2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

doi 10.26089/NumMet.v23r418 УДК 04.272, 004.4'242

Преобразование последовательных Fortran-программ для их распараллеливания на гибридные кластеры в системе SAPFOR

А. С. Колганов
Институт прикладной математики имени М. В. Келдыша РАН (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0002-1384-7484, e-mail: alexander.k.s@mail.ru

Г. Д. Гусев
Московский государственный университет имени М. В. Ломоносова,
Москва, Российская Федерация
ORCID: 0000-0002-3133-0649, e-mail: gr@medhim.ru

Аннотация: Процесс распараллеливания программ может быть затруднен ввиду их оптимизации под последовательное выполнение. Из-за этого полученная параллельная версия может быть неэффективной, а в некоторых случаях распараллеливание оказывается невозможным. Решить указанные проблемы помогают преобразования исходного кода программ. В данной статье рассматривается реализация в системе автоматизированного распараллеливания SAPFOR (System FOR Automated Parallelization) преобразований последовательных Fortran-программ, позволяющих облегчить работу пользователя в системе и существенно снизить трудоемкость распараллеливания программ. Применение реализованных преобразований в системе SAPFOR продемонстрировано на прикладной программе, решающей систему нелинейных дифференциальных уравнений в частных производных. Также было произведено сравнение производительности полученной параллельной версии с версиями, распараллеленными вручную с использованием DVM и MPI технологий.

Ключевые слова: SAPFOR (System FOR Automated Parallelization), автоматизация распараллеливания на кластер, автоматизация преобразований, параллельные вычисления, DVM (Distributed Virtual Memory), кластеры с графическими процессорами.

Для цитирования: Колганов А.С., Гусев Г.Д. Преобразование последовательных Fortran-программ для их распараллеливания на гибридные кластеры в системе SAPFOR // Вычислительные методы и программирование. 2022. **23**, № 4. 288–310. doi 10.26089/NumMet.v23r418.

Transformation of sequential Fortran programs for their parallelization into hybrid clusters in the SAPFOR

Alexander S. Kolganov
Keldysh Institute of Applied Mathematics, Moscow, Russia
ORCID: 0000-0002-1384-7484, e-mail: alexander.k.s@mail.ru

Grigorii D. Gusev
Lomonosov Moscow State University, Moscow, Russia
ORCID: 0000-0002-3133-0649, e-mail: gr@medhim.ru

Abstract: The process of parallelizing programs can be difficult due to their optimization for sequential execution. Because of this, the resulting parallel version may be inefficient, and in

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

some cases parallelization is not possible. Transformations of the source code of programs help to solve these problems. This article discusses the implementation of transformations of sequential Fortran programs in the SAPFOR (System FOR Automated Parallelization) system, which make it possible to facilitate the user's work in the system and significantly reduce the complexity of program parallelization. The application of the implemented transformations in the SAPFOR system is demonstrated on a program that solves a system of non-linear partial differential equations. The performance of the obtained parallel version was also compared with the versions parallelized manually using DVM and MPI technologies.

Keywords: SAPFOR (System FOR Automated Parallelization), parallelization automation for clusters, transformation automation, parallel computing, DVM (Distributed Virtual Memory), GPU clusters.

For citation: A. S. Kolganov, G. D. Gusev, "Transformation of sequential Fortran programs for their parallelization into hybrid clusters in the SAPFOR," Numerical Methods and Programming. **23** (4), 288–310 (2022). doi 10.26089/NumMet.v23r418.

1. Введение. С прекращением роста частоты отдельного процессора вследствие технических и физических ограничений дальнейшее повышение производительности вычислительных систем стало осуществляться через построение многоядерных и многопроцессорных архитектур. Однако написание параллельных программ для подобных систем существенно отличается от написания последовательных программ в силу иного порядка их выполнения. Более того, написание параллельных программ требует от программиста знаний об особенностях аппаратуры системы, на которой программа будет выполняться. Ситуация осложняется появлением новых параллельных архитектур. Для достижения параллелизма на всех уровнях и максимальной эффективности использования вычислительной системы программисту требуется знание различных технологий параллельного программирования (OpenMP [1], MPI [2], CUDA [3], OpenCL [4] и др.), в особенности если вычислительная система является гибридным кластером.

Кластером называется вычислительная система, состоящая из множества вычислительных узлов (каждый из которых может быть многопроцессорным), соединенных высокоскоростной сетью. Гибридный кластер отличается тем, что помимо центрального процессора (central processing unit, CPU) в узле кластера используется ускоритель в виде графического сопроцессора (graphics processing unit, GPU) или многоядерного высокопроизводительного сопроцессора. Примерами таких ускорителей являются графические сопроцессоры фирмы NVIDIA и многоядерные сопроцессоры Intel Xeon Phi.

Особое место среди технологий параллельного программирования занимают модели, основанные на добавлении в программы на последовательных языках (Fortran, C) спецификаций параллелизма, осуществляющих отображение вычислений программ на узлы вычислительной системы. Указанные спецификации оформляются в виде специальных комментариев в Fortran-программах или директив `#pragma` в программах на языке C и не видны для обычных компиляторов. Подобная высокоуровневая модель позволяет иметь одну версию программы как для последовательного, так и для параллельного выполнения, а также упрощает переносимость программ с одной архитектуры на другую. Более того, подобные модели позволяют распараллелить имеющуюся последовательную программу без существенного изменения ее кода.

Однако процесс распараллеливания программы, уже непростой и требующий от программиста знания соответствующей технологии параллельного программирования, может быть существенно затруднен из-за оптимизации программы под последовательное выполнение. В некоторых случаях распараллелить программу без преобразования ее исходного кода не представляется возможным. В совокупности данные факторы делают задачу распараллеливания последовательной программы нетривиальной. Облегчить ситуацию призваны системы автоматизированного распараллеливания программ.

2. Обзор существующих систем. Можно выделить два основных подхода, применяемых при разработке инструментов, автоматизирующих распараллеливание программ. Подход, наиболее понятный для пользователя и поэтому используемый в системе SAPFOR в силу ее направленности на поддержание процесса интерактивного распараллеливания, заключается в последовательном изменении кода программы,

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

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. [5]). 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, например модели многогранников или полиэдральной модели, описывающей поведение распараллеливаемой программы. Этот подход позволяет сформировать 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 должен применяться для создания автоматически распараллеливающих компиляторов [6–10], а не инструментов автоматизации.

Компиляторы Pluto [6] и PPCG [7] направлены на преобразование программ только на языке C, 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 [8] и его дальнейшее расширение для отображения программ на графические ускорители Polly-ACC [9]. Данные инструменты используют LLVM [11] для анализа и преобразования программ и доступны в составе компилятора Clang. Использование низкоуровневого представления (LLVM IR) лишает пользователя возможности как-либо изменять код параллельной программы, но при этом расширяет 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 [12]. 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 [10], осуществляющий спекулятивное распараллеливание программы во время выполнения. Но в данном инструменте реализована возможность распараллеливания только для многоядерных процессоров.

Примером высокоуровневой технологии параллельного программирования является DVM-система [13], созданная в ИПМ имени М.В. Келдыша РАН при активном участии аспирантов и студентов факультета ВМК МГУ имени М.В. Ломоносова. Она предназначена для разработки параллельных программ научно-технических расчетов. Модель DVMH является расширением DVM-системы для гибридных кластеров. В модели используются высокоуровневые директивные языки Fortran-DVMH и C-DVMH [14], которые представляют собой расширение стандартных языков Fortran-95 и C-99 спецификациями параллелизма в виде специальных комментариев и директив `#pragma` соответственно. Далее в данной статье рассматривается только часть DVM-системы, относящаяся к созданию параллельных программ на языке

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Листинг 1. Общий вид директивы в языке Fortran-DVMH
Listing 1. General form of the directive in the Fortran-DVMH language

```
!DVM$ <DVMH-directive>
```

Fortran-DVMH. Общий вид специальных комментариев в языке Fortran-DVMH, называемых директивами, представлен в листинге 1.

В модели DVMH вычислительная система представляется в виде массива виртуальных процессоров (в общем случае — многомерного), который задает пользователь при запуске программы на выполнение. Данный массив также называется решеткой процессоров. С помощью директивы `!DVM$ DISTRIBUTE` пользователь задает отображение массивов, использующихся в программе, на множество виртуальных процессоров. При этом каждое измерение массива либо распределяется на одно из измерений решетки процессоров, либо распределяется на каждый виртуальный процессор целиком (размножается). Если измерение массива распределяется на одно из измерений решетки, то размер данного измерения решетки определяет количество процессов, между которыми будет распределено данное измерение массива. Произведение размеров измерений решетки процессоров определяет общее количество процессов, на которых будет выполняться программа.

Через директиву `!DVM$ ALIGN` задается отображение (выравнивание) элементов одного массива на элементы другого массива. Соответствующие друг другу элементы массивов будут распределены на один виртуальный процессор. Также с помощью данной директивы можно выравнять массив по DVMH-шаблону. По одному шаблону может быть выравнено несколько массивов. DVMH-шаблон задает индексное пространство, которое определяет отображение элементов выровненных по данному шаблону массивов на множество виртуальных процессоров и, в отличие от массивов, не занимает места в памяти.

Параллельный цикл в модели DVMH рассматривается как массив витков цикла. Количество измерений массива определяется степенью тесно-вложенности данного цикла (тесно-вложенные циклы в языке Fortran описаны в разделе 3). С помощью директивы `!DVM$ PARALLEL` задается отображение витков цикла на элементы некоторого массива. Каждый виток цикла будет выполнен на том виртуальном процессоре, на который был распределен соответствующий данному витку элемент массива [14].

Вычислительным регионом (или просто регионом) в модели DVMH называется часть программы с одним входом и одним выходом, которая может быть выполнена на одном или нескольких вычислительных устройствах. В частности, регион может быть выполнен на графическом ускорителе. В листинге 2 приведен общий вид региона, который состоит из блока операторов, заключенных в пару директив. В списке спецификаций `<specs>` указываются входные, выходные и локальные данные для региона.

Листинг 2. Общий вид региона в модели DVMH
Listing 2. General form of the parallel region in DVMH model

```
!DVM$ REGION[<specs>]
!         Block of statements
!DVM$ END REGION
```

Важным дополнением к DVM-системе является система автоматизированного распараллеливания Fortran- и C-программ SAPFOR [15]. Она анализирует исходный код последовательной программы и на основе полученных результатов осуществляет генерацию параллельной версии программы в модели DVMH. Данная система позволяет разрабатывать параллельные программы и распараллеливать последовательные с учетом особенностей архитектур, на которых программа будет выполняться.

Автоматизированное распараллеливание программ основывается на возможности выявления данных, которые могут быть распределены между процессами, и циклов, витки которых могут быть выполнены параллельно. Для этого требуется точный анализ исходного программного кода программы с выявлением зависимостей по данным и управлению в нем. Однако зачастую такого анализа оказывается недостаточно для эффективного распараллеливания и требуются дополнительные средства. Например, спекулятивное выполнение участков программы. Другим методом, позволяющим существенно повысить

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

эффективность распараллеливания, а также расширить множество программ, которые могут быть распараллелены, является преобразование исходного кода программы.

Работа системы SAPFOR начинается с обработки текстовых файлов с кодом входной программы на языке Fortran. При этом строится внутреннее представление программы в виде абстрактного синтаксического дерева. Модуль системы, осуществляющий данный разбор исходного кода входной программы, называется парсером. Процесс распараллеливания программы через систему является итеративным и состоит из этапов анализа и преобразований. Выполнив тот или иной анализ, пользователь получает набор диагностик и сообщений системы, на основе которых предпринимает дальнейшие шаги по получению параллельной версии программы. Например, пользователь может уточнить результаты анализа, если система недостаточно эффективно с ним справилась. Этап преобразований заключается в трансформации исходного кода программы или построении параллельной версии. Если некоторый анализ или преобразование выполнить не удается, система выдает сообщения о проблемах с привязкой к строкам исходной программы.

В зависимости от полученных результатов пользователь может изменить набор и порядок анализов и преобразований и получить другую версию параллельной программы. Следует отметить, что оптимальной последовательности анализов и преобразований участков кода зачастую не существует даже в рамках одной программы (данный аспект рассмотрен в статье [16]). К одному участку кода могут быть применимы несколько преобразований, дающих разный эффект. Система SAPFOR является инструментом, призванным помочь пользователю в непростой задаче распараллеливания последовательных программ.

Этапы анализа и преобразований в системе SAPFOR представляют собой наборы проходов. Проход — это функциональный модуль, решающий отдельную простую задачу. В общем случае работа прохода состоит из двух фаз, на первой из которых каждый файл исходной программы единообразно обрабатывается, а на второй происходит объединение собранных данных. Наличие всех фаз для прохода не является обязательным.

Проходы могут иметь зависимости и связи между собой. Они возникают из-за того, что данные одного из проходов анализа необходимы для работы другого прохода. Например, для прохода “объединение циклов” требуется граф циклов программы, который строится соответствующим проходом. Для указания зависимостей между проходами используется менеджер проходов. Менеджер проходов является структурой, в которой хранятся прямые зависимости между теми или иными проходами. Косвенные зависимости выводятся из прямых автоматически. При запуске пользователем некоторого прохода система SAPFOR выполняет все проходы, от которых он зависит, после чего будет выполнен вызванный пользователем проход [17].

3. Используемые понятия. Циклом в языке Fortran называется повторное выполнение блока операторов и конструкций. Конструкции состоят из нескольких операторов и используются для выполнения управляющих действий, таких как ветвления или циклы. Блок операторов и конструкций цикла называется телом цикла. Однократное выполнение тела цикла называется витком цикла или итерацией. В языке Fortran есть три вида циклов: с параметром, с предусловием и с постусловием [18].

Распараллеливанием цикла является распределение множества его витков между узлами вычислительной системы. Поскольку для такого распределения необходимо точно знать число витков цикла, распараллелен может быть только цикл с параметром.

В листинге 3 приведен пример цикла с параметром. Целочисленная переменная `I` называется итерационной переменной цикла или параметром в языке Fortran. Целочисленные выражения `A` и `B` задают начальное и конечное значения параметра. Будем называть их границами цикла. Необязательное целочисленное значение `C` задает шаг цикла, который по умолчанию равен единице. Отметим, что шаг цикла может быть отрицательным. Совокупность значений `A`, `B` и `C` назовем заголовком цикла.

Листинг 3. Общий вид цикла с параметром в языке Fortran
Listing 3. General form of the DO loop in Fortran

```
1       DO I = A, B, C
2 !        Block of statements
3       ENDDO
```

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Тесно-вложенным называется цикл, состоящий из нескольких последовательно вложенных друг в друга циклов так, что тело каждого цикла (кроме внутреннего) состоит из одного оператора — вложенного в него другого цикла. Тесно-вложенный цикл иначе называется гнездом циклов. Внешний цикл гнезда назовем находящимся на первом уровне вложенности. Вложенный в него цикл — находящимся на втором уровне вложенности, и т.д. Уровень внутреннего цикла определяет максимальный уровень тесной вложенности всего гнезда, иначе называемой степенью тесно-вложенности цикла. Далее в статье рассматриваются только циклы с параметром и не проводится различий между циклом и гнездом циклов, если не указано обратное.

Переменные, использующиеся в теле цикла, который потенциально может быть распараллелен, делятся на приватные и неприватные для данного цикла. Приватной называется переменная, которая на каждом витке цикла получает некоторое значение, и это значение используется только на данном витке цикла. Зависимостью по данным называется ситуация, при которой инструкция (оператор) в программе записывает данные в то же место в памяти, в которое записывает или из которого считывает данные другая инструкция [19]. Пусть возможен путь выполнения программы, при котором инструкция S_1 выполняется до инструкции S_2. Тогда между инструкциями S_1 и S_2 возможна зависимость по данным одного из следующих типов:

- Flow (read after write), если инструкция S_2 считывает данные из того места, куда произвела запись инструкция S_1;
- Anti (write after read), если инструкция S_2 производит запись в то место, откуда считывала данные инструкция S_1;
- Output (write after write), если обе инструкции пишут данные в одно место в памяти.

Отдельно отметим, что если обе инструкции S_1 и S_2 считывают данные из одного места, то зависимости по данным между инструкциями не возникает.

Определением переменной a называется инструкция, которая присваивает или может присваивать значение данной переменной. Путем выполнения (или просто путем) называют последовательность инструкций в программе, которая может быть выполнена при некотором сценарии выполнения программы. Говорят, что определение d переменной a достигает некоторой точки p в программе, если существует путь s от точки, следующей за d, к точке p, вдоль которого определение d не уничтожается. Определение d будет уничтожено на пути s, если на s существует иное определение d' переменной a. В этом случае говорят, что определение d' уничтожает определение d [19].

4. Директивы системы SAPFOR. Для указания системе SAPFOR информации об анализе программы или о необходимости выполнения преобразований пользователь может использовать директивы, которые оформляются в программе в виде специальных комментариев. Общий вид директивы системы SAPFOR представлен в листинге 4. В данном примере `<SPF-directive>` — тип директивы (один из `ANALYSIS`, `TRANSFORM`, `CHECKPOINT`, `PARALLEL`, `PARALLEL_REG`, `END PARALLEL_REG`), а `<specs>` — список спецификаций, разделенных запятыми.

Листинг 4. Общий вид директивы системы SAPFOR
Listing 4. General form of the SAPFOR's directives

```
1 !$SPF <SPF-directive>(<specs>)
```

Директива типа `ANALYSIS` сообщает системе дополнительную информацию о том участке кода программы, в котором она расположена, и используется в сочетании со спецификациями `PRIVATE(<privates>)`, `REDUCTION(<red list>)` и `PARAMETER(<par list>)`. Директива со спецификацией `PRIVATE` может быть расположена либо перед объявлением переменной, либо перед циклом. В списке `<privates>` через запятую указываются переменные. Если директива со спецификацией `PRIVATE` расположена перед циклом, то в списке `<privates>` указываются приватные переменные данного цикла. Директива со спецификацией `REDUCTION` ставится только перед циклами, в списке `<red list>` указываются переменные, по которым в цикле имеется редукция. Также в списке `<red list>` задаются операции, реализующие редукцию. Спецификация `PARAMETER` ставится перед исполняемыми операторами, в списке `<par list>` перечисляются переменные-параметры программы и их значения.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Директива типа TRANSFORM сообщает системе информацию о преобразовании кода программы в том месте, где данная директива расположена, и используется совместно со спецификациями NOINLINE, FISSION(<iters>), EXPAND(<iters>) и SHRINK(<masks>). Директива со спецификацией NOINLINE ставится после заголовка процедуры и указывает системе, что не следует выполнять подстановку данной процедуры. Директивы со спецификациями FISSION, EXPAND и SHRINK ставятся только перед циклами. Спецификация FISSION указывает, что необходимо выполнить разделение данного цикла, в списке <iters> перечисляются некоторые из итерационных переменных цикла. Спецификация EXPAND сообщает системе о необходимости выполнить расширение приватных переменных данного цикла по уровням цикла, соответствующим итерационным переменным из списка <iters>. Спецификация SHRINK в списке <masks> указывает информацию, в соответствии с которой системе необходимо выполнить сужение приватных переменных цикла.

В листинге 5 приводится пример использования директив системы SAPFOR. В строке 1 находится директива ANALYSIS со спецификацией PRIVATE, которая сообщает системе, что переменная A является приватной для тесно-вложенного цикла в строке 3. Директива TRANSFORM в строке 2 указывает системе, что необходимо выполнить преобразование “расширение приватных переменных” по уровням цикла, соответствующим итерационным переменным I и J.

Листинг 5. Пример использования директив системы SAPFOR
Listing 5. Example of using the SAPFOR's directives

```
1 !$SPF ANALYSIS(PRIVATE(A))
2 !$SPF TRANSFORM(EXPAND(I, J))
3       DO I = 1, N1
4          DO J = 1, N2
5             A = I + J
6             B(J, I) = 1 / A
7          ENDDO
8       ENDDO
```

5. Преобразования последовательных программ. В данном разделе описываются преобразования последовательных Fortran-программ. Главное требование, которому должны удовлетворять все преобразования — корректность, т.е. исходная и преобразованная программы должны быть эквивалентны. Эквивалентными будем называть программы, которые имеют равные выходные данные при одинаковых входных данных. Все преобразования рассматриваются для потенциально параллельных циклов. Потенциально параллельным называется такой цикл с параметром (возможно, тесно-вложенный), который может быть автоматически преобразован в параллельный цикл системой SAPFOR с помощью директивы распределения вычислений !$DVM PARALLEL[17]. Такой цикл должен обладать следующими дополнительными свойствами:

- обладать ненулевым количеством витков;
- иметь прямоугольное индексное пространство (это означает, что границы всех уровней гнезда циклов могут быть вычислены до начала выполнения цикла);
- не содержать операторов перехода GOTO внутрь цикла и за его пределы;
- не содержать операторов ввода/вывода;
- не содержать операторов останова;
- измерения используемых в цикле массивов индексируются только регулярными выражениями вида $a * I + b$, где I — итерационная переменная цикла, a и b — целочисленные выражения;
- между витками цикла могут быть только зависимости по приватным или редукционным переменным, а также регулярные зависимости по распределенному массиву;
- не содержать вызовов процедур, внутри которых находятся другие потенциально параллельные циклы.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Отметим, что потенциально параллельный цикл может содержать вызовы процедур в своем теле. Однако, для того чтобы удовлетворять описанным выше свойствам, данная процедура не должна содержать операторов ввода/вывода и операторов останова. Для проверки соблюдения данного условия выполняется межпроцедурный анализ.

5.1. Объединение циклов. Объединением двух подряд идущих в программе циклов является их замещение одним циклом с соответствующим переносом и преобразованием операторов из тел объединяемых циклов. В данной статье рассматривается только объединение потенциально параллельных циклов с равными границами и равным с точностью до знака шагом. Отметим, что в общем случае могут быть объединены циклы с разными, не совпадающими границами и шагами.

Объединение циклов в ряде случаев позволяет преобразовать некоторое множество циклов к более подходящему для распараллеливания виду. Например, если тело цикла состоит из множества вложенных циклов, то их объединение приведет внешний цикл к тесно-вложенному виду, распараллеливание которого может быть более эффективным по сравнению с исходным циклом.

Обозначим через $Vars_1$ и $Vars_2$ множества всех переменных, использующихся в первом и втором объединяемых циклах соответственно. Через $Privates_1$ и $Privates_2$ обозначим множества приватных переменных циклов, которые являются подмножествами множеств всех переменных соответствующих циклов: $Privates_1 \subseteq Vars_1$, $Privates_2 \subseteq Vars_2$. Отметим, что итерационные переменные цикла являются приватными для него.

Рассмотрим возможные варианты объединения двух циклов. В простейшем случае заголовки объединяемых циклов совпадают и между циклами нет зависимости по данным. Последнее обеспечивается условием, что циклы не имеют общих переменных: $Vars_1 \cap Vars_2 = \varnothing$. Пример таких циклов показан в листинге 6.

Листинг 6. Объединение циклов. Пример № 1
Listing 6. Loops combining. Example No 1

```
1       DO I1 = 1, N1
2          DO I2 = 1, N2
3             A(I2, I1) = I1 + I2
4          ENDDO
5       ENDDO
6       DO J1 = 1, N1
7          DO J2 = 1, N2
8             B(J2, J1) = J1 * J2
9          ENDDO
10      ENDDO
```

В данном случае объединение возможно, поскольку отсутствие зависимостей по данным позволяет выполнять такие циклы в произвольном порядке. Для объединения достаточно перенести тело второго цикла внутрь первого, применив единственное необходимое преобразование — переименование итерационных переменных в операторах тела второго цикла на соответствующие им итерационные переменные первого цикла. Результат объединения циклов из листинга 6 показан в листинге 7.

Листинг 7. Результат объединения циклов из листинга 6
Listing 7. The result of loops combining from listing 6

```
1       DO I1 = 1, N1
2          DO I2 = 1, N2
3             A(I2, I1) = I1 + I2
4             B(I2, I1) = I1 * I2
5          ENDDO
6       ENDDO
```

5.2. Объединение циклов с зависимостью по данным. Рассмотрим случай, когда между объединяемыми циклами есть зависимость по данным. Это означает, что циклы используют общие пере-

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

менные: $Vars_1 \cap Vars_2 \neq \varnothing$. Если общие для двух циклов переменные являются приватными для них ($Vars_1 \cap Vars_2 \subseteq Privates_1 \cap Privates_2$), то такая зависимость по данным не препятствует объединению. В примере, показанном в листинге 8, общей для объединяемых циклов является переменная `A`, которая является приватной для обоих циклов.

Листинг 8. Объединение циклов. Пример № 2
Listing 8. Loops combining. Example No 2

```
1       DO I = 1, N
2          A = 1 / I
3          B(I) = A
4       ENDDO
5       DO J = 1, N
6          A = J / 2
7          C(J) = A
8       ENDDO
```

В данном случае объединение возможно и будет корректным. Результат объединения показан в листинге 9.

Листинг 9. Результат объединения циклов из листинга 8
Listing 9. The result of loops combining from listing 8

```
1       DO I = 1, N
2          A = 1 / I
3          B(I) = A
4          A = I / 2
5          C(I) = A
6       ENDDO
```

Если общая для объединяемых циклов переменная является неприватной для каждого из них ($A \in Vars_1 \cap Vars_2,\ A \notin Privates_1,\ A \notin Privates_2$), то объединение возможно только при зависимости типа Output по данной переменной между телами циклов. Зависимости типов Flow и Anti делают корректное объединение циклов невозможным. В примере, показанном в листинге 10, между циклами есть зависимость типа Flow по общей неприватной переменной `A`, объединение невозможно.

Листинг 10. Объединение циклов. Пример № 3
Listing 10. Loops combining. Example No 3

```
1       DO I = 1, N
2          A = A + I
3       ENDDO
4       DO J = 1, N
5          A = A / J
6       ENDDO
```

В случае, когда общей для двух циклов является переменная, приватная только для одного из циклов ($A \in Vars_1 \cap Vars_2,\ A \in Privates_1 \cup Privates_2,\ A \notin Privates_1 \cap Privates_2$), объединение возможно. Однако требуется переименовать такую переменную (т.е. заменить ее другой, аналогичной по типу) в теле того цикла, для которой она является приватной. В примере, приведенном в листинге 11, переменная `A` является приватной для цикла в строке 6 и неприватной для цикла в строке 2.

В данной ситуации для корректного объединения следует переименовать переменную `A` в теле второго цикла и объявить в программе новую переменную `A1` с типом, аналогичным типу переменной `A`. Результат преобразований показан в листинге 12.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Листинг 11. Объединение циклов. Пример № 4
Listing 11. Loops combining. Example No 4

```
      A = A_INIT
      DO I = 1, N
         A = A / I
         B(I) = A
      ENDDO
      DO J = 1, N
         A = J / 2
         C(J) = A
      ENDDO
```

Листинг 12. Результат объединения циклов из листинга 11
Listing 12. The result of loops combining from listing 11

```
      A = A_INIT
      DO I = 1, N
         A = A / I
         B(I) = A
         A1 = I / 2
         C(I) = A1
      ENDDO
```

5.3. Объединение циклов с “разворотом” одного из них. Рассмотрим ситуацию, когда границы двух циклов совпадают, но они “развернуты” друг относительно друга, т.е. левая граница первого цикла совпадает с правой границей второго и левая граница второго цикла совпадает с правой границей первого, а шаги циклов равны по модулю, но противоположны по знаку. Объединение подобных циклов возможно, если “развернуть” один из циклов, а именно поменять его левую и правую границы местами и домножить шаг на -1. Для корректности данного преобразования необходимо, чтобы витки разворачиваемого цикла не имели зависимостей по данным между собой для неприватных переменных. После разворота циклы будут иметь одинаковые границы и шаги и могут быть объединены в соответствии с описанными выше случаями.

Листинг 13. Объединение циклов. Пример № 5
Listing 13. Loops combining. Example No 5

```
      DO I = 1, N
         A(I) = 1 / I
      ENDDO
      DO J = N, 1, -1
         B(J) = J / 2
      ENDDO
```

Листинг 14. Результат объединения циклов из листинга 13
Listing 14. The result of loops combining from listing 13

```
      DO I = 1, N
         A(I) = 1 / I
         B(I) = I / 2
      ENDDO
```

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

В примере, приведенном в листинге 13, один цикл “развернут” относительно другого. Второй из циклов не имеет зависимостей по данным между витками, поэтому его можно “развернуть”, после чего объединить с первым циклом в соответствии с описанными ранее условиями для “неразвернутых” относительно друг друга циклов. Результат объединения показан в листинге 14.

5.4. Разделение циклов. Разделением цикла является замещение несколькими циклами с совпадающими заголовками, тела которых являются разбиением тела исходного цикла. Под разбиением понимается деление множества операторов тела исходного цикла на непересекающиеся непустые подмножества (здесь и далее под множеством операторов понимается линейно упорядоченное множество, элементами которого являются операторы программы; отношение порядка определяется номерами строк исходной программы, в которых располагаются соответствующие операторы). В общем случае такое разбиение не единственно, однако оно должно удовлетворять основному требованию — корректности. Корректным будем называть такое разбиение, при котором разделение цикла по данному разбиению множества его операторов приводит программу к эквивалентному с исходной программой виду.

Преобразование разделения цикла, как и объединение циклов, в некоторых случаях позволяет привести исходный цикл к более подходящему для распараллеливания виду. Например, пусть тело некоторого цикла состоит из множества вложенных циклов, которые не могут быть объединены. Тогда разделение внешнего цикла приведет к образованию множества тесно-вложенных циклов, которые могут быть распараллелены более эффективно по сравнению с исходным циклом.

Пусть имеется цикл с телом, состоящим из множества операторов S, и пусть есть разбиение множества S на подмножества S_1 и S_2 ($S_1 \cap S_2 = \varnothing$, $S_1 \cup S_2 = S$). Достаточным условием корректности разбиения цикла является отсутствие зависимости по данным между любой парой операторов s_1 и s_2, где $s_1 \in S_1$, $s_2 \in S_2$. Справедливость данного утверждения следует из того, что множества операторов S_1 и S_2 могут быть выполнены независимо, поскольку не используют общих данных (за исключением, возможно, тех данных, к которым указанные множества операторов обращаются только для чтения). Данное условие легко распространяется на случай, когда множество S разбивается на большее количество подмножеств.

Листинг 15. Разделение циклов. Пример № 1
Listing 15. Loops fission. Example No 1

```
1       DO I = 1, N
2          A(I) = 1 / I
3          C = I / 2
4          B(I) = C * C
5       ENDDO
```

Рассмотрим цикл, приведенный в листинге 15. Тело данного цикла состоит из трех операторов, среди которых имеется зависимость по данным типа Flow по скалярной переменной `C` между операторами в строках 3 и 4. Других зависимостей по данным в теле цикла нет. Поэтому разбиение множества операторов исходного цикла {2, 3, 4} на подмножества {2, 3} и {4} является некорректным, а разбиение на подмножества {2} и {3, 4} — корректным. Результат разбиения данного цикла показан в листинге 16.

Листинг 16. Результат разделения цикла из листинга 15
Listing 16. The result of loops fission from listing 15

```
1       DO I = 1, N
2          A(I) = 1 / I
3       ENDDO
4       DO I = 1, N
5          C = I / 2
6          B(I) = C * C
7       ENDDO
```

Однако указанное достаточное условие не является необходимым. В приведенном в листинге 17 примере операторы в строках 3 и 5 читают данные из переменной `C`, запись в которую производится

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

в строке 2. Следовательно, существует зависимость по данным типа Flow между парами операторов в строках 2, 3 и 2, 5.

Листинг 17. Разделение циклов. Пример № 2
Listing 17. Loops fission. Example No 2

```
1       DO I = 1, N
2          C = 1 / I
3          A(I) = C
4          C = I / 2
5          B(I) = C * C
6       ENDDO
```

Однако разбиение тела цикла на множества $\{2, 3\}$ и $\{4, 5\}$ является корректным, поскольку оператор в строке 5 считывает данные из переменной `C`, которые были в нее занесены оператором в строке 4. Перекладывая вышеизложенное на термины анализа достигающих определений, в операторе в строке 5 используется определение переменной `C`, порожденное в строке 4. Определение, порожденное в строке 2, уничтожается в строке 4 и не достигает оператора в строке 5. Результат разбиения показан в листинге 18.

Листинг 18. Результат разделения цикла из листинга 17
Listing 18. The result of loops fission from listing 17

```
1       DO I = 1, N
2          C = 1 / I
3          A(I) = C
4       ENDDO
5       DO I = 1, N
6          C = I / 2
7          B(I) = C * C
8       ENDDO
```

Во всех приведенных выше примерах зависимость между операторами тела цикла была по скалярным переменным. Рассмотрим цикл с зависимостью по массиву, приведенный в листинге 19. Несмотря на наличие зависимости типа Flow по массиву `A` между операторами в строках 3 и 6, данный цикл может быть разделен. Преобразование корректно, поскольку массив `A` индексируется по итерационной переменной `I` разделяемого цикла (под индексацией понимается то, что индексом во всех обращениях к массиву `A` в некотором его измерении является итерационная переменная `I`). Благодаря указанному условию каждый элемент массива `A` определяется и используется только на одной итерации цикла по `I`. Это позволяет разделить цикл по операторам, использующим массив `A`, при условии сохранения относительного порядка данных операторов. Результат разделения показан в листинге 20.

Листинг 19. Разделение циклов. Пример № 3
Listing 19. Loops fission. Example No 3

```
1       DO I = 1, N1
2          DO J1 = 1, N2
3             A(J1, I) = I + J1
4          ENDDO
5          DO J2 = 2, N2 - 1
6             B(J2, I) = A(J2 - 1, I) * A(J2 + 1, I)
7          ENDDO
8       ENDDO
```

Исходя из изложенного выше, можно сформулировать условия, при которых разделение цикла корректно. Пусть имеется тесно-вложенный цикл с итерационными переменными $I_1, I_2, \ldots, I_k$ и телом, состо-

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Листинг 20. Результат разделения цикла из листинга 19
Listing 20. The result of loops fission from listing 19

```
      DO I = 1, N1
         DO J1 = 1, N2
            A(J1, I) = I + J1
         ENDDO
      ENDDO
      DO I = 1, N1
         DO J2 = 2, N2 - 1
            B(J2, I) = A(J2 - 1, I) * A(J2 + 1, I)
         ENDDO
      ENDDO
```

ящим из множества операторов S. Пусть имеется разбиение множества S на подмножества $S_1, S_2, \ldots, S_n$. Тогда данное разбиение будет корректным, если выполняются следующие условия:

- ни одно определение из S_i *скалярной* переменной x не достигает ни одного оператора из S_j, в котором переменная x используется на чтение, $\forall i, j = \overline{1,n},\ i \neq j$;
- если между операторами s' и s'' существует зависимость по *массиву* x, то массив x должен индексироваться по итерационным переменным цикла $I_1, I_2, \ldots, I_k$, $s' \in S_i,\ s'' \in S_j,\ \forall i, j = \overline{1,n},\ i \neq j$.

5.5. Расширение приватных переменных. Расширением приватной переменной цикла называется преобразование, при котором к переменной добавляются измерения, соответствующие некоторому количеству внешних уровней гнезда циклов. Соответствие добавленного измерения одному из уровней гнезда, по которым происходит расширение (т.е. одному из циклов гнезда), означает, что размер данного измерения совпадает с количеством итераций цикла, а индексом является итерационная переменная указанного цикла. Отметим, что расширение любой приватной переменной цикла является корректным преобразованием.

Расширение приватной переменной по некоторому уровню из гнезда циклов создает свою "версию" данной переменной для каждого витка уровня. Это позволяет корректно разделить цикл по этому уровню так, что данная переменная будет использоваться в разных образованных циклах. Таким образом, расширение приватной переменной позволяет избавиться от ограничений на разделение цикла из-за зависимости по данным между операторами.

Листинг 21. Расширение приватных переменных. Пример № 1
Listing 21. Extension of private variables. Example No 1

```
      DO I = 1, N1
         DO J1 = 1, N2
            A(J1) = I + J1
         ENDDO
         DO J2 = 2, N2 - 1
            B(J2, I) = A(J2 - 1) * A(J2 + 1)
         ENDDO
      ENDDO
```

Рассмотрим пример, приведенный в листинге 21. На каждой итерации внешнего цикла по `I` массив `A` получает значения в цикле по `J1`, которые потом используются в цикле по `J2`. Массив `A` является приватным и может быть расширен соответственно с циклом по `I`. После преобразования новый массив `A1` будет двумерным, с размером второго измерения равным `N1`. Результат расширения показан в листинге 22.

Расширена может быть и скалярная переменная. В примере в листинге 23 в цикле степени тесновложенности два имеется скалярная приватная переменная `A`. Данная переменная может быть расширена по обоим уровням гнезда. При этом измерение, соответствующее циклу по `I`, будет иметь размер `N1`, а измерение, соответствующее циклу по `J` — `N2`. Результат расширения приведен в листинге 24.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Листинг 22. Результат расширения переменной для цикла из листинга 21
Listing 22. The result of extension of private variables from listing 21

```
1       DO I = 1, N1
2          DO J1 = 1, N2
3             A1(J1, I) = I + J1
4          ENDDO
5          DO J2 = 2, N2 - 1
6             B(J2, I) = A1(J2 - 1, I) * A1(J2 + 1, I)
7          ENDDO
8       ENDDO
```

Листинг 23. Расширение приватных переменных. Пример № 2
Listing 23. Extension of private variables. Example No 2

```
1       DO I = 1, N1
2          DO J = 1, N2
3             A = I + J
4             B(J, I) = 1 / A
5          ENDDO
6       ENDDO
```

Листинг 24. Результат расширения переменной для цикла из листинга 23
Listing 24. The result of extension of private variables from listing 23

```
1       DO I = 1, N1
2          DO J = 1, N2
3             A1(J, I) = I + J
4             B(J, I) = 1 / A1(J, I)
5          ENDDO
6       ENDDO
```

Массивы в языке Fortran хранятся в памяти по столбцам. Из этого следует, что оптимальным порядком обхода многомерного массива в тесно-вложенном цикле будет тот, при котором цикл первого уровня гнезда соответствует крайнему правому измерению массива, цикл второго уровня соответствует второму справа измерению массива, и т.д. Однако в программе возможна ситуация, когда данный порядок не соблюдается.

В цикле, приведенном в листинге 25, трехмерный массив `B` обходится не оптимальным образом. При расширении приватной переменной `A` по циклам `I` и `J` возникает вопрос — в каком порядке следует добавлять измерения. Порядок добавляемых измерений не влияет на корректность преобразования, однако важен при распараллеливании цикла. В модели DVMH при распараллеливании программы многомерные массивы распределяются между узлами вычислительной системы. Если при этом удается выравнять распределяемые данные по некоторому DVMH-шаблону, это позволяет распределить их более эффективно и избежать излишних перераспределений. Выбор порядка добавляемых к переменной измерений потенциально позволяет выравнять образованный массив вместе с обходимыми в цикле массивами по одному DVMH-шаблону, что при распараллеливании оказывается более важным, чем оптимальный порядок обхода введенного массива.

Пусть выполняется расширение переменной C, и пусть в цикле имеется массив D, размерность которого равна сумме размерности расширяемой переменной C (размерность скалярной переменной положим равной нулю) и количества добавляемых к ней измерений, а также который индексируется по итерационным переменным тех уровней гнезда, по которым происходит расширение. Тогда вероятно, что если добавлять измерения к переменной C в соответствии с тем, как индексируемые по тем же переменным измерения расположены в массиве D, то образованный массив C_1 при распараллеливании цикла возможно будет выравнять по одному DVMH-шаблону с массивом D.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Листинг 25. Расширение приватных переменных. Пример № 3
Listing 25. Extension of private variables. Example No 3

```
1       DO I = 1, N1
2          DO J = 1, N2
3             DO K = 1, N3
4                A(K) = I + J + K
5                B(J, K, I) = 1 / A(K)
6             ENDDO
7          ENDDO
8       ENDDO
```

В приведенном в листинге 25 случае в качестве массива, в соответствии с которым можно добавлять измерения при расширении переменной `A` по циклам по `I` и `J`, выступает массив `B`. Размерность массива `B` равна трем, что совпадает с суммой размерности расширяемой переменной (один) и количеством добавляемых измерений (два). Также массив `B` индексируется по итерационным переменным `I` и `J` циклов, по которым происходит расширение. Исходя из этого, измерения к переменной `A` следует добавлять так, чтобы измерение, соответствующее циклу по `J`, было первым для образованного массива `A1`, а измерение, соответствующее циклу по `I`, — третьим. Благодаря этому массивы `B` и `A1` потенциально при распараллеливании могут быть выровнены по одному DVMH-шаблону. Результат расширения приведен в листинге 26.

Листинг 26. Результат расширения переменной для цикла из листинга 25
Listing 26. The result of extension of private variables from listing 25

```
1       DO I = 1, N1
2          DO J = 1, N2
3             DO K = 1, N3
4                A1(J, K, I) = I + J + K
5                B(J, K, I) = 1 / A1(J, K, I)
6             ENDDO
7          ENDDO
8       ENDDO
```

5.6. Сужение приватных переменных. В результате преобразований последовательного кода в программе может возникнуть цикл, в котором будет приватная переменная (массив), имеющая “лишние” измерения. Сужением приватной переменной является преобразование, при котором данная переменная заменяется другой переменной, множество измерений которой является строгим подмножеством измерений исходной переменной, т.е. у сужаемой переменной “удаляется” часть измерений.

Например, после объединения циклов из листинга 27, будет образован цикл, для которого переменная `A` окажется приватной (при условии, что далее в программе значения массива `A`, полученные в данном цикле, не используются). Результат объединения показан в листинге 28.

Листинг 27. Сужение переменных. Пример
Listing 27. Shrinking of private variables. Example

```
1       DO I1 = 1, N1
2          DO I2 = 1, N2
3             A(I1, I2) = I1 + I2
4          ENDDO
5       ENDDO
6       DO J1 = 1, N1
7          DO J2 = 2, N2 - 1
8             B(J1, J2) = A(J1, J2) + J1 * J2
9          ENDDO
10      ENDDO
```

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Листинг 28. Результат объединения циклов из листинга 27
Listing 28. The result of loops combining from listing 27

```
1       DO I1 = 1, N1
2          DO I2 = 1, N2
3             A(I1, I2) = I1 + I2
4          ENDDO
5          DO J2 = 2, N2 - 1
6             B(I1, J2) = A(I1, J2) + I1 * J2
7          ENDDO
8       ENDDO
```

Заметим, что рассматривается объединение двух гнезд циклов, из которых совпадающие границы имеют только циклы, находящиеся на внешнем уровне тесной вложенности. Поэтому в данном случае объединение производится только для внешних циклов.

Листинг 29. Результат сужения переменной для цикла из листинга 28
Listing 29. The result of shrinking of loop's private variable from listing 28

```
1       DO I1 = 1, N1
2          DO I2 = 1, N2
3             A1(I2) = I1 + I2
4          ENDDO
5          DO J2 = 2, N2 - 1
6             B(I1, J2) = A1(J2) + I1 * J2
7          ENDDO
8       ENDDO
```

В образованном цикле приватная переменная `A` имеет "лишнее" измерение, соответствующее итерационной переменной `I1`. На каждой итерации внешнего цикла по `I1` массив `A` заполняется по второму измерению в цикле по `I2`, полученные значения используются в цикле по `J2` и больше не используются нигде. Таким образом, первое измерение массива `A` не используется и может быть удалено, т.е. приватная переменная `A` может быть сужена по первому измерению. При этом она заменяется переменной `A1`, первое и единственное измерение которой соответствует второму измерению исходной переменной `A`. Результат сужения показан в листинге 29.

Таким образом, условие корректности преобразования сужения приватной переменной цикла можно сформулировать следующим образом. Пусть имеется тесно-вложенный цикл с итерационными переменными $I_1, I_2, \ldots, I_k$, и пусть переменная (массив) A является приватной для данного цикла. Тогда если переменная A индексируется по итерационным переменным цикла $I_{i1}, I_{i2}, \ldots, I_{in}$, где $\{I_{i1}, I_{i2}, \ldots, I_{in}\} \subseteq \{I_1, I_2, \ldots, I_k\}$, то переменная A может быть сужена по циклам с итерационными переменными $I_{i1}, I_{i2}, \ldots, I_{in}$.

6. Исследование реализованных преобразований. Все реализованные преобразования являются универсальными и могут быть применены в процессе распараллеливания различных программ. В данном разделе описывается процесс применения преобразований в задаче автоматизированного распараллеливания последовательной версии программы NAS LU в системе SAPFOR.

Программа LU, решающая систему нелинейных дифференциальных уравнений в частных производных, является одной из набора программ (тестов) NAS Parallel Benchmarks, разработанных для оценки производительности параллельных вычислительных систем [20]. Используемая версия набора тестов — 3.3. Исходный код программы содержит около 4000 строк (здесь и далее указывается количество строк в фиксированном формате записи программы). Основными вычислительными процедурами, на которые приходится более 95% всего времени выполнения программы, являются SSOR и RHS. Код данных процедур состоит из 260 (с учетом процедур, подставленных в результате преобразований, — 890) и 430 строк соответственно, или 7% (22%) и 11% от общего объема программы. В процедуре SSOR находится главный итерационный цикл программы, тело которого состоит из нескольких вложенных циклов и вызова процедуры RHS.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Объявление основных массивов, обрабатывающихся в программе LU, приведено в листинге 30. Параметры `ISIZ1`, `ISIZ2` и `ISIZ3` указываются в заголовочном файле и вместе с количеством итераций главного цикла определяют класс задачи.

Листинг 30. Объявление основных массивов программы LU
Listing 30. Declaration of the main arrays of the LU program

```
1       DOUBLE PRECISION U(5, ISIZ1/2*2 + 1, ISIZ2/2*2 + 1, ISIZ3),
2      >                 RSD(5, ISIZ1/2*2 + 1, ISIZ2/2*2 + 1, ISIZ3),
3      >                 FRCT(5, ISIZ1/2*2 + 1, ISIZ2/2*2 + 1, ISIZ3),
4      >                 QS(ISIZ1/2*2+1, ISIZ2/2*2+1, ISIZ3),
5      >                 RHO_I(ISIZ1/2*2+1, ISIZ2/2*2+1, ISIZ3)
```

Как видно из объявления, массивы `U`, `RSD` и `FRCT` совпадают по всем четырем измерениям, а их 2-е, 3-е и 4-е измерения совпадают с тремя измерениями массивов `QS` и `RHO_I` соответственно. Исходя из этого, наилучшим распределением данных для программы LU является распределение указанных массивов на трехмерный массив виртуальных процессоров. При этом на i-е измерение массива виртуальных процессоров распределяется i-е измерение массивов `QS` и `RHO_I` и $(i+1)$-е измерение массивов `U`, `RSD` и `FRCT`, $i = \overline{1,3}$. Первое измерение массивов `U`, `RSD` и `FRCT` размножается между всеми виртуальными процессорами. Наиболее подходящими для распараллеливания оказываются тесно-вложенные циклы степени три, итерационные переменные которых являются индексными выражениями для распределенных измерений данных массивов.

Поскольку основными вычислительными процедурами программы LU являются SSOR и RHS, именно их распараллеливание лежит в основе эффективности полученной параллельной версии программы.

В процедуре SSOR тела нескольких циклов, вложенных в главный итерационный цикл, состоят из вызовов процедур (JACLD, BLTS, JACU, BUTS), внутри которых обрабатываются основные массивы программы. Данные циклы не являются циклами потенциально параллельного вида и не могут быть распараллелены. Для их приведения к нужному виду было применено преобразование “точечная подстановка процедур”. В результате подстановки объем кода процедуры SSOR увеличился до 890 строк. На месте вызовов указанных процедур образовалось множество циклов, которые были преобразованы проходом “объединение циклов” к тесно-вложенному виду (в том числе некоторые из циклов при объединении были “развернуты”). После объединения некоторые приватные массивы в данных циклах оказалось возможным сузить по части измерений. Сужение переменных привело к существенной оптимизации программы уже в рамках последовательного кода (результаты запусков описаны в разделе 7). Полученные циклы степени тесной вложенности три были успешно распараллелены системой по всем уровням и выделены в вычислительный регион.

В процедуре RHS имелось несколько циклов степени тесной вложенности два, тело каждого из которых состояло из множества циклов. Однако вложенные циклы имели несовпадающие границы, из-за чего не могли быть объединены автоматизированно. Для приведения к потенциально параллельному виду потребовалось применить преобразование “разделение циклов”. Для возможности и корректности разделения предварительно были расширены приватные массивы, по которым данные вложенные циклы имели зависимость. Поскольку границы циклов в данной процедуре задавались с помощью переменных, которые инициализировались в другом модуле программы через ее параметры, проход не смог определить количество итераций циклов (для этого требуется анализ достигающих определений на уровне всей программы). Поэтому новые расширенные массивы были объявлены проходом в программе как динамические. Однако использование динамических массивов больших размеров в одной из основных процедур программы неэффективно, поэтому данные массивы были ручным преобразованием объявлены статическими с атрибутом `SAVE`, операторы их выделения и освобождения были удалены.

В результате преобразований тело процедуры RHS было приведено ко множеству потенциально параллельных тесно-вложенных циклов степеней три и четыре, которые были распараллелены системой SAPFOR по всем уровням тесной вложенности и выделены в вычислительный регион.

Преобразования, аналогичные выполненным в RHS, были проведены в процедуре ERHS. В процедуре SETBV были применены преобразования “точечная подстановка процедур”, “разделение циклов” и “объединение циклов”. Также подстановка процедур была выполнена в процедурах ERROR и SETIV. Во

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

всех случаях преобразования позволили привести вычислительные циклы в данных процедурах к потенциально параллельному виду.

В некоторых случаях для приведения цикла к тесно-вложенному виду потребовалось вынесение из цикла или внесение в него инварианта. Данное преобразование было выполнено вручную.

Суммарно преобразованием “точечная подстановка процедур” было вставлено около 800 строк кода. Преобразованиями “объединение циклов” и “разделение циклов” было изменено около 200 и 220 строк соответственно. Преобразованиями “сужение приватных переменных” и “расширение приватных переменных” — 500 и 170 строк соответственно. Ручными преобразованиями (объявление динамических массивов статическими, удаление операторов выделения и освобождения памяти, внесение и вынесение инварианта цикла) было изменено около 50 строк кода. Общее количество расставленных вручную в коде программы директив системы SAPFOR, необходимых для преобразований и построения параллельной версии, составило около 70.

Итого автоматизированными преобразованиями было затронуто около 1890 строк кода (47% от общего объема программы), из которых 1090 (27%) пришлось на преобразования “объединение/разделение циклов” и “сужение/расширение приватных переменных”. Ручными преобразованиями было изменено около 50 (1,3%) строк кода, не считая примерно 70 (1,8%) расставленных директив для системы SAPFOR.

7. Анализ эффективности полученной параллельной программы. Полученная после автоматизированных преобразований и распараллеливания через систему SAPFOR версия программы NAS LU (далее — SAPFOR-версия) была протестирована на гибридном вычислительном кластере K10, состоящем из 16 узлов (характеристики узла кластера приведены в табл. 1). Для сравнения также были протестированы DVMH- и MPI-версии теста LU, которые являются вручную преобразованными и распараллеленными версиями программы в моделях DVMH и MPI соответственно. Далее в таблицах с результатами запусков приведено время выполнения соответствующих версий программы на классе задач D в секундах.

SAPFOR-версия программы LU успешно прошла встроенную в нее проверку правильности вычислений как в последовательном, так и в параллельном вариантах, что свидетельствует о корректности примененных преобразований.

Сравнение исходной (непреобразованной), SAPFOR- и DVMH-версий при последовательном запуске на кластере K10 приведено в табл. 2. Из результатов видно, что преобразования ускорили выполнение процедуры SSOR в SAPFOR-версии по сравнению с исходной в 1.7 раз, однако замедлили выполнение процедуры RHS в 2.3 раза, что в совокупности дало увеличение общего времени выполнения на 2%. Ускорение в SSOR объясняется сужением приватных переменных в циклах, что дало экономию по памяти и уменьшило количество кэш-промахов при работе с данными переменными. Замедление в RHS вызвано расширением приватных переменных в циклах, что имело обратный эффект — увеличение используемой памяти и количества кэш-промахов при работе с ней.

В DVMH-версии преобразование циклов в процедуре RHS было реализовано иначе, чем в SAPFOR-версии. В DVMH-версии основные циклы были приведены к тесно-вложенному виду не через их разделение, а более сложным преобразованием — объединением циклов с разными границами, поэтому расширение приватных переменных не потребовалось. Из-за отсутствия расширения переменных процедура

Таблица 1. Характеристики узла гибридного кластера K10

Table 1. Characteristics of the K10 hybrid cluster node

Процессор (CPU) Central processing unit	2 × Intel Xeon E5-2660
Количество ядер Number of cores	16
Оперативная память RAM	128 GB
Графический ускоритель (GPU) Graphics processing unit	3 × nVidia Fermi M2090
Память графического ускорителя GPU memory	3 × 6 GB

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

Таблица 2. Результаты последовательных запусков, в секундах

Table 2. Results of sequential launches, in seconds

Версия Version	Общее время Total time	RHS	SSOR
Исходная Initial	17 112,9	4 025,4	12 935,2
SAPFOR	17 439,8	9 408,4	7 827,1
DVMH	9 348,5	3 618,4	6 287,5

RHS в DVMH-версии выполняется существенно быстрее (в 2.6 раза) по сравнению с SAPFOR-версией. В итоге DVMH-версия программы при последовательном запуске выполняется почти в 1.9 раза быстрее SAPFOR-версии.

В табл. 3 приведены сравнительные результаты запусков трех версий программы на центральных процессорах (CPU) узлов кластера. При выполнении SAPFOR- и DVMH-версий указывались решетки виртуальных процессоров вида N × N × 1, за исключением запуска на 240 процессах, где использовались решетки 15 × 16 × 1 и 16 × 15 × 1 (в таблице приведен лучший результат). В MPI-версии используется решетка процессоров N × N. Из таблицы видно, что в среднем SAPFOR-версия выполняется медленнее DVMH-версии в 2 раза и быстрее MPI-версии в 1,1 раз (на 10%). При этом разница приблизительно в 2 раза между SAPFOR- и DVMH-версиями сохраняется при любом количестве процессов, а при более чем 144-х процессах разница между SAPFOR- и MPI-версиями почти отсутствует. Итоговое ускорение SAPFOR-версии программы относительно исходной последовательной версии при использовании половины кластера (121 процесс на 8 узлах) составило около 67 раз, а при использовании 15 узлов кластера (240 процессов) — около 120 раз.

Ускорение SAPFOR-версии при переходе от одного используемого процесса к 240 процессам составляет 123 раза, DVMH-версии — 133 раза, а MPI-версии — 112 раз (при переходе от одного к 225 процессам). Это свидетельствует о достаточно хорошей эффективности SAPFOR-версии, полученной автоматизированными преобразованиями, по сравнению с вручную преобразованными версиями.

Таблица 3. Результаты запусков на CPU, в секундах

Table 3. Results of CPU launches, in seconds

Количество процессов Number of processes	SAPFOR-версия SAPFOR-version	DVMH-версия DVMH-version	MPI-версия MPI-version
1	17 439,8	9 348,5	18 290,4
4	4 222,7	2 614,4	4 555,5
9	2 299,8	1 236,5	2 452,8
16	1 492,7	738,5	1 983,0
25	996,6	486,8	1 151,1
36	703,8	340,8	768,9
49	538,3	274,6	678,0
64	412,2	200,4	458,7
81	335,6	164,1	392,1
100	274,2	136,1	300,6
121	255,8	120,1	287,7
144	217,1	106,9	213,3
169	189,5	90,6	180,9
196	166,2	81,7	164,4
225	149,3	76,8	163,2
240	141,9	70,4	—

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

В табл. 4 приведены результаты запусков SAPFOR- и DVMH-версий программы на графических ускорителях (GPU) узлов гибридного кластера. Поскольку при выполнении региона программы на ускорителе все используемые в регионе данные должны поместиться в памяти устройства, существует минимальное количество устройств, требуемых для успешного запуска программы. Для DVMH-версии потребовалось минимум 3 устройства, для SAPFOR-версии оказалось необходимо не менее 16 устройств GPU. Увеличенная потребность в памяти для SAPFOR-версии программы объясняется расширением приватных переменных в процедуре RHS и введением новых массивов больших размерностей. Поскольку все циклы в RHS были заключены при распараллеливании в регион, для его выполнения требуется больший объем памяти по сравнению с DVMH-версией, где приватные переменные не расширялись. Итого общий объем занимаемой памяти на данном классе задачи для SAPFOR-версии оказался равным приблизительно 100 GB, для DVMH-версии — 20 GB.

При запуске на GPU указывались решетки виртуальных процессоров вида $1 \times N \times N$, за исключением запуска на 3 и 42 устройствах (в этих случаях приведен лучший результат для пар решеток $1 \times 1 \times 3$ и $1 \times 3 \times 1$, $1 \times 6 \times 7$ и $1 \times 7 \times 6$ соответственно). Из таблицы видно, что обе версии программы ускорялись при увеличении количества устройств GPU до 25, после чего происходило замедление, вызванное обменами данными между узлами кластера. Итоговое ускорение в наилучшей для обеих версий конфигурации при запуске на 25 устройствах GPU относительно исходной последовательной версии составило около 43 раз для SAPFOR- и около 45 раз для DVMH-версии. Относительная разница времени выполнения двух версий составила менее 5%. При этом поведение двух версий схоже при большом количестве устройств GPU.

Таким образом, при выполнении на CPU SAPFOR-версия оказывается медленнее DVMH-версии приблизительно в 2 раза при любом количестве процессов и в среднем быстрее MPI-версии в 1,1 раза. При выполнении на GPU наилучшие результаты показывают отставание SAPFOR-версии от DVMH-версии на менее чем 5%, что свидетельствует об относительно высокой эффективности полученной автоматизированными преобразованиями параллельной версии программы по сравнению с вручную распараллеленной версией при выполнении на графических ускорителях.

Дальнейшее улучшение эффективности автоматизированного распараллеливания возможно через реализацию преобразования объединения циклов с несовпадающими границами, которое позволит преобразовать циклы в процедуре RHS аналогично тому, как это сделано в DVMH-версии программы. Данное улучшение позволит избежать необходимости расширения приватных переменных, которое приводит к дополнительным затратам по используемой памяти и увеличивает количество кэш-промахов.

8. Заключение. В данной статье были рассмотрены следующие преобразования последовательных Fortran-программ: “объединение/разделение циклов” и “сужение/расширение приватных переменных”. Были описаны условия их применимости и приведены алгоритмы их реализации. Система SAPFOR была дополнена проходами, реализующими данные преобразования.

Применимость реализованных преобразований была исследована в процессе автоматизированного распараллеливания программы NAS LU. Суммарно автоматизированными преобразованиями было изменено более 47% строк кода программы, из которых более половины пришлось на реализованные преобразования. Вручную было изменено около 1.3% строк программы, не считая примерно 70 расставленных

Таблица 4. Результаты запусков на GPU, в секундах

Table 4. Results of GPU launches, in seconds

Количество устройств GPU Number of GPU units	SAPFOR-версия SAPFOR-version	DVMH-версия DVMH-version
3	—	1 236,4
4	—	907,9
9	—	485,5
16	483,5	385,8
25	396,0	379,7
36	454,2	561,5
42	423,5	523,4

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

директив для системы SAPFOR (1.8% от общего объема кода). Данные результаты свидетельствуют о значительном уменьшении трудоемкости распараллеливания программы для пользователя системы благодаря реализованным преобразованиям.

Полученная в результате автоматизированных преобразований и распараллеливания SAPFOR-версия программы LU была протестирована на гибридном вычислительном кластере K10 в сравнении с распараллеленными вручную MPI- и DVMH-версиями. Результаты запусков показали, что при выполнении на CPU SAPFOR-версия оказалась медленнее DVMH-версии приблизительно в 2 раза при любом количестве используемых процессов и в среднем быстрее MPI-версии в 1,1 раза. Все версии имеют схожее поведение при росте числа процессов и примерно равные ускорения относительно последовательного исполнения. При выполнении на GPU в наилучшей для обеих версий конфигурации результаты показали отставание SAPFOR-версии относительно DVMH-версии на менее чем 5%, что является существенно лучшим результатом, чем двукратное отставание при выполнении на CPU.

Таким образом, реализованные преобразования расширили возможности системы SAPFOR и позволили автоматизированно распараллелить программу NAS LU, полученная параллельная версия показала относительно высокую эффективность по сравнению с вручную распараллеленной DVMH-версией программы, в особенности при выполнении на графических ускорителях. Дальнейшее повышение эффективности автоматизированного распараллеливания возможно через реализацию преобразования объединения циклов с несовпадающими границами.

Список литературы

1. OpenMP Specification. https://www.openmp.org/specifications/. Cited October 12, 2022.
2. MPI Documents. https://www.mpi-forum.org/docs/. Cited October 12, 2022.
3. CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/. Cited October 12, 2022.
4. Khronos OpenCL Registry. https://www.khronos.org/registry/OpenCL/. Cited October 12, 2022.
5. *Wolfe M.* High performance compilers for parallel computing // New York: Addison-Wesley, 1995.
6. *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.
7. *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.
8. *Grosser T., Groesslinger A., Lengauer C.* Polly — performing polyhedral optimizations on a low-level intermediate representation // Parallel Processing Letters. 2012. **22**. doi 10.1142/S0129626412500107.
9. *Grosser T., Hoefler T.* Polly-ACC transparent compilation to heterogeneous hardware // Proc. Int. Conf. on Supercomputing. Istambul, Turkey, June 1–3, 2016. New York: ACM Press, 2016. doi 10.1145/2925426.2926286.
10. *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). Stockholm, Sweden, January 2017. https://www.researchgate.net/publication/313059456_APOLLO_Automatic_speculative_POLyhedral_Loop_Optimizer. Cited October 12, 2022.
11. *Lattner C., Adve V.* LLVM: a compilation framework for lifelong program analysis & transformation // Proc. Int. Symposium on Code Generation and Optimization (CGO'04). Palo Alto, California, USA, March 20–24, 2004. doi 10.1109/CGO.2004.1281665.
12. *Doerfert J., Streit K., Hack S., Benaissa Z.* Polly's polyhedral scheduling in the presence of reductions // Proc. 5th International Workshop on Polyhedral Compilation Techniques. 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 12, 2022.
13. Описание DVM-системы. http://dvm-system.org/. Дата обращения: 12 октября 2022.
14. Документация по языкам C-DVMH и Fortran-DVMH. http://dvm-system.org/ru/docs/. Дата обращения: 12 октября 2022.
15. Описание системы SAPFOR. http://keldysh.ru/dvm/SAPFOR/. Дата обращения: 12 октября 2022.
16. *Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А.* Автоматизация распараллеливания программных комплексов // Труды XVIII Всероссийской научной конференции "Научный сервис в сети Интернет", 19–24 сентября 2016, Новороссийск. М.: ИПМ им. М.В. Келдыша, 2016. 76–85.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

17. *Колганов А.С.* Автоматизация распараллеливания Фортран-программ для гетерогенных кластеров: автореф. дисс. канд. физ.-мат. наук. 2020.
18. *Бартеньев О.В.* Современный Фортран. М.: ДИАЛОГ-МИФИ, 2000.
19. *Ахо А.В., Лам М.С., Сети Р., Ульман Д.Д.* Компиляторы: принципы, технологии и инструментарий. М.: Вильямс, 2008.
20. NAS Parallel Benchmarks. https://www.nas.nasa.gov/software/npb.html. Cited October 14, 2022.

Поступила в редакцию
30 июля 2022 г.

Принята к публикации
13 сентября 2022 г.

Информация об авторах

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

Григорий Дмитриевич Гусев — студент; Московский государственный университет имени М. В. Ломоносова, Ленинские горы, 1, 119991, Москва, Российская Федерация.

References

1. OpenMP Specification. https://www.openmp.org/specifications/. Cited October 12, 2022.
2. MPI Documents. https://www.mpi-forum.org/docs/. Cited October 12, 2022.
3. CUDA Toolkit Documentation. https://docs.nvidia.com/cuda/. Cited October 12, 2022.
4. Khronos OpenCL Registry. https://www.khronos.org/registry/OpenCL/. Cited October 12, 2022.
5. M. Wolfe, *High Performance Compilers for Parallel Computing* (Addison-Wesley, New York, 1995).
6. 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.
7. 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.
8. 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.
9. 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.
10. J. M. 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 12, 2022.
11. C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *Proc. Int. Symp. on Code Generation and Optimization. Palo Alto, USA, March 20–24, 2004*, doi 10.1109/CGO.2004.1281665.
12. 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 12, 2022.
13. Description of DVM-system. http://dvm-system.org/. Cited October 12, 2022.
14. Documentation for C-DVM and Fortran-DVMH Languages. http://dvm-system.org/ru/docs/. Cited October 12, 2022.
15. Description of SAPFOR System. http://keldysh.ru/dvm/SAPFOR/. Cited October 12, 2022.
16. V. A. Bakhtin, O. F. Zhukova, N. A. Kataev, et al., "Automation of Parallelization of Software Complexes," in *Proc. Conf. on "Scientific Service on the Internet", Novorossiysk, Russia, September 19–24, 2016* (Keldysh Institute of Applied Mathematics, Moscow, 2016), pp. 76–85.

2022, **23** (4), 288–310. doi 10.26089/NumMet.v23r418

17. A. S. Kolganov, *Automation of Parallelization of Fortran Programs for Heterogeneous Clusters*, Candidate's Dissertation in Mathematics and Physics (Keldysh Institute of Applied Mathematics, Moscow, 2020).
18. O. V. Bartenev, *Modern Fortran* (DIALOG-MEPHI, Moscow, 2000) [in Russian].
19. A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools* (Addison-Wesley, Boston, 2006; Williams, Moscow, 2008).
20. NAS Parallel Benchmarks. https://www.nas.nasa.gov/software/npb.html. Cited October 14, 2022.

Received
July 30, 2022

Accepted for publication
September 13, 2022

Information about the authors

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

Grigorii D. Gusev — Student; Lomonosov Moscow State University, Leninskie Gory, 1, 119991, Moscow, Russia.