**Отчет по лабораторной работе № 6**

Технология CUDA

Студенты:

Потапова Алина,

Запорожская Елисавета

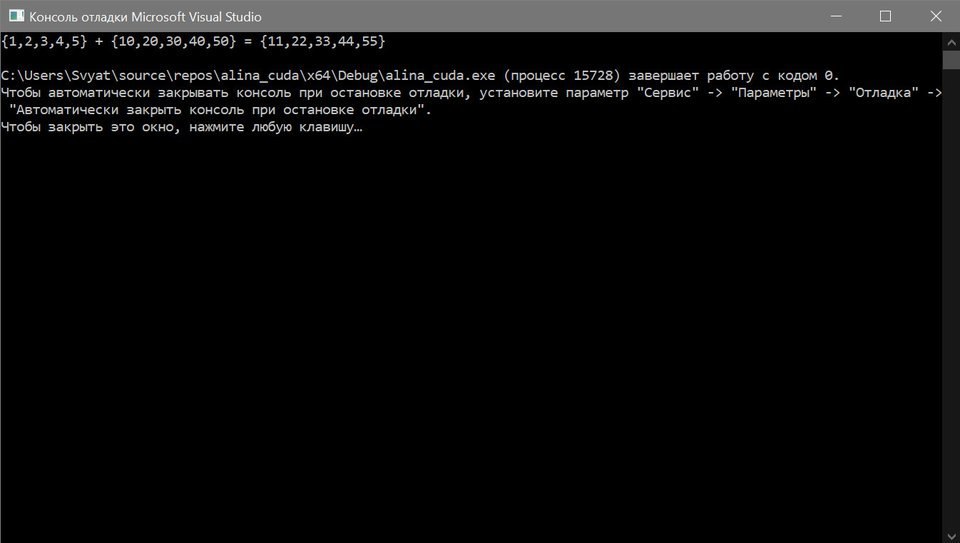
Группа:

3530202/90002

**Main.cpp**

1. #include "cuda\_runtime.h"
2. #include "device\_launch\_parameters.h"
4. #include <stdio.h>
6. cudaError\_t addWithCuda(int \*c, const int \*a, const int \*b, unsigned int size);
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];
12. }
14. int main()
15. {
16. const int arraySize = 5;
17. const int a[arraySize] = { 1, 2, 3, 4, 5 };
18. const int b[arraySize] = { 10, 20, 30, 40, 50 };
19. int c[arraySize] = { 0 };
21. // Add vectors in parallel.
22. cudaError\_t cudaStatus = addWithCuda(c, a, b, arraySize);
23. if (cudaStatus != cudaSuccess) {
24. fprintf(stderr, "addWithCuda failed!");
25. return 1;
26. }
28. printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
29. c[0], c[1], c[2], c[3], c[4]);
31. // cudaDeviceReset must be called before exiting in order for profiling and
32. // tracing tools such as Nsight and Visual Profiler to show complete traces.
33. cudaStatus = cudaDeviceReset();
34. if (cudaStatus != cudaSuccess) {
35. fprintf(stderr, "cudaDeviceReset failed!");
36. return 1;
37. }
39. return 0;
40. }
42. // Helper function for using CUDA to add vectors in parallel.
43. cudaError\_t addWithCuda(int \*c, const int \*a, const int \*b, unsigned int size)
44. {
45. int \*dev\_a = 0;
46. int \*dev\_b = 0;
47. int \*dev\_c = 0;
48. cudaError\_t cudaStatus;
50. // Choose which GPU to run on, change this on a multi-GPU system.
51. cudaStatus = cudaSetDevice(0);
52. if (cudaStatus != cudaSuccess) {
53. fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU installed?");
54. goto Error;
55. }
57. // Allocate GPU buffers for three vectors (two input, one output)
58. cudaStatus = cudaMalloc((void\*\*)&dev\_c, size \* sizeof(int));
59. if (cudaStatus != cudaSuccess) {
60. fprintf(stderr, "cudaMalloc failed!");
61. goto Error;
62. }
64. cudaStatus = cudaMalloc((void\*\*)&dev\_a, size \* sizeof(int));
65. if (cudaStatus != cudaSuccess) {
66. fprintf(stderr, "cudaMalloc failed!");
67. goto Error;
68. }
70. cudaStatus = cudaMalloc((void\*\*)&dev\_b, size \* sizeof(int));
71. if (cudaStatus != cudaSuccess) {
72. fprintf(stderr, "cudaMalloc failed!");
73. goto Error;
74. }
76. // Copy input vectors from host memory to GPU buffers.
77. cudaStatus = cudaMemcpy(dev\_a, a, size \* sizeof(int), cudaMemcpyHostToDevice);
78. if (cudaStatus != cudaSuccess) {
79. fprintf(stderr, "cudaMemcpy failed!");
80. goto Error;
81. }
83. cudaStatus = cudaMemcpy(dev\_b, b, size \* sizeof(int), cudaMemcpyHostToDevice);
84. if (cudaStatus != cudaSuccess) {
85. fprintf(stderr, "cudaMemcpy failed!");
86. goto Error;
87. }
89. // Launch a kernel on the GPU with one thread for each element.
90. addKernel<<<1, size>>>(dev\_c, dev\_a, dev\_b);
92. // Check for any errors launching the kernel
93. cudaStatus = cudaGetLastError();
94. if (cudaStatus != cudaSuccess) {
95. fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
96. goto Error;
97. }
99. // cudaDeviceSynchronize waits for the kernel to finish, and returns
100. // any errors encountered during the launch.
101. cudaStatus = cudaDeviceSynchronize();
102. if (cudaStatus != cudaSuccess) {
103. fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);
104. goto Error;
105. }
107. // Copy output vector from GPU buffer to host memory.
108. cudaStatus = cudaMemcpy(c, dev\_c, size \* sizeof(int), cudaMemcpyDeviceToHost);
109. if (cudaStatus != cudaSuccess) {
110. fprintf(stderr, "cudaMemcpy failed!");
111. goto Error;
112. }
114. Error:
115. cudaFree(dev\_c);
116. cudaFree(dev\_a);
117. cudaFree(dev\_b);
119. return cudaStatus;
120. }

**Пример выполнения программы**

****