Опубликовано в материалах 2-й межвузовской научной конференции по проблемам информатики СПИСОК-2011, с. 377-379.

Е. В. Смирнов
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Применение генетических алгоритмов для локальной оптимизации программного кода

Скорость работы всегда оставалась одним из основных критериев оценки качества программного обеспечения. По этой причине стадия оптимизации в той или иной мере всегда присутствует в процессе разработки. И если затратная тонкая ручная оптимизация применяется только к тем программам, в которых критична каждая лишняя миллисекунда работы, то использование автоматических средств ускорения кода дешево и подходит всем, позволяя при этом добиться заметного повышения скорости.

Одним из этапов работы, который в той или иной мере включают в себя все современные оптимизирующие средства, является так называемая «локальная оптимизация». Эта оптимизация представляет собой поиск определенных небольших последовательностей низкоуровневых инструкций (на уровне ассемблера) и замена их на более эффективный аналог [1]. Локальный оптимизатор использует таблицу правил преобразования фрагментов кода, в которой содержатся неоптимальные наборы инструкций и то, чем их следует заменить. Обычно такая таблица составляется либо вручную (и может содержать лишь ограниченный набор достаточно простых

правил), либо с помощью полного перебора наборов инструкций (что требует значительных затрат времени, как описано в [2,3])

Цель работы

Целью данной работы является проверка эффективности применения генетических алгоритмов как замены полного перебора при поиске лучшего (в данном случае в смысле «работающего быстрее») набора команд.

Ход работы

Была разработана реализация эволюционного процесса, оперирующего наборами инструкций и при этом поддерживающего их корректность (возможность запуска) и эквивалентность исходному набору. Были созданы реализации, позволяющие производить оптимизацию байткода виртуальной машины *Java* и инструкций архитектуры *x86*. На данный момент поддержаны только некоторые подмножества наборов команд целевых платформ, в основном различные арифметические действия и базовые операции с регистрами, памятью, стеком.

Полученное программное решение может быть в дальнейшем расширено для поддержания других целевых платформ. Оно представляет собой универсальное клиент-серверное приложение с поддержкой распределенных вычислений и сохранением всех накопленных успешных оптимизаций в базу данных. Такое решение может в дальнейшем быть использовано либо для накопления наборов правил для использования в обычных локальных оптимизаторах, либо для индивидуальной оптимизации конкретных приложений.

В качестве эксперимента были проведены поиски более быстрых вариантов для нескольких наборов

инструкций, взятых из различных прикладных программ.

Пример

Из неоптимизированного исполняемого файла была взята последовательность из двух инструкций (№1):

```
xor eax, -1
and eax, ebx
```

С помощью широко используемых компиляторов языка *C++ MSVC* и *ICC*, при включенной максимальной оптимизации по скорости, был получен улучшенный вариант (№2, оба компилятора дали одинаковый результат):

```
not eax
and eax, ebx
```

Результатом работы генетического алгоритма стал вариант (№3):

```
xor eax, ebx
and eax, ebx
```

Варианты 2 и 3 превосходят исходный по скорости выполнения приблизительно в **полтора** раза, при этом вариант №3 обгоняет вариант №2 еще на **5-10%**.

Результаты

Результат работы генетического процесса показал его высокую эффективность в деле оптимизации. Были найдены как стандартные решения (например, замена целочисленного умножения на два на сдвиг), так и нетривиальные оптимизации, для нахождения которых с помощью прочих методов потребовался бы глубокий анализ свойств вычисляемых выражений и детальное

знание целевой архитектуры.

Как было показано на примере выше, генетические алгоритмы способны также улучшать результаты работы уже существующих оптимизирующих средств.

Генетические алгоритмы позволяют найти достаточно хорошее решение за намного меньшее число итераций, чем полный перебор. Например, для одной из рассмотренных последовательностей длиной в 11 инструкций, более выгодный вариант в большинстве случаев выращивается за 100-150 поколений, по 100 особей в каждом, тогда как всего возможно около 10^{20} наборов инструкций. Результат при этом оказывается лучше выдаваемого другими средствами на 3-15%.

Список используемых источников

1. *McKeeman W.M.* Peephole Optimization // Communications of the ACM, № 8. Нью-Йорк, 1965. С. 443-444.

2. *Bansal S., Aiken A.* Automatic Generation of Peephole Superoptimizers // Proceedings of the 2006 ASPLOS Conference

3. *Massalin H.* Superoptimizer - A Look at the Smallest Program // ACM SIGPLAN Notices. №22. Нью-Йорк, 1987.