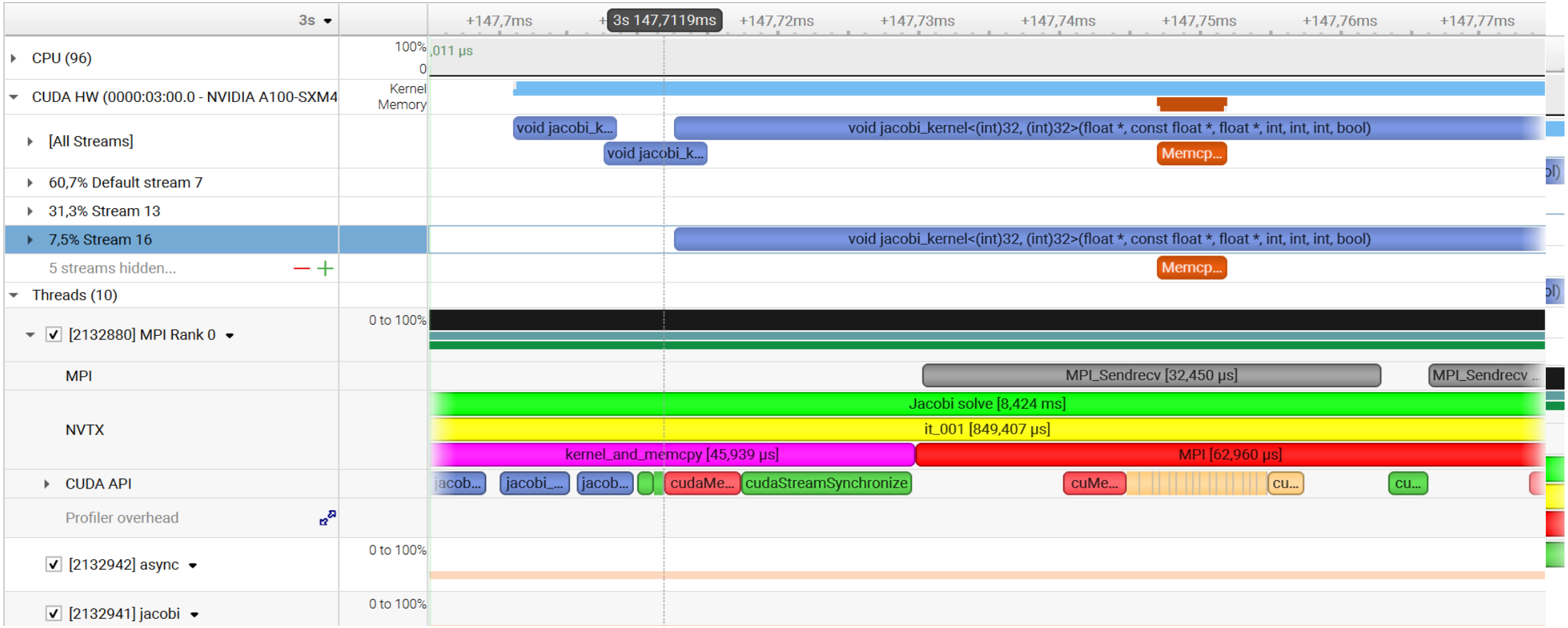
Overlapping communication and computation



Overlapping communication and computation

```
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_top_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx, push_bottom_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);
const int top = rank > 0 ? rank - 1 : (size - 1);
const int bottom = (rank + 1) % size;

cudaStreamSynchronize(push_top_stream)

MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0, a_new + (iy_end * nx), nx,
        MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaStreamSynchronize(push_bottom_stream));

MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0, a_new, nx,
        MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

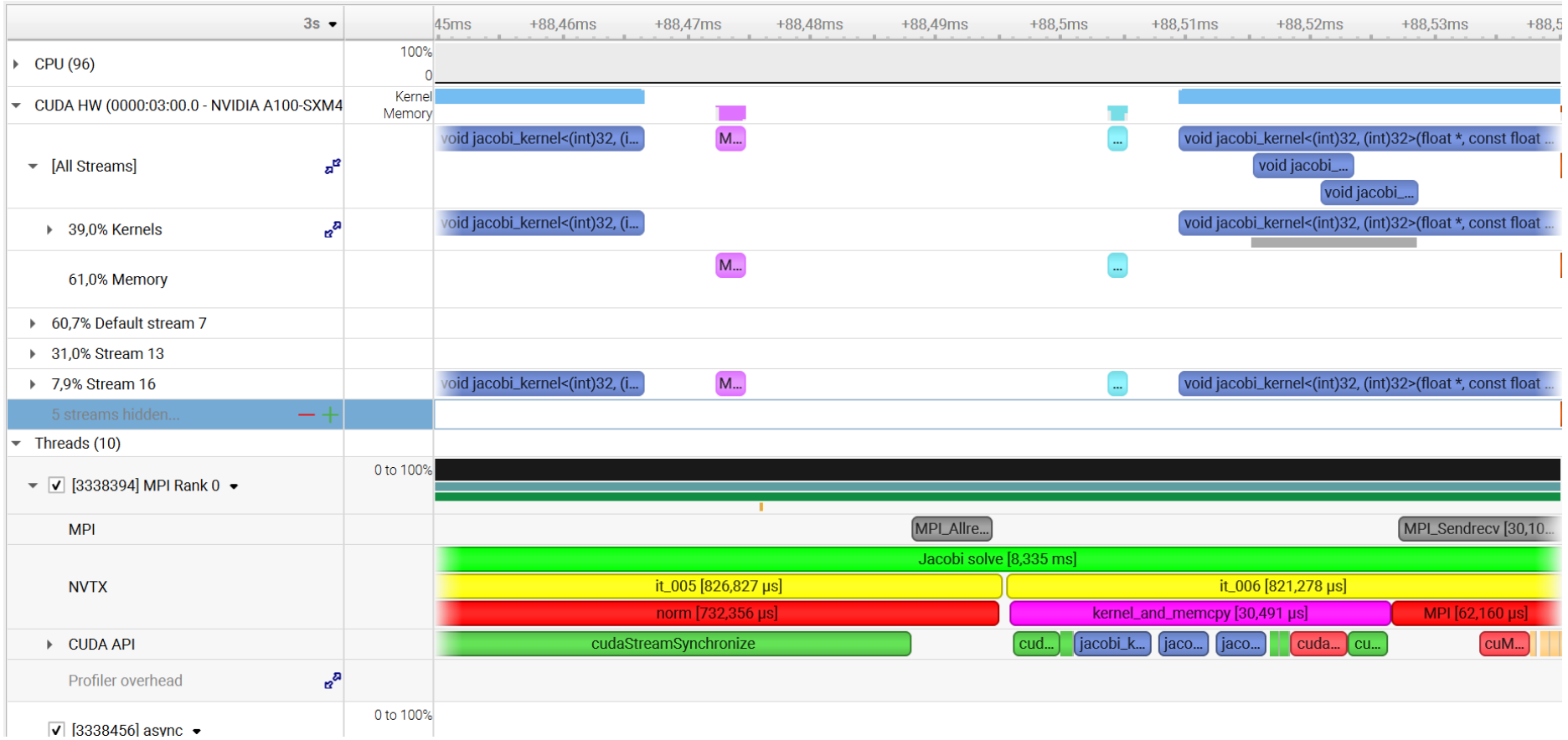
Overlapping communication and computation



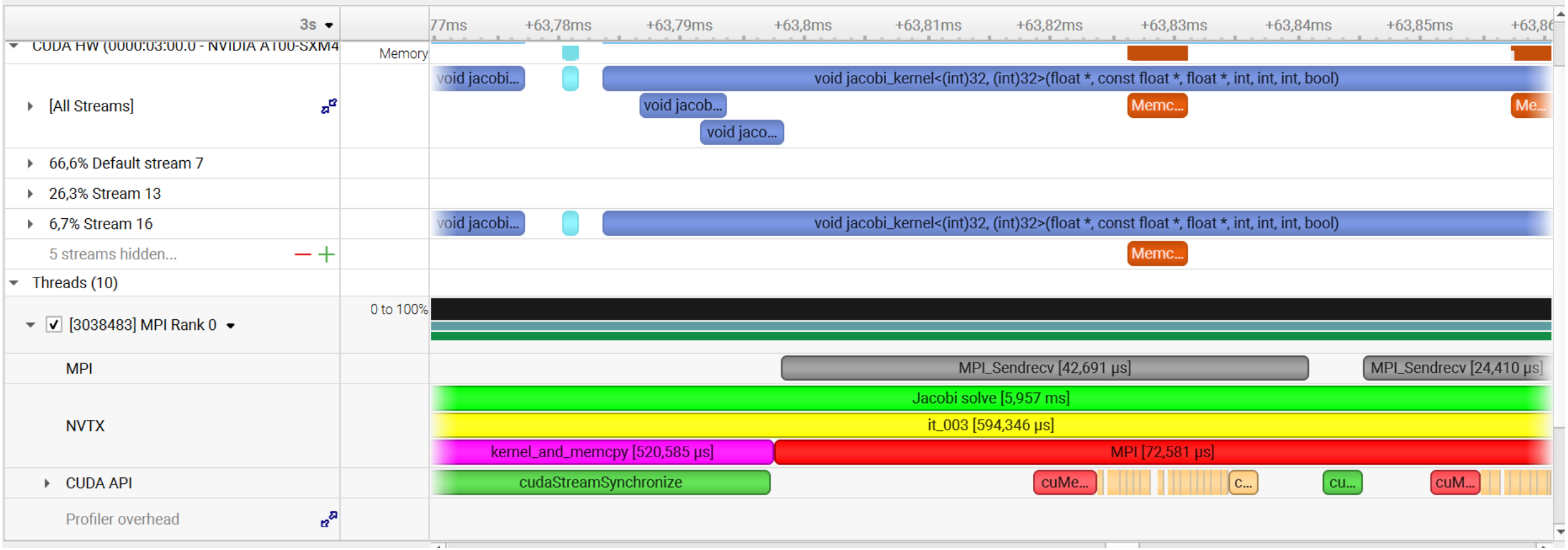
Overlapping communication and computation

```
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_top_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx, push_bottom_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1), nx, compute_stream);
const int top = rank > 0 ? rank - 1 : (size - 1);
const int bottom = (rank + 1) % size;

cudaStreamSynchronize(push_top_stream)

MPI_Sendrecv(a_new + iy_start * nx, nx, MPI_REAL_TYPE, top, 0, a_new + (iy_end * nx), nx,
       MPI_REAL_TYPE, bottom, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
cudaStreamSynchronize(push_bottom_stream));

MPI_Sendrecv(a_new + (iy_end - 1) * nx, nx, MPI_REAL_TYPE, bottom, 0, a_new, nx,
       MPI_REAL_TYPE, top, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

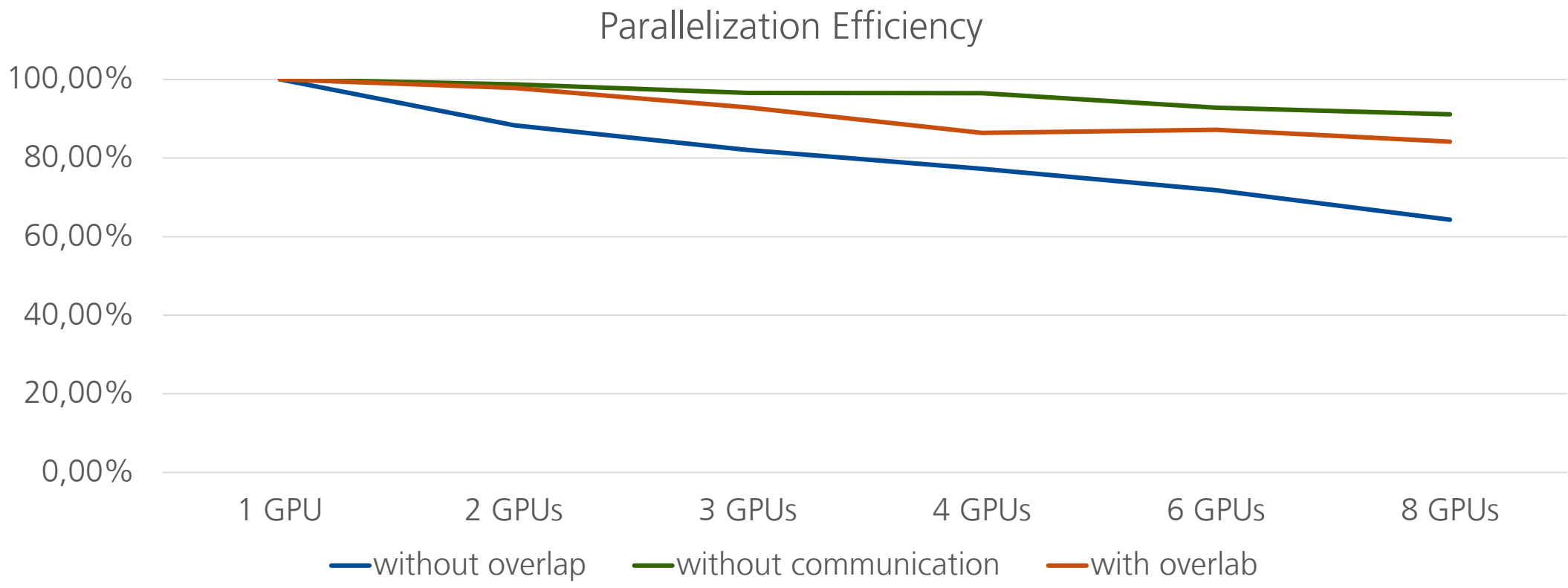
Overlapping Communication and Computation



Avoid Synchronisation



Performance comparison on Jupiter



So – Everything is perfect? But Wait….

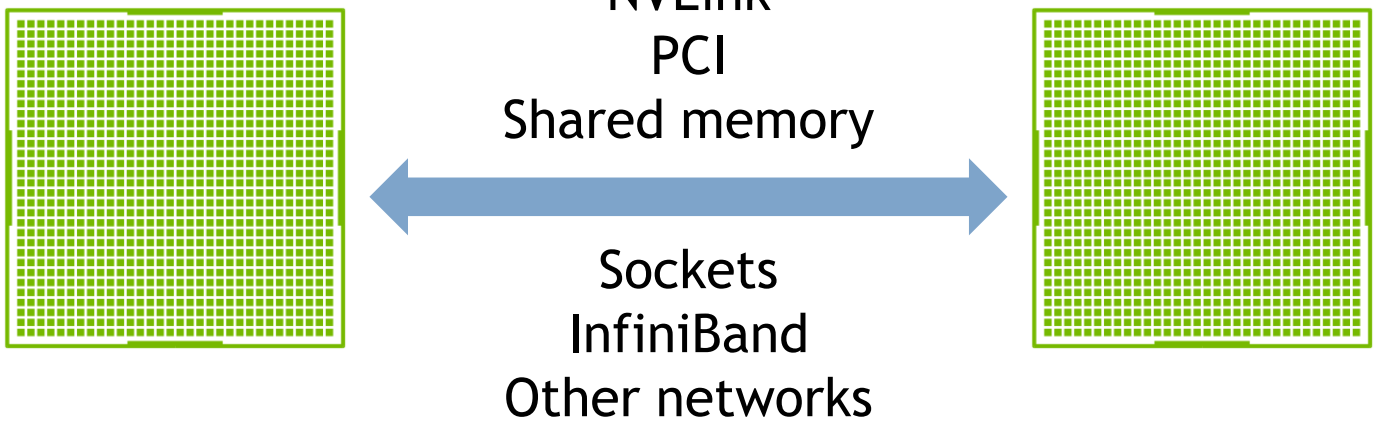
- MPI is **not** aware of CUDA streams
- Explicit synchronization between GPU-compute kernel and CPU communication calls is required
- CUDA-aware MPI is *GPU-memory-aware* communication
- For better efficiency: *CUDA-stream-aware* communication
    - Communication, which is aware of CUDA-streams or use CUDA streams
    - NCCL (and (Host-API) of NVSHMEM)

Optimized inter-GPU communication

NCCL : NVIDIA Collective Communication Library

Communication library running on GPUs, for GPU buffers.

- Library for efficient communication with GPUs
- First: Collective Operations (e.g. Allreduce), as they are required for DeepLearning
- Since 2.8: Support for Send/Recv between GPUs
- Library running on GPU: Communication calls are translated to GPU a kernel (running on a stream)



Binaries : https://developer.nvidia.com/nccl and in NGC containers
Source code : https://github.com/nvidia/nccl
Perf tests : https://github.com/nvidia/nccl-tests

NCCL-API (with MPI) - Initialization

```
MPI_Init(&argc,&argv)
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

ncclUniqueId nccl_uid;

if (rank == 0) ncclGetUniqueId(&nccl_uid);

MPI_Bcast(&nccl_uid, sizeof(ncclUniqueId), MPI_BYTE, 0, MPI_COMM_WORLD));

ncclComm_t nccl_comm;
ncclCommInitRank(&nccl_comm, size, nccl_uid, rank);

…

ncclCommDestroy(nccl_comm);
MPI_Finalize();
```

Communication Calls

- Send/Recv

```
ncclSend(void* sbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
ncclRecv(void* rbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
```

- Collective Operations

```
    ncclAllReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op,           ncclComm_t comm, cudaStream_t stream);
    ncclBroadcast(void* sbuff, void* rbuff, size_t count, ncclDataType_t type,                int root, ncclComm_t comm, cudaStream_t stream);
       ncclReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op, int root, ncclComm_t comm, cudaStream_t stream);
ncclReduceScatter(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op,           ncclComm_t comm, cudaStream_t stream);
    ncclAllGather(void* sbuff, void* rbuff, size_t count, ncclDataType_t type,                           ncclComm_t comm, cudaStream_t stream);
```

Fused Communication Calls

- Multiple calls to `ncclSend()` and `ncclRecv()` should be fused with `ncclGroupStart()` and `ncclGroupEnd()` to
  - Avoid deadlocks
    (if calls need to progress concurrently)
  - For more performance
    (can be more efficiently)

SendRecv:

```
ncclGroupStart();
 ncclSend(sendbuff, sendcount, sendtype, peer, comm, stream);
 ncclRecv(recvbuff, recvcount, recvtype, peer, comm, stream);
ncclGroupEnd();
```

Bcast:

```
ncclGroupStart();
if (rank == root) {
  for (int r=0; r<nranks; r++)
    ncclSend(sendbuff[r], size, type, r, comm, stream);}
ncclRecv(recvbuff, size, type, root, comm, stream);
ncclGroupEnd();
```

Neighbor exchange:

```
ncclGroupStart();
for (int d=0; d<ndims; d++) {
  ncclSend(sendbuff[d], sendcount, sendtype, next[d], comm, stream);
  ncclRecv(recvbuff[d], recvcount, recvtype, prev[d], comm, stream);}
ncclGroupEnd();
```

Jacobi using NCCL

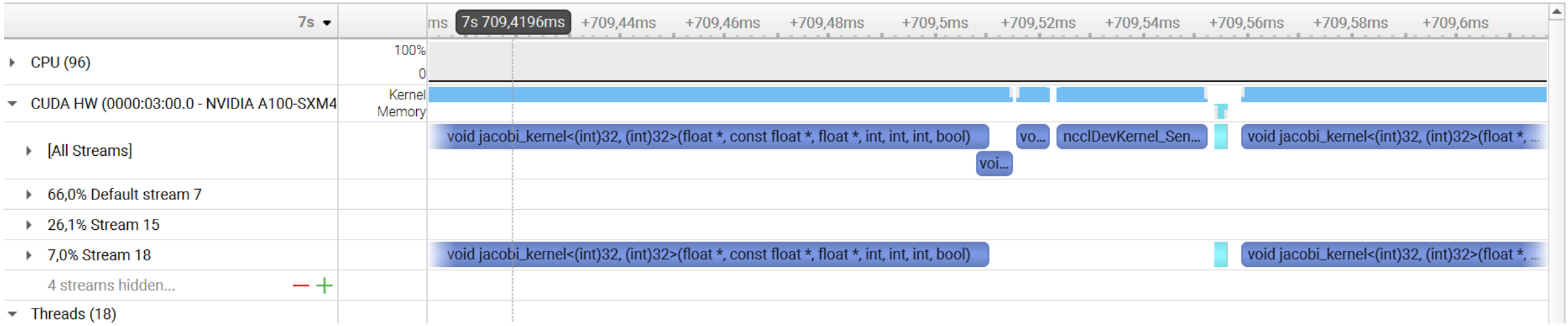
```
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, iy_end, nx, compute_stream);

ncclGroupStart();
ncclRecv(a_new,                     nx, NCCL_REAL_TYPE, top,    nccl_comm, compute_stream)
ncclSend(a_new + (iy_end - 1) * nx, nx, NCCL_REAL_TYPE, bottom, nccl_comm, compute_stream);
ncclRecv(a_new + (iy_end * nx),     nx, NCCL_REAL_TYPE, bottom, nccl_comm, compute_stream);
ncclSend(a_new + iy_start * nx,     nx, NCCL_REAL_TYPE, top,    nccl_comm, compute_stream);
ncclGroupEnd();
```

So far, no Overlap of communication and computation!

Overlapping communication with Computation -> First Try

```
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, (iy_start + 1), nx, push_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1), iy_end, nx,     push_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start, iy_end, nx, compute_stream);

ncclGroupStart();
ncclRecv(a_new,                     nx, NCCL_REAL_TYPE, top,    nccl_comm, push_stream)
ncclSend(a_new + (iy_end - 1) * nx, nx, NCCL_REAL_TYPE, bottom, nccl_comm, push_stream);
ncclRecv(a_new + (iy_end * nx),     nx, NCCL_REAL_TYPE, bottom, nccl_comm, push_stream);
ncclSend(a_new + iy_start * nx,     nx, NCCL_REAL_TYPE, top,    nccl_comm, push_stream)
ncclGroupEnd();
```

Analysis

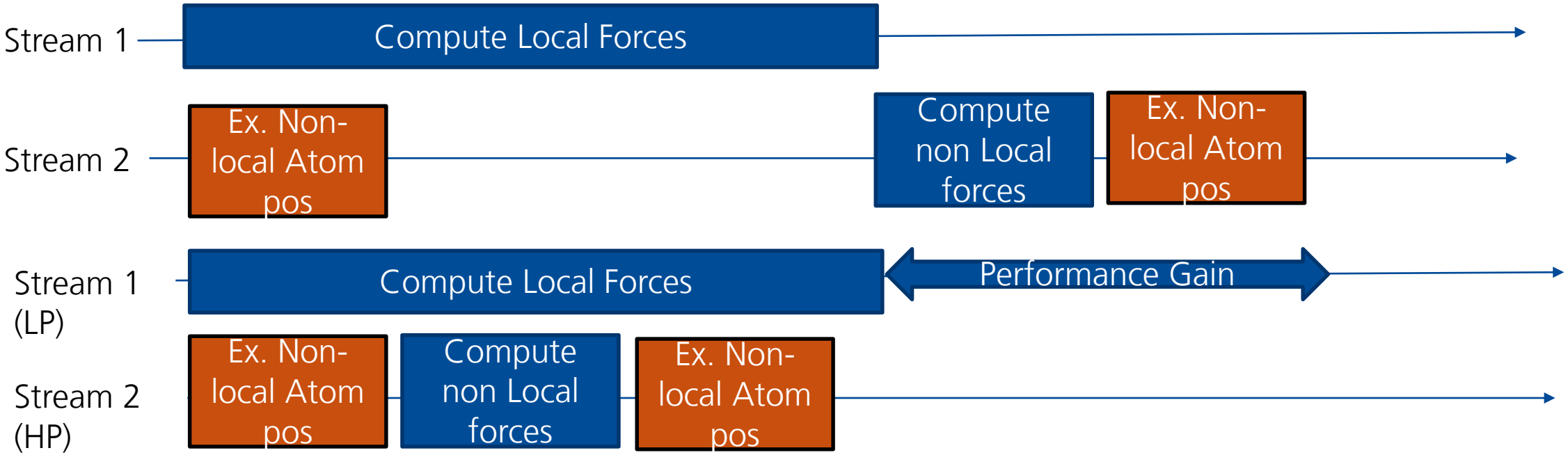

7s
ms
7s 709,4196ms
+709,44ms
+709,46ms
+709,48ms
+709,5ms
+709,52ms
+709,54ms
+709,56ms
+709,58ms
+709,6ms
CPU (96)
100%
0
CUDA HW (0000:03:00.0 - NVIDIA A100-SXM4
Kernel
Memory
[All Streams]
void jacobi_kernel<(int)32, (int)32>(float *, const float *, float *, int, int, int, bool)
vo...
ncclDevKernel_Sen...
void jacobi_kernel<(int)32, (int)32>(float *, ...
voi...
66,0% Default stream 7
26,1% Stream 15
7,0% Stream 18
void jacobi_kernel<(int)32, (int)32>(float *, const float *, float *, int, int, int, bool)
void jacobi_kernel<(int)32, (int)32>(float *, ...
4 streams hidden...
Threads (18)

High Priority Streams

Improve scalability with high priority streams (available on CC 3.5+)

```
cudaStreamCreateWithPriority ( cudaStream_t* pStream, unsigned int flags, int priority )
```

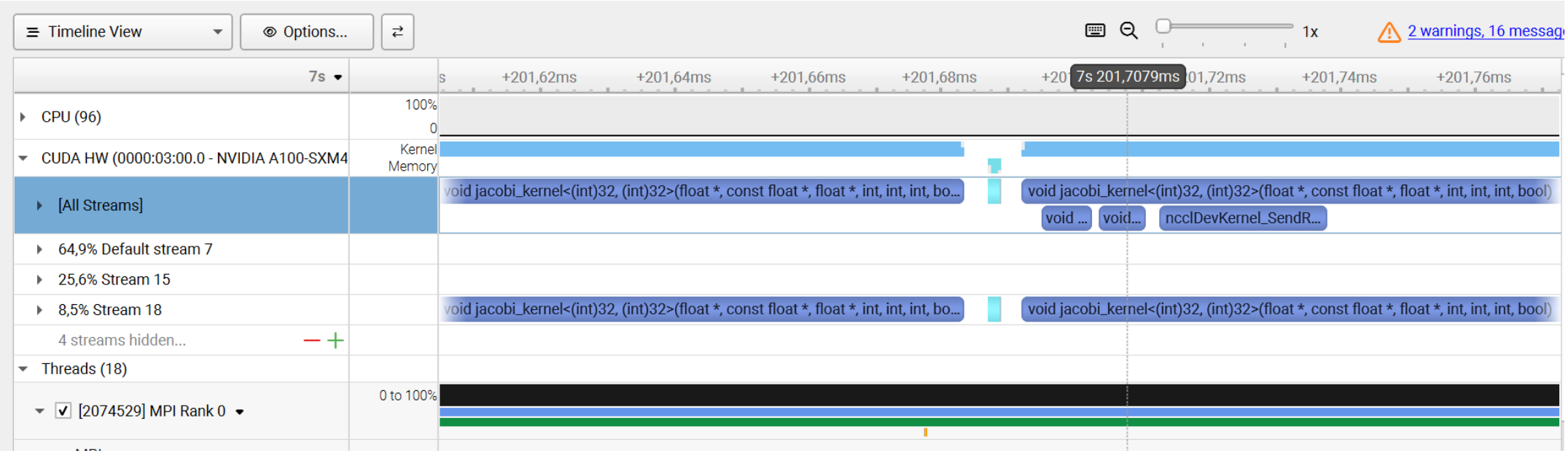
Motivating Example: MD- Simulations



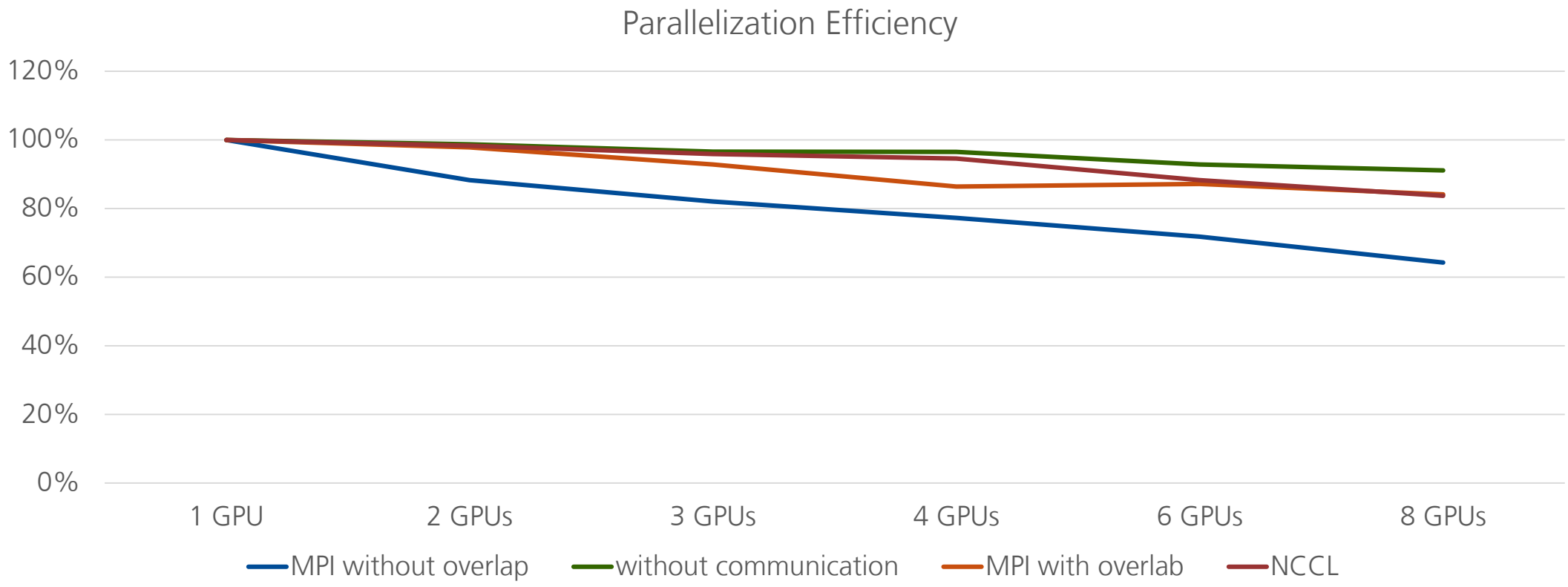
Create Priority Streams

```
int leastPriority = 0;
int greatestPriority = leastPriority;
cudaDeviceGetStreamPriorityRange(&leastPriority, &greatestPriority);

cudaStream_t push_stream, compute_stream;

cudaStreamCreateWithPriority(&compute_stream, cudaStreamDefault, leastPriority));
cudaStreamCreateWithPriority(&push_stream, cudaStreamDefault, greatestPriority);
```

Overlapping with Priority Streams



Performance on Jupiter



Summary

- Asynchronously computing on the GPU while MPI communication allows to hide MPI communication times
- NCCL supports CUDA stream aware communication
- NCCL allows to issue communication request asynchronous with respect to the CPU-thread, but synchronous to CUDA streams
- High priority streams are required to overlap communication and computation