

Оптимизация программ

Введение в оптимизацию программ

Мееров И.Б.

к.т.н., доц., каф. МОСТ, институт ИТММ

Содержание

- Что такое оптимизация?
- Критерии оптимизации
- Требования к оптимизации
- Правила оптимизации
- Жизненный цикл оптимизации
- Бенчмарк
- Инфраструктура
- Оптимизация и параллельность
- Примеры
- Литература

Оптимизация в математике и оптимизация программ

- Оптимизация в математике \neq оптимизация программ
- Оптимизация в математике: ищем глобальный или локальный минимум.
- Оптимизация программ не стремится к поиску оптимального варианта, т.е. варианта, который нельзя улучшить (за исключением маркетинговых соображений).
- Цель — поиск **приемлемого** варианта в смысле некоторых критериев.
- Оптимизация программ — *процесс улучшения программы в смысле некоторых критериев.*

Цель оптимизации программ

Цель оптимизации программ:

получение из **работающего*** варианта программы другого **работающего*** варианта, удовлетворяющего некоторым критериям.

* Принципиальный момент.

Вопрос: как организовать **инфраструктуру**?

Ответ: нужна постоянная проверка **работоспособности**.

// Примеры из практики

Рассматриваемые критерии

Основными **критериями** при оптимизации программ являются:

- **Скорость работы**

Миф: компьютеры работают все быстрее и быстрее. Ничего делать не нужно 😊

// Большие задачи; Real-time; Финансы и др. области

- **Объем используемой памяти**

Миф: много дешевой памяти и будет еще больше. Ничего делать не нужно 😊

// Большие задачи; ограничения на ускорителях

- **Объем места, занимаемого на диске**

Выглядит сомнительным, но вспоминаем про инженерные расчеты и ограничения на кластерах

3. Рассматриваемые критерии.

Скорость работы

Если программа работает медленно:

- **ее не удастся применить на практике**

пример: *Прогноз погоды* на завтра нужен сегодня, а не через неделю

пример: Система технического мониторинга не успеет подать сигнал об опасности

- **с ней откажутся работать потенциальные пользователи**

пример: *Обработка фотографий* – слишком большое время ожидания, и у нас не купят наш графический редактор

- **пользователи потеряют прибыль**

пример: финансовые расчеты, торговые роботы

3. Рассматриваемые критерии.

Объем используемой памяти

Если программа требует слишком много памяти (ОЗУ):

- **с ней откажутся работать потенциальные пользователи**
пример: офисные приложения
- **программа будет работать за неудовлетворительное время**
пример: работа на ускорителях (объем памяти)
- **данные не поместятся в память**
пример: большие задачи (линейная алгебра и др. области)

3. Рассматриваемые критерии.

Объем места, занимаемого на диске

Если программа требует много места на жестком диске:

- **ее могут отказаться приобретать потенциальные пользователи**

пример: пользователей не обрадует, что следующая версия продукта «весит» втрое больше, чем текущая

- **может не хватить отведенного дискового пространства на кластере**

пример: научные и инженерные расчеты, научная визуализация

Рассматриваемые критерии. Взаимодействие

- Скорость (performance)
- Память
- Место на диске

Часто критерии противоречивы:

- Используем дополнительную память – ускоряем работу
- Используем место на диске для однократного расчета и последующего хранения и использования – ускоряем работу (**пример**: кратчайшие пути, см. [Renato Werneck](#), 2010)

Обычно один из критериев является *основным*.

В современных условиях это чаще **скорость работы**, но есть и другие случаи.

Н.В. Важный момент: эффективность масштабируемости (за рамками лекции)

Цели и задачи оптимизации

- Необходимость оптимизации должна быть обоснована
- Существует немало задач, в которых оптимизация не нужна, а иногда и вредна
- Чаще всего, оптимизация затрагивает лишь фрагмент приложения, незначительный в процентном отношении по объему кода
- Не нужно оптимизировать все подряд из любви к искусству!
- Не нужно писать «хакерский код», в котором никто не сможет разобраться (в т.ч. автор через месяц)!

Требования к оптимизации *

- Оптимизация ради оптимизации не только бесполезна, но и вредна.
- **Требования:**
 - Переносимость
 - Небольшая трудоемкость
 - Значимый выигрыш
 - Понятность программы
 - Возможность развития

* По материалам К. Касперски. *Техника оптимизации программ. Эффективное использование памяти.* БХВ-Петербург, 2003.

Требования к оптимизации. Переносимость

- Избегаем ассемблерных вставок.
- Избегаем недокументированных возможностей аппаратуры и средств разработки.

Основная причина:

средства разработки и аппаратура тоже развиваются.

Совместимость.

Требования к оптимизации.

Небольшая трудоемкость

- Оптимизация не должна существенно увеличивать трудоемкость разработки и тестирования!

Основная причина:

Необходимо уложиться в сроки и бюджет и выпустить продукт. **Рамки проекта.**

Требования к оптимизации.

Значимый выигрыш

- Оптимизация должна приносить значимый выигрыш.

Основная причина:

Избегаем **Оптимизации ради оптимизации**. Соразмерность затрат и результата.

- * Есть области, в которых 2% это значимый выигрыш:

Пример: моделирование финансовых рынков.

Требования к оптимизации.

Понятность программы

- Текст программы должен остаться понятным “нормальному” программисту.

Основная причина:
Сопровождение

Требования к оптимизации. Возможность развития

- Необходимо соблюдать в целом проектные решения.

Основная причина:

Модификации неизбежны

Правила оптимизации (1-3)*

- Прежде чем оптимизировать программу, *убеждаемся в ее работоспособности.*
- Основной прирост производительности приходит от алгоритмической оптимизации и выбора структур данных, а не от «трюков» (*ищем эффективные алгоритмы и подходящие СД*).
- Оптимизация кода \neq Ассемблерная реализация (*улучшаем код для увеличения быстродействия в рамках ЯПВУ*).
Используем возможности компилятора.
- Используем оптимизированные библиотеки.

* По материалам К. Касперски. *Техника оптимизации программ. Эффективное использование памяти.* БХВ-Петербург, 2003.


Правила оптимизации (4-5)*

- Перед переписыванием на Ассемблер изучаем ассемблерный листинг после работы компилятора (*можем ли сделать лучше?*).
- Если ничего не помогло, но мы знаем, как правильно реализовать алгоритм при помощи системы команд CPU, *только тогда переходим на Ассемблер.*

* По материалам К. Касперски. *Техника оптимизации программ. Эффективное использование памяти.* БХВ-Петербург, 2003.

Жизненный цикл оптимизации

Предполагаем, что программа написана и работает, но работает недостаточно быстро.

- Подготовить подходящий бенчмарк.
 - Провести тесты.
 - Провести анализ производительности.
 - Выявить проблемы производительности.
 - Выбрать метод решения одной из проблем.
 - Модифицировать код.
- 

Много тонкостей. Далее будут примеры типичных ошибок на каждом этапе схемы.

Бенчмарк*

- **Бенчмарк** — наборы тестовых данных, предназначенные для решения следующих задач:
 - **объективная** оценка производительности приложения;
 - создание полигона для дальнейшей оптимизации.

Бенчмарк должен быть *репрезентативен (!!!), прост в использовании, работать разумное время* (не слишком медленно, но и не слишком быстро), обеспечивать *воспроизводимость результатов*.

* По материалам К. Касперски. *Техника оптимизации программ. Эффективное использование памяти. БХВ-Петербург, 2003.*

Бенчмарк (примеры, проблемы)

- Примеры:
 - Сортировки
 - Архиваторы
 - Линейная алгебра (плотная, разреженная)
- **Примеры проблем:**
 - Неудачно подобранные данные
 - Чтение данных >90% времени

Не совсем про бенчмарк, но тоже важно:

- *Неправильная постановка эксперимента:*
все запуски в одном процессе – некорректные результаты

Средства оптимизации

- Оптимизирующий компилятор.
- Средства профилировки.
- Оптимизирующие библиотеки.
- *Главное средство*: голова программиста, снабженная знаниями основ программирования, теории сложности, алгоритмов и структур данных, архитектуры ВС и др.

// Будет отдельная лекция

Виды оптимизации

- Алгоритмическая оптимизация
- Оптимизация структур данных
- Компиляторная оптимизация
- Использование оптимизированных библиотек
- Программная оптимизация

// Обсуждение с аудиторией

И вновь «виды оптимизации»

- Алгоритмы и структуры данных — тема отдельного курса (все знакомы).
- Основной ресурс — выбор лучшего алгоритма и подходящих структур данных
- Теорию нужно применять грамотно (к вопросу о теоретической оценке констант в сложностных формулах).
 - Пример: Линейный поиск в массиве.

Линейный поиск - 1

```
int Search(int key) {  
    int Index = -1;  
    for (i = 0; i < N; i++)  
        if (A[i] == key) Index = i;  
  
    return Index;  
}
```

Линейный поиск – 2.

Как уменьшить число проверок?

```
int Search(int key) {  
    int Index = -1;  
    for (i = 0; i < N; i++)  
        if (A[i] == key) {  
            Index = i;  
            break;  
        }  
    return Index;  
}
```

Линейный поиск – 3.

Всегда ли это будет эффективно работать?

```
int Search(int key) {  
    int Index, i = 0;  
    A[N] = key;  
    while (A[i] != key) i++;  
    if (i == N) Index = -1; else Index = i;  
    return Index;  
}
```

НЕТ!

Обсуждение примера и того, что с ним сделал компилятор ☺

И вновь «виды оптимизации» (анонс будущих лекций)

■ Компиляторная оптимизация:

- Используем C/C++ и Fortran, оптимизирующие компиляторы
- C лучше, чем C++. Поменьше ООП, STL и т.д. ☺

Компилятор предпочитает C и Fortran.

Если очень нужно, можно сделать умелый гибрид C++ на верхнем уровне и C на нижнем уровне (**примеры**).

- Что нам дает компилятор?
 - Множественные оптимизации: размещение данных, оптимизация циклов, векторизация (SIMD), предвыборка, встраивание функций, настройка точности... **Нужно смотреть отчеты (Qvec-report).**
 - LibM – библиотека математических функций

И вновь «виды оптимизации» (анонс будущих лекций)

■ Использование оптимизированных библиотек:

- Intel Math Kernel Library (есть и другие: Boost, NAG, ACML, GSL, PETSc, SLEPc, солверы и многие другие)
- Проблемно-ориентированные библиотеки (OpenCV...)
- Библиотеки поддержки параллельных вычислений (TBV...)

И вновь «виды оптимизации» (анонс будущих лекций)

■ Программная оптимизация, математика + базовые вещи:

- Векторизация циклов (долой зависимости, поможем компилятору, упростим циклы...)
- Вычисление мат. функций (Обсудим: **LibM**, **SVML**, **VML**)
- Замена делений там, где можно.

Пример: $1 / \text{sqrt}(x) = \text{invsqrt}(x);$

- Эквивалентные преобразования, **пример:** `cndf()` vs. `erf()`
- `float` vs. `double` (обсудим: `x87`, `scalar SSE`, `vector SSE`; не смешивать типы данных!)
- Помним про прогноз ветвлений (где можно, лучше обойтись без них – `lookup table` и др.)
- Убираем рекурсию

И вновь «виды оптимизации» (анонс будущих лекций)

- Программная оптимизация, размещение данных в ОЗУ:
 - Многоуровневая иерархия памяти
 - Локальность (массив лучше, чем список; список — можно обдумать вариант с хранением в непрерывном участке памяти)
 - Пишем циклы в правильном порядке (**пример**: dgemm)
 - Поможем компилятору (одномерные массивы это хорошо)
 - SOA vs. AOS — что лучше?
 - Выравнивание массивов (для векторизации)
 - Выравнивание в структурах

Оптимизация и параллелизм

- Выбор алгоритма
 - Разработка
 - Отладка и тестирование
-

■ Оптимизация, распараллеливание?

Вопрос: в каком порядке? Универсального ответа нет.

Пример: сортировки (MSD, LSD, ...).

В ряде случаев необходимо ставить вопрос о разработке принципиально другого (параллельного) алгоритма.

■ Оптимизация и масштабируемость

Использованные источники

При создании презентации активно использовались материалы книг:

Крис Касперски. *Техника оптимизации программ. Эффективное использование памяти.* — Санкт-Петербург: БХВ-Петербург, 2003.

R. Gerber, A. Bik, K. Smith, X. Tian. The Software Optimization Cookbook.

Литература

- Что читать (области)?
 - Архитектура ЭВМ
 - Теория сложности
 - Алгоритмы и структуры данных
 - Параллельное программирование
 - Оптимизация программ

Какие книги и курсы лекций Вы знаете?

// Обсуждение

Вопросы

- ???