Боресков А. В., Харламов А. А.

Основы работы с технологией CUDA

ДМК Пресс
издательство
Москва, 2010

УДК 32.973.26-018.2
ББК 004.4
Б82

Б82 **Боресков А. В., Харламов А. А.**
Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.: ил.
ISBN 978-5-94074-578-5

Данная книга посвящена программированию современных графических процессоров (GPU) на основе технологии CUDA от компании NVIDIA. В книге разбираются как сама технология CUDA, так и архитектура поддерживаемых GPU и вопросы оптимизации, включающие использование .PTX.

Рассматривается реализация целого класса алгоритмов и последовательностей на CUDA.

К книге прилагается CD, который содержит примеры решения на CUDA реальных задач с большим объемом вычислений из широкого класса областей, включая моделирование нейронных сетей, динамику движения элементарных частиц, геномные исследования и многое другое.

УДК 32.973.26-018.2
ББК 004.4

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-5-94074-578-5

© Боресков А. В., Харламов А. А., 2010
© Оформление, издание, ДМК Пресс, 2010

Содержание

Глава 1. Существующие многоядерные системы. Эволюция GPU. GPGPU 7
1.1. Многоядерные системы 8
1.1.1. Intel Core 2 Duo и Intel Core i7 8
1.1.2. Архитектура SMP 9
1.1.3. BlueGene/L 10
1.1.4. Архитектура GPU 11
1.2. Эволюция GPU 11

Глава 2. Модель программирования в CUDA. Программно-аппаратный стек CUDA 17
2.1. Основные понятия 17
2.2. Расширения языка C 22
2.2.1. Спецификаторы функций и переменных 22
2.2.2. Добавленные типы 23
2.2.3. Добавленные переменные 23
2.2.4. Директива вызова ядра 23
2.2.5. Добавленные функции 24
2.3. Основы CUDA host API 26
2.3.1. CUDA driver API 27
2.3.2. CUDA runtime API 27
2.3.3. Основы работы с CUDA runtime API 31
2.3.4. Получение информации об имеющихся GPU и их возможностях 31
2.4. Установка CUDA на компьютер 34
2.5. Компиляция программ на CUDA 35
2.6. Замеры времени на GPU, CUDA events 41
2.7. Атомарные операции в CUDA 42
2.7.1. Атомарные арифметические операции 42
2.7.2. Атомарные побитовые операции 44
2.7.3. Проверка статуса нитей warp'а 44

Глава 3. Иерархия памяти в CUDA. Работа с глобальной памятью 45
3.1. Типы памяти в CUDA 45
3.2. Работа с константной памятью 46
3.3. Работа с глобальной памятью 47
3.3.1. Пример: построение таблицы значений функции с заданным шагом 49
3.3.2. Пример: транспонирование матрицы 49
3.3.3. Пример: перемножение двух матриц 50

3.4. Оптимизация работы с глобальной памятью 51
3.4.1. Задача об N-телах 55

Глава 4. Разделяемая память в CUDA и ее эффективное использование 59
4.1. Работа с разделяемой памятью 59
4.1.1. Оптимизация задачи об N телах 60
4.1.2. Пример: перемножение матриц 62
4.2. Паттерны доступа к разделяемой памяти 66
4.2.1. Пример: умножение матрицы на транспонированную 69

Глава 5. Реализация на CUDA базовых операций над массивами – reduce, scan, построения гистограмм и сортировки 72
5.1. Параллельная редукция 72
5.2. Нахождение префиксной суммы (scan) 79
5.2.1. Реализация нахождения префиксной суммы на CUDA 80
5.2.2. Использование библиотеки CUDPP для нахождения префиксной суммы 86
5.3. Построение гистограммы 88
5.4. Сортировка 98
5.4.1. Битоническая сортировка 98
5.4.2. Поразрядная сортировка 101
5.4.3. Использование библиотеки CUDPP 102

Глава 6. Архитектура GPU, основы PTX 106
6.1. Архитектура GPU Tesla 8 и Tesla 10 106
6.2. Введение в PTX 108
6.2.1. Типы данных 111
6.2.2. Переменные 112
6.2.3. Основные команды 114

Глава 7. Иерархия памяти в CUDA. Работа с текстурной памятью 121
7.1. Текстурная память в CUDA 122
7.2. Обработка цифровых сигналов 123
7.2.1. Простые преобразования цвета 124
7.2.2. Фильтрация. Свертка 128
7.2.3. Обнаружение границ 134
7.2.4. Масштабирование изображений 137

Глава 8. Взаимодействие с OpenGL 142
8.1. Создание буферного объекта в OpenGL 142
8.2. Использование классов 143

8.3. Пример шума Перлина 147
8.3.1. Применение 150

Глава 9. Оптимизации 152
9.1. PTX-ассемблер 155
9.1.1. Занятость мультипроцессора 156
9.1.2. Анализ PTX-ассемблера 157
9.2. Использование CUDA-профайлера 161

Приложение 1. Искусственные нейронные сети 163
П1.1. Введение 163
П1.1.1. Задачи классификации (Classification) 163
П1.1.2. Задачи кластеризации (Clustering) 164
П1.1.3. Задачи регрессии и прогнозирования 164
П1.2. Модель нейрона 165
П1.3. Архитектуры нейронных сетей 166
П1.4. Многослойный персептрон 166
П1.4.1. Работа с многослойным персептроном 167
П1.4.2. Алгоритм обратного распространения ошибки 169
П1.4.3. Предобработка данных 171
П1.4.4. Адекватность данных 171
П1.4.5. Разбиение на наборы 171
П1.4.6. Порядок действий при работе с многослойным персептроном 172
П1.5. Персептроны и CUDA 173
П1.5.1. Пример задачи реального мира 174
П1.6. Литература 178

Приложение 2. Моделирование распространения волн цунами на GPU 179
П2.1. Введение 179
П2.2. Математическая постановка задачи 181
П2.3. Программная модель 183
П2.4. Адаптация алгоритма под GPU 186
П2.5. Заключение 191
П2.6. Литература 191

Приложение 3. Применение технологии NVIDIA CUDA для решения задач гидродинамики 193
П3.1. Введение 193
П3.2. Сеточные методы 194
П3.2.1. Геометрический многосеточный метод 195
П3.2.2. Алгебраический многосеточный метод 197
П3.2.3. Метод редукции 198

П3.2.4. Оценка эффективности 199
П3.3. Метод частиц 200
П3.4. Статистическая обработка результатов 201
П3.5. Обсуждение 202
П3.6. Литература 203

Приложение 4. Использование технологии CUDA при моделировании динамики пучков в ускорителях заряженных частиц 205
П4.1. Введение 205
П4.2. Особенности задачи 205
П4.3. Использование многоядерных процессоров 208
П4.4. Реализация на графических процессорах 210
П4.5. Результаты 214
П4.6. Литература 216

Приложение 5. Трассировка лучей 218
П5.1. Обратная трассировка лучей 219
П5.1.1. Поиск пересечений 221
П5.1.2. Проблемы трассировки лучей на GPU 222
П5.1.3. Ускорение поиска пересечений 223
П5.2. Оптимизация трассировки лучей для GPU 228
П5.2.1. Экономия регистров 228
П5.2.2. Удаление динамической индексации 229
П5.3. Литература 230

Глава 1
Существующие многоядерные системы. Эволюция GPU. GPGPU

Одной из важнейших характеристик любого вычислительного устройства является его быстродействие. Для математических расчетов быстродействие обычно измеряется в количестве floating-point операций в секунду (*Flops*). При этом довольно часто рассматривается так называемое пиковое быстродействие, то есть максимально возможное число операций с вещественными (*floating-point*) величинами в секунду (когда вообще нет других операций, обращений к памяти и т. п.).

Реальные цифры по загрузке (пиковая vs реальная) по CPU и GPU

На самом деле реальное быстродействие всегда оказывается заметно ниже пикового, поскольку необходимо выполнять другие операции, осуществлять доступ к памяти и т. п.

Для персонального компьютера быстродействие обычно напрямую связано с тактовой частотой центрального процессора (CPU). Процессоры архитектуры x86 за время с момента своего появления в июне 1978 года увеличили свою тактовую частоту почти в 700 раз (с 4,77 МГц у Intel 8086 до 3,33 ГГц у Intel Core i7).

Таблица 1.1. Динамика роста тактовых частот

Год	Тактовая частота	Процессор
1978	4.77 MHz	Intel 8086
2004	3.46 GHz	Intel Pentium 4
2005	3.8 GHz	Intel Pentium 4
2006	2.333 GHz	Intel Core Duo T2700
2007	2.66 GHz	Intel Core 2 Duo E6700
2007	3 GHz	Intel Core 2 Duo E6800
2008	3.33 GHz	Intel Core 2 Duo E8600
2009	3.06 GHz	Intel Core i7 950

Однако если внимательно посмотреть на динамику роста частоты CPU, то становится заметно, что в последние годы рост частоты заметно замедлился, но зато появилась новая тенденция – создание многоядерных процессоров и систем и увеличение числа ядер в процессоре.

Это связано как с ограничениями технологии производства микросхем, так и с тем фактом, что энергопотребление (а значит, и выделение тепла) пропорцио-

нально четвертой степени частоты. Таким образом, увеличивая тактовую частоту всего в 2 раза, мы сразу увеличиваем тепловыделение в 16 раз. До сих пор с этим удавалось справляться за счет уменьшения размеров отдельных элементов микросхем (так, сейчас корпорация Intel переходит на использование технологического процесса в 32 нанометра).

Однако существуют серьезные ограничения на дальнейшую миниатюризацию, поэтому сейчас рост быстродействия идет в значительной степени на счет увеличения числа параллельно работающих ядер, то есть через параллелизм. Скорее всего, эта тенденция сохранится в ближайшее время, и появление 8- и 12-ядерных процессоров не заставит себя долго ждать.

Максимальное ускорение, которое можно получить от распараллеливания программы на N процессоров (ядер), дается законом Амдала (Amdahl Law):

$$S = \frac{1}{(1+P)+\frac{P}{N}}.$$

В этой формуле P – это часть времени выполнения программы, которая может быть распараллелена на N процессоров. Как легко видно, при увеличении числа процессоров N максимальный выигрыш стремится к $\frac{1}{1-P}$. Таким образом, если вы можем распараллелить 3/4 всей программы, то максимальный выигрыш составит 4 раза.

Именно поэтому крайне важно использование хорошо распараллеливаемых алгоритмов и методов.

1.1. Многоядерные системы

Рассмотрим в качестве иллюстрации несколько существующих многоядерных систем и начнем наше рассмотрение с процессоров Intel Core 2 Duo и Intel Core i7.

1.1.1. Intel Core 2 Duo и Intel Core i7

Процессор Intel Core 2 Duo содержит два ядра (P0 и P1), каждое из которых фактически является процессором Pentium M, со своим L1-кешем команд и L1-кешем данных (по 32 Кбайта каждое). Также имеется общий L2-кеш (размером 2 или 4 Мб), совместно используемый обоими ядрами (см. рис. 1.1).

Процессор Core i7 содержит уже четыре ядра (P0, P1, P2 и P3). Каждое из этих ядер обладает своими L1-кешем для данных и L1-кешем для команд (по

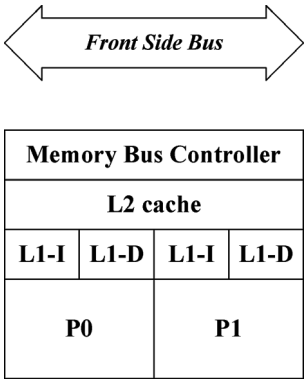


Рис. 1.1. Схема процессора Intel Core 2 Duo

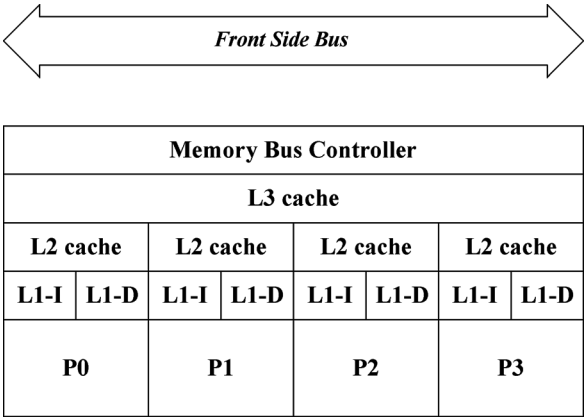


Рис. 1.2. Схема процессора Intel Core i7

32 Кбайта каждое) и L2-кешем (256 Кбайт). Кроме того, имеется общий для всех ядер L3-кеш размером 8 Мбайт.

1.1.2. Архитектура SMP

Кроме этих процессоров, есть также и другие многопроцессорные архитектуры. Одной из таких систем является система на основе симметричной мультипроцессорной архитектуры (Simmetric MultiProcessor Architecture, SMP).

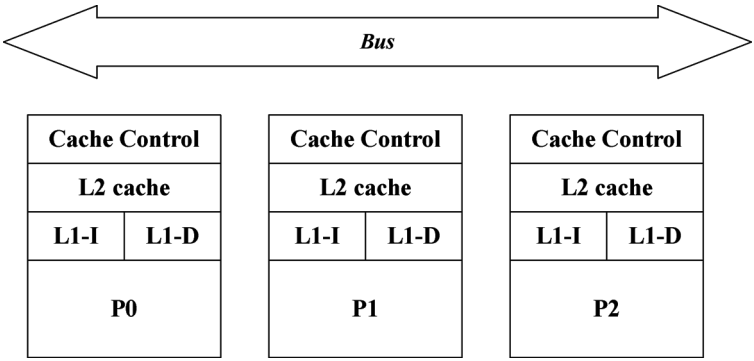


Рис. 1.3. Архитектура SMP-систем

В таких системах каждое ядро содержит свои кеши L1 и L2, и все ядра подсоединены к общей шине. Блок Cache Control отслеживает изменения памяти другими процессорами и обновляет соответствующим образом содержимое кешей – ведь если один и тот же участок памяти содержится в кеше сразу нескольких ядер, то любое изменение этого участка памяти одним из ядер должно быть немедленно передано в кеши других ядер, работающих с данным участком памяти.

Это типичная проблема архитектур с набором процессоров, подключенных к общей шине, – принципиальным моментом для всех многоядерных систем является то, что каждый процессор должен «видеть» целый и корректный образ памяти, что неизбежно ведет к необходимости для каждого процессора отслеживать обращения к памяти всех остальных процессоров для поддержания актуальности своих кешей. Подобная задача имеет квадратичную сложность от числа процессоров.

1.1.3. BlueGene/L

Еще одним примером многопроцессорной архитектуры является суперкомпьютер BlueGene/L. Он состоит из 65 536 двухъядерных узлов (*nodes*). Каждый узел содержит два 770 МГц процессора PowerPC. Каждый из них имеет свои кеши первого и второго уровней, специализированный процессор для *floating-point* вычислений *(Double Hammer FPU*). Оба процессора подключены к общему кешу третьего уровня (L3) размером 4 Мб и имеют собственный блок памяти размером 512 Мб (см. рис. 1.4).

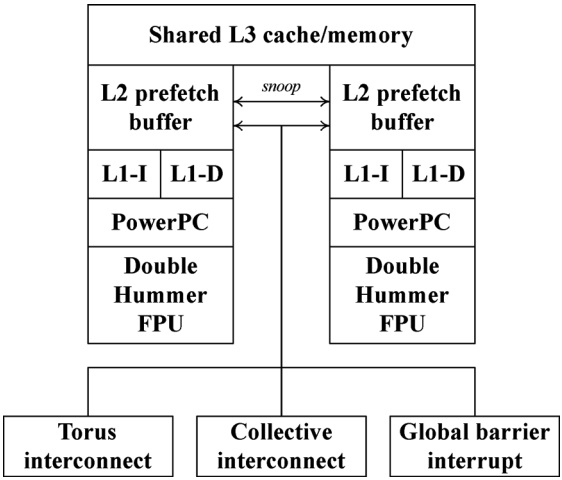


Рис. 1.4. Устройство одного узла в архитектуре BlueGene/L

Отдельные узлы могут соединяться между собой различными способами при помощи набора портов. Таким образом, каждый узел может непосредственно обратиться всего к небольшому числу других узлов, но за счет соединения всех узлов в сеть сообщение может быть передано любому другому узлу за небольшое количество шагов.

Для минимизации числа шагов, необходимых для передачи сообщения от одного узла другому, очень хорошо подходит топология тороидального куба. В ней все узлы образуют куб и у каждого узла есть ровно восемь соседей. При этом если узел лежит на одной из граней куба, то недостающих соседей он берет с противоположной грани (рис. 1.5).