Programming Heterogeneous Parallel Architectures

Intermediate OpenACC

Michael Wolfe

Michael.Wolfe@pgroup.com

http://www.pgroup.com

CEA-INRIA-EDF

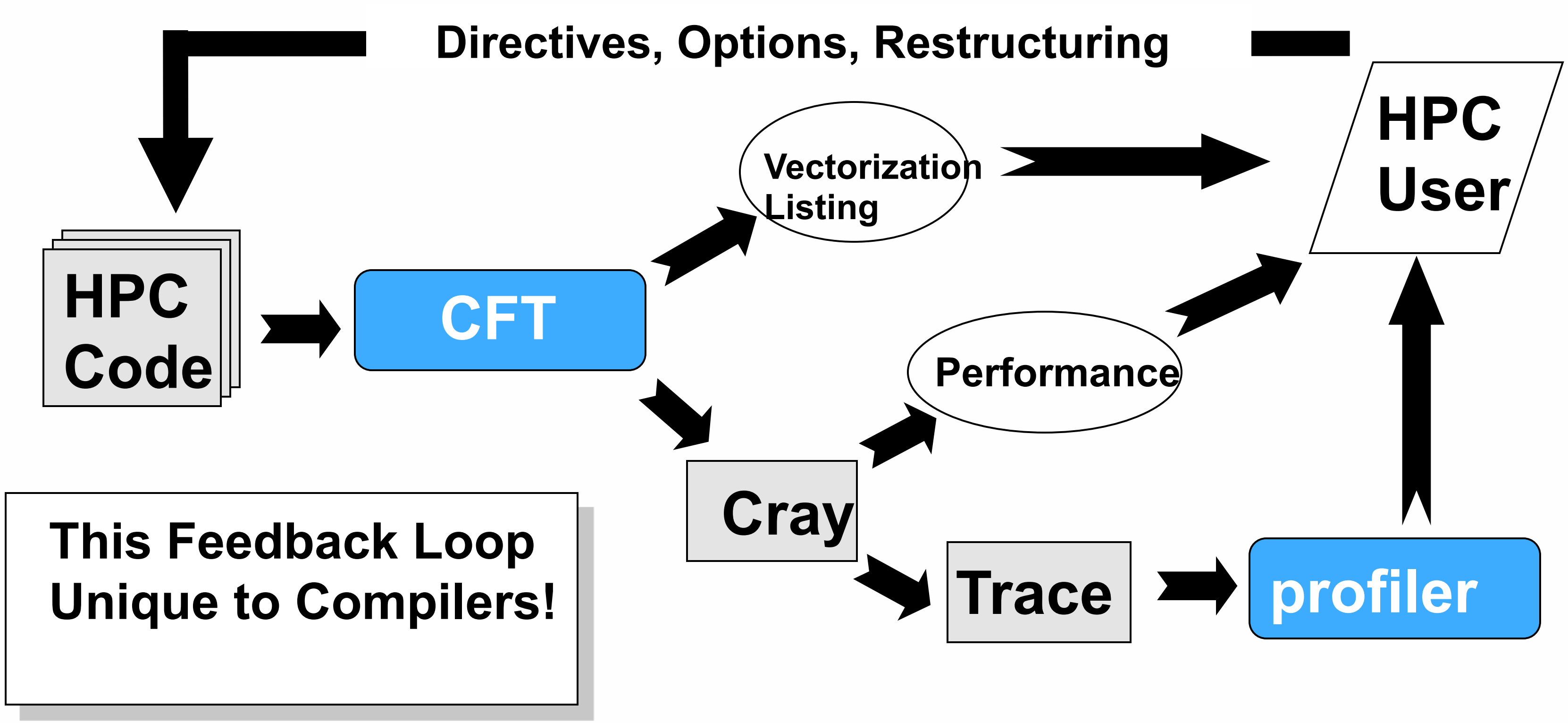
Ecoles d'Eté

2013

Outline

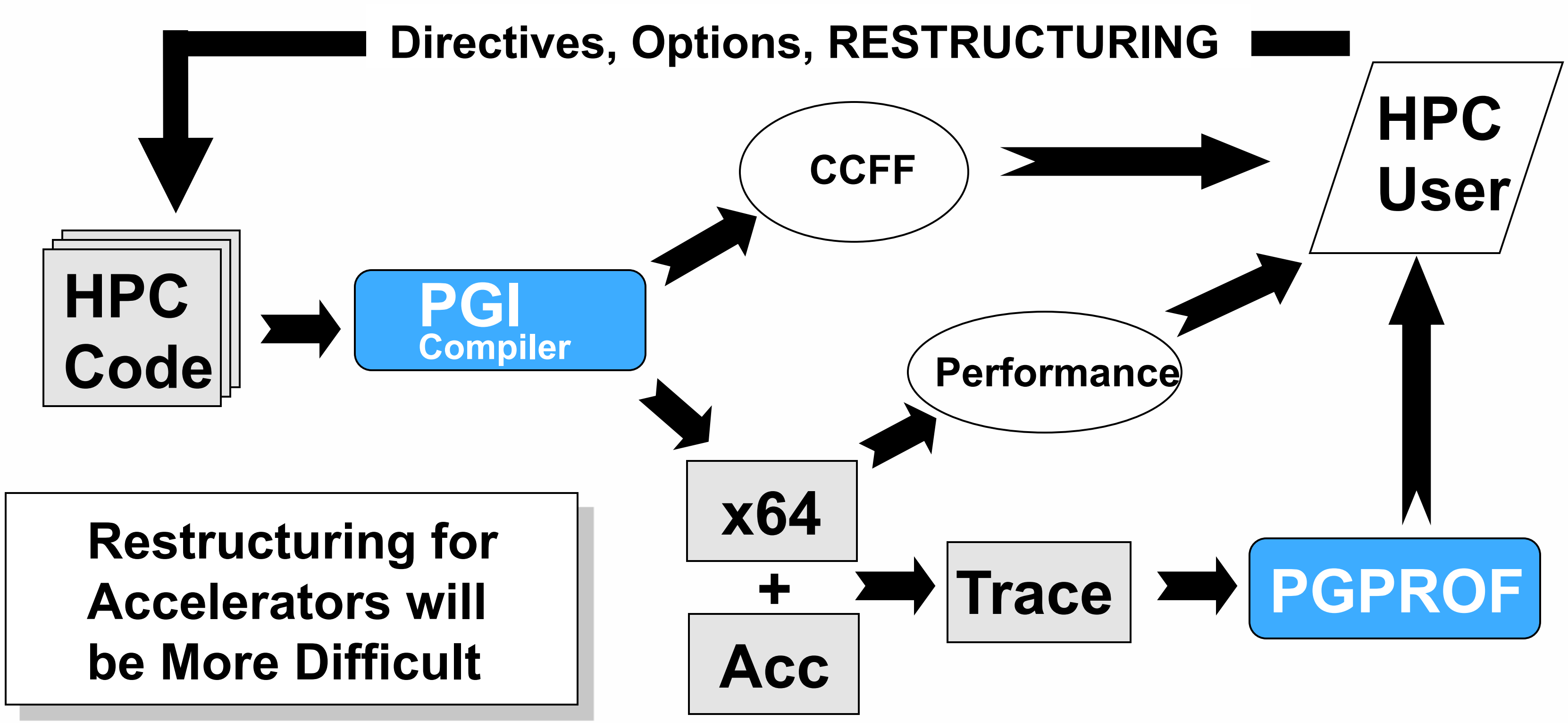
- parallel vs. kernels
- loop schedules
- reductions
- private
- inlining
- data regions
- update directives

How did we make Vectors Work?
Compiler-to-Programmer Feedback – a classic "Virtuous Cycle"



We can use this same methodology to enable effective migration of applications to Multi-core and Accelerators

Compiler-to-Programmer Feedback



Compiler-to-User Feedback

```
% pgfortran -fast -acc -Minfo mm.F90
mm1:
      6, Generating copyout(a(1:m,1:m))
         Generating copyin(c(1:m,1:m))
         Generating copyin(b(1:m,1:m))
      7, Loop is parallelizable
      8, Loop is parallelizable
         Accelerator kernel generated
          7, !$acc do parallel, vector(16)
          8, !$acc do parallel, vector(16)
     11, Loop carried reuse of 'a' prevents parallelization
     12, Loop is parallelizable
         Accelerator kernel generated
          7, !$acc do parallel, vector(16)
         11, !$acc do seq
             Cached references to size [16x16] block of 'b'
             Cached references to size [16x16] block of 'c'
         12, !$acc do parallel, vector(16)
             Using register for 'a'
```

Loop Schedules

```
27, Accelerator kernel generated
    26, !$acc loop gang, vector(16)
    27, !$acc loop gang, vector(16)
```

- **vector loops correspond to `threadidx` indices**
- **gang loops correspond to `blockidx` indices**
- **this schedule has a CUDA schedule**
  <<< dim3(ceil(N/16),ceil(M/16)), dim3(16,16) >>>
- **Compiler strip-mines to protect against very long loop limits**

Loop Directive

- C

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

- Fortran

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

Kernels 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`)

Loop Scheduling Clauses

- Want stride-1 loop to be in 'vector' mode (threadIdx)
  - look at –Minfo messages!
- Want lots of parallelism

Loop Directive Clauses

- Scheduling Clauses
  - `vector` or `vector(n)`
  - `gang` or `gang(n)`
  - `worker` or `worker(n)`
- `independent`
  - use with care, overrides compiler analysis for dependence, private variables
- `private(` *`list`* `)`
  - private data for each iteration of the loop
  - different from local (how?)
- `reduction(` *`red:var`* `)`
  - reduction across the loop

Laplace Equation, Fortran

```
change = tolerance + 1.0
do while(change > tolerance)
  change = 0
  do i = 2, m-1
    do j = 2, n-1
      newa(i,j) = w0*a(i,j) + &
       w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))
      change = max(change,abs(newa(i,j)-a(i,j)))
    enddo
  enddo
  do i = 2, m-1
    do j = 2, n-1
      a(i,j) = newa(i,j)
    enddo
  enddo
enddo
```

Laplace Equation, C

```
do{
  change = 0;
  for( j = 1; j < m-1; ++j ){
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
              w1 * (a[j][i-1] + a[j-1][i] +
                    a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

Laplace Equation, Fortran, OpenMP

```
change = tolerance + 1.0
!$omp parallel shared(change)
do while(change > tolerance)
  change = 0
 !$omp do reduction(max:change) private(i,j)
  do i = 2, m-1
    do j = 2, n-1
      newa(i,j) = w0*a(i,j) + &
       w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))
      change = max(change,abs(newa(i,j)-a(i,j)))
    enddo
  enddo
 !$omp do private(i,j)
  do i = 2, m-1
    do j = 2, n-1
      a(i,j) = newa(i,j)
    enddo
  enddo
enddo
```

Laplace Equation, C, OpenMP

```
#pragma omp parallel shared(change) private(tmp)
do{
  change = 0;
  #pragma omp for private(i,j) reduction(max:change)
  for( j = 1; j < m-1; ++j ){
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
            w1 * (a[j][i-1] + a[j-1][i] +
                  a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

```
change = tolerance + 1.0
!$acc data create(newa(1:m,1:n)) copy(a(1:m,1:n))
do while(change > tolerance)
  change = 0
  !$acc kernels reduction(max:change)
  do i = 2, m-1
    do j = 2, n-1
      newa(i,j) = w0*a(i,j) + &
       w1 * (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))
      change = max(change,abs(newa(i,j)-a(i,j)))
    enddo
  enddo
  do i = 2, m-1
    do j = 2, n-1
      a(i,j) = newa(i,j)
    enddo
  enddo
 !$acc end kernels
enddo
!$acc end data
```

```
#pragma acc data create(newa[0:n][0:m])\
                        copy(a[0:n][0:m])
do{
  change = 0;
  #pragma acc kernels reduction(max:change)
  for( j = 1; j < m-1; ++j ){
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
          w1 * (a[j][i-1] + a[j-1][i] +
                a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

```
#pragma acc data create(newa[0:n][0:m])\
                        copy(a[0:n][0:m])
do{
  change = 0;
  #pragma acc parallel reduction(max:change)
  {
    #pragma acc loop
  for( j = 1; j < m-1; ++j ){
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
          w1 * (a[j][i-1] + a[j-1][i] +
                a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

```
#pragma acc data create(newa[0:n][0:m])\
                 copy(a[0:n][0:m])
do{
  change = 0;
  #pragma acc kernels loop gang(100) reduction(max:change)
  for( j = 1; j < m-1; ++j ){
    #pragma acc loop vector(128) reduction(max:change)
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
          w1 * (a[j][i-1] + a[j-1][i] +
                a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

```
#pragma acc data create(newa[0:n][0:m])\
                  copy(a[0:n][0:m])
do{
  change = 0;
  #pragma acc kernels loop gang(m/2) vector(2) \
     reduction(max:change)
  for( j = 1; j < m-1; ++j ){
    #pragma acc loop vector(128) reduction(max:change)
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
          w1 * (a[j][i-1] + a[j-1][i] +
                a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

```
#pragma acc data create(newa[0:n][0:m])\
                          copy(a[0:n][0:m])
do{
  change = 0;
  #pragma acc parallel loop reduction(max:change)\
    num_gangs(m-2) vector_length(128) gang
  for( j = 1; j < m-1; ++j ){
    #pragma acc loop vector
    for( i = 1; i < n-1; ++i ){
      newa[j][i] = w0*a[j][i] +
          w1 * (a[j][i-1] + a[j-1][i] +
                a[j][i+1] + a[j+1][i]);
      change = fmax(change,fabs(newa[j][i]-a[j][i]));
    }
  }
  tmp = a; a = newa; newa = tmp;
}while( change > tolerance );
```

Building Accelerator Programs

- `pgfortran –acc a.f90`
- `pgcc –acc a.c`
- **Other options:**
  - `-ta=nvidia[,cc1x|cc2x|cc3x]`
    - `default in siterc file:`
      - `set COMPUTECAP=30;`
  - `-ta=nvidia[,cuda5.0]`
    - `default in siterc file:`
      - `set DEFCUDAVERSION=5.0;`
  - `-ta=nvidia,fastmath,nofma`
- **Enable compiler feedback with `–Minfo` or `–Minfo=accel`**

```
#pragma omp parallel num_threads(8)
{
  #pragma omp master
    sum = 0.0;

  #pragma omp for reduction(+:sum)
  for( j = 0; j < m; ++j )
    sum += a[j];

  #pragma omp for
  for( j = 0; j < m; ++j )
    a[j] /= sum;
}
```

```
#pragma acc parallel num_gangs(100)
{
  sum = 0.0;

  #pragma acc loop reduction(+:sum)
  for( j = 0; j < m; ++j )
    sum += a[j];

  #pragma acc loop
  for( j = 0; j < m; ++j )
    a[j] /= sum;
}
```

```
#pragma acc kernels
{
  sum = 0.0;

  #pragma acc loop reduction(+:sum)
  for( j = 0; j < m; ++j )
    sum += a[j];

  #pragma acc loop
  for( j = 0; j < m; ++j )
    a[j] /= sum;
}
```

```
#pragma acc parallel num_gangs(100) num_workers(8)
{
  #pragma acc loop gang private(sum)
  for( j = 0; j < m; ++j ){
    sum = 0.0;

   #pragma acc loop worker reduction(+:sum)
    for( i = 0; i < n; ++i )
      sum += a[j][i];

    #pragma acc loop worker
    for( i = 0; i < m; ++i )
      a[j][i] /= sum;
}
```

C Intrinsics

- C: **#include <accelmath.h>**

acos	asin	atan	atan2
cos	cosh	exp	fabs
fmax	fmin	log	log10
pow	sin	sinh	sqrt
tan	tanh		
acosf	asinf	atanf	atan2f
cosf	coshf	expf	fabsf
fmaxf	fminf	logf	log10f
powf	sinf	sinhf	sqrtf
tanf	tanhf		

Fortran Intrinsics

abs	acos	aint	asin
atan	atan2	cos	cosh
dble	exp	iand	ieor
int	ior	log	log10
max	min	mod	not
real	sign	sin	sinh
sqrt	tan	tanh	

other functions

- **libm routines**
  - `use libm`
  - `#include <accelmath.h>`
- **device builtin routines**
  - `use cudadevice`
  - `#include <cudadevice.h>`

Obstacles to Parallelization

- Computed Index (linearization, look-up)
- While Loops
- Triangular Loops
- “live-out” Variables
- Privatization of Local Arrays
- Function calls
- Device Runtime Errors
- Compiler Errors

Computed Index – Linearization

```
!$acc kernels
  do i = 1, M
    do j = 1, N
      idx = ((i-1)*M)+j
      A(idx) = B(i,j)
    enddo
  enddo
!$acc end kernels
```

```
 pgf90 linearization.f90 -ta=nvidia -Minfo=accel
linear:
    16, No parallel kernels found, accelerator region ignored
    17, Complex loop carried dependence of 'a' prevents parallelization
    18, Complex loop carried dependence of 'a' prevents parallelization
        Parallelization would require privatization of array 'a(:)
```

To fix,
Remove the linearization or Use the "independent" clause

```
!$acc kernels
  do i = 1, M
    do j = 1, N
      A(i,j) = B(i,j)
    enddo
  enddo
!$acc end kernels
```

```
!$acc kernels
!$acc loop independent
  do i = 1, M
    do j = 1, N
      idx = ((i-1)*M)+j
      A(idx) = B(i,j)
    enddo
  enddo
!$acc end kernels
```

Computed Index - Look-up

```
!$acc kernels
  do i = 1, M
    idx = lookup(i)
    do j = 1, N
      A(idx,j) = ((i-1)*M)+j
    enddo
  enddo
!$acc end kernels
```

% pgf90 lookup.f90 -ta=nvidia -Minfo=accel
lookup_test:
    16, Generating copyout(a(:,1:1024))
    17, Parallelization would require privatization of array 'a(:,1:1024)'
        Sequential loop scheduled on host
    19, Loop is parallelizable
        Accelerator kernel generated
        19, !$acc do parallel, vector(256)

```
!$acc kernels
  do i = 1, M
    do j = 1, N
      idx = lookup(j)
      A(i,idx) = ((i-1)*M)+j
    enddo
  enddo
!$acc end kernels
```

% pgf90 lookup1.f90 -ta=nvidia -Minfo=accel
lookup_test:
    16, Generating copyout(a(1:1024,:))
        Generating copyin(cell(1:1024))
    17, Loop is parallelizable
        Accelerator kernel generated
        17, !$acc do parallel, vector(256)
    18, Loop carried reuse of 'a' prevents parallelization
        Inner sequential loop scheduled on accelerator

The Independent or parallel clauses could be used to force parallelization but is not recommended

While Loops

```
!$acc kernels
  i = 0
  do, while (.not.found)
    i = i + 1
    if (A(i) .eq. 102) then
      found = i
    endif
  enddo
!$acc end kernels
```

```
% pgf90 -ta=nvidia -Minfo=accel while.f90
while1:
     17, Accelerator region ignored
     19, Accelerator restriction: invalid loop
```

Change to a rectangular loop

```
!$acc kernels
  do i = 1, N
    if (A(i) .eq. 102) then
      found(i) = i
    else
      found(i) = 0
    endif
  enddo
!$acc end kernels
print *, 'Found at ', maxval(found)
```

```
% pgf90 -ta=nvidia -Minfo=accel while2.f90
while2:
     18, Generating copyin(a(1:1024))
         Generating copyout(found(1:1024))
         Generating compute capability 1.0 binary
         Generating compute capability 1.3 binary
     19, Loop is parallelizable
         Accelerator kernel generated
         19, !$acc do parallel, vector(256)
             Using register for 'found'
```

Triangular Loops

```
!$acc kernels copyout(A)
  do i = 1, M
     do j = i, N
        A(i,j) = i+j
     enddo
  enddo
!$acc end kernels
```

All loop schedules must be rectangular. For triangular loops, the compiler will either serialize the inner loop or make the inner loop rectangular and add an implicit if statement to skip the lower part of the triangle.

Problem: The compiler will copy out the entire array A. The lower triangle contains garbage since it was not initialized. Use “copy(A)” to initialize the values.

"live-out" Variables

```
!$acc kernels
  do i = 1, M
    do j = 1, N
      idx = i+j
      A(i,j) = idx
    enddo
  enddo
!$acc end kernels
print *, idx, A(1,1), A(M,N)
```

```
% pgf90 -ta=nvidia,time -Minfo=accel  liveout.f90
liveout:
    11, Generating copyout(a(1:1024,1:1024))
    12, Loop is parallelizable
        Accelerator kernel generated
        12, !$acc do parallel, vector(256)
    13, Inner sequential loop scheduled on accelerator
    14, Accelerator restriction: induction variable live-out from loop: idx
    15, Accelerator restriction: induction variable live-out from loop: idx
```

Privatize the scalar

```
!$acc kernels
  do i = 1, M
!$acc loop private(idx)
    do j = 1, N
      idx = i+j
      A(i,j) = idx
    enddo
  enddo
!$acc end kernels
  print *, idx, A(1,1), A(M,N)
```

```
% pgf90 -ta=nvidia,time -Minfo=accel liveout2.f90
liveout2:
    10, Generating copyout(a(1:1024,1:1024))
    11, Loop is parallelizable
    13, Loop is parallelizable
        Accelerator kernel generated
        11, !$acc do parallel, vector(16)
        13, !$acc do parallel, vector(16)
```

Privatization of Local Arrays

```
!$acc kernels
  do i = 1, M
    do j = 1, N
      do jj = 1, 10
        tmp(jj) = jj
      end do
      A(i,j) = sum(tmp)
    enddo
  enddo
!$acc end kernels
```

```
% pgf90 -ta=nvidia -Minfo=accel  private.f90
privatearr:
    10, Generating copyout(tmp(1:10))
        Generating compute capability 1.0 binary
        Generating compute capability 1.3 binary
    11, Parallelization would require privatization of array 'tmp(1:10)'
    13, Parallelization would require privatization of array 'tmp(1:10)'
        Sequential loop scheduled on host
    14, Loop is parallelizable
        Accelerator kernel generated
        14, !$acc do parallel, vector(10)
    17, Loop is parallelizable
        Accelerator kernel generated
        17, !$acc do parallel, vector(10)
            Sum reduction generated for tmp$r
```

Privatization of Local Arrays - cont.

```
!$acc kernels
  do i = 1, M
!$acc loop private(tmp)
    do j = 1, N
      do jj = 1, 10
        tmp(jj) = jj
      end do
      A(i,j) = sum(tmp)
    enddo
  enddo
!$acc end kernels
```

```
% pgf90 -ta=nvidia,time -Minfo=accel private2.f90
privatearr2:
    10, Generating copyout(a(1:1024,1:1024))
        Generating compute capability 1.0 binary
        Generating compute capability 1.3 binary
    11, Loop is parallelizable
    13, Loop is vectorizable
        Accelerator kernel generated
        11, !$acc do parallel, vector(16)
        13, !$acc do vector(16)
    14, Loop is parallelizable
    17, Loop is parallelizable
```

Need to privatize local temporary arrays.
Default is to assume that they are shared.

Function Calls

- **Function calls are not allowed within a compute region.**
- **Restriction is due to lack of a device linker and hardware support.**
- **Functions must be inlined, either manually or by the compiler with –Minline or –Mipa=inline.**

Device Errors

call to cuMemcpyDtoH returned error 700: Launch failed

```
   !$acc kernels
   do i = 1, M
     do j = 1, N
       A(i,j) = B(i,j+1) << out-of-bounds
     enddo
   enddo
 !$acc end kernels
```

- Typically occurs when the device kernel gets an execution error, such as a out-of-bounds or other memory access violation.

call to cuMemcpy2D returned error 1: Invalid value

```
  parameter(N=1024,M=512)
  real :: A(M,N), B(M,N)
...
!$acc kernels copyout(A), copyin(B(0:N,1:M+1))  <<< Bad bounds for the copyin
  do i = 1, M
    do j = 1, N
      A(i,j) = B(i,j+1)
    enddo
  enddo
!$acc end kernels
```

- Occurs if there is an error when copying data to/from the device

Data Attributes

- predetermined data attributes
  - loop variables are private
- implicit data attributes
  - array, struct – `present_or_copy`
  - scalar – `firstprivate`
- explicit data attributes
  - in a data clause

Data Clauses

- **C**

```
#pragma acc data copyin(a[0:n]) copyout(r[0:n])
 {
      ....
 }
```

- **Fortran**

```
!$acc data copyin(a(1:n)) copyout(r(1:n))
      ...
!$acc end data
```

Data Clauses

- **C**

```
#pragma acc data copyin(a[0:n][0:m]) \
                 copy(r[0:n][0:m])
 {
      ...
 }
```

- **Fortran**

```
!$acc data copyin(a(1:m,1:n)) copy(r(:,:))
      ...
 !$acc end data
```

Data Clauses

- C

```
#pragma acc data copyin(a[0:n][0:m]) \
                create(r[0:n][0:m])
{
    ...
}
```

- Fortran

```
!$acc data copyin(a(1:m,1:n)) create(r)
    ...
!$acc end data
```

Data Clauses

- **C**

```
#pragma acc data copyin(a[0:n][0:m]) \
                 create(r[1:n-1][1:m-1])
 {
     ...
 }
```

- **Fortran**

```
!$acc data copyin(a(1:m,1:n)) &
        create(r(2:m-1,2:n-1))
     ...
 !$acc end data
```

Data Clauses

- C

```
#pragma acc data copyin(a[0:n][0:m])
  /* data copied to Accelerator here */
 {
     ...
 } /* data copied to Host here */
```

- Fortran

```
!$acc data copyin(a(1:m,1:n))
  ! data copied to Accelerator here
     ...
  ! data copied to Host here
 !$acc end data
```

Fortran Array Sections

- `a(1:n,1:m)` … `copy(a(1:n,1:m))`
  - allocates full array (1:n,1:m)
  - copies full array (1:n,1:m)
- `a(1:n,1:m)`… contiguous subarray `copy(a(2:n-1,1:m))`
  - allocates subarray (2:n-1,1:m)
  - copies subarray (2:n-1,1:m)
  - `a(1:n,1:m)` is NOT present, but subarray IS present
- `a(1:n,1:m)`… noncontiguous `copy(a(2:n-1,2:m-1))`
  - allocates contiguous bounding subarray (1:n,2:m-1)
  - copies subarray (2:n-1,2:m-1)
  - `a(1:n,2:m-1)` IS present, but may not be up to date
  - data copies may take longer

C Array Sections

- `float r[100][200];  copy(r[0:100][0:200])`
  - allocates rectangular array [100][200]
  - copies subarray [0:100][0:200]
- `float r[100][200];  copy(r[1:n][0:200])`
  - allocates subarray [1:n][200] ([n-1][200])
  - copies subarray [1:n][0:200]
- `float r[100][200];  copy(r[1:n][1:m])`
  - allocates bounding subarray [1:n][200]
  - copies subarray [1:n][1:m]

C Array Sections

- `typedef float row[200];`
- `row* r;  copy(r[0:100][0:200])`
  - allocates vector of pointers [100]
  - allocates rectangular array [100][200]
  - fills in pointers [100]
  - copies subarray [0:100][0:200]
- `row* r;  copy(r[1:n][0:200])`
  - allocates vector of pointers [n-1]
  - allocates rectangular subarray [1:n][200]
  - fills in pointers [1:n]
  - copies subarray [1:n][0:200]
- `row* r;  copy(r[1:n][1:m])`
  - allocates vector of pointers [n-1]
  - allocates rectangular subarray [1:n][0:m]
  - fills in pointers [1:n]
  - copies subarray [1:n][1:m]

C Array Sections

- `float *r[100];  copy(r[0:100][0:200])`
  - allocates vector of pointers [100]
  - allocates rectangular array [100][200]
  - fills in pointers [100]
  - copies subarray [0:100][0:200]
- `float *r[100];  copy(r[1:n][0:200])`
  - allocates vector of pointers [n-1]
  - allocates rectangular subarray [1:n][200]
  - fills in pointers [1:n]
  - copies subarray [1:n][0:200]
- `float *r[100];  copy(r[1:n][1:m])`
  - allocates vector of pointers [n-1]
  - allocates rectangular subarray [1:n][0:m]
  - fills in pointers [1:n]
  - copies subarray [1:n][1:m]

C Array Sections

- `float **r;  copy(r[0:100][0:200])`
  - allocates vector of pointers [100]
  - allocates rectangular array [100][200]
  - fills in pointers [100]
  - copies subarray [0:100][0:200]
- `float **r; copy(r[1:n][0:200])`
  - allocates vector of pointers [n-1]
  - allocates rectangular subarray [1:n][200]
  - fills in pointers [1:n]
  - copies subarray [1:n][0:200]
- `float **r; copy(r[1:n][1:m])`
  - allocates vector of pointers [n-1]
  - allocates rectangular subarray [1:n][0:m]
  - fills in pointers [1:n]
  - copies subarray [1:n][1:m]

Data Clauses

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

Data Region

- C

```
#pragma acc data data-clauses if( condition )
{
   ....
}
```

- Fortran

```
!$acc data data-clauses if( condition )
....
!$acc end data
```

- May be nested and may contain compute regions
- May not be nested within a compute region
- May contain procedure calls

Update Directive

- **C**

```
#pragma acc update host( list )
#pragma acc update device( list )
```

- **Fortran**

```
!$acc update host( list )
!$acc update device( list )
```

  - **data must be in a data clause for an enclosing data region**
  - **may be noncontiguous**
  - **implies `present( list )`**
  - **both may be on a single line**
    - `update host( list ) device( list )`

Asynchronous Update Directive

- C

```
#pragma acc update host( list ) async(1)
#pragma acc update device( list ) async(2)
```

- Fortran

```
!$acc update host( list ) async(3)
!$acc update device( list ) async(n)
```

  - async value should be >= 0
  - mapped down to some number of actual async queues
  - updates with same value will execute in program order
  - async with no value is same as `async(acc_async_noval)`
  - `async(acc_async_sync)` is same as no async clause
  - host program may continue

Wait Directive

- C

```
#pragma acc wait( 2 )
#pragma acc wait
```

- Fortran

```
!$acc wait( n, n-1 )
!$acc wait
```

  - wait with no values waits for ALL asynchronous queues
  - wait with multiple values waits for each queue
  - NO implied wait at end of data construct
    - async updates in a data region require wait

Data Regions Across Procedures

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

subroutine bus(x, y)
 real :: x(:), y(:)
 !$acc data copy(x)
 call sub( x, y )
 !$acc end data
```

Data Regions Across Procedures

```
void sub( float* a, float* b, int n ){
  int i;
  #pragma acc kernels copyin(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 );
  }
...
```

Compiler Feedback Messages

- Data related
  - `Generating copyin(b(1:n,1:m))`
  - `Generating copyout(b(2:n-1,2:m-1))`
  - `Generating copy(a(1:n,1:n))`
  - `Generating create(c(1:n,1:n))`
- Loop or kernel related
  - `Loop is parallelizable`
  - `Accelerator kernel generated`
- Barriers to GPU code generation
  - `No parallel kernels found, accelerator region ignored`
  - `Loop carried dependence due to exposed use of ... prevents parallelization`
  - `Parallelization would require privatization of array ...`

Compiler Messages Continued

- **Memory optimization related**
  - **`Cached references to size [(x+2)x(y+2)] block of ‘b’`**
  - **`Non-stride-1 memory accesses for ‘a’`**

Selecting Device

- C

```
#include <openacc.h>
  n = acc_get_num_devices(acc_device_nvidia);
  ...
  acc_set_device_num( 1, acc_device_nvidia);
```

- Fortran

```
use openacc
n = acc_get_num_devices( acc_device_nvidia )
...
call acc_set_device_num( 0, acc_device_nvidia)
```

- Environment Variable

```
setenv ACC_DEVICE_NUM 1
export ACC_DEVICE_NUM=1
```

Directives Summary

- **acc kernels**
  - **if(*condition*)**
  - **copy(*list*) copyin(*list*) copyout(*list*) create(*list*)**
- **acc parallel**
  - **if(*condition*)**
  - **num_gangs(*n*) num_workers(*n*) vector_length(*n*)**
- **acc data**
  - **copy(*list*) copyin(*list*) copyout(*list*) create(*list*)**
  - **if(*condition*)**
- **acc update device(*list*) host(*list*)**
- **acc loop**
  - **gang(*width*) worker(*width*) vector(*width*)**
  - **independent**
  - **private(*list*)**
  - **reduction(*red:var*)**

Runtime Summary

- `acc_get_num_devices( acc_device_nvidia )`
- `acc_set_device_num( n, acc_device_nvidia )`
- `acc_set_device_type( acc_device_nvidia | acc_device_host )`
- `acc_get_device_type()`
- `acc_init( acc_device_nvidia | acc_device_host )`

Environment Variable Summary

- ACC_DEVICE_NUM
- ACC_DEVICE_TYPE
- PGI_ACC_NOTIFY
- PGI_ACC_TIME

Command Line Summary

- `-ta=nvidia`
- `-ta=nvidia,nofma`
- `-ta=nvidia,fastmath`
- `-ta=nvidia,[cc1x|cc2x|cc3x]   (multiple allowed)`
- `-ta=nvidia,maxregcount:n`
- `-ta=nvidia,cuda5.0`

Accelerator Programming Timeline

- late 1990s – GPGPU Programming
- early 2000s – Brook project (and others)
- 2007 – NVIDIA CUDA release, SC07 tutorial
  - HiCUDA, HMPP (and others); EXOCHI: PLDI 2007 (Intel)
- 2008 – PGI Accelerator Programming Model
  - Larrabee paper at SIGGRAPH (Intel)
- 2009 – OpenMP BoF at SC09
  - Prog. Model for Heterogeneous x86: PLDI 2009 (Intel); Rattner/SC09
- 2010 – OpenMP Accelerator committee
- 2011 – OpenACC API 1.0
  - Language Extensions for Offload (LEO, Intel)
- 2013 – OpenACC API 2.0
  - OpenMP 4.0 target directives

Where to get help

- **PGI Customer Support - trs@pgroup.com**
- **PGI User's Forum - http://www.pgroup.com/userforum/index.php**
- **PGI Articles - http://www.pgroup.com/resources/articles.htm http://www.pgroup.com/resources/accel.htm**
- **PGI User's Guide - http://www.pgroup.com/doc/pgiug.pdf**
- **CUDA Fortran Reference Guide - http://www.pgroup.com/doc/pgicudafortug.pdf**

Copyright Notice