УДК 519.688

А. Е. Зименко[1], А. А. Дячук[2]

[1]ЗАТ "Преттль-Кабель Украина", г. Каменец-Подольский
[2]Институт проблем моделирования в энергетике им. Г. Е. Пухова НАН Украины, г. Киев

ПРОГРАММНАЯ МОДЕЛЬ МНОГОПОТОЧНОГО ПРОЦЕССОРА НА БАЗЕ ГРАФИЧЕСКИХ УСКОРИТЕЛЕЙ

Произведен обзор и сравнение систем программирования общего программирования для графических процессоров (GPGPU), выделены особенности системы программирования, а также рассмотрен пример вычисления операции SAXPY (умножение вектора на скаляр и сложение векторов) при помощи GPGPU Nvidia CUDA.

Ключевые слова: *графический процессор, архитектура графического процессора, программные модели, GPGPU, Nvidia CUDA, параллельное программирование.*

Вступление. Анализ вычислительной мощности современных графических процессоров, которые появлялись в последние годы, показывает, что на текущий момент времени она в десятки раз выше мощности многоядерных процессоров (*рис. 1*).

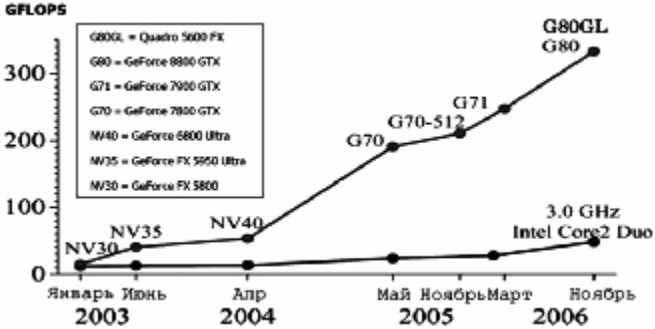

Рис. 1. *Динамика роста вычислительной мощности современных процессоров*

Особый интерес представляют даже не показатели мощности, а динамика ее роста. Если еще три-четыре года назад быстродействие GPU Nvidia GeForce NX 5800 и центральных процессоров было фактически идентичным (около десяти-двадцати гигафлоп), то к концу прошлого года мощности процессоров выросли всего в несколько раз, а GPU Nvidia GeForce 8800 GTX демонстрирует почти три с половиной сотни гигафлоп.

На наш взгляд, подобный отрыв сохранится еще несколько лет, поскольку слишком велико различие в архитектурах (*рис. 2*) универ-

сальных процессоров общего назначения (CPU – central processor unit) и процессоров, ориентированных на обработку графики (GPU – graphic processor unit). Решение задачи максимального использования возможностей графических процессоров представляется весьма многообещающим [1].

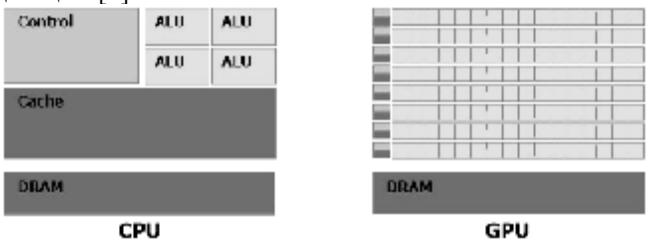

Рис. 2. *Архитектуры CPU и GPU*

Высокий уровень параллельной нагрузки, характерный для трехмерной компьютерной графики, воспроизводимой в режиме реального времени, предъявляет крайне высокие требования к пропускной способности арифметического блока и потоковой памяти, но вместе с тем допускает весьма существенные задержки при выполнении отдельных вычислительных операций, поскольку окончательное изображение выводится с интервалом 16 миллисекунд. Подобные характеристики рабочей нагрузки определяют базовую архитектуру графического процессора. Если процессор общего назначения оптимизируется с учетом требования минимизации задержек, то для графического процессора главное – оптимизация пропускной способности.

Однако специализированная архитектура графических процессоров не всегда подходит для эффективного решения задач. Многие приложения ориентированы на последовательную обработку и для них характерны непредсказуемые всплески обращений к памяти. Например, чтение из памяти для графических процессоров Nvidia составляет 400-600 тактов, а операция сложения 4 такта. При этом следует учитывать архитектуру конкретного графического процессора. Вместе с тем, большинство задач требуют значительных вычислительных ресурсов и, поэтому, хорошо укладываются в характерную для графического процессора схему интенсивной многоядерной арифметической обработки. Для большинства задач важным при обработке больших объемов данных является высокая пропускная способность, и тут уже в полном блеске может продемонстрировать себя подсистема потоковой памяти графического процессора [2].

Нестандартная утилизация производительности графических процессоров может предоставить вычислительные мощности для решения задач в различных областях науки, причем независимо от уникальных сверхсистем. Примером может быть появление технологии Nvidia CUDA и программных продуктов для ее поддержки.

Архитектура Nvidia CUDA. Архитектура CUDA (*рис. 3*) формируется множеством 32-разрядных SIMD-процессоров (Single Instruction Multiple Data, один поток вычислений, обрабатывающих несколько потоков данных). Каждый вычислитель SIMD-процессора – это, по сути, фрагмент полноценного процессора: собственные арифметико-логическое устройство, шина данных, регистровая память и т.д. Общим же для всего SIMD-процессора является декодер команд, управляющий работой всех вычислителей, что и обеспечивает синхронное выполнение ими одной и той же последовательности операций над разными потоками данных.

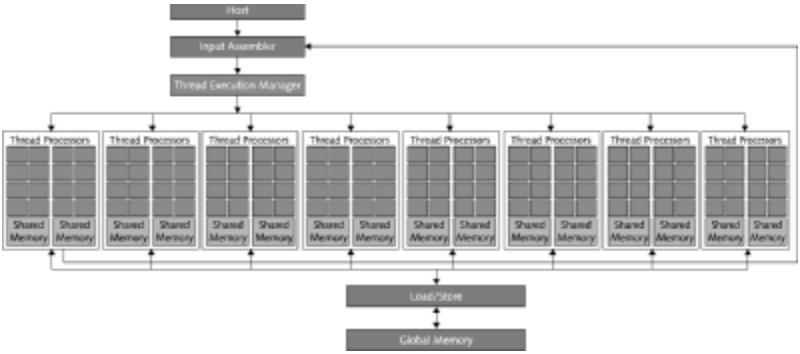

Рис. 3. *Архитектура графического процессора Nvidia GeForce 8*

Основу аппаратных средств CUDA-вычислителя образует потоковый процессор (SP, Streaming Processor), 32-битовое арифметико-логическое устройство которого и предоставляет каждому потоку вычислительные возможности.

Восемь SP, каждый со своим модулем регистровой памяти емкостью 32 KB, объединяются в потоковый мультипроцессор (SM, Streaming Multiprocessor) – вычислительную машину SIMD-архитектуры, имеющую собственные механизм выборки и декодирования команд, независимые кэши команд и данных (констант), диспетчер потоков и блок памяти емкостью 16 KB, разделяемой между всеми восемью SP. Работающий на тактовой частоте 1,35 GHz, SM способен предоставлять более чем семистам потокам вычислительную мощность порядка 20 GFLOPS. SP выполняет одну SIMD-команду за один машинный такт.

Два SM, дополненные более высокоуровневой кэш-памятью команд и данных, образуют кластер обработки текстур (TPC, Texture Processing Cluster). И, наконец, из восьми TPC формируется собственно CUDA-вычислитель, – массив потоковых процессоров (SPA, Streaming Processor Array).

Таким образом, в распоряжении CUDA-программиста имеется вычислительная система, пусть весьма сложная из-за разнообразия

адресных пространств и специфических механизмов, но все же оснащенная 128-мю 32-разрядными арифметико-логическими устройствами. Эти устройства способны за один такт выполнять такие команды, как умножение с накоплением (обычно обозначаются MADD, смысл этой трехоперандной операции понятен из псевдокода A=A+B*C). Потенциальная пиковая производительность такой системы – 346 GFLOPS. Это одновременно и немало, и не очень много, если учесть тот факт, что пиковая производительность четырехъядерных процессоров класса Core 2 Duo совсем немного не дотягивает до 100 GFLOPS [4].

Программные модели. Программная модель CUDA значительно отличается от однопоточной схемы программирования центральных процессоров, и от технологии параллельного программирования графических ускорителей, не поддерживающих технологию CUDA. Основным отличием является присутствие суперскалярного процессора. Суперскалярный процессор представляет собой нечто большее, чем обычный последовательный (скалярный) процессор. В отличие от последнего, он может выполнять несколько операций за один такт. Основными компонентами суперскалярного процессора являются устройства для интерпретации команд, снабженные логикой, позволяющей определить, являются ли команды независимыми, и достаточное число исполняющих устройств. Суперскалярные процессоры реализуют параллелизм на уровне команд.

В однопоточной модели, центральный процессор обрабатывает единственную (отдельную) команду из потока команд, которая обрабатывает последовательные данные. Суперскалярный центральный процессор может направить поток команд через множественные конвейеры, но все-таки, это один поток команд, и степень параллелизма команд строго ограничена данными и зависит от конкретной архитектуры. Даже лучшие четырех, пяти, или шести путевые суперскалярные центральные процессоры могут выполнить в среднем 1.5 команды в такт, что и показывает, почему суперскалярные решения редко содержат больше четырех-путевых конвейеров. SIMD расширения позволяют центральному процессору извлекать не только некоторый параллелизм данных из кода, но и практический предел – три или четыре операции в такт [2].

Другая модель – это универсальное применение графический процессоров (GPGPU, *рис. 4*). Эта модель относительно новая и привлекает к себе все больше внимания в последние годы. По существу, разработчики, для достижения высокой эффективности решения различных задач начали использовать графические процессоры как универсальные процессоры. Здесь понятие “универсальные” означает использование приложений с интенсивными вычислениями в различных научных и технических отраслях. При этом используются пик-

сельные шейдеры графических ускорителей как универсальные с одинарной точностью (FPU) сопроцессоры для операций с плавающей точкой. GPGPU обработка сильно распараллелена, но она трудно реализуема из-за использования памяти вне графического процессора при обработке больших наборов данных. Видеопамять, используемая для текстур и расчета графики, может работать с любым видом данных в приложениях GPGPU, а различные потоки взаимодействуют друг с другом через память вне видеокарты. Эти частые обращения к памяти имеют тенденцию ограничивать производительность [1].

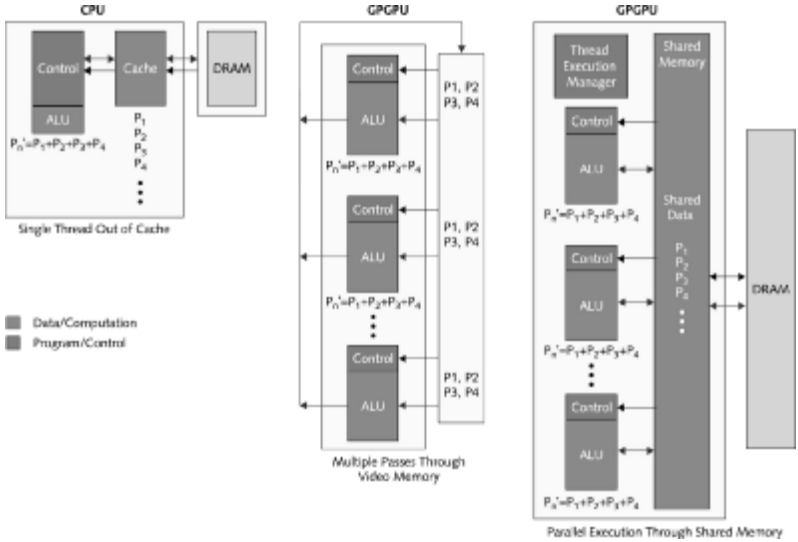

Рис. 4. *Модели высокоэффективных вычислений*

Слева на *рисунке 4* – типичный универсальный центральный процессор, который выполняет единственный поток, выборка потока команд осуществляется из кеша команд и операций с данными, которые проходят через кэш данных от оперативной памяти. Средняя часть *рисунка 4*, иллюстрирует модель GPGPU, которая использует программируемые пиксельные шейдеры GPU для общих вычислений операций. Это сильно распараллеленная модель вычислений, однако ввод-вывод данных связан с частыми доступами к оперативной памяти вне видеокарты.

Справа на *рисунке 4* – модель Nvidia CUDA, с помощью которой кэшируются меньшие порции данных на чипе и разделяется поток данных среди кластера потоковых процессоров (шейдеров).

Особенности Nvidia CUDA. Важная особенность CUDA – это то что, прикладным программистам, которые будут ее использовать, не нужно писать код с определенно разделенными потоками. Аппа-

ратный менеджер потоков обрабатывает их автоматически. Автоматическое управление потоками важно, когда многопоточность масштабируется на тысячи выполняемых потоков. Хотя это "легкие" потоки (каждый работает на маленькой части данных) и вместе с тем полностью самостоятельные потоки в обычном смысле. Каждый поток имеет свой собственный стек, файл регистра, программный счетчик, и свою память [3].

Значение термина "поток" в этом контексте следует уточнить – под потоком понимается некоторая исполняющаяся сущность – а именно функция, значение которой вычисляется на основании определенного набора входных данных. Если решаемая задача допускает формализацию на уровне многочисленных независимо и одновременно выполняемых копий функции, обрабатывающих разные наборы данных, – появляется возможность использовать производительность графического процессора для ускорения вычислений. В CUDA бинарное представление одновременно исполняемых наборов функций-потоков (в системе команд вычислителей графических процессоров) принято называть ядрами (kernels), и каждому потоку присваивать свой специфический идентификатор. Формирование ядер осуществляется с помощью кросс-компилятора, исполняющегося обычным центральным процессором компьютера, из кода, написанного на С-подобном языке программирования.

Второе специфическое для CUDA понятие – блок потоков (thread block). По сути, блок потоков – это исполняемая сущность ядра, потоки которой обладают возможностью коммуникации. То есть, если ядро – это статический бинарный код, то блок потоков – исполняющиеся вычислителями GPU функции-потоки. Здесь также присутствует управляющая сеть. Например, можно задать точку синхронизации потоков одного блока, а исполнение некоторых потоков при этом будет автоматически "заторможено" для одновременного достижения потоками исполнений требуемой точки. Однако, не существует механизмов, позволяющих потокам из разных блоков обмениваться данными и синхронизировать свое исполнение. Так же, как и каждый отдельный поток, блок потоков имеет собственный идентификатор, так что полный адрес потока в CUDA образуется парой < идентификатор блока, идентификатор потока в блоке > [3].

Каждому потоку доступны следующие ресурсы – регистровая память потока, его локальная память, разделяемая в пределах блока потоков память, а также пул адресных пространств, предназначенных для обмена данными с основным вычислителем системы (иными словами – с программно-аппаратными средствами компьютера, в который вставлена используемая в качестве вычислительного акселератора видеокарта, которого принято называть хостом).

После того, как выполнен анализ данных, можно приступать к получению решения в С или С++. Для этой цели, CUDA добавляет

несколько специальных расширений и вызовов API к языку, показанные на *рисунке 5*.

```
Computing y _ ax + y with a serial loop:
void saxpy_serial(int n, float alpha, float *x, float *y)
{
    for(int i = 0; i<n; ++i)
        y[i] = alpha*x[i] + y[i];
}
// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

а)

```
Computing y _ ax + y in parallel using CUDA:
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if( i<n )  y[i] = alpha*x[i] + y[i];
}
// Invoke parallel SAXPY kernel (256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

б)

Рис. 5. *Пример фрагментов программы для Nvidia CUDA*

Первый фрагмент кода (*рис. 5а*) – однопоточная функция для обычного центрального процессора. Второй (*рис. 5б*) – многопоточное выполнение тех же самых функций для CUDA. Обе функции выполняют так называемую SAXPY операцию ($y_i = \alpha x_i + y_i$), умножение вектора на скаляр и сложение векторов, на массиве чисел с плавающей точкой. Эта функция (или "ядро" в CUDA-терминологии) – часть базовой библиотеки линейной алгебры (basic linear algebra subprograms (BLAS)), включенная с CUDA. Расширение global указывает, что функция saxpy_parallel принадлежит CUDA-ядру, которое должно быть откомпилировано для Nvidia GPU, а не для центрального процессора, и это ядро глобально доступно для целой программы. Другая инструкция расширения, в последней строке CUDA примера, использует три пары угольников (<<<nblocks, 256>>>), чтобы определить размер сетки данных и блоков. Первый параметр nblocks определяет измерения сетки в блоке, а второй, в данном случае 256, определяет размер блоков в потоках.

Выводы. Таким образом, применение графических процессоров позволяет ускорить обработку больших объемов данных, в том числе и для быстрого решения современных и сложных задач в научной и технической отраслях. Технология Nvidia CUDA предоставляет про-

граммные и аппаратно-вычислительные мощности для решения сложных задач в различных областях науки, таких как математика, медицина, физика, экономика т.д.

Список использованной литературы:

1. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. – Ver 1.1. – December 2007. – NVIDIA Corporation, 2007.
2. SUPERCOMPUTING 2007 Tutorial: High Performance Computing with CUDA (http:// www.gpgpu.org/sc2007/).
3. ECE 498 AL1: Programming Massively Parallel Processors (http:// courses. ece. uiuc. edu/ ece498/al1/).

A review and comparison of the systems of programming of the general programming is produced for graphic processors (GPGPU), the features of the system of programming are selected, and also the example of calculation of operation of SAXPY (multiplying of vector by a scalar and addition of vectors) is considered through GPGPU Nvidia CUDA.

Key words: *graphic processors unit, Architecture graphic processors unit, program models, GPGPU, General-Purpose computation on graphic processors unit, Nvidia CUDA, parallel programming.*

Отримано: 05.06.2008

УДК 004.942

В. А. Іванюк

Кам'янець-Подільський національний університет

ЛАНЦЮГОВО-ДРОБОВА АПРОКСИМАЦІЯ ІРРАЦІОНАЛЬНИХ ТА ТРАНСЦЕНДЕНТНИХ ПЕРЕДАТНИХ ФУНКЦІЙ ОБ'ЄКТІВ З РОЗПОДІЛЕНИМИ ПАРАМЕТРАМИ

Розроблено та реалізовано алгоритми ланцюгово-дробової апроксимації ірраціональних та трансцендентних передатних функцій об'єктів з розподіленими параметрами, досліджено точність наближень.

Ключові слова: *передатна функція, об'єкти з розподіленими параметрами, ланцюгові дроби, апроксимація.*

Вступ. Передатні функції об'єктів з розподіленими параметрами містять ірраціональні та трансцендентні функції від аргументу p, що значно ускладнює можливості чисельної реалізації. При вирішенні таких задач виникає необхідність проведення їх апроксимації. На сьогоднішній день питання побудови апроксимаційних моделей об'єктів з розподіленими параметрами не знайшли достатньо повного розв'язання.