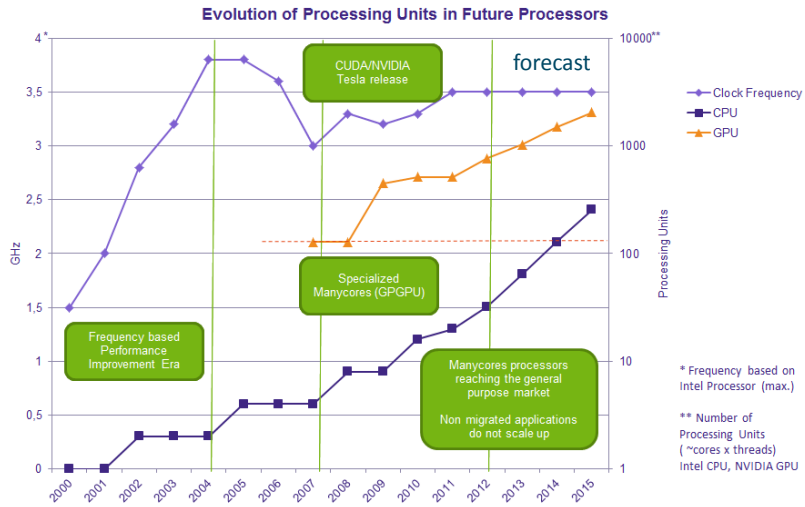
Directive-based Progamming for Accelerators

Florent Lebeau
CAPS entreprise
Ecole Polytechnique, June 2013

Why Accelerators?

Where Are We Going?



Accelerating Critical Paths

- Offloading the critical paths of an application can improve the application performance
- These critical paths are called hotspots
  - i.e the parts of the application where the majority of execution time is spent
  - The hotspots should be compute intensive
  - The hotspots should be data-parallel
- Sometimes the code may have to be modified to exhibit parallelism

CAPS

Many-core Technology Insights

Various Many-Core Paths

- Large number of small cores
- Data parallelism is key
- PCIe to CPU connection

AMD Discrete GPU

- 50+ number of x86 cores
- Support conventional programming
- Vectorization is key
- Run as an accelerator or standalone

INTEL Many Integrated Cores

- Integrated CPU+GPU cores
- Target power efficient devices at this stage
- Shared memory system with partitions

AMD APU

- Large number of small cores
- Data parallelism is key
- Support nested and dynamic parallelism
- PCIe to host CPU or low power ARM CPU (CARMA)

NVIDIA GPU

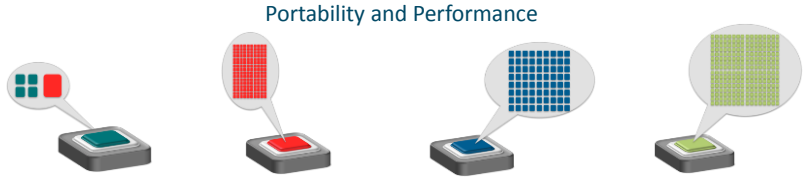
Many-core Programming Models

Applications

Libraries	Directives (OpenACC, OpenHMPP, etc)	Programming languages (CUDA, OpenCL, etc)
Just use as is	Standard and high level	Maximum performance and maintenance cost

CAPS Compilers

Portability and Performance



CAPS

www.caps-entreprise.com

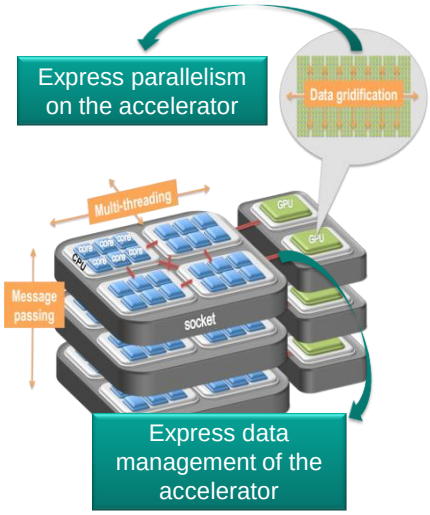
7

Directive-based Programming

CAPS

Directive-based Approaches

- Supplement an existing serial language with directives to express parallelism and data management
- Many variants
  - OpenHMPP
  - PGI Accelerator
  - OpenACC
  - OpenMP 4.0
  - …



Advantages of Directive-based Programming

- Simple and fast development of accelerated applications
- Non-intrusive
- Helps to keep a unique version of code
  - To preserve code assets
  - To reduce maintenance cost
  - To be portable on several accelerators
- Incremental approach
- Enables "portable" performance

OpenMP 4.0

Introduction

- OpenMP is an implementation of multi-threading
  - C/C++/FORTRAN specification published by the OpenMP ARB
    - AMD, CAPS, Intel, IBM, Nvidia, ORNL, PGI, TI, …
  - A master thread forks a number of slave threads
  - Enables task parallelism
  - Enables data parallelism
- OpenMP 4.0 will be the new specification of OpenMP directives
  - Release candidate version available
    - http://www.openmp.org
  - Final version planned summer 2013
  - Integrates accelerator support
  - No implementation yet

Two Kinds of Accelerators

- 1: the accelerator is just another computer
  - Intel Xeon Phi, TI DSPs, …
  - Runs a fairly complete operating system (e.g. linux)
    - Applications, threads, simple memory layout, SIMD instructions, …
  - Full OpenMP can be or is already implemented on that system
- 2: the accelerator is designed for performance
  - Nvidia or AMD GPUs, …
  - No real OS but programming API (e.g. CUDA, OpenCL, …)
    - Compute kernels, complex memory layout, coalescing, …
  - OpenMP cannot be implemented on the device

TARGET Construct

- Specifies a piece of code to be run on the accelerator
- For accelerator type 1, all OpenMP directives are allowed

```
!$omp target
   !$omp parallel num_threads(100)

     !$omp do
     DO i=1,n
       A(i) = compute(i)
     ENDDO

     !$omp barrier

     !$omp do
     DO i=1,n
       B(i) = A(i) + A(n-i-1)
     ENDDO

   !$omp end parallel
!$omp end target
```

TEAMS Construct

- For accelerator type 2 (e.g. GPU), OpenMP cannot be fully implemented
- A new level of parallelism was introduced: TEAMS
  - Basically corresponds to CUDA thread blocks or OpenCL workgroups
- The TEAMS directive creates a group of teams for a given code section
  - Many threads can be launched in each team
  - The master thread in each team starts executing the region

DISTRIBUTE Constructs

- Work can be distributed among several teams inside a TEAMS region thanks to the DISTRIBUTE directive
- Typically associated to a loop nest
- DISTRIBUTE constructs should be nested in a TEAMS section
  - The TEAMS section is itself included in a TARGET construct

Example

```
!$omp target …

   !$omp teams num_teams(1024)

     !$omp distribute
     DO i=1,n

       !$omp parallel do num_threads(256)
       DO j=1,m
         A(i,j) = compute(i,j)
       ENDDO

     ENDDO

   !$omp end teams

!$omp end target
```


TEAMS or not TEAMS?

- The model TARGET-TEAMS-DISTRIBUTE can theoretically be implemented everywhere
  - On Intel Xeon Phi too
- Use TARGET-TEAMS-DISTRIBUTE for portability
- However, the model TARGET enables the maximum of performance on Intel Xeon Phi
- The model TARGET alone cannot work on GPU
  - At least not efficiently
  - Could use a single team

Vectorization

- Ensure vectorization with the SIMD directive
- Applies to a loop
- Not specific to accelerators
  - On GPUs, iterations of the loop are distributed among the threads of a team
  - On other architectures, it controls vectorization units (SSE, AVX, …)

Data Management

- The TARGET DATA construct enables to:
  - Allocate memory space on the accelerator
  - Transfer data to and from the accelerator
- The MAP clause maps a variable from the current task data environment to a variable in the device data environment
  - "Alloc" clause performs an allocation on device only
  - "To" clause initializes the data on device when entering the region
  - "From" clause gets back the data from device when exiting the region

```
!$omp target data map(alloc:W) map(to:X) map(from:Y) map(tofrom:Z)

  ! W, X, W, Z are now mapped on the device
  …
  ! the following code still executes on the host
  …

!$omp end target data
```

Data Transfers

- The TARGET UPDATE directive enables to copy data to and from the accelerator
- The clauses
  - To: performs a transfer from the host to the accelerator device
  - From: performs a transfer from the device to the host

```
!$omp target data map(alloc:W,X,Y,Z)
  …
  !$omp target update to(X,Z)
  …
  !$omp target update from(Y)
  …
!$omp end target data
```


CAPS

OpenACC

Introduction

- Launched by CAPS, Cray, Nvidia and PGI in 2011
  - Allinea, Georgia Tech, U. Houston, ORNL, Rogue Wave, Sandia NL, Swiss National Computing Center, TUD joined in 2012
- Open Standard
- A directive-based approach for programming heterogeneous many-core hardware for C/C++ and FORTRAN applications
- http://www.openacc-standard.com

OpenACC DIRECTIVES FOR ACCELERATORS

CAPS

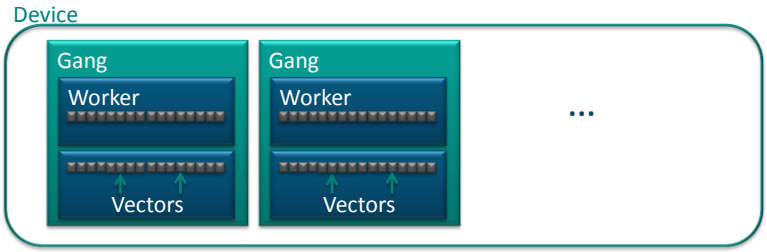
CRAY

NVIDIA

PGI

OpenACC Execution Model

- Host-controlled execution
- Based on three parallelism levels
  - Gangs – coarse grain (e.g. CUDA thread blocks)
  - Workers – fine grain (e.g. CUDA warps)
  - Vectors – finest grain (e.g. CUDA threads)



Kernels Construct

```
$!acc kernels […]

  DO i=1,n
    …
  END DO
  …
  DO j=1,n
    …
  END DO

$!acc end kernels
```

1st Kernel

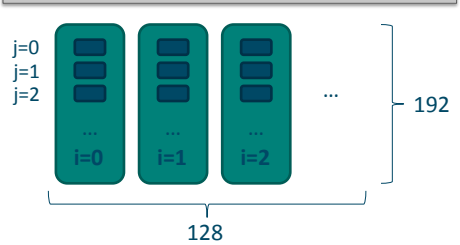
2nd Kernel

- Defines a region of code to be compiled into a sequence of accelerator kernels for execution on the accelerator device
- Typically, each loop nest will be a distinct accelerator kernel
- The number of gangs and workers can be different for each kernel

Loop Constructs

- A *Loop* directive applies to a loop that immediately follow the directive
- Enables to describe the parallelism to use with one of the following clause:
  - Gang for coarse-grain parallelism
  - Worker for middle-grain parallelism
  - Vector for fine-grain parallelism

```
#pragma acc kernels
{
  …
  #pragma acc loop gang(128)
  for(i=0; i < n; i++) {
    #pragma acc loop worker(128)
    for(j=0; j < m; j++) {
      …
    }
  }
  …
}
```



Data Management

- The DATA construct enables to:
  - Allocate device memory
  - Transfer data to and from the device
- The clauses
  - CREATE enables to allocate device memory only
  - COPYIN enable to initialize data when entering the region
  - COPYOUT enables to retrieve results when exiting the region

```
#pragma acc data create(A) \
                 copyin(B) \
                 copyout(C)
{
  #pragma acc kernels
  {
    …
  }
  …
  #pragma acc kernels
  {
    …
  }
}
```


Data Transfers

- Used within explicit or implicit data region
- It performs a data transfer:
  - Between the host and the accelerator device if it is followed by the clause "device"
  - Between the device and the host if it is followed by the clause "host"

```
!$acc kernels copyout(A(1:n)) \
              copyin (B(1:n))
  do i=1,n
    A(i) = B(n - i)
  end do
!$acc end kernels
```

```
!$acc data create( A(1:n), \
                   B(1:n) )

!$acc update device (B(1:n))
!$acc kernels
  do i=1,n
    A(i) = B(n - i)
  end do
!$acc end kernels
!$acc update host (A(1:n))

!$acc end kernels
```

CAPS

CAPS Compilers

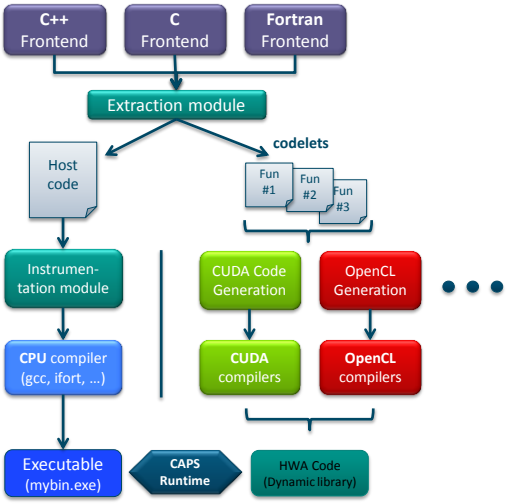
CAPS Compilers

- Take the original application as input and generate another application source code as output
  - Automatically turn the OpenACC source code into a accelerator-specific source code (CUDA, OpenCL)
- Compile the entire hybrid application
- Just prefix the original compilation line with *capsmc* to produce a hybrid application

```
$ capsmc gcc myprogram.c
$ capsmc gfortran myprogram.f90
```

CAPS Compilers Compilation Path

- CAPS Compilers drives all compilation passes
- Host application compilation
  - Calls traditional CPU compilers
  - CAPS Runtime is linked to the host part of the application
- Device code production
  - According to the specified target
  - A dynamic library is built



CAPS

Conclusion

Conclusion

- Many-Core becomes ubiquitous
  - Various accelerator architectures
  - Still as co-processors but going toward on-die integrated
- Programming models converge
  - Momentum with OpenACC directive-based standard
  - Followed by OpenMP with future specification 4.0
  - Key point is portability
- Directives enable fast development of high-level heterogeneous applications
  - For C and FORTRAN code
- Explicit the calls to a hardware accelerator in your code
  - Whatever the target
  - CAPS Compilers supports:
    - Nvidia Tesla GPUs
    - AMD GPUs and APUs
    - X86 Intel Xeon Phi

CAPS www.caps-entreprise.com 33

Accelerator Programming model
Parallelization
CAPS Workbench
Directive-based programming
HPC
Portability
OpenCL
Parallel Computing
CAPS Compilers
Many-Core programming
OpenHMPP
NVIDIA Cuda
Code speedup
GPGPU
OpenACC
High Performance Computing
Performance

CAPS
Visit CAPS Website:
www.caps-entreprise.com

6/5/2013 test-template 34