Additional Compilers, Libraries, and Tools

Philip C. Roth

Useful Information

- **For today only**, to submit jobs to the Keeneland queue you need to add the following to the qsub command line

  -l walltime=…,feature=cudaworkshop

- Or add it to the –l line in your job script

  #PBS –l walltime=…,feature=cudaworkshop

- Be sure to return your OTP token to Bruce Loftis before you leave

Keeneland Software

- Keeneland project funding is mainly for system procurement and effort
- Leveraging freely-available software whenever possible
  - Open source
  - Proprietary but freely available
  - Shared and/or site licenses

Keeneland Software (2)

- The GPU software landscape is changing rapidly
  - New tools and libraries
  - Old tools and libraries with new features
- There are several software packages that may be useful to Keeneland users but that we have not (yet!) used on the system

Categories

- Compilers/translators
- Language bindings
- Libraries
- Performance and debugging tools

- Will focus on software not discussed elsewhere in the tutorial
- *Not an exhaustive list!*

Compilers/Translators

- Tools for producing code that runs on a GPU

- PGI
- HMPP Workbench
- R-Stream
- OpenMPC
- ROSE

The Portland Group

- PGI offers several GPU and GPU-related features
  - Accelerator directives
  - CUDA Fortran
  - CUDA x86

- http://www.pgroup.com

PGI Accelerator

- Directives similar to OpenMP but cause parts of the program to run on a CUDA-capable GPU
- Can control data movement and computation
- Fortran and C/C++
- Add-on feature with PGI 10.x and newer compilers

PGI Accelerator Example

- 5-point weighted stencil
- Data and compute regions
- Data movement optimizations

```
!$acc data region copy(a(1:n,1:m)) local(b(2:n-1,2:m-1)) copyin(w(2:n-1))
  do while(...)
    !$acc region
      do i = 2, n-1
        do j = 2, m-1
          b(i,j) = 0.25*w(i)*(a(i-1,j)+a(i,j-1)+ &
                              a(i+1,j)+a(i,j+1)) &
                 +(1.0-w(i))*a(i,j)
        enddo
      enddo
      do i = 2, n-1
        do j = 2, m-1
          a(i,j) = b(i,j)
        enddo
      enddo
    !$acc end region
  enddo
!$acc end data region
```

http://www.pgroup.com/lit/articles/insider/v2n1a1.htm

PGI CUDA Fortran

- Supports Fortran-based CUDA programs
- Seems to follow the principle of least surprise
  - Similar to CUDA C
  - Differs in ways that one would expect if familiar with differences between “regular” C and Fortran
- Add-on feature with PGI 10.x and newer compilers

PGI CUDA Fortran example:

- Matrix multipliciation host

```
   subroutine mmul( A, B, C )
!
   use cudafor
   real, dimension(:,:) :: A, B, C
   integer :: N, M, L
   real, device, allocatable, dimension(:,:) :: Adev,Bdev,Cdev
   type(dim3) :: dimGrid, dimBlock
!
   N = size(A,1) ; M = size(A,2) ; L = size(B,2)
   allocate( Adev(N,M), Bdev(M,L), Cdev(N,L) )
   Adev = A(1:N,1:M)
   Bdev = B(1:M,1:L)
   dimGrid = dim3( N/16, L/16, 1 )
   dimBlock = dim3( 16, 16, 1 )
   call mmul_kernel<<<dimGrid,dimBlock>>>( Adev,Bdev,Cdev,N,M,L )
   C(1:N,1:M) = Cdev
   deallocate( Adev, Bdev, Cdev )
!
   end subroutine
```

http://www.pgroup.com/lit/articles/insider/v1n3a2.htm

PGI CUDA Fortran example (2):

- Matrix multiplication kernel

```
   attributes(global) subroutine MMUL_KERNEL( A,B,C,N,M,L)
!
   real,device :: A(N,M),B(M,L),C(N,L)
   integer,value :: N,M,L
   integer :: i,j,kb,k,tx,ty
   real,shared :: Ab(16,16), Bb(16,16)
   real :: Cij
!
   tx = threadidx%x ; ty = threadidx%y
   i = (blockidx%x-1) * 16 + tx
   j = (blockidx%y-1) * 16 + ty
   Cij = 0.0
   do kb = 1, M, 16
      ! Fetch one element each into Ab and Bb;  note that 16x16 = 256
      ! threads in this thread-block are fetching separate elements
      ! of Ab and Bb
      Ab(tx,ty) = A(i,kb+ty-1)
      Bb(tx,ty) = B(kb+tx-1,j)
      ! Wait until all elements of Ab and Bb are filled
      call syncthreads()
      do k = 1, 16
         Cij = Cij + Ab(tx,k) * Bb(k,ty)
      enddo
      ! Wait until all threads in the thread-block finish with
      ! this iteration's Ab and Bb
      call syncthreads()
   enddo
   C(i,j) = Cij
!
   end subroutine
```

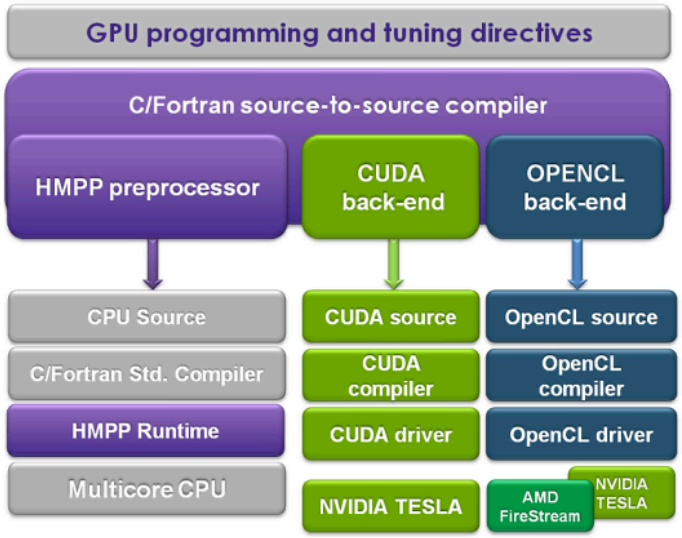
http://www.pgroup.com/lit/articles/insider/v1n3a2.htm

PGI CUDA-x86

- Translates CUDA C for multi-core x86_64 CPUs
- Not yet available
  - Demonstrated prototype at SC10
  - Functionality release in Q2 2011
  - Unified CPU/GPU executables supported in Q4 2011

CAPS Enterprise HMPP Workbench

- Directive based (like OpenMP, PGI Accelerator)
- Supports C and Fortran programs
- Source to source translator
- Back-ends for CUDA and OpenCL
- Can run on host if GPUs are not available



- http://www.caps-entreprise.com/index.php

HMPP Workbench example

- SGEMM

```
#pragma hmpp sgemm codelet, args[m;n;k;alpha;beta;a;b].io=in, &
#pragma hmpp sgemm          args[c].io=inout, target=CUDA:OPENCL
void sgemm(int m, int n, int k,float alpha, float a[m][k],
           float b[k][n],float beta, float c[m][n]);

int main(int argc, char **argv) {
...

/* Allocate device and memory */
#pragma hmpp sgemm allocate, args[c].size={M,N}
/* Prefetch all data, alpha and beta are constant */
#pragma hmpp sgemm advancedload, args[alpha;beta;a;b;c], & …
...
#pragma hmpp sgemm callsite, args[alpha;beta;a;b;c].advancedload=true
sgemm( M, N, K, alpha, t1, t2, beta, t3 );

/* retrieve c output data */
#pragma hmpp sgemm delegatedstore, args[c]
/* release the device */
#pragma hmpp sgemm release
...
return 0;
}
```

http://www.caps-entreprise.com/upload/ckfinder/userfiles/files/caps_hmpp_ds.pdf

Reservoir Labs R-Stream

- "High Level Compiler"
- Source to source translator
  - Translates sequential C code for lower-level "native" compiler
- Uses polyhedral model for automated parallelism
  - Identifies more parallelism than traditional approach
  - Enables coordinated optimizations instead of optimizations in sequential phases
- Earlier work targets optimizations for OpenMP, and Cell processor, more recent work targets GPUs
- https://www.reservoir.com/rstream

ROSE

- Infrastructure for building source-to-source translators
- ROSE-based translators can accept C, C++, and Fortran
- Recently added support for building translators that can generate CUDA and OpenCL code
- Currently working on support for consuming CUDA and OpenCL code
- http://www.rosecompiler.org

OpenMPC

- Translates programs with OpenMPC directives to run on CUDA-enabled devices
- OpenMPC extends OpenMP
  - Original goal of project was to map OpenMP automatically to GPUs
  - Found additional clauses were desirable
    - Controlling data motion
    - Kernel execution parameters like block size
- https://engineering.purdue.edu/paramnt/OpenMPC/

Language Bindings

- Notable examples:
  - PyCUDA, PyOpenCL: call Cuda, OpenCL functions from Python programs
    - Kernels still written in CUDA C or OpenCL
    - But Copperhead uses PyCUDA and allows kernels written in Python
  - PGI CUDA Fortran
  - C++ bindings for OpenCL, available from Khronos OpenCL repository

Libraries

- Libraries for accelerating performance-critical parts of a program

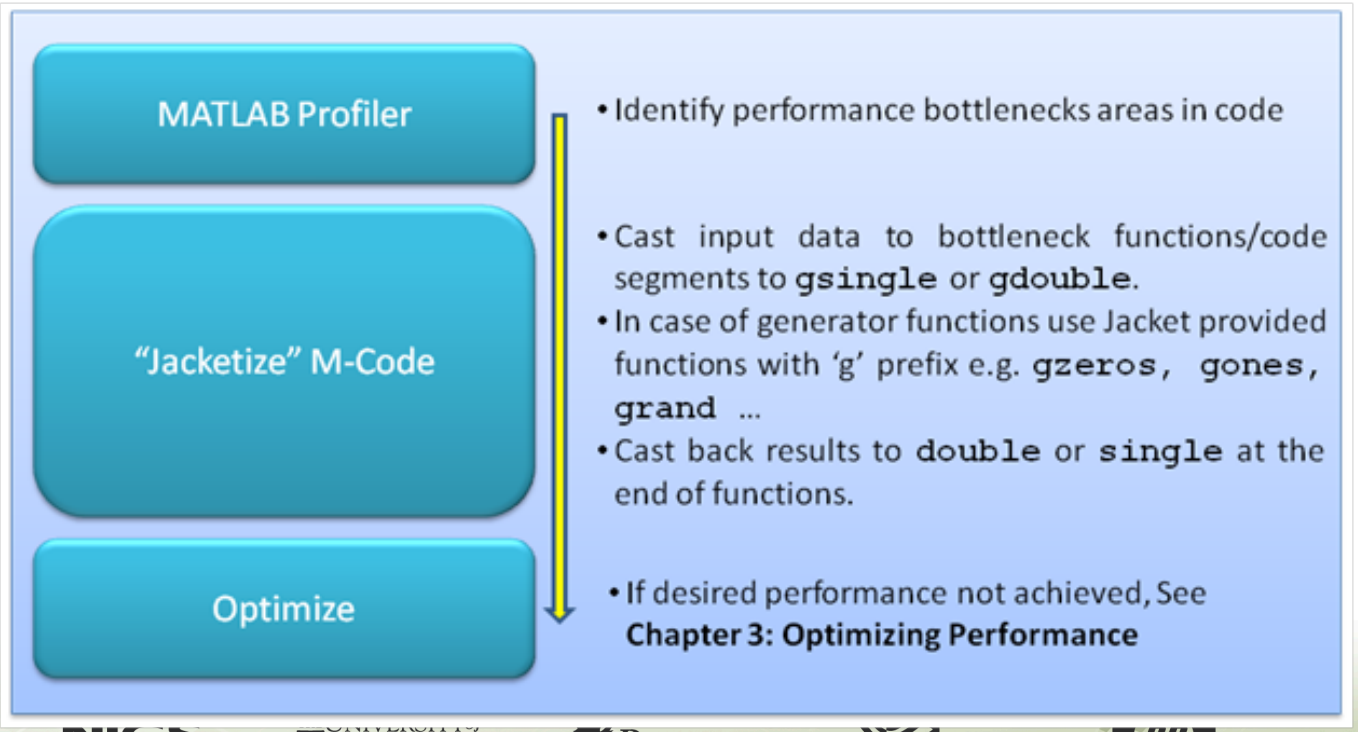
- Jacket
- libJacket
- Thrust
- CUDPP

AccelerEyes Jacket

- GPU-accelerated package for MATLAB programs
- Supports CUDA-capable GPUs
- Provides accelerated versions of many functions/ programming constructs
- Includes accelerated libraries for
  - Image processing
  - Signal processing
  - Statistics
  - Data visualization
- http://www.accelereyes.com

Jacket Workflow

- Simple integration into current workflow
  - Sometimes just a cast of variables to g* forms before passing to functions
  - May also alter control flow constructs such as using GFOR instead of FOR



Jacket Example

- Small 2D FFTs
- Reported 600% speedup on "dual-core Intel machine with NVIDIA Qaudro FX 5800"

```
>>  N = 128; % matrix size
>>  M = 200; % number of tiled matrices
>>
>>  %Create Data
>>  [Ac Bc] = deal(complex( ones(N,N,M, 'single'),0));
>>
>>  % Compute 200 (128x128) FFTs
>>  for ii = 1:M
>>      Ac(:,:,ii) = fft2(Bc(:,:,ii));
>>  end
```

```
>>  N = 128; % matrix size
>>  M = 200; % number of tiled matrices
>>
>>  % Create Data
>>  [Ac Bc] = deal(complex( gones(N,N,M, 'single'),0));
>>
>>  % Compute 200 (128x128) FFTs
>>  gfor ii = 1:M
>>      Ac(:,:,ii) = fft2(Bc(:,:,ii));
>>  gend
>>
>>  % Bring the results back to CPU
>>  Ac = single(Ac);
```

http://wiki.accelereyes.com/wiki/index.php/FFT_GFOR_Example

libJacket

- Library providing access to same core functionality as Jacket
- Enables access to Jacket core from C, C++, .NET, Python, Java, Fortran, and R
- Like Jacket, often requires only changing variable types

libJacket Example

- Compute Pi using Monte Carlo method

```
float gpu_pi_example(const f32& x, const f32& y)
{
    float pi = sum_vector(sqrt(x*x + y*y) < 1) * (4.0f / NSET);
    return pi;
}

void pi_example()
{
    // Compute the random values
    f32 x = f32::rand(NSET);
    f32 y = f32::rand(NSET);

    // Ensure that the data has been copied to the host side so we
    // don't include the cost of bringing the random data to the
    // host in the timing of the CPU computation of PI.
    float *fx = x.host();
    float *fy = y.host();

    // Perform the CPU computation
    gtime_t cpu_start = gtime_now();
    float pi_cpu = cpu_pi_example(fx, fy);
    gtime_t cpu_end = gtime_now();

    // Perform the GPU computation
    gtime_t gpu_start = gtime_now();
    float pi_gpu = gpu_pi_example(x, y);
    gtime_t gpu_end = gtime_now();
```

http://wiki.accelereyes.com/wiki/libjacket/examples_2pi_2pi_8cpp-example.htm

Thrust

- Parallel algorithms for C++ programs
- Uses C++ templates
- Similar interface to C++ standard template library
- Beginning with CUDA 4.0, Thrust will be included with CUDA Toolkit

- http://code.google.com/p/thrust/

What Thrust Provides

- Containers
  - thrust::host_vector<T>
  - thrust::device_vector<T>
  - Iterators over containers
  - Compatible with STL containers
- Algorithms
  - Many standard algorithms, e.g., sort, reductions
  - Operate over containers
  - Support both built-in types and user-defined types
  - Supports overriden low-level operators, e.g., a sum reduction using a '+' operator that makes sense for your custom structure

Thrust Example

- Sort values on GPU

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device (846M keys per second on GeForce GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```

CUDA Data Parallel Primitives (CUDPP)

- Library of data parallel primitives implemented using CUDA
- Programs still use CUDA
  - Initialization
  - Finalization
  - Data transfers
- Addresses largely the same problems as Thrust…
- …but library interface with C API makes CUDPP more widely accessible than Thrust

- http://code.google.com/p/cudpp/

CUDPP Example

- Model: build a plan, then execute the plan

```
CUDPPConfiguration config;
config.op = CUDPP_ADD;
config.datatype = CUDPP_FLOAT;
config.algorithm = CUDPP_SCAN;
config.options = CUDPP_OPTION_FORWARD | CUDPP_OPTION_EXCLUSIVE;

CUDPPHandle scanplan = 0;
CUDPPResult result = cudppPlan(&scanplan, config, numElements, 1, 0);

if (CUDPP_SUCCESS != result)
{
    printf("Error creating CUDPPPlan\n");
    exit(-1);
}
// Run the scan
cudppScan(scanplan, d_odata, d_idata, numElements);
```

Tools

- Tools for improving productivity

- RogueWave TotalView debugger
- Vampir performance tool

TotalView

- Well-known GUI-based parallel debugger for HPC systems
- Recently added support for debugging CUDA programs
- Whereas Allinea's DDT uses NVIDIA's cuda-gdb as the node-level debugger, TotalView implements its own

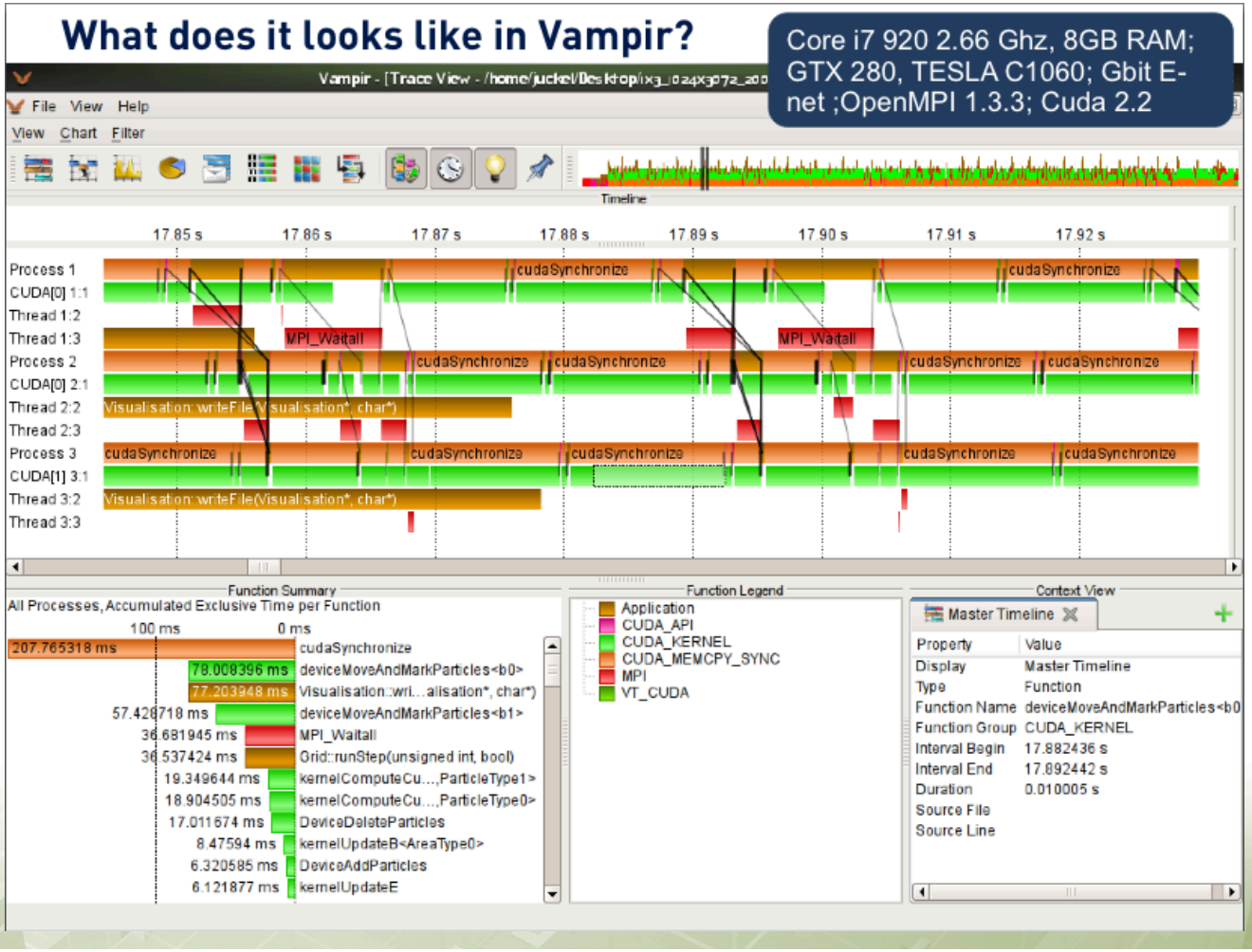
- http://www.roguewave.com/products/totalview-family.aspx

Vampir

- Performance data collection and visualization tool for parallel programs
- Trace-based
- Now supports CUDA and OpenCL programs
- Uses library API wrapping approach
- With CUDA 4.0, uses NVIDIA CUPTI for GPU hardware counter access
- Once available, will use CUPTI callbacks for kernel timing information
- http://www.vampir.eu

Vampir Example Screenshot

- Timeline display of MPI+CUDA program
- Produced with early version of Vampir GPU support



For more information

- http://keeneland.gatech.edu