Use of Accelerate Tools
PGI CUDA FORTRAN
Jacket

Supercomputing Institute
For Advanced Computational Research
e-mail: szhang@msi.umn.edu or help@msi.umn.edu
Tel: 612-624-8858 (direct), 612-626-0802 (help)
Tentative Agenda:

9:30 - 10:00 Intro – GPU computing, Hardware (Jeff)
10:00 - 11:00 Basics of Cuda Programming (Weijun)
11:00 - 12:00 hands-on exercise

1:00 - 1:40 Use of Memory Hierarchy for Performance Enhancement (David)
1:50 – 2:10 hands-on exercise
2:10 – 2:20 Break
2:20 – 3:10 Use of acceleration tools (Shuxia)
   CUDA FORTRAN & Jacket
3:10 – 4:00 hands-on exercises
Survey Questionaires:

Why are you interested in GPU computing?

What kind of applications do you need to accelerate on GPU hardware?

Do you have the computing code(s) already on CPU?  
If yes, in what language is it written (C, FORTRAN or Matlab)?

Do you have a deadline or milestone to get your computing on GPU hardware? When?

Specific need about the hardware (memory, multi-GPU and interconnect need)?

Will you learn CUDA or use the acceleration tools to get your calculations on GPU hardware?

How can we do better for the future GPU workshop:  
Specific topics are you interested?  
Specific acceleration tools?
PGI CUDA FORTRAN

1. A small set of extensions to Fortran
2. Supports and is built up on the CUDA
3. A lower-level explicit programming model
4. Substantial run-time library components
5. An analog to NVIDIA's CUDA C compiler
   Portland License!
CUDA Fortran extensions:
• Declaring variables allocated in the GPU device memory
• Allocating dynamic memory in the GPU device memory
• Copying data between the host memory to the GPU memory
• Writing subroutines and functions to execute on the GPU
• Invoking GPU subroutines from the host
• Allocating pinned memory on the host
• Using asynchronous transfers between the host and GPU
CUDA Fortran Programming

Host code
- Optional: select a GPU
- Allocate device memory
- Copy data to device memory
- Launch kernel(s)
- Copy data from device memory
- Deallocate device memory

Kernel code
- Attributes clause
- Kernel subroutines, device subprograms
- Shared memory
- What is and what is not allowed in a kernel
**CUDA C vs CUDA Fortran**

**CUDA C**
- supports texture memory
- supports Runtime API
- supports Driver API
- `cudaMalloc`, `cudaFree`
- `cudaMemcpy`
- OpenGL interoperability
- Direct3D interoperability
- arrays zero-based
- `threadidx/blockidx` 0-based
- unbound pointers
- pinned allocate routines

**CUDA Fortran**
- no texture memory
- supports Runtime API
- no support for Driver API
- `allocate`, `deallocate`
- Assignments `(A=d)A`
- no OpenGL interoperability
- no Direct3D interoperability
- arrays one-based
- `threadidx/blockidx` 1-based
- allocatable are device/host
- pinned attribute
CUDA Fortran Programming

Key building blocks:
- Use the cudafor module
- Attributes clause
- Kernel subroutines, device subprograms
- Use of memory hierarchy
- Thread Blocks
- What is and what is not allowed in a kernel
### Intrinsic data-types in device subprograms

<table>
<thead>
<tr>
<th>Type</th>
<th>Type Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>1,2,4,8</td>
</tr>
<tr>
<td>logical</td>
<td>1,2,4,8</td>
</tr>
<tr>
<td>real</td>
<td>4,8</td>
</tr>
<tr>
<td>double precision</td>
<td>real(kind=8)</td>
</tr>
</tbody>
</table>
CUDA Fortran Programming

Host code – GPU-related operations
- Optional: select a GPU
- Allocate device memory
- Copy data to device memory
- Launch kernel(s)
- Copy data from device memory
- Deallocate device memory

Device code
- Scalar thread code, limited operations
- Implicitly parallel
  - thread blocks scheduled by hardware on any multiprocessor
  - runs to completion before next kernel
subroutine vadd(A,B,C,N)
real(4) :: A(N), B(N), C(N)
integer :: N
integer :: i
make
C(i) = A(i) + B(i)
enddo
end subroutine

subroutine vadd(A, B, C)
use kmod
real(4), dimension(:) :: A, B, C
real(4), device, allocatable, &
dimension(:):: Ad, Bd, Cd
integer :: N
N = size(A, 1)
allocate(Ad(N), Bd(N), Cd(N))
Ad = A(1:N)
Bd = B(1:N)
call kernel<<< grid, block >>>( Ad, Bd, Cd, N )
call kernel<<< (N+31)/32, 32 >>>( Ad, Bd, Cd, N )
C(1:N) = Cd
deallocate(Ad, Bd, Cd)
end subroutine
module kmod
  use cudafor
contains
  attributes(global) subroutine kernel(A,B,C,N)
    real(4), device :: A(N), B(N), C(N)
    integer, value :: N
    integer :: i
    i = (blockidx%x-1)*32 + threadIdx%x
    if( i <= N ) C(i) = A(i) + B(i)
  end subroutine
end module
Attributes clause for subroutines and/or functions

attributes(host), or by default, host subprogram to be executed on host
   can only be called from another host subprogram
attributes(global) - a kernel subroutine to be executed on the device
   may only be called from the host using a kernel call.
attributes(device) - a device Subprogram, subroutine or function to be executed on the device;
   must be called from a subprogram with the global or device Attribute.

Restrictions

not be recursive, not contain variables with the save or data initialization; may not also have the device or host attribute; not have optional arguments; must not have the pointer attribute.
Attributes clause for variables and arrays

- `attributes(device)` - device variable
  - allocated in the device global array

- `attributes(constant)` – device constant variable
  - allocated in the device constant memory space

- `attributes(shared)` - a device shared variable
  - may only be declared in a device subprogram
  - is allocated in the device shared memory for a thread block
  - can be read or written by all threads in the block

- `attributes(pinned)` - a pinned variable
  - must be an allocatable array
  - will be allocated in host pagelocked memory
Execution Configuration

Call kernel<<<grid,block>>>(arg1, arg2,...) 
Where grid and block – execution configuration integer expression or type(dim3).

Predifined variables of type(dim3) used on host:
block – block%x, block%y, block%z
grid – grid%x, grid%y, grid%z

Predifined variables of type(dim3) used on device:
threadidx – threadidx%x, threadidx%y, threadidx%z
blockidx – blockidx%x, blockidx%y, blockidx%z
blockdim – blockdim%x, blockdim%y, blockdim%z

Asynchronous Concurrent Execution
Asynchronous Concurrent Execution
Concurrent Host and Device Execution - a kernel launch

call cudaThreadSynchronize ! the host program can synchronize and wait for all previously launched or queued kernels

Concurrent Stream Execution
Operations involving the device, including kernel execution and data copies to and from device memory, are implemented using stream queues.

call syncthreads() ! The device program
Hands-on Exercise:
https://www-test2.msi.umn.edu/content/gpu-hands-tutorial

Use of CUDA blas library
Assignment: Modify the code CPU_Sgemm.f90 to call sgemm of cuda blas library for calculating $C = a \times A \times B + b \times C$

Hints: 1.) Compile and run CPU_Sgemm as is for $N = 10000$ to see many Gflops it achieves;
2). Add the following as device interface

Get the tar file
www.msi.umn.edu/~szhang/GPU_Tools.tar
cat README
Hands-on Exercise:

```fortran
program test_CPU_Sgemm
real, allocatable, dimension(:,:) :: a, b, c
!real, device, allocatable, dimension(:,:) :: dA, dB, dC
real :: alpha = 1.0e0
real :: beta  = 1.0e0
print *, "Enter N: "
read(5,*) n
allocate(a(n,n), b(n,n), c(n,n))
a = 2.0e0;b = 1.5e0; c = -9.9e0
!allocate (dA(n,n), dB(n,n), dC(n,n))
!dA = a; dB = b; dC = c
call sgemm('n','n', n, n, n, alpha, a, n, b, n, beta, c, n)
!call sgemm('n','n', n, n, n, alpha, dA, n, dB, n, beta, dC, n)
!c=dC
end
```
Hands-on Exercise:

How to compile

module load pgi
pgfortran -O2 -o CPU_perf CPU_Sgemm.F90 -lblas (or -lacml)
pgfortran -Mcuda -o GPU_perf GPU_Sgemm.F90 -lcublas
pgfortran -o GPU_perf GPU_Sgemm.cuf -lcublas

How to run
/usr/bin/time ./CPU_perf < input
/usr/bin/time ./GPU_perf < input
Jacket

Wraps some of Matlab codes for enhancing their performance by running on GPU

module load jacket matlab
matlab
>> gactivate
>> ghelp % list all functions supported by Jacket
>> ghelp try %

All Jacket functions may be found at:
How can Jacket help?

Partial support - Not every Matlab calculation can benefit

Hot spot – part of the code consumes most of the CPU time

Special functions and toolbox – are they being used? Are they supported by Jacket?

If yes, modify the code according to Jacket's syntax.
## Performance Enhancement of Matlab Calculations

Use of Jacket

- replacement of low-level MATLAB data structures
- GPU computation and acceleration

<table>
<thead>
<tr>
<th>Jacket Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSINGLE</td>
<td>Casts a CPU matrix to a single precision floating point GPU matrix.</td>
<td>A = gsingle(B);</td>
</tr>
<tr>
<td>GDOUBLE</td>
<td>Casts a CPU matrix to a double precision floating point GPU matrix.</td>
<td>A = gdouble(B);</td>
</tr>
<tr>
<td>GLOGICAL</td>
<td>Casts a CPU matrix to a binary GPU matrix. All non zero values are set to '1'. The input matrix can be a GPU or CPU datatype.</td>
<td>A = glogical(B); A = glogical(0:4);</td>
</tr>
<tr>
<td>GINT8, GUINT8, GINT32, GUINT32</td>
<td>Cast a CPU matrix to a signed and unsigned 8-bit or 32-bit integer GPU matrix respectively.</td>
<td>A = gint8(B); A = guint8(B); A = gint32(B); A = guint32(B);</td>
</tr>
<tr>
<td>GZEROS, ZEROS</td>
<td>Create a matrix of zeros on the GPU.</td>
<td>A = gzeros(5,'double'); A = zeros(2,6,gdouble);</td>
</tr>
<tr>
<td>GONES, ONES</td>
<td>Create a matrix of ones on the GPU.</td>
<td>A = gones(5,'double'); A = ones([3 9], gdouble);</td>
</tr>
<tr>
<td>GEYE</td>
<td>Creates an identity matrix on the GPU.</td>
<td>A = eye(5);</td>
</tr>
<tr>
<td>GRAND or RAND</td>
<td>Creates a random matrix on the GPU, with uniformly distributed pseudorandom numbers.</td>
<td>A = rand(5,'double');</td>
</tr>
<tr>
<td>GRANDN</td>
<td>Creates a random matrix on the GPU, with normally distributed pseudorandom numbers.</td>
<td>A = randn(5,'double');</td>
</tr>
</tbody>
</table>
## Basic functions

<table>
<thead>
<tr>
<th>Jacket Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>GHELP</td>
<td>Retrieve information on the Jacket support for any function.</td>
<td>ghelp sum; gactivate;</td>
</tr>
<tr>
<td>GACTIVATE</td>
<td>Used for manual activation of a Jacket license.</td>
<td>gactivate; gselect(0);</td>
</tr>
<tr>
<td>GSELECT</td>
<td>Select or query which GPU is in use.</td>
<td>gfor n = 1:10; % loop body gend;</td>
</tr>
<tr>
<td>GFOR</td>
<td>Executes FOR loop in parallel on GPU.</td>
<td>my_fn = gcompile('filename.m');</td>
</tr>
<tr>
<td>GCOMPILE</td>
<td>Compile M-code directly into a single CUDA kernel.</td>
<td>[B C ...] = my_fn(A)</td>
</tr>
<tr>
<td>GPROFILE</td>
<td>Profile code to compare CPU versus GPU runtimes.</td>
<td>gprofile on; foo; gprofile off; gprofile report;</td>
</tr>
<tr>
<td>GPROFVIEW</td>
<td>Visual representation of profiling data.</td>
<td>gprofview;</td>
</tr>
<tr>
<td>GEVAL</td>
<td>Evaluate computation and leave results on GPU.</td>
<td>geval;</td>
</tr>
<tr>
<td>GSYNC</td>
<td>Block until all queued GPU computation is complete.</td>
<td>gsync(A); gcache;</td>
</tr>
<tr>
<td>GCACHE</td>
<td>Save GPU compiled code for given script.</td>
<td>gload('filename');</td>
</tr>
<tr>
<td>GLOAD</td>
<td>Load from disk directly into the GPU. Requires the Jacket SDK.</td>
<td>gsave('filename', 'filename');</td>
</tr>
<tr>
<td>GSAVE</td>
<td>Save data to disk as text file directly from the GPU. Requires the Jacket SDK.</td>
<td>gread('filename', OFFSET, BYTES);</td>
</tr>
<tr>
<td>GREAD</td>
<td>Load from disk directly into the GPU, with option to specify the byte range. Requires the Jacket SDK.</td>
<td>gwrite('filename', OFFSET, DATA); gplot(A);</td>
</tr>
<tr>
<td>Graphics</td>
<td>Library Functions contained in the Graphics Library.</td>
<td></td>
</tr>
</tbody>
</table>

---

This page is from the University of Minnesota's Driven to Discover publication.
Find the **hotspot** of your code

**mlint** - Check M-files for possible problems

```
>> mlint lengthofline.m  % Display to command line
>> info = mlint('lengthofline') % Store to struct
```

**tic/toc** - accurate timing of each operation/function

```
>> tic; a=rand(1000); toc;
```

**Matlab Profiler** - find where the bottle neck is

```
>> profile on
>> calcultion;
>> profile off
>> profile report
```

**http://www.mathworks.com/contest/protein.cgi/jitstory.html**
Nx = 20;
n = 20;
Df = zeros(n,Nx);
X = ones(n, Nx);
for ii = 1:Nx
    Df(1,ii) = X(1,ii);
    Df(2,ii) = X(2,ii);
end

Option 2
Nx = 20; n = 20;
Df = gzeros(n,Nx);
gfor ii = 1:Nx
    Df(1,ii) = X(1,ii);
    Df(2,ii) = X(2,ii);
gend

Option 1:
Nx = 20; n = 20;
Df = gzeros(n,Nx);
for ii = 1:Nx
    Df(1,ii) = X(1,ii);
    Df(2,ii) = X(2,ii);
end
Jacket examples: gfor

\begin{verbatim}
A = gones(n,n,m);
B = gones(n);
gfor k = 1:2:m
    A(:,:,k) = k*B + sin(k+1);  % expressions
Gend

A = gones(n,2*m);
B = gones(n,m);
gfor k = 2:m
    B(:,k) = A(:,floor(k+.2));
Gend
\end{verbatim}
N = 128*2; % matrix size
M = 256; % number of tiled matrices
%Create Data
tic
[Ac Bc]= ...
deal(complex(ones(N,N,M,'single'),0));
toc
% Compute 200 (128x128) FFTs
tic
for ii = 1:M
    Ac(:,:,ii) = fft2(Bc(:,:,ii));
end
Toc
%Elapsed time
%Elapsed time.
Restriction of gfor

- Iteration independence
- No conditional statements
- No cell array assignment
- Iterator not allowed in colon expressions

Hands-on Exercise:

Get the tar file

www.msi.umn.edu/~szhang/GPU_Tools.tar

Tar -xvf GPU_Tools.tar
module load jacket
Matlab &
On matlab window
<< fft_cpu
<< fft-gpu
References:

http://www.accelereyes.com/support/documentation

Need help?
help@msi.umn.edu
Or call @612-624-8858 or 612-626-0802