



Fast GPU Development with CUDA Libraries

Levi Barnes



3 Ways to Accelerate Applications



Applications

Libraries

“Drop-in”
Acceleration

OpenACC
Directives

Easily Accelerate
Applications

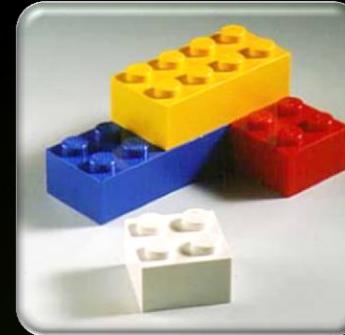
Programming
Languages

Maximum
Flexibility

NVIDIA CUDA Library Approach



- Provide basic building blocks
 - Make them easy to use
 - Make them fast
-
- Provides a quick path to GPU acceleration
 - Enables developers to focus on their “secret sauce”
 - Ideal for applications that use CPU libraries



CUDA Math Libraries



High performance math routines for your applications:

- cuFFT – Fast Fourier Transforms Library
- cuBLAS – Complete BLAS Library
- cuSPARSE – Sparse Matrix Library
- cuRAND – Random Number Generation (RNG) Library
- NPP – Performance Primitives for Image & Video Processing
- Thrust – Templated C++ Parallel Algorithms & Data Structures
- math.h - C99 floating-point Library

Included in the CUDA Toolkit

Free download @ www.nvidia.com/getcuda

Linear Algebra

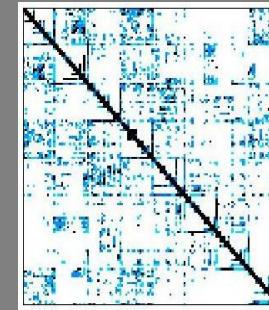


Dense



cuBLAS

Sparse



cuSPARSE



Drop-In Acceleration

```
int N = 1 << 20;
```

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



Drop-In Acceleration (Step 1)

```
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



Drop-In Acceleration (Step 2)

```
int N = 1 << 20;  
cublasInit();
```



Initialize CUBLAS

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

```
cublasShutdown();
```



Shut down CUBLAS

Drop-In Acceleration (Step 3)

```
int N = 1 << 20;  
cublasInit();  
cublasAlloc(N, sizeof(float), (void**)&d_x);  
cublasAlloc(N, sizeof(float), (void*)&d_y);
```



Allocate device vectors

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

```
cublasFree(d_x);  
cublasFree(d_y);  
cublasShutdown();
```



Deallocate device vectors



Drop-In Acceleration (Step 4)

```
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();
```



Transfer data to GPU



Read data back GPU



cuBLAS Interface

```
err = idamax(n, col, 1, size);
```

```
err = dscal(n, val, row, 1);
```

```
dgemm('N','N',m,n,k,A,m,  
B,k,0.0,C,m)
```

```
err = cublasIdamax(hdl, n, col, 1, size);
```

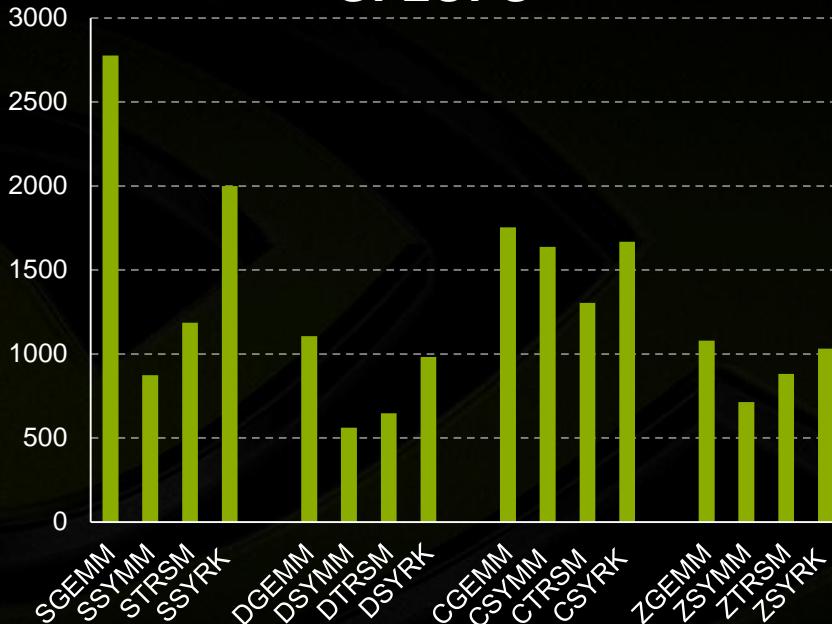
```
err = cublasDscal(hdl, n, val, row, 1);
```

```
cublasDgemm(hdl,'N','N',m,n,k,d_A,m,  
d_B,k,0.0,d_C,m)
```

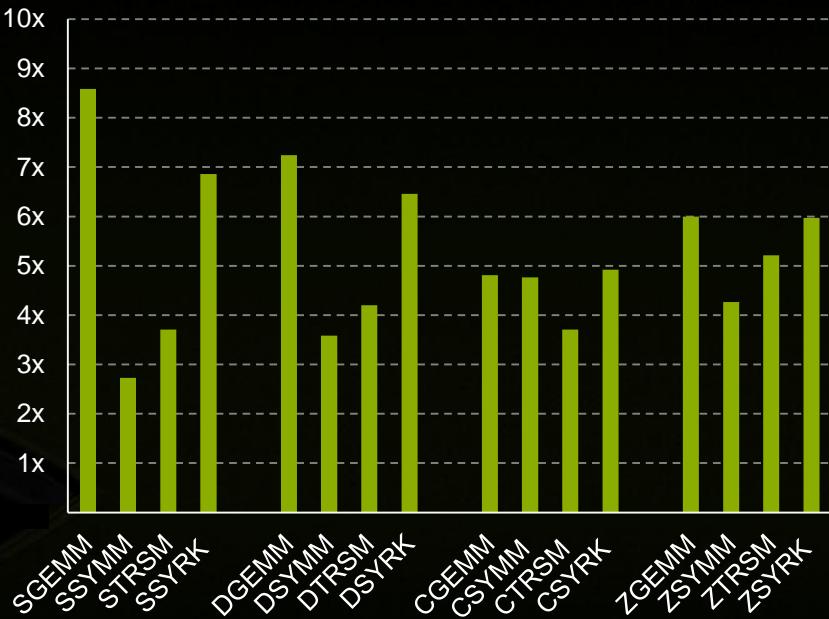
cuBLAS Level 3: >1 TFLOPS double-precision



GFLOPS



Speedup over MKL



- MKL 10.3.6 on Intel SandyBridge E5-2687W @3.10GHz
- CUBLAS 5.0.30 on K20X, input and output data on device

GPU-Callable Libraries



New in CUDA 5.0

Call cuBLAS library function from GPU code

Supported on K20 and K20x only

Encourages third party libraries



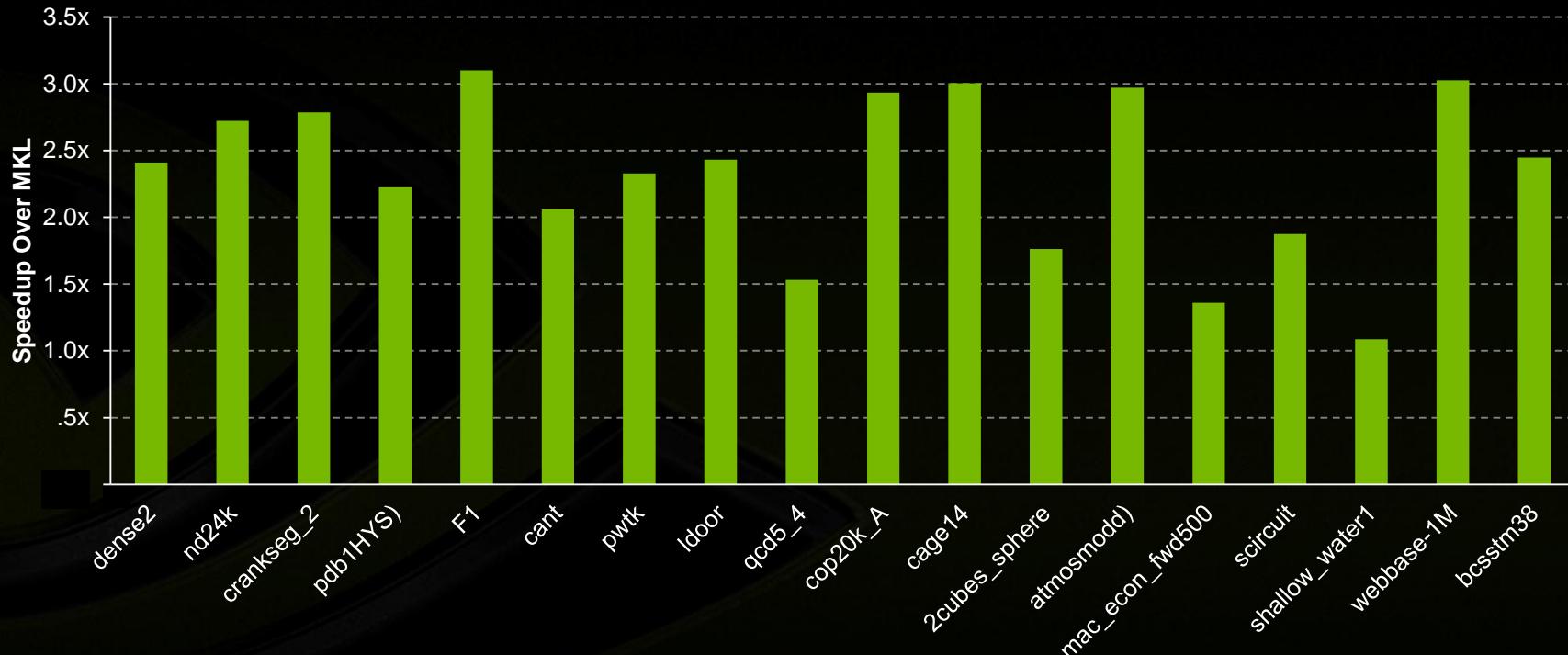
cuSPARSE Interface

```
mkl_dcsrmv(transa, m, k,  
           alpha, descr,  
           val, indx, pntrb, pntre,  
           x, beta, y);
```

```
err = cusparseDcsrmv(hdl, transa, m, k,  
                      nnz, alpha, desrc,  
                      val, indx, col,  
                      x, beta, y);
```

cuSPARSE performance

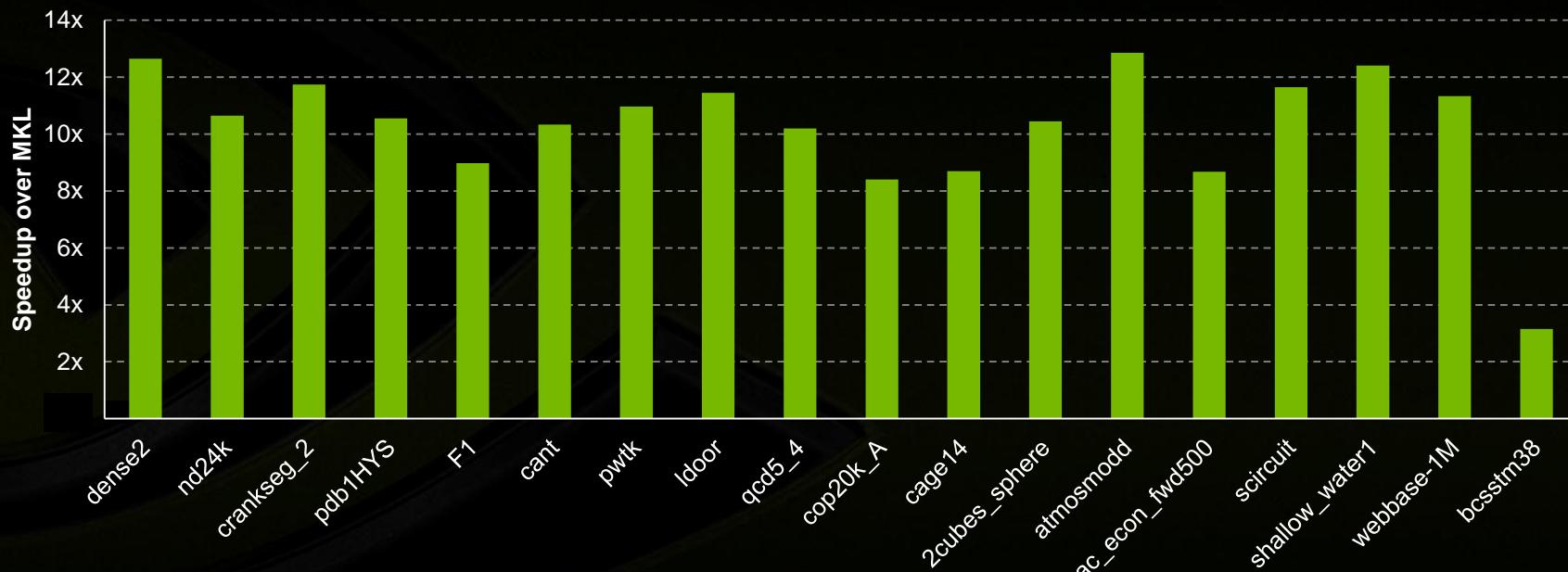
Sparse Matrix x Dense Vector



- 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

cuSPARSE: up to 12x Faster than MKL

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

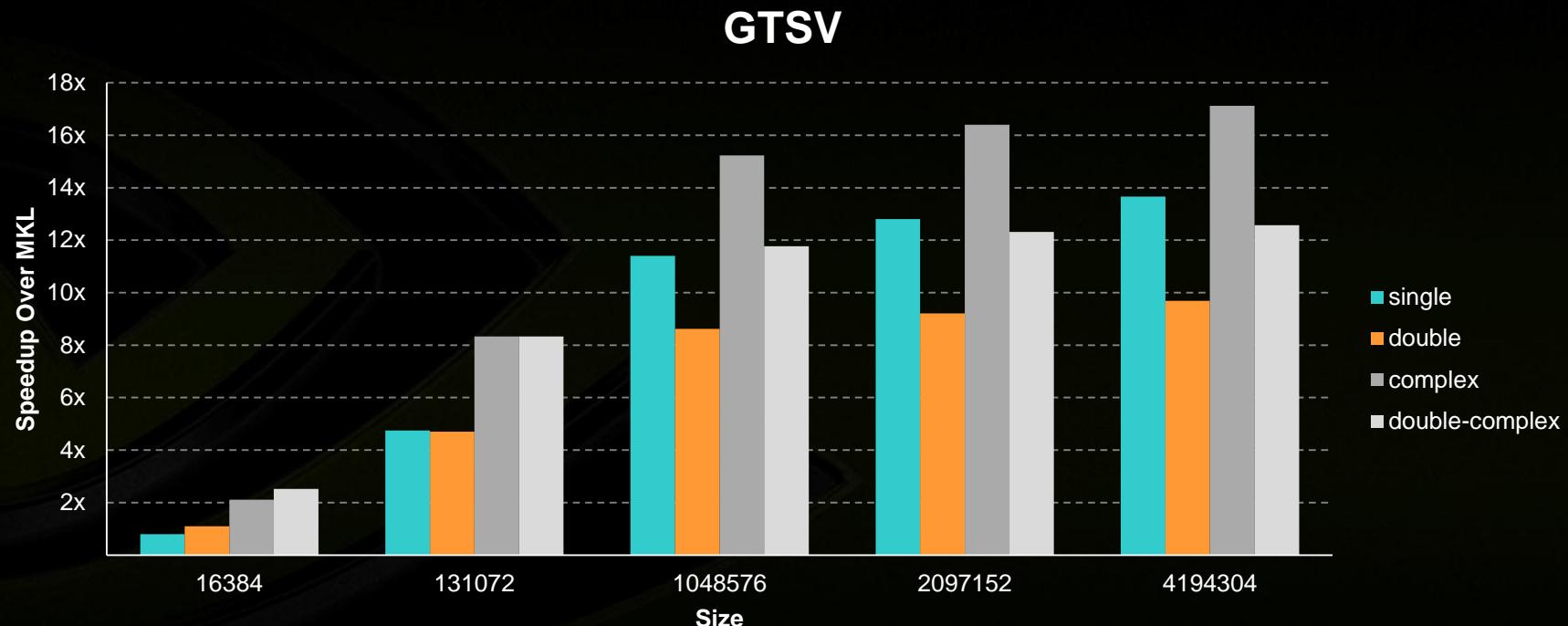


- Average of s/d/c/z routines
- cuSPARSE 5.0 on K20X, input and output data on device
- MKL 10.3.6 on Intel SandyBridge E5-2687W @ 3.10GHz

Tri-diagonal Solver Performance vs. MKL



Speedup for Tri-Diagonal solver (gtsv)



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

Different Approaches to Linear Algebra



- **CULA tools (dense, sparse)**
 - LAPACK based API
 - Solvers, Factorizations, Least Squares, SVD, Eigensolvers
 - Sparse: Krylov solvers, Preconditioners, support for various formats
- `culaSgetrf(M, N, A, LDA, IPIV, INFO)`



GPU Accelerated
Linear Algebra

- **ArrayFire**
 - Array container object
 - Solvers, Factorizations, SVD, Eigensolvers
- `array out = lu(A)`



AccelerEyes



Different Approaches to Linear Algebra (cont.)

▪ MAGMA

- LAPACK conforming API
- Magma BLAS and LAPACK
- High performance by utilizing both GPU and CPU

`magma_sgtrfs(M, N, A, LDA, IPIV, INFO)`

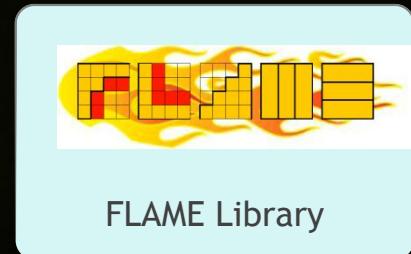


ICL

▪ LibFlame

- LAPACK compatibility interface
- Infrastructure for rapid linear algebra algorithm development

`FLASH_LU_piv(A, p)`



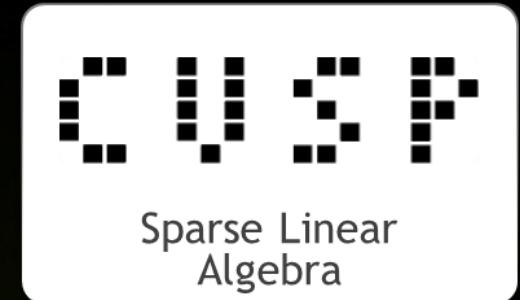
FLAME Library

UT-Austin

Different Approaches to Linear Algebra (cont.)

- CUSP
 - Sparse matrix operations
 - Open source
 - Supports COO, CSR, ELL, DIA, hybrid, etc.
 - Solvers, monitors, preconditioners, etc.

```
cusp::krylov::cg(A, x, b);
```



Toolkits are increasingly supporting GPUs



- PETSc
 - GPU support via extension to Vec and Mat classes
 - Partially dependent on CUSP
 - MPI parallel, GPU accelerated solvers

- Trilinos
 - GPU support in KOKKOS package
 - Used through vector class Tpetra
 - MPI parallel, GPU accelerated solvers



Signal Processing



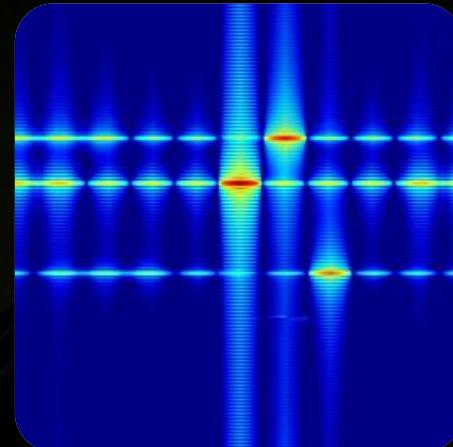
Common Tasks in Signal Processing



Filtering



Correlation

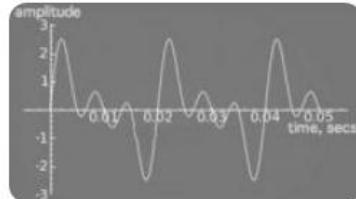


Segmentation





Libraries for GPU Accelerated Signal Processing



NVIDIA cuFFT



NVIDIA NPP

GPU VSIPL

Vector Signal
Image Processing



MATLAB

Parallel Computing
Toolbox



ArrayFire Matrix
Computations

GPULib

GPU Accelerated
Data Analysis

cuFFT



- Interface modeled after FFTW

```
fftw_plan PlanA;  
  
fftw_plan_dft_2d(N, M, &PlanA,  
                 data, data, FFT_FORWARD)  
  
fftw_execute_dft(PlanA, data,  
                 data);
```

```
cufftPlan2d PlanA;  
  
cufftCreatePlan(N, M, &PlanA, CUFFT_C2C);  
  
cufftExecC2C(PlanA, d_data, d_data,  
             CUFFT_FORWARD);
```

- Supports streams and batching (2 and 3-D, too!) for better performance

CUFFT: up to 600 GFLOPS

1D used in audio processing and as a foundation for 2D and 3D FFTs



cuFFT-Single Precision



cuFFT-Double Precision



- CUFFT 5.0.30 on K20X, input and output data on device

NPP features a large set of functions



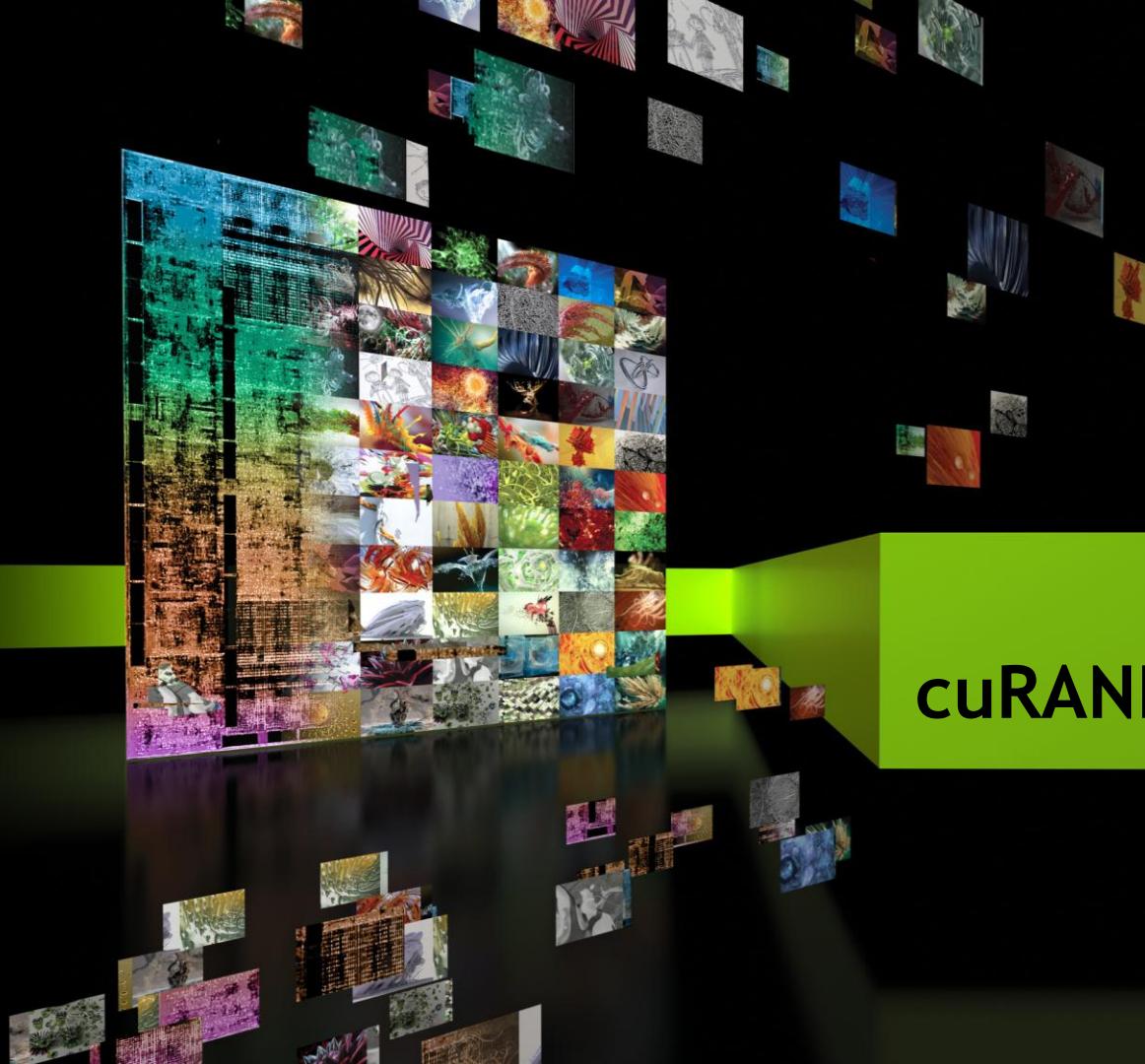
- **Arithmetic and Logical Operations**
 - Point-by-point ops, clamp, threshold, etc.
- **Geometric transformations**
 - Rotate, Warp, Interpolate
- **Compression**
 - jpeg de/compression
- **Image processing**
 - Filter, histogram, statistics



NVIDIA NPP

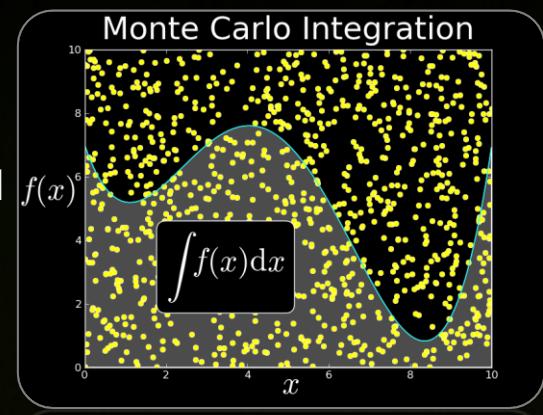


cuRAND



Random Number Generation on GPU

- Generating high quality random numbers in parallel is hard
 - Don't do it yourself, use a library!
- Large suite of generators and distributions
 - XORWOW, MRG323ka, MTGP32, (scrambled) Sobol
 - uniform, normal, log-normal
 - Single and double precision
- Two APIs for cuRAND
 - Called from CPU: Ideal when generating large batches of RNGs on GPU
 - Called from GPU: Ideal when RNGs need to be generated inside a kernel



cuRAND: Host vs Device API

- CPU API

```
#include "curand.h"  
curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_DEFAULT);  
curandGenerateUniform(gen, d_data, n);
```

Generate set of random numbers at once

- GPU API

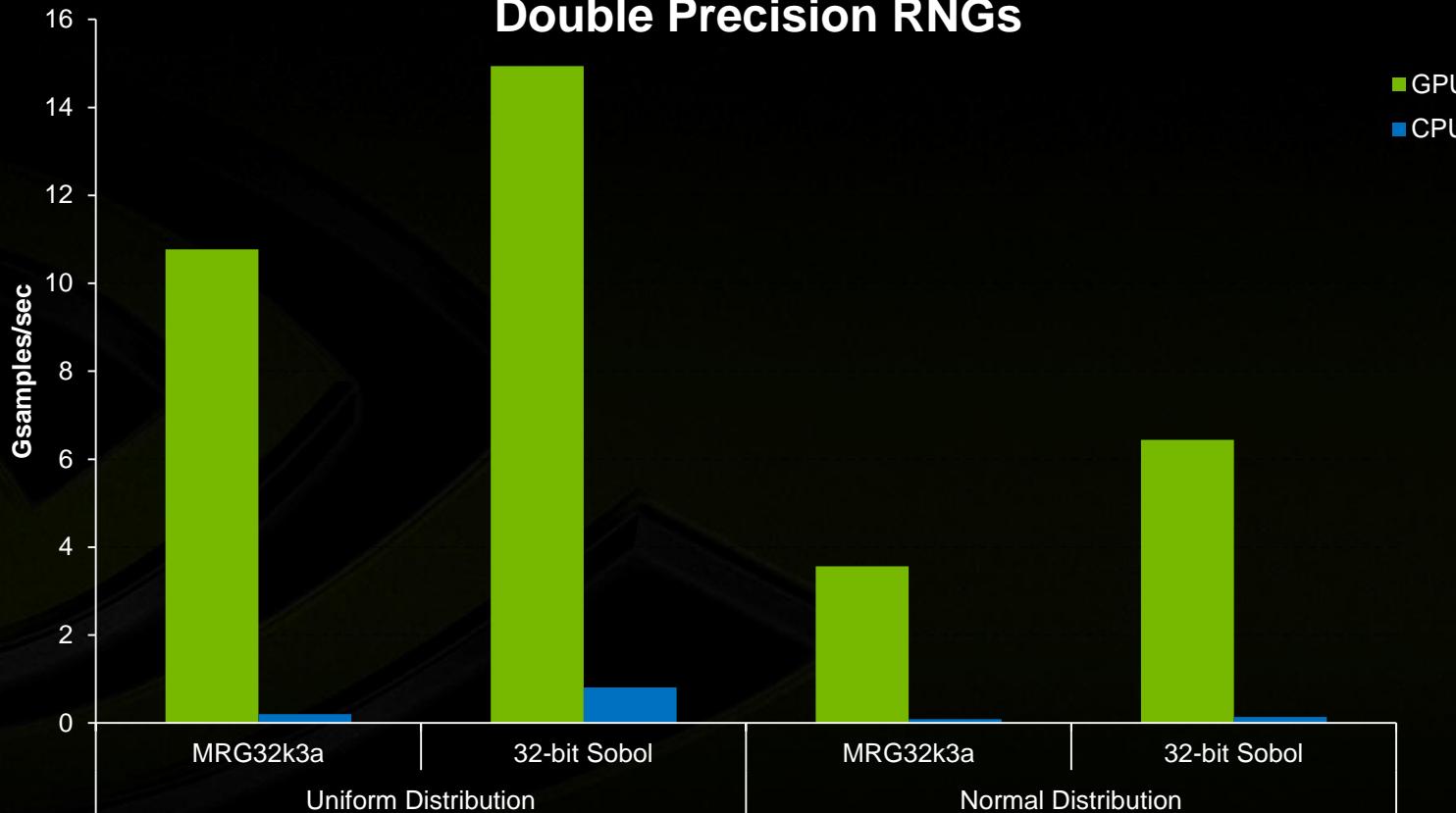
```
#include "curand_kernel.h"  
__global__ void generate_kernel(curandState *state) {  
    int id = threadIdx.x + blockIdx.x * 64;  
    x = curand(&state[id]);  
}
```

Generate random numbers per thread

cuRAND Performance



Double Precision RNGs



Thrust: STL-like CUDA Template Library



- **GPU(device) and CPU(host) vector class**

```
thrust::host_vector<float> H(10, 1.f);
thrust::device_vector<float> D = H;
```

- **Iterators**

```
thrust::fill(D.begin(), D.begin() + 5, 42.f);
float* raw_ptr = thrust::raw_pointer_cast(D);
```

- **Algorithms**

- **Sort, reduce, transformation, scan, ..**

```
thrust::transform(D1.begin(), D1.end(), D2.begin(), D2.end(),
thrust::plus<float>()); // D2 = D1 + D2
```



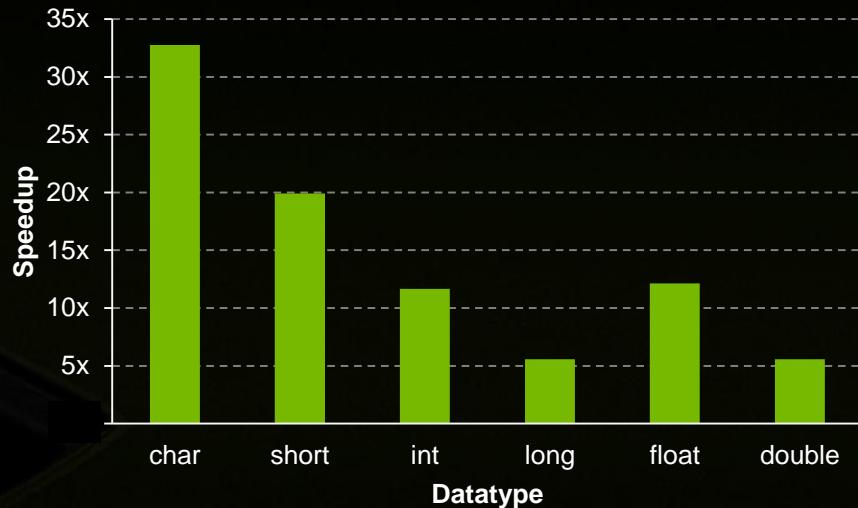
C++ STL Features
for CUDA

Thrust Performance

Various Algorithms (32M int.)
Speedup over TBB



Sort (32M samples)
Speedup over TBB



- Thrust 5.0 on K20X, input and output data on device
- TBB 4.1 on Intel SandyBridge E5-2687W @3.10GHz

math.h: C99 floating-point library + extras



CUDA math.h is **industry proven, high performance, accurate**

- Basic: +, *, /, 1/, sqrt, FMA (all IEEE-754 accurate for float, double, all rounding modes)
- Exponentials: exp, exp2, log, log2, log10, ...
- Trigonometry: sin, cos, tan, asin, acos, atan2, sinh, cosh, asinh, acosh, ...
- Special functions: lgamma, tgamma, erf, erfc
- Utility: fmod, remquo, modf, trunc, round, ceil, floor, fabs, ...
- Extras: rsqrt, rcp, exp10, sinpi, sincos, cospi, erfinv, erfcinv, ...
- New in CUDA 5.0
 - sincospi[f]() and normcdf[inv][f]()
 - sin(), cos() and erfcinvf() more accurate and faster
 - Full list of new features and optimizations:

<http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#math>

<http://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#math-performance-improvements>

CUDA libs with OpenACC



```
cufftExecPlan(plan, d_signal, d_signal)  
...  
#pragma acc data deviceptr(d_signal)  
#pragma acc loop independent  
for(i=0; i<n; i++) d_signal[i] = 2 * d_signal[i];
```

Explore the CUDA (Libraries) Ecosystem



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

developer.nvidia.com/cuda-tools-ecosystem

Log in | Feedback | New Account

Search

NVIDIA DEVELOPER ZONE

DEVELOPER CENTERS TECHNOLOGIES TOOLS RESOURCES COMMUNITY

QUICKLINKS

- The NVIDIA Registered Developer Program
- Registered Developers Website
- NVDeveloper (old site)
- CUDA Newsletter
- CUDA Downloads
- CUDA GPUs
- Get Started - Parallel Computing
- CUDA Spotlights
- CUDA Tools & Ecosystem

FEATURED ARTICLES

INTRODUCING NVIDIA INSIGHT VISUAL STUDIO EDITION 2.2, WITH LOCAL SINGLE GPU CUDA DEBUGGING!

Previous Next

LATEST NEWS

OpenACC Compiler For \$199

Introducing NVIDIA INSIGHT VISUAL STUDIO EDITION 2.2, WITH LOCAL SINGLE GPU CUDA DEBUGGING!

CUDA Spotlight: Lorena Barba, Boston University

Stanford To Host CUDA On Campus Day, April 13, 2012

CUDA Spotlight:

GPU-Accelerated Libraries

Adding GPU-acceleration to your application can be as easy as simply calling a library function. Check out the extensive list of high performance GPU-accelerated libraries below. If you would like other libraries added to this list please [contact us](#).

NVIDIA cuFFT

NVIDIA CUDA Fast Fourier Transform Library (cuFFT) provides a simple interface for computing FFTs up to 10x faster, without having to develop your own custom GPU FFT implementation.

NVIDIA cuBLAS

NVIDIA CUDA BLAS Library (cuBLAS) is a GPU-accelerated version of the complete standard BLAS library that delivers 6x to 17x faster performance than the latest MKL BLAS.

cUBLA tools

CUBLA Tools

GPU-accelerated linear algebra library by EM Photonics, that utilizes CUDA to dramatically improve the computation speed of sophisticated mathematics.

MAGMA

MAGMA

A collection of next gen linear algebra routines. Designed for heterogeneous GPU-based architectures. Supports current LAPACK and BLAS standards.

IMSL Fortran Numerical Library

Developed by RogueWave, a comprehensive set of mathematical and statistical functions that offloads work to GPUs.

NVIDIA cuSPARSE

NVIDIA CUDA Sparse (cuSPARSE)

Matrix library provides a collection of basic linear algebra subroutines used for sparse matrices that delivers over 8x performance boost.

NVIDIA CUSP

NVIDIA CUSP

A GPU accelerated Open Source C++ library of generic parallel algorithms for sparse linear algebra and graph computations. Provides an easy to use high-level interface.

AccelerEyes ArrayFire

AccelerEyes ArrayFire

Comprehensive GPU function library, including functions for math, signal, and image processing, statistics, and more. Interfaces for C, C++, Fortran, and Python.

NVIDIA curAND

NVIDIA curAND

The CUDA Random Number Generation library performs high quality GPU-accelerated random number generation (RNG) over 8x faster than typical CPU only code.

NVIDIA NPP

NVIDIA Performance Primitives

An industry proven, highly accurate collection of standard mathematical functions, providing high performance primitives for image processing.

NVIDIA CUDA Math Library

NVIDIA CUDA Math Library

An industry proven, highly accurate collection of standard mathematical functions, providing high performance primitives for image processing.

Thrust

Thrust

A powerful, open source library of parallel algorithms and data structures. Perform GPU-accelerated sort, scan, transform, and reductions.

GPU Technology Conference 2013

The Premier Event in Accelerated Computing

- Four days - March 18-21, San Jose, CA
- 300+ sessions
- 100+ research posters
- Networking opportunities with experts & colleagues from 40+ countries

It's everything you want to know about accelerated computing -- all in one place!

Visit www.nvidia.com/gtc-compute for more info.

Summary



- CUDA libraries offer high performance for minimal effort
- Robust community of 3rd party libraries
- Familiar interfaces make porting legacy code easy (“drop-in”)
- Enables focus on core IP

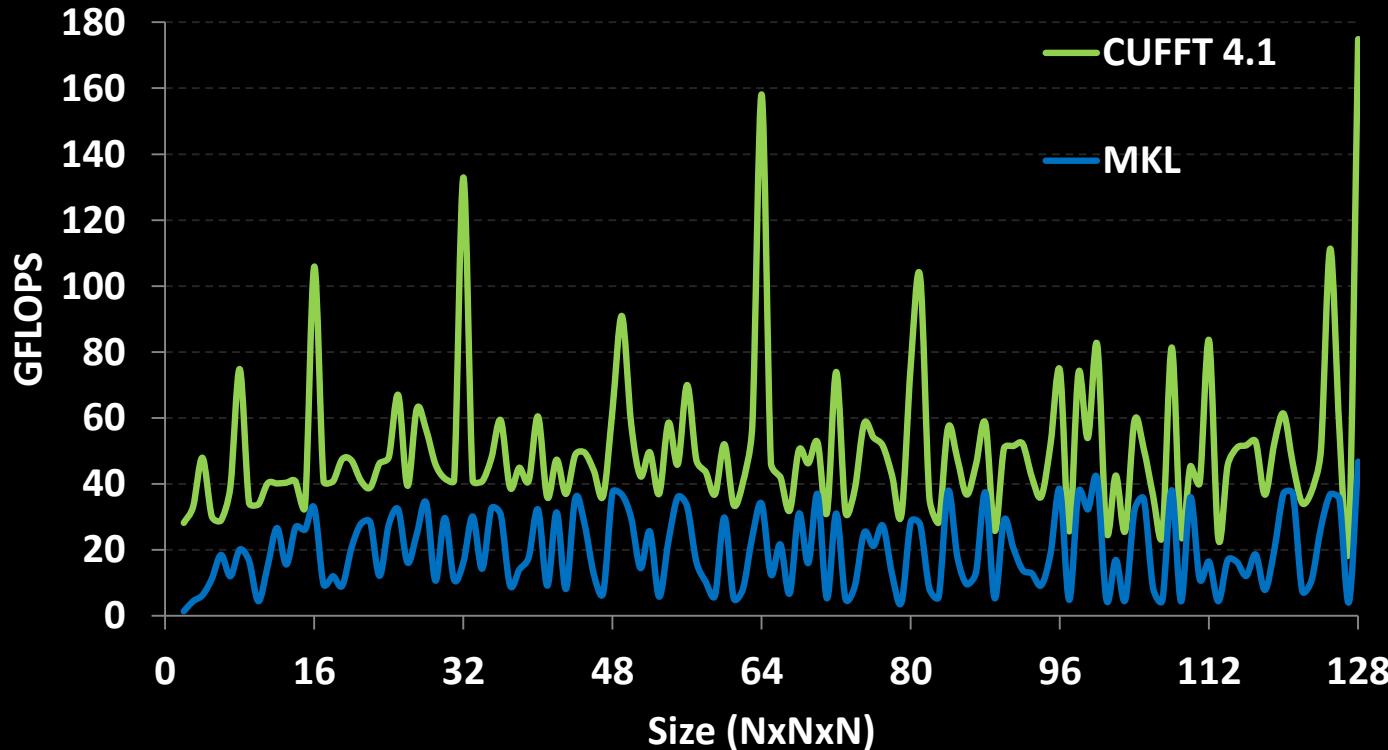
A large, semi-transparent NVIDIA logo watermark is positioned in the center-right area of the slide. It features the iconic green stylized eye icon followed by the word "NVIDIA" in a white, sans-serif font.

Thank you



Optimized 3D transforms

Single Precision All Sizes 2x2x2 to 128x128x128



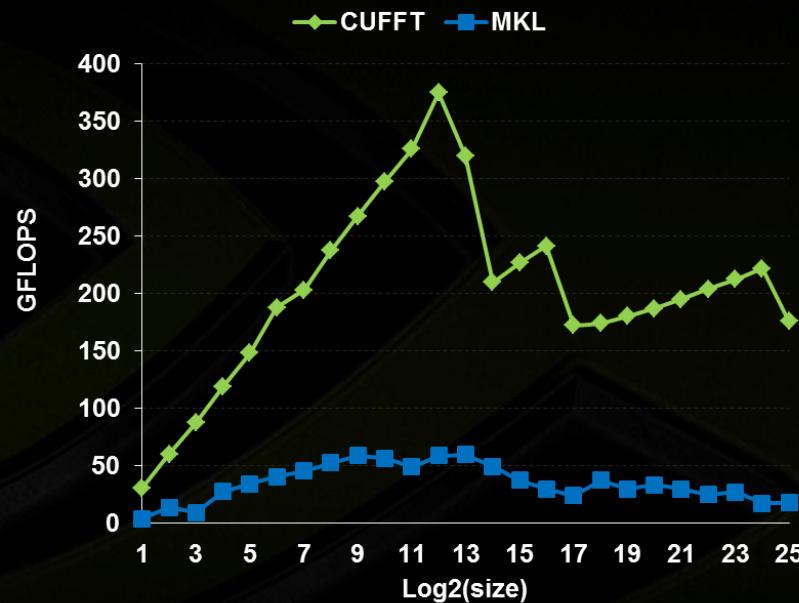
Performance may vary based on OS version and motherboard configuration
© NVIDIA Corporation 2012

- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

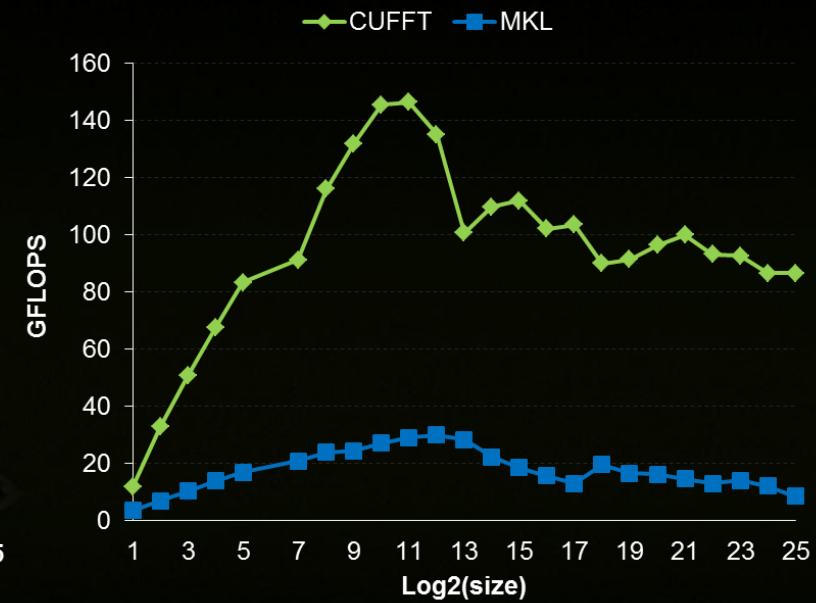
cuFFT Performance -1D



cuFFT Single Precision



cuFFT Double Precision



Performance may vary based on OS version and motherboard configuration

- Measured on sizes that are exactly powers-of-2
- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz