GPU TECHNOLOGY CONFERENCE

NVIDIA

SCALING OPENACC APPLICATIONS ACROSS MULTIPLE GPUS

Michael Wolfe, PGI Compilers & Tools

OpenACC Directives

```
void spmv(float *a, float *v, float *x,
          int nrows, int* rowndx, int* colndx){


#pragma acc parallel loop vector_length(128)
 for(int i=0; i<nrows; ++i ){
   float val=0.0f;
   int n1=rowndx[i], n2=rowndx[i+1];
  #pragma acc loop vector reduction(+:val)
   for(int n=n1; n<n2; ++n)
      val += m[n]*v[colndx[n]];
   x[i] = val;
 }
}
```

OpenACC Directives

```
subroutine spmv(a, v, x, nrows, rowndx, colndx)
 real, dimension(:) :: a, v, x
 integer :: nrows
 integer, dimension(:) :: rowndx, colndx
!$acc parallel loop vector_length(128)
 do i = 1, nrows
   val = 0.0
   n1 = rowndx(i);  n2 = rowndx(i+1)-1
  !$acc loop vector reduction(+:val)
   do n = n1, n2
      val = val + m(n)*v(colndx(n))
   enddo
   x(i) = val
 enddo
end subroutine
```

Multiple GPUs

MPI and OpenACC

- Benefits of using multiple GPUs
- Have each MPI rank use a different GPU (up to #GPUs)
- Each MPI rank has its own context (even when sharing GPU)

MPI and OpenACC

```
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
...
ngpus = acc_get_num_devices( acc_device_nvidia );
if( ngpus ){
    gpunum = rank % ngpus;
    acc_set_device_num( gpunum, acc_device_nvidia );
}else{
    /* no NVIDIA GPUs available */
    acc_set_device_type( acc_device_host );
}
```

MPI and OpenACC

- Benefits of using multiple GPUs
- Have each MPI rank use a different GPU (up to #GPUs)
- Each MPI rank has its own context (even when sharing GPU)
  - Device memory is physical – not paged, not shared
  - Can run out of memory with multiple ranks/GPU

MPI and OpenACC

- Why use multiple GPUs?
- Have each MPI rank use a different GPU (up to #GPUs)
- Each MPI rank has its own context (even when sharing GPU)
  - Device memory is physical – not paged, not shared
  - Can run out of memory with multiple ranks/GPU
- Sharing memory with MPI

```
cudaMalloc( &ptr, size );
MPI_Send( &ptr, ... );  MPI_Recv( &ptr, ... );
acc_map_data( hostptr, ptr, size );
```

OpenMP and OpenACC

- Benefits of using multiple GPUs
- Have each OpenMP thread use a different GPU
  - acc_set_device_num
  - thread/GPU assignment persists across OpenMP parallel regions
- All OpenMP threads share a single context on shared GPU

OpenMP and OpenACC

```
tnum = omp_get_thread_num();
...
ngpus = acc_get_num_devices( acc_device_nvidia );
if( ngpus ){
    gpunum = tnum % ngpus;
    acc_set_device_num( gpunum, acc_device_nvidia );
}else{
    /* no NVIDIA GPUs available */
    acc_set_device_type( acc_device_host );
}
```

OpenMP and OpenACC

- Benefits of using multiple GPUs
- Have each OpenMP thread use a different GPU
- All OpenMP threads share a single context on shared GPU
  - data is shared across threads
  - present_or_copy (pcopy) is your friend
  - disjoint sections of arrays will not be contiguous on device
  - overlapping sections of arrays may fail

OpenACC in OpenMP threads

```
numgpus = acc_get_num_devices( acc_device_nvidia );
#pragma omp parallel num_threads(numgpus)
{

    gpunum = omp_get_thread_num();
    acc_set_device_num( gpunum, acc_device_nvidia );

    #pragma acc data copy( x[0:n] )
    {
        // thread tnum copies x to GPU gpunum
    }
}
```

Data shared across OpenMP threads

```
numgpus = acc_get_num_devices( acc_device_nvidia ); /*2*/
#pragma omp parallel num_threads(numcores) /*4*/
{
    tnum = omp_get_thread_num();
    gpunum = tnum % numgpus;
    acc_set_device_num( gpunum, acc_device_nvidia );

    #pragma acc data copy( x[0:n] )
    {
        // thread 0 and 2 both copy x to GPU 0

    }
}
```

Present_or_copy is your friend

```
numgpus = acc_get_num_devices( acc_device_nvidia ); /*2*/
#pragma omp parallel num_threads(numcores) /*4*/
{
    tnum = omp_get_thread_num();
    gpunum = tnum % numgpus;
    acc_set_device_num( gpunum, acc_device_nvidia );

    #pragma acc data present_or_copy( x[0:n] )
    {   // first thread of (0,2) copies x to GPU 0

    }   // last thread of (0,2) copies x from GPU 0
}
```

Present_or_copy is your friend

```
numgpus = acc_get_num_devices( acc_device_nvidia ); /*2*/
#pragma omp parallel num_threads(numcores) /*4*/
{
    tnum = omp_get_thread_num();
    tpergpu = omp_get_num_threads() / numgpus;
    gpunum = tnum / tpergpu;
    acc_set_device_num( gpunum, acc_device_nvidia );

    #pragma acc data present_or_copy( x[0:n] )
    {   // first thread of (0,1) copies x to GPU 0

    }   // last thread of (0,1) copies x from GPU 0
}
```

Disjoint sections across OpenMP threads

```
numgpus = acc_get_num_devices( acc_device_nvidia ); /*2*/
#pragma omp parallel num_threads(numcores) /*4*/
{   ...
    sz = n/numcores;
    low = tnum * sz;

    #pragma acc data present_or_copy( x[low:sz] )
    {   // thread 0 copies x[0:sz] to GPU 0
        // thread 1 copies x[sz:sz] to GPU 0
        // These are NOT contiguous!
        #pragma acc parallel loop
        for( i = 0; i < sz*2; ++i ) x[i] = ...;
    }
}
```

Overlapping sections across threads

```
numgpus = acc_get_num_devices( acc_device_nvidia ); /*2*/
#pragma omp parallel num_threads(numcores) /*4*/
{   ...
    sz = n/numcores;
    low = tnum * sz;

    #pragma acc data present_or_copy( x[low:sz+1] )
    {   // thread 0 copies x[0:sz+1] to GPU 0
        // thread 1 copies x[sz:sz+1] to GPU 0
        // These overlap!
        // Second reaching thread will fail!
    }
}
```

Multiple GPUs in a Single Thread

- Benefits of using multiple GPUs
- Dynamically switch between GPUs
  - acc_set_device_num
  - async() operations on each GPU
- Data not shared across GPUs

Multiple GPUs in a Single Thread

```
ngpus = acc_get_num_devices( acc_device_nvidia );

for( gpunum = 0; gpunum < ngpus; ++gpunum ){
   acc_set_device_num( gpunum, acc_device_nvidia );
   #pragma acc enter data create(...) // no async in OpenACC
   #pragma acc update device(...) async(gpunum)
   #pragma acc parallel async(gpunum)...
   {...
   }
   #pragma acc update host(...) async(gpunum)
   #pragma acc exit data delete(...) // no async in OpenACC
}
```

Multiple GPUs in a Single Thread

```
ngpus = acc_get_num_devices( acc_device_nvidia );

for( gpunum = 0; gpunum < ngpus; ++gpunum ){
   acc_set_device_num( gpunum, acc_device_nvidia );
   #pragma acc enter data create(...) async(gpunum) //PGI
   #pragma acc update device(...) async(gpunum)
   #pragma acc parallel async(gpunum)...
   {...
   }
   #pragma acc update host(...) async(gpunum)
   #pragma acc exit data delete(...) async(gpunum) //PGI
}
```

Multiple GPUs in a Single Thread

```
ngpus = acc_get_num_devices( acc_device_nvidia );

for( gpunum = 0; gpunum < ngpus; ++gpunum ){
   acc_set_device_num( gpunum, acc_device_nvidia );
   #pragma acc data copyin(...) \
                    copyout(...) async(gpunum) // PGI
   {
     #pragma acc parallel async(gpunum)...
     {...
     }

   }
}
```

It's All About the Data

- MPI + OpenACC
  - Each MPI rank using a different GPU
  - if ranks > GPUs, oversubscribe some GPUs
- OpenMP + OpenACC
  - Each OpenMP thread using a different GPU
  - if threads > GPUs, shared data across threads on some GPUs
- Dynamic GPU selection
  - challenging, but possible, using async operations
- CUDA_VISIBLE_DEVICES=0,2

OpenACC and OpenMP on one loop?

```
#pragma omp parallel for
  #pragma acc parallel loop
    for( i = 1; i < n; ++i )
       x[i] = ...
```