INTRODUCTION TO OPENACC

Hossein Pourreza

hossein.pourreza@umanitoba.ca

March 31, 2016

Acknowledgement: Most of examples and pictures are from PSC (https://www.psc.edu/images/xsedetraining/OpenACC_May2015/introductiontoopenacc.pdf) Please do not redistribute the slides without permission of the original author (John Urbanic).

Logistics of this webinar

- Login to Parallel using your WestGrid username/password
  - ssh -Y username@parallel.westgrid.ca if you are using Linux or Mac OR
  - Use the SSH client of your choice on Windows
    - E.g., putty or MobaXterm (http://mobaxterm.mobatek.net/download.html)

- Copy the `openacc-wg-2016` folder to your home directory using the following command:
  - `cp -r /global/software/examples/openacc-wg-2016 .`
  - Please do not forget the . (dot) at the end

Logistics of this webinar

- Once you are in the openacc-wg-2016 folder, run:

```
sh acc_node.sh
```

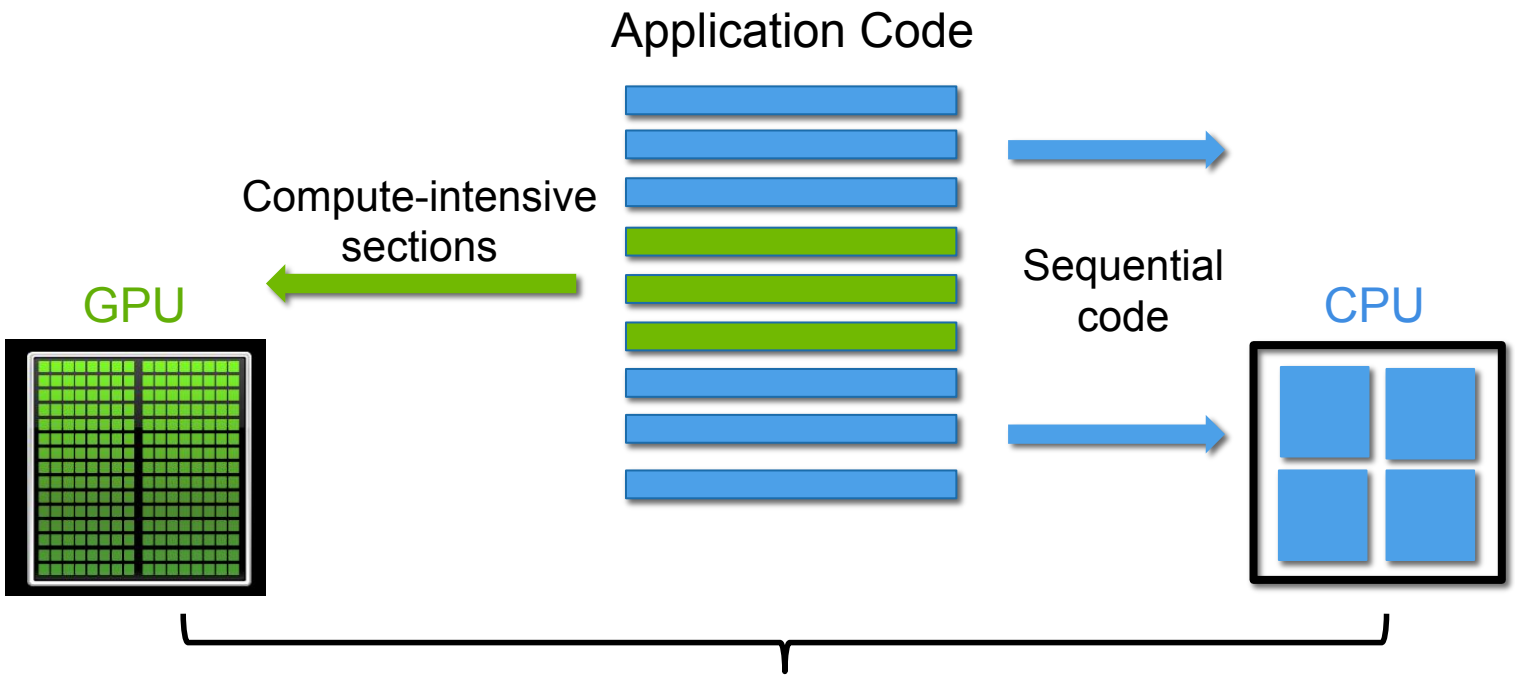
- Please run the `module load pgi/14.6` command to access the PGI compilers

- Please use editors such as vim, nano, etc. to edit the files
  - Please refrain from transferring the files to Windows, editing, and transferring them back to Parallel
  - Copying codes/commands from slides may result in unexpected error messages

GPU: Graphical Processing Unit

- Designed for graphics
- Optimized for parallel tasks with hundreds of cores
- Good at doing simple and repeated calculations
- High memory bandwidth and flops/watt
- We will use NVIDIA GPUs in this webinar

Heterogeneous computing



Picture is adapted from https://computing.llnl.gov/tutorials/2014.09.15-16.NVIDIA-OpenACC.pdf

GPU computing

- Libraries
  - NVIDIA Math Libraries, CULA , MAGMA, etc.
- Compiler directives
  - OpenACC
- Programming languages
  - NVIDIA CUDA toolkit for C/C++, PGI CUDA Fortran, NVIDIA OpenCL, PyCUDA

Increased flexibility and complexity

OpenACC

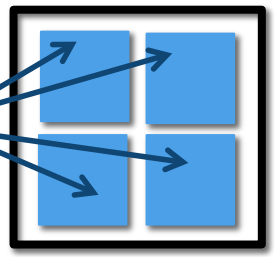
- Programming standard developed by NVIDIA, Cray, CAPS, and PGI
- Includes GPU directives to annotate C/C++ and Fortran code
- Can be used on different platforms such as GPUs (NVIDIA and AMD), Intel Xeon Phi, etc.

Key advantages

- High-level
  - No involvement of OpenCL, CUDA, etc.
- Single source
  - Compile the same program for accelerators or serial execution
- Efficient
  - Comparable to the low-level implementations of the same algorithm
- Performance portable
  - Supports GPU accelerators and co-processors from multiple vendors
- Incremental

OpenACC is similar to OpenMP

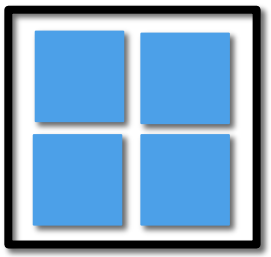
CPU



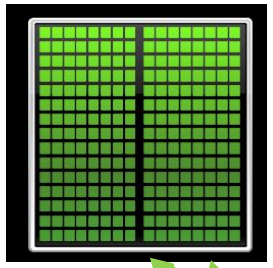
```
main(){
  double pi = 0.0;long i;

#pragma omp parallel for reduction(+:pi)
  for(i=0;i < N;i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }
  printf("pi = %f\n",pi/N);
}
```

CPU



GPU



```
main(){
  double pi = 0.0;long i;

#pragma acc kernels
  for(i=0;i < N;i++)
  {
    double t = (double)((i+0.05)/N);
    pi += 4.0/(1.0+t*t);
  }
  printf("pi = %f\n",pi/N);
}
```

Basics

- In C/C++ all OpenACC directives start with `#pragma acc`
  - These lines will be skipped by non-OpenACC compilers
    - You may get a warning message but it should be OK

```
#pragma acc directive [clause]
{
  …
}
```

- In Fortran all OpenACC directives start with `!$acc`

```
!$acc directive [clause]
    …
!$acc end directive
```

`kernels` directive

The `kernels` construct expresses that a region may contain parallelism and the compiler determines what can safely be parallelized.

```
#pragma acc kernels
{
  for(i=1;i<n;i++){
    a[i] = 0.0;
    b[i] = 1.0;
  }
  for(i=0;i < n;i++)
    a[i] = b[i] + 5;
}
```

Kernel 1

Kernel 2

Kernel: A parallel routine to run on the GPU

`parallel loop` directive

C

```
#pragma acc parallel loop
{
  …
}
```

Fortran

```
!$acc parallel loop
  …
!$acc end parallel loop
```

Similar to OpenMP we can use `parallel for` in C and `parallel do` in Fortran. To keep it consistent, `parallel loop` can be used in both.

`kernels` vs. `parallel loop`

- `kernels`
  - Gives more flexibility to compiler to parallelize loops
  - Covers larger sections of code

- `parallel loop`
  - Requires programmer to analyze the dependencies
  - Is a simple transition from OpenMP

SAXPY

```
#include "ticktock.h"
int main(){
  int n=1e7; float a = 5./3.;int i;
  float *x = malloc(sizeof(float)*n);
  float *y = malloc(sizeof(float)*n);

  //initialize vectors
  for (i=0; i<n; i++){
    x[i] = 2.0f;
    y[i] = (i+1.)*(i-1.);
  }
  tick_tock tt;
  tick(&tt);
  saxpy(n,a,x,y);
  tock(&tt);
  free(x);
  free(y);
  return 0;
}
```

```
void saxpy(int n, float a, float *x,
float *restrict y){
  int i;
  #pragma acc kernels
  for (i=0; i<n; i++)
    y[i] = a * x[i] + y[i];
}//end of saxpy
```

The `restrict` keyword: C99 standard

- Important for optimization of serial as well as OpenACC and OpenMP code.

- Promise to the compiler for a pointer

  ```
  float *restrict y
  ```

  - Meaning: “for the lifetime of y, only it or a value directly derived from it (such as y + 1) will be used to access the object to which it points”

- Limits the effects of pointer aliasing

Compile and run

```
$pgcc -acc -Minfo=accel saxpy.c -o saxpy
$pgf90 –acc –Minfo=accel saxpy.f90 –o saxpy
```

```
saxpy:
  5, Generating present_or_copyin(x[:n])
     Generating present_or_copy(y[:n])
     Generating Tesla code
  6, Loop is parallelizable
     Accelerator kernel generated
  6, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

```
$./saxpy
```

Note: You need to compile your code on the Parallel login node and run it on the Parallel compute node. It will be easier to have two terminals open.

Matrix multiplication

```
#include <stdio.h>
#include "ticktock.h"
#define SIZE 1000
float a[SIZE][SIZE], b[SIZE][SIZE],
c[SIZE][SIZE];
int main(){
  int i,j;
  for(i=0;i<SIZE;i++)
    for(j=0;j<SIZE;j++){
      a[i][j] = (float)(i+j);
      b[i][j] = (float)(i-j);
      c[i][j] = 0.0f;
    }
  tick_tock tt;
  tick(&tt);
  matmul();
  tock(&tt);
  return 0;
}
```

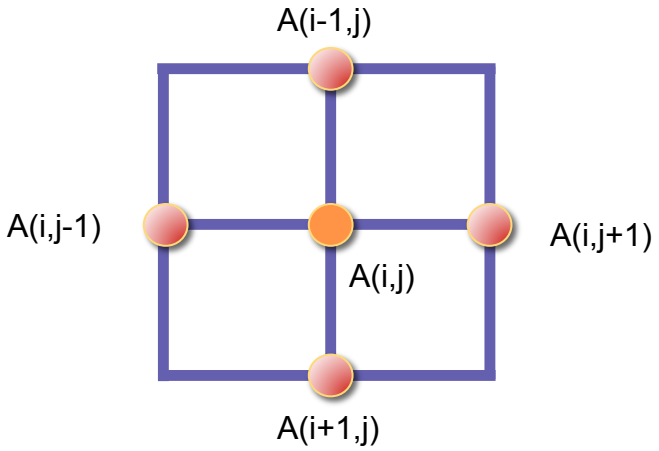
```
void matmul(){
  int i,j,k;
  #pragma acc kernels
  for(i=0;i<SIZE;i++)
    for(j=0;j<SIZE;j++){
      float sum = 0.0f;
      for(k=0;k<SIZE;k++)
        sum += a[i][k] * b[k][j];
      c[i][j] = sum;
    }
}
```

Compile and run

```
$pgcc -acc -Minfo=accel ticktock.c matmul_acc.c -o matmul_acc
$pgf90 –acc –Minfo=accel ticktock.f90 matmul_acc.f90 –o matmul_acc
matmul:
     12, Generating present_or_copyin(a[:][:])
         Generating present_or_copyin(b[:][:])
         Generating present_or_copyout(c[:][:])
         Generating Tesla code
     13, Loop is parallelizable
     14, Loop is parallelizable
         Accelerator kernel generated
         13, #pragma acc loop gang /* blockIdx.y */
         14, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
     16, Loop is parallelizable
$./matmul_acc
```

Fundamental example

- Laplace equation for two dimensional heat conduction problem:



$$A(i,j) = \frac{A_k(i-1,j)+A_k(i+1,j)+A_k(i,j-1)+A_k(i,j+1)}{4}$$

Serial code

```
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++)
      Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
              Temperature_last[i-1][j]+ Temperature_last[i][j+1] +
              Temperature_last[i][j-1]);

  dt = 0.0;

  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
      Temperature_last[i][j] = Temperature[i][j];
    }

  if((iteration % 100) == 0) {
    track_progress(iteration);
  }
  iteration++;
}
```

Compute

Update

Print

Helper functions

```
void initialize(){
  int i,j;
  for(i = 0; i <= ROWS+1; i++)
    for(j = 0; j <= COLUMNS+1; j++)
      Temperature_last[i][j] = 0.0;

  // these boundary conditions never change throughout run
  // set left side to 0 and right to a linear increase
  for(i = 0; i <= ROWS+1; i++){
    Temperature_last[i][0] = 0.0;
    Temperature_last[i][COLUMNS+1] = (100.0/ROWS)*I;
  }

  // set top to 0 and bottom to linear increase
  for(j = 0; j <= COLUMNS+1; j++){
    Temperature_last[0][j] = 0.0;
    Temperature_last[ROWS+1][j] = (100.0/COLUMNS)*j;
  }
}
```

```
void track_progress(int iteration){
  int i;
  printf("-- Iteration: %d --\n", iteration);
  for(i = ROWS-5; i <= ROWS; i++)
    printf("[%d,%d]: %5.2f ", i, i,
           Temperature[i][i]);
  printf("\n");
}
```

OpenACC version

```
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++)
      Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
              Temperature_last[i-1][j]+ Temperature_last[i][j+1] +
              Temperature_last[i][j-1]);

  dt = 0.0;
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
      Temperature_last[i][j] = Temperature[i][j];
    }

  if((iteration % 100) == 0) {
    track_progress(iteration);
  }
  iteration++;
```

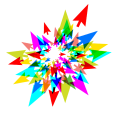
Compile and run

```
$pgcc -acc -Minfo=accel laplace_acc.c -o laplace_acc
$pgf90 –acc -Minfo=accel laplace_acc.f90 –o laplace_acc
main:
     34, Generating present_or_copyout(Temperature[1:1000][1:1000])
         Generating present_or_copyin(Temperature_last[:][:])
         Generating Tesla code
     35, Loop is parallelizable
     36, Loop is parallelizable
         Accelerator kernel generated
         35, #pragma acc loop gang /* blockIdx.y */
         36, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
     42, Generating present_or_copyin(Temperature[1:1000][1:1000])
         Generating present_or_copy(Temperature_last[1:1000][1:1000])
         Generating Tesla code
     43, Loop is parallelizable
     44, Loop is parallelizable
         Accelerator kernel generated
         43, #pragma acc loop gang /* blockIdx.y */
         44, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
         45, Max reduction generated for dt
$./laplace_acc
```

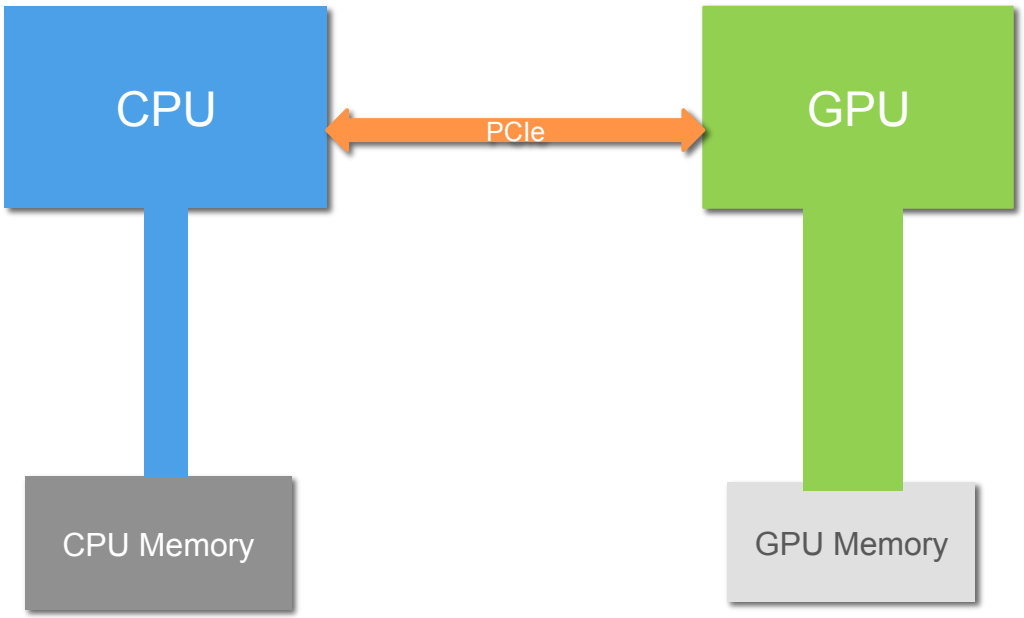
Slower than serial and much slower than OpenMP version!!

Running with profiling on

```
$export PGI_ACC_TIME=1;./laplace_acc
main  NVIDIA  devicenum=0
    time(us): 2,252,965
    34: data region reached 3372 times
        34: data copyin transfers: 3372
             device time(us): total=43,193 max=185 min=4 avg=12
        40: data copyout transfers: 3372
             device time(us): total=32,656 max=217 min=7 avg=9
    34: compute region reached 3372 times
        36: kernel launched 3372 times
            grid: [8x1000]  block: [128]
             device time(us): total=981,984 max=297 min=276 avg=291
            elapsed time(us): total=1,022,141 max=476 min=296 avg=303
    42: data region reached 3372 times
        42: data copyin transfers: 6744
             device time(us): total=107,420 max=63 min=4 avg=15
        49: data copyout transfers: 3372
             device time(us): total=29,125 max=18 min=6 avg=8
    42: compute region reached 3372 times
        44: kernel launched 3372 times
            grid: [8x1000]  block: [128]
             device time(us): total=1,028,970 max=500 min=297 avg=305
            elapsed time(us): total=1,065,287 max=510 min=307 avg=315
        44: reduction kernel launched 3372 times
            grid: [1]  block: [256]
             device time(us): total=29,617 max=207 min=8 avg=8
            elapsed time(us): total=65,491 max=220 min=17 avg=19
```



Data movement



Data management

C

```
#pragma acc data [clause]
{
    …
}
```

Fortran

```
!$acc data [clause]
    …
!$acc end data
```

Data clauses

- `copy( list )`
  - Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.
    - Principal use: For many important data structures in your code, this is a logical default to input, modify and return the data.
- `copyin( list )`
  - Allocates memory on GPU and copies data from host to GPU when entering region.
    - Principal use: Think of this like an array that you would use as just an input to a subroutine.
- `copyout( list )`
  - Allocates memory on GPU and copies data to the host when exiting region.
    - Principal use: A result that isn't overwriting the input data structure.
- `create( list )`
  - Allocates memory on GPU but does not copy.
    - Principal use: Temporary arrays.

Array shaping

- Compilers sometimes cannot determine the size of arrays, so we must specify explicitly using data clauses with an array “shape”.
  - The compiler will let you know if you need to do this. Sometimes, you will want to for your own efficiency reasons.
- C

```
#pragma acc data copyin(a[0:size]), copyout(b[s/4:3*s/4])
```

- Fortran

```
!$acc data copyin(a(1:size)), copyout(b(s/4:3*s/4))
```

- Fortran uses start:end and C uses start:length

`update` directive

```
#pragma acc update host(a)
```

`host`: copies data from the device (GPU) to the host (CPU)

`device`: copies from the host to the device

- Explicitly transfers data between the host and the device
- Useful when you want to update data in the middle of a data region Clauses:

OpenACC solution with data clause

```
#pragma acc data copy(Temperature_last) create(Temperature)
while ( dt > MAX_TEMP_ERROR && iteration <= max_iterations ) {
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++)
      Temperature[i][j] = 0.25 * (Temperature_last[i+1][j] +
              Temperature_last[i-1][j]+ Temperature_last[i][j+1] +
              Temperature_last[i][j-1]);
  dt = 0.0;
  #pragma acc kernels
  for(i = 1; i <= ROWS; i++)
    for(j = 1; j <= COLUMNS; j++){
      dt = fmax( fabs(Temperature[i][j]-Temperature_last[i][j]), dt);
      Temperature_last[i][j] = Temperature[i][j];
    }
  if((iteration % 100) == 0) {
    #pragma acc update host(Temperature)
    track_progress(iteration);
  }
  iteration++;
```

Conclusion

- OpenACC is an easy and quick way to make use of GPUs to run your code faster

- Data dependency and data movement are two important challenges to get a good speedup from GPU-based code

Conclusion

- WestGrid's Tier 2 and Tier 3 support are available to help you with your parallel programming needs

- References and useful links:
  - http://www.openacc.org
  - https://www.psc.edu/images/xsedetraining/OpenACC_May2015/introductiontoopenacc.pdf
  - http://courses.cms.caltech.edu/cs101gpu/2015_lectures/
  - http://www.nvidia.com/docs/IO/116711/SC11-NV-TESLA.pdf
  - http://on-demand.gputechconf.com/gtc/2013/presentations/S3076-Getting-Started-with-OpenACC.pdf
  - https://computing.llnl.gov/tutorials/2014.09.15-16.NVIDIA-OpenACC.pdf