

```
1
2 #include "cuda_runtime.h"
3 #include "device_launch_parameters.h"
4
5 #include <stdio.h>
6 cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size);
7
8 __global__ void addKernel(int *c, const int *a, const int *b)
9 {
10     int i = threadIdx.x;
11     c[i] = a[i] + b[i] + 10000000;
12 }
13
14
15 int main()
16 {
17     不可变常量
18     const int arraySize = 5;
19     const int a[arraySize] = { 1, 2, 3, 4, 5 };
20     const int b[arraySize] = { 10, 20, 30, 40, 50 };
21     int c[arraySize] = { 0 };
22
23     // Add vectors in parallel.
24     cudaError_t cudaStatus = addWithCuda(c, a, b, arraySize);
25     if (cudaStatus != cudaSuccess) {
26         fprintf(stderr, "addWithCuda failed!");
27         return 1;
28     }
29
30     printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
31           c[0], c[1], c[2], c[3], c[4]);
32
33     // cudaDeviceReset must be called before exiting in order for profiling and
34     // tracing tools such as Nsight and Visual Profiler to show complete traces.
35     cudaStatus = cudaDeviceReset();
36     if (cudaStatus != cudaSuccess) {
37         fprintf(stderr, "cudaDeviceReset failed!");
38         return 1;
39     }
40
41     return 0;
42 }
43
44 // Helper function for using CUDA to add vectors in parallel.
45 cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size)
46 {
47     定义设备端变量？
48     int *dev_a = 0;
49     int *dev_b = 0;
50     int *dev_c = 0;
51     cudaError_t cudaStatus;
52
53     // Choose which GPU to run on, change this on a multi-GPU system.
54     cudaStatus = cudaSetDevice(0);
```

```
54     if (cudaStatus != cudaSuccess) {
55         fprintf(stderr, "cudaSetDevice failed! Do you have a CUDA-capable GPU
56             installed?");
57         goto Error;
58     }
59     // Allocate GPU buffers for three vectors (two input, one output)
60     cudaStatus = cudaMalloc((void**)&dev_c, size * sizeof(int));
61     if (cudaStatus != cudaSuccess) {
62         fprintf(stderr, "cudaMalloc failed!");
63         goto Error;
64     }
65
66     cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
67     if (cudaStatus != cudaSuccess) {
68         fprintf(stderr, "cudaMalloc failed!");
69         goto Error;
70     }
71
72     cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
73     if (cudaStatus != cudaSuccess) {
74         fprintf(stderr, "cudaMalloc failed!");
75         goto Error;
76     }
77
78     // Copy input vectors from host memory to GPU buffers.
79     cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(int), cudaMemcpyHostToDevice);
80     if (cudaStatus != cudaSuccess) {
81         fprintf(stderr, "cudaMemcpy failed!");
82         goto Error;
83     }
84
85     cudaStatus = cudaMemcpy(dev_b, b, size * sizeof(int), cudaMemcpyHostToDevice);
86     if (cudaStatus != cudaSuccess) {
87         fprintf(stderr, "cudaMemcpy failed!");
88         goto Error;
89     }
90
91     // Launch a kernel on the GPU with one thread for each element.
92     addKernel<<<1, size>>>>(dev_c, dev_a, dev_b);
93
94     // Check for any errors launching the kernel
95     cudaStatus = cudaGetLastError();
96     if (cudaStatus != cudaSuccess) {
97         fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString
98             (cudaStatus));
99         goto Error;
100     }
101
102     // cudaDeviceSynchronize waits for the kernel to finish, and returns
103     // any errors encountered during the launch.
104     cudaStatus = cudaDeviceSynchronize();
105     if (cudaStatus != cudaSuccess) {
```

```
105     fprintf(stderr, "cudaDeviceSynchronize returned error code %d after
        launching addKernel!\n", cudaStatus);
106     goto Error;
107 }
108
109 // Copy output vector from GPU buffer to host memory.
110 cudaStatus = cudaMemcpy(c, dev_c, size * sizeof(int), cudaMemcpyDeviceToHost);
111 if (cudaStatus != cudaSuccess) {
112     fprintf(stderr, "cudaMemcpy failed!");
113     goto Error;
114 }
115
116 Error:
117     cudaFree(dev_c);
118     cudaFree(dev_a);
119     cudaFree(dev_b);
120
121     return cudaStatus;
122 }
123
```