

---

---

NVIDIA CUDA Software Development Kit (CUDA SDK)  
Release Notes  
Version 2.0 for MAC OSX

---

---

Please, also refer to the release notes of version 2.0 of CUDA, installed by the CUDA Toolkit installer.

---

TABLE OF CONTENTS

---

- I. Quick Start Installation Instructions
  - II. Detailed Installation Instructions
  - III. Creating Your Own CUDA Program
  - IV. Known Issues
  - V. Frequently Asked Questions
  - VI. Change Log
- 

---

I. Quick Start Instructions

---

For more detailed instructions, see section II below.

0. Install the NVIDIA Toolkit & Driver package by executing the file  
NVIDIA-MACOSX-CUDA.pkg

(Note this for MAC OSX Leopard (10.5.x))

1. Install version 2.0 of the NVIDIA CUDA SDK by executing the file  
NVIDIA\_CUDA\_SDK\_MACOSX.pkg

3. Build the SDK project examples.

```
cd /Developer/CUDA
make
```

4. Run the examples:

```
cd /Developer/CUDA/bin/darwin/release
matrixmul
```

(or any of the other executables in that directory)

See the next section for more details on installing, building, and running SDK samples.

---

II. Detailed Instructions for building the SDK

---

1. Build the SDK project examples.
- a. Go to /Developer/CUDA ("cd /Developer/CUDA")
  - b. Build:
    - release configuration by typing "make".
    - debug configuration by typing "make dbg=1".
    - emurelease configuration by typing "make emu=1".
    - emudebug configuration by typing "make emu=1 dbg=1".

Running make at the top level first builds libcutil, a utility library used by the SDK examples (libcutil is simply for convenience -- it is not a part of CUDA and is not required for your own CUDA programs). Make then builds each of the projects in the SDK.

**NOTES:**

- The release and debug configurations require a CUDA-capable GPU to run properly (see Appendix A.1 of the CUDA Programming Guide for a complete list of CUDA-capable GPUs).
- The emurelease and emudebug configurations run in device emulation mode, and therefore do not require a CUDA-capable GPU to run properly.
- You can build an individual sample by typing "make" (or "make emu=1", etc.) in that sample's project directory. For example:

```
cd /Developer/CUDA/projects/matrixmul
make emu=1
```

And then execute the sample with:

```
/Developer/CUDA/bin/darwin/emurelease/matrixmul
```

- To build just libcutil, type "make" (or "make dbg=1") in the "common" subdirectory:

```
cd /Developer/CUDA/common
make
```

4. Run the examples from the release, debug, emurelease, or emudebug directories located in /bin/darwin/[release|debug|emurelease|emudebug].

---

### III. Creating Your Own CUDA Program

---

Creating a new CUDA Program using the NVIDIA CUDA SDK infrastructure is easy. We have provided a "template" project that you can copy and modify to suit your needs. Just follow these steps:

1. Copy the template project

```
cd /Developer/CUDA/projects
cp -r template <myproject>
```

2. Edit the filenames of the project to suit your needs

```
mv template.cu myproject.cu
mv template_kernel.cu myproject_kernel.cu
mv template_gold.cpp myproject_gold.cpp
```

3. Edit the Makefile and source files. Just search and replace all occurrences of "template" with "myproject".

4. Build the project

```
make
```

You can build a debug version with "make dbg=1", an emulation version with "make emu=1", and a debug emulation with "make dbg=1 emu=1".

5. Run the program

```
../../bin/darwin/release/myproject
```

(It should print "Test PASSED")

6. Now modify the code to perform the computation you require. See the CUDA Programming Guide for details of programming in CUDA.

---

### IV. Known Issues

---

---

Note: Please see the CUDA Toolkit release notes for additional issues.

There are currently no known issues with the CUDA Toolkit

---

## V. Frequently Asked Questions

---

The Official CUDA FAQ is available online on the NVIDIA CUDA Forums:  
<http://forums.nvidia.com/index.php?showtopic=36286>

Note: Please also see the CUDA Toolkit release notes for additional Frequently Asked Questions.

---

## VI. Change Log

---

### Release 2.0

- \* Added simpleVotelntrinsics (requires GT200)

### Release 2.0 Beta

- \* Updated to the 2.0 CUDA Toolkit
- \* CUT\_DEVICE\_INIT macro modified to take command line arguments. All samples now support specifying the CUDA device to run on from the command line ("-device=n").
- \* deviceQuery sample: Updated to query number of multiprocessors and overlap flag.
- \* multiGPU sample: Renamed to simpleMultiGPU.
- \* reduction, MonteCarlo, and binomialOptions samples: updated with optional double precision support for upcoming hardware.
- \* simpleAtomics sample: Renamed to simpleAtomicIntrinsics.
- \* 7 new code samples:  
dct8x8, quasirandomGenerator, recursiveGaussian,  
simpleTexture3D, threadMigration, and volumeRender

### Release 1.1

- \* Updated to the 1.1 CUDA Toolkit
- \* Fixed several bugs in common/common.mk
- \* Removed isInteropSupported() from cutil: OpenGL interop now works on multi-GPU systems
- \* MonteCarlo sample: Improved performance. Previously it was very fast for large numbers of paths and options, now it is also very fast for small- and medium-sized runs.
- \* Transpose sample: updated kernel to use a 2D shared memory array for clarity, and optimized bank conflicts.
- \* 15 new code samples:  
asyncAPI, cudaOpenMP, eigenvalues, fastWalshTransform, histogram256,  
lineOfSight, Mandelbrot, marchingCubes, MonteCarloMultiGPU, nbody, oceanFFT,  
particles, reduction, simpleAtomics, and simpleStreams

### Release 1.0

- \* Updated to the 1.0 CUDA Toolkit.
- \* Added 4 new code samples: convolutionTexture, convolutionFFT2D, histogram64, and SobelFilter.
- \* All graphics interop samples now call the cutil library function isInteropSupported(), which returns false on machines with multiple CUDA GPUs, currently (see above).
- \* When compiling in DEBUG mode, CU\_SAFE\_CALL() now calls cuCtxSynchronize() and CUDA\_SAFE\_CALL() and CUDA\_CHECK\_ERROR() now call cudaThreadSynchronize() in order to return meaningful errors. This means that performance might suffer in DEBUG mode.