УДК 004.4
ББК **32.973.202**
Г84

Гримм Р.
Г84 Параллельное программирование на современном языке C++ / пер. с англ. В. Ю. Винника. – М.: ДМК Пресс, 2022. – 618 с.: ил.

ISBN 978-5-97060-957-6

В этой книге рассказывается о подробностях параллельного программирования на современном языке C++, а также приводятся многочисленные примеры кода.

Для тех, кто незнаком с параллельным программированием, дается краткий обзор темы; читатели с опытом работы могут сразу переходить к подробностям: изучить модель памяти, управление потоками, параллельные алгоритмы, сопрограммы в стандарте C++ 20. Глава «Учебные примеры» поможет читателю применить изученную теорию на практике. Дополнительно рассказывается о некоторых новшествах, запланированных в стандарт C++ 23.

Для тех, кто хочет освоить параллельное программирование на одном из наиболее распространенных языков.

УДК 004.4
ББК 32.973.202

ISBN 978-5-97060-957-6 (рус.)

Дизайн обложки разработан с использованием ресурса freepik.com.

Содержание

Отзывы читателей

Барт Вандевустейн

Старший разработчик в компании Esterline

Книга «Параллельное программирование на современном языке C++» – это практический путеводитель, который познакомит вас с параллельным программированием на современном языке C++. Начинающаяся с моделей памяти, содержащая множество готовых к запуску примеров кода, эта книга охватывает большую часть того, что нужно изучить, овладеть навыками параллельного программирования. Помимо поучительных учебных примеров, которые введут читателя в курс дела, обзор вероятных будущих нововведений подогреет ваш интерес ещё более.

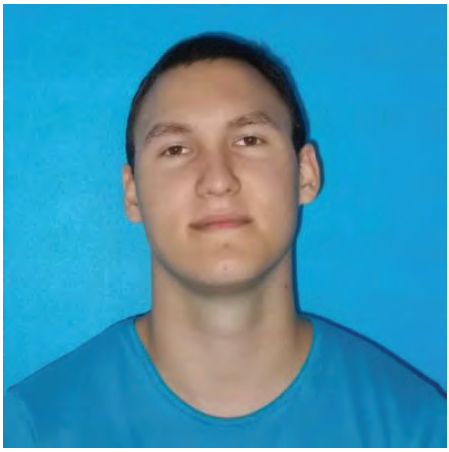
Иан Рив

Старший инженер-программист
по системам хранения данных в компании Dell

«Параллельное программирование на современном языке C++» Райнера Гримма – это хорошо написанная книга, охватывающая теоретические и практические аспекты разработки параллельных программ средствами

имеющихся стандартов языка C++, а также затрагивающая потенциальные изменения из будущего стандарта C++ 20[1]. Автор в подробностях обсуждает применения и эмпирические правила параллельных средств, сопровождая это примерами кода, закрепляющими понимание каждой темы. Информативная и заслуживающая внимания книга!

Роберт Бадеа



Технический руководитель команды

«Параллельное программирование на современном языке C++» – это простейший способ стать знатоком мира многопоточной разработки. В книге излагаются как простые, так и углублённые темы, в ней есть всё, что нужно разработчику, чтобы стать профессионалом в этой области: много содержательного материала, многочисленные готовые к запуску примеры кода вместе с превосходными пояснениями, а также целая глава о подводных камнях. Я наслаждался чтением этой книги и рекомендую её каждому, кто работает с языком C++.

[1] На момент выхода книги стандарт C++ 20 перестал быть будущим. – *Прим. перев.*

Введение

«Параллельное программирование на современном языке C++» – это путешествие по нынешним и будущим средствам параллельного программирования в языке C++.

- Стандарты C++ 11 и C++ 14 предоставляют основные строительные блоки для создания многопоточных и асинхронных программ.
- В стандартной библиотеке C++ 17 появились параллельные алгоритмы. Теперь большинство алгоритмов из стандартной библиотеки можно выполнять последовательным, параллельным или векторизированным образом.
- Развитие средств параллельного программирования на этом не останавливается. В стандарт C++ 20 вошли сопрограммы, а в будущем стандарте C++ 23 можно ожидать поддержку транзакционной памяти, расширенные фьючерсы и другие полезные новшества.

В этой книге рассказывается о подробностях параллельного программирования на современном языке C++, а также приводятся многочисленные примеры кода. Поэтому читатель может сочетать теорию с практикой, чтобы от обеих получить максимум знаний.

Поскольку книга посвящена параллельному программированию, в ней хочется рассказать также и о многочисленных подводных камнях – а ещё о том, как их обойти.

Соглашения

Соглашений будет немного.

Выделение шрифтом

Курсив используется, чтобы выделить важную мысль. **Жирный** шрифт служит для выделения ещё более важных мыслей. `Моноширинным` шрифтом набраны фрагменты кода, команды, ключевые слова языка программирования, имена переменных, типов, функций, классов.

Особые символы

Стрелкой $\Rightarrow$ обозначается логическое следование в математическом смысле: выражение $a \Rightarrow b$ означает «если a, то b».

Особые блоки текста

В особые блоки вынесены советы, предупреждения и важная информация.

Совет. В таких блоках помещены советы и дополнительная информация к материалу главы.

Предупреждение. В таком блоке размещены предупреждения, помогающие избегать ошибки.

Краткие итоги. В таких блоках, размещённых в конце основных глав, кратко повторяются основные положения для лучшего запоминания.

Исходный код

Все примеры исходного кода полны. Это означает, что читатель, имея подходящий компилятор, может компилировать и запускать их. Имя исходного файла показано в комментарии в первой строке листинга. Директива `using namespace std` используется иногда в случае крайней необходимости.

Запуск программ

Компиляция и запуск примеров, относящихся к стандартам C++ 11 и C++ 14, происходят проще всего. Любой современный компилятор языка C++ поддерживает эти стандарты. Компиляторам GCC[1] и clang[2] версию стандарта и требование подключить многопоточную библиотеку нужно передавать в параметрах командной строки. Например, чтобы с помощью компилятора g++ из пакета GCC создать исполняемую программу с именем thread, нужно подать такую команду:

```
g++ -std=c++14 -pthread thread.cpp -o thread
```

Назначение параметров таково:

- `-std=c++14` – использовать стандарт C++ 14;
- `-pthread` – подключить библиотеку pthread для поддержки потоков;
- `thread.cpp` – имя исходного файла;
- `-o thread` – имя исполняемого файла, который должен быть построен.

Таким же образом передаются параметры и компилятору clang. Компилятор, входящий в состав среды Microsoft Visual Studio 17, также поддерживает стандарт C++ 14.

Если современного компилятора под рукой нет, существует множество интерактивных сайтов, позволяющих компилировать программы удалённо. Заметка в блоге Арне Мертца содержит прекрасный обзор таких систем[3].

---

[1] https://gcc.gnu.org/.

[2] https://clang.llvm.org/.

[3] https://arne-mertz.de/2017/05/online-compilers/.

Со стандартами C++ 17, 20 и 23 дело обстоит сложнее[1]. Автор инсталлировал систему HPX[2] (High Performance ParallelX – высокопроизводительные параллельные вычисления) – систему с широким набором возможностей для разработки на языке C++ распределённых приложений любого масштаба. В системе HPX уже реализована параллельная стандартная библиотека из стандарта C++ 17, а также многие относящиеся к параллельному программированию нововведения стандартов C++ 20 и 23.

Как читать эту книгу

Читателям, не имеющим сколько-нибудь серьёзных знаний параллельного программирования, рекомендуется начать с начала – с части «Краткий обзор», чтобы составить общее представление о предмете.

Имея общую картину, можно переходить к части «Параллельное программирование в подробностях». Главу о моделях памяти можно пропустить при первом прочтении книги – кроме случаев, когда именно эта тема и нужна читателю. Глава «Учебные примеры» поможет читателю применить изученную теорию на практике. Некоторые примеры довольно сложны и требуют хорошего понимания моделей памяти.

Глава «Будущее языка C++» не обязательна для изучения. С другой стороны, заглядывать в будущее – это так увлекательно!

Последняя часть, озаглавленная «Дополнительные сведения», содержит разбор вопросов, позволяющих лучше понять основной материал книги и извлечь из неё как можно больше пользы.

Личные замечания

Благодарности

Впервые поделившись в своём англоязычном блоге www.ModernesCpp.com намерением написать эту книгу[3], автор встретил гораздо больший отклик, чем мог ожидать. Около пятидесяти человек изъявили желание ознакомиться с предварительным вариантом книги. Автор благодарен всем этим коллегам, включая дочь Юлитте, которая помогла с вёрсткой, и сына Мариуса, ставшего первым, кому довелось вычитывать текст.

---

[1] На момент выхода этого перевода компилятор MSVC и идущая с ним в комплекте библиотека поддерживают стандарт C++ 20 полностью, а в компиляторах clang и GCC с соответствующими библиотеками не хватает отдельных второстепенных элементов этого стандарта. – *Прим. перев.*

[2] http://stellar.cct.lsu.edu/projects/hpx/.

[3] http://www.modernescpp.com/index.php/looking-for-proofreaders-for-my-new-book-concurrency-with-modern-c.

В алфавитном порядке фамилий: Никос Атанасиу, Роберт Бадеа, Дафидд Вальтерс, Барт Вандевустейн, Анджей Варжинский, Вадим Винник, Джо Дас, Йонас Девлигере, Лассе Натвиг, Эрик Ньютон, Иан Рив, Ранди Хорманн, Энрико Цшемиш.

Автор о себе

Я работал архитектором программного обеспечения, руководителем команды разработчиков и инструктором по программированию на протяжении двух десятилетий. Люблю писать статьи о языках программирования C++, Python, Haskell, а в свободное время – выступать на конференциях. В 2016 году я решил работать на себя и с тех пор занимаюсь организацией и проведением семинаров по современным языкам C++ и Python.

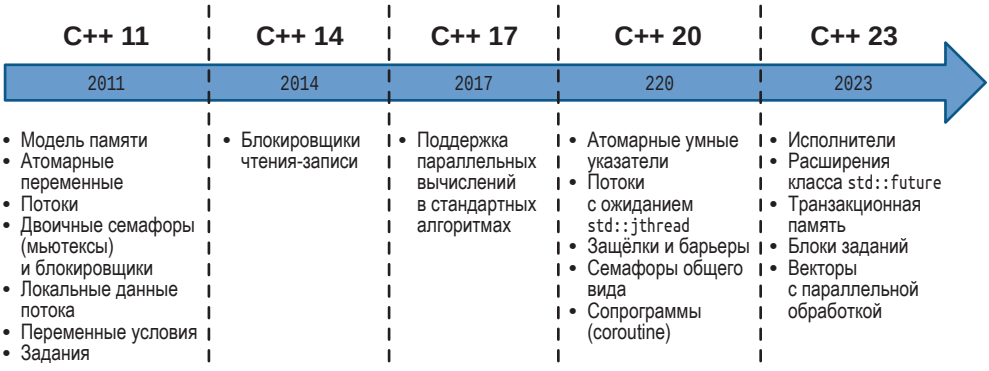
Необычная история возникновения книги

Эту книгу я начинал писать в Оберстдорфе параллельно с заменой тазобедренного сустава. Строго говоря, весь левый тазобедренный сустав заменялся эндопротезом. Первую половину книги я написал, лёжа в клинике и затем проходя реабилитацию. Говоря откровенно, написание книги очень помогло мне в этот сложный период.

Краткий обзор

1. Параллельное программирование и современный язык C++

С выходом стандарта C++ 11 язык получил библиотеку для многопоточного программирования и подходящую модель памяти. Эта библиотека содержит такие строительные блоки, как атомарные переменные, классы для потоков, двоичных семафоров и переменных условия. Они составляют фундамент, на котором в будущих версиях стандарта – например, C++ 20 и C++ 23 – станет возможным определить абстракции более высокого уровня. Однако и в стандарте C++ 11 уже присутствует понятие задания, которое обеспечивает более высокий уровень абстрагирования по сравнению с перечисленными базовыми строительными блоками.



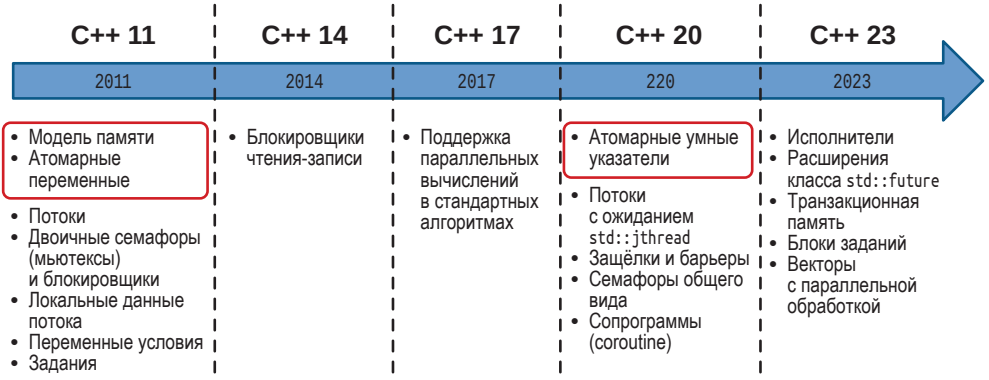
Средства параллельного программирования в языке C++

С некоторым огрублением историю поддержки параллельных вычислений в языке C++ можно разделить на три периода, которые кратко описаны в следующих разделах.

1.1. Стандарты C++ 11 и C++ 14: закладка фундамента

Стандартом C++ 11 поддержка многопоточного программирования впервые добавлена в язык. Она состоит из чётко определённой модели памяти и интерфейса для программирования потоков. С выходом стандарта C++ 14 к этому набору добавлены блокировщики чтения-записи.

1.1.1. Модели памяти



Модели памяти в языке C++

В основе многопоточного программирования лежит чётко определённая *модель памяти*. В ней должны быть отражены следующие аспекты:

- Атомарные операции – операции, выполнение которых не может быть прервано до их полного завершения.
- Частичное упорядочение операций – последовательности операций, порядок выполнения которых не должен нарушаться.
- Видимые эффекты операций – гарантия того, что результат операций над переменными, находящимися в общем доступе, будет заметен из других потоков.

Модель памяти, разработанная для языка C++, испытала заметное влияние предшественника – модели памяти для языка Java. В отличие от последней, однако, язык C++ позволяет снимать требование *временной согласованности* операций, по умолчанию накладываемое на атомарные операции.

Временная согласованность состоит из двух гарантий:

1. Инструкции программы выполняются в том порядке, в котором они записаны в её исходном коде.

2. Существует глобальный порядок выполнения всех операций во всех параллельных потоках.

Понятие модели памяти, в свою очередь, основывается на понятии атомарной операции, выполняемой над данными атомарных типов – для краткости их называют атомарными переменными.

1.1.1.1. Атомарные переменные

Язык C++ содержит набор простых *атомарных типов данных*. К ним относятся логические, символьные, числовые типы и типы указателей, включая различные их вариации. Программист может определить собственный атомарный тип данных, воспользовавшись шаблоном класса `std::atomic`. Этот шаблон позволяет наложить требования синхронизации и упорядочения операций на типы, сами по себе не являющиеся атомарными.

Стандартизированный интерфейс управления потоками – это ядро всей системы средств параллельного программирования на языке C++.

1.1.2. Управление потоками

Средства для работы с потоками в языке C++

Средства многопоточного программирования в языке C++ включают потоки, примитивы синхронизации доступа к общим данным, локальные переменные потоков и задания.

1.1.2.1. Классы для поддержки потоков

В библиотеке языка C++ есть два класса для поддержки потоков: простейший `std::thread`, введённый в стандарте C++ 11, и усовершенствованный `std::jthread`, появившийся в стандарте C++ 20.

1.1.2.1.1. Базовые потоки типа std::thread

Класс `std::thread` служит обёрткой для потока – независимой единицы выполнения в составе программы. Выполняемая единица запускается сразу после своего создания, для этого она должна получить на вход вызываемый объект, который и задаёт, что должно быть сделано в потоке. Вызываемый объект может быть именованной функцией, функциональным объектом или лямбда-функцией.

Код, который создаёт поток, отвечает за его дальнейшую судьбу. Единица выполнения, работающая в потоке, завершается с окончанием работы своего вызываемого объекта. Создатель потока может либо присоединить поток, т. е. подождать его завершения посредством вызова `join`, либо отсоединиться от потока, вызвав функцию `detach`). Поток `t` находится в *присоединяемом состоянии*, если над ним не выполнялась ни одна из операций `join` и `detach`. Если в момент вызова деструктора объект-поток находится в присоединяемом состоянии, деструктор вызывает функцию `std::terminate`, которая приводит к аварийному завершению программы.

Поток, отсоединённый от своего создателя, часто называют потоком-демоном, поскольку он выполняется в фоновом режиме.

Под именем `std::thread` скрывается шаблон класса с переменным числом параметров. Это означает, что при создании потока можно передавать любое число аргументов, так как разное число аргументов могут принимать выполняющиеся в потоке вызываемые объекты.

1.1.2.1.2. Усовершенствованные потоки: класс std::jthread (стандарт C++ 20)

Название `jthread` означает «присоединяемый» (англ. *joinable*) поток. В дополнение ко всему, что умеет делать класс `std::thread`, введённый в стандарте C++ 11, этот класс ожидает завершения потока в деструкторе и поддерживает кооперативное прерывание. Тем самым класс `std::jthread` расширяет интерфейс класса `std::thread`.

1.1.2.2. Данные в совместном доступе

Программисту необходимо координировать доступ к общей переменной, если более одного потока могут одновременно обращаться к ней и если переменная при этом может менять своё значение (т. е. не является константой). Чтение данных из общей переменной в то время, как другой поток помещает в неё новое значение, называется гонкой данных и представляет собой неопределённое поведение. Координация доступа к общей переменной достигается в языке C++ с помощью мьютексов и блокировщиков.

1.1.2.2.1. Мьютексы

Мьютекс (mutex, от англ. *mutual exclusion* – взаимное исключение) позволяет гарантировать, что только один поток может получить доступ к общей переменной в каждый момент времени. Мьютекс запирает и открывает крити-

ческую секцию, внутри которой происходит работа с общей переменной. В стандартной библиотеке языка C++ определены пять различных видов мьютексов. Они позволяют блокировать выполнение рекурсивно, с запросом состояния блокировки, с ограничением времени ожидания или без такого ограничения. Особый вид мьютекса даёт возможность даже нескольким потокам входить в критическую секцию одновременно.

1.1.2.2.2. Блокировщики

Чтобы гарантировать автоматическое освобождение мьютекса, его следует оборачивать в объект-блокировщик (`lock`). Блокировщики реализуют идиому RAII – время запирания мьютекса ограничивается временем жизни блокировщика. В стандарте языка имеются классы `std::lock_guard` и `std::scoped_lock` для простых сценариев использования и классы `std::unique_lock` и `std::shared_lock` – для более сложных (например, для явного освобождения и повторного запирания мьютекса).

1.1.2.2.3. Потокобезопасная инициализация

Если общие данные используются только для чтения, достаточно обеспечить потокобезопасность их инициализации. Язык C++ предоставляет для этого множество средств, включая константные выражения, статические переменные, видимые в определённом блоке, или функцию `std::call_once` вместе с флагом `std::once_flag`.

1.1.2.3. Локальные данные потока

Объявление переменной как локальной для потока (англ. *thread-local*) означает, что каждый поток получит собственную копию такой переменной. В этом случае переменная уже не будет общей для нескольких потоков. Время жизни локальных данных потока ограничено временем выполнения потока-хозяина.

1.1.2.4. Переменные условия

Переменные условия (англ. *condition variables*) позволяют синхронизировать потоки путём отправки сообщений[1]. Один поток выступает отправителем оповещения, а остальные – получателями. Типичная ситуация, для которой хорошо подходят переменные условия, – это очередь между производителями и потребителями данных. При этом условную переменную может использовать как отправитель, так и получатель сообщения. Использование переменных условия может оказаться непростым делом; зачастую более простых решений можно добиться на основе т. н. заданий.

---

[1] Можно также сказать, что переменные условия синхронизируют потоки по наступлении некоторого события. – *Прим. перев.*

1.1.2.5. Кооперативное прерывание потоков (стандарт C++ 20)

Полезное дополнение к средствам управления потоками состоит в возможности прерывать их выполнение – при условии, что в коде потока расставлены точки, где его можно прерывать. Такое прерывание называется кооперативным. Кооперативное прерывание поддерживается классами `std::jthread` и `std::condition_variable`, для его реализации служат классы `std::stop_source`, `std::stop_token` и `std::stop_callback`.

1.1.2.6. Семафоры (стандарт C++ 20)

Семафоры представляют собой механизм управления одновременным доступом к общему ресурсу (и в этом отношении отчасти сходны с мьютексами). Семафор снабжён целочисленным счётчиком, который должен быть неотрицательным. Счётчик инициализируется в конструкторе. Каждый захват семафора уменьшает счётчик на единицу, а освобождение – увеличивает. Если поток пытается зайти под семафор (т. е. захватить его), когда счетчик равен нулю, поток блокируется до тех пор, пока какой-то другой поток не освободит семафор, тем самым нарастив счётчик.

1.1.2.7. Защёлки и барьеры (стандарт C++ 20)

Защёлки и барьеры служат для координирования потоков. Они позволяют блокировать поток до тех пор, пока счётчик не достигнет нуля. Начальное значение счётчика задаётся в конструкторе. Несмотря на сходство названий, барьеры, о которых говорится здесь, не имеют ничего общего с барьерами памяти, на которых основана семантика атомарных операций. Для координации потоков через счётчики служат два класса: `std::latch` и `std::barrier`. Одновременный вызов функций-членов объекта такого класса из разных потоков никогда не приводит к гонке данных.

1.1.2.8. Задания

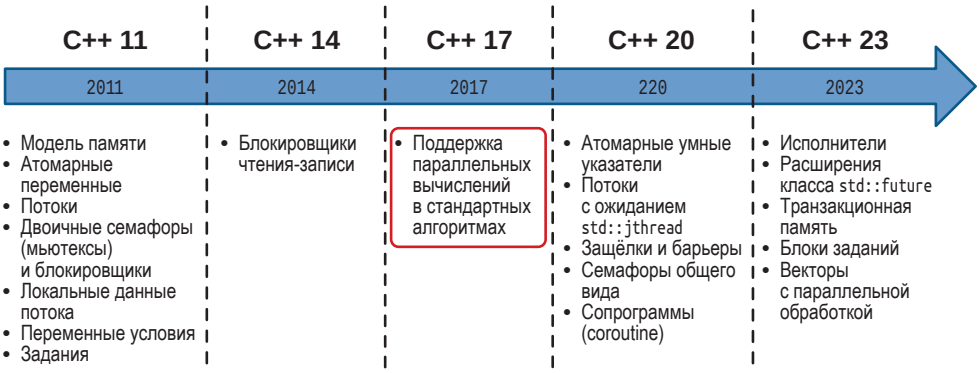
Задания (англ. *tasks*) имеют много общего с потоками. Если поток создаётся и запускается в явном виде, то задание – это нечто, что должно быть рано или поздно выполнено. Реализация стандартной библиотеки C++ сама решит во время выполнения программы, когда задание выполнить, – простым примером может служить функция `std::async`.

Объект-задание похож на канал передачи данных между двумя точками. На основе заданий можно реализовать потокобезопасную коммуникацию между потоками. *Обещание* (англ. *promise*) на одном конце помещает данные в канал, а *фьючерс* на другом конце канала извлекает значение. В роли передаваемых данных могут выступать значения некоторых типов, исключения или просто оповещения о некотором событии. Помимо уже упомянутого `std::async`, стандартная библиотека C++ содержит также шаблоны классов `std::promise` и `std::future`, позволяющие более полно управлять заданиями.

1.1.2.9. Синхронизированные потоки вывода (стандарт C++ 20)

В стандарте C++ 20 появились синхронизированные потоки вывода. Класс `std::basic_syncbuf` представляет собой обёртку над классом `std::basic_streambuf`[1]. Все выводимые символы он накапливает в буфере. Объект-обёртка отправляет накопленные символы в обёрнутый ею поток только в момент своего уничтожения. Следовательно, все сообщения, выведенные в синхронизированный поток, появятся в настоящем потоке вывода в виде единой, цельной последовательности символов. Сообщения из разных потоков при этом перемежаться не могут. Таким образом, из разных потоков можно выводить сообщения в поток `stdout`, не опасаясь путаницы.

1.2. Стандарт C++ 17. Параллельные алгоритмы в стандартной библиотеке



Параллельные алгоритмы в стандарте C++ 17

С появлением стандарта C++ 17 поддержка параллельного программирования языком C++ значительно расширилась, особенно за счёт *параллельных алгоритмов*. Стандарты C++ 11 и C++ 14 содержали лишь простейшие строительные блоки для создания параллельных программ. Эти инструменты были удобны для разработки библиотек или каркасов, но не для разработки приложений. По сравнению со средствами параллельного программирования из стандарта C++ 17, средства стандартов C++ 11 и C++ 14 выглядят словно язык ассемблера.

[1] https://en.cppreference.com/w/cpp/io/basic_streambuf.

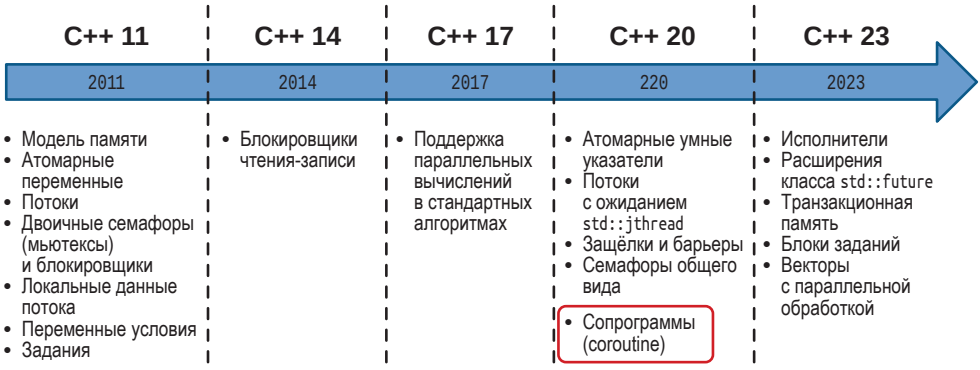
1.2.1. Политики выполнения

В стандарте C++ 17 для большинства алгоритмов из стандартной библиотеки стали доступны параллельные версии. Вызывая тот или иной алгоритм, можно передавать ему так называемую *политику выполнения*. Политика определяет, должен ли алгоритм работать последовательно (`std::execution::seq`), параллельно (`std::execution::par`) или параллельно с дополнительной векторизацией (`std::execution::par_unseq`).

1.2.2. Новые параллельные алгоритмы

В дополнение к 69 старым алгоритмам, получившим возможность выполняться параллельным или параллельно-векторизированным способом, библиотека пополнилась восемью новыми алгоритмами. Эти алгоритмы, предназначенные для свёртки, сканирования и преобразования контейнеров, изначально приспособлены для параллельного выполнения.

1.3. Сопрограммы в стандарте C++ 20



Сопрограммы

Сопрограммы можно представить себе как функции, выполнение которых можно приостанавливать, сохраняя текущее состояние, а затем возобновлять. Сопрограммы хорошо подходят для реализации кооперативной многозадачности, как в некоторых операционных системах, циклов обработки событий, бесконечных списков и конвейеров.

1.4. Учебные примеры

После того как модели памяти и интерфейс для управления потоками будут разобраны теоретически, их использование будет продемонстрировано на нескольких учебных примерах.

1.4.1. Вычисление суммы элементов вектора

Сумму элементов вектора можно вычислить различными способами. Это можно делать последовательно или параллельно с большим или меньшим использованием общих данных. Показатели производительности при этом разительно отличаются.

1.4.2. Потокобезопасное создание объекта-одиночки

Потокобезопасное создание объекта-одиночки – классический пример потокобезопасной инициализации переменной в общем доступе. Существует множество способов сделать это, каждый со своей производительностью.

1.4.3. Поэтапная оптимизация с использованием инструмента CppMem

Начав с небольшой программы, будем шаг за шагом её улучшать. Каждый шаг этой непрерывной оптимизации будем проверять на инструменте под названием CppMem. CppMem – это интерактивное инструментальное средство, позволяющее исследовать поведение небольших участков кода с точки зрения моделей памяти, определённых в языке C++.

1.4.4. Быстрая синхронизация потоков

В стандарте C++ 20 имеется много средств для синхронизации потоков. Так, можно пользоваться переменными условия, типами `std::atomic_flag`, `std::atimic<bool>` или семафорами. В отдельной главе изучим показатели производительности этих способов на примере игры в пинг-понг.

1.4.5. Вариации на тему фьючерсов

Благодаря появлению в языке нового ключевого слова `co_return` появится возможность разработать жадный фьючерс, ленивый фьючерс и фьючерс,

работающий в отдельном потоке, – этому посвящена соответствующая глава. Обилие комментариев должно сделать реализацию лёгкой для понимания.

1.4.6. Модификации и обобщения генераторов

Ключевое слово `co_yield` позволяет создавать бесконечные потоки данных. Отдельная глава посвящена тому, как реализацию бесконечного потока сделать конечной и обобщённой.

1.4.7. Способы управления заданиями

В соответствующей главе рассказывается о том, как сделать сопрограмму, способную пробуждаться, когда нужно. В этом поможет ключевое слово `co_await`.

1.5. Будущее языка C++

Средства параллельного программирования в стандарте C++ 23

«Очень трудно сделать точный прогноз, особенно о будущем», – говорил Нильс Бор.

1.5.1. Исполнители

Исполнитель (executor) содержит набор правил касательно того, где, когда и как выполнять вызываемый объект. Они представляют собой основные блоки, из которых строится выполнение программы, и определяют, должен ли тот или иной код выполняться в произвольном потоке, в пуле потоков или даже в едином потоке без распараллеливания. От них зависят расши-

рения фьючерсов, расширения для работы сетью N4734[1], а также параллельные алгоритмы из стандартной библиотеки; другие средства параллельного программирования из стандартов C++ 20/23, такие как защёлки, барьеры, сопрограммы, транзакционная память и блоки заданий, также смогут использовать исполнители.

1.5.2. Расширенные фьючерсы

Задания, в виде обещаний и фьючерсов появившиеся в стандарте C++ 11, приносят программистам существенную пользу, но у них есть свои недостатки: задания трудно соединять между собой в более крупные единицы. Это ограничение должно исчезнуть благодаря расширениям фьючерсов, появившимся в стандарте C++ 20 и запланированным в стандарте C++ 23. Например, функция-член `then` создаёт фьючерс, который становится готов, когда готово задание-предшественник; функция-член `when_any` – когда готов один из нескольких предшественников, а функция-член `when_all` – когда готовы все предшественники.

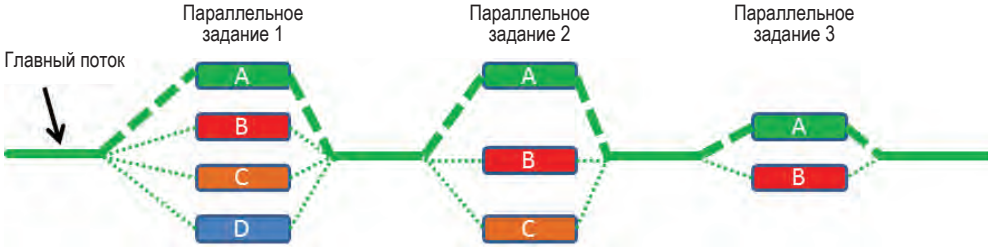
1.5.3. Транзакционная память

Понятие транзакционной памяти основывается на той же идее, что и понятие транзакции в теории баз данных. Транзакция над памятью – это действие, удовлетворяющее первым трем из четырёх требований, предъявляемых к транзакциям в базах данных, известных под названием ACID, а именно требованиям атомарности (atomicity), согласованности (consistency), изолированности (isolation). Требование прочности (durability) имеет смысл для баз данных, но не для разрабатываемой системы транзакционной памяти языка C++. В будущем стандарте ожидаются два вида транзакционной памяти: синхронизированные блоки и атомарные блоки. Оба выполняются с полным упорядочением операций и ведут себя так, будто находятся под глобальной блокировкой. Отличие между ними состоит в том, что в атомарных блоках не может выполняться транзакционно-небезопасный код.

1.5.4. Блоки заданий

Блоки заданий добавляют в язык C++ поддержку идиомы «разветвление–слияние» (fork-join). Следующий рисунок поясняет основную идею блока заданий: на этапе разветвления запускается выполнение ряда заданий, а этап слияния ожидает завершения их всех.

[1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4734.pdf.



Блоки заданий

1.5.5. Библиотека для векторных вычислений

Библиотека для векторных вычислений позволяет воспользоваться распараллеливанием по данным (SIMD – англ. *single instructions*, *multiple data*) при работе с векторными типами. Этот принцип вычислений состоит в том, что каждая операция может выполняться параллельно над несколькими значениями.

1.6. Шаблоны и эмпирические правила

Шаблоны – это хорошо документированные методики, лучшие из устоявшихся на практике. Они «выражают отношение между определённым контекстом, проблемой и решением» (Кристофер Александер). Рассматривая трудности параллельного программирования с более фундаментальной точки зрения, можно получить немалую выгоду. В отличие от главы о шаблонах, глава об устоявшихся практиках посвящена более прагматичным советам о том, как преодолевать те или иные затруднения.

1.6.1. Шаблоны синхронизации

Необходимое условие для возникновения гонок данных – наличие общего доступа потоков к изменяемому состоянию. Шаблоны синхронизации сводятся к двум вопросам: что делать с общим доступом и что делать с изменяемым состоянием.

1.6.2. Шаблоны параллельной архитектуры

В главе о параллельных архитектурах представлены три шаблона. Шаблоны «активный объект» и «монитор» основаны на синхронизации и планировании вызовов функций-членов. Третий шаблон, «полусинхронная архитектура» (англ. *Half-Sync/Half-Async*), более сосредоточен на архитектуре системы и позволяет разделить в ней асинхронную и синхронную обработки.

1.6.3. Эмпирические правила

Параллельное программирование сложно по самой своей сути, поэтому имеет смысл пользоваться накопленным практическим опытом – как в области параллельного программирования в целом, так и, в частности, в том, что касается управления потоками и моделей памяти.

1.7. Структуры данных

Структура данных, которая защищает себя сама таким образом, что гонка данных в ней становится невозможной, называется потокобезопасной. Отдельная глава посвящена трудностям, возникающим при разработке потокобезопасных структур данных с блокировками.

1.8. Сложности параллельного программирования

Создание параллельных программ – сложное дело. Особенно сложным оно становится, если использовать только лишь средства из стандартов C++ 11 и C++ 14. Поэтому в отдельной главе рассказывается о наиболее существенных трудностях. Посвятив специальную главу разбору трудностей параллельного программирования, автор надеется, что читатель будет заранее знать, где подстерегает опасность. В частности, речь в главе пойдёт о состоянии гонок, гонке данных и о мёртвых блокировках (англ. *deadlock*).

1.9. Библиотека для работы со временем

Средства для работы со временем тесно связаны со средствами параллельного программирования. Часто возникает необходимость приостановить выполнение потока на определённый промежуток времени или до наступления определённого момента времени. В состав стандартной библиотеки входят: моменты времени, временные интервалы и часы.

1.10. Обзор инструментального средства CppMem

CppMem – это интерактивный инструмент, позволяющий заглянуть глубоко внутрь модели памяти. Он предоставляет две замечательные услуги. Во-

первых, с его помощью можно верифицировать свой безблокировочный код; во-вторых, можно проанализировать безблокировочный код и получить более точное представление о том, как он работает. В этой книге инструмент CppMem используется часто. Поскольку параметры конфигурации и механизмы работы CppMem довольно сложны, глава даёт лишь общее представление об этом инструменте.

1.11. Пояснение некоторых терминов

Последняя глава книги представляет собой глоссарий важнейших терминов.