CUDA高性能科学计算

2023-2024-1 秋季

公选-06110

Lecture 10

理学院　赖欣

授课平台/讨论QQ群

CUDA高性能科学计算(GX)
课程编号:107016


加速计算基础——CUDA C/C++
8 学时 | 中文 | 90 美元 | C/C++，CUDA ®
有培训证书

大作业

- 任选一个执行时间不少于1分钟的CPU串行程序，判断其是否可以被加速，以及热点位置
- 使用CUDA将其进行加速，并解释加速的实现过程与原理
- 经过测试，在保证结果与串行程序一致的前提下，保证加速过程的正确性
- 最后给出实际测试的加速比和提速效果
- 大作业模板见群文件，大作业电子版提交截止日期2023年12月20日
- 提交课程大作业同时需要附上Nvidia的结课证书
- Nvidia结课证书需要在11月30日之前获取

https://github.com/stars/lxidea/lists/2beaccelerated4cuda

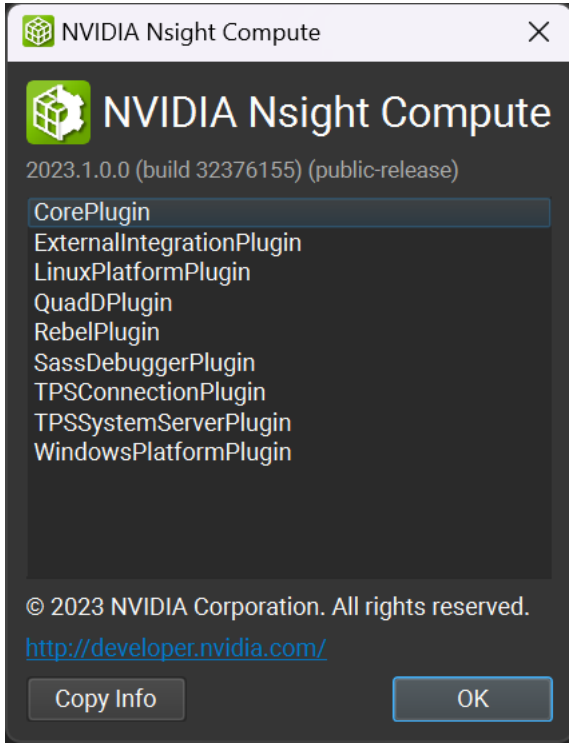
复习

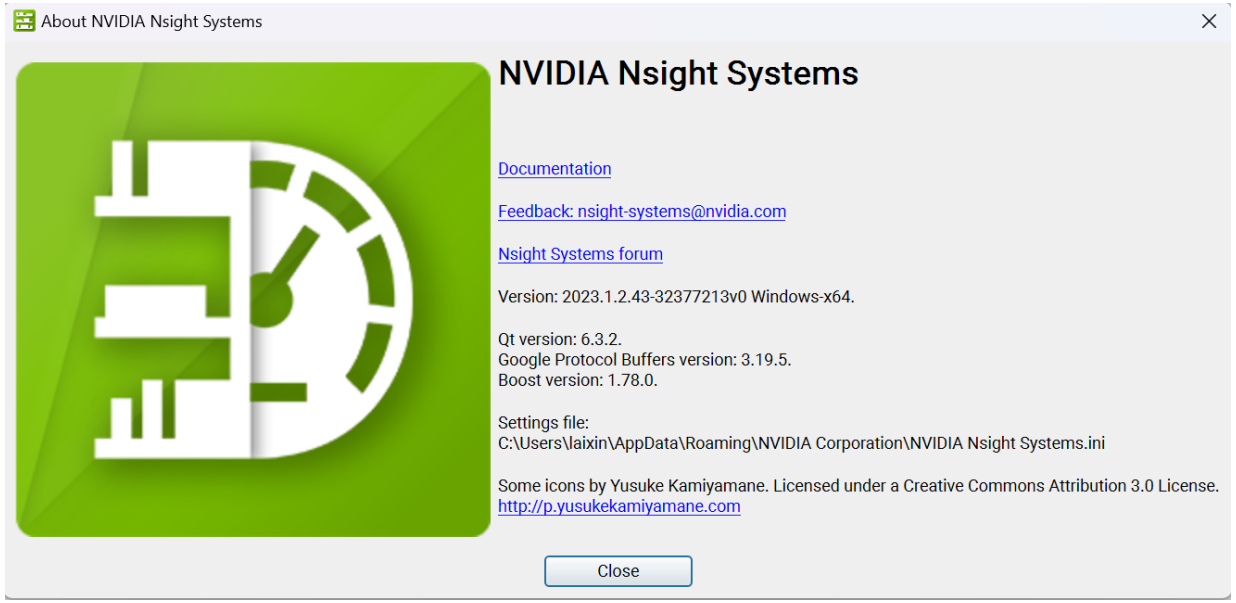
- 异构计算/CPU函数/GPU核函数
- CPU函数-循环/GPU核函数-线程跨网格循环
- CUDA C/C++/核函数 __global__, __device__
- 启动配置/网格/线程块 <=1024/流多处理器个数
- 统一内存unified memory/异步内存预取/数据迁移/页错误
- 默认流/非默认流
- Debug/CUDA错误处理
- cudaGetDevice/cudaGetDeviceProperties
- nsys/nsight-sys

- **平台：**
  - Windows – Visual Studio/Windows Subsystem Linux
  - Linux – CentOS/Suse/Ubuntu/Fedora/OpenSuse/RedHat
  - Mac

Windows平台

- Visual Studio
- Nvidia Nsight Compute
- Nvidia Nsight System





Windows平台

- Visual Studio
- Nvidia Nsight Compute
- Nvidia Nsight System
- Nvidia CUDA Toolkit

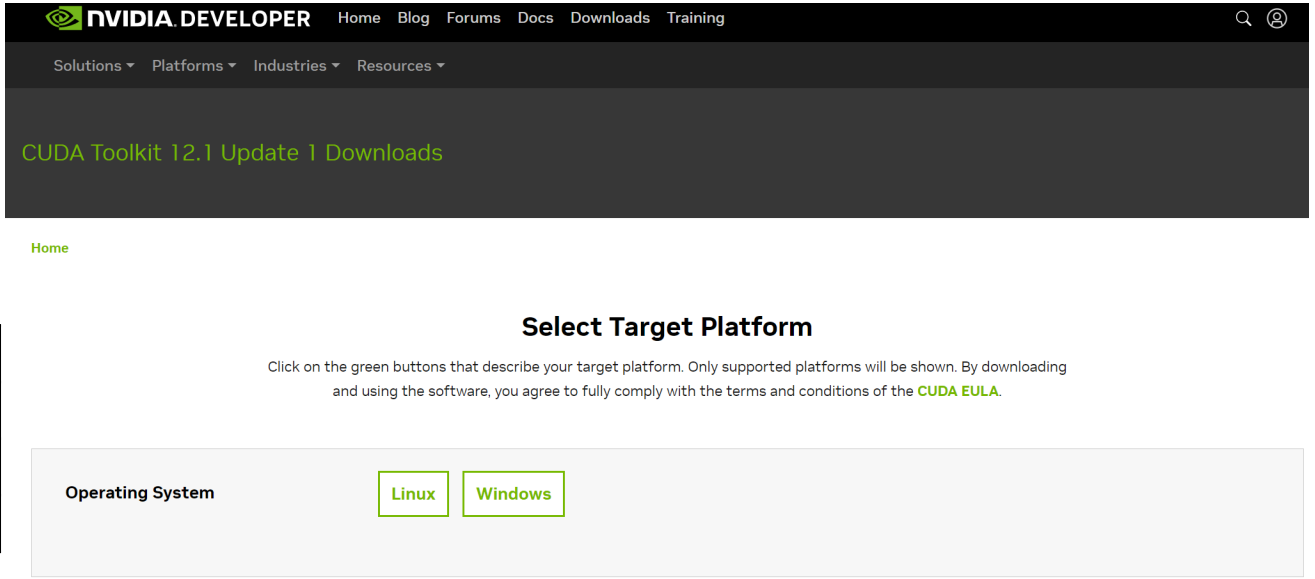
1.1. System Requirements

To use CUDA on your system, you will need the following installed:

- A CUDA-capable GPU
- A supported version of Microsoft Windows
- A supported version of Microsoft Visual Studio
- The NVIDIA CUDA Toolkit (available at https://developer.nvidia.com/cuda-downloads)

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

Operating System	Native x86_64	Cross (x86_32 on x86_64)
Windows 11	YES	NO
Windows 10	YES	NO
Windows Server 2022	YES	NO
Windows Server 2019	YES	NO



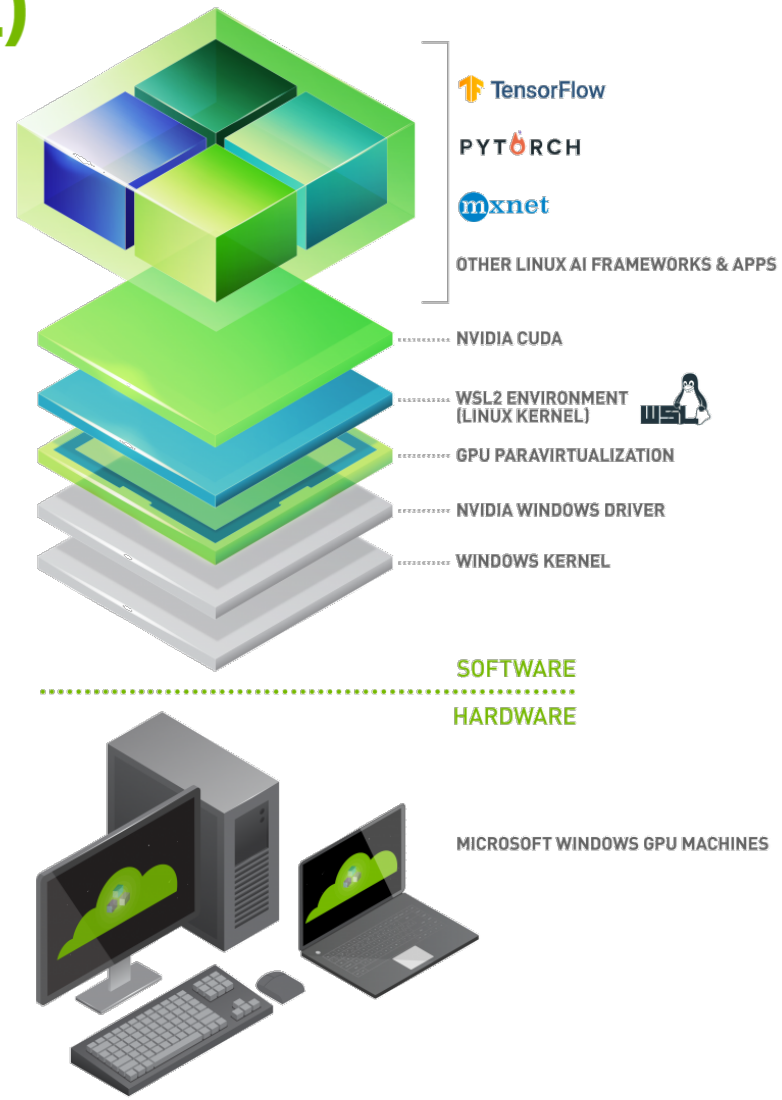
Compiler*	IDE	Native x86_64
MSVC Version 193x	Visual Studio 2022 17.0	YES
MSVC Version 192x	Visual Studio 2019 16.x	YES
MSVC Version 191x	Visual Studio 2017 15.x (RTW and all updates)	YES

https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html#cuda-compiler-new-features
https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html

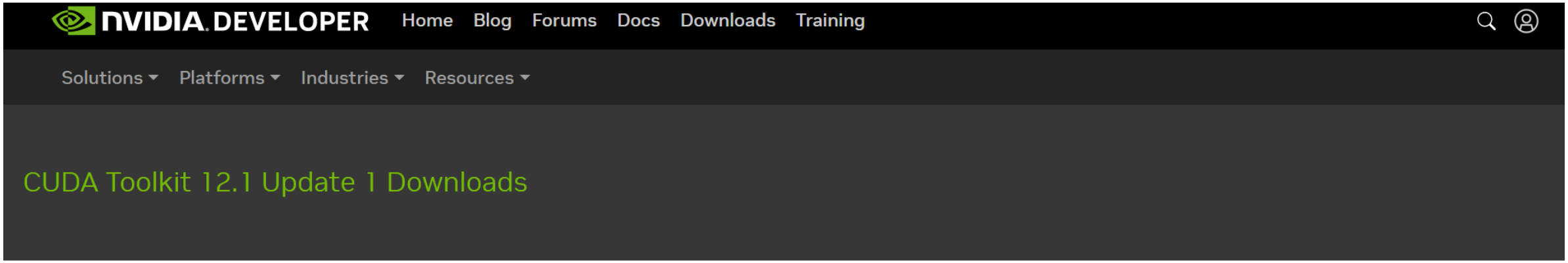
Windows Subsystem Linux

CUDA on Windows Subsystem for Linux (WSL)

- Step 1:
  - Install NVIDIA GeForce Game Ready or NVIDIA RTX Quadro Windows 11 display driver on your system with a compatible GeForce or NVIDIA RTX/Quadro card from https://www.nvidia.com/Download/index.aspx. Refer to the system requirements in the Appendix.)
- Step 2: Install WSL 2
  - wsl.exe –install
  - wsl.exe –update
- Step 3: Set Up a Linux Development Environment



https://developer.nvidia.com/cuda/wsl



Home

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the CUDA EULA.

Operating System	Linux	Windows

Installation Guide and Information

CUDA Installation Guide for Microsoft Windows

https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html

NVIDIA CUDA Installation Guide for Linux

https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

CUDA on WSL User Guide

https://docs.nvidia.com/cuda/wsl-user-guide/index.html

NVIDIA Nsight Visual Studio Edition

https://developer.nvidia.com/nsight-visual-studio-edition

从纯CUDA程序到混合编程

main.cu

```
__global__ void dosth(int *a, size_t n){
    // blahblah
}
int main(int argc, char ** argv) {
    int *a; cudaMallocManaged(&a, size);
    dosth<<<numofBlocks, numofThreads>>>(a,N);
    cudaDeviceSynchronize();
}
#: nvcc –o main main.cu -run
```

C/C++/CUDA C混合编程

main.cpp

class1.h

class1.cpp

class2.h

class2.cpp

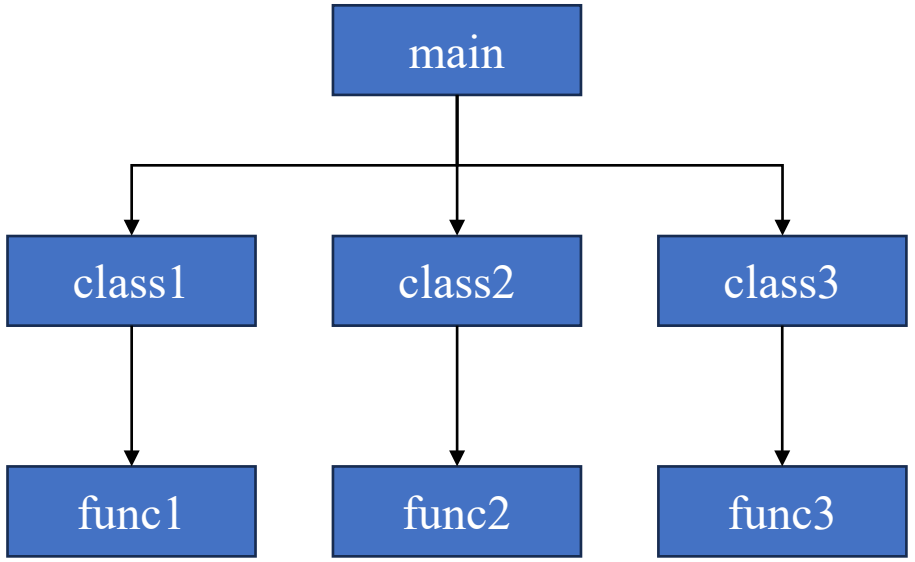
class3.h

class3.cpp

func1.cu

func2.cu

func3.cu



C/C++/CUDA C混合编程

run.cc / run.cpp

```
#include “func_ker.h”
….
func (…);
...
```

func_ker.h

```
…
void func(…);
…
```

func_ker.cu

```
#include “func_ker.h”
#include "cuda_runtime.h"
…
__global__ void func_ker(…) {
…
}
void func(…) {
…
   func_ker<<<256,1024>>>(…);
   cudaDeviceSynchronize();
…
}
```

Visual Studio CUDA 实例程序

kernel.cu

```
#include "cuda_runtime.h,,
#include "device_launch_parameters.h,,
#include <stdio.h>
cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size);
__global__ void addKernel(int *c, const int *a, const int *b)
{
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
int main(){
    const int arraySize = 5;
    const int a[arraySize] = { 1, 2, 3, 4, 5 };
    const int b[arraySize] = { 10, 20, 30, 40, 50 };
    int c[arraySize] = { 0 };
    // Add vectors in parallel.
    cudaError_t cudaStatus = addWithCuda(c, a, b, arraySize);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "addWithCuda failed!");
        return 1;
    }
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
    printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",        c[0], c[1], c[2], c[3], c[4]);
    // cudaDeviceReset must be called before exiting in order for profiling and
    // tracing tools such as Nsight and Visual Profiler to show complete traces.
    cudaStatus = cudaDeviceReset();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceReset failed!");
        return 1;
    }
    return 0;
}
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
// Helper function for using CUDA to add vectors in parallel.
cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size){
    int *dev_a = 0;
    int *dev_b = 0;
    int *dev_c = 0;
    cudaError_t cudaStatus;
 // Choose which GPU to run on, change this on a multi-GPU system.
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU
installed?");
        goto Error;
    }
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
// Allocate GPU buffers for three vectors (two input, one output)    .
    cudaStatus = cudaMalloc((void**)&dev_c, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }
    cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
 cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }
    // Copy input vectors from host memory to GPU buffers.
    cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(int),
cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
 cudaStatus = cudaMemcpy(dev_b, b, size * sizeof(int),
cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }
    // Launch a kernel on the GPU with one thread for each element.
    addKernel<<<1, size>>>(dev_c, dev_a, dev_b);
```

Visual Studio CUDA 实例程序

kernel.cu/continued

```
 // Check for any errors launching the kernel
    cudaStatus = cudaGetLastError();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "addKernel launch failed: %s\n",
cudaGetErrorString(cudaStatus));
        goto Error;    }
    // cudaDeviceSynchronize waits for the kernel to finish, and returns
    // any errors encountered during the launch.
    cudaStatus = cudaDeviceSynchronize();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching
addKernel!\n", cudaStatus);
        goto Error;    }
```

kernel.cu/continued

```
 // Copy output vector from GPU buffer to host memory.
    cudaStatus = cudaMemcpy(c, dev_c, size * sizeof(int),
cudaMemcpyDeviceToHost);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }
Error:    cudaFree(dev_c);
    cudaFree(dev_a);
    cudaFree(dev_b);
        return cudaStatus;
}
```

Visual Studio CUDA/C++混合编程实例

main.cpp

```
#include <stdio.h>
#include "cuda_runtime.h"
extern cudaError_t addWithCuda(int* c, const int* a, const int* b, unsigned int size);
int main(int argc, char* argv)
{
    const int arraySize = 5;
    const int a[arraySize] = { 1, 2, 3, 4, 5 };
    const int b[arraySize] = { 10, 20, 30, 40, 50 };
    int c[arraySize] = { 0 };
```

Visual Studio CUDA/C++混合编程实例

main.cpp/continued

```
// Add vectors in parallel.
    cudaError_t cudaStatus = addWithCuda(c, a, b, arraySize);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "addWithCuda failed!");
        return 1;
    }

    printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
        c[0], c[1], c[2], c[3], c[4]);
```

main.cpp/continued

```
 // cudaDeviceReset must be called before exiting in order for profiling and
    // tracing tools such as Nsight and Visual Profiler to show complete traces.
    cudaStatus = cudaDeviceReset();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceReset failed!");
        return 1;
    }

    return 0;
}
```

Visual Studio CUDA/C++混合编程实例

kernel.cu

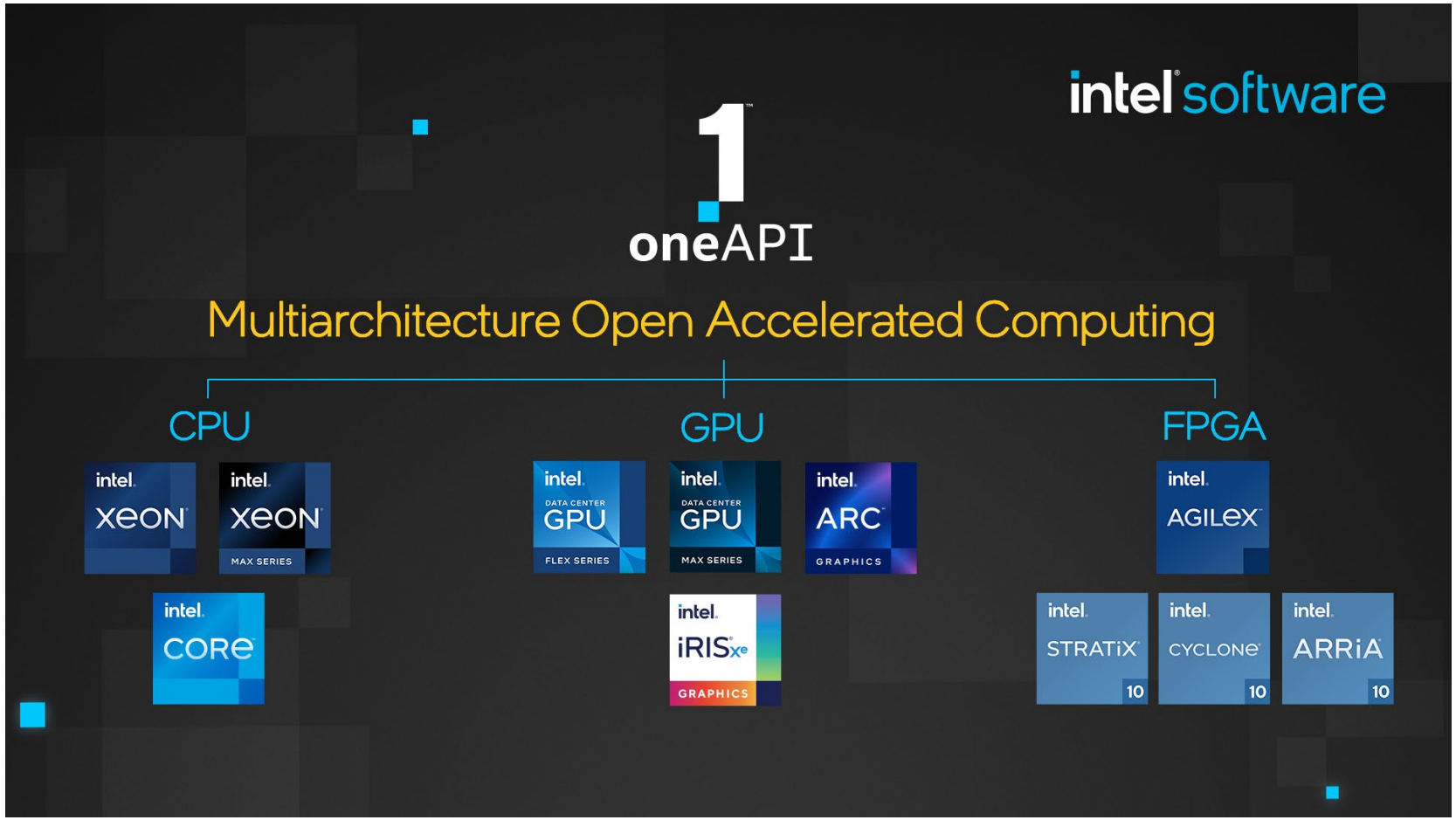
```
#include "cuda_runtime.h,,
#include "device_launch_parameters.h,,
#include <stdio.h>
__global__ void addKernel(int *c, const int *a, const int *b){
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
```

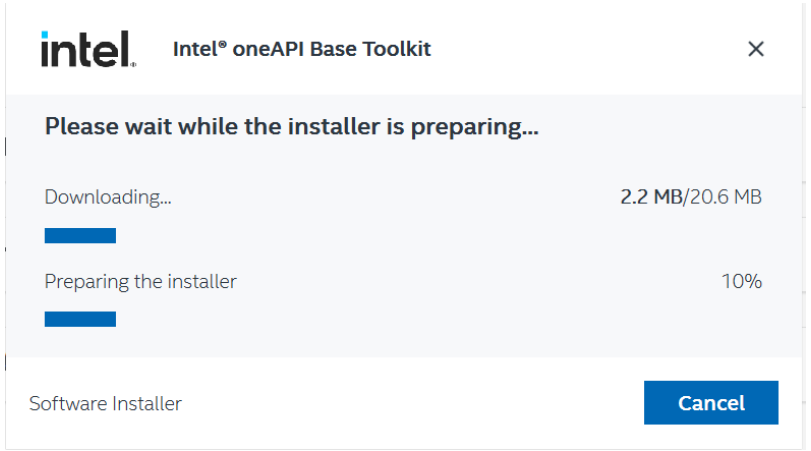
Visual Studio CUDA/C++混合编程实例

kernel.cu/continued

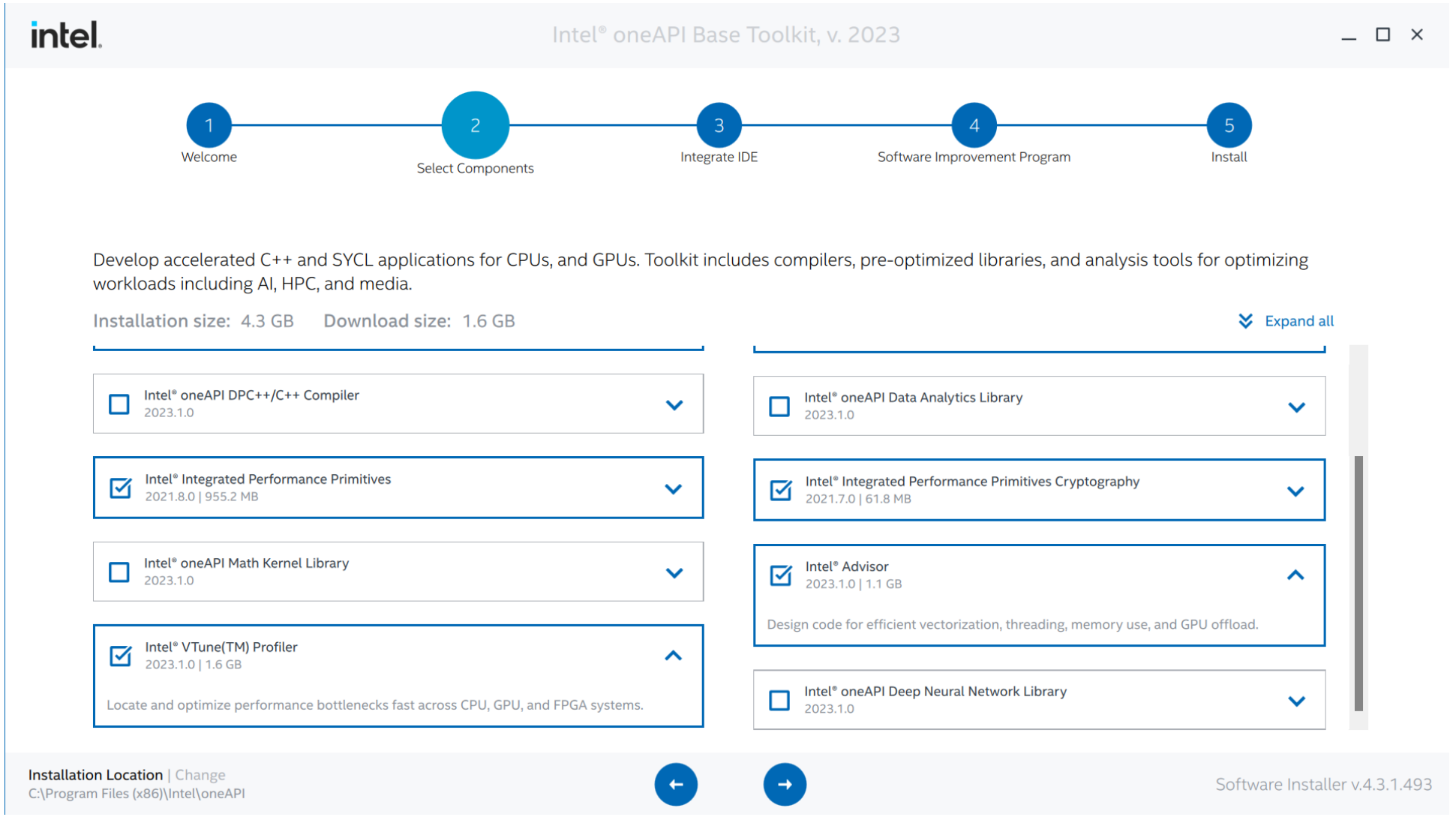
```
// Helper function for using CUDA to add vectors in parallel.
cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size){
    int *dev_a = 0;
    int *dev_b = 0;
    int *dev_c = 0;
    cudaError_t cudaStatus;
.......
 return cudaStatus;
}
```

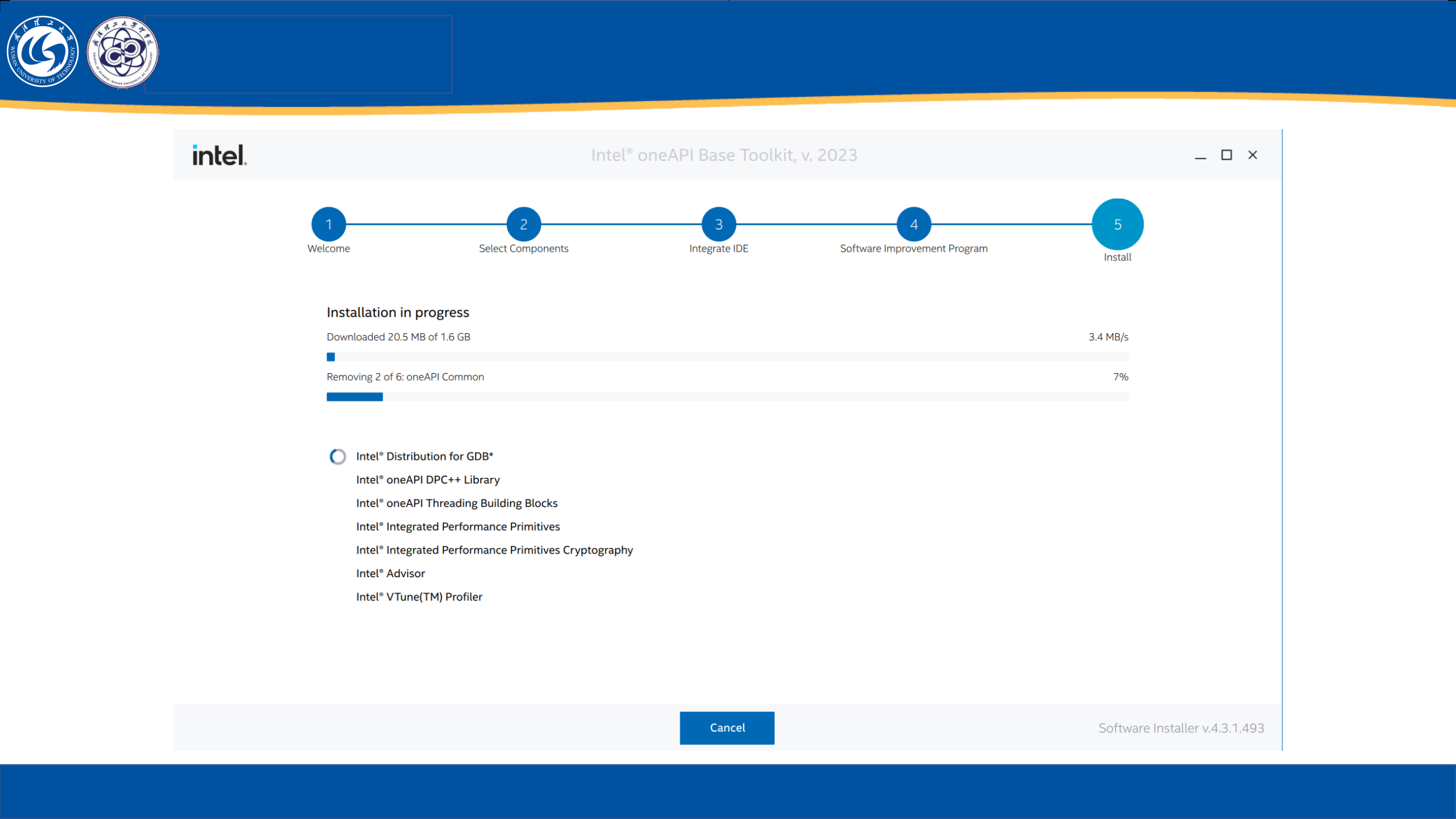
诊断与分析代码中的瓶颈





https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html


intel
Intel® oneAPI Base Toolkit, v. 2023
1
Welcome
2
Select Components
3
Integrate IDE
4
Software Improvement Program
5
Install
Develop accelerated C++ and SYCL applications for CPUs, and GPUs. Toolkit includes compilers, pre-optimized libraries, and analysis tools for optimizing workloads including AI, HPC, and media.
Installation size: 4.3 GB
Download size: 1.6 GB
Expand all
Intel® oneAPI DPC++/C++ Compiler
2023.1.0
Intel® Integrated Performance Primitives
2021.8.0 | 955.2 MB
Intel® oneAPI Math Kernel Library
2023.1.0
Intel® VTune(TM) Profiler
2023.1.0 | 1.6 GB
Locate and optimize performance bottlenecks fast across CPU, GPU, and FPGA systems.
Intel® oneAPI Data Analytics Library
2023.1.0
Intel® Integrated Performance Primitives Cryptography
2021.7.0 | 61.8 MB
Intel® Advisor
2023.1.0 | 1.1 GB
Design code for efficient vectorization, threading, memory use, and GPU offload.
Intel® oneAPI Deep Neural Network Library
2023.1.0
Installation Location | Change
C:\Program Files (x86)\Intel\oneAPI
Software Installer v.4.3.1.493


intel
Intel® oneAPI Base Toolkit, v. 2023
1
Welcome
2
Select Components
3
Integrate IDE
4
Software Improvement Program
5
Install
Installation in progress
Downloaded 20.5 MB of 1.6 GB
3.4 MB/s
Removing 2 of 6: oneAPI Common
7%
Intel® Distribution for GDB*
Intel® oneAPI DPC++ Library
Intel® oneAPI Threading Building Blocks
Intel® Integrated Performance Primitives
Intel® Integrated Performance Primitives Cryptography
Intel® Advisor
Intel® VTune(TM) Profiler
Cancel
Software Installer v.4.3.1.493

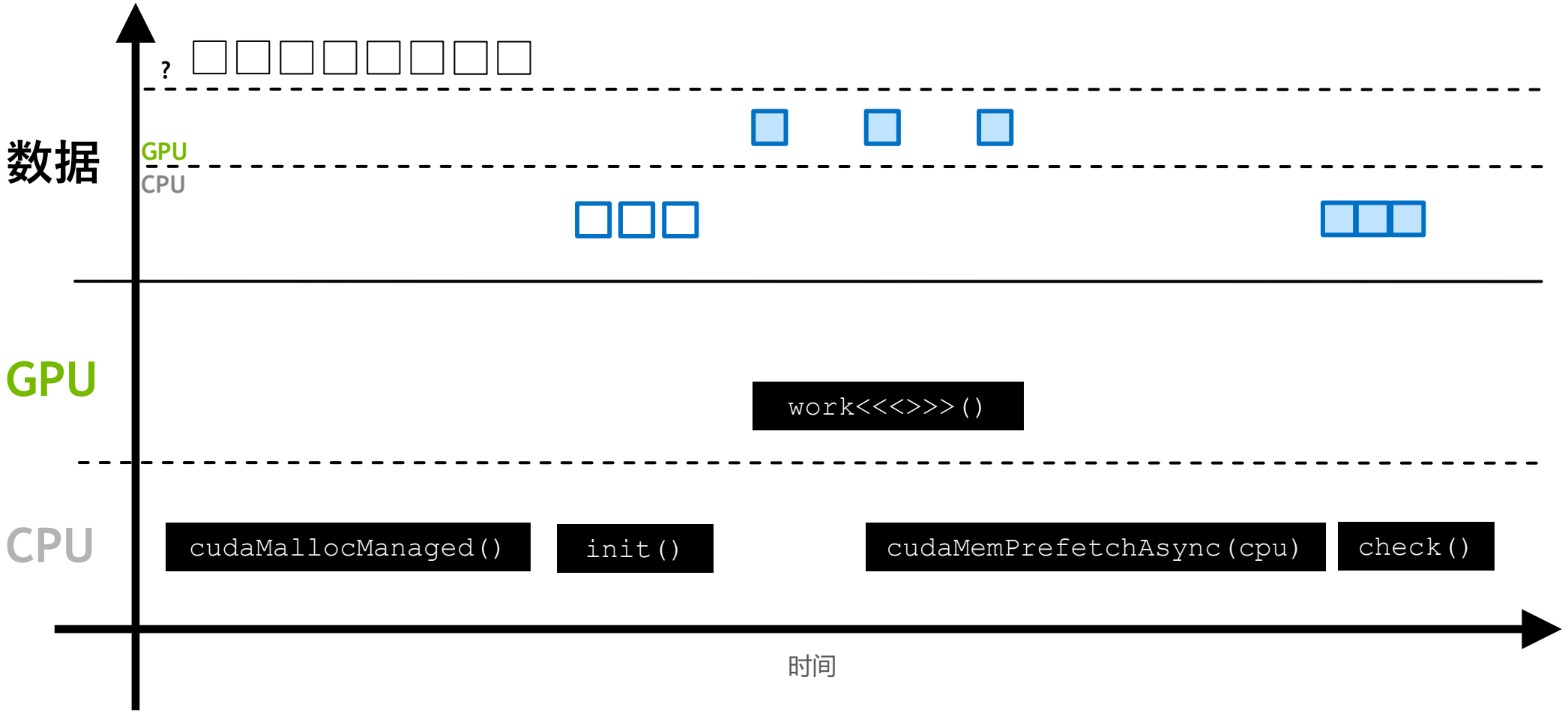
查找程序热点hotspot

https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide/2023-0/prepare-application.html

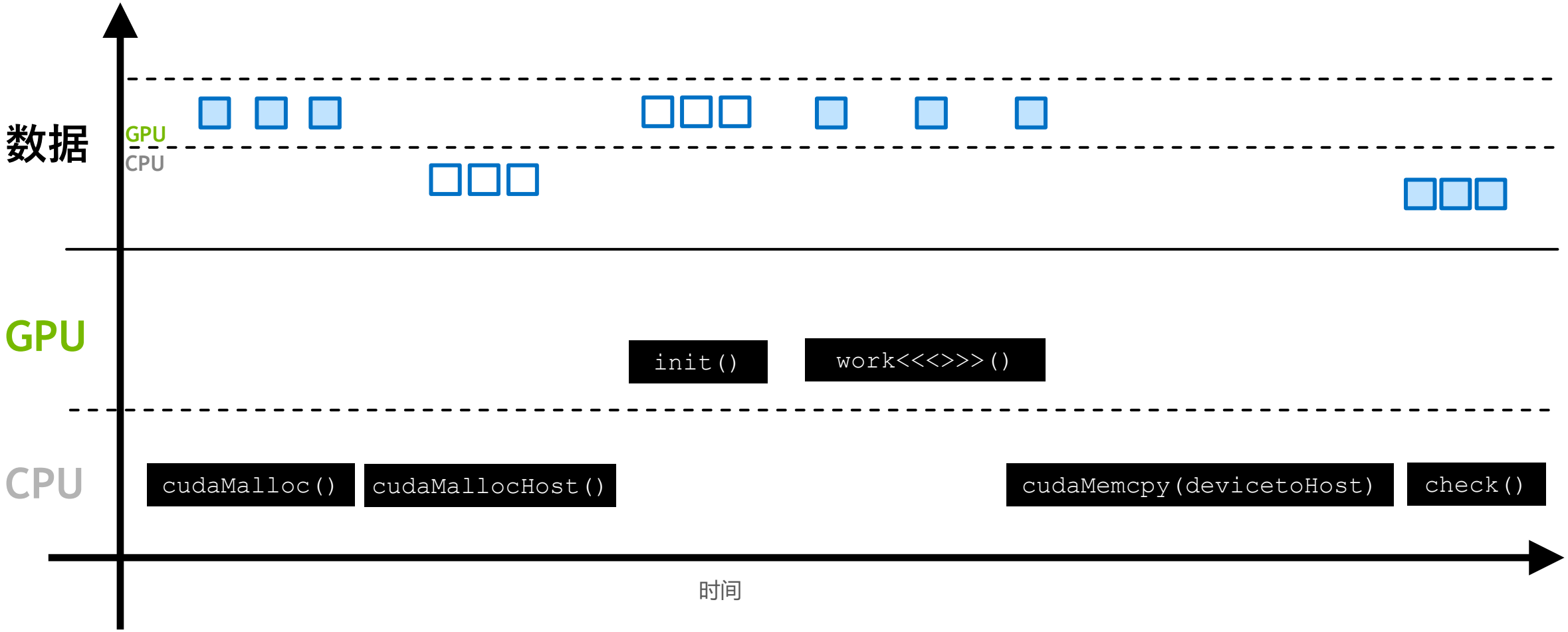
CUDA加速技术

- Unified Memory
  - cudaMallocManaged
  - cudaMalloc
  - cudaMallocHost
- Kernel function
- Multiple streaming
  - Default stream works as a barrier
- Synchronize
  - cudaDeviceSynchronize
- Data Retriving
  - cudaMemPrefetchAsync
  - cudaMemcpyAsync
  - cudaMemcpy

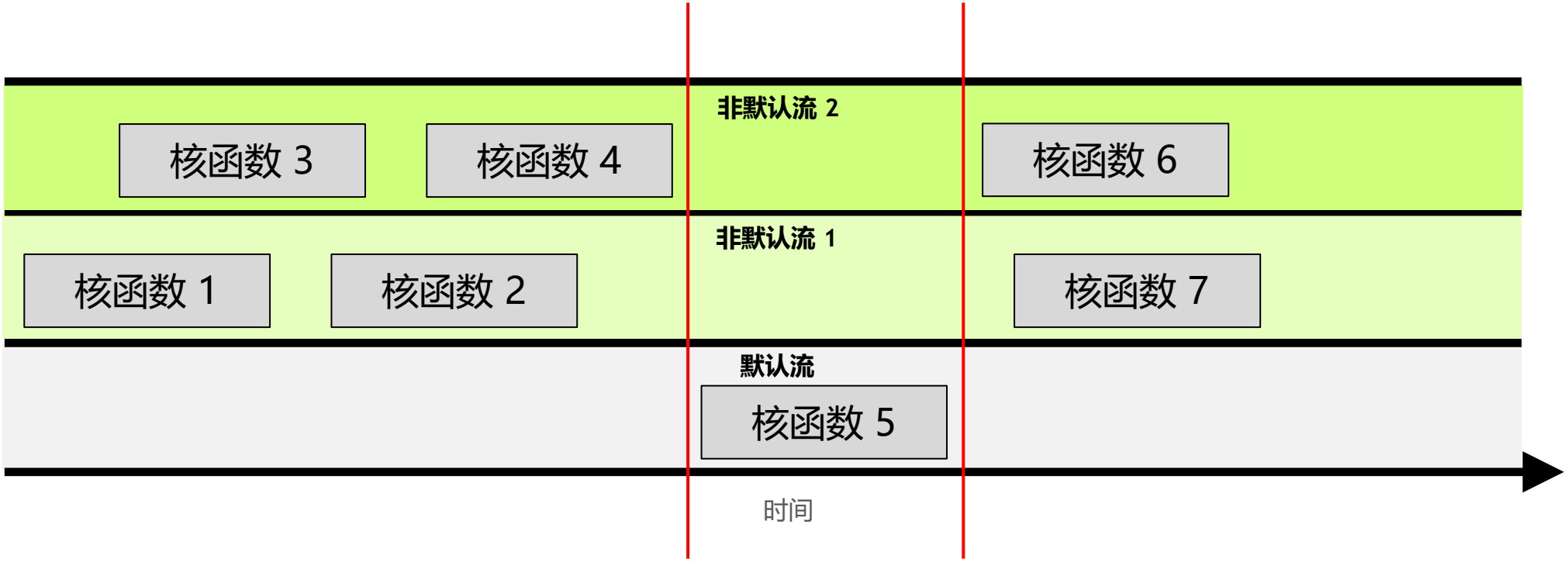
- Unified Memory (cudaMallocManaged)



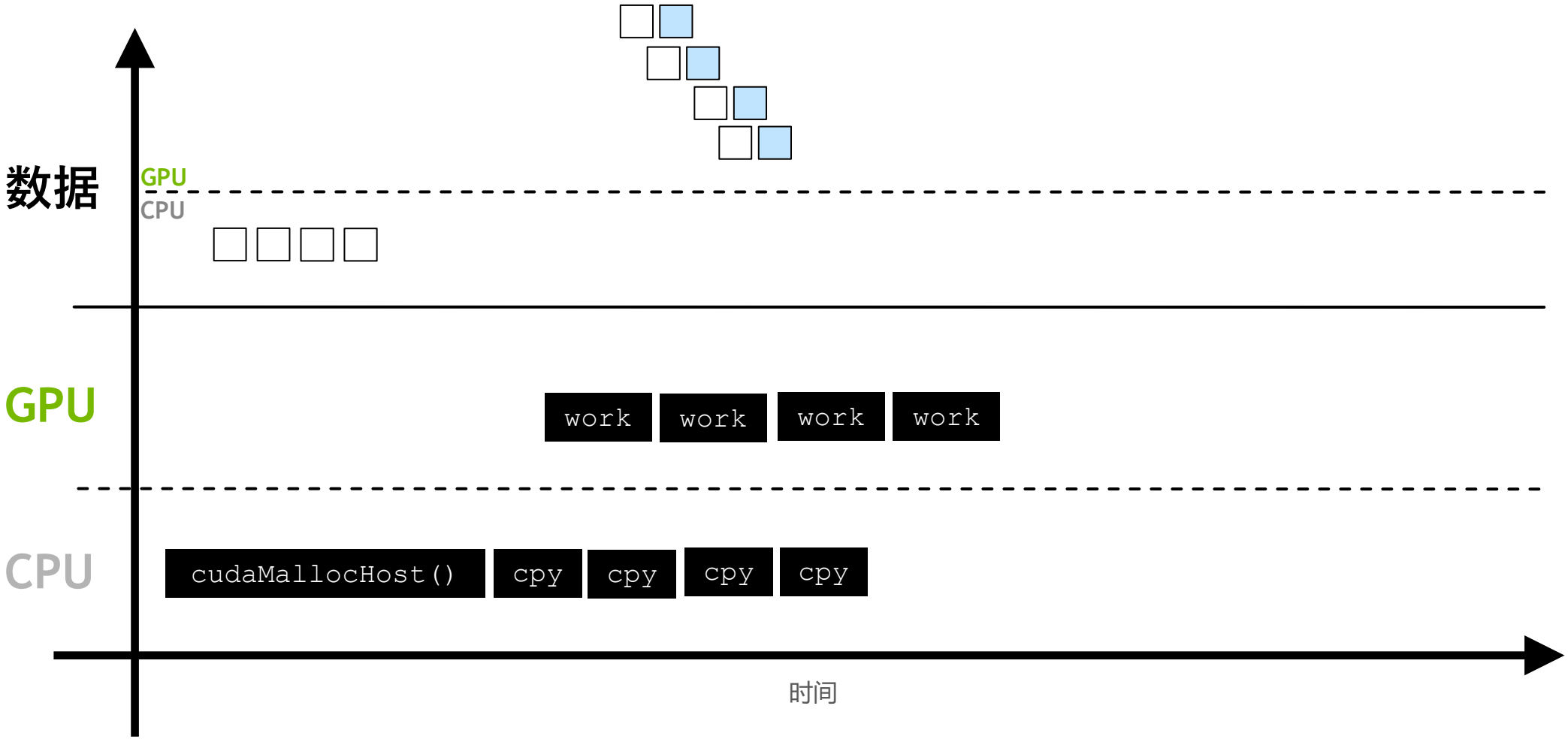
- Unified Memory (cudaMalloc/cudaMallocHost)



- Multiple streaming



- Multiple streaming / cudaMemcpyAsync



- Multiple streaming


NVIDIA