CSE 746 - Parallel and High Performance Computing Lecture 8 - Introduction to OpenACC

Pawel Pomorski, *HPC Software Analyst*
SHARCNET, University of Waterloo
ppomorsk@sharcnet.ca
http://ppomorsk.sharcnet.ca/

Assignment discussion

- Let's figure out why the CPU start code for the assignment was so slow
- Use starting code in: /home/ppomorsk/CSE746_lec8/assignment1_cpu_matmulti
- Possible problem:

```
for (i=0; i < TILE_DIM; i++) {
    sum+= a[row+i*N]*b[i+col*TILE_DIM];      }
```

- a matrix is accessed with stride N in the loop
- add loops to transpose matrix a into a_transp, and use that matrix for sequential access
- compare speeds, try gcc and icc compilers, with -O2 and -O3 flags.

Assignment discussion

- Problem is cache misses
- Even on CPU it pays to access memory sequentially
- Reference:
  What every programmer should know about memory
  http://www.akkadia.org/drepper/cpumemory.pdf

OpenACC

- New standard for parallel computing developed by compiler makers. See: http://www.openacc-standard.org/
- Specified in late 2011, released in 2012
- SHARCNET has the PGI compiler on monk which supports it
- OpenACC works somewhat like OpenMP
- Goal is to provide simple directives to the compiler which enable it to accelerate the application on the GPU
- The tool is aimed at developers aiming to quickly speed up their code without extensive recoding in CUDA
- As tool is very new and this course focuses on CUDA, only a brief demo of OpenACC follows

OpenACC

- Programming framework designed to be easier to use than CUDA while producing useful GPU acceleration
- similar to OpenMP in philosophy
- suggested workflow:
  - examine the serial code you wish to accelerate on the GPU, identify hotspots in code which would execute efficiently on the GPU
  - straightforwardly accelerate those sections with OpenACC directives
  - manage data flow between CPU and GPU if needed
  - if acceleration is sufficient you are done, otherwise recode parts that need better acceleration in CUDA

OpenACC specification

- Documentation

  http://openacc.org/

  Version 1.0 specification - November 2011
  http://openacc.org/

  Handy cheat sheet:
  http://www.nvidia.com/docs/IO/116711/OpenACC-API.pdf

OpenACC at SHARCNET - via PGI compiler

- Documentation

  Full list:
  http://www.pgroup.com/resources/accel.htm

  "Getting started " guide
  https://www.pgroup.com/doc/openACC_gs.pdf

Compiling OpenACC with PGI at Sharcnet

- Need to switch from default compiler to PGI, and set some useful environment variables for automatic profiling before compiling.

```
module unload intel
module load pgi
export ACC_NOTIFY=1
export PGI_ACC_TIME=1
pgcc -acc -Minfo=accel -fast your_code.c
```

OpenACC pragmas

- OpenACC directives specified by pragma mechanism

  #pragma acc clause ...
- in principle, the starting serial code to which pragmas are added need not be changed in any way
- in practice you may have to modify code to make it easier for compiler to parallelize it

Accelerator compute constructs

- Parallel construct

  #pragma acc parallel ...
  structured block

  where structured block in C/C++ is just code in {} brackets
- can also apply to loops or single statements
- note the similarity to OpenMP
  #pragma openmp ...
- OpenACC and OpenMP directives can in fact be used in the same code (these approaches may merge in future)
- in fact CUDA and OpenACC can be used in same code

Parallel construct

- When program enters a parallel construct, tasks are distributed via a hierarchy of parallelism
- OpenACC has three parallelism levels:
  vector
  worker
  gang

Parallel construct

- Somewhat confusingly, the relation of these to CUDA threads, warps and thread blocks is implementation dependent

  Possibilities:

  gang = block,worker = warp, vector = threads of warp

  or, omit worker and just have

  gang = block, vector = threads of block

- Depends on what compiler thinks is best, you can leave it up to compiler, but sometimes need to provide specific values if needed to tune the performance

Parallel construct

- Can think of a parallel construct as a single CUDA kernel
- like OpenMP threads, all OpenACC gangs started by the construct execute the same code redundantly, until they reach a work-sharing construct, then split the work of that construct across threads or gangs

Kernels construct

- similar to parallel construct, but with somewhat different syntax and very important differences
- parallel construct:
  - it **will** accelerate region even if code leads to incorrect results (race conditions etc.) Same approach as OpenMP
  - it creates a single kernel
  - offers greater control, acceleration always happens
- kernels construct:
  - **may** accelerate region if compiler decides loop iterations are independent (i.e. it's safer, correct results more likely)
  - can create multiple kernels, each possibly with different number of gangs/workers/vectors
  - not knowing if acceleration actually occurred can be problematic

Loop directive in OpenACC

- makes it possible to provide more precise instructions on how to accelerate a loop
- can be combined with parallel directive

  #pragma acc parallels loop
- can declare variables:
  shared - all iterations process same variable
  private - each iteration uses variable separately
  first_private - variation of private, each thread copy set to initial value
- note similarity to OpenMP, but it's **not exactly the same**

Loop directive in OpenACC

- reduction clause available

  #pragma acc parallel loop reduction(+:sum)

  for standard reduction (“+” operation), sum is automatically set to zero at start

  max,min and other reduction operations available

From lecture 1: SAXPY with OpenACC

```
...
#include <openacc.h>

void saxpy_openacc(float *restrict vecY, float *vecX, float alpha, int n)
{
     int i;
#pragma acc kernels
     for (i = 0; i < n; i++)
          vecY[i] = alpha * vecX[i] + vecY[i];
}

...
   /* execute openacc accelerated function on GPU */
   saxpy_openacc(y_shadow, x_host, alpha, n);
...
```

- OpenACC automatically builds a kernel function that will run on GPU
- Memory transfers between device and host handled by OpenACC and need not be explicit

“Hello, world!” in OpenACC

```
/* in /home/ppomorsk/CSE746_lec8/hello/hello_acc.c */

int main(){

   int i;
   int N=1024*1024;
   int a[N];

# pragma acc parallel loop
   for(i=0;i<N;i++){
      a[i]=i;
   }

# pragma acc parallel loop
   for(i=0;i<N;i++){
       a[i] = 2*a[i];
   }

   return 0;
   }
```

Compile “Hello, world!” in OpenACC

```
[ppomorsk@mon241:~/CSE746_lec8/hello] pgcc -acc -Minfo=accel -fast hello_acc.c
main:
     20, Accelerator kernel generated
         21, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
             CC 1.0 : 8 registers; 32 shared, 0 constant, 0 local memory bytes
             CC 2.0 : 9 registers; 0 shared, 48 constant, 0 local memory bytes
     20, Generating copyout(a[0:1048576])
         Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
     21, Loop is parallelizable
     25, Accelerator kernel generated
         26, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
             CC 1.0 : 8 registers; 32 shared, 0 constant, 0 local memory bytes
             CC 2.0 : 10 registers; 0 shared, 48 constant, 0 local memory bytes
     25, Generating copy(a[0:1048576])
         Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
     26, Loop is parallelizable
```

Run “Hello, world!” in OpenACC

```
[ppomorsk@mon54:~/CSE746_lec8/hello] ./a.out
launch kernel  file=/home/ppomorsk/CSE746_lec8/hello/hello_acc.c function=main line=20
device=0 grid=4096 block=256 queue=0
launch kernel  file=/home/ppomorsk/CSE746_lec8/hello/hello_acc.c function=main line=25
device=0 grid=4096 block=256 queue=0

Accelerator Kernel Timing data
/home/ppomorsk/CSE746_lec8/hello/hello_acc.c
  main
    25: region entered 1 time
        time(us): total=4622
                  kernels=101 data=4326
        25: kernel launched 1 times
            grid: [4096]  block: [256]
            time(us): total=101 max=101 min=101 avg=101
/home/ppomorsk/CSE746_lec8/hello/hello_acc.c
  main
    20: region entered 1 time
        time(us): total=3516374 init=3512626 region=3748
                  kernels=106 data=3144
        w/o init: total=3748 max=3748 min=3748 avg=3748
        20: kernel launched 1 times
            grid: [4096]  block: [256]
            time(us): total=106 max=106 min=106 avg=106
```

Data clauses in OpenACC

- applies to structured block {} which immediately follows

```
#pragma acc data copy(A[0:N])
{     // start of structured block
//code which uses data in A
}     // end of structured block
```

- dimensions of A need not be specified if known
- can copy just a subarray of A
- multiple data regions can overlap
- unspecified arrays still handled automatically by OpenACC

Data clauses in OpenACC

- if not given specific instructions on how to do it, OpenACC will handle all data movement between the device and host
- unfortunately, in this approach all needed data must be copied to the device at the start of a parallel region and copied back to host at the end. No data survives on the device after leaving parallel region
- this can lead to redundant data movement if one has multiple parallel region
- to avoid this, a way is needed to handle data that is kept on the device after a parallel region exits
- in general one usually needs more precise control over data movement
- this is provided by the data clause in OpenACC

Data clauses in OpenACC

- copyin - for variables with values in host memory that need copying to device memory when entering region
- copyout - for variables with values in device memory that need copying to host memory when exiting region
- copy - for variables that have values in host memory that need to be copied to device memory when entering region, assigned values on the accelerator, then copied back to the host when leaving region
- create - allocates data on device but does not copy between host and device
- present - data must already be present on accelerator from some contain
- present_or_copyin, present_or_copyout,present_or_copy, present_or_create - will carry out these operations if data not present not present on device

“Hello, world!” in OpenACC, with data

```
/* in /home/ppomorsk/CSE746_lec8/hello/hello_acc_data.c */
int main(){

   int i;
   int N=1024*1024;
   int a[N];

# pragma acc data copyout(a[0:N])
{
# pragma acc parallel loop
   for(i=0;i<N;i++){
      a[i]=i;
   }

# pragma acc parallel loop
   for(i=0;i<N;i++){
       a[i] = 2*a[i];
   }
}
   return 0;
   }
```

Compile “Hello, world!” in OpenACC, with data

```
[ppomorsk@mon54:~/CSE746_lec8/hello] pgcc -acc -Minfo=accel -fast  hello_acc_data.c
main:
     20, Generating copyout(a[0:N])
     22, Accelerator kernel generated
         23, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
             CC 1.0 : 8 registers; 36 shared, 0 constant, 0 local memory bytes
             CC 2.0 : 9 registers; 0 shared, 52 constant, 0 local memory bytes
     22, Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
     23, Loop is parallelizable
     27, Accelerator kernel generated
         28, #pragma acc loop gang, vector(256) /* blockIdx.x threadIdx.x */
             CC 1.0 : 8 registers; 36 shared, 0 constant, 0 local memory bytes
             CC 2.0 : 10 registers; 0 shared, 52 constant, 0 local memory bytes
     27, Generating compute capability 1.0 binary
         Generating compute capability 2.0 binary
     28, Loop is parallelizable
```

Run “Hello, world!” in OpenACC, with data

```
[ppomorsk@mon54:~/CSE746_lec8/hello] ./a.out
launch kernel  file=/home/ppomorsk/CSE746_lec8/hello/hello_acc_data.c
function=main line=22 device=0 grid=4096 block=256 queue=0
launch kernel  file=/home/ppomorsk/CSE746_lec8/hello/hello_acc_data.c
function=main line=27 device=0 grid=4096 block=256 queue=0

Accelerator Kernel Timing data
/home/ppomorsk/CSE746_lec8/hello/hello_acc_data.c
  main
    27: region entered 1 time
        time(us): total=187
                  kernels=106 data=0
        27: kernel launched 1 times
            grid: [4096]  block: [256]
            time(us): total=106 max=106 min=106 avg=106
/home/ppomorsk/CSE746_lec8/hello/hello_acc_data.c
  main
    22: region entered 1 time
        time(us): total=235
                  kernels=105 data=0
        22: kernel launched 1 times
            grid: [4096]  block: [256]
            time(us): total=105 max=105 min=105 avg=105
/home/ppomorsk/CSE746_lec8/hello/hello_acc_data.c
  main
    20: region entered 1 time
        time(us): total=3523640 init=3519742 region=3898
                  data=3124
        w/o init: total=3898 max=3898 min=3898 avg=3898
```

Exercise - try “Hello, world!” programs

Code in:

/home/ppomorsk/CSE746_lec8/hello

Exercise - compute Pi

Start serial code can be found in:

/home/ppomorsk/CSE746_lec8/pi

Only one pragma line needs to be added here. Do you need to use reduction? Check and see.

Exercise - Jacobi iteration

Start code can be found in:

/home/ppomorsk/CSE746_lec8/Laplace2D

Use kernels construct

Need 3 pragmas - 2 for the loops, one for data movement, one loop will need reduction

Experiment with different gang,worker,vector arrangements and compare timing

Exercise: Black-Sholes

Start code can be found in:

/home/ppomorsk/CSE746_lec8/blackscholes

Code computes futures pricing for array of stocks via Black-Scholes formula

Add just one parallel pragma

Use the “for if(variable) “ in your construct

Exercise: Game of Life

Start code can be found in:

/home/ppomorsk/CSE746_lec8/gameoflife

Code runs Conway's Game of Life

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life