Лекция 1. Оптимизации компиляторов

Инструменты разработки быстрых программ

13 ноября 2017 г.

Список литературы I

Компиляторы — книги

Kennedy K., Allen J. R. — Optimizing Compilers for Modern Architectures: A Dependence-based Approach. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2002. — ISBN 1-55860-286-0.

Muchnick S. S. — Advanced Compiler Design and Implementation. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1997. — ISBN 1-55860-320-4.

Компиляторы: принципы, технологии и инструментарий. — пер. с англ. / А. Ахо [и др.]. — 2-е изд. — М. : ООО “И. Д. Вильямс”, 2008. — 1184 с. — ISBN 978-5-8459-1349-4.

Список литературы II

Компиляторы

- Clang 6 documentation : — Clang Compiler User's Manual. — URL: http://clang.llvm.org/docs/UsersManual.html (visited on 11/13/2017).
- Intel® C++ Compiler 18.0 Developer Guide and Reference. — 11/10/2017. — URL: https://software.intel.com/en-us/cpp-compiler-18.0-developer-guide-and-reference (visited on 11/13/2017).
- LLVM Language Reference Manual. — URL: http://llvm.org/docs/LangRef.html (visited on 11/13/2017).
- LLVM's Analysis and Transform Passes. — URL: http://llvm.org/docs/Passes.html (visited on 11/13/2017).
- Using the GNU Compiler Collection (GCC). — URL: https://gcc.gnu.org/onlinedocs/gcc-7.2.0/gcc/ (visited on 11/13/2017).

Список литературы III

Параллельное программирование

- *Букатов А. А., Дацюк В. Н., Жегуло А. И.* — Программирование многопроцессорных вычислительных систем. — Ростов-на-Дону : Издательство ООО «ЦВВР», 2003. — 208 с. — ISBN 5-94153-062-5. — URL: http://rsusu1.rnd.runnet.ru/tutor/method/book.pdf.
- Инструменты параллельного программирования в системах с общей памятью : — учебник. /. — К. В. Корняков [и др.]. — М. : Издательство Московского университета, 2010. — 272 с. — (Суперкомпьютерное образование). — ISBN 978-5-211-05931-3.
- Практикум по методам параллельных вычислений : — учебник. /. — А. В. Старченко [и др.] ; под ред. А. В. Старченко. — М. : Издательство Московского университета, 2010. — 200 с. — (Суперкомпьютерное образование). — ISBN 978-5-211-05976-4.

Список литературы IV

Библиотеки

- Developer Reference for Intel® Math Kernel Library 2018 - C. — 11/08/2017. — URL: https://software.intel.com/en-us/mkl-developer-reference-c (visited on 11/13/2017).
- FFTW 3.3.7. — URL: http://www.fftw.org/fftw3_doc/ (visited on 11/13/2017).
- LAPACK Users' Guide. /. — E. Anderson [et al.]. — URL: http://www.netlib.org/lapack/lug/ (visited on 11/13/2017).
- ScaLAPACK Users' Guide. /. — L. S. Blackford [et al.]. — URL: http://www.netlib.org/scalapack/slug/ (visited on 11/13/2017).

Рассматриваемые темы

Темы курса

1. Оптимизирующие компиляторы.
   - Ключи оптимизации.
   - Прагмы оптимизации.
   - Расширения языка.
2. Средства анализа кода.
   - Анализ многопоточных программ.
   - Анализ MPI-программ.
3. Библиотеки численных расчётов.
   - FFTW.
   - ScaLAPACK.
   - MKL.

Базовый блок

Определение

Базовый блок: (basic block) — участок программы, в котором:

- начало — точка перехода;
- последняя операция — переход (условный/безусловный);
- других точек перехода и переходов нет.

SSA-форма

Определение

Граф потока управления: (control flow graph, CFG) — граф программы:

- вершины — все базовые блоки;
- дуги — возможные переходы.

Пример (граф потока управления)

```
double f(double x)
{
  double y = 2 * x;
  while (y > 1)
    y /= 2;
  return y;
}
```

SSA-форма

Определение

SSA-форма: (static single assignment form) — представление программы, в котором:

- каждая переменная присваивается ровно 1 раз;
- каждая переменная определена перед использованием;
- для представления переменных, принимающих значения из разных базовых блоков, используются φ-функции.

Оптимизации

Не компромиссные (no tradeoff)

- Свёртка констант (constant folding).
- Распространение констант (constant propagation).
- Удаление тупиковых записей (dead store elimination).
- Удаление мёртвого кода (dead code elimination).
- Удаление общих подвыражений (common subexpression elemenation).
- Подстановка функций (inlining).
- Планирование инструкций (instruction scheduling).

Компромисс в размере/скорости выполнения кода (space-speed tradeoff)

- Раскрутка циклов (loop unrolling).

Свёртка и распространение констант

Пример (свёртка констант)

```
double y = 2 * sin(0.5) * x;
```

Особенности

- Должна учитывать правила арифметики целевой платформы.

Пример (распространение констант)

```
double pi = 3.14159265358979;
double l = 2 * pi * r;
```

Удаление тупиковых записей (DSE)

Пример (исходный фрагмент)

```
double f(double x)
{
  double y = sin(x);
  double z = y * y;
  y = x + 1;     // тупиковая запись
  return z;
}
```

Достоинства

- Уменьшает нагрузку на ресурсы.
- Ускоряет программу, уменьшает использование энергии.

Недостатки

- Нарушение безопасности.

Удаление мёртвого кода (DCE)

Виды

- Тупиковые записи.
- Недостижимый код.
- Вычисление значений, используемых в мёртвом коде.

Достоинства

- Как и для удаления тупиковых записей.
- Удаляет лишний код-результат других оптимизаций.

Недостатки

- Нарушение безопасности.

Удаление общих подвыражений (CSE)

Пример (исходный фрагмент)

```
x = cos(v) * (1 + sin(u / 2)) + sin(w) * (1 - sin(u / 2));
```

Пример (после оптимизации)

```
t = sin(u / 2);
x = cos(v) * (1 + t) + sin(w) * (1 - t);
```

Особенности удаления общих подвыражений

Достоинства

- В большинстве случаев уменьшает размер и время выполнения программы.
- Может удалять выражения, которые недоступны пользователю (индексирование элементов массива, подстановка макросов, языковые конструкции, . . .)

Недостатки

- Возможное возрастание нагрузки на регистры.

Виды удаления общих подвыражений

Виды

Локальное: ($LCSE$) — в пределах базовых блоков (basic block).

Глобальное: ($GCSE$) — в пределах функций.

Подстановка функций (function inlining)

Пример (исходный фрагмент)

```
double sq(double x)
{
  return x * x;
}

int f(/* ... */)
{
  // ...
  for (i = 0; i < 1000000; ++ i)
    dSum += sq(i + 0.5);
  // ...
}
```

Пример (после подстановки)

```
int f(/* ... */)
{
  // ...
  for (i = 0; i < 1000000; ++ i)
  {
    double dTemp = i + 0.5;
    dSum += dTemp * dTemp;
  }
  // ...
}
```

Особенности подстановки

Достоинства

- Имеет смысл, если функция содержит относительно мало инструкций и использует значительную долю времени выполнения $\Rightarrow$ затраты на вызов (копирование аргументов, инициализация стека) значительны.
- Может повысить эффективность дальнейших преобразований, когда все функции слиты в одну (CSE, ...)

Недостатки

- В большинстве случаев затраты на вызов функций незначительны по сравнению с общим временем работы программы.

Особенности подстановки (окончание)

Ситуации, когда преобразование всегда выгодно

- Есть только одна точка вызова функции.
- Вызов функции требует больше инструкций/памяти, чем исполнение тела функции в месте вызова (простые функции доступа).

Реализация в компиляторе GCC

- Выбирает функции для подстановки на основе ряда эвристик (малый размер, . . .)
- Выполняется в пределах каждого объектного модуля.
- Можно явно запросить при помощи ключевого слова **`inline`**, если возможно.

Особенности подстановки (окончание)

Ситуации, когда преобразование всегда выгодно

- Есть только одна точка вызова функции.
- Вызов функции требует больше инструкций/памяти, чем исполнение тела функции в месте вызова (простые функции доступа).

Реализация в компиляторе GCC

- Выбирает функции для подстановки на основе ряда эвристик (малый размер, . . .)
- Выполняется в пределах каждого объектного модуля.
- Можно явно запросить при помощи ключевого слова **`inline`**, если возможно.

Раскрутка циклов (loop unrolling)

Исходный цикл

```
for (i = 0; i < N; ++ i)
  loop_body(i);
```

Раскрученный порядка n

```
for (i = 0; i < N_new; i += n)
{
  loop_body(i);
  loop_body(i + 1);
  ...
  loop_body(i + n - 1);
}

for (i = N_new; i < N; ++ i)
  loop_body(i);
```

Особенности раскрутки циклов

Преимущества

- Минимизация промахов ветвления (branch penalty).
- Независимые операторы $\Rightarrow$ могут выполняться параллельно.
- Лучшая загрузка оборудования при конвейеризации.
- Может выполняться динамически, если количество итераций неизвестно во время компиляции.

Недостатки

- Увеличение размеров кода (промахи кэша инструкций).
- Противоречие с подстановкой функций.
- Возможно увеличение количества используемых регистров для временных значений.
- Наличие условных операторов сильно замедляет выполнение.

Планирование инструкций (instruction scheduling)

Виды

Локальное: (*local*) — инструкции не могут перемещаться через границы базовых блоков.

Глобальное: (*global*) — инструкции могут перемещаться через границы базовых блоков.

По модулю: (*modulo scheduling*) — перекрытие по времени итераций циклов.

Особенности поведения компилятора GCC

Включение оптимизаций

- Большинство оптимизаций включено, только если в командной строке указан ключ “`-O...`”.
- Иначе оптимизации отключены, даже если в командной строке указаны соответствующие флаги.
- Не все оптимизации управляются напрямую флагами.

Пример (просмотр справки об использовании оптимизаций)

```
gcc -Q --help=optimizers -O3
```

Предустановленные наборы оптимизаций

Ключ	Краткое описание
`-O0`	Минимизация времени компиляции, обеспечение ожидаемых результатов при отладке.
`-O1`	Уменьшение размеров кода и времени исполнения программы при помощи оптимизаций, не занимающих много времени компиляции и не вносящих противоречия между размером кода и временем исполнения.
`-O2`	Почти все оптимизации, не вносящие противоречия между размером кода и временем исполнения. Планирование инструкций.
`-O3`	Оптимизации `-O2` + дополнительные (подстановка функций, ...)

Таблица 1: наборы оптимизаций в GCC/Clang

Предустановленные наборы оптимизаций (окончание)

Ключ	Краткое описание
`-Os`	Оптимизации `-O2`, которые, как правило, не увеличивают размеров кода, + оптимизации, направленные на уменьшение размеров кода.
`-Og`	Оптимизации, не мешающие отладке.
`-Ofast`	Оптимизации `-O3` + оптимизации, которые могут нарушить соответствие поведения программы стандартному.

Таблица 2: наборы оптимизаций в GCC/Clang (окончание)

Ключ	Эквивалент
Отсутствие ключей “`-O...`”	`-O0`
`-O`	`-O1`

Таблица 3: сокращённые записи ключей

Включение оптимизаций в наборы

	-O0	-O1	-O2	-O3	-Os	-Og	-Ofast
-fdce	✓	✓	✓	✓	✓	✓	✓
-fdse	✓	✓	✓	✓	✓	✓	✓
-ftree-ccp	✗	✓	✓	✓	✓	✓	✓
-fgcse	✗	✗	✓	✓	✓	✗	✓
-finline-functions-called-once	✗	✓	✓	✓	✓	✗	✓
-finline-small-functions	✗	✗	✓	✓	✓	✗	✓
-finline-functions	✗	✗	✗	✓	✓	✗	✓
-fschedule-insns2	✗	✗	✓	✓	✓	✗	✓
-fassociative-math	✗	✗	✗	✗	✗	✗	✓
-funroll-loops	✗	✗	✗	✗	✗	✗	✗

Таблица 4: некоторые оптимизации, входящие в наборы

Раскрутка циклов

Пример

```
int init_un(int n, int a[])
  __attribute__ ((optimize (1, "-funroll-loops")));

int init_un(int n, int a[])
{
  int i = 0;
  for (i = 0; i < 100; ++ i)
    a[i] = 0;
}

int init_no(int n, int a[])
{
  // ...
```

Раскрутка циклов (продолжение)

Пример

```
_init_un:
        pushl   %ebp
        movl    %esp, %ebp
        movl    12(%ebp), %edx
        movl    %edx, %eax
        addl    $400, %edx
L2:
        movl    $0, (%eax)
        movl    $0, 4(%eax)
        movl    $0, 8(%eax)
        movl    $0, 12(%eax)
```

Пример (продолжение)

```
        movl    $0, 16(%eax)
        movl    $0, 20(%eax)
        movl    $0, 24(%eax)
        movl    $0, 28(%eax)
        movl    $0, 32(%eax)
        movl    $0, 36(%eax)
        addl    $40, %eax
        cmpl    %edx, %eax
        jne     L2
        popl    %ebp
        ret
```

Раскрутка циклов (продолжение)

Пример (продолжение)

```
_init_no:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    $0, -4(%ebp)
        movl    $0, -4(%ebp)
        jmp     L15
L16:
        movl    -4(%ebp), %eax
        leal    0(,%eax,4), %edx
```

Пример (окончание)

```
        movl    12(%ebp), %eax
        addl    %edx, %eax
        movl    $0, (%eax)
        addl    $1, -4(%ebp)
L15:
        cmpl    $99, -4(%ebp)
        jle     L16
        nop
        leave
        ret
```

Прагмы настроек оптимизаций

Прагмы GCC

```
#pragma GCC optimize ("string" ...)
#pragma GCC push_options
#pragma GCC pop_options
#pragma GCC reset_options
```

Прагмы настроек оптимизаций

Пример

```
void sum(int n, int *a, int *b, int *c)
{
  int i;
#pragma GCC ivdep
// #pragma ivdep
  for (i = 0; i < n; ++ i)
    a[i] = b[i] + c[i];
}
```

Прагмы настроек оптимизаций

Пример

```
void sum(int n, int *restrict a, int *restrict b, int restrict *c)
{
  int i;
  for (i = 0; i < n; ++ i)
    a[i] = b[i] + c[i];
}
```

Прагмы настроек оптимизаций

Пример

```
void conv(int *a, int k, int c, int m)
{
  int i;
#pragma GCC ivdep
  for (i = 0; i < m; ++ i)
    a[i] = a[i + k] * c;
}
```

Векторные расширения

Пример

```
typedef int v4si __attribute__ ((vector_size (16)));

v4si z1 = { a, b, c, d };
v4si z2 = { /* .... */ };
// ...
v4si z, c;
z = (z1 - z2) * (z2 - z3) + 2 * z4 + 3;
c = (z1 < z2);
```

Межмодульные оптимизации

Пример

```
gcc -c -O2 -flto foo.c
gcc -c -O2 -flto bar.c
gcc -o myprog -flto -O2 foo.o bar.o
```