OpenACC Course
Lecture 1: Introduction to OpenACC
September 2015

NVIDIA
Course Objective:

Enable you to accelerate your applications with OpenACC.
Course Syllabus

Oct 1: Introduction to OpenACC
Oct 6: Office Hours
Oct 15: Profiling and Parallelizing with the OpenACC Toolkit
Oct 20: Office Hours
Oct 29: Expressing Data Locality and Optimizations with OpenACC
Nov 3: Office Hours
Nov 12: Advanced OpenACC Techniques
Nov 24: Office Hours

Recordings: https://developer.nvidia.com/openacc-course
Introduction to OpenACC

Jeff Larkin, NVIDIA Developer Technologies
Agenda

Why OpenACC?
Accelerated Computing Fundamentals
OpenACC Programming Cycle
Installing the OpenACC Toolkit
Accessing QwikLabs
Week 1 Homework
Why OpenACC?
OpenACC
Simple | Powerful | Portable

Fueling the Next Wave of Scientific Discoveries in HPC

main()
{
    <serial code>
    #pragma acc kernels
    //automatically runs on GPU
    {
        <parallel code>
    }
}

RIKEN Japan
NICAM- Climate Modeling

7-8x Speed-Up
5% of Code Modified

University of Illinois
PowerGrid- MRI Reconstruction

70x Speed-Up
2 Days of Effort

8000+
Developers
using OpenACC

http://www.openacc.org/content/experiences-porting-molecular-dynamics-code-gpus-cr-ay-x-k7
LS-DALTON

Large-scale application for calculating high-accuracy molecular energies

“OpenACC makes GPU computing approachable for domain scientists. Initial OpenACC implementation required only minor effort, and more importantly, no modifications of our existing CPU implementation.”

Janus Juul Eriksen, PhD Fellow
qLEAP Center for Theoretical Chemistry, Aarhus University

<table>
<thead>
<tr>
<th>Lines of Code Modified</th>
<th># of Weeks Required</th>
<th># of Codes to Maintain</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;100 Lines</td>
<td>1 Week</td>
<td>1 Source</td>
</tr>
</tbody>
</table>

Big Performance

LS-DALTON CCSD(T) Module
Benchmarked on Titan Supercomputer (AMD CPU vs Tesla K20X)
OpenACC Directives

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

Pragma acc data copyin(a,b) copyout(c)
{
   ...
   #pragma acc parallel
   {
      #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
         z[i] = x[i] + y[i];
         ...
      }
   }
   ...
}

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, MIC
Accelerated Computing Fundamentals
Accelerated Computing
10x Performance & 5x Energy Efficiency for HPC

CPU
Optimized for Serial Tasks

GPU Accelerator
Optimized for Parallel Tasks
What is Heterogeneous Programming?

- A few % of Code
- A large % of Time

Compute-Intensive Functions

Rest of Sequential CPU Code
Portability & Performance

Accelerated Libraries
- High performance with little or no code change
- Limited by what libraries are available

Compiler Directives
- High Level: Based on existing languages; simple, familiar, portable
- High Level: Performance may not be optimal

Parallel Language Extensions
- Greater flexibility and control for maximum performance
- Often less portable and more time consuming to implement
Code for Portability & Performance

Libraries
- Implement as much as possible using portable libraries

Directives
- Use directives for rapid and portable development

Languages
- Use lower level languages for important kernels
OpenACC Programming Cycle
Identify Available Parallelism

Express Parallelism

Express Data Movement

Optimize Loop Performance
**Example: Jacobi Iteration**

Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.

Common, useful algorithm

Example: Solve Laplace equation in 2D: $\nabla^2 f(x, y) = 0$

$$A_{k+1}(i, j) = \frac{A_k(i - 1, j) + A_k(i + 1, j) + A_k(i, j - 1) + A_k(i, j + 1)}{4}$$
while ( err > tol && iter < iter_max ) {
    err=0.0;

    for( int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {

            Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] + 
                                A[j-1][i] + A[j+1][i]);

            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }

    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    iter++;
}
Identify Parallelism

while ( err > tol && iter < iter_max ) {
    err=0.0;

    for( int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {
            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }

    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
        }
    }

    iter++;
}
The kernels directive identifies a region that may contain *loops* that the compiler can turn into parallel *kernels*.

```c
#pragma acc kernels
{
    for(int i=0; i<N; i++)
    {
        x[i] = 1.0;
        y[i] = 2.0;
    }

    for(int i=0; i<N; i++)
    {
        y[i] = a*x[i] + y[i];
    }
}
```

The compiler identifies 2 parallel loops and generates 2 kernels.
Parallelize with OpenACC kernels

```
while ( err > tol && iter < iter_max ) {
    err=0.0;

    #pragma acc kernels
    {
        for( int j = 1; j < n-1; j++ ) {
            for(int i = 1; i < m-1; i++) {


                err = max(err, abs(Anew[j][i] - A[j][i]));
            }
        }

        for( int j = 1; j < n-1; j++ ) {
            for( int i = 1; i < m-1; i++ ) {
                A[j][i] = Anew[j][i];
            }
        }
    }

    iter++;
}
```
Building the code

$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c

main:
  40, Loop not fused: function call before adjacent loop
  Generated vector sse code for the loop
  51, Loop not vectorized/parallelized: potential early exits
  55, Generating copyout(Anew[1:4094][1:4094])
  Generating copyin(A[:][:])
  Generating copyout(A[1:4094][1:4094])
  Generating Tesla code
  57, Loop is parallelizable
  59, Loop is parallelizable
  Accelerator kernel generated
  57, #pragma acc loop gang /* blockIdx.y */
  59, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
  63, Max reduction generated for error
  67, Loop is parallelizable
  69, Loop is parallelizable
  Accelerator kernel generated
  67, #pragma acc loop gang /* blockIdx.y */
  69, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
Why did OpenACC slow down here?

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40
Very low Compute/Memcpy ratio

Compute: 5 seconds
Memory Copy: 62 seconds
PCle Copies

104ms/iteration
Excessive Data Transfers

while ( err > tol && iter < iter_max )
{
    err=0.0;

    #pragma acc kernels

    for(int j = 1; j < n-1; j++)
    {
        for(int i = 1; i < m-1; i++)
        {
            err = max(err, abs(Anew[j][i] - A[j][i]));
        }
    }

    ...
Identifying Data Locality

```c
while ( err > tol && iter < iter_max ) {
    err=0.0;

    #pragma acc kernels
    {
        for( int j = 1; j < n-1; j++) {
            for(int i = 1; i < m-1; i++) {
                err = max(err, abs(Anew[j][i] - A[j][i]));
            }
        }
    }

    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++) {
            A[j][i] = Anew[j][i];
        }
    }

    iter++;
}
```

Does the CPU need the data between these loop nests?

Does the CPU need the data between iterations of the convergence loop?
Identify Available Parallelism

Optimize Loop Performance

Express Data Movement

Express Parallelism
Data regions

The `data` directive defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```c
#pragma acc data
{
#pragma acc kernels ...
#pragma acc kernels ...
}
```

Arrays used within the data region will remain on the GPU until the end of the data region.
Data Clauses

`copy ( list )`  Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.

`copyin ( list )`  Allocates memory on GPU and copies data from host to GPU when entering region.

`copyout ( list )`  Allocates memory on GPU and copies data to the host when exiting region.

`create ( list )`  Allocates memory on GPU but does not copy.

`present ( list )`  Data is already present on GPU from another containing data region.

`deviceptr( list )`  The variable is a device pointer (e.g. CUDA) and can be used directly on the device.
Array Shaping

Compiler sometimes cannot determine size of arrays

Must specify explicitly using data clauses and array “shape”

C/C++

#pragma acc data copyin(a[0:nelem]) copyout(b[s/4:3*s/4])

Fortran

!$acc data copyin(a(1:end)) copyout(b(s/4:3*s/4))

Note: data clauses can be used on data, parallel, or kernels
Express Data Locality

```c
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
    err=0.0;
    #pragma acc kernels
    {
        for( int j = 1; j < n-1; j++) {
            for(int i = 1; i < m-1; i++) {
                Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                                   A[j-1][i] + A[j+1][i]);
                err = max(err, abs(Anew[j][i] - A[j][i]));
            }
        }
        for( int j = 1; j < n-1; j++) {
            for( int i = 1; i < m-1; i++ ) {
                A[j][i] = Anew[j][i];
            }
        }
        iter++;
    }
}
```

Copy A to/from the accelerator only when needed.
Create Anew as a device temporary.
Rebuilding the code

```plaintext
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
  40, Loop not fused: function call before adjacent loop
  Generated vector sse code for the loop
  51, Generating copy(A[:,][:])
  Generating create(Anew[:,][:])
  Loop not vectorized/parallelized: potential early exits
  56, Accelerator kernel generated
  56, Max reduction generated for error
  57, #pragma acc loop gang /* blockIdx.x */
  59, #pragma acc loop vector(256) /* threadIdx.x */
  56, Generating Tesla code
  59, Loop is parallelizable
  67, Accelerator kernel generated
  68, #pragma acc loop gang /* blockIdx.x */
  70, #pragma acc loop vector(256) /* threadIdx.x */
  67, Generating Tesla code
  70, Loop is parallelizable
```
Visual Profiler: Data Region

Iteration 1
Iteration 2
Was 104ms
Speed-Up (Higher is Better)

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40

- Single Thread: 1.90X
- 2 Threads: 3.20X
- 4 Threads: 3.74X
- 6 Threads: 3.83X
- 8 Threads: 3.74X
- OpenACC: 19.89X

Socket/Socket: 5.2X
Identify Available Parallelism

Express Parallelism

Express Data Movement

Optimize Loop Performance
The loop Directive

The `loop` directive gives the compiler additional information about the next loop in the source code through several clauses.

- **independent** - all iterations of the loop are independent
- **collapse(N)** - turn the next N loops into one, flattened loop
- **tile(N[,M,...])** - break the next 1 or more loops into tiles based on the provided dimensions.

These clauses and more will be discussed in greater detail in a later class.
Optimize Loop Performance

```c
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
    err=0.0;
    #pragma acc kernels
    {
        #pragma acc loop device_type(nvidia) tile(32,4)
        for( int j = 1; j < n-1; j++ ) {
            for( int i = 1; i < m-1; i++ ) {
                err = max(err, abs(Anew[j][i] - A[j][i]));
            }
        }
        #pragma acc loop device_type(nvidia) tile(32,4)
        for( int j = 1; j < n-1; j++ ) {
            for( int i = 1; i < m-1; i++ ) {
                A[j][i] = Anew[j][i];
            }
        }
    }
    iter++;
}
```

"Tile" the next two loops into 32x4 blocks, but only on NVIDIA GPUs.
Speed-Up (Higher is Better)

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Speed-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Thread</td>
<td>1.90X</td>
</tr>
<tr>
<td>2 Threads</td>
<td>3.20X</td>
</tr>
<tr>
<td>4 Threads</td>
<td>3.74X</td>
</tr>
<tr>
<td>6 Threads</td>
<td>3.83X</td>
</tr>
<tr>
<td>8 Threads</td>
<td>19.89X</td>
</tr>
<tr>
<td>OpenACC</td>
<td>21.22X</td>
</tr>
<tr>
<td>OpenACC Tuned</td>
<td>21.22X</td>
</tr>
</tbody>
</table>
The OpenACC Toolkit
Introducing the New OpenACC Toolkit
Free Toolkit Offers Simple & Powerful Path to Accelerated Computing

PGI Compiler
Free OpenACC compiler for academia

NVProf Profiler
Easily find where to add compiler directives

GPU Wizard
Identify which GPU libraries can jumpstart code

Code Samples
Learn from examples of real-world algorithms

Documentation
Quick start guide, Best practices, Forums

http://developer.nvidia.com/openacc
Download the OpenACC Toolkit

Go to https://developer.nvidia.com/openacc
Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc
- Register for the toolkit
  - If you are an academic developer, be sure to click the check box at the bottom.
Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc
- Register for the toolkit
  - If you are an academic developer, be sure to click the check box at the bottom.
- You will receive an email from NVIDIA
  - Be sure to read the Quick Start Guide
Windows/Mac Developers

- The OpenACC Toolkit is only available on Linux, however...
- The PGI compiler is available on Mac and Windows from [http://www.pgroup.com/support/trial.htm](http://www.pgroup.com/support/trial.htm)
  - You should still register for the OpenACC Toolkit to get the 90 day license.
- The CUDA Toolkit contains the libraries and profiling tools that will be used in this course.
  - Obtaining all examples and guides from the toolkit will still require downloading the full OpenACC toolkit.
Using QwikLabs
Getting access

Go to nvidia.qwiklab.com, log-in or create an account

Sign In or Create a New Account
Homework
Complete “2X in 4 Steps” Qwiklab


This lab is browser-based and should take you roughly 1 hour.
Install the OpenACC Toolkit (Optional)

- Go to [developer.nvidia.com/openacc](https://developer.nvidia.com/openacc)
- Register for the OpenACC Toolkit
- Install on your personal machine. (Linux Only)
Where to find help

• OpenACC Course Recordings - https://developer.nvidia.com/openacc-course
• OpenACC on StackOverflow - http://stackoverflow.com/questions/tagged/openacc
• OpenACC Toolkit - http://developer.nvidia.com/openacc

Additional Resources:

• Parallel Forall Blog - http://devblogs.nvidia.com/parallelforall/
• GPU Technology Conference - http://www.gputechconf.com/
• OpenACC Website - http://openacc.org/
<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 1</td>
<td>Introduction to OpenACC</td>
</tr>
<tr>
<td>Oct 6</td>
<td>Office Hours</td>
</tr>
<tr>
<td>Oct 15</td>
<td>Profiling and Parallelizing with the OpenACC Toolkit</td>
</tr>
<tr>
<td>Oct 20</td>
<td>Office Hours</td>
</tr>
<tr>
<td>Oct 29</td>
<td>Expressing Data Locality and Optimizations with OpenACC</td>
</tr>
<tr>
<td>Nov 3</td>
<td>Office Hours</td>
</tr>
<tr>
<td>Nov 12</td>
<td>Advanced OpenACC Techniques</td>
</tr>
<tr>
<td>Nov 24</td>
<td>Office Hours</td>
</tr>
</tbody>
</table>

Recordings:
https://developer.nvidia.com/openacc-course