УДК 004.432

ВОССТАНОВЛЕНИЕ МНОГОМЕРНОЙ ФОРМЫ ОБРАЩЕНИЙ К ЛИНЕАРИЗОВАННЫМ МАССИВАМ В СИСТЕМЕ SAPFOR

Н. А. Катаев[1], В. Н. Василькин [2]

[1]*Институт прикладной математики им. М.В. Келдыша РАН, г. Москва*
[2]*Московский государственный университет им. М.В. Ломоносова, г. Москва*

[1]kataev_nik@mail.ru, [2]volandtymim@gmail.com

Аннотация

Система автоматизированного распараллеливания SAPFOR (System FOR Automated Parallelization) включает инструменты для анализа и преобразования программ, основной ее целью является снижение сложности распараллеливания программ. Система SAPFOR ориентирована на исследования многоязыковых вычислительных комплексов, разрабатываемых на языках программирования Фортран и Си. Для анализа программ в этой системе используется низкоуровневое их представление в виде LLVM IR, которое позволяет проводить различные оптимизации с целью повышения качества анализа программ. При этом оно теряет некоторые особенности программы, отражаемые ее представлением на языке высокого уровня. Одной из таких особенностей является многомерная структура используемых массивов. Анализ зависимостей по данным является одним из ключевых при исследовании возможности параллельного выполнения программ. При этом такой анализ относится к классу NP-трудных задач. Знание многомерной структуры массивов позволяет во многих случаях учесть структуру индексных выражений в обращениях к массивам и снизить сложность проводимого анализа. Кроме того, использование многомерных массивов позволяет повысить уровень параллелизма в программе за счет использования многомерных решеток процессоров и распараллеливания гнезд циклов, а не отдельных циклов в гнезде. Данная возможность естественным образом поддерживается в DVM-системе. В настоящей работе рассмотрен подход, применяемый в системе SAPFOR для восстановления формы многомерных массивов и обращений к ним по их линеаризованному представлению в LLVM IR. Предложенный подход был

успешно протестирован на различных приложениях, включая тесты производительности из набора NAS Parallel Benchmarks.

***Ключевые слова**: анализ программ, автоматизация распараллеливания, SAPFOR, DVM, LLVM*

ВВЕДЕНИЕ

Использование многомерных массивов характерно для многих вычислительных задач. Например, они позволяют описать свойства объекта в различных точках многомерного вычислительного пространства. Система DVM [1, 2] позволяет учесть многомерную структуру задачи и отобразить измерения массивов на измерения многомерной решетки процессоров. Таким образом, при распараллеливании, в том числе, полуавтоматическом, необходимо учитывать особенности использования многомерных структур данных в программе.

Система SAPFOR [3, 4] выполняет распараллеливание программ с помощью DVMH-языков. Распараллеливание в модели DVM позволяет согласованно распределять данные по процессорам, образующим многомерную процессорную решетку. Использовав директиву *align,* можно указать правила, задающие расположения элементов массивов относительно друг друга на процессорах системы. Данные правила должны быть заданы в виде аффинных выражений, связывающих измерения массивов. Чтобы избежать дополнительных накладных расходов на коммуникации, используемые правила должны отображать элементы массива, используемые на одной и той же итерации цикла на один и тот же процессор. Таким образом, обращения к массивам в циклах и форма индексных выражений в этих обращениях диктуют правила расположения элементов массива друг относительно друга.

Сохранение многомерной структуры данных также позволяет значительно повысить точность и снизить вычислительную сложность анализа зависимостей по данным, который является неотъемлемым этапом распараллеливания программ, как ручного, так и автоматизированного. Если выражения, вычисляющие смещения относительно начала массива в тестируемых обращениях к элементам данного массива, являются линейными относительно индексных переменных объемлющих циклов, то проверка факта наличия/отсутствия зависимости по данным сводится к решению задачи целочисленного линейного программирования, которая является

NP-трудной. Чтобы понизить сложность решаемой задачи, используют эвристики, вводя ограничения на рассматриваемые индексные выражения, и выполняют попарное сравнение индексных выражений, соответствующих одному и тому же измерению массива. Знание многомерной структуры массивов позволяет выделить сравниваемые индексные выражения из линеаризованного представления обращений к массивам. В работе [5] рассмотрены тесты, применяемые в зависимости от вида индексных выражений и обладающие существенно меньшей сложностью.

Стоит отметить, что если размеры измерений массива не являются константными, то линеаризованное представление обращений к элементам массива перестает быть линейным, и проверить наличие зависимостей по данным, не учитывая многомерную структуру массива, становится практически невозможно.

Система SAPFOR при распараллеливании прикладных программ в модели DVM опирается на их представление в виде LLVM IR [6]. Это позволяет проводить анализ программ для разных языков программирования (Фортран, Си), в том числе, решая одну из проблем, характерных для больших вычислительных комплексов (многоязыковость) [7], а также обеспечивает возможность скрытого от пользователя преобразования программ для повышения качества проводимого анализа [8, 9]. Использование LLVM для анализа также устраняет необходимость учитывать синтаксические особенности разных языков и анализировать все многообразие доступных языковых конструкций. Для сохранения специфики исходного языка достаточно воспользоваться отладочной информацией, которая представлена в стандартизованном виде (DWARF [10]) и может быть расширена при необходимости. Недостатком применения LLVM является использование линеаризованного представления обращений к элементам массива, которое полностью соответствует способу хранения массива в памяти, но скрывает многомерную структуру используемых данных. Явным образом в LLVM IR могут быть представлены только массивы с фиксированными размерами каждого измерения (например, тип [100 x [200 x float]] может быть использован для объявления в LLVM IR массива, содержащего 100*200 элементов).

Целью делинеаризации может быть как восстановление формы массивов, которые были многомерными в исходном коде, но оказались линеаризованы при построении LLVM IR, так и поиск для массивов, представленных в исходном коде программы в линеаризованном виде, эквивалентного описания в виде многомер-

ных массивов. В рамках данной работы нас в первую очередь интересует восстановление формы массивов. Это связано с тем, что ограничением, накладываемым на распараллеливаемую программу DVM-системой, является использование именно многомерных массивов в исходном коде программы.

1. ДЕЛИНЕАРИЗАЦИЯ В LLVM

В работе [11] описан подход к делинеаризации, применяемый в LLVM. Будучи частично реализованным в LLVM, данный подход изначально ориентирован на использование в проекте Polly [12], нацеленном на оптимизацию циклов и повышение локальности используемых данных, а также распараллеливание для систем с общей памятью (OpenMP, GPU). В данном случае оптимизация выполняется над LLVM IR, поэтому не требуется соотнесение делинеаризованных массивов с объектами исходного кода. Кроме того, оптимизации подвергаются отдельные циклы, и не требуется согласованная делинеаризация массивов в рамках всей программы. Требование согласованной делинеаризации отличает распараллеливание для систем с распределенной памятью, которое предполагает принятие решений по распределению данных для всей программы. В [11] речь идет о делинеаризации для массивов, измерения которых имеют только переменные размеры. Отдельно в Polly реализована делинеаризация для массивов с фиксированными размерами измерений. В LLVM для представления таких массивов используется встроенный тип многомерного массива, таким образом, делинеаризация фактически не требуется, так как структуру индексных выражений можно восстановить, исходя из типа массива. Массивы, часть измерений которых имеют переменные размеры, а часть – фиксированные, будут рассматриваться как массивы с переменными размерами, поэтому количество измерений и индексные выражения, используемые для доступа по каждому измерению, могут не соответствовать описанию массивов и обращениям к ним в исходной программе. Реализованная в Polly функциональность опирается на используемые в нем структуры данных и лишь частично входит в основную сборку LLVM. В связи с вышесказанным использование предложенных алгоритмов в SAPFOR напрямую оказывается невозможным.

Рассмотрим идею алгоритма, предложенного в [11]. В линеаризованной форме обращение A[S_0]…[S_N–1] к массиву A[D_0]…[D_N–1] будет иметь вид

(1) A + S_0 * D_1 *…* D_N–1 + … + S_N–1.

В данном случае D_I – размер измерения массива с номером I, а S_I – соответствующее индексное выражение, I принимает значения от 0 до N–1. Из данной формулы следуют следующие соотношения:

(2) C_N–1 * D_N–1=GCD(S_0 * D_1 * … * D_N–1, …, S_N–2 * D_N–1),

(3) C_I * D_I=GCD(S_0 * D_1 * … * D_N–1, … S_I–1 * D_I * … D_N–1) / (D_I+1 *…* D_N–1), 0<I<N–1.

Таким образом, чтобы вычислить размер измерения массива с номером I, необходимо найти наибольший общий делитель слагаемых в (1), расположенных слева от слагаемого I, и разделить его на произведение вычисленных ранее размеров измерений от I+1 до N–1.

Множитель C_I может быть не равен 1, например, если в каждом индексном выражении присутствует один и тот же множитель, функция GCD выполняет символьное вычисление наибольшего общего делителя среди всех своих параметров.

Основная сложность алгоритма делинеаризации состоит в том, чтобы выделить из суммы (1), вычисляющей адрес элемента массива, правильное количество слагаемых (равное количеству измерений в массиве), упорядочить их в соответствии с порядком измерений и определить значение коэффициента C_I. Количество слагаемых в сумме (1) может не совпадать с количеством измерений массива, если некоторые из индексных выражений равны нулю, являются константными одновременно с размерами массивов, на которые они умножаются, или при построении формулы (1) было выполнено раскрытие скобок.

Будем называть используемые для делинеаризации слагаемые термами. В работе [11] предложено использовать в качестве термов слагаемые, которые содержат произведение индексной переменной объемлющего цикла на переменные программы. Константные множители в этом случае игнорируются, а коэффициент C_I считается равным 1. Упорядочивание термов выполняется в соответствии с количеством входящих в них множителей. В результате делинеаризованное представление массива может не соответствовать представлению массива в исходной программе. Это является допустимым, так как основная цель делинеаризации в Polly – получить аффинные индексные выражения.

Делинеаризация в SAPFOR основана на идее алгоритма, использованного в Polly, но содержит некоторые особенности, которые позволяют соотнести восста-

новленную многомерную структуру массивов с многомерными массивами в исходной программе.

2. ДЕЛИНЕАРИЗАЦИЯ В SAPFOR

В первую очередь стоит отметить, что мы рассматриваем делинеаризацию для массивов, которые были многомерными в исходной программе. Для остальных массивов делинеаризация может проводиться с целью преобразования исходной программы, чтобы добиться использования в ней многомерных массивов. В этом случае согласованное с исходным кодом восстановление формы не требуется, и может быть использован подход, реализованный в LLVM.

Как было отмечено выше, для массивов, все измерения которых являются известными на этапе компиляции, делинеаризация не требуется. Таким образом, нас будут интересовать массивы, часть или все измерения которых вычисляются во время выполнения программы.

Для вычисления адреса элемента некоторого агрегатного типа данных в LLVM IR используется специальная инструкция getelementptr (см. Рис. 1).

```
%6 = zext i32 %I.0 to i64
%7 = mul nuw nsw i64 %6, %1
%arrayidx11 = getelementptr inbounds [2 x double], [2 x double]* %vla, i64 %7
%8 = zext i32 %J.0 to i64
%arrayidx13 = getelementptr inbounds [2 x double], [2 x double]* %arrayidx11, i64 %8
%arrayidx14 = getelementptr inbounds [2 x double], [2 x double]* %arrayidx13, i64 0, i64 1
```

Рис. 1. Пример LLVM IR вычисляющего смещение для доступа к элементу A[I][J][1] массива размерности M*N*2

Параметрами данной инструкции являются адрес начала участка памяти и одно или несколько значений, используемых для вычисления смещения относительно заданного адреса. В случае многомерных массивов параметрами данной инструкции являются слагаемые из (1), используемые для вычисления смещения по каждому измерению, либо индексное выражение (без умножения на размеры измерений).

Рассмотрим пример вычисления смещения на Рис. 1. Первая инструкция getelementptr сдвигает адрес начала массива A (%vla) на величину I*N (%7), следующая инструкция сдвигает полученный адрес на величину J*2 (%8). Стоит отметить, что умножение на размер последнего измерения массива выполняется неявно инструкцией getelementptr (регистр %8 содержит только значение переменной J). Это

связано с тем, что размер последнего измерения фиксирован, а сам массив A в LLVM IR имеет тип [2 x double] *.

Зависимость количества параметров инструкции getelementptr от количества измерений массива можно считать эвристикой (можно получить эквивалентное LLVM IR, заменив все инструкции getelementptr на Рис. 1 одной с двумя параметрами: адрес массива и предварительно вычисленное смещение), аналогичной той, которая используется в [11] для выделения термов. Это позволяет нам сделать вывод о количестве измерений в массиве и определить количество и порядок термов, необходимых для делинеаризации конкретного обращения к элементу массива.

При этом для вычисления количества измерений используются только инструкции, заведомо обращающиеся к отдельным элементам массива, такие как load и store. Это связано с тем, что другие инструкции, например, инструкции вызова функции, могут принимать в качестве параметра целое измерение массива, и количество операндов в инструкции getelementptr может оказаться меньше количества измерений. Другим источником несоответствия количества операндов и количества измерений может стать использование нулевых индексных выражений. Такие выражения будут опущены в инструкции getelementptr. Чтобы точнее определить количество измерений в массиве, одновременно рассматриваются все обращения к его элементам и среди них выбираются обращения с максимальным количеством измерений.

Во многих ситуациях количество измерений также доступно из отладочной информации. Кроме того, из отладочной информации доступны размеры измерений, известные на этапе компиляции, а также (в некоторых случаях) размеры, заданные с помощью переменных.

Далее, на основе формул (2) и (3) вычисляются размеры неизвестных измерений массива. Чтобы уменьшить вероятность ошибки, наибольший общий делитель вычисляется среди термов, полученных для всех обращений к элементам заданного массива внутри анализируемой функции, которые использовались при определении количества измерений массива. Тот факт, что термы определяются на основе инструкций getelementptr, позволяет рассматривать выражения, не содержащие индексные переменные циклов, а также являющиеся константами. Это также уменьшает вероятность того, что коэффициент C_I окажется отличным от 1, так

как в противном случае все слагаемые, входящие в состав всех индексных выражений по измерениям от 0 до I–1 во всех обращениях к элементам массива, умножались на одну и ту же величину.

Для вычисления наибольшего общего делителя выполняется операция символьного деления термов друг на друга. Допустим, необходимо вычислить наибольший общий делитель (НОД) для последовательности термов T_1, …, T_N. В соответствии с приведенными выше формулами каждый терм представляет собой произведение нескольких множителей. Для вычисления НОД все термы упорядочиваются в соответствии с количеством входящих в них множителей, начиная с наиболее коротких термов. Множители первого терма в последовательности рассматриваются как потенциально возможные множители, образующие НОД всей группы термов (обозначим данную группу множителей DIVS (от англ. divider)). Для оставшихся термов выполняется символьное деление на каждый множитель из DIVS, и если остаток от деления не равен 0, то множитель удаляется из DIVS. После обработки всей последовательности термов в DIVS остаются множители, являющиеся делителями каждого терма последовательности, и НОД принимается равным произведению данных множителей. Если DIVS не содержит ни одного множителя, то НОД найти не удалось, и делинеаризация для рассматриваемого массива оказывается невозможной.

После того как были вычислены размеры измерений массива, вычисляются индексные выражения для каждого обращения к массиву. Начиная с 0 измерения, выполняется деление термов на произведение размеров следующих измерений. Если количество термов для заданного обращения меньше, чем количество измерений массива, и текущий терм не делится на произведение измерений, то индексное выражение считается равным 0. Таким образом восстанавливаются нулевые индексные выражения, опущенные в инструкции getelementptr. Важно отметить, что в процессе восстановления индексных выражений участвуют все обращения к массивам, в том числе, те, которые не могли быть использованы для вычисления количества измерений (например, фактические параметры, передающие в вызов функции целое измерение массива).

Выражения, являющиеся операндами инструкции getelementptr, могут иметь достаточно сложный вид и вложенные приведения типов. Для качественной рабо-

ты алгоритма нахождения наибольшего общего делителя, а также выполнения символьного деления выражений друг на друга, необходимо проведение максимального раскрытия скобок в выражениях и выделение наиболее простых множителей. С точки зрения строгости вычислений, операции приведения типов могут изменять результаты вычислений, по этой причине раскрытие скобок часто оказывается недопустимым в компиляторе и существенно ухудшает результаты делинеаризации в Polly. В системе SAPFOR результаты делинеаризации не используются для генерации кода, на их основе выполняется анализ зависимостей по данным и строится распределение данных, а затем выполняется вставка в исходный код соответствующих DVMH-директив. При необходимости корректность расстановки директив может быть проверена средствами функциональной отладки, входящими в состав DVM-системы. Кроме того, для полного контроля процесса делинеаризации пользователю доступна опция анализатора -fsafe-type-cast, запрещающая небезопасные приведения типов во время анализа.

3. ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ

Чтобы проверить соответствие делинеаризованного представления массива и обращений к нему представлению массива в исходной программе, было проведено тестирование на автоматически сгенерированных тестовых программах. Чтобы максимально покрыть все возможные случаи использования массивов, тесты были классифицированы по следующим параметрам:

- количество измерений массива;
- размеры измерений: переменные (в стиле C99), константные, заданные через перечисления, макросы и литеральные константы разных целочисленных типов;
- динамические и статические массивы;
- локальные (объявленные в теле функции или переданные в качестве параметра) и глобальные массивы;
- различные способы задания индексных выражений: содержащие индексную переменную цикла, константные, содержащие переменные, отличные от индексных переменных;

- наличие приведения типов в объявлениях и обращениях к массивам.

С целью автоматической генерации и запуска тестов, а также анализа результатов запусков одним из авторов данной статьи была разработана библиотека тестирования Ctestgen [13]. Исходный код библиотеки написан на языке Python 3 и доступен на GitHub. Библиотека распространяется под свободной лицензией MIT.

Библиотека состоит из двух модулей: модуля генерации Си-программ через описание их абстрактных синтаксических деревьев и модуля запуска целевой программы на наборах тестов. Использование каждого модуля основано на наследовании базовых абстрактных классов и описании необходимого поведения на языке Python 3.

Каждой генерируемой программе соответствует объект класса Program, который состоит из множества директив подключения заголовочных файлов Include, множества определений макросов Define, множества перечислений Enum, множества глобальных переменных Var и множества функций Function. Функция задается названием, списком аргументов и телом функции – блоком кода. Для создания генератора нужно наследовать абстрактный класс ctestgen.generator.TestGenerator и переопределить метод _generate_programs(), возвращающий абстрактные синтаксические деревья описываемых программ. Пример определения класса для генерации программ с суммирующей функцией, принимающей на вход от 2 до 5 аргументов, приведен в Таблице 1, пример сгенерированной функции для 3 аргументов – в Таблице 2.

Модуль запуска тестов предназначен для запуска целевой программы (в данном случае – анализатора системы SAPFOR) и передаче ей на вход каждой из автоматически сгенерированных тестовых программ. Данный модуль определяет успешность прохождения тестов и собирает метрики запусков. Для запуска программ необходимо наследовать абстрактный класс ctestgen.runner.TestRunner и переопределить метод _on_test(), вызываемый для каждой программы в заданной директории.

Таблица 1. Пример определения класса для генерации программ с суммирующей функцией, принимающей на вход от 2 до 5 аргументов

```
from ctestgen.generator import TestGenerator
```

```
class ExampleTestGenerator(TestGenerator):
  def _generate_programs(self):
    generated_programs = list()
    for i in range(2, 6):
      # Создаем список аргументов функции (от 0 до i)
      sum_arguments = [Int('num_' + str(arg_idx)) for arg_idx in range(i)]
      # Объявляем функцию с заданным именем,  списком аргументов и результатом типа int.
      sum_function = Function('sum_' + str(i) + '_nums', Int, sum_arguments)
      sum_result = Int('sum')
      # Определяем тело функции.
      sum_body = CodeBlock(
        Assignment(VarDeclaration(sum_result), Add(sum_arguments)),
        Return(sum_result)
      )
      sum_function.set_body(sum_body)
      # Создаем программу, содержащую единственную функцию.
      example_program = Program('sum_' + str(i))
      example_program.add_function(sum_function)
      generated_programs.append(example_program)
    return generated_programs
# Генерируем множество программ.
example_generator = ExampleTestGenerator('example_generator_output')
example_generator.run()
```

Каждый сгенерированный тест помимо программы на языке Си 99 содержал ожидаемый результаты делинеаризации, описанные в JSON-формате. Возможность генерации результата делинеаризации в формате JSON также была добавлена в анализатор системы SAPFOR.

Таблица 2. Пример сгенерированной программы

```
int sum_3_nums(int num_0, int num_1, int num_2) {
```

```
 int sum = num_0 + num_1 + num_2;
 return sum;
}
```

В системе SAPFOR модуль делинеаризации используется при построении параллельного варианта исходной последовательной программы, а также при анализе свойств программы, например, определении зависимостей по данным. Как было отмечено в [9], чтобы повысить качество проводимого анализа программы, во многих случаях требуется ее предварительное преобразование. Подход, описанный в [8], позволяет выполнять необходимые преобразования скрыто от пользователя над представлением программы в виде LLVM IR. Общая схема выполнения анализа в системе SAPFOR выглядит следующим образом: выполнение некоторого анализа до преобразования LLVM IR, выполнение преобразования LLVM IR, повторное выполнение анализа и уточнение ранее полученных результатов. При этом результаты анализа будут связаны с объектами исходной непреобразованной программы. Модуль делинеаризации запускается на каждом шаге анализа с целью уточнения результатов анализа зависимостей по данным. Для анализа зависимостей используются наборы тестов, описанных в [5] и реализованных в виде модуля, входящего в состав LLVM. Данный модуль был соответствующим образом модифицирован с целью использования реализованного алгоритма делинеаризации.

Корректная работа модуля делинеаризации, а также модифицированного модуля анализа зависимостей по данным были вручную проверены на тестах производительности NAS Parallel Benchmarks 3.3 [14] и тестовом наборе Polybench/C the Polyhedral Benchmark suite 4.2.1 [15].

ЗАКЛЮЧЕНИЕ

Рассмотрен подход к восстановлению формы многомерных массивов языка C99, представленных в LLVM IR в линеаризованном виде. Предложенный подход опирается на отладочную информацию, доступную в LLVM, и структуру инструкций, используемых для вычисления смещений относительно начала массива. Использование стандартизованного представления отладочной информации позволяет в будущем расширить область применимости предложенного алгоритма на язык Fortran, избежав дополнительного анализа специфичных языковых конструкций.

Предложенный подход основан на идее, используемой в модуле делинеаризации, входящем в состав LLVM, но в отличие от него обеспечивает более точное соответствие результатов делинеаризации описанию многомерных массивов в исходной программе на языке высокого уровня. Данное соответствие является необходимым в силу направленности системы SAPFOR на распараллеливание программ на уровне исходного кода.

Дальнейшие работы планируется направить на совместное использование предложенного подхода и подхода, реализованного в LLVM, для построения эквивалентного многомерного представления массивов, явно заданных в исходной программе в линеаризованном виде, с целью последующей делинеаризации данных массивов в исходном коде программы.

Исходные коды системы SAPFOR доступны на GitHub [16].

СПИСОК ЛИТЕРАТУРЫ

1. *Konovalov N.A., Krukov V.A, Mikhajlov S.N., Pogrebtsov A.A.* Fortan DVM: a Language for Portable Parallel Program Development // Programming and Computer Software, 1995. V. 21. No 1. P. 35–38.

2. *Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л.* Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Вестник Южно-Уральского государственного университета, серия «Математическое моделирование и программирование». 2012. №18 (277), выпуск 12. Челябинск: Издательский центр ЮУрГУ. С. 82–92.

3. *Клинов М.С., Крюков В.А.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник Нижегородского университета им. Н.И. Лобачевского. 2009. № 2. С. 128–134.

4. *Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А.* Автоматизация распараллеливания программных комплексов // Труды XVIII Всероссийской научной конференции «Научный сервис в сети Интернет», Новороссийск, Россия, 19–24 сентября 2016 г. М.: ИПМ им. М.В. Келдыша, 2016. С. 76–85. doi:10.20948/abrau-2016-31

5. *Goff G., Kennedy K., Tseng C.W.* Practical Dependence Testing // Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation (PLDI '91), 1991. ACM, New York, NY, USA, 1991. P. 15–29.

6. *Lattner C., Adve V.* LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation // Proc. of the 2004 International Symposium on Code Generation and Optimization (CGO'04), 2004. Palo Alto, California.

7. *Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Крюков В.А., Кузнецов М.Ю., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А.* Распараллеливание программных комплексов. Проблемы и перспективы // Труды XX Всероссийской научной конференции «Научный сервис в сети Интернет», Новороссийск, Россия, 17–22 сентября 2018 г. М.: ИПМ им. М.В. Келдыша, 2018. С. 63–72. doi:10.20948/abrau-2018-33

8. *Kataev N.A.* Application of the LLVM Compiler Infrastructure to the Program Analysis in SAPFOR // Voevodin V., Sobolev S. (eds) Supercomputing. RuSCDays 2018. Communications in Computer and Information Science, 2018. Vol 965. Springer, Cham. P. 487-499. doi:10.1007/978-3-030-05807-4_41

9. *Катаев Н.А., Козырев В.И.* Преобразование программ на высокоуровневом языке программирования на основе результатов анализа низкоуровневого представления программ в системе SAPFOR // Труды XIII международной конференции «Параллельные вычислительные технологии, ПаВТ'2019», Калининград, Россия, 2–4 апреля 2019 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2019. С. 251–262.

10. Dwarf 3 Standard. URL: http://eagercon.com/dwarf/dwarf3std.htm

11. *Grosser T., Pop S., Ramanujam J., Sadayappan P.* On recovering multi-dimensional arrays in Polly // 5th International Workshop on Polyhedral Compilation Techniques, IMPACT 2015. P. 1–9.

12. Polly – Polyhedral optimizations for LLVM. URL: https://polly.llvm.org/

13. Ctestgen. URL: https://github.com/VolandTymim/ctestgen

14. *Seo S., Jo G., Lee J.* Performance Characterization of the NAS Parallel Benchmarks in OpenCL // 2011 IEEE International Symposium on. Workload Characterization (IISWC), 2011. P. 137–148.

15. PolyBench/C the Polyhedral Benchmark suite. URL: http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/polybench.html

16. SAPFOR. URL: https://github.com/dvm-system

RECONSTRUCTION OF MULTI-DIMENSIONAL FORM OF LINEARIZED ACCESSES TO ARRAYS IN SAPFOR

N. A. Kataev[1], V. N. Vasilkin [2]

[1]*Keldysh Institute of Applied Mathematic RAS;* [2] *Lomonosov Moscow State University*

[1]kataev_nik@mail.ru, [2] volandtymim@gmail.com

Abstract

The system for automated parallelization SAPFOR (System FOR Automated Parallelization) includes tools for program analysis and transformation. The main goal of the system is to reduce the complexity of program parallelization. SAPFOR system is focused on the investigation of multilingual applications in Fortran and C programming languages. The low-level LLVM IR representation is used in SAPFOR for program analysis. This representation allows us to perform various IR-level optimizations to improve the quality of program analysis. At the same time, it loses some features of the program, which are available in its higher level representation. One of these features is the multi-dimensional structure of the arrays. Data dependence analysis is one of the main problems which should be solved to automate program parallelization. Moreover, such an analysis belongs to the class of NP-hard problems. Knowledge of the multidimensional structure of arrays allows in many cases to take into account the structure of index expressions in calls to arrays and reduce the complexity of the analysis. In addition, the use of multi-dimensional arrays allows us to use multi-dimensional processor matrix and to parallelize a whole loop nests, rather than a single loop in the nest. So, parallelism of a program is going to be increased. These opportunities are natively supported in the DVM system. This paper discusses the approach used in the SAPFOR system to recover the form of multi-dimensional arrays by their linearized representation in LLVM IR. The proposed approach has been

successfully evaluated on various applications including performance tests from the NAS Parallel Benchmarks suite.

***Keywords**: program analysis, semi-automatic parallelization, SAPFOR, DVM, LLVM*

REFERENCES

1. *Konovalov N.A., Krukov V.A, Mikhajlov S.N., Pogrebtsov A.A.* Fortan DVM: a Language for Portable Parallel Program Development // Programming and Computer Software. 1995. V. 21. No 1. P. 35–38.

2. *Bakhtin V.A., Klinov M.S., Kriukov V.A., Podderiugina N.V., Pritula M.N., Sazanov Iu.L.* Rasshirenie DVM-modeli parallelnogo programmirovaniia dlia klasterov s geterogennymi uzlami // Vestnik Iuzhno-Uralskogo gosudarstvennogo universiteta, seriia "Matematicheskoe modelirovanie i programmirovanie", No 18 (277), vypusk 12. Cheliabinsk: Izdatelskii tsentr IuUrGU, 2012. P. 82–92.

3. *Klinov M.S., Kriukov V.A.* Avtomaticheskoe rasparallelivanie Fortran-programm. Otobrazhenie na klaster // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo. 2009. No 2. S. 128–134.

4. *Bakhtin V.A., Zhukova O.F., Kataev N.A., Kolganov A.S., Kriukov V.A., Podderiugina N.V., Pritula M.N., Savitskaia O.A., Smirnov A.A.* Avtomatizatsiia rasparallelivaniia programmnykh kompleksov // Trudy XVIII Vserossiiskoi nauchnoi konferentsii “Nauchnyi servis v seti Internet”, Novorossiisk, Russia, 19–24 sentiabria. M.: IPM im. M.V. Keldysha, 2016. S. 76–85. doi:10.20948/abrau-2016-316

5. *Goff G., Kennedy K., Tseng C.W.* Practical Dependence Testing // Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation (PLDI ’91), 1991. ACM, New York, NY, USA, 1991. P. 15–29.

6. *Lattner C., Adve V.* LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation // Proc. of the 2004 International Symposium on Code Generation and Optimization (CGO’04), 2004. Palo Alto, California.

7. *Bakhtin V.A., Zhukova O.F., Kataev N.A., Kolganov A.S., Kriukov V.A., Kuznetsov M.Iu., Podderiugina N.V., Pritula M.N., Savitskaia O.A., Smirnov A.A.* Rasparallelivanie programmnykh kompleksov. Problemy i perspektivy // Trudy XX Vserossiiskoi nauchnoi konferentsii “Nauchnyi servis v seti Internet”, Novorossiisk, Russia, 17–22

sentiabria 2018 g. M.: IPM im. M.V. Keldysha, 2018. S. 63–72. doi:10.20948/abrau-2018-33

8. *Kataev N.A.* Application of the LLVM Compiler Infrastructure to the Program Analysis in SAPFOR // Voevodin V., Sobolev S. (eds) Supercomputing. RuSCDays 2018. Communications in Computer and Information Science, 2018. V. 965. Springer, Cham. P. 487–499. doi:10.1007/978-3-030-05807-4_41

9. *Kataev N.A., Kozyrev V.I.* Preobrazovanie programm na vysokourovnevom iazyke programmirovaniia na osnove rezultatov analiza nizkourovnevogo predstavleniia programm v sisteme SAPFOR // Trudy XIII mezhdunarodnoi konferentsii “Parallelnye vychislitelnye tekhnologii, PaVT'2019”, Kaliningrad, Russia, 2–4 aprelia 2019 g. Korotkie stati i opisaniia plakatov. Cheliabinsk: Izdatelskii tsentr IuUrGU, 2019. S. 251–262.

10. Dwarf 3 Standard. URL: http://eagercon.com/dwarf/dwarf3std.htm

11. *Grosser T., Pop S., Ramanujam J., Sadayappan P.* On recovering multi-dimensional arrays in Polly // 5th International Workshop on Polyhedral Compilation Techniques, IMPACT 2015. P. 1–9.

12. Polly – Polyhedral optimizations for LLVM. URL: https://polly.llvm.org/

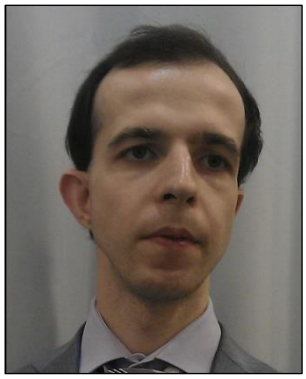
13. Ctestgen. URL: https://github.com/VolandTymim/ctestgen

14. *Seo S., Jo G., Lee J.* Performance Characterization of the NAS Parallel Benchmarks in OpenCL // 2011 IEEE International Symposium on. Workload Characterization (IISWC), 2011. P. 137–148.

15. PolyBench/C the Polyhedral Benchmark suite. URL: http://web.cse.ohio-state.edu/~pouchet.2/software/polybench/polybench.html

16. SAPFOR. URL: https://github.com/dvm-system

СВЕДЕНИЯ ОБ АВТОРАХ



КАТАЕВ Никита Андреевич – научный сотрудник ИПМ им. М.В. Келдыша, специалист в области системного программирования. Сфера научных интересов – компиляторные технологии, автоматизация распараллеливания программ.

Nikita Andreevich KATAEV – Researcher of KIAM RAS, a specialist in system programming. Research interests include compiler technologies, semi-automatic program parallelization.

email: kataev_nik@mail.ru

ВАСИЛЬКИН Владислав Николаевич – студент 1 курса магистратуры факультета ВМиК МГУ им М.В. Ломоносова.

Vladislav Nikolaevich VASILKIN – student of CMC faculty of Lomonosov Moscow State University

email: volandtymim@gmail.com

Материал поступил в редакцию 15 ноября 2019 года