

HandsOnGPUProgramming

November 8, 2019

1 Hands-On GPU Programming

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

1.0.1 Read me first

This tutorial is primarily designed to be executed as a *jupyter* notebook. However, everything can also be done using an *ssh* connection to *ascent.olcf.ornl.gov* in your terminal.

Jupyter Lab execution When using jupyter this notebook will guide you through the step. Note that if you execute a cell multiple times while optimizing the code the output will be replaced. You can however duplicate the cell you want to execute and keep its output. Check the *edit* menu above.

You will always find links to a file browser of the corresponding task subdirectory as well as direct links to the source files you will need to edit as well as the profiling output you need to open locally.

If you want you also can get a terminal in your browser by following the *File -> New -> Terminal* in the Jupyter Lab menu bar.

Terminal fallback The tasks are placed in directories named `[C/FORTRAN]/task[0-6]`. *Note: The tasks using NVHSMEM (4-6) are only available in C.*

The files you will need to edit are always the `poisson2d.(C|F03)` files.

The makefile targets execute everything to compile, run and profile the code. Please take a look at the cells containing the make calls as a guide.

The outputs of profiling runs be placed in the working directory of the current task and are named like `*.pgprof` or `pgprof.*.tar.gz` in case of multiple files. You can use *scp/sftp* to transfer files to your machine and for viewing them in `pgprof/nvprof`.

Viewing profiles in the NVIDIA Visual Profiler / PGI Profiler The profiles generated *pgprof* / *nvprof* should be viewed on your local machine. You can install the PGI Community Edition (`pgprof`) or the NVIDIA CUDA Toolkit on your notebook (Windows, Mac, Linux). You don't need an NVIDIA GPU in your machine to use the profiler GUI.

There are USB Sticks in the room that contain the installers for various platforms, but for reference you can also download it from: * [NVIDIA CUDA Toolkit](#) * [PGI Community Edition](#)

After downloading the profiler output (more infos below) follow the steps outlined in: * [Import Session](#)

In case there is confusion: The PGI Profiler is a slightly modified version (different default settings) of the NVIDIA Visual Profiler. So you can use any of the two to view profiles.

1.1 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!**

```
[ ]: # 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 sure to cd to the right directory first."

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

#cleanall()
```

2 Tasks

This session comes with multiple tasks. All tasks are available in C or FORTRAN and can be found in the [C|Fortan]/task[0-3] subdirectories. There you will also find Makefiles that are set up so that you can compile and submit all necessary tasks.

Please choose from the task below. *If you want to go for the advanced NVSHMEM tasks you should complete Task 2 but can skip Task 3 (or postpone it until the end).*

2.0.1 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

2.0.2 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

2.0.3 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

2.0.4 Survey

- Section 3 Please remember to take the survey !

2.0.5 Make Targets

For all tasks we have defined the following make targets.

- **run:**
run poisson2d
- **poisson2d:**
build poisson2d binary (default)
- **profile:**
profile with pgprof
- **__*.solution__:**
same as above for the solution (e.g. `make poisson2d.solution` or `make run.solution`)

Section ??

2.1 Task 0: Using OpenACC

2.1.1 Description

The goal of this task is to accelerate a CPU Jacobi solver with OpenACC relying on Unified Memory for data movement using `-ta=tesla:managed`.

Your task is to:

- Parallelize Loops with OpenACC parallel loop

Look for **TODOs** in the code.

Look at the output generated by the PGI compiler (enabled by the `-Minfo=accel` option) to see how the compiler parallelizes the code.

Code You can open the source code either in a terminal in an editor. Navigate to `(C|Fortran)/task0/` and open `poisson2d.c` in a editor of your choice.

If your are using the jupyter approach by following the link (for the language of your choice), This will open the source code in an editor in a new browser tab/window.

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

Before executing any of the cells below first execute the next cell to change to the right directory.

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

Compilation and Execution If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call `make` with the desired Section `??`. Alternatively you can just navigate to the right directory and execute `make <target>` in your terminal.

```
[ ]: checkdir('task0')
!make
```

```
[ ]: checkdir('task0')
!make run
```

Profiling You can profile the code by executing the next cell. **After** the profiling finished the output file `poisson2d.pgprof` can be downloaded using the file browser. Then you can import them into `pgprof / nvvp` using the *Import* option in the *File* menu.

```
[ ]: checkdir('task0')
!make profile
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)

Section `??`

2.2 Task 1: Memory Access Patterns

2.2.1 Description

The goal of this task is to fix the memory access pattern of OpenACC accelerated Jacobi Solver. Generate the profile, download the generated profiles and import them into `pgprof / nvprof`. There

use “Global Memory Access Pattern” experiment to analyze the issue.

Look for **TODOs** in the code.

Code

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

Before executing any of the cells below first execute the next cell to change to the right directory.

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

Compilation and Execution If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call **make** with the desired Section **??**. Alternatively you can just navigate to the right directory and execute **make <target>** in your terminal.

```
[ ]: checkdir('task1')
!make
```

```
[ ]: checkdir('task1')
!make run
```

Profiling You can profile the code by executing the next cell. Download the tarball containing the profiles (**pgprof.Task1.poisson2d.tar.gz**) with the File Browser. Then you can import them into pgprof / nvvp using the *Import* option in the *File* menu.

```
[ ]: checkdir('task1')
!make profile
```

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.pgprof**.

```
[ ]: checkdir('task1')
data_frame = pandas.read_csv('poisson2d.efficiency.csv', sep=',')
data_frame
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)
3. [pgprof Quickstart](#)
4. [CUDA Toolkit Documentation - Profiler](#) *pgprof is based on the NVIDIA Visual Profiler*

Section **??**

2.3 Task 2: Apply Domain Decomposition

2.3.1 Description

Your task is to apply a domain decomposition and use MPI for the data exchange. Specifically you should * Handle GPU affinity * Do the Halo Exchange

Look for **TODOs**

When profiling take a look at how kernel and communication times change when you scale to more GPUs.

Code

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

Before executing any of the cells below first execute the next cell to change to the right directory.

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

Compilation If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call `make` with the desired Section `??`. Alternatively you can just navigate to the right directory and execute `make <target>` in your terminal.

```
[ ]: checkdir('task2')
!make poisson2d
```

Running For the Multi-GPU version you can set the number of GPUs / MPI ranks using the variable `NP`. On *Ascent* within a single node you can use up to 6 GPUs.

```
[ ]: checkdir('task2')
!NP=2 make run
```

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

```
[ ]: checkdir('task2')
!NP=1 make run | grep speedup > scale.out
!NP=2 make run | grep speedup >> scale.out
!NP=4 make run | grep speedup >> scale.out
!NP=6 make run | grep speedup >> scale.out
data_frame2 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

!rm scale.out

data_frame2b=data_frame2.iloc[:, [5,7,10,12]].copy()
data_frame2b.rename(columns={5: 'GPUs', 7: 'time [s]', 10: 'speedup', 12:
    ↳ 'efficiency'})
```

Profiling You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (`pgprof.Task2.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.

```
[ ]: checkdir('task2')
      !NP=2 make profile
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)
3. <https://www.open-mpi.org/doc/v3.1/>

Section ??

2.4 Task 3: Hide MPI Communication time

To overlap compute and communication you will need to

- start the copy loop asynchronously
- wait for async copy loop after MPI communication has finished

Look for **TODOs**.

Compare the scaling and efficiency with the results from the previous task. Check for the overlap in the profiler.

Optional: Try to understand how well communication and compute overlap is able to improve efficiency when scaling to more GPUs.

Code

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

Before executing any of the cells below first execute the next cell to change to the right directory.

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

Compilation If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call `make` with the desired Section `??`. Alternatively you can just navigate to the right directory and execute `make <target>` in your terminal.

```
[ ]: checkdir('task3')
      !make poisson2d
```

Running For the Multi-GPU version you can set the number of GPUs / MPI ranks using the variable NP. On *Ascent* within a single node you can use up to 6 GPUs.

```
[ ]: checkdir('task3')
      !NP=2 make run
```

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

```
[ ]: checkdir('task3')
      !NP=1 make run | grep speedup > scale.out
      !NP=2 make run | grep speedup >> scale.out
      !NP=4 make run | grep speedup >> scale.out
      !NP=6 make run | grep speedup >> scale.out
      data_frame3 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

      !rm scale.out

      data_frame3b=data_frame3.iloc[:,[5,7,10,12]].copy()
      data_frame3b.rename(columns={5:'GPUs', 7: 'time [s]', 10:'speedup', 12:
      ↪'efficiency'})
```

Profiling You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task3.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.

```
[ ]: checkdir('task3')
      !NP=2 make profile
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)
3. <https://www.open-mpi.org/doc/v3.1/>

2.5 Tasks using NVSHMEM

The following tasks are using NVSHMEM instead of MPI. NVSHMEM is currently available as early access software. Please read the following carefully before starting these tasks.

- *NVSHMEM early access 0.3.2* is installed on Ascent. It is provided under the license in [LICENSE_NVSHMEM.md](#).
 - If you want to continue using the NVSHMEM early access version beyond this tutorial you need to apply for early access at <https://developer.nvidia.com/nvshmem>
-

NVSHMEM enables efficient communication among GPUs. It supports an API for direct communication among GPUs, either initiated by the CPU or by GPUs inside of compute kernels. Inside compute kernels, NVSHMEM also supports direct load/store accesses to remote memory over PCIe or NVLink. The ability to initiate communication from inside kernels eliminates GPU-host-synchronization and associated overheads. It can also benefit from latency tolerance mechanisms available within GPUs. The tasks illustrate that progressing from an MPI-only app to an app that uses NVSHMEM can be straightforward.

NOTE: Covering all feature of NVSHMEM, including communication calls in kernels, is not easily accessible through OpenACC and also exceed the scope of this tutorial. However, the OpenACC examples should give you a basic introduction to NVSHMEM.

You can check the developer guide and the other presentations

References

1. <http://www.openacc.org>
 2. [OpenACC Reference Card](#)
 3. [OpenSHMEM 1.4 Specification](#)
 4. [NVSHMEM 0.3 EA Developer Guide](#)
-

2.6 Task 4: Replace MPI with host side NVSHMEM

To replace MPI from Section ?? with NVSHMEM you will need to connect the NVSHMEM library to MPI and replace all MPI communication calls related to the halo exchange with the corresponding NVSHMEM functions:

- Include NVSHMEM API header (`nvshmem.h`)
- Include NVSHMEM API extensions header (`nvshmemx.h`)
- Initialize NVSHMEM and connect to MPI (`nvshmemx_init_attr`)
- Allocate symmetric memory (`nvshmem_alloc`) for A on the device and use the OpenACC map function to tell OpenACC to use it as device memory for A
- Replace `MPI_Sendrecv` calls with SHMEM calls (`nvshmem_double_put`)
- Insert NVSHMEM barriers to ensure correct execution (`nvshmem_barrier_all`)

For interoperability with OpenSHMEM NVSHMEM can also be set up to prefix all calls to NVSHMEM with `nv`. Please make sure to use these version, e.g. use `nvshmem_barrier` instead of `shmem_barrier`. The developer guide mostly uses the unprefix versions.

Look for TODOs.

Code

- [C Version](#)

Before executing any of the cells below first execute the next cell to change to the right directory.

```
[ ]: %cd $basedirC/task4
```

Compilation If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call `make` with the desired Section `??`. Alternatively you can just navigate to the right directory and execute `make <target>` in your terminal.

```
[ ]: checkdir('task4')
      !make poisson2d
```

Running For the Multi-GPU version you can set the number of GPUs / MPI ranks using the variable `NP`. On *Ascent* within a single node you can use up to 6 GPUs.

```
[ ]: checkdir('task4')
      !NP=2 make run
```

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

```
[ ]: checkdir('task4')
      !NP=1 make run | grep speedup > scale.out
      !NP=2 make run | grep speedup >> scale.out
      !NP=4 make run | grep speedup >> scale.out
      !NP=6 make run | grep speedup >> scale.out
      data_frame4 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

      !rm scale.out

      data_frame4b=data_frame4.iloc[:, [5,7,10,12]].copy()
      data_frame4b.rename(columns={5: 'GPUs', 7: 'time [s]', 10: 'speedup', 12:
      ↪ 'efficiency'})
```

Profiling You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (`pgprof.Task4.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.

```
[ ]: checkdir('task4')
      !NP=2 make profile
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)
3. [OpenSHMEM 1.4 Specification](#)
4. [NVSHMEM 0.3 EA Developer Guide](#)

2.7 Task 5: Make communication asynchronous

NVSHMEM allows you to put communications in *CUDA streams* / *OpenACC async queues*. This allows the CPU already set up communication and kernel launches while the GPU is still communicating, effectively hiding the time spend in API calls.

To do this you need to: * use the `async` and `wait` keywords in the OpenACC pragmas to execute the kernels asynchronously in the OpenACC default queue * replace `nvshmem_double_put` calls with the `nvshmemx_double_put_on_stream` version. use `acc_get_cuda_stream` and `acc_get_default_async` to get the `cudaStream_t` `cudaStream` corresponding to the OpenACC default async queue. * make sure to synchronize before copying the data back to the CPU

Look for **TODOs**.

Compare the scaling and efficiency with the results from the previous task and the MPI versions. Check for asynchronous execution in the profiler.

Optional: Try to understand how well communication and compute overlap is able to improve efficiency when scaling to more GPUs.

Code

- [C Version](#)

Before executing any of the cells below first execute the next cell to change to the right directory.

```
[ ]: %cd $basedirC/task5
```

Compilation If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call `make` with the desired Section `??`. Alternatively you can just navigate to the right directory and execute `make <target>` in your terminal.

```
[ ]: checkdir('task5')
!make poisson2d
```

Running For the Multi-GPU version you can set the number of GPUs / MPI ranks using the variable `NP`. On *Ascent* within a single node you can use up to 6 GPUs.

```
[ ]: checkdir('task5')
!NP=2 make run
```

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

```
[ ]: checkdir('task5')
!NP=1 make run | grep speedup > scale.out
!NP=2 make run | grep speedup >> scale.out
!NP=4 make run | grep speedup >> scale.out
!NP=6 make run | grep speedup >> scale.out
```

```
data_frame5 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

!rm scale.out

data_frame5b=data_frame5.iloc[:,[5,7,10,12]].copy()
data_frame5b.rename(columns={5:'GPUs', 7: 'time [s]', 10:'speedup', 12:
↪'efficiency'})
```

Profiling You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (pgprof.Task5.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.

```
[ ]: checkdir('task5')
!NP=2 make profile
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)
3. [OpenSHMEM 1.4 Specification](#)
4. [NVSHMEM 0.3 EA Developer Guide](#)

2.8 Task 6: Use direct load/store to remote memory

NVSHMEM allows you to put communications in the GPU kernels. However, the `nvshmem_put` / `nvshmem_get` calls are not easily available in OpenACC kernels. However, for *intranode* communication when all GPUs can use P2P (as in the nodes in Ascent and Summit) you can get a pointer to a remote GPUs memory using `nvshmem_ptr`.

To do this you need to: * use the `nvshmem_ptr` to get pointers to your neighboring (top/bottom) `d_A` allocation * when setting `A` to `Anew` also update the halos of your neighbors. You need to use the `deviceptr` keyword to use `d_Atop` / `d_Abottom` device pointers in an OpenACC directly. * add the needed `nvshmem_barrier`.

- Additional task: Similar to the previous version you can use asynchronous execution here.

Look for **TODOs**.

Compare the scaling and efficiency with the results from the previous tasks and the MPI versions. Check for asynchronous execution in the profiler.

Code

- [C Version](#)

Before executing any of the cells below first execute the next cell to change to the right directory.

```
[ ]: %cd $basedirC/task6
```

Compilation If you are using the jupyter notebook approach you can execute the cells below. They will put you in the right directory. There you can call `make` with the desired Section `??`. Alternatively you can just navigate to the right directory and execute `make <target>` in your terminal.

```
[ ]: checkdir('task6')
      !make poisson2d
```

Running For the Multi-GPU version you can set the number of GPUs / MPI ranks using the variable `NP`. On *Ascent* within a single node you can use up to 6 GPUs.

```
[ ]: checkdir('task6')
      !NP=2 make run
```

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

```
[ ]: checkdir('task6')
      !NP=1 make run | grep speedup > scale.out
      !NP=2 make run | grep speedup >> scale.out
      !NP=4 make run | grep speedup >> scale.out
      !NP=6 make run | grep speedup >> scale.out
      data_frame5 = pandas.read_csv('scale.out', delim_whitespace=True, header=None)

      !rm scale.out

      data_frame5b=data_frame5.iloc[:, [5,7,10,12]].copy()
      data_frame5b.rename(columns={5: 'GPUs', 7: 'time [s]', 10: 'speedup', 12:
      ↪ 'efficiency'})
```

Profiling You can profile the code by executing the next cell. **After** the profiling completed download the tarball containing the profiles (`pgprof.Task6.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.

```
[ ]: checkdir('task6')
      !NP=2 make profile
```

References

1. <http://www.openacc.org>
2. [OpenACC Reference Card](#)
3. [OpenSHMEM 1.4 Specification](#)
4. [NVSHMEM 0.3 EA Developer Guide](#)

3 Survey

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

