An Introduction to OpenACC

William Killian

Department of Computer and Information Science
University of Delaware

CISC 879 — Advanced Parallel Programming
Methods of Accelerating Applications

- Targeting Multi-core CPUs
- Targeting Multi-node CPUs
- Targeting GPUs
Methods of Accelerating Applications

- Targeting Multi-core CPUs
  - OpenMP
  - PThreads, QThreads, etc ...

- Targeting Multi-node CPUs

- Targeting GPUs
Methods of Accelerating Applications

- Targeting Multi-core CPUs
  - OpenMP
  - PThreads, QThreads, etc …

- Targeting Multi-node CPUs
  - MPI

- Targeting GPUs
Methods of Accelerating Applications

- Targeting Multi-core CPUs
  - OpenMP
  - PThreads, QThreads, etc ...

- Targeting Multi-node CPUs
  - MPI

- Targeting GPUs
  - CUDA
Methods of Accelerating Applications

- Targeting Multi-core CPUs
  - OpenMP
  - PThreads, QThreads, etc ...
  - OpenCL
- Targeting Multi-node CPUs
  - MPI
- Targeting GPUs
  - CUDA
  - OpenCL
Methods of Accelerating Applications

- Targeting Multi-core CPUs
  - OpenMP
  - PThreads, QThreads, etc …
  - OpenCL
- Targeting Multi-node CPUs
  - MPI
- Targeting GPUs
  - CUDA
  - OpenCL

What’s left? …OpenACC!
Methods of Accelerating Applications

- Targeting Multi-core CPUs
  - OpenMP
  - PThreads, QThreads, etc …
  - OpenCL
- Targeting Multi-node CPUs
  - MPI
- Targeting GPUs
  - CUDA
  - OpenCL
  - OpenACC

What’s left? …**OpenACC**!
What is OpenACC?

- is an API using compiler directives that
- allows for small segments of code, called kernels, to be run on the GPU and
- requires little to no modifications to the original program
- is compatible with C/C++ and Fortran

Reason Behind Formation

OpenACC was formed to help create and foster a cross platform API that would allow any scientist or programmer to easily accelerate their application on modern many-core and multi-core processors using directives.
History of OpenACC

- Initially collaboration between CAPS Entreprise, Cray Inc., The Portland Group (PGI), and NVIDIA
- Built from OpenMP-style directives
  - #pragma omp parallel vs. #pragma acc parallel
  - Creators of OpenACC are all members of the OpenMP Working Group on accelerators
- Standardized in November 2011 at SuperComputing 2011
- Compilers available from Cray, CAPS, and PGI
- Potential API merge with OpenMP in the future?
Directives Overview

Format

```
#pragma acc directive-name [clause [[,] clause] …]
```

Possible directives are:

- **parallel**: starts parallel execution on the accelerator
- **kernels**: defines a region that should be converted to a kernel
- **data**: defines contiguous data to be allocated on the accelerator
- **host_data**: makes the address of accelerator data available on the host
- **loop**: defines type of parallelism to apply to proceeding loop
Directives Overview (continued)

**cache**
defines elements or subarrays that should be fetched into cache

**declare**
defines that a variable should be allocated in accelerator memory

**update**
update all or part of host memory from device memory, or vice versa

**wait**
forces program to wait for completion of asynchronous activity
Clauses Overview

Each directive can have zero (or more) clauses associated.

**Example clauses are:**

if (e)  
condition used to determine if command should be executed (data transfer, accelerator computation, etc)

async [(n)]  
tells the current command to be executed asynchronously. Used with wait for synchronization.

**Clauses found in either kernels or parallel directives:**

reduction (op:list)

private (list)

firstprivate (list)
Clauses Overview

Each directive can have zero (or more) clauses associated. **Example clauses are:**

- `if (e)`: condition used to determine if command should be executed (data transfer, accelerator computation, etc).
- `async [(n)]`: tells the current command to be executed asynchronously. Used with `wait` for synchronization.

**Clauses found in either kernels or parallel directives:**

- `reduction (op:list)`: similar to OpenMP.
- `private (list)`
- `firstprivate (list)`
Clauses — parallel and loop

**Clauses — parallel directive**

- `num_gangs(e)` specify the number of gangs to execute in the region
- `num_workers(e)` specify number of workers to launch in each gang
- `vector_length(e)` define vector length to use

**Clauses — loop directive**

- `collapse(n)` specifies # of loops associated
- `gang(e)` distribute across gang
- `worker(e)` distribute across worker (within gang)
- `vector(e)` operate in SIMD (within gang or worker)
- `seq` execute sequentially on the accelerator
- `independent` tell the compiler loops are data-independent
### Clauses — Data Operations (optional with most directives)

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy(list)</code></td>
<td>transfer to/from device</td>
</tr>
<tr>
<td><code>copyin(list)</code></td>
<td>transfer to device</td>
</tr>
<tr>
<td><code>copyout(list)</code></td>
<td>transfer from device</td>
</tr>
<tr>
<td><code>create(list)</code></td>
<td>allocate on device</td>
</tr>
<tr>
<td><code>present(list)</code></td>
<td>data which is already on the device</td>
</tr>
<tr>
<td><code>deviceptr(list)</code></td>
<td>used to inform which variables are device pointers (opposed to host)</td>
</tr>
<tr>
<td><code>device_resident(list)</code></td>
<td>allocate on device instead of host</td>
</tr>
</tbody>
</table>
Clauses — Data Operations (optional with most directives)

- `pcopy(list)`
  - transfer to/from device
- `pcopyin(list)`
  - transfer to device
- `pcopyout(list)`
  - transfer from device
- `pcreate(list)`
  - allocate on device
- `present(list)`
  - data which is already on the device
- `deviceptr(list)`
  - used to inform which variables are device pointers (opposed to host)
- `device_resident(list)`
  - allocate on device instead of host

Checks for presence before issuing data command
Clauses — host_data and update

Clauses — use with host_data directive

use_device (list) make the device address data available in host code

Clauses — use with update directive

host (list) variables to copy from device to host
device (list) variables to copy from host to device
We mentioned data clauses such as copy and create but never went over what we do when the memory was dynamically allocated (using malloc).

**What should we do?**

We can specify the size of the data!

A is our array of size $n$ but we need to provide a hint to OpenACC

**Solution:**

```plaintext
#pragma acc kernels copyin(A[0:n])
```

**Note**

This will also work for 2-dimensional arrays i.e. $A[0:m*n]$
Combining Clauses

Observation

Similar to OpenMP, we can combine directives

- `#pragma acc parallel loop [clause [[,] clause]...]`
- `#pragma acc kernels loop [clause [[,] clause]...]`

Notice

A loop must directly follow, similar to `parallel for` in OpenMP
Runtime calls allow the programmer to obtain information about the host and accelerators during runtime, instead of compile time.

**List of library routines:**

```c
int acc_get_num_devices (acc_device_t); // gets number of devices of passed type
int acc_set_device_type (acc_device_t); // sets device type to use
acc_device_t acc_get_device_type (); // gets current device type
void acc_set_device_num (int, acc_device_t); // sets device based on index and type
void acc_get_device_num (acc_device_t); // gets current device number
```
Runtime Routines — Synchronization

```c
int acc_async_test (int);
tests to see if a specified async. tasks are completed

int acc_async_test_all ();
tests to see if all async. tasks are completed

void acc_async_wait (int);
waits until specified async. task is completed

void acc_async_wait_all ();
waits until all async. tasks are completed
```
Runtime Routines — Setup and Teardown

```c
void acc_init (acc_device_t);
initialize OpenACC runtime for passed device type
void acc_shutdown (acc_device_t);
shut down connection to passed device type
int acc_on_device (acc_device_t);
tells the program whether it’s executing on passed device type
void* acc_malloc (size_t);
allocates memory on the device
void acc_free (void*);
frees memory on the device
```
Compiling OpenACC

There are a few different compilers available for OpenACC.

We will be using HMPP Workbench 3.2.1 by CAPS Entreprise.

In addition to OpenACC, HMPP Workbench also supports another directive-based accelerator language, HMPP (and consequently OpenHMPP).
Compiling using Built-in (supplied) Makefiles

- Obtain HMPP Workbench 3.2.1 (see website for details)
- Extract the tarball
- Extract the OpenACC Labs tarball
- Navigate to the OpenACC_Labs/CUDA/C/ directory
- Copy an existing lab
- Edit the code
- Invoke “make”
Compiling using the command line

Sample Invocation

hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
Compiling using the command line

Sample Invocation

```
hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
```

- hmpp compiler
Compiling using the command line

Sample Invocation

```
hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
```

- **** hmpp compiler
- **** specify OpenACC codelet target (CUDA or OPENCL)
Compiling using the command line

**Sample Invocation**

```
hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
```

- `hmpp` compiler
- specify OpenACC codelet target (CUDA or OPENCL)
- force proper codelet creation for compilation
Compiling using the command line

Sample Invocation

```
hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
```

- hmpp compiler
- specify OpenACC codelet target (CUDA or OPENCL)
- force proper codelet creation for compilation
- compiler to use for host code
Compiling using the command line

Sample Invocation

```
hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
```

- **hmpp** compiler
- specify OpenACC codelet target (CUDA or OPENCL)
- force proper codelet creation for compilation
- compiler to use for host code
- flags for host compiler
Compiling using the command line

Sample Invocation

```
hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c
```

- `hmpp` compiler
- Specify OpenACC codelet target (CUDA or OPENCL)
- Force proper codelet creation for compilation
- Compiler to use for host code
- Flags for host compiler
- Specify output file
Compiling using the command line

Sample Invocation

hmpp --openacc-target=CUDA --codelet-required
gcc -O2 -o mvmult mvmult.c

- hmpp compiler
- specify OpenACC codelet target (CUDA or OPENCL)
- force proper codelet creation for compilation
- compiler to use for host code
- flags for host compiler
- specify output file
- source file(s)
Problem Overview

**Problem**

Given two vectors, $A$ and $B$, each of size $n$, we wish to compute the per-component addition and store the result into $C$.

**Pseudocode**

```c
int i;
for (i = 0; i < N; ++i) {
    C[i] = A[i] + B[i];
}
```
C Implementation

```c
1  const int N = 1000;
2  float A [N];
3  float B [N];
4  float C [N];
5  int i;
6
7  // Initialization Loop
8  for (i = 0; i < N; ++i) {
9      A [i] = i;
10     B [i] = 2* i - 1;
11  }
12
13  // Computation Loop
14
15  for (i = 0; i < N; ++i) {
16      C [i] = A [i] + B [i];
17  }
```
OpenACC Implementation

```c
1  const int N = 1000;
2  float A [N];
3  float B [N];
4  float C [N];
5  int i;

7  // Initialization Loop
8  for (i = 0; i < N; ++i) {
9    A [i] = i;
10   B [i] = 2 * i - 1;
11  }

13  // Computation Loop
14  #pragma acc kernels loop independent copyin(A,B), copyout(C)
15  for (i = 0; i < N; ++i) {
16    C [i] = A [i] + B [i];
17  }
```
Execution time of Vector-Vector Addition

**Graph:**
- **Y-axis:** Execution Time (us)
- **X-axis:** Problem Size
- **Legend:**
  - GPU
  - GPU+Transfer
  - CPU

**Data Points:**
- 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304
Problem Overview

Problem
Given two matrices, $A$ and $B$, with $A$ having dimensions $m \times p$ and $B$ having dimensions $p \times n$, we wish to compute the row-column inner product into $C$, a matrix with dimensions $m \times n$.

Pseudocode
```c
int i, j, k;
for (i = 0; i < M; ++i)
    for (j = 0; j < N; ++j) {
        C [i][j] = 0;
        for (k = 0; k < P; ++k)
            C [i][j] += A [i][k] * B [k][j];
    }
```
#define INDEX(M,N,i,j) (i + j * M)

int main() {
  float* A; float* B; float* C;
  int i, j, k;

  A = (float*) malloc (M * P * sizeof (float));
  B = (float*) malloc (P * N * sizeof (float));
  C = (float*) malloc (M * N * sizeof (float));

  for (i = 0; i < P; ++i) {
    for (j = 0; j < M; ++j)
      A [INDEX(M,P,i,j)] = (float) rand () / RAND_MAX;
    for (k = 0; k < N; ++k)
      B [INDEX(P,N,k,i)] = (float) rand () / RAND_MAX;
  }
Matrix-Matrix Multiplication

C Implementation (Computation)

```
for (i = 0; i < M; ++i) {
    for (j = 0; j < N; ++j) {
        float sum = 0.0f;
        for (k = 0; k < P; ++k) {
            sum += A [INDEX(M,P,i,k)] * B [INDEX(P,N,k,j)];
        }
        C [INDEX(M,N,i,j)] = sum;
    }
}
```
int m, n, p;

m = M; n = N; p = P;

// computation

#pragma acc kernels copyin(A[0:m*p],B[0:p*n]), copyout(C[0:m*n])
{
    #pragma acc loop independent
    for (i = 0; i < M; ++i) {
        #pragma acc loop independent
        for (j = 0; j < N; ++j) {
            float sum = 0.0f;
            for (k = 0; k < P; ++k) {
                sum += A [INDEX(M,P,i,k)] * B [INDEX(P,N,k,j)];
            }
            C [INDEX(M,N,i,j)] = sum;
        }
    }
}

free (A);
Matrix-Matrix Multiplication

Execution time of Matrix-Matrix Multiplication

- C1060
- K20
- CPU

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Execution Time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10^0</td>
</tr>
<tr>
<td>16</td>
<td>10^1</td>
</tr>
<tr>
<td>32</td>
<td>10^2</td>
</tr>
<tr>
<td>64</td>
<td>10^3</td>
</tr>
<tr>
<td>128</td>
<td>10^4</td>
</tr>
<tr>
<td>256</td>
<td>10^5</td>
</tr>
<tr>
<td>512</td>
<td>10^6</td>
</tr>
<tr>
<td>1024</td>
<td>10^7</td>
</tr>
<tr>
<td>2048</td>
<td>10^8</td>
</tr>
<tr>
<td>4096</td>
<td>10^9</td>
</tr>
</tbody>
</table>
Summary

- OpenACC makes targeting *accelerators* much easier
- Designed for use by *scientists* to make GPGPU much easier
- Syntax similar to *OpenMP*
- Compiling with *HMPP Workbench 3.2.1* can target CUDA or OpenCL

OpenACC Reference API

http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf