

به نام خدا
دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

برنامه‌نویسی چندهسته‌ای

گزارش کار آزمایش ۵

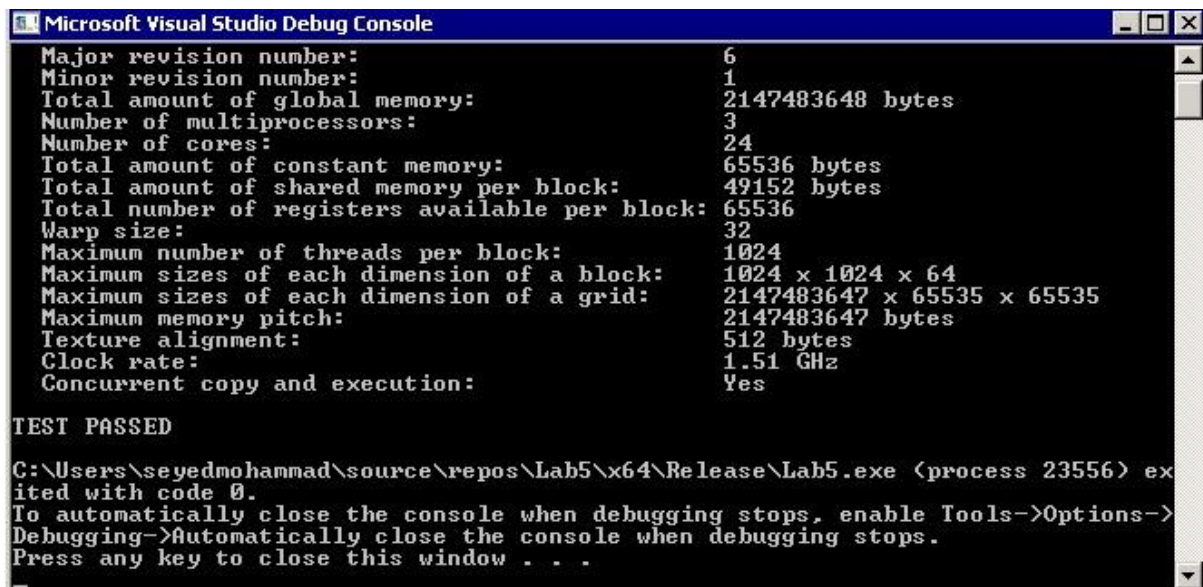
سید محمد حجازی حسینی

۹۷۳۳۰۲۰

بهار ۱۴۰۱

کد مربوط به جمع vector در vector.c و کد مربوط به نشان دادن id ها در print_id.c در پوشه‌ی Code قرار دارند.

گام (۱)



```
Microsoft Visual Studio Debug Console

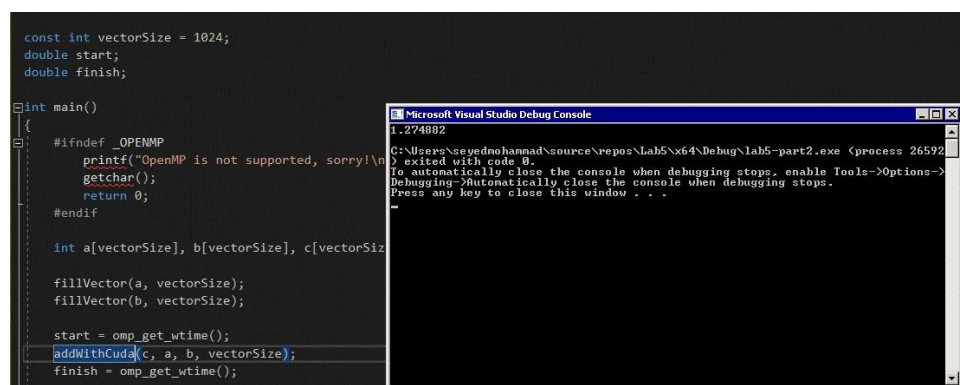
Major revision number: 6
Minor revision number: 1
Total amount of global memory: 2147483648 bytes
Number of multiprocessors: 3
Number of cores: 24
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 65536
Warp size: 32
Maximum number of threads per block: 1024
Maximum sizes of each dimension of a block: 1024 x 1024 x 64
Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Clock rate: 1.51 GHz
Concurrent copy and execution: Yes

TEST PASSED

C:\Users\seyedmohammad\source\repos\Lab5\x64\Release\Lab5.exe (process 23556) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

گام (۲)

کد به درستی با addWithCuda اجرا می‌شود:



```
const int vectorSize = 1024;
double start;
double finish;

int main()
{
    #ifndef _OPENMP
        printf("OpenMP is not supported, sorry!\n");
        getchar();
        return 0;
    #endif

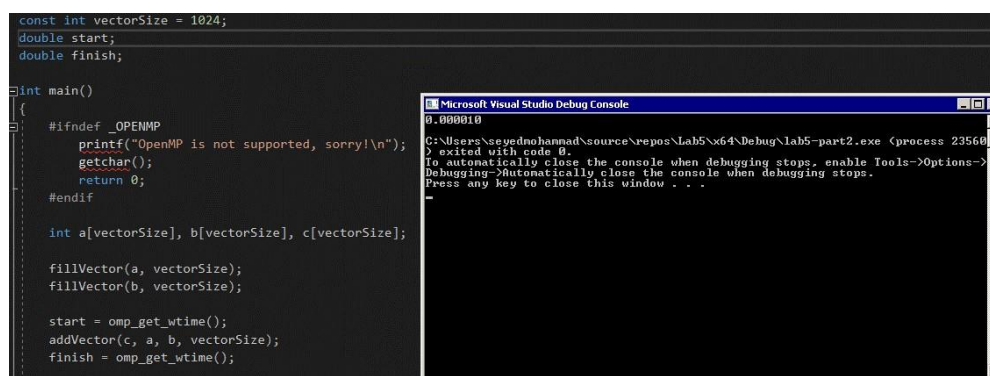
    int a[vectorSize], b[vectorSize], c[vectorSize];

    fillVector(a, vectorSize);
    fillVector(b, vectorSize);

    start = omp_get_wtime();
    addWithCuda(c, a, b, vectorSize);
    finish = omp_get_wtime();
}
```

```
Microsoft Visual Studio Debug Console
1.274882
C:\Users\seyedmohammad\source\repos\Lab5\x64\Debug\lab5-part2.exe (process 26592) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

برای اجرای سریال:



```
const int vectorSize = 1024;
double start;
double finish;

int main()
{
    #ifndef _OPENMP
        printf("OpenMP is not supported, sorry!\n");
        getchar();
        return 0;
    #endif

    int a[vectorSize], b[vectorSize], c[vectorSize];

    fillVector(a, vectorSize);
    fillVector(b, vectorSize);

    start = omp_get_wtime();
    addVector(c, a, b, vectorSize);
    finish = omp_get_wtime();
}
```

```
Microsoft Visual Studio Debug Console
0.000010
C:\Users\seyedmohammad\source\repos\Lab5\x64\Debug\lab5-part2.exe (process 23560) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

که بسیار بدتر می‌شود و علت آن رابطهی PCI Express است که به نسبت مقدار کم بردار بسیار کند است.

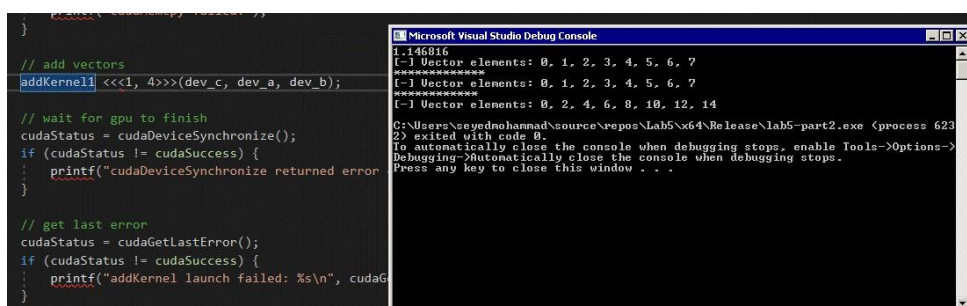
گام ۳)

برای روش اول کد kernel به این صورت نوشته می‌شود:

```
__global__ void addKernel1(int *c, const int *a, const int *b)
{
    int i = threadIdx.x;
    int offset = n * i;
    for (int j = 0; j < n; j++) {
        c[j + offset] = a[j + offset] + b[j + offset];
    }
}
```

با محاسبه‌ی offset برای هر thread کاری می‌کنیم که هر کدام ۱۰ خانه را حساب کند.

برای روش اول مقدار کم برای نمایش نتایج درستی، $block = n = 2$ و تعداد thread ها برابر ۴ است:



```
// add vectors
addKernel1 <<<1, 4>>>(dev_c, dev_a, dev_b);

// wait for gpu to finish
cudaStatus = cudaDeviceSynchronize();
if (cudaStatus != cudaSuccess) {
    printf("cudaDeviceSynchronize returned error\n");
}

// get last error
cudaStatus = cudaGetLastError();
if (cudaStatus != cudaSuccess) {
    printf("addKernel launch failed: %s\n", cudaGetLastError().message.c_str());
}
```

Microsoft Visual Studio Debug Console

```
1.146816
[-] Vector elements: 0, 1, 2, 3, 4, 5, 6, 7
*****
[-] Vector elements: 0, 1, 2, 3, 4, 5, 6, 7
*****
[-] Vector elements: 0, 2, 4, 6, 8, 10, 12, 14
C:\Users\seyedmohammad\source\repos\Lab5\64\Release\lab5-part2.exe (process 623)
2) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

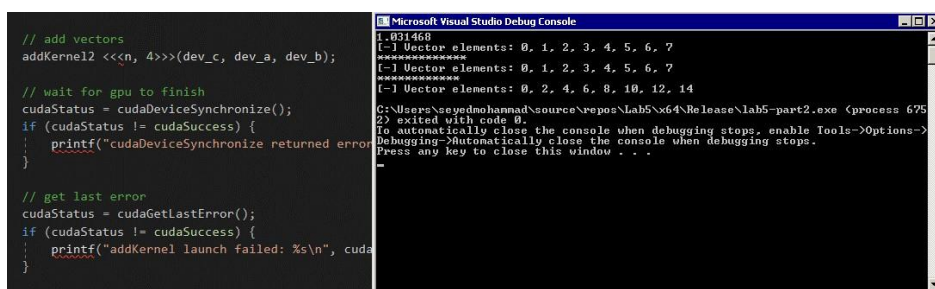
برای روش دوم به این صورت عمل می‌کنیم:

```
__global__ void addKernel2(int *c, const int *a, const int *b)
{
    int id = blockDim.x * blockIdx.x + threadIdx.x;

    while (id < n * 1024) {
        c[id] = a[id] + b[id];
        id += blockDim.x;
    }
}
```

ابتدا با محاسبه‌ی thread id برای مشخص کردن خانه‌ی اولی که باید محاسبه کند شروع می‌کنیم. سپس به اندازه‌ی thread های درون هر block به ترتیب جلو می‌رویم.

مقدار کم برای نمایش نتایج درست، $block = n = 2$ و تعداد thread ها برابر ۴ است:



```
// add vectors
addKernel2 <<<n, 4>>>(dev_c, dev_a, dev_b);

// wait for gpu to finish
cudaStatus = cudaDeviceSynchronize();
if (cudaStatus != cudaSuccess) {
    printf("cudaDeviceSynchronize returned error\n");
}

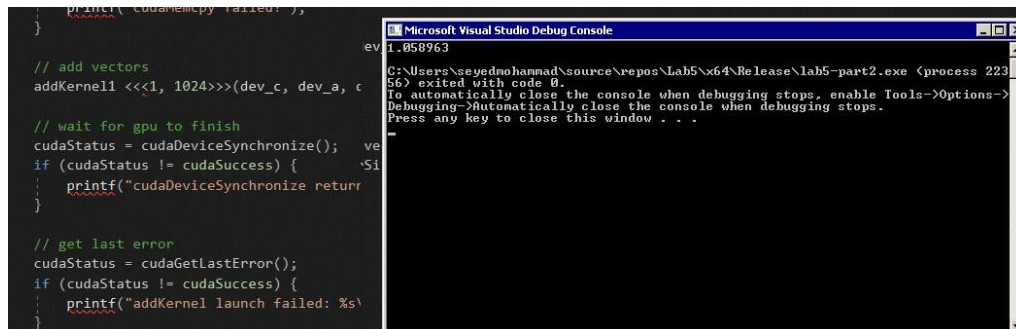
// get last error
cudaStatus = cudaGetLastError();
if (cudaStatus != cudaSuccess) {
    printf("addKernel launch failed: %s\n", cudaGetLastError().message.c_str());
}
```

Microsoft Visual Studio Debug Console

```
1.031460
[-] Vector elements: 0, 1, 2, 3, 4, 5, 6, 7
*****
[-] Vector elements: 0, 1, 2, 3, 4, 5, 6, 7
*****
[-] Vector elements: 0, 2, 4, 6, 8, 10, 12, 14
C:\Users\seyedmohammad\source\repos\Lab5\64\Release\lab5-part2.exe (process 675)
2) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

برای عدد بزرگ $size = 50 * 1024$ دو روش را مقایسه می‌کنیم؛ اعداد بزرگ‌تر مثل $100 * 1024$ تست شدند و خطای stack overflow می‌گیریم.

روش اول:

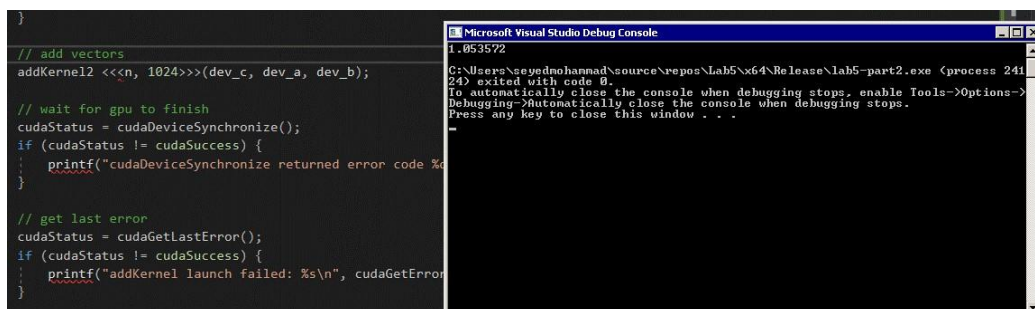


The screenshot shows a C++ code editor on the left and a Microsoft Visual Studio Debug Console on the right. The code in the editor includes comments and function calls for adding vectors and synchronizing the GPU. The debug console shows the execution path and the exit code 0, indicating a successful run.

```
printf("cudaMemcpy failed: %s", cudaGetLastError());  
}  
  
// add vectors  
addKernel1 <<<n, 1024>>>(dev_c, dev_a, dev_b);  
  
// wait for gpu to finish  
cudaStatus = cudaDeviceSynchronize();  
if (cudaStatus != cudaSuccess) {  
    printf("cudaDeviceSynchronize returned error code %d\n", cudaStatus);  
}  
  
// get last error  
cudaStatus = cudaGetLastError();  
if (cudaStatus != cudaSuccess) {  
    printf("addKernel launch failed: %s\n", cudaGetLastError());  
}
```

Microsoft Visual Studio Debug Console
1.058963
C:\Users\seyyedmohammad\source\repos\Lab5\64\Release\lab5-part2.exe (process 22356) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

روش دوم:



The screenshot shows a C++ code editor on the left and a Microsoft Visual Studio Debug Console on the right. The code in the editor is similar to the first method but uses a different kernel launch syntax. The debug console shows the execution path and the exit code 0, indicating a successful run.

```
addKernel2 <<<n, 1024>>>(dev_c, dev_a, dev_b);  
  
// wait for gpu to finish  
cudaStatus = cudaDeviceSynchronize();  
if (cudaStatus != cudaSuccess) {  
    printf("cudaDeviceSynchronize returned error code %d\n", cudaStatus);  
}  
  
// get last error  
cudaStatus = cudaGetLastError();  
if (cudaStatus != cudaSuccess) {  
    printf("addKernel launch failed: %s\n", cudaGetLastError());  
}
```

Microsoft Visual Studio Debug Console
1.053572
C:\Users\seyyedmohammad\source\repos\Lab5\64\Release\lab5-part2.exe (process 24124) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

با روش اول 1.0589 و با روش دوم 1.053 ثانیه جواب می‌گیریم. با این مقدار size تفاوت زیادی احساس نمی‌شود.

```

const int block = 2;
const int threadPerBlock = 64;
const int idsNum = 4;
const int vectorSize = block * threadPerBlock * idsNum;

int main()
{
    int a[vectorSize];
    getIds(a, vectorSize);
    for (int i = 0; i < block * threadPerBlock; i++) {
        int index = i * idsNum;
        printf("Calculated GThread: %d - Block: %d - Wrap: %d - LThread:%d\n",
            a[index + 0], a[index + 1], a[index + 2], a[index + 3]);
    }
    return EXIT_SUCCESS;
}

```

ابتدا یک تابع به اندازه‌ی $2 \times 64 \times 4$ در نظر می‌گیریم و آن را در کارت گرافیک allocate می‌کنیم. سپس kernel را حساب کرده و این آرایه را از GPU به CPU انتقال می‌دهیم و سپس در CPU، ID های مورد نظر را print می‌کنیم.

کد kernel مربوط به آن:

```

__global__ void printKernel(int *a)
{
    int globalID = blockDim.x * blockIdx.x + threadIdx.x;
    int index = globalID * idsNum;
    a[index] = globalID;           // Global Id
    a[index + 1] = blockIdx.x;    // Block Id
    a[index + 2] = threadIdx.x / warpSize; // Wrap Id
    a[index + 3] = threadIdx.x;   // Local Id
}

```

خروجی همانطور هست که انتظار داریم:

```

Microsoft Visual Studio Debug Console
Calculated GThread: 0 - Block: 0 - Wrap: 0 - LThread:0
Calculated GThread: 1 - Block: 0 - Wrap: 0 - LThread:1
Calculated GThread: 2 - Block: 0 - Wrap: 0 - LThread:2
Calculated GThread: 3 - Block: 0 - Wrap: 0 - LThread:3
Calculated GThread: 4 - Block: 0 - Wrap: 0 - LThread:4
Calculated GThread: 5 - Block: 0 - Wrap: 0 - LThread:5
Calculated GThread: 6 - Block: 0 - Wrap: 0 - LThread:6
Calculated GThread: 7 - Block: 0 - Wrap: 0 - LThread:7
Calculated GThread: 8 - Block: 0 - Wrap: 0 - LThread:8
Calculated GThread: 9 - Block: 0 - Wrap: 0 - LThread:9
Calculated GThread: 10 - Block: 0 - Wrap: 0 - LThread:10
Calculated GThread: 11 - Block: 0 - Wrap: 0 - LThread:11
Calculated GThread: 12 - Block: 0 - Wrap: 0 - LThread:12
Calculated GThread: 13 - Block: 0 - Wrap: 0 - LThread:13
Calculated GThread: 14 - Block: 0 - Wrap: 0 - LThread:14
Calculated GThread: 15 - Block: 0 - Wrap: 0 - LThread:15
Calculated GThread: 16 - Block: 0 - Wrap: 0 - LThread:16
Calculated GThread: 17 - Block: 0 - Wrap: 0 - LThread:17
Calculated GThread: 18 - Block: 0 - Wrap: 0 - LThread:18
Calculated GThread: 19 - Block: 0 - Wrap: 0 - LThread:19
Calculated GThread: 20 - Block: 0 - Wrap: 0 - LThread:20
Calculated GThread: 21 - Block: 0 - Wrap: 0 - LThread:21
Calculated GThread: 22 - Block: 0 - Wrap: 0 - LThread:22
Calculated GThread: 23 - Block: 0 - Wrap: 0 - LThread:23
Calculated GThread: 24 - Block: 0 - Wrap: 0 - LThread:24
Calculated GThread: 25 - Block: 0 - Wrap: 0 - LThread:25
Calculated GThread: 26 - Block: 0 - Wrap: 0 - LThread:26
Calculated GThread: 27 - Block: 0 - Wrap: 0 - LThread:27
Calculated GThread: 28 - Block: 0 - Wrap: 0 - LThread:28
Calculated GThread: 29 - Block: 0 - Wrap: 0 - LThread:29
Calculated GThread: 30 - Block: 0 - Wrap: 0 - LThread:30
Calculated GThread: 31 - Block: 0 - Wrap: 0 - LThread:31
Calculated GThread: 32 - Block: 0 - Wrap: 1 - LThread:32
Calculated GThread: 33 - Block: 0 - Wrap: 1 - LThread:33
Calculated GThread: 34 - Block: 0 - Wrap: 1 - LThread:34
Calculated GThread: 35 - Block: 0 - Wrap: 1 - LThread:35
Calculated GThread: 36 - Block: 0 - Wrap: 1 - LThread:36
Calculated GThread: 37 - Block: 0 - Wrap: 1 - LThread:37
Calculated GThread: 38 - Block: 0 - Wrap: 1 - LThread:38
Calculated GThread: 39 - Block: 0 - Wrap: 1 - LThread:39
Calculated GThread: 40 - Block: 0 - Wrap: 1 - LThread:40
Calculated GThread: 41 - Block: 0 - Wrap: 1 - LThread:41
Calculated GThread: 42 - Block: 0 - Wrap: 1 - LThread:42
Calculated GThread: 43 - Block: 0 - Wrap: 1 - LThread:43
Calculated GThread: 44 - Block: 0 - Wrap: 1 - LThread:44
Calculated GThread: 45 - Block: 0 - Wrap: 1 - LThread:45
Calculated GThread: 46 - Block: 0 - Wrap: 1 - LThread:46
Calculated GThread: 47 - Block: 0 - Wrap: 1 - LThread:47
Calculated GThread: 48 - Block: 0 - Wrap: 1 - LThread:48

```