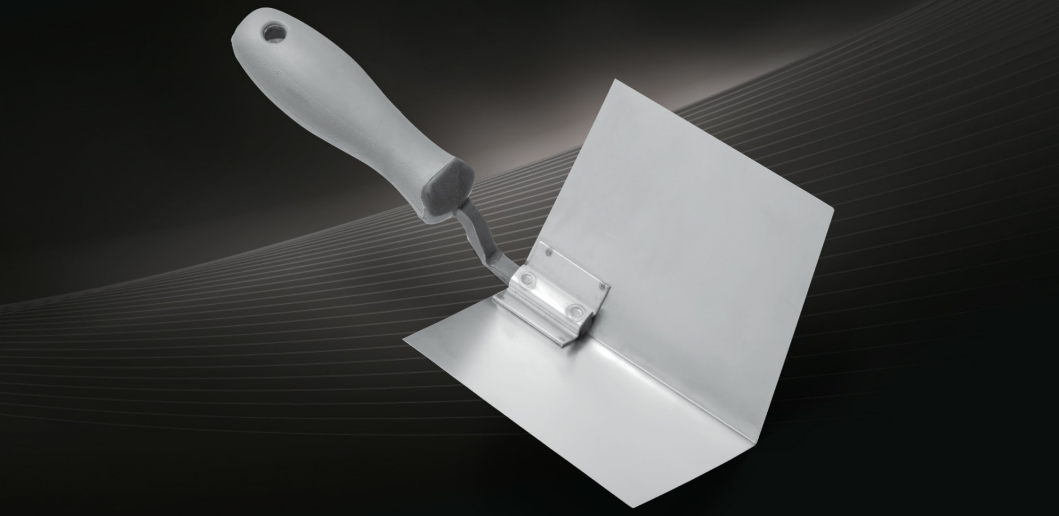
Microsoft

C++ AMP

Построение массивно параллельных программ с помощью Microsoft Visual C++



Кейт Грегори
Эйд Миллер

ДМК Пресс
издательство

Кэйт Грегори
Эйд Миллер

C++ AMP:

построение массивно параллельных программ с помощью Microsoft Visual C++

Microsoft®

C++ AMP: Accelerated Massive Parallelism with Microsoft® Visual C++®

Kate Gregory
Ade Miller

C++ AMP:

построение массивно параллельных программ с помощью Microsoft Visual C++

Кэйт Грегори
Эйд Миллер

2-е издание, электронное

ДМК ПРЕСС
издательство
Москва, 2023

УДК 004.438C++AMP
ББК 32.973.202-018.2
Г79

Грегори, Кэйт**.**
Г79 C++ AMP: построение массивно параллельных программ с помощью Microsoft Visual C++ / К. Грегори, Э. Миллер ; пер. с англ. А. А. Слинкина. — 2-е изд., эл. — 1 файл pdf : 413 с. — Москва : ДМК Пресс, 2023. — Систем. требования: Adobe Reader XI либо Adobe Digital Editions 4.5 ; экран 10". — Текст : электронный.

ISBN 978-5-89818-518-3

C++ Accelerated Massive Parallelism (C++ AMP) — разработанная корпорацией Microsoft технология ускорения написанных на C++ приложений за счет исполнения кода на оборудовании с распараллеливанием по данным, например, на графических процессорах. Модель программирования в C++ AMP основана на библиотеке, устроенной по образцу STL, и двух расширениях языка C++, интегрированных в компилятор Visual C++ 2012. Она в полной мере поддерживается инструментами Visual Studio, в том числе IntelliSense, отладчиком и профилировщиком. Благодаря C++ AMP свойственная гетерогенному оборудованию производительность становится доступна широким кругам программистов.

В книге показано, как воспользоваться всеми преимуществами C++ AMP в собственных приложениях. Помимо описания различных черт C++ AMP, приведены примеры различных подходов к реализации различных алгоритмов в реальных приложениях.

Издание предназначено для программистов, уже работающих на C++ и стремящихся повысить производительность существующих приложений.

УДК 004.438C++AMP
ББК 32.973.202-018.2

Электронное издание на основе печатного издания: C++ AMP: построение массивно параллельных программ с помощью Microsoft Visual C++ / К. Грегори, Э. Миллер ; пер. с англ. А. А. Слинкина. — Москва : ДМК Пресс, 2013. — 412 с. — ISBN 978-5-94074-896-0. — Текст : непосредственный.

ISBN 978-5-89818-518-3

Посвящается Брайану, который всегда был моим секретным оружием, и моим детям, теперь уже почти взрослым, которые считают, что мама, пишущая книги, – это нормально.

– Кэйт Грегори

Посвящается Сюзанне – той единственной, что в сто раз лучше, чем я заслуживаю.

– Эйд Миллер

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

На протяжении большей части истории развития компьютеров мы были свидетелями экспоненциального роста производительности скалярных процессоров. Но теперь этому пришел конец. Мы находимся в начале эры гетерогенных параллельных вычислений. В мире, где всем приложениям нужно больше вычислительных мощностей, а мощность всех вычислительных систем – от мобильных устройств до облачных кластеров – ограничена, будущие вычислительные платформы просто обязаны стать гетерогенными. Так, самые мощные в мире суперкомпьютеры все чаще строятся как кластеры на базе комбинации ЦП и ГП (графический процессор). И хотя программные интерфейсы первого поколения, такие как CUDA и OpenCL, позволили приступить к созданию новых библиотек и приложений для подобных систем, выявилась настоятельная необходимость в средствах, которые обеспечили бы гораздо более высокую продуктивность при разработке гетерогенного параллельного ПО.

Основная трудность заключается в том, что любой программный интерфейс, повышающий продуктивность в этой области, должен еще и предоставлять программисту средства, необходимые для достижения требуемой производительности. Интерфейс C++ AMP, предложенный Microsoft, – крупный шаг на пути решения этой проблемы. Это простое и элегантное расширение языка C++, призванное устранить два самых заметных недостатка прежних интерфейсов. Во-первых, прежние подходы плохо сочетаются с общепринятой практикой построения программ на C++, так как модели параллельного программирования на основе ядра трудно увязать с организацией классов в приложении. Во-вторых, унаследованная от C индексация динамически выделяемых массивов затрудняет управление локальностью.

Я с радостью обнаружил, что для решения первой проблемы C++ AMP поддерживает в параллельном коде циклические конструкции и объектно-ориентированные средства C++, а для решения второй – конструкцию *array_view*. Подход на основе *array_view* – это задел на

будущее, позволяющий уже сейчас создавать приложения, которые сумеют воспользоваться всеми преимуществами грядущих архитектур с объединенным адресным пространством. Многие программисты, имеющие опыт работы с CUDA и OpenCL, находят программирование в стиле C++ AMP интересным, элегантным и эффективным.

Не менее важен, на мой взгляд, и тот факт, что интерфейс C++ AMP открывает возможности для широкого распространения многочисленных новаторских способов преобразования программы компилятором, в частности, выбора размещения данных и гранулярности потоков. Он также позволяет реализовать оптимизацию перемещения данных во время выполнения. Благодаря таким усовершенствованиям продуктивность программиста резко возрастет.

В настоящее время интерфейс C++ AMP реализован только в Windows, но спецификация открыта и, скорее всего, будет реализована и на других платформах. Заложенный в C++ AMP потенциал раскроется по-настоящему, когда поставщики других платформ начнут предлагать его реализацию (если это произойдет).

Издание этой книги знаменует важный этап в развитии гетерогенных параллельных вычислений. Я ожидаю, что теперь количество разработчиков, способных продуктивно работать в такой среде, существенно увеличится. Я горжусь выпавшей мне честью написать предисловие к этой книге и принять участие в этом движении. И, что еще более важно, отдаю должное группе инженеров Microsoft, создавших C++ AMP и тем самым поспособствовавших движению в этом направлении.

Вэнь-Мей Ху
Профессор, кафедра имени Сандерса (AMD),
факультет электротехники и вычислительной техники
университета штата Иллинойс в Урбане и Шампейне,
технический директор компании MulticoreWare, Inc.

ОБ АВТОРАХ

Эйд Миллер в настоящее время работает главным системным архитектором в компании Microsoft Studios. Ранее занимал различные должности в Microsoft, в том числе являлся руководителем проекта по платформам «больших данных», над которым работал совместно с группой разработки Windows HPC Server. Был также руководителем команд гибкой разработки в группе «Patterns & Practices». Интересуется главным образом параллельными и распределенными вычислениями, а также методами улучшения качества разработки ПО за счет правильной организации работ.

Эйд – соавтор книг «Parallel Programming with Microsoft .NET» и «Parallel Programming with Microsoft Visual C++». Он много пишет и выступает на тему параллельных вычислений и своего опыта гибкой разработки ПО в Microsoft и других компаниях.

Кэйт Грегори программирует на C++ уже больше двадцати лет и хорошо известна как преподаватель, лектор и автор. Управление проектами, обучение, составление технической документации и выступления на различных мероприятиях отнимают большую часть ее времени, но тем не менее она умудряется писать код каждую неделю. Кэйт – автор более десятка книг, она регулярно выступает на конференциях DevTeach, TechEd (в США, Европе и Африке), TechDays и других. Кэйт удостоена звания C++ MVP, является спонсором-основателем группы пользователей .NET в Торонто, основателем группы пользователей в Восточном Торонто и преподает по временному контракту в университете Трент в Питерборо. С января 2002 года является региональным директором Microsoft в Торонто, а в январе 2004 года удостоена звания Microsoft Most Valuable Professional по Visual C++. В июне 2005 стала региональным директором года, а в феврале 2011 – Visual C++ MVP 2010 года. Ее компания, Gregory Consulting Limited, расположенная в сельской местности в районе озера Онтарио, помогает заказчикам осваивать новые технологии и адаптироваться к изменяющимся условиям ведения бизнеса.

ВВЕДЕНИЕ

C++ Accelerated Massive Parallelism (C++ AMP) – разработанная корпорацией Microsoft технология ускорения написанных на C++ приложений за счет исполнения кода на оборудовании с распараллеливанием по данным, например, на графических процессорах (ГП). Она рассчитана не только на современную параллельную аппаратуру в виде ГП и APU (Accelerated Processing Unit – ускоренный процессорный элемент), но и на поддержку будущего оборудования – с целью защитить вложения в разработку кода. Спецификация C++ AMP открыта. Microsoft реализовала ее поверх DirectX, обеспечив тем самым переносимость на различные аппаратные платформы. Но другие реализации могут опираться на иные технологии, поскольку в спецификации DirectX нигде не упоминается.

Модель программирования в C++ AMP основана на библиотеке, устроенной по образцу STL, и двух расширениях языка C++, интегрированных в компилятор Visual C++ 2012. Она в полной мере поддерживается инструментами Visual Studio, в том числе IntelliSense, отладчиком и профилировщиком. Благодаря C++ AMP свойственная гетерогенному оборудованию производительность становится доступна широким кругам программистов.

В этой книге показано, как воспользоваться всеми преимуществами C++ AMP в собственных приложениях. Помимо описания различных черт C++ AMP, приведены примеры различных подходов к реализации некоторых общеупотребительных алгоритмов в реальных приложениях. Полный код примеров, равно как и код, представленный в отдельных главах, можно скачать и изучить.

Для кого предназначена эта книга

Цель этой книги – помочь программистам на C++ в освоении технологии C++ AMP, от базовых концепций до более сложных средств. Если вас интересует, как воспользоваться преимуществами гетерогенного оборудования для повышения производительности существующих функций приложения или добавления совершенно новых,

которые раньше не удавалось реализовать из-за ограничений на быстродействие, то эта книга для вас.

Прочитав ее, вы будете лучше понимать, как и где лучше всего применить C++ в своем приложении. Вы узнаете, как работать с имеющимися в Microsoft Visual Studio 2012 средствами отладки и профилирования для поиска ошибок и оптимизации производительности.

Предполагаемые знания

Предполагается, что читатель знаком с разработкой ПО в среде Windows C++, с концепциями объектно-ориентированного программирования и со стандартной библиотекой C++ (которую часто называют STL по названию ее предтечи – библиотеки Standard Template Library). Знакомство с общими понятиями параллельной обработки было бы не лишним, но не обязательно. В некоторых примерах используется DirectX, но ни для их использования, ни для понимания входящего в них кода C++ AMP, опыт работы с DirectX не требуется.

Общие сведения о языке C++ можно получить из книги Бьярна Страуструпа «Язык программирования C++» (Бином, 2012). В настоящей книге используются многие языковые и библиотечные средства, появившиеся только в стандарте C++11 и пока еще не нашедшие отражения в печатных изданиях. Хороший обзор имеется в работе Скотта Мейерса «Presentation Materials: Overview of the New C++ (C++11)», которую можно приобрести через Интернет у издательства Artima Developer по адресу *http://www.artima.com/shop/overview_of_the_new_cpp*. Хорошее введение в стандартную библиотеку приведено в книге Nicolai M. Josuttis's The C++ Standard Library: A Tutorial and Reference (2nd Edition) (Addison-Wesley Professional, 2012).

В примерах также нередко используются библиотеки Parallel Patterns Library и Asynchronous Agents Library. Та и другая неплохо описаны в книге «Parallel Programming with Microsoft Visual C++» (Microsoft Press, 2011) Колина Кэмпбелла (Colin Campbell) и Эйда Миллера (Ade Miller). Бесплатная версия этой книги имеется также на сайте MSDN по адресу *http://msdn.microsoft.com/en-us/library/gg675934.aspx*.

Для кого не предназначена эта книга

Эта книга не предназначена для изучения языка C++ или его стандартной библиотеки. Предполагается, что читатель владеет тем и

другим на рабочем уровне. Книга также не является общим введением в параллельное или многопоточное программирование. Если вы не знакомы с этими темами, рекомендуем сначала прочитать книги, упомянутые в предыдущем разделе.

Организация материала

Книга состоит из 12 глав, каждая из которых посвящена одной из сторон программирования с помощью C++ AMP. Кроме того, в книгу включены три примера, демонстрирующих применение основных возможностей C++ AMP в реальных программах. Полный код примеров и фрагменты из других глав можно скачать с сайта CodePlex.

Глава 1 «Общие сведения и подход C++ AMP»	Введение в графические процессоры, гетерогенные вычисления, организацию параллелизма на ЦП. Обзор того, как C++ AMP позволяет задействовать всю мощь современных гетерогенных систем.
Глава 2 «Пример: программа NBody»	Моделирование задачи N тел с помощью C++ AMP
Глава 3 «Основы C++ AMP»	Краткое описание библиотеки и изменений в языке, составляющих C++ AMP, а также некоторых правил, которым должна следовать программа.
Глава 4 «Разбиение на блоки»	Рассматривается вопрос о разбиении вычисления на группы потоков, называемые блоками (tile), которые имеют общий доступ к сверхбыстрому программируемому кэшу.
Глава 5 «Пример: блочный вариант программы NBody»	Описывается вариант программы NBody из главы 2 с использованием разбиения на блоки.
Глава 6 «Отладка»	Обзор технических приемов и средств отладки приложений на базе C++ AMP в Visual Studio.
Глава 7 «Оптимизация»	Дополнительные сведения о факторах, которые влияют на производительность приложений на базе C++ AMP, и о том, как добиться максимального быстродействия.
Глава 8 «Пример: программа Reduction»	Демонстрируются различные подходы к реализации простого вычисления и их влияние на производительность.
Глава 9 «Работа с несколькими ускорителями»	Как использовать несколько ГП для достижения максимальной производительности. Рассматривается комбинированный параллелизм и применение ЦП для обеспечения максимального эффективного использования ГП.

Глава 10 «Пример: программа Cartoonizer»	Комплексный пример, в котором сочетается параллелизм на уровне ЦП с параллелизмом в духе C++ AMP и поддержкой нескольких ускорителей.
Глава 11 «Интероперабельность с графикой»	Использование C++ AMP в сочетании с DirectX.
Глава 12 «Советы, хитрости и рекомендации»	Описываются менее распространенные ситуации и среды и объясняется, как разрешать некоторые типичные проблемы.
Приложение «Другие ресурсы»	Онлайновые ресурсы, техническая поддержка и учебные курсы для желающих обогатить свои знания о C++ AMP.

Принятые соглашения

В этой книге применяются следующие соглашения.

- Текст, заключенный в рамочку и помеченный, например, словом «**Примечание**», содержит дополнительные сведения или описание альтернативных способов выполнить действие.
- Знак **+** между названиями двух клавиш, например **Alt+Tab**, означает, что эти клавиши нужно нажать одновременно.
- Вертикальная черта между двумя или более пунктами меню (например, **File | Close**), означает, что сначала нужно выбрать первое меню или пункт меню, затем следующее и т. д.

Требования к системе

Для сборки и запуска примеров из этой книги необходимо следующее оборудование и программное обеспечение.

- Операционная системе Microsoft Windows 7 с пакетом обновлений Service Pack 1 или Windows 8 (x86 или x64). Примеры должны также собираться и запускаться в системах Windows Server 2008 R2 (x64) и Windows Server 2012 (x64), но тестирование для них не производилось.
- Любое издание Visual Studio 2012 (для профилирования, описанного в главах 7 и 8, необходимо издание Professional или Ultimate).
- Для сборки программы NBody понадобится DirectX SDK (версия от июня 2010).

- Компьютер, оснащенный процессором с тактовой частотой 1,6 Ггц или выше. Рекомендуется четырехъядерный процессор.
- Оперативная память объемом не ниже 1 ГБ (для 32-разрядных ОС) или 2 ГБ (для 64-разрядных ОС).
- 10 ГБ свободного места на жестком диске (для установки Visual Studio 2012).
- Жесткий диск с частотой вращения 5400 об/мин.
- Видеокарта с поддержкой DirectX 11 (для примеров использования C++ AMP) и монитор с разрешением 1024×768 или выше (для Visual Studio 2012).
- Накопитель DVD-ROM (если Visual Studio 2012 устанавливается с DVD-диска).
- Соединение с Интернетом для скачивания ПО и примеров.

Примеры кода

Почти во всех главах имеются примеры, позволяющие интерактивно осваивать изложенный в тексте новый материал. Исходный код примеров можно скачать в виде ZIP-файла со страницы *http://go.microsoft.com/FWLink/?Linkid=260980*.

Примечание. Помимо примеров кода, вы должны установить Visual Studio 2012 и DirectX SDK (версия от июня 2010). Устанавливайте последнюю доступную версию каждого продукта.

Установка примеров кода

Для установки примеров кода выполните следующие действия.

1. Скачайте ZIP-файл с исходным кодом со страницы книги на сайте CodePlex: *http://ampbook.codeplex.com/*. Последняя рекомендуемая версия находится на вкладке Downloads.
2. Прочитайте лицензионное соглашение с конечным пользователем (если будет предложено). Если вы согласны с его условиями, отметьте флажок Accept и нажмите кнопку **Next**.
3. Распакуйте архив в любую папку и откройте файл BookSamples.sln в Visual Studio 2012.

Примечание. Если лицензионное соглашение не появилось, с ним можно ознакомиться по адресу *http://ampbook.codeplex.com/license*. Копия соглашения включена также в архив с исходным кодом.

Использование примеров кода

После распаковки архива создается папка Samples, содержащая три подпапки.

- **CaseStudies:** содержит все три примера из глав 2, 8 и 10. Каждый пример находится в отдельной папке.
  - **NBody:** гравитационная модель для задачи N тел.
  - **Reduction:** несколько реализаций алгоритма редукции с целью демонстрации различных подходов к повышению производительности.
  - **Cartoonizer:** приложение для обработки изображений, которое мультиплицирует изображения, загруженные с диска или снятые видеокамерой.
- **Chapter 4, 7, 9, 11, 12:** содержат код примеров из соответствующей главы.
- **ShowAmpDevices:** простая утилита для вывода списка всех поддерживающих C++ AMP устройств в данном компьютере.

Папка верхнего уровня Samples содержит файл решения для Visual Studio 2012, BookSamples.sln, включающий все перечисленные выше проекты. Решение должно компилироваться без ошибок и предупреждений в конфигурациях Debug и Release для платформ Win32 и x64. Для каждого проекта имеется также собственный файл решения – на случай, если вы захотите загружать их по отдельности.

Благодарности

Любая книга – плод усилий не одного человека. У этой книги два автора, но у нас было еще и много помощников. Мы выражаем благодарность группе C++ AMP в Microsoft, сотрудники которой далеко не ограничивались рецензированием черновиков и ответами на многочисленные вопросы: Амиту Агарвалу (Amit Agarwal), Дэвиду Кэллахану (David Callahan), Чарльзу Фу (Charles Fu), Джерри Хиггинсу (Jerry Higgins), Йосси Леванони (Yossi Levanoni), Дону Маккреди (Don McCrady), Лукашу Мендакевичу (Łukasz Mendakiewicz), Дэниэлу Моту (Daniel Moth), Бхарату Майсору Нанджундаппа (Bharath Mysore Nanjundappa), Пуджа Нагпалу (Pooja Nagpal), Джеймсу Рэппу (James Rapp), Саймону Выбрански (Simon Wybranski), Линь Ли Чжану (Lingli Zhang) и Вэй Рон Чжу (Weirong Zhu) (корпорация Microsoft).

У группы C++ AMP также имеется блог, на котором представлены поистине бесценные материалы. Многие из перечисленных выше рецензентов пишут статьи в этом блоге. Кроме того, особый интерес вызвали у нас статьи следующих авторов: Стив Дэйц (Steve Deitz), Кэвин Гао (Kevin Gao), Паван Кумар (Pavan Kumar), Пол Мэйби (Paul Maybee), Джо Мэйо (Joe Mayo), Игорь Островский (Igor Ostrovsky) (корпорация Microsoft).

Эд Эсси (Ed Essey) и Дэниэл Мот стояли у истоков проекта и подали издательству O'Reilly и авторам идею написать книгу о C++ AMP. Они же координировали взаимодействие с группой разработки C++ AMP.

Выражаем также благодарность Расселлу Джонсу (Russell Jones), Холли Бауэр (Holly Bauer) и Кэрол Уитни (Carol Whitney), которые отвечали за редактирование и производство книги, а также Ребекке Демарест, художнику.

Мы также считаем большой удачей возможность публиковать черновые варианты книги на сайте Safari благодаря программе Rough Cuts издательства O'Reilly. Свое мнение о них высказали многие читатели. Мы благодарны им за потраченное время и проявленный интерес. Особенно полезны были замечания Бруно Букара (Bruno Boucard) и Вейкко Эева (Veikko Eeva).

Замеченные опечатки и поддержка книги

Мы приложили все усилия, чтобы в книге и сопроводительных материалах не было ошибок. Опечатки и ошибки, обнаруженные после выхода книги из печати, публикуются на сайте oreilly.com по адресу:

http://go.microsoft.com/FWLink/?Linkid=260979

Здесь же есть возможность сообщить нам о других ошибках.

Для получения дополнительной поддержки отправьте сообщение электронной почты в отдел поддержки книг издательства Microsoft Press по адресу *mspinput@microsoft.com*.

Просьба иметь в виду, что эти адреса не предназначены для поддержки программных продуктов Microsoft.

Нам важно ваше мнение

Удовлетворение читательских запросов – важнейший приоритет издательства Microsoft Press, и ваши отклики – главное наше достояние. Оставьте свое мнение об этой книге на странице

http://www.microsoft.com/learning/booksurvey

Заполнение анкеты не займет у вас много времени и можете быть уверены, что мы внимательно изучим все ваши замечания и предложения. Заранее благодарны!

Оставайтесь на связи

Продолжим беседу! Наш адрес на Твиттере:

http://twitter.com/MicrosoftPress.

ГЛАВА 1. Общие сведения и подход C++ AMP

В этой главе:

- Что означает GPGPU? Что такое гетерогенные вычисления?
- Технологии распараллеливания вычислений на ЦП.
- Подход C++ AMP.

Что означает GPGPU? Что такое гетерогенные вычисления?

Разработчикам ПО привычно приспосабливаться к изменяющемуся миру. У нашей индустрии изменение мира стало уже почти рутиной. Мы изучаем новые языки, осваиваем новые методологии, начинаем использовать новые парадигмы человеко-машинного интерфейса и считаем само собой разумеющимся, что программу всегда можно улучшить. Когда на некотором пути мы упираемся в стену и уже не можем сделать версию n+1 лучше версии n, мы находим другой путь. Последним из таких путей, на который готовы встать некоторые разработчики, являются гетерогенные вычисления.

В этой главе мы рассмотрим, что делалось для повышения производительности раньше; это поможет понять, в какую стену мы уперлись сейчас. Вы узнаете об основных различиях между ЦП и ГП, двумя потенциальными компонентами гетерогенного решения, и о том, какие задачи поддаются ускорению с помощью этих приемов распаралле-

ливания. Затем мы рассмотрим применяемые ныне виды параллелизма на ЦП и ГП и познакомимся с концепциями технологии C++ AMP, подготовив почву для детального изучения в последующих главах.

История роста производительности

В середине 70-х годов прошлого века компьютеры, находящиеся в распоряжении одного человека, были в диковинку. Термин «персональный компьютер» появился в 1975 году. За прошедшие с тех пор десятилетия идея компьютера на каждом рабочем столе перестала восприниматься как амбициозная и, быть может, недостижимая цель, а превратилась в обыденность. Теперь на многих столах стоит даже *несколько* компьютеров, да не только в кабинетах, а и в гостиных. Многие носят в кармане смартфоны – тоже компьютеры, хоть и совсем маленькие. За первые 30 лет экстенсивного роста компьютеры не только стали дешевле и популярнее, но и быстрее. Каждый год производители выпускали кристаллы со все более высокой тактовой частотой, с большим объемом кэш-памяти и, как следствие, более производительные. У разработчиков вошло в привычку добавлять в программы всё новые и новые функции. И если из-за этого программа начинала работать медленнее, то разработчики не особо переживали; все равно через полгода-год появятся более быстрые машины, и программа снова станет «шустрой» и отзывчивой. Это было время так называемых «бесплатных завтраков», когда наращивание функциональности программ обеспечивалось повышением производительности оборудования. В конечном итоге производительность порядка гигафлопс – миллиардов операций над числами с плавающей точкой в секунду – стала досягаемой и экономически доступной.

К сожалению, примерно в 2005 году «бесплатные завтраки» кончились. Производители продолжали увеличивать количество транзисторов на одном кристалле, но физические ограничения – в частности, тепловыделение кристалла – уже не дают повышать тактовую частоту. Но рынок – как всегда – требовал всё более и более мощных компьютеров. Для удовлетворения спроса производители стали выпускать многоядерные машины – с двумя, четырьмя и более процессорами. Когда-то цель «каждому пользователю по процессору» считалась труднодостижимой, но по завершении эры бесплатных завтраков пользователям стало недостаточно одноядерного компьютера – сначала настольного, потом – ноутбука, а теперь уже и смартфона. В последние пять-шесть лет стало обычным делом иметь параллельный

суперкомпьютер на каждом рабочем столе, в каждой гостиной и в каждом кармане.

Но одно лишь добавление процессорных ядер ничего не ускоряет. Программы можно грубо разделить на две большие группы: поддерживающие и не поддерживающие параллелизм. Программа без поддержки параллелизма обычно задействует лишь половину, четверть или одну восьмую часть имеющихся ядер. Она ютится на единственном ядре, упуская возможность ускорить работу, когда пользователь приобретает новую машину с большим числом ядер. Разработчики же, научившиеся писать программы, работающие тем быстрее, чем больше имеется процессорных ядер, могут обеспечить почти линейный прирост быстродействия – вдвое на двухъядерных машинах, вчетверо на четырехъядерных и т. д. Информированные потребители недоумевают, почему некоторые разработчики игнорируют дополнительные возможности повысить производительность программ.

Гетерогенные платформы

В те же пять-шесть лет, на которые пришелся расцвет многоядерных компьютеров с несколькими процессорами, не стояли на месте и производители графических карт. Но вместо двух или четырех ядер, как в ЦП, графические процессоры (ГП) оснащались десятками, а то и сотнями ядер. Эти ядра сильно отличаются от имеющихся в ЦП. Первоначально они разрабатывались для повышения скорости вычислений, специфичных для машинной графики, например для определения цвета конкретного пикселя на экране. ГП справляется с этой работой гораздо быстрее ЦП, а поскольку на современной графической карте графических процессоров так много, открывается возможность массивного параллелизма. Разумеется, очень быстро возникло непреодолимое желание приспособить ГП для численных расчетов, *не относящихся* к графике. Машина, оснащенная многоядерными ЦП и ГП, на одном или на разных кристаллах, или даже кластер подобных машин называется гетерогенным суперкомпьютером. Очевидно, очень скоро гетерогенные суперкомпьютеры окажутся на каждом рабочем столе, в каждой гостиной и в каждом кармане.

В начале 2012 года типичный ЦП имел четыре ядра с двойной гиперпоточностью и насчитывал примерно миллиард транзисторов. Компьютеры высшего класса могут достигать при вычислениях с двойной точностью пиковой производительности 0,1 терафлопс (100 гигафлопс). Типичный ГП в начале 2012 года имел 32 ядра с 32 ни-

тями[1] в каждом и примерно вдвое больше транзисторов, чем ЦП. ГП высшего класса могут достигать при вычислениях с одинарной точностью производительности 30 терафлопс – в 30 раз больше пиковой производительности ЦП.

Примечание. Одни ГП поддерживают вычисления с двойной точностью, другие – нет, но данные о производительности обычно приводятся для вычислений с одинарной точностью.

Причина такой высокой производительности ГП не в количестве транзисторов или даже ядер. Дело в пропускной способности памяти – у ЦП она составляет около 20 гигабайт в секунду (ГБ/с), а у ГП – 150 ГБ/с. ЦП рассчитан на исполнение кода общего вида – с многозадачностью, вводом/выводом, виртуализацией, многоуровневым вычислительным конвейером и произвольной выборкой. Напротив, ГП проектируются для исполнения кода обработки графических данных, с распараллеливанием по данным, оснащаются программируемыми процессорами с фиксированными функциями, одноуровневым вычислительным конвейером и ориентированы на последовательную выборку. На самом деле, сверхвысокой производительности ГП достигают только на тех задачах, на которые рассчитаны, а не на задачах общего вида. У ГП есть еще одна особенность, быть может, даже более важная, чем быстродействие, – низкое энергопотребление. Для ЦП характерно энергопотребление порядка 1 Гфлопс/вт, а для ГП – примерно 10 Гфлопс/вт.

Во многих приложениях мощность, необходимая для выполнения конкретного вычисления, важнее затрачиваемого времени. Например, портативные устройства –смартфоны и ноутбуки – работают от аккумулятора, поэтому пользователи зачастую выбирают не самые быстрые, а самые энергетически экономичные приложения. Это существенно и для ноутбуков, пользователи которых ожидают, что при эксплуатации приложений, выполняющих большой объем вычислений, заряда аккумулятора хватит на целый день. Становится нормой ожидать нескольких ЦП – и ГП тоже – даже на таких небольших устройствах, как смартфоны. Некоторые устройства умеют включать и отключать питание отдельных ядер для продления срока работы от аккумулятора. В таких условиях перенос части вычислений на ГП может означать разницу между «приложением, которое невозможно

1 В программировании общего назначения слово thread обычно переводится как «поток», но в контексте программирования ГП чаще употребляется термин «нить» во избежание конфликтов со словом stream. В дальнейшем мы будем придерживаться именно этой терминологии. *Прим. перев.*

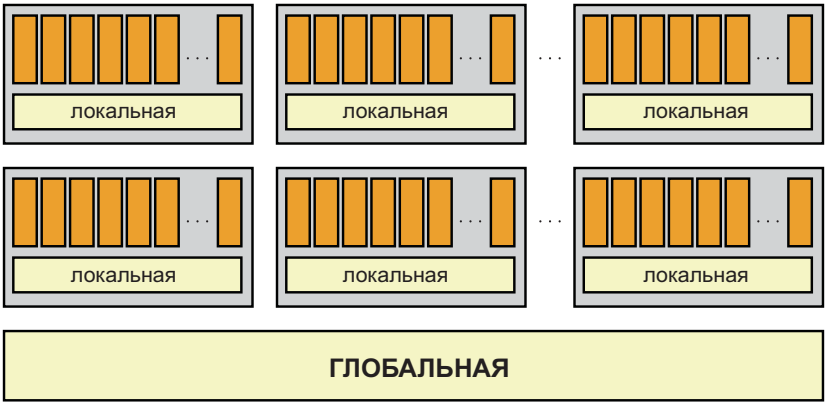
использовать вне офиса, потому что “жрет” батарейку» и «приложением, без которого я жизни себе не представляю». На другом конце спектра находятся центры обработки данных, для которых значительная доля эксплуатационных затрат приходится на оплату энергоснабжения. Двадцатипроцентная экономия энергии при выполнении сложного расчета в ЦОД или в облаке напрямую отражается на счете за электроэнергию.

Следует также учитывать доступ к памяти со стороны процессорных ядер. С точки зрения быстродействия, размер кэш-памяти может оказаться важнее тактовой частоты, поэтому ЦП оснащаются большими кэшами, чтобы у процессора всегда были данные для обработки и ядру как можно реже приходилось ждать завершения выборки данных из памяти. Для ЦП характерно повторное обращение к одним и тем же данным, что и позволяет получить от кэширования реальный выигрыш. Напротив, у типичного ГП кэш-память невелика, зато имеется много нитей, часть из которых всегда готова к работе. ГП может осуществлять предвыборку данных для компенсации задержки памяти, но поскольку данные, скорее всего, будут использоваться однократно, кэширование дает меньший выигрыш и не так необходимо. Чтобы от такого подхода была польза, в идеале должно быть очень много данных, над которыми производится относительно простое вычисление.

Но, пожалуй, самое важное различие заключается в технологии программирования. Для программирования ЦП существует много давно и прочно зарекомендовавших себя языков и инструментов. С точки зрения выразительной мощи и производительности, C++ стоит на первом месте, предлагая разработчику абстракции и развитые библиотеки, но не отнимая возможность низкоуровневого контроля. Выбор средств универсального программирования ГП (general-purpose GPU programming – GPGPU) куда более ограничен, и в большинстве случаев подразумевает нишевую или экзотическую модель программирования. Из-за этого ограничения лишь в немногих областях и задачах удавалось в полной мере задействовать способность ГП к «перемалыванию чисел». По той же причине большая часть разработчиков не стремилась изучать взаимодействие с ГП. Но разработчикам необходим способ повысить быстродействие приложений или сократить энергопотребление в конкретном вычислении. Сегодня таким способом может стать использование ГП. Идеальным было бы такое решение, которое позволило бы воспользоваться преимуществами ГП сегодня и другими формами гетерогенных вычислений – в будущем.

Архитектура ГП

Выше уже отмечалось, что ГП оснащен одноуровневым вычислительным конвейером, небольшим кэшем и большим числом нитей, выполняющих последовательную выборку из памяти. Нити не являются независимыми, а объединены в группы. В продуктах компании NVIDIA эти группы называются *канатами* (warps), а в продуктах AMD – *волновыми фронтами* (wavefront). В этой книге мы будем употреблять термин «канат». Канаты работают совместно, могут сообща обращаться к одной области памяти и взаимодействовать между собой. Для чтения локальной памяти требуется всего четыре такта, а для чтения более объемной (до 4 Гб) глобальной памяти – от 400 до 600 тактов. Когда одна группа нитей блокирована в ожидании результатов чтения, может исполняться другая группа. ГП способен очень быстро переключаться между группами нитей. Доступ к памяти организован таким образом, что чтение производится гораздо быстрее, когда соседние нити обращаются к соседним ячейкам. Если же отдельные нити в группе обращаются к ячейкам памяти, далеко отстоящим от тех, которые читают остальные нити в той же группе, то производительность резко падает.



Архитектуры ЦП и ГП существенно различаются. Программисты, работающие на языках высокого уровня, обычно не задумываются об архитектуре ЦП. Компоненты низкого уровня, операционные системы и оптимизирующие компиляторы обязаны принимать во внимание эти факторы, но при этом они ограждают «обычные» приложения от аппаратных деталей. Рекомендации и эвристические правила, которые вам, возможно, кажутся очевидными, иногда вовсе не являются