Параллельные и высокопроизводительные вычисления



Роберт Роби
Джулиана Замора

MANNING
ДМК ПРЕСС ИЗДАТЕЛЬСТВО

УДК 004.421
ББК 32.972
Р58

Роби Р., Замора Дж.
Р58 Параллельные и высокопроизводительные вычисления / пер. с англ. А. В. Логунова. – М.: ДМК Пресс, 2021. – 800 с.: ил.

ISBN 978-5-97060-936-1

Параллельное программирование позволяет распределять задачи обработки данных между несколькими процессорами, существенно повышая производительность. В книге рассказывается, как с минимальными трудозатратами повысить эффективность ваших программ. Вы научитесь оценивать аппаратные архитектуры и работать со стандартными инструментами отрасли, такими как OpenMP и MPI, освоите структуры данных и алгоритмы, подходящие для высокопроизводительных вычислений, узнаете, как экономить энергию на мобильных устройствах, и даже запустите масштабную симуляцию цунами на батарее из GPU-процессоров.

Издание предназначено для опытных программистов, владеющих языком высокопроизводительных вычислений, таким как C, C++ или Fortran.

УДК 004.421
ББК 32.972

Original English language edition published by Manning Publications USA. Russian-language edition copyright © 2021 by DMK Press. All rights reserved.

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.

ISBN 978-1-6172-9646-8 (англ.)
ISBN 978-5-97060-936-1 (рус.)

© Manning Publications, 2021
© Перевод, оформление, издание, ДМК Пресс, 2021

Оглавление

Часть I ▪ ВВЕДЕНИЕ В ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ 36
1 ▪ Зачем нужны параллельные вычисления? 38
2 ▪ Планирование под параллелизацию 75
3 ▪ Пределы производительности и профилирование 102
4 ▪ Дизайн данных и модели производительности 134
5 ▪ Параллельные алгоритмы и шаблоны 179

Часть II ▪ CPU: ПАРАЛЛЕЛЬНАЯ РАБОЧАЯ ЛОШАДКА 233
6 ▪ Векторизация: флопы забесплатно 236
7 ▪ Стандарт OpenMP, который «рулит» 273
8 ▪ MPI: параллельный становой хребет 328

Часть III ▪ GPU: РОЖДЕНЫ ДЛЯ УСКОРЕНИЯ 385
9 ▪ Архитектуры и концепции GPU 389
10 ▪ Модель программирования GPU 430
11 ▪ Программирование GPU на основе директив 458
12 ▪ Языки GPU: обращение к основам 510
13 ▪ Профилирование и инструменты GPU 558

Часть IV ▪ ЭКОСИСТЕМЫ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ 590
14 ▪ Аффинность: перемирие с вычислительным ядром 592
15 ▪ Пакетные планировщики: наведение порядка в хаосе 633
16 ▪ Файловые операции для параллельного мира 654
17 ▪ Инструменты и ресурсы для более качественного исходного кода 691

1 Зачем нужны параллельные вычисления?

Эта глава охватывает следующие ниже темы:

- что такое параллельные вычисления и почему они приобретают все бóльшую значимость;
- где в современном оборудовании существует параллелизм;
- почему важен параллелизм приложений;
- программные подходы для задействования параллелизма.

В современном мире вы столкнетесь со многими задачами, требующими широкого и эффективного использования вычислительных ресурсов. Большинство приложений, требующих производительности, традиционно относится к научной области. Однако, согласно прогнозам, приложения искусственного интеллекта (ИИ) и машинного обучения станут преобладающими пользователями крупномасштабных вычислений. Несколько примеров таких приложений включают:

- моделирование мегапожаров для оказания помощи пожарным командам и населению;
- моделирование цунами и штормовых волн от ураганов (см. главу 13, в которой приводится простая модель цунами);
- распознавание голоса для компьютерных интерфейсов;
- моделирование распространения вируса и разработка вакцин;
- моделирование климатических условий на десятилетия и столетия;

- распознавание изображений для технологии беспилотных автомобилей;
- оснащение аварийных бригад работающими симуляциями таких источников опасности, как наводнение;
- снижение энергопотребления мобильных устройств.

С помощью описанной в этой книге технологии вы сможете справляться с более крупными задачами и наборами данных, одновременно выполняя симуляции в десять, сто или даже в тысячу раз быстрее. Типичные приложения оставляют большую часть вычислительных способностей современных компьютеров незадействованными. Параллельные вычисления представляют собой ключ к выявлению потенциала ваших компьютерных ресурсов. Итак, что такое параллельные вычисления и как их можно использовать, чтобы подстегнуть свои приложения?

Параллельные вычисления – это исполнение большого числа операций в отдельный момент времени. Полное эксплуатирование параллельных вычислений не происходит автоматически. Это требует некоторых усилий от программиста. И прежде всего вы должны определить и выявить потенциал параллелизма в приложении. Потенциальный параллелизм, или *конкурентность*, означает, что вы подтверждаете безопасность выполнения операций в любом порядке по мере появления системных ресурсов. А при параллельных вычислениях существует еще одно, дополнительное требование: эти операции должны выполняться в одно и то же время. Для того чтобы это происходило, вы также должны правильно задействовать ресурсы с целью одновременного исполнения этих операций.

Параллельные вычисления вводят новые трудности, которых нет в последовательном мире. Для того чтобы приспособиться к дополнительным сложностям параллельного исполнения, нам необходимо поменять наши мыслительные процессы, но с практикой это будет становиться второй натурой. Данная книга начинает ваше открытие в том, как получать доступ к мощи параллельных вычислений.

Жизнь представляет массу примеров параллельной обработки, и эти примеры часто становятся основой для вычислительных стратегий. На рис. 1.1 показана кассовая линия супермаркета, где цель состоит в том, чтобы клиенты быстро оплачивали товары, которые они хотят приобрести. Это можно сделать, наняв нескольких кассиров для обработки или проверки клиентов по одному за раз. В этом случае квалифицированные кассиры смогут быстрее исполнять процесс оплаты покупок, в результате чего клиенты будут покидать магазин быстрее. Еще одна стратегия состоит в задействовании большого числа касс самообслуживания и разрешении клиентам исполнять процесс самостоятельно. Эта стратегия требует меньше человеческих ресурсов от супермаркета и позволяет открыть больше линий по обработке клиентов. Клиенты, возможно, не смогут оплачивать покупки так же эффективно, как обученный кассир, но, возможно, больше клиентов сможет быстро выполнять оплату из-за увеличения параллелизма, что приведет к сокращению очередей.

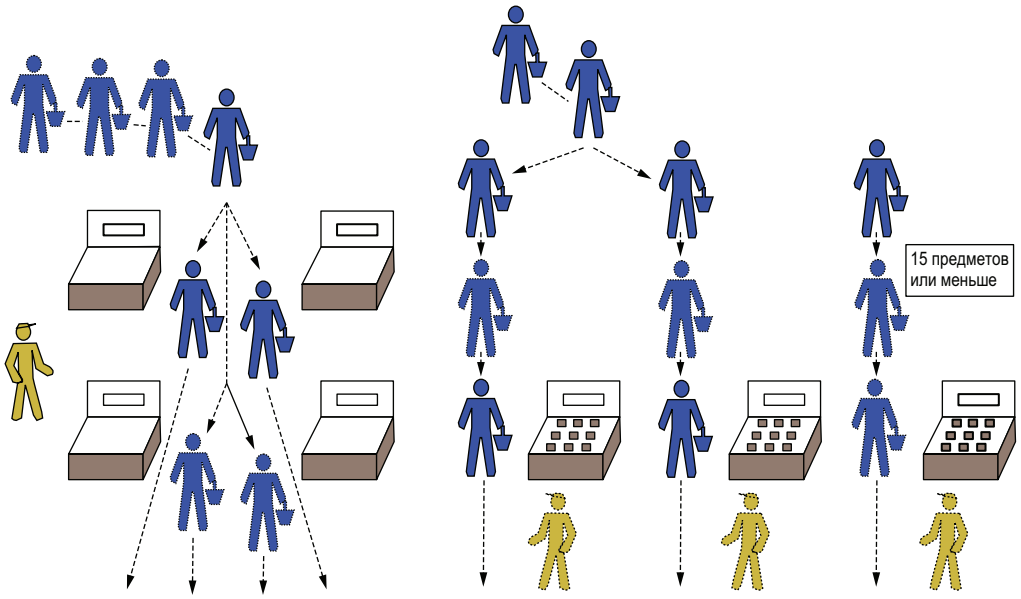


Рис. 1.1 Повседневный параллелизм в очередях в кассы супермаркетов. Кассиры (с кепками) обрабатывают свою очередь клиентов (с корзинами). Слева один кассир одновременно обрабатывает четыре полосы для каждой самостоятельной оплаты покупки. Справа один кассир требуется для каждой полосы оплаты покупки. Каждый вариант влияет на затраты супермаркета и темпы оплаты покупок

Мы решаем вычислительные задачи, разрабатывая *алгоритмы*, перечень шагов для достижения желаемого результата. В аналогии с супермаркетом процесс самообслуживания при оплата покупки представляет собой алгоритм. В данном случае сюда входит выгрузка товаров из корзины, сканирование товаров для получения цены и оплата товаров. Указанный алгоритм является последовательным (или сериальным); он должен следовать этому порядку. Если данная задача должна исполняться сотнями клиентов, то алгоритм оформления покупок многочисленных клиентов содержит параллелизм, которым можно воспользоваться. Теоретически нет никакой зависимости между любыми двумя клиентами, которые проходят процесс оплаты покупки. Используя несколько кассовых линий или станций самообслуживания, супермаркеты демонстрируют параллелизм, тем самым увеличивая скорость, с которой покупатели покупают товары и покидают магазин. Каждый вариант выбора в том, как мы имплементируем этот параллелизм, приводит к разным затратам и выгодам.

ОПРЕДЕЛЕНИЕ *Параллельные вычисления* – это практика выявления и раскрытия параллелизма в алгоритмах с выражением их в нашем программном обеспечении и пониманием затрат, преимуществ и ограничений выбранной имплементации.

В конечном итоге параллельные вычисления всецело касаются *производительности*. Сюда входит не только скорость, но и масштаб задачи и энергоэффективность. Наша цель в этой книге состоит в том, чтобы дать вам понимание широты современной области параллельных вычислений и познакомить с достаточным числом наиболее часто используемых языков, технических приемов и инструментов, чтобы обеспечить возможность уверенно браться за параллельно-вычислительный проект. Важные решения о том, как встраивать параллелизм, часто принимаются в начале проекта. Продуманный дизайн является важным шагом на пути к успеху. Уклонение от шага дизайна может привести к проблемам гораздо позже. Не менее важно сохранять реалистичность ожиданий и знать имеющиеся ресурсы и характер проекта.

Еще одна цель этой главы состоит в ознакомлении вас с терминологией, используемой в параллельных вычислениях. Время от времени это будет делаться, направляя вас к глоссарию в приложении C с целью краткого ознакомления с терминологией, когда вы будете читать эту книгу. Поскольку эта область и технология постепенно расширяются, использование многих терминов участниками параллельного сообщества зачастую бывает небрежным и неточным. В связи с возросшей сложностью оборудования и параллелизма в приложениях важно с самого начала применять терминологию четко и недвусмысленно.

Добро пожаловать в мир параллельных вычислений! По мере того как вы будете углубляться, технические приемы и подходы будут становиться более естественными, и вы обнаружите, что их сила завораживает. Задачи, к решению которых вы никогда не думали приступать, станут обыденным делом.

1.1 Почему вы должны изучить параллельные вычисления?

Будущее будет параллельным. Увеличение последовательной производительности достигло плато, поскольку варианты дизайна процессоров достигли пределов миниатюризации, тактовой частоты, мощности и даже тепла. На рис. 1.2 показаны тренды тактовой частоты (скорости, с которой команда может исполняться), энергопотребления, числа вычислительных ядер (или просто ядер для краткости) и производительности оборудования с течением времени для товарных процессоров.

В 2005 году число ядер резко возросло с одного до нескольких. В то же время тактовая частота и энергопотребление выровнялись. Теоретическая производительность неуклонно повышалась, поскольку производительность пропорциональна произведению тактовой частоты и числа ядер. Этот сдвиг в сторону увеличения числа ядер, а не тактовой частоты указывает на то, что достигуть наиболее идеальную производительность центрального процессора (CPU) можно только за счет параллельных вычислений.

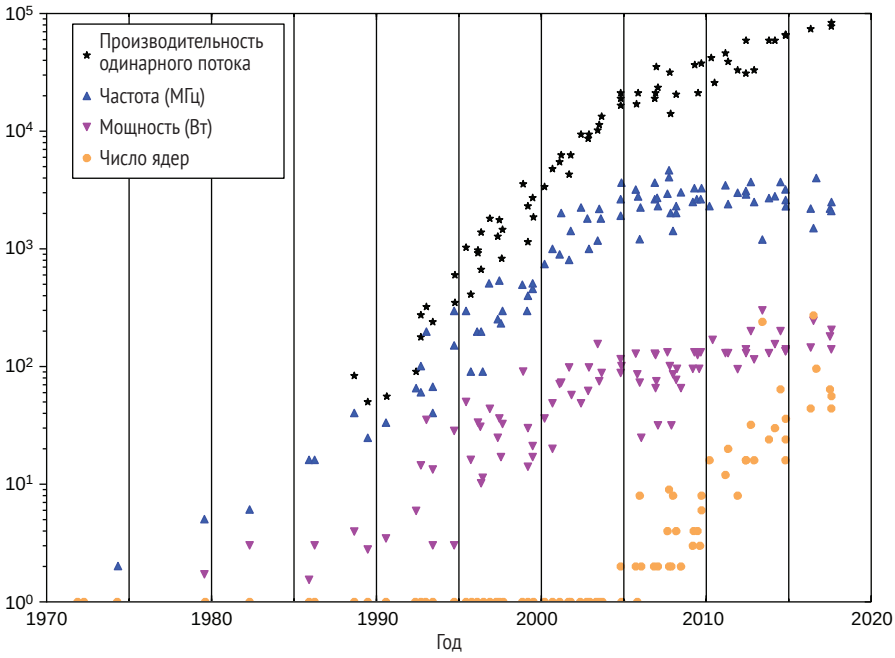


Рис. 1.2 Производительность одинарного потока, тактовая частота процессора (МГц), потребляемая мощность процессора (Вт) и число ядер процессора с 1970 по 2018 год. Эра параллельных вычислений начинается примерно в 2005 году, когда число ядер в процессорных микросхемах начинает расти, в то время как тактовая частота и энергопотребление снижаются, но производительность неуклонно растет (Горовиц и соавт. и Рупп, https://github.com/karlrupp/microprocessor-trend-data**)**

Современное вычислительное оборудование потребительского класса оснащается несколькими центральными процессорами (CPU) и/или графическими процессорами (GPU), которые обрабатывают несколько наборов команд одновременно. Эти небольшие системы часто соперничают по вычислительной мощности с суперкомпьютерами двадцатилетней давности. В целях полного использования вычислительных ресурсов (на ноутбуках, рабочих станциях, смартфонах и т. д.) требуется, чтобы вы, программист, обладали практическими знаниями об инструментах, служащих для написания параллельных приложений. Вы также должны понимать аппаратные функциональности, которые подстегивают параллелизм.

Поскольку существует много разных параллельных аппаратных функциональностей, это создает программисту новые сложности. Одной из таких функциональностей является гиперпотокообразование (hyper-threading)[1], представленное компанией Intel. Наличие двух очередей

[1] Гиперпотокообразование (Hyper-Threading) – это термин, пришедший из компании Intel, для обозначения одновременного множественного потокообразования (simultaneous multithreading, SMT). Это процесс, в котором центральный процессор (CPU) разбивает каждое свое физическое ядро на вир-

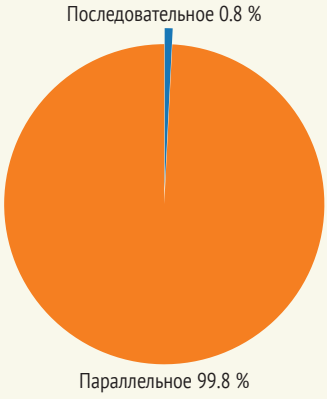
команд, чередующих работу с аппаратными логическими модулями, позволяет одному физическому ядру выглядеть в операционной системе (OS) как два ядра. Векторные процессоры являются еще одной аппаратной функциональностью, которая начала появляться в товарных процессорах примерно в 2000 году. Они исполняют несколько инструкций сразу. Ширина в битах векторного процессора (также именуемого модулем векторной обработки) определяет число команд, которые должны исполняться одновременно. Отсюда модуль векторной обработки шириной 256 бит может исполнять одновременно четыре 64-битовые команды (двойной точности) или восемь 32-битовых команд (одинарной точности).

Пример

Давайте возьмем 16-ядерный CPU с гиперпотокообразованием и векторным модулем шириной 256 бит, который обычно используется в домашних настольных компьютерах. Последовательная программа, использующая одинарное ядро и не имеющая векторизации, использует только 0.8 % теоретических обрабатывающих способностей этого процессора! Расчет выглядит так:

16 ядер × 2 гиперпотока × (векторный модуль шириной 256 бит) / (64-бит двойной точности) = 128-путный параллелизм,

где 1 последовательный путь / 128 параллельных путей = 0.008, или 0.8 %. На следующем ниже рисунке показано, что это составляет мизерную долю суммарной вычислительной мощности процессора.



Последовательное приложение обращается только к 0.8 % вычислительной мощности 16-ядерного процессора

Умение проводить расчет теоретических и реалистичных ожиданий последовательной и параллельной производительности, как показано в этом примере, имеет большую важность. Мы обсудим это подробнее в главе 3.

туальные ядра, именуемые потоками (threads). Например, в большинстве процессоров Intel с двумя ядрами гиперпотокообразование используется для обеспечения четырех потоков. – *Прим. перев.*

Несколько усовершенствований в инструментах разработки программного обеспечения помогло добавить параллелизм в наши инструментарии, и в настоящее время научно-исследовательское сообщество делает больше, но до устранения разрыва в производительности еще далеко. Это ложится большой нагрузкой на нас, разработчиков программного обеспечения, чтобы получать максимальную отдачу от процессоров нового поколения.

К сожалению, разработчики программного обеспечения отстали в адаптации к этому фундаментальному изменению вычислительной мощности. Кроме того, транзит существующих приложений в сторону использования современных параллельных архитектур может оказаться обескураживающе сложным из-за бурного развития новых языков программирования и интерфейсов прикладного программирования (API). Но хорошая работающая осведомленность о вашем приложении, способность видеть и выявлять параллелизм, а также глубокое понимание имеющихся инструментов обязательно принесут существенные преимущества. Какие именно преимущества увидят приложения? Давайте посмотрим поближе.

1.1.1 Каковы потенциальные преимущества параллельных вычислений?

Параллельные вычисления могут сокращать время до решения[1], повышать энергоэффективность вашего приложения и позволять решать более масштабные задачи на существующем в настоящее время оборудовании. Сегодня параллельные вычисления больше не являются единственной областью деятельности крупнейших вычислительных систем. Указанная технология теперь присутствует на всех настольных компьютерах или ноутбуках и даже на портативных устройствах. Она позволяет каждому разработчику программного обеспечения создавать параллельное программное обеспечение в своих локальных системах, тем самым значительно расширяя возможности для новых приложений.

Передовые исследования как в промышленности, так и в научных кругах открывают новые области для параллельных вычислений по мере того, как интерес расширяется от научных вычислений до машинного обучения, больших данных, компьютерной графики и потребительских приложений. Появление новых технологий, таких как самоуправляемые автомобили, компьютерное зрение, распознавание голоса и искусственный интеллект, требует крупных вычислительных способностей как внутри потребительских устройств, так и в сфере разработки, где необходимо использовать и обрабатывать массивные тренировочные наборы данных. И в научных вычислениях, которые долгое время были исключительной областью параллельных вычислений, тоже появляются новые, захваты-

[1] Время до решения (time to solution) – это продолжительность времении между предобработкой и постобработкой данных при решении поставленной задачи. – *Прим. перев.*

вающие возможности. Распространение удаленных датчиков и портативных устройств, которые могут подавать данные в более масштабные и реалистичные вычисления для более оптимального информирования при принятии решений, имеющих отношение к природным и техногенным катастрофам, позволяет получать более обширные данные.

Следует помнить, что параллельные вычисления как таковые не являются самоцелью. Правильне сказать, цели – это то, что является результатом параллельных вычислений: сокращение времени выполнения, выполнение более крупных вычислений или снижение энергопотребления.

Более быстрое время выполнения с бóльшим числом вычислительных ядер

Сокращение времени выполнения приложения, или его *ускорение*, часто считается первостепенной целью параллельных вычислений. И действительно обычно это имеет наибольшие последствия. Параллельные вычисления способны ускорять интенсивные вычисления, обработку мультимедиа и операции с большими данными независимо от того, требуются ли для обработки ваших приложений дни или даже недели, или же результаты необходимы в реальном времени.

В прошлом программист тратил больше усилий на последовательную оптимизацию, чтобы выжать несколько процентных улучшений. Теперь есть потенциал для улучшения на порядки, с многочисленными вариантами на выбор. Это создает новую проблему в разведывании возможных параллельных парадигм – больше возможностей, чем численность программистов. Но глубокое знание вашего приложения и понимание возможностей параллелизма непременно поведут вас по более ясному пути к сокращению времени выполнения вашего приложения.

Более крупные размеры задач с бóльшим числом вычислительных узлов

Выявляя параллелизм в своем приложении, вы можете масштабировать размер вашей задачи вертикально до размерностей, недоступных для последовательного приложения. Это связано с тем, что объем вычислительных ресурсов обуславливает то, что можно сделать, а параллелизм позволяет работать с большими ресурсами, предоставляя возможности, которые раньше никогда не рассматривались. Более крупные размеры обеспечиваются за счет большего объема основной памяти, дискового хранилища, пропускной способности сетей и диска, а также центральных процессоров (CPU). По аналогии с супермаркетом, как упоминалось ранее, выявление параллелизма эквивалентно найму большего числа кассиров либо открытию большего числа касс самостоятельной оплаты покупок с целью обслуживания большего и растущего числа клиентов.

Энергоэффективность: делая больше с меньшими затратами

Одной из новых областей влияния параллельных вычислений является энергоэффективность. С появлением параллельных ресурсов в наладонных устройствах параллелизм может ускорять работу приложений. Это позволяет устройству быстрее возвращаться в спящий режим и дает

возможность использовать более медленные, но более параллельные процессоры, потребляющие меньше энергии. Отсюда перенос работы тяжеловесных мультимедийных приложений на GPU-процессоры может оказать более существенное влияние на энергоэффективность, а также значительно повысить производительность. Чистый результат использования параллелизма снижает энергопотребление и продлевает срок службы батареи, что является сильным конкурентным преимуществом в этой рыночной нише.

Еще одна область, в которой важность энергоэффективности неоспорима, – это удаленные датчики, сетевые устройства и операционные устройства, развернутые на местах, таких как удаленные метеостанции. Нередко без больших источников питания эти устройства должны быть способны функционировать малыми пакетами с небольшим объемом ресурсов. Параллелизм расширяет возможности, которые могут имплементироваться на этих устройствах, и разгружает работу центральной вычислительной системы в растущем тренде, который называется *периферийным вычислением* (edge compute). Перемещение вычислений на самый край сети обеспечивает возможность вести обработку в источнике данных, сжимая их в меньший результирующий набор, который легче отправлять по сети.

Точный расчет энергетических затрат приложения является сложной задачей без прямых измерений энергопотребления. Однако вы можете оценить затраты путем умножения расчетной тепловой мощности производителя на время выполнения приложения и число используемых процессоров. Расчетная тепловая мощность (thermal design power, TDP) – это скорость, с которой энергия расходуется при типичных эксплуатационных нагрузках. Потребление энергии для вашего приложения можно оценить по формуле:

$$P = (N \text{ Процессоров}) \times (R \text{ Ватт/Процессоры}) \times (T \text{ часов}),$$

где P – это потребление энергии, N – число процессоров, R – расчетная тепловая мощность и T – время выполнения приложения.

Пример

16-ядерный процессор Intel Xeon E5-4660 имеет расчетную тепловую мощность 120 Вт. Предположим, что ваше приложение использует 20 из этих процессоров в течение 24 часов до полного завершения работы. Оценочное потребление энергии для вашего приложения составляет:

$$P = (20 \text{ Процессоров}) \times (120 \text{ Вт/Процессоры}) \times (24 \text{ часа}) = 57.60 \text{ кВт/ч}.$$

В общем случае GPU имеют более высокую расчетную тепловую мощность, чем современные CPU, но потенциально способны сократить время работы или потребовать всего нескольких GPU для получения того же результата. Можно применить ту же формулу, что и раньше, где N теперь рассматривается как число GPU.

Пример

Предположим, что вы портировали свое приложение на мульти-GPU-платформу. Теперь вы можете выполнять свое приложение на четырех GPU NVIDIA Tesla V100 за 24 часа! GPU Tesla V100 от NVIDIA имеет максимальную тепловую мощность 300 Вт. Оценочное потребление энергии для вашего приложения составляет:

P = (4 GPU-процессора) × (300 Вт/GPU-процессоры) × (24 часа) = 28.80 кВт/ч.

В этом примере приложение с ускорением на основе GPU работает с вдвое меньшими энергозатратами, чем версия только для CPU. Обратите внимание, что в этом случае, даже несмотря на то, что время до решения остается прежним, затраты энергии сокращаются вдвое!

Достижение снижения энергозатрат с помощью ускорителей, таких как GPU-процессоры, требует наличия в приложении достаточного параллелизма, который можно выявить. Это позволяет эффективно пользоваться ресурсами устройства.

Параллельные вычисления могут снизить затраты

Фактические денежные затраты становятся все более заметной озабоченностью для коллективов разработчиков программного обеспечения, пользователей программного обеспечения и исследователей в равной степени. По мере роста размеров приложений и систем нам необходимо проводить анализ затрат и выгод на имеющихся у нас ресурсах. Например, в следующих крупных системах высокопроизводительных вычислений (HPC) затраты на электроэнергию, по прогнозам, в три раза превысят стоимость приобретения оборудования.

Затраты на использование также способствовали развитию облачных вычислений в качестве альтернативы, которая все чаще внедряется в научных кругах, стартапах и отраслях промышленности. Как правило, поставщики облачных служб выставляют счета в зависимости от типа и объема используемых ресурсов, а также количества времени, потраченного на их использование. Хотя GPU-процессоры, как правило, стоят дороже, чем CPU в расчете на единицу времени, в некоторых приложениях могут использоваться GPU-ускорители, чтобы обеспечивать достаточное сокращение времени выполнения по сравнению с расходами на CPU с целью снижения затрат.

1.1.2 Предостережения, связанные с параллельными вычислениями

Параллельные вычисления не являются панацеей. Многие приложения недостаточно велики или не требуют достаточного времени выполнения, чтобы нуждаться в параллельных вычислениях. У некоторых может даже не быть достаточного внутреннего параллелизма, который можно

было бы эксплуатировать. Кроме того, транзит приложений на использование мультиядерного и многоядерного оборудования (GPU) требует выделенных усилий, которые могут временно отвлекать внимание от прямых исследований или продуктовых целей. Сначала следует осознать целесообразность затрат времени и усилий. Всегда важнее обеспечить функционирование приложения и генерирование им желаемого результата, прежде чем его ускорять и масштабировать его вертикально до более крупных задач.

Мы настоятельно рекомендуем вам начать свой параллельно-вычислительный проект с плана. Важно знать варианты, которые доступны для ускорения приложения, а затем выбрать наиболее подходящий для вашего проекта. После этого крайне важно иметь разумную оценку затраченных усилий и потенциальных выгод (с точки зрения стоимости в долларах, энергопотребления, времени до решения и других показателей, которые могут представлять важность). В этой главе мы начнем давать вам авансом знания и навыки для принятия решений по параллельно-вычислительным проектам.

1.2 Фундаментальные законы параллельных вычислений

В последовательных вычислениях все операции ускоряются по мере увеличения тактовой частоты. В отличие от них в случае параллельных вычислений нам необходимо немного поразмыслить и модифицировать наши приложения так, чтобы задействовать параллельное оборудование в полной мере. Почему так важен объем параллелизма? Для того чтобы в этом разобраться, давайте взглянем на законы параллельных вычислений.

1.2.1 Предел на параллельные вычисления: закон Амдала

Нам необходим подход к расчету потенциального ускорения вычисления, основываясь на объеме параллельного кода. Это можно сделать, используя Закон Амдала, предложенный Джином Амдалом в 1967 году. Указанный закон описывает ускорение задачи фиксированного размера по мере увеличения числа процессоров. Следующее ниже уравнение это показывает, где P – это параллельная доля кода, S – последовательная доля, означающая, что $P + S = 1$, и N – это число процессоров:

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

Закон Амдала подчеркивает, что независимо от того, какой быстрой мы делаем параллельную часть кода, мы всегда будем лимитированы

последовательной частью. На рис. 1.3 показан этот предел. Такое масштабирование задачи фиксированного размера называется сильным масштабированием.

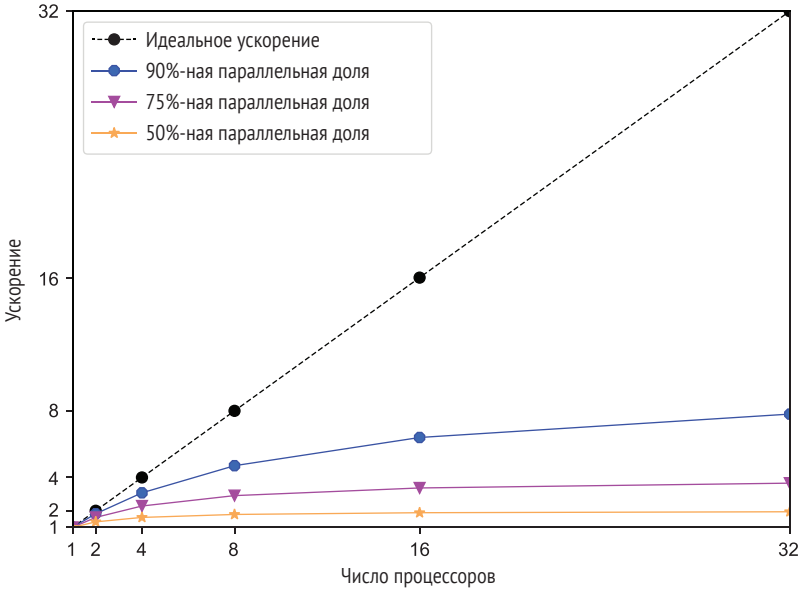


Рис. 1.3 Ускорение для задачи фиксированного размера в соответствии с законом Амдала показано как функция от числа процессоров. Линии показывают идеальное ускорение, когда параллелизуется 100 % алгоритма, а также для 90, 75 и 50 %. Закон Амдала гласит, что ускорение лимитировано долями кода, которые остаются последовательными

ОПРЕДЕЛЕНИЕ *Сильное масштабирование* представляет время до решения в зависимости от числа процессоров для задачи фиксированного суммарного размера.

1.2.2 Преодоление параллельного предела: закон Густафсона–Барсиса

В 1988 году Густафсон и Барсис указали на то, что прогоны параллельного кода должны увеличивать размер задачи по мере добавления большего числа процессоров. Это дает нам альтернативный подход к расчету потенциального ускорения нашего приложения. Если размер задачи растет пропорционально числу процессоров, то ускорение теперь выражается как

$$\text{Ускорение}(N) = N - S \times (N - 1),$$

где N – это число процессоров, а S – последовательная доля, как и раньше. Как следствие, более крупная задача может быть решена за одно и то

же время с использованием большего числа процессоров. Это предоставляет дополнительные возможности для привлечения параллелизма. И действительно увеличение размера задачи вместе с числом процессоров имеет смысл, потому что пользователь приложения хочет извлекать выгоду не только из мощности дополнительного процессора, но и хочет использовать дополнительную память. Масштабирование времени выполнения для этого сценария, показанное на рис. 1.4, называется слабым масштабированием.

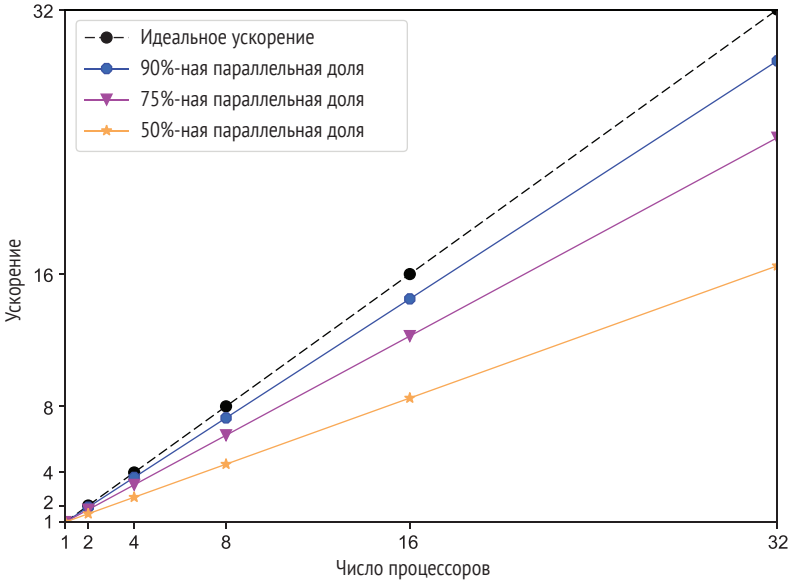


Рис. 1.4 Ускорение для случаев, когда размер задачи увеличивается вместе с увеличением числа доступных процессоров в соответствии с законом Густафсона–Барсиса, показано как функция числа процессоров. Линии показывают идеальное ускорение, когда параллелизуется 100 % алгоритма, а также для 90, 75 и 50 %

ОПРЕДЕЛЕНИЕ *Слабое масштабирование* представляет время до решения в зависимости от числа процессоров для задачи фиксированного размера в расчете на процессор.

На рис. 1.5 наглядно показана разница между сильным и слабым масштабированием. Аргументация слабого масштабирования в отношении того, что размер вычислительной сетки должен оставаться постоянным на каждом процессоре, позволяет эффективно пользоваться ресурсами дополнительного процессора. С позиций сильного масштабирования все внимание сконцентрировано в первую очередь на ускорении вычисления. На практике важны как сильное, так и слабое масштабирование, поскольку они решают разные пользовательские сценарии.

Термин «масштабируемость» часто используется для обозначения возможности добавлять больше параллелизма в аппаратное или про-

граммное обеспечение и существования совокупного предела возможного улучшения. В то время как традиционный фокус внимания лежит на масштабировании времени выполнения, мы выдвигаем аргумент, что масштабирование памяти часто имеет бóльшую значимость.

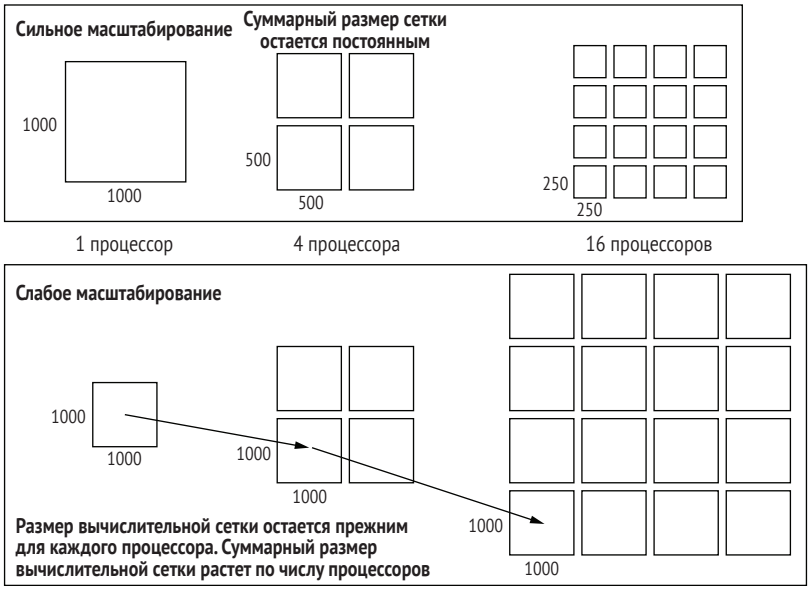


Рис. 1.5 Сильное масштабирование сохраняет тот же совокупный размер задачи и распределяет ее по дополнительным процессорам. При слабом масштабировании размер вычислительной сетки остается одинаковым для каждого процессора, а суммарный размер увеличивается

На рис. 1.6 показано приложение с лимитированной масштабируемостью памяти. *Реплицированный массив* (R) – это набор данных, который дублируется во всех процессорах. *Распределенный массив* (D) разбивается на разделы и распределяется между процессорами.

Например, в игровом симуляторе 100 персонажей могут быть распределены между 4 процессорами по 25 персонажей на каждом процессоре. Но карта игрового поля может быть скопирована на каждый процессор. На рис. 1.6 реплицированный массив дублируется по всей вычислительной сетке. Поскольку этот рисунок относится к слабому масштабированию, размер задачи увеличивается вместе с увеличением числа процессоров. Для 4 процессоров массив в 4 раза больше на каждом процессоре.

Поскольку размер задачи растет, вскоре на процессоре не хватит памяти для выполнения задания. Лимитированное масштабирование времени выполнения означает, что задание выполняется медленно; лимитированное масштабирование памяти означает, что задание не может выполняться вообще. Это также тот случай, что если память приложения может быть распределена, то время выполнения обычно масштабируется тоже. Однако обратное не обязательно верно.

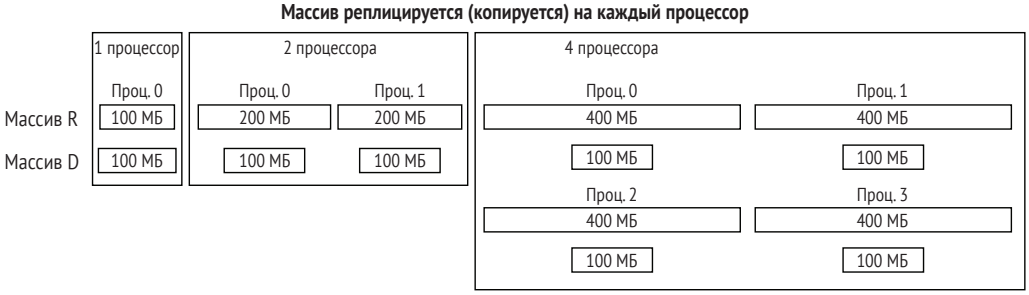


Рис. 1.6 Распределенные массивы остаются того же размера, что и задача, а число процессоров удваивается (слабое масштабирование). Но реплицированные (скопированные) массивы нуждаются во всех данных на каждом процессоре, при этом память быстро растет вместе с увеличением числа процессоров. Даже если время выполнения масштабируется слабо (остается постоянным), требования к памяти лимитируют масштабируемость

Одна из точек зрения на интенсивное вычислительное задание состоит в том, что каждый байт памяти затрагивается в каждом цикле обработки, а время выполнения является функцией от размера памяти. Сокращение размера памяти обязательно приведет к сокращению времени выполнения. Таким образом, первоначальное внимание в параллелизме должно быть направлено на сокращение размера памяти по мере увеличения числа процессоров.

1.3 Как работают параллельные вычисления?

Параллельные вычисления требуют сочетания понимания оборудования, программного обеспечения и параллелизма при разработке приложения. Это больше, чем просто передача сообщений или потокообразование. Современное оборудование и программное обеспечение предоставляют массу разных возможностей для параллелизации вашего приложения. Некоторые из этих возможностей могут быть скомбинированы, чтобы обеспечить еще большую эффективность и ускорение.

Важно иметь понимание параллелизации в вашем приложении и того, как разные аппаратные компоненты позволяют вам ее выявлять. Кроме того, разработчики должны понимать, что между вашим исходным кодом и оборудованием ваше приложение должно проходить дополнительные слои, включая компилятор и операционную систему (рис. 1.7).

Как разработчик, вы несете ответственность за слой прикладного программного обеспечения, в котором содержится ваш исходный код. В исходном коде вы выбираете язык программирования и параллельные программные интерфейсы, которые используете для задействования опорного оборудования. В дополнение вы принимаете решение о том, как разбить свою работу на параллельные модули. Компилятор служит для транслирования вашего исходного кода в форму, исполнимую вашим оборудованием. Получив в свое распоряжение эти команды,

операционная система управляет их исполнением на компьютерном оборудовании.

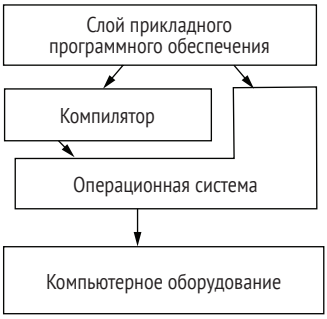


Рис. 1.7 Параллелизация выражается в слое прикладного программного обеспечения, который соотносится с компьютерным оборудованием через компилятор и OS

Мы покажем вам на примере процесс введения параллелизации в алгоритм с помощью прототипного приложения. Указанный процесс происходит в слое прикладного программного обеспечения, но требует понимания компьютерного оборудования. Пока что мы воздержимся от обсуждения выбора компилятора и операционной системы. Мы будем поступательно добавлять каждый слой параллелизации, чтобы у вас имелась возможность видеть принцип ее работы. В каждой параллельной стратегии мы объясним характер влияния доступного оборудования на сделанный выбор. Цель этого состоит в том, чтобы продемонстрировать то, как аппаратные функциональности влияют на параллельные стратегии. Мы классифицируем параллельные подходы, которые разработчик может принять, на:

- параллелизацию на основе процессов (process-based parallelization);
- параллелизацию на основе потоков (thread-based parallelization);
- векторизацию (vectorization);
- обработку в потоковом режиме (stream processing).

Следуя этому примеру, мы введем модель, которая поможет вам рассуждать о современном оборудовании. Эта модель разбивает современное вычислительное оборудование на отдельные компоненты и различные вычислительные устройства. В эту главу включен упрощенный взгляд на память. Более подробный обзор иерархии памяти представлен в главах 3 и 4. Наконец, мы подробнее обсудим слои приложения и программное обеспечение.

Как уже упоминалось, мы классифицируем параллельные подходы, которые разработчик может принять, на параллелизацию на основе процессов, параллелизацию на основе потоков (т. е. виртуальных ядер), векторизацию и обработку в потоковом режиме. Параллелизация на основе отдельных процессов с их собственными пространствами памяти может представлять собой распределенную память на разных узлах компьютера или внутри узла. Обработка в потоковом режиме обычно связана с GPU-процессорами. Модель современного оборудования и прикладного программного обеспечения поможет вам лучше понять принцип

планирования переноса вашего приложения на текущее параллельное оборудование.

1.3.1 Пошаговое ознакомление с примером приложения

В целях этого введения в параллелизацию мы рассмотрим подход на основе параллелизма данных. Это одна из наиболее распространенных стратегий применения параллельных вычислений. Мы выполним вычисления на пространственной вычислительной сетке, состоящей из регулярной двумерной (2D) решетки прямоугольных элементов или ячеек. Шаги (резюмированные здесь и подробно описанные позже) для создания пространственной расчетной сетки и подготовки к вычислению таковы:

1. дискретизировать (подразделить) задачу на более мелкие ячейки или элементы;
2. определить вычислительное ядро (операцию) для выполнения на каждом элементе вычислительной сетки;
3. добавить следующие слои параллелизации на CPU-процессорах и GPU-процессорах для выполнения расчета:
   - *векторизацию* – работать на более одной единице данных за один раз;
   - *потоки* – развертывать более одного вычислительного маршрута для привлечения большего числа процессорных ядер;
   - *процессы* – отделять программные экземпляры для распространения расчета по отдельным пространствам памяти;
   - *выгрузку расчета на GPU-процессоры* – отправлять данные в GPU для расчета.

Мы начинаем с двумерной задачной области участка пространства. Для целей иллюстрации мы будем использовать 2D-изображение вулкана Кракатау (рис. 1.8) в качестве примера. Целью наших расчетов может быть моделирование шлейфа вулканического выброса, возникающего в результате цунами или раннее обнаружение извержения вулкана с использованием машинного обучения. Для всех этих вариантов скорость вычислений имеет решающее значение, если мы хотим, чтобы реальновременные результаты информировали наши решения.



Рис. 1.8 Пример двухмерной пространственной области для численной симуляции. Численные симуляции обычно предусматривают стенсильные операции (см. рис. 1.11) или большие матрично-векторные системы. Эти типы операций часто используются при моделировании жидкостей для продуцирования предсказаний времени прибытия цунами, прогнозов погоды, распространения дымового шлейфа и других процессов, необходимых для принятия обоснованных решений