Introduction to OpenACC
Our First OpenACC Program

Stefano Markidis

Two Key-Points

1. OpenACC is a set of compiler directives, library routines and environment variables for programming GPUs
2. We write our first OpenACC code for matrix multiplication using `#pragma acc`, `parallel loop` and `copyin` and `copyout` directive clauses

What is OpenACC?

- The OpenACC API provides a set of
    1. compiler directives
    2. library routines
    3. environment variables

OpenACC
More Science, Less Programming

- Use to write data-parallel FORTRAN, C/C11 programs that run on accelerator devices, including GPUs.
    - It is an extension to the host language.
- The OpenACC specification was initially developed by the Portland Group (PGI, now under Nvidia) and Cray with support from CAPS.

History of OpenACC

- Around 2010, compiler vendors started to look at directives for GPUs.
  - Everyone developed their own set of directives.
  - Moved some of the burden from the programmer to the compiler.
    - Mostly Nvidia as the target.
  - Not intended to be only for GPUs but it has ended up that way
- Rather than wait for OpenMP to incorporate accelerators, Nvidia, CAPS, Cray & PGI create the OpenACC board.
- The board release OpenACC standard at **SC11**.
- OpenMP 4 has support for accelerators and will likely replace OpenACC
  - Our installed gcc compilers do not support OpenMP for accelerators

Run Code on GPU – Use `pragma acc`

```
void matMulAcc(float *P, const float *M, const float *N,
                                int Mh, int Mw, int Nw){

    #pragma acc parallel loop copyin(M[0:Mh*Mw])
                               copyin(N[0:Nw*Mw]) copyout(P[0:Mh*Nw])
    for (int i=0; i < Mh; i++) {
        #pragma acc loop
        for (int j=0; j < Nh; j++) {
            float sum = 0.0;
             for (int k=0; k < Mw; k++) {
                 float a = M[i*Mw+k]) ;
                 float b = N[k*Nw+j]) ;
                 sum += a*b
             }
             P[i*Nw+j] = sum;
        }
    }
}
```

- The code in the figure is almost identical to the sequential version, except for the two lines with `#pragma acc`
  - to provide, to the compiler, information that is not specified in the standard language.
- OpenACC uses the compiler directive mechanism to extend the base language.

Loop executed on GPU - `#pragma acc parallel loop`

```
void matMulAcc(float *P, const float *M, const float *N,
                                int Mh, int Mw, int Nw){

    #pragma acc parallel loop copyin(M[0:Mh*Mw])
                              copyin(N[0:Nw*Mw]) copyout(P[0:Mh*Nw])
    for (int i=0; i < Mh; i++) {
        #pragma acc loop
        for (int j=0; j < Nh; j++) {
            float sum = 0.0;
             for (int k=0; k < Mw; k++) {
                 float a = M[i*Mw+k]) ;
                 float b = N[k*Nw+j]) ;
                 sum += a*b
             }
             P[i*Nw+j] = sum;
        }
    }
}
```

The `#pragma acc parallel loop` tells the compiler to generate code for the `i` loop so that the loop iterations are executed in parallel on the accelerator

Data Movement with OpenACC – `copyin` and `copyout`

```
void matMulAcc(float *P, const float *M, const float *N,
                                int Mh, int Mw, int Nw){

    #pragma acc parallel loop copyin(M[0:Mh*Mw])
                              copyin(N[0:Nw*Mw]) copyout(P[0:Mh*Nw])
    for (int i=0; i < Mh; i++) {
        #pragma acc loop
        for (int j=0; j < Nh; j++) {
            float sum = 0.0;
             for (int k=0; k < Mw; k++) {
                 float a = M[i*Mw+k]) ;
                 float b = N[k*Nw+j]) ;
                 sum += a*b
             }
             P[i*Nw+j] = sum;
        }
    }
}
```

The `copyin` clause and the `copyout` clause specify how the matrix data should be transferred between the host and the accelerator

Work-Sharing - #pragma acc loop

```
void matMulAcc(float *P, const float *M, const float *N,
                              int Mh, int Mw, int Nw){

    #pragma acc parallel loop copyin(M[0:Mh*Mw])
                              copyin(N[0:Nw*Mw]) copyout(P[0:Mh*Nw])
    for (int i=0; i < Mh; i++) {
        #pragma acc loop
        for (int j=0; j < Nh; j++) {
            float sum = 0.0;
             for (int k=0; k < Mw; k++) {
                float a = M[i*Mw+k]) ;
                float b = N[k*Nw+j]) ;
                sum += a*b
             }
             P[i*Nw+j] = sum;
        }
    }
}
```

The `#pragma acc loop` instructs the compiler to map the inner `j` loop using work-sharing.

We used it to also to parallelize over `i`

Compiling OpenACC

We will use the `pgcc` compiler from Portland Group available on the GPU cluster:

```
pgcc -acc -ta=tesla:cc3x,nvidia,lineinfo -Minfo=accel code.c -o code
```

OpenACC flag

OpenACC Pros/Cons

- **Pro:** OpenACC programmers can often start with writing a sequential version and then annotate their sequential program with OpenACC directives.
- **Pro:** OpenACC provides an incremental path for moving legacy applications to accelerators.
  - Adding directives disturbs the existing code less than other approaches
- **Pro:** A non-OpenACC compiler is not required to understand and process OpenACC directives, therefore it can just ignore the directives and compile the rest of the program as usual
- **Con:** Highly-tuned CUDA code has higher performance than OpenACC versions

To Summarize

- OpenACC is a set of compiler directives, library routines and environment variables for programming GPUs
- We have written our first OpenACC code for matrix multiplication using `#pragma acc`, `parallel loop` and `copyin` and `copyout` directive clauses