CUDA TOOLKIT 8
Everything you need to accelerate applications

Comprehensive C/C++ development environment

Out of box performance on Pascal

Unified Memory on Pascal enables simple programming with large datasets

New critical path analysis quickly identifies system-level bottlenecks

developer.nvidia.com/cuda-toolkit
CUDA 8 — WHAT’S NEW

**PASCAL SUPPORT**
- New Architecture
- NVLINK
- HBM2 Stacked Memory
- Page Migration Engine

**UNIFIED MEMORY**
- Larger Datasets
- Demand Paging
- New Tuning APIs
- Data Coherence & Atomics

**LIBRARIES**
- New nvGRAPH library
- cuBLAS improvements for Deep Learning

**DEVELOPER TOOLS**
- Critical Path Analysis
- 2x Faster Compile Time
- OpenACC Profiling
- Debug CUDA Apps on Display GPU
MASSIVE LEAP IN PERFORMANCE

With CUDA 8 on Pascal

![Graph showing speedup vs Dual Socket Broadwell for various applications with different hardware configurations.](image-url)

- **NAMD**
- **VASP**
- **MILC**
- **HOOMD Blue**
- **AMBER**
- **Caffe/Alexnet**

**CPU:** Xeon E5-2697v4, 2.3 GHz, 3.6 GHz Turbo

**Caffe Alexnet:** batch size of 256 images; VASP, NAMD, HOOMD-Blue, and AMBER average speedup across a basket of tests
UNIFIED MEMORY
UNIFIED MEMORY
Dramatically lower developer effort

Past Developer View

Starting with Kepler and CUDA 6

System Memory ↔ GPU Memory ↔ Unified Memory
CUDA 8: PASCAL UNIFIED MEMORY

Larger datasets, simpler programming, higher performance

CUDA 8

Pascal GPU

CPU

Unified Memory

Allocate Beyond GPU Memory Size

ENABLE LARGE DATA MODELS

Oversubscribe GPU memory
Allocate up to system memory size

SIMPLER DATA ACCESS

CPU/GPU Data coherence
Unified memory atomic operations

TUNE UNIFIED MEMORY PERFORMANCE

New APIs for Pre-fetching & Read duplication
Usage hints via cudaMemAdvise API

Feature Available in GA
CUDA 8 UNIFIED MEMORY — EXAMPLE

Allocating 4x more than P100 physical memory

```c
void foo() {
    // Allocate 64 GB
    char *data;
    size_t size = 64*1024*1024*1024;
    cudaMallocManaged(&data, size);
}
```

64 GB unified memory allocation on P100 with 16 GB physical memory

Transparent - No API changes

Works on Pascal & future architectures
CUDA 8 UNIFIED MEMORY — EXAMPLE
Accessing data simultaneously by CPU and GPU codes

```c
__global__ void mykernel(char *data) {
    data[1] = 'g';
}

void foo() {
    char *data;
    cudaMallocManaged(&data, 2);
    mykernel<<<...>>>(data);
    // no synchronize here
    data[0] = 'c';
    cudaFree(data);
}
```

Both CPU code and CUDA kernel accessing ‘data’ simultaneously
Possible with CUDA 8 unified memory on Pascal
ON-DEMAND PAGING
New use cases such as graph algorithms

Higher Performance with Unified Memory on Maximum Flow

- Unified Memory Baseline (no explicit tuning)
- Unified Memory with Migration Hints (optimized memory placement)

Performance over CPU only

Application working set / GPU memory size

Large Data Set
GPU MEMORY OVERSUBSCRIPTION
Many domains would benefit

Combustion
Many species & improved accuracy

Quantum chemistry
Larger systems

Ray-tracing
Larger scenes to render

Higher Throughput with Large Grid Sizes on AMR Codes

GRID Size

<table>
<thead>
<tr>
<th>GRID Size</th>
<th>Tesla K40 (12GB)</th>
<th>Tesla P100 (16GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0.016 GB</td>
<td>0.016 GB</td>
</tr>
<tr>
<td>64</td>
<td>0.062 GB</td>
<td>0.062 GB</td>
</tr>
<tr>
<td>128</td>
<td>0.33 GB</td>
<td>0.33 GB</td>
</tr>
<tr>
<td>192</td>
<td>1.43 GB</td>
<td>1.43 GB</td>
</tr>
<tr>
<td>256</td>
<td>2.58 GB</td>
<td>2.58 GB</td>
</tr>
<tr>
<td>384</td>
<td>8.69 GB</td>
<td>8.69 GB</td>
</tr>
<tr>
<td>512</td>
<td>20.59 GB</td>
<td>20.59 GB</td>
</tr>
<tr>
<td>768</td>
<td>69.5 GB</td>
<td>69.5 GB</td>
</tr>
</tbody>
</table>

Simulate More Species w/ Accurate results
LIBRARIES
GRAPH ANALYTICS

Insight from connections in big data

SOCIAL NETWORK ANALYSIS

CYBER SECURITY / NETWORK ANALYTICS

GENOMICS

... and much more: Parallel Computing, Recommender Systems, Fraud Detection, Voice Recognition, Text Understanding, Search
INTRODUCING NVGRAPH

Accelerate graph analytics applications

Deliver results up to 3x faster than CPU-only
Solve graphs with up to 2.5 Billion edges on 1x M40
Accelerates a wide range of graph analytics apps

<table>
<thead>
<tr>
<th>PAGERANK</th>
<th>SINGLE SOURCE SHORTEST PATH</th>
<th>SINGLE SOURCE WIDEST PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Robotic path planning</td>
<td>IP routing</td>
</tr>
<tr>
<td>Recommendation engines</td>
<td>Power network planning</td>
<td>Chip design / EDA</td>
</tr>
<tr>
<td>Social Ad placement</td>
<td>Logistics &amp; supply chain planning</td>
<td>Traffic sensitive routing</td>
</tr>
</tbody>
</table>

developer.nvidia.com/nvgraph
ADDITIONAL LIBRARY IMPROVEMENTS
cuBLAS, nvBLAS, cuBLAS-XT, and cuSPARSE

cuBLAS-XT and nvBLAS will now accept argument from device memory, so users can keep data in device memory and chain together accelerated operations.

Improved new Strided Batched GEMM are easier to use and yield higher performance on newer GPUs.

Improved SGEMM performance for small arrays where m and n are not a multiple of the tile size.

8-bit integer input and output for certain cuBLAS routines.
DEVELOPER TOOLS
DEPENDENCY ANALYSIS
Easily find the critical kernel to optimize

The longest running kernel is not always the most critical optimization target
DEPENDENCY ANALYSIS

Visual Profiler

Unguided Analysis

Generating critical path

Dependency Analysis

Functions on critical path
DEPENDENCY ANALYSIS

Visual profiler

Launch copy_kernel
MemCpy HtoD [sync]

Inbound dependencies

MemCpy DtoH [sync]

Outbound dependencies
OPENACC PROFILING

OpenAcc -> Driver API -> Compute correlation

OpenAcc timeline

OpenAcc -> Source Code correlation

OpenAcc Properties
PROFILE CPU CODE IN VISUAL PROFILER

New CPU profiling with CUDA 8

Profile execution times on host function calls

View CPU code function hierarchy
MORE CUDA 8 PROFILER FEATURES

Unified memory profiling

NVLink topology and bandwidth profiling
COMPILER
2X FASTER COMPILE TIMES ON CUDA

NVCC Speedups on CUDA 8

- Average total compile times (per translation unit)
- Intel Core i7-3930K (6-cores) @ 3.2GHz
- CentOS x86_64 Linux release 7.1.1503 (Core) with GCC 4.8.3 20140911
- GPU target architecture sm_52

Performance may vary based on OS and software versions, and motherboard configuration
__global__ template <typename F, typename T>
void apply(F function, T *ptr) {
  *ptr = function(ptr);
}

int main(void) {
  float *x;
  cudaMallocManaged(&x, 2);

  auto square = [=] __host__ __device__ (float x) { return x*x; };

  apply<<<1, 1>>>(square, &x[0]);
  ptr[1] = square(&x[1]);
  cudaFree(x);
}

Call lambda from device code

__host__ __device__ lambda

Pass lambda to CUDA kernel

... or call it from host code

Experimental feature in CUDA 8.
`nvcc --expt-extended-lambda`
HETEROGENEOUS C++ LAMBDA

Usage with thrust

```c++
void saxpy(float *x, float *y, float a, int N) {
    using namespace thrust;
    auto r = counting_iterator(0);

    auto lambda = [=] __host__ __device__ (int i) {
        y[i] = a * x[i] + y[i];
    };

    if(N > gpuThreshold)
        for_each(device, r, r+N, lambda);
    else
        for_each(host, r, r+N, lambda);
}
```

__host__ __device__ lambda

Use lambda in thrust::for_each on host or device

Experimental feature in CUDA 8.
`nvcc --expt-extended-lambda`
OTHER FEATURES
GPUDIRECT ASYNC*
Achieve higher MPI application performance across large clusters

Eliminate CPU as the critical path in the GPU initiated data transfers

GPU directly triggers data transfers without CPU coordination

CPU is unblocked to perform other tasks

Between two nodes: each node is Hybridge Xeon CPU + 1 Mellanox Connectx-6 FDR adapter + 1 NVIDIA K80m GPU

*Supported on Tesla and Quadro GPUs with Mellanox InfiniBand HCA
Run CUDA applications on virtualized infrastructures, using hypervisor pass-through:

**TESTED CONFIGURATIONS**

<table>
<thead>
<tr>
<th>OS / GPU</th>
<th>RHEL7_2</th>
<th>Ubuntu14_04</th>
<th>Windows10</th>
<th>Windows 2012R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>M60</td>
<td>x86-64</td>
<td>—</td>
<td>x86-64</td>
<td>—</td>
</tr>
<tr>
<td>M40</td>
<td>—</td>
<td>x86-64</td>
<td>—</td>
<td>x86-64</td>
</tr>
<tr>
<td>M4</td>
<td>x86-64</td>
<td>—</td>
<td>—</td>
<td>x86-64</td>
</tr>
<tr>
<td>K80</td>
<td>x86-64</td>
<td>ppc64</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>K40</td>
<td>ppc64</td>
<td>ppc64</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
CUDA 8 PLATFORM SUPPORT
New OS and compilers added

OS
Linux: Fedora 23, Ubuntu 16.04, SLES 11 SP4
Mac: OS X 10.12

Compiler
Windows: VS 2015 and VS 2015 Community
Linux: PGI C++ 16.1/16.4, Clang 3.7, ICC 16.0
CUDA 8 - GET STARTED TODAY
Everything You Need to Accelerate Applications

Release Candidate Available Now

Join developer program to download CUDA & provide feedback

General Availability in August

developer.nvidia.com/cuda-toolkit
THANK YOU