Лекция 3. Оптимизации, управляемые профилированием

Инструменты разработки быстрых программ

27 ноября 2017 г.

Ввведение

Определение

Оптимизация, управляемая профилированием: (*profile-guided optimization, PGO, profile-directed feedback, . . .*) — техника автоматической оптимизации, использующая профилирование для улучшения производительности времени выполнения программы, а также её размера.

Поддерживающие компиляторы, минимальная версия

- GCC 3.4
- clang 3.5.0
- Intel C/C++/Fortran Compiler 7.0 (?)
- Microsoft Visual C++ 2005

Порядок выполнения оптимизации

Порядок выполнения

1. Инструментирование (instrumenting) или подключение отсчётов (sampling)
2. Тренировка (training)
   - Несколько раз
3. Оптимизирующая компиляция с обратной связью (feedback compilation)

Способы сбора данных

Свойство	Инструментирование	Семплирование
Сохранение размера кода и времени выполнения	✗	✓
Воспроизводимость результатов	✓	✗
Построение покрытия кода	✓	✗
Бóльшая детальность результатов	✓	✗

Таблица 1: сравнение методов сбора информации

Запуск этапов оптимизации для GCC

Пример (Этапы 1–3)

```
rem 1 ------------------------------------------------
g++ in_1.cpp -c -o in_1_inst.o -O2 -fprofile-generate=dir
...
g++ in_1_inst.o ... -o prog_inst -fprofile-generate=dir
rem 2 ------------------------------------------------
prog_inst
...
rem 3 ------------------------------------------------
g++ in_1.cpp -c -o in_1_pgo.o -O2 -fprofile-use=dir
...
g++ in_1_pgo.o -o prog_pgo -fprofile-use=dir
```

Запуск этапов для Intel C++ Compiler

Пример (Этапы 1–3)

```
rem 1 ----------------------------------------------------
icl /Qprof-gen=threadsafe /O2 /Qprof-dir dir in_1.cpp ...
rem /Qcov-gen
icl  in_1..obj ... /Fe prog_inst
rem 2 ----------------------------------------------------
prog_inst
...
rem 3 ----------------------------------------------------
icl /Qprof-use /Qipo /Qprof-dir dir in_1.cpp ...
```

Запуск этапов для Intel C++ Compiler, аппаратно

Пример (Этапы 1–3)

```
# 1 ------------------------------------------------
icl -prof-gen-sampling -g in_1.cpp ...
icl  in_1..o ... prog_inst
# 2 ------------------------------------------------
/opt/intel-compiler/bin64/amplxe-pgo-report.sh prog_inst
...
# 3 ------------------------------------------------
icl -prof-gen-sampling -g -ipo in_1.cpp ...
```

Использование оптимизаций

Правила запуска этапов

- При использовании семплирования на (1) желательно отключить подстановку функций, остальные оптимизации лучше оставить.
- Желательно минимизировать изменения в коде после (2) до (3).
- Если изменений кода много $\Rightarrow$ повторить (1).
- Желательно запускать (2) на типичных данных.

Реализация инструментирования

Собираемые данные

- Вход/выход функции: время.
- Вход/выход в циклы (внутренние/внешние): время, итерации.

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Граф вызовов функций (Call graph)

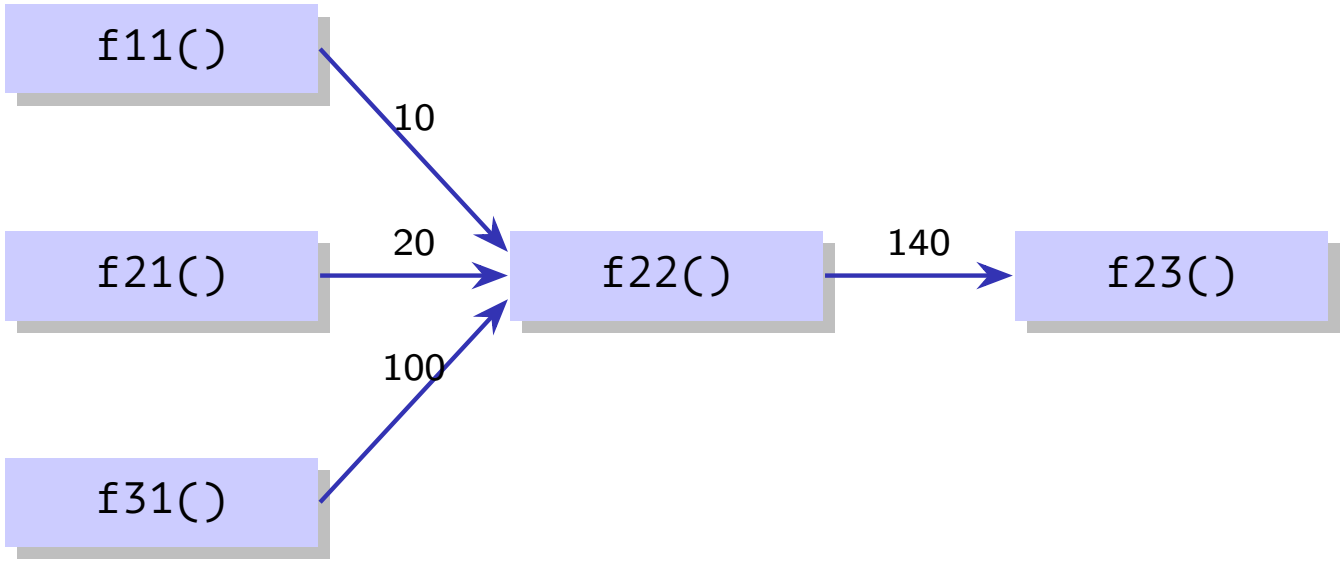

Рис. 1: граф вызовов функций вместе со статистикой

Граф вызовов функций (Call graph)

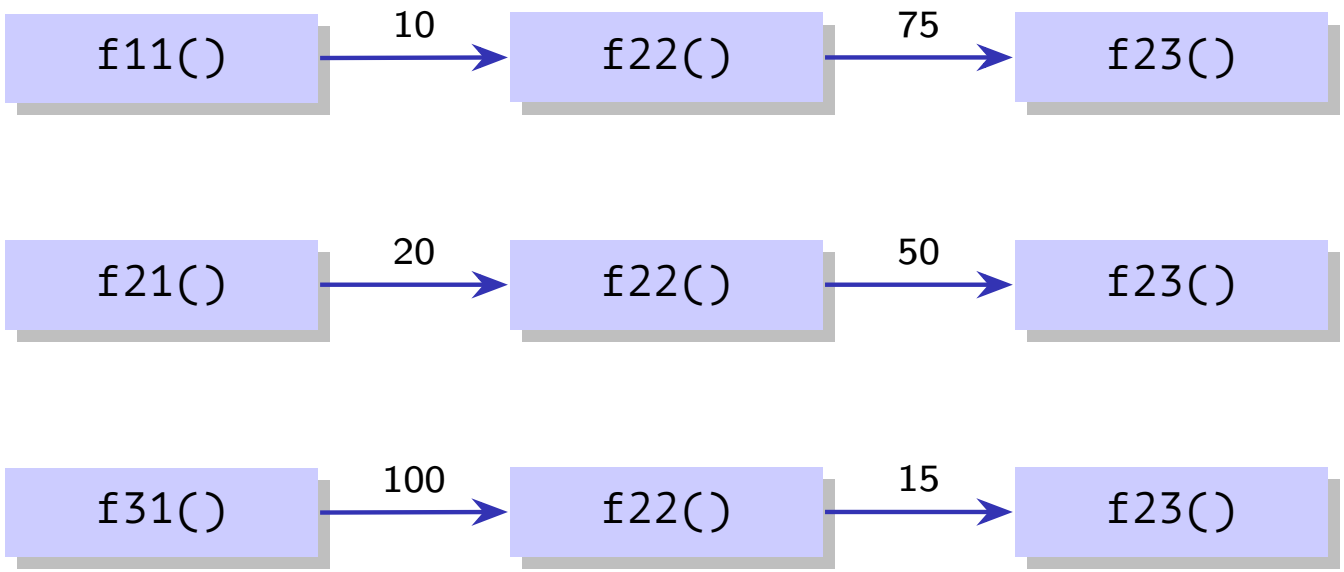

Рис. 1: граф вызовов функций вместе со статистикой

Граф вызовов функций (Call graph)

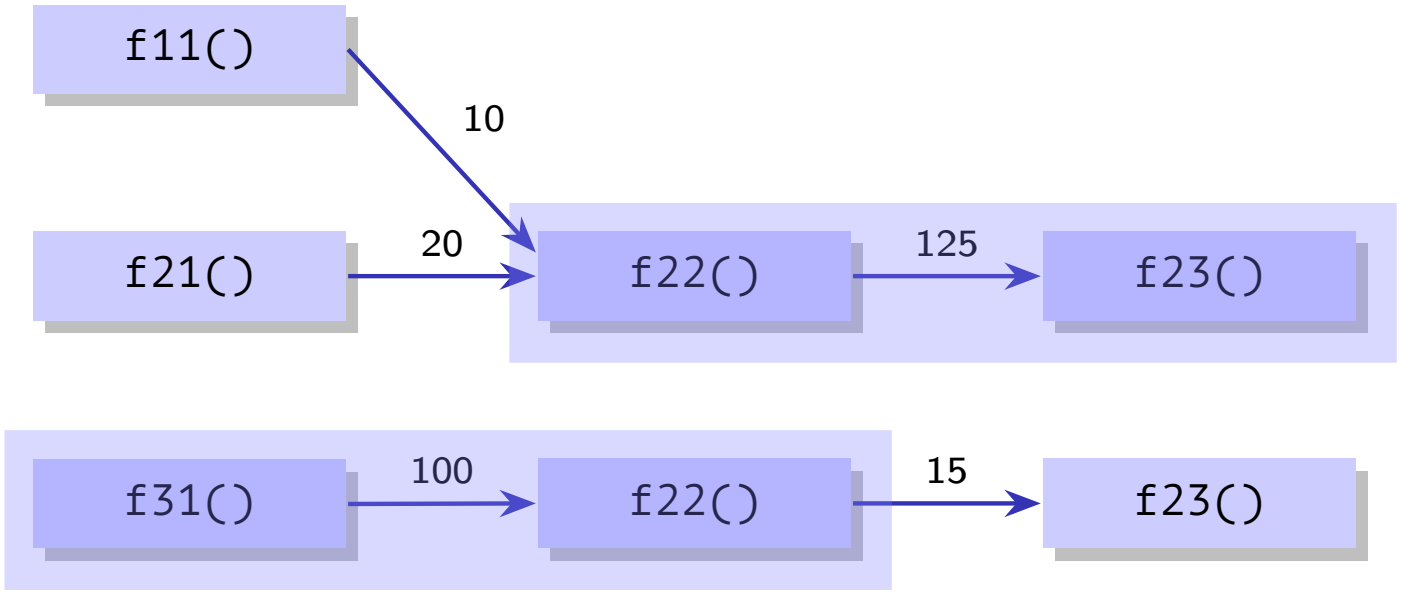

Рис. 1: граф вызовов функций вместе со статистикой

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Предсказание виртуального вызова

Пример (исходный)

```
void process(Base *pBase)
{
  // ...
  pBase->f();
  // ...
}
```

Пример (преобразованный фрагмент)

```
void process(Base *pBase)
{
  // ...
  if (typeid (*pBase) == typeid (Derived1))
    // подстановка pBase->Derived1::f();
  else
    pBase->f();
  // ...
}
```

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Граф потока управления (Control flow graph)

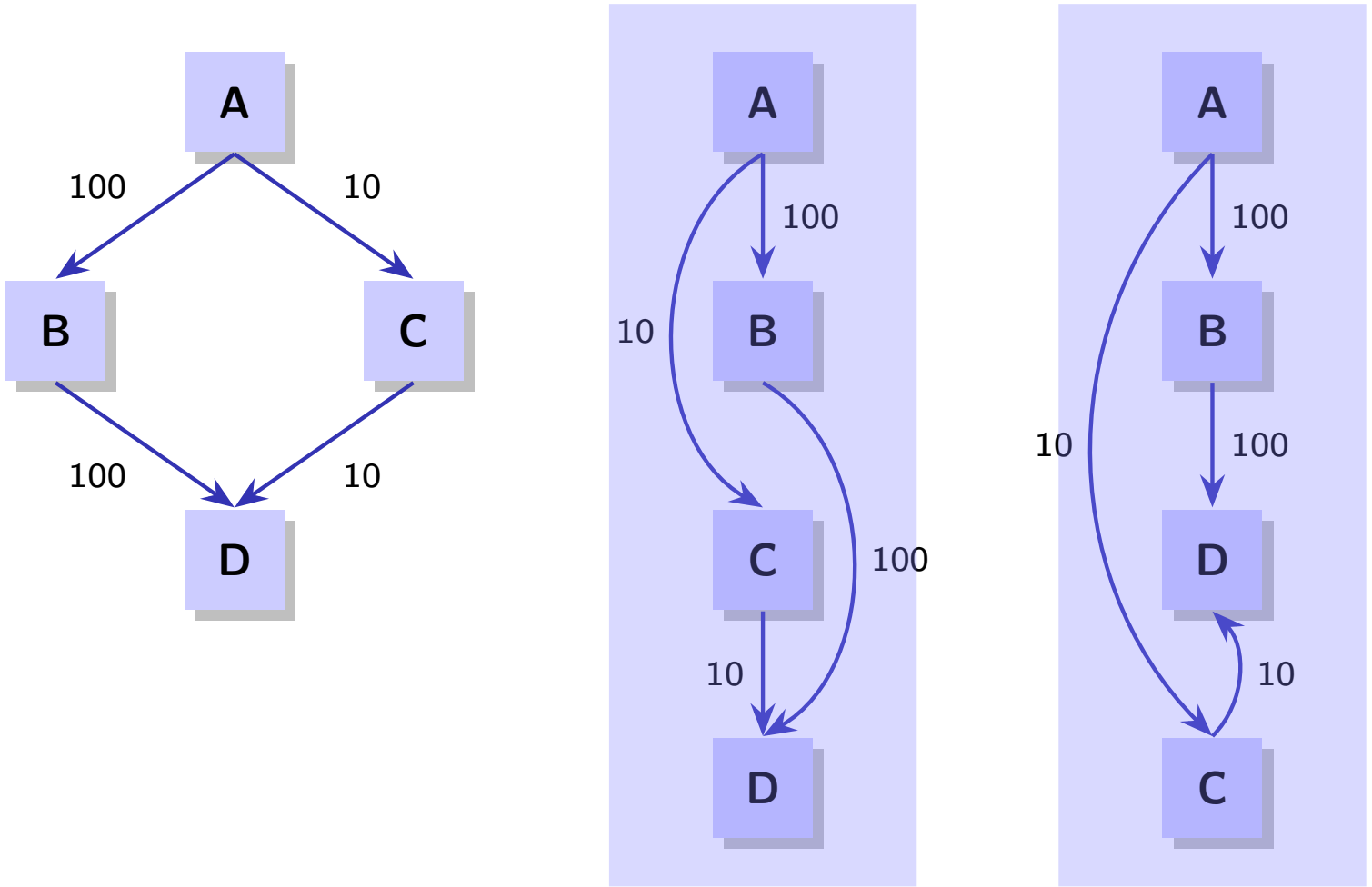

Рис. 2: оптимизации с использованием графа потока управления

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Граф потока управления (Control flow graph)

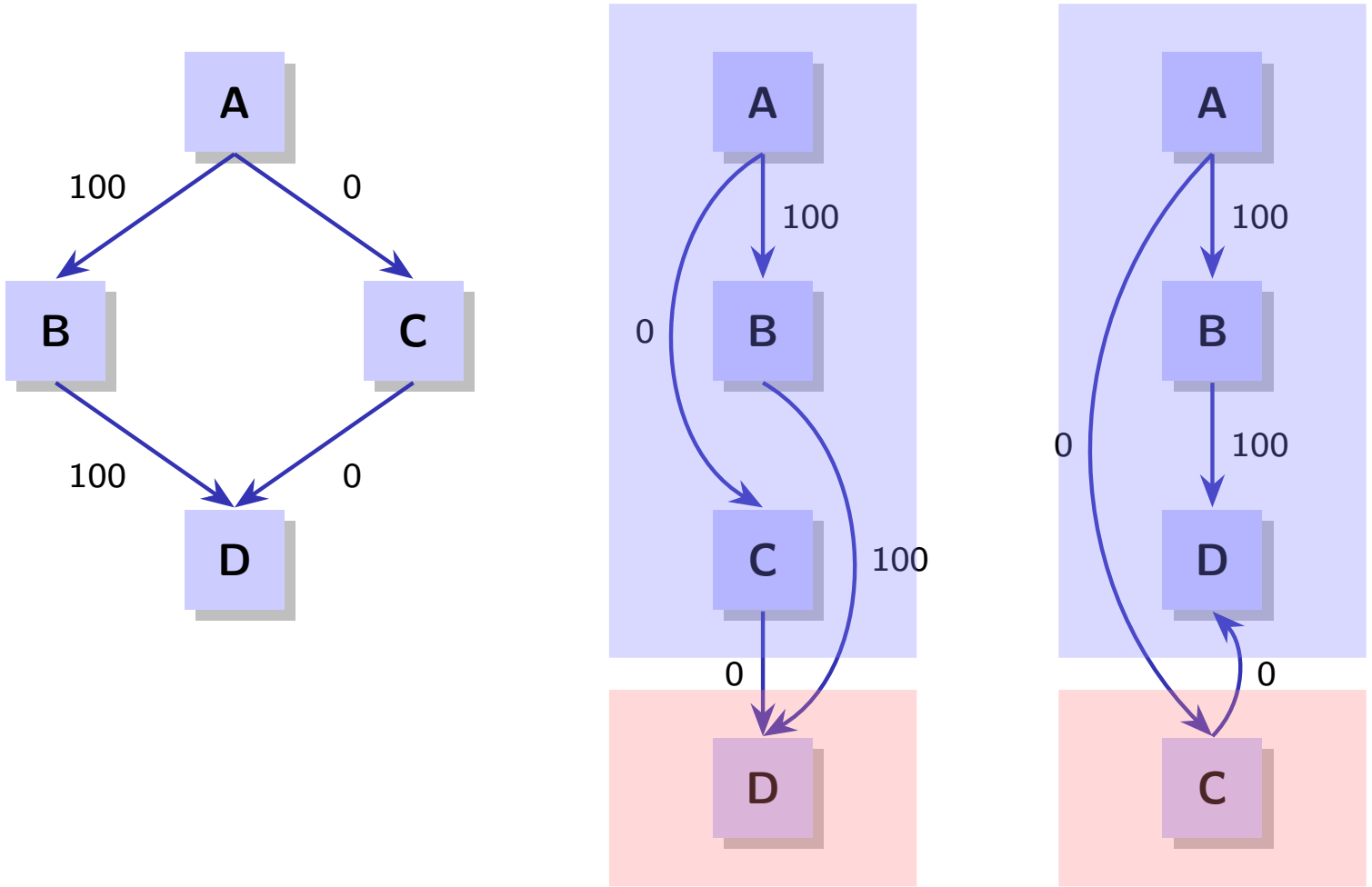

Рис. 2: оптимизации с использованием графа потока управления

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Граф вызовов функций (Call graph)

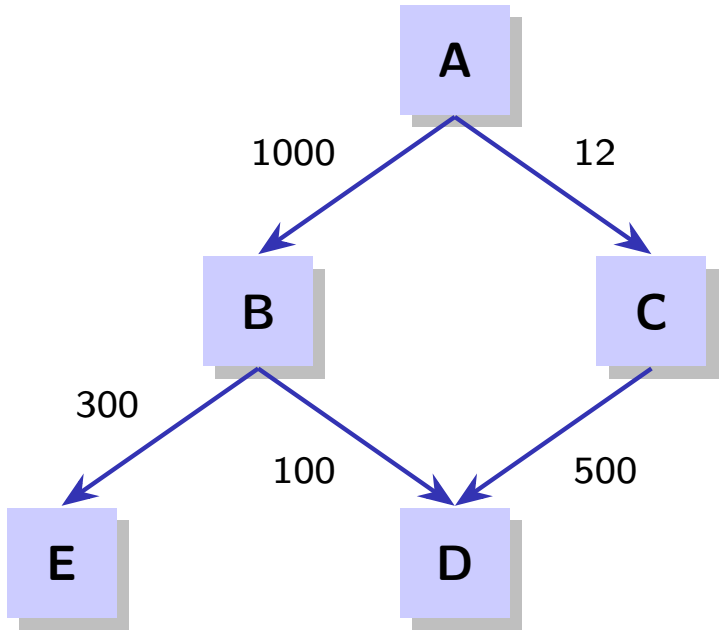

Рис. 3: оптимизации с использованием графа вызовов

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Оптимизация условного ветвления

Пример (исходный)

```
switch (n)
{
  case 1:
    // ...
    break;
  case 2:
    // ...
    break;
  default:
    // ...
}
```

Пример (преобразованный)

```
switch (n)
{
  default:
    // ...
    break;
  case 1:
    // ...
    break;
  case 2:
    // ...
    break;
}
```

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Использование профилирования

Оптимизации, управляемые профилированием

- Подстановка функций (inlining)
- Предсказание виртуального вызова (virtual call speculation)
- Размещение регистров (register allocation)
- Оптимизация базовых блоков (basic block optimization)
- Выделение мёртвого кода (dead code separation)
- Выделение обработчиков исключения (exception handling code separation)
- Размещение функций (function layout)
- Оптимизация условного ветвления (conditional branch optimization)
- Оптимизация встроенных функций обработки памяти (memory intrinsics)
- Оптимизации размера/скорости (speed/size optimizations)
- Векторизация циклов (loop vectorization)

Пример: сортировка с проверкой

Пример

```
int main()
{
  const int cnMax = 1000000000;
  IntVector v(cnMax);
  const time_t cuStart = time(0);
  generate(v.begin(), v.end(), Generator(cnMax));
  sort(v.begin(), v.end());
  const IntVector::const_iterator ci = adjacent_find(
    v.begin(), v.end(), greater <int> ());
  const bool cbSorted = (ci == v.end());
  const time_t cuEnd = time(0);
  // ...
}
```

Варианты сборки

Пример

```
> g++ sort.cpp -O0 -o a_no
> g++ sort.cpp -O2 -o a_opt
> g++ sort.cpp -O2 -o a_prof -fprofile-generate=profile
> g++ sort.cpp -O2 -o a_pgo -fprofile-use=profile
```

Результаты экспериментов

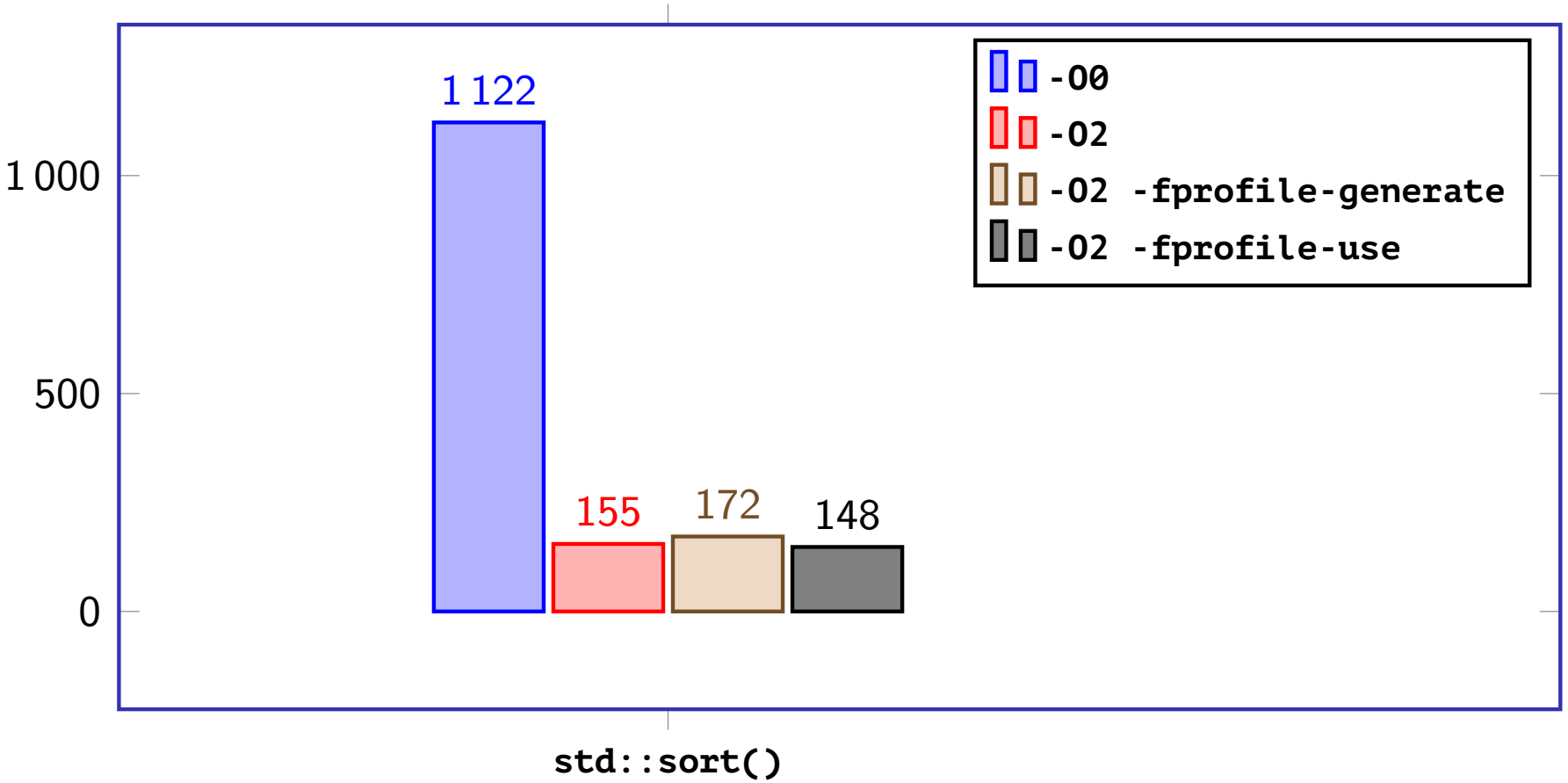

Рис. 4: результаты запуска программ

Ссылки


Asthana A. — Visual C++ Team Blog : Profile Guided Optimization (PGO) — Under the Hood. — 05/27/2013. — URL: https://blogs.msdn.microsoft.com/vcblog/2013/05/27/profile-guided-optimization-pgo-under-the-hood/ (visited on 11/27/2017).


Clang 6 documentation : — Clang Compiler User's Manual : Profile Guided Optimization. — URL: http://clang.llvm.org/docs/UsersManual.html#profile-guided-optimization (visited on 11/27/2017).


Intel® C++ Compiler 18.0 Developer Guide and Reference : — Profile-Guided Optimization (PGO). — URL: https://software.intel.com/en-us/cpp-compiler-18.0-developer-guide-and-reference-profile-guided-optimization-pgo (visited on 11/27/2017).


Profile-Guided Optimizations. — URL: https://msdn.microsoft.com/en-us/library/e7k32f4k.aspx (visited on 11/27/2017).