Содержание

От издательства 10

Предисловие 11

Глава 1. Введение 14

1.1. Традиционное использование параллелизма 14
1.2. Параллелизм в современном аппаратном обеспечении 16
1.3. Основные концепции параллельного программирования 18
1.4. Краткий обзор содержания книги 20

Глава 2. Архитектура параллельного компьютера 23

2.1. Архитектура процессора и тенденции развития технологии 24
2.2. Потребление электроэнергии в процессорах 29
2.3. Время доступа к памяти 32
2.3.1. Время доступа к DRAM 33
2.3.2. Многопоточность для сокрытия времени доступа к памяти 34
2.3.3. Кеши для уменьшения среднего времени доступа к памяти 35
2.4. Классификация параллельных архитектур Флинна 36
2.5. Организация памяти параллельных компьютеров 38
2.5.1. Компьютеры с распределенной памятью 39
2.5.2. Компьютеры с совместно используемой памятью 43
2.6. Параллелизм на уровне потоков 47
2.6.1. Одновременная многопоточность 48
2.6.2. Мультиядерные процессоры 49
2.6.3. Архитектура мультиядерных процессоров 51
2.7. Соединительные сети 57
2.7.1. Свойства соединительных сетей 59
2.7.2. Сети прямого соединения 61
2.7.3. Варианты встраивания 67
2.7.4. Динамические соединительные сети 70
2.8. Маршрутизация и коммутация 76
2.8.1. Алгоритмы маршрутизации 77
2.8.2. Маршрутизация в сети омега 85
2.8.3. Коммутация 88
2.8.4. Механизмы управления потоком 96
2.9. Иерархия кешей и памяти 97
2.9.1. Характеристики кешей 99
2.9.2. Стратегии записи 108
2.9.3. Согласованность кеша 110

2.9.4. Согласованность памяти ... 120
2.10. Примеры аппаратного параллелизма ... 126
2.10.1. Архитектуры Intel Cascade Lake и Ice Lake ... 127
2.10.2. Список Top500 ... 131
2.11. Упражнения к главе 2 ... 132

Глава 3. Модели параллельного программирования ... 137
3.1. Модели для параллельных систем ... 138
3.2. Параллельный режим в программах ... 141
3.3. Уровни распараллеливания ... 144
3.3.1. Распараллеливание на уровне машинных инструкций ... 144
3.3.2. Распараллеливание по данным ... 146
3.3.3. Распараллеливание цикла ... 148
3.3.4. Функциональное распараллеливание ... 151
3.3.5. Явное и неявное представление распараллеливания ... 153
3.3.6. Паттерны параллельного программирования ... 156
3.4. SIMD-вычисления ... 162
3.4.1. Выполнение векторных операций ... 162
3.4.2. SIMD-инструкции ... 164
3.5. Распределения данных для массивов ... 166
3.5.1. Распределение данных для одномерных массивов ... 167
3.5.2. Распределение данных для двумерных массивов ... 169
3.5.3. Параметризованное распределение данных ... 170
3.6. Обмен информацией ... 171
3.6.1. Совместно используемые переменные ... 171
3.6.2. Операции обмена данными ... 173
3.7. Вычисление произведения матрицы на вектор в параллельном режиме ... 181
3.7.1. Параллельное вычисление скалярных произведений ... 182
3.7.2. Параллельное вычисление линейных комбинаций ... 185
3.8. Процессы и потоки ... 186
3.8.1. Процессы ... 188
3.8.2. Потоки ... 189
3.8.3. Механизмы синхронизации ... 194
3.8.4. Разработка эффективных и корректных потоковых программ ... 197
3.9. Перспективные методики параллельного программирования ... 200
3.10. Упражнения к главе 3 ... 203

Глава 4. Анализ производительности параллельных программ ... 207
4.1. Оценка производительности компьютерных систем ... 208
4.1.1. Оценка производительности ЦП ... 209
4.1.2. MIPS и MFLOPS ... 211
4.1.3. Производительность процессоров в иерархии памяти ... 213
4.1.4. Программы эталонных тестов ... 215
4.2. Метрики производительности для параллельных программ ... 221
4.2.1. Время выполнения в параллельном режиме и затраты ... 221

4.2.2. Коэффициент ускорения вычислений и эффективность 222
4.2.3. Слабая и сильная масштабируемость 226
4.3. Измерение энергозатрат и метрики энергии 232
4.3.1. Методики измерения производительности и потребления энергии 233
4.3.2. Моделирование потребления мощности и энергии для DVFS 239
4.3.3. Метрики энергопотребления для параллельных программ 240
4.4. Асимптотическое время для глобальных операций обмена данными 243
4.4.1. Реализация глобальных операций обмена данными 244
4.4.2. Операции обмена данными в гиперкубе 251
4.4.3. Операции обмена данными в полном двоичном дереве 260
4.5. Анализ времени параллельного выполнения 264
4.5.1. Скалярное произведение в параллельном режиме 264
4.6. Параллельные вычислительные модели 269
4.6.1. Модель PRAM 269
4.6.2. Модель BSP 273
4.6.3. Модель LogP 275
4.7. Планирование и разделение на блоки цикла 278
4.7.1. Планирование цикла 279
4.7.2. Разбиение цикла на блоки 290
4.8. Упражнения 293

Глава 5. Программирование передачи сообщений 299

5.1. Введение в MPI 300
5.1.1. MPI-операции двухточечного обмена данными 303
5.1.2. Взаимоблокировки при двухточечном обмене данными 308
5.1.3. Неблокирующие операции двухточечного обмена данными 311
5.1.4. Режимы обмена данными 315
5.2. Групповые операции обмена данными 318
5.2.1. Групповые операции обмена данными в MPI 318
5.2.2. Взаимоблокировки при выполнении групповых операций обмена данными 334
5.2.3. Неблокирующие групповые операции обмена данными 337
5.3. Группы процессов и коммуникаторы 339
5.3.1. Группы процессов в MPI 340
5.3.2. Топологии процессов 345
5.3.3. Хронометраж и удаление процессов 350
5.4. Дополнительные темы 351
5.4.1. Динамическая генерация и управление процессами 351
5.4.2. Односторонний обмен данными 354
5.5. Упражнения к главе 5 364

Глава 6. Многопоточное программирование 369

6.1. Программирование с использованием Pthreads 370
6.1.1. Создание и объединение потоков 372
6.1.2. Согласование потоков с помощью Pthreads 376

6.1.3. Условные переменные 384
6.1.4. Расширенный механизм блокировки 388
6.1.5. Однократная инициализация 390
6.2. Паттерны параллельного программирования с использованием Pthreads 391
6.2.1. Реализация пула задач 391
6.2.2. Распараллеливание методом конвейеризации 397
6.2.3. Реализация модели клиент–сервер 404
6.3. Дополнительные функциональные возможности Pthreads 409
6.3.1. Атрибуты потока и отмена его выполнения 409
6.3.2. Планирование потоков с помощью Pthreads 416
6.3.3. Инверсия приоритета 420
6.3.4. Специализированные данные потоков 424
6.4. Java Threads 426
6.4.1. Создание потоков в Java 426
6.4.2. Синхронизация потоков Java 430
6.4.3. Ожидание и оповещение 439
6.4.4. Расширенные паттерны синхронизации 445
6.4.5. Планирование потоков в Java 450
6.4.6. Пакет `java.util.concurrent` 451
6.5. OpenMP 458
6.5.1. Директивы компилятора 460
6.5.2. Подпрограммы среды выполнения 468
6.5.3. Согласование и синхронизация потоков 469
6.5.4. Модель задач OpenMP 475
6.6. Упражнения к главе 6 481

Глава 7. Программирование GPU общего назначения 486

7.1. Архитектура графических процессоров 487
7.2. Введение в программирование с использованием CUDA 494
7.3. Синхронизация и совместно используемая память 500
7.4. Планирование потоков в CUDA 505
7.5. Эффективный доступ к памяти и методика тайлинга 507
7.6. Введение в OpenCL 513
7.7. Упражнения к главе 7 516

Глава 8. Алгоритмы решения систем линейных уравнений 518

8.1. Метод исключения Гаусса 519
8.1.1. Метод исключения Гаусса и LU-разложение матрицы 520
8.1.2. Параллельная реализация с циклическим по строкам распределением данных 523
8.1.3. Параллельная реализация с распределением в шахматном порядке 527
8.1.4. Анализ времени выполнения в параллельном режиме 534
8.2. Прямые методы решения систем линейных уравнений с ленточной структурой 539

8.2.1. Дискретизация уравнения Пуассона 539
8.2.2. Трехдиагональные системы 545
8.2.3. Обобщение для ленточных матриц 557
8.2.4. Решение дискретизированного уравнения Пуассона 559
8.3. Итеративные методы решения систем линейных уравнений 562
8.3.1. Стандартные итеративные методы 563
8.3.2. Параллельная реализация итерационного метода Якоби 567
8.3.3. Параллельная реализация итерационного метода Гаусса–Зейделя 569
8.3.4. Итерационный метод Гаусса–Зейделя для разреженных систем 571
8.3.5. Красно-черное упорядочение 574
8.4. Метод сопряженных градиентов 580
8.4.1. Последовательный метод сопряженных градиентов 581
8.4.2. Параллельная реализация метода сопряженных градиентов 583
8.5. Разложение Холецкого для разреженных матриц 588
8.5.1. Последовательный алгоритм 588
8.5.2. Схема хранения для разреженных матриц 594
8.5.3. Реализация для совместно используемых переменных 596
8.6. Упражнения к главе 8 602

Список литературы 606

Предметный указатель 618

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

Инновации в архитектуре аппаратных средств, такие как гиперпоточность или мультиядерные процессоры, делают ресурсы параллельных вычислений доступными для компьютерных систем в различных областях, включая настольные и портативные компьютеры, мобильные устройства и встроенные системы. Но эффективное использование ресурсов параллельных вычислений требует применения методов параллельного программирования. Сегодня многие стандартные программные продукты уже основаны на концепциях параллельного программирования для эффективного использования аппаратных ресурсов мультиядерных процессоров. Эта тенденция сохраняется, и потребность в параллельном программировании распространяется на все области разработки программного обеспечения. Область применения существенно расширится по сравнению со сферой научных вычислений, которая на протяжении многих лет была основным объектом применения параллельных вычислений. Расширение области использования параллельных вычислений приведет к огромной потребности в разработчиках программного обеспечения с навыками параллельного программирования. Некоторые производители микросхем уже требуют включить параллельное программирование в качестве стандартного курса в учебные программы по информатике. Более поздней тенденцией является использование графических процессоров (GPU), которые могут содержать несколько тысяч ядер, для выполнения ресурсоемких неграфических приложений.

В этой книге рассматриваются новые разработки в области архитектуры процессоров и параллельного аппаратного оборудования. Кроме того, представлены важные методы параллельного программирования, необходимые для разработки эффективных программ для мультиядерных процессоров, а также для параллельных кластерных систем или суперкомпьютеров. Описываются архитектуры как общего, так и распределенного адресного пространства. Основная цель книги – представить методы параллельного программирования, которые можно использовать в разнообразных ситуациях для многочисленных прикладных областей, и предоставить читателю возможность разрабатывать корректные и эффективные параллельные программы. Для достижения этой цели и демонстрации того, как такие методы можно применять в реальных приложениях, приведено множество примеров программ и упражнений. Книга может быть использована как учебник для студентов, а также как справочник для специалистов. Материал книги уже много лет используется на курсах параллельного программирования в различных университетах.

Третье издание книги по параллельному программированию на английском языке представляет собой обновленную и исправленную версию, основанную на втором издании книги 2013 г. Три предыдущих издания на

немецком языке вышли в 2000, 2007 и 2012 гг. соответственно. Самое последнее издание включает расширенное обновление главы, посвященной компьютерной архитектуре и анализу производительности, с учетом новых разработок, таких как вопросы энергопотребления. Описание OpenMP было расширено и теперь отражает концепцию задач OpenMP. Глава, посвященная программированию передачи сообщений, была расширена и обновлена и теперь включает новые функции MPI, такие как расширенные операции редукции и неблокирующие операции коллективной связи. Глава, посвященная программированию графических процессоров, также была обновлена. Кроме того, тщательно переработаны были и все остальные главы.

Книга состоит из трех основных частей, охватывающих все области параллельных вычислений: архитектуру параллельных систем, модели и среды параллельного программирования, а также реализацию эффективных прикладных алгоритмов. Основное внимание сосредоточено на методах параллельного программирования, необходимых для различных архитектур.

Первая часть содержит обзор архитектуры параллельных систем, включая организацию кеша и памяти, сетевых соединений, методы маршрутизации и коммутации, а также технологии, актуальные для современных и будущих мультиядерных процессоров. Также освещены вопросы энергопотребления.

Во второй части представлены модели параллельного программирования, модели производительности и среды параллельного программирования для передачи сообщений, а также модели совместно используемой памяти, включая интерфейс передачи сообщений (MPI), Pthreads, потоки Java и OpenMP. Для каждой из этих сред параллельного программирования описаны базовые концепции, а также более продвинутые методы программирования, что позволяет читателю писать и запускать семантически корректные и эффективные с точки зрения вычислений параллельные программы. Паттерны параллельного проектирования, такие как конвейерная обработка, клиент–сервер или пулы задач, представлены для различных сред, чтобы проиллюстрировать методы параллельного программирования и облегчить реализацию эффективных параллельных программ для широкого спектра прикладных областей. Подробно описаны модели производительности и методы анализа времени выполнения, поскольку они являются необходимым условием достижения эффективности и высокой производительности. В этой части приводится подробное описание архитектуры графических процессоров, а также содержится введение в методы программирования для графических процессоров общего назначения с упором на CUDA и OpenCL. Приводятся примеры программирования, демонстрирующие использование конкретных описанных методов программирования.

В третьей части методы параллельного программирования из второй части применяются к репрезентативным алгоритмам научных вычислений. Главное внимание сосредоточено на основных методах решения систем линейных уравнений, которые играют важную роль во многих методиках научного моделирования. Следует отметить, что здесь основной темой является анализ алгоритмической структуры различных алгоритмов как основы распараллеливания, а не только математические характеристики методов

решения. Для каждого алгоритма в книге обсуждаются разные варианты распараллеливания с использованием разных методов и стратегий.

Многие коллеги и студенты помогли улучшить качество этой книги. Мы хотели бы поблагодарить всех за помощь и конструктивную критику. За многочисленные исправления благодарим Роберта Дитце (Robert Dietze), Йорга Дюмлера (Jörg Dümmler), Марвина Фербера (Marvin Ferber), Михаэля Хофмана (Michael Hofmann), Ральфа Хофмана (Ralf Hoffmann), Сашу Гунольда (Sascha Hunold), Томаса Якобса (Thomas Jakobs), Оливера Клёкнера (Oliver Klöckner), Матиаса Корха (Matthias Korch), Ронни Крамера (Ronny Kramer), Рафаэла Куниса (Raphael Kunis), Йенса Ланга (Jens Lang), Изабель Мюльман (Isabel Mühlmann), Джона О'Донелла (John O'Donnell), Андреаса Прелля (Andreas Prell), Карстена Шольтеса (Carsten Scholtes), Михаэля Швинда (Michael Schwind) и Йеспера Трэфа (Jesper Träff). Большое спасибо Томасу Якобсу (Thomas Jakobs), Матиасу Корху (Matthias Korch), Карстену Шольтесу (Carsten Scholtes) и Михаэлю Швинду (Michael Schwind) за помощь с примерами программ и упражнениями. Мы благодарим Монику Глазер (Monika Glaser) и Луизу Штайнбах (Luise Steinbach) за их помощь и поддержку при верстке книги в LaTeX. Мы также благодарим всех, кто принимал участие в написании трех редакций этой книги на немецком языке. Было очень приятно сотрудничать с издательством Springer Verlag при работе над этой книгой. Мы особенно благодарим Ральфа Герстнера (Ralf Gerstner) за его постоянную поддержку.

Байройт и Хемниц, январь 2023 г. Томас Раубер
Гудула Рюнгер

Глава 1

Введение

Краткое содержание главы

Параллельное программирование становится все более важным для разработки программного обеспечения сегодня и в будущем. Во введении описывается более традиционное использование параллелизма в научных вычислениях с использованием суперкомпьютеров, а также режим параллельных вычислений, доступный в современном оборудовании, который делает доступным использование параллелизма в более широком классе приложений. Основные концепции параллельного программирования представлены менее чем на двух страницах с помощью неформального определения ключевых понятий, таких как декомпозиция задач или потенциальный параллелизм, и переноса их в соответствующий контекст. Приведено обобщенное описание содержания этой книги по параллельному программированию и даны предложения по структуре учебного курса.

1.1. Традиционное использование параллелизма

Параллельное программирование и разработка эффективных параллельных программ уже в течение многих лет прочно закрепились в сфере высокопроизводительных научных вычислений. Моделирование научных задач – важная область естественных и технических наук, приобретающая все большее значение. Более точное моделирование или моделирование более крупных задач приводит к увеличению спроса на вычислительную мощность и объем памяти. В последние десятилетия исследования, ориентированные на повышение производительности, также включали разработку новых параллельных аппаратных и программных технологий, поэтому наблюдается устойчивый прогресс в области параллельных высокопроизводительных вычислений. Характерными примерами являются: моделирование прогноза погоды на основе сложных математических моделей, включающих урав-

нения в частных производных, или моделирование аварий в автомобильной промышленности на основе метода конечных элементов. Есть и другие примеры: разработка лекарственных средств и приложения компьютерной графики для кино и рекламной индустрии.

В зависимости от конкретного способа применения компьютерное моделирование является основным методом получения желаемого результата или используется для замены либо расширения физических экспериментов. Типичным примером первой области применения является прогнозирование погоды, где необходимо предсказать будущее развитие атмосферы, что можно получить только путем моделирования. Во второй области применения компьютерное моделирование используется для получения результатов, которые являются более точными, чем результаты практических экспериментов, или которые можно выполнить с меньшими затратами. Примером является использование моделирования для определения сопротивления воздуха при движении транспортных средств: по сравнению с классическим экспериментом в аэродинамической трубе, компьютерное моделирование может дать более точные результаты, поскольку в моделирование может быть включено относительное движение транспортного средства по отношению к земной поверхности. В аэродинамической трубе это невозможно, так как транспортное средство всегда остается на месте. Краш-тесты транспортных средств представляют собой очевидный пример того, как компьютерное моделирование может быть выполнено с меньшими затратами.

Компьютерное моделирование часто требует больших вычислительных усилий. Таким образом, низкая производительность используемой компьютерной системы может существенно ограничить моделирование и снизить точность полученных результатов. Использование высокопроизводительной системы позволяет проводить более масштабное моделирование, что приводит к улучшенным результатам, поэтому для выполнения компьютерного моделирования обычно используются параллельные компьютеры. Сегодня кластерные системы, построенные из серверных узлов, широко доступны и в настоящее время также часто используются для параллельного моделирования. Кроме того, мультиядерные процессоры внутри узлов обеспечивают дополнительный параллелизм, который можно использовать для быстрых вычислений. Чтобы использовать параллельные компьютеры или кластерные системы, выполняемые вычисления должны быть разделены на несколько частей, которые назначаются параллельно используемым ресурсам. Эти вычислительные части должны быть независимы друг от друга, а применяемый алгоритм должен обеспечивать достаточное количество независимых вычислений, чтобы стать пригодным для параллельного выполнения. Обычно это относится к научному моделированию, которое часто использует одно- или многомерные массивы в качестве структур данных и организует свои вычисления во вложенных циклах. Чтобы получить параллельную программу для выполнения в параллельном режиме, алгоритм должен быть сформулирован на подходящем языке программирования. Параллельное выполнение часто управляется специализированными библиотеками времени выполнения или директивами компилятора, добавляемыми в стандартный

язык программирования, например в C, Fortran или Java. В этой книге описаны методы программирования, необходимые для создания эффективных параллельных программ. Также представлены широко распространенные системы и среды выполнения.

1.2. Параллелизм в современном аппаратном обеспечении

Параллельное программирование – важный аспект высокопроизводительных научных вычислений, но раньше оно занимало обособленную нишу во всей области аппаратных и программных продуктов. Однако в последнее время параллельное программирование выходит за пределы этой ниши и становится основным методом разработки программного обеспечения из-за радикальных изменений в технологиях аппаратного обеспечения.

Крупные производители микросхем начали выпускать процессоры с несколькими энергоэффективными вычислительными блоками на одном кристалле, которые имеют независимое управление и могут одновременно обращаться к одной и той же памяти. Обычно термин «ядро» (core) используется для отдельных вычислительных блоков, а термин «мультиядерный» (multicore) – для всего процессора, имеющего несколько ядер. Таким образом, использование мультиядерных процессоров превращает каждый настольный компьютер в небольшую параллельную систему. Ориентация технологического развития на мультиядерные процессоры была вызвана физическими причинами, поскольку тактовую частоту микросхем с постоянно увеличивающимся количеством транзисторов невозможно наращивать прежними темпами без существенного перегрева.

Мультиядерные архитектуры в виде отдельных мультиядерных процессоров, систем с совместно используемой памятью из нескольких мультиядерных процессоров или кластеров мультиядерных процессоров с иерархической сетью соединений в немалой степени воздействуют на разработку программного обеспечения. В 2022 г. четырехъядерные и восьмиядерные процессоры становятся стандартом для обычных настольных компьютеров, а микросхемы с числом ядер до 64 уже доступны для использования в высокопроизводительных системах. На основе закона Мура можно предсказать, что количество ядер на микросхеме процессора будет удваиваться каждые 18–24 месяца, а через несколько лет обычный процессор может состоять из десятков или сотен ядер, причем некоторые из ядер будут выделены для конкретных целей, например управления сетью, шифрования и дешифрования или графики [138]. Бóльшая часть ядер будет доступна для прикладных программ, обеспечивая огромный потенциал производительности. Еще одна тенденция в сфере параллельных вычислений – использование графических процессоров для ресурсоемких приложений. Архитектуры графических процессоров предоставляют сотни специализированных вычислительных ядер, которые могут выполнять вычисления в параллельном режиме.

Пользователи компьютерной системы заинтересованы в получении выгоды от повышения производительности, обеспечиваемого мультиядерными процессорами. Если такая возможность будет обеспечена, то пользователи вправе ожидать, что их прикладные программы будут работать быстрее и содержать все больше и больше дополнительных функций, которые невозможно было интегрировать в предыдущие версии программного обеспечения из-за слишком высоких требований к вычислительной мощности. Для этого обязательно должна быть обеспечена поддержка со стороны операционной системы, например путем использования выделенных ядер по прямому назначению или посредством параллельного запуска нескольких пользовательских программ, если это возможно. Но при наличии большого количества ядер, что произойдет в ближайшем будущем, также возникает необходимость выполнения одной прикладной программы на нескольких ядрах. Наилучшей ситуацией для разработчика программного обеспечения стало бы существование автоматического преобразователя, который принимает на вход последовательную программу и генерирует параллельный выполняемый код, который эффективно работает на новых архитектурах. Если бы такой преобразователь был доступен, разработка программного обеспечения могла бы продолжаться в том же ключе, что и раньше. Но, к сожалению, опыт исследований по распараллеливанию компиляторов за последние 20 лет показал, что для многих последовательных программ невозможно автоматически создать достаточный уровень параллелизма. Следовательно, необходима определенная помощь со стороны программиста, и прикладные программы должны быть соответствующим образом реструктурированы.

Для автора программного обеспечения развитие нового аппаратного обеспечения в направлении мультиядерной архитектуры представляет собой сложную задачу, поскольку существующее программное обеспечение необходимо реструктурировать для параллельного выполнения, чтобы воспользоваться преимуществами дополнительных вычислительных ресурсов. В частности, разработчики программного обеспечения больше не могут рассчитывать на то, что увеличение вычислительной мощности аппаратуры может автоматически использоваться их программными продуктами. Вместо этого требуются дополнительные усилия на программном уровне, чтобы воспользоваться преимуществами увеличенной вычислительной мощности. Если компания-разработчик программного обеспечения сможет преобразовать свое программное обеспечение так, чтобы оно эффективно работало на новых мультиядерных архитектурах, она, вероятнее всего, получит преимущество перед своими конкурентами.

В сфере практического применения языков и сред параллельного программирования проводятся многочисленные исследования, цель которых – облегчить параллельное программирование путем предоставления поддержки на надлежащем уровне абстракции. Но уже сейчас существует множество эффективных методик и рабочих сред. В этой книге мы предлагаем общий обзор и представляем важные методы программирования, позволяющие читателю разрабатывать эффективные параллельные программы. Существует несколько аспектов, которые необходимо учитывать при разработке параллельной программы независимо от того, какая конк-

ретная среда или система используется. В следующем разделе приводится краткий обзор этих аспектов.

1.3. Основные концепции параллельного программирования

Первый шаг в параллельном программировании – проектирование параллельного алгоритма или программы для конкретной прикладной общей задачи (problem). Проектирование начинается с декомпозиции вычислений приложения на несколько частей, называемых задачами (tasks), которые можно вычислять в параллельном режиме на ядрах или процессорах параллельного аппаратного оборудования. Декомпозиция на задачи может представлять собой сложный и трудоемкий процесс, поэтому обычно существует несколько различных возможных вариантов декомпозиции для одного прикладного алгоритма. Размер задач (например, в количестве инструкций) обозначается термином «степень разбиения» (granularity; иногда просто «грануляция»), и обычно существует возможность выбора задач различных размеров. Правильное определение задач в приложении – одна из основных интеллектуальных проблем в разработке любой параллельной программы, практически не поддающаяся автоматизации. Потенциальный параллелизм (potential parallelism) – это постоянное свойство, присущее прикладному алгоритму, воздействующее на выбор способа разделения приложения на задачи.

Задачи приложения кодируются на языке программирования или в среде с поддержкой параллельного режима и распределяются по процессам (process) или потокам (threads), которым, в свою очередь, назначаются для выполнения физические вычислительные устройства. Распределение задач по процессам или потокам называется планированием (scheduling). Планирование определяет порядок выполнения задач и может осуществляться вручную в исходном коде или средой программирования во время компиляции либо динамически во время выполнения. Распределение процессов или потоков по физическим устройствам, процессорам или ядрам называется маппингом (mapping) и обычно осуществляется системой времени выполнения, но иногда программист может повлиять на маппинг. Задачи прикладного алгоритма могут быть независимыми, но также возможна их зависимость друг от друга, создающая зависимости по данным или по управлению. Зависимости по данным или по управлению могут потребовать определенного порядка выполнения параллельных задач: если для задачи требуются данные, создаваемые другой задачей, то выполнение первой задачи может начаться только после того, как вторая задача действительно сгенерирует эти данные и предоставит требуемую информацию. Таким образом, зависимости между задачами являются ограничивающими условиями для планирования. Кроме того, в параллельных программах необходима синхронизация (synchronization) и координация потоков и процессов для обеспечения корректного выполнения.

Методы синхронизации и координации в параллельных вычислениях тесно связаны со способом обмена информацией между процессами и потоками, а это зависит от организации памяти в аппаратном обеспечении.

При упрощенной классификации организации памяти различаются вычислительные устройства с совместно используемой памятью (shared memory) и устройства с распределенной памятью (distributed memory). Часто термин «поток» связывается с совместно используемой памятью, а термин «процесс» – с распределенной памятью. В вычислительных устройствах первого типа глобальная совместно используемая память хранит данные приложения, и доступ к ней могут получить все процессоры или ядра аппаратных систем. Обмен информацией между потоками осуществляется через совместно используемые переменные, в которые выполняет запись один поток, а чтение – другой. Корректное поведение всей программы в целом должно обеспечиваться синхронизацией между потоками, чтобы доступ к совместно используемым данным являлся согласованным, т. е. поток не должен считывать элемент данных до того, как полностью завершится операция записи того же элемента данных, выполняемая другим потоком. В зависимости от языка программирования или среды синхронизация осуществляется системой времени выполнения или программистом. В вычислительных устройствах с распределенной памятью для каждого процессора существует собственная память, доступ к которой разрешен только этому процессору, поэтому синхронизация операций доступа к памяти не требуется. Обмен информацией осуществляется посредством передачи данных из одного процессора в другой через сетевые соединения с помощью операций обмена данными (communication operations).

Специализированные операции барьера (barrier operations) предоставляют другую форму координации (согласования), доступную для устройств с совместно используемой и распределенной памятью. Все процессы или потоки в точке барьера синхронизации должны ждать до тех пор, пока все прочие процессы или потоки также достигнут этой точки. Только после того, как все процессы или потоки завершат выполнение кода перед барьером, они смогут продолжить работу, выполняя код после барьера.

Важным аспектом параллельных вычислений является время параллельного выполнения (parallel execution time), состоящее из времени вычисления на процессорах или ядрах и времени, необходимого для обмена данными либо синхронизации. Время параллельного выполнения должно быть меньше времени последовательного выполнения на одном процессоре, иначе проектирование параллельной программы не имеет смысла. Время параллельного выполнения – это время от начала выполнения приложения на первом процессоре до завершения выполнения приложения на всех процессорах. На это время влияет распределение работы по процессорам или ядрам, время для обмена информацией и/или синхронизации, а также интервалы времени простоя (idle times), когда процессор не может выполнять полезную работу, а просто ждет наступления некоторого события. В общем случае снижение времени параллельного выполнения достигается, когда рабочая нагрузка равномерно распределяется по процессорам или ядрам – это обозначается термином «балансирование нагрузки» (load balancing), и ког-

да издержки (накладные расходы) на обмен информацией, синхронизацию и интервалы простоя являются минимальными. Поиск конкретного варианта планирования и стратегии маппинга, которые приводят к правильному балансу нагрузки, часто становится трудной задачей из-за множества взаимозависимых факторов. Например, снижение издержек на обмен информацией может привести к дисбалансу нагрузки, тогда как правильное балансирование нагрузки может потребовать увеличения накладных расходов на обмен информацией или синхронизацию.

Для числовой оценки времени выполнения параллельных программ используются такие параметры измерения стоимости, как коэффициент ускорения вычислений (speedup) и коэффициент эффективности (efficiency), которые сравнивают итоговое время параллельного выполнения со временем последовательного выполнения на одном процессоре. Существуют различные способы измерения стоимости или времени выполнения параллельной программы и большое разнообразие моделей оценки параллельности на основе уже разработанных и используемых моделей параллельного программирования. Эти модели позволяют установить связь между конкретным параллельным аппаратным оборудованием и более абстрактными языками программирования и средами с поддержкой параллельного режима.

1.4. Краткий обзор содержания книги

Здесь приведено описание структуры остальной части книги. В главе 2 содержится обзор важных аспектов аппаратного обеспечения параллельных вычислительных систем и новых разработок, таких как тенденция к переходу на мультиядерные архитектуры. В частности, в этой главе рассматриваются важные аспекты организации памяти с совместно используемым и распределенным адресным пространством, а также широко применяемые сетевые соединения с соответствующими топологическими свойствами. Поскольку иерархические структуры памяти с несколькими уровнями кешей могут оказывать существенное воздействие на производительность (параллельных) вычислительных систем, они также рассматриваются в этой главе. Здесь же подробно описана архитектура мультиядерных процессоров. Основная цель главы – предоставить подробный обзор самых значимых аспектов архитектуры параллельных компьютеров, имеющих важное значение для параллельного программирования и для разработки эффективных параллельных программ.

В главе 3 рассматриваются широко распространенные модели и парадигмы параллельного программирования, а также способы представления параллелизма, присущего алгоритмам, в среде параллельного выполнения для обеспечения эффективного параллельного выполнения. Важной частью этой главы является описание механизмов согласования (координации) параллельных программ, включая координацию и операции обмена данными. Также описаны механизмы обмена информацией между вычислительными ресурсами для различных моделей памяти. Глава 4 полностью посвящена

анализу производительности параллельных программ. В ней представлены широко применяемые методы измерения производительности и стоимости, которые также применяются для последовательных программ, и методы измерения, специально разработанные для параллельных программ. Особое внимание уделено общеизвестным паттернам обмена данными для архитектур с распределенным адресным пространством и их эффективным реализациям для конкретных структур взаимодействия.

В главе 5 описана разработка параллельных программ для распределенных адресных пространств. Приведено подробное описание интерфейса MPI (Message Passing Interface), самой распространенной программной среды для распределенных пространств адресов памяти. В этой главе рассматриваются важные свойства и библиотечные функции MPI и демонстрируются методики программирования, которые должны использоваться для создания эффективных MPI-программ. В главе 6 рассматривается разработка параллельных программ для совместно используемых адресных пространств. Для разработки чаще всего применяются такие программные среды, как Pthreads, Java threads и OpenMP. Все три среды подробно описаны с одновременным рассмотрением методик программирования для создания эффективных параллельных программ. Многочисленные примеры помогают лучше понять соответствующие концепции и избежать часто возникающих ошибок, приводящих к снижению производительности или, возможно, к возникновению более серьезных проблем, таких как взаимоблокировки (deadlocks) и состояние гонки (race condition). Также представлены примеры программ и паттерны параллельного программирования. В главе 7 представлены методики программирования для выполнения неграфических приложений на графических процессорах (GPU), например программ для научных вычислений. В этой главе описана архитектура графических процессоров, а основное внимание сосредоточено на программной среде CUDA (Compute Unified Device Architecture) компании Nvidia. Здесь приведено краткое описание OpenCL. В главе 8 рассматриваются алгоритмы численного анализа как типичные примеры, а также показано, как последовательные алгоритмы можно преобразовать в параллельные программы на систематической основе.

Основная задача этой книги – предоставление читателю методик программирования, необходимых для разработки эффективных параллельных программ для различных архитектур, и достаточного количества примеров, позволяющих читателю применять эти методики в программах из разнообразных прикладных областей. В частности, чтение и практическое использование этой книги является неплохим способом обучения разработке программного обеспечения для современных параллельных архитектур, включая мультиядерные архитектуры.

Содержимое этой книги можно использовать в учебных курсах по теме «Параллельные вычисления» с различными акцентами. Все главы являются самодостаточными, поэтому их можно использовать по отдельности. Тем не менее перекрестные ссылки на материал из других глав могут оказаться полезными. Таким образом, различные курсы по теме «Параллельные вычисления» можно скомпоновать из глав книги как отдельных независимых модулей. Упражнения и задания представлены отдельно для каждой главы.

Для курса по программированию мультиядерных систем рекомендуется использовать главы 2, 3 и 6. А именно глава 6 предоставляет обзор соответствующих программных сред и методик. Для общего курса по параллельному программированию можно воспользоваться главами 2, 5 и 6. В этих главах представлены методики программирования для распределенных и совместно используемых адресных пространств. Для курса по параллельным числовым алгоритмам вполне подходят главы 5 и 8, дополнительно можно включить главу 6. В этих главах рассматриваются практически применяемые параллельные алгоритмы, а также требуемые для них методики программирования. Для обобщающего курса по параллельным вычислениям можно использовать главы 2, 3, 4, 5 и 6, дополнив их выборочно взятыми приложениями из главы 8. В зависимости от конкретной специализации в каждый из перечисленных выше курсов можно включить программирование GPU из главы 7. Новые и дополнительные материалы можно найти на веб-странице этой книги: ai2.inf.uni-bayreuth.de/ppbook3.

Глава 2

Архитектура параллельного компьютера

Краткое содержание главы

Возможность параллельного выполнения вычислений в большой степени зависит от архитектуры платформы выполнения, которая определяет, как вычисления программы могут быть связаны с доступными ресурсами для поддержки параллельного выполнения. В этой главе дается обзор общей архитектуры параллельных компьютеров, которая включает в себя организацию памяти, параллелизм на уровне потоков и мультиядерные процессоры, сети взаимосвязей, маршрутизацию и коммутацию, а также кеши и иерархии памяти. Также рассматривается вопрос энергоэффективности.

В разделе 2.1 приводится подробный обзор применения параллельного режима на одном процессоре или ядре процессора. Использование доступных ресурсов одного ядра процессора на уровне инструкций может привести к существенному увеличению производительности. В разделе 2.2 внимание сосредоточено на важных аспектах энергопотребления процессоров. В разделе 2.3 рассматриваются методики, влияющие на время доступа к памяти и играющие важную роль в улучшении производительности (параллельных) программ. Раздел 2.4 представляет классификацию Флинна, а в разделе 2.5 описана организация памяти параллельных платформ. В разделе 2.6 представлена архитектура мультиядерных процессоров и описано применение параллелизма на основе потоков для одновременной многопоточной обработки.

В разделе 2.7 описаны сети взаимосвязей, соединяющие ресурсы параллельных платформ и используемые для обмена данными и информацией между этими ресурсами. Сети взаимосвязей также играют важную роль

в мультиядерных процессорах для соединений между ядрами микросхемы процессора. В этом разделе рассматриваются статические и динамические сети взаимосвязей и их важные характеристики, например диаметр, пропускная способность сечения и связность различных типов сетей, а также встраивание сетей в другие сети. В разделе 2.8 описаны методики маршрутизации для выбора путей в сетях и методики коммутации для передачи сообщений по заданному пути. В разделе 2.9 рассматриваются иерархические структуры памяти последовательных и параллельных платформ, а также обсуждается согласованность (когерентность) кеша и целостность памяти для платформ с совместно используемой памятью. В разделе 2.10 показаны примеры применения параллелизма в современных архитектурах компьютеров с описанием архитектуры процессоров Intel Cascade Lake и Ice Lake, с одной стороны, и списка Top500 – другой.

2.1. Архитектура процессора и тенденции развития технологии

Микросхемы процессоров являются главными компонентами компьютеров. Учитывая тенденции, наблюдаемые в отношении процессорных микросхем в последние годы, можно делать прогнозы на будущее.

Важной характеристикой производительности является тактовая частота (clock frequency; также clock rate или clock speed) процессора, т. е. количество тактовых циклов в секунду с единицей измерения Герц = 1/секунда, сокращенно Гц = 1/с. Тактовая частота f определяет период тактовых импульсов (clock cycle time) t процессора как $t = 1/f$, что обычно является временем, необходимым для выполнения одной инструкции. Таким образом, повышение тактовой частоты приводит к более быстрому выполнению программы, следовательно, к улучшению производительности.

В интервале между 1987 г. и 2003 г. среднее ежегодное увеличение тактовой частоты приблизительно на 40 % можно было наблюдать в процессорах для настольных компьютеров [103]. С 2003 г. тактовая частота таких процессоров остается практически неизменной, и в ближайшем будущем не следует ожидать ее существенного увеличения [102, 134]. Причиной такой линии развития является тот факт, что увеличение тактовой частоты приводит к росту энергопотребления в основном из-за тока утечки, преобразующегося в тепло, что приводит к необходимости более интенсивного охлаждения. При использовании существующей самой современной технологии охлаждения процессоры с тактовой частотой около 4 ГГц невозможно охлаждать постоянно без огромных дополнительных усилий.

Другим важным фактором, влияющим на развитие процессоров, являются технические усовершенствования в сфере производства процессоров. Микросхема процессора состоит из транзисторов. Количество транзисторов, содержащихся в микросхеме, можно использовать как приблизительную оценку сложности и производительности процессора. Закон Мура (Moore's