

# Содержание

<b>Предисловие</b>	<b>17</b>
Извинения за код	19
Использование примеров кода	19
Соглашения, использованные в книге	20
Об авторе	20
Об изображении на обложке	20
Ждем ваших отзывов!	21
<b>Глава 1. Обзор оптимизации</b>	<b>23</b>
Оптимизация — часть разработки программного обеспечения	24
Эффективность оптимизации	25
Оптимизируйте!	25
Наносекунда туда, наносекунда сюда	28
Стратегии оптимизации кода на C++	28
Используйте компилятор получше; используйте компилятор лучше	29
Использование лучших алгоритмов	30
Использование лучших библиотек	32
Уменьшение количества выделений памяти и копирований	33
Устранение вычислений	33
Использование лучших структур данных	34
Увеличение параллельности	34
Оптимизация управления памятью	35
Резюме	35
<b>Глава 2. Оптимизация, влияющая на поведение компьютера</b>	<b>37</b>
Ложь о компьютерах, в которую верит C++	38
Правда о компьютерах	39
Медленная память	40
Недоступность байтов	41
Одни обращения к памяти медленнее других	41
Остроконечные и тупоконечные слова	42
Количество памяти ограничено	43
Медленное выполнение команд	43
Трудное принятие решений	44
Множественные потоки выполнения	45
Вызовы операционной системы являются дорогостоящими	46
C++ тоже лжет	46
Не все инструкции одинаково дорогие	47
Инструкции выполняются не по порядку	47
Резюме	48

<b>Глава 3. Измерение производительности</b>	<b>49</b>
Оптимизирующее мышление	50
Производительность должна быть измерена	50
Оптимизация — большая игра	51
Правило 90/10	51
Закон Амдала	53
Проведение экспериментов	54
Ведение лабораторного журнала	56
Измерение базовой производительности и постановка целей	57
Улучшить можно только измеряемое	60
Профилирование выполнения программы	60
Длительно работающий код	63
“Полузнайство” об измерении времени	64
Измерение времени с помощью компьютеров	69
Преодоление проблем измерений	77
Создание класса-секундомера	81
Хронометраж функции в тесте	85
Оценка стоимости кода для поиска узких мест	86
Оценка стоимости отдельных инструкций C++	87
Оценка стоимости циклов	88
Другие пути поиска узких мест	89
Резюме	90
<b>Глава 4. Оптимизация использования строк</b>	<b>91</b>
Почему строки представляют собой проблему	91
Строки используют динамическое выделение памяти	92
Строки как значения	92
Строки выполняют массу копирований	93
Первая попытка оптимизации строк	94
Использование модифицирующих операций для устранения временных значений	96
Уменьшение работы с памятью с помощью резервирования	96
Устранение копирования строкового аргумента	97
Устранение разыменовании с помощью итераторов	98
Устранение копирования возвращаемого значения	99
Использование массивов символов вместо строк	100
Итоги первой попытки оптимизации	102
Вторая попытка оптимизации строк	102
Использование лучшего алгоритма	102
Использование лучшего компилятора	104
Использование лучшей библиотеки для работы со строками	105
Использование лучшего менеджера памяти	109

Устранение преобразования строк	110
Преобразование строк в стиле C в <code>std::string</code>	111
Преобразование между кодировками	111
Резюме	112
<b>Глава 5. Оптимизация алгоритмов</b>	<b>113</b>
Временная стоимость алгоритмов	115
Временная стоимость в наилучшем, среднем и наихудшем случаях	117
Амортизированная временная стоимость	118
Прочие стоимости	118
Оптимизации сортировки и поиска	118
Эффективные алгоритмы поиска	119
Временная стоимость алгоритмов поиска	119
Все поиски равноценны при малых $n$	120
Эффективные алгоритмы сортировки	121
Временная стоимость алгоритмов сортировки	121
Замена сортировки с плохой производительностью в наихудшем случае	122
Использование информации о входных данных	123
Шаблоны оптимизации	123
Предвычисления	124
Отложенные вычисления	125
Пакетирование	126
Кеширование	126
Специализация	127
Группировка	127
Подсказки	128
Оптимизация ожидаемого пути	128
Хеширование	128
Двойная проверка	129
Резюме	129
<b>Глава 6. Оптимизация переменных в динамической памяти</b>	<b>131</b>
Переменные C++	132
Длительность хранения переменной	132
Владение переменными	135
Объекты-значения и объекты-сущности	136
API динамических переменных C++	138
Автоматизация владения интеллектуальными указателями	140
Динамические переменные имеют стоимость времени выполнения	143
Уменьшение использования динамических переменных	144
Статическое создание экземпляров класса	145
Использование статических структур данных	146
Использование <code>std::make_shared</code> вместо <code>new</code>	150

Не следует разделять владение без необходимости	150
Использование “главного указателя” для владения динамическими переменными	152
Уменьшение количества перераспределений динамических переменных	152
Предварительное выделение памяти для динамических переменных для предотвращения перераспределений	152
Создание динамических переменных вне циклов	153
Устранение излишнего копирования	154
Устранение нежелательного копирования в определении класса	155
Устранение копирования при вызове функции	156
Устранение копирования при возврате из функции	158
Библиотеки без копирования	160
Реализация идиомы “копирования при записи”	161
Срезы	162
Реализация семантики перемещения	163
Нестандартная семантика копирования: болезненный хак	163
std::swap(): семантика перемещения для бедных	164
Разделяемое владение сущностями	165
Перемещающая часть семантики перемещения	166
Изменения кода для использования семантики перемещения	167
Тонкости семантики перемещения	168
Плоские структуры данных	171
Резюме	172
<b>Глава 7. Оптимизация инструкций</b>	<b>173</b>
Удаление кода из циклов	174
Кеширование конечного значения цикла	175
Применение более эффективных инструкций циклов	175
Изменение направления цикла	176
Устранение инвариантного кода из циклов	177
Удаление ненужных вызовов функций из циклов	177
Удаление скрытых вызовов функций из циклов	180
Удаление дорогих медленно меняющихся вызовов из циклов	182
Перемещение циклов в функции для снижения накладных расходов при вызовах	183
Выполняйте некоторые действия пореже	184
И все остальное	186
Удаление кода из функций	186
Стоимость вызовов функций	186
Объявление коротких функций встраиваемыми	190
Определение функций до их первого использования	191
Устранение неиспользуемого полиморфизма	191
Удаление неиспользуемых интерфейсов	192
Выбор реализации во время компиляции с помощью шаблонов	196
Исключение применения идиомы PIMPL	196

Устранение вызовов кода в DLL	198
Используйте статические функции-члены вместо функций-членов экземпляров	199
Перенесение виртуального деструктора в базовый класс	199
Оптимизация выражений	200
Упрощение выражений	201
Группирование констант	202
Используйте менее дорогостоящие операторы	203
Использование целочисленной арифметики вместо арифметики с плавающей точкой	203
double может быть быстрее, чем float	205
Замена итеративных вычислений аналитическими выражениями	206
Идиомы оптимизации потока управления	207
Применение switch вместо if-elseif-else	208
Применение виртуальных функций вместо switch или if	208
Используйте обработку исключений без стоимости	209
Резюме	211
<b>Глава 8. Использование лучших библиотек</b>	<b>213</b>
Оптимизация использования стандартной библиотеки	213
Философия стандартной библиотеки C++	214
Вопросы применения стандартной библиотеки C++	215
Оптимизация существующих библиотек	217
Изменения должны быть небольшими	218
Добавление функций, а не изменение функциональности	219
Проектирование оптимизированных библиотек	219
Кодировать на скорую руку — обречь себя на долгую муку	219
При разработке библиотек скупость является добродетелью	221
Принятие решений о выделении памяти вне библиотеки	221
Если сомневаетесь, выбирайте скорость	222
Оптимизация функций проще оптимизации каркасов	222
Плоские иерархии наследования	223
Упрощение цепочки вызовов	223
Упрощение проектирования слоев	223
Избегайте динамического поиска	225
Остерегайтесь “функций Бога”	226
Резюме	227
<b>Глава 9. Оптимизация сортировки и поиска</b>	<b>229</b>
Таблицы “ключ/значение” с использованием std::map и std::string	230
Инструментарий для повышения производительности поиска	231
Выполнение базовых измерений	232
Идентификация оптимизируемой деятельности	232
Разделение оптимизируемой деятельности	233

Изменение или замена алгоритмов и структур данных	234
Использование процесса оптимизации для пользовательских абстракций	236
Оптимизация поиска с использованием <code>std::map</code>	237
Применение символьных массивов фиксированного размера в качестве ключей <code>std::map</code>	237
Использование строк в стиле C в качестве ключей <code>std::map</code>	238
Использование <code>std::set</code> , когда ключ является значением	241
Оптимизация поиска с использованием заголовочного файла <code>&lt;algorithm&gt;</code>	241
Таблица “ключ/значение” для поиска в последовательных контейнерах	243
<code>std::find()</code> : очевидное имя, стоимость — $O(n)$	244
<code>std::binary_search()</code> : не возвращает значения	245
Бинарный поиск с использованием <code>std::equal_range()</code>	245
Бинарный поиск с использованием <code>std::lower_bound()</code>	246
Самостоятельное кодирование бинарного поиска	247
Самостоятельное кодирование бинарного поиска с использованием <code>strcmp()</code>	248
Оптимизация поиска в хешированных таблицах “ключ/значение”	248
Хеширование с использованием <code>std::unordered_map</code>	249
Хеширование с фиксированными символьными массивами в качестве ключей	250
Хеширование с ключами в виде строк с завершающими нулевыми символами	251
Хеширование с пользовательской хеш-таблицей	253
Цена абстракций Степанова	254
Оптимизация сортировки с использованием стандартной библиотеки C++	255
Резюме	257
<b>Глава 10. Оптимизация структур данных</b>	<b>259</b>
Знакомство с контейнерами стандартной библиотеки	259
Последовательные контейнеры	260
Ассоциативные контейнеры	260
Эксперименты с контейнерами стандартной библиотеки	261
<code>std::vector</code> и <code>std::string</code>	266
Следствия перераспределения для производительности	267
Вставка и удаление в <code>std::vector</code>	268
Итерирование <code>std::vector</code>	270
Сортировка <code>std::vector</code>	271
Поиск в <code>std::vector</code>	271
<code>std::deque</code>	271
Вставка и удаление в <code>std::deque</code>	273
Итерирование <code>std::deque</code>	275
Сортировка <code>std::deque</code>	275
Поиск в <code>std::deque</code>	275
<code>std::list</code>	275
Вставка и удаление в <code>std::list</code>	278
Итерирование <code>std::list</code>	278

Сортировка <code>std::list</code>	278
Поиск в <code>std::list</code>	279
<code>std::forward_list</code>	279
Вставка и удаление в <code>std::forward_list</code>	280
Итерирование <code>std::forward_list</code>	280
Сортировка <code>std::forward_list</code>	281
Поиск в <code>std::forward_list</code>	281
<code>std::map</code> и <code>std::multimap</code>	281
Вставка и удаление в <code>std::map</code>	282
Итерирование <code>std::map</code>	284
Сортировка <code>std::map</code>	285
Поиск в <code>std::map</code>	285
<code>std::set</code> и <code>std::multiset</code>	285
<code>std::unordered_map</code> и <code>std::unordered_multimap</code>	286
Вставка и удаление в <code>std::unordered_map</code>	289
Итерирование <code>std::unordered_map</code>	290
Поиск в <code>std::unordered_map</code>	290
Другие структуры данных	290
Резюме	292
<b>Глава 11. Оптимизация ввода-вывода</b>	<b>293</b>
Рецепты для чтения файлов	293
Создание экономной сигнатуры функции	295
Сокращение цепочек вызовов	297
Снижение количества перераспределений	297
Использование большего входного буфера	299
Использование построчного чтения	300
Еще одно сокращение цепочек вызовов	302
Бесполезные вещи	303
Запись файлов	303
Чтение из <code>std::cin</code> и запись в <code>std::cout</code>	304
Резюме	305
<b>Глава 12. Оптимизация параллельности</b>	<b>307</b>
Введение в параллельные вычисления	308
Экскурсия по зоопарку параллелизма	309
Чередующееся выполнение	313
Последовательная согласованность	314
Гонки	315
Синхронизация	316
Атомарность	317
Возможности параллельности в C++	319
Потоки	320
Обещания и фьючерсы	321

Асинхронные задания	323
Мьютексы	325
Блокировки	326
Условные переменные	327
Атомарные операции над общими переменными	330
Будущие возможности параллелизма C++	333
Оптимизация многопоточных программ C++	334
Предпочитайте <code>std::async</code> , а не <code>std::thread</code>	335
Создавайте потоков столько же, сколько имеется ядер	337
Реализуйте очередь заданий и пул потоков	338
Выполняйте ввод-вывод в отдельном потоке	339
Программа без синхронизации	339
Удаление кода запуска и завершения	342
Более эффективная синхронизация	343
Уменьшайте критические разделы	344
Ограничивайте количество параллельных потоков	344
Избегайте громового стада	346
Избегайте очереди на блокировку	346
Уменьшайте конкуренцию	347
Не пользуйтесь активным ожиданием в одноядерных системах	348
Не ждите вечно	349
Собственные мьютексы могут быть неэффективными	349
Ограничивайте длину очереди вывода производителя	349
Библиотеки для параллельных вычислений	350
Резюме	351
<b>Глава 13. Оптимизация управления памятью</b>	<b>353</b>
API управления памятью C++	354
Жизненный цикл динамических переменных	354
Функции для выделения и освобождения памяти	355
Построение динамических переменных с помощью выражений <code>new</code>	358
Уничтожение динамических переменных с помощью выражения <code>delete</code>	361
Явный вызов деструктора уничтожает динамическую переменную	362
Высокопроизводительные диспетчеры памяти	363
Диспетчеры памяти для конкретных классов	365
Диспетчер памяти для блоков фиксированного размера	366
Арена блоков	369
Добавление <code>operator new()</code> для конкретного класса	371
Производительность диспетчера памяти для блоков фиксированного размера	372
Вариации диспетчера памяти для блоков фиксированного размера	372
Небезопасные с точки зрения параллельности диспетчеры более эффективны	373



Пользовательские аллокаторы стандартной библиотеки	374
Минимальный аллокатор в C++11	376
Дополнительные определения для аллокатора C++98	378
Аллокатор блоков фиксированного размера	382
Аллокатор блоков фиксированного размера для строк	384
Резюме	385
<b>Предметный указатель</b>	<b>387</b>