Introduction to OpenACC

Jeff Larkin

OpenACC
DIRECTIVES FOR ACCELERATORS

What is OpenACC?

- A common directive programming model for today’s GPUs
  - Announced at SC11 conference
  - Offers portability between compilers
    - Drawn up by: NVIDIA, Cray, PGI, CAPS
    - Multiple compilers offer portability, debugging, permanence
  - Works for Fortran, C, (and maybe) C++
    - Standard available at www.OpenACC-standard.org
    - Initially implementations targeted at NVIDIA GPUs
- Current version: 1.0 (November 2011)
  - Version 2.0 RFC released at SC12 (expected 1Q13)
- Compiler support:
  - PGI Accelerator: released product in 2012
  - Cray CCE: released product in 2012
  - CAPS: released product in Q1 2012

The OpenACC™ API
QUICK REFERENCE GUIDE

The OpenACC Application Program Interface describes a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

Most OpenACC directives apply to the immediately following structured block or loop; a structured block is a single statement or a compound statement (C or C++) or a sequence of statements (Fortran) with a single entry point at the top and a single exit at the bottom.

CAPS
CRAY THE SUPERCOMPUTER COMPANY
NVIDIA
PGI

Version 1.0, November 2011
All four companies intend to work within OpenMP organization to merge OpenACC and create a common specification that extends OpenMP to support accelerators.

© 2011 OpenACC-standard.org all rights reserved.

NVIDIA CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

OpenACC Portability Goals

- Compiler Portability
  - Different compilers should support the same directives/pragmas and runtime library
  - Work is currently underway to standardize a compliance test suite.
- Device Portability
  - Designed to be high level enough to support any of today’s or tomorrow’s accelerators.
  - Eliminate the need for separate code branches for CPU and GPUs.
- Performance Portability
  - Since OpenACC only annotated the code, well-written code should perform well on either the CPU or GPU

OPENACC BASICS

Directive Syntax

- Fortran
  `!$acc directive [clause [,] clause] …]`
  Often paired with a matching end directive surrounding a structured code block
  `!$acc end directive`
- C
  `#pragma acc directive [clause [,] clause] …]`
  Often followed by a structured code block

Important Directives to Know

- `!$acc parallel`
  - Much like `!$omp parallel`, defines a region where loop iterations may be run in parallel
  - Compiler has the freedom to decompose this however it believes is profitable
- `!$acc kernels`
  - Similar to parallel, but loops within the kernels region will be independent kernels, rather than one large kernel.
  - Independent kernels and associated data transfers may be overlapped with other kernels

kernels: Your first OpenACC Directive

Each loop executed as a separate *kernel* on the GPU.

```
!$acc kernels
     do i=1,n
        a(i) = 0.0
        b(i) = 1.0
        c(i) = 2.0
     end do

     do i=1,n
        a(i) = b(i) + c(i)
     end do
!$acc end kernels
```

kernel 1

kernel 2

Kernel:
A parallel function that runs on the GPU

Important Directives to Know

- `!$acc data`
  - Defines regions where data may be left on the device
  - Useful for reducing PCIe transfers by creating temporary arrays or leaving data on device until needed
- `!$acc host_data`
  - Define a region in which host (CPU) arrays will be used, unless specified with `use_device()`
  - The `use_device()` clause exposes device pointer to the CPU
  - Useful for overlapping with CPU computation or calling library routines that expect device memory

Important Directives to Know

- `!$acc wait`
  - Synchronize with asynchronous activities.
  - May declare specific conditions or wait on all outstanding requests
- `!$acc update`
  - Update a host or device array within a data region
  - Allows updating parts of arrays
  - Frequently used around MPI

Important Directives to Know

- `!$acc loop`
  - Useful for optimizing how the compiler treats specific loops.
  - May be used to specify the decomposition of the work
  - May be used to collapse loop nests for additional parallelism
  - May be used to declare kernels as independent of each other

Important Terminology

- Gang
  - The highest level of parallelism, equivalent to CUDA Threadblock. (num_gangs => number of threadblocks in the grid)
  - A “gang” loop affects the “CUDA Grid”
- Worker
  - A member of the gang, equivalent to CUDA thread within a threadblock (num_workers => threadblock size)
  - A “worker” loop affects the “CUDA Threadblock”
- Vector
  - Tightest level of SIMT/SIMD/Vector parallelism, reoughly equivalent to CUDA warp or SIMD vector length (vector_length should be a multiple of warp size)
  - A ‘vector” loop affects the SIMT parallelism

- Declaring these on particular loops in your loop nest will affect the decomposition of the problem to the hardware

Other Directives

`async` clause	Declares that control should return to the CPU immediately. If an integer is passed to async, that integer can be passed as a handle to wait
`cache` construct	Cache data in software managed data cache (CUDA shared memory).
`declare` directive	Specify that data is to allocated in device memory for the duration of an implicit data region created during the execution of a subprogram.

Runtime Library Routines

Fortran

`use openacc` or
`#include "openacc_lib.h"`

C

`#include "openacc.h"`

```
acc_get_num_devices
acc_set_device_type
acc_get_device_type
acc_set_device_num
acc_get_device_num
acc_async_test
acc_async_test_all
```

```
acc_async_wait
acc_async_wait_all
acc_shutdown
acc_on_device
acc_malloc
acc_free
acc_init
```

Environment and Conditional Compilation

`ACC_DEVICE` *`device`*	Specifies which device type to connect to.
`ACC_DEVICE_NUM num`	Specifies which device number to connect to.
`_OPENACC`	Preprocessor directive for conditional compilation. Set to OpenACC version

USING OPENACC

Identify High-level, Rich Loop Nests

- Use your favorite profiling tool to identify hotspots at the highest level possible.
    - If there’s not enough concurrency to warrant CUDA, there’s not enough to warrant OpenACC either.

CrayPAT Loop-level profile

```
  100.0% | 117.646170 | 13549032.0 |Total
|-------------------------------------------------------
|  75.4% |  88.723495 | 13542013.0 |USER
||-------------------------------------------------------
||  10.7% |  12.589734 |  2592000.0 |parabola_
|||-------------------------------------------------------
3||   7.1% |   8.360290 |  1728000.0 |remap_.LOOPS
4||        |            |            | remap_
5||        |            |            |  ppmlr_
|||||-------------------------------------------------------
6|||||   3.2% |   3.708452 |   768000.0 |sweepx2_.LOOP.2.li.35
7|||||        |            |            | sweepx2_.LOOP.1.li.34
8|||||        |            |            |  sweepx2_.LOOPS
9|||||        |            |            |   sweepx2_
10||||        |            |            |    vhone_
6|||||   3.1% |   3.663423 |   768000.0 |sweepx1_.LOOP.2.li.35
7|||||        |            |            | sweepx1_.LOOP.1.li.34
8|||||        |            |            |  sweepx1_.LOOPS
9|||||        |            |            |   sweepx1_
10||||        |            |            |    vhone_
|||||=======================================================
3||   3.6% |   4.229443 |   864000.0 |ppmlr_
||||-------------------------------------------------------
4|||   1.6% |   1.880874 |   384000.0 |sweepx2_.LOOP.2.li.35
5|||        |            |            | sweepx2_.LOOP.1.li.34
6|||        |            |            |  sweepx2_.LOOPS
7|||        |            |            |   sweepx2_
8|||        |            |            |    vhone_
4|||   1.6% |   1.852820 |   384000.0 |sweepx1_.LOOP.2.li.35
5|||        |            |            | sweepx1_.LOOP.1.li.34
6|||        |            |            |  sweepx1_.LOOPS
7|||        |            |            |   sweepx1_
8|||        |            |            |    vhone_
|||=======================================================
```

Place OpenMP On High-level Loops

- Using OpenMP allows debugging issues of variable scoping, reductions, dependencies, etc. easily on the CPU
  - CPU toolset more mature
  - Can test anywhere
- Cray will soon be releasing Reveal, a product for scoping high-level loop structures.
- Who knows, this may actually speed-up your CPU code!

Focus on Vectorizing Low-Level Loops

- Although GPUs are not strictly vector processors, vector inner loops will benefit both CPUs and GPUs
  - Eliminate dependencies
  - Reduce striding
  - Remove invariant logic
  - …
- Compiler feedback is critical in this process

Finally, Add OpenACC

- Once High-Level parallelism with OpenMP and Low-Level vector parallelism is exposed and debugged, OpenACC is easy.

```
#ifdef _OPENACC
!$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w,&
!$acc&  svel0,xa, xa0, dx, dx0, dvol, f, flat,&
!$acc&  para,radius, theta, stheta) reduction(max:svel)
#else
!$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w,&
!$omp&  svel0,xa, xa0, dx, dx0, dvol, f, flat,&
!$omp&  para,radius, theta, stheta) reduction(max:svel)
#endif
```

Differences between OpenMP and OpenACC

- Things that are different between OpenMP and OpenACC
  - Cannot have CRITICAL REGION down callchain
  - Cannot have THREADPRIVATE
  - Vectorization is much more important
  - Cache/Memory Optimization much more important
  - No EQUIVALENCE
  - Private variables not necessarily initialized to zero.

```
#ifdef _OPENACC
!$acc parallel loop private( k,j,i,n,r, p, e, q, u, v, w,&
!$acc&  svel0,xa, xa0, dx, dx0, dvol, f, flat, para,radius,&
!$acc&  theta, stheta) reduction(max:svel)
#else
!$omp parallel do private( k,j,i,n,r, p, e, q, u, v, w, svel0,&
!$omp& xa, xa0, dx, dx0, dvol, f, flat, para,radius, &
!$omp& theta, stheta) reduction(max:svel)
#endif
```

But Now It Runs Slower!

- Every time I’ve gone through this process, the code is slower at this step than when I started.

- OpenACC is not automatic, you’ve still got work to do…
  - Improve data movement
  - Adjust loop decomposition
  - File bugs?

Optimizing Data Movement

- Compilers will be cautious with data movement a likely move more data that necessary.
  - If it’s left of ‘=‘, it will probably be copied from the device.
  - If it’s right of ‘=‘, it will probably be copied to the device.

- The CUDA Profiler can be used to measure data movement.
- The Cray Compiler also has the CRAY_ACC_DEBUG runtime environment variable, which will print useful information.
  - See `man intro_openacc` for details.

Optimizing Data Movement

- Step 1, place a data region around the simulation loop
  - Use this directive to declare data that needs to be copied in, copied out, or created resident on the device.
  - Use the present clause to declare places where the compiler may not realize the data is already on the device (within function calls, for example)
- Step 2, use an update directive to copy data between GPU and CPU inside the data region as necessary

Keep data on the accelerator with acc_data region

```
!$acc data copyin(cix,ci1,ci2,ci3,ci4,ci5,ci6,ci7,ci8,ci9,ci10,ci11,&
!$acc& ci12,ci13,ci14,r,b,uxyz,cell,rho,grad,index_max,index,&
!$acc& ciy,ciz,wet,np,streaming_sbuf1, &
!$acc&    streaming_sbuf1,streaming_sbuf2,streaming_sbuf4,streaming_sbuf5,&
!$acc&    streaming_sbuf7s,streaming_sbuf8s,streaming_sbuf9n,streaming_sbuf10s,&
!$acc&    streaming_sbuf11n,streaming_sbuf12n,streaming_sbuf13s,streaming_sbuf14n,&
!$acc&    streaming_sbuf7e,streaming_sbuf8w,streaming_sbuf9e,streaming_sbuf10e,&
!$acc&    streaming_sbuf11w,streaming_sbuf12e,streaming_sbuf13w,streaming_sbuf14w, &
!$acc&    streaming_rbuf1,streaming_rbuf2,streaming_rbuf4,streaming_rbuf5,&
!$acc&    streaming_rbuf7n,streaming_rbuf8n,streaming_rbuf9s,streaming_rbuf10n,&
!$acc&    streaming_rbuf11s,streaming_rbuf12s,streaming_rbuf13n,streaming_rbuf14s,&
!$acc&    streaming_rbuf7w,streaming_rbuf8e,streaming_rbuf9w,streaming_rbuf10w,&
!$acc&    streaming_rbuf11e,streaming_rbuf12w,streaming_rbuf13e,streaming_rbuf14e, &
!$acc&    send_e,send_w,send_n,send_s,recv_e,recv_w,recv_n,recv_s)
  do ii=1,ntimes
         o o o
      call set_boundary_macro_press2
      call set_boundary_micro_press
      call collisiona
      call collisionb
      call recolor
```

Update the Host for Communication

```
!$acc parallel_loop private(k,j,i)
  do j=0,local_ly-1
    do i=0,local_lx-1
      if (cell(i,j,0)==1) then
        grad (i,j,-1) = (1.0d0-wet)*db*press
      else
        grad (i,j,-1) = db*press
      end if
      grad (i,j,lz)    = grad(i,j,lz-1)
    end do
  end do
!$acc end parallel_loop
!$acc update host(grad)
  call mpi_barrier(mpi_comm_world,ierr)
  call grad_exchange
!$acc update device(grad)
```

But we would rather not send the entire grad array back – how about…

Packing the buffers on the accelerator

```
!$acc data present(grad,recv_w,recv_e,send_e,send_w,recv_n,&
!$acc&                       recv_s,send_n,send_s)
!$acc parallel_loop
      do k=-1,lz
        do j=-1,local_ly
          send_e(j,k) = grad(local_lx-1,j        ,k)
          send_w(j,k) = grad(0         ,j        ,k)
        end do
      end do
!$acc end parallel_loop
!$acc update host(send_e,send_w)
      call mpi_irecv(recv_w, bufsize(2),mpi_double_precision,w_id, &
           tag(25),mpi_comm_world,irequest_in(25),ierr)
          o  o  o
      call mpi_isend(send_w, bufsize(2),mpi_double_precision,w_id, &
           tag(26),& mpi_comm_world,irequest_out(26),ierr)
      call mpi_waitall(2,irequest_in(25),istatus_req,ierr)
      call mpi_waitall(2,irequest_out(25),istatus_req,ierr)
!$acc update device(recv_e,recv_w)
!$acc parallel
!$acc loop
      do k=-1,lz
        do j=-1,local_ly
          grad(local_lx  ,j         ,k) = recv_e(j,k)
          grad(-1        ,j         ,k) = recv_w(j,k)
```

Optimizing Kernels

- The compiler has freedom to schedule loops and kernels as it thinks is best, but the programmer can override this.
- First you must know how the work was decomposed.
    - Feedback from compiler at build time
    - Feedback from executable at runtime
    - CUDA Profiler

Adjusting Decomposition

- Adjust the number of gangs, workers, and or vector length on your `parallel` or `kernels` region
    - num_gangs, num_workers, vector_length
- Add `loop` directives to individual loop declaring them as gang, worker, or vector parallelism

Further Optimizing Kernels

- Use `loop collapse()` to merge loops and increase parallelism at particular levels
- Use compiler's existing directives regarding loop optimizations
  - Loop unrolling
  - Loop fusion/fission
  - Loop blocking
- Ensure appropriate data access patterns
  - Memory coalescing, bank conflicts, and striding are just as important with OpenACC as CUDA/OpenCL
  - This will likely help when using the CPU as well.

Interoperability

- OpenACC plays well with others; CUDA C, CUDA Fortran, Libraries
- If adding OpenACC to an existing CUDA code, the `deviceptr` data clause allows using existing data structures.
- If adding CUDA or a library call to an OpenACC code, use `host_data` and `use_device` to declare CPU or GPU memory use.

Interoperability Advice

- OpenACC provides a very straightforward way to manage data structures without needing 2 pointers (host & device), so use it at the top level.
- CUDA provides very close-to-the-metal control, so it can be used for very highly tuned kernels that may be called from OpenACC
- Compilers do complex tasks such as reductions very well, so let them.

OpenACC Homework

OpenACC Homework

- I have provided a starting point at http://users.nccs.gov/~larkin/OpenACC_Exercises.zip
- A simple Fortran matrix multiplication kernel has been provided, your assignment is to parallelize the kernel via OpenACC.
- Instructions have been provided for the Fortran kernel to add the directives in stages.
- Although the instructions use PGI, please feel free to try CCE as well. If you do, consider using the parallel directive in place of kernels.
- Please consider doing the option step 0 first.