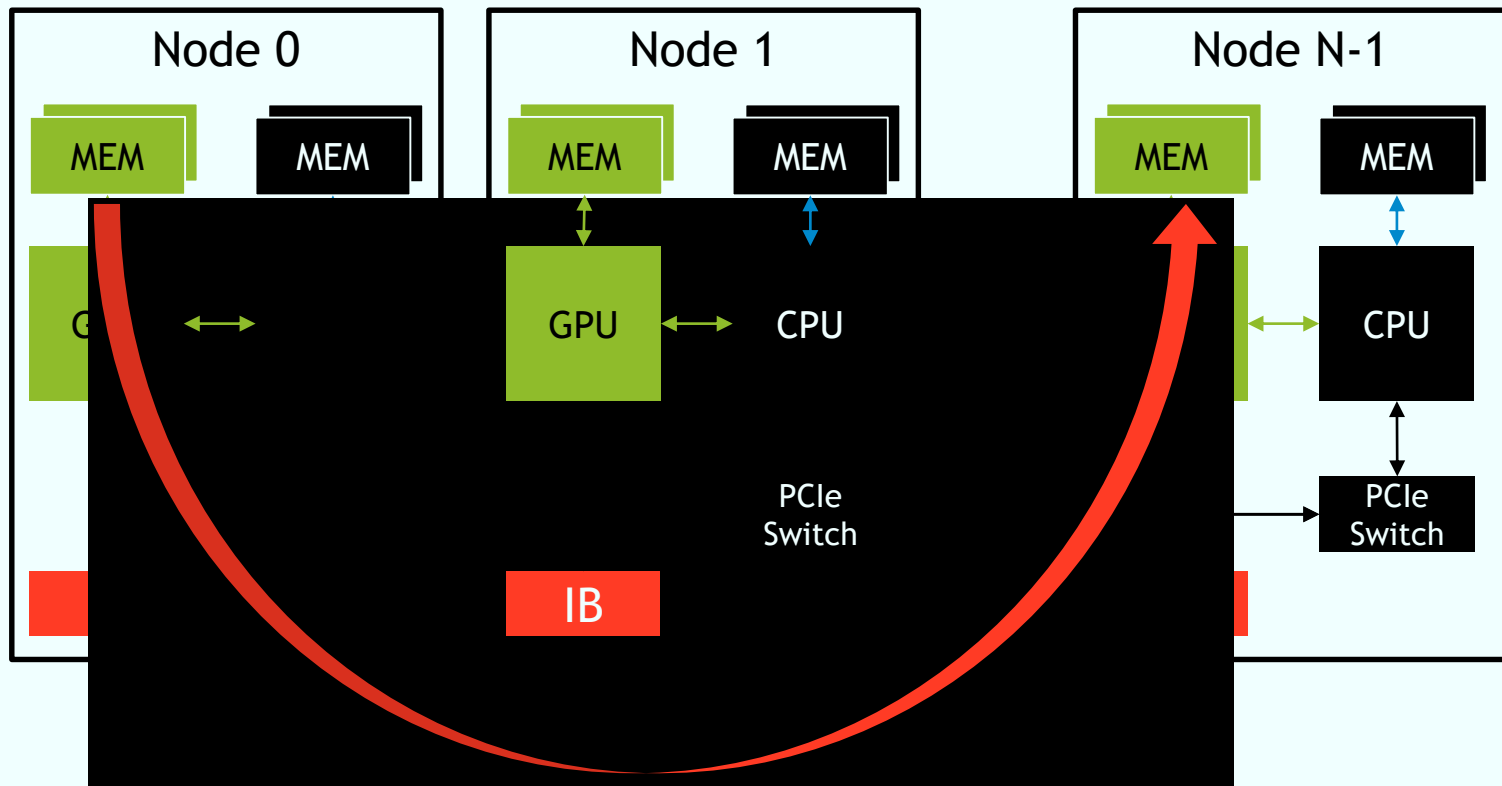


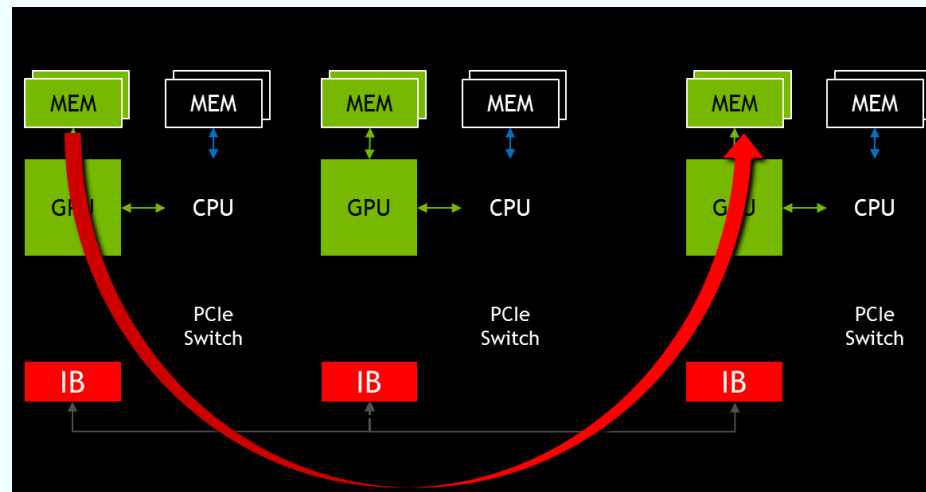
Multi GPU Programming with MPI and OpenACC

04.10.2017 | J. Kraus (NVIDIA)

MPI+OpenACC



MPI+OpenACC



```
//MPI rank 0
#pragma acc host_data use_device( sbuf )
MPI_Send(sbuf, size, MPI_DOUBLE, n-1, tag, MPI_COMM_WORLD);

//MPI rank n-1
#pragma acc host_data use_device( rbuf )
MPI_Recv(rbuf, size, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

What you will learn

- What MPI is
- How to use MPI for inter GPU communication with OpenACC
- How to use pgprof for MPI+OpenACC applications
- How to hide MPI communication times

Message Passing Interface - MPI

- Standard to exchange data between processes via messages
 - Defines API to exchanges messages
 - Pt. 2 Pt.: e.g. MPI_Send, MPI_Recv
 - Collectives, e.g. MPI_Allreduce
- Multiple implementations (open source and commercial)
 - Binding for C/C++, Fortran, Python, ...
 - E.g. MPICH, OpenMPI, MVAPICH, IBM Platform MPI, Cray MPT, ...

MPI – A minimal program

```
#include <mpi.h>

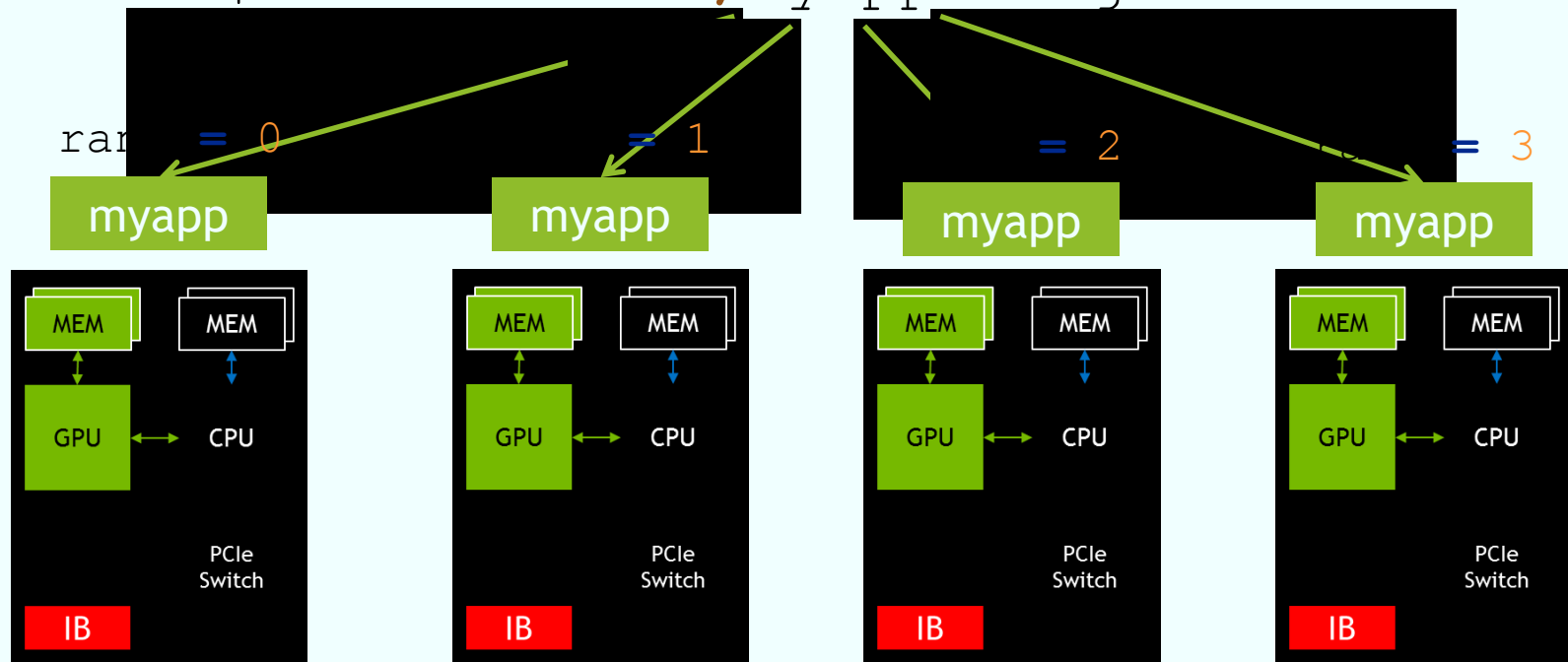
int main(int argc, char *argv[]) {
    int rank, size;
    /* Initialize the MPI library */
    MPI_Init(&argc, &argv);
    /* Determine the calling rank and total number of ranks */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    /* Call MPI routines like MPI_Send,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

Remark: Almost all MPI routines return an error value which should be checked. The examples and tasks leave that out for brevity.

MPI – Compiling and Launching

```
$ mpicc -o myapp myapp.c
```

```
$ srun -n 4 ./myapp <args>
```



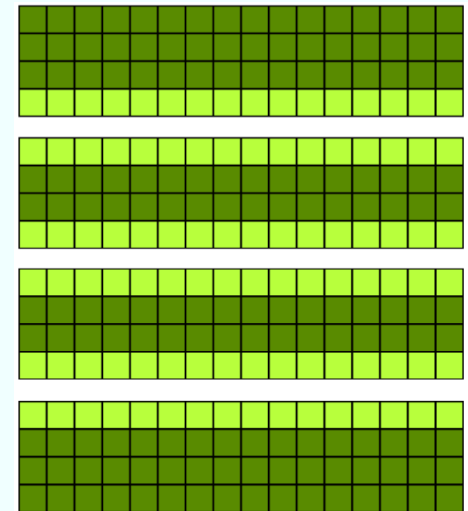
Example: Jacobi Solver

Solves the 2D-Poisson Equation on a rectangle

$$\Delta u(x, y) = e^{-10 \cdot (x^2 + y^2)} \quad \forall (x, y) \in \Omega \setminus \delta\Omega$$

Periodic boundary conditions

Domain decomposition with stripes



Horizontal Stripes

Example: Jacobi Solver – Single GPU

While not converged

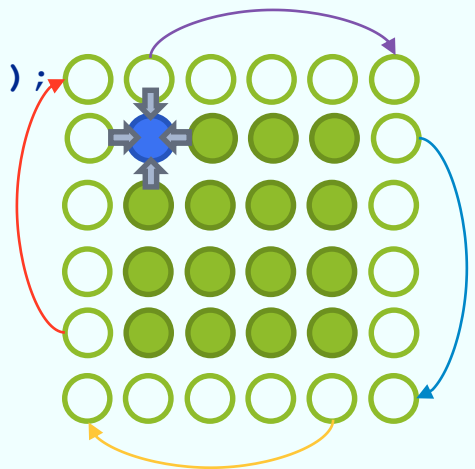
Do Jacobi step:

```
for (int iy = 1; iy < ny-1; ++iy)
  for (int ix = 1; ix < nx-1; ++ix)
    Anew[iy*nx+ix] = -0.25f * (rhs[iy*nx+ix]
      - (A[iy*nx+(ix-1)] + A[iy*nx+(ix+1)]
      + A[(iy-1)*nx+ix] + A[(iy+1)*nx+ix]));
```

Copy Anew to A

Apply periodic boundary conditions

Next iteration



Domain Decomposition

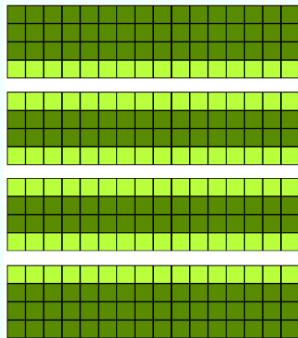
Different ways to split the work between processes:

Minimizes number of neighbors:

- Communicate to less neighbors
- Optimal for latency bound communication

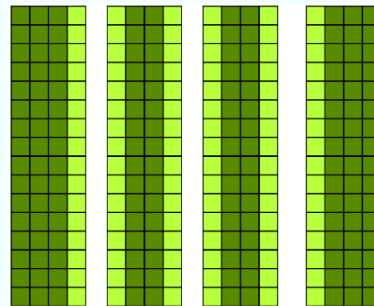
Minimizes surface area/volume ratio:

- Communicate less data
- Optimal for bandwidth bound communication



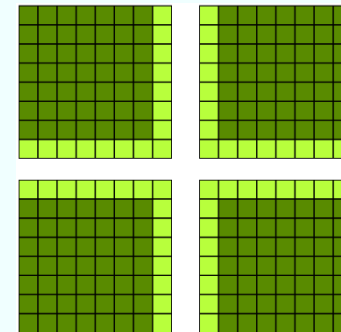
Horizontal Stripes

Contiguous if data is row-major



Vertical Stripes

Contiguous if data is column-major



Tiles

Example: Jacobi Solver – Multi GPU

While not converged

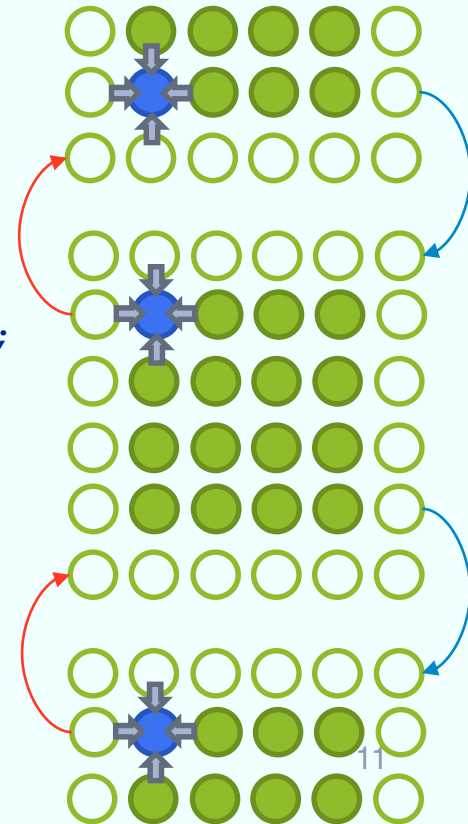
Do Jacobi step:

```
for (int iy = iy_start; iy < iy_end; ++iy)
  for (int ix = 1; ix < NX-1; ++ix)
    Anew[iy*nx+ix] = -0.25f * (rhs[iy*nx+ix]
      - (A[iy*nx+(ix-1)] + A[iy*nx+(ix+1)]
      + A[(iy-1)*nx+ix] + A[(iy+1)*nx+ix]));
```

Copy Anew to A

Apply periodic boundary conditions and exchange halo with 2 neighbors

Next iteration



Handling GPU Affinity

- Rely on process placement (with on rank per GPU)*

```
#if _OPENACC
```

```
acc_device_t device_type = acc_get_device_type();
```

```
int ngpus=acc_get_num_devices(device_type);
```

```
int devicenum=rank%ngpus;
```

Alternative (JURECA):

```
int devicenum = atoi(getenv("MPI_LOCALRANKID"));
```

```
#endif /*_OPENACC*/
```

```
#pragma acc set device_num( devicenum )
```

```
#if _OPENACC // Or using the API:
```

```
acc_set_device_num(devicenum,device_type);
```

```
#endif /*_OPENACC*/
```

* This assumes the node is homogeneous, i.e. that all the GPUs are the same. If you have different GPUs in the same node then you may need some more complex GPU selection

Profiling MPI+OPENACC applications

Embed MPI rank in output filename, process name, and context name

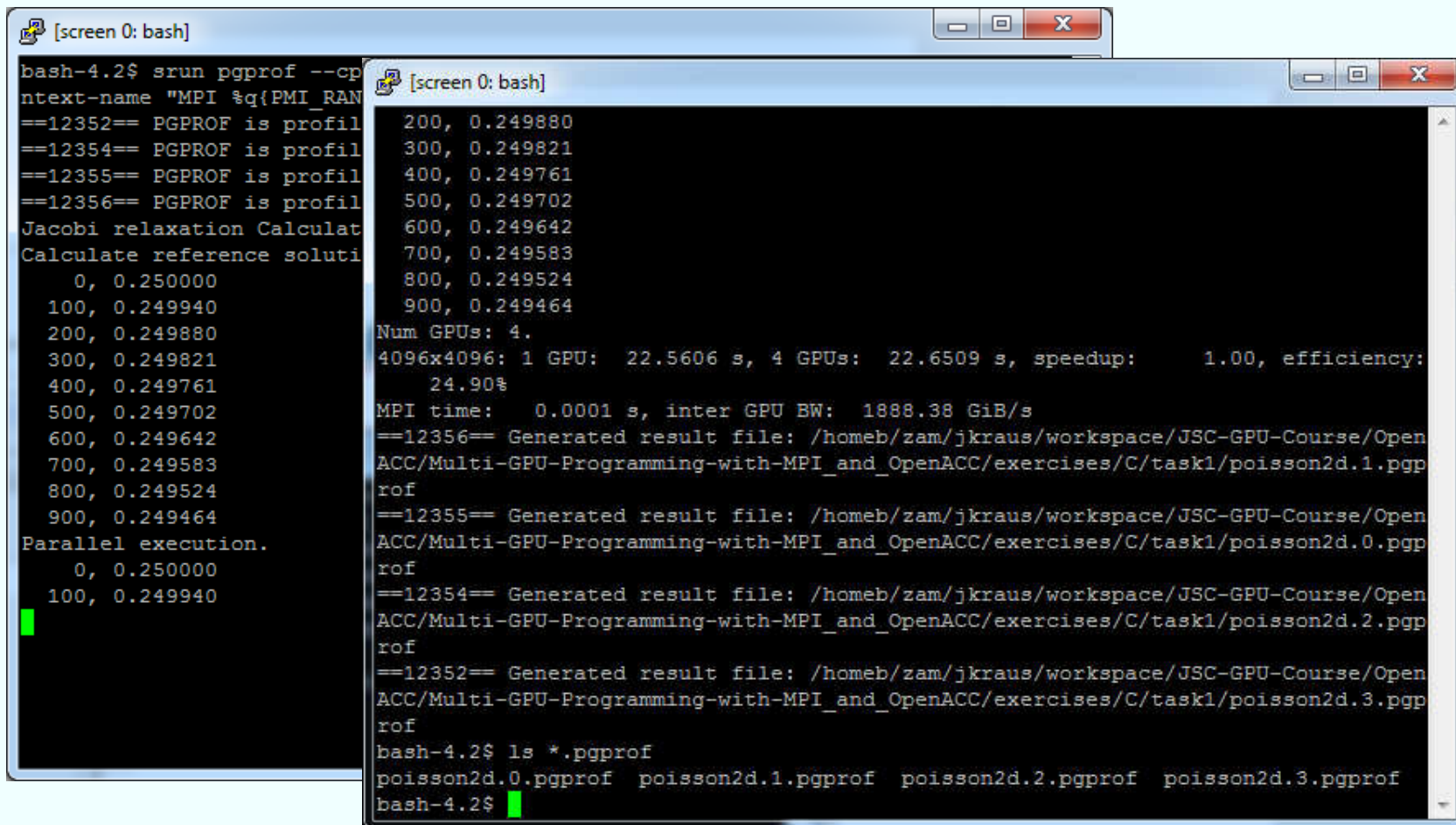
```
srun pgprof --cpu-profiling off  
            --output-profile profile.%q{PMI_RANK} \\  
            --process-name "rank %q{PMI_RANK}" \\  
            --context-name "rank %q{PMI_RANK}"
```

JURECA: PMI_RANK

OpenMPI: OMPI_COMM_WORLD_RANK

MVAPICH2: MV2_COMM_WORLD_RANK

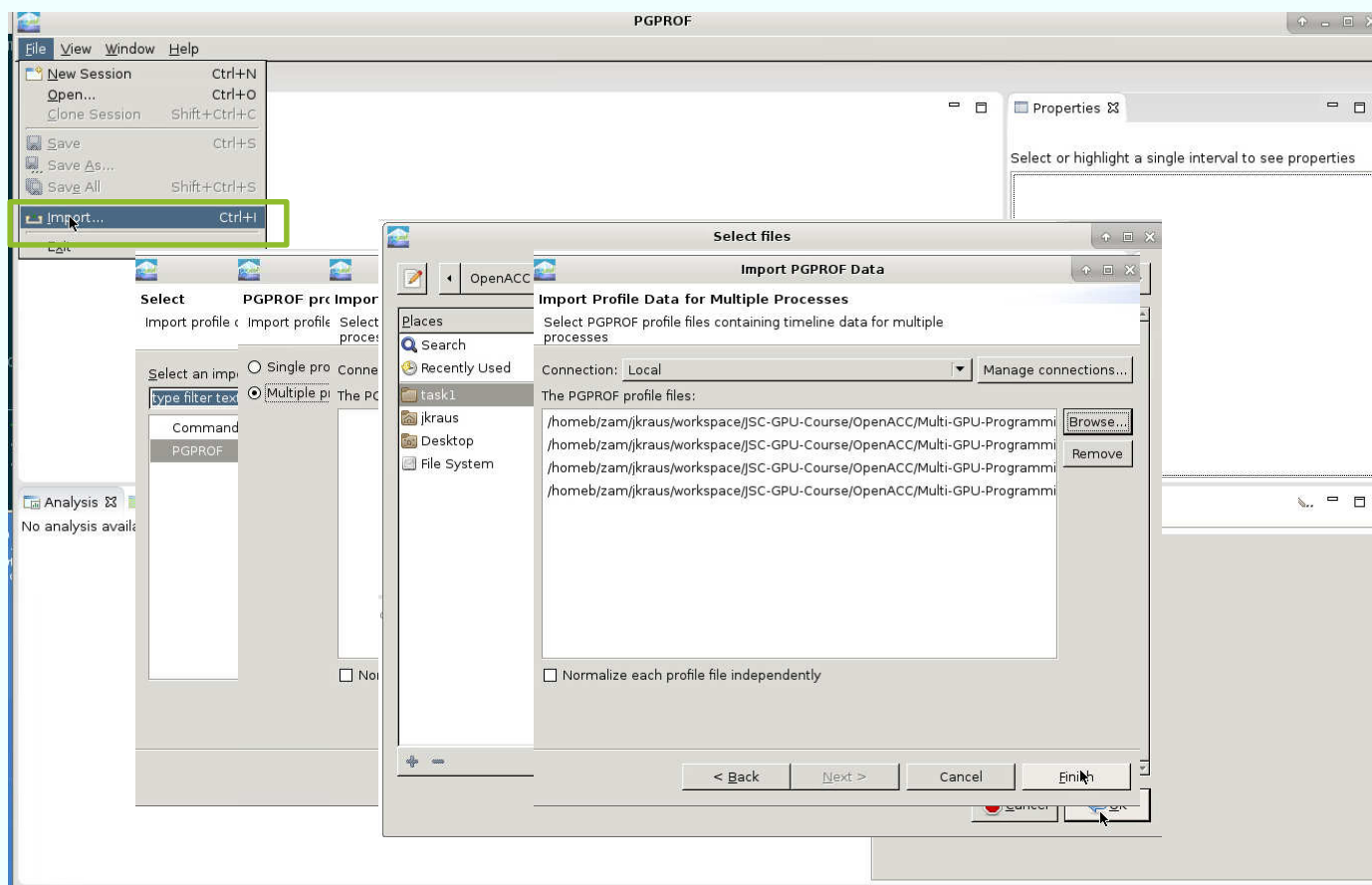
Profiling MPI+OpenACC applications



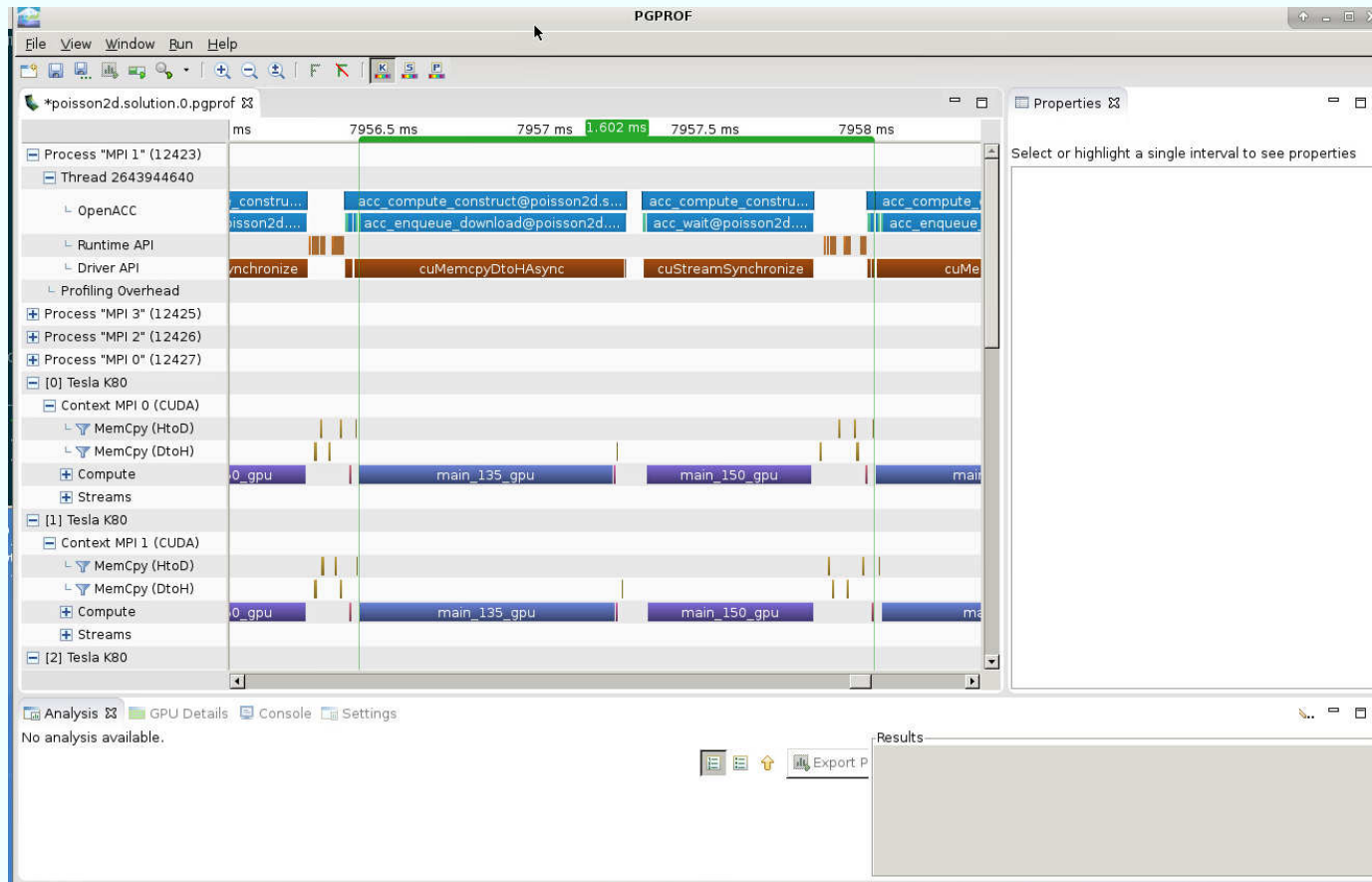
```

[screen 0: bash]
bash-4.2$ srun pgprof --cp
ntext-name "MPI %q{PMI_RAN
==12352== PGPROF is profil
==12354== PGPROF is profil
==12355== PGPROF is profil
==12356== PGPROF is profil
Jacobi relaxation Calculat
Calculate reference soluti
  0, 0.250000
 100, 0.249940
 200, 0.249880
 300, 0.249821
 400, 0.249761
 500, 0.249702
 600, 0.249642
 700, 0.249583
 800, 0.249524
 900, 0.249464
Parallel execution.
  0, 0.250000
 100, 0.249940
[screen 0: bash]
200, 0.249880
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Num GPUs: 4.
4096x4096: 1 GPU: 22.5606 s, 4 GPUs: 22.6509 s, speedup: 1.00, efficiency:
24.90%
MPI time: 0.0001 s, inter GPU BW: 1888.38 GiB/s
==12356== Generated result file: /homeb/zam/jkraus/workspace/JSC-GPU-Course/Open
ACC/Multi-GPU-Programming-with-MPI_and_OpenACC/exercises/C/task1/poisson2d.1.pgpr
rof
==12355== Generated result file: /homeb/zam/jkraus/workspace/JSC-GPU-Course/Open
ACC/Multi-GPU-Programming-with-MPI_and_OpenACC/exercises/C/task1/poisson2d.0.pgpr
rof
==12354== Generated result file: /homeb/zam/jkraus/workspace/JSC-GPU-Course/Open
ACC/Multi-GPU-Programming-with-MPI_and_OpenACC/exercises/C/task1/poisson2d.2.pgpr
rof
==12352== Generated result file: /homeb/zam/jkraus/workspace/JSC-GPU-Course/Open
ACC/Multi-GPU-Programming-with-MPI_and_OpenACC/exercises/C/task1/poisson2d.3.pgpr
rof
bash-4.2$ ls *.pgprof
poisson2d.0.pgprof poisson2d.1.pgprof poisson2d.2.pgprof poisson2d.3.pgprof
bash-4.2$
  
```

Profiling MPI+OpenACC applications



Profiling MPI+OpenACC applications

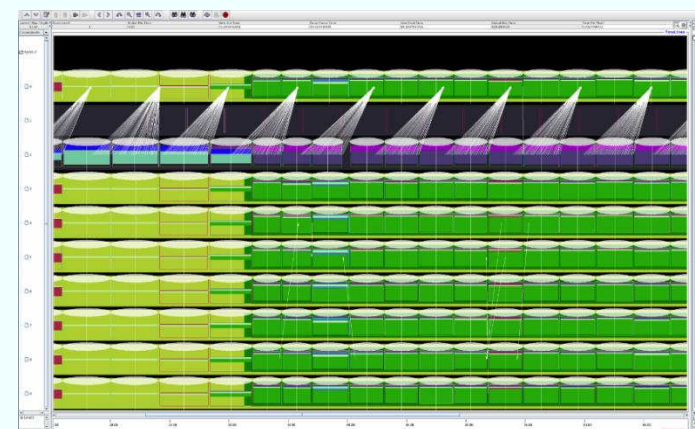
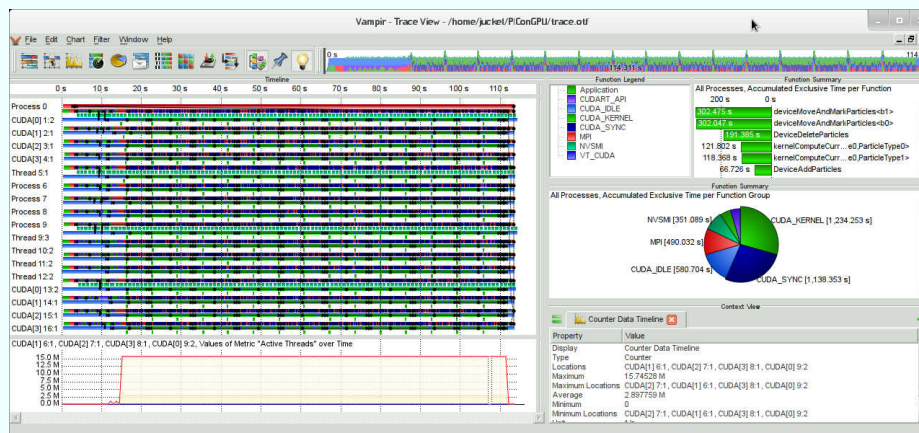


Profiling MPI+OpenACC applications

Multiple parallel profiling tools are CUDA-aware

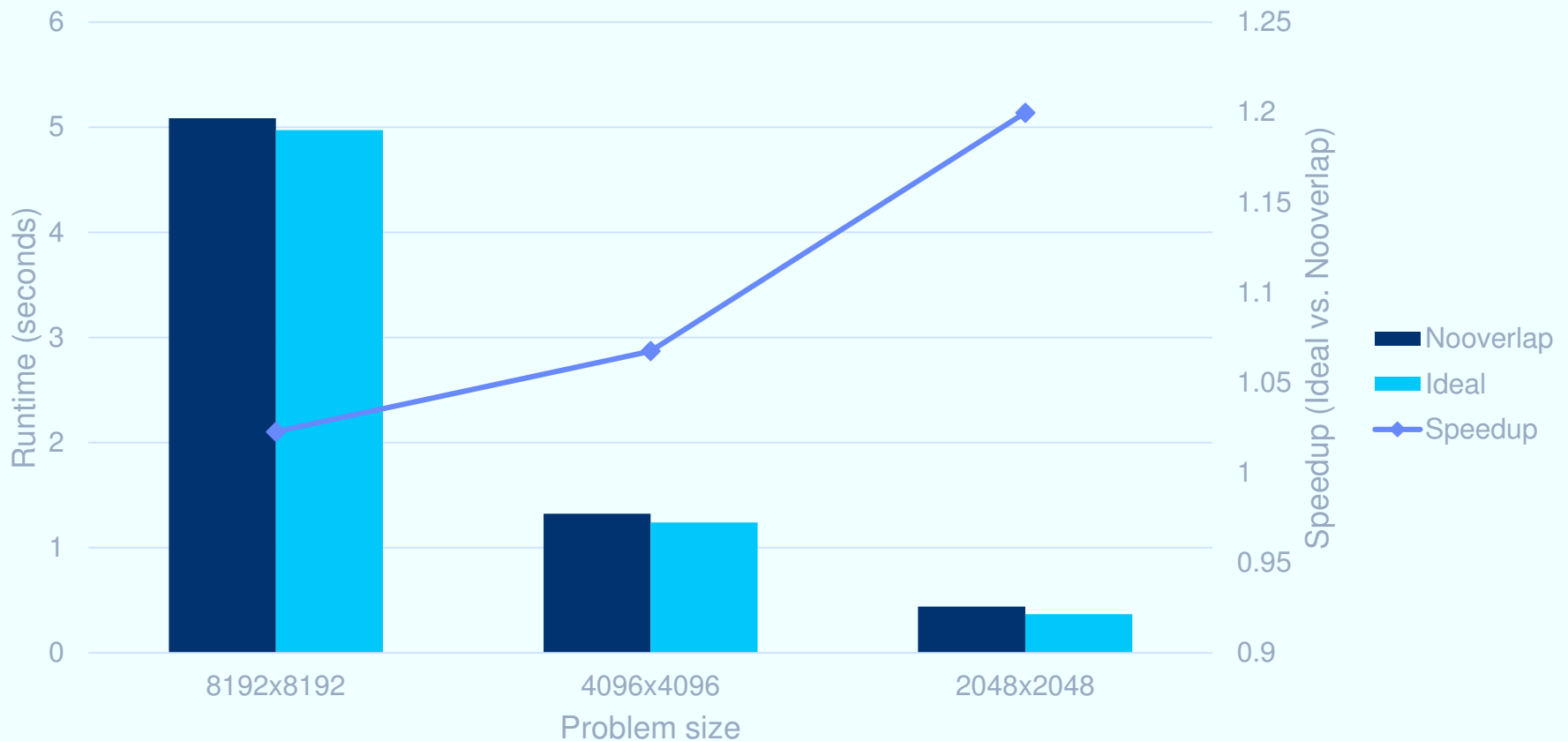
- Score-P
- Vampir
- Tau

These tools are good for discovering MPI issues as well as basic CUDA performance inhibitors.

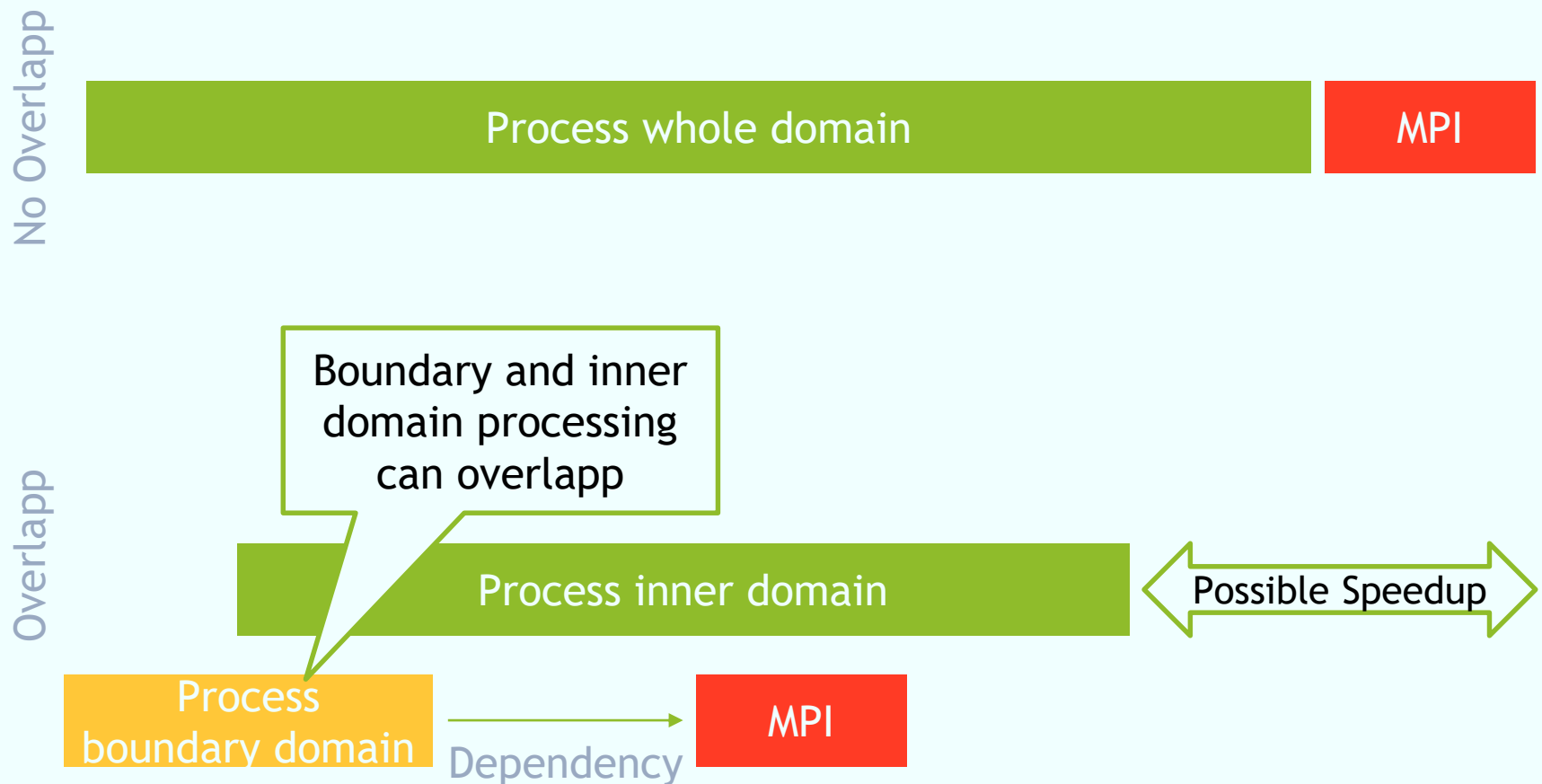


Communication + Computation Overlap

MVAPICH2-GDR 2.2b – PGI 16.9 - 2 Tesla K80



Communication + Computation Overlap



Communication + Computation Overlap

```
#pragma acc parallel loop
for ( ... )
    //Process boundary
#pragma acc parallel loop async
for ( ... )
    //Process inner domain

#pragma acc host_data use_device ( A )
{
    //Exchange halo with top and bottom neighbor
    MPI_Sendrecv( A... );
    //...
}
//wait for iteration to finish
#pragma acc wait
```

Scalability Metrics For Success

- Serial Time: T_s :
 - How long it takes to run the problem with a single process
- Parallel Time: T_p
 - How long it takes to run the problem with multiple processes
- Number of Processes: P
 - The number of Processes operating on the task at hand
- Speedup: $S = T_s / T_p$
 - How much faster is the parallel version vs. serial. (optimal is P)
- Efficiency: $E = S / P$
 - How efficient are the processors used (optimal is 1)

Hands-on Setup Environment

```
$ module load PGI/17.3-GCC-5.4.0  
$ module load MVAPICH2/2.2-GDR  
$ export TMPDIR=/tmp
```

Task 1: Apply domain decomposition

Handle GPU affinity

Look for TODOs

Halo Exchange

```
$ make
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla poiss
[...]
srun ./poisson2d
Jacobi relaxation Calculation: 4096 x 4096 mesh
[...]
Num GPUs: 4.
4096x4096: 1 GPU: 18.9390 s,
MPI time: 0.0001 s, inter G
```

Make Targets:

```
run:          run poisson2d (default)
poisson2d:    build poisson2d binary
profile:      profile with pgprof
*.solution:   same as above with solution
               (poisson2d.solution.*)
```

Task 2: Hide MPI communication time

Start copy loop asynchronously

Look for TODOs

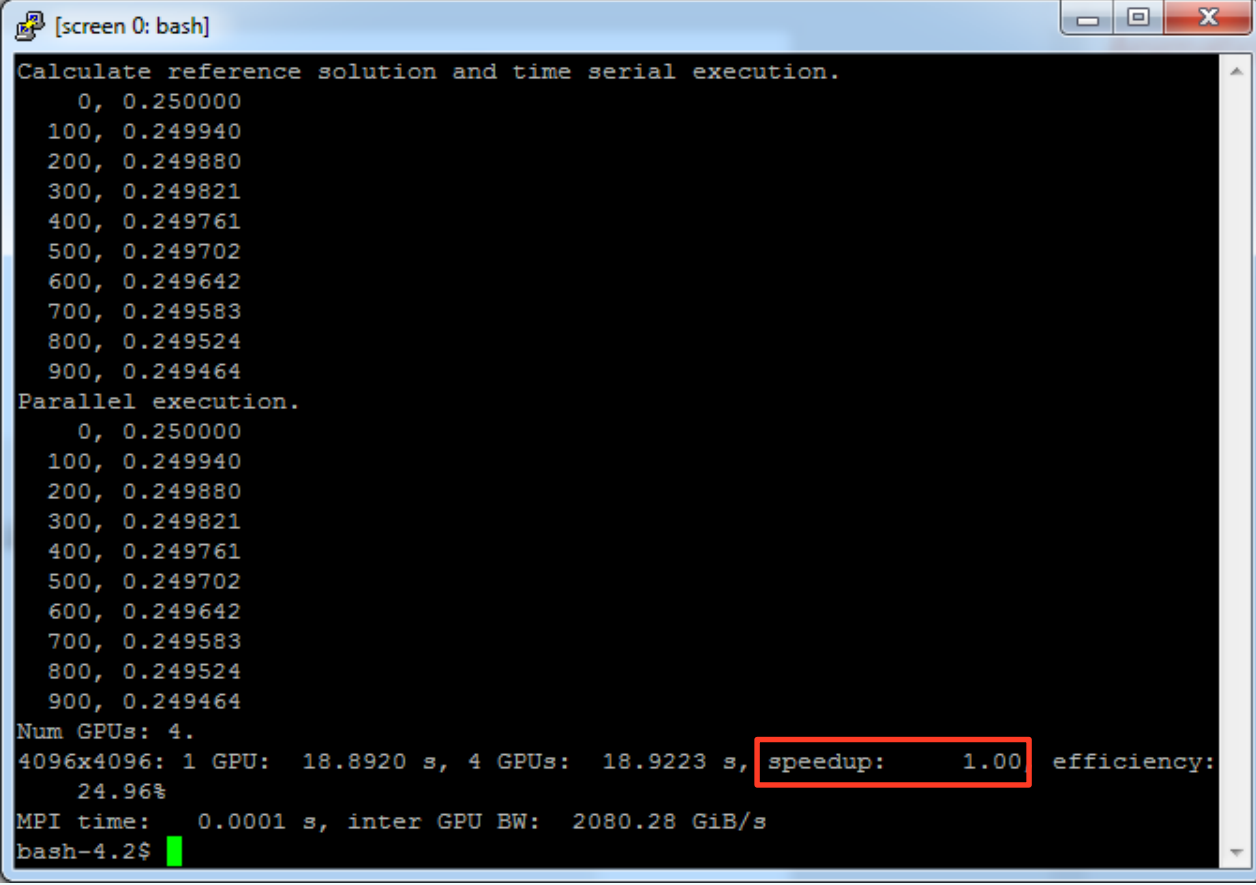
Wait for async copy loop after MPI comm. is done

```
$ make
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla poiss
[...]
srun ./poisson2d
Jacobi relaxation Calculation: 4096 x 4096 mesh
[...]
Num GPUs: 4.
4096x4096: 1 GPU:    4.6780 s,
MPI time:    0.0846 s, inter G
```

Make Targets:

```
run:          run poisson2d (default)
poisson2d:    build poisson2d binary
profile:      profile with pgprof
*.solution:   same as above with solution
               (poisson2d.solution.*)
```

Task 1: Initial Version



```
[screen 0: bash]
Calculate reference solution and time serial execution.
  0, 0.250000
 100, 0.249940
 200, 0.249880
 300, 0.249821
 400, 0.249761
 500, 0.249702
 600, 0.249642
 700, 0.249583
 800, 0.249524
 900, 0.249464
Parallel execution.
  0, 0.250000
 100, 0.249940
 200, 0.249880
 300, 0.249821
 400, 0.249761
 500, 0.249702
 600, 0.249642
 700, 0.249583
 800, 0.249524
 900, 0.249464
Num GPUs: 4.
4096x4096: 1 GPU:  18.8920 s, 4 GPUs:  18.9223 s, speedup:    1.00 efficiency:
          24.96%
MPI time:   0.0001 s, inter GPU BW:  2080.28 GiB/s
bash-4.2$
```

Task 1: Solution

```
//Initialize MPI and determine rank and size
```

```
MPI_Init(&argc, &argv);
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
#pragma acc set device_num( rank )
```

```
real* restrict const A = (real*) malloc(nx*ny*sizeof(real));
```

```
real* restrict const Aref = (real*) malloc(nx*ny*sizeof(real));
```

```
real* restrict const Anew = (real*) malloc(nx*ny*sizeof(real));
```

```
real* restrict const rhs = (real*) malloc(nx*ny*sizeof(real));
```

Task 1: Solution

```
// Ensure correctness if ny%size != 0
int chunk_size = ceil( (1.0*ny)/size );
int iy_start = rank * chunk_size;
int iy_end = iy_start + chunk_size;
// Do not process boundaries
iy_start = max( iy_start, 1 );
iy_end = min( iy_end, ny - 1 );
```

Task 1: Solution

```
int top = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#pragma acc host_data use_device( A ) {
    //1. Sent row iy_start (first modified row) to top receive lower boundary (iy_end)
    //from bottom
    MPI_Sendrecv( A+iy_start*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                  A+iy_end*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE );

    //2. Sent row (iy_end-1) (last modified row) to bottom
    //receive upper boundary (iy_start-1) from top
    MPI_Sendrecv( A+(iy_end-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                  A+(iy_start-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
```

Task 1: Solution

```

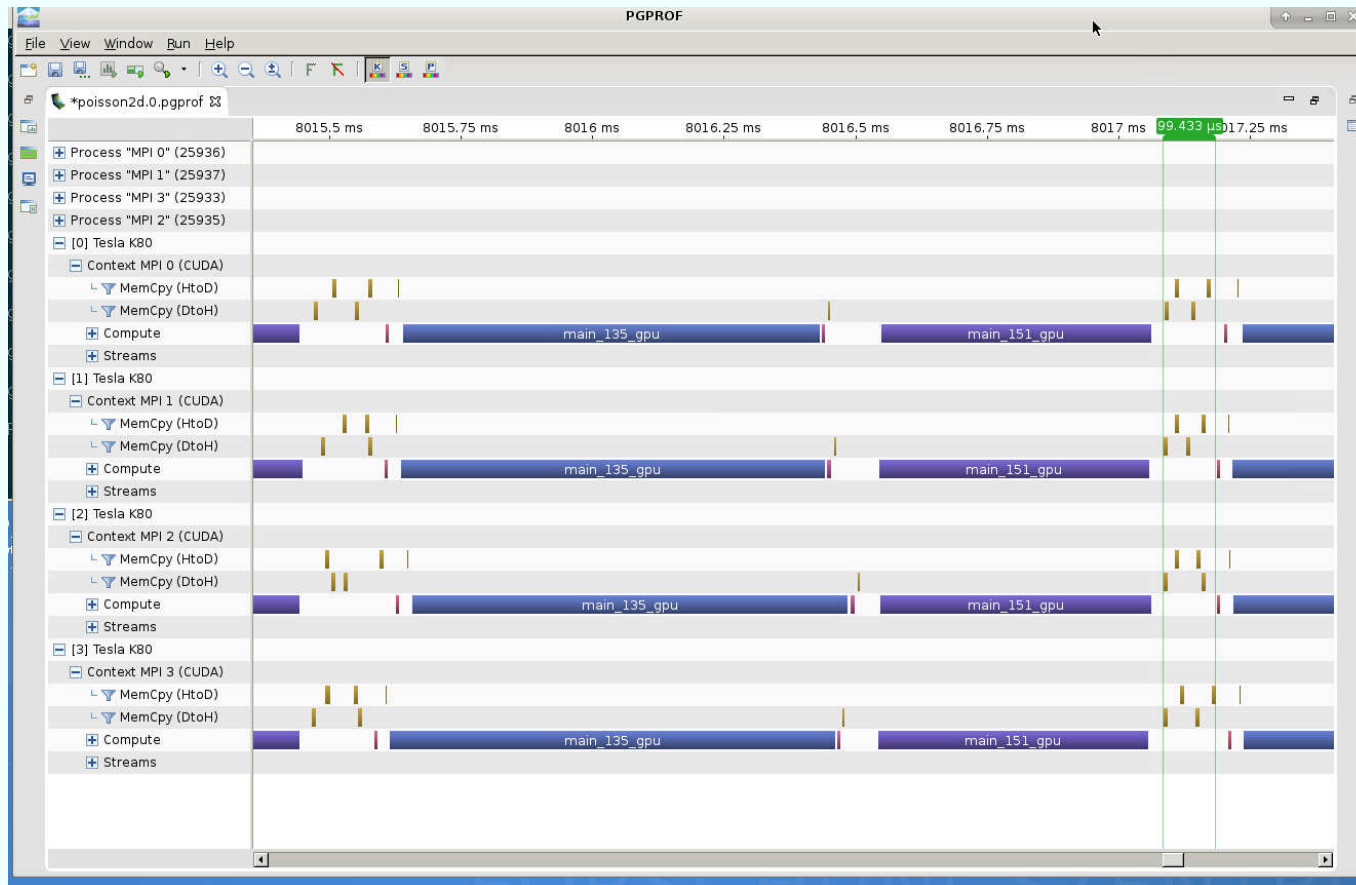
[screen 0: bash]
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Warning *** The GPU and IB selected are not on the same socket.
*** This configuration may not deliver the best performance.
Warning *** The GPU and IB selected are not on the same socket.
*** This configuration may not deliver the best performance.
Parallel execution.
  0, 0.250000
100, 0.249940
200, 0.249880
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Num GPUs: 4.
4096x4096: 1 GPU:   4.6872 s, 4 GPUs:   1.3239 s, speedup:    3.54 efficiency:
          88.51%
MPI time:   0.0840 s, inter GPU BW:   1.45 GiB/s
bash-4.2$
  
```

Task 2: Initial Version

```

jureca.fz-juelich.de - PuTTY
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Warning *** The GPU and IB selected are not on the same socket.
*** This configuration may not deliver the best performance.
Warning *** The GPU and IB selected are not on the same socket.
*** This configuration may not deliver the best performance.
Parallel execution.
  0, 0.250000
100, 0.249940
200, 0.249880
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Num GPUs: 4.
4096x4096: 1 GPU:   4.6595 s, 4 GPUs:   1.3242 s, speedup:   3.52, efficiency:
87.97%
MPI time:   0.0835 s, inter GPU BW:   1.46 GiB/s
bash-4.2$
  
```

Task 2: Initial Version



Task 2: Solution

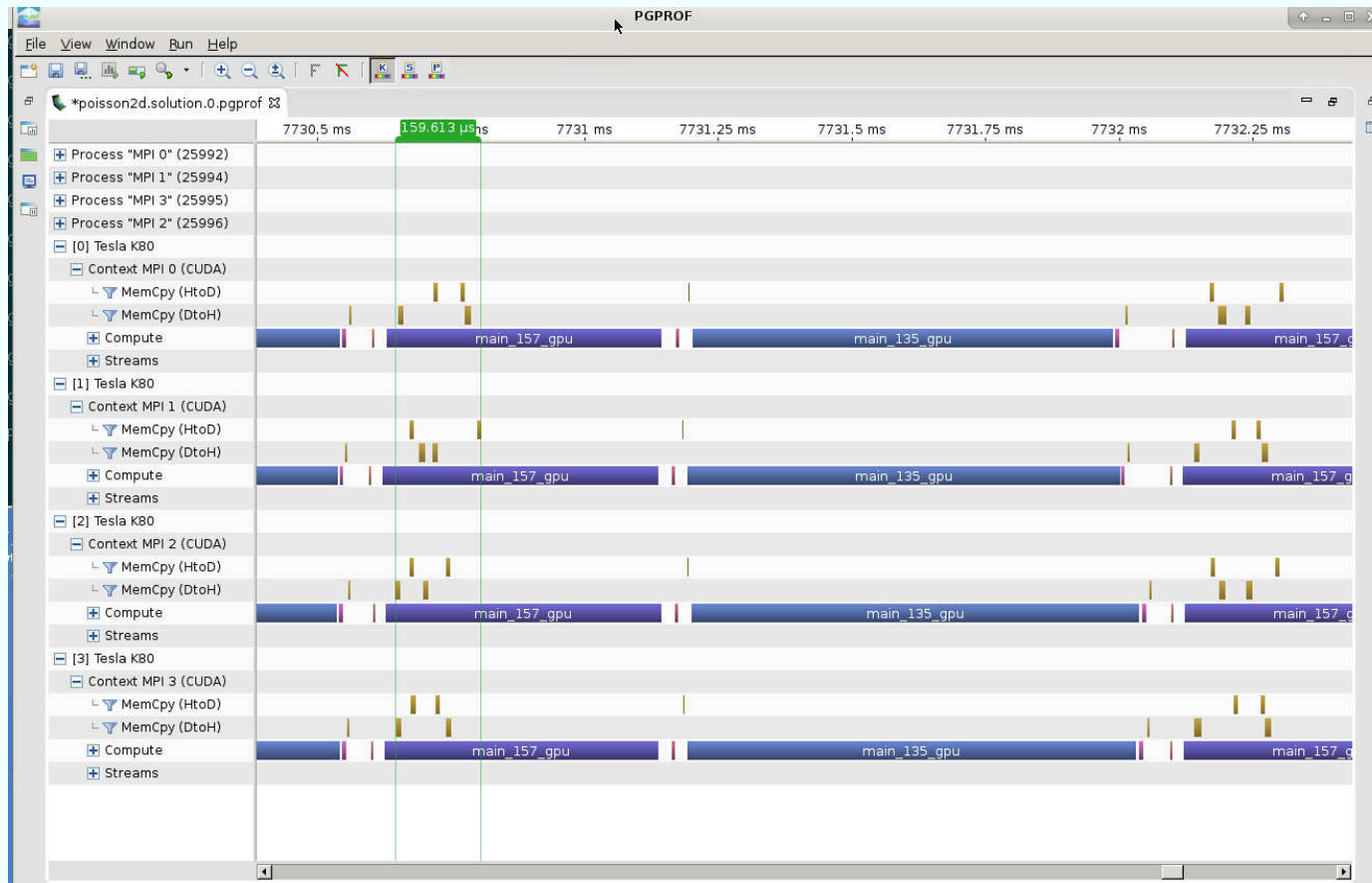
```
#pragma acc parallel loop present(A,Anew)
for( int ix = ix_start; ix < ix_end; ix++ ) {
    A[(iy_start)*nx+ix] = Anew[(iy_start)*nx+ix];
    A[(iy_end-1)*nx+ix] = Anew[(iy_end-1)*nx+ix];
}
#pragma acc parallel loop present(A,Anew) async
for (int iy = iy_start+1; iy < iy_end-1; iy++) {
    for( int ix = ix_start; ix < ix_end; ix++ ) {
        A[iy*nx+ix] = Anew[iy*nx+ix];
    }
}
int top = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#pragma acc host_data use_device( A )
{
    MPI_Sendrecv( A+iy_start*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                  A+iy_end*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE );
    MPI_Sendrecv( A+(iy_end-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, bottom, 0,
                  A+(iy_start-1)*nx+ix_start, (ix_end-ix_start), MPI_REAL_TYPE, top , 0,
                  MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
#pragma acc wait
```

Task 2: Solution

```

jureca.fz-juelich.de - PuTTY
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Warning *** The GPU and IB selected are not on the same socket.
*** This configuration may not deliver the best performance.
Warning *** The GPU and IB selected are not on the same socket.
*** This configuration may not deliver the best performance.
Parallel execution.
0, 0.250000
100, 0.249940
200, 0.249880
300, 0.249821
400, 0.249761
500, 0.249702
600, 0.249642
700, 0.249583
800, 0.249524
900, 0.249464
Num GPUs: 4.
4096x4096: 1 GPU: 4.6648 s, 4 GPUs: 1.2595 s, speedup: 3.70, efficiency:
92.59%
MPI time: 0.0979 s, inter GPU BW: 1.25 GiB/s
bash-4.2$
  
```

Task 2: Solution



Communication + Computation Overlap

MVAPICH2-GDR 2.2b – PGI 16.9 - 2 Tesla K80

