## **Accelerating Applications with CUDA C/C++**

CUDA

<https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/contents.html>

CUDA C++ Best Practices

<https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html>

Download CUDA

<https://developer.nvidia.com/cuda-downloads>

NVIDIA CUDA Compiler Driver NVCC

<https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html>

Options for Steering GPU Code Generation

<https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html#options-for-steering-gpu-code-generation>

GPU Feature List

<https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html#gpu-feature-list>

About CUDA

<https://developer.nvidia.com/about-cuda>

Profiler User’s Guide

<https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview>

Memory optimizations

<https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#memory-optimizations>

Memory management

<https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html#group__CUDART__MEMORY>

Hardware Implementation

<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#hardware-implementation>

cuFFT

<https://developer.nvidia.com/cufft>

nvGRAPH

<https://developer.nvidia.com/nvgraph>

cuBLAS

<https://developer.nvidia.com/cublas>

cuDNN

<https://developer.nvidia.com/cudnn>

GPU-ACCELERATING PRACTICE APPLICATION

<https://en.wikipedia.org/wiki/Mandelbrot_set>

<https://github.com/sol-prog/Mandelbrot_set>

GPU-Accelerated Libraries for Computing

<https://developer.nvidia.com/gpu-accelerated-libraries>

Environment Quick Start

This is a quick-start for users who just want to get going.

Use the nvidia AMI on AWS ( 10 minutes):  
<https://github.com/NVIDIA/nvidia-docker/wiki/Deploy-on-Amazon-EC2>

Get started with nvidia-docker (5 minutes):  
<https://github.com/NVIDIA/nvidia-docker>Get started with the CUDA development image (5 minutes):

"docker pull nvidia/cuda:9.1-devel" [https://hub.docker.com/r/nvidia/cuda/](https://github.com/NVIDIA/nvidia-docker)

Run CUDA code

!nvcc -arch=sm\_70 -o first-parallel 02-first-parallel/01-first-parallel.cu -run

Profile the executable that was just compiled using nsys profile.

!nsys profile --stats=true ./multi-thread-vector-add

Note on error handling

Finally, in order to catch errors that occur asynchronously, for example during the execution of an asynchronous kernel, it is essential to check the status returned by a subsequent synchronizing CUDA runtime API call, such as cudaDeviceSynchronize, which will return an error if one of the kernels launched previously should fail.

The [*CUDA Best Practices Guide*](http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#memory-optimizations), a highly recommended followup to this and other CUDA fundamentals labs, recommends a design cycle called **APOD**: **A**ssess, **P**arallelize, **O**ptimize, **D**eploy. In short, APOD prescribes an iterative design process, where developers can apply incremental improvements to their accelerated application's performance, and ship their code. As developers become more competent CUDA programmers, more advanced optimization techniques can be applied to their accelerated code bases.

Understanding Unified Memory behavior is a fundamental skill for CUDA developers, and serves as a prerequisite to many more advanced memory management techniques.

//good approximation conversion of ni and ni (100 and 200) to 32 x (100/32+1) = 128 //and 16 x (200/16+1) = 208

dim3 threads\_per\_block(32, 16, 1);

dim3 number\_of\_blocks((nj/threads\_per\_block.x)+1,(ni/threads\_per\_block.y)+1, 1);

When accelerating applications, or optimizing already-accelerated applications, take a scientific and iterative approach. Profile your application after making changes, take note, and record the implications of any refactoring on performance. Make these observations early and often: frequently, enough performance boost can be gained with little effort such that you can ship your accelerated application. Additionally, frequent profiling will teach you how specific changes to your CUDA code bases impact its actual performance: knowledge that is hard to acquire when only profiling after many kinds of changes in your code bases.

When nsys profile gives the amount of time that a kernel takes to execute, the host-to-device page faults and data migrations that occur during this kernel's execution are included in the displayed execution time.