C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators
Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
(Mode-4 : GPUs)

Lecture Topic:
GPU Computing – CUDA / OpenACC

Venue : CMSD, UoHYD ; Date : October 15-18, 2013

An Overview of OpenACC

Lecture Outline

Following topics will be discussed

- Part-I : An introduction to OpenACC
- Part-II : The OpenACC Pragmas
- Part-III: OpenACC Basic Examples
- Part-IV : Summary

Venue : CMSD, UoHYD ; Date : Oct 15-18, 2013

Source : NVIDIA & References given in the presentation

Introduction to OpenACC

OpenACC®

DIRECTIVES FOR ACCELERATORS

- OpenACC: http://www.openacc-standard.org/
- Source : NVIDIA, NVIDIA-PGI & References

3 Ways to Accelerate Applications

Applications

Libraries	Open ACC Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility

OpenACC Standard

NVIDIA. CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

Source : NVIDIA, PGI, CRAY, CAPS, & References given in the presentation

OpenACC : Open Prog. Stanadard for Par. Comp.

"OpenACC will enable programmers to easily develop portable applications that maximize the performance and power efficiency benefits of the hybrid CPU/GPU architecture of Titan."

--Buddy Bland, Titan Project Director, Oak Ridge National Lab

"OpenACC is a technically impressive initiative brought together by members of the OpenMP Working Group on Accelerators, as well as many others. We look forward to releasing a version of this proposal in the next release of OpenMP."

--Michael Wong, CEO OpenMP Directives Board

Source : NVIDIA & References given in the presentation

OpenACC : The standard for GPU Devices

Easy: Directives are the easy path to accelerate compute intensive applications

Open: OpenACC is an open GPU directives standard, making GPU programming straightforward and portable across parallel and multi-core processors

Powerful: GPU Directives allow complete access to the massive parallel power of a GPU

Source : NVIDIA & References given in the presentation

OpenACC : High-level, with low-level access

- Compiler directives to specify parallel regions in C, C++, Fortran
  - OpenACC compilers offload parallel regions from host to accelerator
  - Portable across OSes, host CPUs, accelerators, and compilers
- Create high-level heterogeneous programs
  - Without explicit accelerator initialization,
  - Without explicit data or program transfers between host and accelerator
- Programming model allows programmers to start simple
  - Enhance with additional guidance for compiler on loop mappings, data location, and other performance details
- Compatible with other GPU languages and libraries
  - Interoperate between CUDA C/Fortran and GPU libraries
  - e.g. CUFFT, CUBLAS, CUSPARSE, etc.

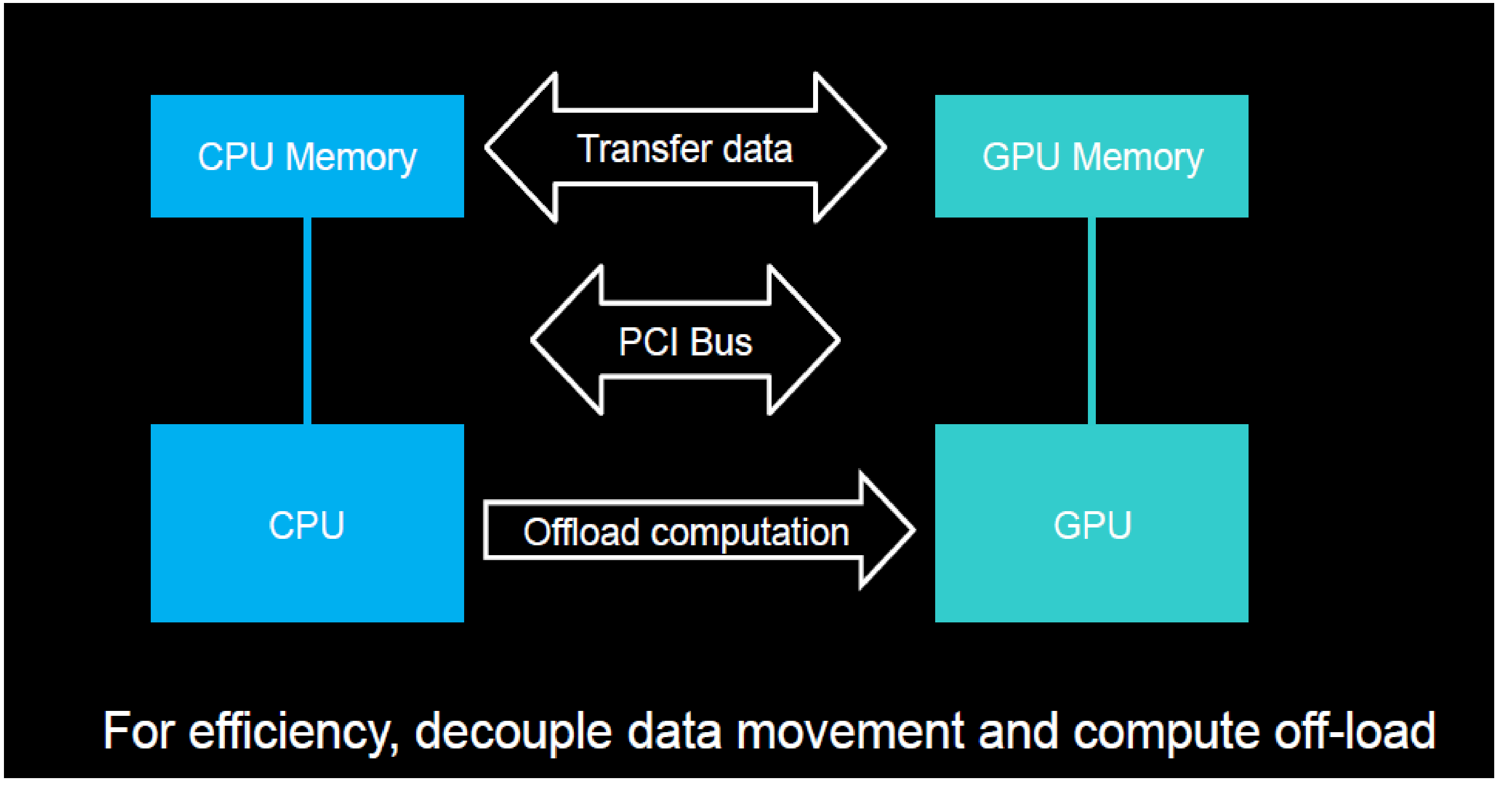
Source : NVIDIA & References given in the presentation

OpenACC : High-level, with low-level access

- ❖ Full OpenACC 1.0 Specification available online http://www.openacc-standard.org
- ❖ Quick reference card also available
- ❖ Beta implementations available now from PGI, Cray, and CAPS
- ❖ Information is given in References

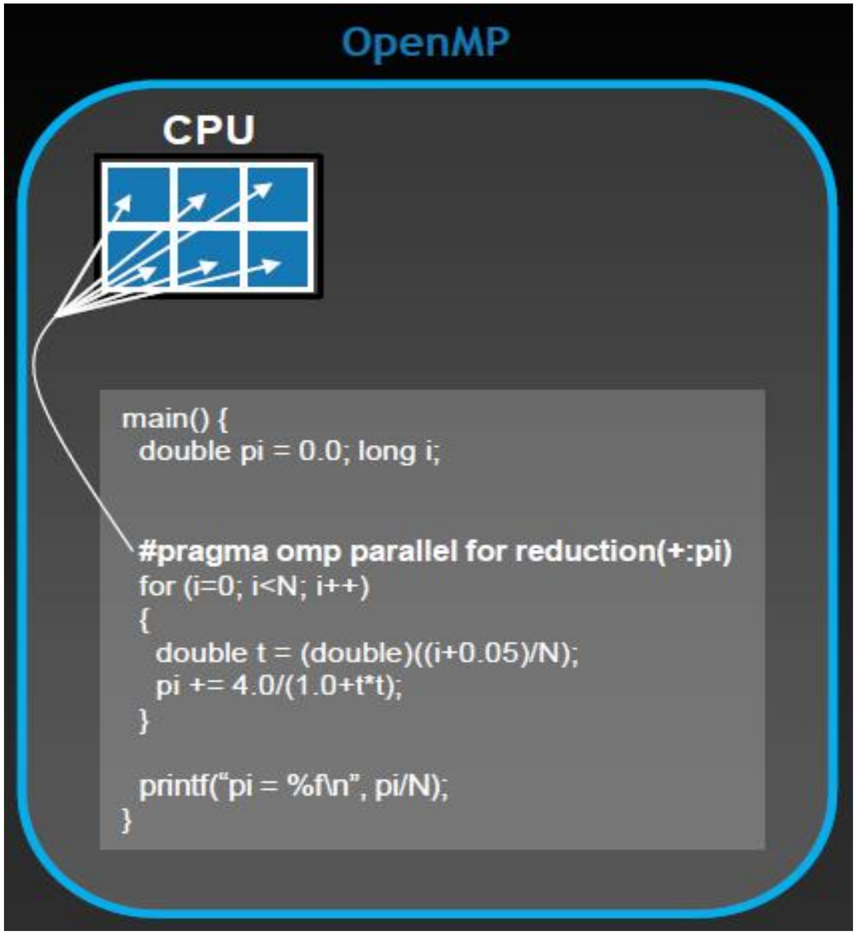
Source : NVIDIA & References given in the presentation

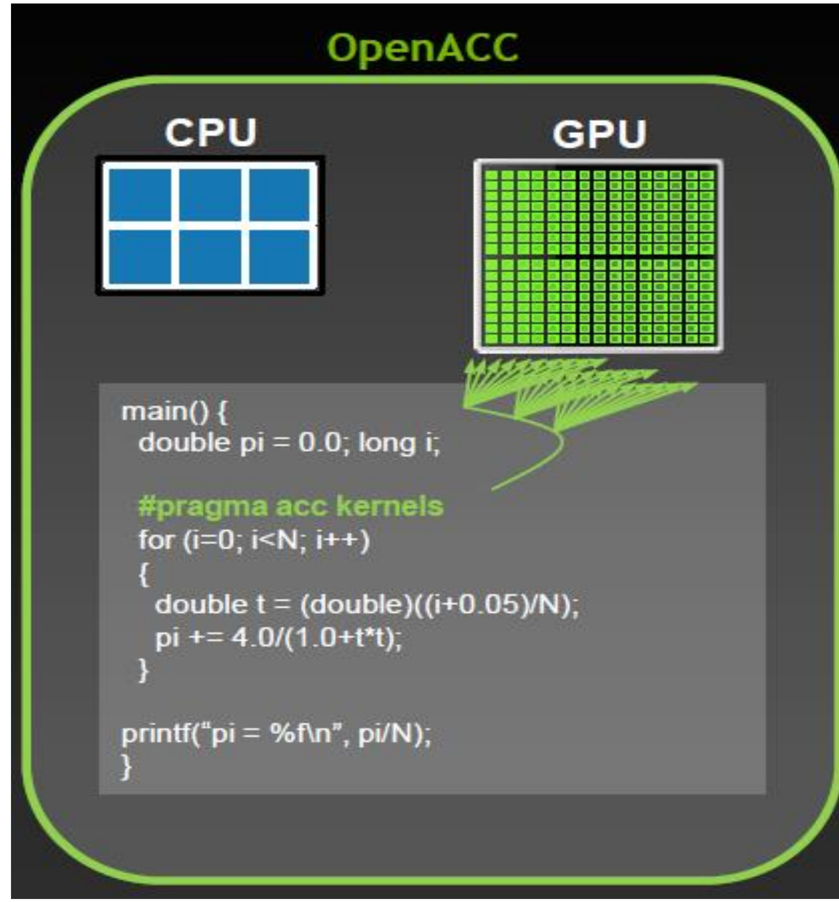
OpenACC Basic Concepts



Source : NVIDIA & References given in the presentation

Familiar to OpenMP Programmers





<u>Source</u> : NVIDIA & References given in the presentation

OpenACC Compile & Run

Compile and run

C:

pgcc –acc -ta=nvidia -Minfo=accel –o saxpy_acc saxpy.c

Fortran:

pgf90 –acc -ta=nvidia -Minfo=accel –o saxpy_acc saxpy.f90

Compiler output:

```
pgcc -acc -Minfo=accel -ta=nvidia -o saxpy_acc saxpy.c
saxpy:
8, Generating copyin(x[:n-1])
     Generating copy(y[:n-1])
     Generating compute capability 1.0 binary
     Generating compute capability 2.0 binary
9, Loop is parallelizable
     Accelerator kernel generated
      9, #pragma acc loop worker, vector(256) /* blockIdx.x threadIdx.x */
         CC 1.0 : 4 registers; 52 shared, 4 constant, 0 local memory bytes; 100% occupancy
         CC 2.0 : 8 registers; 4 shared, 64 constant, 0 local memory bytes; 100% occupancy
```

Source : NVIDIA & References given in the presentation

What is OpenACC?

- Accelerator programming API standard to program accelerators
  - Portable across operating systems and various types of host CPUs and GPU accelerators.
  - Allows parallel programmers to provide simple hints, known as "**directives**," to the compiler, identifying which areas of code to accelerate, without requiring programmers to modify or adapt the underlying code itself.
  - Aimed at incremental development of accelerator code
- Effort driven by vendors with the input from users/ applications

Source : NVIDIA & References given in the presentation

OpenACC Vendor Support

- The current vendors support OpenACC are: Cray: High-Level GPU directives
  - **PGI: PGI accelerator directives**
  - **CAPS Enterprise: HMPP**
  - **NVIDIA: CUDA, OpenCL**
  - **Others: As this defacto standard gains traction**
- Strong interaction with the OpenMP accelerator subcomittee with input from other institutions

Source : NVIDIA & References given in the presentation

Impact of OpenACC

- Phase 1: First Standardization of High-Level GPU directives. [Short-term, Mid-term]
  - Heavily influenced by NVIDIA hardware.

- Phase 2: Experiences from OpenACC will drive the effort of OpenMP for Accelerators
  - More general solution
  - Might take years to develop
  - Better interoperability with OpenMP

Source : NVIDIA & References given in the presentation

Overview of the OpenACC directives

- Directives facilitate code development for accelerators
- Provide the functionality to:
  - Initiate accelerator startup/shutdown
  - Manage data or program transfers between host (CPU) and accelerator
  - Scope data between accelerator and host (CPU)
  - Manage the work between the accelerator and host.
  - Map computations (loops) onto accelerators
  - Fine-tune code for performance

Source : NVIDIA & References given in the presentation

Execution Model

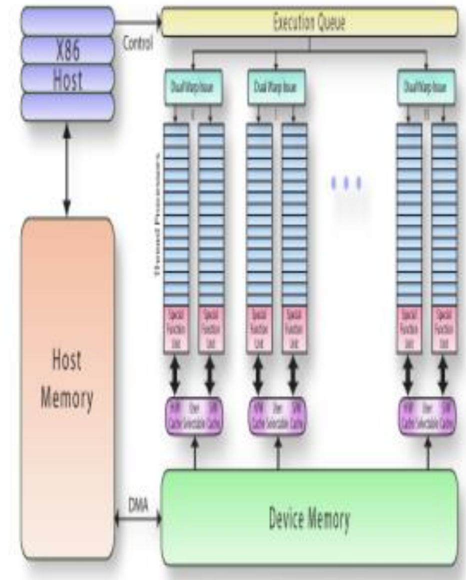
- Bulk of computations executed in CPU, compute intensive regions offloaded to accelerators
- Accelerators execute parallel regions:
  - Use work-sharing and kernel directives
  - Specification of coarse and fine grain parallelization
- The **host** is responsible for
  - Allocation of memory in accelerator
  - Initiating data transfer
  - Sending the code to the accelerator
  - Waiting for completion
  - Transfer the results back to host
  - De-allocating memory
  - Queue sequences of operations executed by the device

Source : NVIDIA & References given in the presentation

Execution Model

❖ **Parallelism:**

- Support coarse-grain parallelism
  - Fully parallel across execution units
  - Limited synchronizations across
  - coarse-grain parallelism
- Support for fine-grain parallelism
  - Often implemented as SIMD
  - Vector operations



- Programmer need to understand the differences between them.
  - Efficiently map parallelism to accelerator
  - Understand synchronizations available
- Compiler may detect data hazards
  - Does not guarantee correctness of the code

Source : NVIDIA & References given in the presentation

Memory Model

- Host + Accelerator memory may have completely separate memories
  - **Host may not be able to read/write device memory that is not mapped to a shared virtual addressed.**
- All data transfers must be initiated by host
  - **Typically using direct memory accesses (DMAs)**
- Data movement is implicit and managed by compiler
- Device may implement weak consistency memory model
  - **Among different execution units**
  - **Within execution unit: memory coherency guaranteed by barrier**

Source : NVIDIA & References given in the presentation

Memory Model (2)

- Programmer must be aware of:
  - Memory bandwidth affects compute intensity
  - Limited device memory
  - Assumptions about cache:
    - **Accelerators may have software or hardware managed cache**
    - **May be limited to read only data**
- Caches are managed by the compiler with hints by the programmer
- Compiler may **auto-scope** variables based on static information or enforce runtime checks.

Source : NVIDIA & References given in the presentation

Categories of OpenACC APIs

- Accelerator Parallel Region / Kernels Directives
- Loop Directives
- Data Declaration Directives
- Data Regions Directives
- Cache directives
- Wait / update directives
- Runtime Library Routines
- Environment variables

Source : NVIDIA & References given in the presentation

Directives Format

- **C/C++:**

#pragma acc *directive-name [clause [,clause]…] new-line*

- **Fortran:**

!$acc *directive-name [clause [, clause]…]*

c$acc *directive-name [clause [, clause]…]*

$acc *directive-name [clause [, clause]…]

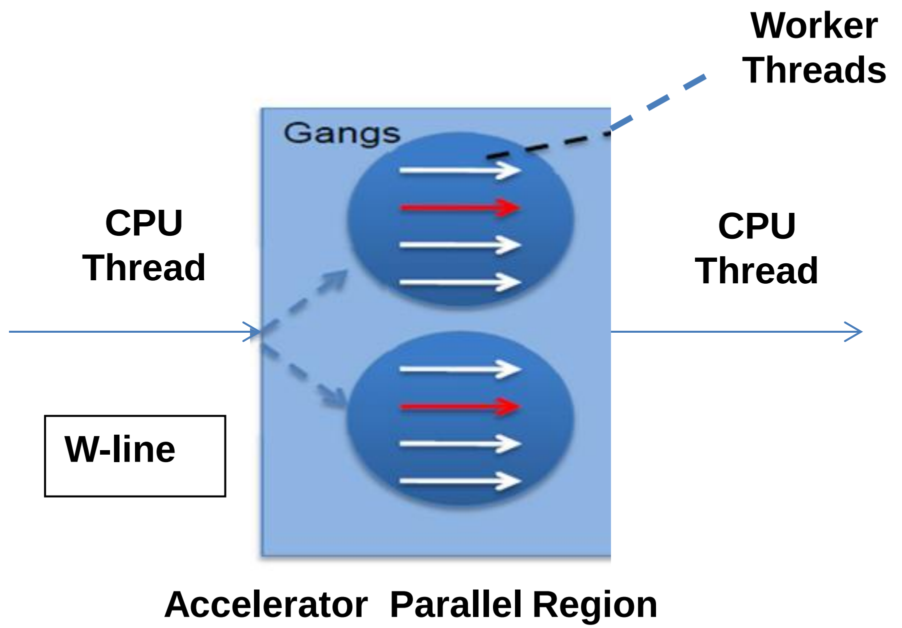
Source : NVIDIA & References given in the presentation

OpenACC Parallel Directive

- Starts parallel execution on accelerator
- Specified by:
  - **#pragma acc parallel [clause [,clause]…] new-line structured block**
- When encountered:
  - Gangs of workers threads are created to execute on accelerator
  - One worker in each gang begins executing the code following the structured block
  - Number of gangs/workers remains constant in parallel region

Source : NVIDIA & References given in the presentation

OpenACC Parallel Directive



<u>Source</u> : NVIDIA & References given in the presentation

OpenACC Parallel Directive (2)

❖ The clauses for the ***!$acc parallel*** directive are:

- 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)

Source : NVIDIA & References given in the presentation

OpenACC Parallel Directive (3)

- The clauses for the **!$acc parallel** directive are:
  - present (list)
  - present_or_copy (list)
  - present_or_copyin (list)
  - present_or_copyout (list)
  - present_or_create (list)
  - deviceprt (list)
- If **async** is not present, there is an implicit barrier at the end of accelerator parallel region.
- **present_or_copy** default for aggregate types (arrays)
- **private or copy** default for scalar variables

Source : NVIDIA & References given in the presentation

OpenACC Kernel Directive

- Defines a region of a program that is to be compiled into a sequence of kernels for execution on the accelerator
- Each loop nest will be a different kernel
- Kernels launched in order in device
- Specified by:
  - ***#pragma acc kernels [clause [,clause]…] new-line***
    structured block

Source : NVIDIA & References given in the presentation

OpenACC Kernel Directive (2)

- Kernels directive may not contain nested parallel or kernel directive
- Configuration of gangs and worker thread may be different for each kernel
- The clauses for the ***!$acc kernels*** directive are: if(condition)
  - async [(scalar-integer-expression)]
  - copy (list)
  - copyin (list)
  - copyout (list)
  - create (list)
  - private (list)
  - firstprivate (list)

Source : NVIDIA & References given in the presentation

OpenACC Kernel Directive (3)

- The clauses for the ***!$acc kernels*** directive are: present (list)
  - present_or_copy (list)
  - present_or_copyin (list)
  - present_or_copyout (list)
  - present_or_create (list)
  - deviceprt (list)
- If **async** is present, kernels or parallel region will execute asynchronous on accelerator
- **present_or_copy** default for aggregate types (arrays)
- **private or copy** default for scalar variables

Source : NVIDIA & References given in the presentation

OpenACC Parallel/Kernel Clauses

- **if clause**
  - Optional clause to decide if code should be executed on accelerator or host
- **async clause**
  - Specifies that a parallel accelerator or kernels regions should be executed asynchronously
  - The host will evaluate the integer expression of the async clause to test or wait for completion with the wait directive
- **num_gangs clause**
  - Specifies the number of gangs that will be executed in the accelerator parallel region
- **num_workers clause**
  - Specifies the number of workers within each gang for a accelerator parallel region

Source : NVIDIA & References given in the presentation

OpenACC Parallel/Kernel Clauses

- **vector_length clause**
  - Specifies the vector length to use for the vector or SIMD operations within each worker of a gang
- **private clause**
  - A copy of each item on the list will be created for each gang
- **firstprivate clause**
  - A copy of each item on the list will be created for each gang and initialized with the value of the item in the host
- **reduction clause**
  - Specifies a reduction operation to be perform across gangs using a private copy for each gang.
  - Support for: +, *, max, min, &, |, &&, ||
  - Other operators available in Fortran: .neqv., .eqv.

Source : NVIDIA & References given in the presentation

OpenACC Data Directive

- The data construct defines scalars, arrays and subarrays to be allocated in the accelerator memory for the duration of the region.
- Can be used to control if data should be copied-in or out from the host
- Specified by:
  - *#pragma acc data [clause [,clause]…] new-line structured block*

Source : NVIDIA & References given in the presentation

OpenACC Data Directive

- The clauses for the ***!$acc data*** directive are:
  - if(condition)
  - copy (list)
  - copyin (list)
  - copyout (list)
  - create (list)
  - present (list)
  - present_or_copy (list)
  - present_or_copyin (list)
  - present_or_copyout (list)
  - present_or_create (list)
  - deviceptr (list)

Source : NVIDIA & References given in the presentation

OpenACC Data Directive

- ❖ 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

Source : NVIDIA , PGI & References given in the presentation

OpenACC Data Directive (2)

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)

Source : NVIDIA & References given in the presentation

OpenACC Data Directive (3)

- 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)

Source : NVIDIA & References given in the presentation

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**

Source : NVIDIA & References given in the presentation

OpenACC Loop Directive (2)

- 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

Source : NVIDIA & References given in the presentation

OpenACC Loop Clauses

- 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

Source : NVIDIA & References given in the presentation

OpenACC Loop Clauses

- 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, worker or vector

Source : NVIDIA & References given in the presentation

OpenACC Cache Directive

- Specifies array elements or subarrays that should be fetched into the highest level of the cache for the body of the loop.

- Specified by:
  - **#pragma acc cache(list) new-line**

Source : NVIDIA & References given in the presentation

OpenACC Combined Directive

- Some directives can be combined into a single one

- Combined directives are specified by:

  - ***#pragma acc parallel loop [clause [,clause]…] new-line***
    for loop
  - ***#pragma acc kernels loop [clause [,clause]…] new-line***
    for loop

Source : NVIDIA & References given in the presentation

OpenACC Declare Directive

- Used in the variable declaration section of program to specify that a variable should be allocated, copy-in/out in an implicit data region of a function, subroutine or program .

- If specified within a Fortran Module, the implicit data region is valid for the whole program.

- Specified by:
  - ***#pragma acc declare [clause [,clause]…] new-line***

Source : NVIDIA & References given in the presentation

OpenACC Declare Directive (2)

- The clauses for the **!$acc data** directive are: copy (list)
  - copyin (list)
  - copyout (list)
  - create (list)
  - present (list)
  - present_or_copy (list)
  - present_or_copyin (list)
  - present_or_copyout (list)
  - present_or_create (list)
  - deviceptr (list)
  - device_resident (list)

Source : NVIDIA & References given in the presentation

OpenACC Update Directive

- Used within a data region to update / synchronize the values of the arrays on both the host or accelerator

- Specified by:
  - #pragma acc update [clause [,clause]…] new-line

- The clauses for the **!$acc update** directive are:
  - host (list)
  - device (list)
  - if (condition)
  - async [(scalar-integer-expression)]

Source : NVIDIA & References given in the presentation

OpenACC Wait Directive

- It causes the program to wait for completion of an asynchronous activity such as an accelerator parallel, kernel region or update directive
- Specified by:
  - #pragma acc wait [(scalar-integer-expression)] new-line
- It will test and evaluate the integer expression for completion
- If no argument is specified, the host process will wait until all asynchronous activities have completed
- Can be specified per CPU/Thread basis.

Source : NVIDIA & References given in the presentation

OpenACC runtime calls

- int acc_get_num_devices(acc_device_t)
- void acc_set_device_type(acc_device_t)
- acc_device_t acc_get_device_type()
- acc_set_device_num(int, acc_device_t)
- int acc_get_device_num(acc_device_t)
- int acc_async_test(int)
- int acc_async_test_all()
- void acc_async_wait(int)
- void acc_async_wait_all()
- void acc_init(acc_device_t)
- void acc_shutdown (acc_device_t)
- int acc_on_device(acc_device_t)
- void* acc_malloc(size_t)
- void acc_free(void*)

setenv ACC_DEVICE_TYPE
NVIDIA setenv
ACC_DEVUCE_NUM 1
Environment Variables

Source : NVIDIA & References given in the presentation

OpenACC runtime calls

- Some vendors will provide implementations of OpenACC at the end of this year.
- The OpenACC Cray implementation is available
- Use OpenACC as the standard GPU programming directives
- applications users are starting to use
- Visit References for runtime calls

Source : NVIDIA & References given in the presentation

References Acknowledgement s

References :

1. OpenACC: www.openacc-standard.org/
2. GPU Computing with OpenACC Directives Presented by John Urbanic, Pittsburgh Supercomputing Center Authored by Mark Harris NVIDIA Corporation
3. Cray OpenACC http://www.openacc-standard.org/content/cray-even
4. CAPS – OpenACC : http://www.caps-entreprise.com/index.php
5. http://www.caps-entreprise.com/fr/page/index.php?id=148&p_p=36 CAPS OpenACC COMPILER
6. PGI OpenACC : www.pgroup.com/resources/accel.htm
7. http://www.opengpu.net/EN/attachments/154_HiPEAC2012_OpenGPU_nVidia.pdf OPENACC DIRECTIVES FOR ACCELERATORS –NVIDIA
8. http://www.pgroup.com/doc/openACC_gs.pdf PGI OpenACC Compilers Getting Started Guide Version 12.3
9. *Introduction to OpenACC;* Oscar Hernandez, Richard Graham, Computer Science and Mathematics (CSM), Application Performance Tools Group,Oak Ridge National Laboratories, U.S Dept. of Energy
10. *GPU Programming with CUDA and OpenACC*; Axel Koehler – NVIDIA
11. http://www.nvidia.com/docs/IO/116711/OpenACC-API.pdf The OpenACC™ API QUICK RE FEREN CE GUIDE
12. http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf The OpenACC™ Application Programming Interface Version 1.0 November, 2011

Source : NVIDIA & References given in the presentation

Questions

Thank You

- **Any Questions?**
- OpenACC: http://www.openacc-standard.org/
- Source : NVIDIA & References

Source : NVIDIA & References given in the presentation