Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Архитектура параллельных вычислительных систем»

Студент	Никифорова Ирина Андреевна
Группа	РК6-12М
Тип задания	лабораторная работа
Тема лабораторной работы	№2. CUDA

Студент _____ **Никифорова И.А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Жук Д.М.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2020 г.

Оглавление

Задание на лабораторную работу

С помощью неявной разностной схемы решить нестационарное уравнение теплопроводности для прямоугольной пластины размером 5*5см. Начальное значение температуры пластины - 20 градусов.

Граничные условия следующие: левая граница теплоизолирована, на нижней границе поддерживается 20, на остальной части границы температура 800 градусов.

Полученную программу сделать параллельной с помощью технологии CUDA в местах использования метода Гаусса. Изучить влияние количества потоков на скорость выполнения программы.

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы – познакомиться с библиотекой CUDA и попробовать ее использование для прикладной программы.

Описание выполненных изменений

Для выполнения задания была использована уже написанная ранее (в 6 семестре) программа.

Чтобы использовать библиотеку CUDA, в программу был добавлен заголовочный файл *cuda_runtime.h*, вся программа была переписана на язык Си, а двумерные массивы развернуты для удобства в одномерные.

Параллельно были выполнены два одинаковых блока программы, связанные с вычитанием строк в методе Гаусса. Листинг 1 показывает этот участок.

Листинг 1. Параллельное преобразование строк

```
for (int j = threadIdx.x; j < size; j+=blockDim.x) {
    A[size * k + j] -= A[size * i + j] * coeff;
}
```

Кроме того, в основную функцию программы *count_state* были внесены изменения, связанные с выделением памяти на GPU, ее очищением и копированием с CPU на GPU и обратно. Листинг 2 показывает текущую функцию *count_state*. В комментариях, помеченных словом CUDA, указан типичный процесс работы с памятью для программ на CUDA, который был использован при написании программы.

Листинг 2. Функция *count_state*

```
// рассчитать состояние
void count_state(int k) {
   // --- CUDA 1. Выделение памяти хоста и заполнение ее ---

   // расчет начальных условий
   if (k == 0) {
       begin_state();
       return;
   }
```

```
    // создание исходных матриц СЛАУ
    int size = (I_MAX + 1) * (J_MAX + 1);
    int cmax = size - 1;    // максимальный номер строки в векторе c

    double* C = (double*)malloc(sizeof(double) * size);
    double* A = (double*)malloc(sizeof(double) * size * size);

    prepare_state(C, A, size, cmax, k);

    // --- CUDA 2. Выделение памяти девайса и заполнение ее ---
    double *dC, *dA;
    cudaMalloc((void**)(&dC), sizeof(double) * size);
    cudaMalloc((void**)(&dA), sizeof(double) * size * size);

    // --- CUDA 3. Перенос данных из хоста девайсу ---
    cudaMemcpy((void*)dC, (void*)C, sizeof(double) * size,
cudaMemcpyHostToDevice);
    cudaMemcpy((void*)dA, (void*)A, sizeof(double) * size * size,
cudaMemcpyHostToDevice);

    // --- CUDA 4. Вызов ядра для решения СЛАУ и синхронизация ---
    Gauss<<<1,32>>>(dA, size, dC);
    cudaDeviceSynchronize();

    // --- CUDA 5. Перенос данных из девайса хосту ---
    cudaMemcpy((void*)C, (void*)dC, sizeof(double) * size,
cudaMemcpyDeviceToHost);

    // перенос значений из вектора C в матрицу p
    int gi = 0;
    for (int i = 0; i < I_MAX + 1; i++) {
        for (int j = 0; j < J_MAX + 1; j++) {
            p[i][j][k] = C[gi];
            gi++;
        }
    }

    // --- CUDA 6. Очистка всех видов памяти ---
    cudaFree(dC);
    cudaFree(dA);

    free(C);
    free(A);
}
```

Замеры производительности

Были проведены измерения скорости работы программы в зависимости от размера задачи и количества потоков. Замеры производились на компьютере с видеокартой GeForce GTX 1060 3GB/PCIe/SSE2.

Размер задачи менялся следующим образом:

1. сетка 10×10, время 10с;
2. сетка 20×20, время 10с;

Количество тредов - 1, 32. Большее количество тредов приводит к ошибкам в расчетах. Скорее всего, это связано с характеристиками видеокарты или с неочевидной ошибкой в программе.

По результатам сделанных замеров была составлена таблица 1.

Таблица 1. Результаты тестов на производительность (в секундах).

	10×10×10	20×20×10
1	3.271	141.946
32	0.526	6.425

Как видно из таблицы 1, при разделении только этой небольшой части программы для выполнения на графической карте, удалось достичь существенного улучшения производительности.

Большое время работы при втором варианте задачи связано также с тем, что не был убран вывод результатов в консоль для проверки правильности выполнения.

Выводы

В работы были изучены основы работы с библиотекой CUDA. Она была применена для выполнения теплового расчета. Использование параллельных вычислений для метода Гаусса позволило сократить время работы приблизительно в 6 и более раз, что при увеличении размерности является существенным преимуществом.

Использованная литература

1. Лекционные материалы по дисциплине «Архитектура параллельных вычислительных систем».
2. https://www.dns-shop.ru/product/16cbd7e56dc93330/videokarta-asus-geforce-gtx-1060-dual-oc-dual-gtx1060-o3g/characteristics/
3. https://drmini.ru/computers/os/get-program-execution-time-in-the-shell.html
4. https://cuda-tutorial.readthedocs.io/en/latest/tutorials/tutorial01/
5. https://docs.nvidia.com/cuda/cuda-quick-start-guide/index.html