Programming NVIDIA GPUs with OpenACC Directives

Michael Wolfe
michael.wolfe@pgroup.com

PGI®

http://www.pgroup.com/accelerate

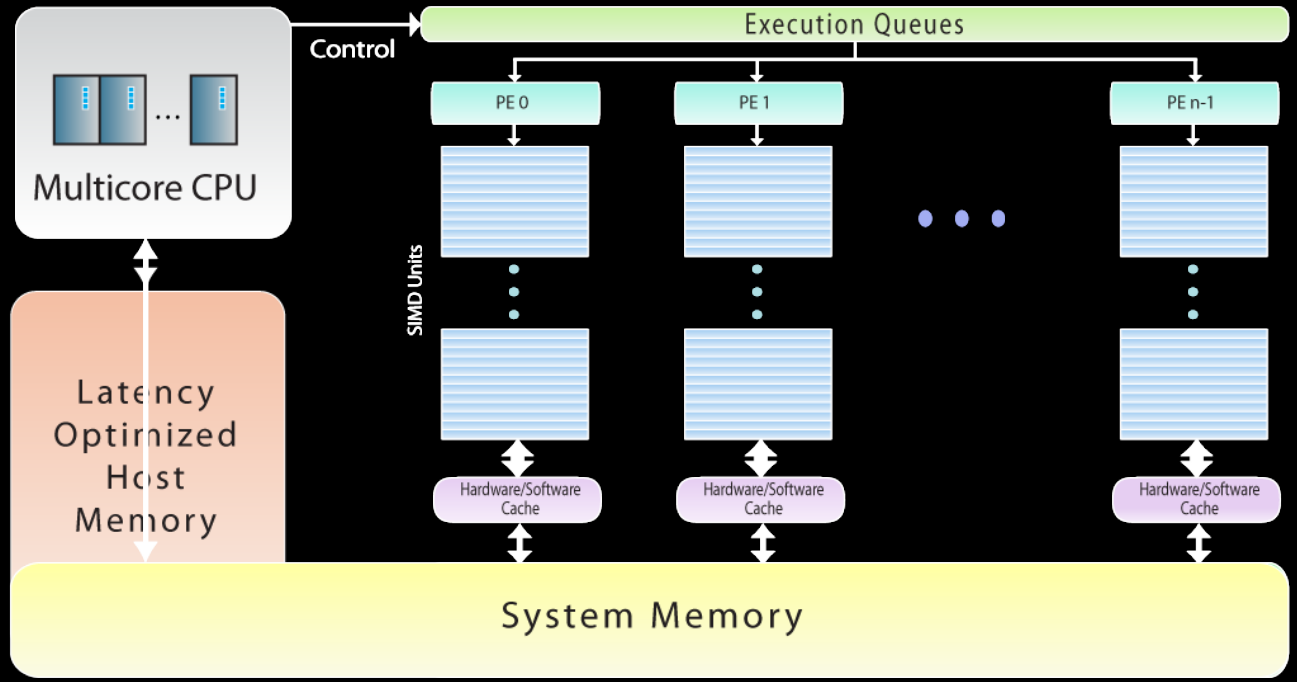
Programming NVIDIA GPUs with OpenACC Directives

Michael Wolfe
mwolfe@nvidia.com

PGI®

http://www.pgroup.com/accelerate

The New HPC Node Architecture



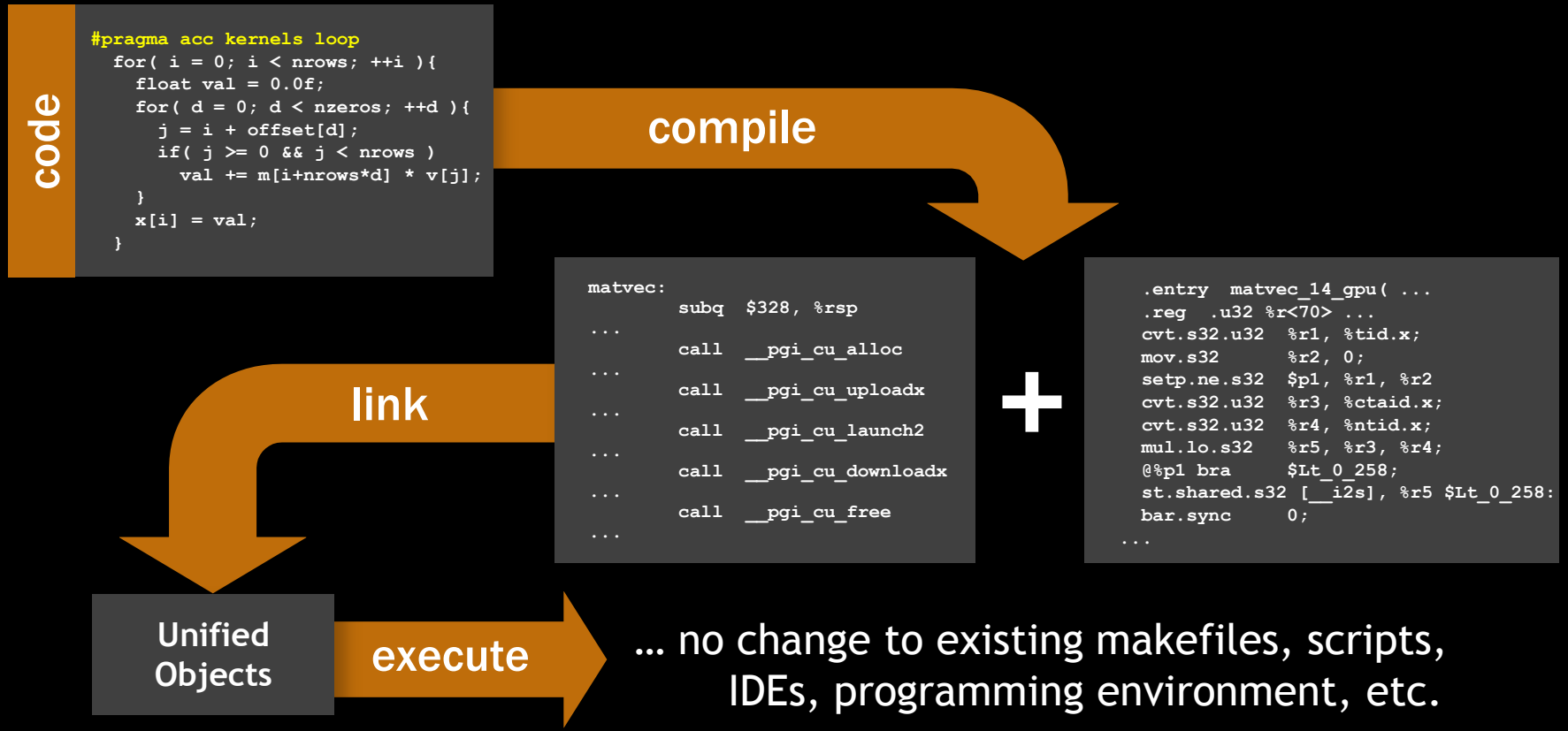
OpenACC Coding Example

```
#pragma acc data copy(b[0:n*m]) create(a[0:n*m])
{
  for (iter = 1; iter <= p; ++iter){
    #pragma acc parallel loop present(b[0:n*m], a[0:n*m])
      for (i = 1; i < n-1; ++i)
        for (j = 1; j < m-1; ++j)
          a[i*m+j]=w0*b[i*m+j]+
                   w1*(b[(i-1)*m+j]+b[(i+1)*m+j]+
                       b[i*m+j-1]+b[i*m+j+1])+
                   w2*(b[(i-1)*m+j-1]+b[(i-1)*m+j+1]+
                       b[(i+1)*m+j-1]+b[(i+1)*m+j+1]);
      tmp = a;
      a = b;
      b = tmp;
  }
}
```

OpenACC™ API

- CAPS, Cray, NVIDIA, PGI (and more)
- Directives similar to OpenMP
  - control data movement to/from device memory
  - control parallel loops on the device
- OpenACC 2.0 features
  - procedure calls
  - nested parallelism
  - unstructured data lifetimes

Code, Compile & Run Workflow is Unchanged



… no change to existing makefiles, scripts, IDEs, programming environment, etc.

OpenACC Coding Example

```
#pragma acc data copy(b[0:n*m]) create(a[0:n*m])
{
  for (iter = 1; iter <= p; ++iter){
    #pragma acc parallel loop present(b[0:n*m], a[0:n*m])
      for (i = 1; i < n-1; ++i)
        for (j = 1; j < m-1; ++j)
          a[i*m+j]=w0*b[i*m+j]+
                   w1*(b[(i-1)*m+j]+b[(i+1)*m+j]+
                       b[i*m+j-1]+b[i*m+j+1])+
                   w2*(b[(i-1)*m+j-1]+b[(i-1)*m+j+1]+
                       b[(i+1)*m+j-1]+b[(i+1)*m+j+1]);
      tmp = a;
      a = b;
      b = tmp;
  }
}
```

OpenACC Coding Example

```
for (iter = 1; iter <= p; ++iter){
  #pragma acc parallel loop present(b[0:n*m], a[0:n*m])
    for (i = 1; i < n-1; ++i){
      #pragma acc loop vector
      for (j = 1; j < m-1; ++j)
        a[i*m+j]=w0*b[i*m+j]+
                 w1*(b[(i-1)*m+j]+b[(i+1)*m+j]+
                     b[i*m+j-1]+b[i*m+j+1])+
                 w2*(b[(i-1)*m+j-1]+b[(i-1)*m+j+1]+
                     b[(i+1)*m+j-1]+b[(i+1)*m+j+1]);
    }
    tmp = a;
    a = b;
    b = tmp;
}
```

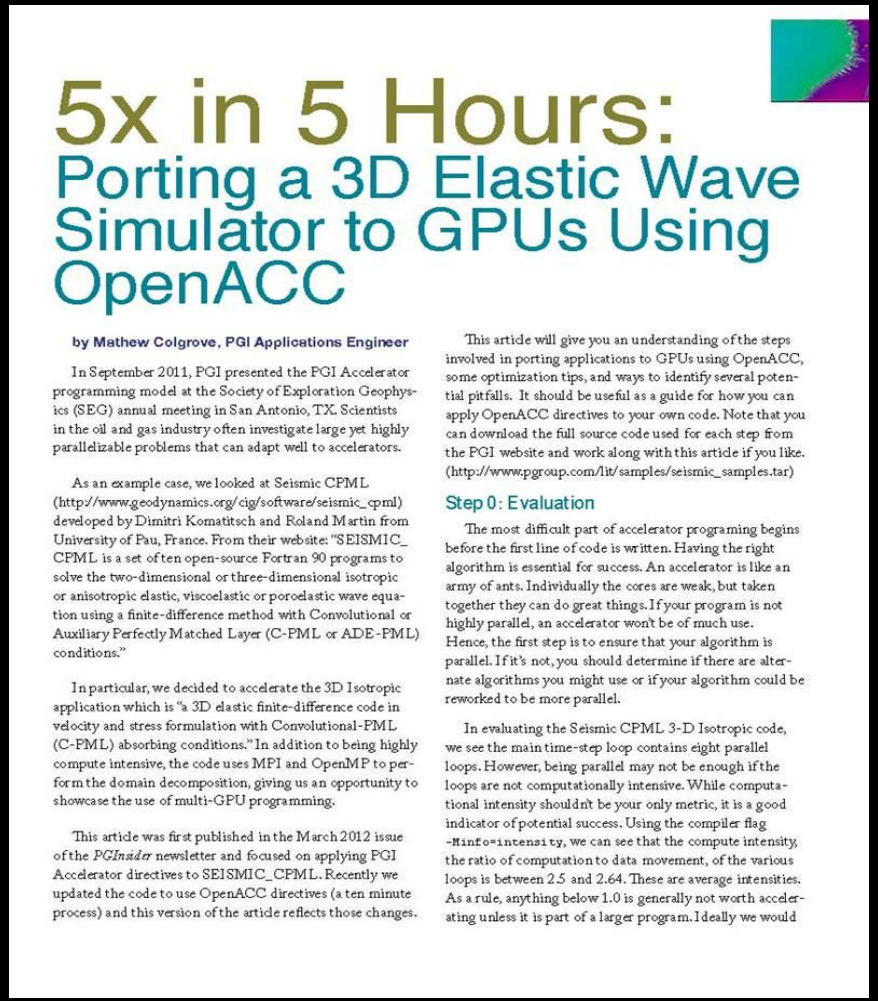
Performance Portability

```
% pgcc -acc -ta=nvidia relax.c

relax:
      6, Generating present(b[0:n*m])
         Generating present(a[0:n*m])
      7, Accelerator kernel generated
         8, #pragma acc loop gang /* blockIdx.x */
        10, #pragma acc loop vector(256) /* threadidx.x */
      7, Generating NVIDIA code
         Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
         Generating compute capability 3.0 binary
     10, Loop is parallelizable
```

Accelerating SEISMIC_CPML from the University of Pau

Read this article online at www.pgroup.com/pginsider



5x in 5 Hours:
Porting a 3D Elastic Wave Simulator to GPUs Using OpenACC

by Mathew Colgrove, PGI Applications Engineer

In September 2011, PGI presented the PGI Accelerator programming model at the Society of Exploration Geophysics (SEG) annual meeting in San Antonio, TX. Scientists in the oil and gas industry often investigate large yet highly parallelizable problems that can adapt well to accelerators.

As an example case, we looked at Seismic CPML (http://www.geodynamics.org/cig/software/seismic_cpml) developed by Dimitri Komatitsch and Roland Martin from University of Pau, France. From their website: "SEISMIC_CPML is a set of ten open-source Fortran 90 programs to solve the two-dimensional or three-dimensional isotropic or anisotropic elastic, viscoelastic or poroelastic wave equation using a finite-difference method with Convolutional or Auxiliary Perfectly Matched Layer (C-PML or ADE-PML) conditions."

In particular, we decided to accelerate the 3D Isotropic application which is "a 3D elastic finite-difference code in velocity and stress formulation with Convolutional-PML (C-PML) absorbing conditions." In addition to being highly compute intensive, the code uses MPI and OpenMP to perform the domain decomposition, giving us an opportunity to showcase the use of multi-GPU programming.

This article was first published in the March 2012 issue of the *PGInsider* newsletter and focused on applying PGI Accelerator directives to SEISMIC_CPML. Recently we updated the code to use OpenACC directives (a ten minute process) and this version of the article reflects those changes.

This article will give you an understanding of the steps involved in porting applications to GPUs using OpenACC, some optimization tips, and ways to identify several potential pitfalls. It should be useful as a guide for how you can apply OpenACC directives to your own code. Note that you can download the full source code used for each step from the PGI website and work along with this article if you like. (http://www.pgroup.com/lit/samples/seismic_samples.tar)

Step 0: Evaluation

The most difficult part of accelerator programing begins before the first line of code is written. Having the right algorithm is essential for success. An accelerator is like an army of ants. Individually the cores are weak, but taken together they can do great things. If your program is not highly parallel, an accelerator won't be of much use. Hence, the first step is to ensure that your algorithm is parallel. If it's not, you should determine if there are alternate algorithms you might use or if your algorithm could be reworked to be more parallel.

In evaluating the Seismic CPML 3-D Isotropic code, we see the main time-step loop contains eight parallel loops. However, being parallel may not be enough if the loops are not computationally intensive. While computational intensity shouldn't be your only metric, it is a good indicator of potential success. Using the compiler flag `-Minfo=intensity`, we can see that the compute intensity, the ratio of computation to data movement, of the various loops is between 2.5 and 2.64. These are average intensities. As a rule, anything below 1.0 is generally not worth accelerating unless it is part of a larger program. Ideally we would

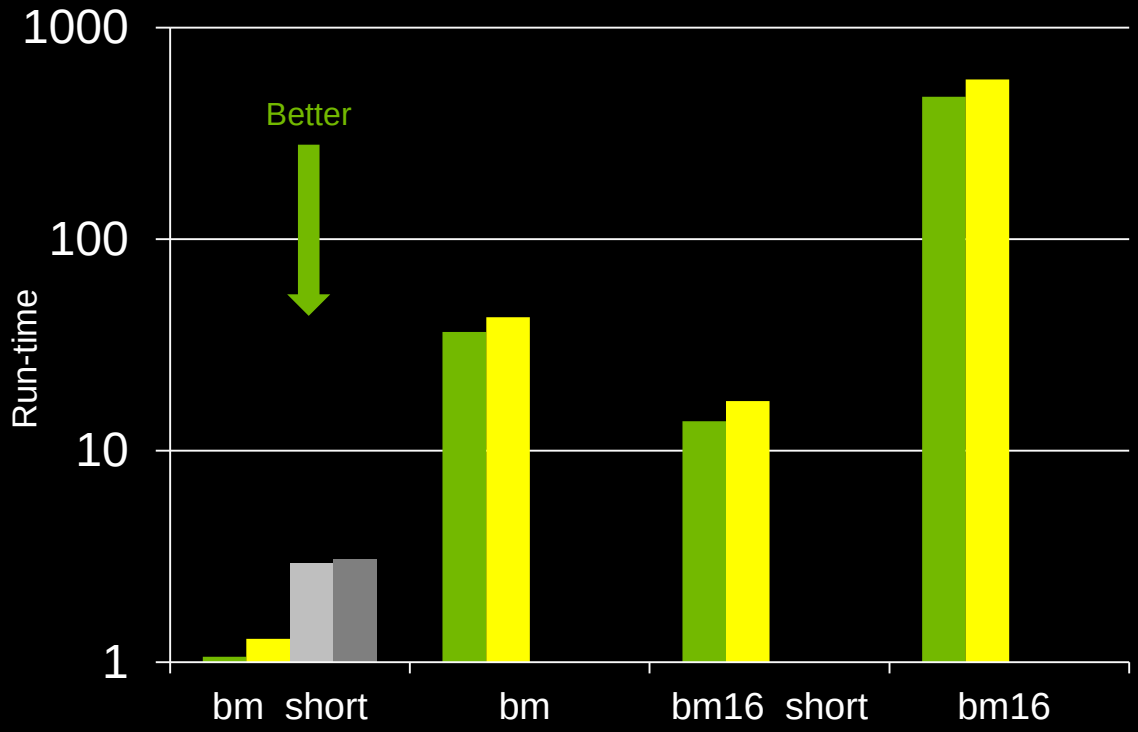
SEISMIC_CPML Timings

Version	MPI Processes	OpenMP Threads	GPUs	Time (sec)	Approx. Programming Time (min)
Original MPI/OMP	2	4	0	951	
ACC Steps 1/2	2	0	2	3100	10
ACC Step 3	2	0	2	550	60
ACC Step 4	2	0	2	124	120
ACC Step 5	2	0	2	120	120

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

5x in 5 hours!

Cloverleaf mini-App Performance



K20X CUDA

K20X OpenACC

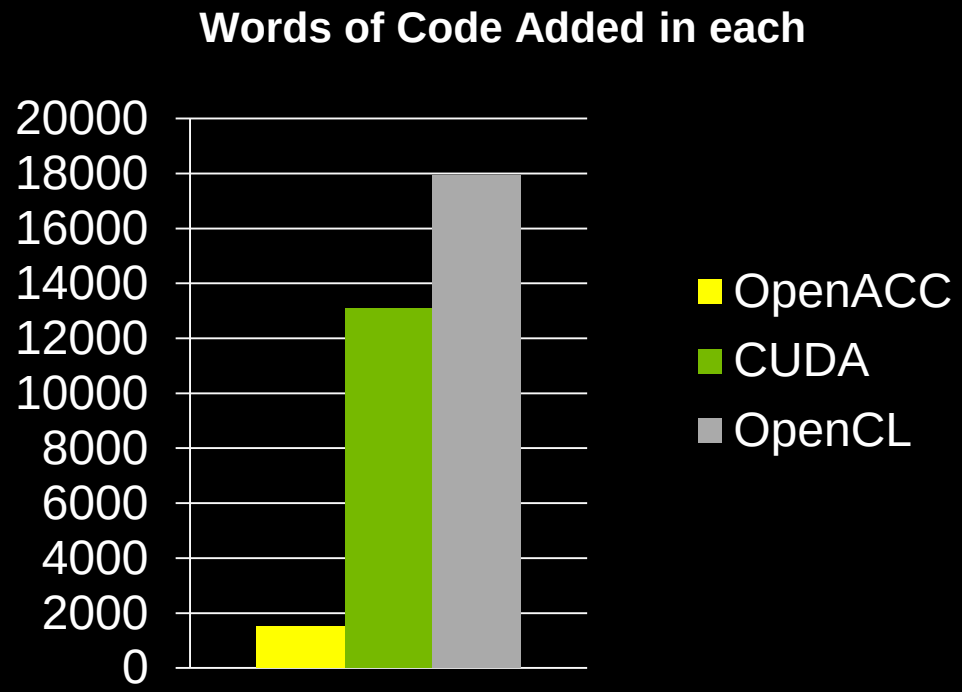
Dual-socket CPU C

Dual-socket CPU Fortran

NVIDIA benchmarks: dual-socket Intel Xeon E5-2667

Cloverleaf is a Trinity/Coral mini-app benchmark developed by AWE https://github.com/Warwick-PCAV/CloverLeaf/wiki/Performance-Table

OpenACC: Performance with Less Effort



Cloverleaf: http://www.computer.org/csdl/proceedings/sccompanion/2012/4956/00/4956a465-abs.html

OpenACC Applications Porting Activity

Geology	Weather/Climate/Ocean	Plasma & Combustion	Fluid Dynamics / Cosmology	Chemistry
AWP-ODC	CAM-SE	Cloverleaf	PMH bv DELPASS	GAMESS CCSD(T)
EMGS ELAN	COSMO Physics	GENE	DNS	GAUSSIAN
Seismic CPML	FIM	GTC	MiniGHOST	MiniMD
SPECFM3D	GEOS-5	LULESH	RAMSES	Quantum Espresso
TeraP	Harmonie	S3D	UPACS	
	HBM		X-ECHO	
	ICON			
	NICAM			
	NEMO GYRE			
	NIM			
	PALM-GPU			
	ROMS			
	WRF			

Other US efforts: 8 new OpenACC efforts begin May 2013

- Almost all Fortran, some C/C++
  - Most OpenACC + MPI / OpenMP
  - Some OpenACC + libraries + CUDA
  - C++ are all “mini Apps”
- Many are 100K to 1M+ lines of code
  - 5 to 50 kernels of multi-disciplinary science
- PGI, Cray, CAPS OpenACC compilers all being used
- 24 different lead developers
  - 10 Europe, 3 Asia, 12 North America

OpenACC 2.next Development

- Struct/Derived type support
  - array members of struct / derived type
- C++ support
  - class members, class member functions, templated classes, STL <<vector>>
- Bit-exact option
- Profiler interface

Easy?

- **Streams:** Parallel programming made easy
- **NESL:** Making parallel programming easy and portable
- **CxC:** Makes parallel programming easy and efficient
- **ParLab:** Goal to make it easy to write correct, scalable parallel programs
- **UPCRC:** Make parallel programming synonymous with programming
- **Swift:** The easy scripting language for parallel computing

Using OpenACC Directives and PGI Accelerator Compilers

- Appropriate algorithm (think nested parallel loops)
- Appropriate data structure (vectors, arrays, simple indexing)
- Read the –Minfo messages
- Manage data moving to and from GPU (CUDA or data regions)
- Optimize, tune for strides, locality
- Accelerator-enabled and Host-only in same binary
- Performance portability

http://www.pgroup.com/accelerate