CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

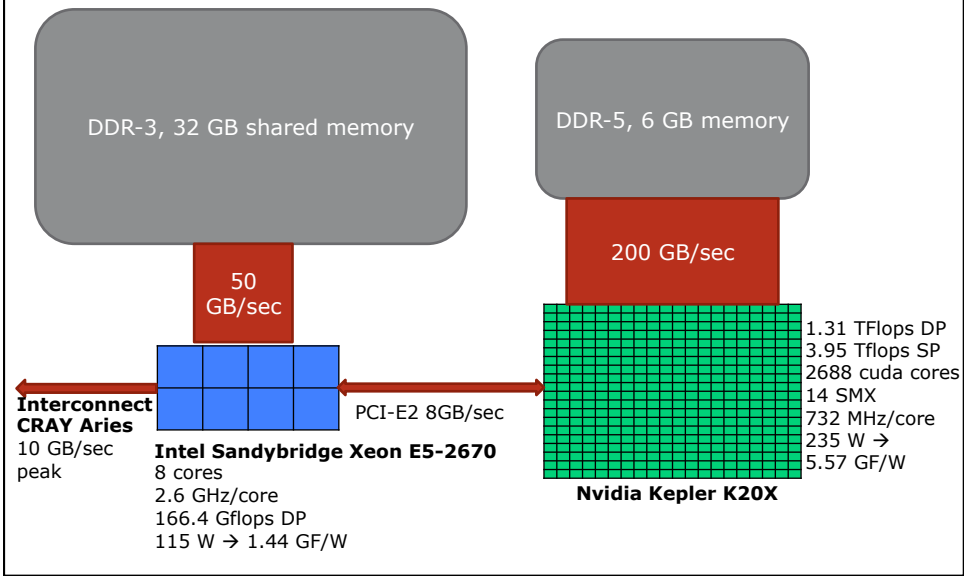
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction to OpenACC

Claudio Gheller
cgheller@cscs.ch

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Processor architecture (Piz Daint)



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

A few relevant terms

- **Thread → single computational unit (maps to CUDA core)**
- **Vector → SIMD thread set**
- **Warp → pool of threads sharing the same instructions and the same shared memory**
- **Shared memory → fast cache memory accessible from all the threads in a pool (=warp)**
- **Threadblock (or simply Block) → pool of warps**
- **Global memory → GPU "RAM" memory**

For the moment, just names. We will give them significance in the next hours…

3

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

GPU's important facts (1)

- You need **a lot of parallel tasks** (i.e. loop iterations) to keep GPU busy
  - Each parallel task maps to a thread in a threadblock
  - You need a lot of threadblocks per streaming multiprocessor (SM) to **hide memory latency**
  - Not just 2688 parallel tasks, but 10^4 to 10^6 or more
  - In a loop-based code, treat iterations as tasks
- Your inner loop must **vectorise** (at least with vector length of **32**)
  - So we can use all 32 threads in a warp with **shared instruction stream**

4

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

GPU's important facts (2)

- Memory should be accessed in the correct order
  - Global memory access is done with **(sequential) vector loads**
  - For good performance, want **as few** of these as possible
  - so all the threads in warp should collectively load a contiguous block of memory at the same point in the instruction stream: this is known as "**coalesced memory access**"
  - So vectorised loop index should be fastest-moving index of each array

5

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

GPU's important facts (3)

- No internal mechanism for synchronising between threadblocks
  - **Synchronisation must be handled by host**
    - even though all threadblocks share same global memory
  - Fortunately launching kernels is cheap
    - GPU threadteams are "lightweight"

- Data transfers between CPU and GPU are very expensive
  - You need to address "**data locality**"
  - Keeping data in the **right place for as long as it is needed** is crucial
  - You should port as much of the application as possible
    - This probably means **porting more than you expected**

6

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

GPU's important facts (4)

- **GPUs can give very good performance**
- but you need to be aware of the underlying architecture
- porting a real application to GPU(s) requires some hard work
  - Amdahl says you need to port a lot of the profile to see a speed-up
    - bad news: to see 10x speedup, need to port at least 90% of the application profile
    - good news: if profile very peaked, 90% of time may be spent in, say, 40% of code
  - even before you worry about the costs of data transfers

7

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

GPU programming models

API currently available

- **CUDA**
  - NVIDIA product, best performance, low portability
- **OpenCL**
  - Open standard, API similar to CUDA, high portability
- **OpenACC**
  - **Directive based, our focus**
- **Libraries**
  - MAGMA, cuFFT, Thrust, CULA, cuBLAS

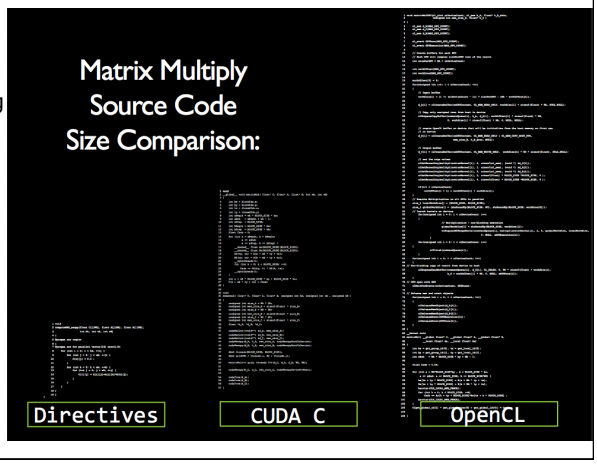
8

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Why OpenACC?

CUDA and OpenCL are quite low-level and closely coupled to the GPU, but:

- User needs to rewrite kernels in specialist language:
  - Hard to write and debug
  - Hard to port to new accelerator
- Multiple versions of kernels in codebase
  - Hard to add new functionality



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

OpenACC: http://www.openacc-standard.org/

Accelerator programming API standard

- **Allows parallel programmers to provide simple hints, known as "directives" to the compiler, identifying which areas of code to accelerate**
- **Aimed at incremental development of accelerator code**
- **Supported by various vendors**
  - Cray
  - PGI
  - CAPS
  - gcc: work started in late 2013

10

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

OpenACC: directive-based programming

+ **Based on original source code (Fortran, C, C++)**
  + Easier to maintain/port/extend code
  + Users with OpenMP experience find it a familiar programming model
  + Compiler handles repetitive coding (cudaMalloc, cudaMemcpy...)
  + Compiler handles default scheduling; user tunes only where needed

– **Possible performance sacrifice**

11

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

OpenACC directives

- **Directives facilitate code development for accelerators**
- **Provide the functionality to:**
  - Manage data transfers between host (CPU) and accelerator
  - Manage the work between the accelerator and host.
  - Manage computations (loops) onto accelerators
  - Tune code for performance

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Adding directives and building

Modify original source code with directives

- Non-executable statements (comments, pragmas)
  - Can be **ignored** by non-accelerating compiler
  - Cray Compiler (CCE) **-hnoacc** also suppresses compilation
- Sentinel: **acc**
  - **C/C++**:

```
/* C/C++ example */
#pragma acc *
{structured block}
```

  - **Fortran**:

```
! Fortran example
!$acc *
<structured block>
!$acc end *
```

- Continuation to extra lines allowed
  - **C/C++: ** (at end of line to be continued)
  - **Fortran:**
    - Fixed form: **c$acc&** or **!$acc&** on continuation line
    - Free form: **&** at end of line to be continued
      - continuation lines can start with either **!$acc** or **!$acc&**

13

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Conditional compilation

- **In theory, OpenACC code should be identical to CPU**
- only difference are the directives (i.e. comments)

- **In practise, the final code will be (possibly highly) different**
- Substantial code refacotring is needed:
  - usually for performance reasons
  - usually better OpenACC code is better CPU code

14

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Execution Model

- **CPU is the "driver"**
- **Computing intensive regions offloaded to accelerators**
- **Accelerators execute parallel and kernel regions**
- **The host is responsible for:**
  - Allocation de-allocation of memory in accelerator
  - Data transfers (from-to the accelerator)
  - Sending the code to the accelerator
  - Waiting for completion
  - Queue sequences of operations executed by the device
- **Work is scheduled in Gangs, Workers, Vectors (mapping to CUDA blocks, warps, threads)**

15

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

OpenACC main directives classes

- **Accelerator Parallel Region / Kernels Directives**
  - Parallel region (**!$acc parallel [clause]**): part of the code executed in parallel on the accelerator keeping gangs, workers and vector constant
  - Kernel (**!$acc kernel [clause]**): region of a program that is to be compiled into a sequence of kernels for execution on the accelerator. When program encounters a kernels construct, it will launch sequence of kernels in order on the device. Number and configuration of gangs of workers and vector length may be different for each kernel.
- **Loop Directives**
- **Data directives**
- **Synchronization directives**
- **Cache directives**

16

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Memory Model

- **Host and Accelerator have completely separate memories (interconnected via PCI Express bus – SLOW!!!)**
- **All data transfers are initiated by host**
- **Data movement is implicit and managed by compiler**
- **Programmer must be aware of:**
  - Moving data from CPU to accelerator and back is extremely slow
  - Memory access affects compute intensity
  - Limited device memory

17

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

A first example

```
PROGRAM main
  INTEGER :: a(N)
  <stuff>
!$acc parallel loop
  DO i = 1,N
   a(i) = i
  ENDDO
!$acc end parallel loop
!$acc parallel loop
  DO i = 1,N
   a(i) = 2*a(i)
  ENDDO
!$acc end parallel loop
  <stuff>
END PROGRAM main
```

- Two accelerator parallel regions
  - Compiler creates two kernels
    - Loop iterations automatically divided across GPU threads
  - First kernel initialises array
    - Compiler will determine a is write-only
  - Second kernel updates array
    - Compiler will determine a is read-write
  - Breaking parallel region=barrier
    - No barrier directive

Note: Same code can still be compiled for the CPU

18

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Data scoping

- **In a serial code (or pure MPI), there are no complications**
- **In a thread-parallel code (OpenACC, OpenMP etc.) things are more complicated:**
  - Some data will be the same for each thread (e.g. the main data array)
    - The threads can (and usually should) share a single copy of this data
  - Some data will be different (e.g. loop index values)
    - Each thread will need it's own private copy of this data
- **Data scoping arranges this. It is done:**
  - automatically (by the compiler) or explicitly (by the programmer)
- **If the data scoping is incorrect, we get:**
  - incorrect (and inconsistent) answers ("race conditions"), and/or
  - a memory footprint that is too large to run

19

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Data scoping (2)

- In OpenMP, we have the following data clauses
  - shared, private, firstprivate
- In OpenACC
  - private, firstprivate are just the same
- **Shared variables are more complicated** in OpenACC because we also need to think about data movements to/from GPU

OpenACC parallel regions:

- scalars and loop index variables are private by default
- arrays are shared by default
  - the compiler chooses which shared-type: copyin, copyout, etc.
- explicit data clauses over-ride automatic scoping decisions
- You can also add the default(none) clause
  - you have to do everything explicitly (or you get a compiler error)

20

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

GPU data management

When creating an accelerated region, data management can be explicitly specified through the following data clauses

- a comma-separated collection of variable names, array names, or subarray specifications. Compiler allocates and manage a copy of variable or array in device memory, creating a visible device copy of variable or array.
- **Data clauses:**

copy
copyin
copyout
create
present
present_or_copy
present_or_copyin
present_or_copyout
present_or_create
if
deviceptr

Example:
#pragma acc parallel loop copyin(a, b), copyout (c)

!$acc parallel loop copyin(a, b), copyout (c)

21

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Copy

copy

- To declare that variables, arrays or subarrays **have values in host memory that need to be copied to device memory, and are assigned values on accelerator that need to be copied back to host.** Data copied to device memory before entry to region, and data copied back to host memory when region complete.

22

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Copyin

copyin

- To declare that variables, arrays or subarrays have values in host memory that need to be copied to device memory. **Implies that data need not be copied back from device memory to the host memory.** Data copied to device memory upon entry to region. If a subarray specified, then only that subarray needs to be copied.

23

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Copyout

copyout

- To declare that variables, arrays or subarrays have values in device memory that **need to be copied back to host memory at end of accelerator region. Implies that data need not be copied to device memory from host memory.** Data copied back to host memory upon exit from region.
  If a subarray specified, then only that subarray needs to be copied.

24

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Create and Present

Create

- used to declare that the variables, arrays, subarrays or common blocks in the *var-list* need to be allocated (created) in the device memory **but the values in the local memory are not needed on the accelerator, and any values computed and assigned on the accelerator are not needed back in local memory**. On a data construct or compute construct, the data is allocated in device memory upon entry to the region, and deallocated upon exit from the region. **No data in this clause will be copied between the local and device memories**.

Present

- used to declare that the variables or arrays in the *var-list* are **already present in device memory** due to data regions that contain this region, such as data constructs within procedures that call the procedure containing this construct, or an enter data directive or runtime API routine called before this routine.

25

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Present_or_copy (in, out, create)

present_or_copy (**pcopy**)

- used to tell the implementation to test whether each of the variables or arrays on the *var-list* is already present in the accelerator memory, as with the **present** clause.
- If the data is already present, **the program behaves as with the present clause**. No new device memory will be allocated and no data will be moved to or from the device memory.
- If the data is not present, **the program behaves as with the copy (copyin, copyout, create) clause**.

26

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

If and Deviceptr

if clause is optional; **When an if clause appears, the program will conditionally allocate memory on, and move data to and/or from the device**. When the *condition* in the **if** clause evaluates to zero in C or C++, or **.false.** in Fortran, no device memory will be allocated, and no data will be moved. When the *condition* evaluates to nonzero in C or C++, or **.true.** in Fortran, the data will be allocated and moved as specified. At most one **if** clause may appear.

deviceptr
This clause is used to declare that the pointers in the *var-list* are **device pointers**, so the data need not be allocated or moved between the host and device for this pointer.

27

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Data clauses in action

```
PROGRAM main
  INTEGER :: a(N)
  <stuff>
!$acc parallel loop copyout(a)
  DO i = 1,N
   a(i) = i
  ENDDO
!$acc end parallel loop
!$acc parallel loop copy(a)
  DO i = 1,N
   a(i) = 2*a(i)
  ENDDO
!$acc end parallel loop
  <stuff>
END PROGRAM main
```

- **We could choose to make the data movements explicit**
  - maybe because we want to
  - maybe also use default(none) clause
  - or maybe compiler is overcautious

Note:
- Array a is needlessly moved from/to GPU between kernels
  - This will have a big impact on performance

28

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Data regions

- **Data regions allow data to remain on the accelerator**
  - e.g. for processing by multiple accelerator kernels
  - specified arrays only move at start/end of data region
- **Data regions only label a region of code**
  - they do not define or start any sort of parallel execution
  - just specify GPU memory allocation and data transfers
  - can contain host code, nested data regions and/or device kernels
- **Be careful:**
  - Inside data region we have two copies of each of the specified arrays
  - These only synchronise at the start/end of the data region
    - and only following the directions of the explicit data clauses

29

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Defining data regions

Two ways to define data regions:

- Structured data regions (procedural programming):
  - Fortran: **!$acc data [data-clauses] … !$acc end data**
  - C/C++: **#pragma acc data [date-clauses] {…}**
- Unstructured data regions (object oriented programming):
  - Fortran: **!$acc enter data [data-clauses] …**
    !$acc exit data [data-clauses]
  - C/C++: **#pragma enter data [data-clauses] …**
    #pragma exit data [data-clauses]

30

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Sharing GPU data between subprograms

```
PROGRAM main
  INTEGER :: a(N)
  <stuff>
!$acc data copyout(a)
!$acc parallel loop
  DO i = 1,N
   a(i) = i
  ENDDO
!$acc end parallel loop
  CALL double_array(a)
!$acc end data
  <stuff>
END PROGRAM main
```

```
SUBROUTINE double_array(b)
  INTEGER :: b(N)
!$acc parallel loop present(b)
  DO i = 1,N
   b(i) = double_scalar(b(i))
  ENDDO
!$acc end parallel loop
END SUBROUTINE double_array
```

```
INTEGER FUNCTION double_scalar(c)
  INTEGER :: c
  double_scalar = 2*c
END FUNCTION double_scalar
```

- **present clause uses GPU version of b without data copy**
- Original calltree structure of program can be preserved

- **One kernel is now in subroutine (maybe in separate file)**
  - OpenACC 1.0: function calls inside parallel regions required inlining
  - OpenACC 2.0: compilers support nested parallelism

31

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Data scoping recap

- **parallel regions:**
- scalars and loop index variables are **private** by default
- arrays are **shared** by default
  - the compiler chooses which shared-type: copyin, copyout, etc.
- explicit data clauses over-ride automatic scoping decisions

- **data regions:**
- only shared-type scoping clauses are allowed
- there is **NO** default/automatic scoping
  - un-scoped variables on data regions
    - will be scoped at each of the enclosed **parallel** region automatically, unless the programmer does this explicitly
    - this probably leads to unwanted data-movements or large arrays
  - using data region scoping in enclosed **parallel** regions:
    - same routine: omit scoping clauses on enclosed parallel directives
    - different routine: use **present** clause on enclosed parallel directives

32

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Summary

- **Compute regions**
  - created using parallel or kernels directives
- **Data regions**
  - created using data or enter/exit data directives
- **Data clauses are applied to:**
  - accelerate loopnests: parallel and kernels directives
    - here they over-ride relevant parts of the automatic compiler analysis
    - you can switch off all automatic scoping with default(none)
  - data regions: data directive (plus enter/exit data in OpenACC v2)
    - Note there is no automatic scoping in data regions (arrays or scalars)
  - Shared clauses (copy, copyin, copyout, create)
    - supply list of scalars, arrays (or array sections)
  - Private clauses (private, firstprivate, reduction)
    - only apply to accelerated loopnests (parallel and kernels directives)
  - present clause (used for nested data/compute regions)

33

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Summary (cont.ed)

- **The emphasis in this introduction has been on**
- explaining data scoping and using data regions

- **Why?**
- **because optimising data movements is far more important than tuning**
  - minimising data transfers typically speeds up GPU execution by 10x-100x
  - performance tuning maybe gains you 2x-3x
    - and you can't start to get this until you first stop useless data movements

34