ИПМ им.М.В.Келдыша РАН

Абрау-2019 • Труды конференции

Н.А. Катаев, В.Н. Василькин

Восстановление обращений к многомерным массивам в системе SAPFOR

Рекомендуемая форма библиографической ссылки

Катаев Н.А., Василькин В.Н. Восстановление обращений к многомерным массивам в системе SAPFOR // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 413-423. — URL: http://keldysh.ru/abrau/2019/theses/90.pdf doi:10.20948/abrau-2019-90

Размещена также презентация к докладу

Восстановление обращений к многомерным массивам в системе SAPFOR

Н.А. Катаев[1], В.Н. Василькин[2]

[1] ИПМ им. М.В. Келдыша РАН
[2] МГУ им. М.В. Ломоносова

Аннотация. Представление программ в виде LLVM IR позволяет проводить различные оптимизации с целью повышения качества анализа программ в системе SAPFOR (System FOR Automated Parallelization). Являясь единым для разных высокоуровневых языков, LLVM IR позволяет исследовать многоязыковые вычислительные комплексы. При этом оно теряет некоторые особенности программы, отражаемые ее представлением на языке высокого уровня. Одной из таких особенностей является многомерная структура используемых массивов. Знание многомерной структуры повышает точность анализа зависимостей по данным, так как линеаризованное представление массивов, размеры измерений которых являются переменными, может не быть аффинным выражением и к нему не удастся применить методы целочисленного линейного программирования. 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 данной работе рассматривается 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.

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

Reconstruction of multi-dimensional arrays in SAPFOR

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

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

Abstract. Low-level representation of programs in the form of LLVM IR allows for various optimizations to improve the quality of program analysis in SAPFOR

(System FOR Automated Parallelization). Being the same for different high-level languages, LLVM IR allows us to explore multilingual applications. 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. Multi-dimensional view improves the accuracy of data dependency analysis, since the linearized representation of arrays with parametric sizes may not be an affine expression and it will not be possible to apply integer linear programming to analyze it. 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.

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

1. 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 и полу-автоматическом, необходимо учитывать особенности использования многомерных структур данных в программе.

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 [3] 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 [4, 5] 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 многомерных 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.

2. 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 статье [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 выше сказанным 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] предлагается 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 восстановленную многомерную структуру массивов с многомерными массивами в исходной программе.

3. 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 реализованные 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 работе [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 ситуация 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 неизвестные измерения массива. Чтобы уменьшит вероятность ошибки, наибольший общий делитель вычисляется среди термов, полученных для всех обращений к элементам заданного массива внутри анализируемой функции. 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 обращения к элементам массива, умножались на одну и ту же величину.

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**.

Выражения, являющиеся операндами инструкции **getelementptr**, могут иметь достаточно сложный вид и иметь вложенные приведения типов. Для качественной работы 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 менять результаты вычислений, по этой причине раскрытие скобок часто оказывается недопустимо 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 над процессом делинеаризации пользователю доступна опция анализатора **-fsafe-type-cast**, запрещающая небезопасные приведения типов во время анализа.

4. 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 основывается 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.

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.

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

Помимо автоматического тестирования вручную была проверена корректная работа модуля делинеаризации на тестах производительности NAS Parallel Benchmarks 3.3 [14] и тестовом наборе Polybench/C the Polyhedral Benchmark suite 4.2.1 [15].

5. Заключение

В данной работе 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].

Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект 18-01-00851-а, 17-01-00820-а.

Литература

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. vol. 21, no. 1, 1995. — 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", №18 (277), выпуск 12 — Челябинск: Издательский центр ЮУрГУ, 2012. — С. 82-92
3. Goff G., Kennedy K., Tseng CW. Practical Dependence Testing // Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation (PLDI ’91). — ACM, New York, NY, USA, 1991. — P. 15–29
4. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник Нижегородского университета им. Н.И. Лобачевского, № 2, 2009. — С. 128–134.
5. 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 // Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2016. — С. 76-85. — doi:10.20948/abrau-2016-31
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). — Palo Alto, California, 2004
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 // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018 г., г. Новороссийск). — М.:

ИПМ им. М.В.Келдыша, 2018. — С. 63-72. — URL: http://keldysh.ru/abrau/2018/theses/33.pdf 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, vol 965. — Springer, Cham, 2018 — 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 // Параллельные вычислительные технологии – XIII международная конференция, ПаВТ'2019, г. Калининград, 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. – C. 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.

References

. 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. vol. 21, no. 1, 1995. — 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", №18 (277), vypusk 12 — Cheliabinsk: Izdatelskii tsentr IuUrGU, 2012. — P. 82-92
3. Goff G., Kennedy K., Tseng CW. Practical Dependence Testing // Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation (PLDI ’91). — ACM, New York, NY, USA, 1991. — P. 15–29
4. Klinov M.S., Kriukov V.A. Avtomaticheskoe rasparallelivanie Fortran-programm. Otobrazhenie na klaster // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo, № 2, 2009. — P. 128–134.
5. 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 // Nauchnyi servis v seti Internet: trudy XVIII Vserossiiskoi nauchnoi konferentsii (19-24 sentiabria 2016 g., g. Novorossiisk). — M.: IPM im. M.V.Keldysha, 2016. — P. 76-85. — doi:10.20948/abrau-2016-316.
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). — Palo Alto, California, 2004
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 // Nauchnyi servis v seti Internet: trudy XX Vserossiiskoi nauchnoi konferentsii (17-22 sentiabria 2018 g., g. Novorossiisk). — M.: IPM im. M.V.Keldysha, 2018. — P. 63-72. — URL: http://keldysh.ru/abrau/2018/theses/33.pdf 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, vol 965. — Springer, Cham, 2018 — 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 // Parallelnye vychislitelnye tekhnologii – XIII mezhdunarodnaia konferentsiia, PaVT'2019, g. Kaliningrad, 2–4 aprelia 2019 g. Korotkie stati i opisaniia plakatov. — Cheliabinsk: Izdatelskii tsentr IuUrGU, 2019 — P. 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. – C. 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.