УДК 004.272, 004.4'242 doi 10.26089/NumMet.v21r432

ОПЫТ ПРИМЕНЕНИЯ МЕХАНИЗМА ОБЛАСТЕЙ ДЛЯ ПОЭТАПНОГО РАСПАРАЛЛЕЛИВАНИЯ ПРОГРАММНЫХ КОМПЛЕКСОВ С ПОМОЩЬЮ СИСТЕМЫ SAPFOR

А. С. Колганов[1]

Одна из основных сложностей разработки параллельной программы для кластера — необходимость принятия глобальных решений по распределению данных и вычислений с учетом свойств всей программы, а затем выполнения кропотливой работы по модификации программы и ее отладки. Большой объем программного кода, а также многомодульность, многовариантность и многоязыковость, затрудняют принятие решений по согласованному распределению данных и вычислений. Опыт использования предыдущей системы САПФОР показал, что при распараллеливании на кластер больших программ и программных комплексов необходимо уметь распараллеливать их постепенно, начиная с наиболее времяемких фрагментов и постепенно добавляя новые фрагменты, пока не достигнем желаемого уровня эффективности параллельной программы. С этой целью предыдущая система была полностью переработана, и на ее основе была создана новая система SAPFOR (System FOR Automated Parallelization). В данной статье будет рассмотрен опыт применения метода частичного распараллеливания, идея которого заключается в том, что распараллеливанию подвергается не вся программа целиком, а ее части (области распараллеливания) — в них заводятся дополнительные экземпляры требуемых данных, производится распределение этих данных и соответствующих им вычислений.

Ключевые слова: SAPFOR (System FOR Automated Parallelization), автоматизация распараллеливания, параллельные вычисления, DVM (Distributed Virtual Memory), инкрементальное распараллеливание для кластера.

1. Введение. Развитие высокопроизводительных вычислительных систем в настоящее время в основном происходит за счет создания разнообразных параллельных архитектур — многоядерных процессоров на x86 и Power архитектурах, ARM процессоров, графических процессоров, Intel Xeon Phi процессоров, ПЛИС и т.д. Несмотря на такое разнообразие архитектур, применение вычислителей с одним типом архитектуры оказывается недостаточным для эффективного решения многих задач из класса HPC или обработки больших данных (BigData), поэтому существует большое количество вычислительных кластеров, объединяющих параллельные вычислители различных архитектур в высокопроизводительную вычислительную систему.

Большое распространение получили гибридные кластеры, в узлах которых содержатся вычислители разной архитектуры, например центральные процессоры и графические ускорители. Гибридная архитектура кластера вместе с распределенной памятью сильно усложняет разработку параллельных программ или отображение существующих последовательных программ в эффективные параллельные. Разработка программ для высокопроизводительных кластеров различной архитектуры продолжает оставаться исключительно сложным делом, доступным довольно узкому кругу специалистов. Основная причина — это низкий уровень современной технологии автоматизации разработки параллельных программ.

В настоящее время практически все параллельные программы для гибридных кластеров разрабатываются с использованием низкоуровневых средств передачи сообщений (например, MPI или SHMEM), а также с использованием некоторых средств параллельного программирования на общей памяти для задействования нескольких ядер центрального процессора (например, OpenMP, pthreads, TBB) или графического процессора (CUDA [1], OpenACC). Такие гибридные программы трудно разрабатывать, сопровождать и повторно использовать при создании новых программ.

[1] Институт прикладной математики им. М. В. Келдыша РАН (ИПМ РАН), Миусская пл., 4, 125047, Москва; науч. сотр., e-mail: alexander.k.s@mail.ru

Для решения определенного класса задач имеются специализированные библиотеки, которые упрощают процесс написания программ. В других случаях приходится использовать языки параллельного программирования. Но трудности возникают не только при написании программ на языках параллельного программирования, но и при отладке таких программ. Очень часто разработка параллельной программы начинается с написания и отладки последовательной. Процесс распараллеливания отлаженной последовательной программы целесообразно максимально автоматизировать, а в идеале — осуществлять полностью автоматически, без участия программиста. Проблема автоматического распараллеливания исследуется достаточно давно. Для систем на общей памяти существует достаточно много инструментов, позволяющих в той или иной степени преобразовать последовательную программу в параллельную автоматически. Среди таких инструментов можно отметить Polaris, CAPO, WPP, SUIF, VAST/Parallel, OSCAR, ParallelWare, Intel Parallel Studio XE. К примеру, Intel предоставляет достаточно мощные средства для анализа программы с целью ее последующего распараллеливания на многоядерные процессоры с помощью OpenMP.

Для распараллеливания программы на общей памяти требуется распределить на ядра процессора только те вычисления, которые, в основном, сосредоточены в циклах. В отличие от систем с общей памятью, на системах с распределенной памятью необходимо произвести не только распределение вычислений, но и распределение данных, которое должно быть согласовано с распределением вычислений. Помимо этого, необходимо обеспечить на каждом процессоре доступ к удаленным данным — данным, которые расположены на других процессорах.

Для обеспечения эффективного доступа к удаленным данным требуется определить те данные, которые должны пересылаться с одного процессора на другой. Для этого требуется производить анализ индексных выражений не только внутри одного гнезда цикла (как в случае распараллеливания на общую память), но и между разными циклами для оптимизации коммуникационных обменов (группировки обменов для уменьшения количества пересылок данных и совмещения обменов с вычислениями). На текущий момент можно выделить следующие понятия в области распараллеливания программ [2]:

- **Распараллеливание программ** — процесс отображения последовательной программы пользователем на параллельную архитектуру, состоящий в ее преобразовании для эффективного выполнения на параллельной вычислительной системе и заключающийся в переписывании программы на специальный язык (например, MPI, CUDA, OpenCL) или во вставке специальной разметки (например, OpenMP, OpenACC, DVM [3]);
- **Автоматическое распараллеливание** — процесс отображения последовательной программы компиляторами (например, Intel, PGI) на параллельную архитектуру, состоящий в автоматическом ее преобразовании без участия пользователя для эффективного выполнения на параллельной вычислительной системе;
- **Автоматизированное распараллеливание** — процесс отображения последовательной программы на параллельную архитектуру, состоящий в ее преобразовании, в котором пользователь принимает активное участие. Автоматизированное распараллеливание может включать в себя процесс автоматического распараллеливания.

Последний подход является наиболее перспективным в силу практической невозможности преобразования любой последовательной программы в эффективную параллельную без участия пользователя. Для автоматизации распараллеливания программ были созданы высокоуровневые языки параллельного программирования, такие как HPF, OpenMP-языки, DVM-языки [3], CoArray Fortran, UPC, Titanium, Chapel, X10, Fortress, а также создавались системы автоматизации распараллеливания программ, такие как CAPTools/Parawise, FORGE Magic/DM, BERT77, ParalWare Trainer, Appolo, САПФОР [4], которые используют для отображения на параллельные вычислительные системы как языки низкого уровня, так и языки высокого уровня.

Несмотря на разнообразие различных систем автоматического или автоматизированного распараллеливания, используемые решения в каждой из них для отображения последовательной программы в параллельную одинаковы. Все системы используют анализ и/или преобразование абсолютно всей исходной последовательной программы. В случае отображения на общую память отказаться от распараллеливания какой-то части программы менее болезненно, чем в случае отображения на кластер. А в некоторых случаях невозможность распараллелить какую-то часть программы при отображении на кластер приводит к отказу от распараллеливания всей программы.

Ясно, что в случае распараллеливания больших программных комплексов статический и динамический анализ всего кода являются трудоемкой задачей [5, 6]. И зачастую для частичного распараллеливания полный анализ кода не требуется. Но на данный момент не существует автоматических или автоматизированных систем распараллеливания, которые позволяют отобразить на гибридный кластер только часть большой последовательной программы.

Для обеспечения возможности постепенного (инкрементального) распараллеливания на кластер в системе SAPFOR было введено понятие области распараллеливания [5, 6, 7]. Это позволяет последовательно переходить от рассмотрения отдельных небольших областей к более крупным областям, вплоть до целой программы, сохраняя при этом преемственность ранее принятых решений по распараллеливанию отдельных областей и уточняя их при необходимости. Области распараллеливания могут быть построены автоматически на основе данных о времени выполнения, полученных с помощью профилирования последовательной программы, либо заданы вручную пользователем. Области отражают те участки кода исходной программы, которые будут рассматриваться системой SAPFOR.

2. Обзор существующих решений. Высокопроизводительные кластеры появились достаточно давно. Но тенденции построения суперкомпьютеров в настоящее время почти не изменились. Для того чтобы достичь большой вычислительной мощности, используется большое количество узлов, объединенных между собой коммуникационной сетью. Каждый из узлов может использовать как процессоры общего назначения, так и процессоры иной архитектуры, например графические процессоры или Intel Xeon Phi. Использование различных архитектурных решений в узле кластера обусловлено, прежде всего, более высокой энергоэффективностью. Наращивание количества узлов в какой-то момент станет невозможным и экономически нецелесообразным. Поэтому существует некоторый баланс между количеством узлов кластера и тем, из чего состоит вычислительный узел кластера.

Для использования таких больших вычислительных мощностей (порядка нескольких десятков петафлопс) требуется не только распараллелить вычисления в узле кластера, но и эффективно задействовать множество узлов. Тем самым разработка программ для кластеров оказывается значительно сложнее, чем, например, разработка программ для общей памяти с использованием OpenMP или OpenACC расширений. Программист должен распределить данные между процессорами, а программу представить в виде совокупности процессов, взаимодействующих между собой посредством обмена сообщениями.

Ввиду такой сложности было бы естественным создание инструмента, который будет автоматически преобразовывать последовательную программу в параллельную программу для кластера. Известно, что для векторных процессоров подобные средства широко и успешно использовались и используются до сих пор. Однако автоматическое распараллеливание для кластеров гораздо сложнее по следующим причинам: требуется минимизация коммуникационных затрат для достижения эффективности распараллеливания; требуется распределить не только вычисления, но и данные, а также обеспечить на каждом процессоре доступ к удаленным данным, то есть к таким данным, которые расположены на других процессорах; распределение вычислений и данных должно быть произведено согласованно. Несогласованность распределения вычислений и данных приведет к значительному увеличению времени коммуникаций.

Распараллеливание существующей последовательной программы распадается на две подзадачи — анализ последовательной программы и преобразование данной программы в параллельную. Автоматизировать можно каждую из этих подзадач. Попытки полностью автоматизировать обе подзадачи для параллельных вычислительных систем с распределенной памятью, проведенные в 90-х годах [8], привели к пониманию того, что полностью автоматическое распараллеливание для таких вычислительных систем реальных производственных программ возможно только в очень редких случаях. Поэтому исследования в области автоматического распараллеливания для параллельных вычислительных систем с распределенной памятью практически были прекращены, а результаты тех исследований, которые были продолжены, оказались неудачными — не удалось получить большие ускорения при выполнении на кластере больших производственных программ.

Как уже было отмечено, автоматизация распараллеливания ведется в двух направлениях — создание высокоуровневых языков параллельного программирования и создание систем автоматизированного распараллеливания, например BERT77 [9], Parawise [10], FORGE.

Высокие требования к эффективности выполнения параллельных программ и изменения в архитектуре параллельных вычислительных систем привели к тому, что в настоящее время нет ни одного общепризнанного высокоуровневого языка параллельного программирования для современных кластеров. В системах автоматизированного распараллеливания на программиста стали возлагаться ответственные

решения не только по уточнению свойств его последовательной программы, но и по ее отображению на параллельную вычислительную систему. Это является серьезным недостатком, вызывающим огромные трудности при использования таких систем.

Таким образом, из всех известных крупных систем автоматизированного распараллеливания на кластер нет ни одной, которая стабильно развивается и поддерживается. Вышеперечисленные системы являются больше академическими проектами, нежели промышленными системами. Основными недостатками таких систем являются:

- Попытка распараллелить всю программу целиком, составить для этой программы согласованное решение по распределению данных;
- Сильное вовлечение программиста в принятие ответственных решений по распределению данных.

По сути, такие системы сложно назвать даже автоматизированными системами для распараллеливания на кластер, потому что они осуществляют некоторый анализ программ, а также помогают расставлять некоторые спецификации параллелизма в коде программы, но только по согласованию с программистом.

3. Понятие области распараллеливания в системе SAPFOR. Областью распараллеливания (ОР) будем называть совокупность исполняемых операторов исходного кода программы, описываемых с помощью фрагментов. Фрагмент — это множество исполняемых операторов в рамках одной области вложенности (функция, процедура, цикл, условный оператор и т.д.) с одним входом и несколькими выходами. По умолчанию (если не было обозначено никаких других областей) вся программа рассматривается как одна область распараллеливания с именем *DEFAULT*, которая содержит в себе все исполняемые операторы.

В случае наличия пользовательских ОР область по умолчанию системой SAPFOR не рассматривается. Введем понятие вложенности и пересечения областей. В рамках одной области видимости определение данных понятий является тривиальным. Для процедур и их вызовов данные понятия определяются следующим образом.

Пусть есть три процедуры *A, B, C*. Явным вызовом процедуры B из процедуры A будем называть вызов процедуры B непосредственно из тела процедуры A. Косвенным вызовом процедуры B из процедуры A будем называть такой вызов процедуры B, который содержится в каком-либо теле процедур, вызываемых непосредственно из тела процедуры A, но при этом отсутствует явный вызов процедуры B из процедуры A. Под явным объявлением области распараллеливания будем понимать выделение последовательности исполняемых операторов с помощью пары директив:

```
!$SPF PARALLEL_REG
    ...
!$SPF END PARALLEL_REG
```

Две различные ОР будут вложенными в том случае, если явное объявление ОР #1 находится в процедуре A и явное объявление ОР #2 находится в процедуре B, причем B вызывается из процедуры A явно или косвенно (листинг 1).

ОР будут пересекающимися по процедуре C, если явное объявление ОР #1 находится в процедуре A и явное объявление ОР #2 находится в процедуре B, причем A не вызывает B, а B не вызывает A ни явно, ни косвенно, но и A и B вызывают процедуру C явно или косвенно (листинг 2).

Листинг 1. Пример вложенных областей распараллеливания

```
SUBROUTINE A
    !$SPF PARALLEL_REG regA
        ...
        call B
    !$SPF END PARALLEL_REG
END

SUBROUTINE B
    !$SPF PARALLEL_REG regB
    ...
    !$SPF END PARALLEL_REG
END
```

Листинг 2. Пример вложенных областей распараллеливания

```
SUBROUTINE A
    !$SPF PARALLEL_REG regA
        ...
        call C
    !$SPF END PARALLEL_REG
END

SUBROUTINE B
    !$SPF PARALLEL_REG regB
        ...
        call C
    !$SPF END PARALLEL_REG
END

SUBROUTINE C
    ...
END
```

Каждая область имеет свой уникальный идентификатор, называемый именем области. Совокупность разных фрагментов (внутри функции, а также в разных файлах) с одинаковым именем рассматривается системой SAPFOR как одна область распараллеливания. Для каждой из областей распараллеливания с разными именами строятся разные и независимые решения по распределению данных и вычислений.

3.1. Правила расстановки областей распараллеливания и разрешения конфликтов. Вложенные области распараллеливания в системе SAPFOR не допускаются ни в рамках одной области видимости, ни между процедурами. Поэтому для снятия данных ограничений, а также реализации концепции областей распараллеливания, в системе SAPFOR были реализованы преобразования, приводящие программу в соответствующий вид. Рассмотрим данные алгоритмы преобразования.

Распределенные массивы (такие массивы, которые будут распределены на узлы кластера) и *нераспределенные* массивы (такие массивы, которые будут размножены на узлах кластера) в модели DVMH отличаются по использованию их в программе. Чтобы ограничить область действия распределенных массивов, которые появятся после распараллеливания программы с ОР системой SAPFOR, введем соответствующие правила преобразования явных областей распараллеливания. Для каждой ОР необходимо заменить используемые в ней массивы на массивы-копии по следующим правилам. Для случая массивов, передаваемых в качестве параметров процедуре или функции:

- необходимо добавить локальные массивы такого же размера, как и массивы, передаваемые в процедуру в качестве параметров. Такие локальные массивы мы будем называть массивами-копиями;
- для всех параметров процедуры, которые являются массивами, необходимо сделать их переименование в теле процедуры для тех операторов, которые используются в области распараллеливания;
- необходимо добавить копирование входных массивов в новые массивы-копии после входа в область распараллеливания и добавить копирование выходных массивов из массивов-копий после выхода из нее.

Для программ на языке Фортран при необходимости замены глобальных массивов, используемых в common–блоке, создается новый common–блок и туда помещаются массивы-копии для каждого массива из соответствующего common–блока. В случае использования модуля достаточно создать массивы-копии в данном модуле. Для каждого массива из common–блока или модуля создается только один массив-копия вне зависимости от количества ОР, в котором он может использоваться. По аналогии с локальными массивами в глобальные массивы-копии копируются исходные массивы до входа в область, а после выхода из области выполняется обратное копирование. Таким образом, массивы-копии призваны “ограничить” действия системы SAPFOR в рамках выделенных фрагментов распараллеливания.

Если проект состоит из нескольких функций и нескольких файлов, то возникает возможность разной расстановки ОР. Считается, что если ОР содержит в себе вызовы процедур или функций, то тела данных функций или процедур автоматически включаются в данную область. Если ОР располагается внутри процедуры и она покрывает все исполняемые операторы, то все вызовы этой процедуры автоматически считаются включенными в эту область. В связи с этим возникают следующие случаи:

- первый и самый простой случай — области распараллеливания не пересекаются даже с учетом вложенности процедур. В данном случае ничего дополнительно предпринимать не нужно;

- области не пересекаются, но есть процедура, вызов которой есть и вне всех ОР, и внутри одной ОР. По правилам языка Fortran–DVMH данная процедура не может принимать и распределенные, и нераспределенные массивы. Таким образом, в этой процедуре нельзя использовать *INHERIT* указание (в модели DVMH данное указание позволяет унаследовать распределение массива, переданного в качестве параметра в процедуре). Значит, необходимо сделать копию процедуры с учетом вложенных процедур, а также заменить вызов процедуры в тех ОР, где она используется;
- области пересекаются по некоторым процедурам, то есть имеются общие процедуры, которые вызываются из разных областей. Данные процедуры будем называть конфликтными. В общем случае каждую из конфликтных процедур надо продублировать необходимое количество раз и заменить соответствующие вызовы в выделенных областях распараллеливания. Дублирования процедуры не требуется, если конфликтные процедуры содержат обращения только к нераспределенным массивам (например, приватным или локальным в этой процедуре) или вообще не содержат обращений к массивам.

Создание массивов-копий в процедурах выполняется на верхнем уровне, то есть создавать массивы-копии для массивов, которые передаются через параметры, во вложенных процедурах нет необходимости, так как такие массивы-копии уже были созданы ранее, а необходимые процедуры — продублированы.

4. Некоторые возможности языка Fortran–DVMH. Система SAPFOR выполняет отображение последовательной Фортран-программы на гетерогенный кластер. В качестве выходного языка используется язык Fortran–DVMH [11]. Язык Fortran–DVMH представляет собой язык Фортран 95, расширенный спецификациями параллелизма. Эти спецификации оформлены в виде специальных комментариев, которые называются директивами. Директивы FDVMH можно условно разделить на три подмножества:

- Распределение данных (директивы *DISTRIBUTE, REDISTRIBUTE, ALIGN, REALIGN*).
- Распределение вычислений (директива *PARALLEL*).
- Спецификация удаленных данных (директивы *SHADOW_RENEW, REMOTE_ACCESS, ACROSS, REDUCTION*).

Модель параллелизма FDVMH базируется на специальной форме параллелизма по данным: одна программа — множество потоков данных. В этой модели одна и та же программа выполняется на каждом процессоре, но каждый процессор выполняет свое подмножество операторов в соответствии с распределением данных.

В модели FDVMH пользователь вначале определяет массивы, которые должны быть распределены между процессорами (распределенные данные). Эти массивы специфицируются директивами отображения данных (*DISTRIBUTE, ALIGN*). Остальные переменные (распределяемые по умолчанию) отображаются по одному экземпляру на каждый процессор (размноженные данные). Размноженная переменная должна иметь одно и то же значение на каждом процессоре за исключением переменных в параллельных конструкциях. Модель FDVMH определяет два уровня параллелизма: параллелизм по данным и параллелизм задач. Системой SAPFOR реализуется только первый уровень параллелизма.

Параллелизм по данным реализуется распределением витков тесно вложенного гнезда цикла между процессорами. При этом каждый виток такого параллельного цикла полностью выполняется на одном процессоре. Операторы вне параллельного цикла выполняются по правилу собственных вычислений — каждый процессор выполняет только вычисления собственных данных, то есть тех данных, которые распределены на этот процессор. При вычислении значения собственной переменной в параллельных циклах процессору могут потребоваться как значения собственных переменных, так и значения несобственных (удаленных) переменных. Все удаленные переменные должны быть указаны в директивах доступа к удаленным данным:

- *REMOTE_ACCESS* — позволяет получить доступ ко всем секциям массива, который не отображен на текущий процессор.
- *SHADOW_RENEW* — позволяет обеспечить актуальность так называемых “теневых граней” для каждого распределенного массива.
- *ACROSS* — позволяет выполнять параллельный цикл, который имеет регулярную зависимость по данным в таком цикле.
- *REDUCTION* — позволяет выполнять редукционные операции в параллельном цикле.

Можно отметить следующие преимущества FDVMH при выборе выходного языка параллельного программирования для системы SAPFOR:

- Специальные комментарии являются высокоуровневыми спецификациями параллелизма в терминах последовательной программы.
- Отсутствуют низкоуровневые передачи данных и синхронизации в исходном коде параллельной программы.
- "Последовательный" стиль программирования.
- Спецификации параллелизма "невидимы" для стандартных компиляторов.
- Существует только один экземпляр программы для последовательного и параллельного счета.
- Большое количество оптимизаций, реализованных в компиляторе и библиотеке поддержки DVM-системы.

Рассмотрим необходимые в данной статье спецификации DVMH-модели на примере для модельной программы SOR (Successive Over Relaxation) — последовательной верхней релаксации. Основной цикл, имеющий зависимости по данным, для квадратной матрицы порядка N представлен в листинге 3.

Листинг 3. Пример вложенных областей распараллеливания

```
!DVM$ DISTRIBUTE ( BLOCK, BLOCK) :: ARRAY
!DVM$ ALIGN A(I,J) WITH ARRAY(I,J)
..........
!DVM$ PARALLEL (J,I) ON A(I,J), ACROSS(A(1:1,1:1)), PRIVATE(S), REDUCTION(MAX(EPS))

DO J = 2, N - 1
   DO I = 2, N - 1
      S = A(I,J)
      A(I,J) = W * (A(I+1,J)+A(I,J+1)+A(I-1,J)+A(I,J-1)) / 4 + (1-W)*A(I,J)
      EPS = MAX(EPS, ABS(S - A(I,J)))
   ENDDO
ENDDO
```

Для того чтобы распараллелить данный цикл с помощью языка Fortran–DVMH, необходимо указать компилятору, как сопоставлены витки цикла с распределенными массивами, используемыми в программе. Также необходимо указать все зависимости между витками циклов по массивам. Для этого в языке Fortran–DVMH существует спецификация ACROSS.

Директива *DISTRIBUTE* задает распределение массива между узлами процессорной решетки. Директива *ALIGN* задает отображение одного массива на другой, то есть говорит о том, что нужно распределить элементы массива *A(I,J)* там же, где и элементы массива *ARRAY(I,J)*. Директива *PARALLEL* указывает соответствие отображаемых витков цикла с измерениями распределенных массивов, используемых в цикле. Директива *PRIVATE* [12] указывает список приватных переменных в цикле, таких переменных, которые будут локализованы в рамках одного витка цикла, а *REDUCTION* — список редукционных операций и соответствующих им редукционных переменных.

5. Применение механизма областей распараллеливания в системе SAPFOR: автоматизация. Использование механизма областей распараллеливания хорошо подходит в тех случаях, когда программист сталкивается с незнакомой ему программой. Чем больше рассматриваемая программа, тем сложнее процесс ее распараллеливания на кластер, так как необходимо представлять структуру программы, знать ее поведение при различных входных данных. Рассмотрим процесс распараллеливания последовательной исходной программы на языке Фортран с применением областей распараллеливания. Конечная цель — отображение всей программы на кластер с графическими процессорами. Будем действовать поэтапно: сначала отобразим программу на кластер, далее на кластер с многоядерными процессорами и GPU.

В качестве демонстрационной программы будем использовать Block Tridiagonal Solver [13] (тест BT из пакета тестов NAS версии 3.3), решающий трехмерную систему уравнений Навье–Стокса для сжимаемой жидкости или газа. В данной программе используется трехдиагональная схема и метод переменных направлений. Исходная версия данной программы содержит 3200 строк кода в фиксированном формате Фортран, 17 файлов и 19 процедур. Рассматриваемая программа является упрощенной версией используемой в инженерных расчетах программы, поэтому этапы загрузки и сохранения результатов счета были

Таблица 1

Профилирование последовательной программы BT

Название процедуры	Время выполнения, с	Количество строк
Вся программа	636 (100%)	3200 (100%)
X_SOLVE	156 (24%)	320 (10%)
Y_SOLVE	196 (30%)	320 (10%)
Z_SOLVE	208 (32%)	320 (10%)
COMPUTE_RHS	70 (11%)	306 (9.5%)

упрощены разработчиками данного пакета тестов. Подробное описание алгоритма, а также используемый метод приведены в статьях [14, 15, 16] и документации [17] к тестовому пакету.

Все описанные результаты запусков в данном разделе были получены на суперкомпьютере K60 [18] (раздел с GPU) с использованием компиляторов Intel версии 18 и CUDA версии 10.0. На разделе с GPU K60 в каждом узле установлены два 16-ядерных CPU Intel Xeon Gold 6142 и четыре GPU Tesla V100.

5.1. Профилирование программы. Для того чтобы понимать важность каждой процедуры, необходимо выполнить профилирование программы. Разработчики программы BT встроили профилирование программы на основе замеров времени выполнения вычислительных частей. Данная возможность может быть отключена без перекомпиляции программы. Основное время занимает вычислительная процедура *ADI*, которая, в свою очередь, вызывает четыре процедуры: три процедуры *X_SOLVE, Y_SOLVE, Z_SOLVE*, которые реализуют метод переменных направлений, а также вычислительную процедуру *COMPUTE_RHS*. Время выполнения последовательной программы и перечисленных выше процедур представлено в табл. 1. Из табл. 1 видно, что 40% кода выполняются более 95% времени.

5.2. Частичное распараллеливание программы с помощью областей. Проанализировав структуру программы, можно попытаться ее распараллелить снизу вверх, идя от самых внутренних процедур до самой внешней. Для того чтобы локализовать действие системы SAPFOR только на рассматриваемых процедурах, необходимо каким-то образом расставить области распараллеливания. Расставим 4 области распараллеливания — каждую времяемкую процедуру поместим в свою область.

Стоит отметить, что процедуры *X/Y/Z_SOLVE* являются похожими, так как реализуют метод переменных направлений. Суть данного метода заключается в том, что в каждой из процедур происходит прямая и обратная прогонки по соответствующему измерению массива: по X, Y и Z. Соответственно, по этим координатам присутствует зависимость по данным в соответствующей процедуре. Такая зависимость по данным обусловлена выбором численного метода для решения рассматриваемой задачи газо- и гидродинамики. Далее, после рассмотрения только одной из них (например, *X_SOLVE*), будет подразумеваться, что аналогичные действия необходимо проделать и для остальных. В других процедурах зависимости по массиву отсутствуют.

Описанные зависимости в процедурах *X/Y/Z_SOLVE* классифицируются системой SAPFOR как регулярные зависимости по данным, которые могут быть устранены с помощью директивы ACROSS в модели DVMH.

Для дальнейшего распараллеливания требуется выполнять анализ с помощью системы SAPFOR и устранять возникающие в циклах проблемы, которые мешают их распараллеливанию. В данном случае потребовалась расстановка трех директив системе SAPFOR для приватизации массивов *FJAC*, *NJAC*, *LHS* перед циклами в процедурах *X/Y/Z_SOLVE* (по одной директиве на каждую процедуру), а также восьми директив, обозначающих начало и конец области распараллеливания. Процесс распараллеливания занял пару минут времени системы SAPFOR (с учетом получения параллельной версии) и не более 15 минут времени программиста на получение и анализ данных от системы SAPFOR. В результате распараллеливания были получены следующие требования выравнивания данных:

- **область X_SLV:**
  ALIGN $(*, j, k)$ *with TEMPLATE_0* $(*, j, k)$ *:: QS, SQUARE*
  ALIGN $(*, *, j, k)$ *with TEMPLATE_0* $(*, j, k)$ *:: U*
  ALIGN $(*, i, j, k)$ *with TEMPLATE_0* (i, j, k) *:: RSH*
- **область Y_SLV:**
  ALIGN $(i, *, k)$ *with TEMPLATE_1* $(i, *, k)$ *:: QS, SQUARE, RHO_I*

ALIGN $(*, i, *, k)$ *with TEMPLATE_ 1* $(i, *, k)$ *:: U*
ALIGN $(*, i, j, k)$ *with TEMPLATE_ 1* (i, j, k) *:: RSH*

- **область Z_ SLV:**
  ALIGN $(i, j, *)$ *with TEMPLATE_ 2* $(i, j, *)$ *:: QS, SQUARE, RHO_ I*
  ALIGN $(*, i, j, *)$ *with TEMPLATE_ 2* $(i, j, *)$ *:: U*
  ALIGN $(*, i, j, k)$ *with TEMPLATE_ 2* (i, j, k) *:: RSH*
- **область COMP:**
  ALIGN (i, j, k) *with TEMPLATE_ 3* (i, j, k) *:: US, WS, QS, VS*
  ALIGN (i, j, k) *with TEMPLATE_ 3* (i, j, k) *:: SQUARE, RHO_ I*
  ALIGN $(*, i, j, k)$ *with TEMPLATE_ 3* (i, j, k) *:: RSH, U, FORCING*

Для каждой области был построен свой вариант распределения данных и вычислений со своим DVM-шаблоном [11]. Шаблоном в DVMH-модели называется виртуальный массив, не занимающий места в памяти, который требуется для создания дерева выравнивания всех массивов в программе между собой. Звездочки в отображении слева от ключевого слова *"with"* означают, что данное измерение не будет распределено (то есть будет размножено на все процессоры при запуске программы). Звездочки справа от ключевого слова *"with"* означают, что на данное измерение массива или шаблона не будет производиться отображений.

Можно заметить, что в рассматриваемом случае нельзя построить общего решения для всех областей сразу: так или иначе, любое выбранное отображение на шаблон будет конфликтовать с отображением в других процедурах, так как существуют массивы с одними и тем же именем в разных областях распараллеливания, у которых, например, в одной области происходит отображение на одни измерения шаблона, а в другой — на другие (к примеру, массивы *QS, U, SQUARE*). Конфликт заключается в том, что для одного и того же массива требуется разное распределение данных между процессорами — требуется разделять данные между процессами у разных измерений массива в разных областях.

Данная проблема связана с тем, что программа не приведена к потенциально параллельной форме. Под потенциально параллельной формой понимается такая форма программы, которую система SAPFOR может автоматически преобразовать в эффективную параллельную программу без участия программиста. Для получения эффективной параллельной программы требуются преобразования кода, приводящие программу в потенциально параллельный вид, такие как "разделение циклов" и "расширение приватизируемых переменных". Эти преобразования реализованы в системе SAPFOR и будут применены далее.

Выберем одну из 32 параллельных схем (по 8 схем для каждой области распараллеливания: каждое измерение шаблона может быть распределено или не распределено, т.е. размножено на каждый процесс, в терминах DVM-системы) с лучшей оценкой от системы SAPFOR и запустим ее на выполнение на двух узлах на 64 процессах. Полученная параллельная программа в модели DVMH содержит в себе 285 DVM-директив распределения данных и вычислений. В дальнейшем, когда будет произведено объединение областей, количество DVM-директив сократится.

В табл. 2 приведены результаты запуска полученной параллельной программы, в скобках у параллельной версии указано "чистое" время выполнения области, а общее время содержит в себе также накладные расходы на вход и выход из области. По результатам видно, что распараллеленная системой SAPFOR программа хорошо ускоряется и показывает до 76% эффективности при использовании 64 процессов. Но это "чистое" ускорение, которое может быть получено в пределе при нулевых накладных расходах на коммуникации при использовании нескольких узлов кластера. Если учитывать общее время, то получаем пятикратное замедление программы относительно исходной последовательной по причине того, что

Таблица 2

Профилирование первой параллельной версии программы BT с областями

Название процедуры	Исходная, с	Параллельная v1, с	"Чистое" ускорение
Вся программа	636	2960	
X_ SOLVE	156	700 (3.6)	43.3
Y_ SOLVE	196	320 (4.8)	40.8
Z_ SOLVE	208	580 (4.2)	49.5
COMPUTE_ RHS	70	1120 (2.0)	35

входы в области распараллеливания и выходы из них содержатся в итерационном цикле и выполняются на каждой итерации, а каждый вход и выход содержит в себе копирование массивов для сохранения корректного выполнения оставшейся части программы.

5.3. Объединение областей распараллеливания. Рассмотрим варианты объединения областей. При объединении нераспределяемое измерение массива, обозначенное "*", поглощает распределяемое. Если мы хотим оставить измерение распределяемым, то необходимо устранить "проблемы", мешающие распределению конкретного распределения в конкретной области. Каждая область содержит в себе свой DVM-шаблон, который может быть распределен по следующим измерениям:
для области *X_SLV* — *TEMPLATE_0 (*, j, k)*, для области *Y_SLV* — *TEMPLATE_1 (i, *, k)*,
для области *Z_SLV* — *TEMPLATE_2 (i, j, *)* и для области COMP — *TEMPLATE_3 (i, j, k)*.

Получается 3 варианта объединения:

- *COMP+Z_SLV+Y_SLV* с отказом распределения по 2 и 3 измерениям;
- *COMP+X_SLV+Y_SLV* с отказом распределения по 1 и 2 измерениям;
- *COMP+X_SLV+Z_SLV* с отказом распределения по 1 и 3 измерениям.

Выполним объединение областей, например, COMP+X_SLV+Y_SLV и получим следующее распределение данных:

- **область Z_SLV:**
  *ALIGN (i, j, *) with TEMPLATE_2 (i, j, *) :: QS, SQUARE, RHO_I*
  ALIGN (, i, j, *) with TEMPLATE_2 (i, j, *) :: U*
  ALIGN (, i, j, *) with TEMPLATE_2 (i, j, k) :: RSH*

- **область COMP:**
  ALIGN (i, j, k) with TEMPLATE_3 (i, j, k) :: US, WS, QS, VS
  ALIGN (i, j, k) with TEMPLATE_3 (i, j, k) :: SQUARE,RHO_I
  ALIGN (, i, j, k) with TEMPLATE_3 (i, j, k) :: RSH,U,FORCING*

Выберем одну из 16 параллельных схем с лучшей оценкой от системы SAPFOR и запустим ее на выполнение на двух узлах на 64 процессах. Полученная параллельная программа в модели DVMH содержит в себе 244 DVM-директив распределения данных и вычислений (против 285 директив в первой версии). В табл. 3 приведены результаты запуска полученной параллельной программы.

Таблица 3

Профилирование второй параллельной версии программы BT с областями

Название процедуры	Исходная, с	Параллельная v2, с	"Чистое" ускорение
Вся программа	636	1620	
X_SOLVE	156	3.6 (3.6)	43.3
Y_SOLVE	196	4.8 (4.8)	40.8
Z_SOLVE	208	580 (4.2)	49.5
COMPUTE_RHS	70	1002 (2.0)	35

Таким образом, вторая версия параллельной программы ускорилась почти в два раза по сравнению с первой версией параллельной программы, но по-прежнему наблюдается замедление по сравнению с исходной последовательной версией программы. Далее невозможно выполнять объединение так, чтобы нераспределяемые измерения массивов не поглощали распределяемые. Выполним распространение решения области $COMP$ и сделаем объединение в одну область, а также вынесем данную область за пределы итерационного цикла. В результате распараллеливания с одной областью были получены следующие требования выравнивания данных:

$$ALIGN\ (i,j,k)\ with\ TEMP\_GLOBAL\ (*,i,j,k)\ ::\ US,\ WS,\ QS,\ VS,$$
$$ALIGN\ (i,j,k)\ with\ TEMP\_GLOBAL\ (*,i,j,k)\ ::\ SQUARE,\ RHO\_I,$$
$$ALIGN\ (m,i,j,k)\ with\ TEMP\_GLOBAL\ (m,i,j,k)\ ::\ RSH,\ U,\ FORCING.$$

Для того чтобы все измерения массивов были эффективно распределены на узлы кластера, необходимо не распределять первое измерение шаблона, а остальные — по желанию. В такой постановке есть всего 8 вариантов распределения шаблона: когда распределено только одно из измерений (3 варианта), когда распределены только два измерения (3 варианта) и когда распределены все три измерения или все три размножены.

В результате системой SAPFOR будет получена третья версия параллельной программы. Исходя из того, что в процедурах *X/Y/Z_SOLVE* не было произведено никаких преобразований исходного кода, при выборе какого-либо распределения данных возникнет несоответствие выбранного распределения данных при распределении вычислений. После выполнения оценки полученных 8 схем системой SAPFOR вариант с распределением только последнего измерения шаблона оказывается наилучшим. Полученная версия параллельной программы в модели DVMH содержит в себе 211 DVM-директив распределения данных и вычислений (против 244 директив во второй версии). В табл. 4 отражены результаты запуска третьей версии параллельной программы. Из-за конфликтов распределения вычислений процедура *Z_SOLVE* плохо ускоряется, так как требуются перераспределения данных, которые порождают "лишние" коммуникационные обмены.

Таблица 4

Профилирование третьей параллельной версии программы BT с областями

Название процедуры	Исходная, с	Параллельная v3, с	Ускорение
Вся программа	636	45	**14**
X_SOLVE	156	3.6	43.3
Y_SOLVE	196	4.8	40.8
Z_SOLVE	208	30	6.9
COMPUTE_RHS	70	6.0	11.6

В данной версии в итерационном цикле отсутствуют накладные расходы на вход и выход из области распараллеливания, тем самым получается ускорить программу в 14 раз по сравнению с исходной версией без каких-либо преобразований. Но программа по-прежнему распараллелена не полностью. Для полного распараллеливания распространим найденные решения на всю программу. В итоге будет получена та же самая программа с таким же распределением данных, но она будет содержать уже 125 DVM-директив распределения данных и вычислений (против 211 директив в третьей версии). Время выполнения не будет отличаться от приведенных ранее, так как все вспомогательные процедуры находятся вне итерационного цикла.

Можно отметить, что все манипуляции над исходным кодом программы проводились с помощью модуля визуализации системы SAPFOR. Выполнялось указание свойств программы с помощью SPF-директив для приватизации переменных, а также расстановка областей распараллеливания. Далее запускались алгоритмы устранения конфликтов областей и построения распределения данных и параллельных схем. То есть не было необходимости изменять исходный код вручную, и программа оставалась наиболее близкой к исходной. При этом удалось получить приемлемое ускорение в 14 раз на 64 ядрах (двух узлах кластера), практически ничего не делая с программой.

6. Применение механизма областей распараллеливания в системе SAPFOR: преобразования. Для дальнейшего распараллеливания программы требуется ее преобразование, так как для использования графического ускорителя и большого количества процессов и нитей необходим больший ресурс параллелизма, чем есть в существующей программе. Исходная версия программы была оптимизирована для последовательного выполнения, поэтому в *X/Y/Z_SOLVE* процедурах прямая и обратная прогонки выполняются в одном цикле, из-за чего из трех циклов параллельным может быть только один. По каждому из трех циклов для самой большой задачи всего около 1000 витков, поэтому требуется увеличение ресурса параллелизма хотя бы до двух циклов, что даст 1000^2 параллельных витков. Для увеличения ресурса параллелизма необходимо разделить циклы, чтобы образовать тесно вложенные независимые между собой циклы. А для этих целей необходимо выполнить преобразование "расширение приватизируемых переменных", чтобы каждый виток цикла работал со своей "копией" приватизируемого массива.

Как уже было отмечено выше, “расширение приватизируемых переменных” и “разделение циклов” выполняется системой SAPFOR по указанию директивы пользователя в коде программы и запуска соответствующего преобразования в модуле визуализации. В процедуру *X_SOLVE* для выполнения соответствующих преобразований необходимо вставить одну директиву для расширения приватизируемых переменных и одну директиву для разделения циклов. Аналогично для *Y/Z_SOLVE*. В процедуре *COMPUTE_RHS* необходимо также выполнить разделение циклов для образования трехмерных тесно вложенных гнезд циклов. Для этого необходимо добавить еще 4 директивы для разделения циклов.

Таким образом, необходимо расставить 10 директив в коде программы и запустить два преобразования через модуль визуализации системы SAPFOR, чтобы получить параллельную версию, которая будет лучше масштабироваться на большое количество процессов, а также сможет выполняться на графическом ускорителе. На данный момент система SAPFOR позволяет указать пользователю посредством анализа кода через систему визуализации на проблемы, мешающие параллельному выполнению тех или иных циклов, но не позволяет автоматически определять необходимые для этого преобразования.

Преобразование “расширение приватизируемых переменных” происходит таким образом, чтобы для каждого витка цикла данная переменная (или массив) были собственными, то есть чтобы переменные из секции PRIVATE перестали быть приватными. Это требуется для того, чтобы можно было устранить зависимость между двумя частями циклов и сделать разделение этих циклов корректным.

Для этих целей вводится новый массив и к нему добавляется необходимое количество измерений, равное количеству тесно вложенных гнезд циклов, для которого указано данное преобразование. В данной программе массивы *FJAC, NJAC* были трехмерными с формой $(5, 5, N)$, а *LHS* — четырехмерным с формой $(5, 5, 3, N)$, где N — размерность решаемой задачи. После преобразования размерность массивов увеличилась на два и их формы стали $(5, 5, N, N, N)$ и $(5, 5, 3, N, N, N)$ соответственно. Вследствие чего увеличилась и занимаемая ими память: для класса C ($N = 162$) массивы *FJAC, NJAC* стали занимать в памяти 850 МБ вместо 31 КБ, для класса D ($N = 408$) массивы *FJAC, NJAC* стали занимать в памяти 13.5 ГБ вместо 80 КБ. Соответственно массив *LHS* занимает еще в три раза больше памяти, чем *FJAC* и *NJAC.*

Полученная программа не является оптимальной, и здесь есть простор для оптимизаций — массивы *FJAC* и *NJAC* можно “удалить”, если вручную подставить все вычисления в вычислительном цикле, чтобы не выполнять их расширение. Также можно сократить количество измерений массива *LHS.* Данные оптимизации считаются не формализуемыми и нетривиальными для автоматического выполнения.

В табл. 5 представлены результаты запусков и количество строк кода полученной последовательной программы после преобразований. За счет увеличения количества обрабатываемой памяти после расширения приватных массивов наблюдается замедление программы почти в 2.3 раза, хотя общее количество строк кода увеличилось незначительно. Корректность программы проверяется “автоматически” с помощью встроенной функции в саму программу BT по завершении ее выполнения.

Таблица 5

Профилирование исходной и преобразованной последовательных программ BT

Название процедуры	Исходная	Преобразованная
Вся программа	636 с (3200 строк)	1494 с (3510 строк)
X_SOLVE	156 с (320 строк)	440 с (416 строк)
Y_SOLVE	196 с (320 строк)	480 с (413 строк)
Z_SOLVE	208 с (320 строк)	494 с (424 строки)
COMPUTE_RHS	70 с (306 строк)	78 с (337 строк)

После применения преобразований полученная параллельная версия в модели DVMH содержит 198 DVM-директив за счет того, что количество циклов, которое можно распараллелить, увеличилось. Также изменился и сам код программы. В табл. 6 представлены результаты запусков новой версии программы в модели DVMH, полученной с помощью SAPFOR на тех же двух узлах и 64 процессах. По сравнению с последовательной новая параллельная версия программы ускорилась в 25 раз, или в 10.6 раз по сравнению с исходной версией.

Таким образом, данная программа может быть выполнена на гибридном кластере с задействованием нитей и графических ускорителей. Например, ее можно выполнить на двух 16-ядерных CPU за 93.3 секун-

Таблица 6

Профилирование преобразованной параллельной версии программы BT

Название процедуры	Исходная	Параллельная v4, с	Ускорение
Вся программа	1494 (636)	60	25 (10.6)
X_SOLVE	440 (156)	18	24 (8.6)
Y_SOLVE	480 (196)	19	25 (10.3)
Z_SOLVE	494 (208)	20	25 (10.4)
COMPUTE_RHS	78 (70)	2.0	39 (35)

ды или на одном графическом процессоре за 50 секунд. Полученное время для графического процессора можно оценить с точки зрения производительности на потраченный ватт энергии. Так, графический процессор при полной нагрузке тратит 300 Вт, в то время как четыре CPU в общей сложности более 600 Вт, что дает в 2.25 раз большую энергоэффективность при запуске на GPU, чем при использовании CPU.

Для получения выигрыша на одном GPU можно попытаться применить преобразование не ко всей программе, а только для *Z_SOLVE* и *COMPUTE_RHS*. Судя по результатам из таблицы 5, получается примерно 1.5-кратное замедление последовательной программы вместо 2.3-кратного после выполнения частичного преобразования. Вследствие такого объединения будет получено время выполнения, указанное в табл. 7.

Таблица 7

Время запусков частично преобразованной параллельной версии программы BT на GPU

	Исходная, с	Параллельная v5, с	Ускорение
Вся программа	636	38.4	16.5

В итоге получается ускорение в 16.5 раз на одном графическом ускорителе, что лучше, чем было получено на двух узлах кластера при задействовании 64 процессов, — в 14 раз (табл. 4). При этом пользователь не менял вручную текст исходного алгоритма, а пользовался подсказками системы SAPFOR и выполнял автоматизированные преобразования “проблемных” мест с помощью системы посредством указания необходимых директив. Трудоемкость такого преобразования кода значительно ниже, чем ручная модификация кода. Несмотря на то что общее количество строк выросло незначительно (с 3200 строк кода до 3510 строк кода в преобразованной версии), преобразованию подверглась значительная часть кода — около 45%.

7. Ручная оптимизация и адаптация программы для кластера с несколькими GPU. Как было отмечено выше, для эффективного выполнения вычислительных регионов на нескольких GPU требуется дополнительная оптимизация, которая позволит получить еще больший выигрыш при использовании гетерогенного кластера. Большинство оптимизаций для GPU выполняется с целью минимизации количества чтений и записей в глобальную (общую) память и вынесения как можно большего количества вычислений на регистры (по-простому — в скалярные переменные с их последующей приватизацией для цикла). На современных архитектурах GPU скорость доступа к кэшу и регистровой памяти в два-три раза выше, чем к глобальной памяти.

Известно, что все скалярные переменные и массивы с константными размерами могут быть помещены на регистры. Таким образом, необходимо перенести выполнение всех важных вычислений на скалярные переменные и приватизируемые массивы. Данные оптимизации необходимо выполнить для процедур *X/Y/Z_SOLVE*, так как они получились наиболее времязатратными после отображения последовательной версии на GPU.

Для инициализации временного массива *LHS* используются два дополнительных массива *FJAC* и *NJAC*. Общее правило инициализации можно описать следующим формулами (для всех $i = 1..N$, $m1 = 1..5$, $m2 = 1..5$):

$$LHS\,(m1, m2, 1, i) = FJAC\,(m1, m2, i-1) + NJAC\,(m1, m2, i-1),$$
$$LHS\,(m1, m2, 2, i) = FJAC\,(m1, m2, i) + NJAC\,(m1, m2, i),$$
$$LHS\,(m1, m2, 3, i) = FJAC\,(m1, m2, i+1) + NJAC\,(m1, m2, i+1).$$

Видно, что для инициализации i-го элемента требуются $(i-1)$-й, i-й, $(i+1)$-й элементы двух других массивов, то есть необходимо хранить и вычислять всего три элемента массивов *FJAC* и *NJAC* вместо N. Тем самым можно снизить количество данных, которые нужно хранить в глобальной памяти GPU в несколько раз и сделать эти массивы приватизируемыми. Такая оптимизация увеличит количество вычислений в три раза, так как нам необходимо вычислить три элемента для каждого i-го элемента массива *LHS* вместо того, чтобы вычислять все N элементов заранее. Но так как вычисления занимают порядка 2–16 тактов, а запрос к памяти порядка 300–500 тактов, то такие накладные расходы будут "покрыты".

Для выполнения такой оптимизации требуется подставить непосредственно все вычисления *FJAC* и *NJAC* в инициализацию массива LHS, а также удалить ненужные инициализации массивов *FJAC/NJAC.* Также можно вынести все вычисления на регистры для массива *LHS* — завести приватный массив и заменить использование *LHS* на введенный приватный массив. Для сохранения корректного счета в приватный массив требуется загрузить данные до начала вычислений и сохранить необходимые данные после.

Детально изучив код процедуры, например *X_ SOLVE,* состоящий примерно из 350 строк, можно заметить, что в прямой и обратной прогонке участвует массив *LHS* $(5, 5, 1, N)$ вместо *LHS* $(5, 5, 3, N)$, остальные элементы используются в промежуточных вычислениях и могут быть вынесены на регистры, что позволит в три раза сократить количество сохраняемых данных на GPU в глобальной памяти.

После выполнения описанных выше оптимизаций над последовательной программой получится эффективная параллельная программа для GPU в модели DVHM как по памяти, так и по вычислительной нагрузке. Описанная работа требует некоторых знаний об устройстве параллельной архитектуры GPU, а также требует экспериментального подхода, так как не всегда описанные действия могут привести к желаемому результату, а оценить то или иное преобразование с точки зрения эффективности (особенно для GPU) достаточно тяжело или же невозможно.

В отличие от ручного распараллеливания с использованием технологий CUDA и MPI, пользователю приходится модифицировать код исходной последовательной программы, а затем с помощью систем SAPFOR и DVM получать параллельную и оценивать результат. Данный подход существенно сокращает трудоемкость, так как фактически необходимо написать один и тот же алгоритм другим способом, подходящим под ту или иную архитектуру, в данном случае — под графический процессор или кластер с большим количеством ядер.

Таким образом, в рамках одной процедуры, пользователю достаточно произвести модификацию последовательной программы, проверить ее корректность как последовательной. Если рассматривать полностью ручное распараллеливание, то необходимо модифицировать какую-то версию полученной программы на MPI+CUDA, которая будет в несколько раз более "объемной", чем исходная версия. Также необходимо потратить время на отладку полученной программы как параллельной между процессами, так и параллельной на GPU внутри узла. Такая отладка под силу далеко не каждому программисту.

Сравним результаты работы полученной эффективной программы на классе D с MPI-версией этой же программы, написанной разработчиками тестов NAS. В результате выполненных преобразований DVMH-программа сможет работать на одном GPU при таком наборе данных, а также хорошо масштабироваться на несколько GPU. В табл. 8 представлены результаты работы параллельных программ на кластере K60. Программа MPI была запущена на 256 процессах, или на 16 узлах кластера, то есть были задействованы все доступные ресурсы CPU. Для достижения не меньшего ускорения полученную DVMH-программу достаточно запустить только на двух GPU на одном узле.

Таблица 8

Время запусков оптимизированной параллельной версии программы BT, класс D

	Количество устройств	Время, с	Ускорение
Последовательная	1 CPU	13206	1
MPI	256 CPU	83	159
SAPFOR	1 GPU	108	123
SAPFOR	2 GPU	78	170
SAPFOR	4 GPU	60	221

8. Заключение. Благодаря введенному в системе SAPFOR инкрементальному подходу по распараллеливанию программ удается сначала сузить область действия системы на наиболее времязатратные участки кода, а затем распространить полученные решения на всю программу, если это возможно. На примере распараллеливания программы, решающей 3-мерную систему уравнений Навье–Стокса для сжимаемой жидкости или газа, можно сделать следующие выводы.

В отличие от распараллеливания с использованием стандартных технологий и расширений MPI, OpenMP и CUDA, разработка, преобразования и поддержка кода осуществляются программистом в последовательной программе. Полученная при этом DVMH-программа с помощью системы SAPFOR может задействовать многоядерные процессоры и графические ускорители, что позволит тратить меньше вычислительных ресурсов кластера с большей эффективностью. После дополнительных ручных преобразований над DVMH-программой можно получить более эффективный код для GPU, а также лучшую масштабируемость на кластер, так как и GPU, и большое количество процессов содержат много вычислительных ядер, для задействования которых требуется большой ресурс параллелизма.

Оценим трудоемкость использования предложенного подхода. Исходная версия программы занимала порядка 3200 строк кода, полученная эффективная параллельная программа с помощью системы SAPFOR с DVMH-указаниями занимает порядка 3600 строк кода. MPI-версия программы, написанная разработчиками тестов, занимает порядка 7600 строк кода. К этому нужно прибавить еще как минимум столько же, если мы хотим выйти на GPU с помощью технологии CUDA, так как каждый параллельный цикл должен быть скопирован или перенесен в ядро, которое будет выполняться на GPU. Помимо этого, требуется дополнить программу выделением и освобождением памяти, ее перемещением. Если мы хотим добавить OpenMP, то это также увеличит количество кода. То есть получается, что для написания такой же программы вручную требуется более 16000 строк кода. При этом сильно увеличивается сложность программы, ее отладки и сопровождения.

Эффективность получаемой с помощью системы SAPFOR параллельной версии зависит от того, насколько исходная программа была близка к потенциально параллельной форме. Ясно, что после ручной адаптации параллельная программа будет получаться лучше, чем после системы автоматизации. Здесь можно отметить тот факт, что сложность распараллеливания программы с нуля с использованием системы SAPFOR сильно понижается, а также понижается трудоемкость необходимых преобразований для приведения программы в потенциально параллельную форму, так как, по сути, программисту не нужно знать такие технологии, как OpenMP, CUDA и MPI. Достаточно иметь представление о том, какие проблемы могут возникать в циклах для их распараллеливания, например приватизация данных или редукционные операции. Для приведенной выше программы необходимо было переписать алгоритм более эффективным способом, что позволило повысить ускорение на одном GPU с 16.5 раз до 123 раз.

Ввиду описанных сложностей и трудоемкости процесса распараллеливания на GPU, на данный момент не существует инструмента, который позволил бы в автоматическом или полуавтоматическом режиме получить программу на кластер с графическими процессорами. Существуют некоторые работы по ручному распараллеливанию некоторых программ на один GPU с помощью технологии CUDA [19] и OpenCL [20], но эффективность распараллеливания достаточно низкая даже в рамках одного GPU. С помощью системы SAPFOR, а также с помощью ручной доводки программы на одном GPU на кластере K60 полученная программа ускоряется в 123 раза, а ускорение на четырех GPU – в 221 раз (45% эффективности по сравнению с одним GPU). Эффективность MPI-программы составляет примерно 62% на 256 ядрах.

В предложенном методе инкрементального распараллеливания на кластер в системе SAPFOR существует недостаток — отсутствует процесс автоматического объединения областей распараллеливания, а также их автоматической расстановки. В дальнейшем планируется реализовать необходимые алгоритмы для поиска наиболее важных областей распараллеливания, а также их последующего объединения в автоматическом режиме, что позволит еще больше снизить трудоемкость частичного распараллеливания программных комплексов.

СПИСОК ЛИТЕРАТУРЫ

1. NVidia CUDA Zone. https://developer.nvidia.com/cuda-zone.
2. *Колганов А.С., Катаев Н.А., Титов П.А.* Автоматизированное распараллеливание задачи моделирования распространения упругих волн в средах со сложной 3D геометрией поверхности на кластеры разной архитектуры // Вестн. Уфимского гос. авиацион. техн. ун-та. 2017. **21**, № 3. 87–96.
3. DVM-система. http://dvm-system.org.

4. *Клинов М.С., Крюков В.А.* Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестн. Нижегородского гос. ун-та им. Н.И. Лобачевского. 2009. № 2, 128–134.
5. *Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А.* Автоматизация распараллеливания программных комплексов // Тр. XVIII Всеросс. научной конференции "Научный сервис в сети Интернет". М.: ИПМ РАН, 2016. 76–85.
6. *Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Королев Н.Н., Крюков В.А., Кузнецов М.Ю., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А.* Инкрементальное распараллеливание для кластеров в системе САПФОР // Тр. XIX Всеросс. научной конференции "Научный сервис в сети Интернет". М.: ИПМ РАН, 2017. 48–52.
7. *Колганов А.С., Яшин С.В.* Автоматическое инкрементальное распараллеливание больших программных комплексов с помощью системы SAPFOR // Тр. Междунар. научной конференции "Параллельные вычислительные технологии" (ПаВТ'2019). Челябинск: Издательский центр ЮУрГУ, 2019. 275–287.
8. *Banerjee P., Chandy J.A., Gupta M., et al.* An overview of the PARADIGM compiler for distributed-memory multicomputers," http://www.cs.cmu.edu/∼745/papers/paradigm.pdf.
9. Система BERT77: automatic and efficient parallelizer for FORTRAN. http://www.sai.msu.su/sal/C/3/BERT_77.html.
10. Система ParaWise. http://www.parallelsp.com/.
11. Описание модели DVMH. URL: http://dvm-system.org/static_data/docs/FDVMH-user-guide-ru.pdf.
12. *Колганов А.С., Королев Н.Н.* Статический анализ приватных переменных в системе автоматизированного распараллеливания Фортран-программ // Тр. Междунар. научной конференции "Параллельные вычислительные технологии" (ПаВТ'2018). Челябинск: Издательский центр ЮУрГУ, 2018. 286–294.
13. Тесты NAS parallel benchmarks. https://www.nas.nasa.gov/publications/npb.html.
14. *Bailey D.H., Barton J.T.* The NAS kernel benchmark program. Report TM-86711. Moffett Field: NASA Ames Research Center, 1985.
15. *Pulliam T.H.* Efficient solution methods for the Navier–Stokes Equations. Rhode-Saint-Genése: Von Kármán Inst. for Fluid Dynamics, 1986.
16. *Jameson A., Schmidt W., Turkel E.* Numerical solution of the Euler equations by finite volume methods using Runge–Kutta time stepping schemes // AIAA Paper 81-1259. 1981. doi: 10.2514/6.1981-1259.
17. The NAS Parallel Benchmarks. https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf.
18. Суперкомпьютер K60. https://keldysh.ru/.
19. Распараллеленные тесты NAS с помощью CUDA. https://www.tu-chemnitz.de/informatik/PI/sonstiges/downloads/npb-gpu/index.php.en.
20. Распараллеленные тесты NAS с помощью OpenCL. http://aces.snu.ac.kr/software/snu-npb/.

Поступила в редакцию
26 июня 2020

An Experience of Applying the Parallelization Regions for the Step-by-Step Parallelization of Software Packages Using the SAPFOR System

A. S. Kolganov[1]

[1] *Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Miusskaya ploshchad' 4, Moscow, 125047, Russia; Scientist, e-mail: alexander.k.s@mail.ru*

Received June 26, 2020

Abstract: The main difficulty in developing a parallel program for a cluster is the need to make global decisions on the distribution of data and computations, taking into account the properties of the entire program, and then doing the hard work of modifying the program and debugging it. A large amount of code as well as multimoduling, multivariant and multilanguage, make it difficult to make decisions on a consistent distribution of data and computations. The experience of using the previous SAPFOR system showed that, when parallelizing large programs and software packages for a cluster, one should be able to parallelize them gradually, starting with the most time-intensive fragments and gradually adding new fragments until we reach the desired level of parallel program efficiency. For this purpose, the previous system was completely redesigned and a new system SAPFOR (System FOR Automated Parallelization) was created. To solve this problem, the method of incremental or partial parallelization will be considered in this paper. The idea of this method is that not the entire program is

subjected to parallelization, but only its parts (parallelization regions) where additional versions of the required data are created and distributed and the corresponding computations are performed. This paper also discusses the application of automated mapping of programs to a cluster using the proposed incremental parallelization method and using the example of a NPB (NAS Parallel Benchmarks) software package.

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

References

1. NVidia CUDA Zone. https://developer.nvidia.com/cuda-zone. Cited November 15, 2020.

2. A. S. Kolganov, N. A. Kataev, and P. A. Titov, "Automated Parallelization of a Simulation Method of Elastic Wave Propagation in Media with Complex 3D Geometry Surface on High-Performance Heterogeneous Clusters," Vestn. Ufa Aviatsion. Tekh. Univ. **21** (3), 87–96 (2017).

3. DVM–system. http://dvm-system.org/. Cited November 15, 2020.

4. M. S. Klinov and V. A. Kryukov, "Automatic Parallelization of Fortran Programs. Mapping to Cluster," Vestn. Lobachevskii Univ. Nizhni Novgorod, No. 2, 128–134 (2009).

5. V. A. Bakhtin, O. F. Zhukova, N. A. Kataev, et al., "Automation of Software Package Parallelization," in *Proc. XVIII All-Russian Conference on Scientific Service on the Internet, Novorossiysk, Russia, September 19–24, 2016* (Keldysh Institute of Applied Mathematics, Moscow, 2016), pp. 76–85.

6. V. A. Bakhtin, O. F. Zhukova, N. A. Kataev, et al., "Incremental Parallelization for Clusters in the SAPFOR System," in *Proc. XIX All-Russian Conference on Scientific Service on the Internet, Novorossiysk, Russia, September 18–23, 2017* (Keldysh Institute of Applied Mathematics, Moscow, 2017), pp. 48–52.

7. A. S. Kolganov and S. V. Yashin, "Automatic Incremental Parallelization of Large Software Systems Using the SAPFOR System," in *Proc. Int. Conf. on Parallel Computing Technologies, Kaliningrad, Russia, April 2–4, 2019* (South Ural State Univ., Chelyabinsk, 2019), pp. 275–287.

8. P. Banerjee, J. A. Chandy, M. Gupta, et al., "An Overview of the PARADIGM Compiler for Distributed-Memory Multicomputers," http://www.cs.cmu.edu/∼745/papers/paradigm.pdf. Cited November 15, 2020.

9. BERT77 system: Automatic and Efficient Parallelizer for FORTRAN. http://www.sai.msu.su/sal/C/3/BERT_77.html. Cited November 15, 2020.

10. ParaWise System. http://www.parallelsp.com/. Cited November 15, 2020.

11. DVMH model. http://dvm-system.org/static_data/docs/FDVMH-user-guide-ru.pdf. Cited November 15, 2020.

12. A. S. Kolganov and N. N. Korolev, "Static Analysis of Private Variables in the System of Automated Parallelization of Fortran Programs," in *Proc. Int. Conf. on Parallel Computing Technologies, Rostov-on-Don, Russia, April 2–6, 2018* (South Ural State Univ., Chelyabinsk, 2018), pp. 286–294.

13. NAS Parallel Benchmarks. https://www.nas.nasa.gov/publications/npb.html. Cited November 15, 2020.

14. D. H. Bailey and J. T. Barton, *The NAS Kernel Benchmark Program*, Report TM-86711 (NASA Ames Research Center, Moffett Field, 1985).

15. T. H. Pulliam, *Efficient Solution Methods for the Navier–Stokes Equations* (Von Kármán Inst. for Fluid Dynamics, Rhode-Saint-Genése, 1986).

16. A. Jameson, W. Schmidt, and E. Turkel, "Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge–Kutta Time Stepping Schemes", AIAA Paper 81-1259 (1981). doi: 10.2514/6.1981-1259

17. The NAS Parallel Benchmarks.
https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf. Cited November 15, 2020.

18. K60 Supercomputer. https://keldysh.ru/. Cited November 15, 2020.

19. NAS Parallel Benchmarks with CUDA.
https://www.tu-chemnitz.de/informatik/PI/sonstiges/downloads/npb-gpu/index.php.en. Cited November 15, 2020.

20. NAS Parallel Benchmarks with OpenCL. http://aces.snu.ac.kr/software/snu-npb/. Cited November 15, 2020.