Approaches to GPU Computing

Libraries, OpenACC Directives, and Languages
Add GPUs: Accelerate Applications
GPUs Accelerate Science
Small Changes, Big Speed-up

Use GPU to Parallelize Compute-Intensive Functions

Rest of Sequential CPU Code
Why more than one approach?
3 Ways to Accelerate Applications

Applications

Libraries
“Drop-in” Acceleration

OpenACC Directives
Easily Accelerate Applications

Programming Languages
Maximum Performance
Easy, High-Quality Acceleration

- **Ease of use:** Using libraries enables GPU acceleration without in-depth knowledge of GPU programming

- **“Drop-in”:** Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes

- **Quality:** Libraries offer high-quality implementations of functions encountered in a broad range of applications

- **Performance:** NVIDIA libraries are tuned by experts
GPU Accelerated Libraries
“Drop-in” Acceleration for your Applications

- NVIDIA cuBLAS
- NVIDIA cuRAND
- NVIDIA cuSPARSE
- NVIDIA NPP
- GPU VSIPL
- CULA tools
- MAGMA
- NVIDIA cuFFT
- Rogue Wave Software
- ArrayFire
- CUSP
- Thrust

Vector Signal Image Processing
GPU Accelerated Linear Algebra
Matrix Algebra on GPU and Multicore
IMSL Library
ArrayFire Matrix Computations
Sparse Linear Algebra
C++ STL Features for CUDA
3 Steps to CUDA-accelerated application

**Step 1:** Substitute library calls with equivalent CUDA library calls

```
saxpy ( ... ) ➤ cublasSaxpy ( ... )
```

**Step 2:** Manage data locality

- **with CUDA:** `cudaMalloc()`, `cudaMemcpy()`, etc.
- **with CUBLAS:** `cublasAlloc()`, `cublasSetVector()`, etc.

**Step 3:** Rebuild and link the CUDA-accelerated library

```
nvcc myobj.o -l cublas
```
int N = 1 << 20;

// Perform SAXPY on 1M elements: d_y[] = a*d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

Add “cublas” prefix and use device variables
```c
int N = 1 << 20;
cublasInit();
cublasAlloc(N, sizeof(float), (void**)&d_x);
cublasAlloc(N, sizeof(float), (void**)&d_y);

cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

// Perform SAXPY on 1M elements: d_y[] = a*d_x[] + d_y[]
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);

cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);

cublasFree(d_x);
cublasFree(d_y);
cublasShutdown();
```
Explore the CUDA (Libraries) Ecosystem

CUDA Tools and Ecosystem described in detail on NVIDIA Developer Zone:

developer.nvidia.com/cuda-tools-ecosystem
3 Ways to Accelerate Applications

- Applications
- Libraries
  - “Drop-in” Acceleration
- OpenACC Directives
  - Easily Accelerate Applications
- Programming Languages
  - Maximum Performance
OpenACC Directives

Program myscience
... serial code ...
$acc kernels
  do k = 1,n1
  do i = 1,n2
    ... parallel code ...
  enddo
  enddo
$acc end kernels
... End Program myscience

Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

Your original Fortran or C code
Familiar to OpenMP Programmers

```
main() {
    double pi = 0.0; long i;
    #pragma omp parallel for reduction(+:pi)
    for (i=0; i<N; i++)
    {
        double t = (double)((i+0.05)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi = %f\n", pi/N);
}
```

```
main() {
    double pi = 0.0; long i;
    #pragma acc kernels
    for (i=0; i<N; i++)
    {
        double t = (double)((i+0.05)/N);
        pi += 4.0/(1.0+t*t);
    }
    printf("pi = %f\n", pi/N);
}
```
OpenACC
Open Programming Standard for Parallel Computing

“OpenACC will enable programmers to easily develop portable applications that maximize the performance and power efficiency benefits of the hybrid CPU/GPU architecture of Titan.”

--Buddy Bland, Titan Project Director, Oak Ridge National Lab

“OpenACC is a technically impressive initiative brought together by members of the OpenMP Working Group on Accelerators, as well as many others. We look forward to releasing a version of this proposal in the next release of OpenMP.”

--Michael Wong, CEO OpenMP Directives Board
Easy: Directives are the easy path to accelerate compute intensive applications

Open: OpenACC is an open GPU directives standard, making GPU programming straightforward and portable across parallel and multi-core processors

Powerful: GPU Directives allow complete access to the massive parallel power of a GPU
Directives: Easy & Powerful

Real-Time Object Detection
Global Manufacturer of Navigation Systems

Valuation of Stock Portfolios using Monte Carlo
Global Technology Consulting Company

Interaction of Solvents and Biomolecules
University of Texas at San Antonio

5x in 40 Hours  2x in 4 Hours  5x in 8 Hours

“Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications.”

-- Developer at the Global Manufacturer of Navigation Systems
A Very Simple Exercise: SAXPY

**SAXPY in C**

```c
void saxpy(int n,
       float a,
       float *x,
       float *restrict y)
{
    #pragma acc kernels
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}
...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...```

**SAXPY in Fortran**

```fortran
subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
$!acc kernels
    do i=1,n
        y(i) = a*x(i)+y(i)
    enddo
$!acc end kernels
end subroutine saxpy
...
$ Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...```
OpenACC Specification and Website

- Full OpenACC 1.0 Specification available online
  
  http://www.openacc.org

- Implementations available now from PGI and CAPS
Start Now with OpenACC Directives

Sign up for a free trial of the directives compiler now!

Free trial license to PGI Accelerator

Tools for quick ramp

www.nvidia.com/gpudirectives
3 Ways to Accelerate Applications

- Libraries
  - “Drop-in” Acceleration

- OpenACC Directives
  - Easily Accelerate Applications

- Programming Languages
  - Maximum Performance
void saxpy_serial(int n,  
    float a,  
    float *x,  
    float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);

__global__
void saxpy_parallel(int n,  
    float a,  
    float *x,  
    float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096,256>>>(n,2.0,x,y);

CUDA Fortran

- Program GPU using Fortran
  - Key language for HPC
- Simple language extensions
  - Kernel functions
  - Thread / block IDs
  - Device & data management
  - Parallel loop directives
- Familiar syntax
  - Use allocate, deallocate
  - Copy CPU-to-GPU with assignment (=)

```fortran
module mymodule
    contains
    subroutine saxpy(n,a,x,y)
        real :: x(:), y(:), a,
        integer n, i
        attributes(value) :: a, n
        i = threadIdx%x+(blockIdx%x-1)*blockDim%x
        if (i<=n) y(i) = a*x(i) + y(i);
    end subroutine saxpy
end module mymodule

program main
    use cudafor
    use mymodule
    real, device :: x_d(2**20), y_d(2**20)
    x_d = 1.0; y_d = 2.0
    call saxpy<<<4096,256>>>(2**20,3.0,x_d,y_d,)
    y = y_d
    write(*,*) 'max error=', maxval(abs(y-5.0))
end program main
```

http://developer.nvidia.com/cudafortran
CUDA C++ features enable sophisticated and flexible applications and middleware.

```cpp
template <typename T>
struct Functor {
    __device__ Functor(_a) : a(_a) {}
    __device__ T operator(T x) { return a*x; }
    T a;
};

template <typename T, typename Oper>
__global__ void kernel(T *output, int n) {
    Oper op(3.7);
    output = new T[n]; // dynamic allocation
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)
        output[i] = op(i); // apply functor
}
```

3+ Ways to Accelerate Applications

Applications

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

Programming Languages

Maximum Performance
Rapid Parallel C++ Development

- Resembles C++ STL
- High-level interface
  - Enhances developer productivity
  - Enables performance portability between GPUs and multicore CPUs
- Flexible
  - CUDA, OpenMP, and TBB backends
  - Extensible and customizable
  - Integrates with existing software
- Open source

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

// transfer data to device (GPU)
thrust::device_vector<int> d_vec = h_vec;

// sort data on device
thrust::sort(d_vec.begin(), d_vec.end());

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

Six Ways to SAXPY

Programming Languages for GPU Computing
Single precision **Alpha X Plus Y (SAXPY)**

Part of Basic Linear Algebra Subroutines (BLAS) Library

\[ z = \alpha x + y \]

\[ x, y, z : \text{vector} \]
\[ \alpha : \text{scalar} \]

GPU SAXPY in multiple languages and libraries

A menagerie* of possibilities, not a tutorial

*technically, a *program chrestomathy*: http://en.wikipedia.org/wiki/Chrestomathy
Parallel C Code

```c
void saxpy(int n,
    float a,
    float *x,
    float *y)
{
    #pragma acc kernels
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];

    ...

    // Perform SAXPY on 1M elements
    saxpy(1<<20, 2.0, x, y);
    ...
}
```

Parallel Fortran Code

```fortran
subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    !$acc kernels
    do i=1,n
        y(i) = a*x(i)+y(i)
    enddo
    !$acc end kernels
end subroutine saxpy

...
! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
CUBLAS Library

Serial BLAS Code

```
int N = 1<<20;
...
// Use your choice of blas library
// Perform SAXPY on 1M elements
blas_saxpy(N, 2.0, x, 1, y, 1);
```

Parallel cuBLAS Code

```
int N = 1<<20;

cublasInit();
cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);
// Perform SAXPY on 1M elements
cublasSaxpy(N, 2.0, d_x, 1, d_y, 1);
cublasGetVector(N, sizeof(y[0]), d_y, 1, y, 1);
cublasShutdown();
```

You can also call cuBLAS from Fortran, C++, Python, and other languages

http://developer.nvidia.com/cublas
### Standard C

```c
void saxpy(int n, float a, float *x, float *y) {
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

int N = 1<<20;

// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

### Parallel C

```c
__global__
void saxpy(int n, float a, float *x, float *y) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, d_x, d_y);
cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

```cpp
int N = 1<<20;
std::vector<float> x(N), y(N);
...

// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
                y.begin(), y.end(),
                2.0f * _1 + _2);
```

```cpp
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);
...

thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;

// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
                   d_y.begin(), d_y.begin(),
                   2.0f * _1 + _2);
```

www.boost.org/libs/lambda

http://thrust.github.com
module mymodule contains
  subroutine saxpy(n, a, x, y)
    real :: x(:), y(:), a
    integer :: n, i
    do i=1,n
      y(i) = a*x(i)+y(i)
    enddo
  end subroutine saxpy
end module mymodule

program main
  use mymodule
  real :: x(2**20), y(2**20)
  x = 1.0, y = 2.0
  ! Perform SAXPY on 1M elements
  call saxpy(2**20, 2.0, x, y)
end program main

module mymodule contains
  subroutine saxpy(n, a, x, y)
    attributes(global) real :: x(:), y(:), a
    attributes(value) integer :: n, i
    i = threadIdx%x+(blockIdx%x-1)*blockDim%x
    if (i<=n) y(i) = a*x(i)+y(i)
  end subroutine saxpy
end module mymodule

program main
  use cudafor; use mymodule
  real, device :: x_d(2**20), y_d(2**20)
  x_d = 1.0, y_d = 2.0
  ! Perform SAXPY on 1M elements
  call saxpy<<<4096,256>>>(2**20, 2.0, x_d, y_d)
end program main

import numpy as np

from copperhead import *
import numpy as np

@cu
def saxpy(a, x, y):
    return [a * xi + yi
            for xi, yi in zip(x, y)]

x = np.arange(2**20, dtype=np.float32)
y = np.arange(2**20, dtype=np.float32)

cpu_result = saxpy(2.0, x, y)

http://numpy.scipy.org

http://copperhead.github.com

Standard Python

Copperhead: Parallel Python
Enabling Endless Ways to SAXPY

Developers want to build front-ends for Java, Python, R, DSLs.

Target other processors like ARM, FPGA, GPUs, x86

CUDA Compiler Contributed to Open Source LLVM
## Recommended Approaches

<table>
<thead>
<tr>
<th>Area</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical analytics</td>
<td>MATLAB, Mathematica, LabVIEW</td>
</tr>
<tr>
<td>Fortran</td>
<td>OpenACC, CUDA Fortran</td>
</tr>
<tr>
<td>C</td>
<td>OpenACC, CUDA C</td>
</tr>
<tr>
<td>C++</td>
<td>Thrust, CUDA C++</td>
</tr>
<tr>
<td>Python</td>
<td>PyCUDA</td>
</tr>
<tr>
<td>C#</td>
<td>GPU.NET</td>
</tr>
</tbody>
</table>
How to get started

www.nvidia.com/cudazone

www.nvidia.com/getcuda
GPUs with fricken laserbeams!

- Created by Intellectual Ventures to help fight malaria in third world countries
- Image detection and targeting is done with NVIDIA GPUs
GTC 2013 | March 18-21 | San Jose, CA
The Smartest People. The Best Ideas. The Biggest Opportunities.

Opportunities for Participation:

**SPEAK** - Showcase your work among the elite of graphics computing
- Call for Sessions: August 2012
- Call for Posters: October 2012

**REGISTER** - learn from the experts and network with your peers
- Use promo code **GM10SIGG** for a 10% discount

**SPONSOR** - Reach influential IT decision-makers

Learn more at [www.gputechconf.com](http://www.gputechconf.com)