Introduction to GPU Computing
Add GPUs: Accelerate Science Applications
Small Changes, Big Speed-up

- Use GPU to Parallelize Compute-Intensive Functions
- Rest of Sequential CPU Code

Application Code
3 Ways to Accelerate Applications

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

- **Libraries**: “Drop-in” Acceleration
- **OpenACC Directives**: Easily Accelerate Applications
- **Programming Languages**: Maximum Flexibility
## Libraries: Easy, High-Quality Acceleration

<table>
<thead>
<tr>
<th>Ease of use:</th>
<th>Using libraries enables GPU acceleration without in-depth knowledge of GPU programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Drop-in”:</td>
<td>Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes</td>
</tr>
<tr>
<td>Quality:</td>
<td>Libraries offer high-quality implementations of functions encountered in a broad range of applications</td>
</tr>
<tr>
<td>Performance:</td>
<td>NVIDIA libraries are tuned by experts</td>
</tr>
</tbody>
</table>
Some GPU-accelerated Libraries

- 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
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

- Libraries
  - “Drop-in” Acceleration

- OpenACC Directives
  - Easily Accelerate Applications

- Programming Languages
  - Maximum Flexibility
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
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
OpenACC
The Standard for GPU Directives

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
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

Applications

Libraries
“Drop-in” Acceleration

OpenACC Directives
Easily Accelerate Applications

Programming Languages
Maximum Flexibility
GPU Programming Languages

**Numerical analytics**
- MATLAB, Mathematica, LabVIEW

**Fortran**
- OpenACC, CUDA Fortran

**C**
- OpenACC, CUDA C

**C++**
- Thrust, CUDA C++

**Python**
- PyCUDA, Copperhead

**C#**
- GPU.NET
CUDA C

### Standard C Code

```c
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);
```

### Parallel C Code

```c
__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 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
}
```

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

```cpp
// 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());
```

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 (=)

```
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
```

More Programming Languages

Python
- PyCUDA

C# .NET
- GPU.NET
- tidepowered

Numerical Analytics
- MATLAB
- Wolfram Mathematica 8
Get Started Today

These languages are supported on all CUDA-capable GPUs.
You might already have a CUDA-capable GPU in your laptop or desktop PC!

**CUDA C/C++**

**Thrust C++ Template Library**
http://developer.nvidia.com/thrust

**CUDA Fortran**

**GPU.NET**
http://tidepowerd.com

**MATLAB**
http://www.mathworks.com/discovery/matlab-gpu.html

**PyCUDA (Python)**
http://mathema.tician.de/software/pycuda

**Mathematica**
Thank you

developer.nvidia.com