Досліджені архітектури ядер CPU та GPU. Реалізовано алгоритм SAXPY на GPU. Отримані залежності часу виконання алгоритму на CPU і GPU від розмірів векторів. Зроблено висновок про доцільність використання GPU для SIMD–обчислень

Ключові слова: CPU, GPU, SAXPY, ядро, архітектура, нитка

Исследованы архитектуры ядер CPU и GPU. Реализован алгоритм SAXPY на GPU. Получены зависимости времени выполнения алгоритма на CPU и GPU от размеров векторов. Сделан вывод о целесообразности применения GPU для SIMD–вычислений

Ключевые слова: CPU, GPU, SAXPY, ядро, архитектура, блок, нить

Research of architecture of kernel CPU and GPU is executed. Algorithm SAXPY on GPU is realised. Dependences of the performance time of algorithm on CPU and GPU from the sizes of vectors are received. The conclusion is drawn on expediency of application GPU for SIMD–calculations

Key words: CPU, GPU, SAXPY, core, architecture, block, thread

УДК 004.27

SIMD-РЕАЛИЗАЦИЯ АЛГОРИТМА SAXPY НА GPU

С.Ю. Скрупский
Студент*
Контактный тел.: 8 (061) 267-25-45
E-mail: 88sts88@mail.ru

Р.К. Кудерметов
Кандидат технических наук, доцент, заведующий кафедрой
Кафедра компьютерных систем и сетей*
Контактный тел.: 8 (061) 220-28-90
E-mail: krk@zntu.edu.ua
*Национальный технический университет

1. Введение

Необходимость решения сложных задач с огромными объемами вычислений привели к появлению многопроцессорных вычислительных систем, что позволило значительно увеличить производительность в решении таких задач, однако стоимость каждого ядра или узла такой системы по-прежнему остается высокой. Анализ вычислительной мощности современных графических процессоров, которые появлялись в последние годы, показывает, что на текущий момент времени она в несколько раз выше мощности многоядерных процессоров и при этом – дешевле. Разница между CPU (central processor unit) и GPU (graphic processor unit) заключается в принципиальном различии их архитектур [1] (рис. 1).

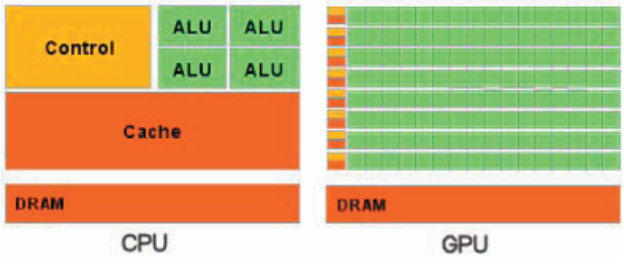

Рис. 1 Архитектуры CPU - слева и GPU - справа

Ядра CPU созданы для исполнения одного потока последовательных инструкций с максимальной производительностью, а GPU проектируются для быстрого исполнения большого числа параллельно выполняемых потоков инструкций. Разработчики CPU стараются добиться выполнения как можно большего числа инструкций параллельно, для увеличения производительности. Для этого, начиная с процессоров Intel Pentium, появилось суперскалярное выполнение, обеспечивающее выполнение двух инструкций за такт, а Pentium Pro “отличился” внеочередным выполнением инструкций. Но у параллельного выполнения последовательного потока инструкций есть определённые базовые ограничения: с увеличением количества исполнительных блоков кратного увеличения скорости не добиться. У видеочипов работа простая и распараллеленная изначально. GPU принимает группу полигонов, проводит все необходимые операции, возвращает пиксели. Обработка полигонов и пикселей независима, их можно обрабатывать параллельно, отдельно друг от друга. Поэтому, из-за изначально параллельной организации работы в GPU используется большое количество исполнительных блоков, которые легко загрузить, в отличие от последовательного потока инструкций для CPU. Кроме того, современные GPU также могут исполнять больше одной инструкции за такт (dual issue). Так, архитектура Tesla в некоторых условиях запускает на исполнение операции MAD+MUL или MAD+SFU одновременно. Не все центральные

процессоры имеют встроенные контроллеры памяти, а у всех GPU обычно есть по несколько контроллеров, вплоть до восьми 64-битных каналов в чипе NVIDIA GT-200. На видеокартах применяется более быстрая память, в результате видеочипам доступна в несколько раз большая пропускная способность памяти, что весьма важно для параллельных расчётов, оперирующих с огромными потоками данных. CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеочипы могут поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько штук. И если переключение с одного потока на другой для CPU стоит сотни тактов, то GPU переключает несколько потоков за один такт.

Таким образом, основой эффективного использования мощи GPU для неграфических расчётов является распараллеливание алгоритмов на множество исполнительных блоков, имеющихся на видеочипах.

2. Постановка задачи

Рассмотрим векторный алгоритм SAXPY(scalar a x+y), смысл которого заключается в вычислении скалярного произведения константы на один вектор и прибавление другого вектора, и который очень часто используется в линейной алгебре, в тестах высокопроизводительных вычислительных систем. Для экспериментальной оценки эффективности GPU в решении вычислительных задач разработаем SIMD–реализацию алгоритма, адаптированного под архитектуру ядра графического процессора. Проведем испытание этого алгоритма и измерим время его выполнения на CPU, на GPU и на GPU с учетом пересылок данных из ОЗУ в видеопамять и обратно.

3. Реализация векторного алгоритма SAXPY на GPU

Верхний уровень ядра GPU состоит из блоков, которые группируются в сетку размерностью N1 * N2 (рис. 2).

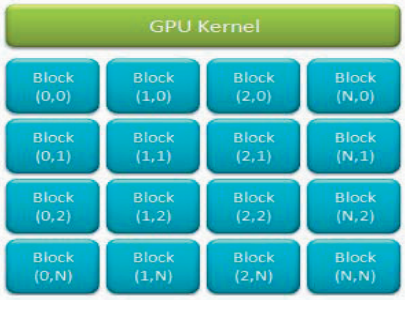

Рис. 2. Верхний уровень ядра GPU

Каждый кластер состоит из укрупнённого блока текстурных выборок и двух-трех потоковых мультипроцессоров, состоящих из восьми вычислительных устройств и двух суперфункциональных блоков. Все инструкции выполняются по принципу SIMD, когда одна инструкция применяется ко всем потокам.

Каждый блок состоит из нитей (легковесных потоков), которые являются непосредственными исполнителями вычислений. Нити в блоке сформированы в виде трехмерного массива (рис. 3).

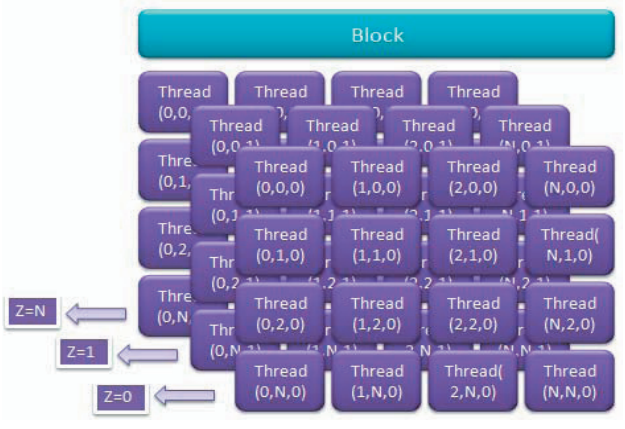

Рис. 3. Организация каждого блока GPU

Аппаратный менеджер потоков обрабатывает их автоматически. Автоматическое управление потоками важно, когда многопоточность масштабируется на тысячи выполняемых потоков. Каждый поток имеет свой собственный стек, файл регистра, программный счетчик и свою память. Связь GPU с памятью представлена на рис. 4.

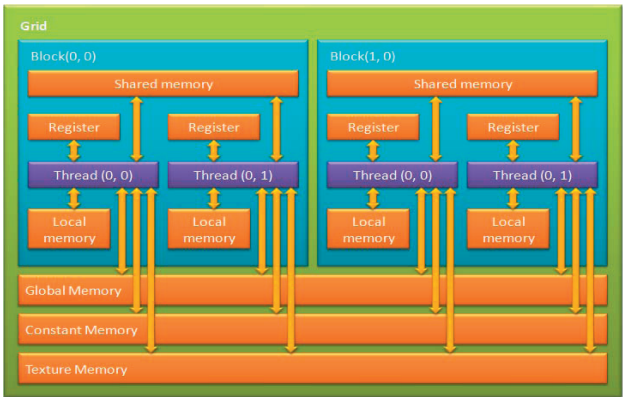

Рис. 4. Организация памяти GPU

Таким образом, в распоряжении разработчика имеется вычислительная система, оснащенная 128 или более (в зависимости от конкретной модели) 32-разрядными арифметико-логическими устройствами.

Программный код реализации SAXPY на GPU и на CPU написан на расширении языка C++, предоставленным компанией NVIDIA [2]. Блок-схема SIMD–реализации алгоритма SAXPY на GPU приведена на рис. 5.

Функция

```
__global__void addVector(int N,float k,float* dev_vec1,
float* dev_vec2)
{ int idx = blockIdx.x * blockDim.x + threadIdx.x;
if(idx<n) dev_vec2 [idx] = k* dev_vec1 [idx] + dev_vec2 [idx];}
```

вычисляет SAXPY на GPU параллельно. Здесь используются такие переменные:

• blockIdx.x - индекс текущего блока в вычислении на GPU, имеет тип uint3.

• blockDim.x - размерность блока, имеет тип dim3. Позволяет узнать размер блока, выделенного при текущем вызове ядра.

• threadIdx.x - индекс текущей нити в вычислении на GPU, имеет тип uint3.

Особое внимание следует уделить вызову функции ядра GPU в основной программе (рис. 6).

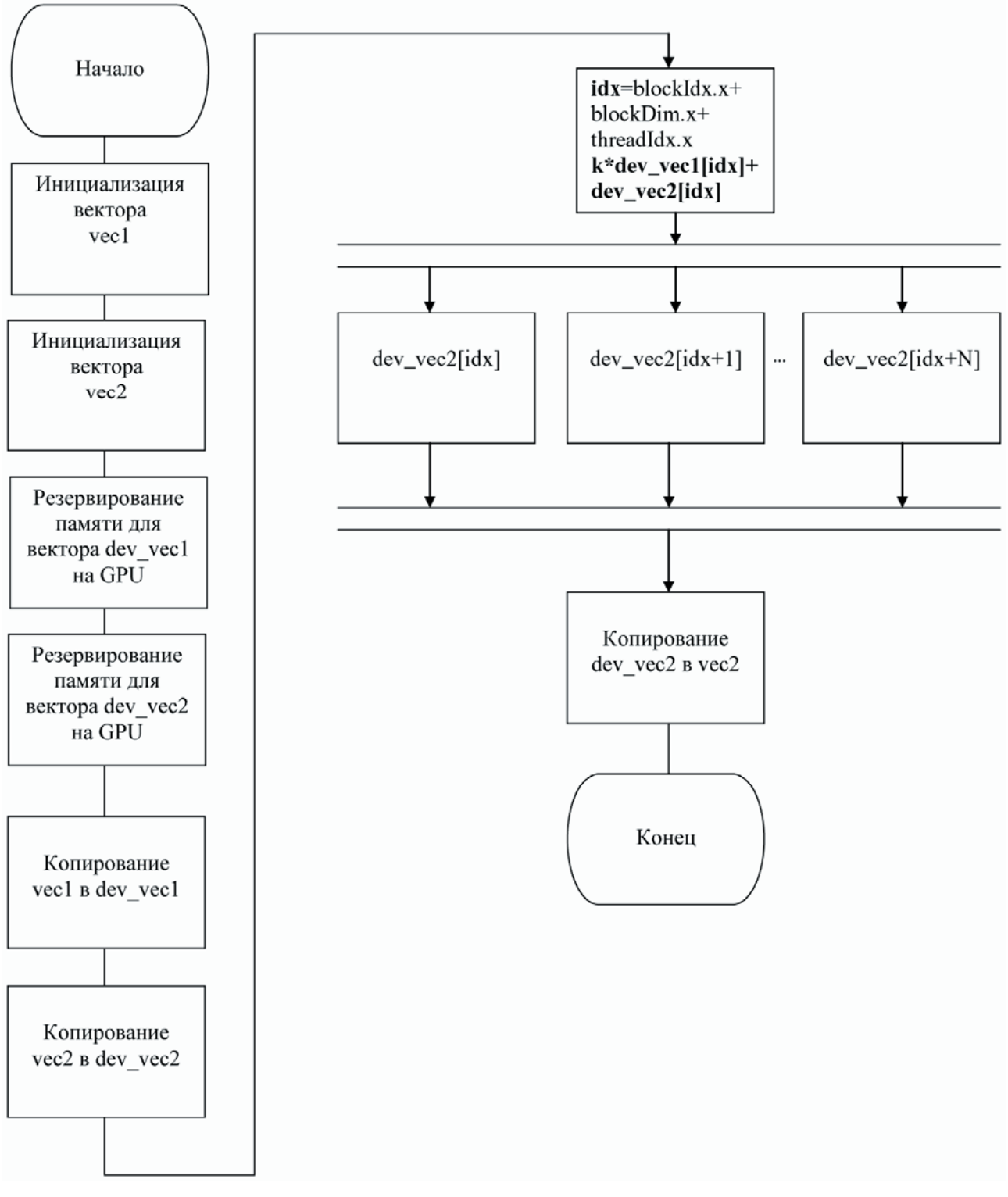

Рис. 5. Блок-схема SIMD–реализации алгоритма SAXPY на GPU

Вызов функции ядра GPU в основной программе:

```
dim3 threads = dim3(512, 1); //число нитей на блок
dim3 blocks = dim3(N / threads.x, 1); //число блоков в сетке
addVector<<<blocks,threads>>>(N,k,dev_vec1,dev_vec2); //вызов GPU
```

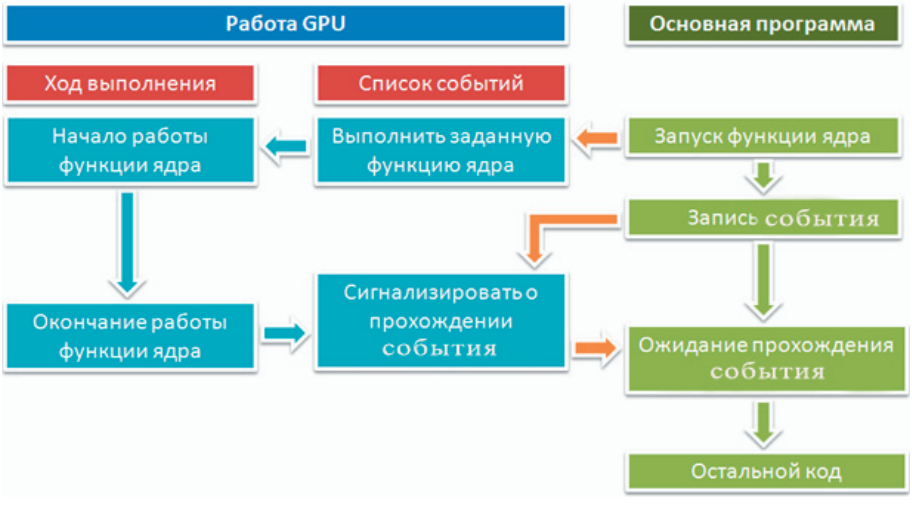

Рис. 6. Запуск функции ядра GPU

4. Результаты экспериментов

Замеры времени выполняются по событиям (events). Время выполнения функции ядра GPU – это разница между временем записи события до вызова функции ядра GPU и после ее выполнения. В процессе тестирования оценивалось время выполнения алгоритма на CPU, на GPU и на GPU с учетом времени пересылок данных из ОЗУ в видеопамять и обратно. Тестирование выполнялось на компьютере следующей конфигурации:

- Системная плата: ASUS P5B DELUX на Intel P965;
- CPU: Intel Core 2 DUO E6420, 2.13GHz;
- GPU: Nvidia 9800 GTX+, 765MHz;
- ОЗУ: DDR2 800MHz.

Результаты тестирования представлены на рис. 7.

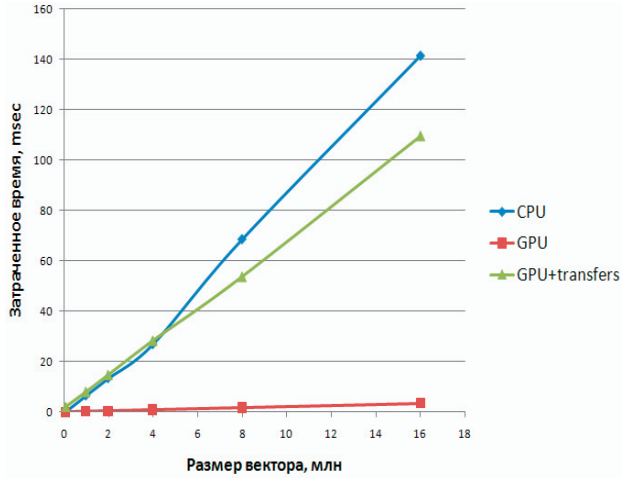

Рис 7. Результаты тестирования алгоритма на CPU и на GPU

Как видно из рисунка 7, время, затрачиваемое GPU на выполнение алгоритма SAXPY, значительно меньше времени, необходимого CPU для реализации того же алгоритма. Слабым местом системы с векторными вычислениями средствами GPU являются пересылки данных в цепочке ОЗУ → Северный мост → GPU → Северный мост → ОЗУ.

Время, затрачиваемое на такие пересылки, зависит от конкретной аппаратуры и тактовых частот проводников. Количество таких пересылок необходимо минимизировать.

Выводы

Таким образом, показана эффективность применения GPU для SIMD – реализации алгоритма SAXPY. Графический процессор может быть использован в качестве арифметического сопроцессора CPU для выполнения векторных вычислений.

В исследованиях применялась одноядерная видеокарта, характерная для обычных рабочих станций. Более трудоемкие задачи могут быть решены, например на четырех двухъядерных видеокартах по технологии 4-way SLI [2], которые позволят распараллелить задачу на 8 независимых потоков.

Литература

1. Берилло А. NVIDIA CUDA — неграфические вычисления на графических процессорах / А. Берилло. //ixbt.com. – 2008. - №4. –с. 18-25.
2. NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. Ver 2.1. / NVIDIA Corporation. – [S.I.] : NVIDIA, 2008. – 111 p.