
# KFUPM HPC Workshop

**April 29-30 2015**


## Mohamed Mekias

**HPC Solutions Consultant**


**Introduction to OpenAcc**

# What is OpenACC?

- **A set of directive-based extensions to C, C++ and Fortran that allow you to annotate regions of code and data for offloading from a CPU host to an attached Accelerator**

- **Borrowed the ease of use of OpenMP directive programming**
- **For detailed OpenACC standard visit:**
  - **www.openacc-standard.org**

# OpenAcc directives

- **#pragma acc  directive-name [clause [,clause]…] new-line**
  - **#pragma acc parallel [clause [,clause]…] new-line**
    **{structured block}**
  - **When encountered this directive causes a gang of worker threads are created to execute on the accelerator**
  - **One worker in each gang begins executing the code following the structured block**
  - **Number of gangs/workers remains constant in parallel region**

# OpenAcc directives

- **The clauses for #pragma acc parallel:**
  - **if( condition)**
  - **async [(scalar-integer-expression)]**
  - **num_gangs (scalar-integer-expression)**
  - **num_workers (scalar-integer-expression)**
  - **vector_length (scalar-integer-expression)**
  - **reduction (operator:list)**
  - **copy (list)**
  - **copyout (list)**
  - **create (list)**
  - **private (list)**
  - **firstprivate (list)**
  - **present (list)**
  - **present_or_copy (list)**
  - **present_or_copyin (list)**
  - **present_or_copyout (list)**
  - **present_or_create (list)**
  - **deviceprt (list)**

# OpenAcc directives

- **The clauses for #pragma acc parallel loop:**
  - **if( condition)**
  - **async [(scalar-integer-expression)]**
  - **num_gangs (scalar-integer-expression)**
  - **num_workers (scalar-integer-expression)**
  - **vector_length (scalar-integer-expression)**
  - **reduction (operator:list)**
  - **copy (list)**
  - **copyout (list)**
  - **create (list)**
  - **private (list)**
  - **firstprivate (list)**
  - **present (list)**
  - **present_or_copy (list)**
  - **present_or_copyin (list)**
  - **present_or_copyout (list)**
  - **present_or_create (list)**
  - **deviceprt (list)**

# Details of clauses

**copy clause**

**– Specifies items that need to be copied-in from the host to accelerator, and then copy-out at the end of the region**

**– Allocates accelerator memory for the copy items.**

- **copy-in clause**

**– Specifies items that need to be copied-in to the accelerator memory**

**– Allocates accelerator memory for the copy-in items**

- **copy-out clause**

**– Specifies items that need to be copied-out to the accelerator memory allocates accelerator memory for the copy-out items**

**create clause**

**– Specifies items that need to allocated (created) in the accelerator memory**

**– The values of such items are not needed by the host**

**• copy-in clause**

**– Specifies items that need to be copied-in to the accelerator memory**

**– Allocates accelerator memory for the copy-in items**

**• present clause**

**– Specifies items are already present in the accelerator memory**

**– The items were already allocated on other data, parallel or kernel regions. (i.e. inter-procedural calls)**

- **present_or_copy clause**

**– Tests if a data item is already present in the accelerator. If not, it will allocate the item in the accelerator and copy-in and out its value from/to the host**

- **present_or_copyin clause**

**– Test if a data item is already present in the accelerator. If not, it will allocate the item in the accelerator and copy-in its value from the host**

- **present_or_copyout clause**

**– Test if a data item is already present in the accelerator. If not, it will allocate the item in the accelerator and copy-out its value to the host**

- **present_or_create clause**

**– Test if a data item is already present in the accelerator. If not, it will allocate the item in the accelerator (no initialization)**

# OpenAcc Loop Directive

- **Used to describe what type of parallelism to use to execute the loop in the accelerator.**

- **Can be used to declare loop-private variables, arrays and reduction operations.**

- **Specified by:**
  - **#pragma acc loop [clause [,clause]…] new-line**
    **//for loop**

**The clauses for the !$acc loop directive are:**

- **collapse (n)**
- **gang [( scalar-integer-expression )]**
- **worker [( scalar-integer-expression )]**
- **vector [( scalar-integer-expression )]**
- **seq**
- **independent**
- **private (list)**
- **reduction ( operator : list)**

- **collapse directive**

**– Specifies how many tightly nested loops are associated with the loop construct**

**gang clause**

**– Within a parallel region: it specifies that the loop iteration need to be distributed among gangs.**

**– Within a kernel region: that the loop iteration need to be distributed among gangs. It can also be used to specify how many gangs will execute the iteration of a loop**

- **worker clause**

**– Within a parallel region: it specifies that the loop iteration need to be distributed among workers of a gang.**

**– Within a kernel region: that the loop iteration need to be distributed among workers of a gang. It can also be used to specify how many workers of a gang will execute the iteration of a loop**

- **seq clause**

**– Specifies that a loop needs to be executed sequentially by the Accelerator**

**vector clause**

**– Within a parallel region: specifies that the loop iterations need to be in vector or SIMD mode. It will use the vector length specified by the parallel region**

**– Within a kernel region: specifies that the loop iterations need to be in vector or SIMD mode. If an argument is specified, the iterations will be processed in vector strips of that length.**

- **independent clause**

**– Specifies that there are no data dependences in the loop**

- **private clause**

**– Specifies that a copy of each item on the list will be created for each iterations of the loop.**

- **reduction clause**

**– Specifies that a reduction need to be perform associated to a gang.**

# OpenACC directives examples

```
#pragma acc data copy(a[0:n]) copyin(b[0:n])
{
    //some code
    axpy(a,b,1.0,n);
    //some more code
}
void axpy( float* y, float* x, float a, int n ){
    #pragma acc parallel loop present(a[0:n],b[0:n])
    for(int i=0;i<n;i++)
        y[i] +=a*x[i];
}
```

# A simple working example: Estimating pi

```
#include <stdio.h>
//#include <omp.h>
#define N 1000000

  int main(void) {
  double pi = 0.0f; long i;
  //#pragma omp parallel for reduction(+:pi)
  #pragma acc parallel loop reduction(+:pi)
  for (i=0; i<N; i++) {
     double t= (double)((i+0.5)/N);
     pi +=4.0/(1.0+t*t);
  }
  printf("pi=%16.15f\n",pi/N);
  return 0;
  }
```

# Building your OpenAcc program

- PGI compiler installed on gpu00 & gpu01
  - Login to one of these two nodes
  - “module load pgi”
- To compile your code
  - **pgcc -acc -ta=tesla:cc35 test_openacc.c [more compiler options]**
- Very useful to set environment variable
  - **PGI_ACC_NOTIFY=3**
  - **You will see what the compiler is doing for you**
- **If you want to see what GPU do you have and its compute capability**
  - **Type the command pgaccelinfo**

# More examples