Лабораторная работа № 6

Программирование графических процессоров (NVIDIA CUDA)

Цель: Ознакомиться с технологией NVIDIA CUDA. Научиться компилировать и запускать программы, содержащие CUDA- код.

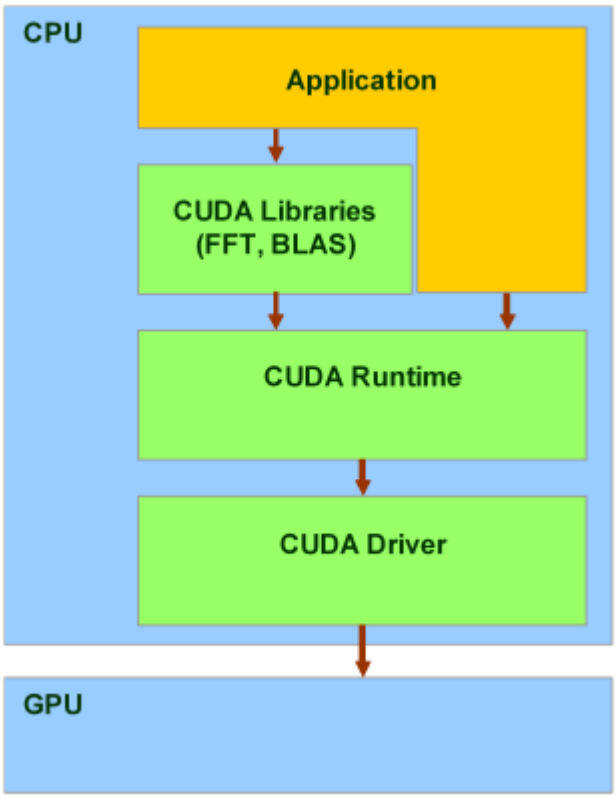
Теоретические сведения

CUDA (англ. Compute Unified Device Architecture) — технология GPGPU (англ. General-Purpose computing on Graphics Processing Units – вычисления общего назначения на графических процессорах), позволяющая программистам реализовывать на упрощённом языке программирования Си алгоритмы, выполнимые на графических процессорах видеокарт GeForce восьмого поколения и старше. Технология CUDA разработана компанией Nvidia. Фактически CUDA позволяет включать в текст Си программы специальные функции. Эти функции пишутся на упрощённом языке программирования Си и выполняются на графических процессорах Nvidia.

Чтобы понять, какие преимущества приносит перенос расчётов на видеокарты, приведём усреднённые цифры, полученные исследователями по всему миру. В среднем, при переносе вычислений на GPU, во многих задачах достигается ускорение в 5-30 раз, по сравнению с быстрыми универсальными процессорами.

Состав NVIDIA CUDA

CUDA включает два API: высокого уровня (CUDA Runtime API) и низкого (CUDA Driver API), хотя в одной программе одновременное использование обоих невозможно, нужно использовать или один или другой. Высокоуровневый работает «сверху» низкоуровневого, все вызовы runtime транслируются в простые инструкции, обрабатываемые низкоуровневым Driver API. Но даже «высокоуровневый» API предполагает знание основ об устройстве и работе видеокарт NVIDIA.



Есть и ещё один уровень, даже более высокий — две библиотеки:

CUBLAS — CUDA вариант BLAS (Basic Linear Algebra Subprograms), предназначенный для вычислений задач линейной алгебры и использующий прямой доступ к ресурсам GPU;

CUFFT — CUDA вариант библиотеки Fast Fourier Transform для расчёта быстрого преобразования Фурье, широко используемого при обработке сигналов. Поддерживаются следующие типы преобразований: complex-complex (C2C), real-complex (R2C) и complex-real (C2R).

Написание программ на CUDA

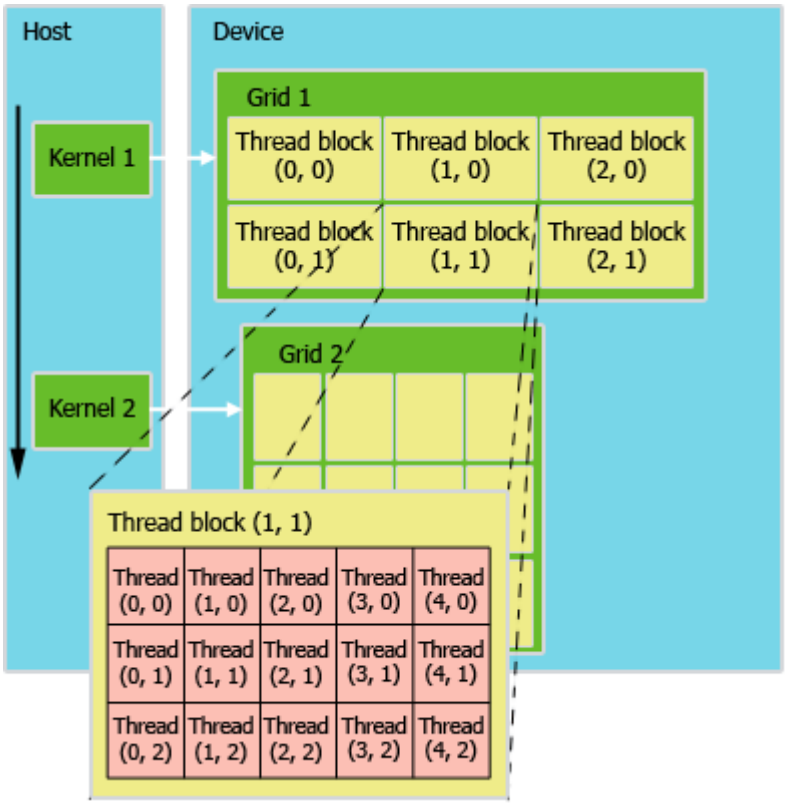
Для разработки собственных программ следует разбираться в базовых архитектурных особенностях видеочипов NVIDIA. Все инструкции в видеокарте выполняются по принципу SIMD, когда одна инструкция применяется ко всем потокам в warp (в CUDA это группа из 32 потоков — минимальный объём данных, обрабатываемых мультипроцессорами). Этот способ выполнения назвали SIMT (single instruction multiple threads — одна инструкция и много потоков).

При исполнении программы, центральный процессор выполняет свои порции кода, а GPU выполняет CUDA код с наиболее тяжелыми параллельными вычислениями. Эта часть, предназначенная для GPU,

называется ядром (kernel). В ядре определяются операции, которые будут исполнены над данными.

Таким образом, CUDA использует параллельную модель вычислений, когда каждый из SIMD процессоров выполняет ту же инструкцию над разными элементами данных параллельно. GPU является вычислительным устройством(device) для центрального процессора(host), обладающим собственной памятью и обрабатывающим параллельно большое количество потоков. Ядром (kernel) называется функция для GPU, исполняемая потоками. Видеочип отличается от CPU тем, что может обрабатывать одновременно сотни тысяч потоков, что обычно для графики, которая хорошо распараллеливается.

Модель программирования в CUDA предполагает группирование потоков. Потоки объединяются в блоки потоков (thread block) — одномерные или двумерные сетки потоков, взаимодействующих между собой при помощи разделяемой памяти и точек синхронизации. Программа (ядро, kernel) исполняется над сеткой (grid) блоков потоков (thread blocks), см. рисунок ниже. Одновременно исполняется одна сетка. Каждый блок может быть одно-, двух- или трехмерным по форме, и может состоять из 512 потоков на текущем аппаратном обеспечении.



Пример программы на CUDA

В качестве примера рассмотрим задачу умножения квадратной матрицы на вектор.

Исходные данные:

`A[n][n]` – матрица размерности `n x n`;

`b[n]` – вектор, состоящий из `n` элементов.

Результат:

`c[n]` – вектор из `n` элементов.

```
//Последовательный алгоритм умножения матрицы на вектор
for (i=0; i<n; i++)
{
          c[i]=0;
          for (j=0; j<m; j++)
          {
               c[i]=A[i][j]*b[j];
          }
}
```

Теперь рассмотрим решение этой задачи на видеокарте. Следующий код иллюстрирует пример вызова функции CUDA:

```
// инициализация CUDA
if(!InitCUDA())   { return 0; }

int Size = 1000;
// обычные массивы в оперативной памяти
float *h_a,*h_b,*h_c;
h_a = new float[Size*Size];
h_b = new float[Size];
h_c = new float[Size];

for (int i=0;i<Size;i++)  // инициализация массивов a и b
{
  for (int k=0;k<Size;k++)
     {
       h_a[i*Size+k]=1;
     }
  h_b[i]=2;
}

// указатели на массивы в видеопамяти
float *d_a,*d_b,*d_c;

// выделение видеопамяти
```

```
cudaMalloc((void **)&d_a, sizeof(float)*Size*Size);
cudaMalloc((void **)&d_b, sizeof(float)*Size);
cudaMalloc((void **)&d_c, sizeof(float)*Size);

// копирование из оперативной памяти в видеопамять
CUDA_SAFE_CALL(cudaMemcpy(d_a, h_a, sizeof(float)*Size*Size,
                                cudaMemcpyHostToDevice) );
CUDA_SAFE_CALL(cudaMemcpy(d_b, h_b, sizeof(float)*Size,
                                cudaMemcpyHostToDevice) );

// установка количества блоков
dim3 grid((Size+255)/256, 1, 1);
// установка количества потоков в блоке
dim3 threads(256, 1, 1);

// вызов функции
MatrVectMul<<< grid, threads >>> (d_c, d_a, d_b,Size);

// копирование из видеопамяти в оперативную память
CUDA_SAFE_CALL(cudaMemcpy(h_c, d_c, sizeof(float)*Size,
                                cudaMemcpyDeviceToHost) );

// освобождение памяти
CUDA_SAFE_CALL(cudaFree(d_a));
CUDA_SAFE_CALL(cudaFree(d_b));
CUDA_SAFE_CALL(cudaFree(d_c));
```

В принципе структуру любой программы с использованием CUDA можно представить аналогично рассмотренному выше примеру. Таким образом, можно предложить следующую последовательность действий:

1) инициализация CUDA
2) выделение видеопамяти для хранения данных программы
3) копирование необходимых для работы функции данных из оперативной памяти в видеопамять
4) вызов функции CUDA
5) копирование возвращаемых данных из видеопамяти в оперативную
6) освобождение видеопамяти

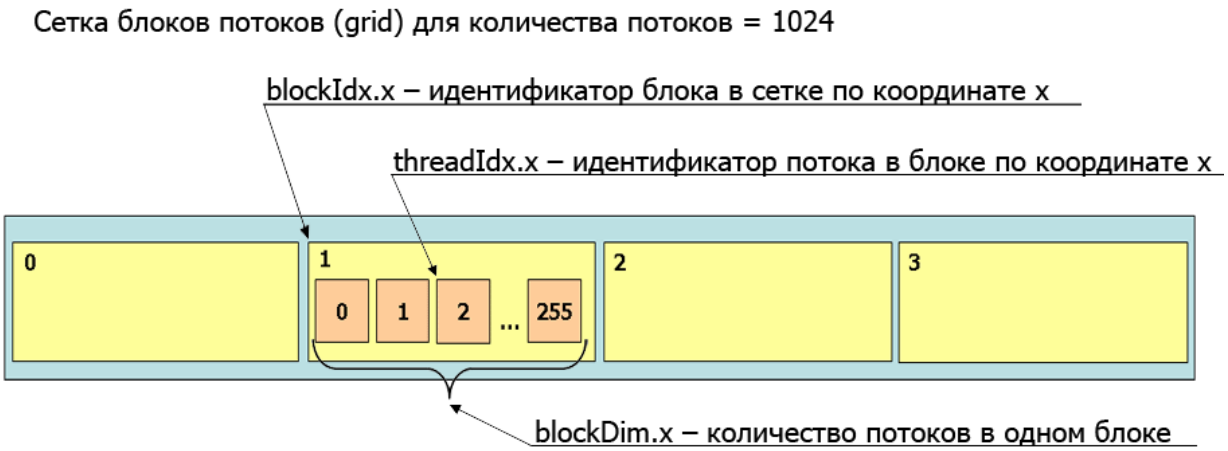
Пример функции, исполнимой на видеокарте

```
extern "C" __global__ void MatrVectMul(float *d_c, float *d_a,
float *d_b, int Size)

    {

            int i = blockIdx.x*blockDim.x+threadIdx.x;
            int k;
```

```
            d_c[i]=0;
            for (k=0;k<Size;k++)
              {
                 d_c[i]+=d_a[i*Size+k]*d_b[k];
              }
}
```

Здесь: `threadIdx.x` – идентификатор потока в блоке по координате x, `blockIdx.x` – идентификатор блока в гриде по координате x, `blockDim.x` – количество потоков в одном блоке.

Пока же следует запомнить, что таким образом получается уникальный идентификатор потока (в данном случае `i`), который можно использовать в программе, работая со всеми потоками как с одномерным массивом.



Одномерная сетка блоков потоков, количество блоков = 4;
и одномерная сетка потоков в блоке, количество потоков в блоке = 256.
Уникальный идентификатор потока (i) = blockIdx.x*blockDim.x+threadIdx.x,
например, i=1*256+2=258

Важно помнить, что функция, предназначенная для исполнения на видеокарте, не должна обращать к оперативной памяти. Подобное обращение приведет к ошибке. Если необходимо работать с каким-либо объектом в оперативной памяти, предварительно его надо скопировать в видеопамять, и обращаться из функции CUDA к этой копии.

Среди основных особенностей CUDA следует отметить отсутствие поддержки двойной точности (типа `double`). Также для функций CUDA установлено максимальное время исполнения, отсутствует рекурсия, нельзя объявить функцию с переменным числом аргументов.

Следует отметить следующее, что функция, работающая на видеокарте, должна выполняться не более 1 секунды. Иначе, функция будет не завершена, и программа завершится с ошибкой.

Для синхронизации потоков в блоке существует функция `__syncthreads()`, которая ждет, пока все запущенные потоки отработают до этой точки. Функция `__syncthreads()` необходима, когда данные, обрабатываемые одним потоком, затем используются другими потоками.

Замер времени в CUDA

Для измерения времени выполнения отдельных частей программы и анализа скорости выполнения того или иного участка кода удобно использовать встроенные в CUDA функции по замеру времени.

```
unsigned int timer;

// создание таймера
cutCreateTimer(&timer);

// запуск таймера
cutStartTimer(timer);

… // код, время исполнения которого необходимо замерить

// остановка таймера
cutStopTimer( timer);

// получение времени
printf("time: %f (ms)\n", cutGetTimerValue(timer));

// удаление таймера
cutDeleteTimer( timer);
```

Задания

Необходимо написать программу согласно варианту, при этом реализовать 2 функции: одну для выполнения на процессоре, вторую для выполнения на видеокарте. Затем сравнить результаты (возвращаемые значения) и скорость работы.

Задание к выполнению лабораторной работы

Задание №1.

Разработайте программу, для вычисления значения функции f(x) на отрезке [1,N+1] с

шагом h=N/ k, где N – номер варианта и составьте таблицу:

k	время расчета на CPU	время расчета на GPU
N		
$N*10^2$		
$N*10^4$		
$N*10^6$		
...		

a = N, b = N*2, k=N/2, z = N^2

1. $y = \sqrt{x-1} + \frac{1}{x-3};$
2. $y = \frac{1}{k * \sqrt{2\pi}} * e^{\frac{-(x-a)^2}{2k^2}};$
3. $y = \frac{1}{2b} * e^{\frac{-|x-a|}{b}};$
4. $y = \frac{\sin x}{2\cos^2 x} - \cos x - \frac{3}{2} tgx;$
5. $y = \cos x(\ln\left|2 - e^{-|a+x|}\right|);$
6. $y = \frac{e^{\sin^2 x} + \ln|tgx|}{\sin x};$
7. $y = a * e^{-ax} * \sin x;$
8. $y = \frac{\sin^3\left|ax^3 + bx^2 - ab\right|}{\sqrt{\left|ax^3 + bx^2 - ab\right|}};$
9. $y = \frac{\sqrt{|\cos x|}}{\sqrt{1+x^2}};$
10. $y = \frac{1 + \sin\sqrt{x+1}}{\cos(12z-4)};$
11. $y = x - \frac{x^3}{3} + \frac{x^5}{5};$

12. $y = \dfrac{x^2 - 7x + 10}{x^2 - 8x + 12};$

13. $y = \dfrac{\cos x}{a - 2x} + 16x \cdot \cos(xz) - 2;$

14. $y = \left|x^2 - x^3\right| - \dfrac{7x}{x^3 - 15x};$

15. $y = e^{-x} - \cos x + \sin(2xz);$

16. $y = e^{tgx} + \cos(x - z);$

17. $y = \sin\sqrt{x+1} - \sin\sqrt{x-1};$

18. $y = e^x - \dfrac{z^2 + 12xz - 3x^2}{18z - 1};$

19. $y = x - 10\sin x + \left|x^2 - xe^5\right|;$

20. $y = x - 10e^{\sin x} + \cos(x - z);$

21. $y = 2ctg(3x) - \dfrac{1}{12x^2 + 7x - 5};$

22. $y = e^x - x - 2 + (1 + x)^2;$

23. $y = 2ctg(3x) - \dfrac{\sin\cos x}{\sin(1 + x^2)};$

24. $y = e^x - 4x + (z - \sqrt{|x|});$

25. $y = \left(\dfrac{z+1}{x-1}\right)^2 + 18xz^2;$

26. $y = \left(1 + \dfrac{1}{x^2}\right)^2 - 12x^2 z:$

Задание №2.

Варианты заданий

1. Разработайте программу для нахождения минимального значения среди элементов вектора.
2. Разработайте программу для нахождения максимального значения среди элементов вектора.
3. Разработайте программу для нахождения суммы всех элементов вектора.
4. Разработайте программу для нахождения произведения всех элементов вектора.
5. Разработайте программу для вычисления скалярного произведения двух векторов.

6. Разработайте программу решения задачи поиска максимального значения среди минимальных элементов строк матрицы.
7. Разработайте программу решения задачи поиска минимального значения среди максимальных элементов строк матрицы.
8. Разработайте программу для решения задачи транспонирования матрицы.
9. Разработайте программу для нахождений произведения вектора на матрицу.
10. Разработайте программу для нахождения минимального значения среди элементов матрицы.
11. Выполнить сложение двух матриц одинакового размера.
12. Найти сумму максимальных элементов строк матрицы.
13. Найти площадь выпуклого многоугольника, заданного координатами вершин.
14. Найти в данном тексте все палиндромы.
15. Найти в тексте все вхождения данного образца.
16. Дана последовательность вещественных чисел. Сократить количество десятичных разрядов после запятой каждого числа до двух.
17. Дана последовательность арифметических выражений, операндами которых являются однозначные числа, а число операций не больше двух. Найти значения всех выражений.
18. Дана матрица вещественных чисел. Преобразовать матрицу таким образом, чтобы элементы ее строк шли по убыванию.
19. Вывести все согласные, которые отсутствуют в данном тексте.
20. Дана матрица вещественных чисел. Преобразовать матрицу таким образом, чтобы элементы ее столбцов шли по убыванию.

Содержание отчета

1. Название и цель работы.
2. Задание к выполнению лабораторной работы согласно варианту.
3. Описание алгоритма решения задачи в виде блок-схем или словесное. Описание используемых параллельных методов вычислений.
4. Программа в виде исходных кодов (с поясняющими комментариями), а также в откомпилированном виде для демонстрации на ЭВМ.
5. Примеры работы программы на тестовых данных.

Выводы по работе.

Контрольные вопросы

1. Что подразумевается под терминами *kernel*, *host* и *device*?
2. Объясните понятия *grid*, *thread block*, *thread*.
3. На каких видеокартах доступна технология CUDA?
4. Как вы считаете, будут ли перечисленные выше задания выполняться быстрее для *Size=5* на видеокарте, чем на процессоре и почему?
5. Что общего в технологиях OpenMP и Nvidia CUDA и чем они различаются (с точки зрения программирования)?