RWTH AACHEN UNIVERSITY

Introduction to Directive-based GPGPU Programming

PGI Accelerator and OpenACC

Sandra Wienke, M.Sc.
wienke@rz.rwth-aachen.de

PPCES 2012

Links

PGI Accelerator

- PGI Accelerator Home: http://www.pgroup.com/resources/accel.htm
- PGI Compiler User‘s Guide: http://www.pgroup.com/doc/pgiug.pdf
- PGI User Forum: http://www.pgroup.com/userforum/index.php

OpenACC

- OpenACC Home: www.openacc-standard.org/
- Specification: http://www.openacc-standard.org/Downloads/OpenACC.1.0.pdf
- OpenACC + OpenMP: http://terboven.wordpress.com/2011/11/16/openmp-and-openacc/

Agenda

- **Directive-based GPU Programming**
  - PGI Accelerator & OpenACC
- **Execution & Memory model**
- **Constructs**
  - Syntax
  - Compiler Feedback
  - Offload regions
  - Loops
  - Data regions & clauses
  - Misc
- **Runtime Library Routines**

Paradigm *PGI Accelerator*

- **Directive-based programming model for accelerators**
  - Delegates responsibility for low-level GPU programming tasks to compiler
  - → Higher-level than CUDA, OpenCL
  - User-directed programming
  - More productive development process, but in general “knowledge” still needed
- **Developed by the Portland Group (PGI)**
- **Support**
  - C and Fortran
  - Works only on NVIDIA GPUs currently
- **Timeline**
  - Jun’09: First specification and support of accelerator programming model (compiler 9.0)
  - Nov’10: Specification 1.3 of accelerator programming model
  - March’12: Full support of specification 1.3 (compiler 12.3)

Paradigm *OpenACC*

- **Directive-based programming model for accelerators**
  - Similar approach like PGI Accelerator
- **Industry standard**
- **Introduced by CAPS, Cray, NVIDIA, the Portland Group (PGI)**
  - Compiler by PGI, Cray, CAPS
- **Support**
  - C,C++ and Fortran
  - Works only on NVIDIA GPUs currently
- **Timeline**
  - Nov'11: Specification 1.0
  - Jan'12: Cray compiler understands parts of OpenACC API
  - March'12: PGI compiler understands parts of OpenACC API
- **Promising step towards OpenMP for accelerators**
  - PGI, Cray, CAPS, NVIDIA are members of subcommittee "OpenMP for Acc"
  - Intention to integrate lessons learned

Agenda

- Directive-based GPU Programming
  - PGI Accelerator & OpenACC
- **Execution & Memory model**
- Constructs
  - Syntax
  - Compiler Feedback
  - Offload regions
  - Loops
  - Data regions & clauses
  - Misc
- Runtime Library Routines

Execution & Memory Model

- **Host-directed execution model**
- **Separate host and device memories**
- **Host**
  - Executes most of the program
  - Allocates accelerator memory
  - Initiates data copy from host memory to accelerator
  - Sends kernel code to accelerator
  - Queues kernels for execution on accelerator
  - Waits for kernel completion
  - Initiates data copy from accelerator to host memory
  - Deallocates accelerator memory
- **Accelerator**
  - Executes kernel (one after another)
  - Concurrently, may transfer data between host and accelerator

→ Most tasks can be done by compiler/ runtime

Agenda

- Directive-based GPU Programming
  - PGI Accelerator & OpenACC
- Execution & Memory model
- **Constructs**
  - Syntax
  - Compiler Feedback
  - Offload regions
  - Loops
  - Data regions & clauses
  - Misc
- Runtime Library Routines

Constructs: Syntax & Feedback

Syntax

C

```
#pragma acc directive-name [clauses]
```

Fortran

```
!$acc directive-name [clauses]
```

PGI Acc

OpenACC

Compiler feedback

- Whether an accelerator kernel could be generated
- Which loop schedule is used
- Where/which data is copied

Getting Started (PGI Accelerator)

RWTH AACHEN UNIVERSITY

Compiling (use PGI Compiler)

```
module switch intel pgi/12.1          # on our cluster
pgcc  -ta=nvidia,4.0,cc20 saxpy.c
pgf90 -ta=nvidia,4.0,cc20 saxpy.F90
```

- **nvidia**: Obligatory. Accelerator regions are offloaded to NVIDIA GPU.
- **4.0**: Defines the version of the used CUDA toolkit
- **cc20**: Set compute capability 2.0

  →Sets certain architecture features, e.g. enabling double precision floating point operations

Tips

- **-ta=nvidia,time**: Getting simple timing information
- `export ACC_NOTIFY=1`: Prints message when kernel is executed

green = background information
PGI Accelerator

Example SAXPY – Serial (host)

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* restrict x; float* restrict y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE




  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }



  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

Example SAXPY – Directives for GPGPUs

RWTH AACHEN UNIVERSITY

PGI Acc

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* restrict x; float* restrict y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE

  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

OpenACC

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* x; float* y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE

  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

Example SAXPY – Directives for GPGPUs

RWTH AACHEN UNIVERSITY

PGI Acc

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* restrict x; float* restrict y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE
#pragma acc data region copyin(x[0:n-1])
{
#pragma acc region copy(y[0:n-1])

#pragma acc for
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
#pragma acc region copy(y[0:n-1])

#pragma acc for
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
}
  free(x); free(y); return 0;
}
```

OpenACC

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* x; float* y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE
#pragma acc data copyin(x[0:n])
{
#pragma acc parallel copy(y[0:n])

#pragma acc loop
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
#pragma acc parallel copy(y[0:n])

#pragma acc loop
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
}
  free(x); free(y); return 0;
}
```

Example SAXPY – Directives for GPGPUs

RWTH AACHEN UNIVERSITY

PGI Acc

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* restrict x; float* restrict y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE
#pragma acc data region copyin(x[0:n-1])
{
#pragma acc region copy(y[0:n-1])

#pragma acc for parallel vector(256)
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
#pragma acc region copy(y[0:n-1])

#pragma acc for parallel vector(256)
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
}
  free(x); free(y); return 0;
}
```

OpenACC

```
int main(int argc, const char* argv[]) {
  int n = 10240; float a = 2.0f;
  float* x; float* y;
  x = (float*) malloc(n * sizeof(float));
  y = (float*) malloc(n * sizeof(float));
  // Initialize x, y

  // Run SAXPY TWICE
#pragma acc data copyin(x[0:n])
{
#pragma acc parallel copy(y[0:n])
                     vector_length(256)
#pragma acc loop gang vector
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
#pragma acc parallel copy(y[0:n])
                     vector_length(256)
#pragma acc loop gang vector
  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }
}
  free(x); free(y); return 0;
}
```

Constructs: Offload regions

RWTH AACHEN UNIVERSITY

Offload region

- Region maps to a CUDA kernel function

PGI Acc

C

```
#pragma acc region [clauses]
```

Fortran

```
!$acc region [clauses]

!$acc end region
```

OpenACC

C/C++

```
#pragma acc parallel [clauses]
```

Fortran

```
!$acc parallel [clauses]

!$acc end parallel
```

Constructs: Loops

Share work of loops

- Loop work gets distributed among threads on GPU (in certain schedule)

PGI Acc

C
```
#pragma acc for[clauses]
```

Fortran
```
!$acc do[clauses]
```

OpenACC

C/C++
```
#pragma acc loop[clauses]
```

Fortran
```
!$acc loop[clauses]
```

Loop schedules (clauses)

- Specifies the mapping of loop-level parallelism onto acc parallelism
- Distributes work into thread blocks ……..
- Distributes work into warps ……
- Distributes work into threads within warp/ thread block ……
- Executes loop sequentially on the device.

C, Fortran	C/C++, Fortran
parallel	gang
	worker
vector	vector
seq	seq

PGI Acc | OpenACC

Constructs: Data regions & clauses

RWTH AACHEN UNIVERSITY

Data region

- Decouples data movement from offload regions

PGI Acc

C
```
#pragma acc data region [clauses]
```

Fortran
```
!$acc data region [clauses]

!$acc end region
```

OpenACC

C/C++
```
#pragma acc data [clauses]
```

Fortran
```
!$acc data [clauses]

!$acc end data
```

Data clauses (for data/offload regions)

- Triggers data movement of denoted arrays

- H2D-copy at region start + D2H at region end …
- Only H2D-copy at region start ……..
- Only D2H-copy at region end ……..
- Allocates data on device, no copy to/from host ..
- Data is already on device ……..…

PGI Acc	OpenACC
C, Fortran	C/C++, Fortran
copy	copy
copyin	copyin
copyout	copyout
local	create
	present

Constructs: Misc

Reductions

- Reduction code will be generated by compiler

PGI Acc

- Reductions should be automatically recognized by the compiler
- May need hints by creating temporary variables for reduction

OpenACC

C/C++, Fortran

`reduction(op:list)`

- For parallel and loop constructs
- Operantors: +,*,max,min,…

Update clause/ directive

- Updates the content of arrays that exist on host and device
- Either update the host-side or the device-side

Cache clause/ directive

- Prioritizes data for placement in the highest level of data cache on GPU

Agenda

- Directive-based GPU Programming
  - PGI Accelerator & OpenACC
- Execution & Memory model
- Constructs
  - Syntax
  - Compiler Feedback
  - Offload regions
  - Loops
  - Data regions & clauses
  - Misc
- **Runtime Library Routines**

Runtime library routines

Interface to runtime routines & data types

PGI Acc

C
```
#include "accel.h"
```

Fortran
```
use accel_lib
```

OpenACC

C/C++
```
#include "openacc.h"
```

Fortran
```
use openacc
```

Initialization of device

- E.g. to exclude initialization time from computation time

PGI Acc

C, Fortran
```
acc_init(devicetype)
```

OpenACC

C/C++
```
acc_init(devicetype)
```

Setting device to run on

…