OpenACC 2.5 and Beyond

Michael Wolfe

PGI compiler engineer

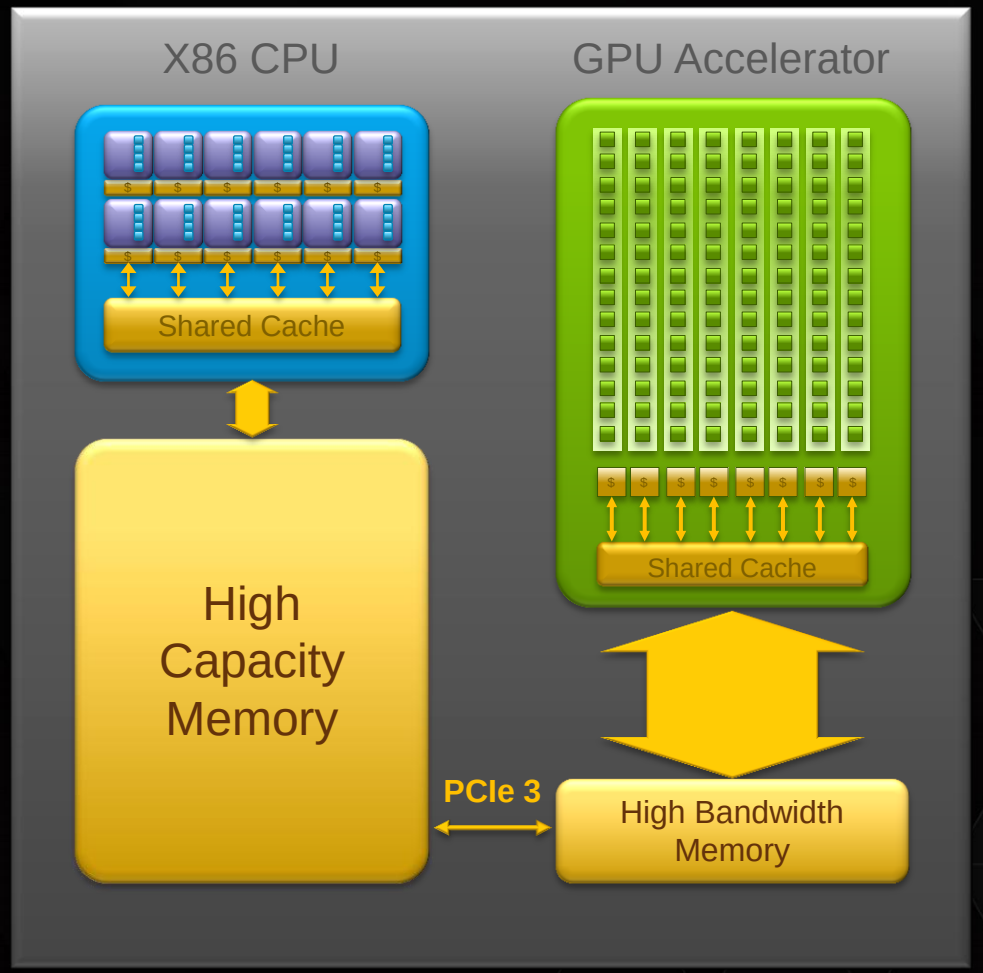
michael.wolfe@pgroup.com

OpenACC Timeline

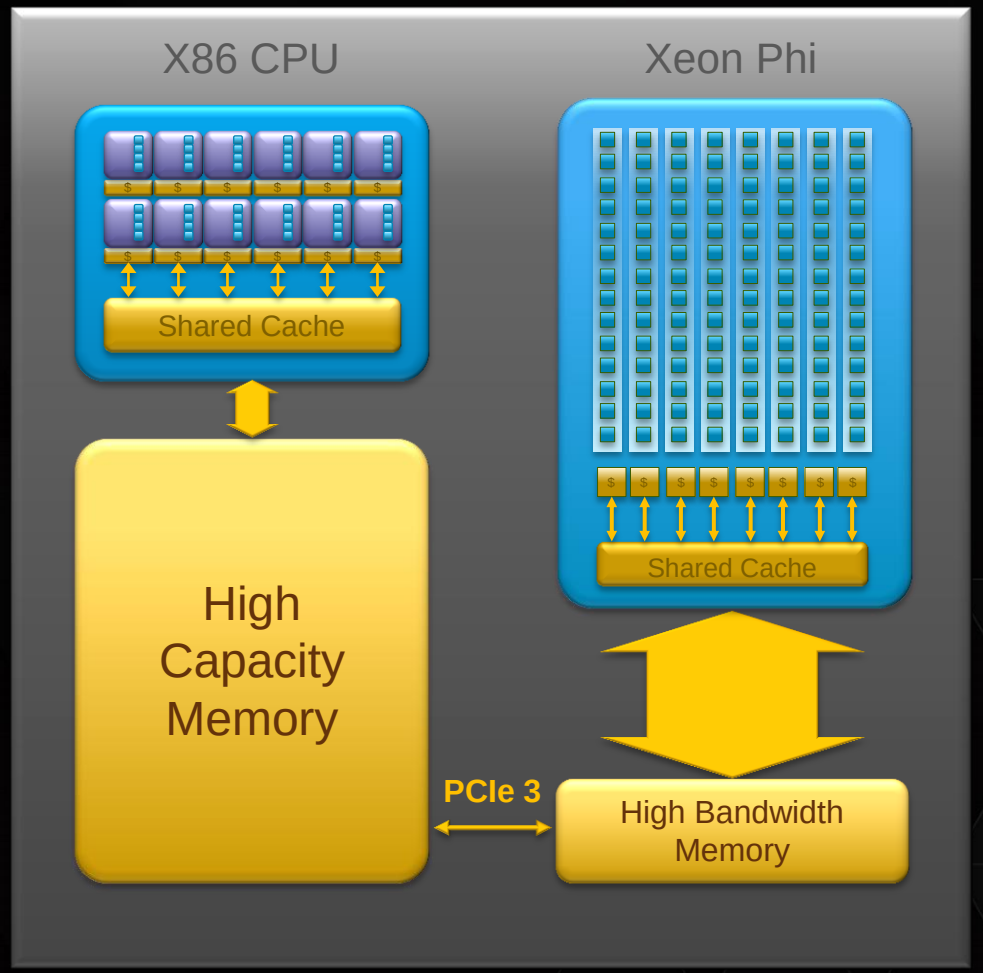
- 2008 – PGI Accelerator Model (targeting NVIDIA GPUs)
- 2011 – OpenACC 1.0 (targeting NVIDIA GPUs, AMD GPUs)
  - data regions, compute regions, gang/worker/vector
- 2013 – OpenACC 2.0
  - procedures, dynamic data lifetimes
- 2015 – OpenACC 2.5
  - minor fixes, additions
- 2015/16 – OpenACC 3.0
  - deep copy

S5388 – OpenACC for Fortran Programmers, Wednesday, 2pm
S5233 – OpenACC and C++, Thursday, 9am
S5196 – Comparing OpenMP and OpenACC, Thursday 4pm
S5864 – Hangout – Thursday 5pm
S5198 – Panel on GPU Computing with OpenACC and OpenMP, Friday 11am

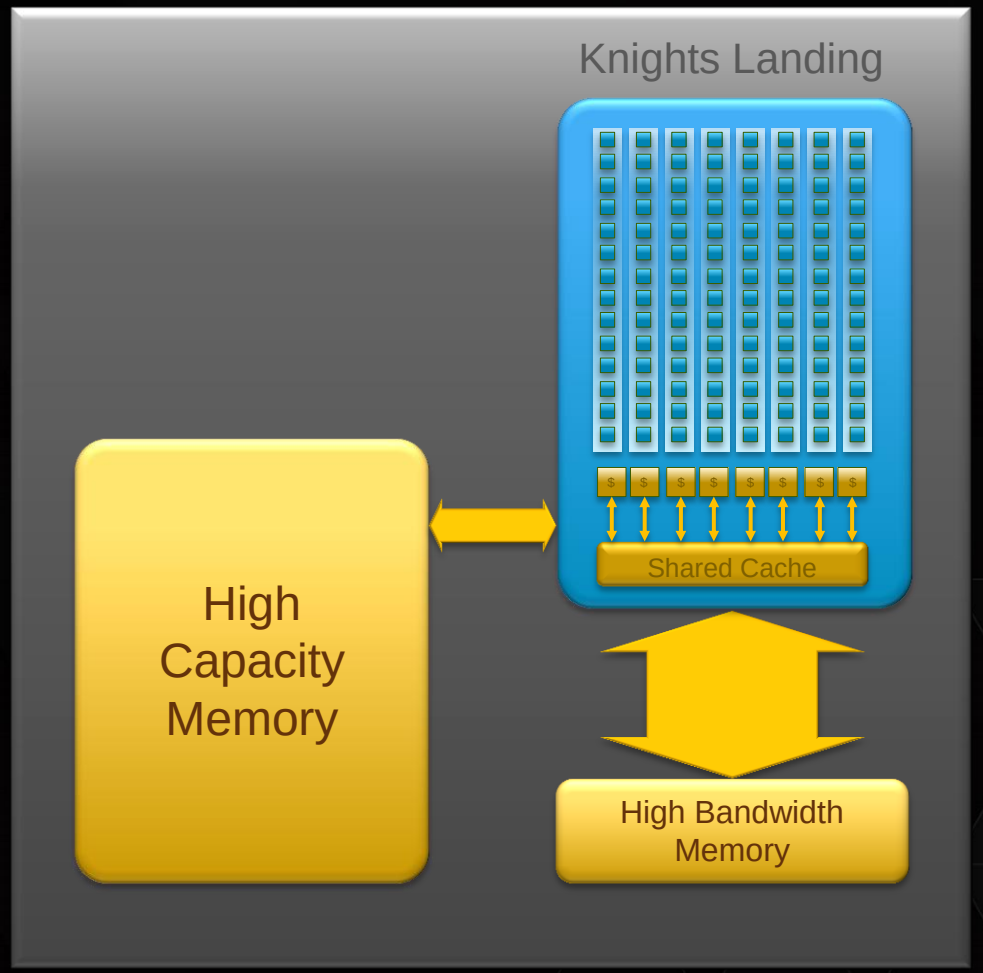
Modern HPC Node



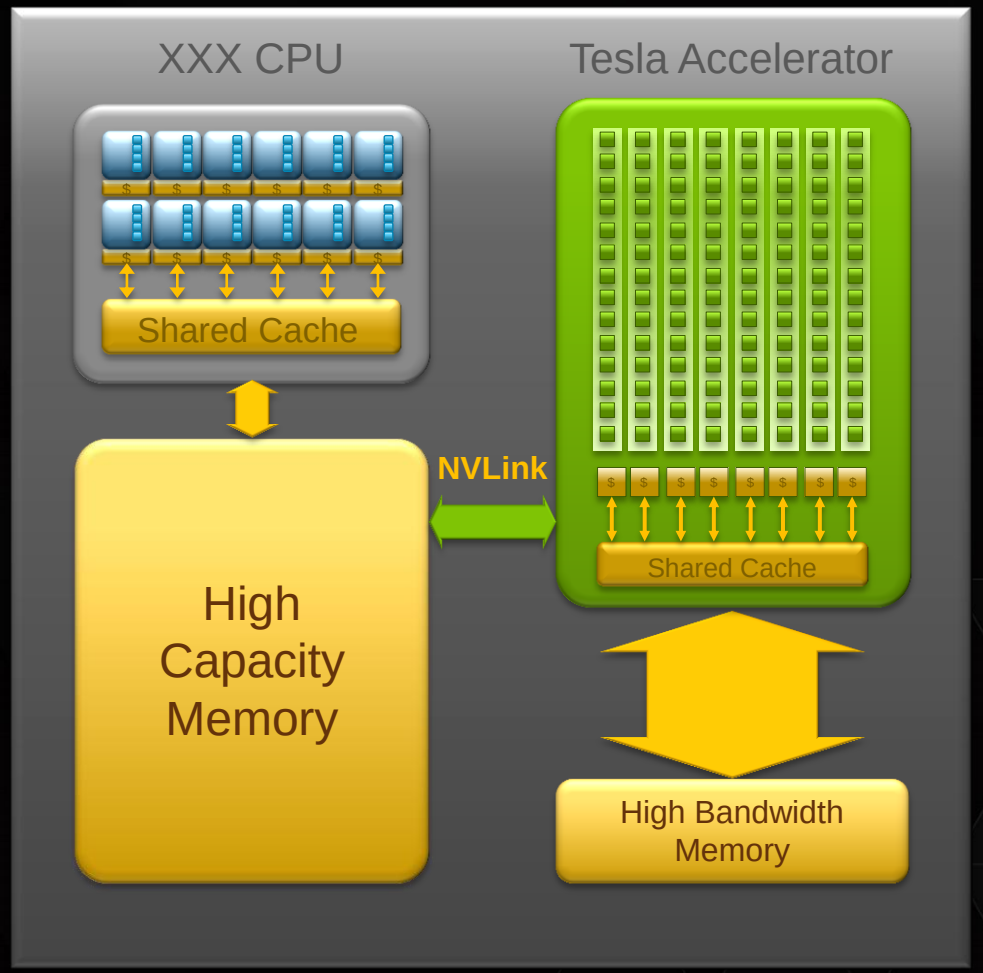
Modern HPC Node



Modern HPC Node



Modern HPC Node



OpenACC 2.5 Topics

- Data Clause Changes
- API Routines Equivalent to Directives, and Vice Versa
- New Restrictions
- Some minor fixes (changes)
- Profile Tool Interface

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
#pragma acc data copy(a[0:n])
{
    foo( a );
}
....
void foo( float* a ){
    #pragma acc data copy(a[0:n])
    {
        #pragma acc parallel...
    }
}
```

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
#pragma acc data copy(a[0:n])
{
    foo( a );
}
....
void foo( float* a ){
    #pragma acc enter data \
                        copyin(a[0:n])
    #pragma acc parallel...
}
```

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
#pragma acc data copy(a[0:n])
{
    foo( a );
}
....
void foo( float* a ){
    #pragma acc enter data \
                        copyin(a[0:n])
    #pragma acc parallel...
    #pragma acc exit data \
                       copyout(a[0:n])
}
```

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
void foo( float* a, float* b ){
  #pragma acc data copyin(a[0:n]) \
                   copyout(b[0:n])
  {
     #pragma acc parallel loop
     for(i=0;i<n;++i)
        b[i] = a[i]*expf(a[i]);
  }
}


float* x;

foo( x, x );
```

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
module m
real, allocatable :: a(:)
!$acc declare create(a)
end module
...
use m
allocate(a(1:n))
...
!$acc parallel loop
 do i = 1, n
   a(i) = a(i) * 2.0
 enddo
!$acc update self(a)
```

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
void foo( float* a, float* b ){
  #pragma acc update self(a[0:n])
  for(i=0;i<n;++i)
    b[i] = a[i]*expf(a[i]);
  #pragma acc update device(b[0:n])
}
...
foo( x, y );
...
#pragma acc data copy(x[0:n],y[0:n])
{
    ...
    foo( x, y );
    ...
}
```

OpenACC 2.5 Data Clause Changes

- copy* == present_or_copy*
  - create == present_or_create
- reference counting behavior exposed
- exit data delete/copyout changes
- define behavior in the face of aliases
- real, allocatable::a
  !$acc declare create(a)
- update device(a) or update self(a)
  - if ‘a’ is not present ==> no-op
- default(present)

```
#pragma acc parallel loop \
   default(present)
for( i=0;i<n;++i )
  a[i] = b[i]*x + c[i]*y;
```

OpenACC 2.5 API / Directive Equivalence

- acc_init
- acc_shutdown
- acc_set_device_num
- enter data copyin() async
- update device() async

OpenACC 2.5 API / Directive Equivalence

- acc_init
- acc_shutdown
- acc_set_device_num
- enter data copyin() async
- update device() async

```
acc_init( acc_device_nvidia );
```

```
#pragma acc init(nvidia)
```

OpenACC 2.5 API / Directive Equivalence

- acc_init
- acc_shutdown
- acc_set_device_num
- enter data copyin() async
- update device() async

```
acc_set_device_num( 1,
                    acc_device_nvidia );
```

```
#pragma acc set device(nvidia,num:1)
```

OpenACC 2.5 API / Directive Equivalence

- acc_init
- acc_shutdown
- acc_set_device_num
- enter data copyin() async
- update device() async

```
#pragma acc enter data \
                copyin( a[0:n] ) async(1)

bytes = n*sizeof(a[0]);
acc_copyin( a, bytes );
acc_copyin_async( a, bytes, 1 );
```

OpenACC 2.5 API / Directive Equivalence

- acc_init
- acc_shutdown
- acc_set_device_num
- enter data copyin() async
- update device() async

```
#pragma acc update\
                device( a[0:n] ) async(1)

bytes = n*sizeof(a[0]);
acc_update_device( a, bytes );
acc_update_device_async( a, bytes,
                                 1 );
```

New Restrictions

- reduction in orphaned routine gang
- reduction on struct member, array element C++ class member
- acc routine requires one of gang / worker / vector / seq

New Restrictions

- reduction in orphaned routine gang
- reduction on struct member, array element C++ class member
- acc routine requires one of gang / worker / vector / seq

```
#pragma acc routine gang
float test(float* a, float* b, int
n){
  float sum = 0.0;
  #pragma acc loop gang \
                reduction(+:sum)
  for(int i=0; i<n; ++i){
    a[i] = dosomething( b, i, n );
    sum = sum + a[i];
  }
  return sum;
}
```

New Restrictions

- reduction in orphaned routine gang
- reduction on struct member, array element C++ class member
- acc routine requires one of gang / worker / vector / seq

```
class F{
  float s;
  float* data;
  ...
  void count(){
    #pragma acc parallel loop \
        reduction(+:s)
    for(int i=0; i<size(); ++i)
       if(data[i]>0) ++s;
  }
  ...
}
```

New Restrictions

- reduction in orphaned routine gang
- reduction on struct member, array element C++ class member
- acc routine requires one of gang / worker / vector / seq

```
#pragma acc routine seq
float area( float x1, float y1,
            float x2, float y2 ){
    float a =(x2-x1)*(y2-y1);
    if( a < 0 ) a = -a;
    return a;
}
```

Minor Clarifications or Changes

• set default async queue	API routine
• eliminate Fortran header file	use openacc module instead
• fix routine bind()	only applies to call site
• fix loop auto	loop auto implies auto-parallelization
• acc cache(x[i:3])	x[j] must lie between x[i] and x[i+2]
• acc parallel loop private(x)	bind private(x) to loop, not parallel
• acc parallel loop reduction(+:x)	bind reduction() to loop, not parallel
• Latex instead of Word	simpler multi-author revision control

Profile Tools Interface

- Register Runtime Event Callbacks
- Device events
- Data events
- Compute events
- Synchronization Events
- interface

Initialization, Shutdown

Alloc, Free, Upload, Download, Construct

Launch, Construct

Explicit / Implicit Wait

prof_info{devnum, async, srcfile, lineno,...}

event_info{evtype,bytes,ptrs,krnname,...}

api_info{devtype,vendor,API handles,...}

- **S5139 - Showing the Missing Middle: Enabling OpenACC Performance Analysis**
- Guido Juckeland (TU Dresden - ZIH)

OpenACC 3.0 - Deep Copy

```
template<typename T>class vector{
    T *_begin, *_end, *_dataend;
public:
    inline T& operator[]( size_t i ) const { return _begin[i]; }
    inline size_t size(){ return _end-_begin; }
    vector( size_t _sz){ _begin = new T[_sz];
                         _end = _begin+_sz; _dataend = _end; }
    ~myvector(){ delete [] _begin; }...
}

vector<VTYPE> V;
... new V(n); ...
#pragma acc parallel loop copy(V[0:n])
 for( size_t i = 0; i < n; ++i )
   V[i] = Randomize(V[i]);
```

OpenACC Timeline

- 2008 – PGI Accelerator Model (targeting NVIDIA GPUs)
- 2011 – OpenACC 1.0 (targeting NVIDIA GPUs, AMD GPUs)
  - data regions, compute regions, gang/worker/vector
- 2013 – OpenACC 2.0
  - procedures, dynamic data lifetimes
- 2015 – OpenACC 2.5
  - minor fixes, additions
- 2015/16 – OpenACC 3.0
  - deep copy
- More information: www.openacc.org, www.pgroup.com/openacc
- michael.wolfe@pgroup.com