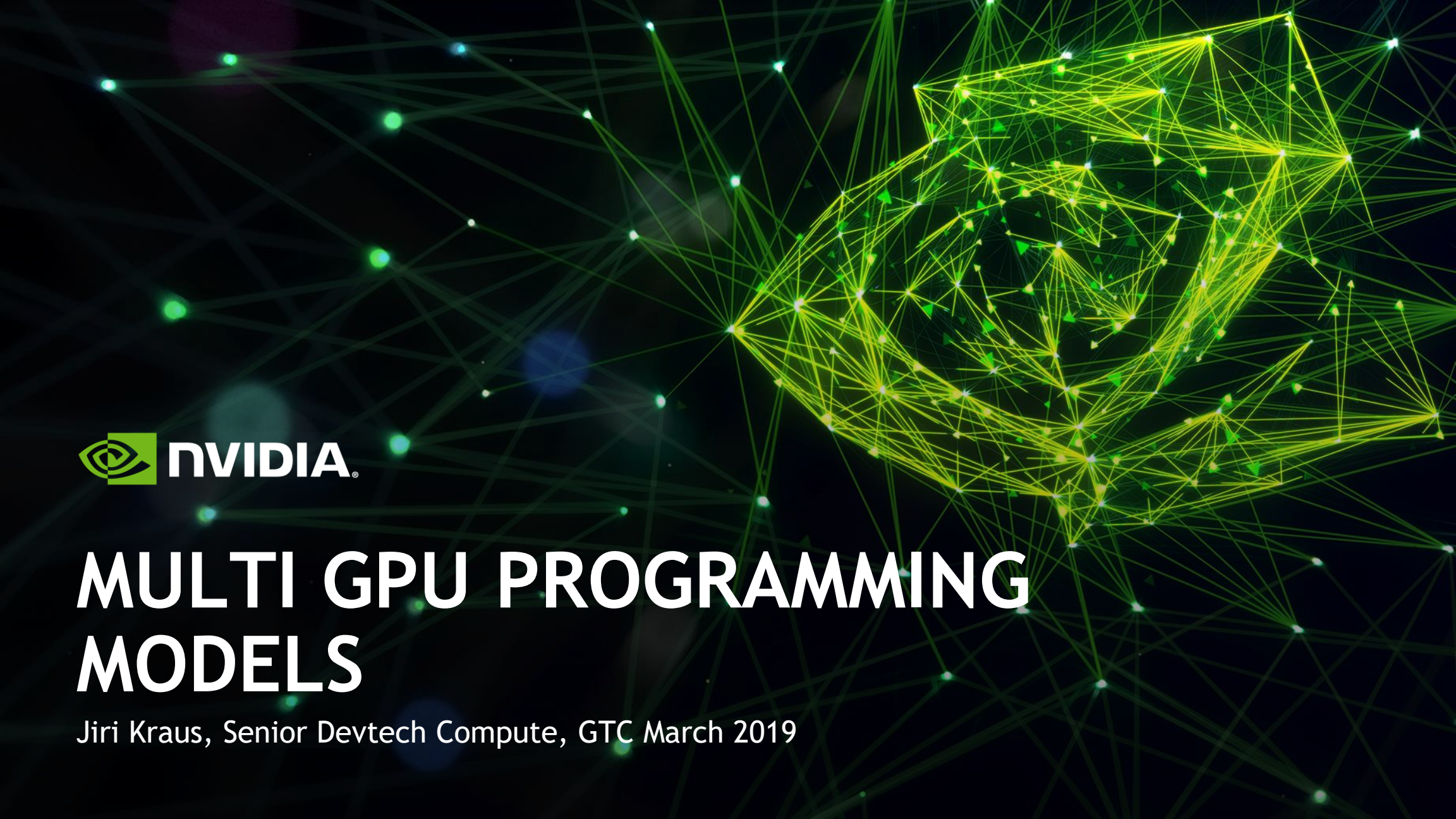


MULTI GPU PROGRAMMING MODELS

Jiri Kraus, Senior Devtech Compute, GTC March 2019

MOTIVATION

Why use multiple GPUs?

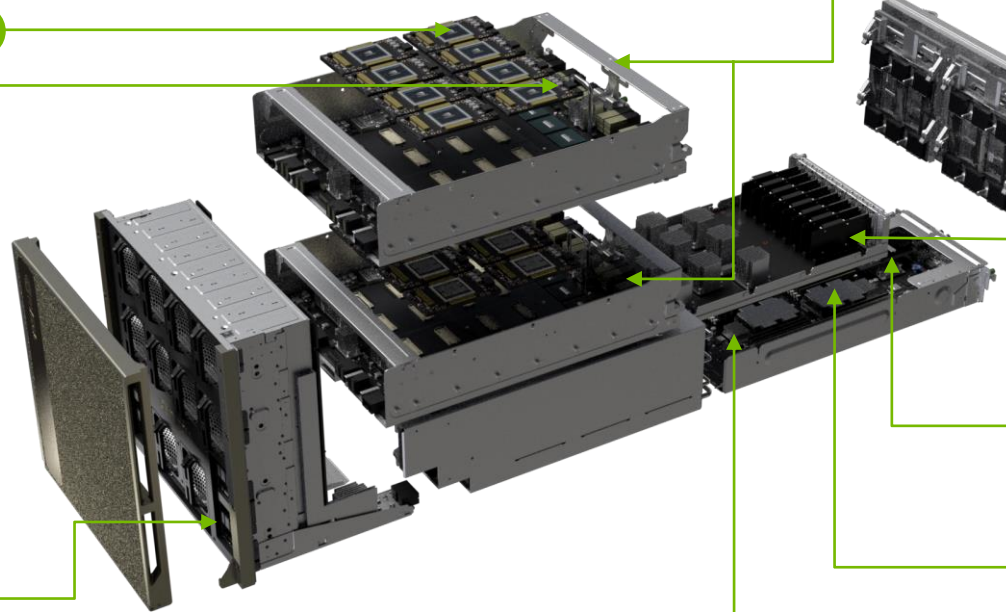
Need to compute larger, e.g. bigger networks, car models, …

Need to compute faster, e.g. weather prediction

Better energy efficiency with dense nodes with multiple GPUs

DESIGNED TO TRAIN THE PREVIOUSLY IMPOSSIBLE

NVIDIA DGX-2



1 **NVIDIA Tesla V100 32GB**

2 **Two GPU Boards**
8 V100 32GB GPUs per board
6 NVSwitches per board
512GB Total HBM2 Memory
interconnected by
Plane Card

3 **Twelve NVSwitches**
2.4 TB/sec bi-section
bandwidth

4 **Eight EDR Infiniband/100 GigE**
1600 Gb/sec Total
Bi-directional Bandwidth

5 **Two Intel Xeon Platinum CPUs**

6 **1.5 TB System Memory**

7 **30 TB NVME SSDs**
Internal Storage

8 **Two High-Speed Ethernet**
10/25/40/100 GigE

EXAMPLE: JACOBI SOLVER

Solves the 2D-Laplace Equation on a rectangle

$$\Delta \boldsymbol{u}(\boldsymbol{x}, \boldsymbol{y}) = \mathbf{0} \; \forall \; (\boldsymbol{x}, \boldsymbol{y}) \in \Omega \backslash \boldsymbol{\delta}\Omega$$

Dirichlet boundary conditions (constant values on boundaries) on left and right boundary

Periodic boundary conditions on top and bottom boundary

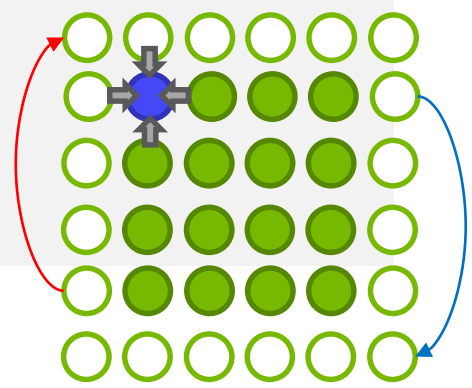
EXAMPLE: JACOBI SOLVER

Single GPU

While not converged

Do Jacobi step:

```
for( int iy = 1; iy < ny-1; iy++ )
for( int ix = 1; ix < nx-1; ix++ )
    a_new[iy*nx+ix] = -0.25 *
                      -( a[ iy    *nx+(ix+1)] + a[ iy    *nx+ix-1]
                       + a[(iy-1)*nx+ ix    ] + a[(iy+1)*nx+ix   ] );
```



Apply periodic boundary conditions

Swap `a_new` and `a`

Next iteration

DOMAIN DECOMPOSITION

Different Ways to split the work between processes:

Minimize number of neighbors:

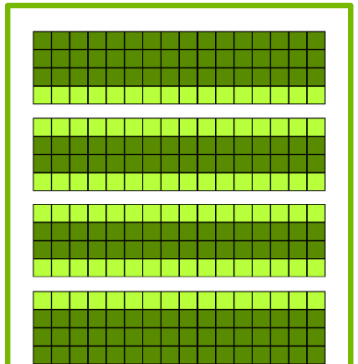
Communicate to less neighbors

Optimal for latency bound communication

Minimize surface area/volume ratio:

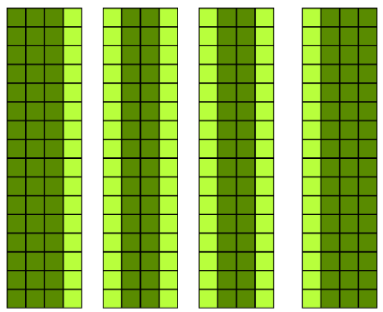
Communicate less data

Optimal for bandwidth bound communication



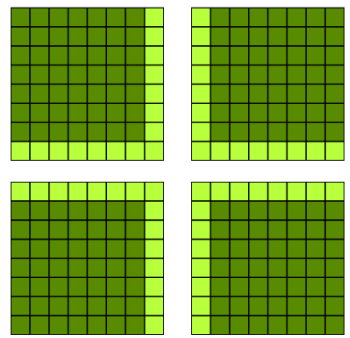
Horizontal Stripes

Contiguous if data is row-major



Vertical Stripes

Contiguous if data is column-major



Tiles

EXAMPLE: JACOBI SOLVER

Multi GPU

While not converged

Do Jacobi step:

```
for (int iy = iy_start; iy < iy_end; iy++ )
for( int ix = 1; ix < nx-1; ix++ )
    a_new[iy*nx+ix] = -0.25 *
                      -( a[ iy    *nx+(ix+1)] + a[ iy    *nx+ix-1]
                       + a[(iy-1)*nx+ ix    ] + a[(iy+1)*nx+ix   ] );
```

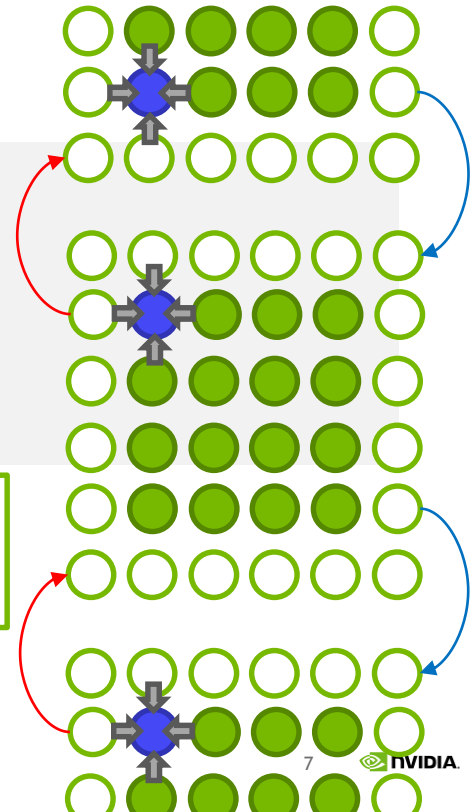
Apply periodic boundary conditions

Exchange halo with 2 neighbors

One-step with ring exchange

Swap `a_new` and `a`

Next iteration



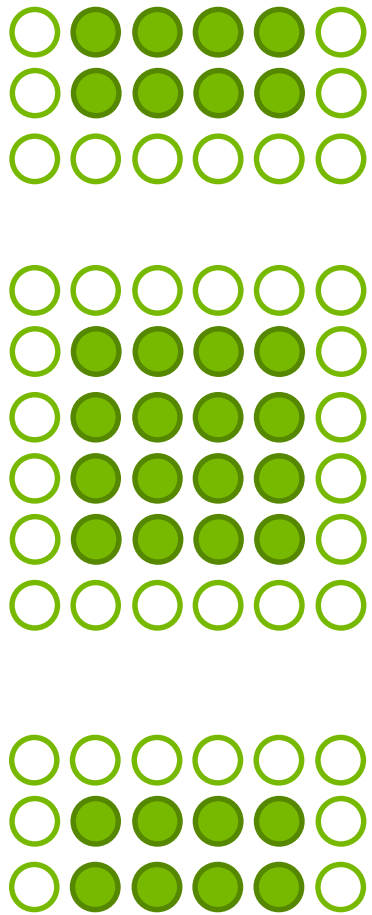
SINGLE THREADED MULTI GPU PROGRAMMING

```
while ( l2_norm > tol && iter < iter_max ) {
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1); const int bottom = (dev_id+1)%num_devices;
        cudaSetDevice( dev_id );
        cudaMemsetAsync(l2_norm_d[dev_id], 0 , sizeof(real) );
        jacobi_kernel<<<dim_grid,dim_block>>>( a_new[dev_id], a[dev_id], l2_norm_d[dev_id],
                                               iy_start[dev_id], iy_end[dev_id], nx );
        cudaMemcpyAsync( l2_norm_h[dev_id], l2_norm_d[dev_id], sizeof(real), cudaMemcpyDeviceToHost );
        cudaMemcpyAsync( a_new[top]+(iy_end[top]*nx), a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
        cudaMemcpyAsync( a_new[bottom], a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
    }
    l2_norm = 0.0;
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
        cudaSetDevice( dev_id ); cudaDeviceSynchronize();
        l2_norm += *(l2_norm_h[dev_id]);
    }
    l2_norm = std::sqrt( l2_norm );
    for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) std::swap(a_new[dev_id],a[dev_id]);
    iter++;
}
```

EXAMPLE JACOBI

Top/Bottom Halo

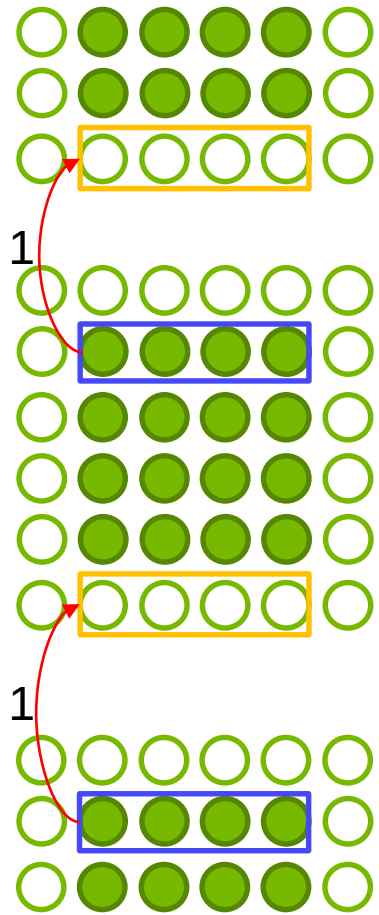
```
cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
```



EXAMPLE JACOBI

Top/Bottom Halo

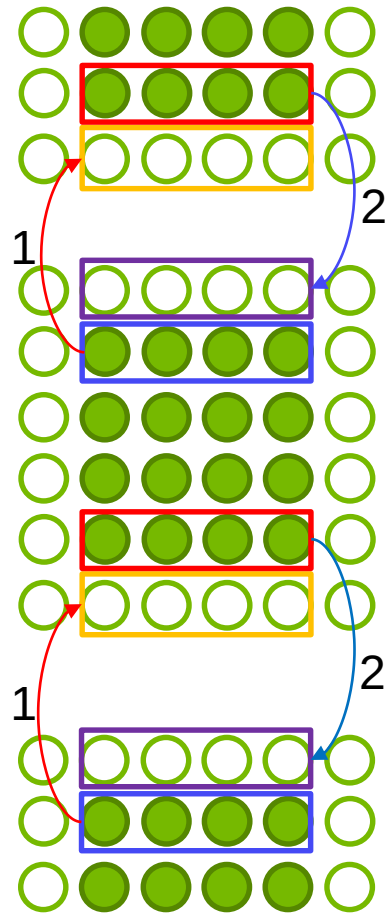
```
cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);
```



EXAMPLE JACOBI

Top/Bottom Halo

```
cudaMemcpyAsync(
  a_new[top]+(iy_end[top]*nx),
  a_new[dev_id]+iy_start[dev_id]*nx, nx*sizeof(real), ...);


cudaMemcpyAsync(
  a_new[bottom],
  a_new[dev_id]+(iy_end[dev_id]-1)*nx, nx*sizeof(real), ...);
```

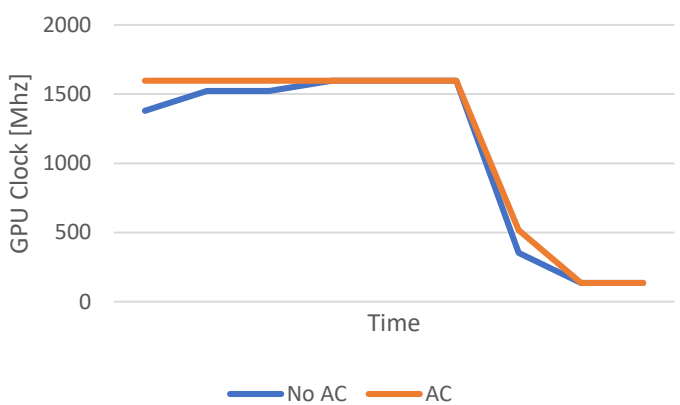


CONTROLLING GPU BOOST

using application clocks

Application can safely run at max clocks

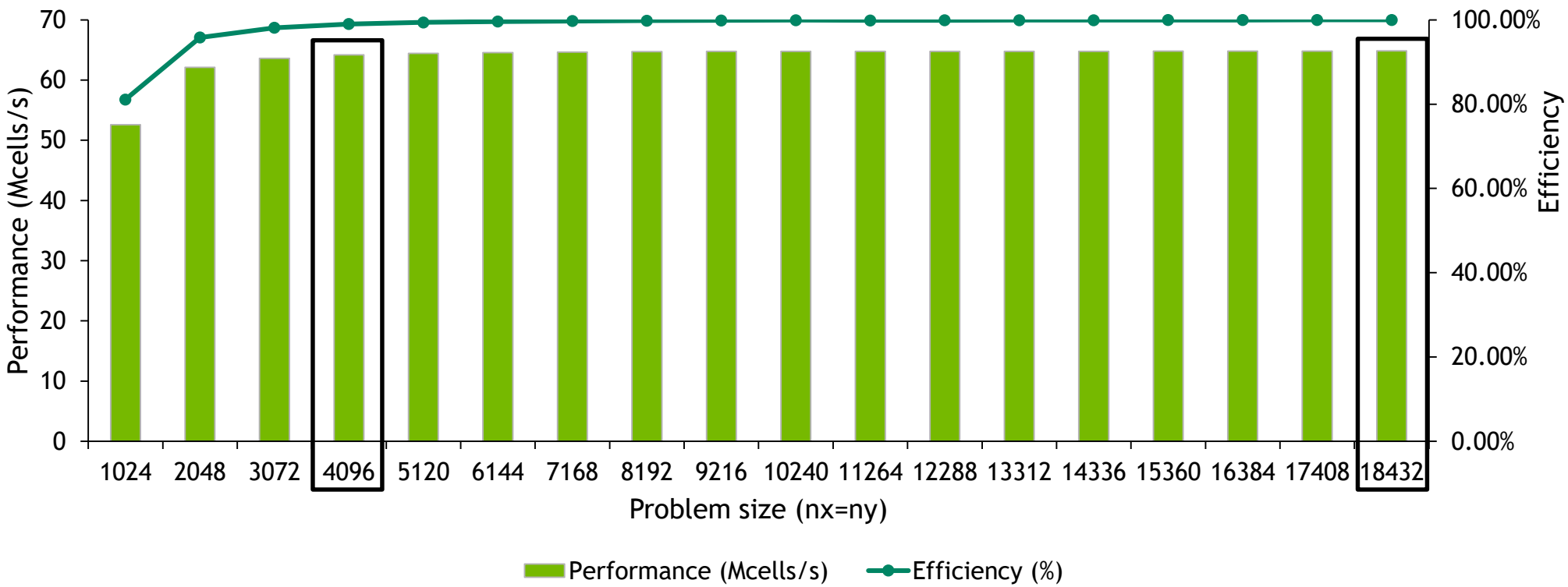
Short runtime of the benchmark makes spinning clocks up visible:




```
$ sudo nvidia-smi -ac 958,1597
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:34:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:36:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:39:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:3B:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:57:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:59:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:5C:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:5E:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:B7:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:B9:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:BC:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:BE:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:E0:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:E2:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:E5:00.0
Applications clocks set to "(MEM 958, SM 1597)" for GPU 00000000:E7:00.0
All done.
```

EXAMPLE: JACOBI SOLVER

Single GPU performance vs. problem size – Tesla V100 SXM3 32 GB



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, Reported Performance is the minimum of 5 repetitions

SCALABILTY METRICS FOR SUCCESS

Serial Time: T_s: How long it takes to run the problem with a single GPU

Parallel Time: T_p: How long it takes to run the problem with multiple GPUs

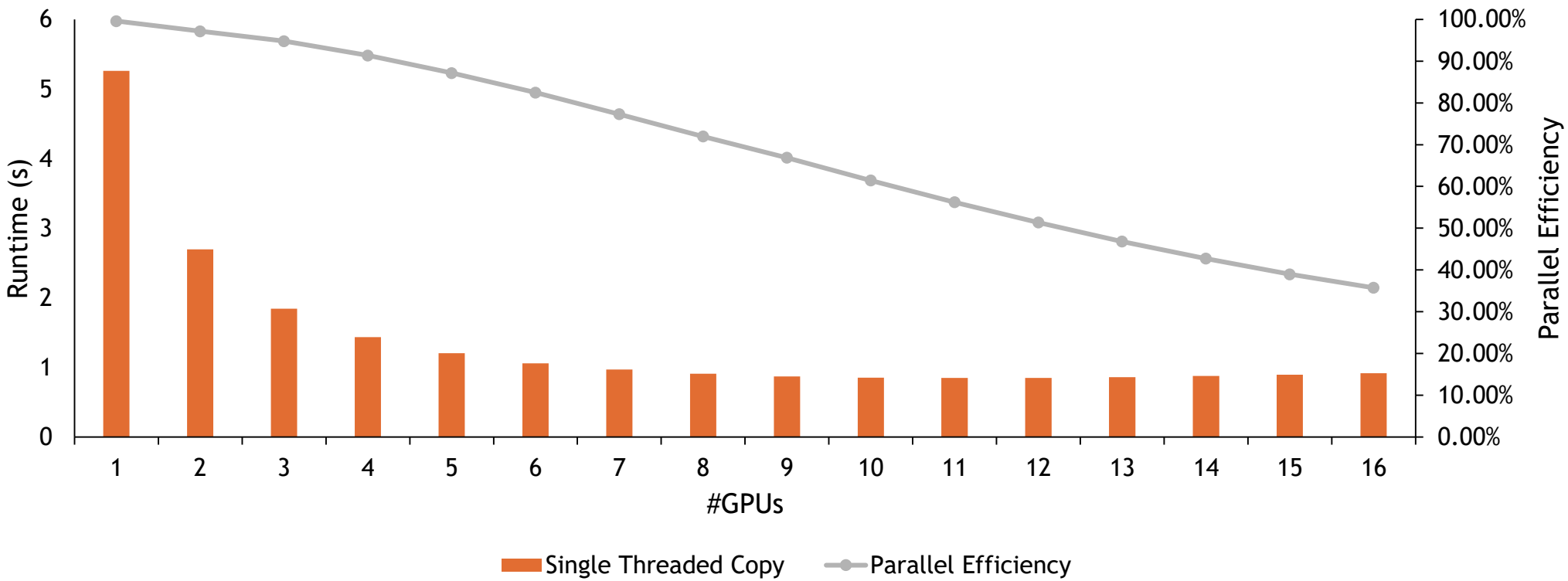
Number of GPU: P: The number of GPUs operating on the task at hand

Speedup: $S = \frac{T_s}{T_p}$: How much faster is the parallel version vs. serial. (optimal is P)

Efficiency: $E = \frac{S}{P}$: How efficient are the GPUs used (optimal is 1)

MULTI GPU JACOBI RUNTIME

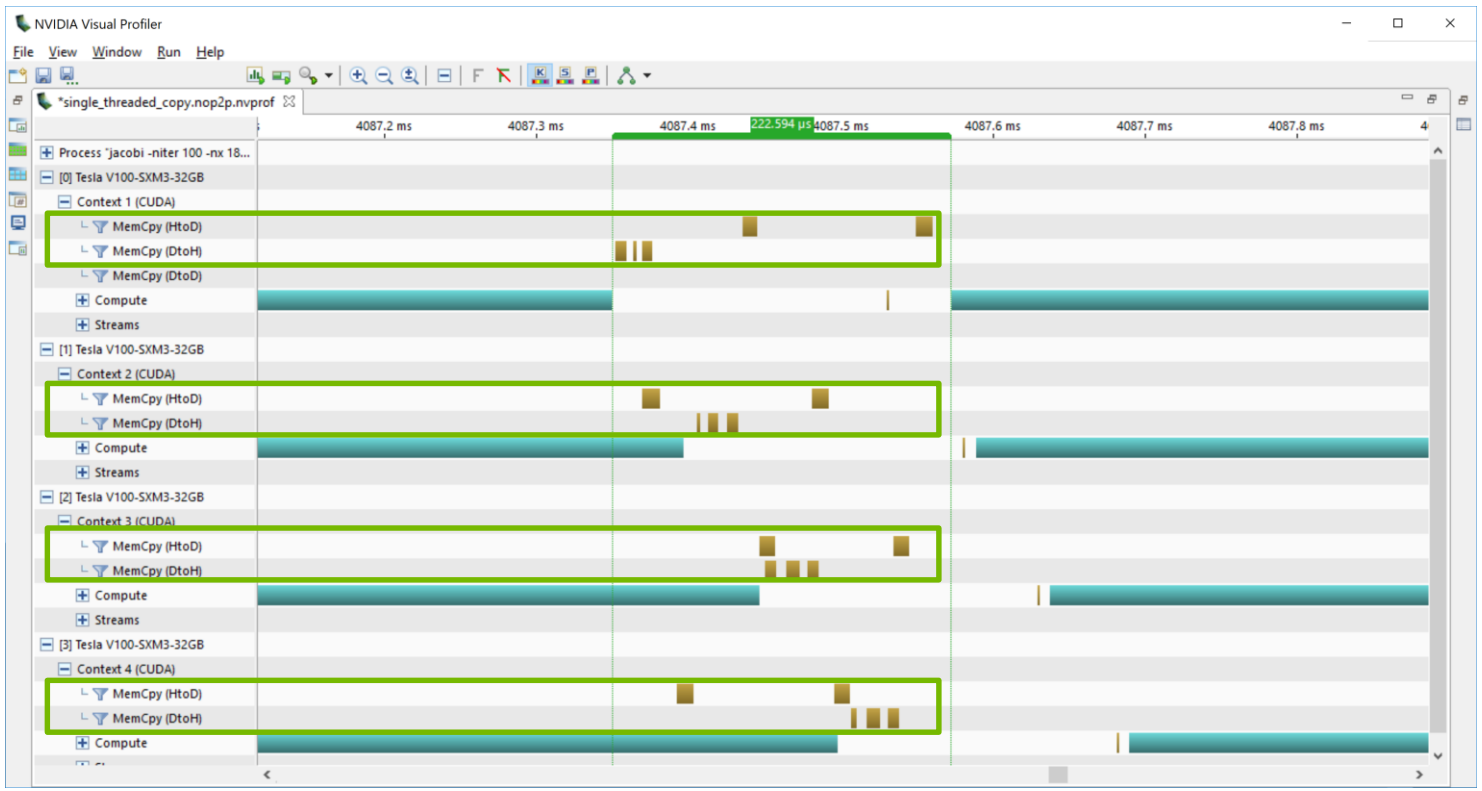
DGX-2 - 18432 x 18432, 1000 iterations



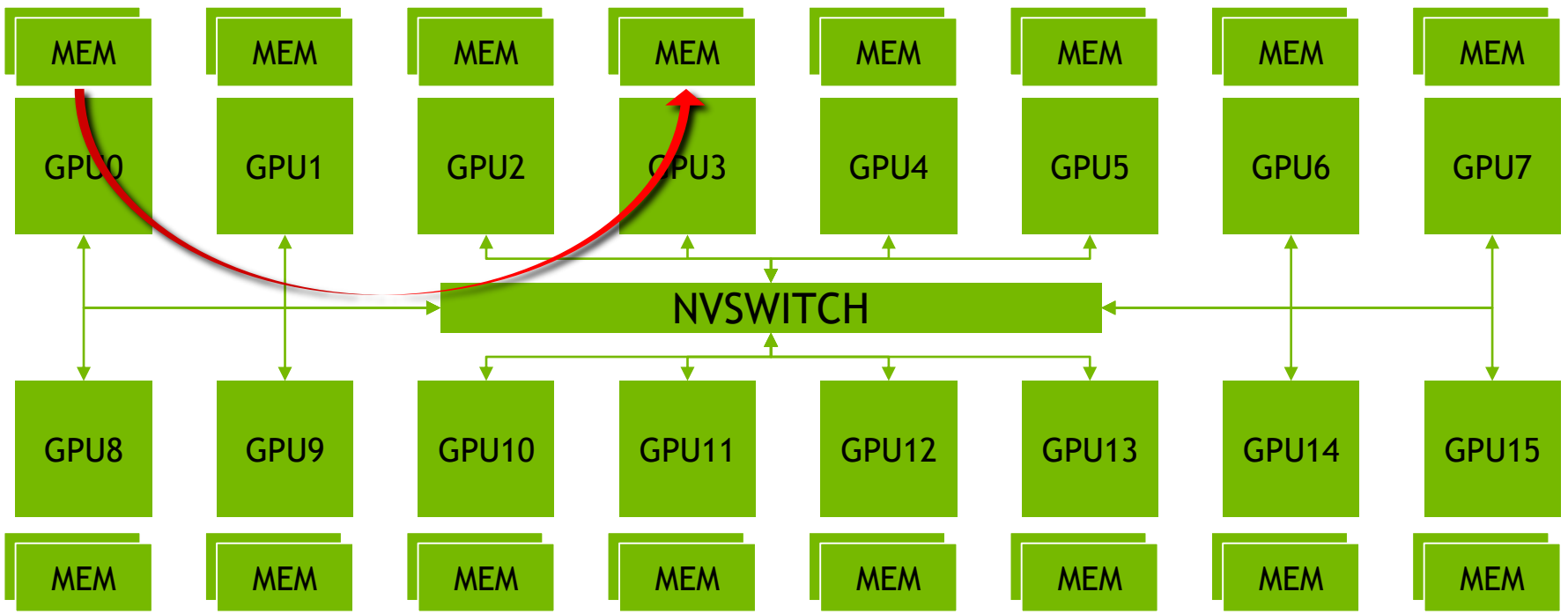
Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MULTI GPU JACOBI NVVP TIMELINE

Single Threaded Copy 4 V100 on DGX-2



GPUDIRECT P2P



Maximizes intra node inter GPU Bandwidth

Avoids Host memory and system topology bottlenecks

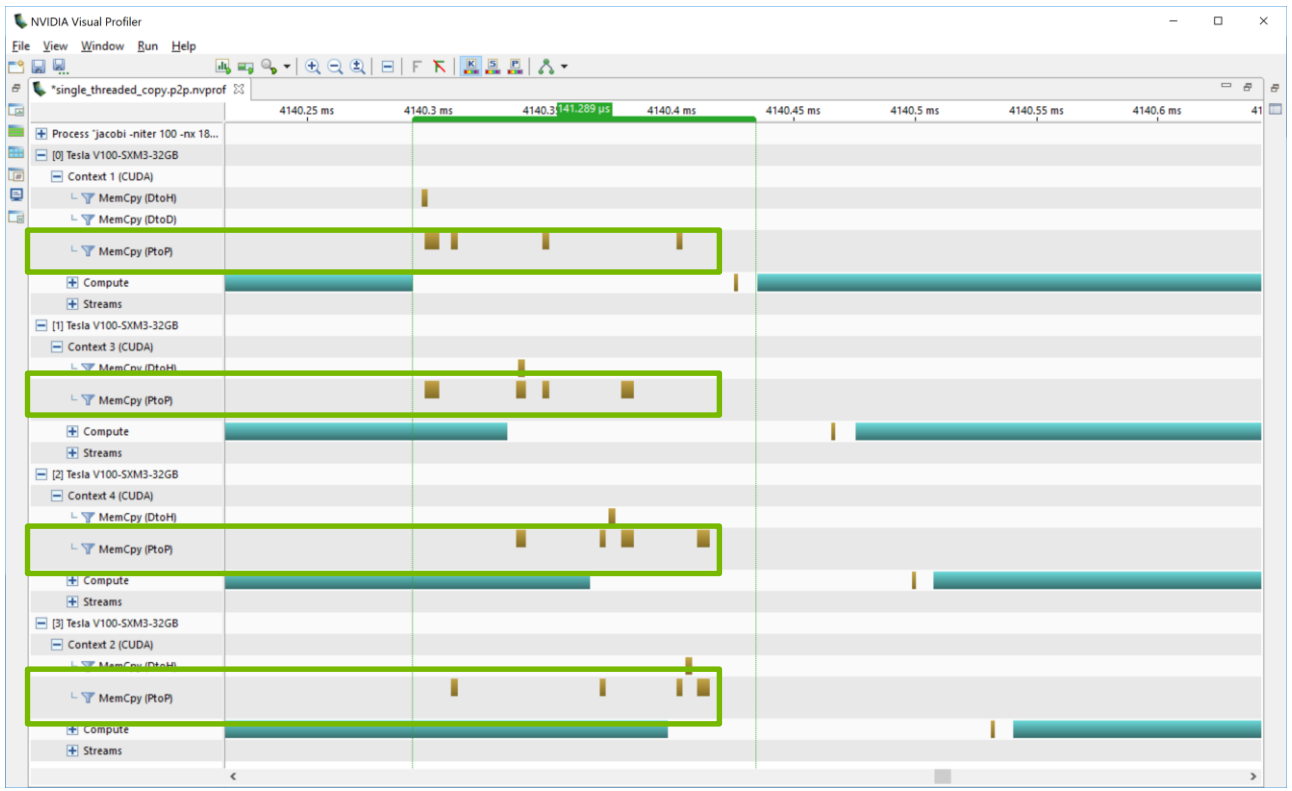
GPUDIRECT P2P

Enable P2P

```
for ( int dev_id = 0; dev_id < num_devices; ++dev_id ) {
    cudaSetDevice( dev_id );
    const int top = dev_id > 0 ? dev_id - 1 : (num_devices-1);
    int canAccessPeer = 0;
    cudaDeviceCanAccessPeer ( &canAccessPeer, dev_id, top );
    if ( canAccessPeer )
        cudaDeviceEnablePeerAccess ( top, 0 );
    const int bottom = (dev_id+1)%num_devices;
    if ( top != bottom ) {
        cudaDeviceCanAccessPeer ( &canAccessPeer, dev_id, bottom );
        if ( canAccessPeer )
           cudaDeviceEnablePeerAccess ( bottom, 0 );
    }
}
```

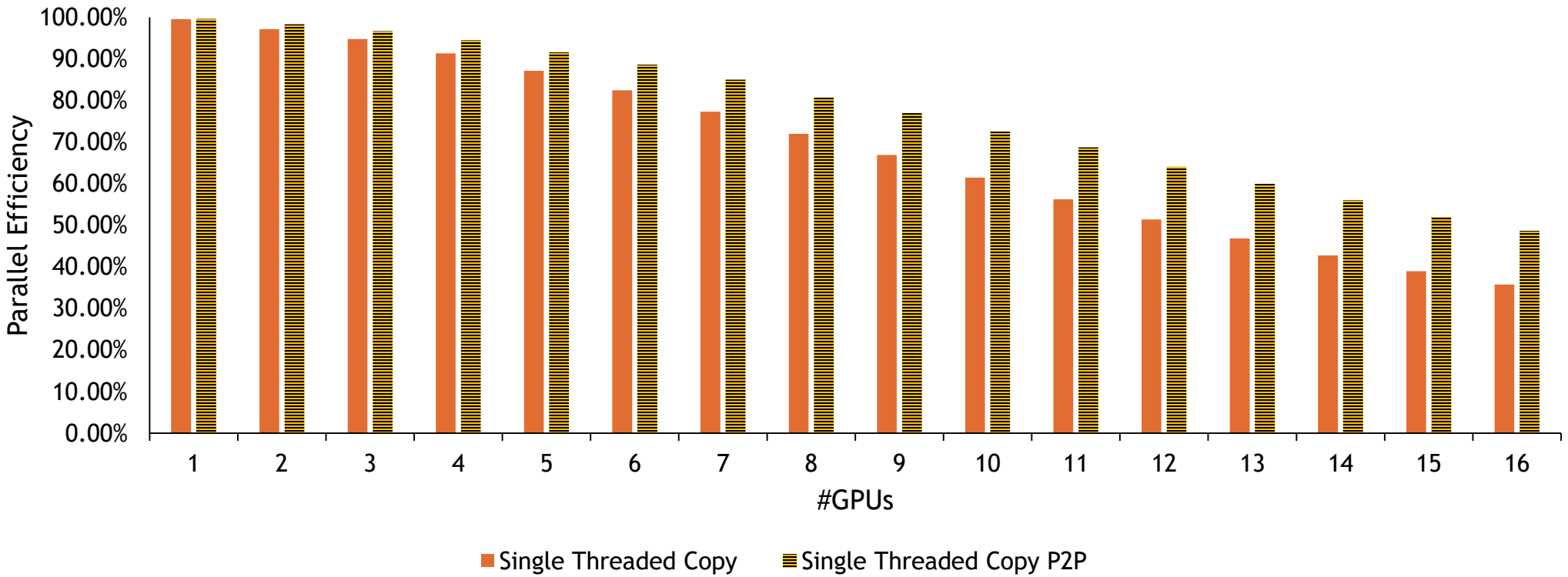
MULTI GPU JACOBI NVVP TIMELINE

Single Threaded Copy 4 V100 on DGX-2 with P2P



MULTI GPU JACOBI RUNTIME

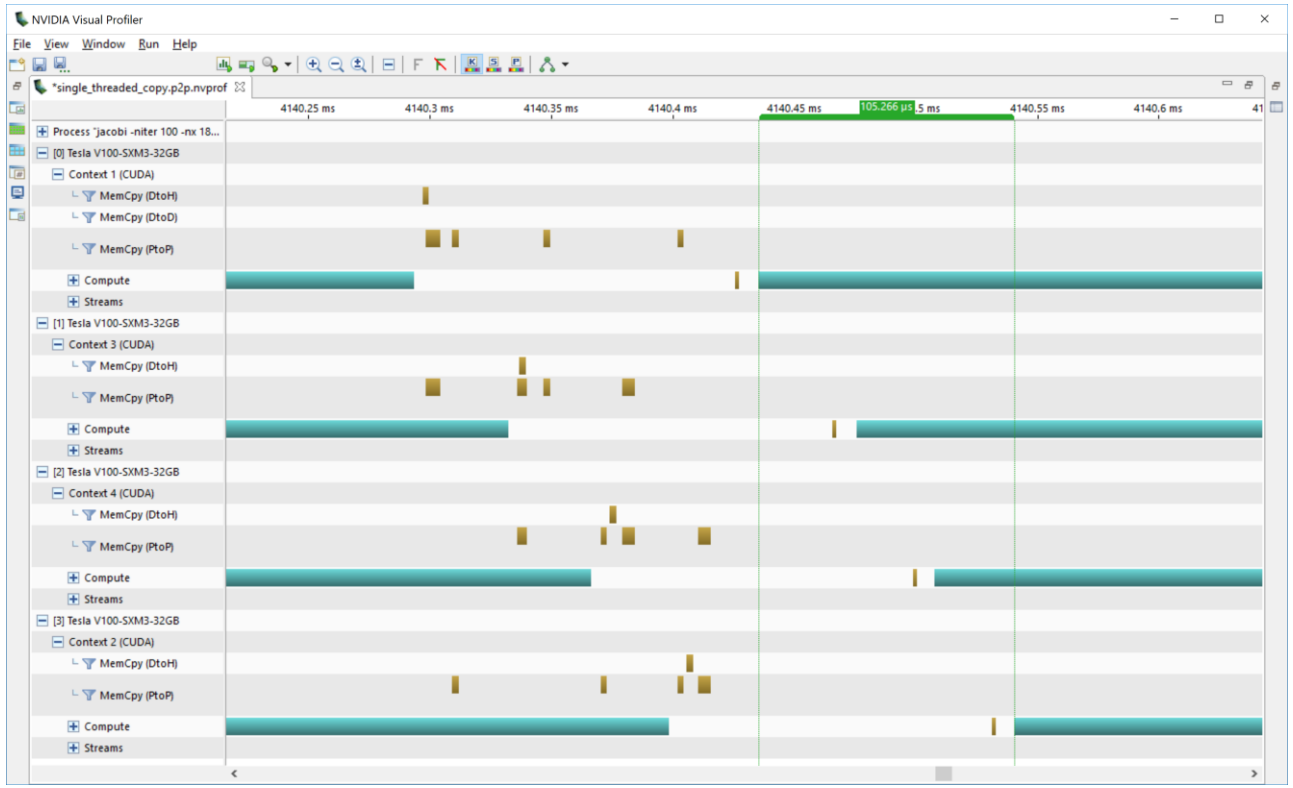
DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MULTI GPU JACOBI NVVP TIMELINE

Single Threaded Copy 4 V100 on DGX-2 with P2P



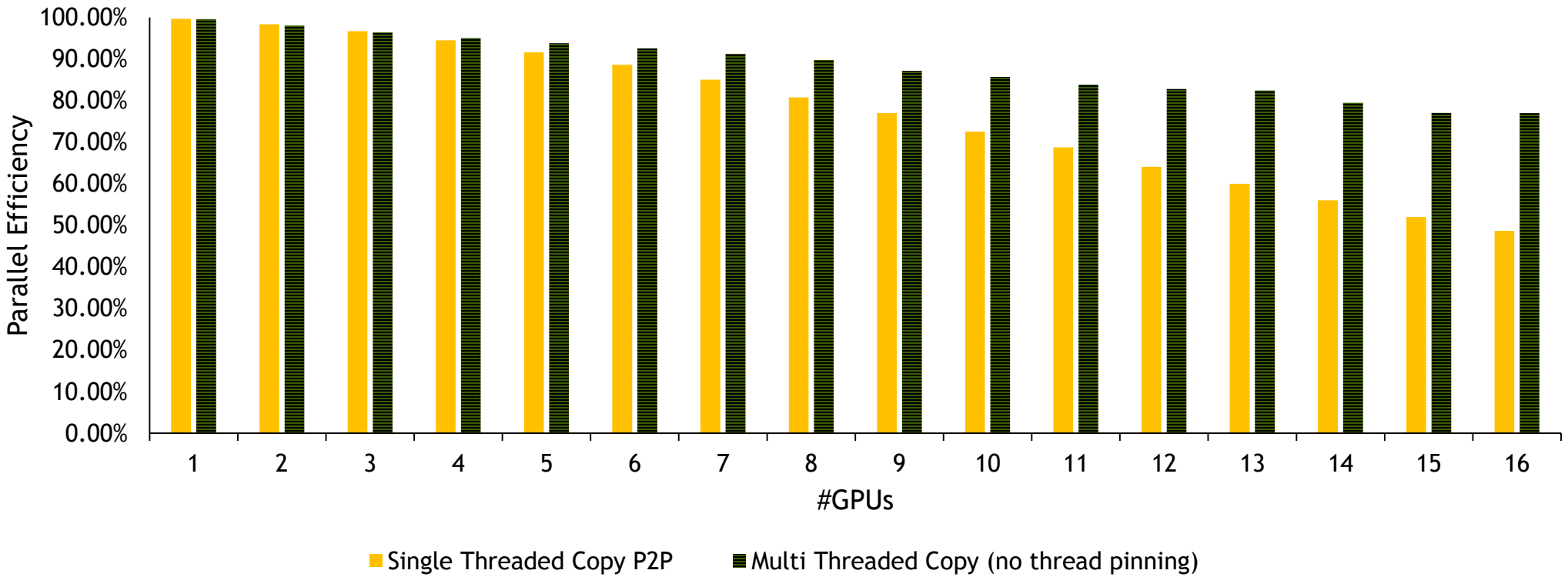
MULTI THREADED MULTI GPU PROGRAMMING

Using OpenMP

```
int num_devices = 0;

cudaGetDeviceCount( &num_devices );

#pragma omp parallel num_threads( num_devices )

{

    int dev_id = omp_get_thread_num();

    cudaSetDevice( dev_id );

}
```

MULTI GPU JACOBI RUNTIME

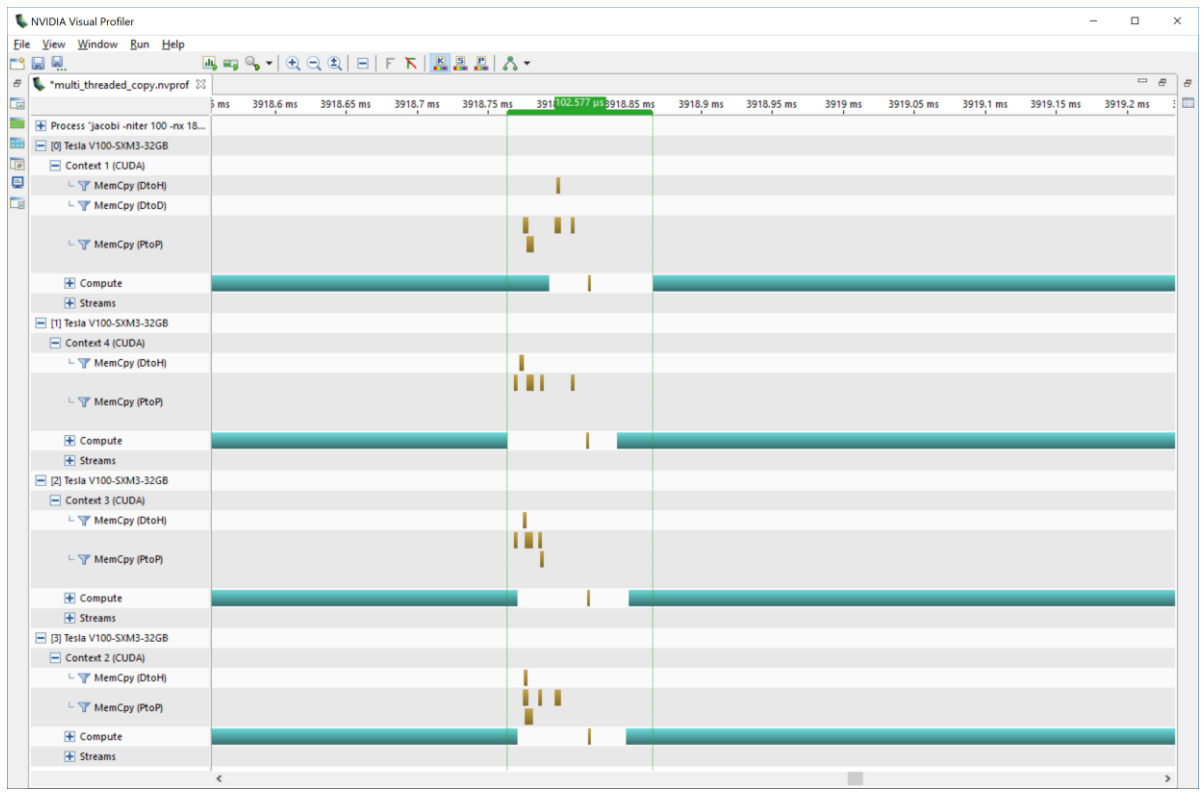
DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded Copy 4 V100 on DGX-2 with P2P



NVIDIA

GPU/CPU AFFINITY

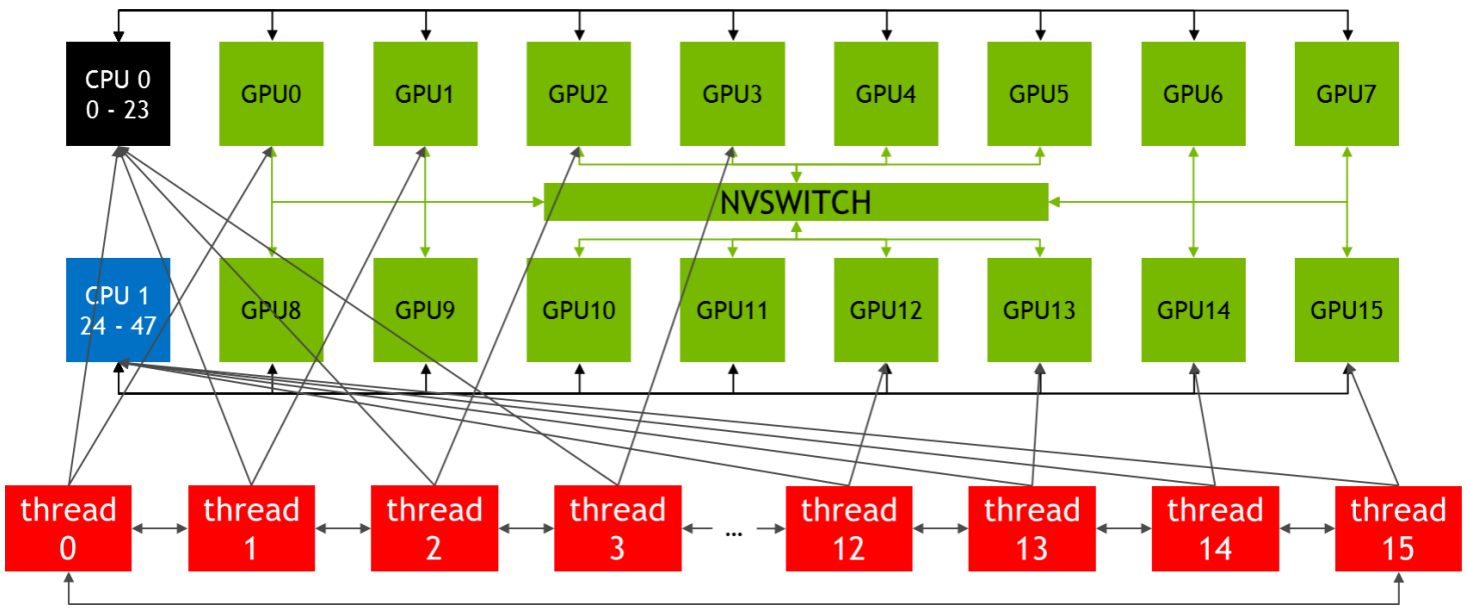
Querying system topology with `nvidia-smi topo -m`

```
$ nvidia-smi topo -m
```

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	GPU8	GPU9	GPU10	GPU11	GPU12	GPU13	GPU14	GPU15	...	CPU Affinity	
GPU0	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	CPU 0
GPU1	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU2	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU3	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU4	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU5	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU7	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	0-23,48-71	
GPU8	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	NV6	...	24-47,72-95	CPU 1
GPU9	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	NV6	...	24-47,72-95	
GPU10	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	NV6	...	24-47,72-95	
GPU11	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	NV6	...	24-47,72-95	
GPU12	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	NV6	...	24-47,72-95	
GPU13	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	NV6	...	24-47,72-95	
GPU14	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	NV6	...	24-47,72-95	
GPU15	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	NV6	X	...	24-47,72-95	

GPU/CPU AFFINITY

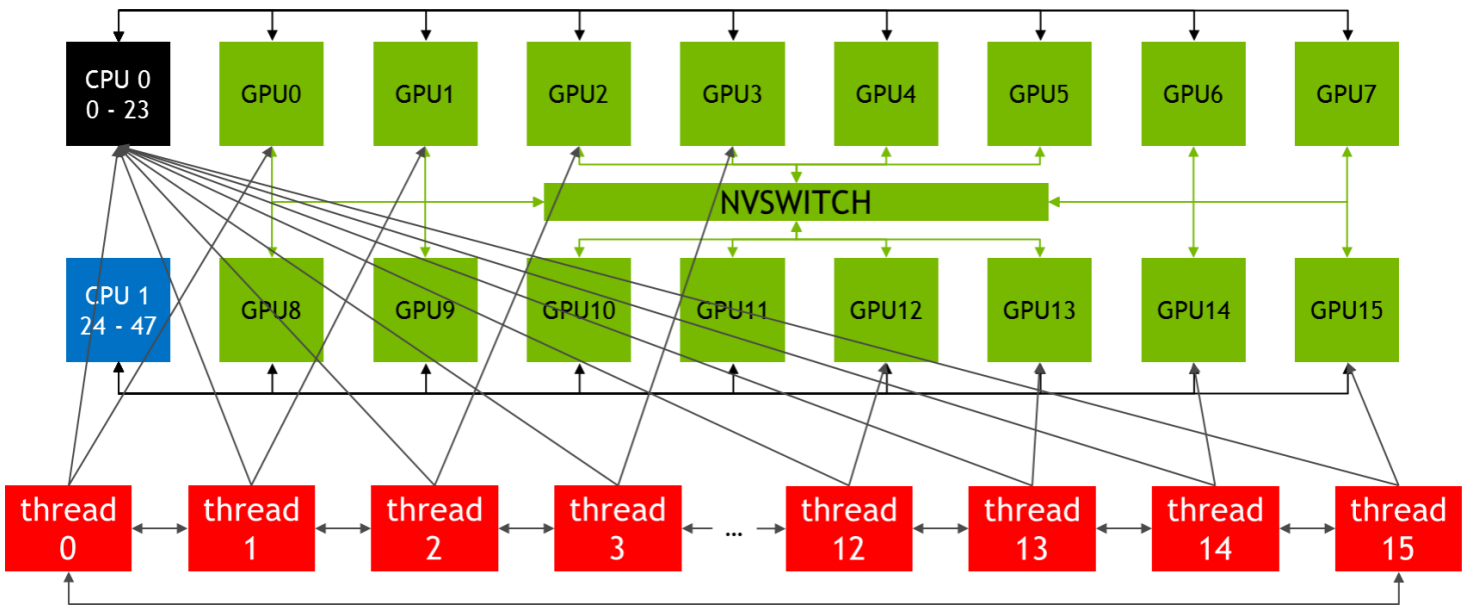
Using OpenMP env. vars.




```
export OMP_PROC_BIND=TRUE
export OMP_PLACES="{0},{1},{2},{3},{4},{5},{6},{7},{24},{25},{26},{27},…,{31}"
```

GPU/CPU AFFINITY

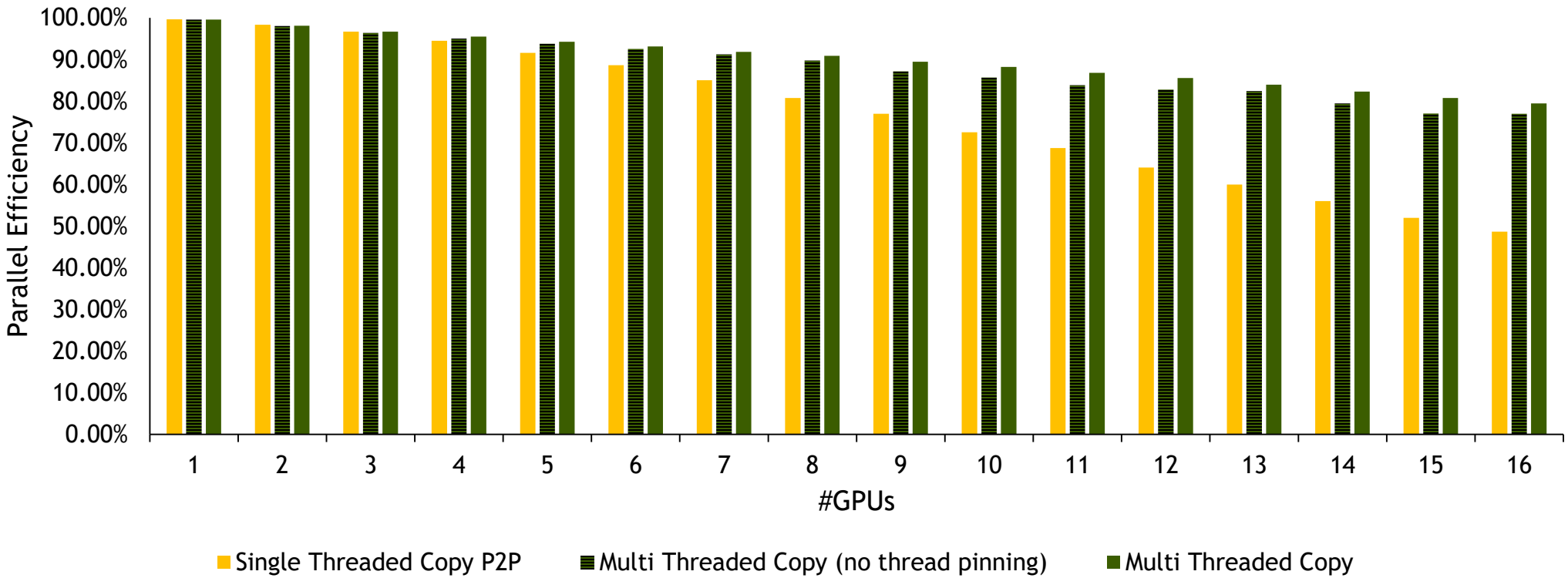
Using OpenMP env. vars.




```
export OMP_PROC_BIND=TRUE
export OMP_PLACES="{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10},{11},…,{15}"
```

MULTI GPU JACOBI RUNTIME

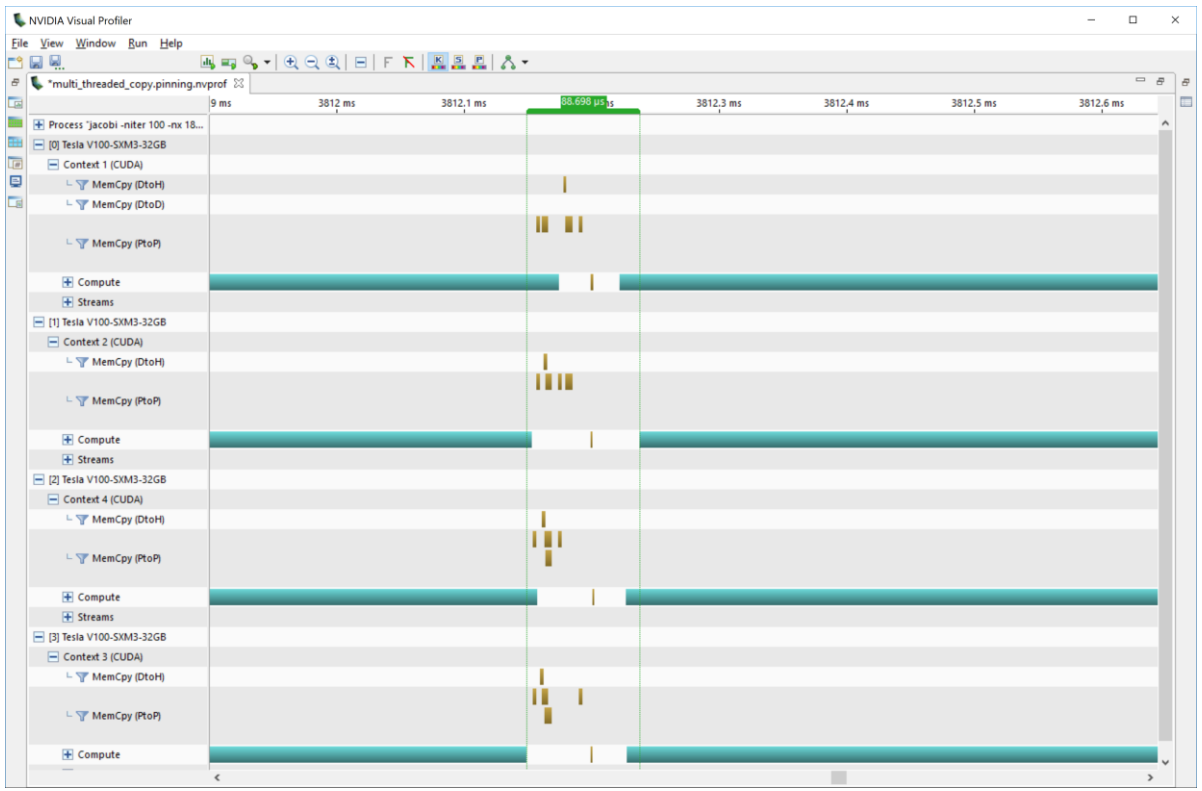
DGX-2 - 18432 x 18432, 1000 iterations



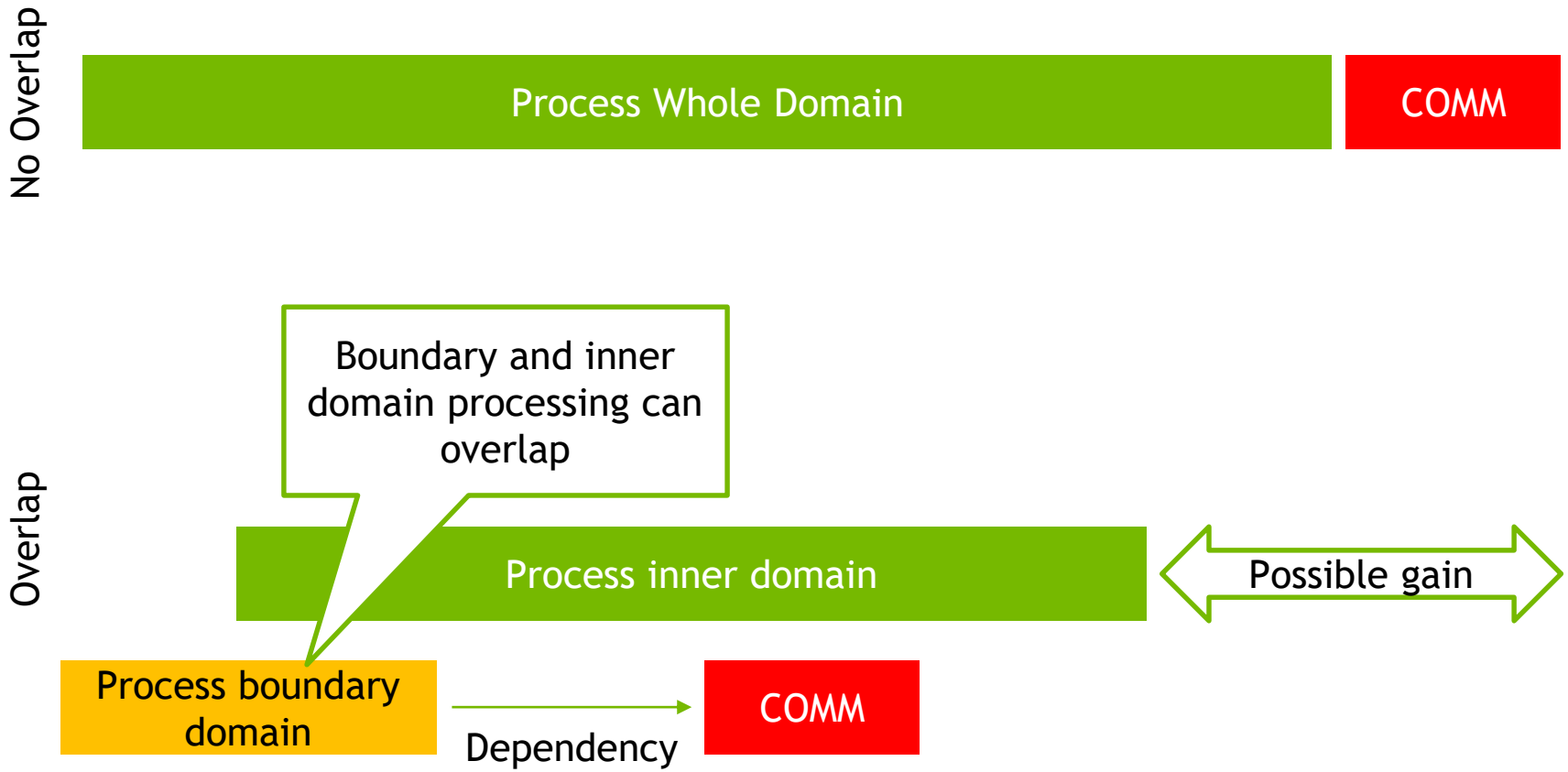
Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded Copy 4 V100 on DGX-2 with P2P and thread pinning



COMMUNICATION + COMPUTATION OVERLAP



COMMUNICATION + COMPUTATION OVERLAP

```
//Compute bulk
cudaStreamWaitEvent(compute_stream,push_top_done[(iter%2)][dev_id],0);
cudaStreamWaitEvent(compute_stream,push_bottom_done[(iter%2)][dev_id],0);
jacobi_kernel<<<dim_grid,dim_block,0,compute_stream>>>(a_new[dev_id],a,l2_norm_d,(iy_start+1),(iy_end[dev_id]-1),nx);

//Compute boundaries
cudaStreamWaitEvent( push_top_stream, reset_l2norm_done, 0 );
cudaStreamWaitEvent( push_top_stream, push_bottom_done[(iter%2)][top], 0 );
jacobi_kernel<<<nx/128+1,128,0,push_top_stream>>>( a_new[dev_id],a,l2_norm_d,iy_start,(iy_start+1),nx);
cudaStreamWaitEvent(push_bottom_stream,reset_l2norm_done,0);
cudaStreamWaitEvent(push_bottom_stream,push_top_done[(iter%2)][bottom], 0 ) ;
jacobi_kernel<<<nx/128+1,128,0,push_bottom_stream>>>( a_new[dev_id],a,l2_norm_d,(iy_end[dev_id]-1),iy_end[dev_id],nx);

//Apply periodic boundary conditions and exchange halo
cudaMemcpyAsync(a_new[top]+(iy_end[top]*nx),a_new[dev_id]+iy_start*nx,nx*sizeof(real),cudaMemcpyDeviceToDevice,push_top_stream);
cudaEventRecord(push_top_done[((iter+1)%2)][dev_id],push_top_stream);
cudaMemcpyAsync(a_new[bottom],a_new[dev_id]+(iy_end[dev_id]-1)*nx,nx*sizeof(real),cudaMemcpyDeviceToDevice,push_bottom_stream);
cudaEventRecord(push_bottom_done[((iter+1)%2)][dev_id],push_bottom_stream);
```

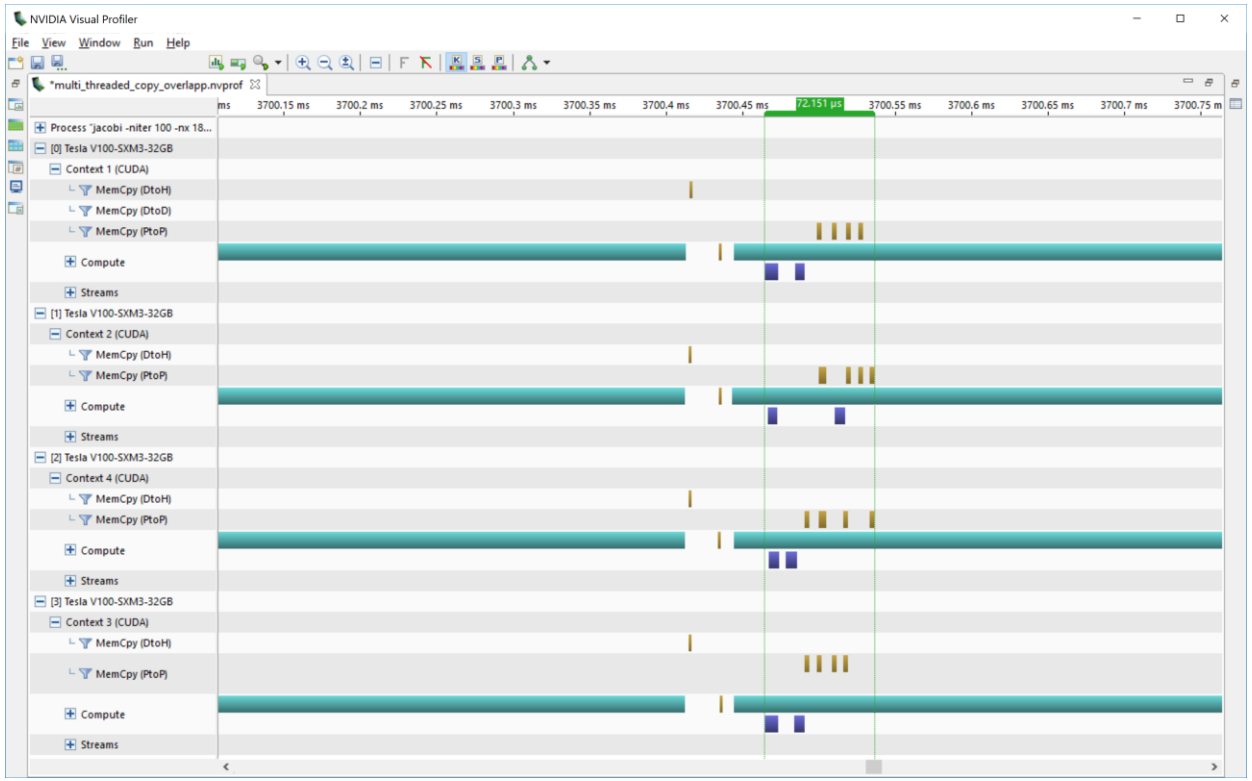
COMMUNICATION + COMPUTATION OVERLAP

High Priority Streams

```
int leastPriority = 0;

int greatestPriority = leastPriority;

cudaDeviceGetStreamPriorityRange ( &leastPriority, &greatestPriority );

cudaStreamCreateWithPriority ( &compute_stream, cudaStreamDefault, leastPriority );

cudaStreamCreateWithPriority ( &push_top_stream, cudaStreamDefault, greatestPriority );

cudaStreamCreateWithPriority ( &push_bottom_stream, cudaStreamDefault, greatestPriority );
```

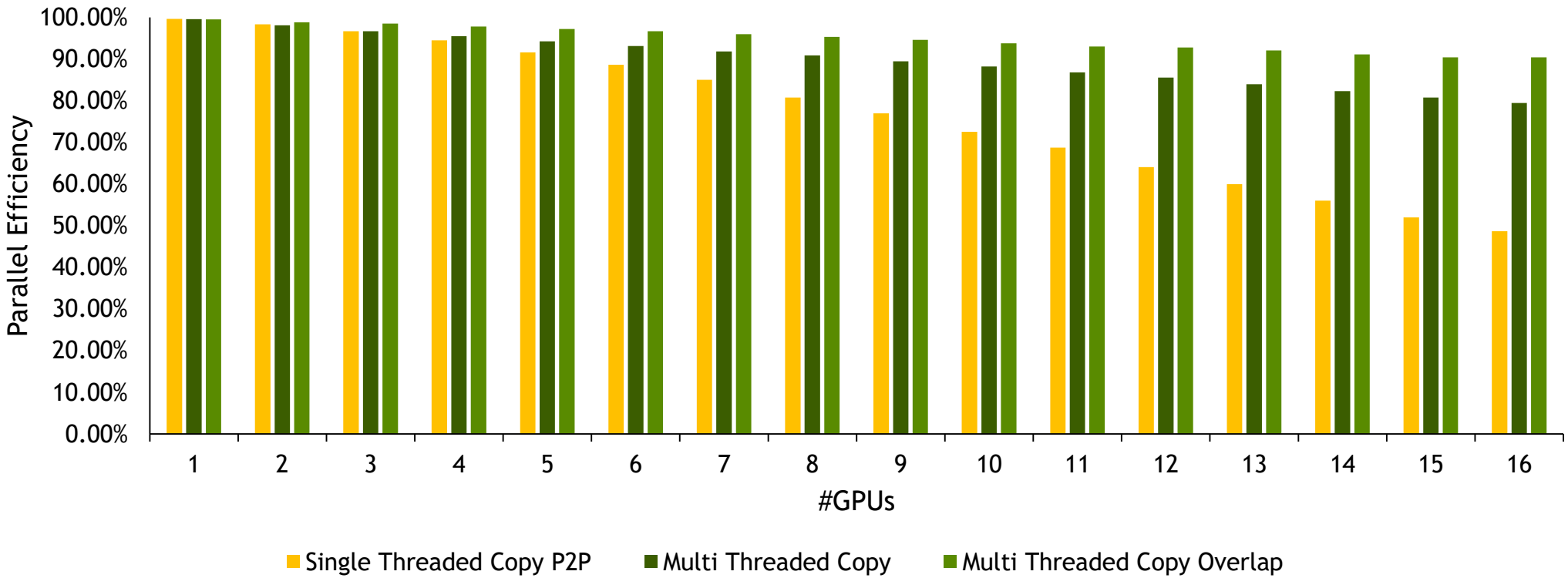
MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded Copy Overlap 4 V100 on DGX-2 with P2P



MULTI GPU JACOBI RUNTIME

DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MULTI THREADED MULTI GPU PROGRAMMING

Using OpenMP and P2P Mappings

```
while ( l2_norm > tol && iter < iter_max ) {
    cudaMemsetAsync(l2_norm_d, 0 , sizeof(real), compute_stream );
    #pragma omp barrier
    cudaStreamWaitEvent( compute_stream, compute_done[iter%2][top], 0 );
    cudaStreamWaitEvent( compute_stream, compute_done[iter%2][bottom], 0 );
    jacobi_kernel<<<dim_grid,dim_block,0,compute_stream>>>(
        a_new[dev_id], a, l2_norm_d, iy_start, iy_end[dev_id], nx,
        a_new[top], iy_end[top], a_new[bottom], 0 );
    cudaEventRecord( compute_done[(iter+1)%2][dev_id], compute_stream );
    cudaMemcpyAsync(l2_norm,l2_norm_d,sizeof(real),cudaMemcpyDeviceToHost,compute_stream);

    // l2_norm reduction btw threads skipped ...

    #pragma omp barrier
    std::swap(a_new[dev_id],a); iter++;
}
```

MULTI THREADED MULTI GPU PROGRAMMING

Using OpenMP and P2P Mappings

```
__global__ void jacobi_kernel( ... ) {
    const int ix = bIdx.x*bDim.x+tIdx.x;
    const int iy = bIdx.y*bDim.y+tIdx.y + iy_start;
    real local_l2_norm = 0.0;
    if ( iy < iy_end && ix >= 1 && ix < (nx-1) ) {
        const real new_val = 0.25 * ( a[ iy * nx + ix + 1 ] + a[ iy * nx + ix - 1 ]
                                    + a[ (iy+1) * nx + ix ] + a[ (iy-1) * nx + ix ] );
        a_new[ iy * nx + ix ] = new_val;
        if ( iy_start == iy ) a_new_top[ top_iy *nx + ix ] = new_val;
        if ( (iy_end - 1) == iy ) a_new_bottom[ bottom_iy*nx + ix ] = new_val;
        real residue = new_val - a[ iy * nx + ix ];
        local_l2_norm += residue * residue;
    }
    atomicAdd( l2_norm, local_l2_norm );
}
```

MULTI THREADED MULTI GPU PROGRAMMING

L2 norm reduction

```
cudaMemcpyAsync( l2_norm_h,l2_norm_d,sizeof(real),cudaMemcpyDeviceToHost,compute_stream );
#pragma omp barrier
#pragma omp single
{ l2_norm = 0.0; }
#pragma omp barrier
cudaStreamSynchronize( compute_stream );
#pragma omp atomic
l2_norm += *(l2_norm_h);
#pragma omp barrier
#pragma omp single
{ l2_norm = std::sqrt( l2_norm ); }
#pragma omp barrier
```

Can be hidden if L2 norm check is delayed.

MULTI THREADED MULTI GPU PROGRAMMING

Delayed L2 norm reduction

```
cudaMemcpyAsync( l2_norm_h[ curr ],l2_norm_d[ curr ],sizeof( real ),
                 cudaMemcpyDeviceToHost,compute_stream );
cudaEventRecord( copy_done[ curr ], compute_stream );
```

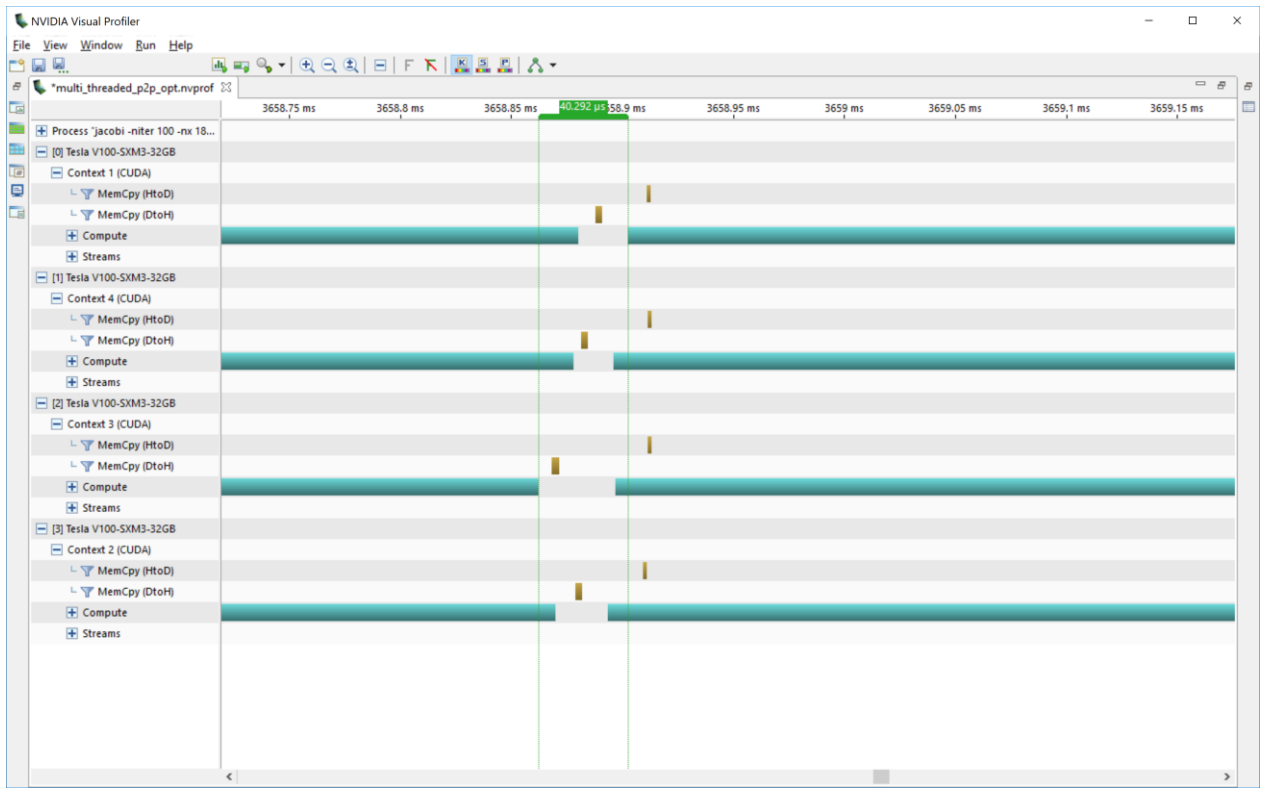
Issue H2D copy of current L2 norm

```
cudaEventSynchronize( copy_done[ prev ] );
#pragma omp atomic
l2_norm[ prev ] += *( l2_norm_h[ prev ] );
#pragma omp barrier
l2_norm[ prev ] = std::sqrt( l2_norm[ prev ] );
#pragma omp barrier
l2_norm[ prev ] = 0.0;
```

Check L2 norm of last iteration (hidden)

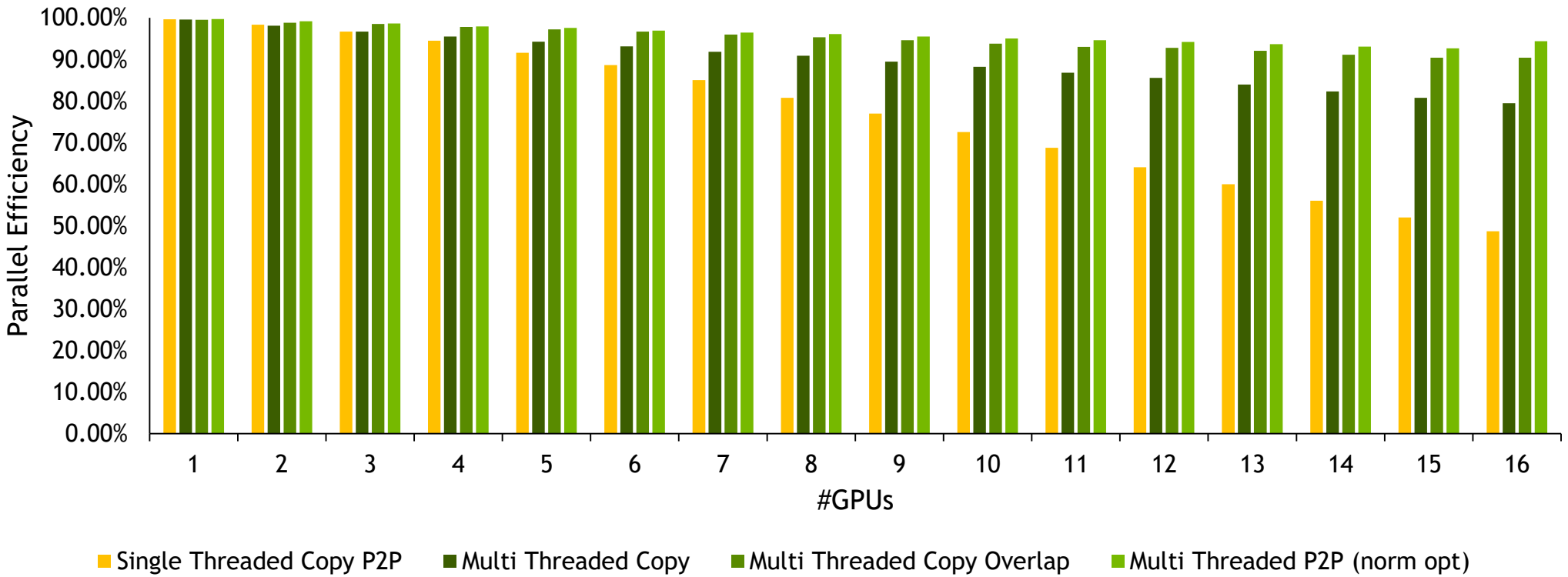
MULTI GPU JACOBI NVVP TIMELINE

Multi Threaded P2P 4 V100 and application clocks on DGX-2



MULTI GPU JACOBI RUNTIME

DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MESSAGE PASSING INTERFACE - MPI

Standard to exchange data between processes via messages

- Defines API to exchanges messages
  - Point to Point: e.g. `MPI_Send`, `MPI_Recv`
  - Collectives: e.g. `MPI_Reduce`

Multiple implementations (open source and commercial)

- Bindings for C/C++, Fortran, Python, …
- E.g. MPICH, OpenMPI, MVAPICH, IBM Platform MPI, Cray MPT, …

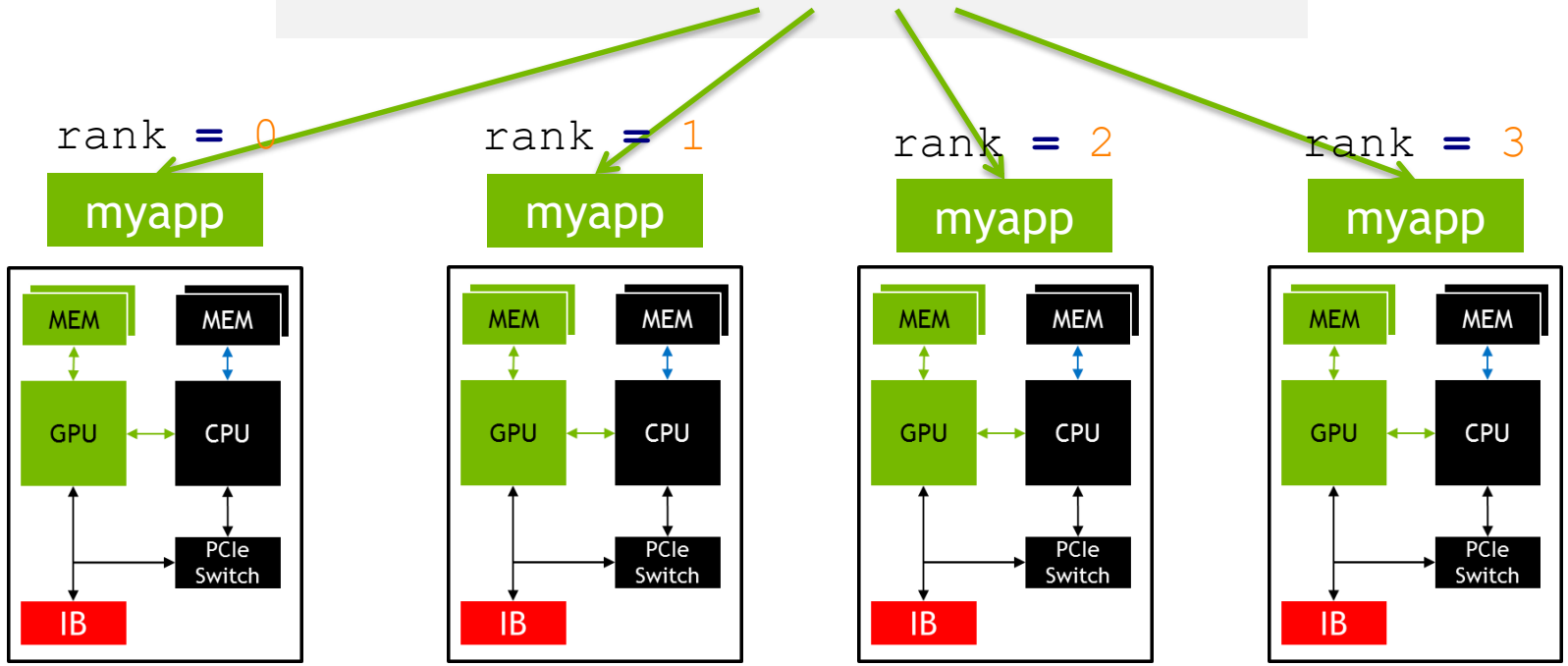
MPI - SKELETON

```c
#include <mpi.h>
int main(int argc, char *argv[]) {
    int rank,size;
    /* Initialize the MPI library */
    MPI_Init(&argc,&argv);
    /* Determine the calling process rank and total number of ranks */
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    /* Call MPI routines like MPI_Send, MPI_Recv, ... */
    ...
    /* Shutdown MPI library */
    MPI_Finalize();
    return 0;
}
```

MPI

Compiling and Launching

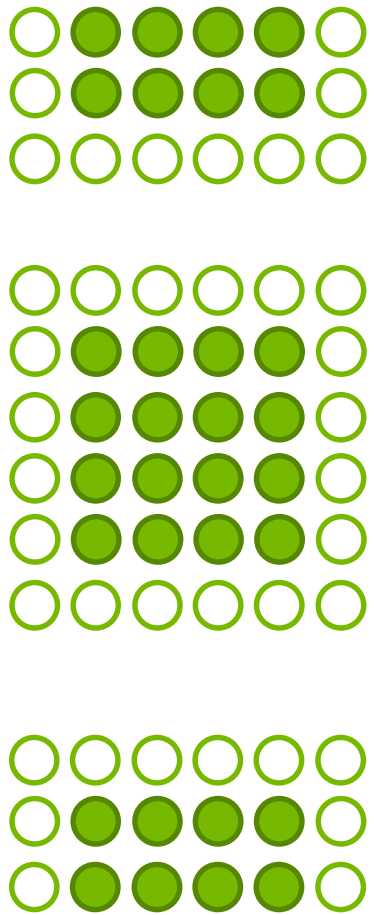
```
$ mpicc -o myapp myapp.c
$ mpirun -np 4 ./myapp <args>
```



EXAMPLE JACOBI

Top/Bottom Halo

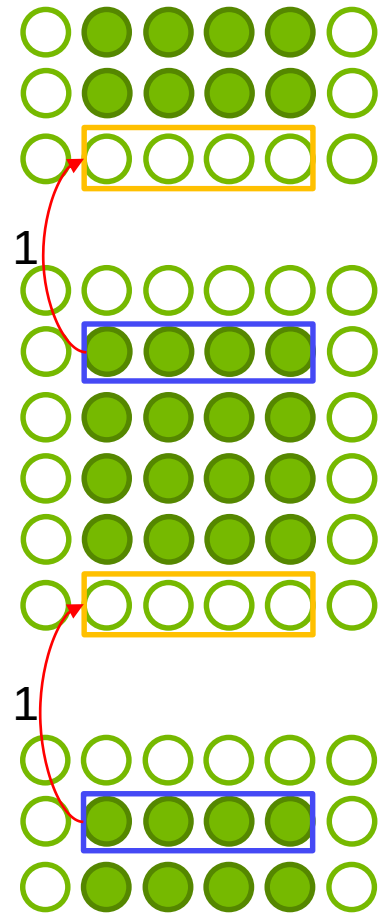
```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE JACOBI

Top/Bottom Halo

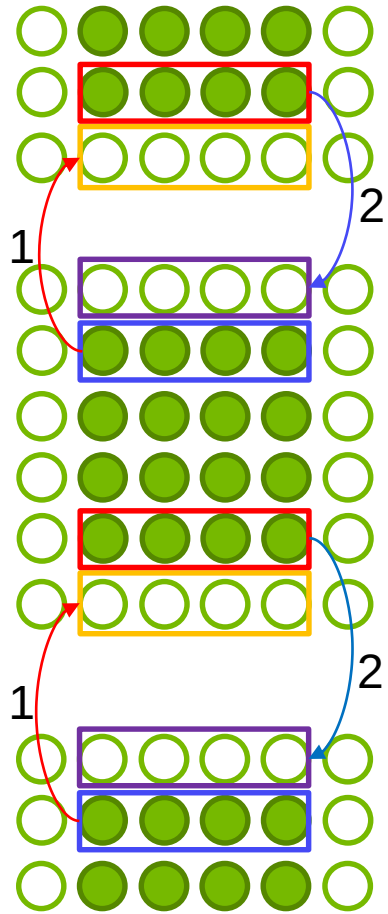
```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```



EXAMPLE JACOBI

Top/Bottom Halo

```
MPI_Sendrecv(a_new+iy_start*nx, nx, MPI_FLOAT, top , 0,
             a_new+(iy_end*nx), nx, MPI_FLOAT, bottom, 0,
             MPI_COMM_WORLD, MPI_STATUS_IGNORE);

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



HANDLING MULTIPLE MULTI GPU NODES

GPU-affinity

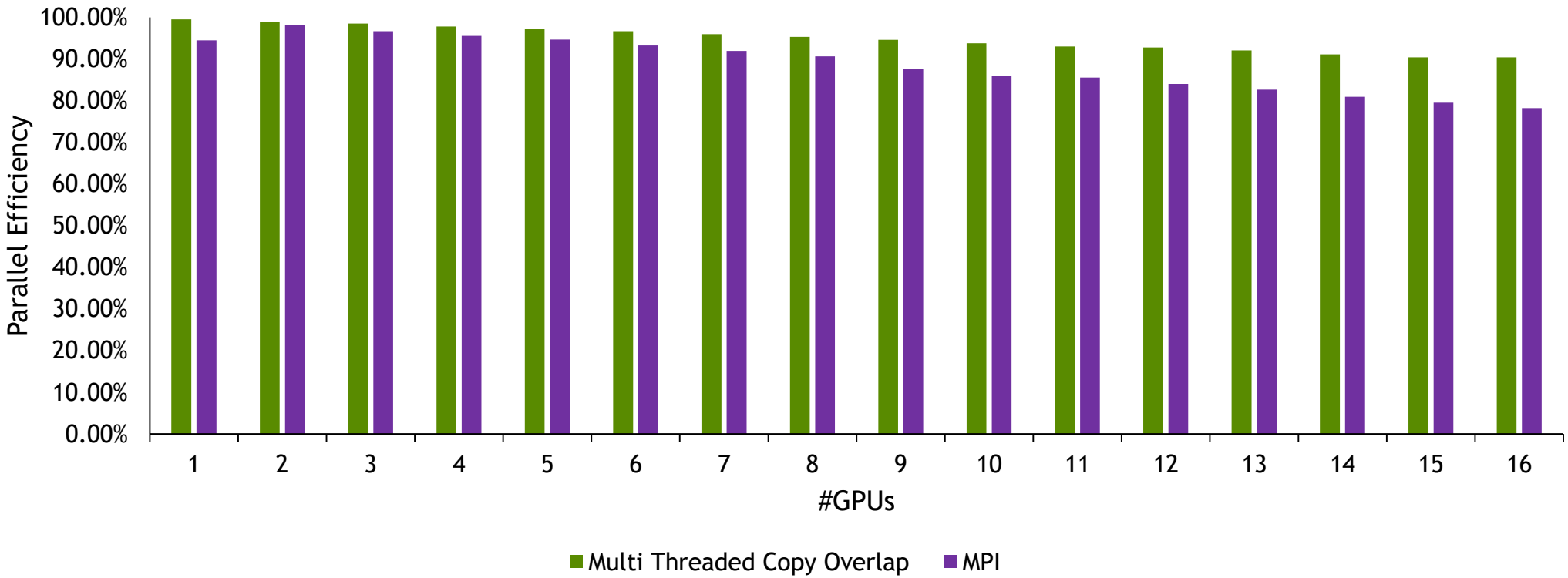
Use local rank:

```
int local_rank = -1;

MPI_Comm_rank(local_comm,&local_rank);

int num_devices = 0;

cudaGetDeviceCount(&num_devices);

cudaSetDevice(local_rank % num_devices);
```

(see backup slides for further details on how to create `local_comm`.)

MULTI GPU JACOBI RUNTIME

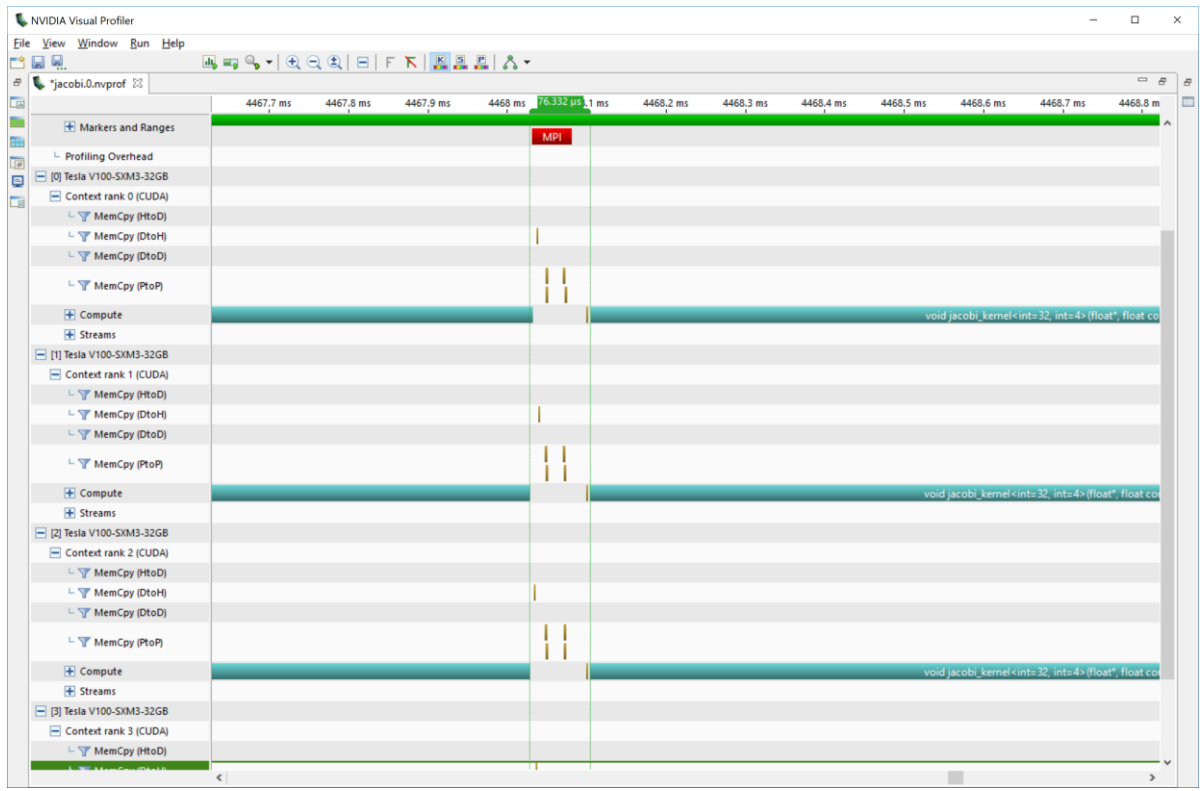
DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

MULTI GPU JACOBI NVVP TIMELINE

MPI 4 V100 on DGX-2

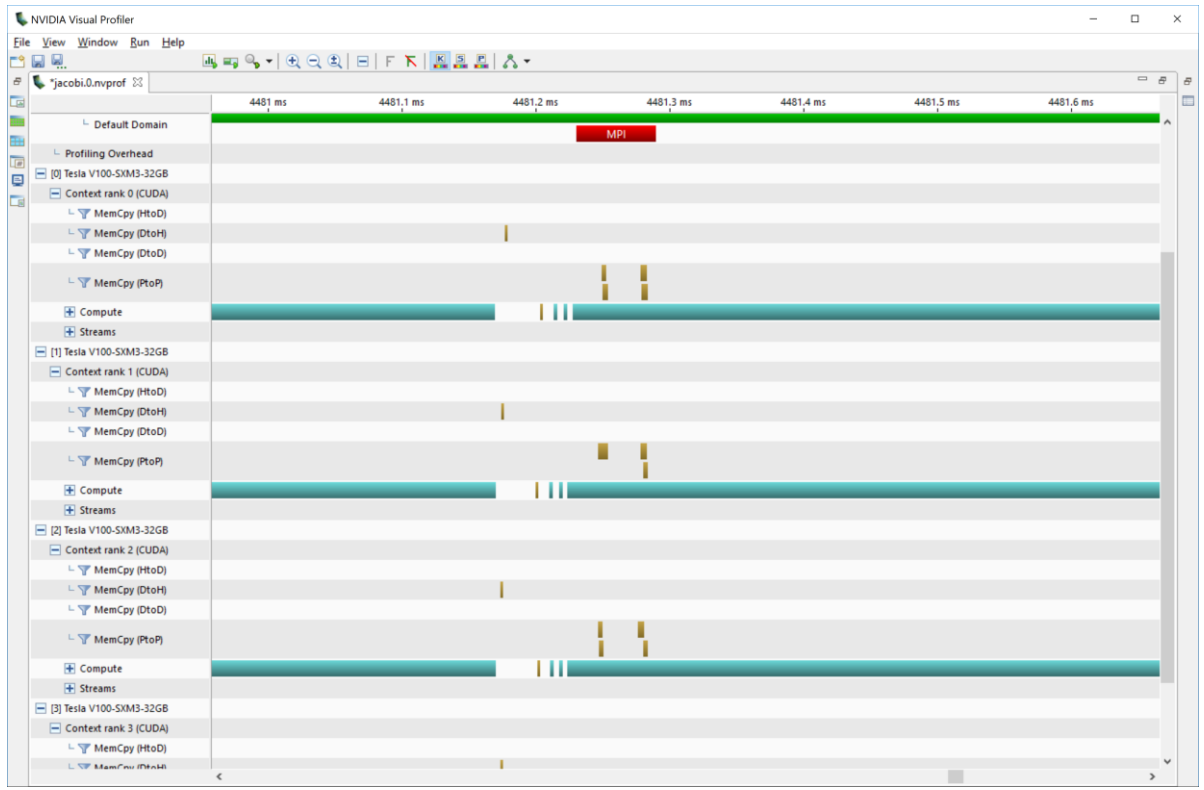


COMMUNICATION + COMPUTATION OVERLAP

```
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 );
```

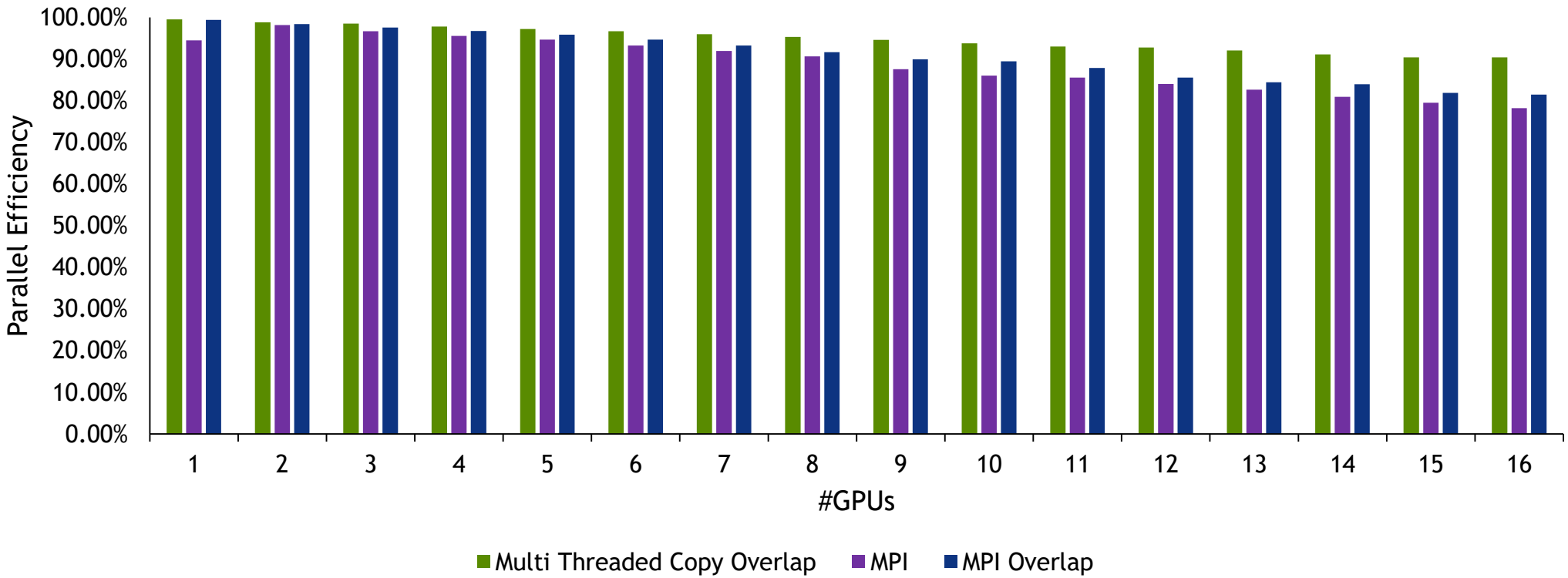
MULTI GPU JACOBI NVVP TIMELINE

MPI Overlapping 4 V100 on DGX-2



MULTI GPU JACOBI RUNTIME

DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

NVSHMEM

Implementation of OpenSHMEM, a Partitioned Global Address Space (PGAS) library

- Defines API to (symmetrically) allocate memory that is remotely accessible
- Defines API to access remote data
  - One-sided: e.g. `shmem_putmem, shmem_getmem`
  - Collectives: e.g. `shmem_broadcast`

NVSHMEM features

- Symmetric memory allocations in device memory
- Communication API calls on CPU (standard and stream-ordered)
- Allows kernel-side communication (API and LD/ST*)
- Interoperable with MPI

NVSHMEM - SKELETON

```
#include <shmem.h>
#include <shmemx.h>
int main(int argc, char *argv[]) {
    ...
    MPI_Comm mpi_comm;
    shmemx_init_attr_t attr;
    mpi_comm = MPI_COMM_WORLD;
    attr.mpi_comm = &mpi_comm;
    shmemx_init_attr (SHMEMX_INIT_WITH_MPI_COMM, &attr);
    int npes = shmem_n_pes();
    int mype = shmem_my_pe();
    ...
    return 0;
}
```

NVSHMEM - HOST CODE

```
a = (float *) shmem_malloc(nx*(chunk_size+2)*sizeof(float));
a_new = (float *) shmem_malloc(nx*(chunk_size+2)*sizeof(float));
…
while ( l2_norm > tol && iter < iter_max ) {
    …
    jacobi_kernel<<<dim_grid,dim_block,0,compute_stream>>>(
        a_new, a, l2_norm_d, iy_start, iy_end, nx,
        top, iy_end_top, bottom, iy_start_bottom );
    shmemx_barrier_all_on_stream(compute_stream);
     …
}
shmem_barrier_all();
shmem_free(a);
shmem_free(a_new);
```

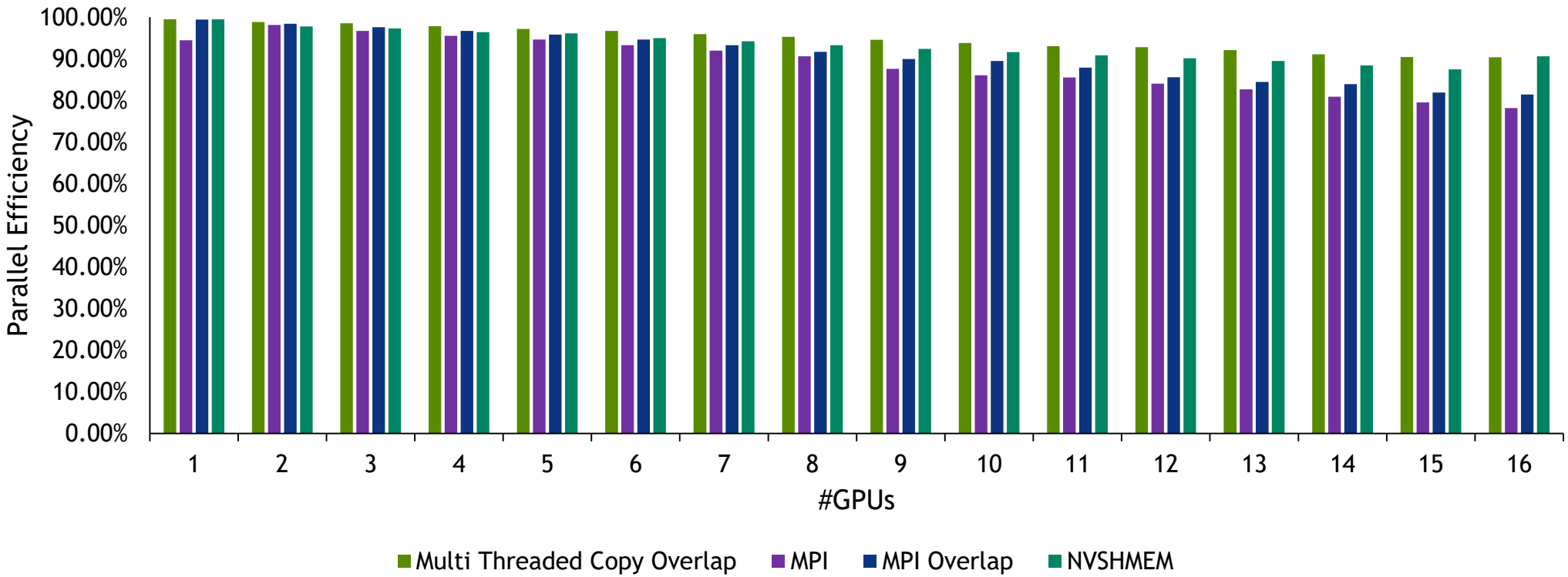
NVSHMEM - KERNEL

```
__global__ void jacobi_kernel( ... ) {
    const int ix = bIdx.x*bDim.x+tIdx.x;
    const int iy = bIdx.y*bDim.y+tIdx.y + iy_start;
    float local_l2_norm = 0.0;
    if ( iy < iy_end && ix >= 1 && ix < (nx-1) ) {
        const float new_val = 0.25 * ( a[ iy * nx + ix + 1 ] + a[ iy * nx + ix - 1 ]
                                     + a[ (iy+1) * nx + ix ] + a[ (iy-1) * nx + ix ] );
        a_new[ iy * nx + ix ] = new_val;
        if ( iy_start == iy )
            shmem_float_p(a_new + top_iy*nx + ix, new_val, top_pe);
        if ( iy_end == iy )
            shmem_float_p(a_new + bottom_iy*nx + ix, new_val, bottom_pe);
        float residue = new_val - a[ iy * nx + ix ];
    }
    atomicAdd( l2_norm, local_l2_norm );
}
```

Optimized for DGX-2/NVLink, other approach might be better for portable performance

MULTI GPU JACOBI RUNTIME

DGX-2 - 18432 x 18432, 1000 iterations



Benchmarksetup: DGX-2 with OS 4.0.5, GCC 7.3.0, CUDA 10.0 with 410.104 Driver, CUB 1.8.0, CUDA-aware OpenMPI 4.0.0, NVSHMEM EA2 (0.2.3), GPUs@1597Mhz AC, , Reported Runtime is the minimum of 5 repetitions

NOT COVERED IN THIS TALK I

MPI with GPUDirect Async support (under development)

Enables MPI communication to follow CUDA stream ordering

Avoids unwanted CPU/GPU synchronization

NOT COVERED IN THIS TALK II

NCCL: Accelerating multi-GPU collective communications

GOAL:

- A library for collective communication using CUDA kernels for reduction and data movement.

APPROACH:

- Allreduce, Reduce, Broadcast, ReduceScatter and Allgather primitives, similar to MPI primitives.
- CUDA oriented : works on CUDA pointers only, enqueues operations to CUDA streams.
- Supports any mapping between processes, threads and GPUs per thread to integrate into any hybrid model.

CONCLUSION

Thank you for your attention!

	Programming Models	GPUDirect P2P/RDMA	Multi Node
Single Threaded	CUDA	Improves Perf.	No
Multi Threaded	CUDA + OpenMP/TBB/…	Improves Perf.	No
Multi Threaded P2P	CUDA + OpenMP/TBB/…	Required	No
MPI	CUDA + MPI	Improves Perf.	Yes
NVSHMEM*	CUDA + NVSHMEM (MPI interop.)	Required	Yes

CE9104 - Connect with the Experts: Multi-GPU Programming:

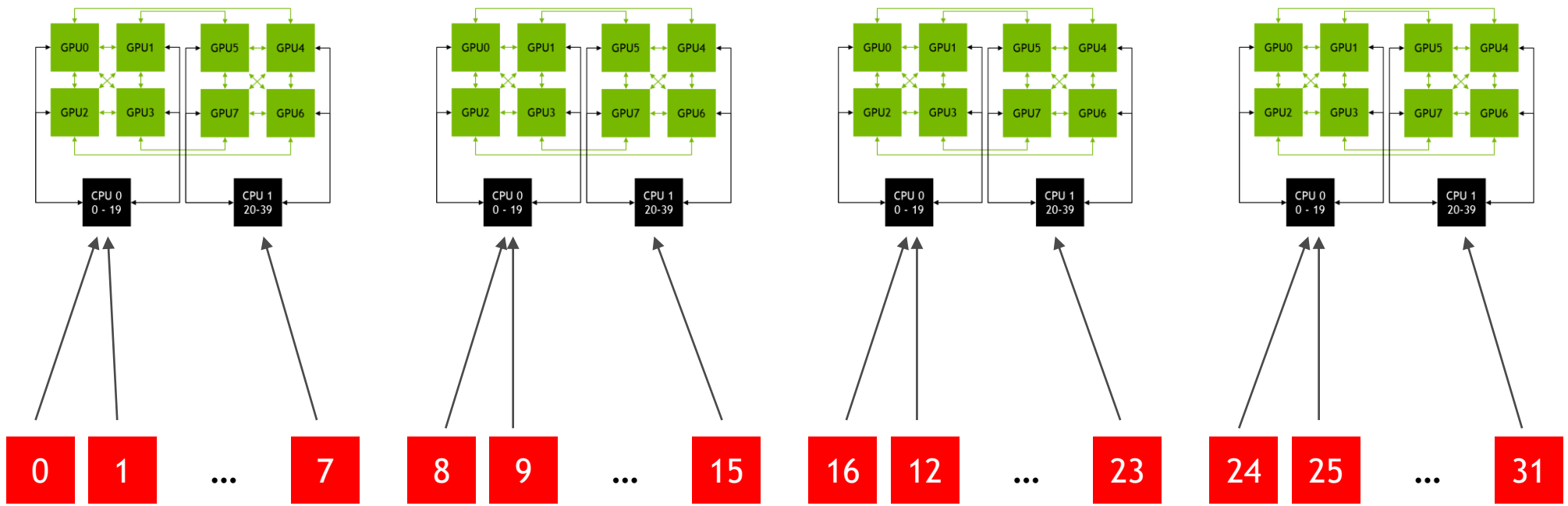
3PM Today (directly after this talk) - SJCC Hall 3 Pod C (Concourse Level)

Source is on GitHub: https://github.com/NVIDIA/multi-gpu-programming-models

*Please reach out to nvshmem@nvidia.com for an early access to NVSHMEM


BACKUP

HANDLING MULTIPLE MULTI GPU NODES

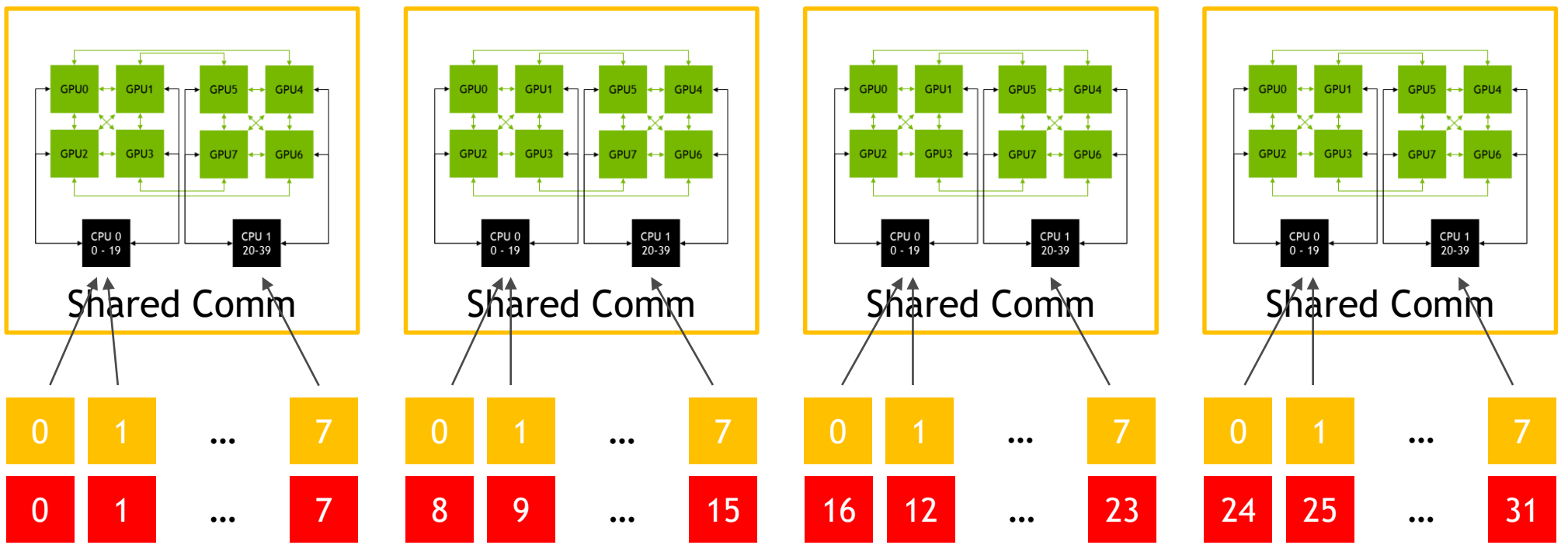


HANDLING MULTIPLE MULTI GPU NODES

How to determine the local rank? – MPI-3

```
MPI_Comm local_comm;

MPI_Info info;

MPI_Info_create(&info);

MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, rank, info, &local_comm);

int local_rank = -1;

MPI_Comm_rank(local_comm,&local_rank);

MPI_Comm_free(&local_comm);

MPI_Info_free(&info);
```

HANDLING MULTIPLE MULTI GPU NODES



MULTI GPU JACOBI NVVP TIMELINE

MPI 1 V100 on DGX-2