Languages, Libraries and Development Tools for GPU Computing
GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems.

Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs.

“Jack Dongarra
Professor, University of Tennessee
Director of the Innovative Computing Laboratory
Author of LINPACK”
Small Changes, Big Speed-up

Use GPU to Parallelize Compute-Intensive Functions

Rest of Sequential CPU Code
In testing our key applications, the Tesla GPUs delivered speed-ups that we had never seen before, sometimes even orders of magnitude.

Satoshi Matsuoka
Professor, Global Scientific Information and Computing Center, Tokyo Institute of Technology
Technical Lead for the TSUBAME Supercomputer
CUDA Parallel Computing Platform

Programming Approaches
- Libraries: “Drop-in” Acceleration
- OpenACC Directives: Easily Accelerate Apps
- Programming Languages: Maximum Flexibility

Development Environment
- Nsight IDE
  - Linux, Mac and Windows
  - GPU Debugging and Profiling
- CUDA-GDB debugger
  - NVIDIA Visual Profiler

Open Compiler Tool Chain
- Enables compiling new languages to CUDA platform, and CUDA languages to other architectures

Hardware Capabilities
- SMX
- Dynamic Parallelism
- HyperQ
- GPUDirect

Programming Languages
- Code Optimization
- Interleaved Execution
3 Ways to Accelerate Applications

Libraries
“Drop-in” Acceleration

OpenACC Directives
Easily Accelerate Applications

Programming Languages
Maximum Flexibility
GPU Accelerated Libraries
“Drop-in” Acceleration for your Applications

NVIDIA cuBLAS
NVIDIA cuSPARSE
NVIDIA NPP
NVIDIA cuFFT

MAGMA
GPU Accelerated Linear Algebra
Matrix Algebra on GPU and Multicore

CULA tools
Vector Signal Image Processing

GPU VSIPL

ROGUE WAVE SOFTWARE
IMSL Library

CenterSpace NMath

ArrayFire
Building-block Algorithms

Thrust
C++ Templated Parallel Algorithms
cuBLAS Level 2 Performance

Up to 1 TFLOPS sustained performance and 7x faster than Intel MKL

Performance may vary based on system configuration

- cuBLAS 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz
ZGEMM Performance vs. Matrix Size

- cuBLAS 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz

Performance may vary based on system configuration
cuSPARSE: Sparse linear algebra routines

- Sparse matrix-vector multiplication & triangular solve
- Tri-diagonal solver up to 16x faster vs. Intel MKL
- Incomplete-LU & -Cholesky preconditioners (ilu0 and ic0)
- Format conversion: HYB, CSR, BlockCSR, COO, CSC, dense

\[
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix} = \alpha \begin{bmatrix}
1.0 & & & \\
2.0 & 3.0 & & \\
& & 4.0 & \\
5.0 & 6.0 & 7.0 &
\end{bmatrix} \begin{bmatrix}
1.0 \\
2.0 \\
3.0 \\
4.0 \\
\end{bmatrix} + \beta \begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
\end{bmatrix}
\]

developer.nvidia.com/cusparse
cuSPARSE: up to 12x Faster than MKL

Sparse Matrix x 6 Dense Vectors
(useful for block iterative solvers)

Performance may vary based on system configuration

- Average of s/d/c/z routines
- cuSPARSE 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz
NVIDIA Performance Primitives

- Thousands of GPU-accelerated image & signal processing functions
  Arithmetic, Logic, Conversions, Filters, Statistics, etc.
- Up to 40x faster performance than Intel IPP

developer.nvidia.com/content/graphcuts-using-npp

* NPP 4.1, NVIDIA C2050 (Fermi)
* IPP 6.1, Dual Socket Core™ i7 920 @ 2.67GHz
CenterSpace NMath
GPU-accelerated math for .NET languages

NMath 5.3 Premium Edition
- Dense matrix operations & Fast Fourier Transforms
- Singular Value Decomposition (SVD) and QR decomposition
- LU factorization, Least squares, and Eigenvalue routines

- No GPU programming experience required
- No changes to existing NMath code required
- Automatic fallback when there’s no GPU

Sign up at:
centerspace.net/nmath-premium-beta

2-4x faster than CPU-only
GPU-Aware MPI Libraries
Integrated Support for GPU Computing

GPUDirect™ P2P Transfers

GPUDirect™ RDMA

OpenMPI
MVAPICH
IBM Platform MPI
Cray MPI

developer.nvidia.com/gpudirect
// 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());

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

thrust.github.com or developer.nvidia.com/thrust
HEMI: Fully Portable C++ GPU Computing

- Functions compile and run on either CPU or GPU
- C++ classes can be used on CPU and GPU
- Simplified memory management
- Minimize platform-specific code

```
HEMI_KERNEL(solve)(float *out, float *n, int N) { ... }
HEMI_KERNEL_LAUNCH(solve, gDim, bDim, 0, 0, output, input, N);
```

github.com/harrism/hemi
3 Ways to Accelerate Applications

- **Libraries**: “Drop-in” Acceleration
- **OpenACC Directives**: Easily Accelerate Applications
- **Programming Languages**: Maximum Flexibility
Focus on Expressing Parallelism

With Directives, tuning work focuses on *expressing parallelism*, which makes codes inherently better.

Example: Application tuning work using directives for new Titan system at ORNL

- **S3D**
  Research more efficient combustion with next-generation fuels

- **CAM-SE**
  Answer questions about specific climate change adaptation and mitigation scenarios

  - Tuning top 3 kernels (90% of runtime)
  - 3 to 6x faster on CPU+GPU vs. CPU+CPU
  - But also improved all-CPU version by 50%

  - Tuning top key kernel (50% of runtime)
  - 6.5x faster on CPU+GPU vs. CPU+CPU
  - Improved performance of CPU version by 100%
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 Standard

www.nvidia.com/gpudirectives
A Very Simple Exercise: SAXPY

**SAXPY in C**

```c
void saxpy(int n,
    float a,
    float *x,
    float *restrict y)
{
    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(n), y(n), a
    integer :: n, i

    do i=1,n
        y(i) = a*x(i)+y(i)
    enddo
end subroutine saxpy
...
! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...```
A Very Simple Exercise: SAXPY OpenMP

SAXPY in C

```c
void saxpy(int n,
    float a,
    float *x,
    float *restrict y)
{
    #pragma omp parallel for
    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(n), y(n), a
    integer :: n, i

    !$omp parallel do
    do i=1,n
        y(i) = a*x(i)+y(i)
    enddo
    !$omp end parallel do
end subroutine saxpy

... ! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
```
void saxpy(int n,
    float a,
    float *x,
    float *restrict y)
{
    #pragma acc parallel loop
    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);
...

A Very Simple Exercise: SAXPY OpenACC

SAXPY in C

SAXPY in Fortran

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

...  ! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
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
3 Ways to Accelerate Applications

Applications

Libraries
“Drop-in” Acceleration

OpenACC Directives
Easily Accelerate Applications

Programming Languages
Maximum Flexibility
Opening the CUDA Platform with LLVM

CUDA compiler source contributed to open source LLVM compiler project

SDK includes specification documentation, examples, and verifier

Anyone can add CUDA support to new languages and processors

Learn more at developer.nvidia.com/cuda-llvm-compiler
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);

developer.nvidia.com/cuda-toolkit
CUDA C++: Develop Generic Parallel Code

CUDA C++ features enable sophisticated and flexible applications and middleware.

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

developer.nvidia.com/cuda-toolkit
PGI CUDA Fortran

- Simple Fortran 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
  attributes(global) 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

developer.nvidia.com/cuda-fortran
Compile Python for Parallel Architectures

- Anaconda Accelerate from Continuum Analytics
  - NumbaPro array-oriented compiler for Python & NumPy
  - Compile for CPUs or GPUs (uses LLVM + NVIDIA Compiler SDK)

- Fast Development + Fast Execution: Ideal Combination

Free Academic License

http://continuum.io
CUDA Python with NumbaPro

<table>
<thead>
<tr>
<th></th>
<th>1024² Mandelbrot Time</th>
<th>Speedup v. Pure Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Python</td>
<td>4.85s</td>
<td>--</td>
</tr>
<tr>
<td>NumbaPro (CPU)</td>
<td>0.11s</td>
<td>44x</td>
</tr>
<tr>
<td>CUDA Python (K20)</td>
<td>0.004s</td>
<td>1221x</td>
</tr>
</tbody>
</table>

```python
def mandel(x, y, max_iters):
    zr, zi = 0.0, 0.0
    for i in range(max_iters):
        newzr = (zr*zr-zi*zi)+x
        zi = 2*zr*zi+y
        zr = newzr
        if (zr*zr+zi*zi) >= 4:
            return i
    return 255

@cuda.jit(argtypes=[uint8[:, :], f8, f8, f8, f8, uint32])
def mandel_kernel(img, xmin, xmax, ymin, ymax, iters):
    x, y = cuda.grid(2)
    if x < img.shape[0] and y < img.shape[1]:
        img[y, x] = mandel(min_x+x*((max_x-min_x)/img.shape[0]),
                           min_y+y*((max_y-min_y)/img.shape[1]), iters)

gimage = np.zeros((1024, 1024), dtype = np.uint8)
d_image = cuda.to_device(gimage)
mandel_kernel[(32, 32), (32, 32)](d_image, -2.0, 1.0, -1.0, 1.0, 20)
d_image.to_host()```
## GPU Programming Languages

<table>
<thead>
<tr>
<th>Category</th>
<th>Languages</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>CUDA C++, Thrust, Hemi, ArrayFire</td>
</tr>
<tr>
<td>Python</td>
<td>Anaconda Accelerate, PyCUDA</td>
</tr>
<tr>
<td>.NET</td>
<td>CUDAfy.NET, Alea.cuBase</td>
</tr>
</tbody>
</table>

NVIDIA® Nsight™ Eclipse Edition for Linux and MacOS

CUDA-Aware Editor
- Automated CPU to GPU code refactoring
- Semantic highlighting of CUDA code
- Integrated code samples & docs

Nsight Debugger
- Simultaneously debug CPU and GPU
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging

Nsight Profiler
- Quickly identifies performance issues
- Integrated expert system
- Source line correlation

developer.nvidia.com/nsight
NVIDIA® Nsight™ Visual Studio Edition

CUDA Debugger
- Debug CUDA kernels directly on GPU hardware
- Examine thousands of threads executing in parallel
- Use on-target conditional breakpoints to locate errors

CUDA Memory Checker
- Enables precise error detection

System Trace
- Review CUDA activities across CPU and GPU
- Perform deep kernel analysis to detect factors limiting maximum performance

CUDA Profiler
- Advanced experiments to measure memory utilization, instruction throughput and stalls
Debugging Solutions
Command Line to Cluster-Wide

- NVIDIA Nsight Eclipse & Visual Studio Editions
- NVIDIA CUDA-GDB for Linux & Mac
- NVIDIA CUDA-MEMCHECK for Linux & Mac
- Allinea DDT with CUDA Distributed Debugging Tool
- TotalView for CUDA for Linux Clusters

developer.nvidia.com/debugging-solutions
Performance Analysis Tools
Single Node to Hybrid Cluster Solutions

NVIDIA Nsight
Eclipse & Visual Studio Editions

NVIDIA Visual Profiler

Vampir Trace Collector

TAU Performance System

PAPI CUDA Component

Under Development

developer.nvidia.com/performance-analysis-tools
Job Scheduling & Cluster Management

- IBM Platform Computing: LSF, HPC, Cluster Manager
- Bright Computing: Bright Cluster Manager
- Adaptive Computing
- PBS Works: PBS Professional
- Ganglia: NVML Plugin for GPUs
- Univa: Univa Grid Engine

developer.nvidia.com/cuda-tools-ecosystem
**GPU Management: nvidia-smi**

Command-line access to:
- Approximate GPU utilization
- Approximate memory footprint
- Number of GPUs
- ECC state
- Driver version

Inspect and modify GPU state
- ECC mode
- Exclusive use mode

Included in the NVIDIA driver, also available via NVML API

---

<table>
<thead>
<tr>
<th>GPU Name</th>
<th>Bus-Id</th>
<th>Disp.</th>
<th>Volatile Uncorr. ECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Tesla K20X</td>
<td>0000:03:00.0</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>N/A 30C P8 28W / 235W</td>
<td>0% 12MB / 6143MB</td>
<td>0% Default</td>
<td></td>
</tr>
<tr>
<td>1 Tesla K20X</td>
<td>0000:85:00.0</td>
<td>Off</td>
<td>Off</td>
</tr>
<tr>
<td>N/A 28C P8 26W / 235W</td>
<td>0% 12MB / 6143MB</td>
<td>0% Default</td>
<td></td>
</tr>
</tbody>
</table>

Compute processes:
<table>
<thead>
<tr>
<th>GPU Memory</th>
</tr>
</thead>
</table>

No running compute processes found
Online Resources

- UDACITY: www.udacity.com
- NVIDIA Developer Zone: developer.nvidia.com
- NVIDIA Documentation: docs.nvidia.com
- NVIDIA DevTalk: devtalk.nvidia.com
- NVIDIA Software Development Kit (CUDA, C++)
- NVIDIAicides.com
- NVIDIA CUDA Toolkit: en.wikipedia.org/wiki/NVIDIA_CUDA_Toolkit
- NVIDIA CUDA Programming: Stack Overflow: www.stackoverflow.com
Thank You!