УДК 004.04

АНАЛИЗ МЕТОДОВ И ТЕХНОЛОГИЙ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ ДЛЯ ОБРАБОТКИ ИНФОРМАЦИИ

Кудухов Алан Нодарович

аспирант
Северо-Кавказский горно-металлургический институт
(Государственный технологический университет), Владикавказ

apriori-journal.ru

Аннотация. В статье изложены основы параллельных вычислений и особенности организации обработки программных потоков в операционной системе Windows, а также проведен сравнительный анализ архитектур CUDA и CPU.

Ключевые слова: процесс; поток; ядро; кэш-память; CUDA; CPU.

ANALYSIS METHODS AND PARALLEL COMPUTING TECHNOLOGIES FOR INFORMATION PROCESSING

Kuduhov Alan Nodarovich

postgraduate student
North Caucasian Institute of Mining and Metallurgy
(State technological university), Vladikavkaz

Abstract. The article describes the basics of parallel computing and feature of the organization of processing program threads in Windows operating system, and also the comparative analysis of the CUDA architecture and CPU.

Key words: process; thread; kernel; cache-memory; CUDA; CPU.

В современном мире все больше роль играют технологии, обеспечивающие хранение, анализ и обработку данных. Связанно это с наблюдаемым с конца прошлого века лавинообразным ростом информации. Современные задачи и приложения, связанные с анализом данных, предъявляют особые требования к вычислительным ресурсам, значительно превышающих мощность отдельных компьютеров [1].

В течение 30 лет одним из основных методов повышения производительности компьютеров было увеличении тактовой частоты. В первых персональных компьютерах, появившихся в начале 1980-х годов, генератор тактовых импульсов внутри CPU работал на частоте 1 МГц. В настоящее время тактовая частота процессора в большинстве настольных компьютеров составляет от 1 до 4 ГГц, т.е. примерно в 1000 раз быстрее своих прародителей. Необходимо отметить, что увеличение частоты тактового генератора – далеко не единственный способ повышения производительности вычислений, но всегда был наиболее надежным из всех [2].

В последние годы инженеры оказались перед необходимостью искать замену этому традиционному источнику повышения быстродействия. Из-за фундаментальных ограничений при производстве интегральных схем уже невозможно рассчитывать на увеличение тактовой частоты процессора как средство получения дополнительной производительности от существующих архитектур. Ограничение на потребляемую мощность и на тепловыделение, а также быстро приближающийся физический предел размера транзистора заставляют инженеров и производителей искать новые решение данной проблемы.

Рассмотрим особенности функционирования графического процессора. У них своя собственная архитектура вычислений, и до недавнего времени было довольно трудно представить, чтобы графические чипы помогали процессору в каких-либо вычислениях, кроме как для них предназначающихся (обработка графики, поддержка GUI, векторные вы-

числения). Но теперь появились технологии, позволяющие производить вычисления с использованием графических процессоров, поддерживающих технологию GPGPU (произвольных вычислений на графических процессорах). Одна из таких технологий – CUDA (Compute Unified Device Architecture), позволяющая обрабатывать трудоемкие алгоритмы и большие массивы данных, а также осуществляющая вычисления связанные с математическими моделями и формулами в десятки, и даже в сотни раз быстрее и при этом не обязательно покупать огромные вычислительные центры и мощные сервера.

Чтобы понять, чем отличается архитектура CUDA от других существующих технологий распараллеливания, надо понять, чем стандартные многопроцессорные системы отличаются от графических процессоров.

Рассмотрим стандартную архитектуру процессора:

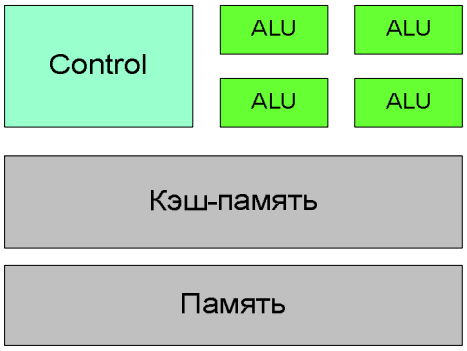

Рис. 1. Архитектура центрального процессора

Как показано выше (см. рис. 1), большую часть площади кристалла занимает *кэш-память*, а вычислительные модули (арифметико-логическое устройство, АЛУ) занимают лишь четверть кристалла. Кроме того, каждый отдельный АЛУ является полноценным центральным процессором. Он способен поддерживать все аппаратные прерывания, может работать со всеми устройствами ввода/вывода, что, несомненно, полезно для его функционирования как центрального процессора, однако становится излишним для его использования как векторного вычислительного модуля.

Кэш-память представляет собой высокоскоростное запоминающее устройство небольшой емкости для временного хранения данных, значительно более быстро действующее, чем основная память, но в отличие от оперативной памяти, не адресуемое и непосредственно "невидимое" для программиста (см. рис. 2).

Основным назначением кэш-памяти являются:

- ✓ обеспечение быстрого доступа к интенсивно используемым данным;
- ✓ согласование интерфейсов процессора и контроллера памяти;
- ✓ упреждающая загрузка данных;
- ✓ отложенная запись.

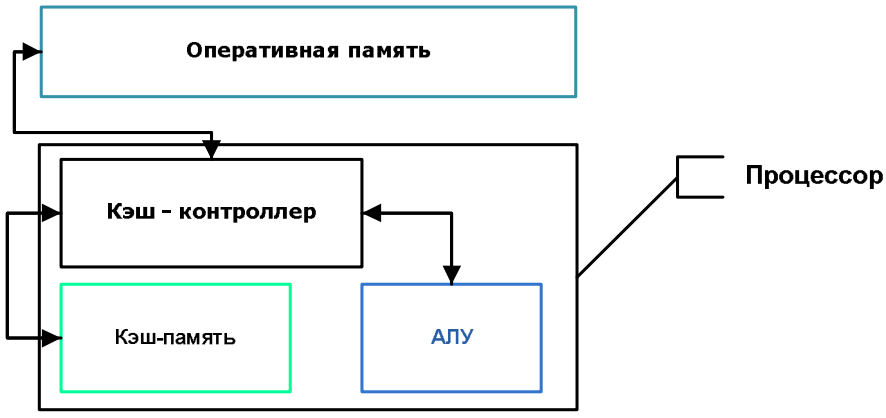

Рис. 2. Расположение кэш-памяти в иерархии оперативной памяти

Перехватывая запросы к основной памяти, кэш-контроллер проверяет: есть ли действительно копия затребованных данных в кэш-памяти. Если такая копия там действительно присутствует, то произошло так называемое *кэш-попадание*, в противном случае говорят о *промахе*, и тогда данные переадресуются к основной памяти. Кроме того, потоки исполнения (код программы), которые выполняются на центральном процессоре, являются очень «*тяжеловесными*», ими управляет операционная система, поэтому их не может быть много (максимальное количество измеряется несколькими тысячами). Для лучшего понимания всей сложности параллельного программирования в операционной системе

Windows, рассмотрим такие важные понятия как: процесс, поток исполнения и контекст переключения.

Процесс обычно определяют как экземпляр выполняемой программы, и он состоит из двух компонентов:

- ✓ объект ядра, через который операционная система управляет процессом. Там же храниться статическая информация о процессе;
- ✓ адресного пространства, в котором содержится код и данные всех EXE и DLL библиотек.

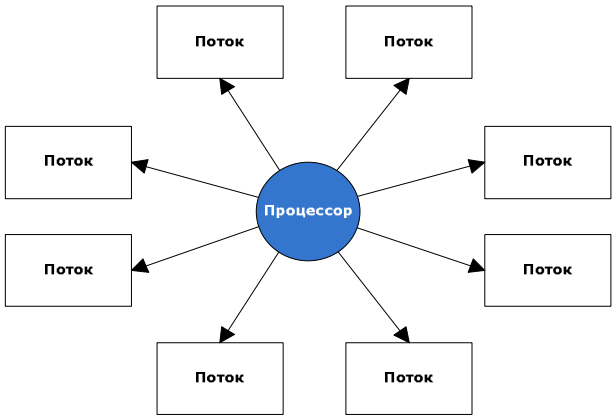

Рис. 3. Планирование потоков в операционной среде Windows

Процессы инертны. Чтобы процесс что-нибудь выполнил, в нем нужно создать поток выполнения. Именно потоки отвечают за исполнения кода, содержащегося в адресном пространстве процесса. В принципе, один процесс может владеть несколькими потоками и тогда они одновременно исполняют код в адресном пространстве процесса. Для этого каждый поток выполнения должен располагать собственным набором регистров процессоров и собственным стеком. В каждом процессе есть минимум один поток выполнения [3].

Чтобы все потоки работали, операционная система отводит каждому из них определенное процессорное время (квант). Выделяя потокам отрезки времени по принципу карусели, она создает тем самым иллю-

зию одновременного выполнения потоков. Рис. 3 иллюстрирует распределение процессорного времени между потоками на одной машине с одним процессором. Когда заканчивается время потока, операционная система прерывает его выполнение, т.е. переключается на другой поток. При этом обязательно происходит следующее:

- ✓ значения регистров процессора исполняющегося в данный момент потока сохраняются в структуре контекста, которая располагается в ядре потока;
- ✓ из набора имеющихся потоков выбирается тот, которому будет передано управление;
- ✓ значение из выбранной структуры контекста потока загружаются в регистры процессора.

После переключение контекста, процессор исполняет выбранный поток, пока не истечет выделенное потоку время, после этого снова происходит переключение контекста. Поэтому потоки в операционной системе Windows, являются «*тяжеловесными*», вследствие чего создают разные парадигмы параллельного программирования на CPU.

Рассмотри код метода на языке C#, который создает отдельный поток.

```
public static void CreateThread()
{
    Console.WriteLine("Основной поток - {0}", Thread.CurrentThread.ManagedThreadId);
    Thread thread = new Thread(() =>
    {
      Console.WriteLine("Новый поток - {0}", Thread.CurrentThread.ManagedThreadId);
    });
    thread.Start();
}
```

Рис. 4. Создание потока на языке программирования C#

Из кода видно, что объект (***Thread***), который создает поток, передает данный поток операционной системе, точнее планировщику потоков, для запуска и выполнения. Таким образом, данный процесс обработки потоков отнимает много времени у операционной системы. Технология CUDA, упрощает написание параллельного кода, в частности она, предлагает использовать параллелизм на уровне данных.

И так, CUDA – это архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию графического процессора.

Рассмотрим схематическую архитектуру графического адаптера.

Рис. 5. Схематичное изображение графического адаптера

Главная часть графического процессора приходится на арифметико-логическое устройство (АЛУ) при этом на долю кэш-памяти и устройства управления отводится очень мало места (см. рис. 5). Ядра на GPU, значительно проще, чем ядра CPU, они могут выполнять только математические операции и не способны к самостоятельной деятельности.

Вычислительная архитектура CUDA основана на концепции «одна команда – множество данных» (*Single Instruction Multiple Data, SIMD*).

Концепция SIMD подразумевает, что одна инструкция позволяет одновременно обработать множество данных. Мультипроцессор – это многоядерный SIMD процессор, позволяющий в каждый определенный момент времени выполнять на всех ядрах только одну инструкцию.

Основная цель анализируемой архитектуры – это размещение на кристалле нескольких десятков процессорных ядер, с собственной памятью, каждая из которых выполняет несколько сотен программных потоков[4]. Процессорные ядра, называемые в CUDA – терминологии мультипроцессорами, имеют собственный доступ к глобальной памяти,

расположенной на графическом процессоре и устройство обменивается данными с CPU через шину PCI-Express.

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*). Одновременно исполняется одна сетка. Каждый блок может быть одного, двух или трехмерным по форме, и может состоять из 512 потоков на текущем аппаратном обеспечении.

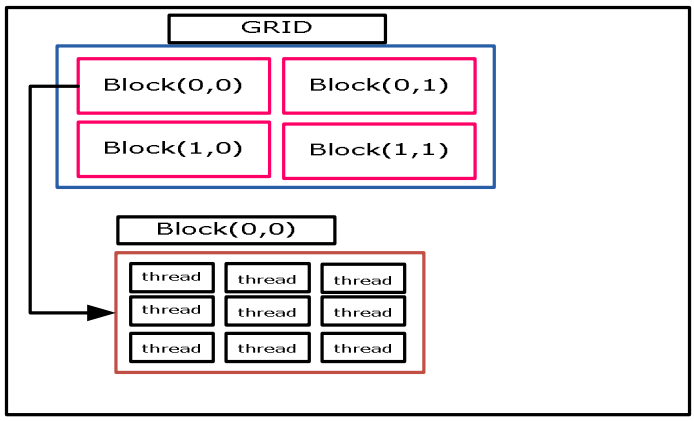

Рис. 6. Вычислительное устройство GPU

Блоки потоков выполняются в виде небольших групп, называемых варп (warp), размер которых – 32 потока. Это минимальный объём данных, который может обрабатываться в мультипроцессорах. И так как это не всегда удобно, CUDA позволяет работать и с блоками, содержащими от 64 до 512 потоков.

Группировка блоков в сетки позволяет уйти от ограничений и применить ядро к большему числу потоков за один вызов. Это помогает и при масштабировании. Если у GPU недостаточно ресурсов, он будет выполнять блоки последовательно. В противном случае, блоки могут выполняться параллельно, что важно для оптимального распределения

работы на видеоадаптерах разного уровня, начиная мобильными и заканчивая интегрированными.

Технология параллельного программирования CUDA упрощает параллелизм данных, превращая графический процессор, в мощный сопроцессор для CPU.

Вывод

Таким образом, на сегодняшний день параллельные вычисления становятся стандартом в разработке крупномасштабных приложений, анализе и обработке данных. Важные задачи в области интеллектуального анализа данных, а также разработки эффективных алгоритмов невозможны без применения параллельного вычисления. Как показывает проведенный анализ, технология CUDA позволяет облегчить написание параллельного кода, посредством использования концепции SIMD, также переносит вычисления на графический процессор.

Список использованных источников

1. Сухорослов О.В. Новые технологии распределенного хранения и обработки больших массивов данных // Всерос. конкурсный отбор обзорно-аналит. стат. по приоритетному направлению «Информационно-телекоммуникационные системы». М., 2008. 40 с.
2. Сандерс Д., Кэндрот Э. Технология CUDA в примерах. Введение в программирование графических процессоров.
3. Джеффри Рихтер. Создание эффективныхWin32-приложений с учетом спецификации 64-разрядной версии Windows.
4. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М.: ДМК-Пресс, 2010. 232 с.