Согласованное представление Fortran-программ в компиляторе PARFOR на разных уровнях абстракции

Н.А. Катаев[1]

1 Институт прикладной математики им. М.В. Келдыша РАН

Аннотация. Распараллеливающий компилятор PARFOR, выделенный в отдельный инструмент из системы автоматизированного распараллеливания SAPFOR, предназначен для разработки параллельных программ на языках с неявным параллелизмом Fortran и C. Целью разработки компилятора PARFOR является автоматическое преобразование программы на уровне исходного кода для последующего выполнения на вычислительном кластере. Неявное параллельное программирование предполагает возможность описания свойств исходной последовательной программы, чтобы снизить сложность поиска необходимых преобразований программы. Распараллеливание выполняется в модели DVMH на языках Fortran-DVMH и C-DVMH соответственно. При этом для анализа программ используются возможности системы компиляторов LLVM, а преобразования основаны на представлении программ в виде абстрактного синтаксического дерева (AST). В данной работе рассматривается подход, позволяющий согласованно рассматривать программы на двух уровнях абстракции: уровне исходного кода и низкоуровнего представления, задействовав потенциал низкоуровневого представления в отношении анализа программ и сохранив понятный для пользователя вид отображаемой информации. В работе также рассматривается возможность исследования преобразованного LLVM IR, чтобы повысить точно проводимого анализа. Также приводятся основные причины выбора LLVM в качестве инструментальной базы для разработки компилятора PARFOR.

Ключевые слова: неявный параллелизм, преобразование программ, анализ программ, распараллеливание программ, SAPFOR, DVM, LLVM

Consistent representation of Fortran programs in PARFOR compiler at different levels of abstraction

N.A. Kataev[1]

1 Keldysh Institute of Applied Mathematics (Russian Academy of Sciences)

Abstract. PARFOR parallelizing compiler is the main part of SAPFOR (System For Automated Parallelization). This compiler can be applied as a stand-alone tool to exploit implicit parallelism in traditional sequential languages Fortran and C. It was primarily designed to perform source-to-source transformation of a sequential program for execution on parallel architectures with distributed memory. Implicitly parallel programming assumes that programmers may describe the properties of the original sequential program in order to reduce the complexity of finding an optimal program specific sequence of transform passes. PARFOR focuses on parallel programming in DVMH model. CDVMH and Fortran-DVMH languages are used. PARFOR relies on LLVM (Low Level Virtual Machine) compiler infrastructure to examine a program. However, program transformations are based on a higher level program representation in the form of an abstract syntax tree (AST). In this paper, we consider an approach that allows us to consistently represent programs at two levels of abstraction: the source-level (AST) and the low-level level (LLVM IR). The considered questions involve the interpretation of information derived from LLVM IR to retain a user-friendly description of analysis results. In addition, the possibilities of analysis of the transformed LLVM IR are explored to improve the quality of the source program analysis. We also discuss the reasons which argue the choice of LLVM as a base for PARFOR development.

Keywords: implicitly parallel programming, program transformation, program analysis, program parallelization, SAPFOR, DVM, LLVM

1. Введение

Повсеместное распространение многопроцессорных вычислительных систем привело к необходимости разработки параллельных программ, способных выполняться на системах разной архитектуры. Для этого приходится одновременно задействовать различные технологии параллельного программирования (MPI, OpenMP, CUDA, OpenACC), либо использовать высокоуровневые модели, объединяющие сразу несколько технологий (DVM [1, 2], XcalableACC [3]). Постоянное усложнение моделей параллельного программирования стимулирует использование языков с неявным параллелизмом для создания параллельных программ вычислительного характера. Особое внимание уделяется языкам Фортран и Си/Си++, поскольку их в основном и используют программисты при решении задач, наиболее остро требующих распараллеливания.

В большинстве случаев попытка применения конкретных технологий параллельного программирования для распараллеливания произвольных последовательных программ наталкивается на необходимость преобразования исходных программ. Целью таких преобразований может быть устранение зависимостей в программе, препятствующих параллельному выполнению участков кода, оптимизация доступов к памяти, особенно актуальная в случае использования GPU. Может потребоваться изменение структуры хранения данных для обеспечения возможности их хранения на процессорах в случае

использования распределённой памяти, а также изменение структуры циклов, чтобы корректно отобразить вычисления на процессоры, владеющие соответствующими порциями данных. В крайнем случае, от программиста может потребоваться выбор другого алгоритма, решающего ту же задачу, но обладающего большим потенциалом параллелизма.

Это значительно сужает возможности автоматического распараллеливания. Выходом может стать следование определенным правилам при написании параллельных программ и использование специальных указаний в коде или флагов компилятора для задания свойств объектов программы. В отличие от моделей параллельного программирования, основанных на вставке директив, описывающих параллелизм в программе, данный подход предполагает описание свойств исходной последовательной программы, чтобы, руководствуясь ими, компилятор смог выполнить необходимые преобразования и определить соответствующие спецификации параллелизма самостоятельно [4].

Такой подход реализован в автоматически распараллеливающем компиляторе Paralax [5], ориентированном на выявление крупнозернистого параллелизма и организацию конвейерного выполнения программы. Компилятор Paralax использует аннотации, размещаемые программистом в коде программы, для выявления ее свойств. Например, некоторые переменные могут быть помечены как «мертвые», для функции можно указать наличие побочных эффектов, многие аннотации направлены на описание использования указателей в программе.

Описание свойств исходной программы также используется в инструменте Parallware [6], ориентированном на обучение параллельному программированию. Так, например, пользователя просят гарантировать отсутствие пересечения по памяти (aliasing) между массивами, являющимися параметрами функции. С точки зрения стандарта C99 такого же эффекта можно добиться, не прибегая к средствами диалоговой оболочки, достаточно указать ключевое слово ‘restrict’ в объявлении формальных параметров функции. Стоит отметить, что в данный момент возможности Parallware значительно ограничены и в первую очередь рассчитаны на обучение параллельному программированию на небольших программах, а не на распараллеливание больших программных комплексов.

Упомянутые инструменты не получили в данный момент широкого распространения и на данном этапе являются скорее исследовательскими проектами, а не компиляторами, пригодными для распараллеливания больших вычислительных задач. Другим важным недостатком является их ориентированность на системы с общей памятью, при этом Parallware

поддерживает генерацию кода для графических процессоров за счет вставки директив OpenACC.

Система SAPFOR [7, 8] нацелена на взаимодействие с пользователем в интерактивном режиме, и обеспечивает поддержание разной степени вовлеченности программиста в процесс распараллеливания (ручное преобразование программы, ручной и автоматический выбор необходимых преобразований, описание свойств программы, профилирование и автоматизированный выбор времяемких фрагментов, которые должны быть рассмотрены в первую очередь). Система ориентирована на распараллеливании в модели DVMH [1,2] и тем самым охватывает системы разной архитектуры, позволяя проводить распараллеливание для гибридных вычислительных кластеров.

В состав системы входит автоматически распараллеливающий компилятор PARFOR, который отвечает за автоматическое принятие решений о необходимости преобразования программы, распределение данных и вычислений, оптимизацию коммуникаций. Компилятор PARFOR может быть выделен в отдельный инструмент для реализации описанного выше подхода и поддержания возможности неявного параллельного программирования.

Компилятор PARFOR, как и система SAPFOR, выполняет распараллеливание программы на уровне исходного кода. Это позволяет пользователю оценить принятые решения, внести коррективы в параллельную версию программы, программа становится более переносимой. Кроме того, использование модели DVMH значительно снижает сложность построения параллельной версии программы, так как служит переходом к низкоуровневым технологиям параллельного программирования (MPI, CUDA, OpenMP). В результате в компиляторе программа должна быть представлена на уровне близком к уровню исходного кода, это существенно отличает PARFOR от обычных компиляторных инфраструктур, таких как GCC [9] или LLVM [10,11], использующих низкоуровневые представления программы. В свою очередь это не позволяет напрямую воспользоваться широкими возможностями данных систем по анализу и преобразованию программ.

В данной работе рассматривается подход, используемый в PARFOR и позволяющий воспользоваться преимуществами низкоуровневого представления программы в системе LLVM в области анализа программ, сохранив при этом возможность преобразования и распараллеливания программы на уровне исходного кода.

2. Выбор инструментальной базы

Первоначальная версия системы SAPFOR [7,8], ориентированная на Fortran, опиралась на внутреннее представление, образованное абстрактным

синтаксическим деревом (AST) и базой данных, отражающей основную структуру и свойства программ.

Структура информации, хранимой в базе данных, была продиктована потребностями модуля, ответственного за построение вариантов параллельных программ (эксперта), и являлась некоторой достаточно грубой абстракцией, описывающей исходную программу. База данных выступала в качестве интерфейса для передачи необходимой информации эксперту и получения от него описания спецификаций параллелизма, которые должны быть добавлены в исходную программу. На основе информации, хранимой в базе данных, был реализован алгоритм определения скалярных переменных, которые должны быть объявлены в параллельной программе как приватные [12]. Существенным преимуществом было то, что реализованный алгоритм фактически не зависел от языка исходной программы. Однако, для более сложных анализов и тем более преобразования программы (приватизация массивов, определение зависимостей по данным, анализ указателей, распространение констант, копий, преобразование циклов) возможностей базы данных было недостаточно.

Анализаторы и генераторы кода, входящие в SAPFOR, использовали абстрактное синтаксическое дерево (AST) программы для построения базы данных и генерации параллельной версии программы соответственно. Для построения AST применялась библиотека Sage++ [13]. Sage++ – это объектно-ориентированная библиотека, разработанная специально для выполнения преобразований на уровне исходного кода. К сожалению, она уже долгое время не развивается и ответственность за поддержание новых языковых стандартов, а также устранение ошибок в исходной версии библиотеки ложится на разработчиков системы SAPFOR. В данный момент поддерживается только Fortran 95. Еще более существенным моментом является отсутствие реализации большинства известных алгоритмов анализа и преобразования программ, применяемых в современных компиляторах.

Следовательно, для построения компилятора PARFOR, а также дальнейшего развития системы SAPFOR, встает вопрос поиска новой инструментальной базы. Среди рассмотренных инструментов можно выделить Cetus [14] и ROSE [15].

Система Cetus разрабатывается с использованием Java в университете Пердью с 2004 года. В данный момент поддерживаются стандарты ANSI C89/ISO C90, поддержка Fortran отсутствует. Реализация базовых алгоритмов анализа, часто оказывается слишком консервативной (например, наличие операторов ‘goto’ в циклах препятствует анализу, в качестве редукционных операций допускаются только операции сложения). Стандартные опции компиляторов также не поддерживаются, это значительно затрудняет использования Cetus в системах сборки (таких как make) для работы с большими программными комплексами.

Система ROSE разрабатывается в Ливерморской национальной лаборатории им. Э. Лоуренса (LLNL). Основными поддерживаемыми языками

являются C(89/98), C++(98/11), Fortran (77/95/2003), Java, Python, Haskell. Несмотря, на реализацию большого количества алгоритмов анализа, возможности использования системы ограничены. Опыт использования системы для полуавтоматического преобразования программ [16] показал наличие ошибок в реализации некоторых модулей, а ориентированность авторов системы на исследования в области анализа бинарного кода, делает перспективы быстрого их исправления туманными.

Современные компиляторные инфраструктуры, такие как GCC и LLVM опираются на свои собственные внутренние представления программ и реализуют большое число алгоритмов анализа и преобразования. Широкое их распространение и постоянное развитие не дает повода сомневаться в их надежности и своевременном исправлении обнаруживаемых ошибок.

Однако существенным недостатком является низкий уровень используемых внутренних представлений. Например, LLVM IR – это программный код, состоящий из строго типизированного набора RISC-подобных инструкций. Для доступа к памяти используется небольшое множество специально выделенных инструкций. Программный код представлен в форме однократного присваивания (SSA) [17], исключением являются инструкции доступа к памяти. При этом LLVM IR содержит подробную отладочную информацию о программе, которая может быть использована с целью распространения результатов анализа LLVM IR на исходный код программы. Более того, открытость исходных кодов, наличие подробной документации и объектно-ориентированная структура исходных кодов позволяют расширять возможности LLVM. Также важно отметить, что LLVM IR, генерируемый по исходному коду точно соответствует операциям исходной программы: все оптимизации выполняются в виде отдельных проходов. Для LLVM IR доступны фронтенды как с языков C/C++ (Clang) [18], так и с Fortran (Flang) [19].

3. Внутреннее представление программы в компиляторе PARFOR

Реализация преобразований программ на уровне исходного кода в компиляторе PARFOR опирается на представление программ в виде AST, в то время как анализ в основном выполняется над низкоуровневым LLVM IR. При этом выполнение преобразований на уровне LLVM IR позволяет повысить точность анализа исходной программы и необходимо для работы многих реализованных в LLVM алгоритмов (определение индукционных и редукционных переменных, вычисление границ циклов, уточнение результатов анализа указателей). В тоже время данные преобразования никак не сказываются на исходной программе. Результаты анализа, полученные для программы в виде LLVM IR, могут быть распространены на программу на исходном языке высокого уровня благодаря наличию отладочной информации. Отладочная информация в LLVM стандартизована и представлена в соответствии со стандартом Dwarf [20]. Таким образом, она минимально

зависит от фронтенда, используемого для построения LLVM IR. Это позволяет реализовать общие алгоритмы сопоставления AST и LLVM IR и, более того, использовать для построения AST инструменты отличные от фронтендов. Для C/C++ используется Clang, а для Fortran Sage++, несмотря на то, что для генерации LLVM IR применяется Flang.

Чтобы реализовать согласованное представление программы на двух уровнях абстракции была разработана структура данных, называемая деревом псевдонимов.

Каждая вершина дерева – это объединение участков памяти. Участок памяти характеризуется адресом начала и размером. Два участка памяти попадают в одну вершину дерева, если они потенциально могут пересекаться. Для проверки пересечений по памяти используются реализованные в LLVM проходы анализа. Один и тот же элементарный участок памяти может относиться только к одной вершине дерева. У одной вершины может быть множество дочерних вершин. Разбиение делается таким образом, что объединение всех участков памяти родительской вершины покрывает объединение участков памяти дочерней. Исключением является память, используемая в вызываемой функции. Такая память описывается отдельным видом вершин и пересекается (не обязательно покрывая) с участками памяти дочерних вершин. Участки памяти, относящиеся к соседним вершинам, не пересекаются.

Построение дерева псевдонимов выполняется в два этапа. На первом этапе участки памяти описываются низкоуровневыми объектами, задействованными в LLVM IR. Построенное дерево (alias tree) используется для анализа программы и детального описания полученных результатов на низком уровне. Затем дерево псевдонимов обобщается на основе отладочной информации, которая используется для описания участков памяти в обобщенном дереве (source-level alias tree). Низкоуровневые участки памяти, которые не могут быть представлены с помощью отладочной информации, обобщаются до объемлющих их участков памяти.

Использование отладочной информации для описания участков памяти, позволяет дереву псевдонимов (source-level alias tree) сохранять свою структуру при преобразованиях LLVM IR, необходимых для повышения качества анализа.

Одним из таких преобразований является проход SROA (Scalar Replacement of Aggregates), реализованный в LLVM. Данный проход разделяет память агрегатного типа (структуры, массивы константной длины) на отдельные скалярные участки памяти (соответствующие элементам структур и массивов) и размещает скалярные участки памяти на регистрах. При этом отладочная информация модифицируется в соответствии с выполненными преобразованиями и позволяет определить, каким переменным исходной программе соответствуют регистры. В результате может быть выполнен анализ свойств переменных исходной программы, в том числе отдельно для каждого элемента переменной агрегатного типа.

В LLVM IR используется бесконечный регистровый файл, и обращения к регистрам представлены в SSA-форме. Следствием такого преобразования является упрощение индексных выражений в обращениях к массивам и приведение их к виду, зависящему от индуктивных переменных цикла.

Отладочная информация, доступная для переменных, размещенных на регистрах, используется, чтобы уточнить дерево псевдонимов (source-level alias tree) после выполнения данного преобразования.

4. Заключение

Использование языков с неявным параллелизмом (C/C++, Fortran) является соблазнительной возможностью для разработки параллельных программ, при условии, что программы разрабатываются по определенным правилам и для уточнения их свойств программист размещает специальные указания в исходном коде. Для реализации данной возможности из системы SAPFOR был выделен в виде отдельного инструмента компилятор PARFOR.

В качестве инструментальной базы для анализа программ PARFOR опирается на LLVM, при этом для преобразования программ на уровне исходного кода используется абстрактное синтаксическое дерево для программы на соответствующем языке (Fortran или C).

Чтобы обеспечить согласованное представление программы на данных уровнях, была разработана новая структура данных – дерево псевдонимов, опирающееся на отладочную информацию, хранимую в LLVM IR.

Рассмотренное дерево псевдонимов обладает следующими свойствами: (1) обобщает низкоуровневое представление участков памяти на уровне LLVM IR до объектов исходной программы; (2) напрямую не зависит от исходного языка и используемых фронтендов; (3) адаптирует свою структуру к преобразованиям LLVM IR, которые не затрагивают структуру памяти исходной программы.

Работа выполнена при поддержке РФФИ (проекты 16-07-01014, 17-01-00820, 18-01-00851) и программы президиума РАН № 26 «Фундаментальные основы создания алгоритмов и программного обеспечения для перспективных сверхвысокопроизводительных вычислений».

Литература

1. Konovalov, N.A., Krukov, V.A, Mikhajlov, S.N., Pogrebtsov, A.A.: Fortan DVM: a Language for Portable Parallel Program Development. In: Programming and Computer Software. vol. 21, no. 1, pp. 35-38, 1995
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. Hitoshi Murai, Masahiro Nakao, Takehiro Shimosaka, Akihiro Tabuchi, Taisuke Boku, and Mitsuhisa Sato. "XcalableACC - a Directive-based Language Extension for Accelerated Parallel Computing", Supercomputing'14 poster, New Orleans, LA, USA, Nov. 2014
4. Wen-mei Hwu, Shane Ryoo, Sain-Zee Ueng, John H. Kelm, Isaac Gelado, Sam S. Stone, Robert E. Kidd, Sara S. Baghsorkhi, Aqeel A. Mahesri, Stephanie C. Tsao, Nacho Navarro, Steve S. Lumetta, Matthew I. Frank, and Sanjay J. Patel. 2007. Implicitly parallel programming models for thousand-core microprocessors. In Proceedings of the 44th annual Design Automation Conference (DAC '07). ACM, New York, NY, USA, 754-759. DOI=http://dx.doi.org/10.1145/1278480.1278669
5. H. Vandierendonck et al. The Paralax Infrastructure: Automatic Parallelization with a Helping Hand. In PACT, 2010.
6. Manuel Arenaz, and Sergio Ortega, and Ernesto Guerrero, and Fernanda Foertter. Parallware Trainer: Interactive Tool for Experiential Learning of Parallel Programming using OpenMP and OpenACC // EduHPC-17: Workshop on Education for High-Performance Computing. URL: https://grid.cs.gsu.edu/~tcpp/curriculum/sites/default/files/paper%206_0.pdf
7. М.С. Клинов, В.А. Крюков. Автоматическое распараллеливание Фортран-программ. Отображение на кластер. Вестник Нижегородского университета им. Н.И. Лобачевского, 2009, № 2, С. 128–134.
8. Бахтин В.А., Клинов М.С., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Автоматическое отображение программ на языке фортран на кластеры с графическими процессорами // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2014. Т. 3. № 3. С. 86-96.
9. GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/
10. The LLVM Compiler Infrastructure. URL: http://llvm.org
11. Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the International Symposium on Code Generation and Optimization (GCO), pages 75–87, 2004.
12. Катаев Н.А. Статический анализ последовательных программ в системе автоматизированного распараллеливания САПФОР // Вестник Нижегородского университета им. Н.И. Лобачевского. - Н. Новгород: Изд-во ННГУ. N5(2), 2012, с. 359-366
13. pC++/Sage++. URL: http://www.extreme.indiana.edu/sage/.
14. Lee, S. I., Johnson, T. A., Eigenmann, R.: Cetus an extensible compiler infrastructure for source-to-source transformation. In: International Workshop on Languages and Compilers for Parallel Computing, pp. 539-553. Springer, Berlin, Heidelberg (2003)
15. ROSE compiler infrastructure, http://rosecompiler.org/.

16. Baranov, M.S., Ivanov, D.I., Kataev, N.A., Smirnov, A.A.: Automated parallelization of sequential C-programs on the example of two applications from the field of laser material processing. // CEUR Workshop Proceedings 1st Russian Conference on Supercomputing Days 2015, vol. 1482, pp. 536, 2015.
17. R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, F. K. Zadeck. An Efficient Method of Computing Static Single Assignment Form // Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 25-35, January, 1989
18. Clang: a C language family frontend for LLVM, https://clang.llvm.org/.
19. GitHub - flang-compiler/flang. URL: https://github.com/flang-compiler/flang.
20. Dwarf 3 Standard. URL: http://eagercon.com/dwarf/dwarf3std.htm

References

1. Konovalov, N.A., Krukov, V.A, Mikhajlov, S.N., Pogrebtsov, A.A.: Fortan DVM: a Language for Portable Parallel Program Development. In: Programming and Computer Software. vol. 21, no. 1, pp. 35-38, 1995
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. S. 82-92
3. Hitoshi Murai, Masahiro Nakao, Takehiro Shimosaka, Akihiro Tabuchi, Taisuke Boku, and Mitsuhisa Sato. "XcalableACC - a Directive-based Language Extension for Accelerated Parallel Computing", Supercomputing'14 poster, New Orleans, LA, USA, Nov. 2014
4. Wen-mei Hwu, Shane Ryoo, Sain-Zee Ueng, John H. Kelm, Isaac Gelado, Sam S. Stone, Robert E. Kidd, Sara S. Baghsorkhi, Aqeel A. Mahesri, Stephanie C. Tsao, Nacho Navarro, Steve S. Lumetta, Matthew I. Frank, and Sanjay J. Patel. 2007. Implicitly parallel programming models for thousand-core microprocessors. In Proceedings of the 44th annual Design Automation Conference (DAC '07). ACM, New York, NY, USA, 754-759. DOI=http://dx.doi.org/10.1145/1278480.1278669
5. H. Vandierendonck et al. The Paralax Infrastructure: Automatic Parallelization with a Helping Hand. In PACT, 2010.
6. Manuel Arenaz, and Sergio Ortega, and Ernesto Guerrero, and Fernanda Foertter. Parallware Trainer: Interactive Tool for Experiential Learning of Parallel Programming using OpenMP and OpenACC // EduHPC-17: Workshop on Education for High-Performance Computing. URL: https://grid.cs.gsu.edu/~tcpp/curriculum/sites/default/files/paper%206_0.pdf
7. M.S. Klinov, V.A. Kriukov. Avtomaticheskoe rasparallelivanie Fortran-programm. Otobrazhenie na klaster. // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo, 2009, № 2, S. 128–134.
8. Bakhtin V.A., Klinov M.S., Kolganov A.S., Kriukov V.A., Podderiugina N.V., Pritula M.N. Avtomaticheskoe otobrazhenie programm na iazyke fortran na

klastery s graficheskimi protsessorami // Vestnik Iuzhno-Uralskogo gosudarstvennogo universiteta. Seriia: Vychislitelnaia matematika i informatika. 2014. T. 3. № 3. S. 86-96.
9. GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/
10. The LLVM Compiler Infrastructure. URL: http://llvm.org
11. Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the International Symposium on Code Generation and Optimization (GCO), pages 75–87, 2004.
12. Kataev N.A. Staticheskii analiz posledovatelnykh programm v sisteme avtomatizirovannogo rasparallelivaniia SAPFOR // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo. - N. Novgorod: Izd-vo NNGU. N5(2), 2012, s. 359-366
13. pC++/Sage++. URL: http://www.extreme.indiana.edu/sage/.
14. Lee, S. I., Johnson, T. A., Eigenmann, R.: Cetus an extensible compiler infrastructure for source-to-source transformation. In: International Workshop on Languages and Compilers for Parallel Computing, pp. 539-553. Springer, Berlin, Heidelberg (2003)
15. ROSE compiler infrastructure, http://rosecompiler.org/.
16. Baranov, M.S., Ivanov, D.I., Kataev, N.A., Smirnov, A.A.: Automated parallelization of sequential C-programs on the example of two applications from the field of laser material processing. // CEUR Workshop Proceedings 1st Russian Conference on Supercomputing Days 2015, vol. 1482, pp. 536, 2015.
17. R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, F. K. Zadeck. An Efficient Method of Computing Static Single Assignment Form // Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 25-35, January, 1989
18. Clang: a C language family frontend for LLVM, https://clang.llvm.org/.
19. GitHub - flang-compiler/flang. URL: https://github.com/flang-compiler/flang.
20. Dwarf 3 Standard. URL: http://eagercon.com/dwarf/dwarf3std.htm