Introduction to OpenACC

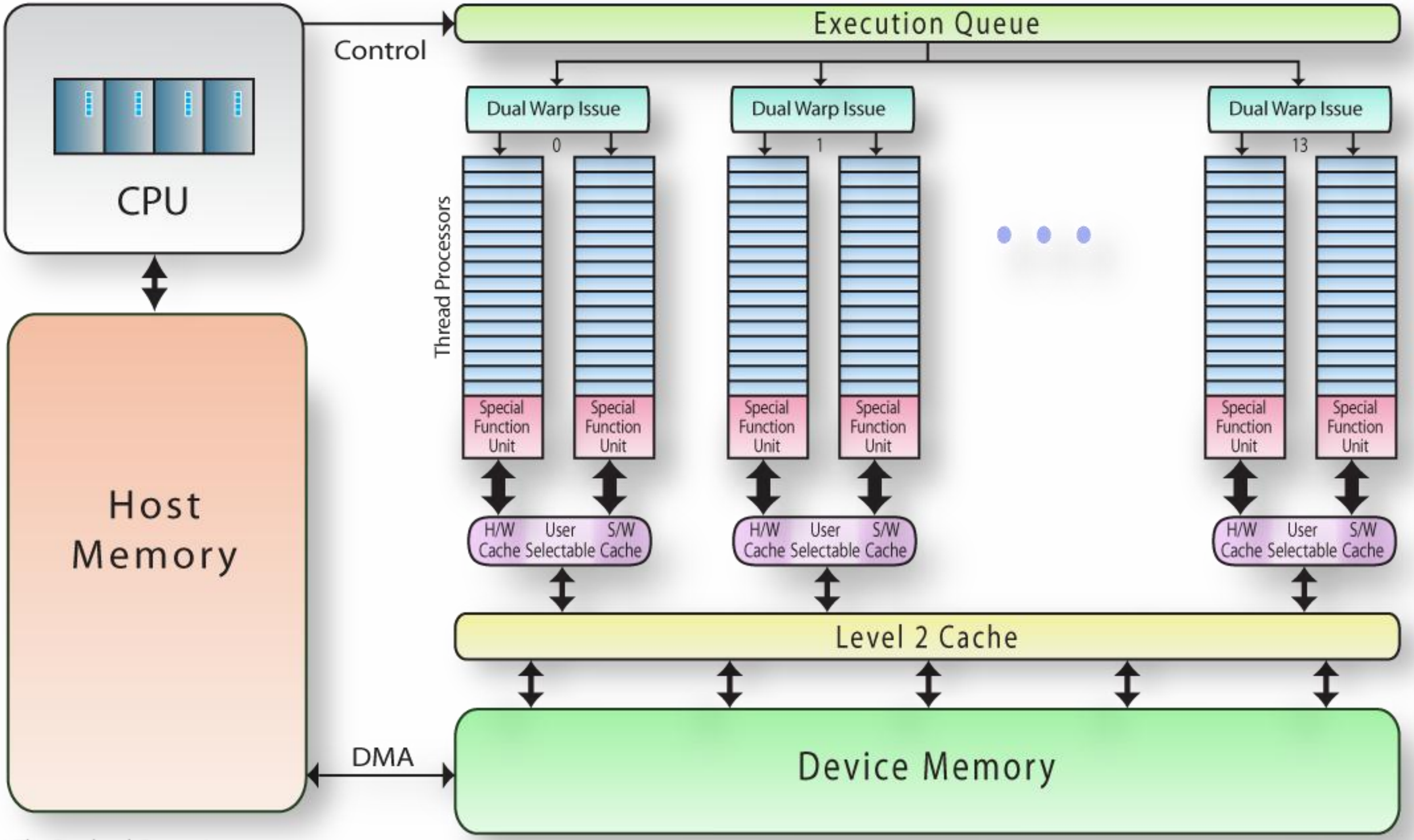
CUG 2012

Brent Leback
Brent.Leback@pgroup.com
http://www.pgroup.com

Accelerate!

What is OpenACC?

- PGI Introduced its Accelerator Programming Model in 2008
    - It has gone through a few iterations, is currently at version 1.3
- Goal then and now is to produce a higher–level model than CUDA, aimed at scientists and engineers
- Wanted something that is directive-based, in the spirit of OpenMP
- A few alternate but similar models were introduced later
    - CAPS, Cray
- OpenMP committee attempted to standardize a model in 2010
- Meeting of the minds last fall, OpenACC 1.0 announced at SC11
- See www.openacc.org


Execution Queue
Control
CPU
Dual Warp Issue
0
Dual Warp Issue
1
Dual Warp Issue
13
Thread Processors
Special Function Unit
H/W Cache
User Selectable
S/W Cache
Host Memory
Level 2 Cache
DMA
Device Memory

GPU Architecture Features

- Optimized for high degree of regular parallelism
- Outer do-all parallelism is fully parallel across the multiprocessors
- SIMD parallelism within a multiprocessor, which can synchronize and share data
- High bandwidth memory, support for ECC
- Highly multithreaded (slack parallelism) with hardware thread scheduling
- Non-coherent hw data caches, sw managed shared memory
- No multiprocessor memory model guarantees
  - Low-level atomic functions available, but not generally recommended

GPU Programming Constants

The Program must:

- Initialize/Select the GPU to run on
- Allocate data on the GPU
- Move data from host, or initialize data on GPU
- Launch kernel(s)
- Gather results from GPU
- Deallocate data

CUDA Fortran, an Explicit Language

```
use vaddmod
real, device, dimension(:), allocatable :: da, db, dc

allocate( da(1:n), db(1:n), dc(1:n) )

db = b
dc = c

call vadd<<<min((n+255)/256,65535),256>>>( da, db, dc, n )

a = da

deallocate( da, db, dc )
```

Implicit Model

```
!$acc kernels loop
do i = 1, n
   a(i) = b(i) + c(i)
enddo
```

```
#pragma acc kernels loop
for( i = 0; i < n; ++i )
   a[i] = b[i] + c[i];
```

Compiler determines

- Need to allocate a, b, c of length n on the device
- Copyin b and c
- Need to generate and call a kernel for the specified operation, decide on a launch configuration
- Copyout a
- Deallocate a, b, and c

Data Directives

```
 C    INITIALIZE CONSTANTS AND ARRAYS
 C
       CALL ALLOC
!$ACC DATA CREATE(U(:NP1,:MP1), V(:NP1,:MP1))
!$ACC& CREATE(UNEW(:NP1,:MP1), VNEW(:NP1,:MP1))
!$ACC& CREATE(PNEW(:NP1,:MP1), UOLD(:NP1,:MP1))
!$ACC& CREATE(VOLD(:NP1,:MP1), POLD(:NP1,:MP1))
!$ACC& CREATE(CU(:NP1,:MP1), CV(:NP1,:MP1))
!$ACC& CREATE(P(:NP1,:MP1), Z(:NP1,:MP1))
!$ACC& CREATE(H(:NP1,:MP1), PSI(:NP1,:MP1))

       CALL INITAL

        . . .

!$ACC END DATA
```

- **Defines a region where arrays should be allocated on the device**
- **Often just one large data region per program**
- **Clauses define copy behavior**
- **Use present clauses in subprograms**

The Kernel Construct

```
#pragma acc kernels loop
    copyin(b[0:n*m]) copy(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]);
```

- **Most like the PGI Accelerator region**
- **Contains loop constructs, can generate multiple kernels**
- **Compiler is free to schedule kernels onto hardware, but schedule can be directed with clauses**
- **Currently requires tightly nested loops, no undersubscribed gangs**

The Parallel Construct

```
!$acc parallel
! Do some redundant gang work here
!$acc loop gang
do j = 1, n
    p1 = posin(j,1)
    p2 = posin(j,2)
    p3 = posin(j,3)
    f1 = 0.0; f2 = 0.0; f3 = 0.0
!$acc loop worker, reduction(+:f0,f1,f2)
    do i = 1, n
        r1 = posin(i,1) - p1
        r2 = posin(i,2) - p2
        r3 = posin(i,3) - p3
        distsqr = r1*r1 + r2*r2 + r3*r3
        . . .
```

- **Most like an OpenMP Parallel region**
- **Contains loop constructs, generates one kernel**
- **Compiler schedule is fixed by num_gangs and num_workers**
- **One worker in each gang executes redundantly until a work-sharing loop is encountered**

Some Simple Compiler Tips

- The compiler flag to enable OpenACC is
  - -acc [= strict | verystrict]
  - Use this in combination with the –ta=nvidia target options
- Use the –Minfo=accel option to enable compiler feedback
- PGI Accelerator Model and OpenACC can coexist in the same program, as can CUDA Fortran and OpenACC. They can share features.
- Use the PGI_ACC_TIME environment variable to get a quick accounting of data transfer between host and device, and some quick kernel statistics

New/Upcoming Features in PGI Accelerator Compilers

- **CUDA-x86 Compilers officially released in January**
- **PGI OpenCL Compilers for ARM announced in March**
- **CUFFT interface modules for CUDA Fortran in PGI 12.5**
- **Support for OpenACC parallel construct in PGI 12.5**
- **Support for CUDA 4.2, Kepler coming soon**
- **CUDA Fortran support for textures coming soon**
- **PGI OpenACC release 1.0 in June or July 2012**
- **Work on generating PTX directly, with dwarf, is underway**
- **PGI Accelerator Model 2.0 specification will be out by ISC12**