Acknowledgements

To the great Nvidia people, for sharing with me ideas, material, figures, presentations, ... Particularly, for this presentation:

Mark Ebersole (webinars and slides):
  - CUDA 6.0 overview.
  - Optimizations for Kepler.

Mark Harris (SC’13 talk, webinar and “parallel for all” blog):
  - CUDA 6.0 announcements.
  - New hardware features in Maxwell.
Talk contents [49 slides]

1. The evolution of CUDA [6 slides]
2. CUDA 6.0 support [5]
3. Compiling and linking (CUDA 5.0 only) [3]
4. Dynamic parallelism (CUDA 5 & 6) [6]
5. New tools for development, debugging and optimization (CUDA 5 & 6) [1]
6. GPUDirect-RDMA (CUDA 5 & 6) [4]
7. Unified memory (CUDA 6.0 only) [13]
I. The evolution of CUDA
The impressive evolution of CUDA

Year 2008

100,000,000 CUDA-capable GPUs
150,000 CUDA downloads
1 supercomputer
60 university courses
4,000 academic papers

Year 2014

500,000,000 CUDA-capable GPUs
2,100,000 CUDA downloads
52 supercomputers
780 courses
40,000 academic papers

The CUDA software is downloaded once every minute.

Manuel Ujaldon - Nvidia CUDA Fellow
Worldwide distribution of CUDA university courses
Summary of GPU evolution

2001: First many-cores (vertex and pixel processors).
2003: Those processor become programmable (with Cg).
2006: Vertex and pixel processors unify.
2007: CUDA emerges.
2008: Double precision floating-point arithmetic.
2010: Operands are IEEE-normalized and memory is ECC.
2012: Wider support for irregular computing.
2014: The CPU-GPU memory space is unified.
Still pending: Reliability in clusters and connection to disk.
The CUDA family picture

GPU Computing Applications

Libraries and Middleware
- CUFFT
- CUBLAS
- CURAND
- CUSPARSE
- CULA
- MAGMA
- Thrust
- NPP
- VSIPL
- SVM
- OpenCurrent
- PhysX
- OptiX
- iray
- MATLAB
- Mathematica

Programming Languages
- C
- C++
- Fortran
- Java
- Python
- Wrappers
- DirectCompute
- Directives (e.g. OpenACC)

CUDA-Enabled NVIDIA GPUs

- Kepler Architecture (compute capabilities 3.x)
  - GeForce 600 Series
  - Quadro Kepler Series
  - Tesla K20
  - Tesla K10

- Fermi Architecture (compute capabilities 2.x)
  - GeForce 500 Series
  - GeForce 400 Series
  - Quadro Fermi Series
  - Tesla 20 Series

- Tesla Architecture (compute capabilities 1.x)
  - GeForce 200 Series
  - GeForce 9 Series
  - GeForce 8 Series
  - Quadro FX Series
  - Quadro Plex Series
  - Quadro NVS Series
  - Tesla 10 Series

Entertainment
Professional Graphics
High Performance Computing
CUDA 5 highlights

- **Dynamic Parallelism:** Spawn new parallel work from within GPU code (from GK110 on).
- **GPU Object Linking:** Libraries and plug-ins for GPU code.
- **New Nsight Eclipse Edition:** Develop, Debug, and Optimize... All in one tool!
- **GPUDirect:** RDMA between GPUs and PCI-express devices.

- **CUDA 5.5 is an intermediate step:** Smoothes the transition towards CUDA 6.0.
CUDA 6 highlights

Unified Memory:
- CPU and GPU can share data without much programming effort.

Extended Library Interface (XT) and Drop-in Libraries:
- Libraries much easier to use.

GPUDirect RDMA:
- A key achievement in multi-GPU environments.

Developer tools:
- Visual Profiler enhanced with:
  - Side-by-side source and disassembly view showing.
  - New analysis passes (per SM activity level), generates a kernel analysis report.
- Multi-Process Server (MPS) support in nvprof and cuda-memcheck.
- Nsight Eclipse Edition supports remote development (x86 and ARM).
II. CUDA 6.0 support
(operating systems and platforms)
Operating systems

Windows:
- XP, Vista, 7, 8, 8.1, Server 2008 R2, Server 2012.

Linux:
- Fedora 19.
- RHEL & CentOS 5, 6.
- OpenSUSE 12.3.
- SUSE SLES 11 SP2, SP3.
- Ubuntu 12.04 LTS (including ARM cross and native), 13.04.

Mac:
- OSX 10.8, 10.9.
Platforms (depending on OS).
CUDA 6 Production Release


<table>
<thead>
<tr>
<th>Platforms</th>
<th>Windows</th>
<th>Linux</th>
<th>Mac OSX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows 8.1</td>
<td></td>
<td>Notebook</td>
<td></td>
</tr>
<tr>
<td>Windows 7</td>
<td></td>
<td>Desktop</td>
<td></td>
</tr>
<tr>
<td>Windows Vista</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows XP</td>
<td></td>
<td>Desktop</td>
<td></td>
</tr>
<tr>
<td><strong>Getting Started Guide</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distribution</th>
<th>x86 64-bit</th>
<th>x86 32-bit</th>
<th>ARMv7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora 19</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>OpenSUSE 12.3</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>RHEL 6</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>CentOS 6</td>
<td>RUN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RHEL 5</td>
<td>RUN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CentOS 5</td>
<td>RUN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLES 11 (SP2 &amp; SP3)</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>Ubuntu 13.04</td>
<td>DEB</td>
<td>RUN</td>
<td>DEB</td>
</tr>
<tr>
<td>Ubuntu 12.04</td>
<td>DEB</td>
<td>RUN</td>
<td>DEB</td>
</tr>
<tr>
<td>L4T, Linux for Tegra</td>
<td>LINK</td>
<td>LINK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OSX Release</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.8</td>
<td>PKG</td>
</tr>
<tr>
<td>10.9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Getting Started Guide</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Manuel Ujaldon - Nvidia CUDA Fellow
GPUs for CUDA 6.0

CUDA Compute Capabilities 3.0 (sm_30, 2012 versions of Kepler like Tesla K10, GK104):
- Do not support dynamic parallelism nor Hyper-Q.
- Support unified memory with a separate pool of shared data with auto-migration (a subset of the memory which has many limitations).

- Support dynamic parallelism and Hyper-Q.
- Support unified memory, with similar restrictions than CCC 3.0.

CUDA Compute Capabilities 5.0 (sm_50, 2014 versions of Maxwell like GeForce GTX750Ti, GM107-GM108):
- Full support of dynamic parallelism, Hyper-Q and unified memory.
Deprecations

Things that tend to be obsolete:
- Still supported.
- Not recommended.
- New developments may not work with it.
- Likely to be dropped in the future.

Some examples:
- 32-bit applications on x86 Linux (toolkit & driver).
- 32-bit applications on Mac (toolkit & driver).
- G80 platform / sm_10 (toolkit).
Dropped support

- cuSPARSE "Legacy" API.
- Ubuntu 10.04 LTS (toolkit & driver).
- SUSE Linux Enterprise Server 11 SP1 (toolkit & driver).
- Mac OSX 10.7 (toolkit & driver).

Mac Models with the MCP79 Chipset (driver)

- iMac: 20-inch (early ’09), 24-inch (early ’09), 21.5-inch (late ’09).
- MacBook Pro: 15-inch (late’08), 17-inch (early’09), 17-inch (mid’09), 15-inch (mid ’09), 15-inch 2.53 GHz (mid’09), 13-inch (mid’09).
- Mac mini: Early ’09, Late ’09.
- MacBook Air (Late ’08, Mid ’09).
III. Compiling and linking
CUDA 4.0: Whole-program compilation and linking

CUDA 4 required a single source file for a single kernel. It was not possible to link external device code.

Include files together to build
CUDA 5.0: Separate Compilation & Linking

Now it is possible to compile and link each file separately:

That way, we can build multiple object files independently, which can later be linked to build the executable file.
CUDA 5.0: Separate Compilation & Linking

We can also combine object files into static libraries, which can be shared from different source files when linking:

- To facilitate code reuse.
- To reduce the compilation time.

- This also enables closed-source device libraries to call user-defined device callback functions.
IV. Dynamic parallelism in CUDA 5 & 6
Dynamic parallelism allows CUDA 5.0 to improve three primary issues:

- Execution
- Performance
- Programmability

- Data-dependent execution
- Recursive parallel algorithms
- Dynamic load balancing
- Thread scheduling to help fill the GPU
- Library calls from GPU kernels
- Simplify CPU/GPU division
Familiar syntax and programming model

```c
int main() {
    float *data;
    setup(data);
    A <<< ... >>> (data);
    B <<< ... >>> (data);
    C <<< ... >>> (data);
    cudaDeviceSynchronize();
    return 0;
}

__global__ void B(float *data) {
    do_stuff(data);
    X <<< ... >>> (data);
    Y <<< ... >>> (data);
    Z <<< ... >>> (data);
    cudaDeviceSynchronize();
    do_more_stuff(data);
}
```
Before CUDA 6.0: Tight limit on Pending Launch Buffer (PLB)

Applications using dynamic parallelism can launch too many grids and exhaust the pre-allocated pending launch buffer (PLB).

- Result in launch failures, sometimes intermittent due to scheduling.
- PLB size tuning can fix the problem, but often involves trial-and-error.
CUDA 6.0 uses an extended PLB (EPLB)

- EPLB guarantees all launches succeed by using a lower performance virtualized launch buffer, when fast PLB is full.
  - No more launch failures regardless of scheduling.
  - PLB size tuning provides direct performance improvement path.
  - Enabled by default.
CUDA 6.0: Performance improvements in key use cases

- Kernel launch.
- Repeated launch of the same set of kernels.
- `cudaDeviceSynchronize()`.
- Back-to-back grids in a stream.
Performance improvements on dynamic parallelism

![Graph showing performance improvements between CUDA 5, CUDA 5.5, and CUDA 6.](image URL)
V. New tools for development, debugging and optimization
New features in Nvidia Nsight, Eclipse Edition, also available for Linux and Mac OS

CUDA-aware editor:
- Automated CPU to GPU code refactoring.
- Semantic highlighting of CUDA code.
- Integrated code samples & docs.

Nsight debugger
- Simultaneously debugging of CPU and GPU code.
- Inspect variables across CUDA threads.
- Use breakpoints & single step debugging.

Nsight profiler
- Quickly identifies bottlenecks in source lines and using a unified CPU-GPU trace.
- Integrated expert system.
- Fast edit-build-profile optimization cycle.
VI. GPU Direct
Communication among GPU memories

GPU Direct 1.0 was released in Fermi to allow communications among GPUs within CPU clusters.
Kepler + CUDA 5 support GPUDirect-RDMA
[Remote Direct Memory Access]

- This allows a more direct transfer between GPUs.
- Usually, the link is PCI-express or InfiniBand.
GPUDirect-RDMA in Maxwell

The situation is more complex in CUDA 6.0 with unified memory.
Preliminary results using GPUDirect-RDMA (better perf. ahead w. CUDA 6.0 & OpenMPI)

Inter-node latency using:
- Tesla K40m GPUs (no GeForces).
- MPI MVAPICH2 library.
- ConnectX-3, IVB 3GHz.

Better MPI Applic. Scaling:
- Code: HSG (bioinformatics).
- 2 GPU nodes.
- 4 MPI processes each node.
VII. Unified memory
The idea

- CPU
- GPU
- DDR3
- GDDR5
- PCI-express (~10 GB/s.)
- Dual-, tri- or quad-channel (~100 GB/s.)
- Main memory
- Video memory

256, 320, 384 bits (~300 GB/s.)

Kepler+ GPU

Unified memory

Manuel Ujaldon - Nvidia CUDA Fellow
Unified memory contributions

- Simpler programming and memory model:
  - Single pointer to data, accessible anywhere.
  - Eliminate need for `cudaMemcpy()`.
  - Greatly simplifies code porting.

- Performance through data locality:
  - Migrate data to accessing processor.
  - Guarantee global coherency.
  - Still allows `cudaMemcpyAsync()` hand tuning.
## System requirements

<table>
<thead>
<tr>
<th></th>
<th>Required</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPU</strong></td>
<td>Kepler (GK10x+) or Maxwell (GM10x+)</td>
<td>Limited performance in CCC 3.0 and CCC 3.5</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>64 bits</td>
<td></td>
</tr>
<tr>
<td><strong>Windows</strong></td>
<td>7 or 8</td>
<td>WDDM &amp; TCC no XP/Vista</td>
</tr>
<tr>
<td><strong>Linux</strong></td>
<td>Kernel 2.6.18+</td>
<td>All CUDA-supported distros, not ARM</td>
</tr>
<tr>
<td><strong>Linux on ARM</strong></td>
<td>ARM64</td>
<td></td>
</tr>
<tr>
<td><strong>Mac OSX</strong></td>
<td></td>
<td>Not supported in CUDA 6.0</td>
</tr>
</tbody>
</table>
## CUDA memory types

<table>
<thead>
<tr>
<th>Zero-Copy (pinned memory)</th>
<th>Unified Virtual Addressing</th>
<th>Unified Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA call</td>
<td>cudaMallocHost(&amp;A, 4);</td>
<td>cudaMallocManaged(&amp;A, 4);</td>
</tr>
<tr>
<td>Allocation fixed in</td>
<td>Main memory (DDR3)</td>
<td>Video memory (GDDR5)</td>
</tr>
<tr>
<td>Local access for</td>
<td>CPU</td>
<td>Home GPU</td>
</tr>
<tr>
<td>PIC-e access for</td>
<td>All GPUs</td>
<td>Other GPUs</td>
</tr>
<tr>
<td>Other features</td>
<td>Avoid swapping to disk</td>
<td>No CPU access</td>
</tr>
<tr>
<td>Coherency</td>
<td>At all times</td>
<td>Between GPUs</td>
</tr>
<tr>
<td>Full support in</td>
<td>CUDA 2.2</td>
<td>CUDA 1.0</td>
</tr>
</tbody>
</table>

![Diagram showing memory types and features]
Additions to the CUDA API

**New call: `cudaMallocManaged()`**
- Drop-in replacement for `cudaMalloc()` allocates managed memory.
- Returns pointer accessible from both Host and Device.

**New call: `cudaStreamAttachMemAsync()`**
- Manages concurrently in multi-threaded CPU applications.

**New keyword: `__managed__`**
- Global variable annotation combines with `__device__`.
- Declares global-scope migratable device variable.
- Symbol accessible from both GPU and CPU code.
A preliminar example:
Sorting the elements from a file

<table>
<thead>
<tr>
<th>CPU code in C</th>
<th>GPU code in CUDA 6.0</th>
</tr>
</thead>
</table>
| 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);
} | 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);
} |
Before unified memory

A “deep copy” is required:

- We must copy the structure and everything that it points to. This is why C++ invented the copy constructor.
- CPU and GPU cannot share a copy of the data (coherency). This prevents `memcpy` style comparisons, checksumming and other things.

```c
struct dataElem {
    int prop1;
    int prop2;
    char *text;
}
```
The code required without unified memory

```c
void launch(dataElem *elem) {
    dataElem *g_elem;
    char *g_text;

    int textlen = strlen(elem->text);

    // Allocate storage for struct and text
    cudaMemcpy(g_elem, elem, sizeof(dataElem));
    cudaMemcpy(g_text, elem->text, textlen);
    cudaMemcpy(&(g_elem->text), &g_text, sizeof(g_text));

    // Finally we can launch our kernel, but
    // CPU and GPU use different copies of "elem"
    kernel<<< ... >>>(g_elem);
}
```

Two addresses and two copies of the data
The code required WITH unified memory

void launch(dataElem *elem) {
    kernel<<< ... >>>(elem);
}

What remains the same:
- Data movement.
- GPU accesses a local copy of text.

What has changed:
- Programmer sees a single pointer.
- CPU and GPU both reference the same object.
- There is coherence.

To pass-by-reference vs. pass-by-value you need to use C++.
An example: Linked lists

Almost impossible to manage in the original CUDA API.

The best you can do is use pinned memory:

- Pointers are global: Just as unified memory pointers.
- Performance is low: GPU suffers from PCI-e bandwidth.
- GPU latency is very high, which is particularly important for linked lists because of the intrinsic pointer chasing.
Linked lists with unified memory

Can pass list elements between CPU & GPU.
No need to move data back and forth between CPU and GPU.
Can insert and delete elements from CPU or GPU.
But program must still ensure no race conditions (data is coherent between CPU & GPU at kernel launch only).
Unified memory: Summary

- Drop-in replacement for cudaMalloc().
  - cudaMemcpy() now optional.
- Greatly simplifies code porting.
  - Less Host-side memory management.
- Enables shared data structures between CPU & GPU
  - Single pointer to data = no change to data structures.
- Powerful for high-level languages like C++.
Unified memory: Future developments

CUDA 6: Ease of Use
- Single Pointer to Data
- No Memcopy Required
- Coherence @ launch & sync
- Shared C/C++ Data Structures

Next: Optimizations
- Prefetching
- Migration Hints
- Additional OS Support

Maxwell
- System Allocator Unified
- Stack Memory Unified
- HW-Accelerated Coherence
VIII. Resources and bibliography
CUDA Zone:
Basic web resource for a CUDA programmer

[developer.nvidia.com/cuda-zone]
CUDA 6 Production Release.
Free download for all platforms and users

[developer.nvidia.com/cuda-downloads]

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux</th>
<th>Mac OS X</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>64-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 8.1</td>
<td>Notebook</td>
<td>EXE</td>
</tr>
<tr>
<td>Windows 7</td>
<td>Desktop</td>
<td>EXE</td>
</tr>
<tr>
<td>Windows Vista</td>
<td>Desktop</td>
<td>EXE</td>
</tr>
<tr>
<td>Windows XP</td>
<td>Desktop</td>
<td>EXE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distribution</th>
<th>x86 64-bit</th>
<th>x86 32-bit</th>
<th>ARMv7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fedora 19</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>OpenSUSE 12.3</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>RHEL 6</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>CentOS 6</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>RHEL 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CentOS 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLES 11 (SP2 &amp; SP3)</td>
<td>RPM</td>
<td>RUN</td>
<td></td>
</tr>
<tr>
<td>Ubuntu 13.04</td>
<td>DEB</td>
<td>RUN</td>
<td>DEB</td>
</tr>
<tr>
<td>Ubuntu 12.04</td>
<td>DEB</td>
<td>RUN</td>
<td>DEB</td>
</tr>
<tr>
<td>L4T, Linux for Tegra</td>
<td>LINK</td>
<td>LINK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OSX Release</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.8</td>
<td></td>
</tr>
<tr>
<td>10.9</td>
<td>PKG</td>
</tr>
</tbody>
</table>

Getting Started Guide
CUDA books: From 2007 to 2013

GPU Gems series [developer.vidia.com/content/GPU Gems3/gpugems3_part01.html]

List of CUDA books in [www.nvidia.com/object/cuda_books.html]
Guides for developers and more documents

- Getting started with CUDA C: Programmers guide.
  [docs.nvidia.com/cuda/cuda-c-programming-guide]
- For tough programmers: The best practices guide.
  [docs.nvidia.com/cuda/cuda-c-best-practices-guide]
- The root web collecting all CUDA-related documents:
  [docs.nvidia.com/cuda]

where we can find, additional guides for:
- Installing CUDA on Linux, MacOS and Windows.
- Optimize and improve CUDA programs on Kepler platforms.
- Check the CUDA API syntax (runtime, driver and math).
- Learn to use libraries like cuBLAS, cuFFT, cuRAND, cuSPARSE, ...
- Deal with basic tools (compiler, debugger, profiler).
Choices to accelerate your applications on GPUs and material for teaching CUDA

[developer.nvidia.com/cuda-education-training] (also available from the left lower corner of the CUDA Zone)

CUDA Education & Training

Accelerate Your Applications
Learn using step-by-step instructions, video tutorials and code samples.
- Accelerated Computing with C/C++
- Accelerate Applications on GPUs with OpenACC Directives
- Accelerated Numerical Analysis Tools with GPUs
- Drop-in Acceleration on GPUs with Libraries
- GPU Accelerated Computing with Python

Teaching Resources
Get the latest educational slides, hands-on exercises and access to GPUs for your parallel programming courses.
- Parallel Programming Training Materials
- NVIDIA Research & Academic Programs

Sign up to join the Accelerated Computing Educators Network. This network seeks to provide a collaborative area for those looking to educate others on massively parallel programming. Receive updates on new educational material, access to CUDA Cloud Training Platforms, special events for educators, and an educators focused news letter.

Sign-up Today!
Courses on-line (free access)

More than 50,000 registered users from 127 countries over the last 6 months. An opportunity to learn from CUDA masters:

- Prof. Wen-Mei Hwu (Univ. of Illinois).
- Prof. John Owens (Univ. of California at Davis).
- Dr. David Luebke (Nvidia Research).

There are two basic options:

- Introduction to parallel programming: [www.udacity.com]
- Heterogeneous parallel programming: [www.coursera.org]

If you do not have a CUDA-enabled GPU, you can even request 90 minutes tokens on Amazon EC2 instances (cloud computing):

[nvidia.qwiklab.com]

Only a supported web browser is required.
Tutorials and webinars

Presentations recorded at GTC (Graphics Technology Conference):

- 383 talks from 2013.
- More than 500 available from 2014.

[www.gputechconf.com/gtcnew/on-demand-gtc.php]

Webinars about GPU computing:

- List of past talks on video (mp4/wmv) and slides (PDF).
- List of incoming on-line talks to be enrolled.

[developer.nvidia.com/gpu-computing-webinars]

CUDACasts:

[bit.ly/cudacasts]
Examples of webinars about CUDA 6.0

GTC Express Webinar Program

GTC Express is a year-round extension of the great content available at GTC. Each month, top developers, scientists, researchers, and industry practitioners present innovative and thought-provoking webinars focused on the GPU-enabled work they're doing and sharing how GPUs are transforming their fields.

Register below for upcoming webinars and explore recordings of previous webinars.

### GTC EXPRESS SCHEDULE AT-A-GLANCE

<table>
<thead>
<tr>
<th>Date</th>
<th>Title</th>
<th>Speaker</th>
<th>Register Now</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 3, 2014, 9:00 AM PDT</td>
<td>The Next Steps for Folding@home</td>
<td>Vijay Pande, Professor, Stanford University</td>
<td></td>
</tr>
<tr>
<td>May 14, 2014, 10:00 AM PDT</td>
<td>CUDA 6: Performance Overview</td>
<td>Jonathan Cohen, Senior Manager, CUDA Libraries and Algorithms, NVIDIA</td>
<td></td>
</tr>
<tr>
<td>May 13, 2014, 9:00 AM PDT</td>
<td>An Overview of AMBER 14 - Creating the World's Fastest Molecular Dynamics Software Package</td>
<td>Ross C. Walker, University of California San Diego, Scott Le Grand, Amazon Web Services, and Adrian Roitberg, University of Florida</td>
<td></td>
</tr>
<tr>
<td>May 7, 2014, 10:00 AM PDT</td>
<td>CUDA 6: Drop-in Performance Optimized Libraries</td>
<td>NVIDIA DevTech Team</td>
<td></td>
</tr>
<tr>
<td>May 1, 2014, 10:00 AM PDT</td>
<td>CUDA 6: Unified Memory</td>
<td>Mark Ebersole, CUDA Educator, NVIDIA</td>
<td></td>
</tr>
<tr>
<td>April 23, 2014, 11:00 AM PDT</td>
<td>CUDA 6 Features Overview</td>
<td>Ujval Kapasi, CUDA Product Manager, NVIDIA</td>
<td></td>
</tr>
</tbody>
</table>
Developers

Sign up as a registered developer:
  [www.nvidia.com/paralleldeveloper]
  Access to exclusive developer downloads.
  Exclusive access to pre-release CUDA installers like CUDA 6.0.
  Exclusive activities an special offers.

Meeting point with many other developers:
  [www.gpucomputing.net]

GPU news and events:
  [www.gpgpu.org]

Technical questions on-line:
  NVIDIA Developer Forums: [devtalk.nvidia.com]
  Search or ask on: [stackoverflow.com/tags/cuda]
Developers (2)

List of CUDA-enabled GPUs:
[developer.nvidia.com/cuda-gpus]

And a last tool for tuning code: The CUDA Occupancy Calculator
[developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls]
Future developments

Nvidia’s blog contains articles unveiling future technology to be used within CUDA. It is the most reliable source about what’s next (subscription recommended):

[devblogs.nvidia.com/parallelforsall]

Some recommended articles:

“5 Powerful New Features in CUDA 6”, by Mark Harris.
“Jetson TK1: Mobile Embedded Supercomputer Takes CUDA Everywhere”, by Mark Harris.
“NVLINK, Pascal and Stacked Memory: Feeding the Appetite for Big Data”, by Denis Foley.
“CUDA Pro Tip: Increase Application Performance with NVIDIA GPU Boost”, by Mark Harris.
“CUDA 6.0 Unified Memory”, by Mark Ebersole.
Thanks!

You can always reach me in Spain at the Computer Architecture Department of the University of Malaga:

- e-mail: ujaldon@uma.es
- Phone: +34 952 13 28 24.

Or, more specifically on GPUs, visit my web page as Nvidia CUDA Fellow:

http://research.nvidia.com/users/manuel-ujaldon