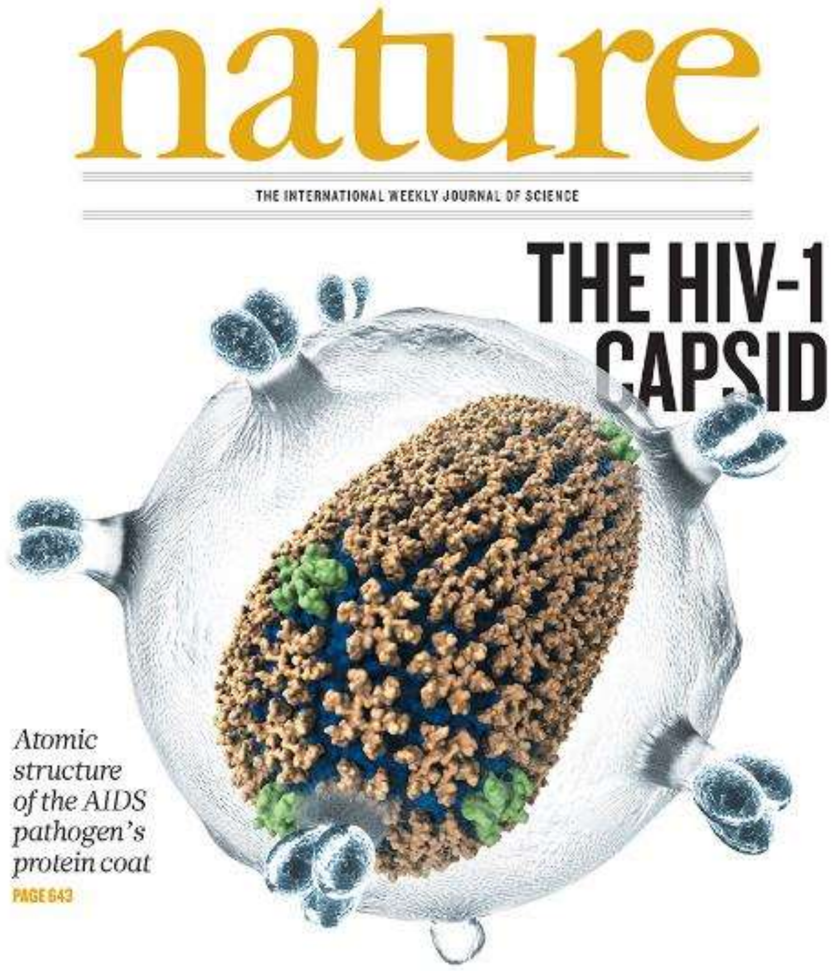
GPU TECHNOLOGY CONFERENCE

ACCELERATED COMPUTING WITH TESLA AND CUDA

MARK HARRIS, NVIDIA
LARRY BROWN, NVIDIA


nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
THE HIV-1 CAPSID
Atomic structure of the AIDS pathogen's protein coat
PAGE 643

ACCELERATING DISCOVERIES

USING A SUPERCOMPUTER POWERED BY 3,000 TESLA PROCESSORS, UNIVERSITY OF ILLINOIS SCIENTISTS PERFORMED THE FIRST ALL-ATOM SIMULATION OF THE HIV VIRUS AND DISCOVERED THE CHEMICAL STRUCTURE OF ITS CAPSID — “THE PERFECT TARGET FOR FIGHTING THE INFECTION.”

WITHOUT GPUS, THE SUPERCOMPUTER WOULD NEED TO BE 5X LARGER FOR SIMILAR PERFORMANCE.

FROM HPC TO ENTERPRISE DATACENTERS

Oil & Gas

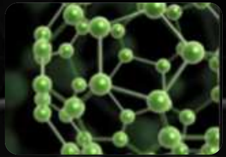
Schlumberger

BR PETROBRAS

Eni

Chevron

Statoil



Higher Ed

HARVARD School of Engineering and Applied Sciences

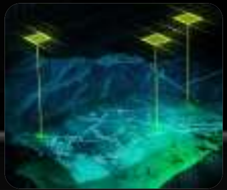
STANFORD UNIVERSITY

Georgia Tech

ETH

Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

UNIVERSITY OF CAMBRIDGE



Government

Air Force Research Laboratory

Raytheon

NASA

Naval Research Laboratory

Supercomputing

CSCS

NCSA

Tokyo Institute of Technology

OAK RIDGE National Laboratory

Lawrence Livermore National Laboratory

Finance

J.P.Morgan

BARCLAYS

STANDARD LIFE

BNP PARIBAS

MUREX

Consumer Web

Baidu 百度

salesforce

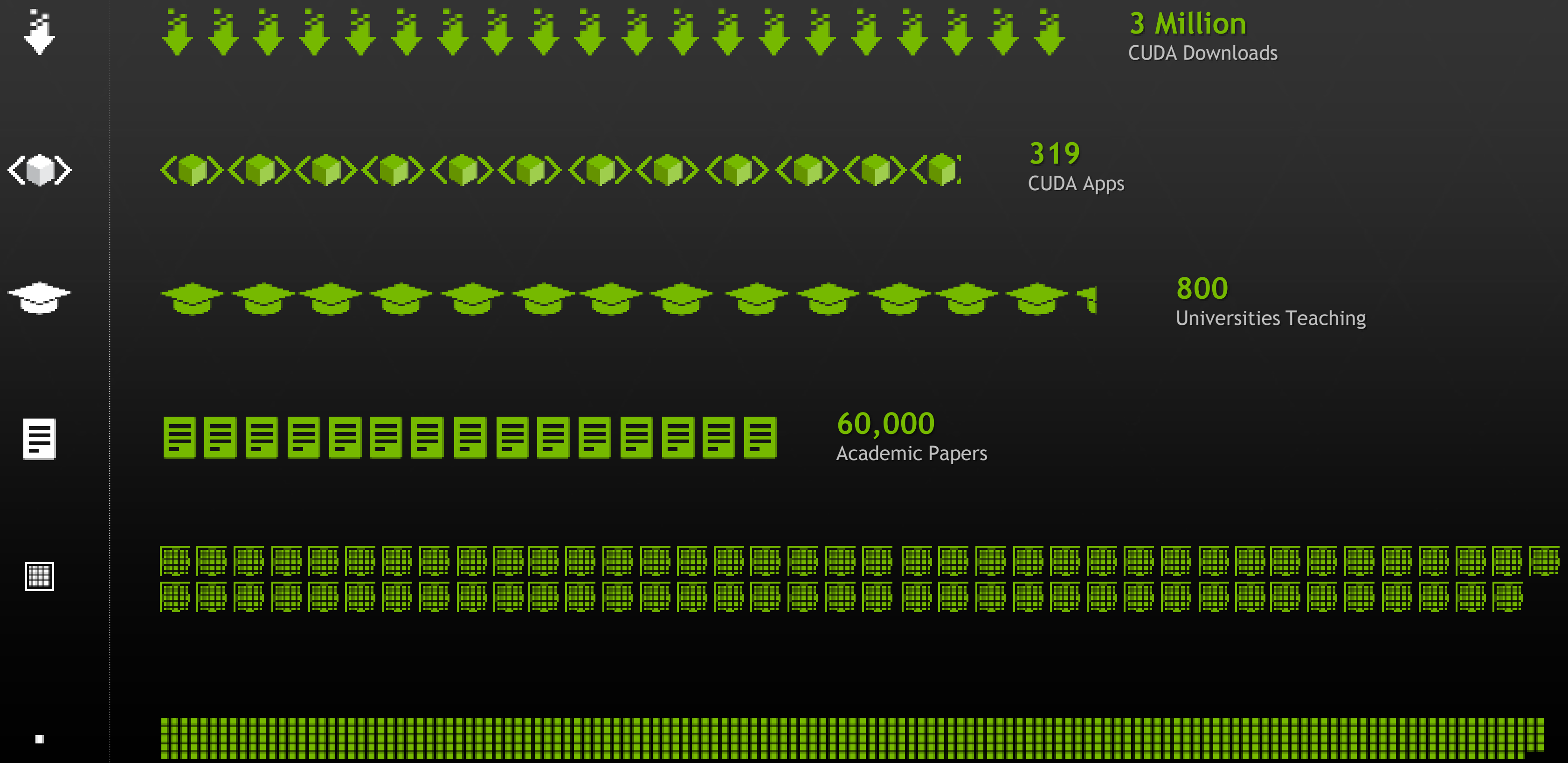
SHAZAM

amazon.com

Yandex

10X GROWTH IN GPU COMPUTING

2008	2015
150,000 CUDA Downloads	3 Million CUDA Downloads
27 CUDA Apps	319 CUDA Apps
60 Universities Teaching	800 Universities Teaching
4,000 Academic Papers	60,000 Academic Papers
6,000 Tesla GPUs	450,000 Tesla GPUs
77 Supercomputing Teraflops	54,000 Supercomputing Teraflops



TESLA ACCELERATED COMPUTING PLATFORM

Data Center Infrastructure

System Solutions

CRAY
DELL
IBM
hp
lenovo

Communication

Mellanox TECHNOLOGIES
MVAPICH
OPEN MPI

Infrastructure Management

Adaptive COMPUTING
Bright Computing
IBM PLATFORM COMPUTING

Development

Programming Languages

C/C++
Fortran
OpenACC
python

Development Tools

allinea DDT
PGI
VAMPIR

Software Solutions

Kitware
MATLAB
ROGUE WAVE SOFTWARE

GPU Accelerators
GPU Boost

Interconnect
GPU Direct
NVLink

System Management
NVML

Compiler Solutions
LLVM

Profile and Debug
CUPTI

Accelerated Libraries
cuBLAS

Enterprise Services Support & Maintenance

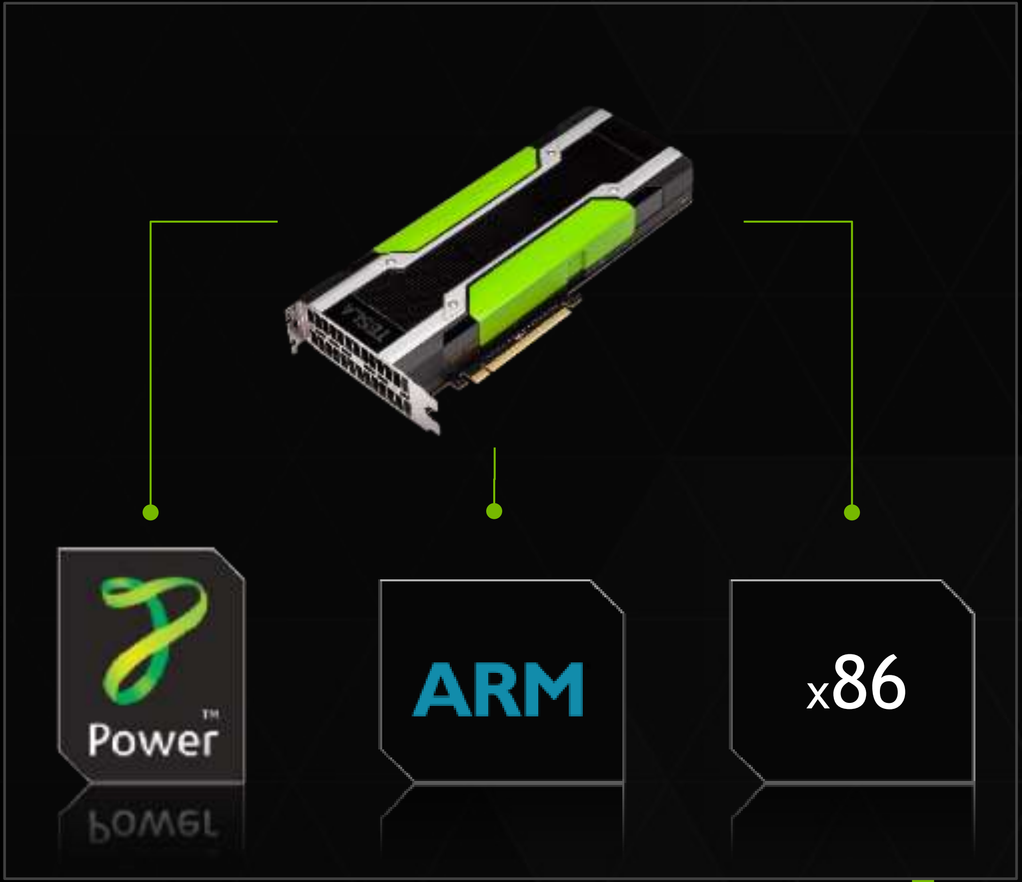
COMMON PROGRAMMING APPROACHES

Across Heterogeneous Architectures

Libraries — AmgX, cuDNN, cuBLAS, OpenCV, Thrust

Compiler Directives — OpenACC

Programming Languages — C/C++, Fortran, python, Java

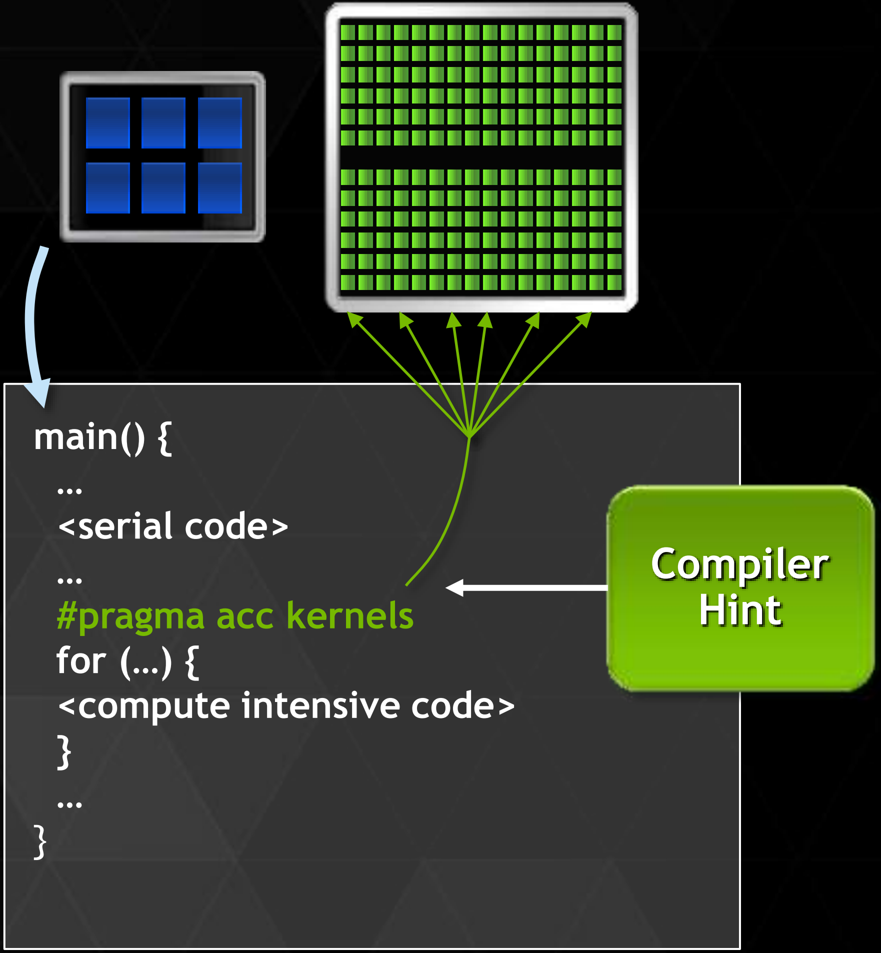


GPU-ACCELERATED LIBRARIES

"Drop-in" Acceleration for Your Applications

Linear Algebra	NVIDIA cuBLAS, cuSPARSE, cuSOLVER	CULA tools	MAGMA	AmgX	
Numerical & Math	IMSL®	NVIDIA Math Lib	NVIDIA cuRAND		
Algorithms and Data Analytics	Thrust	ArrayFire	cuDNN	NVBIO	
Signal, Image, Video, Vision	cuFFT	NVIDIA PERFORMANCE PRIMITIVES	NVIDIA Video Encode	OpenCV	NVIDIA NPP

OPENACC: OPEN, SIMPLE, PORTABLE



```
main() {
  …
  <serial code>
  …
  #pragma acc kernels
  for (…) {
  <compute intensive code>
  }
  …
}
```

- Open Standard
- Easy, Compiler-Driven Approach
- Portable

CAM-SE Climate
6x Faster on GPU
Top Kernel: 50% of Runtime

CUDA C/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
int n = 4096*256;
saxpy_serial(n, 2.0, x, y);
```

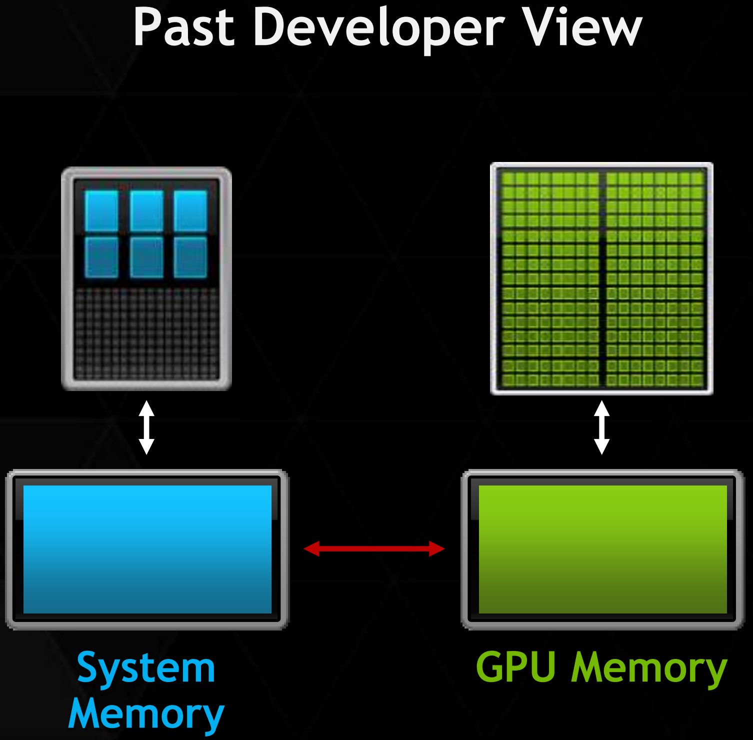
Parallel C Code

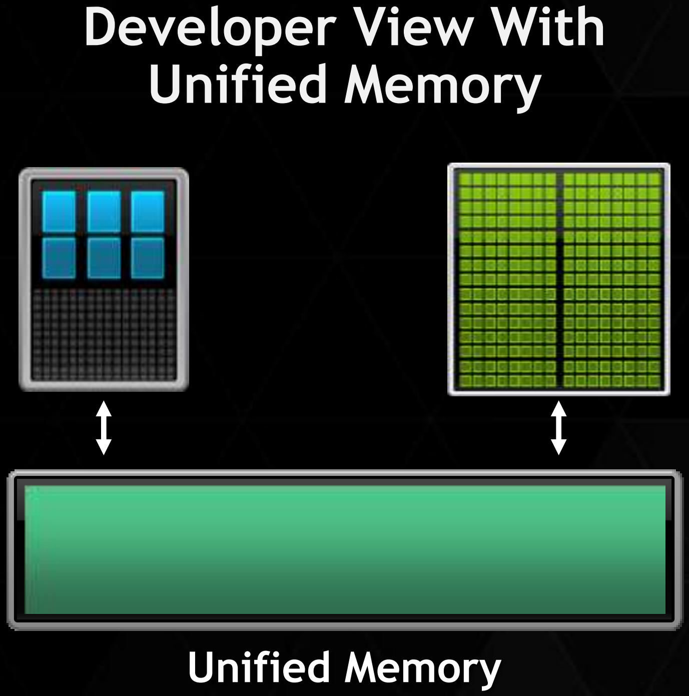
```
__global__
void saxpy_parallel(int n,
                    float a,
                    float *x,
                    float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

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

http://developer.nvidia.com/cuda-toolkit

Unified Memory

Dramatically Lower Developer Effort





SUPER SIMPLIFIED MEMORY MANAGEMENT

CPU Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

CUDA 6 Code with Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

UNIFIED MEMORY DELIVERS

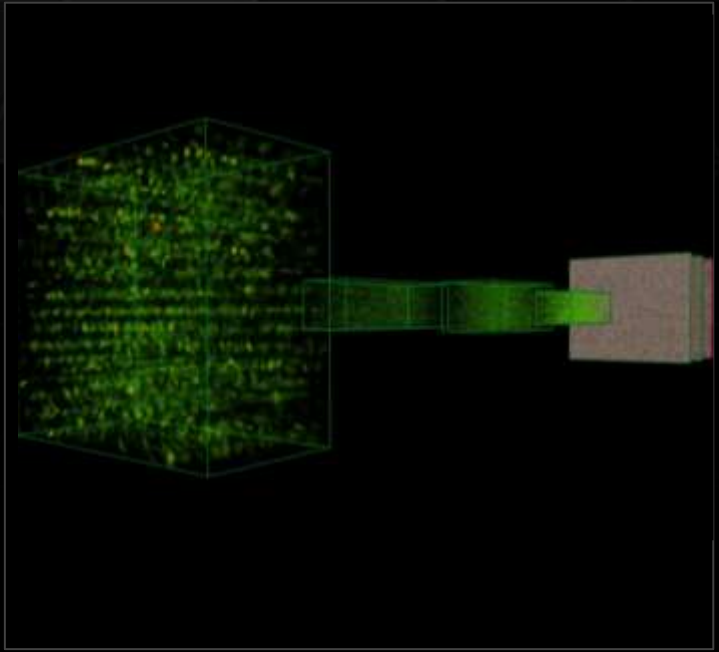
1. Simpler Programming & Memory Model

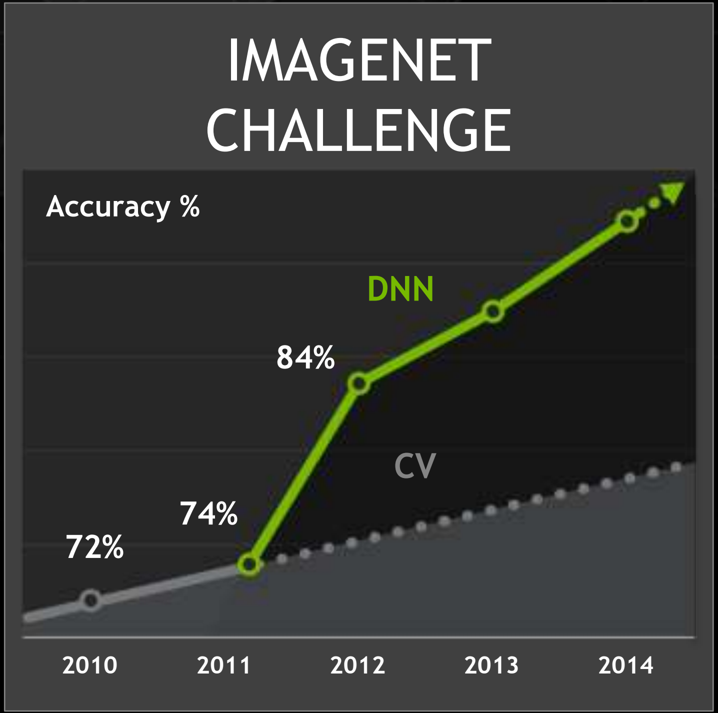
- Single pointer to data, accessible anywhere
- Tight language integration
- Greatly simplifies code porting

2. Performance Through Data Locality

- Migrate data to accessing processor
- Guarantee global coherency
- Still allows *cudaMemcpyAsync()* hand tuning

THE BIG BANG





“Deep Image: Scaling up Image Recognition”

— Baidu: 5.98%, Jan. 13, 2015

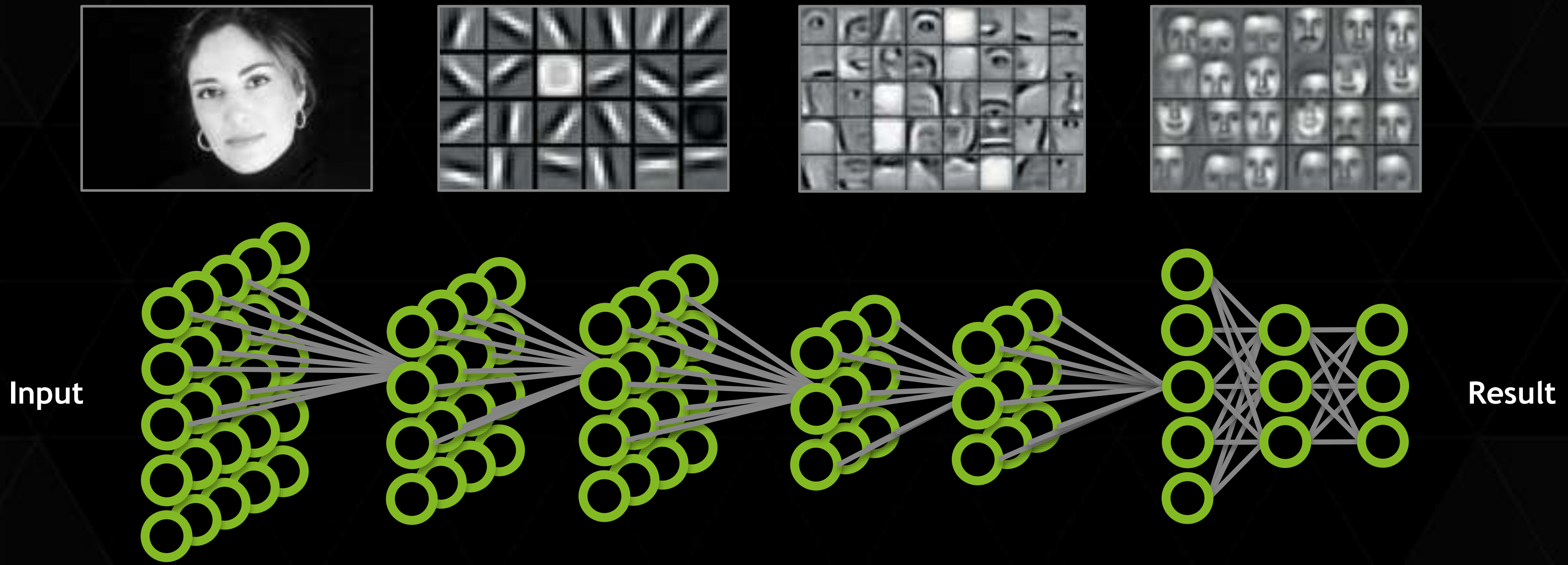
“Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”

— Microsoft: 4.94%, Feb. 6, 2015

“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariant Shift”

— Google: 4.82%, Feb. 11, 2015

MACHINE LEARNING USING DEEP NEURAL NETWORKS



Hinton et al., 2006; Bengio et al., 2007; Bengio & LeCun, 2007; Lee et al., 2008; 2009

Visual Object Recognition Using Deep Convolutional Neural Networks
Rob Fergus (New York University / Facebook) http://on-demand-gtc.gputechconf.com/gtcnew/on-demand-gtc.php#2985

GPU-ACCELERATED DEEP LEARNING

Adobe Alibaba.com

Baidu 百度 facebook

flickr YAHOO! Google

IBM Microsoft

NUANCE twitter

START-UPS

Capio clarifai clarify Dato

EMOTIENT enlitic ersatz labs EyeEm

herta security 科大讯飞 iFLYTEK Intelligent Voice iQIYI 爱奇艺

Letv乐视网 megvii MetaMind NERVANA SYSTEMS

rbeus SENSETIME Sogou搜狗 云知声 Unisound

北京文安 VIONVISION zebra MEDICAL.VISION

Image Detection

Face Recognition

Gesture Recognition

Video Search & Analytics

Speech Recognition & Translation

Image and Video Understanding

Recommendation Engines

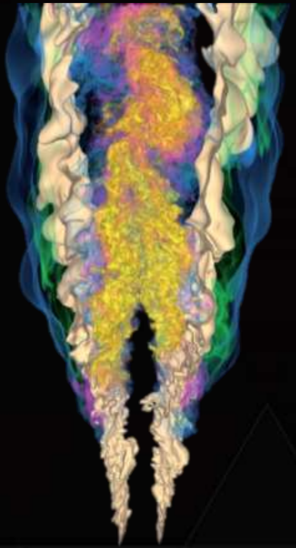
Indexing & Search

CUDA 7

C++11

```
[&](char c) {
  for (auto x : letters)
    if (c == x) return true;
  return false;
}
```

cuSOLVER



Runtime Compilation

GPU TECHNOLOGY CONFERENCE

"C++11 FEELS LIKE A NEW LANGUAGE"

- Bjarne Stroustrup, creator of C++
  - "Pieces fit together better… higher-level style of programming"

- Auto, Lambda, range-based for, initializer lists, variadic templates, more…

- Enable using --std=c++11 (not required for MSVC)

```
nvcc --std=c++11 myprogram.cu –o myprogram
```

Examples in this talk:
nvda.ly/Kty6M

Useful C++11 overviews:
http://www.stroustrup.com/C++11FAQ.html
http://herbsutter.com/elements-of-modern-c-style/

A SMALL C++11 EXAMPLE

- Count the number of occurrences of letters x, y, z and w in text

```
__global__
void xyzw_frequency(int *count, char *text, int n)
{
    const char letters[] { 'x','y','z','w' };

    count_if(count, text, n, [&](char c) {
        for (const auto x : letters)
            if (c == x) return true;
        return false;
    });
}
```

Initializer List

Lambda Function

Range-based For Loop

Automatic type deduction

Output:

```
Read 3288846 bytes from "warandpeace.txt"
counted 107310 instances of 'x', 'y', 'z', or 'w' in "warandpeace.txt"
```

LAMBDA

- count_if() increments count for each element of data for which p is true:

```
template <typename T, typename Predicate>
__device__ void count_if(int *count, T *data, int n, Predicate p);
```

- Predicate is a function object. In C++11, this can be a Lambda:

```
const char letters[] {
   'x','y','z','w' };
```

```
[&](char c) {
    for (const auto x : letters)
        if (c == x) return true;
    return false;
}
```

Lambda: *Closure*

Unnamed function object capable of capturing variables in scope.

AUTO AND RANGE-FOR

- Auto tells the compiler to deduce variable type from initializer

```
for (const auto x : letters) {
    if (x == c) return true;
}
```

- Range-based For Loop is equivalent to:

```
for (auto x = std::begin(letters); x != std::end(letters); x++) {
    if (x == c) return true;
}
```

  - Use with arrays of known size, or any object that defines `begin()` / `end()`

GPU TECHNOLOGY CONFERENCE

CUDA GRID-STRIDE LOOPS

- Common idiom in CUDA C++

```
template <typename T, typename Predicate>
__device__ void count_if(int *count, T *data, int n, Predicate p)
{
    for (int i = blockDim.x * blockIdx.x + threadIdx.x;
         i < n;
         i += gridDim.x * blockDim.x)
    {
        if (p(data[i])) atomicAdd(count, 1);
    }
}
```

Verbose, bug-prone…

- Decouple grid & problem size, decouple host & device code

http://devblogs.nvidia.com/parallelforall/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/

GPU TECHNOLOGY CONFERENCE

CUDA GRID-STRIDE RANGE-FOR

- Simpler and clearer to use C++11 range-based for loop:

```
template <typename T, typename Predicate>
__device__ void count_if(int *count, T *data, int n, Predicate p)
{
    for (auto i : grid_stride_range(0, n)) {
        if (p(data[i])) atomicAdd(count, 1);
    }
}
```

- C++ allows range-for on any object that implements begin() and end()
- We just need to implement `grid_stride_range()`...

GRID-STRIDE RANGE HELPER

- Just need a strided range class. One I like: http://github.com/klmr/cpp11-range/
  - Forked and updated to work in __device__ code: http://github.com/harrism/cpp11-range

```
#include "range.hpp”

template <typename T>
__device__
step_range<T> grid_stride_range(T begin, T end) {
    begin += blockDim.x * blockIdx.x + threadIdx.x;
    return range(begin, end).step(gridDim.x * blockDim.x);
}
```

- Enables simple, bug-resistant grid-stride loops in CUDA C++

```
for (auto i : grid_stride_range(0, n)) { ... }
```

THRUST: RAPID PARALLEL C++ DEVELOPMENT

- Resembles C++ STL
- Open source
- Productive High-level API
  - CPU/GPU Performance portability
- Flexible
  - CUDA, OpenMP, and TBB backends
  - Extensible and customizable
  - Integrates with existing software
- Included in CUDA Toolkit
  - CUDA 7 includes new Thrust 1.8

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

http://thrust.github.io

GPU TECHNOLOGY CONFERENCE

C++11 AND THRUST: AUTO

- Naming complex Thrust iterator types can be troublesome:

```
typedef typename device_vector<float>::iterator FloatIterator;
typedef typename tuple<FloatIterator,
                       FloatIterator,
                       FloatIterator> FloatIteratorTuple;
typedef typename zip_iterator<FloatIteratorTuple> Float3Iterator;

Float3Iterator first =
    make_zip_iterator(make_tuple(A0.begin(), A1.begin(), A2.begin()));
```

- C++11 auto makes it easy! Variable types automatically deduced:

```
auto first =
    make_zip_iterator(make_tuple(A0.begin(), A1.begin(),
A2.begin()));
```

GPU TECHNOLOGY CONFERENCE

C++11 AND THRUST: LAMBDA

- C++11 lambda makes a powerful combination with Thrust algorithms.

```
void xyzw_frequency_thrust_host(int *count, char *text, int n)
{
  const char letters[] { 'x','y','z','w' };

  *count = thrust::count_if(thrust::host, text, text+n, [&](char c) {
    for (const auto x : letters)
      if (c == x) return true;
    return false;
  });
}
```

- Here we apply thrust::count_if on the host, using a lambda predicate

NEW: DEVICE-SIDE THRUST

- Call Thrust algorithms from CUDA device code

```
__global__
void xyzw_frequency_thrust_device(int *count, char *text, int n)
{
  const char letters[] { 'x','y','z','w' };

  *count = thrust::count_if(thrust::device, text, text+n, [=](char c) {
    for (const auto x : letters)
      if (c == x) return true;
    return false;
  });
}
```

Device Execution

Device Lambda

- Device execution uses Dynamic Parallelism kernel launch on supporting devices
- Can also use `thrust::cuda::par` execution policy

GPU TECHNOLOGY CONFERENCE

NEW: DEVICE-SIDE THRUST

- Call Thrust algorithms from CUDA device code

```
__global__
void xyzw_frequency_thrust_device(int *count, char *text, int n)
{
  const char letters[] { 'x','y','z','w' };

  *count = thrust::count_if(thrust::seq, text, text+n, [=](char c) {
    for (const auto x : letters)
      if (c == x) return true;
    return false;
  });
}
```

Sequential Execution Within each CUDA thread

GPU TECHNOLOGY CONFERENCE

MORE THRUST IMPROVEMENTS IN CUDA 7

- Faster algorithms
  - thrust::sort: 300% faster for user-defined types, 50% faster for primitive types
  - thrust::merge: 200% faster
  - thrust::reduce_by_key: 25% faster
  - thrust::scan: 15% faster

- API Support for CUDA streams argument (concurrency between threads)

```
thrust::count_if(thrust::cuda::par.on(stream1), text, text+n, myFunc());
```

NEW LIBRARY: CUSOLVER

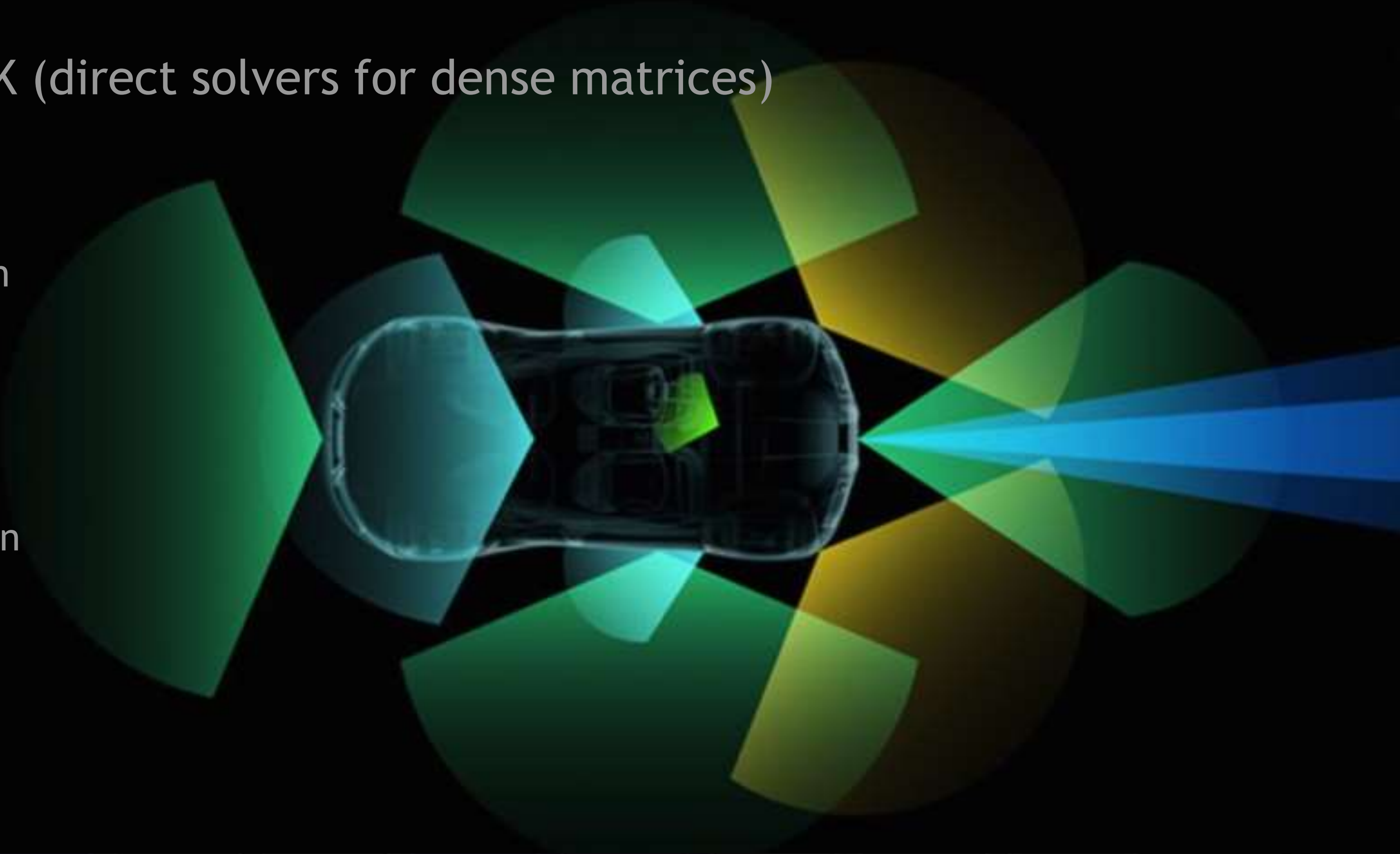
- Routines for solving sparse and dense linear systems and Eigen problems

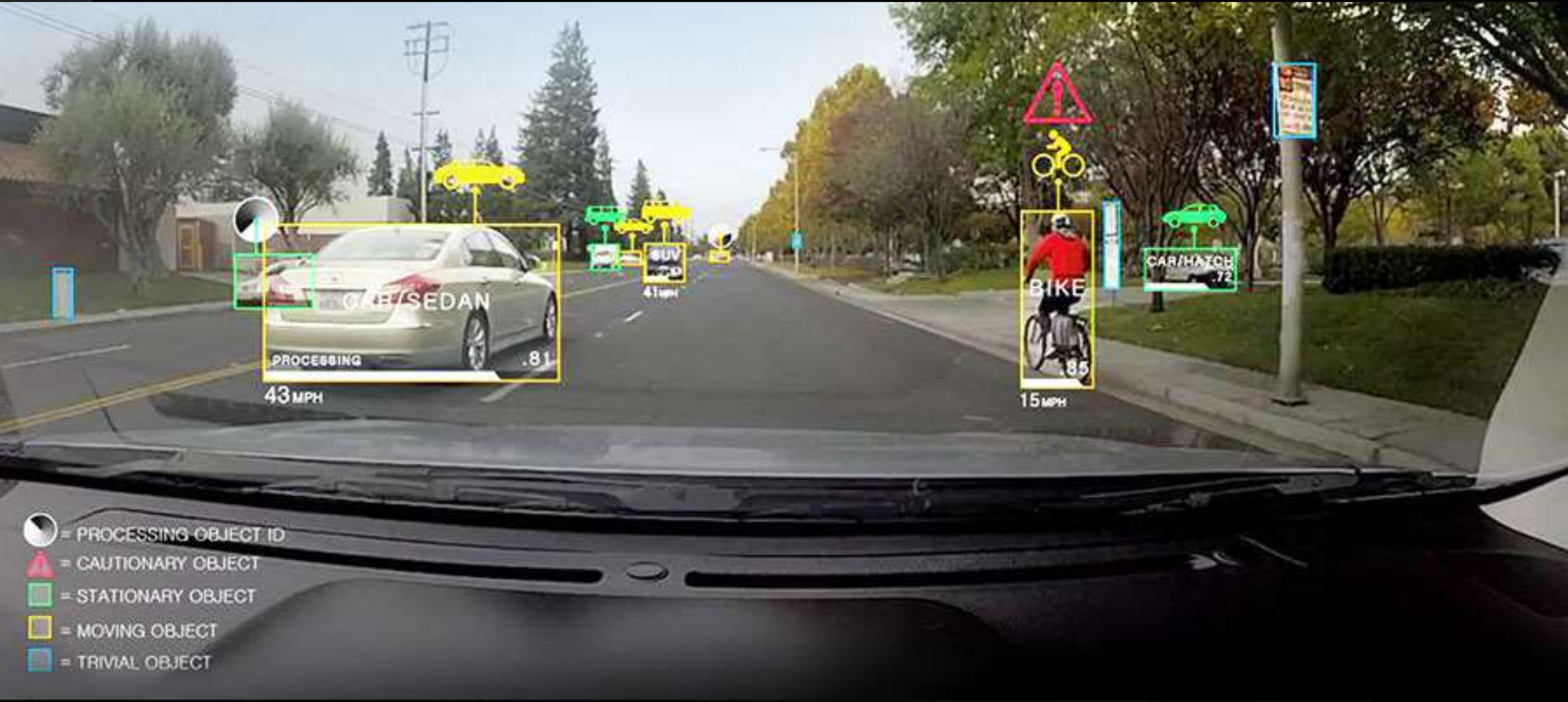
- 3 APIs:
  - Dense,
  - Sparse
  - Refactorization

GPU TECHNOLOGY CONFERENCE

cuSOLVER DENSE

- Subset of LAPACK (direct solvers for dense matrices)
  - Cholesky / LU
  - QR, SVD
  - Bunch-Kaufman
  - Batched QR
- Useful for:
  - Computer vision
  - Optimization
  - CFD

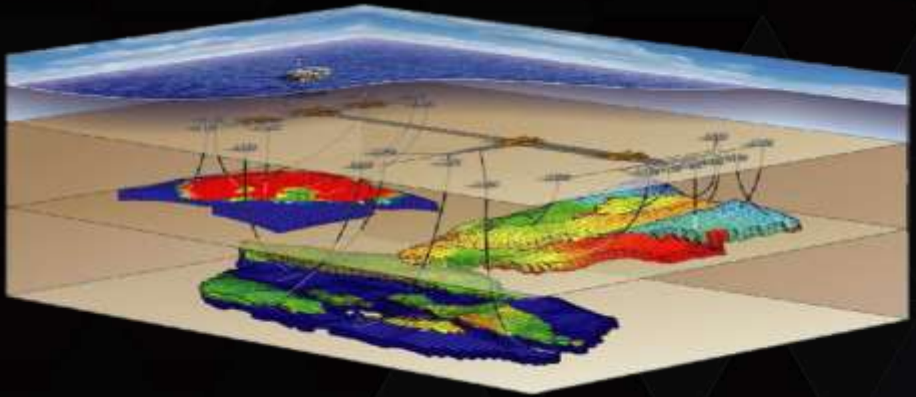



CAR/SEDAN
PROCESSING
.81
43 MPH
SUV
41 MPH
BIKE
.85
15 MPH
CAR/HATCH
.72
= PROCESSING OBJECT ID
= CAUTIONARY OBJECT
= STATIONARY OBJECT
= MOVING OBJECT
= TRIVIAL OBJECT

cuSOLVER SPARSE API

- Sparse direct solvers based on QR factorization
  - Linear solver A*x = b (QR or Cholesky-based)
  - Least-squares solver min|A*x – b|
  - Eigenvalue solver based on shift-inverse
  - A*x = \lambda*x
  - Find number of Eigenvalues in a box

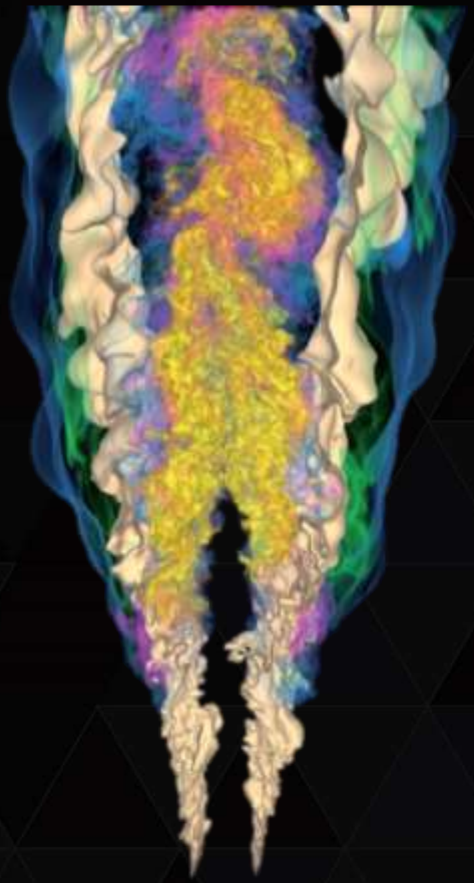
- Useful for:
  - Well models in Oil & Gas
  - Non-linear solvers via Newton’s method
  - Anywhere a sparse-direct solver is required



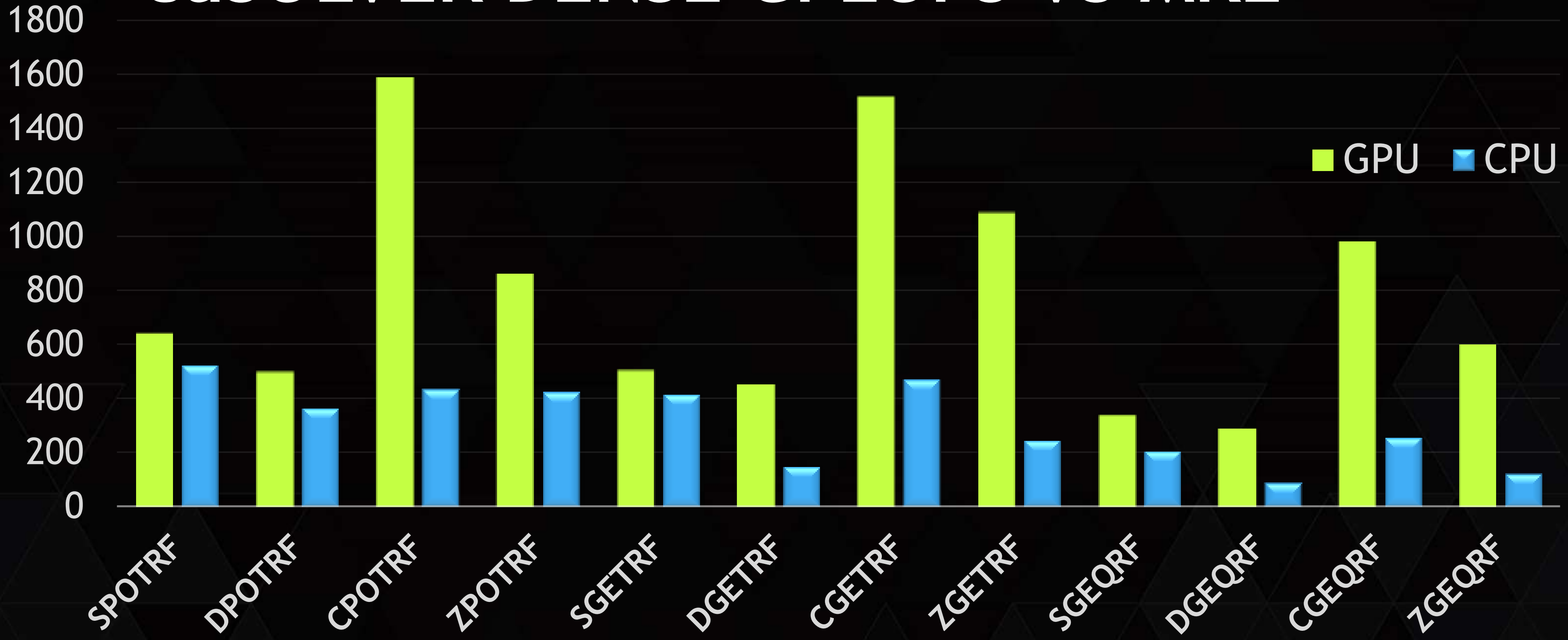
cuSOLVER REFACTORIZATION API

- LU-based sparse direct solver
  - Requires factorization to already be computed (e.g. using KLU)
- Batched version
  - Many small matrices to be solved in parallel

- Useful for:
  - SPICE
  - Combustion simulation
  - Chemically reacting flow calculation
  - Other types of ODEs, mechanics

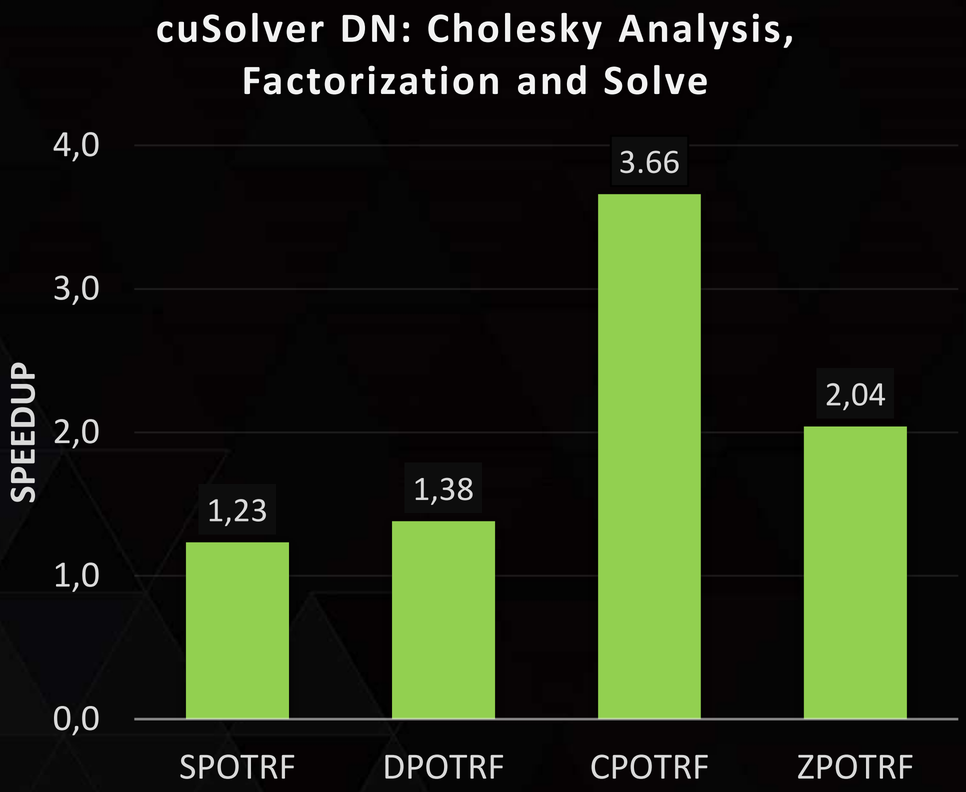


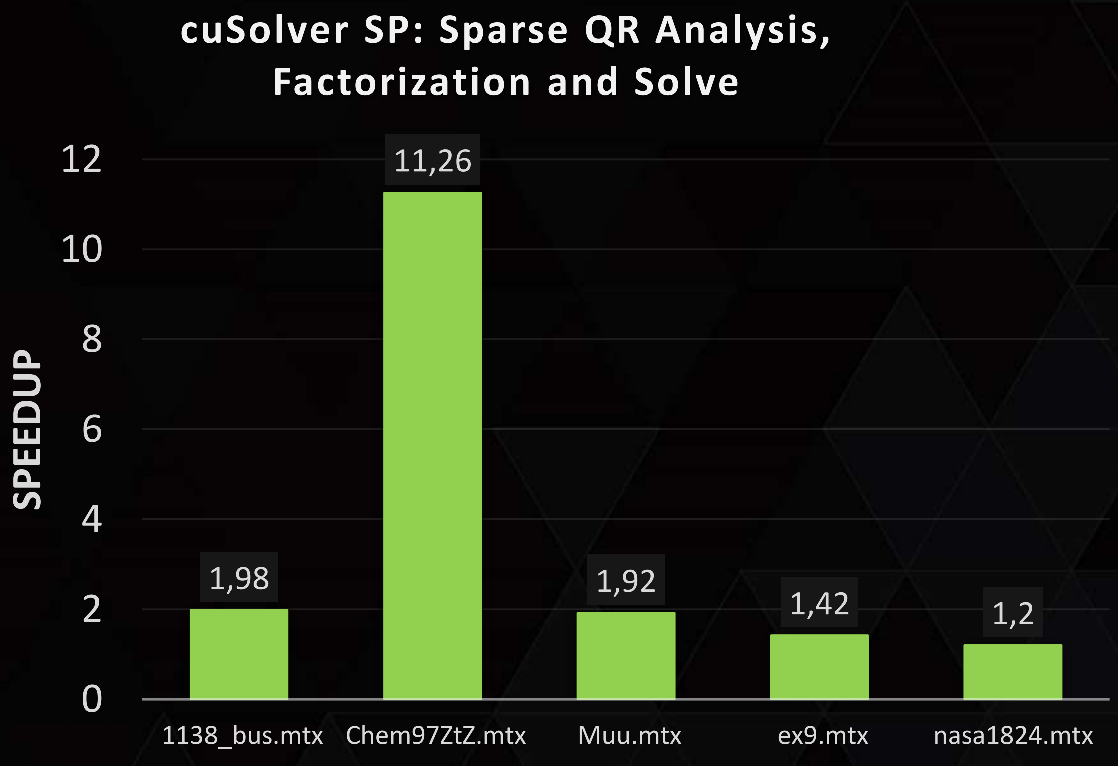
cuSOLVER DENSE GFLOPS VS MKL



GPU:K40c M=N=4096
CPU: Intel(R) Xeon(TM) E5-2697 v3 CPU @ 3.60GHz, 14 cores
MKL v11.04

cuSOLVER SPEEDUP

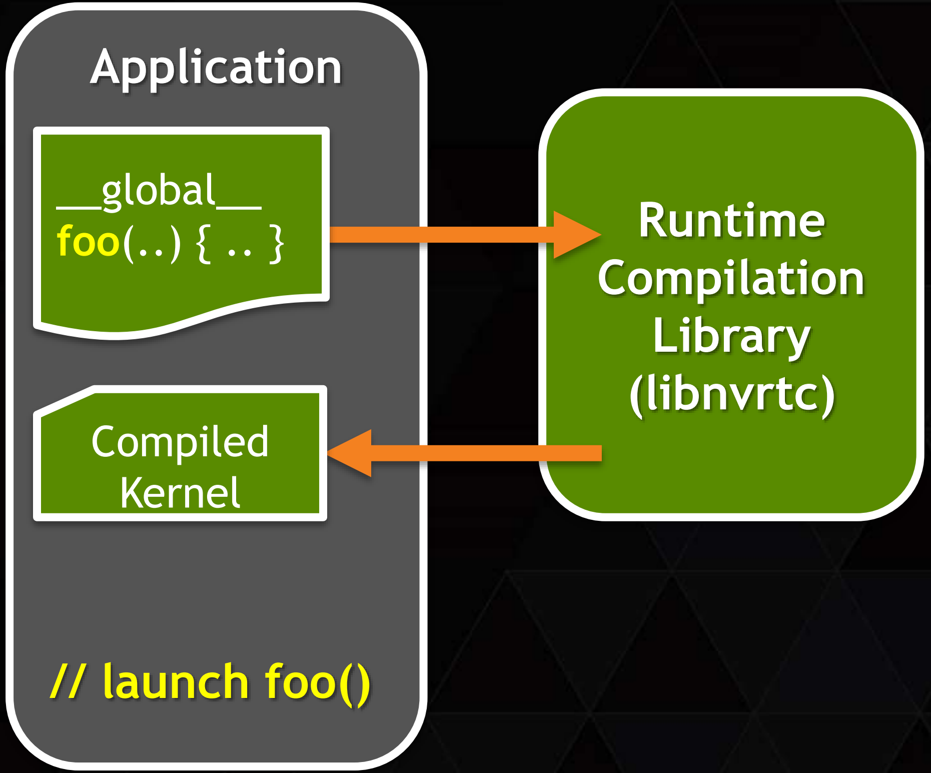




GPU:K40c M=N=4096
CPU: Intel(R) Xeon(TM) E5-2697v3 CPU @ 3.60GHz, 14 cores
MKL v11.04 for Dense Cholesky, Nvidia csr-QR implementation for CPU and GPU

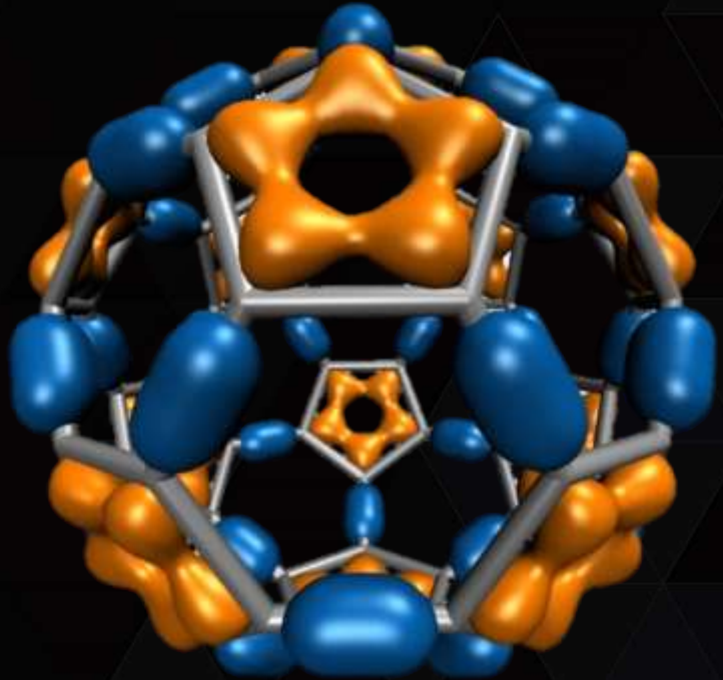
CUDA RUNTIME COMPILATION

- Compile CUDA kernel source at run time
  - Compiled kernels can be cached on disk
- Runtime C++ Code Specialization
  - Optimize code based on run-time data
  - Unroll loops, eliminate references, fold constants
  - Reduce compile time and compiled code size
- Enables runtime code generation, C++ template-based DSLs

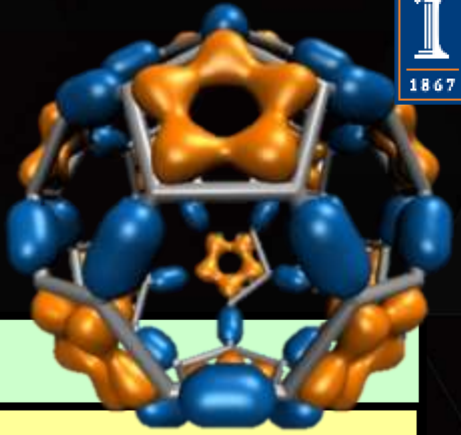


HIGHER PERF FOR DATA-DRIVEN ALGORITHMS

- Example: Visualization of Molecular Orbitals
  - Expensive to compute and cache
- GPUs enable interactivity and animation
  - Provides insight into simulation results
- Generate input-specific kernels at runtime for 1.8 speedup
- Courtesy John Stone, Beckman Institute, UIUC

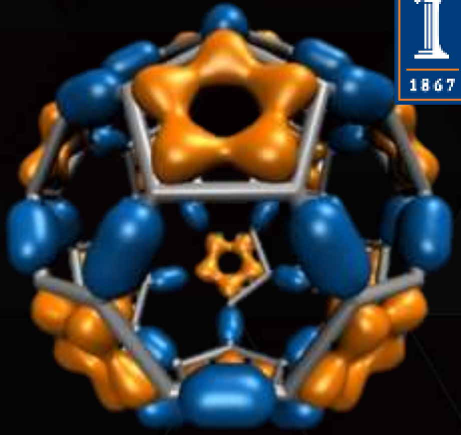

C60: “buckyball”

MOLECULAR ORBITAL KERNEL



```
Loop over atoms (1 to ~200) {

    Loop over electron shells for this atom type (1 to ~6) {

        Loop over primitive functions for shell type (i: 1 to ~6) {
            Data-driven, short loop trip count → high overhead

            Dynamic kernel generation and run-time compilation
            Unroll entirely, resulting in 1.8x speed boost!
        }

        Loop over angular momenta for this shell type (1 to ~15) {}

    }

}
```

MOLECULAR ORBITAL KERNEL

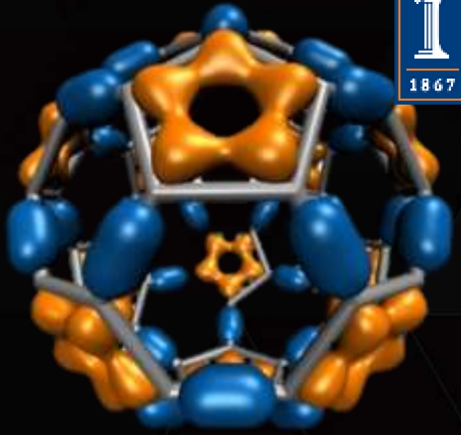


- Original inner loop
- Short trip count → high loop overhead

```
Loop over primitive functions for shell type (i: 1 to ~6) {
  float exponent = const_basis_array[prim_counter];
  float contract_coeff = const_basis_array[prim_counter + 1];
  contracted_gto += contract_coeff * expf(-exponent*dist2);
  prim_counter += 2;
}
```

- But #primitive functions known at initialization time

MOLECULAR ORBITAL KERNEL



- Fully unrolled inner loop
- Eliminate array lookups for exponents & coefficients

```
contracted_gto =  1.832937 * expf(-7.868272*dist2);
contracted_gto += 1.405380 * expf(-1.881289*dist2);
contracted_gto += 0.701383 * expf(-0.544249*dist2);
```

- 1.8x overall speedup!

GPU TECHNOLOGY CONFERENCE

BEYOND CUDA 7

PARALLEL PROGRAMMING APPROACHES

- Prescriptive Parallelism
  - Program specifies details of parallel execution configuration
  - More programmer control
  - Greater programmer responsibility

```
xyzw_frequency<<<blockSize, nBlocks>>>
           (count, text, len);
```

- Descriptive Parallelism
  - Program indicates parallel regions
  - Compiler / runtime determine execution configuration
  - More performance portable
  - Greater compiler responsibility

```
thrust::count_if(thrust::device, d, d+n,
  [&](char c){…});
```

DESCRIPTIVE KERNEL LAUNCHES

- Enable launching CUDA kernels without prescribing parallelism
  - This: `launch(xyzw_frequency, count, text, len);`
  - Instead of this:

```
xyzw_frequency<<<blockSize, nBlocks>>>(count, text, len);
```

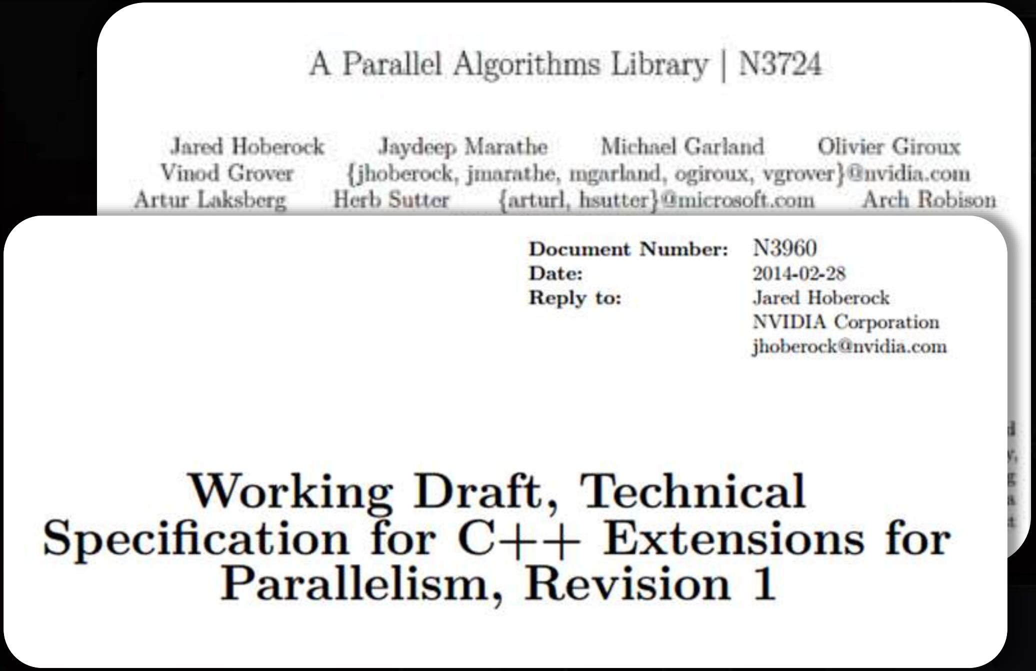
- The library / runtime chooses execution configuration
  - Based on device and kernel attributes
  - Easier, more portable

- Prototype in hemi open-source library
  - http://github.com/harrism/hemi (in “apk” branch)

PARALLEL STL

```
std::vector<int> vec = ...

// previous standard sequential loop
std::for_each(vec.begin(), vec.end(), f);

// explicitly sequential loop
std::for_each(std::seq, vec.begin(), vec.end(),
f);

// permitting parallel execution
std::for_each(std::par, vec.begin(), vec.end(),
f);
```

- Complete set of parallel primitives: for_each, sort, reduce, scan, etc.
- ISO C++ committee voted unanimously to accept as official tech. specification working draft



A Parallel Algorithms Library | N3724

Jared Hoberock Jaydeep Marathe Michael Garland Olivier Giroux
Vinod Grover {jhoberock, jmarathe, mgarland, ogiroux, vgrover}@nvidia.com
Artur Laksberg Herb Sutter {arturl, hsutter}@microsoft.com Arch Robison

Document Number:	N3960
Date:	2014-02-28
Reply to:	Jared Hoberock NVIDIA Corporation jhoberock@nvidia.com

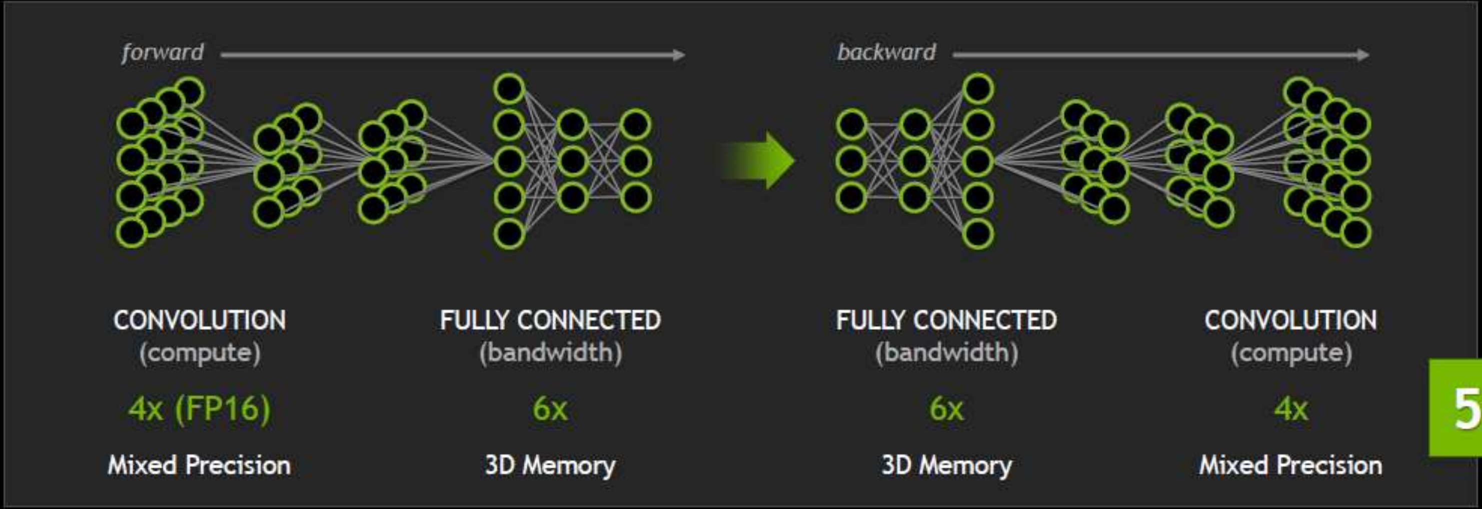
Working Draft, Technical Specification for C++ Extensions for Parallelism, Revision 1

N3960 Technical Specification Working Draft:
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4352.html
Prototype:
https://github.com/n3554/n3554

GPU TECHNOLOGY CONFERENCE

MIXED PRECISION COMPUTATION

PASCAL 10X MAXWELL



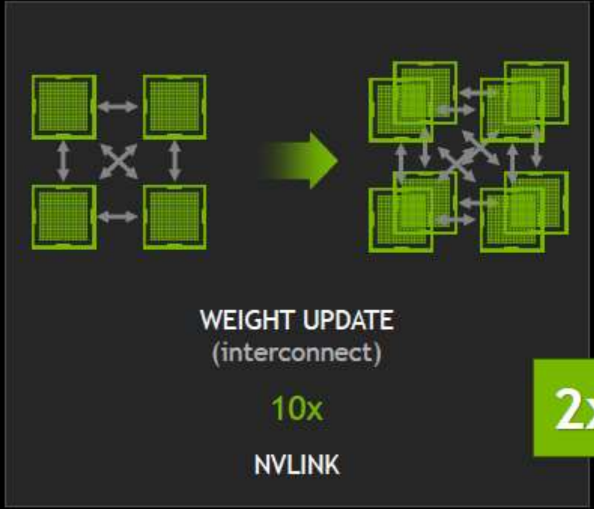
CONVOLUTION (compute) 4x (FP16) Mixed Precision

FULLY CONNECTED (bandwidth) 6x 3D Memory

FULLY CONNECTED (bandwidth) 6x 3D Memory

CONVOLUTION (compute) 4x Mixed Precision

5x



WEIGHT UPDATE (interconnect) 10x NVLINK

2x

* *Very rough estimates*

MIXED PRECISION COMPUTATION

- *half precision* (fp16) data type in addition to single (fp32), double (fp64)

- fp16: half the bandwidth, twice the throughput
- Format: s1e5m10
- Range ~ -6*10^-8 … 6*10^4 as it includes denormals

- Limitations
  - Limited precision: 11-bit mantissa
  - Vector operations only: 32-bit register holds 2 fp16 values

FP16 SUPPORT IN CUDA

Developer API

Half & half2 datatypes

Vector ops
- Convert 16<->32
- Compare
- FMA ops

Arithmetic

cuBLAS: HGEMM

cuDNN: forward convolution

cuFFT: smaller input sizes

Storage/data exchange:

E.g. SGEMM_EX (Math in FP32)

cuDNN Forward/Backward training path

cuFFT

Your Needs

?

GPU TECHNOLOGY CONFERENCE

Examples in this talk:
nvda.ly/Kty6M

THANK YOU
harrism@nvidia.com

JOIN THE CONVERSATION
#GTC15

@harrism

JOIN NVIDIA REGISTERED DEVELOPERS

- Members get access to the latest software releases and tools
  - Notifications about developer events and webinars
  - Report bugs and request for feature enhancements
  - Exclusive activities and special offers

Join for free:
developer.nvidia.com