GPU Teaching Kit

Accelerated Computing

# Lecture 21.2 - Related Programming Models: OpenACC

## OpenACC Subtleties

# Objective

- To understand some important and sometimes subtle details in OpenACC programming
  - parallel loops
  - simple examples to illustrate basic concepts and functionalities

# Parallel vs. Loop Constructs

```
#pragma acc parallel loop copyin(M[0:Mh*Mw])
copyin(N[0:Mw*Nw]) copyout(P[0:Mh*Nw])
for (int i=0; i<Mh; i++) {
…
}
```

is equivalent to:

```
#pragma acc parallel copyin(M[0:Mh*Mw]) copyin(N[0:Mw*Nw])
copyout(P[0:Mh*Nw])
{
    #pragma acc loop
    for (int i=0; i<Mh; i++) {
        …
    }
}
```

(a parallel region that consists of a single loop)

# More on Parallel Construct

```
#pragma acc parallel copyout(a) num_gangs(1024) num_workers(32)
{
    a = 23;
}
```

1024*32 workers will be created. a=23 will be executed redundantly by all 1024 gang leads

- A parallel construct is executed on an accelerator
- One can specify the number of gangs and number of workers in each gang
  - Equivalent to CUDA blocks and threads

# What Does Each “Gang Loop” Do?

```
#pragma acc parallel num_gangs(1024)
{
    for (int i=0; i<2048; i++) {
        …
    }
}
```

```
#pragma acc parallel num_gangs(1024)
{
#pragma acc loop gang
    for (int i=0; i<2048; i++) {
        …
    }
}
```

# Worker Loop

```
#pragma acc parallel num_gangs(1024) num_workers(32)
{
    #pragma acc loop gang
    for (int i=0; i<2048; i++) {
        #pragma acc loop worker
        for (int j=0; j<512; j++) {
            foo(i,j);
        }
    }
}
```

1024*32=32K workers will be created, each executing 1M/32K = 32 instance of foo()

# A More Substantial Example

- Statements 1, 3, 5, 6 are redundantly executed by 32 gangs

```
#pragma acc parallel num_gangs(32)
{
    Statement 1;
    #pragma acc loop gang
    for (int i=0; i<n; i++) {
        Statement 2;
    }
    Statement 3;
    #pragma acc loop gang
    for (int i=0; i<m; i++) {
        Statement 4;
    }
    Statement 5;
    if (condition) Statement 6;
}
```

# A More Substantial Example

- The iterations of the n and m for-loop iterations are distributed to 32 gangs
- Each gang could further distribute the iterations to its workers
  - The number of workers in each gang will be determined by the compiler/runtime

```
#pragma acc parallel num_gangs(32)
{
    Statement 1;
    #pragma acc loop gang
    for (int i=0; i<n; i++) {
        Statement 2;
    }
    Statement 3;
    #pragma acc loop gang
    for (int i=0; i<m; i++) {
        Statement 4;
    }
    Statement 5;
    if (condition) Statement 6;
}
```

# Avoiding Redundant Execution

- Statements 1, 3, 5, 6 will be executed only once
- Iterations of the n and m loops will be distributed to 32 workers

```
#pragma acc parallel
num_gangs(1) num_workers(32)
{
    Statement 1;
    #pragma acc loop worker
    for (int i=0; i<n; i++) {
        Statement 2;
    }
    Statement 3;
    #pragma acc loop worker
    for (int i=0; i<m; i++) {
        Statement 4;
    }
  Statement 5;
   if (condition)  Statement 6;
}
```

# Kernel Regions

- Kernel constructs are descriptive of programmer intentions
  - The compiler has a lot of flexibility in its use of the information
- This is in contrast with Parallel, which is prescriptive of the action for the compile follow

```
#pragma acc kernels
{
    #pragma acc loop gang(1024)
    for (int i=0; i<2048; i++) {
        a[i] = b[i];
    }
    #pragma acc loop gang(512)
    for (int j=0; j<2048; j++) {
        c[j] = a[j]*2;
    }
    for (int k=0; k<2048; k++) {
        d[k] = c[k];
    }
}
```

# Kernel Regions

- Code in a kernel region can be broken into multiple CUDA/OpenCL kernels
- The i, j, k loops can each become a kernel
  - The k-loop may even remain as host code
- Each kernel can have a different gang/worker configuration

```
#pragma acc kernels
{
    #pragma acc loop gang(1024)
    for (int i=0; i<2048; i++) {
        a[i] = b[i];
    }
    #pragma acc loop gang(512)
    for (int j=0; j<2048; j++) {
        c[j] = a[j]*2;
    }
    for (int k=0; k<2048; k++) {
        d[k] = c[k];
    }
}
```

GPU Teaching Kit