Оптимизация кода функционально-потокового языка программирования Пифагор*

В.С. Васильев[1]

Сибирский федеральный университет, институт Космических и информационных технологий[1]

Язык Пифагор предназначен для параллельного программирования и поддерживает функционально-потоковую модель вычислений и специальные конструкции для описания параллелизма. Ответственность за эффективное преобразование этих конструкций в команды вычислителя ложится на кодогенератор или интерпретатор. При этом программа может быть написана неэффективно, что обуславливает актуальность применения методов оптимизации. Целью работы является разработка инструментов оптимизации кода для языка программирования Пифагор.

Язык Пифагор в контексте оптимизации обладает рядом особенностей поэтому в рамках работы выполняется не только адаптация ряда известных методов оптимизации, но и разработка новых. Программа на языке Пифагор представляет собой набор функций. Каждая функция задает информационный граф, отражающий зависимости по данным между операторами. Графы, являющиеся исходными данными и формируемые в результате работы оптимизатора, представляют собой текстовые файлы. Формат файлов, а также процесс их интерпретации описан в [1].

В связи с тем, что управление вычислениями в Пифагор осуществляется по готовности данных - графы потока управления, графы зависимостей по данным и графы программных зависимостей (ГПЗ), построенные для программ этого языка будут эквивалентны [2–4]. Кроме того, в связи с отсутствием в языке оператора разрушающего присваивания, любая программа на Пифагор всегда находится в форме статического единственного присваивания [5], и ГПЗ будет эквивалентен def-use графу [6], однако в def-use графе выделяются не только def-use, но и use-def, def-def зависимости.

В связи с тем, что при оптимизации зачастую требуется выполнять поиск «фрагмента определенного вида», при этом переходить как в направлении def-use, так и use-def зависимостей – то наиболее подходящей формой представления программ для языка Пифагор является def-use граф. Однако в этом графе в связи с указанными выше особенностями языка будут отсутствовать def-def зависимости.

Несмотря на то, что в языке Пифагор нет явно выделенного оператора цикла, повторяющиеся вычисления могут задаваться с помощью рекурсии или параллельных списков. Для обоих вариантов возможны преобразования, аналогичные оптимизации инварианта цикла [7]. В процессе оптимизации инвариант выносится из блока повторяющихся вычислений во вспомогательную функцию. При оптимизации инварианта в вычислениях над списками в программе выполняется поиск конструкций следующего вида:

```
1. Len << XList:|;
2. YList << (Y, Len):dup;
3. (XList, YList):#:[]:foo;
```

В данном случае функция *foo* выполняется над параллельным списком пар соответствующих элементов списков *XList* и *YList* (в общем случае возможно произвольное число таких элементов). Элемент *YList* в каждой паре создается с помощью операции *dup* (копирования), поэтому для каждой итерации (вызова функции *foo*) это значение будет одина-

* Исследование выполняется при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

ковым. В связи с этим инвариантом в функции *foo* можно считать вычисления, зависящие только от констант и неизменяемых параметров (таких как элементы *YList*).

В модели вычислений языка Пифагор определен ряд эквивалентных преобразований, которые можно использовать при оптимизации. В процессе преобразования из графа выбираются подграфы определенного вида и заменяются на более простые в вычислительном плане эквивалентные подграфы [8].

Наиболее интересным с точки зрения реализации для языка Пифагор является оптимизация общих подвыражений [9]. Для проведения оптимизации необходимо найти во входном графе программы подграфы, выполняющие одинаковые последовательности операций над одинаковыми аргументами. Затем - выбрать из них подграфы, которые можно заменить с учетом иерархической вложенности задержанных списков. При удалении подграфа выполняется замена обращений к его вершинам на обращения к аналогичным вершинам базового подграфа.

В результате проведенной работы разработан программный продукт реализующий представленные методы оптимизации и позволяющий повысить эффективность формируемого выходного представления, используемого при выполнении программы.

Литература

1. Легалов А.И., Васильев В.С., Матковский И.В., Ушакова М.С. Инструментальная поддержка создания и трансформации функционально-потоковых параллельных программ. Труды Института системного программирования РАН. № 5 (29), 2017. - С. 165-184.

2. Keshav Pingali, Micah Beck, Richard« Johnson, Mayan Moudgill, Paul Stodghill, Dependence Flow Graph: An Algebraic Approach to Program Dependencies. Department of Computer Science, Cornell University, Ithaca, NY 14853

3. Heffernan, M., Wilken, K. Data-Dependency Graph Transformations for Instruction Scheduling // Journal of Scheduling, 8(5), October 2005. - 427-451

4. Nick P. Johnson, Taewook Oh, Ayal Zaks, David I. August, Fast Condensation of the Program Dependence Graph, Acm Sigplan Notices 48 (6), June 2013. - 39-50

5. R. Cytron, J. Ferrante, B. Rosen, M. Wegman, K. Zadeck., Efficiently Computing Static Single Assignment Form and the Control Dependence Graph, ACM Transactions on Programming Languages and Systems, 13(4), 1991. - 451-490

6. Дроздов А. Ю., Новиков С. В., Боханко А. С., Галазин А. Б., Глобальный граф потока данных и его роль в проведении оптимизирующих преобразований программ. // Высокопроизводительные вычислительные системы и микропроцессоры: сборник трудов ИМВС РАН, Выпуск N8, 2005

7. Ахо А.В., Лам М.С, Сети Р., Ульман Дж.Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер.с англ. - М. : "И.Д. Вильямс". - 2008. - 1184 с.

8. Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии : журнал. — 2005. — Т. 10. — № 1. — С. 71-89.

9. Филиппов А.Н. Метод нумерации значений и использование его результатов в оптимизациях программ // Информационные технологии. - No4, 2009. - С. 43 — 49