1 示例一

```
#include <stdio.h>

__global__ void hello_from_gpu() {
    printf("Hello World from the GPU!\n");
}

int main(void) {
    hello_from_gpu<<<1, 1>>>();
    cudaDeviceSynchronize();
    return 0;
}
```

```
#include <math.h>
#include <stdio.h>

const double EPSILON = 1.0e-15;
const double a = 1.23;
const double b = 2.34;
const double c = 3.57;
void __global__ add(const double *x, const double *y, double
↪  *z);
void check(const double *z, const int N);

int main(void) {
  // 原书中的 N 值为 1 亿（作者所用显卡的显存有 8G），已经超过我
↪  现用显卡的显存（2G）；
  // 再加上其他方面对显存的消耗，所以这里将 N 值设为 7 千万。
  const int N = 50000000;
  const int M = sizeof(double) * N;
  // 下面是在主机中分配内存
  double *h_x = (double*)malloc(M);
  if (h_x == NULL) {
```

```
19      printf("h_x: Failed to allocate host memory!\n");
20      return 1;
21    }
22    double *h_y = (double*)malloc(M);
23    if (h_y == NULL) {
24      printf("h_y: Failed to allocate host memory!\n");
25      free(h_x);
26      return 1;
27    }
28    double *h_z = (double*)malloc(M);
29    if (h_z == NULL) {
30      printf("h_z: Failed to allocate host memory!\n");
31      free(h_x);
32      free(h_z);
33      return 1;
34    }
35
36    // 数组元素初始化
37    for (int n = 0; n < N; ++n) {
38      h_x[n] = a;
39      h_y[n] = b;
40    }
41
42    double *d_x, *d_y, *d_z;
43    // 在显卡中分配显存
44    cudaError err;// 判断显存分配是否成功
45    err = cudaMalloc((void **)&d_x, M);
46    if (err != cudaSuccess) {
47      printf("d_x: cudaMemcpy Error!\n");
48      free(h_x);
49      free(h_y);
50      free(h_z);
51      return 1;
```

```
52    }
53    err = cudaMalloc((void **)&d_y, M);
54    if (err != cudaSuccess) {
55      printf("d_y: cudaMemcpy Error!\n");
56      free(h_x);
57      free(h_y);
58      free(h_z);
59      cudaFree(d_x);
60      return 1;
61    }
62    err = cudaMalloc((void **)&d_z, M);
63    if (err != cudaSuccess) {
64      printf("d_z: cudaMemcpy Error!\n");
65      free(h_x);
66      free(h_y);
67      free(h_z);
68      cudaFree(d_x);
69      cudaFree(d_y);
70      return 1;
71    }
72    // 把主机中的数据复制到显存中
73    cudaMemcpy(d_x, h_x, M, cudaMemcpyHostToDevice);
74    cudaMemcpy(d_y, h_y, M, cudaMemcpyHostToDevice);
75
76    // 设置线程块大小为 128，网格大小为 N/128=546875。
77    const int block_size = 128;
78    const int grid_size = N / block_size;
79    add<<<grid_size, block_size>>>(d_x, d_y, d_z);
80
81    // 把计算结果从显存复制到主机
82    cudaMemcpy(h_z, d_z, M, cudaMemcpyDeviceToHost);
83    // 检查结果
84    check(h_z, N);
```

```

  // 释放内存、显存
  free(h_x);
  free(h_y);
  free(h_z);
  cudaFree(d_x);
  cudaFree(d_y);
  cudaFree(d_z);
  return 0;
}

// 相加
void __global__ add(const double *x, const double *y, double
↪  *z) {
  // blockDim.x 的数值等于执行配置中变量 block_size 的数值，在
↪  本例中为 128;
  // blockIdx.x 指定一个线程在一个网格中的线程块指标，其取值范
↪  围从 0 到 gridDim.x-1。本例中，即为 0 到 546874;
  // threadIdx.x 指定一个线程在一个线程块中的线程指标，其取值范
↪  围从 0 到 blockDim.x-1。本例中，即为 0 到 127。
  // 显存中的数组与网格、线程块对应，通过下面计算索引值，让每个
↪  线程对应的数组进行计算，实现高效的并行计算。
  const int n = blockDim.x * blockIdx.x + threadIdx.x;
  z[n] = x[n] + y[n];
}

// 判断两个浮点数是否相等。注意，不能使用运算符 ==，而要将两个数
↪  的差的绝对值与一个很小的数进行比较。
// 本例中，假定当两个双精度浮点数的差的绝对值小于 1e-15
↪  (EPISILON) 时它们就是相等的。
void check(const double *z, const int N) {
  bool has_error = false;
  for (int n = 0; n < N; ++n) {
```

```
    if (fabs(z[n] - c) > EPSILON) {
      has_error = true;
      printf("z[%d] is: %f, c is: %f\n", n, z[n], c);
    }
  }
  printf("%s\n", has_error ? "Has errors": "No errors");
}
```