Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет
Петра Великого

Физико-механический институт

Высшая школа прикладной математики и вычислительной физики

Курсовая работа

по дисциплине
"Использование графических процессоров для вычислений"

Направление подготовки 01.04.02 «Прикладная математика и информатика»
Направленность (профиль) 01.04.02_01 Математическое моделирование и искусственный интеллект

Выполнил
студент гр. 5040102/20101 _____ Шао Ц.
подпись

Руководитель
Кандидат наук _____ Чуканов В. С.
подпись

Санкт-Петербург
2023 г.

Содержание

Список таблиц

Глава 1.

Вычисления общего назначения на графических процессорах

1.1. Введение

Вычисления общего назначения на графических процессорах (англ. *General-purpose computing on graphics processing units*, аббр ***GPGPU***) используют графический процессор, который обрабатывает графические задачи для расчета вычислительных задач общего назначения, первоначально обрабатываемых центральным процессором. Эти общие вычислительные задачи обычно не имеют ничего общего с обработкой графики. Поскольку современные графические процессоры обладают мощными возможностями параллельной обработки и программируемыми конвейерами, графические процессоры также могут обрабатывать неграфические данные. Вычисления общего назначения на графических процессорах значительно превосходят традиционные приложения с центральными процессорами по производительности, особенно когда они сталкиваются с одиночным потоком команд, множественным потоков данных(SIMD), когда объем операций обработки данных намного превышает потребность в планировании и передаче данных.

На данной работе с помощью этой технологии реализовано 2 алгоритма, первый пример из которых является более простым — сложением 2-х векторов, а второй является более сложным — перемножением 2-х матриц. Причем во втором алгоритме еще дополнительно реализовано с помощью разделяемой памятью, обращение к которой быстрее чем обращение к глобальной памяти. Сравнена производительность *CPU* и *GPU*.

1.2. Реализация

Программа выполняется с помощью графического процессора *NVIDIA* со следующими характеристиками:

- Имя устройства: NVIDIA GeForce GTX 1060 3GB
- Total global memory: 3071 MB
- Shared memory per block: 49152
- Registers per block: 65536
- Warp size: 32
- Memory pitch: 2147483647
- Max threads per block: 1024
- Max threads dimensions: $x = 1024, y = 1024, z = 64$
- Max grid size: $x = 2147483647, y = 65535, z = 65535$
- Clock rate: 1708500
- Total constant memory: 65536
- Compute capability: 6.1
- Texture alignment: 512
- Device overlap: 1
- Multiprocessor count: 9
- Kernel execution timeout enabled: true

1.3. Пример

1.3.1. Сложение 2-х векторов

Результат сложения 2-х векторов в *GPU*

Генерируем 2 вектора длиной 10 случайными числами и сложим их в *GPU*

$$v_1 = \big(18, 99, 52, 51, 66, 89, 62, 61, 99, 39\big)$$

$$v_2 = \big(49, 83, 52, 44, 77, 45, 36, 20, 63, 55\big)$$

После сложения получится $v_3 = v_1 + v_2$

$$v_3 = \big(67, 182, 104, 95, 143, 134, 98, 81, 162, 94\big)$$

С помощью *GPU* результат сложения векторов правилен.

Сравнение времени выполнения

Ниже выделена таблица, показывающая сравнение времени выполнения операции сложения 2-х векторов в *CPU* и *GPU*.

Таблица 1.1. Сравнение времени выполнения операции сложения 2-х векторов в *CPU* и *GPU*

Длина векторов N	Время на *CPU*	Время на *GPU*	Ускорение
1×10^7	23.16 ms	0.89 ms	26.02
2×10^7	46.78 ms	1.63 ms	28.70
3×10^7	67.61 ms	2.39 ms	28.29
4×10^7	89.44 ms	3.12 ms	28.67

По вышеперечисленным данным можно сказать, что во-первых как в *CPU* так и в *GPU* время выполнения программы удовлетворяет известной оценке $T(n) = O(n)$. Во-вторых в *GPU* программа значительно быстрее выполняется чем в *GPU*, причем ускорение около 26–28 раз.

1.3.2. Перемножение 2-х матриц

Результат перемножения 2-х матриц

Генерируем 2 матрицы случайными числами и перемножаем их в *GPU*

$$M_1 = \begin{pmatrix} 6 & 4 & 5 & 4 & 9 \\ 2 & 7 & 4 & 4 & 7 \\ 6 & 3 & 7 & 3 & 3 \\ 9 & 2 & 0 & 6 & 8 \end{pmatrix} \quad M_2 = \begin{pmatrix} 2 & 6 & 0 & 6 & 4 & 0 \\ 9 & 3 & 5 & 4 & 5 & 4 \\ 2 & 1 & 8 & 7 & 0 & 0 \\ 0 & 0 & 1 & 1 & 4 & 8 \\ 3 & 4 & 2 & 1 & 9 & 4 \end{pmatrix}$$

После перемножения получится $M_3 = M_1 \, M_2$

$$M_3 = \begin{pmatrix} 85 & 89 & 82 & 100 & 141 & 84 \\ 96 & 65 & 85 & 79 & 122 & 88 \\ 62 & 64 & 80 & 103 & 78 & 48 \\ 60 & 92 & 32 & 76 & 142 & 88 \end{pmatrix}$$

В *GPU* результат перемножения матриц правилен.

Сравнение времени выполнения

Сначала сравниваем время выполнения операции перемножения 2-х матриц в *CPU* и *GPU* без разделяемой памяти.

Таблица 1.2. Сравнение времени выполнения операции сложения 2-х векторов в *CPU* и *GPU* без разделяемой памяти

Размер матрицы 1	Размер матрицы 2	Время на *CPU*	Время на *GPU* без разделяемой памяти	Ускорение
(1000, 1500)	(1500, 2000)	17048.83 ms	27.44 ms	621.31
(1500, 2000)	(2000, 2500)	44231.43 ms	71.59 ms	617.84
(2000, 2500)	(2500, 3000)	89295.26 ms	140.09 ms	637.41
(2500, 3000)	(3000, 3500)	154817.64 ms	252.35 ms	613.50

Так же как в алгоритме сложения векторов в *CPU* и в *GPU* время выполнения программы удовлетворяет известной оценке $T(n) = O(n^3)$. В алгоритме перемножения матриц не как в алгоритме сложения векторов, ускорение *GPU* относительно *CPU* более велико — около 610 – 640 раз.

Теперь сравниваем время выполнения операции перемножения 2-х матриц в *GPU* без разделяемой памяти и с разделяемой памятью.

Таблица 1.3. Сравнение времени выполнения операции сложения 2-х векторов в *GPU* без разделяемой памяти и с разделяемой памятью

Размер матрицы 1	Размер матрицы 2	Время на *GPU* без разделяемой памяти	Время на *GPU* с разделяемой памятью	Ускорение
(1000, 1500)	(1500, 2000)	27.44 ms	14.59 ms	1.88
(1500, 2000)	(2000, 2500)	71.59 ms	36.06 ms	1.98
(2000, 2500)	(2500, 3000)	140.09 ms	72.34 ms	1.93
(2500, 3000)	(3000, 3500)	252.35 ms	128.16 ms	1.97

1.4. Заключение

В целом *GPU* программа значительно быстрее выполняется чем в *CPU*, причем чем сложнее алгоритм, тем больше ускорение. Например в сложении векторов ускорение на 26–28 раз, а в перемножении матриц ускорение больше 600 раз. Во-вторых с использованием разделяемой памяти, обращение к которой быстрее чем к глобальной памяти, можно еще ускорить программа, причем ускорение около на 1.9 – 2 раза. Поэтому можно будет пользоваться данной технологией в других сферах, например в компьютерной графике, обработке и фильтрации изображений и так далее.