Министерство цифрового развития, связи
и массовых коммуникаций Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Кафедра прикладной математики и кибернетики

ЛАБОРАТОРНАЯ РАБОТА №8

По дисциплине: «Программирование графических процессоров»

Выполнили:

Студенты 3 курса группы ИП-111
Корнилов А.А.,
Попов М.И.,
Толкач А.А.

Проверил:

Профессор кафедры ПМиК
Малков Е.А.

Новосибирск, 2024

Задание:

- Реализовать вычисление на GPU произведения матриц, используя CUDA API (“сырой код”) и, отдельно, используя библиотеку cuBLAS. Сравнить время выполнения программ при различной размерности матриц.

Цель: освоить использование библиотеки cuBLAS.

Оборудование: Видеокарта GTX 1050TI (Pascal)

Выполнение работы:

Для выполнения работы была написана программа с использование функции cublasSgemm для перемножения матриц, в отличии от других программ для CUDA для cuBLAS нужно объявлять её дескриптор в нашем случае (cublasHandle_t handle), далее слеует обычное выделение памяти и вызов функции cublasSgemm в которой мы указываем CUBLAS_OP_N для указывает что не нужно транспонировать для умножения, размеры матриц NNN, скаляр для первой матрицы alpha равный 1.0f – то есть * на 1 входную матрицу, указываем входные матрицы d_A и d_B и их размеры NN, beta выходной скаляр который прибавляется к выходной матрице, выходная матрица d_C и её размер N :

```
#include <iostream>
#include <iomanip>
#include <cuda_runtime.h>
#include <cublas_v2.h>

void initMatrix(float *matrix, int rows, int cols) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            matrix[i * cols + j] = i + j;
        }
    }
}
void printMatrix(float *matrix, int rows, int cols) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            std::cout << matrix[i * cols + j] << "\t";
        }
        std::cout << std::endl;
    }
}

int main() {
    const int num =  1 << 2;
    int N = 3 * num;
    float elapsedTime = 0;
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
```

```
    float *h_A = new float[N * N];
    float *h_B = new float[N * N];
    float *h_C = new float[N * N];

    initMatrix(h_A, N, N);
    initMatrix(h_B, N, N);

    cublasHandle_t handle;
    cublasCreate(&handle);

    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, N * N * sizeof(float));
    cudaMalloc(&d_B, N * N * sizeof(float));
    cudaMalloc(&d_C, N * N * sizeof(float));

    cudaMemcpy(d_A, h_A, N * N * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, N * N * sizeof(float), cudaMemcpyHostToDevice);

    float alpha = 1.0f, beta = 0.0f;

    cudaEventRecord(start, 0);
    cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, N, N, N, &alpha, d_A, N, d_B, N,
&beta, d_C, N);
    cudaDeviceSynchronize();
    cudaEventRecord(stop, 0);

    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsedTime, start, stop);
    std::cout << "Time using cuBLAS code: " <<  std::setprecision(15) <<
elapsedTime <<  std::endl;

    cudaMemcpy(h_C, d_C, N * N * sizeof(float), cudaMemcpyDeviceToHost);

    std::cout << "Matrix A:" << std::endl;
    printMatrix(h_A, N, N);
    std::cout << std::endl;

    std::cout << "Matrix B:" << std::endl;
    printMatrix(h_B, N, N);
    std::cout << std::endl;

    std::cout << "End matrix C:" << std::endl;
    printMatrix(h_C, N, N);

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    cublasDestroy(handle);
    delete[] h_A;
    delete[] h_B;
    delete[] h_C;
    return 0;
}
```

Листинг 1 – программа LR08_1.cu

Команда компиляции и результат работы программы:

```
PS D:\Projects\CUDA_CMake\LR08\src> nvcc .\LR08_1G.cu -lcublas
LR08_1G.cu
tmpxft_00042630_00000000-10_LR08_1G.cudafe1.cpp
  Создается библиотека a.lib и объект a.exp
PS D:\Projects\CUDA_CMake\LR08\src> .\a.exe
Time using cuBLAS code: 0.647104024887085
Matrix A:
0    1    2    3    4    5    6    7    8
1    2    3    4    5    6    7    8    9
2    3    4    5    6    7    8    9    10
3    4    5    6    7    8    9    10   11
4    5    6    7    8    9    10   11   12
5    6    7    8    9    10   11   12   13
6    7    8    9    10   11   12   13   14
7    8    9    10   11   12   13   14   15
8    9    10   11   12   13   14   15   16

Matrix B:
0    1    2    3    4    5    6    7    8
1    2    3    4    5    6    7    8    9
2    3    4    5    6    7    8    9    10
3    4    5    6    7    8    9    10   11
4    5    6    7    8    9    10   11   12
5    6    7    8    9    10   11   12   13
6    7    8    9    10   11   12   13   14
7    8    9    10   11   12   13   14   15
8    9    10   11   12   13   14   15   16

End matrix C:
204  240  276  312  348  384  420  456  492
240  285  330  375  420  465  510  555  600
276  330  384  438  492  546  600  654  708
312  375  438  501  564  627  690  753  816
348  420  492  564  636  708  780  852  924
```

384	465	546	627	708	789	870	951	1032
420	510	600	690	780	870	960	1050	1140
456	555	654	753	852	951	1050	1149	1248
492	600	708	816	924	1032	1140	1248	1356

Также была написана программа используя «сырые» компоненты CUDA для перемножения матриц.

```
#include <iostream>
#include <cuda_runtime.h>
#include <iomanip>

__global__ void matrixMultiply(float *a, float *b, float *c, int n) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < n && col < n) {
        int sum = 0;
        for (int i = 0; i < n; ++i) {
            sum += a[row * n + i] * b[i * n + col];
        }
        c[row * n + col] = sum;
    }
}
void initMatrix(float *matrix, int rows, int cols) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            matrix[i * cols + j] = i + j;
        }
    }
}
void printMatrix(float *matrix, int rows, int cols) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            std::cout << matrix[i * cols + j] << "\t";
        }
        std::cout << std::endl;
    }
}

int main() {
    const int num =  1 << 12;
    int N = 3 * num;
    float elapsedTime = 0;
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    float *h_A = new float[N * N];
    float *h_B = new float[N * N];
    float *h_C = new float[N * N];
```

```
    initMatrix(h_A, N, N);
    initMatrix(h_B, N, N);

    float *d_A, *d_B, *d_C;
    cudaMalloc(&d_A, N * N * sizeof(float));
    cudaMalloc(&d_B, N * N * sizeof(float));
    cudaMalloc(&d_C, N * N * sizeof(float));

    cudaMemcpy(d_A, h_A, N * N * sizeof(float), cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, N * N * sizeof(float), cudaMemcpyHostToDevice);

    dim3 blockSize(1024, 1024);
    dim3 gridSize((N + blockSize.x - 1) / blockSize.x, (N + blockSize.y - 1) /
blockSize.y);

    cudaEventRecord(start, 0);
    matrixMultiply<<<gridSize, blockSize>>>(d_A, d_B, d_C, N);
    cudaDeviceSynchronize();
    cudaEventRecord(stop, 0);

    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsedTime, start, stop);
    std::cout << "Time using CUDA code: " <<  std::setprecision(15) << elapsedTime
<<  std::endl;

    cudaMemcpy(h_C, d_C, N * N * sizeof(float), cudaMemcpyDeviceToHost);

/*    std::cout << "Matrix A:" << std::endl;
    printMatrix(h_A, N, N);
    std::cout << std::endl;

    std::cout << "Matrix B:" << std::endl;
    printMatrix(h_B, N, N);
    std::cout << std::endl;

    std::cout << "End matrix C:" << std::endl;
    printMatrix(h_C, N, N);*/

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    delete[] h_A;
    delete[] h_B;
    delete[] h_C;
    return 0;
}
```

Результат работы программы:

```
PS D:\Projects\CUDA_CMake\LR08\src> nvcc .\LR08_2G.cu
LR08_2G.cu
tmpxft_00004d9c_00000000-10_LR08_2G.cudafe1.cpp
  Создается библиотека a.lib и объект a.exp
```

```
PS D:\Projects\CUDA_CMake\LR08\src> .\a.exe
Time using CUDA code: 0.526880025863647
```

К сожалению nvprof не получилось применить к программе.

	CUDA	cuBLAS
1 << 2	0.411648005247116	0.721920013427734
1 << 4	0.485983995199203	0.650367975234985
1 << 6	0.843871994018555	1.64681601524353
1 << 8	1.406495988368988	5.50540781021118
1 << 10	2.516704022884369	38.0118408203125
1 << 12	8.503935992717743	1456.44482421875

В результате программы мы видим что с повышение размера матрицы, время работы с cuBLAS повышается

Вывод: в ходе выполнения лабораторной работы, была исследована и применена работа с библиотекой cuBLAS.