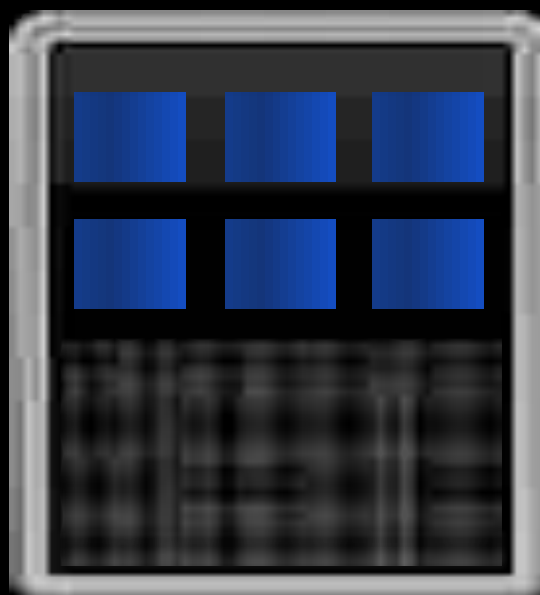




Approaches to GPU Computing

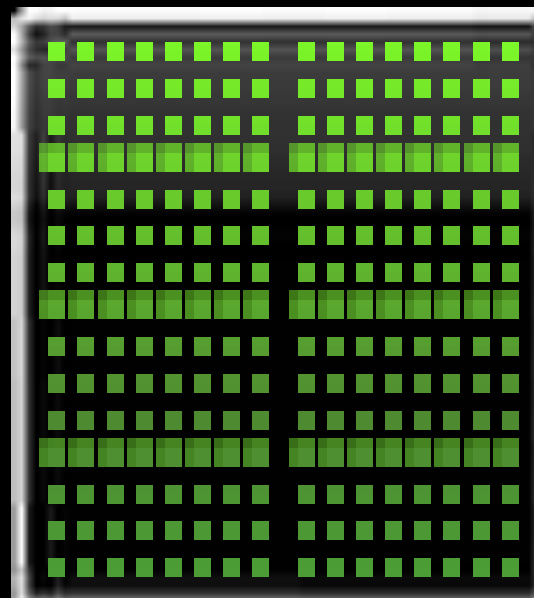


CPU



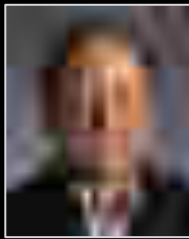
+

GPU



“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.”



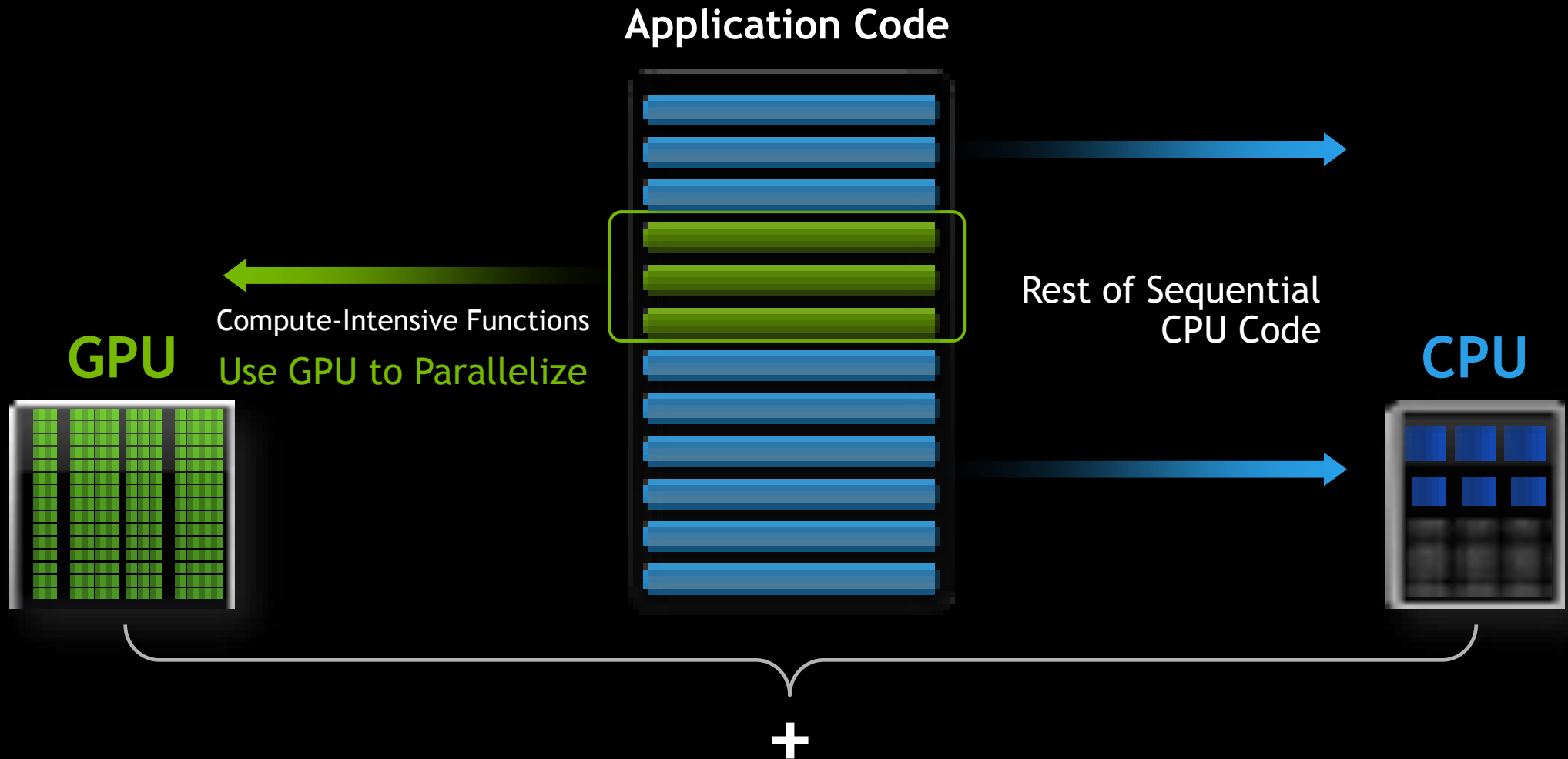
Jack Dongarra

Professor, University of Tennessee

Director of the Innovative Computing Laboratory

Author of LINPACK

Small Changes, Big Speed-up



CUDA Parallel Computing Platform

Programming
Approaches

Libraries

“Drop-in” Acceleration

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



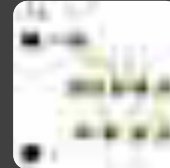
Enables compiling new languages to CUDA platform, and
CUDA languages to other architectures

Hardware
Capabilities

SMX



Dynamic Parallelism



HyperQ



GPUDirect



Accelerating Your Applications

Applications

Libraries

“Drop-in”
Acceleration

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



Matrix Algebra on
GPU and Multicore



GPU Accelerated
Linear Algebra



Vector Signal
Image Processing



NVIDIA cuRAND



IMSL Library



CenterSpace NMath



Building-block
Algorithms



C++ Templated
Parallel Algorithms

Thrust: Rapid Parallel C++ Development



- **High-level C++ Template Library**
 - Host and Device containers mimic STL
 - Enhances developer productivity
 - Enables performance portability
- **Flexible**
 - Backends for CUDA, OpenMP, TBB
 - Extensible and customizable
 - Integrates easily with existing code
- **Open source** ([thrust.github.com](https://github.com/thrust))

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


GPU-Aware MPI Libraries

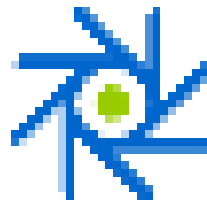
Integrated Support for GPU Computing



OpenMPI



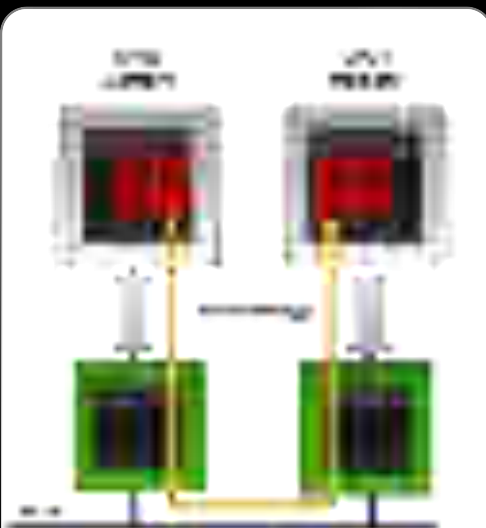
MVAPICH



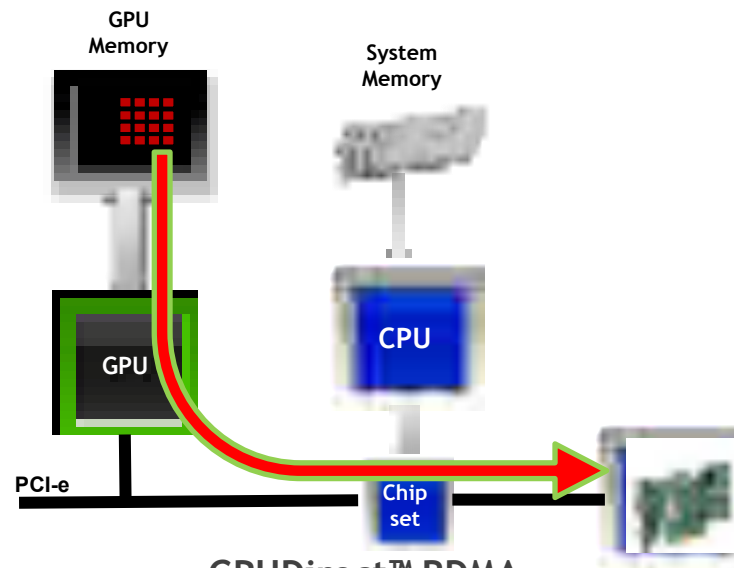
IBM Platform MPI



Cray MPI



GPUDirect™ P2P Transfers



GPUDirect™ RDMA

Accelerating Your Applications

Applications

Libraries

“Drop-in”
Acceleration

Directives

Easily Accelerate
Applications

Programming
Languages

Maximum
Flexibility

Accelerating Your Applications

Applications

Libraries

“Drop-in”
Acceleration

Directives

Easily Accelerate
Applications

Programming
Languages

Maximum
Flexibility

CUDA C

Standard C Code

```
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

```
__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++: Develop Generic Parallel Code

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

Class hierarchies

__device__ methods

Templates

Operator overloading

Functors (function objects)

Device-side new/delete

More...

```
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
}
```

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
```

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



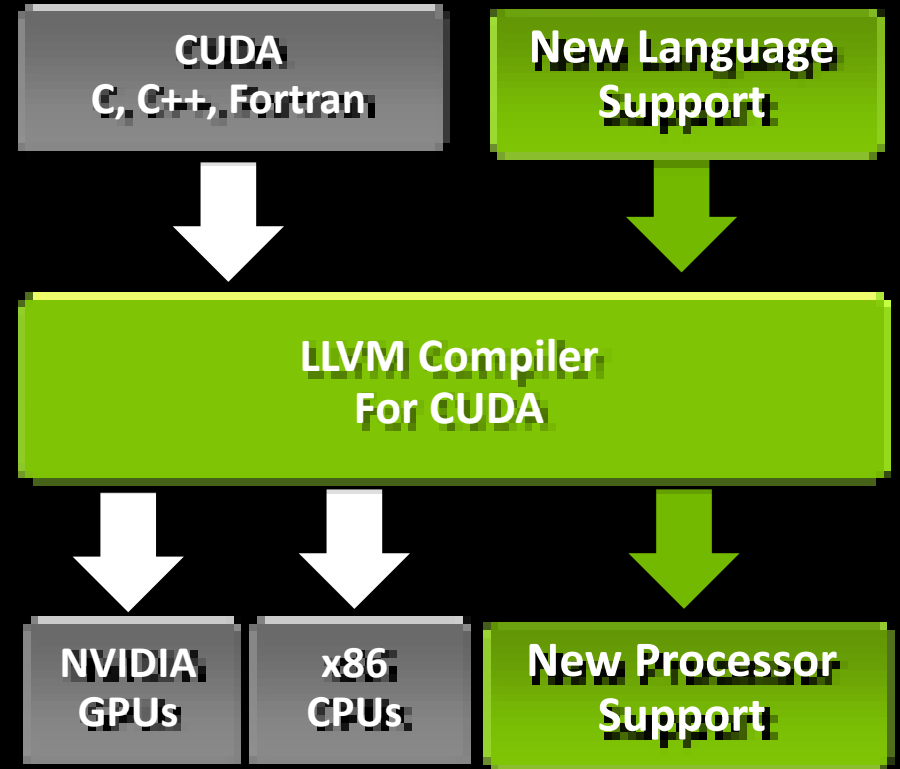
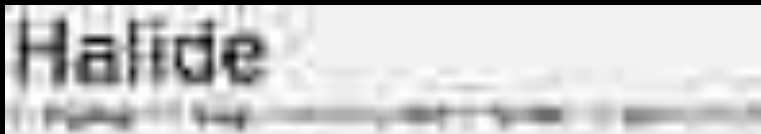
Free Academic License

www.continuum.io


Enabling More Programming Languages



Mozilla Rust



GPU Programming Languages

Numerical analytics 

MATLAB, Mathematica, LabVIEW

Fortran 

OpenACC, CUDA Fortran

C 

OpenACC, CUDA C

C++ 

CUDA C++, Thrust, ArrayFire

Python 

Anaconda Accelerate, PyCUDA

Domain-Specific 

CUDAfy.NET, Alea.cuBase

Development Tools

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

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



System Trace

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

CUDA Memory Checker

- Enables precise error detection



CUDA Profiler

- Advanced experiments to measure memory utilization, instruction throughput and stalls

Performance Analysis Tools

Single Node to Hybrid Cluster Solutions



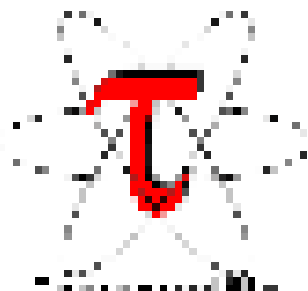
NVIDIA Nsight
Eclipse & Visual Studio Editions



NVIDIA Visual Profiler



Vampir Trace Collector



TAU Performance System



PAPI CUDA Component



Under Development

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

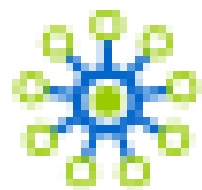


Allinea DDT with CUDA
Distributed Debugging Tool



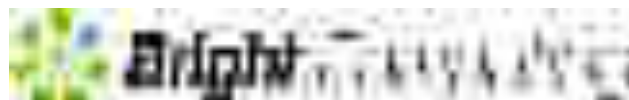
TotalView for CUDA
for Linux Clusters

Job Scheduling & Cluster Management



IBM
Platform
Computing

LSF, HPC, Cluster Manager



Bright Cluster Manager



PBS Works

PBS Professional



NVML Plugin for GPUs



Univa Grid Engine

GPU Management: nvidia-smi

Query

- GPU accounting & utilization metrics
- Power draw, limits
- Clock data (target, current, available)
- Serial Numbers and Version info

```
% nvidia-smi
Sat Mar 22 08:34:44 2014
+-----+
| NVIDIA-SMI 331.37      Driver Version: 331.37      |
+-----+
| GPU  Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+
|  0   Tesla K10.G2.8GB    On         | 0000:01:00.0    Off  |           0         |
| N/A   33C    P0         45W/ 117W | 150MiB/ 3583MiB |      0%      Default |
+-----+-----+
|  1   Tesla K10.G1.8GB    On         | 0000:02:00.0    Off  |           0         |
| N/A   30C    P8         23W/ 117W | 12MiB/ 3583MiB |      0%      Default |
+-----+-----+
```

Modify

- Target clocks
- Compute mode, ECC, persistence
- Power cap
- Reset GPU

```
+-----+
| Compute processes:                        GPU Memory |
| GPU      PID  Process name                  Usage        |
+-----+-----+
|    0      12587  ./nvidia-healthmon                  138MiB      |
+-----+-----+
```

Included in the NVIDIA drivers package, also available via NVML API

Thank You!

