2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

doi 10.26089/NumMet.v22r415 УДК 04.272, 004.4'242

Дополнительное распараллеливание MPI программ с помощью системы SAPFOR

Н. А. Катаев
Институт прикладной математики им. М. В. Келдыша РАН (ИПМ РАН), Москва, Российская Федерация
ORCID: https://orcid.org/0000-0002-7603-4026, e-mail: kataev_nik@mail.ru
А. С. Колганов
Институт прикладной математики им. М. В. Келдыша РАН (ИПМ РАН), Москва, Российская Федерация
ORCID: https://orcid.org/0000-0002-1384-7484, e-mail: alexander.k.s@mail.ru

Аннотация: Системы SAPFOR и DVM были спроектированы и предназначены для упрощения разработки параллельных программ научно-технических расчетов. Главной целью системы SAPFOR является автоматизация процесса отображения последовательных программ на параллельные архитектуры в модели DVMH. В некоторых случаях пользователь системы SAPFOR может рассчитывать на полностью автоматическое распараллеливание, если программа была написана или приведена к потенциально параллельному виду. DVMH модель представляет собой расширение стандартных языков C и Fortran спецификациями параллелизма, которые оформлены в виде директив и не видимы стандартным компиляторам. В статье будет рассмотрено автоматизированное дополнительное распараллеливание существующих MPI-программ с помощью системы SAPFOR, где, в свою очередь, будут использованы новые возможности DVMH модели по распараллеливанию циклов в MPI программе внутри узла. Данный подход позволяет существенно снизить трудоемкость распараллеливания MPI программ на графические ускорители и многоядерные процессоры, сохранив при этом удобство сопровождения уже написанной программы. Данная возможность в системе SAPFOR была реализована для языков Fortran и C. Эффективность данного подхода показана на примере некоторых приложений из пакета NAS Parallel Benchmarks.

Ключевые слова: SAPFOR, DVMH, MPI, автоматизация распараллеливания, дополнительное распараллеливание, ускорители, гетерогенные кластеры.

Для цитирования: Колганов А. С., Катаев Н. А. Дополнительное распараллеливание MPI программ с помощью системы SAPFOR // Вычислительные методы и программирование. 2021. **22**, 239–251. doi 10.26089/NumMet.v22r415.

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

Additional parallelization of existing MPI programs using SAPFOR

Nikita A. Kataev
Keldysh Institute of Applied Mathematics, Moscow, Russia
ORCID: https://orcid.org/0000-0002-7603-4026, e-mail: kataev_nik@mail.ru
Alexander S. Kolganov
Keldysh Institute of Applied Mathematics, Moscow, Russia
ORCID: https://orcid.org/0000-0002-1384-7484, e-mail: alexander.k.s@mail.ru

Abstract: The SAPFOR and DVM systems are primarily designed to simplify the development of parallel programs of scientific-technical calculations. SAPFOR is a software development suite that aims to produce a parallel version of a sequential program in a semi-automatic way. The fully automatic parallelization is also possible if the program is well-formed and satisfies certain requirements. SAPFOR uses the DVMH directive-based programming model to expose parallelism in the code. The DVMH model introduces CDVMH and Fortran-DVMH (FDVMH) programming languages which extend the standard C and Fortran languages by parallelism specifications. We present MPI-aware extension of the SAPFOR system that exploits opportunities provided by the new features of the DVMH model to extend existing MPI programs with intra-node parallelism. In that way, our approach reduces the cost of parallel program maintainability and allows an MPI program to utilize accelerators and multi-core processors. SAPFOR extension has been implemented for both Fortran and C programming languages. In this paper, we use the NAS Parallel Benchmarks to evaluate the performance of generated programs.

Keywords: SAPFOR, DVMH, MPI, automation of parallelization, additional parallelization, accelerators, heterogeneous clusters.

For citation: A. S. Kolganov, N. A. Kataev, "Additional parallelization of existing MPI programs using SAPFOR," Numerical Methods and Programming. **22** (4), 239–251 (2021). doi 10.26089/NumMet.v22r415.

1. Введение. Многопроцессорные вычислительные системы уже достаточно давно используются для проведения расчетов, накопился достаточно большой объем параллельных программ, способных задействовать несколько узлов вычислительного кластера. При этом одним из основных инструментов разработки таких программ остается стандарт MPI. Но появившиеся сравнительно недавно гетерогенные вычислительные системы, узлы которых образованы многоядерными процессорами и ускорителями, требуют применения дополнительных средств разработки параллельных программ. Чтобы эффективно использовать все имеющиеся вычислительные ресурсы, в MPI-программы приходится добавлять дополнительные спецификации параллелизма. Низкоуровневые модели параллельного программирования (CUDA, OpenCL, POSIX Threads) дают полный контроль за процессом выполнения программы. Но в сочетании с тем, что разработчику также приходится разбираться в структуре исходной MPI-программы, которая может быть существенно сложнее аналогичной последовательной программы, низкоуровневые модели существенно увеличивают сложность разработки кода. Отладка таких MPI-программ, для которых было выполнено дополнительное распараллеливание, также является трудоемкой задачей.

Решением в данной ситуации может быть использование высокоуровневых моделей программирования (OpenMP, OpenACC), узкоспециализированных языков программирования (DSL) [1–3], а также параллельных библиотек общего назначения [4, 5]. Дополнением к этому может стать создание инструментальных средств, автоматизирующих процесс дополнительного распараллеливания MPI программ.

Стандарты OpenMP и OpenACC активно развиваются в указанном выше направлении, но все еще не позволяют в полной мере задействовать все ресурсы вычислительного узла. В современных компиляторах данные стандарты реализованы не полностью [6], а для получения существенного выигрыша

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

от их использования пользователю приходится разбираться в том, как высокоуровневые спецификации влияют на выполнение программы, погружаясь в детали их реализации в конкретном компиляторе и подбирая соответствующие оптимизационные опции. Более того, в данный момент отсутствуют удобные высокоуровневые средства как для функциональной отладки, так и для отладки эффективности данных программ.

Если рассматривать автоматически распараллеливающие компиляторы как отдельные инструменты, то их возможности оказываются существенно ограниченными и не позволяют выполнить приемлемое распараллеливание в узле кластера для реальных программ.

В данной ситуации перспективным видится смешанный подход, объединяющий использование высокоуровневой модели параллельного программирования в качестве целевого способа описания параллелизма в программе, и системы автоматизации, которая осуществляет наиболее трудоемкие и приводящие к ошибкам этапы распараллеливания автоматически, но предоставляет пользователю контроль за ходом распараллеливания, достаточный для получения эффективных программ.

В качестве высокоуровневой модели может выступать модель DVMH [7, 8], включающая в себя DVMH-языки параллельного программирования, являющиеся директивными расширениями стандартных последовательных языков программирования C и Fortran. Одним из основных преимуществ DVMH-языков, наряду с высокоуровневой семантикой используемых конструкций, являются реализованные оптимизации времени выполнения программы. Все это существенно снижает сложность разработки инструментов, автоматизирующих процесс распараллеливания. Кроме того, DVM-система включает в свой состав инструменты для функциональной отладки и отладки эффективности создаваемых программ.

Система SAPFOR (System FOR Automated Parallelization) [9, 10] может выступать в качестве инструмента автоматизации параллельного программирования. С одной стороны, в состав системы входит автоматически распараллеливающий компилятор [11], а с другой — SAPFOR предоставляет пользователю дополнительные возможности по контролю за процессом распараллеливания как с использованием графического интерфейса пользователя [12], так и с помощью дополнительного указания свойств программы непосредственно в ее исходном коде. Чтобы расширить доступные возможности по анализу программ, в системе SAPFOR наряду со статическим анализом [13] также поддерживается динамический анализ [14]. Система предоставляет пользователю набор реализованных преобразований для языков Fortran и C (подстановка функций, удаление мертвого кода, распространение выражений и др.), которые пользователь может выбрать для автоматического выполнения.

Изначально модель DVMH была разработана с целью скрыть от пользователя не только параллелизм внутри узла, но и обеспечить распараллеливание программ для систем с распределенной памятью. Но в последних реализациях компиляторов с DVMH-языков был добавлен дополнительный режим, направленный на использование возможностей DVMH-модели в MPI-программах. В данной статье рассматриваются новые возможности системы SAPFOR, которые предназначены для автоматизации дополнительного распараллеливания MPI-программ и опираются на возможности, добавленные в систему DVM.

Статья состоит из введения, четырех разделов и заключения. В разделе 2 приведен краткий обзор работ, направленных на автоматизацию разработки параллельных программ для систем с общей памятью (многоядерных процессоров и графических ускорителей). В разделе 3 рассматриваются возможности, добавленные в DVMH модель для реализации дополнительного внутриузлового параллелизма в MPI-программах. Раздел 4 посвящен системе SAPFOR и особенностям, связанным с автоматизацией дополнительного распараллеливания MPI-программ. Результаты вычислительных экспериментов, охватывающие распараллеливание некоторых программ из набора NAS Parallel Benchmarks [15], приведены в разделе 5.

2. Обзор существующих работ. Можно выделить два основных подхода, применяемых при разработке инструментов, автоматизирующих распараллеливание программ. Подход, наиболее понятный для пользователя и поэтому используемый в SAPFOR в силу ее направленности на поддержание процесса интерактивного распараллеливания, заключается в последовательном изменении кода программы, чтобы впоследствии привести ее к параллельному виду. Многие из выполняемых преобразований оказываются общими для разных программ и могут быть автоматизированы (подстановка функций, разделение и слияние циклов, разворот цикла и др. [16]). Часть из этих преобразований уже реализована в системе SAPFOR и при необходимости может быть запрошена пользователем. Альтернативой этому подходу является второй подход — использование математической модели, описывающей поведение распараллеливаемой программы (модель многогранников, или полиэдральная модель). Этот подход позволяет сформи-

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

ровать оптимизационную задачу, решение которой фактически описывает переход от исходной программы к ее параллельной версии за один шаг преобразования.

Одним из основных недостатков второго подхода является большое количество ограничений, накладываемых на распараллеливаемую программу: в результате такой подход годится, в первую очередь, для локального применения к хорошо структурированным участкам кода (SCoP). Кроме того, степень изменений, вносимых в код программы, которые порождает решение оптимизационной задачи, оказывается такова, что пользователь фактически лишается возможности участвовать в процессе распараллеливания, даже если результирующая параллельная программа остается на языке высокого уровня. Таким образом, альтернативный подход в первую очередь подходит для автоматически распараллеливающих компиляторов [17–21], а не инструментов автоматизации.

Компиляторы Pluto [17] и PPCG [18] направлены на преобразование только C-программ, при этом Pluto выполняет распараллеливание только на многоядерные процессоры (с использованием OpenMP), в то время как PPCG опирается на CUDA или OpenCL для отображения программ на графические ускорители. Оба инструмента сохраняют высокий уровень исходного языка, но вносимые изменения негативно сказываются на возможности восприятия кода неподготовленным программистом. Существенным недостатком является необходимость явно задавать фрагменты исходного кода, которые должны быть оптимизированы. Для этого используется специальная директива `scop`. Так как оптимизация выполняется локально, то от задания данной директивы существенно зависит результат распараллеливания. Например, если соседние гнезда циклов поместить в разные области распараллеливания, то PPCG добавит в код программы вызовы CUDA, выполняющие обмены с графическим ускорителем в начале и в конце каждой области: глобальная оптимизация обменов не выполняется. Кроме того, Pluto и PPCG не обрабатывают участки кода, содержащие редукционные операции, и оставляют соответствующий код последовательным. В этом случае вычисления не могут быть полностью выполнены на ускорителях и требуются дополнительные обмены данными с центральным процессором.

Другим примером применения модели многогранников является инструмент Polly [19] и его дальнейшее расширение для отображения программ на графические ускорители Polly-ACC [20]. Данные инструменты используют LLVM [22] для анализа и преобразования программ и доступны в составе компилятора Clang. Использование низкоуровневого представления (LLVM IR) лишает пользователя возможности каклибо изменять код параллельной программы, но при этом расширяется применимость данных инструментов к языкам, для которых может быть построено LLVM IR. В отличие от Pluto и PPCG, оптимизируемые участки кода определяются автоматически и также выполняется попытка оптимизировать обмены с графическим ускорителем. Отдельно была реализована возможность распараллеливания редукционных операций [23]. Хотя оптимизируемые участки кода определяются автоматически, выполнить распараллеливание удается не всегда: инструменты опираются на статический анализ кода, возможности которого ограничены, особенно если в программах используются указатели и адресная арифметика. Чтобы понять, как привести программу в распараллеливаемую форму, пользователю может потребоваться изучить возникшие проблемы анализа, описываемые в терминах низкоуровневого представления LLVM IR.

Еще одним интересным инструментом является Apollo [21], осуществляющий спекулятивное распараллеливание программы во время выполнения. Но в данном инструменте реализована возможность распараллеливания только для многоядерных процессоров.

3. Возможности модели DVMH для распараллеливания MPI программ. Обычно DVM подход [7] предполагает распараллеливание последовательной программы, в которую программист вставляет директивы распределения данных, а затем — директивы распределения вычислений. После этого компилятор DVMH преобразует такую программу в программу в модели MPI+OpenMP+CUDA, которая затем с помощью библиотеки времени выполнения (runtime system, RTS) выполняется на гетерогенном кластере. RTS берет на себя распределение данных по узлам кластера согласно тому, как были расставлены директивы распределения данных, а затем отображает соответствующие этим данным вычисления в параллельных циклах. Таким образом, применение данного подхода в MPI-программах потребует от программиста либо наличия исходной последовательной версии, либо перевода MPI программы обратно в последовательную версию. Зачастую перевод MPI-программы в исходную последовательную попросту невозможен либо требует достаточно больших усилий. Кроме того, в этом случае достаточно велика вероятность совершения ошибки.

Для решения описанных проблем в DVM системе был введен специальный режим, в котором RTS не выполняет никаких межпроцессорных взаимодействий и не препятствует работе MPI-функций, исполь-

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

зуемых в программе пользователя. Чтобы задействовать данный режим, требуется специальная сборка DVM системы, в результате которой RTS работает локально в каждом MPI процессе вне зависимости от того, на какой процессорной решетке была запущена программа.

Как уже было отмечено выше, изначально DVMH модель предполагает отображение витков параллельных циклов на элементы распределенных массивов в терминах DVMH. В MPI программе пользователь сам берет на себя ответственность за распределение данных. Поэтому потребовалось ввести новую возможность в DVMH модель для того, чтобы отобразить параллельные циклы в такой программе на графические ускорители и многоядерные процессоры. Данная возможность позволяет локально в каждом MPI процессе распараллеливать витки параллельных циклов без привязки к элементам массивов. При этом, несмотря на отсутствие распределения данных в рамках модели DVMH, сохраняется возможность использования оптимизаций времени выполнения программ, доступных в DVM системе. К ним относятся: использование автоматической трансформации данных на графическом процессоре, а также использование интеллектуального механизма управления перемещением данных между памятью CPU и GPU; использование оптимизирующих настроек RTS системы. Также сохраняется возможность использования удобных средств по отладке и анализу производительности MPI+DVMH программ.

В модели DVMH распределение данных выполняется при помощи выравнивания (директивы `align`), которое каждому элементу массива `A` ставит в соответствие элемент или секцию массива `B`. При отображении на какой-либо процессор элемента массива `B` на этот же процессор будет распределен и элемент массива `A` в соответствии с правилами выравнивания данных. Распределение вычислений в модели DVMH выполняется с учетом распределения данных. Правило отображения параллельного гнезда циклов задается при помощи спецификации `on`, которая позволяет связать циклы тесно вложенного гнезда с измерениями распределенных массивов.

Данная информация позволяет компилятору и RTS осуществлять ряд оптимизаций, которые могут существенно повысить эффективность выполнения параллельной программы. Например, зная о том, как связаны массивы между собой, и зная правила отображения цикла, RTS может определить оптимальное для данного цикла представление массивов в памяти вычислительного устройства и выполнить динамическую трансформацию массивов только в тех случаях, когда это требуется. Вследствие этого, на графическом ускорителе все обращения к глобальной памяти будут выполняться CUDA-нитями одного варпа наилучшим образом: соседние нити блока будут обращаться к соседним ячейкам памяти, в результате чего параллельный цикл может выполняться до 10 раз быстрее [24].

При распараллеливании MPI программ в модели DVMH описанные ранее директивы `distribute`, `align` и `on` не используются. Программист сам реализует распределение данных и распределение вычислений по узлам кластера с помощью MPI. Для задания связи между витками параллельных циклов и используемыми в них массивами в DVMH модель была введена новая спецификация `tie` [25], а также понятие нераспределенного параллельного цикла. При помощи данной спецификации программист передает RTS информацию о том, с каким циклом в гнезде есть связь у конкретного измерения массива (а также направление этой связи — прямое или обратное), если связи нет, то указывается `«*»`. Измерение массива может быть отображено только на один из параллельных циклов в гнезде.

Для параллельного выполнения циклов с регулярными зависимостями по данным на графических ускорителях в DVM системе реализован метод гиперплоскостей. Все элементы, лежащие на одной гиперплоскости, могут быть вычислены независимо друг от друга. При таком порядке выполнения витков цикла снова возникает проблема эффективного доступа к глобальной памяти в силу того, что параллельно обрабатываются не локально расположенные в памяти элементы массивов. Для эффективного выполнения таких циклов в RTS реализована диагональная трансформация, в результате которой соседние элементы массивов на диагоналях (в плоскости необходимых измерений) располагаются в соседних ячейках памяти, что позволяет применять технику выполнения цикла с зависимостями по гиперплоскостям без значительной потери производительности на операциях доступа к глобальной памяти графического ускорителя. Для обеспечения эффективного выполнения таких циклов в MPI +DVMH программе совместно применяются спецификации `tie` и `across` для всех необходимых массивов.

На листингах 1 и 2 показаны примеры нераспределенного параллельного цикла, который может выполняться на многоядерном процессоре или графическом процессоре.

Для того чтобы параллельный цикл мог быть отображен на многоядерные CPU или GPU, необходимо его окружить спецификацией `region`, которая обозначает начало и конец вычислительной области [8]. В вычислительную область выделяется часть программы с одним входом и одним выходом для возмож-

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

Листинг 1. Нераспределенный параллельный цикл в языке CDVMH
Listing 1. Undistributed parallel loop in CDVMH language

```
#pragma dvm region targets(CUDA) out(A)
{
#pragma dvm parallel([i][j][k]) tie(A[i][j][k])
for (int i = Li; i <= Hi; i++)
  for (int j = Lj; j <= Hj; j++)
    for (int k = Lk; k <= Hk; k++)
         A[i][j][k] = ....
}
```

Листинг 2. Нераспределенный параллельный цикл в языке FDVMH
Listing 2. Undistributed parallel loop in FDVMH language

```
!DVM$ REGION TARGETS(HOST) OUT(A)
!DVM$ PARALLEL(I,J,K) TIE (A(K,J,I))
 DO I = Li, Hi
   DO J = Lj, Hj
     DO K = Lk, Hk
       A(K,J,I) = ....
!DVM$ END REGION
```

ного ее выполнения на многоядерном процессоре или графическом ускорителе. Статически или динамически вложенные регионы не допускаются. Для выполнения вычислительных регионов на GPU требуется соблюдение некоторых ограничений на ввод/вывод и вызовы процедур из параллельных циклов. Регион может быть выполнен только на CPU или только на GPU. Данная возможность обеспечивается указанием необязательной спецификации `target`.

Фрагменты кода, которые не входят в вычислительные регионы, всегда выполняются на центральном процессоре, что приводит к необходимости актуализации данных между памятью CPU и памятью GPU. Для выполнения необходимых обменов в программе должны быть размещены директивы `actual` и `get_actual`.

Директива `get_actual` делает все необходимые обновления для того, чтобы в памяти центрального процессора были актуальные (то есть самые новые) значения данных в указанных в списке массивах и скалярах. Директива является исполняемой и выполняется в той точке программы, где она была поставлена. Директива `actual` объявляет тот факт, что указанные в списке массивы и скаляры имеют самые новые значения в памяти центрального процессора. Значения указанных переменных и элементов массивов, находящиеся в памяти ускорителей, считаются устаревшими и перед использованием будут при необходимости обновлены. В данном случае помечается только сам факт того, что данные были обновлены на CPU, но физического копирования в точке постановки директивы не происходит.

Отложенная семантика директив актуализации позволяет RTS избегать избыточной передачи данных. Более того, нет никакой разницы, все ли регионы выполняются на GPU или какая-то их часть предназначена для выполнения на CPU. Директивы актуализации влияют только на передачу данных между регионами и последовательными фрагментами кода. Система поддержки в любых случаях будет управлять необходимой передачей данных в автоматическом режиме. Реализованная автоматизация перемещения данных между CPU и GPU в DVMH модели позволяет системе SAPFOR автоматически расставлять директивы актуализации данных оптимальным образом.

4. Автоматизация дополнительного распараллеливания MPI программ с помощью системы SAPFOR. Для того чтобы задействовать весь новый потенциал для распараллеливания MPI-программ с помощью DVMH-модели, необходимо добавить в код программы спецификацию `tie`, а также изменить подход к анализу программы в системе SAPFOR: больше нет необходимости выполнять распределение данных и в соответствии с ним распределение вычислений.

На первом этапе системой SAPFOR для распараллеливания отбираются самые внешние гнезда тесновложенных циклов. Каждое гнездо циклов проверяется на предмет того, может ли оно выполняться параллельно: из цикла не должно быть побочных выходов, не должно быть неразрешимых зависимостей между витками цикла, цикл должен быть в канонической форме и др. [11]. Для каждого цикла также определяются входные и выходные данные, которые используются в нем, так как в компиляторе CDVMH

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

не реализован межпроцедурный анализ и в основном все данные считаются как входными, так и выходными. Консервативная информация о входных и выходных данных в вычислительных областях может привести к дополнительным копированиям между GPU и CPU.

На следующем этапе системе SAPFOR необходимо проанализировать все обращения к памяти, которые выполняются в теле рассматриваемого гнезда циклов. В каждом выражении в обращении к массиву ищутся индуктивные переменные циклов для того, чтобы выполнить сопоставление измерений массивов с витками циклов. Для этого выполняется попытка привести выражения в форму, зависящую от номера витка цикла за счет анализа внутреннего представления программы (expression propagation, scalar evolution). В силу того, что распределение данных строится пользователем с помощью MPI, нет необходимости определять точное соответствие измерений массивов с витками циклов, которое в DVMH-модели выражается с помощью линейной связи. Тем самым система SAPFOR может распараллеливать циклы и с косвенной адресацией, так как нам достаточно установить факт использования индуктивной переменной цикла в том или ином измерении массива.

После того как был определен набор параллельных циклов, системе SAPFOR необходимо разделить программу на вычислительные регионы, в которые будут входить параллельные циклы, и на внерегионное пространство, которое будет выполняться на CPU в последовательном режиме. Алгоритм расстановки регионов состоит из двух этапов. На первом этапе каждый параллельный цикл окружается своим собственным регионом. А на втором этапе происходит объединение соседних регионов в один для уменьшения накладных расходов на вход и выход из региона.

После того как программа была поделена на регионы, необходимо обеспечить консистентность данных между CPU и GPU. Чтобы уменьшить количество копирований, необходима расстановка директив актуализации данных оптимальным образом. DVM-система обеспечивает ряд оптимизаций времени выполнения программы, что облегчает работу системе SAPFOR. Например, если пользователь вставит две одинаковые директивы `get_actual` подряд, то копирование будет выполнено только один раз.

Основная цель на данном этапе работы системы SAPFOR — определить корректный набор данных, который необходимо обновить в памяти GPU или CPU. Для данного анализа используются ранее собранные входные и выходные данные для каждого из циклов. Также используется анализ псевдонимов в случае косвенной адресации в программах на языке Си. Система SAPFOR выполняет актуализацию только тех переменных, которые используются на чтение или запись внутри вычислительных регионов. На основе полученных данных система SAPFOR размещает директиву `actual` после каждого присваивания переменной, которое выполняется во внерегионном пространстве. Директива `get_actual` ставится перед теми операторами, в которых присутствует чтение из переменных, которые были в свою очередь модифицированы в каком-либо вычислительном регионе.

Промышленные программы обычно состоят из большого количества процедур и функций. Система SAPFOR определяет три класса таких процедур: встроенные процедуры (например, математические функции), пользовательские функции и внешние процедуры. В случае наличия внешних процедур системе SAPFOR приходится принимать консервативное решение вставлять директивы актуализации всех используемых в вычислительных регионах переменных до и после вызова, так как тело процедуры недоступно для анализа. Для уменьшения накладных расходов на копирование данных пользователь может подсказать системе SAPFOR, например может указать, что процедура не содержит побочных эффектов. Так как функции MPI являются внешними и информация об использовании в параметрах этих функций переменных является стандартизированной, мы выделили их в системе SAPFOR в отдельный четвертый класс — класс MPI-функций, что позволяет расставлять директивы актуализации оптимально без подсказок пользователя.

Использование указателей в программах может негативно сказаться на результатах анализа в системе SAPFOR, так как это препятствует выполнению некоторых преобразований внутреннего представления программы. В частности, используемый в системе SAPFOR анализ редукционных операций не способен обработать переменные, каким-либо образом участвующие в операциях адресной арифметики. Так как редукционная операция — это коллективная операция, выполняемая сразу всеми/несколькими процессорами, то для обмена значениями соответствующих переменных будут использованы функции библиотеки MPI. При этом для описания передаваемых данных будет использован указатель на буфер с данными (возможно, состоящий из единственной переменной).

Чтобы расширить возможности системы SAPFOR и осуществить анализ редукционных операций, в MPI-программах было реализовано дополнительное преобразование внутреннего представления програм-

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

мы в системе. Данное преобразование скрыто от пользователя и не влияет на исходный код распараллеливаемой программы, но расширяет возможности для анализа программ. Для тех циклов, в которых была предварительно выявлена зависимость по некоторой переменной, участвующей в операциях адресной арифметики, заводится локальный дубликат этой переменной и обращения к переменной в цикле заменяются на обращения к ее дубликату. Это позволяет воспользоваться уже имеющимся в SAPFOR анализом редукционных переменных.

Возможности анализа предыдущих версий SAPFOR также были ограничены необходимостью во многих случаях выполнять предварительную подстановку функций на уровне исходного кода программы [11]. Нами была добавлена возможность определять функции, которые необходимо подставить, автоматически на основе предварительных результатов анализа программы. Более того, выполнение подстановки функций было перенесено на уровень внутреннего представления программы и, таким образом, исходный код программы остается неизменным, если его преобразование не требуется для задания спецификаций параллелизма.

5. Вычислительные эксперименты. Рассмотрим возможности дополнительного распараллеливания MPI-программ, предоставляемые системой SAPFOR совместно с системой DVM, на примере распараллеливания вычислительных приложений BT, CG и EP из пакета NAS Parallel Benchmarks.

Эффективность выполнения исходных MPI-программ, а также полученных с помощью системы SAPFOR MPI-программ с DVMH-директивами, была исследована на суперкомпьютере K60 [26], состоящем из процессоров Intel Xeon Gold 6142v4 и графических ускорителей NVIDIA V100 GPUs с архитектурой Volta. Каждый узел содержит два 16ти ядерных процессора (CPU), связанных посредством общей памяти (архитектура NUMA), и четыре графических ускорителя (GPU). Пиковая производительность узла составляет примерно 60 TFLOPs в вычислениях одинарной точности и 30 TFLOPs в вычислениях двойной точности. Таким образом, для достижения высокой производительности параллельных программ, достаточно задействовать небольшое количество вычислительных узлов. Для вычислительных экспериментов были использованы максимально возможные ресурсы четырех узлов: для MPI-программ использовалось до 128 ядер CPU, а для MPI+DVHM-программ — до 128 ядер CPU и 16 GPU. Время выполнения различных версий MPI-программ, написанных на языках Fortran и C, приведены в табл. 1 и 2 соответственно.

Таблица 1. Время выполнения в секундах программ на языке Fortran, NPB 3.3 класс D
Table 1. Times in seconds of Fortran programs, NPB 3.3 class D

	MPI программы MPI programs			Преобразованные MPI программы Converted MPI programs			MPI программы + **FDVMH** MPI programs + **FDVMH**		
	BT	CG	EP	BT	CG	EP	BT	CG	EP
1 узел 1 node	665.1	397.5	93.68	785.29	376.8	83.34	63.3	80.99	0.62
2 узла 2 nodes	361.6	209.6	46.53	428.07	229.61	42.06	50.3	42.6	0.38
4 узла 4 nodes	196.8	91.3	23.3	232.36	96.67	25.16	45.5	34.3	0.17

Таблица 2. Время выполнения в секундах программ на языке Си, NPB 3.3 класс D
Table 2. Times in seconds of C programs, NPB 3.3 class D

	MPI программы MPI programs			Преобразованные MPI программы Converted MPI programs			MPI программы + **CDVMH** MPI programs + **CDVMH**		
	BT	CG	EP	BT	CG	EP	BT	CG	EP
1 узел 1 node	694.6	326.16	98.41	768.5	328.8	99.37	97.7	186.12	0.67
2 узла 2 nodes	386	218.9	49.29	421.3	214.3	50.05	75.7	96.75	0.38
4 узла 4 nodes	215	89.2	24.71	229.06	92.46	26.5	67.33	71.79	0.17

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

Время выполнения оригинальных версий, написанных разработчиками пакета NAS Parallel Benchmark, приведено в первой группе столбцов (MPI-программы). Изначально рассматриваемые программы были написаны только на Fortran, поэтому потребовалось выполнить перевод рассматриваемых программ на язык C вручную.

Чтобы получить автоматически распараллеливаемые версии программ, были выполнены их предварительные преобразования (подстановка функций, объединение циклов, сужение размерности приватных массивов) [11]. Вторая группа столбцов (Преобразованные MPI-программы) показывает время выполнения преобразованных версий. Большинство преобразований было выполнено системой SAPFOR, другая часть преобразований, не реализованных на данный момент в системе или специфичных для конкретной программы и трудно формализуемых в виде отдельного преобразования, была выполнена вручную.

Время выполнения программ, полученных после автоматического распараллеливания преобразованных версий, приведено в третьей группе столбцов (MPI-программы + DVMH).

Для выполнения исходных и преобразованных MPI программ использовалось максимально возможное количество ядер CPU одного, двух или четырех узлов. А для выполнения MPI+DVHM-программ использовалось максимально возможное количество GPU в узле. Исходя из приведенных времен можно сделать вывод, что преобразования практически не замедляют выполнение программы, позволяя использовать систему SAPFOR для автоматического распараллеливания циклов на GPU и многоядерные CPU в рамках одного узла.

На данный момент компиляторы CDVMH и FDVMH реализуют различный набор оптимизаций при выполнении конвертации кода. В компиляторе CDVMH отсутствуют некоторые оптимизации, доступные в компиляторе FDVMH, что является основной причиной в различии ускорений MPI+DVMH-программ. В некоторых случаях компилятор FDVMH способен дополнительно распараллелить не тесно вложенные циклы. Это позволяет значительно повысить эффективность выполнения программы CG на графическом процессоре.

Максимально возможное ускорение приложения BT с использованием четырех графических процессоров составляет около 10.5 раз по сравнению с 32 MPI-процессами. Ускорение выполнения программ CG и EP на графических процессорах при использовании двух узлов постоянно и не зависит от конфигурации. В данном случае оно составляет примерно 5 раз и 151 раз соответственно. При использовании четырех узлов для программы CG не хватает данных для вычисления, поэтому ускорение падает до 2.6 раз.

Такая разница в полученном ускорении в разных приложениях обусловлена их разной алгоритмической сложностью. Программа EP производит большое количество независимых вычислений, причем обращений к памяти практически нет. Тем самым в данном приложении можно видеть наибольшее полученное ускорение. В программе CG основная доля времени приходится на умножение разреженной матрицы на вектор, что приводит к косвенной индексации и снижению общей производительности в целом.

И наконец, программа BT состоит из большого количества вычислений, большого количества обращений к памяти, а также циклов с регулярными зависимостями по данным. В результате этого, в программе появляется большое количество межпроцессорных обменов, которые сильно сказываются на масштабируемости программы при переходе с одного узла на два и более. Рис. 1 показывает соотношение времен, затрачиваемых программой на вычисления и на накладные расходы (коммуникации, пересылки CPU–GPU). Из графиков видно, что количество коммуникаций и связанных с ними копирований данных с CPU на GPU и обратно при увеличении количества процессов вдвое не сокращается и не зависит от количества процессов. При этом время, затрачиваемое на выполнение параллельных циклов на графическом процессоре, пропорционально уменьшается в соответствии с количеством использованных устройств.

6. Заключение. В статье были рассмотрены новые возможности модели DVMH и их применения в системе SAPFOR для автоматизированного дополнительного распараллеливания существующих MPI-программ на примере некоторых программ из пакета NAS Parallel Benchmarks.

Предлагаемый нами подход к разработке параллельных программ сочетает модель программирования на основе директив (DVMH) и инструменты автоматизации и взаимодействия с пользователем (SAPFOR). Разрабатываемые системы взаимно дополняют друг друга, что, в свою очередь, позволяет получить существенное преимущество по сравнению с другими моделями параллельного программирования, такими как MPI+OpenMP или MPI+OpenACC, при разработке параллельных программ для гетерогенных кластеров.

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

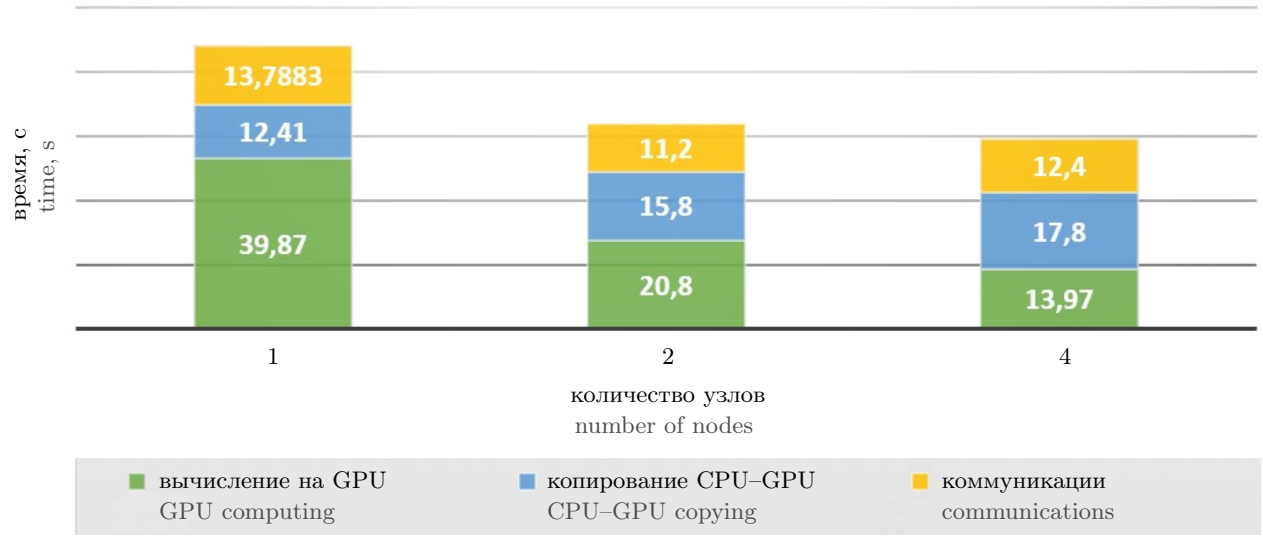

Рис. 1. Соотношение времени вычислений и коммуникаций на примере BT

Fig. 1. The ratio of computation time to communication time in the BT application

Прежде всего, в систему SAPFOR входит автоматически распараллеливающий компилятор, который успешно справляется с потенциально параллельными программами без вмешательств со стороны пользователя. Если же реализованный в системе SAPFOR анализ кода не справляется, то пользователь может помочь системе с помощью соответствующих директив, указав недостающие свойства программы, либо выполнить с помощью системы SAPFOR необходимые преобразования, которые не приводят к снижению производительности и при этом повышают уровень ее потенциальной параллельности. Можно отметить, что преобразования производятся на уровне исходного кода и не требуют детального изучения параллельных архитектур и языков параллельного программирования.

Благодаря заложенным в модели DVMH автоматизации и оптимизации перемещения данных между памятью CPU и памятью GPU SAPFOR может эффективно автоматизированно отображать (а в некоторых случаях полностью автоматически) существующие MPI-программы на графические процессоры. Кроме того, система SAPFOR опирается на оптимизации, осуществляемые DVM-системой во время выполнения программы: трансформацию массивов во время выполнения программы для обеспечения наилучшего доступа к памяти GPU, динамическую компиляцию и оптимизацию CUDA-обработчиков во время выполнения программы, эффективное выполнение параллельных циклов с регулярными зависимостями по данным на GPU и др. Это позволяет достичь высокой эффективности получаемых MPI+DVMH-программ. В то же время данные оптимизации скрыты от пользователя и доступны через высокоуровневые спецификации параллелизма DVMH-модели, таким образом упрощая поддержку разрабатываемых параллельных кодов.

В DVM-систему также входят средства анализа эффективности выполнения программ. Выдаваемые характеристики выполнения программ описывают проблемы потери эффективности понятными пользователю терминами. Все получаемые характеристики выполнения привязаны к DVMH-директивам и исходному коду. В дальнейшем система SAPFOR сможет использовать полученные результаты анализа эффективности для оптимизаций MPI-программы.

Таким образом, системы SAPFOR и DVM могут значительно сократить усилия, необходимые для дополнительного распараллеливания MPI-программ, позволяющего задействовать все доступные внутри узла ресурсы (многоядерные процессоры и графические ускорители), а также существенно повысить эффективность выполнения параллельной MPI-программы. Мы надеемся, что данный подход поможет эффективной разработке и оптимизации масштабируемых программ для суперкомпьютеров.

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

Список литературы

1. *Ragan-Kelley J., Barnes C., Adams A., Paris S., Durand F., Amarasinghe S.P.* Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines // SIGPLAN Notices. 2013. **48**, N 6. 519–530. doi 10.1145/2499370.2462176.
2. *Beaugnon U., Kravets A., van Haastregt S., Baghdadi R., Tweed D., Absar J., Lokhmotov A.* Vobla: a vehicle for optimized basic linear algebra // SIGPLAN Notices. 2014. **49**, N 5. 115–124. doi 10.1145/2666357.2597818.
3. *Zhang Y., Yang M., Baghdadi R., Kamil S., Shun J., Amarasinghe S.* Graphit: a high-performance graph DSL // Proc. ACM Program. Lang. 2018. **2**, Issue OOPSLA. 121:1–121:30. doi 10.1145/3276491.
4. *An P., Jula A., Rus S., Saunders S., Smith T., Tanase G., Thomas N., Amato N., Rauchwerger L.* STAPL: An adaptive, generic parallel C++ library // Lecture Notes in Computer Science. Vol. 2624. Berlin: Springer, 2003. 193–208. https://doi.org/10.1007/3-540-35767-X_13
5. *Bell N., Hoberock J.* Thrust: a productivity-oriented library for CUDA // GPU Computing Gems Jade Edition. 2012. pp. 359–371. doi 10.1016/B978-0-12-385963-1.00026-5.
6. Компиляторы и инструменты OpenMP. https://www.openmp.org/resources/openmp-compilers-tools/. Дата обращения: 15 октября 2021.
7. *Konovalov N.A., Krukov V.A, Mikhajlov S.N., Pogrebtsov A.A.* Fortran DVM: a language for portable parallel program development // Programming and Computer Software. 1995. **21**, N 1. 35–38.
8. *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 ЮУрГУ. Сер. Математическое моделирование и программирование. 2012. № 12. 82–92.
9. *Клинов М.С., Крюков В.А.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестн. Нижегородского гос. ун-та им. Н.И. Лобачевского. 2009. № 2. 128–134.
10. *Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В.* Диалог с программистом в системе автоматизации распараллеливания САПФОР // Вестн. Нижегородского гос. ун-та им. Н.И. Лобачевского. 2012. № 5(2). 242–245.
11. *Kataev N.* LLVM based parallelization of C programs for GPU // Communications in Computer and Information Science. Vol. 1331. Cham: Springer, 2020. 436–448. https://doi.org/10.1007/978-3-030-64616-5_38.
12. *Kataev N.* Interactive parallelization of C programs in SAPFOR // CEUR Workshop Proceedings. Vol. 2784. 2020. 139–148. http://ceur-ws.org/Vol-2784/
13. *Kataev N.* Application of the LLVM compiler infrastructure to the program analysis in SAPFOR // Communications in Computer and Information Science. Vol. 965. Cham: Springer, 2018. 487–499. doi 10.1007/978-3-030-05807-4_41.
14. *Kataev N., Smirnov A., Zhukov A.* Dynamic data-dependence analysis in SAPFOR // CEUR Workshop Proceedings. Vol. 2543. 2020. 199–208. http://ceur-ws.org/Vol-2543/
15. NAS Parallel Benchmarks. https://www.nas.nasa.gov/publications/npb.html (Дата обращения: 15 октября 2021).
16. *Wolfe M.* High performance compilers for parallel computing. New York: Addison-Wesley, 1995.
17. *Bondhugula U., Hartono A., Ramanujam J., Sadayappan P.* A practical automatic polyhedral parallelizer and locality optimizer // SIGPLAN Notices. 2008. **43**, N 6. 101–113. doi 10.1145/1379022.1375595.
18. *Verdoolaege S., Juega J. C., Cohen A., Gomez J. I., Tenllado C., Catthoor F.* Polyhedral parallel code generation for CUDA // ACM Trans. Archit. Code Optim. 2013. **9**, N 4. 1–23. doi 10.1145/2400682.2400713.
19. *Grosser T., Groesslinger A., Lengauer C.* Polly — performing polyhedral optimizations on a low-level intermediate representation // Parallel Processing Letters. 2012. **22**, N 04. doi 10.1142/S0129626412500107.
20. *Grosser T., Hoefler T.* Polly-ACC transparent compilation to heterogeneous hardware // Proc. 2016 Int. Conf. on Supercomputing. Istambul, Turkey, June 1–3, 2016. New York: ACM Press, 2016. doi 10.1145/2925426.2926286.
21. *Caamano J.M.M., Sukumaran-Rajam A., Baloian A., Selva M., Clauss P.* APOLLO: Automatic speculative POLyhedral Loop Optimizer // Proc. 7th Int. Workshop on Polyhedral Compilation Techniques (IMPACT 2017), January 23, 2017, Stockholm, Sweden. https://www.researchgate.net/publication/313059456_APOLLO_Automatic_speculative_POLyhedral_Loop_Optimizer
22. *Lattner C., Adve V.* LLVM: a compilation framework for lifelong program analysis & transformation // Proc. 2004 Int. Symp. on Code Generation and Optimization (CGO'04). San Jose (Palo Alto), USA, March 20–24, 2004. doi 10.1109/CGO.2004.1281665, https://llvm.org/pubs/2004-01-30-CGO-LLVM.pdf
23. *Doerfert J., Streit K., Hack S., Benaissa Z.* Polly's polyhedral scheduling in the presence of reductions // Proc. 5th Int. Workshop on Polyhedral Compilation Techniques (IMPACT 2015), January 19, 2015, Amsterdam, The Netherlands. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1054.1804&rep=rep1&type=pdf
24. *Бахтин В.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н.* Методы динамической настройки DVMH-программ на кластеры с ускорителями // Труды Международной конференции "Суперкомпьютерные дни в России 2015", 28–29 сентября 2015, Москва. М.: Изд-во Моск. гос. ун-та, 2015. 257–269.

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

25. *Бахтин В.А., Захаров Д.А., Крюков В.А., Колганов А.С., Поддерюгина Н.В., Притула М.Н., Смирнов А.А.* Дополнительное распараллеливание MPI-программ с помощью DVM-системы // Научный сервис в сети Интернет: труды XXII Всероссийской научной конференции (21–25 сентября 2020 г., Москва), М.: ИПМ им. М.В.Келдыша, 2020, С. 80–100. https://doi.org/10.20948/abrau-2020-29
26. Гетерогенный кластер К60. https://www.kiam.ru/MVS/resourses/k60.html. Дата обращения: 15 октября 2021.

Поступила в редакцию
15 сентября 2021

Принята к публикации
4 октября 2021

Информация об авторах

Никита Андреевич Катаев — научн. сотр., Институт прикладной математики им. М. В. Келдыша РАН (ИПМ РАН), Миусская пл., д. 4, 125047, Москва, Российская Федерация.

Александр Сергеевич Колганов — к.ф.-м.н., научн. сотр., Институт прикладной математики им. М. В. Келдыша РАН (ИПМ РАН), Миусская пл., д. 4, 125047, Москва, Российская Федерация.

References

1. J. Ragan-Kelley, C. Barnes, A. Adams, et al., "Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines," SIGPLAN Not. **48** (6), 519–530 (2013). doi 10.1145/2499370.2462176
2. U. Beaugnon, A. Kravets, S. van Haastregt, et al., "Vobla: A Vehicle for Optimized Basic Linear Algebra," SIGPLAN Not. **49** (5), 115–124 (2014). doi 10.1145/2666357.2597818
3. Y. Zhang, M. Yang, R. Baghdadi, et al., "Graphit: A High-Performance Graph DSL," Proc. of the ACM on Program. Lang. **2**, (OOPSLA) (2018). doi 10.1145/3276491
4. P. An, A. Jula, S. Rus, et al., "STAPL: An Adaptive, Generic Parallel C++ library," in Lecture Notes in Computer Science (Springer, Heidelberg, 2003), Vol. 2624, pp. 193–208. https://doi.org/10.1007/3-540-35767-X_13. Cited October 15, 2021.
5. N. Bell and J. Hoberock, "Thrust: A Productivity-Oriented Library for CUDA," GPU Computing Gems Jade Edition (2012), pp. 359–371. https://doi.org/10.1016/B978-0-12-385963-1.00026-5. Cited October 15, 2021.
6. OpenMP Compilers & Tools. https://www.openmp.org/resources/openmp-compilers-tools/. Cited October 15, 2021.
7. N. A. Konovalov, V. A. Krukov, S. N. Mikhajlov, and A. A. Pogrebtsov, "Fortan DVM: A Language for Portable Parallel Program Development," Program. Comput. Softw. **21** (1), 35–38 (1995).
8. V. A. Bakhtin, M. S. Klinov, V. A. Krukov, et al., "Extension of the DVM-Model of Parallel Programming for Clusters with Heterogeneous Nodes," Vestn. Yuzhn. Ural. Gos. Univ. Ser. Mat. Model. Programm. No. 12, 82–92 (2012).
9. M. S. Klinov and V. A. Kryukov, "Automatic Parallelization of Fortran Programs. Mapping to Cluster," Vestn. Lobachevskii Univ. Nizhni Novgorod, No. 2, 128–134 (2009).
10. V. A. Bakhtin, I. G. Borodich, N. A. Kataev, et al., "Dialogue with a Programmer in the Automatic Parallelization Environment SAPFOR," Vestn. Lobachevskii Univ. Nizhni Novgorod, No. 5(2), 242–245 (2012).
11. N. Kataev, "LLVM Based Parallelization of C Programs for GPU," in *Communications in Computer and Information Science* (Springer, Cham, 2020), Vol. 1331, pp. 436–448. https://doi.org/10.1007/978-3-030-64616-5_38. Cited October 15, 2021.
12. N. Kataev, "Interactive Parallelization of C Programs in SAPFOR," CEUR Workshop Proc., Vol. 2784 (2020), pp. 139–148. http://ceur-ws.org/Vol-2784/. Cited October 15, 2021.
13. N. Kataev, "Application of the LLVM Compiler Infrastructure to the Program Analysis in SAPFOR," in *Communications in Computer and Information Science* (Springer, Cham, 2019), Vol. 965, pp. 487–499. doi 10.1007/978-3-030-05807-4_41.
14. N. Kataev, A. Smirnov, and A. Zhukov, "Dynamic Data-Dependence Analysis in SAPFOR," CEUR Workshop Proc. Vol. 2543 (2020), pp. 199–208. http://ceur-ws.org/Vol-2543/. Cited October 15, 2021.
15. NAS Parallel Benchmarks. https://www.nas.nasa.gov/publications/npb.html. Cited October 15, 2021.
16. M. Wolfe, *High Performance Compilers for Parallel Computing* (Addison-Wesley, New York, 1995).
17. U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A Practical Automatic Polyhedral Parallelizer and Locality Optimizer," SIGPLAN Not. **43** (6), 101–113 (2008). doi 10.1145/1379022.1375595

2021, **22** (4), 239–251. doi 10.26089/NumMet.v22r415

18. S. Verdoolaege, J. C. Juega, A. Cohen, et al., "Polyhedral Parallel Code Generation for CUDA," ACM Trans. Archit. Code Optim. **9** (4), 1–23 (2013). doi 10.1145/2400682.2400713.
19. T. Grosser, A. Groesslinger, and C. Lengauer, "Polly — Performing Polyhedral Optimizations on a Low-Level Intermediate Representation," Parallel Process. Lett. **22** (2012). doi 10.1142/S0129626412500107
20. T. Grosser and T. Hoefler, "Polly-ACC Transparent Compilation to Heterogeneous Hardware," in *Proc. Int. Conf. on Supercomputing, Istambul, Turkey, June 1–3, 2016* (ACM Press, New York, 2016), doi 10.1145/2925426.2926286.
21. J. M. Caamano, A. Sukumaran-Rajam, A. Baloian, et al., "APOLLO: Automatic Speculative POLyhedral Loop Optimizer," in *Proc. 7th Int. Workshop on Polyhedral Compilation Techniques (IMPACT 2017), Stockholm, Sweden, January 23, 2017*, https://www.researchgate.net/publication/313059456_APOLLO_Automatic_speculative_POLyhedral_Loop_Optimizer. Cited October 15, 2021.
22. C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *Proc. Int. Symp. on Code Generation and Optimization San Jose (Palo Alto), USA, March 20–24, 2004*, doi 10.1109/CGO.2004.1281665, https://llvm.org/pubs/2004-01-30-CGO-LLVM.pdf. Cited October 15, 2021.
23. J. Doerfert, K. Streit, S. Hack, and Z. Benaissa, "Polly's Polyhedral Scheduling in the Presence of Reductions," in *Proc. 5th Int. Workshop on Polyhedral Compilation Techniques (IMPACT 2015), Amsterdam, The Netherlands, January 19, 2015*, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1054.1804&rep=rep1&type=pdf. Cited October 15, 2021.
24. V. Bakhtin, A. Kolganov, V. Krukov, et al., "Dynamic Tuning Methods of DVMH-Programs for Clusters with Accelerators," in *Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia, September 28–29, 2015* (Mosk. Gos. Univ., Moscow, 2015), pp. 257–269.
25. V. A. Bakhtin, D. A. Zakharov, V. A. Krukov, et al., "Additional parallelization of MPI programs using DVM-system," in *Scientific service & Internet: proceedings of the 22nd All-Russian Scientific Conference, Moscow, Russia, September 21–25, 2020,* (Keldysh Institute of Applied Mathematics, Moscow, 2020), pp. 80–100. doi 10.20948/abrau-2020-29.
26. Heterogeneous cluster K60. https://www.kiam.ru/MVS/resourses/k60.html. Cited October 15, 2021.

Received
September 15, 2021

Accepted for publication
October 4, 2021

Information about the authors

Nikita A. Kataev — Researcher, Keldysh Institute of Applied Mathematics, Miusskaya ploshchad' 4, 125047, Moscow, Russia.

Alexander S. Kolganov — Ph.D., Researcher, Keldysh Institute of Applied Mathematics, Miusskaya ploshchad' 4, 125047, Moscow, Russia.