Overview of GPU Programming Using OpenACC

Nick Featherstone, feathern.colorado.edu
University of Colorado Boulder

Slides: https://github.com/ResearchComputing/RMACC
Mar 17, 2017

Overview

- What is OpenACC?
- Compiling with OpenACC
- How do we use OpenACC directives?
- Data Movement
- Parallelization
- Asynchronous Operations

NOTE: With 90 minutes for slides + exercises, we are only scratching the surface!

Why Program for a GPU?

- Because its there … on your laptop… not doing much.
- Increasingly common component of many HPC systems.
  - e.g., Titan/Summit ORNL

Why Not?

- I don’t want to learn CUDA
- I don’t have TIME to learn CUDA
- Will my code be portable?

Enter OpenACC

- Stands for “Open Accelerators”
- Developed by Cray, *Nvidia*, CAPS, *PGI*
- Informal definition:

  An OpenMP-like way of programming for graphics cards without needing to learn CUDA
- Directive-based
- Easy to modify existing code
- Similar in flavor to OpenMP

Compilation

pgfortran -acc -fast -ta=tesla -Minfo=accel -o a.out a.f90

- -acc : parse OpenACC directives
- -fast : common PGI optimizations
- -ta=tesla : target tesla architecture
- -Minfo=accel : reports on parallelized regions of code

How do we use it?

- Place directives into your code
- Directives provide instructions for parallelization & data movement
- Directives begin with !$acc (Fortran) or #pragma (C)
- Accompanied by { } brackets in C

Fortran

```
!$acc  directive-name

   Original Code
```

C / C++

```
#pragma acc  directive-name
   {
   Original Code
    }
```

Parallelization: Essentials

- Identify region that we wish to parallelize
- Use **kernels** or **parallel** directive
- **kernels:** compiler decides what to parallelize
  - "kernel": a function that runs in parallel on the GPU
  - tells the compiler to auto-generate a kernel(s) for this region of code
- **parallel:** user decides what to parallelize
  - requires additional directives
  - Runs redundant copies of code region on GPU if no additional directives provided
  - often used in conjunction with **loop** directive
- **loop:** Instructs compiler to parallelize loop within parallel region. Launches a loop kernel.

Fortran

```
!$acc  kernels
      Original Code
!$acc end kernels
```

or

```
!$acc  parallel
!$acc loop
      Loop Code
!$acc end parallel
```

or

```
!$acc  parallel loop
      Loop Code
!$acc end parallel loop
```

C/C++

```
#pragma acc  kernels
    {
     Original Code
     }
```

or

```
#pragma acc  parallel
    {
     #pragma acc loop
    Original Code
     }
```

or

```
#pragma acc  parallel loop
    {
     Original Code
     }
```

Data Movement: Essentials

- Data can be moved onto and off of the GPU via **data** directive
- Data regions identify regions of code wherein data exists on the GPU and is shared by all active kernels

```
!$acc  data copy(arr1,arr2)
    Code
!$acc end data
```

```
#pragma acc  data copy(arr1,arr2)
    {
        Code }
```

- **copy** clause controls data movment:
  - Entering data region: arr1 and arr2 allocated on GPU; values copied from host to GPU
  - Exiting data region: values copies from GPU to host

This is enough to get started…

Fortran

```
!$acc data copy(arr1,arr2,arr3)
!$acc  parallel
!$acc loop
      Do k = 1, nz
      Do j = 1, ny
      Do i = 1, nx
            arr3(i,j,k) = arr1(i,j,k)+arr2(i,j,k)
      Enddo
      Enddo
      Enddo
!$acc end parallel
```

Interactive Labs on Summit

- As part of this tutorial, you have access to Summit's GPU nodes.
- Summit GPU nodes:
  - 7 nodes in this reservation
  - 24 Haswell cores
  - 128 GB RAM
  - Nvidia Tesla K80 x 2
- Feel free to use your own laptop + PGI compiler as well

Logging Onto Summit

- We will run our jobs on Summit.
- All jobs will be submitted to the Summit batch scheduler (SLURM)
- Jobs should be submitted from an scompile node
- SCOMPILE reached via *two* steps:
  1. ssh username@tutorial-login.Colorado.edu
  2. ssh scompile
- *Immediately* type “module load pgi” (do this any time you open a new scompile window)

Exercise Logistics

- Each exercise is contained within unique directory:
  - E.g. OpenACC/Lab/ex1
- Exercise directories contain:
  - Makefile
  - jobscript
  - Source code (e.g., ex1.f90)
  - A solution directory with similarly named files.
- Exercises provided in free-form Fortran (forgiving syntax)
- To build the code, type “Make”
- To submit the job, type “sbatch jobscript”
- To check job status, type “squeue –u $USER”

Example/Exercise

3-D Diffusion Problem

$$f_{ijk,t+1} = 1/6 \left(f_{i-1,\,jkt} + f_{i+1,jkt} + f_{i,\,j-1,kt} + f_{i,j+1,kt} + f_{ij,k-1,t} + f_{ij,k+1t} \right)$$

- Consider the code in OpenACC/Lab/ex1/ex1.f90
- Running “make” will create serial code (ex1.serial) and gpu-accelerated code (ex1.gpu)
- Compile this code. Let’s examine the output.

Running this Code

Let's examine ex1/jobscript…

Size of 3-D domain controlled at run-time:

./ex5.{gpu,serial} –nx X –ny Y –nz Z –nt T

- -nx X : set number of x grid points to X
- -ny Y : set number of "y" grid points to Y
- -nz Z : set number of "z" grid points to Z
- -nt T : set number of time steps to T
- X,Y,Z default to 256
- T defaults to 100

Exercise

- Modify the ex1.f90 code to use the **parallel** and **loop** directives.
- Note: **loop** parallelizes only the outer loop by default.
- **present** clause gives compiler hint that data already on GPU
- Can parallelize inner loops as well via **collapse** clause
  - Parallelize outer 2 loops: **loop collapse(2)**
  - Parallelize outer 3 loops: **loop collapse(3)**
  - Try a few …

```
!$acc  parallel
!$acc loop collapse(n) present(arr1,arr2)
    Loop Code
!$acc end parallel
```

Exercise

- Consider the code in OpenACC/Lab/ex2/ex2.f90
- Earlier, data remained on GPU throughout calculation
- If our code is using MPI, *something* has to come off the GPU *during* the calculation…
- Revise placement of the data region so that CPU gets updated copy of var prior to calling ghost_zone_comm
- Ghost_zone_comm: stub routine where parallel communication would take place.
- Assume we need to communicate the top/bottom, front/back, and left/right edges of domain at edge time step.

PGProf

- That was a bit slow
- We seem to be doing a lot of data copying. Is this causing the slowdown?
- Let's analyze the code
- If using your laptop, run **pgprof** from ex2 directory
- Otherwise, follow along with me.

Selective Data Movement

- We don't have to copy everything all the time…
- Within a data region, we can use the **update** directive
  - use **host** clause to copy out to CPU
  - use **device** clause to copy onto GPU

```
!acc update device(arr1,arr2)
!$acc  parallel loop
collapse(3)
      …
!$acc end parallel loop
!acc update host(arr3)
```

Selective Data Movement

Fortran

```
!$acc data copy(arr1,arr2,arr3)
!acc update device(arr1,arr2)
!$acc  parallel loop collapse(3)
      Do k = 1, nz
      Do j = 1, ny
      Do i = 1, nx
            arr3(i,j,k) = arr1(i,j,k)+arr2(i,j,k)
      Enddo
      Enddo
      Enddo
!$acc end parallel loop
!acc update host(arr3)
```

Selective Data Movement

- We don't have to copy an entire array
- Can copy slices as well, e.g.

```
!acc update device( arr1( : , : , 10), arr2(1, : , :) )
!$acc  parallel loop collapse(3)
      …
!$acc end parallel loop
!acc update host( arr3(:, :, 1) )
```

Exercise

- Consider the code in OpenACC/Lab/ex3/ex3.f90
- Return the data region to its original location outside the time-stepping loop
- Use update host and update device to move only data that needs to be transferred…
- What needs to be transferred?

Asynchronous Operations

- Maybe the CPU could be doing some work too…
- Maybe we would like to overlap data transfer and computation
- Use the **async** clause and **wait** directive

```
!acc update host(arr1) async(queue #)
!$acc  parallel loop collapse(3) async(queue #)
      gpu code
!$acc end parallel loop
cpu code    ! can execute while loop is running on GPU
!acc wait   ! blocks until instructions in all queues complete
```

Asynchronous Operations

- Instructions within same queue are executed sequentially
- Data transfer in queue 2 overlaps with calculation in queue 1

```
!acc update host(arr1) async(1)
!$acc  parallel loop collapse(3) async(1)
      gpu code 1
!$acc end parallel loop

!acc update host(arr2) async(2)
!$acc  parallel loop collapse(3) async(2)
      gpu code 2
!$acc end parallel loop
!acc wait    ! Blocks until instructions in all queues complete
```

Exercise 4

- Consider the code in OpenACC/Lab/ex4/ex4.f90
- Try to overlap data transfer and computation
- Add **async** clauses to the loop and update directives.
- Use the queue defined within each k-iteration
- Place a **wait** directives at appropriate locations
- Can you get a speedup?