УДК 004.4'422

Удаление немёртвых процедур, не влияющих на результат программы

***В.Е.Шампаров*[1], *А.Л.Маркин*[1]**
[1]АО «МЦСТ»

Одной из задач оптимизирующей компиляции является удаление так называемого мёртвого кода, результаты выполнения которого не используются. Алгоритм выявления и удаления мёртвых инструкций внутри процедуры известен [1] и применяется в популярных оптимизирующих компиляторах [2, 3].

Процедуру назовём мёртвой, если она не содержит побочных эффектов и все операции её вызова также мертвы. Удаление мёртвых процедур выполняется по правилам, аналогичным классическому алгоритму DCE.

В рамках данной работы выявлены такие процедуры, которые работают со статическими объектами, имеющими ограниченную процедурой лексическую область видимости, и не влияют на результат исполнения программы при любых входных данных. Классический метод не позволяет удалять подобные процедуры, так как формально они не мертвы. Поэтому был разработан охарактеризованный ниже алгоритм выявления и удаления немёртвых процедур, не влияющих на результаты программы. Он реализован в рамках компилятора LCC для микропроцессоров семейства «Эльбрус».

Алгоритм предполагает два прохода по представлению программы. В первом проходе каждая процедура преобразуется к виду без возвращаемого значения, если оно не используется после всех вызовов процедуры. Во втором проходе удаляются вызовы всех процедур, соответствующих следующим условиям:

1. На процедуру не был взят адрес
2. Процедура листовая, иными словами не вызывает других процедур
3. Процедура не имеет возвращаемого значения
4. Внутри процедуры нет: записей по указателю, volatile чтений и записей
5. Все записи по имени работают с локальными или статическими объектами процедуры

Измерение изменения производительности при применении оптимизации проводилось на наборе бенчмарков SPEC CPU2006 [4], причём компиляцию выполнили на машине с архитектурой процессора x86-64 с пиковым набором опций, а исполнение — на машине с архитектурой процессора «Эльбрус». В таблице 1 показаны результаты измерения. В колонках «отношение времён компиляции» и «отношение времён выполнения» показаны отношения времён соответствующей работы без преобразования к времени работы с преобразованием. В порядке строк приводятся только результаты четырёх тестов с наибольшими отклонениями от 1 по скорости компиляции, результаты двух тестов с наибольшими отклонениями от 1 по скорости исполнения и геометрическое среднее от результатов всех тестов.

Данные таблицы 1 показали, что ускорение компиляции достигает 1,18, замедление — 0,83, и при этом доля времени работы алгоритма не превышает 0,5%. Из этого можно сделать вывод, что применение данного алгоритма меняет контекст для применения других оптимизаций.

Для сравнения была выполнена попытка измерить те же отношения на компьютере с архитектурой процессора x86-64 с компилятором gcc версии 7.1.0 с опциями -flto -O3. Установлено, что в ходе LTO оптимизаций этот компилятор не удаляет процедуры, которые были удалены реализованным алгоритмом.

Дальнейшее улучшение алгоритма возможно в следующих направлениях: адаптация алгоритма для сборки в помодульном режиме и добавление анализа нелистовых процедур.

Название теста	Отношение времён компиляции	Отношение времён выполнения	Доля времени работы дополнения
400.perlbench	0,91	1,00	0,001%
403.gcc	0,83	1,00	0,06%
444.namd	1,13	1,00	0,005%
462.libquantum	1,03	1,01	0,02%
483.xalancbmk	1,18	1,00	0,4%

433.milc	0,98	0,99	0,01%
Gmean	0,99	1,00	0,02%

Табл. 1. Ускорение работы тестов из набора SPEC-2006 (больше 1 – ускорение, меньше 1 – замедление)

Литература

1. *Muchnick S.* Advanced compiler design and implementation. Morgan Kaufman. –San Francisco, pp. 592-593. 1997.
2. *Lattner C.* LLVM: An infrastructure for multi-stage optimization. University of Illinois, p. 26. 2002.
3. *Novillo D.* GCC. An Architectural Overview, Current Status and Future Directions. // Proceedings of the Linux Symposium, – Tokyo, 2006. Pp. 185-200.
4. http://www.spec.org/ [Электронный ресурс]