2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

doi 10.26089/NumMet.v23r424 УДК 04.272, 004.4'242

Автоматизированное распараллеливание программ для гетерогенных кластеров с помощью системы SAPFOR

Н. А. Катаев
Институт прикладной математики имени М. В. Келдыша РАН (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0002-7603-4026, e-mail: kataev_nik@mail.ru

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

Аннотация: В статье рассмотрен подход к автоматизированному распараллеливанию программ для кластеров с помощью системы SAPFOR (System FOR Automated Parallelization). Главной целью системы SAPFOR является автоматизация процесса отображения последовательных программ на параллельные архитектуры в модели DVMH, которая является моделью программирования, основанной на директивах. Помимо этого, система SAPFOR позволяет выполнять автоматически некоторый класс преобразований над исходным кодом программы по запросу пользователя через графический интерфейс. На определенных классах задач пользователь системы SAPFOR может рассчитывать на полностью автоматическое распараллеливание, если программа была написана или приведена к потенциально параллельному виду. Также в статье описаны подходы к построению схем распределения данных и вычислений на распределенную память в модели DVMH. Эффективность полученных алгоритмов построения схем распределения данных и вычислений продемонстрирована на примере некоторых приложений из пакета NAS Parallel Benchmarks.

Ключевые слова: SAPFOR (System FOR Automated Parallelization), DVMH, автоматизация распараллеливания, распределение данных, распределение вычислений, гетерогенные кластеры.

Для цитирования: Катаев Н.А., Колганов А.С. Автоматизированное распараллеливание программ для гетерогенных кластеров с помощью системы SAPFOR // Вычислительные методы и программирование. 2022. **23**, № 4. 379–394. doi 10.26089/NumMet.v23r424.

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

Automated parallelization of programs for heterogeneous clusters using the SAPFOR system

Nikita A. Kataev
Keldysh Institute of Applied Mathematics, Moscow, Russia
ORCID: 0000-0002-7603-4026, e-mail: kataev_nik@mail.ru

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

Abstract: This paper has proposed an approach to the automated parallelization of programs for heterogeneous computational clusters. This approach is implemented in SAPFOR (System FOR Automated Parallelization). SAPFOR is a software development suite that aims to produce a parallel version of a sequential program in a semi-automatic way. SAPFOR uses the DVMH directive-based programming model to expose parallelism in the code. SAPFOR also implements different source-to-source transformations and gives the user opportunity to control the parallelization process through the graphical user interface. Fully automatic parallelization is also possible if the program is well-formed and satisfies certain requirements. This paper has described an approach which allows SAPFOR to automate selection of data and computation distribution. We use the NAS Parallel Benchmarks to evaluate the performance of generated programs.

Keywords: SAPFOR (System FOR Automated Parallelization), DVMH, parallelization automation, data distribution, distribution of computations, heterogeneous clusters.

For citation: N. A. Kataev, A. S. Kolganov, "Automated parallelization of programs for heterogeneous clusters using the SAPFOR system," Numerical Methods and Programming. **23** (4), 379–394 (2022). doi 10.26089/NumMet.v23r424.

1. Введение. Высокопроизводительные вычислительные технологии получили широкое распространение среди большого количества областей науки: вычислительная гидродинамика, исследования климата и окружающей среды, нейронные сети и искусственный интеллект, и многие другие. Сложились разнообразные подходы к организации параллельных вычислений, предполагающие использование различных технологий параллельного программирования [1]. При этом при выборе и применении предпочтительных подходов для решения конкретной задачи часто приходится сталкиваться с определенными трудностями [2].

Среди них можно выделить необходимость одновременного применения различных технологий программирования (MPI+OpenMP, MPI+OpenMP+CUDA и т.д.), чтобы задействовать все доступные вычислительные ресурсы. При этом прикладной программист должен обладать знаниями, позволяющими применять каждую из них. Множество используемых технологий может меняться по мере развития доступной вычислительной аппаратуры, что усложняет сопровождение и развитие уже написанных программных комплексов. Кроме того, запуск вычислительной задачи на сложной, часто гибридной, вычислительной системе может сопровождаться необходимостью подбора многочисленных параметров для достижения максимальной производительности. В сложившейся ситуации крайне желательной становится автоматизация максимального количества этапов, составляющих процесс разработки и сопровождения параллельной программы.

Одним из направлений такой автоматизации является разработка единых подходов, охватывающих сразу несколько уровней параллелизма и позволяющих отображать программу на различные архитектуры вычислителей. Примером единого подхода является стандарт SYCL [3], который добавляет параллелизм в последние версии языка C++ для поддержки параллельного выполнения на GPU, CPU и FPGA. Параллельная программа должна явно описывать параллельное выполнение, при этом оставаясь программой на стандартном языке C++, а ответственность за реализацию параллелизма ложится на используемые компиляторы. Помимо открытых реализаций данного стандарта активно развивается коммерческая версия

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

от Intel (oneAPI). В исходном виде SYCL ориентирован на системы с общей памятью, но на его основе также разрабатывается открытый проект Celerity [4] для отображения программ на кластеры, оснащенные в узлах ускорителями.

Альтернативой такому подходу может быть применение директивных расширений стандартных последовательных языков программирования XcalabalACC [5], DVMH [6, 7]. Преимущество этих подходов заключается в том, что они позволяют сначала разрабатывать и отлаживать последовательную программу, а потом добавлять в нее спецификации параллелизма, в то время как подходы, аналогичные SYCL, требуют, чтобы программа изначально разрабатывалась с использованием конструкций, описывающих параллелизм.

Являясь достаточно универсальной для описания различных уровней параллелизма, доступных на гибридном вычислительном кластере, DVMH модель скрывает конкретные технологии программирования внутри реализации компиляторов с DVMH языков. Это, в свою очередь, позволяет при необходимости расширять множество поддерживаемых архитектур, избегая значительных изменений в DVMH модели, и обеспечивает переносимость существующих DVMH программ. Еще одним уровнем автоматизации, который обеспечивает DVM система, является возможность динамической настройки запускаемых параллельных программ на выделенные для их выполнения вычислительные ресурсы [8].

Несмотря на то что модели программирования, опирающиеся на директивные расширения существующих языков, являются высокоуровневыми, их применение прикладным программистом все равно требует достаточных знаний в области параллельных вычислений и может быть сопряжено с трудностями. Способствовать решению данной проблемы может дальнейшая автоматизация процесса распараллеливания, связанная с созданием автоматизирующих систем, которые упрощают перевод последовательной программы в параллельную. Однако полностью автоматическое распараллеливание произвольных программ сталкивается со значительными трудностями, не позволяющими достичь приемлемой эффективности получаемых параллельных версий. Поэтому на рассматриваемые последовательные программы могут накладываться существенные ограничения, а пользователь получает возможность описывать свойства программ, которые невозможно проанализировать [9–12].

В связи с этим перспективным видится смешанный подход, который объединяет использование высокоуровневой модели параллельного программирования, системы автоматизации, ответственной за выполнение наиболее трудоемких этапов распараллеливания, и возможность для пользователя контролировать ход распараллеливания и принимать в нем активное участие [13].

В качестве инструмента автоматизации может выступать система SAPFOR (System FOR Automated Parallelization) [14, 15], ориентированная на использование DVMH языков как целевых. С одной стороны, система включает автоматически распараллеливающий компилятор, при этом инкапсулированные в DVMH модели возможности по динамической настройке запускаемых параллельных программ упрощают его разработку. С другой стороны, система обладает широкими возможностями по статическому и динамическому анализу программ [16, 17], автоматизирует выполнение преобразований исходной программы, позволяя пользователю выбирать фрагменты программы, которые должны быть преобразованы. Графический интерфейс пользователя позволяет управлять процессом распараллеливания [18].

В предыдущих работах [13, 19] рассматривались возможности системы SAPFOR, направленные на распараллеливание программ для систем с общей памятью (мультипроцессор, GPU). Данная статья охватывает дальнейшее расширение возможностей системы SAPFOR в направлении отображения программ на многоядерные гибридные вычислительные кластеры. В статье основное внимание уделяется алгоритмам распределения данных и вычислений между узлами кластера в модели DVMH.

Статья состоит из трех разделов и заключения. В разделе 2 приведен краткий обзор работ, направленных на автоматизацию разработки параллельных программ для систем с распределенной памятью. Раздел 3 посвящен алгоритмам распределения данных и вычислений, реализованным в системе SAPFOR. Результаты вычислительных экспериментов, охватывающие распараллеливание некоторых программ из набора NAS Parallel Benchmarks [20], приведены в разделе 4. Выводы приведены в заключении.

2. Обзор существующих работ. Распараллеливание для кластера обладает существенными особенностями, отличающими его от распараллеливания для систем с общей памятью. Разработчику приходится учитывать размещение данных на узлах системы наряду с распределением вычислений между отдельными вычислительными устройствами, чтобы обеспечить доступ к удаленным данным, максимально сократив при этом коммуникационные издержки. Таким образом, важную роль начинает играть требование локальности данных, используемых на каждом узле, и необходимость сбалансированного распреде-

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

ления данных, чтобы равномерно загрузить вычислительные устройства работой. Ситуация усугубляется тем, что приходится принимать глобальные решения в рамках всей программы в целом, так как отдельные ее фрагменты могут накладывать противоречивые требования, что в конечном счете приведет к дополнительным коммуникациям, направленным на перераспределение данных.

В связи с этим рассматриваются различные подходы к распараллеливанию программ на вычислительный кластер. Так как процесс распараллеливания для систем с распределенной памятью можно разделить на три этапа: распределение данных, распределение вычислений, организация коммуникаций для доступа к удаленным данным или перераспределение данных, то автоматизации могут подвергаться только некоторые из них.

Например, в работе [21] рассматривается подход к построению оптимального распределения вычислений и организации коммуникаций, опирающийся на применение полиэдральной модели распараллеливаемой программы, для предварительно заданного фиксированного распределения данных. Подход, основанный на предварительно заданном распределении данных, также применялся в инструменте [22], при этом интересно, что инструмент обеспечивал пользователя диалоговой оболочкой, позволяющей управлять процессом распараллеливания, задавать распределение данных и управлять преобразованиями программ. Распределение вычислений выполнялось автоматически и основывалось на правиле собственных вычислений, когда запись значений выполняется на том процессоре, который владеет записываемыми данными. Подход, описанный в [21], позволяет ослабить это ограничение, в том числе обеспечивая размножение данных между узлами.

Инструмент Molly [11] расширяет возможности компилятора Polly [23] для систем с общей памятью, построенного на базе LLVM [24] и основанного на использовании полиэдральной модели. При этом вводится специальный тип данных, описывающий распределяемые массивы, что позволяет контролировать отсутствие операций адресной арифметики, применяемых к распределяемым данным. Распределение данных выполняется равными блоками распределяемых массивов между процессами, причем выравнивание данных друг на друга не предусмотрено, а для распределения вычислений применяется правило собственных вычислений. Предполагается, что в дальнейшем пользователь сможет корректировать как распределение данных, так и вычислений, задавая соответствующие директивы. Кроме того, накладываются дополнительные ограничения на структуру распараллеливаемых фрагментов (SCoP), вводится требование глобальности распределяемых данных, что позволяет избежать подробного межпроцедурного анализа, редукционные операции также не поддерживаются.

Подход с использованием директив, описывающих распределение данных, также предложен в работе [25]. Директивы содержат большое количество параметров, позволяющих гибко управлять требуемым распределением данных. Также предлагается нестандартное распределение массивов с перекрытиями, что позволяет сократить частоту обменов данными. При этом использование специальных директив для описания такого распределения упрощает разработку программы и уменьшает вероятность ошибок при задании сложных индексных выражений в обращениях к массивам.

Подход, основанный на оптимизации распределения вычислений и необходимых коммуникаций, также приведен в работе [26], расширяющей возможности полиэдрального компилятора Pluto [27] для систем с общей памятью. При этом распределение данных как таковое не требуется, а размещение данных на узлах в каждый конкретный момент определяется выполняемыми над ними вычислениями. Данный подход не предусматривает глобального принятия решений по распределению данных, которое максимально бы соответствовало распределению вычислений, что может привести к увеличению частоты и объема коммуникаций.

Построение распределения данных наряду с распределением вычислений и оптимизацией коммуникаций было реализовано в инструменте Paradigm [28]. Исследования были направлены на распараллеливание последовательных программ на языке Fortran–77. Инструмент предполагал поддержку достаточно широкого класса задач, в том числе за счет поддержки нерегулярных вычислений и распараллеливания циклов с зависимостями за счет организации конвейерного выполнения. При этом в работе отмечаются ограничения, связанные с определением правила выравнивания измерения массивов, а экспериментальные результаты приводятся для небольших вычислительных ядер.

В статье [29] рассматривается подход в виде высокоуровневой модели параллельных вычислений BSF (Bulk Synchronous Farm), являющейся расширением модели BSP и основанной на методе программирования SPMD и фреймворке “мастер–рабочий”. Модель BSF ориентирована на численные итерационные методы с высокой временно́й сложностью. Модель параллельных вычислений BSF представляет собой

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

блочно-синхронную ферму, ориентирована на многопроцессорные системы с кластерной архитектурой и архитектурой MPP. BSF-компьютер представляет собой множество однородных процессорных узлов с приватной памятью, соединенных сетью, позволяющей передавать данные от одного процессорного узла к другому. Среди процессорных узлов выделяется один, называемый узлом-мастером (или кратко мастером). Остальные узлы называются узлами-рабочими (или просто рабочими). В BSF-компьютере должен быть по крайней мере один узел мастер и один рабочий.

3. Построение схем распределения данных и вычислений в системе SAPFOR.

3.1. Варианты отображения последовательной программы на кластер. Распределение данных — одна из основных проблем отображения последовательной программы на кластер в случае использования регулярных сеток. Для эффективного задействования всей мощности вычислительного кластера требуется учитывать специфику обработки данных в циклах последовательной программы, так как зачастую в них содержится основная вычислительная нагрузка. Рассмотрим следующие варианты отображения последовательной программы на кластер:

- выполняется только распределение вычислений, а данные остаются размноженными на всех узлах. При таком подходе существенно упрощается преобразование последовательной программы в параллельную для кластера, так как все данные дублируются на каждом узле кластера. В свою очередь, каждый цикл может “требовать” свое распределение, что легко реализуется данной моделью;
- выполняется распределение как данных, так и вычислений. При таком подходе необходимо учитывать интересы всех циклов, участвующих в распараллеливании. В этом случае в силу того, что на каждом узле присутствует только часть данных, необходимо выполнять их пересылки для корректного выполнения последовательных участков программы.

Система SAPFOR преобразует исходную программу в параллельную с использованием модели DVMH. Данная модель использует второй вариант отображения последовательной программы на кластер — отображение как данных, так и вычислений на узлы кластера. В связи с этим необходимо обеспечить такое распределение данных, чтобы количество коммуникаций между процессорами, а также их объем были как можно меньше. Следует отметить, что с помощью модели DVMH можно реализовать и первый вариант, но, во-первых, параллельная программа будет требовать столько же памяти на каждом узле, что и последовательная, и, во-вторых, эффективность выполнения такой программы может быть ниже из-за большого объема коммуникаций, возникающих в момент перераспределения данных.

Алгоритм распределения данных состоит из двух этапов. На первом этапе выполняется межпроцедурный анализ всей программы: анализируются циклы и используемые в них массивы. Затем собранная информация обобщается и выполняется поиск распределения данных с как можно меньшими издержками по пересылке данных между узлами.

3.2. Определение распределяемых массивов. По умолчанию система SAPFOR считает все массивы в программе распределяемыми. Распределяемый массив — это такой массив, для которого необходимо построить распределение данных с помощью DVMH-директив и выполнить соответствующее выбранному распределению отображение вычислений в параллельных циклах и одиночных операторах. Но не все массивы могут быть распределенными. Чтобы снизить количество распределяемых данных, системе SAPFOR необходимо уметь распознавать такие ситуации либо в автоматическом, либо в полуавтоматическом режиме. Рассмотрим случаи, когда массив не требует распределения.

К первой категории массивов, которые не требуется распределять, относятся те массивы, которые были определены как приватизируемые или редукционные системой SAPFOR или были указаны в соответствующих спецификациях пользователем в исходном коде программы или через графический интерфейс. Данные массивы являются вспомогательными в местах использования (в основном в циклах), при этом использовать в циклах распределенный массив как приватизируемый или редукционный запрещено DVM-системой.

Ко второй категории относятся массивы, которые участвуют в операторах ввода-вывода, а также массивы, которые передаются как параметры во внешние процедуры (например, процедуры стандартных библиотек либо процедуры, которые недоступны для анализа системе SAPFOR). В операторах ввода-вывода разрешается задавать только по одному массиву и только целиком из-за ограничений DVM-системы (т.е. можно выводить целиком весь массив). Все остальные случаи отменяют распределение данного массива. Для того чтобы не отменять распределение массивов, которые участвуют в сложных операторах ввода-вывода, а также во внешних процедурах, требуется заводить массивы-копии и встав-

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

лять копирования до и после соответствующих операторов. В текущий момент в системе SAPFOR такое преобразование не автоматизировано.

К третьей категории относятся массивы, которые система SAPFOR автоматически отфильтровывает по тем или иным причинам. Процесс фильтрации состоит в том, чтобы запретить использовать распределяемые массивы, которые в дальнейшем будут отображены на разные деревья выравнивания и соответственно на разные DVMH-шаблоны в общем гнезде потенциально параллельных циклов. Разные деревья выравнивания образуются из не связанных между собой графов массивов. Также отфильтровываются массивы, объявленные в глобальной области видимости, которые могли быть использованы в процедурах, вызываемых из цикла, без явного использования в самом цикле. Такое ограничение связано с тем, что в DVMH-модели во всех параллельных циклах необходимо "видеть" все используемые массивы. Те данные, которые используются в вызываемых из цикла процедурах, но не используются в циклах, считаются "невидимыми" для DVMH-компилятора, что приведет к ошибке конвертации программы и ее выполнения.

После выполнения фильтрации происходит распространение состояния потенциальной распределенности массивов в соответствии со связями фактических и формальных параметров процедур в программе. По массивам, которые не были отфильтрованы, будет построен граф массивов, который в свою очередь будет отображен в дерево выравнивания в модели DVMH.

3.3. Анализ программы: построение связей циклов и массивов. На данном этапе системой SAPFOR для каждой функции в исходной программе выполняется анализ всех ее операторов. Для каждого обращения к массиву, находящемуся внутри гнезда циклов, для каждого из измерений (отдельно и независимо) выполняется следующий анализ:

- выполняется поиск косвенной адресации в обращении. Косвенная адресация не может быть эффективно распараллелена DVMH-моделью в случае структурированных сеток;
- выполняется поиск более чем одной итерационной переменной цикла в индексном выражении в обращении к массиву. Для корректного отображения данных и связывания распределения вычислений с распределением данных необходимо однозначное соответствие индексного выражения в обращении к массиву с итерационной переменной одного из циклов в гнезде. Если имеется связь с одним циклом, то выполняется попытка сопоставления индексного выражения с итерационной переменной цикла I с шаблоном $a * I + b$ и вычисления данных коэффициентов;
- выполняется проверка всех измерений массива. Если обращение к массиву используется на запись, проверяется факт того, что все индексные выражения в обращении к массиву имеют хотя бы одну итерационную переменную цикла. Если итерационная переменная цикла не встречается в индексных выражениях, то для рассматриваемого цикла отмечается факт наличия неопределенной записи в массиве.

Важно отметить, что вся информация привязывается к циклам, т.е. для каждого цикла формируется список всех обращений к массивам. Для того чтобы цикл был оптимально распараллелен системой SAPFOR, требуется, чтобы каждое индексное выражение в обращении по конкретному измерению массива содержало только одну итерационную переменную цикла или, иными словами, должно быть однозначное отображение измерений массива на циклы.

После обработки функции будет построена общая информация о "хороших" обращениях к массивам с привязкой этих обращений к циклам с коэффициентами $a*I+b$, где $a, b > 0$ — вычисленные константы, $a \neq 0$, а I — итерационная переменная цикла. Остальные обращения к массивам будут порождать неизбежные обмены между узлами кластера.

3.4. Анализ программы: построение графа массивов. Граф массивов является основной структурой данных, на основе которой строятся распределения данных и вычислений. Построение графа массивов обусловлено выбором целевой модели при создании параллельной версии программы. DVMH-модель требует выполнения правила собственных вычислений: каждый процессор изменяет только собственные данные, т.е. данные, которые распределены на этот процессор. Кроме того, вычисление одной итерации цикла должно целиком выполняться на одном процессоре и, следовательно, элементы массивов, вычисляемые на одной итерации, оказываются связанными между собой и должны быть размещены на одном процессоре.

Для соблюдения данного правила необходимо использовать взаимное выравнивание массивов между собой с помощью спецификации ALIGN. После распределения массивов с помощью спецификаций

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

Листинг 1. Пример программы на языке C, который был использован для построения графа массивов на рис. 1
Listing 1. An example of a C program which is used to build a graph of arrays in Fig. 1

```
1  int A[40][50], B[40][50];
2  void foo(int N, int M) {
3    for (int I = 0; I < N; ++I) {
4      for (int K = 0; K < M; ++K)
5        A[I][K] = B[I][K] + B[K][K];
6    }
7    for (int I = 0; I < N; ++I) {
8      for (int K = 0; K < M - 1; ++K)
9        A[I][K] = B[I][K] + B[I][K + 1];
10   }
11 }
```

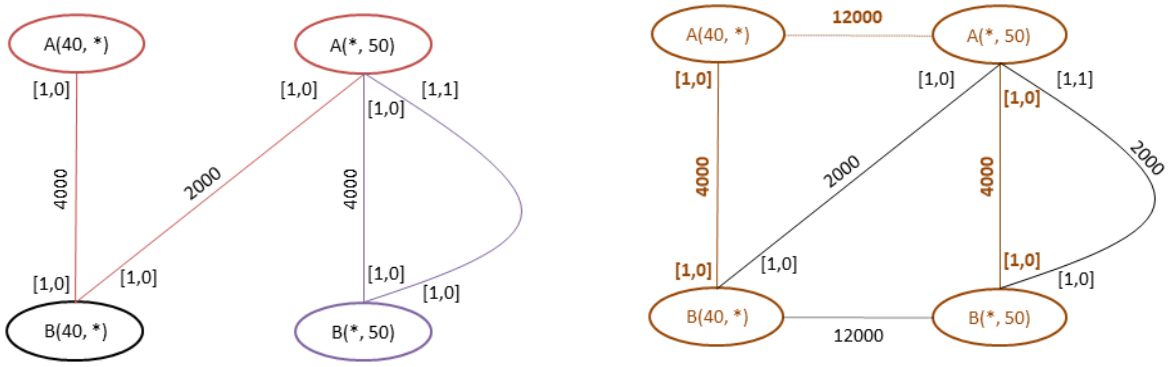

Рис. 1. Граф массивов для программы из листинга 1: a) конфликты в графе; b) максимальное остовное дерево
Fig. 1. Graph of arrays for the program in Listing 1: a) conflicts in the graph b) the maximum spanning tree

DISTRIBUTE и ALIGN получается дерево выравнивания, которое описывает связи между всеми массивами. Правило собственных вычислений требует, чтобы все массивы, используемые в одном цикле, принадлежали одному дереву выравнивания.

Пример графа массивов для программы из листинга 1 приведен на рис. 1 a. Рассмотрим подробнее процесс построения графа.

Граф массивов представлен в системе SAPFOR в формате CSR (Compressed Sparse Rows) и является неориентированным. Каждое измерение массива становится узлом графа, а дуги показывают связь одного измерения массива с другим. Каждая дуга в графе массивов связывает одно измерение массива Arr_1 с измерением массива Arr_2. Для заполнения графа массивов необходимо обработать информацию о вычисленных коэффициентах a, b в обращениях к массивам и их связях с циклом. Дуги в графе массивов характеризуются коэффициентами выравнивания, вычисленными на основе коэффициентов a и b (см. атрибуты в квадратных скобках около дуг графа на рис. 1), а также способом доступа к элементу массива (чтение или запись) в породившем дугу операторе программы.

Дуги добавляются по следующему принципу (W или Write означает обращение на запись, R или Read означает обращение на чтение):

- связываются измерения массивов, обращения по которым присутствуют в левой части операторов присваивания в цикле с типом дуги запись-запись (связь $W - W$) и весом $Loop_w * SumB$;
- связываются измерения массивов Arr_1 и Arr_2, причем обращение к массиву Arr_1 содержится в левой части операторов присваивания, а обращение к массиву Arr_2 содержится в правой части операторов присваивания или в условиях IF, причем не обязательно, чтобы массивы Arr_1 и Arr_2 были в одном операторе. Связь создается с типом дуги запись–чтение (связь $W - R$) по данному циклу с весом $Loop_w * SumB$;

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

- в случае отсутствия операций записи в массивы связываются измерения массивов, обращения по которым присутствуют в правой части операторов присваивания или условиях IF в данном цикле, причем два обращения к разным массивам не обязаны быть в одном операторе. Связь создается с типом дуги чтение–чтение (связь $R-R$) по данному циклу с весом $Loop_w * SumB$.

Под $SumB$ понимается совокупное количество байт, которое потребуется передать другим процессорам в случае неудовлетворения конкретной связи с циклом. В худшем случае необходимо передать целиком все измерение массива, отображенное на соответствующий цикл. Количество байт вычисляется из размерности типа используемого массива и номера измерения массива. Данные о размерах массивов всегда известны системе SAPFOR и должны быть получены либо от статического, либо от динамического анализов, либо от пользователя, иначе невозможно построить дерево выравнивания в модели DVMH.

Под $Loop_w$ понимается вес цикла. В зависимости от количества арифметических операций и обращений к массивам в телах циклов тот или иной цикл с меньшим количеством витков может выполняться дольше аналогичного цикла, но с бóльшим количеством витков. Вес цикла оценивается статическим образом либо путем динамического профилирования (получение времени выполнения данного цикла). Вес цикла показывает, сколько раз цикл был выполнен за все время работы программы. Например, если цикл выполняется всего один раз (цикл инициализации), то можно пожертвовать количеством коммуникаций в данном цикле в пользу итерационного цикла, который может выполняться сотни, а то и тысячи раз, где каждый лишний переданный байт будет серьезно сказываться на производительности программы в целом. В случае недостаточности информации для оценки веса цикла система SAPFOR полагает $Loop_w = 1.0$, что означает равенство всех циклов в программе.

В случае добавления дуги с одинаковыми вершинами и атрибутами (коэффициенты выравнивания и направление доступа) происходит увеличение веса данной дуги путем суммирования текущего веса и веса добавляемой дуги. Данное правило позволит выделить наиболее важные связи, что даст возможность уменьшить количество пересылок между процессами при выборе определенной схемы распределения данных.

Для того чтобы отличать дуги по типу связи ($W-W, W-R, R-R$), необходимо расставить приоритеты. Тип связи $W-W$ имеет самый высший приоритет, однако все дуги с типом $W-W$ имеют равный приоритет между собой. Это объясняется тем, что неудовлетворение данной связи приведет к невозможности распараллеливания данного цикла, так как не будет выполнено правило собственных вычислений для связываемых массивов, что в итоге повлечет за собой бóльшие коммуникации (так как все обращения к распределенным массивам на чтение в таком цикле в худшем случае должны быть “покрыты” с помощью доступа к удаленным данным). Дуги с типом связи $W-R$ имеют приоритет над дугами с типом связи $R-R$. Дуги каждого из типов $W-R$ и $R-R$ имеют также равный приоритет между собой.

Чтобы обеспечить соотношение приоритетов для разных типов дуг, был реализован следующий алгоритм выделения дуг по приоритетам. Сначала посчитаем общую сумму всех дуг с типом $R-R$, $S1 = \sum_{[R-R]}$. Затем прибавим число $S1$ к дугам с типом $W-R$. Тем самым мы “выделим” приоритет $W-R$ типа над дугами типа $R-R$, сохранив между тем равный приоритет между схожим типом. Затем вычислим общую сумму весов всех дуг с типами связи $W-R$ и $R-R$, $S2 = S1 + \sum_{[W-R]}$ и добавим теперь число $S2$ ко всем дугам с типом $W-W$. Таким образом, будет выполнено правило приоритета дуг: вес любой дуги с типом $W-W$ будет больше, чем $W-R$ и $R-R$, вес любой дуги с типом $W-R$ будет больше, чем $R-R$. На рис. 1 веса дуг указаны в виде числовых атрибутов, приписанных каждой дуге графа.

3.5. Поиск наилучшего связывания массивов. После того как был построен общий граф массивов, необходимо выбрать то множество дуг, которое будет отражать наилучшие связи между массивами и минимизировать коммуникации между процессорами. Таким образом, необходимо построить усеченный граф массивов (такой граф массивов, который не содержит циклов, порождающих конфликтные ситуации) для последующего создания распределения данных. Конфликты могут быть двух типов. Рассмотрим данные типы конфликтов:

- наличие цикла в графе массивов, для вершин которого нельзя построить единственный вариант выравнивания измерений массивов между собой (где каждая вершина соответствует измерению массива). На рис. 1 а порождающие конфликт данного типа отмечены фиолетовым. Такие конфликты будем называть конфликтами первого типа (1);

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

- присутствует явная или косвенная дуга (через другие дуги графа) между двумя измерениями одного и того же массива. На рис. 1 а порождающие конфликт данного типа отмечены красным. Такие конфликты будем называть конфликтами второго типа (2).

Рассмотрим два подхода, которые позволяют сократить количество дуг в графе массивов и найти совокупность наиболее важных дуг. Первый подход — перебор совокупности наиболее важных дуг в графе. В данном подходе мы выполняем перебор всевозможных наборов дуг и их оценку общего веса. Набор дуг с максимальным весом и будет решением данной задачи. Решение данной задачи можно представить в виде нескольких этапов.

Первый этап — получение всех простых циклов в графе массивов. Простой цикл в графе — это замкнутый цикл без повторного прохода по ребру или посещения вершины дважды, за исключением начальной и конечной вершин. В системе SAPFOR цикл представляет собой набор дуг графа массивов с сохранением веса.

Нахождение всех простых циклов в графе массивов является NP-трудной задачей, поэтому для ограничения поиска по времени и ресурсам в случае больших графов вводятся два параметра, которые позволят ограничить этот поиск: максимальный размер цикла (размером цикла будем называть количество входящих в него дуг), который необходимо добавить в список простых циклов, и максимальная длина просматриваемой цепочки при рекурсивном поиске таких циклов в графе.

Первый параметр используется для ограничения по памяти, второй — по времени поиска всех простых циклов. Можно отметить, что накладываемые ограничения не всегда позволяют найти все простые циклы в графе массивов, а это значит, что алгоритм не всегда может найти точное решение для больших графов массивов. С другой стороны — чем больше размер (длина) цикла, тем больше массивов он связывает между собой. Тем самым для получения наилучшего выравнивания массивов между собой не обязательно находить все простые циклы в графе.

Второй этап — обработка найденных простых циклов. После нахождения всех простых циклов (далее просто циклов) происходит их сортировка по размеру (длине), а также разделение на независимые группы по длине. Затем для каждого цикла выполняется сортировка его дуг по весу и все циклы в каждой группе упорядочиваются по суммарному весу. Данные сортировки позволят наиболее быстрым образом обеспечить поиск и удаление конфликтных дуг в графе массивов.

Для устранения конфликта (1) можно удалить любую из дуг цикла, например с минимальным весом. Для устранения конфликта (2) необходимо удалить такую дугу, чтобы явная или косвенная связь между двумя измерениями массивов, которая выражена дугами графа, пропала. Если есть конфликтные циклы, то запускается процесс устранения конфликтов. Рассмотрим такой подход, который позволяет удалять конфликтные и неконфликтные дуги. После применения данного подхода мы получим усеченный граф массивов, который не содержит циклов.

Каждый цикл характеризуется суммарным весом всех его дуг или просто общим весом. Ранее все циклы были отсортированы с учетом их веса и размерности от самых маленьких до самых больших циклов по размерности (длине), а циклы с одинаковой размерностью были отсортированы по убыванию их веса.

Для устранения конфликтов запускается рекурсивная процедура. Цель данной процедуры — удалить дуги с минимальным суммарным общим весом, что позволит получить наиболее оптимальное с точки зрения обменов между узлами распределение данных на основе построения графа массивов и задания соответствующих весов. В начале процедуры имеем нулевой общий вес удаленных дуг и пустой список удаляемых дуг. Выбираем очередной цикл из списка полученных конфликтных циклов. Пытаемся по очереди удалить каждую из дуг данного цикла и смотрим, что получается:

- если это первая дуга цикла, то удаляем эту дугу, прибавляем вес этой дуги к общему весу всех удаленных дуг и заносим эту дугу в список удаляемых. После удаления данной дуги некоторые циклы из полученного списка перестанут быть циклами и, соответственно, не будут принимать участие в дальнейшем выборе. Далее рекурсивно вызываем эту процедуру. Рекурсивный вызов завершается в том случае, если нет больше циклов с конфликтами. В этом случае получен суммарный вес и список необходимых для удаления дуг;
- если эта дуга цикла не первая, то у нас уже имеется общий вес и список удаляемых дуг на каком-то конкретном уровне рекурсивной вложенности вызова рассматриваемой процедуры. Также мы имеем текущий суммарный вес на текущем уровне рекурсивной вложенности. Если сумма веса очередной удаляемой дуги и текущего общего суммарного веса больше, чем найденный наименьший общий вес

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

удаляемых дуг, то рекурсивный вызов удаления этой дуги не выполняется, так как удаление этой дуги повлечет за собой увеличение общего веса всех удаляемых дуг (таким образом выполняется отсечение перебора всех вариантов наборов дуг для удаления). Если мы завершили рекурсивный вызов и получили меньший вес, чем был найден до этого, то корректируются общий вес и соответствующий список дуг для удаления.

После того как мы получили список дуг для удаления, происходит формирование усеченного графа массивов, такого графа, который не содержит циклов. Затем необходимо проверить, не возникнут ли конфликты второго типа в усеченном графе, или нет ли таких путей в графе, которые косвенно (через другие массивы и соответствующие им дуги) связывают измерения одного и того же массива. Данная конфликтная ситуация не образует цикл, но требует разрешения.

Для этого необходимо для каждого массива для всех уникальных комбинаций пар его измерений добавить фиктивную дугу между этими измерениями с очень большим весом (например, бóльшим, чем сумма всех весов в графе) и повторить весь алгоритм поиска простых циклов и удаления конфликтов. Если мы нашли конфликтный цикл, то мы удаляем дугу. Из-за особенности алгоритма (применение сортировок и удаление дуг с минимальным совокупным весом) будет выбрана не фиктивная дуга, а существующая дуга в графе. Таким образом, будут разрешены все конфликты второго типа.

Стоит отметить, что после такой операции по данному графу массивов не всегда можно построить выравнивание всех массивов между собой, особенно если в графе было много конфликтов или были применены ограничения на поиск простых циклов, что является минусом данного алгоритма. Оценка сложности данного алгоритма является экспоненциальной в зависимости от количества вершин в графе, что накладывает ограничения на его применимость на больших программах. Данный алгоритм может быть использован на сравнительно небольших программах, где количество узлов графа массивов не более 10.

Альтернативным алгоритмом поиска наиболее важных дуг является модифицированный алгоритм поиска минимального остовного дерева. Минимальное остовное дерево — такое дерево, которое является максимальным по включению ребер подграфом, не имеющее циклов, и в котором сумма весов ребер минимальна. Если исходный граф связный, то будет построено остовное дерево, если же в исходном графе несколько несвязных компонент, то результатом будет остовный лес.

В нашей задаче требуется найти набор дуг с наибольшим весом. Таким образом, без изменения общности алгоритма поиска минимального остовного дерева можно искать максимальное остовное дерево — такое дерево, в котором сумма весов ребер максимальна. Была использована самая простая реализация — алгоритм Дейкстры–Прима.

Для программы из листинга 1 максимальное остовное дерево для графа массивов приведено на рис. 1 b. Для того чтобы устранить конфликты второго типа, перед построением максимального остовного дерева в граф массивов были добавлены фиктивные дуги (отмечены пунктирными линиями на рисунке) между всеми различными измерениями одного и того же массива. В качестве весов для добавленных дуг была использована сумма весов всех остальных дуг в графе массивов. Это гарантирует, что фиктивные дуги войдут в состав максимального остовного дерева, а для разрешения конфликтов второго типа, при их наличии, будут удалены какие-либо дуги графа, отличные от фиктивных.

Недостатком данного алгоритма является тот факт, что решение получается не самое лучшее, как в случае полного перебора, так как не выполняется перебор всех цепочек дуг и их сравнение между собой. Главное достоинство такого подхода заключается в линейной оценке его сложности в зависимости от количества вершин в графе. Данное достоинство вместе с возможностью распараллелить данный алгоритм делает возможным его использование на любой программе с любым количеством массивов.

3.6. Создание вариантов распределения данных. После того как был получен усеченный граф (далее граф массивов), который связывает измерения массивов в соответствии с их использованием в циклах программы, можно построить варианты (схемы) распределения данных.

Так как граф может быть несвязным, в нем ищутся все деревья, которые связывают массивы друг с другом. После такого поиска мы знаем, сколько поддеревьев у нас есть и какие массивы в эти поддеревья входят. Для каждого дерева создается свой шаблон в DVMH-модели — DVMH TEMPLATE, который и будет распределяться с помощью директивы DISTRIBUTE и на который будут выровнены все массивы данного дерева. Шаблон представляет собой виртуальный массив, под который не отводится память в программе.

Шаблон создается по массиву с наибольшей размерностью, а среди одинаковых массивов одной размерности выбирается тот, который занимает больше памяти. После того как создан шаблон и найден

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

массив, по которому строится этот шаблон, в граф добавляются дуги, связывающие измерения этого массива и измерения этого шаблона с весом 1.0 и типом связи $R - R$. Таким образом, в графе массивов появляется шаблон, до которого можно "добраться" по связям между массивами и узнать выравнивание на него.

Так как с шаблоном связан только один из массивов, необходимо уметь вычислять связь с шаблоном и для других массивов. Наибольшая сложность, которая может возникнуть при вычислении таких связей, — некратные коэффициенты при выравнивании на шаблон. Такие коэффициенты могут возникать в том случае, когда измерение одного массива требуется распределить, например, с раздвижкой $3 * i$, а измерение второго массива требуется распределить на то же измерение шаблона с раздвижкой $2 * i$. Таким образом, требуется вычислить наименьшее общее кратное и изменить соответствующие атрибуты в графе.

Варианты распределения данных создаются для каждого шаблона независимо, количество вариантов для каждого шаблона будет равно $2^{d(T_i)}$, а общее их количество — $\sum_{i=1}^{K} 2^{d(T_i)}$, где $d(T_i)$ — размерность шаблона (каждое измерение считается распределенным либо размноженным), K — количество построенных шаблонов.

Правила выравнивания массивов на шаблон в модели DVMH не зависят от выбранного в дальнейшем распределения этих шаблонов. Для каждого массива из поддерева, который не является шаблоном, выполняется поиск связи его измерений с шаблоном в этом поддереве. Поиск связи для конкретного измерения массива запускается в том случае, если это измерение присутствует в графе массивов. В результате поиска может быть не найдено связи с шаблоном по конкретному измерению, такое измерение будет размножено (указана * в спецификации ALIGN).

3.7. Создание директив распределения вычислений. Для создания директивы распределения вычислений необходимо найти массив из графа массивов, на который будет отображаться пространство витков потенциально параллельного цикла. Если в рассматриваемом цикле присутствует запись всего в один массив — он и будет выбран. Если массивов на запись несколько, то выбор происходит прежде всего среди массивов, которые участвуют в спецификации ACROSS, если таковая имеется. В модели DVMH требуется, чтобы цикл был отображен на массив, который участвует в спецификации ACROSS. Данная спецификация позволяет организовать конвейерное выполнение циклов с регулярными зависимостями по данным, необходимость ее задания для цикла определяется системой SAPFOR автоматически на основе результатов анализа программы.

В результате алгоритма поиска наилучшего выравнивания данных, который описан выше, был получен усеченный граф массивов. Конфликтные ситуации первого и второго типов были разрешены с помощью удаления соответствующих ребер этого графа. При этом интересы тех или иных циклов могли быть не учтены, что может привести к невозможности создания директивы распределения вычислений для этих циклов.

Если цикл не содержит ограничений на распараллеливание (считается потенциально параллельным), то директива "привязывается" к соответствующему циклу в дереве циклов. Для директивы заполняется информация о массиве, на который требуется отобразить вычисления, правила отображения, а также сохраняются свойства цикла, которые были получены в результате его анализа системой SAPFOR и должны быть описаны в параллельной программе в виде спецификаций DVMH языков: приватные и редукционные переменные, информация о регулярных зависимостях по данным, теневые грани массивов и элементы массивов, которые должны быть отдельно получены с удаленного процессора с помощью спецификации REMOTE_ACCESS.

Далее выполняется объединение полученных директив для каждого из циклов, если имеет место тесная вложенность. Если цикл не тесно вложенный, то запускается преобразование, которое позволяет на основе информации, полученной от статического и динамического анализов, произвести объединение циклов и сделать их тесно вложенными (внесение инварианта цикла). Данное преобразование выполняется "на лету" и позволяет устранить не тесную вложенность, если это возможно, без потери результатов анализа и построенных структур.

В итоге самый верхний цикл будет содержать объединенную информацию от всех тесно вложенных циклов, следующий по уровню вложенности цикл будет содержать объединенную информацию от всех тесно вложенных циклов, которые расположены ниже по уровню вложенности и т.д. Такой подход позволяет выбирать разные циклы для распараллеливания программы в зависимости от выбранного варианта распределения данных.

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

В зависимости от выбранного варианта распределения данных (шаблонов) будут выбраны соответствующие директивы распределения вычислений, а также созданы дополнительные директивы перераспределения данных и доступа к удаленным данным, если это потребуется.

4. Вычислительные эксперименты. В данном разделе исследуется эффективность программ в модели DVMH, получаемых в результате применения описанных алгоритмов распределения данных и вычислений. Нами было выполнено сравнение параллельных версий, полученных с помощью системы SAPFOR для вычислительных приложений BT, CG и EP из пакета NAS Parallel Benchmarks [20], с MPI-версиями данных программ, написанными их разработчиками вручную.

NAS Parallel Benchmarks — комплекс тестов, позволяющий оценивать производительность суперкомпьютеров. Тесты разработаны и поддерживаются в NASA Advanced Supercomputing (NAS) Division (ранее NASA Numerical Aerodynamic Simulation Program), расположенном в NASA Ames Research Center. Версия 3.3 пакета NPB включает в себя 11 тестов. Существует последовательная реализация этих тестов, а также параллельная с использованием MPI.

Программа BT (Block Tridiagonal) решает синтетическую систему нелинейных дифференциальных уравнений в частных производных (трехмерная система уравнений Навье–Стокса для сжимаемой жидкости или газа), используя блочную трехдиагональную схему с методом переменных направлений. Данная программа содержит большое количество вычислений на байт данных. Программа EP (Embarrassingly Parallel) вычисляет интеграл методом Монте-Карло, в первую очередь предназначена для измерения первичной вычислительной производительности плавающей арифметики. Данная программа содержит большое количество независимых вычислений и почти полное отсутствие обращений к оперативной памяти. Программа CG (Conjugate Gradient) вычисляет приближение к наименьшему собственному значению большой разреженной симметричной положительно определенной матрицы с использованием “inverse iteration” вместе с методом сопряженных градиентов в качестве подпрограммы для решения СЛАУ. В данной программе — малая интенсивность вычислений на байт данных, а также косвенная индексация к памяти.

Выбранные программы показывают разный характер вычислительной способности и разный шаблон доступа к оперативной памяти, что позволяет оценить качество распараллеливания в разных сценариях.

Исследование эффективности было выполнено на суперкомпьютере K10 [30], состоящем из процессоров Intel Xeon E5-2660 и графических ускорителей NVIDIA Tesla M2090. Для компиляции использовался установленный на K10 Intel MPI версии 13 с опциями -O2, также CUDA Toolkit версии 9.0. Каждый узел содержит два восьмиядерных процессора (CPU), связанных посредством общей памяти (архитектура NUMA), и три графических ускорителя (GPU). Для вычислительных экспериментов были использованы максимально возможные ресурсы только процессоров одного, двух и девяти узлов (графические ускорители не использовались). Время выполнения MPI программ и DVMH программ, полученных с помощью SAPFOR с использованием языков FDVMH и CDVMH, приведены в табл. 1.

Время выполнения оригинальных версий, написанных разработчиками пакета NAS Parallel Benchmark, приведено в столбцах, названных MPI.

Изначально рассматриваемые последовательные программы были написаны только на Fortran, поэтому потребовалось выполнить перевод рассматриваемых программ на язык C вручную. Чтобы получить автоматически распараллеливаемые версии программ с помощью системы SAPFOR, были выполнены их предварительные преобразования (подстановка функций, объединение циклов, сужение размерности приватных массивов и др.) [19]. Столбцы, названные DVM, показывают время выполнения этих парал-

Таблица 1. Время выполнения в секундах программ на языке Fortran и C, NPB 3.3 класс C

Table 1. Execution time in seconds for Fortran and C programs, NPB 3.3 class C

	Fortran						C					
	BT		CG		EP		BT		CG		EP	
	MPI	DVM	MPI	DVM	MPI	DVM	MPI	DVM	MPI	DVM	MPI	DVM
1 узел	100	80	21.7	25.9	27.4	22.6	123.6	97.1	35.9	25.8	30.6	28
4 узла	34	24.8	11.04	47	7.09	5.67	34.7	30	10.2	47	7.85	7
9 узлов	16.26	14.9	7.51	60.1	3.25	2.52	17.10	16.57	7.05	60.7	3.43	3.13

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

лельных версий программ, разработанных на языках FDVMH и CDVMH. 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.

Программы BT и EP, полученные с помощью системы SAPFOR, показывают практически схожие ускорения, что и MPI программы, написанные вручную разработчиками тестов. Но программа CG, начиная с 16 процессов (1 узел) и более, замедляется по отношению к MPI программе. Это связано прежде всего с тем, что основная доля времени приходится на умножение разреженной матрицы на вектор, что приводит к косвенной индексации. Данная особенность не может быть эффективно распараллелена в текущей модели на регулярных сетках. Для этого требуется модификация системы SAPFOR для использования нового расширения DVMH-модели для нерегулярных сеток. При распараллеливании в текущей модели системе SAPFOR постоянно требуется выполнять одну коллективную операцию по пересылке удаленных данных между всеми процессами, что приводит к замедлению на их большом количестве. Если использовать графические ускорители, то 16 процессов будет достаточно для достижения высокой производительности, а пересылки не будут так сильно сказываться.

5. Заключение. В статье был рассмотрен подход к автоматизированному распараллеливанию программ для кластеров с помощью системы SAPFOR. Работа полученных параллельных программ была продемонстрирована на примере некоторых приложений 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+CUDA, при разработке параллельных программ для гетерогенных кластеров.

Основная сложность при распараллеливании программ на кластер заключается в минимизации накладных расходов, вызванных коммуникационными обменами между вычислительными узлами. В системе SAPFOR был реализован алгоритм распределения данных, учитывающий глобальные связи между массивами программы. Такие связи порождаются совместным использованием разных массивов в одном цикле и необходимостью соблюдения правила собственных вычислений при распараллеливании циклов в модели DVMH. Выбор между противоречивыми связями выполняется на основе оценки потенциально возможных коммуникаций в случае несоблюдения одной из связей.

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, используя соответствующие директивы для указания недостающих свойств программы, либо выполнить с помощью системы SAPFOR необходимые преобразования, которые не приводят к снижению производительности и при этом повышают доступный уровень параллелизма. Можно отметить, что преобразования производятся на уровне исходного кода и не требуют детального изучения параллельных архитектур и языков параллельного программирования.

В сочетании с ранее выполненными работами [13, 19] система SAPFOR может выполнять распараллеливание в модели DVMH на вычислительные системы в разной конфигурации, используя узлы кластера, а также графические ускорители и многоядерные процессоры внутри узла.

Таким образом, системы SAPFOR и DVM могут значительно сократить усилия, необходимые для распараллеливания программ, и позволяют задействовать все доступные внутри узла ресурсы (многоядерные процессоры и графические ускорители). Мы надеемся, что данный подход должен помочь эффективной разработке и оптимизации масштабируемых программ для суперкомпьютеров.

Список литературы

1. *Штейнберг Б.Я., Штейнберг О.Б.* Преобразования программ — фундаментальная основа создания оптимизирующих распараллеливающих компиляторов // Программные системы: теория и приложения. 2021. **12**, № 1. 21–113. doi 10.25209/2079-3316-2021-12-1-21-113.
2. *Czarnul P., Proficz J., Drypczewski K.* Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems // Scientific Programming. 2020. **2020**. Article ID 4176794. doi 10.1155/2020/4176794.

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

3. SYCL Academy. https://sycl.tech. Cited December 8, 2022.
4. Celerity. High-level C++ for accelerator clusters. https://celerity.github.io. Cited December 8, 2022.
5. *Murai H., Nakao M., Shimosaka T., et al.* XcalableACC — a directive-based language extension for accelerated parallel computing // Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis, New Orleans, USA, November 16–21, 2014. https://pro-env.riken.jp/data/2014/post266s2-file3.pdf. Cited December 8, 2022.
6. *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.
7. *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. Серия: Вычислительная математика и информатика. 2012. № 18, вып. 12. 82–92.
8. *Бахтин В.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н.* Методы динамической настройки DVMH-программ на кластеры с ускорителями // Суперкомпьютерные дни в России: Труды международной конференции. Москва, 28–29 сентября 2015 г. М.: Изд-во МГУ, 2015. 257–268.
9. *Hwu W.-M., Ryoo S., Ueng S.-Z., Kelm J.H., Gelado I., Stone S.S., Kidd R.E., Baghsorkhi S.S., Mahesri A.A., Tsao S.C., Navarro N., Lumetta S.S., Frank M.I., Patel S.J.* Implicitly parallel programming models for thousand-core microprocessors // Proceedings of the 44th Annual Design Automation Conference. New York: ACM Press, 2007. 754–759. doi 10.1145/1278480.1278669.
10. *Baghdadi R., Beaugnon U., Cohen A., et al.* PENCIL: a platform-neutral compute intermediate language for accelerator programming // Proc. Int. Conf. on Parallel Architecture and Compilation Techniques, San Francisco, USA, October 18–21, 2015. Washington DC: IEEE Press, 2015. 138–149. doi 10.1109/PACT.2015.17.
11. *Kruse M.* Introducing Molly: distributed memory parallelization with LLVM // arXiv preprint: 1409.2088v1[cs.PL]. Ithaca: Cornell Univ. Library, 2014. https://doi.org/10.48550/arXiv.1409.2088. Cited December 8, 2022.
12. *Vandierendonck H., Rul S., De Bosschere K.* The Paralax infrastructure: automatic parallelization with a helping hand // Proc. Int. Conf. on Parallel Architectures and Compilation Techniques, Vienna, Austria, September 11–15, 2010. New York: ACM Press, 2010. 389–400. doi 10.1145/1854273.1854322.
13. *Катаев Н.А., Колганов А.С.* Дополнительное распараллеливание MPI программ с помощью системы SAPFOR // Вычислительные методы и программирование. 2021. **22**, № 4. 239–251. doi 10.26089/NumMet.v22r415.
14. *Клинов М.С., Крюков В.А.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник ННГУ. 2009. № 2. 128–134.
15. *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 // Вестник ННГУ. 2012. № 5(2). 242–245.
16. *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.
17. *Kataev N., Smirnov A., Zhukov A.* Dynamic data-dependence analysis in SAPFOR // CEUR Workshop Proc. Vol. 2543. 2020. 199–208. doi 10.20948/abrau-2019-62.
18. *Kataev N.* Interactive parallelization of C programs in SAPFOR // CEUR Workshop Proc. Vol. 2784. 2020. 139–148.
19. *Kataev N.* LLVM based parallelization of C programs for GPU // Communications in Computer and Information Science. Vol. 1331. Cham: Springer, 2020. 436–448. doi 10.1007/978-3-030-64616-5_38.
20. NAS Parallel Benchmarks. https://www.nas.nasa.gov/publications/npb.html. Cited December 8, 2022.
21. *Amarasinghe S.P., Lam M.S.* Communication optimization and code generation for distributed memory machines // ACM SIGPLAN Not. 1993. **28**, N 6. 126–138. doi 10.1145/155090.155102.
22. *Zima H.P., Bast H.-J., Gerndt M.* SUPERB: a tool for semi-automatic MIMD/SIMD parallelization // Parallel Comput. 1988. **6**, N 1. 1–18. doi 10.1016/0167-8191(88)90002-6.
23. *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.
24. *Lattner C., Adve V.* LLVM: a compilation framework for lifelong program analysis and transformation // Proc. Int. Symp. on Code Generation and Optimization (CGO'04). Palo Alto, USA, March 20–24, 2004. doi 10.1109/CGO.2004.1281665.
25. *Гервич Л.Р., Кравченко Е.Н., Штейнберг Б.Я., Юрушкин М.В.* Автоматизация распараллеливания программ с блочным размещением данных // Сиб. журн. вычисл. математики. 2015. **18**, № 1. 41–53.

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

26. *Bondhugula U.* Compiling affine loop nests for distributed-memory parallel architectures // Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis, Denver, USA, November 17–22, 2013. doi 10.1145/2503210.2503289.
27. *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 10.1145/1375581.1375595.
28. *Banerjee P., Chandy J.A., Gupta M., et al.* The paradigm compiler for distributed-memory multicomputers // Computer. 1995. **28**, N 10. 37–47. doi 10.1109/2.467577.
29. *Sokolinsky L.B.* BSF: a parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems // Journal of Parallel and Distributed Computing. 2021. **149**. 193–206. doi 10.1016/j.jpdc.2020.12.009.
30. Гибридный вычислительный кластер К-10. https://www.kiam.ru/MVS/resourses/k10.html. Дата обращения: 8 декабря 2022.

Поступила в редакцию
1 октября 2022 г.

Принята к публикации
1 ноября 2022 г.

Информация об авторах

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

References

1. B. Ya. Steinberg and O. B. Steinberg, "Program Transformations as the Base for Optimizing Parallelizing Compilers," Program Systems: Theory and Applications **12**, Issue 1, 21–113 (2021). doi 10.25209/2079-3316-2021-12-1-21-113.
2. P. Czarnul, J. Proficz, and K. Drypczewski, "Survey of Methodologies, Approaches, and Challenges in Parallel Programming Using High-Performance Computing Systems," Sci. Program. **2020**, Article ID 4176794 (2020). doi 10.1155/2020/4176794.
3. SYCL Academy. https://sycl.tech. Cited December 8, 2022.
4. Celerity. High-level C++ for Accelerator Clusters. https://celerity.github.io. Cited December 8, 2022.
5. H. Murai, M. Nakao, T. Shimosaka, et al., "XcalableACC — a Directive-Based Language Extension for Accelerated Parallel Computing," in *Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis, New Orleans, USA, November 16–21, 2014*. https://pro-env.riken.jp/data/2014/post266s2-file3.pdf. Cited December 8, 2022.
6. N. A. Konovalov, V. A. Krukov, S. N. Mikhajlov, and A. A. Pogrebtsov, "Fortran DVM: a Language for Portable Parallel Program Development," Program. Comput. Softw. **21** (1), 35–38 (1995).
7. 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. Program. No. 12, 82–92 (2012).
8. V. A. Bakhtin, A. S. Kolganov, V. A. 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. 257–268.
9. W.-M. Hwu, S. Ryoo, S.-Z. Ueng, et al., "Implicitly Parallel Programming Models for Thousand-Core Microprocessors," in *Proc. 44th Annual Design Automation Conference, San Diego, USA, June 4–8, 2007* (ACM Press, New York, 2007), pp. 754–759. doi 10.1145/1278480.1278669.
10. R. Baghdadi, U. Beaugnon, A. Cohen, et al, "PENCIL: A Platform-Neutral Compute Intermediate Language for Accelerator Programming," in *Proc. Int. Conf. on Parallel Architecture and Compilation Techniques, San Francisco, USA, October 18–21, 2015* (IEEE Press, Washington, DC, 2015), pp. 138–149. doi 10.1109/PACT.2015.17.
11. M. Kruse, *Introducing Molly: Distributed Memory Parallelization with LLVM*, arXiv preprint: 1409.2088v1[cs.PL] (Cornell Univ. Library, Ithaca, 2014), https://doi.org/10.48550/arXiv.1409.2088. Cited December 8, 2022.

2022, **23** (4), 379–394. doi 10.26089/NumMet.v23r424

12. H. Vandierendonck, S. Rul, and K. De Bosschere, "The Paralax Infrastructure: Automatic Parallelization with a Helping Hand," in *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques, Vienna, Austria, September 11–15, 2010* (ACM Press, New York, 2010), pp. 389–400. doi 10.1145/1854273.1854322.
13. N. A. Kataev and A. S. Kolganov, "Additional Parallelization of Existing MPI Programs Using SAPFOR," Numer. Methods Program. **22** (4), 239–251 (2021). doi 10.26089/NumMet.v22r415.
14. M. S. Klinov and V. A. Krukov, "Automatic Parallelization of Fortran Programs. Mapping to Cluster," Vestn. Lobachevskii Univ. Nizhni Novgorod, No. 2, 128–134 (2009).
15. 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).
16. N. Kataev, "Application of the LLVM Compiler Infrastructure to the Program Analysis in SAPFOR," in *Communications in Computer and Information Science* (Springer, Cham, 2018), Vol. 965, pp. 487–499. doi 10.1007/978-3-030-05807-4_41.
17. N. Kataev, A. Smirnov, and A. Zhukov, "Dynamic Data-Dependence Analysis in SAPFOR," CEUR Workshop Proc. Vol. 2543 (2020), pp. 199–208. doi 10.20948/abrau-2019-62.
18. N. Kataev, "Interactive Parallelization of C Programs in SAPFOR," CEUR Workshop Proc. Vol. 2784 (2020), pp. 139–148.
19. 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. doi 10.1007/978-3-030-64616-5_38.
20. NAS Parallel Benchmarks. https://www.nas.nasa.gov/publications/npb.html. Cited December 8, 2022.
21. S. P. Amarasinghe and M. S. Lam, "Communication Optimization and Code Generation for Distributed Memory Machines," ACM SIGPLAN Not. **28** (6), 126–138 (1993). doi 10.1145/155090.155102.
22. H. P. Zima, H.-J. Bast, and M. Gerndt, "SUPERB: A Tool for Semi-Automatic MIMD/SIMD Parallelization," Parallel Comput. **6** (1), 1–18 (1988). doi 10.1016/0167-8191(88)90002-6.
23. 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.
24. C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation," in *Proc. Int. Symp. on Code Generation and Optimization (CGO'04), Palo Alto, USA, March 20–24, 2004*. doi 10.1109/CGO.2004.1281665.
25. L. R. Gervich, E. N. Kravchenko, B. Ya. Shteinberg, and M. V. Yurushkin, "Automatic Program Parallelization with a Block Data Distribution," Numer. Analysis Appl. **8** (1), 35–45 (2015). doi 10.1134/S1995423915010048.
26. U. Bondhugula, "Compiling Affine Loop Nests for Distributed-Memory Parallel Architectures," in *Proc. Int. Conf. on High Performance Computing, Networking, Storage and Analysis, Denver, USA, November 17–22, 2013*. doi 10.1145/2503210.2503289.
27. 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 10.1145/1375581.1375595.
28. P. Banerjee, J. A. Chandy, M. Gupta, et al., "The Paradigm Compiler for Distributed-Memory Multicomputers," Computer **28** (10), 37–47 (1995). doi 10.1109/2.467577.
29. L. B. Sokolinsky, "BSF: A Parallel Computation Model for Scalability Estimation of Iterative Numerical Algorithms on Cluster Computing Systems," J. Parall. Distrib. Comput. **149**, 193–206 (2021). doi 10.1016/j.jpdc.2020.12.009.
30. Heterogeneous cluster K10. https://www.kiam.ru/MVS/resourses/k10.html. Cited December 8, 2022.

Received
October 1, 2022

Accepted for publication
November 1, 2022

Information about the authors

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

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