Parallelware Tool Workshop

Learning parallelization of real applications from the ground-up

Manuel Arenaz | October 17, 2019

NERSC

appentra
make code parallel

OpenACC
More Science, Less Programming

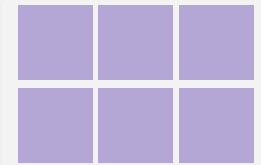
OpenMP
Enabling HPC since 1997

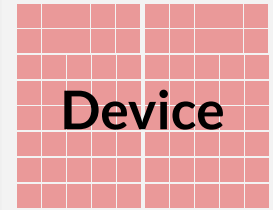
Agenda

8:15 - 8:45	*Morning refreshment and coffee*
8:45 - 9:00	*Welcome and introductions*
9:00 - 9:30	**Lecture 1: An introduction to OpenMP/OpenACC optimizations for CPUs/GPUs**
9:30 - 10:15	Lecture 2: A wider set of code patterns: compute patterns, memory patterns and flow patterns
10:15 - 10:30	Break
10:30 - 11:00	Lecture 3: Minimizing data transfers
11:00 - 11:30	Lecture 4: Optimizing memory usage
11:30 - 12:00	Lecture 5: Exploiting massive parallelism
12:00 - 13:00	Working lunch (hands-on activities)
13:00 - 14:00	Practical 5A: Parallelizing the calculation of HEAT
14:00 - 17:00	Hands-on time with your code
17:00 pm	*Close*

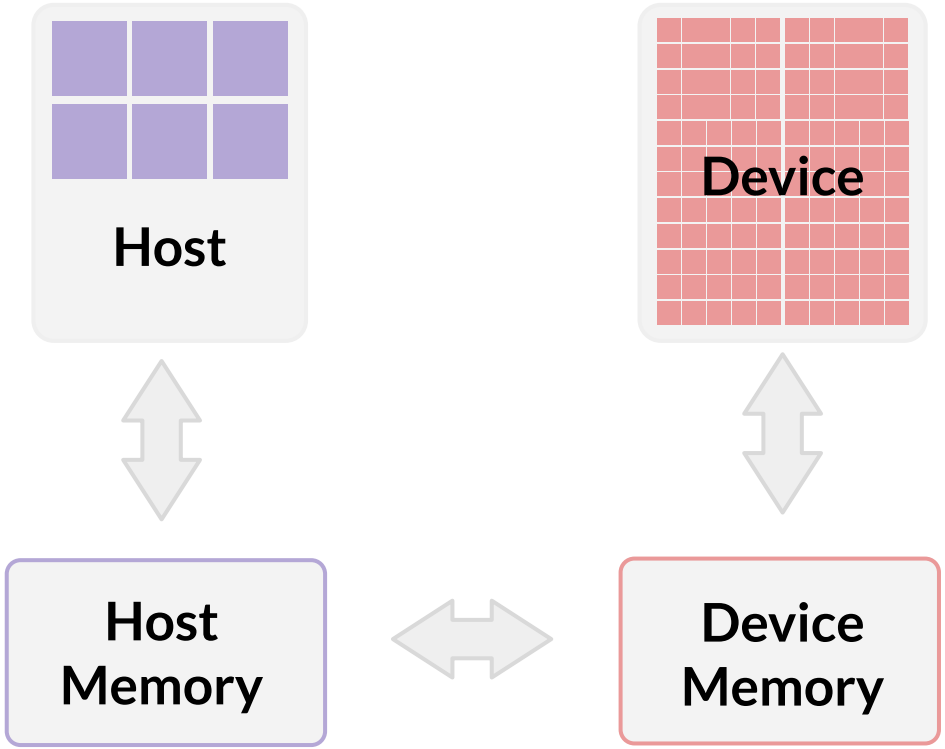
What are the differences between CPUs and GPUs?

- First, the number of cores available in the hardware
  - GPUs have many many more cores than CPUs
- Second, the grouping of the threads at the hardware level
  - In CPUs, the threads are not grouped and all the threads are executed at the same time
  - In GPUs, the threads are grouped and all the threads in a group are executed at the same time.
- Third, the complexity of the memory design
  - In CPUs, all the threads access to all the memory
  - In GPUs, there are constraints in the memory that can be accessed by the threads (e.g., cache, texture, scratchpad, global).
- Fourth, execution of instructions in vector mode
  - Both CPUs and GPUs exploit vector processing, although different “flavours” of it.





The GPU Accelerator Model



The GPU Execution Model

- Use of a host-driven execution model.
- Sequential code runs on a conventional processor.
- Computationally intensive parallel pieces of code (kernels) run on an accelerator such as a GPU.

- To maximize performance, high-performance applications generally conform to the following three rules of accelerator programming:
  - Transfer the data onto the device and keep it there.
  - Give the device enough work to do.
  - Focus on data reuse within the device(s) to avoid memory bandwidth bottlenecks

Why OpenMP/OpenACC for GPU programming?

- **GPUs have a reputation for being difficult to use because of programming**
  - OpenMP/OpenACC aim to change that!
  - OpenMP/OpenACC offer acceleration of scientific computing without significant programming effort.
- **OpenMP/OpenACC improve portability and readability of the code compared to other methods for using accelerators**
  - OpenMP/OpenACC codes use directives, which are added to the original code and can be safely ignored by compilers not supporting them.
- **Don't need to fully understand the specifics of the hardware you are using**
  - A limited understanding is still helpful to understand performance
- **Minimizes the need for code refactoring**
  - Though some refactoring may help with performance
- **Support for C/C++ and Fortran**
  - Note that our examples are in C! But the same methods apply to Fortran code

What are OpenMP/OpenACC?

- **Method for using multicore CPUs and GPUs**
  - Extension for C, C++ and Fortran

- **Uses compiler directives:**
  - Specify loops/regions to be offloaded to the GPU
  - Also runtime library functions and environment variables

- **Host/Accelerator programming model**
  - The CPU controls the accelerator
  - Uses the concepts of threads and tasks

- **OpenMP/OpenACC are focused on portability**

OpenMP/OpenACC Directive Syntax

- The **`omp/acc`** sentinel:
  - Tells the compiler that the text that follows is OpenMP/OpenACC, a directive

C and C++:

```
#pragma omp directive-name [clause [[,] clause]…]
#pragma acc directive-name [clause [[,] clause]…]
```

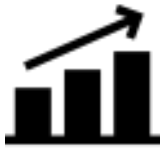
Fortran:

```
!$omp directive-name [clause [[,] clause]…]
!$acc directive-name [clause [[,] clause]…]
```

Available OpenMP/OpenACC compilers

	OpenMP Enabling HPC since 1997	OpenACC More Science, Less Programming
PGI 18.4	4.5	2.6
GCC 9	4.5	2.5
CCE 8.7	4.5	2.0
Clang 10	4.5	-
Intel Compiler 17	4.5	-

Benefits and Limitations of OpenMP/OpenACC



Benefits

- High-level platform independent
- Simple
- Portable

Limitations

- Cannot represent architectural specifics of devices without making the code less portable
- Some optimizations cannot be coded in these high-level APIs, and are only possible in lower-level APIs (e.g. CUDA, OpenCL)
- And sometimes this comes at a cost of performance

Challenge: Performance optimization on CPU/GPU

- **Typical optimizations for high-performance on the CPU**
  - Programming aware of low-level CPU features
    - Small number of threads available in the hardware
    - Instruction Set Architecture (ISA): eg. vector/simd instructions
    - Cache memory: space locality and time locality in cache lines in direct/associative caches
  - Code transformations for CPU-aware programming:
    - Loop transformations: fusion, fision, streap-mining, tiling/blocking
    - Data-layout transformations: array flattening, convert AoS to SoA, array padding

- **Typical optimizations for high-performance on the GPU**
  - Programming aware of low-level GPU features
    - Large number of threads available in the hardware
    - Warps are sets of threads that execute the same instruction, even doing nothing if needed
    - Complex memory system: scratch pad, texture memory, cache memory, global memory
  - Code transformations for GPU-aware programming:
    - Avoid thread divergence: prefer recomputation to conditional execution
    - Data-layout transformations: array flattening, convert AoS to SoA

Use cases: Performance optimization on CPU/GPU

Use case #1: Minimizing data transfers

- On GPUs: Transfer data from CPU to GPU and reuse it!
- On CPUs: Create threads and reuse them!

Use case #2: Optimizing memory usage

- On GPUs: Watch your data structure design as it may break your code!
- On CPUs: Hardware keeps memory consistency, so focus mostly on locality!

Use case #3: Exploiting massive parallelism

- On GPUs: Scale-up to thousands of threads!
- On CPUs: Limited number of threads, so not so important as on GPUs!