Directive-based Accelerator Programming With OpenACC

Mathew Colgrove
mathew.colgrove@pgroup.com
July 2013

PGI
- C99, C++, F2003 Compilers
  - Optimizing
  - Vectorizing
  - Parallelizing
- Graphical parallel tools
  - PGDBG® debugger
  - PGPROF® profiler
- AMD, Intel, NVIDIA
- PGI Unified Binary™
- Linux, OS X, Windows
- Visual Studio & Eclipse integration
- GPGPU Features
  - PGI Accelerator™, OpenACC
  - CUDA Fortran/C/C++
  - CUDA-x86
  - OpenCL (ARM CPUs only)
Talk Roadmap

- Introduction to OpenACC
- Compute Constructs
- Loop Directives
- Demo – Himeno
- Data Region
- Porting Example – Seismic CPML
- Future OpenACC features
OpenACC
Open Programming Standard for Parallel Computing

“PGI OpenACC will enable programmers to easily develop portable applications that maximize the performance and power efficiency benefits of the hybrid CPU/GPU architecture of Titan.”

--Buddy Bland, Titan Project Director, Oak Ridge National Lab

“OpenACC is a technically impressive initiative brought together by members of the OpenMP Working Group on Accelerators, as well as many others. We look forward to releasing a version of this proposal in the next release of OpenMP.”

--Michael Wong, CEO OpenMP Directives Board
OpenACC Abstract Machine Architecture

- Multicore CPU
- Execution Queues
- SIMD Units
- Hardware/Software Cache
- Stream Optimized Device Memory

©2012 The Portland Group, Inc.
OpenACC Directives

Program myscience
  ... serial code ...
  !$acc kernels
  do k = 1,n1
    do i = 1,n2
      ... parallel code ...
    enddo
  enddo
  !$acc end kernels
End Program myscience

CPU

GPU

Portable compiler hints

Compiler parallelizes code

Designed for multicore CPUs & many core GPUs / Accelerators

Your original Fortran or C code

OpenACC Compiler Directive
#pragma acc kernels loop
for( i = 0; i < nrows; ++i ){
    float val = 0.0f;
    for( d = 0; d < nzeros; ++d ){
        j = i + offset[d];
        if( j >= 0 && j < nrows )
            val += m[i+nrows*d] * v[j];
    }
    x[i] = val;
}
#pragma acc data \
    copy(b[0:n][0:m]) \
create(a[0:n][0:m]) 
 {
    for (iter = 1; iter <= p; ++iter){
        #pragma acc kernels 
        {
            for (i = 1; i < n-1; ++i){
                for (j = 1; j < m-1; ++j){
                    a[i][j]=w0*b[i][j] +
                    w1*(b[i-1][j]+b[i+1][j]+
                        b[i][j-1]+b[i][j+1])+
                    w2*(b[i-1][j-1]+b[i-1][j+1]+
                        b[i+1][j-1]+b[i+1][j+1]);
                }
            }
            for( i = 1; i < n-1; ++i )
            for( j = 1; j < m-1; ++j )
            b[i][j] = a[i][j];
        }
    }
}
Why use OpenACC Directives?

- Productivity
  - Higher level programming model
  - a la OpenMP

- Portability
  - ignore directives, portable to the host
  - portable across different accelerators
  - *performance portability*

- Performance feedback
Matrix Multiply Source Code Size Comparison:

Directives

CUDA C

OpenCL
Talk Roadmap

- Introduction to OpenACC
- **Compute Constructs**
- Loop Directives
- Demo – Himeno
- Data Region
- Porting Example – Seismic CPML
- Future OpenACC features
Kernels Construct

- **C**

```c
#pragma acc kernels clause...
{
    for( i = 0; i < n; ++i ) r[i] = a[i]*2.0f;
}
```

- **Fortran**

```fortran
$acc kernels clause...
    do i = 1,n
        r(i) = a(i) * 2.0
    enddo
$acc end kernels
```
Parallel Construct

- C

```c
#pragma acc parallel clause...
{
    #pragma acc loop gang vector
    for( i = 0; i < n; ++i ) r[i] = a[i]*2.0f;
}
```

- Fortran

```fortran
 !$acc parallel clause...
 !$acc loop gang vector
 do i = 1,n
   r(i) = a(i) * 2.0
 enddo
 !$acc end parallel
```
Kernels vs. Parallel

- Kernels Construct
  - Derived from PGI Accelerator Model
  - More implicit giving the compiler more freedom to create optimal code for a given accelerator
  - Works best for tightly nested loops
  - May require some additional ‘hints’ to the compiler
    - i.e. C99 restrict keyword

- Parallel Construct
  - Based on OpenMP “workshare”
  - Create parallel gangs that execute redundantly
  - Each gang executes a portion of a work-sharing loop
  - More explicit requiring some user intervention

http://www.pgroup.com/lit/articles/insider/v4n2a1.htm
Talk Roadmap

- Introduction to OpenACC
- Compute Constructs
- Loop Directives
- Demo – Himeno
- Data Region
- Porting Example – Seismic CPML
- Future OpenACC features
Loop Directive

- **C**

  ```c
  #pragma acc loop clause...
  for( i = 0; i < n; ++i ){
    ....
  }
  ```

- **Fortran**

  ```fortran
  !$acc loop clause...
  do i = 1, n
  ```

Note: Compute Constructs and the Loop directive may be combined
Loop Directive Clauses

- **independent**
  - use with care, overrides compiler analysis for dependence, private variables (kernels only, implied with parallel)

- **private(list)**
  - private data for each iteration of the loop

- **reduction(red:var)**
  - reduction across the loop

**Scheduling Clauses**

- **vector** or **vector(width)**
- **gang** or **gang(width)**
- **worker** or **worker(width)**
- **seq**
Loop Scheduling Clauses

- !$acc loop gang
  - runs in ‘gang’ mode only (blockIdx)
  - does not declare that the loop is in fact parallel (use independent)

- !$acc loop gang(32)
  - runs in ‘parallel’ mode only with gridDim == 32 (32 blocks)

- !$acc loop vector(128)
  - runs in ‘vector’ mode (threadIdx) with blockDim == 128 (128 threads)
  - vector size, if present, must be compile-time constant

- !$acc loop gang vector(128)
  - strip mines loop
  - inner loop runs in vector mode, 128 threads (threadIdx)
  - outer loop runs across thread blocks (blockIdx)
Time for a Demo

- Himeno
  - Compute constructs
  - Loop schedules
  - Compiler Feed-back messages
  - Profiling
Talk Roadmap

- Introduction to OpenACC
- Compute Constructs
- Loop Directives
- Demo – Himeno
- Data Region
- Porting Example – Seismic CPML
- Future OpenACC features
Data Region

- C
  
  ```c
  #pragma acc data
  {
    ....
  }
  ```

- Fortran
  
  ```fortran
  !$acc data
  ....
  !$acc end data
  ```

- May span across host code and multiple compute regions
- May be nested
- May not be nested within a compute region
- Data is not implicitly synchronized between the host and device
Data Clauses

- **Data clauses**
  - `copy(list)`
  - `copyin(list)`
  - `copyout(list)`
  - `create(list)`
  - `present(list)`
  - `present_or_copy(list)`
  - `present_or_copyin(list)`
  - `present_or_copyout(list)`
  - `present_or_create(list)`
  - `deviceptr(list)`
  - `pcopy(list)`
  - `pcopyin(list)`
  - `pcopyout(list)`
  - `pcreate(list)`
Data Attributes

- predetermined data attributes
  - loop variables are private
- implicit data attributes
  - array, struct – \texttt{present\_or\_copy}
  - scalar – \texttt{firstprivate}
- explicit data attributes
  - in a data clause
Data Regions Across Procedures

```fortran
subroutine sub( a, b )
   real :: a(:), b(:)
   !$acc kernels pcopyin(b)
   do i = 1,n
      a(i) = a(i) * b(i)
   enddo
   !$acc end kernels
   ...
end subroutine

subroutine bus(x, y)
   real :: x(:), y(:)
   !$acc data copy(x)
   call sub( x, y )
   !$acc end data
```
void sub( float* a, float* b, int n ){
    int i;
    #pragma acc kernels pcopyin(b[0:n])
    for( i = 0; i < n; ++i )
        a[i] *= b[i];
...
}

void bus( float* x, float* y, int n ){
    #pragma acc data copy(x[0:n])
{
    sub( x, y, n );
}    ...

Update Directives

- `update host(list)`
- `update device(list)`
  - Data must be in a data allocate clause for an enclosing data region
  - Both may be on a single line
    - `update host(list) device(list)`
- Data update clauses on data construct (PGI)
  - `updatein(list)` or `update device(list)`
  - `updateout(list)` or `update host(list)`
  - Shorthand for update directive just inside data construct
Additional Concepts not covered today

- Asynchronous Data Movement
- Asynchronous Compute
- Wait Directive
- Cache Directive
- Host_data Directive
- OpenACC and CUDA interoperability
- OpenACC Runtime Library Calls
- OpenACC Environment Variables
- Multi-device programming
- Multi-target accelerators (Unified Binary)
- Performance Optimization
- Data Layout
Talk Roadmap

- Introduction to OpenACC
- Compute Constructs
- Loop Directives
- Demo – Himeno
- Data Region
- Porting Example – Seismic CPML
- Future OpenACC features
Porting SEISMIC CPML

- Set of ten open-source Fortran90 programs
- Solves two-dimensional or three-dimensional isotropic or anisotropic elastic, viscoelastic or poroelastic wave equation.
- Uses finite-difference method with Convolutional or Auxiliary Perfectly Matched Layer (C-PML or ADE-PML) conditions
- Developed by Dimitri Komatitsch and Roland Martin from University of Pau, France.
- Accelerated source used is taken from the 3D elastic finite-difference code in velocity and stress formulation with Convolutional-PML (C-PML) absorbing conditions.

http://www.geodynamics.org/cig/software/seismic_cpml
Full Article: http://www.pgroup.com/lit/articles/insider/v4n1a3.htm
Step 1: Evaluation

- Is my algorithm right for a GPU?
  - SEISMIC_CPML models seismic waves through the earth. Has an outer time step loop with 9 inner parallel loops. Uses MPI and OpenMP parallelization.
- Good candidate for the GPU, but not ideal.
Step 2: Add Compute Regions

```c
!$acc kernels
  do k = kmin,kmax
    do j = NPOINTS_PML+1, NY-NPOINTS_PML
      do i = NPOINTS_PML+1, NX-NPOINTS_PML
        total_energy_kinetic = total_energy_kinetic + 0.5d0 * rho*(vx(i,j,k)**2 + vy(i,j,k)**2 + vz(i,j,k)**2)
        epsilon_xx = ((lambda + 2.d0*mu) * sigmaxx(i,j,k) - lambda * sigmayy(i,j,k) - lambda*sigmazz(i,j,k)) / (4.d0 * mu * (lambda + mu))
        epsilon_yy = ((lambda + 2.d0*mu) * sigmayy(i,j,k) - lambda * sigmaxx(i,j,k) - lambda*sigmazz(i,j,k)) / (4.d0 * mu * (lambda + mu))
        epsilon_zz = ((lambda + 2.d0*mu) * sigmazz(i,j,k) - lambda * sigmaxx(i,j,k) - lambda*sigmayy(i,j,k)) / (4.d0 * mu * (lambda + mu))
        epsilon_xy = sigmaxy(i,j,k) / (2.d0 * mu)
        epsilon_xz = sigmaxz(i,j,k) / (2.d0 * mu)
        epsilon_yz = sigmayz(i,j,k) / (2.d0 * mu)
        total_energy_potential = total_energy_potential + 0.5d0 * (epsilon_xx * sigmaxx(i,j,k) + epsilon_yy * sigmayy(i,j,k) + &
                        epsilon_xx * sigmaxx(i,j,k)+ 2.d0 * epsilon_xy * sigmaxy(i,j,k) + 2.d0*epsilon_xz * sigmaxz(i,j,k)+2.d0*epsilon_yz * sigmayz(i,j,k))
      enddo
    enddo
  enddo
!$acc end kernels
```
% pgfortran -Mmpi=mpich2 -fast -acc -Minfo=accel
seismic_CPML_3D_isotropic_MPI_OACC_1.F90 -o gpu1.out
seismic_cpml_3d_iso_mpi_openmp:
 1107, Generating copyin(vz(11:91,11:631,kmin:kmax))
  Generating copyin(vy(11:91,11:631,kmin:kmax))
  Generating copyin(vx(11:91,11:631,kmin:kmax))
  Generating copyin(sigmaxx(11:91,11:631,kmin:kmax))
  Generating copyin(sigmayy(11:91,11:631,kmin:kmax))
  Generating copyin(sigmazz(11:91,11:631,kmin:kmax))
  Generating copyin(sigmaxy(11:91,11:631,kmin:kmax))
  Generating copyin(sigmaxz(11:91,11:631,kmin:kmax))
  Generating copyin(sigmayz(11:91,11:631,kmin:kmax))
  Generating compute capability 1.3 binary
  Generating compute capability 2.0 binary
1108, Loop is parallelizable
1109, Loop is parallelizable
1110, Loop is parallelizable
  Accelerator kernel generated
1108, !$acc do gang, vector(4)
1109, !$acc do gang, vector(4)
1110, !$acc do vector(16)
1116, Sum reduction generated for total_energy_kinetic
1134, Sum reduction generated for total_energy_potential
### Initial Timings

<table>
<thead>
<tr>
<th>Version</th>
<th>MPI Processes</th>
<th>OpenMP Threads</th>
<th>GPUs</th>
<th>Time (sec)</th>
<th>Approx. Programming Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original MPI/OMP</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>951</td>
<td></td>
</tr>
<tr>
<td>ACC Step 1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3031</td>
<td>10</td>
</tr>
</tbody>
</table>

**System Info:**
- 4 Core Intel Core-i7 920 Running at 2.67Ghz
- Includes 2 Tesla C2070 GPUs
- Problem Size: 101x641x128

**Why the slowdown?**
Step 3: Optimize Data Movement

!$acc data
!$acc     copyin(a_x_half,b_x_half,k_x_half,
!$acc     a_y_half,b_y_half,k_y_half,
!$acc     a_z_half,b_z_half,k_z_half,
!$acc     a_x,a_y,a_z,b_x,b_y,b_z,k_x,k_y,k_z,
!$acc     sigmaxx,sigmaxz,sigmayx,sigmayy,sigmayz,sigmazz,
!$acc     memory_dvx_dx,memory_dvy_dx,memory_dvz_dx,
!$acc     memory_dvx_dy,memory_dvy_dy,memory_dvz_dy,
!$acc     memory_dvx_dz,memory_dvy_dz,memory_dvz_dz,
!$acc     memory_dsigmaxx_dx, memory_dsigmayx_dy,
!$acc     memory_dsigmaxz_dz, memory_dsigmayx_dx,
!$acc     memory_dsigmaxx_dx, memory_dsigmayz_dy,
!$acc     memory_dsigmayy_dy, memory_dsigmayz_dz,
!$acc     memory_dsigmazz_dz)

do it = 1,NSTEP
    .... Cut ....
enddo
!$acc end data
### Timings Continued

<table>
<thead>
<tr>
<th>Version</th>
<th>MPI Processes</th>
<th>OpenMP Threads</th>
<th>GPUs</th>
<th>Time (sec)</th>
<th>Approx. Programming Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original MPI/OMP</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>951</td>
<td></td>
</tr>
<tr>
<td>ACC Step 1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3031</td>
<td>10</td>
</tr>
<tr>
<td>ACC Step 2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>124</td>
<td>180</td>
</tr>
</tbody>
</table>

System Info:
4 Core Intel Core-i7 920 Running at 2.67Ghz
Includes 2 Tesla C2070 GPUs

Data movement time now only 5 seconds!
Step 4: Fine Tune Schedule

```c
!*acc do vector(4)
   do k=k2begin,NZ_LOCAL
       kglobal = k + offset_k
!*acc do gang, vector(4)
   do j=2, NY
!*acc do gang, vector(16)
   do i=1, NX-1
       value_dvx_dx = (vx(i+1,j,k)-vx(i,j,k)) * ONE_OVER_DELTAX
       value_dvy_dy = (vy(i,j,k)-vy(i,j-1,k)) * ONE_OVER_DELTAY
       value_dvz_dz = (vz(i,j,k)-vz(i,j,k-1)) * ONE_OVER_DELTAZ
```
## Final Timings

<table>
<thead>
<tr>
<th>Version</th>
<th>MPI Processes</th>
<th>OpenMP Threads</th>
<th>GPUs</th>
<th>Time (sec)</th>
<th>Approx. Programming Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original MPI/OMP</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>951</td>
<td></td>
</tr>
<tr>
<td>ACC Step 1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3031</td>
<td>10</td>
</tr>
<tr>
<td>ACC Step 2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>124</td>
<td>180</td>
</tr>
<tr>
<td>ACC Step 3</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>120</td>
<td>120</td>
</tr>
</tbody>
</table>

7x in 5 Hours!
# Cluster Timings

<table>
<thead>
<tr>
<th>Version</th>
<th>Size</th>
<th>MPI Processes</th>
<th>OpenMP Threads</th>
<th>GPUs</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI/OMP</td>
<td>101x641x3072</td>
<td>24</td>
<td>96</td>
<td>0</td>
<td>2081</td>
</tr>
<tr>
<td>MPI/ACC</td>
<td>101x641x3072</td>
<td>24</td>
<td>0</td>
<td>24</td>
<td>446</td>
</tr>
</tbody>
</table>

System Info: 8 Nodes  
2 socket, 6 Core Intel X5675 Running at 3.06Ghz with 3 Tesla C2070 GPU  
Compiler: PGI 2012 version 12.5

Just under 5x!
Talk Roadmap

- Introduction to OpenACC
- Compute Constructs
- Loop Directives
- Demo – Himeno
- Data Region
- Porting Example – Seismic CPML
- Future OpenACC features
OpenACC 2.0 Highlights

- Procedure calls, separate compilation
- Nested parallelism
- Data management features and global data
- atomic operations
Currently Procedure Calls in Compute Regions Must Be Inlined

```c
#pragma acc parallel loop num_gangs(200)...
for( int i = 0; i < n; ++i ){
    v[i] += rhs[i];
    matvec( v, x, a, i, n );
    // must inline matvec
}
```

```bash
pgcc  -Minline a.c
pgcc  -Mextract=lib:mylib b.c
pgcc  -Minline=lib:mylib a.c
pgcc  -Minline=levels:10 a.c
pgcc  -Minline=levels:2,foo,phoo,bar,lib:mylib a.c
```
#pragma acc routine worker
extern void matvec(float* v, float* x, ...);

...
#pragma acc parallel loop num_gangs(200)...
for (int i = 0; i < n; ++i ){
    v[i] += rhs[i];
    matvec( v, x, a, i, n );
    // procedure call on the device
}

#pragma acc routine worker
void matvec( float* v, float* x,
             float* a, int i,
             int n ){
    float xx = 0;
    #pragma acc loop reduction(+:xx)
    for (int j = 0; j < n; ++j )
        xx += a[i*n+j]*v[j];
    x[i] = xx;
}
OpenACC 2.0 routine bind

```c
#pragma acc routine worker bind(mvdev)
extern void matvec(float* v, float* x, ...);
...
#pragma acc parallel loop num_gangs(200)...
for( int i = 0; i < n; ++i ){
    v[i] += rhs[i];
    matvec( v, x, a, i, n );
}
```
void matvec( float* v, float* x,
             float* a, int i, int n ){
    float xx=0.0;
    for( int j = 0; j < n; ++j )
        xx += a[i*n+j]*v[j];
    x[i] = xx;
}

#pragma acc routine worker nohost
void mvdev( float* v, float* x,
            float* a, int i, int n ){
    float xx = 0.0;
    #pragma acc loop reduction(+:xx)
    for( int j = 0; j < n; ++j )
        xx += a[i*n+j]*v[j];
    x[i] = xx;
}
Nested Parallelism

#pragma acc routine
extern void matvec(float* v, float* x,...);
...
#pragma acc parallel loop ...
for( int i = 0; i < n; ++i )
    matvec( v, x, i, n );

#pragma acc routine
matvec(...){
    #pragma acc parallel loop
    for( int i = 0; i < n; ++i ){...}
Nested Parallelism cont.

```c
#pragma acc routine
extern void matvec(float* v, float* x, ...);
...
#pragma acc parallel num_gangs(1)
{
    matvec(v0, x0, i, n);
    matvec(v1, x1, i, n);
    matvec(v2, x2, i, n);
}

#pragma acc routine
matvec(...){
#pragma acc parallel loop
    for(int i = 0; i < n; ++i){...}
```
Dynamic Data Lifetimes

void init( int n ){
...
#pragma acc enter data copyin( x[0:100] )
...
#pragma acc enter data present_or_create( y[0:n] )
...
}

void fini( int n ){
...
#pragma acc exit data delete( x[0:100] )
...
#pragma acc exit data copyout( y[0:n] )
...
}
Global Data

float x[1000];
#pragma acc declare create(x)
// static allocation, host + device

float y[1000];
#pragma acc declare device_resident(y)
// static allocation, device-only

float z[10000];
#pragma acc declare link(z)
// static allocation on host, dynamic allocation on device
atomic operations

#pragma acc parallel loop
for( j = 1; j < n-1; ++j ){
    y = x[j];
    i = y & 0xf;
    #pragma acc atomic update
        ++bin[i];
}

// essentially the OpenMP atomic operations
// atomic update, read, write, capture
// only native data lengths supported
Other OpenACC 2.0 Features Not Covered Today

- Device-specific tuning, multiple devices
- Multiple host thread support
- Loop directive additions
- Asynchronous behavior additions
- New API routines
Key Value-add of the PGI Accelerator Compilers

PGI compilers are designed to enable performance-portable programming for all many-core and GPU Accelerator-based systems without having to re-program for each new successive hardware advancement.
Copyright Notice

© Contents copyright 2013, The Portland Group, Inc. This material may not be reproduced in any manner without the expressed written permission of The Portland Group.

PGFORTRAN, PGF95, PGI Accelerator and PGI Unified Binary are trademarks; and PGI, PGCC, PGC++, PGI Visual Fortran, PVF, PGI CDK, Cluster Development Kit, PGPROF, PGDBG, and The Portland Group are registered trademarks of The Portland Group Incorporated. Other brands and names are the property of their respective owners.