ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКОГО УСКОРИТЕЛЯ ДЛЯ РЕШЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ

Л.Ф. Зимянин, С.И. Речицкий
Беларусь, Минск

Введение. Развитие индустрии компьютерных игр вывело на рынок персональных компьютеров специальные графические ускорители *GPU* (*Graphical Processing Unit*), предназначенные для быстрого рисования изображений на экране монитора. Для достижения анимации в реальном времени GPU должны выполнять огромное количество операций с плавающей точкой. В связи с тем, что *GPU* по своей архитектуре близки к *SIMD* моделям программирования и позволяют выполнять пользовательские программы, они могут быть применены для решения вычислительных задач и, совместно с CPU, существенно повысить производительность компьютера.

Работа посвящена проблемам интеграции *GPU* и *CPU* при решении матричных задач большого объема.

1. Схема вычислений на GPU. Архитектура *GPU* представлена на рисунке. Основными данными в *GPU* являются треугольники, каждой вершине которого назначаются некоторые свойства: позиция в пространстве, текстурные координаты. Эти данные проходят через блок обработки вершин (*БОВ*), основная задача которых вычисление экранных координат треугольника и параметров освещения каждой вершины в соответствии с заданной программой.

Блоки обработки пикселей (*БОП*) модели выполняют заполнение внутренней области треугольников модели. Именно эти блоки представляют наибольший интерес при решении поставленной задачи, поскольку по совокупности блоки обработки пикселей превосходят по мощности в несколько раз блоки обработки вершин.

Для организации вычислений на *GPU* предлагается следующая схема:

1. Закодировать входных данных в виде текстуры;

2. Разработать программу для *БОП*, реализующую процесс обработки текстуры:

3. Результаты вычислений из видеопамяти прочитать в системную память. Следует отметить, что этот процесс является достаточно медленным из-за особенностей шины данных между видео и системной памятью.

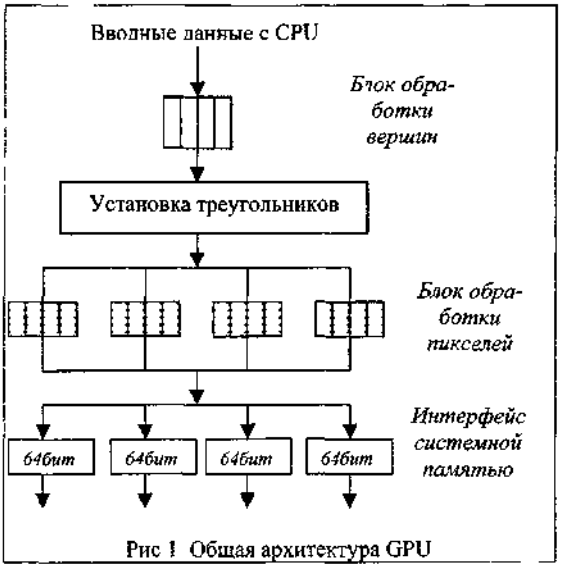

Рис 1 Общая архитектура GPU

В рассматриваемом классе задач основными объектами являются вектора и матрицы (общего вида или специального, например, ленточные, разреженные и т.д.). Эти объекты будем представлять в виде текстуры, каждый элемент которой представляется четырехмерным вектором чисел с плавающей точкой.

Матрица размером M x N кодируется с помощью двумерной текстуры размерности $4\lfloor \frac{M}{4} \rfloor \times N$. Каждый элемент текстуры содержит *4* элемента исходной матрицы.

Вектор размерности N кодируется в виде двумерной текстуры размерности m x m, где $m = \sqrt{\lceil \frac{N}{4} \rceil}$ следующим образом: компоненте x_n ставиться в соответствие элемент матрицы с индексами $(i, j) = (\lfloor k/m \rfloor, k \bmod m)$, где $k=\lfloor N/4 \rfloor$–номер тетрады, содержащей элемент x_n.

Следует отметить ряд ограничений схемы вычислений на GPU.

1. Максимальный размер матриц или векторов ограничен размером *Video RAM*. Поэтому реально мы можем работать с потоками данных ограниченного размера (например, не более 2048 элементов);

2. Узким местом является также медленный интерфейс с системной памятью (для графических операций этот интерфейс не имеет большого значения). Поэтому в схеме вычислений не рекомендуются использовать частые пересылки данных в системную память

2. Потоковая модель вычислений. Это модель вычислений, которая реализуется в языке *BrookGPU* /1/. Язык *BrookGPU* является расширением стандартного *ANSI C* и предназначен для облегчения запуска приложений на GPU. Основным элементом этой модели является поток. Потоком назовём совокупность одинаковым образом обрабатываемых элементов. В некотором смысле поток является эквивалентом массива, но основным его отличием является именно то, что все элементы потока обрабатываются одинаковым образом. Потоки могут быть как одномерными, так и многомерными. Над потоком можно выполнять две основные операции – это *ядро* и *редукция*. Ядро не меняет размерность потока и применяется для вычисления выходных значений, на основании одного или нескольких входных потоков данных. Редукция применяется для вычисления некоторой характеристики потока, к примеру, суммы или скалярного произведения всех его элементов.

3. Результаты экспериментов.

В работе приведен сравнительный анализ выполнения матричных и векторных операций на *CPU* и *GPU*. В качестве теста был использован пакет *BLAS (Basic Linear Algebra Subprograms)*. Тестирование было проведено на системе с процессором *Athlon XP 2500+* с *512 MB DDR* памяти и установленной видеокартой на базе процессора *GeForceFX 5900 XT*. Результаты тестирования приведены в таблице.

Таблица. Результаты экспериментов

Тестовая задача	***Производительность (Мфлопс/с)***		
	CPU	***GPU***	
		DX-9	***Open GL***
SDOT	4.161	227.951	103.819
SAXPY	4.925	699.051	213.995
SGEMV	5.347	374.857	177.898

В качестве тестовых задач были выбраны следующие алгоритмы:

SDOT – итерационное выполнение операций $y=y * x$;

SAXPY – итерационное выполнение операций $y=y+\alpha x$;

SGEMV – итерационное выполнение операций $y=+\alpha Ax$ и $y=\beta y+\alpha A^T x$.

В алгоритмах α, β – константы, x и y –вектора размерности *1024*, A– матрица *1024x1024*.

Подготовка задач для выполнения на *GPU* выполнено с помощью компилятора *BrookGPU*.

Заключение.

Эксперименты показывают, что GPU может быть эффективно использован для решения численных задач, причем для некоторых классов задач наблюдается существенное повышение производительности.

Литература

1 I Buck, T Foley, D Horn, J Sugerman, K Fatahalian, M Houston, and P Hanrahan Brook for GPUs Stream Computing on Graphics Hardware ACM Transactions on Graphics, 23(3) 777–786, 2004

2 Jens Kruger Linear Algebra on GPUs. Technische Universitat Munchen