

HandsOnGPUProgramming_Solution

November 17, 2019

1 Hands-On GPU Programming

Supercomputing 2019 Tutorial "Application Porting and Optimization on GPU-Accelerated POWER Architectures", November 18th 2019

1.1 Solutions

This contains the output for the solutions.

The solutions are described in the solution section. Please navigate to the corresponding directory to find the solution profiles and sources.

1.1.1 GPU Programming

- Section ?? Accelerate a CPU Jacobi solver with OpenACC relying on Unified Memory for data movement using `ta=tesla:managed`
- Section ?? Fix memory access pattern of OpenACC accelerated Jacobi Solver

1.1.2 Multi-GPU with MPI

- Section ?? Use MPI to make OpenACC accelerated Jacobi Solver scale to multiple GPUs
- Section ?? Hide MPI communication time by overlapping communication and computation in a MPI+OpenACC multi GPU Jacobi Solver

1.1.3 Multi-GPU with NVSHMEM (*Advanced -- C only*)

- Section ?? Use NVSHMEM instead of MPI
- Section ?? Put NVSHMEM calls on stream to hide API calls and GPU/CPU synchronization

1.1.4 Survey

- Please remember to take the Section 3 !

1.2 Setup

Please **select your language choice (C or FORTRAN) below** by making sure your choice is uncommented and comment out the other language. Then execute the cell by hitting **Shift+Enter**!

```
In [1]: # select language here
        LANGUAGE='C'
        #LANGUAGE='FORTRAN'

        ## You should not touch the remaining code in the cell
        import os.path
        import pandas

        try: rootdir
        except NameError: rootdir = None
        if(not rootdir):
            rootdir=%pwd
        basedir=os.path.join(rootdir,LANGUAGE)
        basedirC=os.path.join(rootdir,'C')

        print ("You selected {} for the exercises.".format(LANGUAGE))

        def checkdir(dir):
            d=%pwd
            assert(d.endswith(dir) or d.endswith(dir+'p') or d.endswith(dir+'m')), "Please make"

        def cleanall():
            # clean up everything -- use with care
            for t in range(4):
                d='%s/task%i'%(basedir,t)
                %cd $d
                !make clean

        #cleanall()
```

You selected C for the exercises.

```
In [2]: %cd $basedir/task0
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task0
```

2 Solutions

Below are suggested solutions. This is only a short description of the solution, but the `poisson2d.solution.(c|F03)` files linked below have the full source code. If you want to run / profile the solutions feel free to duplicate the cells for the tasks and change the Section ?? to the *.solution ones.

2.1 Solution 0:

```
#pragma acc parallel loop
for (int ix = ix_start; ix < ix_end; ix++)
{
    #pragma acc loop
    for( int iy = iy_start; iy < iy_end; iy++ )
    {
        Anew[iy*nx+ix] = -0.25 * (rhs[iy*nx+ix] - ( A[iy*nx+ix+1] + A[iy*nx+ix-1]
                                                    + A[(iy-1)*nx+ix] + A[(iy+1)*nx+ix] ));
        error = fmaxr( error, fabsr(Anew[iy*nx+ix]-A[iy*nx+ix]));
    }
}
```

Code

- [C Version](#)
- [Fortran Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. **After** the profiling finished the output file `poisson2d.solution.pgprof` can be downloaded from here: [C Version](#) / [Fortran Version](#).

```
In [3]: %cd $basedir/task0
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task0
```

```
In [4]: checkdir('task0')
        !make poisson2d.solution
```

```
pgcc -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,managed poisson2d_serial.c -o pois
pgcc -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,managed poisson2d.solution.c poisson2
poisson2d.solution.c:
```

```
main:
```

```
66, Generating Tesla code
67, #pragma acc loop gang /* blockIdx.x */
68, #pragma acc loop vector(128) /* threadIdx.x */
66, Generating implicit copyout(A[:])
68, Loop is parallelizable
88, Generating Tesla code
89, #pragma acc loop gang /* blockIdx.x */
90, #pragma acc loop vector(128) /* threadIdx.x */
94, Generating implicit reduction(max:error)
88, Generating implicit copyin(A[:],rhs[:])
```

```

    Generating implicit copyout(Anew[:])
90, Loop is parallelizable
98, Generating Tesla code
    99, #pragma acc loop gang /* blockIdx.x */
    100, #pragma acc loop vector(128) /* threadIdx.x */
98, Generating implicit copyin(Anew[:])
    Generating implicit copyout(A[:])
100, Loop is parallelizable
106, Generating Tesla code
    107, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
106, Generating implicit copyin(A[:])
    Generating implicit copyout(A[nx*(ny-1)+1:2046])
111, Generating Tesla code
    112, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
111, Generating implicit copy(A[:])

```

```

In [5]: checkdir('task0')
        !make run.solution

```

```

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS ./poisson2d.solution
Job <25658> is submitted to default queue <batch>.

```

```

<<Waiting for dispatch ...>>

```

```

<<Starting on login1>>

```

```

Jacobi relaxation Calculation: 2048 x 2048 mesh

```

```

Calculate reference solution and time serial CPU execution.

```

```

    0, 0.249999
    100, 0.249760
    200, 0.249522
    300, 0.249285
    400, 0.249048

```

```

GPU execution.

```

```

    0, 0.249999
    100, 0.249760
    200, 0.249522
    300, 0.249285
    400, 0.249048

```

```

2048x2048: 1 CPU:   5.4111 s, 1 GPU:   0.1905 s, speedup:   28.40

```

```

In [6]: checkdir('task0')
        !make profile.solution

```

```

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS pgprof -f --cpu-profiling off --
Job <25659> is submitted to default queue <batch>.

```

```

<<Waiting for dispatch ...>>

```

```

<<Starting on login1>>

```

```

==77763== PGPROF is profiling process 77763, command: ./poisson2d.solution 10

```

```

==77763== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.pgprof

```

```

Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial CPU execution.
    0, 0.249999
GPU execution.
    0, 0.249999
2048x2048: 1 CPU:   0.1194 s, 1 GPU:   0.0179 s, speedup:    6.67
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.pgprof .

```

Section ??

2.2 Solution 1:

Swap the ix and iy loops to make sure that ix is the fastest running index

```

#pragma acc parallel loop
for (int iy = iy_start; iy < iy_end; iy++)
{
    for( int ix = ix_start; ix < ix_end; ix++ )
    {
        Anew[iy*nx+ix] = -0.25 * (rhs[iy*nx+ix] - ( A[iy*nx+ix+1] + A[iy*nx+ix-1]
                                                    + A[(iy-1)*nx+ix] + A[(iy+1)*nx+ix] ));
        error = fmaxr( error, fabsr(Anew[iy*nx+ix]-A[iy*nx+ix]));
    }
}

```

Code

- [C Version](#)
- [Fortran Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. **After** the profiling finished the output file `poisson2d.solution.pgprof` can be downloaded from here: [C Version](#) / [Fortran Version](#).

```
In [7]: %cd $basedir/task1
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task1
```

```
In [8]: checkdir('task1')
        !make poisson2d.solution
```

```
pgcc -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,managed,lineinfo poisson2d_serial.c
pgcc -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,managed,lineinfo poisson2d.solution.c
poisson2d.solution.c:
main:
```

```

66, Generating Tesla code
    67, #pragma acc loop gang /* blockIdx.x */
    68, #pragma acc loop vector(128) /* threadIdx.x */
66, Generating implicit copyout(A[:])
68, Loop is parallelizable
88, Generating Tesla code
    89, #pragma acc loop gang /* blockIdx.x */
    90, #pragma acc loop vector(128) /* threadIdx.x */
    94, Generating implicit reduction(max:error)
88, Generating implicit copyin(A[:],rhs[:])
    Generating implicit copyout(Anew[:])
90, Loop is parallelizable
98, Generating Tesla code
    99, #pragma acc loop gang /* blockIdx.x */
    100, #pragma acc loop vector(128) /* threadIdx.x */
98, Generating implicit copyin(Anew[:])
    Generating implicit copyout(A[:])
100, Loop is parallelizable
106, Generating Tesla code
    107, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
106, Generating implicit copyin(A[:])
    Generating implicit copyout(A[nx*(ny-1)+1:2046])
111, Generating Tesla code
    112, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
111, Generating implicit copy(A[:])

```

```

In [9]: checkdir('task1')
        !make run.solution

```

```

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS ./poisson2d.solution
Job <25660> is submitted to default queue <batch>.

```

```

<<Waiting for dispatch ...>>

```

```

<<Starting on login1>>

```

```

Jacobi relaxation Calculation: 2048 x 2048 mesh

```

```

Calculate reference solution and time serial CPU execution.

```

```

    0, 0.249999
    100, 0.249760
    200, 0.249522
    300, 0.249285
    400, 0.249048

```

```

GPU execution.

```

```

    0, 0.249999
    100, 0.249760
    200, 0.249522
    300, 0.249285
    400, 0.249048

```

```

2048x2048: 1 CPU: 5.3929 s, 1 GPU: 0.1903 s, speedup: 28.33

```

```

In [10]: checkdir('task1')
         !make profile.solution

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS pgprof -f --cpu-profiling off --
Job <25661> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
==77997== PGPROF is profiling process 77997, command: ./poisson2d.solution 3
==77997== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.timeline
Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial CPU execution.
    0, 0.249999
GPU execution.
    0, 0.249999
2048x2048: 1 CPU:  0.0437 s, 1 GPU:  0.0164 s, speedup:      2.66
bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS pgprof -f --cpu-profiling off --
Job <25662> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
==79400== PGPROF is profiling process 79400, command: ./poisson2d.solution 3
==79400== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==79400== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.metrics
Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial CPU execution.
    0, 0.249999
GPU execution.
    0, 0.249999
2048x2048: 1 CPU:  0.0475 s, 1 GPU: 12.3314 s, speedup:      0.00
bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS pgprof -f --cpu-profiling off --
Job <25663> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
==78235== PGPROF is profiling process 78235, command: ./poisson2d.solution 3
==78235== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==78235== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.efficiency
Jacobi relaxation Calculation: 2048 x 2048 mesh
Calculate reference solution and time serial CPU execution.
    0, 0.249999
GPU execution.
    0, 0.249999
2048x2048: 1 CPU:  0.0483 s, 1 GPU:  0.6638 s, speedup:      0.07
pgprof --csv -i /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.efficiency.pgprof 2>&1
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.*.pgprof .
tar -cvzf pgprof.poisson2d.Task1.solution.tar.gz poisson2d.solution.*.pgprof
poisson2d.solution.efficiency.pgprof
poisson2d.solution.metrics.pgprof
poisson2d.solution.timeline.pgprof

```

For the *Global Memory Load/Store Efficiency* the make profile command also generated a CSV file that you can import and view with the cell below.

If you purely work in a terminal you can view the same output by running `pgprof -i poisson2d.efficiency.solution.pgprof`.

```
In [11]: data_frame_solution = pandas.read_csv('poisson2d.solution.efficiency.csv', sep=',')
        data_frame_solution
```

```
Out[11]:
```

	Device	Kernel	Invocations	Metric Name \
0	Tesla V100-SXM2-16GB (0)	main_98_gpu	3	gld_efficiency
1	Tesla V100-SXM2-16GB (0)	main_98_gpu	3	gst_efficiency
2	Tesla V100-SXM2-16GB (0)	main_106_gpu	3	gld_efficiency
3	Tesla V100-SXM2-16GB (0)	main_106_gpu	3	gst_efficiency
4	Tesla V100-SXM2-16GB (0)	main_94_gpu__red	3	gld_efficiency
5	Tesla V100-SXM2-16GB (0)	main_94_gpu__red	3	gst_efficiency
6	Tesla V100-SXM2-16GB (0)	main_66_gpu	1	gld_efficiency
7	Tesla V100-SXM2-16GB (0)	main_66_gpu	1	gst_efficiency
8	Tesla V100-SXM2-16GB (0)	main_88_gpu	3	gld_efficiency
9	Tesla V100-SXM2-16GB (0)	main_88_gpu	3	gst_efficiency
10	Tesla V100-SXM2-16GB (0)	main_111_gpu	3	gld_efficiency
11	Tesla V100-SXM2-16GB (0)	main_111_gpu	3	gst_efficiency

	Metric Description	Min	Max	Avg
0	Global Memory Load Efficiency	90.866222%	91.051373%	90.962535%
1	Global Memory Store Efficiency	88.956522%	88.956522%	88.956522%
2	Global Memory Load Efficiency	94.722222%	94.722222%	94.722222%
3	Global Memory Store Efficiency	88.956522%	88.956522%	88.956522%
4	Global Memory Load Efficiency	99.756335%	99.756335%	99.756335%
5	Global Memory Store Efficiency	25.000000%	25.000000%	25.000000%
6	Global Memory Load Efficiency	0.000000%	0.000000%	0.000000%
7	Global Memory Store Efficiency	100.000000%	100.000000%	100.000000%
8	Global Memory Load Efficiency	91.850475%	91.857005%	91.854824%
9	Global Memory Store Efficiency	88.845486%	88.845486%	88.845486%
10	Global Memory Load Efficiency	25.000000%	25.000000%	25.000000%
11	Global Memory Store Efficiency	25.000000%	25.000000%	25.000000%

Section ??

2.3 Solution 2:

Set the GPU used by the rank using `#pragma acc set device_num`

```
//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));
```

Apply domain decomposition

```
// 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 );
```

Exchange data

```
//Periodic boundary conditions
int top    = (rank == 0) ? (size-1) : rank-1;
int bottom = (rank == (size-1)) ? 0 : rank+1;
#pragma acc host_data use_device( A )
{
    double start_mpi = MPI_Wtime();
    //1. Sent row iy_start (first modified row) to top receive lower boundary (iy_end) from bo
    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)
    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 );
    mpi_time += MPI_Wtime() - start_mpi;
}
```

Code

- [C Version](#)
- [Fortran Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task2.solution.poisson2d.tar.gz) with the File Browser. Then you can import them into pgprof / nvvp using the *Import* option in the *File* menu. Remember to use the *Multiple processes* option in the assistant.

```
In [12]: %cd $basedir/task2
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task2
```

```
In [13]: checkdir('task2')
```

```
!make poisson2d.solution
```

```
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d_serial.c -o poisson2d_serial:
```

```
36, Generating present(Anew[:,rhs:],Aref[:])
39, Generating update device(rhs[:ny*nx],Aref[:ny*nx])
42, Generating Tesla code
43, #pragma acc loop gang /* blockIdx.x */
44, #pragma acc loop vector(128) /* threadIdx.x */
49, Generating implicit reduction(max:error)
44, Loop is parallelizable
53, Generating Tesla code
54, #pragma acc loop gang /* blockIdx.x */
55, #pragma acc loop vector(128) /* threadIdx.x */
55, Loop is parallelizable
61, Generating Tesla code
62, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
66, Generating Tesla code
67, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
78, Generating update self(Aref[:ny*nx])
```

```
mpicc -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d.solution.c poisson2d.solution.c:
```

```
main:
```

```
71, Generating enter data create(Aref[:ny*nx],rhs[:ny*nx],A[:ny*nx],Anew[:ny*nx])
87, Generating present(Aref[:],A[:])
Generating Tesla code
88, #pragma acc loop gang /* blockIdx.x */
89, #pragma acc loop vector(128) /* threadIdx.x */
89, Loop is parallelizable
140, Generating update device(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)],rhs[nx*iy_start:])
143, Generating present(A[:,rhs:],Anew[:])
Generating Tesla code
144, #pragma acc loop gang /* blockIdx.x */
145, #pragma acc loop vector(128) /* threadIdx.x */
149, Generating implicit reduction(max:error)
145, Loop is parallelizable
157, Generating present(Anew[:,A[:])
Generating Tesla code
158, #pragma acc loop gang /* blockIdx.x */
159, #pragma acc loop vector(128) /* threadIdx.x */
159, Loop is parallelizable
184, Generating present(A[:])
```

```

Generating Tesla code
185, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
195, Generating update self(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)])
213, Generating exit data delete(rhs[:1],Aref[:1],A[:1],Anew[:1])

```

```
In [14]: checkdir('task2')
```

```
!NP=2 make run.solution
```

```
bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25664> is submitted to default queue <batch>.
```

```
<<Waiting for dispatch ...>>
```

```
<<Starting on login1>>
```

```
Jacobi relaxation Calculation: 4096 x 4096 mesh
```

```
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: 2.
```

```
4096x4096: 1 GPU: 1.3190 s, 2 GPUs: 0.7096 s, speedup: 1.86, efficiency: 92.94%
```

```
MPI time: 0.0424 s, inter GPU BW: 2.88 GiB/s
```

```
In [15]: checkdir('task2')
```

```
!NP=2 make profile.solution
```

```
bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25665> is submitted to default queue <batch>.
```

```
<<Waiting for dispatch ...>>
```

```
<<Starting on login1>>
```

```
==78468== PGPROF is profiling process 78468, command: ./poisson2d.solution 10
```

```

==78469== PGPROF is profiling process 78469, command: ./poisson2d.solution 10
==78469== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task2.1
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.
    0, 0.250000
Parallel execution.
    0, 0.250000
Num GPUs: 2.
4096x4096: 1 GPU:  0.0226 s, 2 GPUs:  0.0129 s, speedup:      1.75, efficiency:      87.45%
MPI time:  0.0007 s, inter GPU BW:      1.70 GiB/s
==78468== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task2.1
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task2.NP2.?.pgprof  .
tar -cvzf pgprof.poisson2d.Task2.solution.tar.gz poisson2d.solution.Task2.NP2.?.pgprof
poisson2d.solution.Task2.NP2.0.pgprof
poisson2d.solution.Task2.NP2.1.pgprof

```

Scaling You can do a simple scaling run for up to all 6 GPUs in the node by executing the next cell.

```

In [16]: checkdir('task2')
        !NP=1 make run.solution | grep speedup > scale.out
        !NP=2 make run.solution | grep speedup >> scale.out
        !NP=4 make run.solution | grep speedup >> scale.out
        !NP=6 make run.solution | grep speedup >> scale.out
        data_frameS2 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

        !rm scale.out

        data_frameS2b=data_frameS2.iloc[:,[5,7,10,12]].copy()
        data_frameS2b.rename(columns={5:'GPUs', 7: 'time [s]', 10:'speedup', 12:'efficiency'})

<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>

```

```

Out[16]:
   GPUs  time [s] speedup efficiency
0      1   1.4053   0.93,    93.06%
1      2   0.7154   1.83,    91.56%
2      4   0.4211   3.13,    78.21%
3      6   0.3121   4.20,    70.05%

```

Section ??

2.4 Solution 3:

Update the boundaries first.

```
#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];
}
```

Start the interior loop asynchronously so it can overlap with the MPI communication and wait at the end for the completion.

```
#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];
    }
}
```

```
//Periodic boundary conditions
```

```
int top    = (rank == 0) ? (size-1) : rank-1;
```

```
int bottom = (rank == (size-1)) ? 0 : rank+1;
```

```
#pragma acc host_data use_device( A )
```

```
{
```

```
    double start_mpi = MPI_Wtime();
```

```
    //1. Sent row iy_start (first modified row) to top receive lower boundary (iy_end) from bo
```

```
    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)
```

```
    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 );
```

```
    mpi_time += MPI_Wtime() - start_mpi;
```

```
}
```

```
#pragma acc wait
```

Code

- [C Version](#)
- [Fortran Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task2.solution.poisson2d.tar.gz) with the File Browser. Then you can import them into pgprof / nvvp using the *Import* option in the *File* menu. Remember to use the *Multiple processes* option in the assistant.

```
In [17]: %cd $basedir/task3
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task3
```

```
In [18]: checkdir('task3')
        !make poisson2d.solution
```

```
mpicc -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d_serial.c -o poisson2d_serial:
poisson2d_serial:
```

```
36, Generating present(Anew[:,rhs:],Aref[:])
39, Generating update device(rhs[:ny*nx],Aref[:ny*nx])
42, Generating Tesla code
43, #pragma acc loop gang /* blockIdx.x */
44, #pragma acc loop vector(128) /* threadIdx.x */
49, Generating implicit reduction(max:error)
44, Loop is parallelizable
53, Generating Tesla code
54, #pragma acc loop gang /* blockIdx.x */
55, #pragma acc loop vector(128) /* threadIdx.x */
55, Loop is parallelizable
61, Generating Tesla code
62, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
66, Generating Tesla code
67, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
78, Generating update self(Aref[:ny*nx])
```

```
mpicc -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d.solution.c poisson2d.solution.c:
poisson2d.solution.c:
```

```
main:
```

```
71, Generating enter data create(rhs[:ny*nx],Aref[:ny*nx],A[:ny*nx],Anew[:ny*nx])
87, Generating present(Aref[:],A[:])
Generating Tesla code
88, #pragma acc loop gang /* blockIdx.x */
89, #pragma acc loop vector(128) /* threadIdx.x */
89, Loop is parallelizable
140, Generating update device(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)],rhs[nx*iy_start:ny*iy_end])
143, Generating present(A[:,rhs:],Anew[:])
Generating Tesla code
144, #pragma acc loop gang /* blockIdx.x */
145, #pragma acc loop vector(128) /* threadIdx.x */
149, Generating implicit reduction(max:error)
145, Loop is parallelizable
157, Generating present(Anew[:,A[:])
```

```

    Generating Tesla code
158, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
163, Generating present(Anew[:],A[:])
    Generating Tesla code
164, #pragma acc loop gang /* blockIdx.x */
165, #pragma acc loop vector(128) /* threadIdx.x */
165, Loop is parallelizable
191, Generating present(A[:])
    Generating Tesla code
192, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
202, Generating update self(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)])
220, Generating exit data delete(rhs[:1],Aref[:1],A[:1],Anew[:1])

```

```

In [19]: checkdir('task3')
        !NP=2 make run.solution

```

```

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25670> is submitted to default queue <batch>.

```

```

<<Waiting for dispatch ...>>

```

```

<<Starting on login1>>

```

```

Jacobi relaxation Calculation: 4096 x 4096 mesh

```

```

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: 2.

```

```

4096x4096: 1 GPU: 1.3172 s, 2 GPUs: 0.6964 s, speedup: 1.89, efficiency: 94.57%

```

```

MPI time: 0.0561 s, inter GPU BW: 2.17 GiB/s

```

```

In [20]: checkdir('task3')
         !NP=2 make profile.solution

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25671> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
==79190== PGPROF is profiling process 79190, command: ./poisson2d.solution 10
==79192== PGPROF is profiling process 79192, command: ./poisson2d.solution 10
==79192== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task3.1
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.
    0, 0.250000
Parallel execution.
    0, 0.250000
Num GPUs: 2.
4096x4096: 1 GPU:  0.0301 s, 2 GPUs:  0.0126 s, speedup:      2.39, efficiency:   119.53%
MPI time:  0.0009 s, inter GPU BW:    1.34 GiB/s
==79190== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task3.1
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task3.NP2.?.pgprof .
tar -cvzf pgprof.poisson2d.Task3.solution.tar.gz poisson2d.solution.Task3.NP2.?.pgprof
poisson2d.solution.Task3.NP2.0.pgprof
poisson2d.solution.Task3.NP2.1.pgprof

```

Scaling You can do a simple scaling run for up to all 6 GPUs in the node by executing the next cell.

```

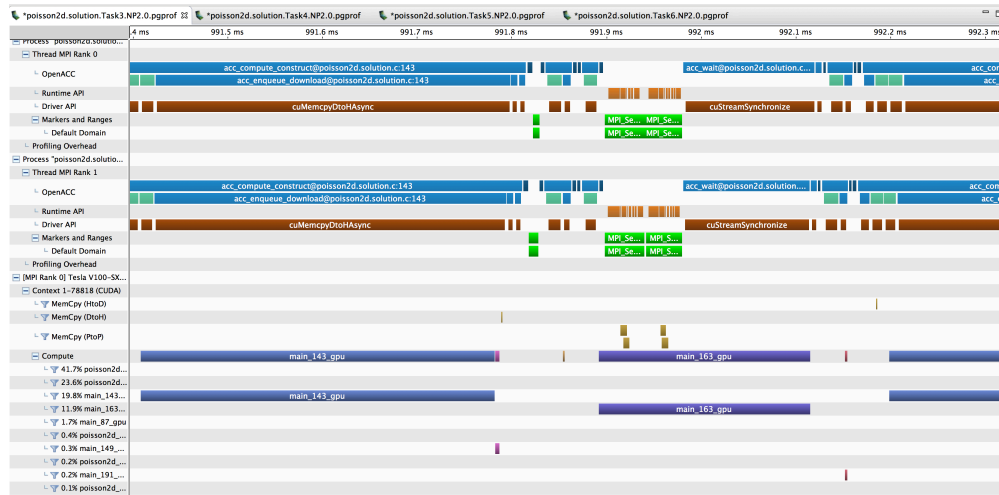
In [21]: checkdir('task3')
         !NP=1 make run.solution | grep speedup > scale.out
         !NP=2 make run.solution | grep speedup >> scale.out
         !NP=4 make run.solution | grep speedup >> scale.out
         !NP=6 make run.solution | grep speedup >> scale.out
         data_frameS3 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

         !rm scale.out

         data_frameS3b=data_frameS3.iloc[:, [5,7,10,12]].copy()
         data_frameS3b.rename(columns={5: 'GPUs', 7: 'time [s]', 10: 'speedup', 12: 'efficiency'})

<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>

```

Solution3.png

```
Out[21]:
```

	GPUs	time [s]	speedup	efficiency
0	1	1.3815	0.95,	94.79%
1	2	0.6968	1.90,	94.91%
2	4	0.3990	3.30,	82.56%
3	6	0.2720	4.81,	80.18%

The overlap of compute and communication can be seen in the profiler, e.g. as shown below.
Section ??

2.5 Solution 4:

First, include NVSHMEM headers

```
#include <nvshmem.h>
#include <nvshmemx.h>
```

and initialize NVSHMEM with MPI

```
MPI_Comm mpi_comm = MPI_COMM_WORLD;
nvshmemx_init_attr_t attr;
attr.mpi_comm = &mpi_comm;
nvshmemx_init_attr(NVSHMEMX_INIT_WITH_MPI_COMM, &attr);
```

Allocate device memory and map it top the host allocation for OpenACC

```
real *d_A = (real *)nvshmem_malloc(nx * ny * sizeof(real));
map(A, d_A, nx * ny * sizeof(real));
```

Calculate the right locations on the remote GPUs and communicate data

```

// Periodic boundary conditions
int top = (rank == 0) ? (size - 1) : rank - 1;
int bottom = (rank == (size - 1)) ? 0 : rank + 1;
int iy_start_top = top * chunk_size;
int iy_end_top = iy_start_top + chunk_size;

// Do not process boundaries
iy_start_top = max(iy_start_top, 1);
iy_end_top = min(iy_end_top, ny - 1);

int iy_start_bottom = bottom * chunk_size;
int iy_end_bottom = iy_start_bottom + chunk_size;

// Do not process boundaries
iy_start_bottom = max(iy_start_bottom, 1);
iy_end_bottom = min(iy_end_bottom, ny - 1);

// Halo exchange
#pragma acc host_data use_device(A)
{
    double start_mpi = MPI_Wtime();
    nvshmem_double_put((double *) (A + iy_end_top * nx + ix_start),
                      (double *) (A + iy_start * nx + ix_start), (ix_end - ix_start), top);
    nvshmem_double_put((double *) (A + (iy_start_bottom - 1) * nx + ix_start),
                      (double *) (A + (iy_end - 1) * nx + ix_start), (ix_end - ix_start),
                      bottom);
    nvshmem_barrier_all();
    mpi_time += MPI_Wtime() - start_mpi;
}

```

Finally, remember to deallocate:

```
nvshmem_free(d_A);
```

Code

- [C Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task4.solution.poisson2d.tar.gz) with the File Browser. Then you can import them into pgprof / nvvp using the *Import* option in the *File* menu. Remember to use the *Multiple processes* option in the assistant.

```
In [22]: %cd $basedir/task4
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task4
```

```

In [23]: checkdir('task4')
         !make poisson2d.solution

mpicxx -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d_serial.c -o po
poisson2d_serial(int, int, double, double *, double *, int, int, const double *):
    37, Generating present(Anew[:],rhs[:],Aref[:])
    39, Generating update device(rhs[:ny*nx],Aref[:ny*nx])
    40, Generating Tesla code
        43, #pragma acc loop gang /* blockIdx.x */
        44, #pragma acc loop vector(128) /* threadIdx.x */
        49, Generating implicit reduction(max:error)
    44, Loop is parallelizable
    51, Generating Tesla code
        54, #pragma acc loop gang /* blockIdx.x */
        55, #pragma acc loop vector(128) /* threadIdx.x */
    55, Loop is parallelizable
    58, Generating Tesla code
        62, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    65, Generating Tesla code
        67, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    77, Generating update self(Aref[:ny*nx])
mpicxx -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned -I/gpfs/wolf/trn003/world-sl
poisson2d.solution.c:
main:
    90, Generating enter data create(Aref[:ny*nx],rhs[:ny*nx],A[:ny*nx],Anew[:ny*nx])
   101, Generating present(Aref[:],A[:])
        Generating Tesla code
        105, #pragma acc loop gang /* blockIdx.x */
        106, #pragma acc loop vector(128) /* threadIdx.x */
   106, Loop is parallelizable
   162, Generating update device(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)],rhs[nx*iy_start:]
   163, Generating present(A[:],rhs[:],Anew[:])
        Generating Tesla code
        166, #pragma acc loop gang /* blockIdx.x */
        167, #pragma acc loop vector(128) /* threadIdx.x */
        171, Generating implicit reduction(max:error)
   167, Loop is parallelizable
   177, Generating present(Anew[:],A[:])
        Generating Tesla code
        180, #pragma acc loop gang /* blockIdx.x */
        181, #pragma acc loop vector(128) /* threadIdx.x */
   181, Loop is parallelizable
   214, Generating present(A[:])
        Generating Tesla code
        217, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
   227, Generating update self(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)])
   246, Generating exit data delete(rhs[:1],Aref[:1],A[:1],Anew[:1])

```

```

In [24]: checkdir('task4')
         !NP=2 make run.solution

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25676> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
WARN: IB HCA and GPU are not connected to a PCIe switch so IB performance can be limited depen
Jacobi relaxation Calculation: 4096 x 4096 mesh
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: 2.
4096x4096: 1 GPU:   1.3188 s, 2 GPUs:   0.7398 s, speedup:   1.78, efficiency:   89.13%
MPI time:   0.0644 s, inter GPU BW:   1.90 GiB/s

```

```

In [25]: checkdir('task4')
         !NP=2 make profile.solution

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25677> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
==79915== PGPROF is profiling process 79915, command: ./poisson2d.solution 10
==79914== PGPROF is profiling process 79914, command: ./poisson2d.solution 10
WARN: IB HCA and GPU are not connected to a PCIe switch so IB performance can be limited depen
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.

```

```

0, 0.250000
Parallel execution.
0, 0.250000
Num GPUs: 2.
4096x4096: 1 GPU: 0.0226 s, 2 GPUs: 0.0131 s, speedup: 1.72, efficiency: 86.13%
MPI time: 0.0010 s, inter GPU BW: 1.27 GiB/s
==79915== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task4.1
==79914== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task4.1
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task4.NP2.?.pgprof .
tar -cvzf pgprof.poisson2d.Task4.solution.tar.gz poisson2d.solution.Task4.NP2.?.pgprof
poisson2d.solution.Task4.NP2.0.pgprof
poisson2d.solution.Task4.NP2.1.pgprof

```

Scaling You can do a simple scaling run for up to all 6 GPUs in the node by executing the next cell.

```

In [26]: checkdir('task4')
!NP=1 make run.solution | grep speedup > scale.out
!NP=2 make run.solution | grep speedup >> scale.out
!NP=4 make run.solution | grep speedup >> scale.out
!NP=6 make run.solution | grep speedup >> scale.out
data_frameS4 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

!rm scale.out

data_frameS4b=data_frameS4.iloc[:,[5,7,10,12]].copy()
data_frameS4b.rename(columns={5:'GPUs', 7: 'time [s]', 10:'speedup', 12:'efficiency'})

<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>

```

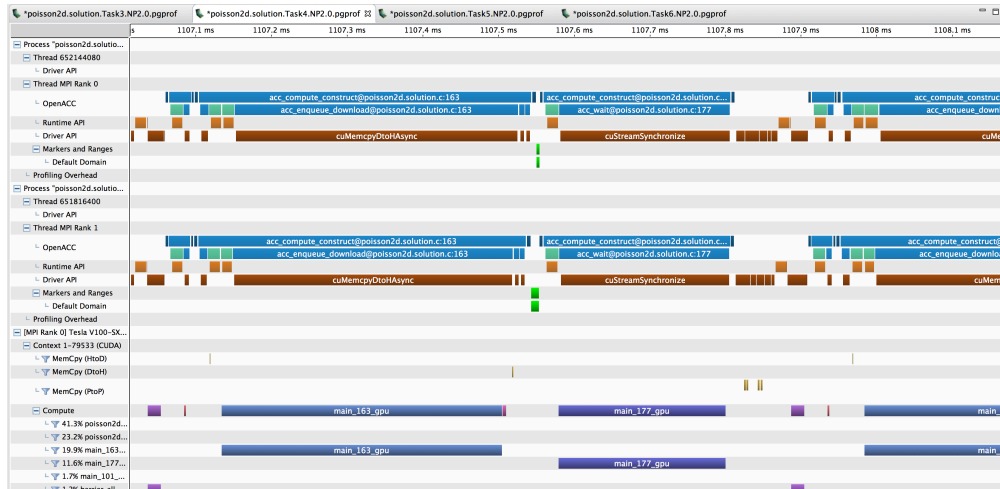
```

Out[26]:
  GPUs  time [s] speedup efficiency
0     1   1.3714   0.96,    95.91%
1     2   0.7460   1.76,    88.19%
2     4   0.4706   2.80,    70.05%
3     6   0.3308   3.91,    65.18%

```

The communication using NVSHMEM and the barrier executed as a kernel on the device can be seen in the profiler, e.g. as shown below.

Section ??



Solution4.png

2.6 Solution 5:

Basically all kernels in the while loop can use the async keyword. Please take a look in the solution source code. They will all use the OpenACC default async queue.

To also place the halo exchange in the queue use:

```
#pragma acc host_data use_device(A)
{
    nvshmemx_double_put_on_stream(
        (double *) (A + iy_end_top * nx + ix_start),
        (double *) (A + iy_start * nx + ix_start), (ix_end - ix_start), top,
        (cudaStream_t) acc_get_cuda_stream(acc_get_default_async()));
    nvshmemx_double_put_on_stream(
        (double *) (A + (iy_start_bottom - 1) * nx + ix_start),
        (double *) (A + (iy_end - 1) * nx + ix_start), (ix_end - ix_start), bottom,
        (cudaStream_t) acc_get_cuda_stream(acc_get_default_async()));
}
nvshmemx_barrier_all_on_stream((cudaStream_t) acc_get_cuda_stream(acc_get_default_async()));
```

Finally when copying out data make sure to wait on all device computation first:

```
#pragma acc update self(A [(iy_start - 1) * nx : ((iy_end - iy_start) + 2) * nx]) wait
```

Code

- [C Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task5.solution.poisson2d.tar.gz) with the File Browser. Then you can import them into pgprof / nvvp using the *Import* option in the *File* menu. Remember to use the *Multiple processes* option in the assistant.

```
In [27]: %cd $basedir/task5
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task5
```

```
In [28]: checkdir('task5')
```

```
!make poisson2d.solution
```

```
mpicxx -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d_serial.c -o poisson2d_serial(int, int, double, double *, double *, int, int, const double *):
```

```
37, Generating present(Anew[:,rhs:],Aref[:])
```

```
39, Generating update device(rhs[:ny*nx],Aref[:ny*nx])
```

```
40, Generating Tesla code
```

```
43, #pragma acc loop gang /* blockIdx.x */
```

```
44, #pragma acc loop vector(128) /* threadIdx.x */
```

```
49, Generating implicit reduction(max:error)
```

```
44, Loop is parallelizable
```

```
51, Generating Tesla code
```

```
54, #pragma acc loop gang /* blockIdx.x */
```

```
55, #pragma acc loop vector(128) /* threadIdx.x */
```

```
55, Loop is parallelizable
```

```
58, Generating Tesla code
```

```
62, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

```
65, Generating Tesla code
```

```
67, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

```
77, Generating update self(Aref[:ny*nx])
```

```
mpicxx -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned -I/ccsopen/home/mathiasw/nv  
poisson2d.solution.c:
```

```
main:
```

```
90, Generating enter data create(Aref[:ny*nx],rhs[:ny*nx],A[:ny*nx],Anew[:ny*nx])
```

```
101, Generating present(Aref[:,A[:])
```

```
Generating Tesla code
```

```
105, #pragma acc loop gang /* blockIdx.x */
```

```
106, #pragma acc loop vector(128) /* threadIdx.x */
```

```
106, Loop is parallelizable
```

```
137, Generating update device(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)],rhs[nx*iy_start:]
```

```
138, Generating present(A[:,rhs:],Anew[:])
```

```
Generating Tesla code
```

```
141, #pragma acc loop gang /* blockIdx.x */
```

```
142, #pragma acc loop vector(128) /* threadIdx.x */
```

```
146, Generating implicit reduction(max:error)
```

```
142, Loop is parallelizable
```

```
154, Generating present(Anew[:,A[:])
```

```
Generating Tesla code
```

```
157, #pragma acc loop gang /* blockIdx.x */
```

```
158, #pragma acc loop vector(128) /* threadIdx.x */
```

```
158, Loop is parallelizable
```

```
192, Generating present(A[:])
```

```

Generating Tesla code
195, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
205, Generating update self(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)])
223, Generating exit data delete(rhs[:1],Aref[:1],A[:1],Anew[:1])

```

In [29]: `checkdir('task5')`

```
!NP=2 make run.solution
```

```
bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25682> is submitted to default queue <batch>.
```

```
<<Waiting for dispatch ...>>
```

```
<<Starting on login1>>
```

```
WARN: IB HCA and GPU are not connected to a PCIe switch so IB performance can be limited depend
```

```
Jacobi relaxation Calculation: 4096 x 4096 mesh
```

```
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: 2.
```

```
4096x4096: 1 GPU: 1.3210 s, 2 GPUs: 0.6750 s, speedup: 1.96, efficiency: 97.86%
```

In [30]: `checkdir('task5')`

```
!NP=2 make profile.solution
```

```
bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25683> is submitted to default queue <batch>.
```

```
<<Waiting for dispatch ...>>
```

```
<<Starting on login1>>
```

```
==80646== PGPROF is profiling process 80646, command: ./poisson2d.solution 10
```



```

==80644== PGPROF is profiling process 80644, command: ./poisson2d.solution 10
==80646== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task5.1
WARN: IB HCA and GPU are not connected to a PCIe switch so IB performance can be limited depend
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.
    0, 0.250000
Parallel execution.
    0, 0.250000
Num GPUs: 2.
4096x4096: 1 GPU:  0.0227 s, 2 GPUs:  0.0120 s, speedup:      1.89, efficiency:    94.65%
==80644== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task5.1
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task5.NP2.?.pgprof .
tar -cvzf pgprof.poisson2d.Task5.solution.tar.gz poisson2d.solution.Task5.NP2.?.pgprof
poisson2d.solution.Task5.NP2.0.pgprof
poisson2d.solution.Task5.NP2.1.pgprof

```

Scaling You can do a simple scaling run for up to all 6 GPUs in the node by executing the next cell.

```

In [31]: checkdir('task5')
        !NP=1 make run.solution | grep speedup > scale.out
        !NP=2 make run.solution | grep speedup >> scale.out
        !NP=4 make run.solution | grep speedup >> scale.out
        !NP=6 make run.solution | grep speedup >> scale.out
        data_frameS5 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

        !rm scale.out

        data_frameS5b=data_frameS5.iloc[:, [5,7,10,12]].copy()
        data_frameS5b.rename(columns={5: 'GPUs', 7: 'time [s]', 10: 'speedup', 12: 'efficiency'})

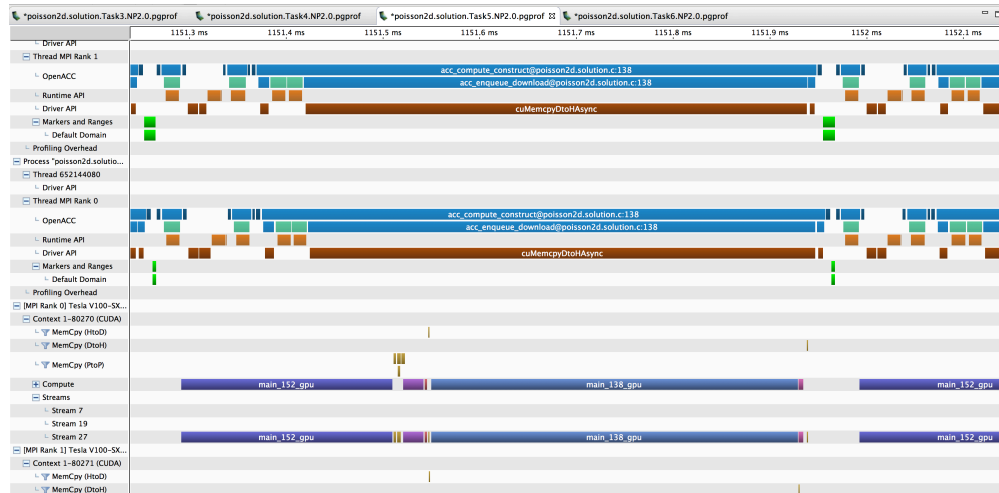
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>

```

```

Out[31]:
   GPUs  time [s]  speedup  efficiency
0      1    1.3004    1.01,    101.04%
1      2    0.6705    1.95,     97.67%
2      4    0.3879    3.41,     85.14%
3      6    0.2745    4.81,     80.25%

```



Solution5.png

The asynchronous execution and execution in the same stream can be seen in the profiler, e.g. as shown below.

Section ??

2.7 Solution 6:

The most important part here is to get an `nvshmem_ptr` pointing to the symmetric `d_A` allocation of your top and bottom neighbor.

```
real * restrict d_Atop = (real *)nvshmem_ptr(d_A, top);
real * restrict d_Abottom = (real *)nvshmem_ptr(d_A, bottom);
```

When updating A from Anew make sure to also update A on your top and bottom neighbor if you are at the boundary:

```
#pragma acc parallel loop present(A, Anew) deviceptr(d_Atop, d_Abottom) async
for (int iy = iy_start; iy < iy_end; iy++) {
    for (int ix = ix_start; ix < ix_end; ix++) {
        A[iy * nx + ix] = Anew[iy * nx + ix];
        if(iy == iy_start){// this also needs to go to the lower halo region of my upper neighbor
            d_Atop[iy_end_top * nx + ix] = Anew[iy * nx + ix];
        }
        if(iy == iy_end - 1){// this also needs to go to the upper halo region of my bottom neighbor
            d_Abottom[(iy_start_bottom - 1) * nx + ix] = Anew[iy * nx + ix];
        }
    }
}
```

We can then remove the explicit `nvshmem_put` calls on completely. But remember to still keep the barrier.

```
nvshmemx_barrier_all_on_stream((cudaStream_t)acc_get_cuda_stream(acc_get_default_async()));
```

Code

- [C Version](#)

Compiling, Running and Profiling You can compile, run and profile the solution with the next cells. You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task6.solution.poisson2d.tar.gz) with the File Browser. Then you can import them into pgprof / nvvp using the *Import* option in the *File* menu. Remember to use the *Multiple processes* option in the assistant.

```
In [32]: %cd $basedir/task6
```

```
/autofs/nccsopen-svm1_home/mathiasw/sc19-tutorial-openpower/4-GPU/HandsOn/Solution/C/task6
```

```
In [33]: checkdir('task6')
        !make poisson2d.solution
```

```
mpicxx -c -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned poisson2d_serial.c -o po
poisson2d_serial(int, int, double, double *, double *, int, int, const double *):
    37, Generating present(Anew[:,rhs:],Aref[:])
    39, Generating update device(rhs[:ny*nx],Aref[:ny*nx])
    40, Generating Tesla code
        43, #pragma acc loop gang /* blockIdx.x */
        44, #pragma acc loop vector(128) /* threadIdx.x */
        49, Generating implicit reduction(max:error)
    44, Loop is parallelizable
    51, Generating Tesla code
        54, #pragma acc loop gang /* blockIdx.x */
        55, #pragma acc loop vector(128) /* threadIdx.x */
    55, Loop is parallelizable
    58, Generating Tesla code
        62, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    65, Generating Tesla code
        67, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    77, Generating update self(Aref[:ny*nx])
mpicxx -DUSE_DOUBLE -Minfo=accel -fast -acc -ta=tesla:cc70,pinned -I/ccsopen/home/mathiasw/nv
poisson2d.solution.c:
main:
    95, Generating enter data create(Aref[:ny*nx],rhs[:ny*nx],A[:ny*nx],Anew[:ny*nx])
    106, Generating present(Aref[:,A[:])
        Generating Tesla code
        110, #pragma acc loop gang /* blockIdx.x */
        111, #pragma acc loop vector(128) /* threadIdx.x */
    111, Loop is parallelizable
    158, Generating update device(rhs[nx*iy_start:nx*(iy_end-iy_start)],A[nx*(iy_start-1):nx*(
    159, Generating present(A[:,rhs:],Anew[:])
        Generating Tesla code
        162, #pragma acc loop gang /* blockIdx.x */
```

```

163, #pragma acc loop vector(128) /* threadIdx.x */
167, Generating implicit reduction(max:error)
163, Loop is parallelizable
174, Generating present(Anew[:,A[:]])
    Generating Tesla code
177, #pragma acc loop gang /* blockIdx.x */
179, #pragma acc loop vector(128) /* threadIdx.x */
179, Loop is parallelizable
190, Generating present(A[:])
    Generating Tesla code
193, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
203, Generating update self(A[nx*(iy_start-1):nx*((iy_end-iy_start)+2)])
219, Generating exit data delete(rhs[:1],Aref[:1],A[:1],Anew[:1])

```

In [34]: checkdir('task6')

```
!NP=2 make run.solution
```

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25688> is submitted to default queue <batch>.

<<Waiting for dispatch ...>>

<<Starting on login1>>

WARN: IB HCA and GPU are not connected to a PCIe switch so IB performance can be limited depend

Jacobi relaxation Calculation: 4096 x 4096 mesh

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: 2.

4096x4096: 1 GPU: 1.3196 s, 2 GPUs: 0.6641 s, speedup: 1.99, efficiency: 99.34%

```

In [35]: checkdir('task6')
         !NP=2 make profile.solution

bsub -W 60 -nnodes 1 -Is -P TRN003 jsrun -n 1 -c 1 -g ALL_GPUS -a 2 -c ALL_CPUS -d cyclic -b p
Job <25689> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on login1>>
==81382== PGPROF is profiling process 81382, command: ./poisson2d.solution 10
==81383== PGPROF is profiling process 81383, command: ./poisson2d.solution 10
==81382== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task6.1
WARN: IB HCA and GPU are not connected to a PCIe switch so IB performance can be limited depend
Jacobi relaxation Calculation: 4096 x 4096 mesh
Calculate reference solution and time serial execution.
    0, 0.250000
Parallel execution.
    0, 0.250000
Num GPUs: 2.
4096x4096: 1 GPU:  0.0225 s, 2 GPUs:  0.0118 s, speedup:      1.91, efficiency:    95.50%
==81383== Generated result file: /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task6.1
mv /gpfs/wolf/trn003/scratch/mathiasw//poisson2d.solution.Task6.NP2.?.pgprof .
tar -cvzf pgprof.poisson2d.Task6.solution.tar.gz poisson2d.solution.Task6.NP2.?.pgprof
poisson2d.solution.Task6.NP2.0.pgprof
poisson2d.solution.Task6.NP2.1.pgprof

```

Scaling You can do a simple scaling run for up to all 6 GPUs in the node by executing the next cell.

```

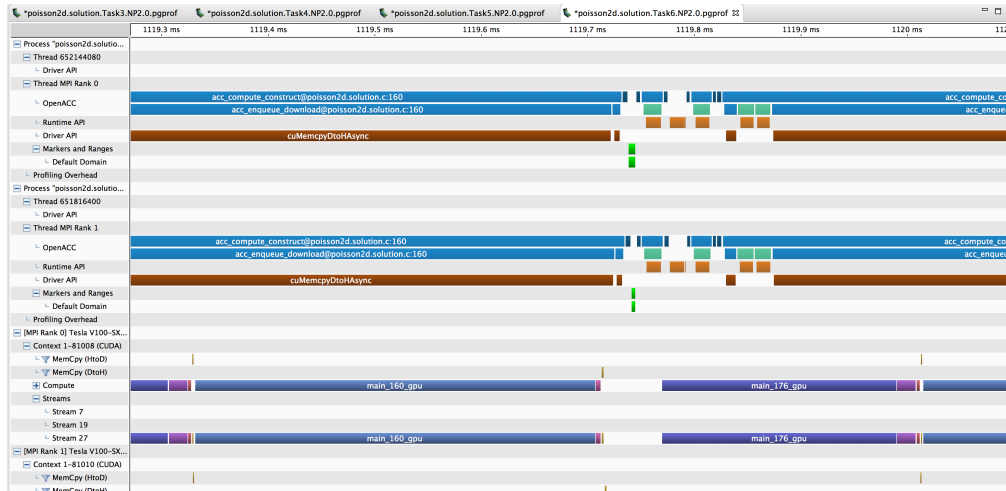
In [36]: checkdir('task6')
         !NP=1 make run.solution | grep speedup > scale.out
         !NP=2 make run.solution | grep speedup >> scale.out
         !NP=4 make run.solution | grep speedup >> scale.out
         !NP=6 make run.solution | grep speedup >> scale.out
         data_frameS5 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

         !rm scale.out

         data_frameS5b=data_frameS5.iloc[:,[5,7,10,12]].copy()
         data_frameS5b.rename(columns={5:'GPUs', 7: 'time [s]', 10:'speedup', 12:'efficiency'})

<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>
<<Waiting for dispatch ...>>
<<Starting on login1>>

```



Solution6.png



eval.png

```
Out [36]:   GPUs  time [s] speedup efficiency
          0     1    1.2964   1.01,   101.26%
          1     2    0.6714   1.94,    96.87%
          2     4    0.3810   3.46,    86.47%
          3     6    0.2641   4.87,    81.16%
```

The missing of device copies can be seen in the profiler, e.g. as shown below. There are only kernels running mostly back-to-back, only interrupted by the global reduction.

Section ??



3 Survey

Please remember to take some time and fill out the survey <http://bit.ly/sc19-eval>.