Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

Кочармин Михаил Дмитриевич

Разработка и реализация алгоритма распараллеливания фортран-программ на общую память

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
д.ф.-м.н., профессор
Крюков Виктор Алексеевич

Научный консультант:
к.ф.-м.н.
Колганов Александр Сергеевич

Москва, 2024

Аннотация

Разработка и реализация алгоритма распараллеливания
фортран-программ на общую память

Кочармин Михаил Дмитриевич

В данной работе рассматривается решение задачи автоматического распараллеливания фортран-программ для случая целевой системы с общей оперативной памятью. Задача решается в рамках доработки системы SAPFOR с использьзованием модели DVMH в качестве технологии распараллеливания.

Во введении приводятся некоторые сведения о распараллеливании, как общепринятые, так и те, что относятся только к системам SAPFOR и DVM, приводится описание этих систем. Далее, во второй главе описывается проблема распараллеливания на общую память с точки зрения DVM и с точки зрения автоматизации, рассматриваюся подходы к её решению. В этой главе так же определяются основные цели работы, описываются основные сценарии использования разрабатываемого фунционала. В третьей главе подробно описывается ход решения поставленной задачи, который был разбит на три этапа – теоретическую подготовку, реализацию и тестирование. В четвёртой главе приведено описание существовавших и добавленных алгоритмов, о которых шла речь в главе 3. В пятой главе приводятся результаты исследования эффективности получаемого распараллеливания на примере программ из пакета NAS Parallel Benchmarks.

Содержание

1 Введение

В современном мире спрос на вычисления растёт быстрее, чем тактовая частота работы процессоров, что обусловлено физическими проблемами и ограничениями. Из-за этого вычислительные системы наращивают свою мощь за счёт увеличения числа вычислительных ядер. Это привело к тому, что в настоящее время стала актуальна тема параллельных вычислений. Различные технологии параллельного программирования дают людям возможность эффективно использовать многоядерные процессоры и графические ускорители.

Создание параллельного кода – задача, требующая больших трудозатрат и высокой квалификации. Есть технологии, которые требуют от разработчиков большого внимания к деталям распараллеливания. Они интегрируются в логику программ, что позволяет писать максимально эффективный код, что, в свою очередь, ведёт к дополнительному, иногда огромному, усложнению структур программ. Примерами таких технологий являются библиотеки параллельного программирования pthreads, CUDA, HIP, OpenCL.

Для борьбы со сложностью разработки параллельного кода, были созданы средства, с помощью которых можно добиться распараллеливания не изменяя логику работы программ. При таком подходе можно потерять в эффективности по сравнению с предыдущими технологиями, но сэкономить человеческие ресурсы. Такие технологии обычно предоставляют программисту лёгкий у удобный интерфейс, оформленный в виде специальных директив компилятору. Сюда можно отнести технологии OpenMP, OpenACC, DVM.

Наконец, существуют инструменты, которые ещё сильнее снижают трудозатраты процесса распараллеливания. С помощью них можно в автоматизированном режиме получать параллельные программы из последовательных. Часто это накладывает ограничения на возможности распараллеливания, но ввиду нехватки человеческих ресурсов такой подход в определённых случаях может быть оправдан. К этому классу относятся инструменты такие как CAPTools/Parawise, FORGE Magic/DM, ParalWare Trainer,

SAPFOR.

Больших успехов в направлении экономии человеческих ресурсов достигли учёные из Института прикладной математики им. М.В. Келдыша Российской академии наук [1]. В Институте были разработаны и активно поддерживаются два средства параллельного программирования: это DVM-система [2] – программный комплекс для компиляции, выполнения и отладки параллельного кода, а также инструмент для автоматизированного распараллеливания программ – система SAPFOR [3], [4].

Перед тем, как перейти к основной теме данной работы, в этой главе будут описаны некоторые сведения о параллельном программированнии, как общеизвестные, так и относящиеся исключительно к DVM и SAPFOR, на базе которых и проводилась работа.

1.1 Общие сведения и термины из параллельного программирования

Когда речь идёт о высокоуровневых технологиях распараллеливания (OpenMP, OpenACC, DVM), говорят о распараллеливании витков циклов. На это есть несколько причин:

- обычно, большая часть вычислительной работы в программах находится именно в циклах;
- анализировать абстрактные последовательности операторов с точки зрения возможности их параллельного выполнения сложнее, чем анализировать зависимости между витками циклов;
- явное указание распараллеливаемых частей кода влечёт за собой изменения в структуре кода, в то время как указание параллельного цикла обычно выглядит как директива перед его заголовком;

Более того, принято рассматривать только циклы, находящиеся в *канонической форме*, то есть циклы, у которых есть итерационная переменная, проходящая заданный диапазон значений с заданным шагом:

```
1       ...
2       DO i = 1, 10
3           ...
4       ENDDO
5       ...
```

Листинг 1: пример цикла в канонической форме на языке фортран с итерационной переменной `i`, принимающая все значения от 1 до 10.

Иногда приходится иметь дело с вложенными циклами, у которых нет операторов между заголовками и концами циклов. Их совокупность называют *гездом тесновложенных циклов*, или просто гнездом циклов. С точки зрения параллелизма, в общем случае выгоднее распараллеливать не только самый верхний цикл в гнезде, а гездо целиком, увеличивая тем самым итерационное пространство.

```
1       ...
2       DO i = 1, 10
3           DO j = 2, 20
4               DO k = 3, 30
5               ...
6               ENDDO
7           ENDDO
8       ENDDO
9       ...
```

Листинг 2: пример гнезда из трёх тесновложенных циклов.

Будем говорить, что витки цикла (или гнезда циклов) имеют *зависимость по данным*, если в нём существуют два витка i_0 и i_1 такие, что в последовательной программе виток i_0 записывает какие-либо данные, а виток i_1 их считывает.

```
1       ...
2       DO i = 3, 10
3           A(i) = A(i - 1) + A(i - 2)
4       ENDDO
5       ...
```

Листинг 3: пример цикла с зависимостью по данным, где каждый виток с номером i записывает данные, которые читают витки $i + 1$ и $i + 2$.

В общем случае такие циклы нельзя распараллеливать, так как в них важен порядок итераций, но существует несколько разных приёмов, позволяющих в некоторых случаях избегать зависимости по данным.

Важным этапом распараллеливания является определение того, какие данные являются локальными или общими по отношению к виткам циклов. Если скалярная переменная или массив перед использованием переопределяется (перезаписываются значения элементов соответственно) на той же самой итерации, то говорят, что такая переменная является *приватной переменной* или *приватным массивом* соответственно. Из определения следует, что счётчик цикла всегда приватен.

```
1       ...
2       DO i = 1, 10
3           DO j = 1, 100
4               TMP(j) = j
5           ENDDO
6
7           t = TMP(100)
8           ...
9       ENDDO
10      ...
```

Листинг 4: пример цикла по `i` с приватным массивом `TMP` и приватными переменными `i` (счётчик), `j`, `t`.

Похожим является класс *редукционных переменных*. Он состоит из переменных, значения которых после цикла зависят от каждой итерации цикла, но не зависят от порядка их выполнения. При этом бинарная операция, с помощью которых объединяются частичные значения, называется *редукционной*.

```
1       ...
2       sum = 0
3       DO i = 1, 10
4           sum = sum + A(i)
5       ENDDO
6
7       write(*, *) sum
8       ...
```

Листинг 5: пример цикла с редукционной переменной `sum` (редукционная операция – сложение).

1.2 DVM-система и модель параллелизма по данным

DVM-система (Distributed Virtual Memory или Distributed Virtual Machine) представляет собой набор программных средств, которые позволяют пользователю производить следующие операции:

- компилировать параллельные программы на языках C-DVMH (расширение языка Си) и Fortran-DVMH (расширение языка фортран);
- запускать скомпилированные программы;

- отлаживать скомпилированные программы (на предмет производительности и корректности распараллеливания);

DVM-система ведёт работу с программами, написанными на языках C-DVMH или Fortran-DVMH, которые представляют собой расширения для языков Си и фортран соответственно. Они позволяют распараллеливать последовательные программы при помощи добавления директив – прагм (для языка Си) или спецкомментариев (для языка фортран). Похожий подход используется в технологиях OpenMP, OpenACC и др.

Целевая вычислительная система модели DVMH рассматривается как совокупность узлов, каждый из которых имеет свою оперативную память. Чаще всего такие узлы состоят из процессора, к которому подключено несколько ускорителей.

При этом, DVMH-модель основана на параллелизме по данным. Это означает, что DVM, в соответствии с заданными пользователем директивами, распределяет данные (в данном случае элементы массивов) по доступным узлам вычислительной системы. После этого, по принципу собственных вычислений, согласно которому каждый узел выполняет те и только те вычисления, которые относятся к данным, расположенным в его оперативной памяти, происходит распределение вычислений.

1.3 SAPFOR

Система SAPFOR (System FOR Automated Parallelization) предназначена для автоматизации процесса распараллеливания программ с использованием технологии DVMH. Она, так же как и DVM система, поддерживает языки Си и фортран. Система SAPFOR разделена на два проекта, каждый их которых отвечает за свой язык. В данной работе рассматривается только часть, связанная с языком фортран.

С помощью системы SAPFOR можно производить не только само распараллеливание, но и множество других преобразований, позволяющих приводить программы к *потенциально параллельному виду* – к форме, в которой она может быть автоматически переведена в параллельную без участия пользователя. Для удобства проведения

таких манипуляций была создана графическая оболочка, которая предоставляет пользователям системы удобный интерфейс для использования системы SAPFOR.

Также система SAPFOR предоставляет набор директив, с помощью которых пользователь может управлять поведением системы при обработке программы. Например, с помощью таких директив можно задавать различные свойства программы, давать указания на расстановку контрольных точек, ограничивать распараллеливание. Сейчас системой поддерживается около десяти различных директив.

Рассмотрим подробнее две из них, которые будут упомянаться далее в работе, более пристально. Это директивы приватизации и редукции. Директива приватизации, как и все директивы системе SAPFOR, представлена в виде комментария, начинающегося с префикса `$SPF`. Далее идёт тело директивы, которое имеет вид `ANALYSIS (PRIVATE(...))`, где вместо троеточия должен быть указан список приватных переменных (скаляров или массивов) через запятую. Следующий пример демонстрирует использование директивы `PRIVATE` системы SAPFOR:

```
1  ...
2  !$SPF ANALYSIS(PRIVATE(B))
3      DO I = 1, N
4          DO J = 1, N
5              B(J) = J
6          END DO
7
8          A(I) = B(A(I))
9      END DO
10 ...
```

Листинг 6: пример задания приватного массива `B` для цикла на строке 3 с помощью директивы системе SAPFOR.

Хотя скаляры и можно объявлять в директиве приватизации, на практике обычно

это не используется, потому что в системе SAPFOR при распараллеливании автоматически производится анализ приватных скалярных переменных.

Аналогично устроена директива `!$SPF ANALYSIS (REDUCTION(OP(...)))`, задающая редукцию по переменным из списка, стоящем вместо троеточия. `OP` в директиве задаёт редукционную операцию. Далее приведён приведён пример использования этой директивы:

```
1 ...
2 !$SPF ANALYSIS(REDUCTION (MAX(EPS)))
3     DO I = 1, N
4         DO J = 1, N
5             EPS = MAX(EPS, ABS(B(I,J) - A(I,J)))
6             A(I, J) = B(I, J)
7         END DO
8     END DO
9 ...
```

Листинг 7: пример задания редукции по переменной `EPS` с редукционной операцией взятия максимума.

2 Распараллеливание на общую память

При написании параллельных DVMH-программ пользователь решает две задачи: он должен найти оптимальный способ распределения данных и обозначить циклы, которые могут выполняться параллельно. На практике нередко возникают ситуации, когда для рассматриваемой программы решить первую задачу трудно или невозможно, но вторая задача решается успешно, то есть программа обладает хорошим потенциалом для распараллеливания.

Основной проблемой, возникающей при попытке построить схему распределения данных, является то, что разные циклы для их распараллеливания могут требовать разных, конфликтующих, схем распределения данных. При распараллеливании практически значимых программ, в силу их объёмности, такие конфликты возникают повсеместно. Эта проблема подробно рассматривается в [5].

Выходом из такой ситуации является рассмотрение частного случая – распараллеливания на общую память. При таком распараллеливании предполагается, что целевая вычислительная система состоит из единственного устройства. Это ограничение позволяет обойти потребность в распределении данных, так как все данные располагаются в общей оперативной памяти устройства.

В качестве устройства для запуска программы, распараллеленной на общую память, можно рассматривать многопоточный процссор или графический ускоритель (видеокарту).

2.1 Реализация в DVM-системе

Как описывалось во введении, распараллеливание в модели DVM происходит при помощи специальных директив. При привычном распараллеливании с распределением данных, в основном, используются следующие директивы:

- `DISTRIBUTE` – предназначена для распределения элементов массивов по узлам;

- `ALIGN` – отображает элементы нескольких массивов на заданный распределённый массив и распределяет их на соответствующие узлы;
- `REGION` – определяет область кода, которую следует выполнять параллельно;
- `PARALLEL ... ON` – определяет цикл, витки которого следует выполнять параллельно, при этом задавая отображение пространства витков цикла на распределённый массив;

```
1  ...
2  !DVM$ DISTRIBUTE (BLOCK,BLOCK) :: A
3  !DVM$ ALIGN B(I,J) WITH A(I,J)
4
5  !DVM$ REGION
6  !DVM$ PARALLEL (I,J) ON A(I,J), REDUCTION (MAX(EPS))
7      DO I = 1, N
8          DO J = 1, N
9              EPS = MAX(EPS, ABS(B(I,J) - A(I,J)))
10             A(I, J) = B(I, J)
11         END DO
12     END DO
13 !DVM$ END REGION
14 ...
```

Листинг 8: пример распараллеленного в модели DVM гнезда циклов. Здесь происходит отображение элементов массива B на элементы массива A, распределение массива A и распараллеливание гнезда циклов. При запуске на кластере каждый узел получит непрерывные и примерно равные секции массива A, точно такие же секции B и соответствующие этим элементам итерации цикла.

Чтобы дать пользователю возможность распараллеливать программы на общую память, в синтаксис языка DVM-системы была добавлена новая форма директивы PARALLEL для распараллеливания без распределения данных.

Новый вариант директивы PARALLEL отличается от исходного тем, что в ней отсутствует клауза ON. При использовании таких директив не надо указывать распределённые массивы. Сами директивы распределения данных также не нужны в таких программах. Более того, в таких директивах PARALLEL не должны присутствовать клаузы доступа к удалённым данных, такие как SHADOW_RENEW, SHADOW_COMPUTE и REMOTE_ACCESS.

```
1  ...
2  !DVM$ REGION
3  !DVM$ PARALLEL (I,J), REDUCTION (MAX(EPS)), TIE(A(I, J),
     ↪  B(I, J))
4      DO I = 1, N
5          DO J = 1, N
6              EPS = MAX(EPS, ABS(B(I,J) - A(I,J)))
7              A(I, J) = B(I, J)
8          END DO
9      END DO
10 !DVM$ END REGION
11 ...
```

Листинг 9: пример распарараллеливания гнезда циклов на общую память.

Также в таком варианте директивы PARALLEL может присутствовать клауза TIE, сопоставляющая итерации цикла с массивом. Она используется в DVM-системе для улучшения производительности выходного исполняемого кода.

2.2 Мотивация поддержки режима распараллеливания на общую память в системе SAPFOR

Поскольку система SAPFOR не способна проанализировать все тонкости логики работы программы, процесс построения оптимальной схемы распределения данных затрудняется. Поэтому аналогичная потребность в функционале распараллеливания на общую память возникает и в системе SAPFOR. Таким образом, реализация нового режима работы системы SAPFOR для распараллеливания на общую память стала для данной работы основной целью, которая детально описывается в последующих параграфах.

2.3 Постановка задачи, определение целей работы

Так возникает задача добавления в систему SAPFOR дополнительного сценария работы – распараллеливания фортран-программ без построения схемы распределения данных и с использованием директив `PARALLEL` без клаузы `ON`.

Требуется, чтобы новый режим мог обходить ограничения, накладываемые распределением данных, тем самым расширяя класс распараллеливаемых программ на общую память по сравнению со стандартным распараллеливанием на кластер. В частности, вставляемые директивы должны корректно описывать все данные, используемые в цикле, их область хранения в памяти и зависимости типа ACROSS, если они есть.

Также от добавляемого функционала естественно потребовать его *корректность*: при условии, если входная программа корректная, то система SAPFOR должна выдавать правильную параллельную программу, которая должен успешно компилироваться и выполняться.

Помимо этого, результирующее распараллеливание должно быть эффективным, то есть давать приемлемый прирост производительности за счёт многопоточного выполнения: не замедлять программу существенно в худших случаях и получать распаралле-

ливание, конкурирующие с ручным в лучших случаях. В это требование дополнительно входит расстановка оптимизирующих клауз `TIE` в директивы `PARALLEL` везде, где это возможно.

Поскольку работа ведётся в рамках доработки системы SAPFOR, решение должно быть в неё интегрировано должным образом:

- в системе SAPFOR каждое преобразование оформляется в виде последовательности *проходов* – алгоритмов, которые логически разделены на блоки. Поэтому новый функционал должен быть реализован в виде нового прохода (или в виде последовательности новых проходов);
- новый режим распараллеливания должен учитывать и правильно обрабатывать директивы системы SAPFOR;
- должна быть добавлена возможность вызова этого режима через диалоговую графическую оболочку;
- вместе с самим распараллеливанием должен быть добавлен функционал анализа входного кода, который выдавал бы информацию по распараллеливанию без фактической вставки директив. Аналогичный анализ в системе SAPFOR есть и для режима распараллеливания на кластер. Он служит для выявления проблем при автоматизированном распараллеливании;

Ещё одной подцелью работы является проверка выполнения этих требований путём проведения тестирования на выбранном множестве программ, которые должны содержать достаточное количество распараллеливаемых циклов разных видов.

Рассмотрим небольшой пример ожидаемого распараллеливания на общую память на отрывке программы, реализующей алгоритм Якоби (см. Рис). Система SAPFOR должна успешно вставить все присутствующие директивы DVM-системы. При запуске анализа кода диалоговая система должна сообщить, что внешний цикл распараллелить нельзя (и указать причину), и что внутренние циклы распараллеливаются без препятствий.

```
1  ...
2      do  it = 1,itmax
3          eps = 0.
4  !DVM$ REGION
5  !DVM$ PARALLEL (j,i), PRIVATE (i,j),TIE (a(i,j),b(i,j)),
       ↪ REDUCTION (max (eps))
6          do  j = 2,l - 1
7              do  i = 2,l - 1
8                  eps = max (eps,abs (b(i,j) - a(i,j)))
9                  a(i,j) = b(i,j)
10             enddo
11         enddo
12 !DVM$ PARALLEL (j,i), PRIVATE (i,j),TIE (a(i,j),b(i,j))
13         do  j = 2,l - 1
14             do  i = 2,l - 1
15                 b(i,j) = calculate (a(i - 1,j) + a(i,j -
                       ↪  1) + a(i + 1,j) + a(i,j + 1))
16             enddo
17         enddo
18 !DVM$ END REGION
19         if (eps .lt. maxeps)  goto 3
20     enddo
21 3   continue
22 ...
```

Листинг 10: пример ожидаемого распараллеливания.

В дополнение к описанному, существует ещё несколько сценариев использования системы SAPFOR, при котором может пригодиться новый функционал. Во-первых, с по-

мощью режима распараллеливания на общую память можно будет распараллеливать программы, написаные на технологии MPI, что может дать дополнительное ускорение. Во-вторых, данный режим может быть полезен и при обычном распараллеливании с распределением данных: с его помощью можно предварительно оценивать программы на предмет потенциала к распараллеливанию, например заранее узнавать, какие циклы могут быть распараллелены, а какие нет. Используя полученную информацию, можно оценивать вероятность того, что распараллеливание даст положительный эффект ещё до создания схемы распределения данных.

2.4 Обзор существующих решений

Исследования по теме автоматизированного распараллеливания программ ведутся достаточно давно. На данный момент существует не так много широко используемых средств, позволяющих получать параллельный код. Среди средств, которые в той или иной степени помогают в процессе распараллеливания программ можно выделить следующие: Polaris, CAPO, WPP, SUIF, VAST/Parallel, OSCAR, ParallelWare, Intel Parallel Studio XE.

Однако, открытых реализаций алгоритмов распараллеливания почти нет, поэтому за основу решения был взят уже существующий в системе SAPFOR режим распараллеливания с распределением данных. Этот подход не только помог учесть спицифику языков фортран и DVM, но и позволил минимизировать объём внесённых в систему SAPFOR изменений.

3 Построение решения

Весь процесс построения решения можно разделить на три основных этапа – исследование кодовой базы системы SAPFOR, внесение необходимых правок и тестирование реализованного функционала.

3.1 Первый этап – подготовка

Основной целью данного этапа стал анализ внутреннего устройства системы SAPFOR, так как было принято решение реализовывать распараллеливание как часть её функционала. На данном этапе было проведено исследование исходного кода системы, которое дало ответы на следующие вопросы:

- как происходит запуск распараллеливания с распределением данных;
- какой код и в какой последовательни работает при распараллеливании с распределением данных;
- какие органичивающие проверки планируется убрать в проходе распараллеливания на общую память;
- какие структуры данных используются;
- использование каких структур данных нужно избежать;
- какие существующие алгоритмы (с можификациями или без) можно (и нужно) переиспользовать в добавляемом режиме;

Путём чтения исходного кода, документации и общения с авторами системы все эти вопросы были разрешены. Далее излагаются основные полученные сведения.

Как уже упомяналось ранее, структурно система SAPFOR состоит из множества алгоритмов, которые логически разделены на отдельные блоки – проходы. Каждый проход выполняет свою функцию и может зависеть от других проходов. При запуске

прохода перед ним запускаются все проходы, от которых он зависит непосредственно или транзитивно, при чём каждый проход запускается не больше одного раза. Таким образом в системе выстраивается дерево зависимостей проходов.

Конечным проходом при распараллеливании с распределением данных является проход с названием INSERT_PARALLEL_DIRS, который производит вставку созданных директив в код. Главные его зависимости – проход CREATE_PARALLEL_DIRS создания параллельных директив (без вставки) по полученной информации из анализа циклов. Анализ циклов производится следующей группой проходов:

- LOOP_ANALYZER_COMP_DIST;
- LOOP_ANALYZER_DATA_DIST_S2;
- LOOP_ANALYZER_DATA_DIST_S1;
- LOOP_ANALYZER_DATA_DIST_S0;

Они заполняют структуры, описывающие циклы, анализируют обращения к массивам внутри циклов, отображают обращения к массивам на циклы. Также они создают и заполняют структуру данных, позволяющую строить распределение данных – *граф измерений массивов*. Вместе с этим происходит вызов прохода CREATE_TEMPLATE_LINKS, который по построенному графу создаёт схему распределения данных. До них в работает ещё множество других проходов (всего порядка семидесяти), которые не относятся к распараллеливанию напрямую. Эти проходы не касаются распределения данных, поэтому они пристально не рассматривались. Часть дерева зависимостей прохода вставки параллельных директив изображена на Рис. 1:

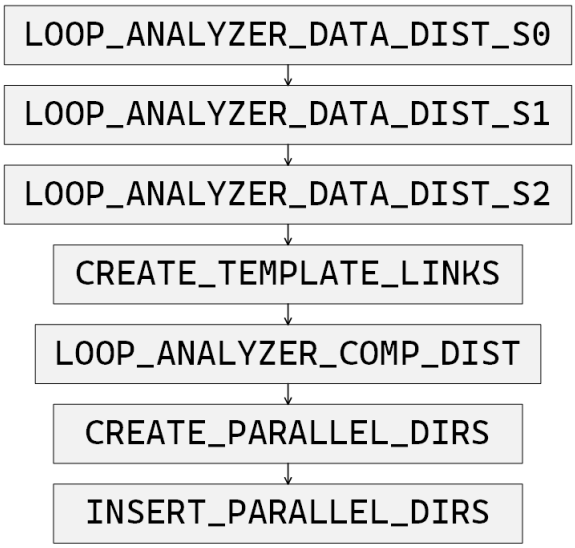

Рис. 1: упрощённая схема работы проходов при распараллеливании с распределением данных.

Таким образом, было установлено, что для успешной реализации распараллеливания на общую память необходимо и достаточно внести изменения в эти семь проходов.

3.2 Второй этап – реализация

Новый режим работы распараллеливания на общую память был оформлен как отдельных проход. По аналогии этот проход был назван INSERT_PARALLEL_DIRS_NODIST, что подчёркивает отсутвие построения распределения данных. Поскольку INSERT_PARALLEL_DIRS осуществляет только вставку созданных заранее директив, его функционал менять не пришлось, поэтому внутри INSERT_PARALLEL_DIRS_NODIST вызывается код прохода INSERT_PARALLEL_DIRS.

Само создание текста параллельных директив, которое происходило в проходе CREATE_PARALLEL_DIRS, подверглось правкам. Оттуда полностью убран код, отвечающий за директивы распределения данных. Также изменён код для констру-

ирования параллельных директив: убрано создание приставки ON в директиве и клауз удалённого доступа к данным, среди которых `SHADOW_RENEW` и `REMOTE_ACCESS`. Также был добавлен алгоритм заполнения клауз `TIE`. Оказалось, что новая версия CREATE_PARALLEL_DIRS стала намного компактнее и было решено не создавать для неё отдельный проход и сделать частью INSERT_PARALLEL_DIRS_NODIST.

Также, новый проход не унаследовал зависимости от проходов CREATE_TEMPLATE_LINKS и LOOP_ANALYZER_DATA_DIST_S*, поскольку они отвечали только за распределение данных.

Проход LOOP_ANALYZER_COMP_DIST пришлось переработать. Он отвечал за анализ обращений к массивам внутри цикла и распараллеливание на основе этой информации. Преобразованный проход получил название LOOP_ANALYZER_NODIST.

В результате схема зависимостей изменилась следующим образом:

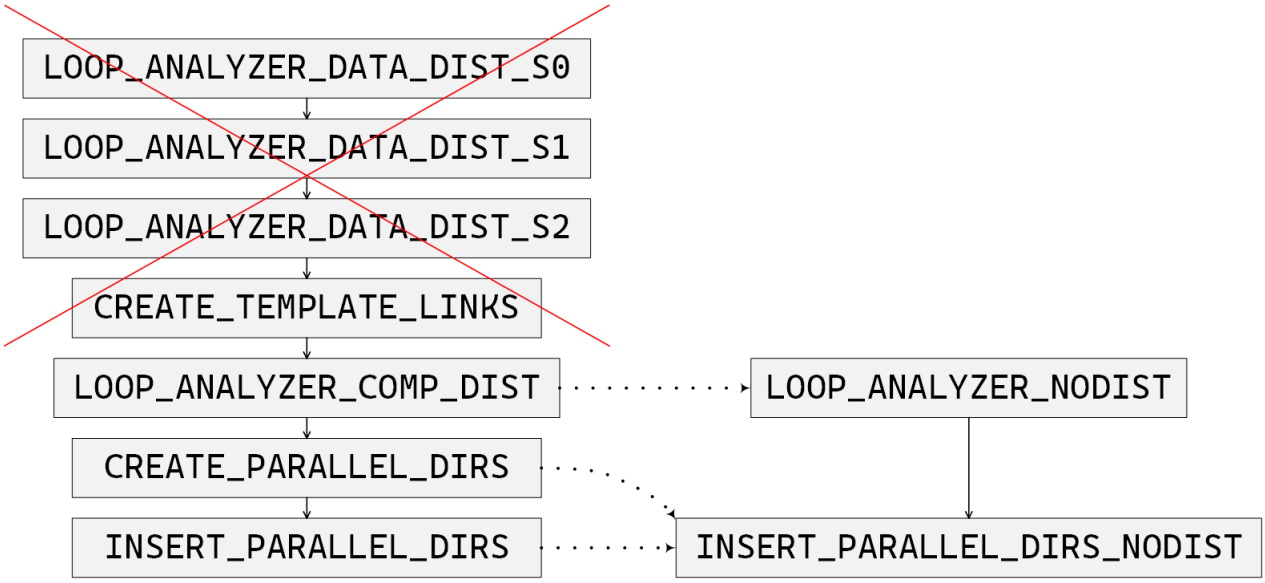

Рис. 2: сравнение схемы работы проходов режима распараллеливания с распределением данных (слева) с новым режимом распараллеливания на общую память (справа).

Далее, следует сказать о ещё одной немаловажной части работы – интеграции с инструментом для визуализации (диалоговой оболочкой) системы SAPFOR. Здесь тре-

бовалось реализовать два основных сценария работы. Первый основной сценарий – запуск распараллеливания на общую память. Требовалось предоставить интерфейс для вызова прохода визуализатором. С этим не возникло сложностей, так как для этого потребовалось написать простую функцию `SPF_SharedMemoryParallelization`, которую вызывает диалоговая оболочка и которая в свою очередь запускает проход INSERT_PARALLEL_DIRS_NODIST. Второй сценарий работы предполагал запуск анализа распараллеливания без фактической вставки директив. Аналгично запуску распараллеливания, была добавлена возможность вызывать из визуализатора проход LOOP_ANALYZER_NODIST.

3.3 Третий этап – тестирование и решение возникших проблем

Поскольку работа ведётся с исходным кодом фортран-программ, множество возможных вариантов входных программ слишком велико, чтобы предусмотреть абсолютно все ситуации на этапе реализации. Из-за этого было необходимо произвести тщательное тестирование, чтобы выявить основные случаи некорректного поведения системы SAPFOR.

Тестирование проводилось как на небольших модельных примерах, так и на больших примерах практически используемых программ. Суммарно за всё тестирование было обнаружено порядка тридцати различных примеров некорректного поведения добавленного режима. Далее описаны наиболее содержательные из найденных ошибок и то, как они были исправлены.

Первая из них касалась обработки приватных массивов. При распределении данных в DVM-системе запрещается распределять массивы, которые являются приватными хотябы для одного цикла. Поэтому в системе SAPFOR такие массивы не отображаются на циклы при построении схемы распределения данных. При распараллеливании на общую память DVM система допускает использование одного массива в качестве приватного и неприватного для разных циклов. Поэтому для режима распараллеливания на общую память было добавлено отображение на цикл всех неприватных для

него массивов (в частности, это нужно для заполнения клаузы `TIE`). Это привело к правкам в проходе LOOP_ANALYZER_NODIST.

Ещё одна проблема связана с обработкой передаваемых в процедуры секций массивов. Дело в том, что в DVM-системе запрещено использование в разных циклах пересекающихся по памяти различных секций массивов. Рассмотрим следующий пример:

```
 1 program p
 2     ...
 3     integer:: A(100)
 4     call foo(A)
 5
 6 !DVM$ REGION
 7 !DVM$ PARALLEL (i), TIE(A(i))
 8     do i = 1, 100
 9         A(i) = ...
10     endo
11 !DVM$ END REGION
12 end
13
14 subroutine foo(A)
15     integer:: A(50)
16
17 !DVM$ REGION
18 !DVM$ PARALLEL (i), TIE(A(i))
19     do i = 1, 50
20         A(i) = ...
21     endo
22 !DVM$ END REGION
```

```
23 end
```

Листинг 11: пример, вызывающий ошибку выполнения.

В нём нём есть два параллельных цикла (строки 8 и 19). Один из них использует полный массив A (который объявлен на строке 3). Другой же использует секцию массива A, которая содержит его первые 50 элементов (объявление на строке 15). Запуск этого примера приводит к ошибке выполнения системы DVM. При этом если бы в подпрограмме `foo` было объявлено, что массив имеет размер 100 (то есть фактически передавался бы массив целиком), то ошибки бы не было.

В случае аналогичного распараллеливания с распределением, ошибок выполнения DVM не возникает. Из-за этого в цепь проходов распараллеливания на общую память был добавлен новый проход, который получил название SELECT_ARRAY_DIM_CONF. Он запускается после анализа циклов и фильтрует параллельные циклы так, чтобы в них не было пересечений по памяти используемых массивов.

Таким образом схема проходов нового режима получила окончательный вид, представленный на Рис. 3:

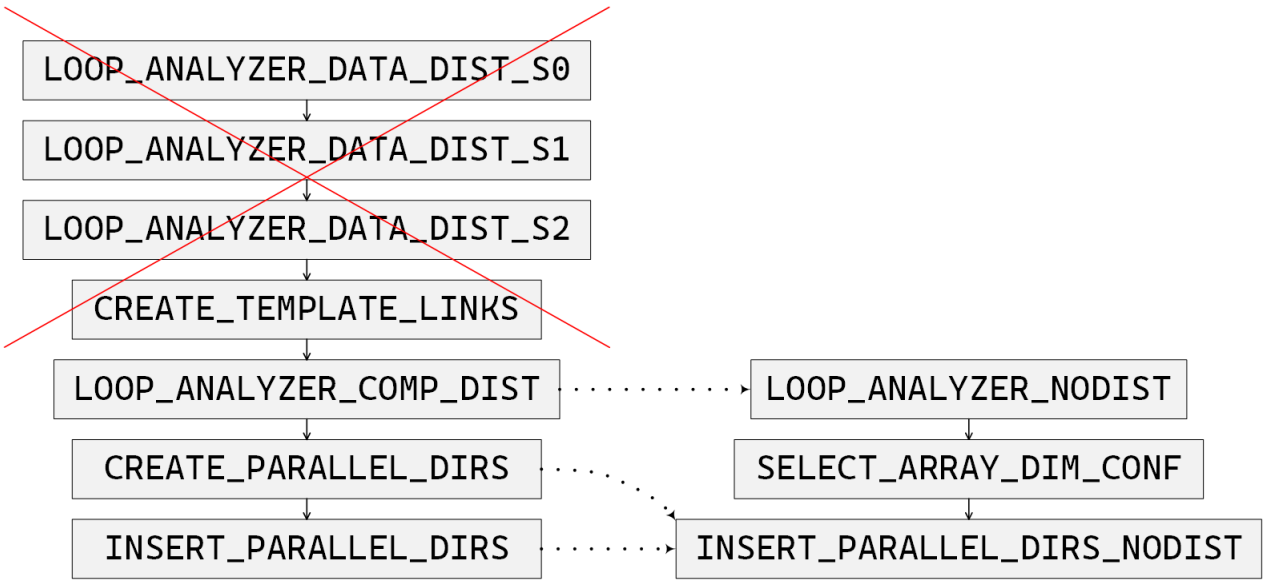

Рис. 3: итоговая схема работы проходов при распределении на общую память.

4 Описание алгоритмов

В предыдущей главе поверхностно описывались функции проходов, работающих при распараллеливании с распределением данных и их аналогов для случая общей памяти. В этой главе приводится подробное описание алгоритмов, с помощью которых реализованы эти проходы.

Весь исходный код системы SAPFOR написан на языке C++. Программа, поступающая на вход системе, переводится в абстрактное синтаксическое дерево операторов (АСД). В качестве реализации абстрактного синтаксического дерева используется библиотека Sage++ [6]. С её помощью входную программу можно анализировать, модифицировать и переводить обратно в код на фортране.

Помимо структур библиотеки Sage++, в системе SAPFOR введён ряд типов данных, упрощающих разработку. Среди них можно выделить следующие наиболее используемые:

- `Array` – тип, отражающий массив входной программы. Помимо прочего, содержит информацию о его имени, размерности, определении, местах использования;
- `FuncInfo` – отражает процедуру входной программы. Содержит информацию о названии, определении, точках её вызова, подключенных common-блоков, именах её формальных аргументов, информацию о наличие побочных эффектов и т. д.;
- `LoopGraph` – отражает цикл. Имеет информацию о месте расположения, вложенных циклах, вызовах функций внутри цикла, операциях над массивами внутри цикла. Если цикл находится в канонической форме, дополнительно содержит название итерационной переменной, границах и шаге итерирования, глубину гнезда тесновложенных циклов. Кроме этого структура содержит набор флагов, отмечающих интересующие с точки зрения распараллеливания свойства;
- `ParallelDirective` – структура, отражающая созданную директиву распараллеливания. Содержит информацию о размере распараллеливаемого гнезда, о кла-

узах `SHADOW_RENEW`, `ACROSS`, `REMOTE_ACCESS`, `REDUCTION`, `PRIVATE` и используемых в них массивах;

4.1 Алгоритмы распараллеливания с распределением данных

Рассмотрим подробнее устройство проходов, отвечающих за построение схемы распределения данных и распараллеливании (см. Рис. 1).

Начнём с прохода LOOP_ANALYZER_DATA_DIST_S0. На вход он получает АСД, заполненные структуры `FuncInfo` и `LoopGraph` для всей программы. Результатом данного прохода является заполненный граф измерений массивов. В проходе происходит итерация по всем структурам `FuncInfo`. Для каждой процедуры производятся следующие действия:

- выполняется проход по всем операторам процедуры;
- если этот оператор содержит запись в элемент массива или чтение из массива и этот массив распределяемый, то есть он не является нигде приватным или редукционным, происходит отображение этой операции на содержащие её циклы;
- отображение заключается в сохранении записей вида *<цикл, массив, тип операции, номер измерения, A, B>* во временную структуру. Такая запись создаётся для каждого измерения, обращение по которому в текущей операции имеет вид $A*i+B$, где A и B - целые числа, i - счётчик цикла;
- далее происходит добавление полученных записей об обращениях к массив в специальную структура данных – граф измерений массивов. Эта структура используется для построения схемы распределения данных и представляет собой граф, вершинами которого являются измерения массивов, в дугами – связи между измерениями массивов согласно их использованию в циклах программы (см. пример на Рис. 4). Подробно построение графа массивов описано в [7];
- после обработки всех функций проход завершает работу;

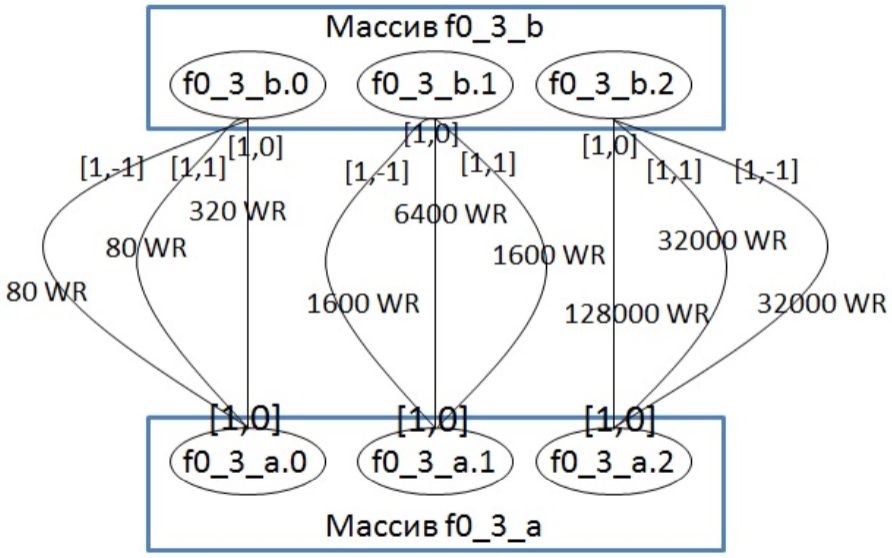

Рис. 4: пример графа измерений массивов.

Далее запускается проход LOOP_ANALYZER_DATA_DIST_S1. Входные данные у него такие же, как и у LOOP_ANALYZER_DATA_DIST_S0. Его цель – собрать информацию о зависимостях по данным в циклах и записать её в поля структуры `LoopGraph`.

В этом проходе также происходит итерация по всем процедурам программы и производятся следующие действия:

- опять выполняется проход по всем операторам процедуры;
- точно так же обрабатываются обращения к массивам с сохранением записей вида *<цикл, массив, тип операции, номер измерения, A, B>*;
- вместо добавления информации в граф измерений массивов, выявляются зависимости по данным внутри циклов путём построения специального графа зависимостей;

- для каждого цикла, который имеет зависимости по данным, заносится пометка в структуру `LoopGraph` о том, что цикл имеет зависимость по данным и не может быть распараллелен;

- исключения составляют зависимости фиксированной длинны, то есть такие, что номера записывающих и читающих витков отличаются на фиксированную величину. Такие циклы могут быть распараллелены с помощью директивы ACROSS (см. пример на Рис. 5 в главе 5);

- после обработки всех функций проход завершает работу;

Проход LOOP_ANALYZER_DATA_DIST_S2 получает на вход граф измерений массивов. Результатом прохода является усечённый граф измерений массивов (такой граф измерений массивов, который не содержит циклов, порождающих конфликтные ситуации) для последующего создания распределения данных. В основе лежат алгоритмы поиска простыв циклов в графе и построения минимального остовного дерева.

Проход CRATE_TEMPLATE_LINKS получает на вход оптимизированный граф измерений массивов, в котором отсутствуют циклы. Поскольку в этом графе нет циклов, все его компоненты связности представляют собой деревья. В каждом таком дереве находится массив с наибольшей размерностью и по нему создаётся служебный массив-шаблон, на который выравниваются все массивы рассматриваемого дерева. Таким образом, на вход этот проход принимает граф измерений массивов, а на выходе получаются директивы распределения и выравнивания данных, представленные структурой `DataDirective`, хранящей всю необходимую информацию.

После этого запускается проход LOOP_ANALYZER_COMP_DIST распределения вычислений согласно построенному распределению данных. На вход он получает АСД программы, получает АСД, структуры `FuncInfo`, `LoopGraph`, граф измерений массивов и построенное распределение данных. Общая схема работы совпадает с LOOP_ANALYZER_DATA_DIST_S0 и LOOP_ANALYZER_DATA_DIST_S1: происходит итерация по всем функциям и заполнение кортежей *<цикл, массив, тип*

операции, номер измерения, A, B>. После заполнения этой информации, происходят следующие действия:

- по отдельности рассматривается каждый цикл;
- по группе флагов структуры `LoopNode` определяется, есть ли факторы, препятствующие распараллеливанию цикла;
- если нет, среди всех массивов, используемых в цикле, выбирается лучший для того, чтобы относительно него распределить витки циклы. Предпочтение отдаётся массивам, в которые происходит запись и которые имеют наибольшее количество измерений. Если таких нет, выбираются массивы, из которых происходит чтение;
- для каждого цикла без ограничений на распараллеливание создаётся структура типа `ParallelDirective`, в которую записывается выбранный массив и другая информация, такая как ACROSS-зависимости и др., извлекаемая в том числе из графа измерений массивов;
- полученные структуры `ParallelDirective` сохраняются в соответствующих структурах `LoopNode`;
- после создание всех директив, для каждого тесновложенного гнезда параллельных циклов происходит объединение `ParallelDirective` для того, чтобы распараллеливался не только самый верхний цикл, а всё гнездо;

Следом вызывается проход CREATE_PARALLEL_DIRS. На вход он получает построенные директивы распределения данных и директивы распараллеливания циклов в виде структур `DataDirective` и `ParallelDirective`.

Сначала рассматриваются директивы распределения данных. Для каждой директивы, по информации из структуры, строится её текстовое представление: в виде строк конструируются DVM-директивы `ALIGN` и `DISTRIBUTE`. Далее аналогичный процесс происходит с директивами распараллеливания: формируются директивы `PARALLEL`

`ON` и все нужные клаузы, такие как `PRIVATE`, `REDUCTION`, `ACROSS`, `SHADOW_RENEW`, `REMOTE_ACCESS`. Текстовые директивы DVM-системы сохраняются в виде множества структур `CreatedDirective`, которые хранят текст директивы и строку, перед которой её необходимо будет вставить.

Наконец, проход INSERT_PARALLEL_DIRS, получающий множество созданных директив, производит их вставку в виде комментариев в АСД обрабатываемой программы перед соответствующими операторами.

4.2 Алгоритмы распараллеливания на общую память

Как было описано ранее, проходы LOOP_ANALYZER_DATA_DIST_S* и CREATE_TEMPLATE_LINKS отвечают сугубо за распределение данных, поэтому работа по распараллеливанию на общую память начинается сразу с аналога прохода LOOP_ANALYZER_COMP_DIST, который получил название LOOP_ANALYZER_NODIST.

На вход он также принимает АСД программы, множество структур `LoopGraph` и `FuncInfo` (но без графа измерений массивов). Отличия в работе начинаются со способа построения кортежей *<цикл, массив, тип операции, номер измерения, A, B>*: при распределении данных такие записи не сохранялись для массивов, которые являются приватными в каком либо цикле программы. При распараллеливании на общую память есть возможность снять это ограничение и отображать на цикл все его неприватные массивы, даже если для других циклов они являются приватными.

Далее, появилась возможность обойти отображение витков цикла на распределяемый массив. Преобразованный алгоритм не запускает поиск наилучшего распределённого массива для распределения вычислений и не заполняет соответствующее поле в структуре `ParallelDirective`, хотя заполнение других полей, таких как ACROSS-зависимости, остаёься актуальным.

На следующем этапе работает новый проход SELECT_ARRAY_DIM_CONF решающий проблему распараллеливания циклов, использующих секции массивов.

Подробно проблема описывалась в предыдущей главе. На вход алгоритм принимает множество структур `LoopNode` с заполенными полями параллельных директив `ParallelDirective`. Так же передаётся *граф связей между массивами*. Вершинами этого ориентированного графа являются структуры `Array`. Дуги строятся по следующему правилу: пусть массив B является формальным аргументом процедуры F. Тогда в графе существует дуга от массива A к B в том и только том случае, если есть вызов процедуры F с предачей секции массива A в качестве фактического аргумента для B.

Перед тем как рассматривать работу алгоритма, введём два понятия. Будем называть масив *главным*, если он не является формальным аргументом функции, в которой он определён. Заметим, что в графе связей между массивами главными будут те и только те, которые не имеют входящих дуг. Также назовём *конфигурацией* n-мерного массива кортеж из n чисел a_i, в котором a_i является длинной i-го измерения.

Алгоритм работает следующим образом:

- для каждого главного массива строится множество конфигураций массивов, достижимых из главного в графе связей массивов;
- для каждого главного массива выбирается лучшая конфигурация – такая, которая соовтветствует наибольшему суммарному количеству элементов (суммарное кол-во элементов вычисляется как произведение элементов кортежа);
- рассматриваются все невыбранные конфигурации. Все циклы, которые используют хоть один массив, конфигурация которого не была выбрана на прошлом шаге, отстраняются от распараллеливания путём поднятия специального флага `hasAccessToSubArray` в структуре `LoopGraph` и удаления в ней директивы распараллеливания, если она была;

Таким образом, после данного прохода от распараллеливания будут отстранены циклы, которые могут спровоцировать ошибку выполнения системы DVM.

Далее в проходе INSERT_PARALLEL_DIRS_NODIST запускается алгоритм получения текстового представления директив распараллеливания `ParallelDirective`,

который имет следующие отличия от случая распараллеливания с распределением:

- директивы распределения данных не используются проходом и не обрабатываются;
- в параллельных директивах не происходит конструирование клаузы `ON`;
- не происходит конструирование клауз `SHADOW_RENEW` и `REMOTE_ACCESS`;
- добавлено конструирование клауз `TIE`. Для заполнения списка отображаемых массивов используется сохранённая информация об обращениях к массивам с прохода LOOP_ANALYZER_NODIST;

В результате получается множество созданных директив `CreatedDirective`, содержащее только директивы распараллеливания на общую память.

Завершает распараллеливание на общую память вставка директив путём вызова кода прохода INSERT_PARALLEL_DIRS с передачей множества созданных директив `CreatedDirective`.

5 Исследование эффективности

Немаловажную роль в оценке качества выполненной работы играет тестирование на предмет эффективности получаемого распараллеливания. Для этих целей был выбран специальный пакет программ, который широко используется в сфере исследования эффективности распараллеливания – пакет *NAS Parallel Benchmarks* [8].

5.1 NAS Parallel Benchmarks

NAS Parallel Benchmarks – это пакет из десяти программ, разработанных для тестирования производительности многопоточных вычислительных систем, восемь из них написаны на фортране, остальные – на языке Си. Тесты на фортране включают следующие программы:

- BT, SP, LU – **B**lock **T**ri-diagonal, **S**calar **P**entadiagonal, **L**ower-**U**pper. Решают синтетическую систему нелинейных диффиренциальных уравнений в частных производных (3-мерная система уравнений Навье — Стокса для сжимаемой жидкости или газа), используя три алгоритма: блочная трёхдиагональная схема с методом переменных направлений (BT), скалярная пятидиагональная схема (SP) и метод симметричной последовательной верхней релаксации (алгоритм SSOR, задача LU);
- CG – **C**onjugate **G**radient. Приближение к наименьшему собственному значению большой разреженной симметричной положительно-определённой матрицы с использованием метода обратных итераций вместе с методом сопряжённых градиентов в качестве подпрограммы для решения СЛАУ;
- EP – **E**mbarrassingly **P**arallel. Генерация независимых нормально распределённых случайных величин;
- FT – **F**ourier **T**ransform. Решение трёхмерного уравнения в частных производных при помощи быстрого преобразования Фурье;

- MG – **M**ulti-**G**rid. Аппроксимация решения трёхмерного дискретного уравнения Пуассона при помощи V-циклового многосеточного метода;
- UA – **U**nstructured **A**daptive mesh. Решение уравнения теплопроводности с учётом диффузии и конвекции в кубе. Источник тепла подвижен, сетка нерегулярна и меняется каждые 5 шагов;

Каждая программа пакета имеет параллельные версии на OpenMP и на MPI. Кроме того, некоторые тесты имеют также несколько вариантов распараллеливания. Так, например, для теста LU можно выбрать конвеерное распараллеливание, распараллеливание по гиперплоскостям и DO-ACROSS версию. Это распараллеливание было произведено создателями пакета, профессионалами в области параллельных вычислений, поэтому было принято решение взять эти версии за эталон распараллеливания.

Также этот пакет позволяет запускать все тесты на разных *классах*, что даёт возможность подбирать для конкретной вычислительной системы оптимальный размер задачи, чтобы тест работал не слишком быстро и не слишком долго. Для удобства в каждую программу встроены средства для замера времени работы теста и самопроверка результатов вычислений.

5.2 Процесс распараллеливания

Часто для того, чтобы получить эффективный параллельный вариант программы с помощью системы SAPFOR, требуется провести ряд преобразований исходного кода. При распараллеливании тестов NAS приходилось совершать много преобразований трёх видов: слияние файлов, подстановка процедур, указание приватных и редукционных массивов.

Слияние файлов представляет собой несложное преобразование, объединяющие все файлы программы в один. Это преобразование нужно из-за проблем при раздельной компиляции фортран-файлов системой DVM. При помощи диалоговой оболочки это преобразование можно легко делать «в один клик».

В некоторых местах распараллеливание циклов осложнено наличием в теле цикла вызовов процедур. Чтобы распараллелить такие циклы, можно попытаться подставить тело процедуры вместо её вызова. Так как в диалоговой оболочке есть проходы анализа кода и подстановки процедур, можно также без больших усилий найти циклы, которые не распараллеливаются из-за вызовов процедур и произвести подстановку в нужных местах.

Нередко в коде встречаются приватные и редукционные массивы. Так как система SAPFOR в данный момент не поддерживает автоматическое обнаружение таких массивов, их приходится вручную указывать при помощи директив `!$SPF ANALYSIS (PRIVATE)` и `!$SPF ANALYSIS (REDUCTION)`.

Отдельного внимания потребовала программа LU. Основная вычислительная нагрузка программы приходится на алгоритм SSOR (метод симмтричной последовательной верхней релаксации). Он представляет собой пару гнёзд циклов глубины два. Рассмотрим только первое из этих гнёзд, так как второе распараллеливается полностью аналогично. Возможности параллельного выполнения витков этого цикла мешает только зависимость по данным массива `rsd`: каждая итерация с номером *(i, j)* использует элементы, которые вычисляются витками *(i - 1, j)*, *(i, j - 1)*. В такой ситуации витки имеют частичный порядок и некоторые группы витков могут выполняться параллельно. В пакете есть три реализации такого распараллеливания:

- версия с использованием так называемого *конвейерного параллелизма*. В ней поддержание корректной последовательности витков цикла осуществляется за счёт использования служебного синхронизационного массива и примитивов синхронизации OpenMP `atomic read` и `atomic write`;
- в другой версии, которая называется DO-ACROSS, используется похожая схема, только синхронизация реализована механизмом упорядоченного выполнения витков цикла. Для этого в OpenMP есть специальная директива ORDERED;
- последняя версия использует *параллелизм по гиперплоскостям*. При наличии та-

кой зависимости, витки с номерами *(i, j)*, у которых сумма $i + j$ совпадает, образуют гиперплоскости и могут выполняться параллельно (см. Рис. 5);

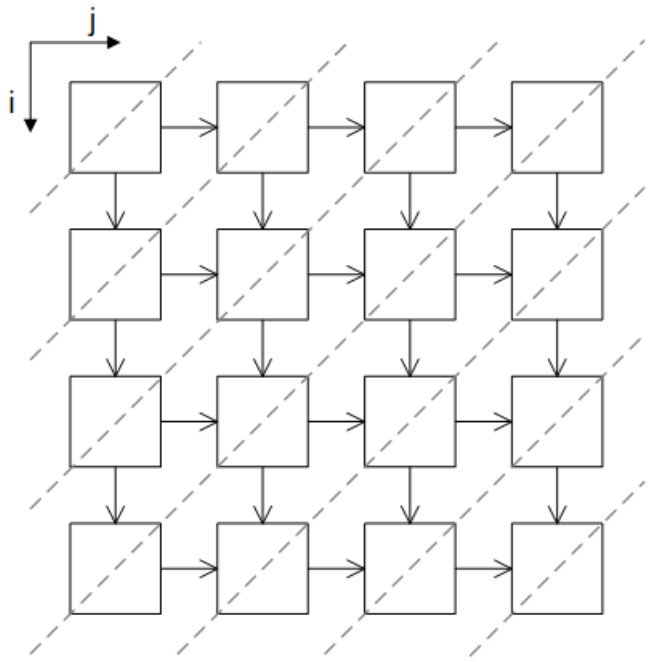

Рис. 5: разделение итерационного пространства на гиперплоскости. Внутри одной гиперплоскости элементы могут считаться параллельно.

В силу того, что SAPFOR не способен проанализировать логику синхронизаций и зависимости между разными гиперплоскостями, первый и третий варинты распараллелить не удалось. Однако во втором варианте система SAPFOR успешно распознала зависимость по данным и распараллелила цикл с использованием клаузы `ACROSS`:

```
1 ...
2 !DVM$    REGION
3 !DVM$    PARALLEL (k,j), PRIVATE (...), TIE(..., rsd(*,*,
    ↪ j,k)), ACROSS (rsd(0:0,0:0,1:0,1:0))
4           do  k = 2,nz - 1
```

```
5               do  j = jst,jend
6                   ...
7               enddo
8           enddo
9  !DVM$    END REGION
10 ...
```

Листинг 12: пример вставленной клаузы ACROSS, задающую зависимость размера 1 по последним двум измерениям массива `rsd`. В теле цикла происходит чтение элементов массива `rsd(*,*,j-1,k)`, `rsd(*,*,j,k-1)`, `rsd(*,*,j-1,k-1)` и запись в элементы массива `rsd(*,*,j,k)`.

Таким образом, в результате автоматизированного распараллеливания были получены параллельные версии тестов BT, CG, EP, FT, SP, LU. Остальные тесты (то есть MG и UA) распараллелить не удалось, даже при помощи преобразований. Причиной этому служит наличие в этих тестах нетривиальной логики, которая не позволяет системе SAPFOR производить распараллеливание.

5.3 Результаты запусков

После получения распараллеленных на общую память версий тестов был проведен их запуск на различных вычислительных устройствах. Произведено сравнение полученных версий с параллельными версиями на OpenMP. Запуск на процессоре Intel Core i7 980x (6 ядер, 12 потоков, компилятор mpiifort с уровнем оптимизации -O3) показал следующие результаты (см. Рис. 6 и 7):

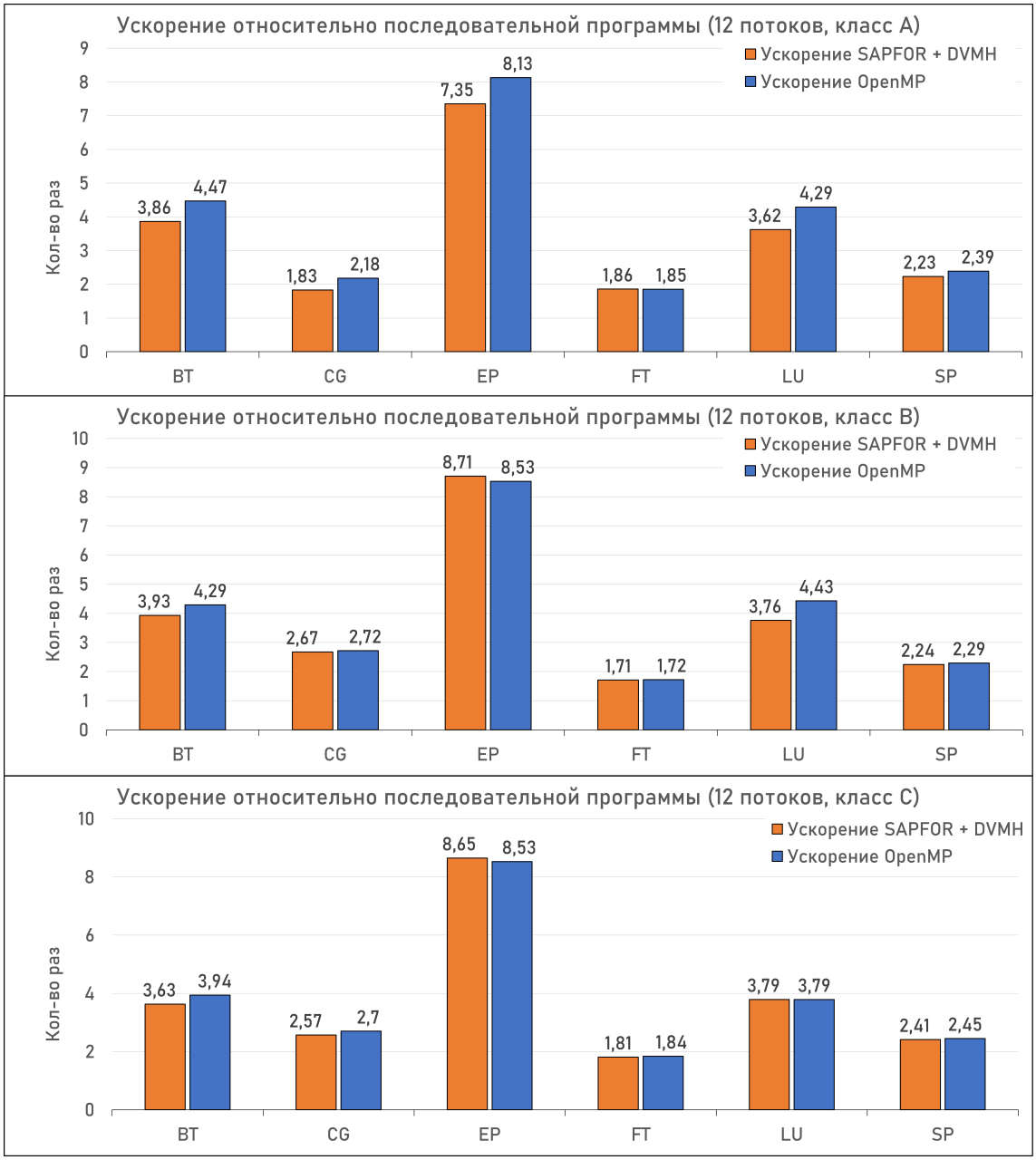

Рис. 6: замер времени работы полученных версий на DVMH и эталонных версий на OpenMP на процессоре Intel Core i7 980x (12 потоков).

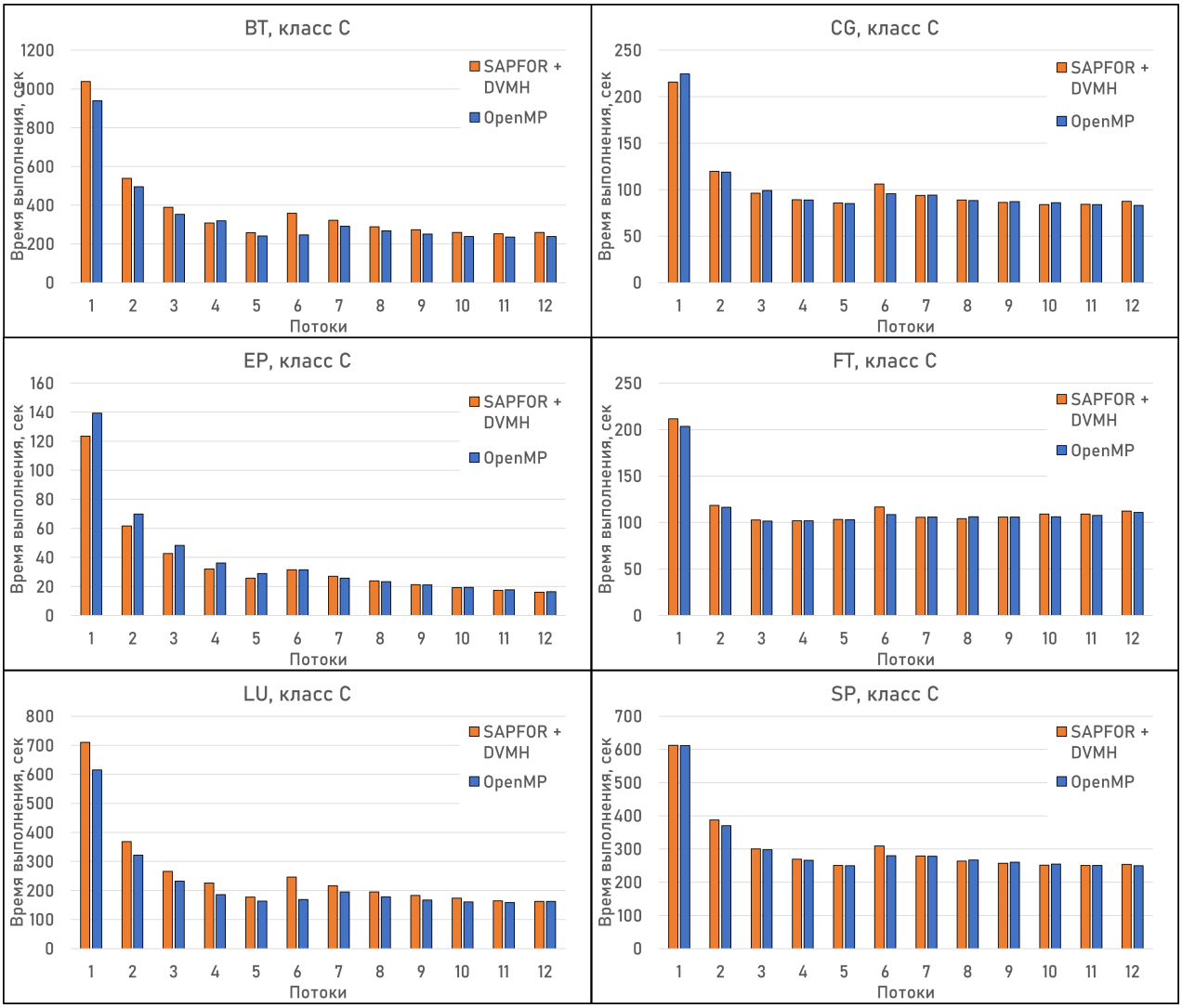

Рис. 7: сравнение времени работы тестов на различном количестве потоков на процессоре Intel Core i7 980x (класс C).

Исследуя данные графики, можно сделать следующие выводы:

- сами по себе тесты ускоряются не одинаково: тесты CG, FT, SP дают максимальное ускорение примернов два раза, BT и LU – в четыре с половиной, EP – в восемь с половиной;

- на одном и том же классе разные тесты сильно отличаются по времени работы (например, тест BT на 12 потоках работает около 400 секунд, а EP - около 20ти секунд);
- разрыв между DVM-версиями и OpenMP-версиями на классе A составляет около 10-20% в пользу OpenMP; на тесте FT разрыва нет вовсе;
- с ростом размера задач разрыв уменьшается до 0-5% на классе C;
- на классе C тесты FT, LU, SP показывают одинаковое ускорение, на BT и CG сохраняется замедление DVM-версий, DVM-версия EP ускоряется немного лучше, чем OpenMP-версия;
- время работы DVM-версий и OpenMP-версий на различном количестве используемых потоков также отличается несущественно, с ростом количества потоков это различие уменьшается;
- есть примеры, на которых DVM-версии быстрее, чем версии на OpenMP и наоборот;

Дополнительно были получены результаты работы на графических ускорителях. При тестировании использовались видеокарты Nvidia GeForce GTX 1660 Ti и Nvidia GeForce GTX 1050 Ti.

В процессе запусков возникли проблемы с тестом FT. Параллельная версия этого теста содержит приватные массивы больших размеров, что приводило к переполнению памяти видеокарты. Поэтому было принято решение отказаться от запуска программы FT на ускорителях.

Результаты остальных запусков представлены на Рис. 8:

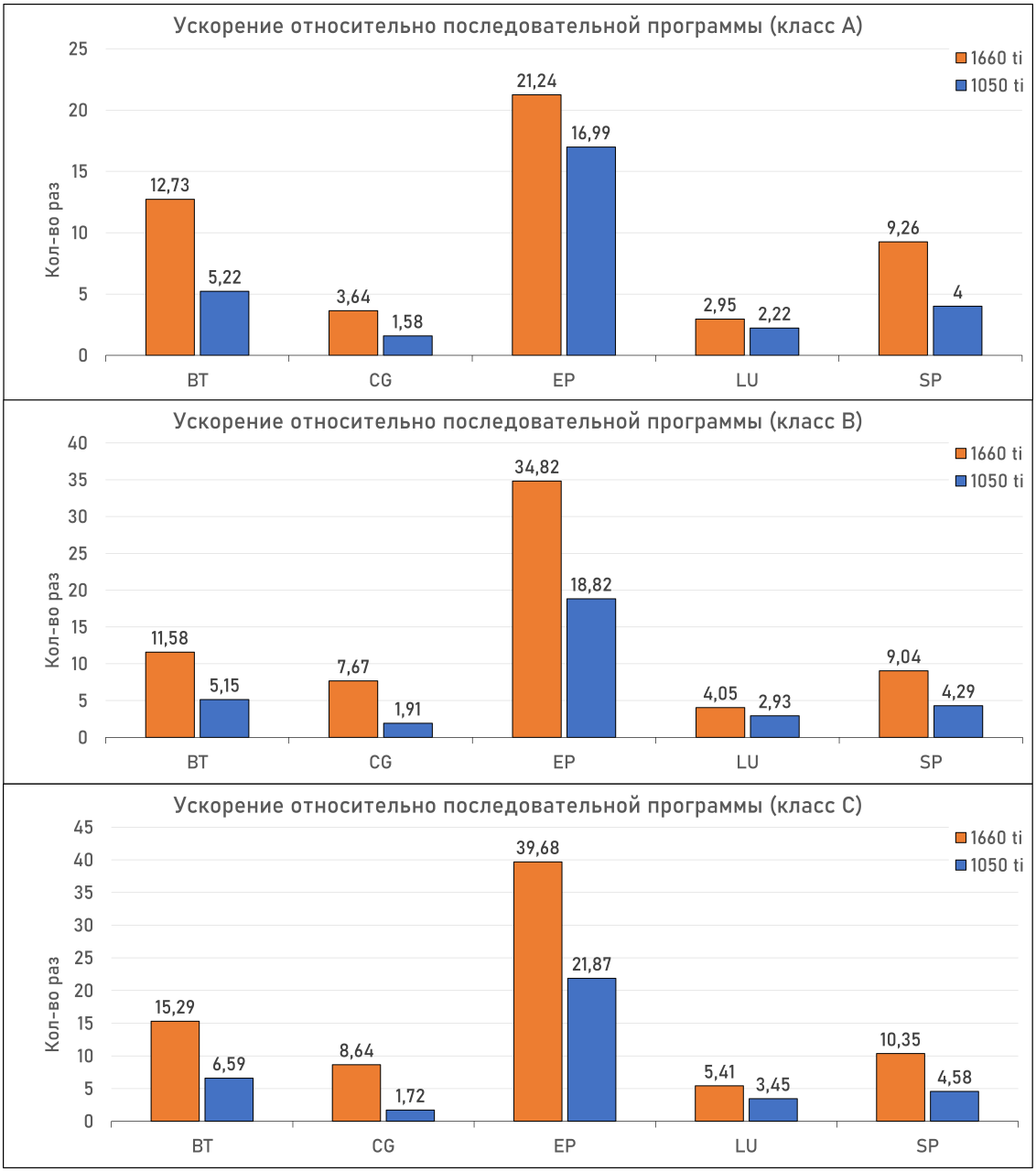

Рис. 8: демонстрация ускорения работы полученных версий на ускорителях Nvidia относительно запуска последовательной программы на процессоре Intel Core i7 980x.

Относительно результатов работы полученных версий на видеокартах спаведливы

следующие факты:

- тест BT на видеокарте ускорился в 15,3 раз (против 3,6 раз на 12ти потоках на классе C);
- CG – ускорение в 8,6 раз против 2,7;
- EP – ускорение в 39,7 раз против 8,5 – самое высокое ускорение;
- LU – ускорение в 5,4 раз против 3,8 – видеокарта почти не даёт выигрыш по сравнению с многопоточным процессором;
- SP – ускорение в 10,4 раз против 2,5;
- с ростом мощности видеокарты растёт и ускорение (модель GeForce GTX 1660 Ti современней и мощней чем GeForce GTX 1050 Ti);

6 Заключение

Итак, была проведена работа по исследованию и улучшению системы автоматизированного распараллеливания SAPFOR. В качестве разультата в систему SAPFOR был добавлен новый режим работы распараллеливания на общую память, что позволило расширить класс распараллеливаемых программ.

Разработанный проход был должным образом интегрирован в систему SAPFOR. Были поддержаны как директивы системе SAPFOR, так и взаимодействие с диалоговой оболочкой, в частности функция запуска прохода анализа кода. Добавленный проход использует как преобразованные версии уже существоваших алгоритмов, так и новые решения.

Добавленный код был тщательно протестирован. Было выполнено тестирование на большом множестве различных программ на предмет корректности получаемого параллельного DVMH-кода. Также с помощью нового прохода были распараллены тесты из пакета NAS Parallel Benchmarks.

Распараллеливание программ из пакета NAS Parallel Benchmarks вызвало некоторые трудности, большинство из которых удалось преодолеть с помощью графической оболочки системы SAPFOR. В результате для большей части тестов были получены эффективные параллельные версии на языке DVM, конкурирующие с эталонными версиями на OpenMP. В доказательство их эффективности были приведены результаты запусков на различных устройствах: на многопоточном процессоре и на графических ускорителях.

Результаты работы были представлены на конференции *«Ломоносовские чтения 2024»* и опубликованы в её сборнике тезисов [9].

Таким образом, все поставленные цели были выполнены, что дало решение исходной задачи автоматизации распараллеливания на общую память.

Список литературы

[1] сайт Института прикладной математики им. М.В. Келдыша Российской академии наук. https://www.keldysh.ru/.

[2] сайт DVM-системы. http://dvm-system.org/.

[3] *Н.А. Катаев, С.А. Черных.* Автоматизация распараллеливания программ в системе SAPFOR / С.А. Черных Н.А. Катаев // Научный сервис в сети Интернет: труды XXII Всероссийской научной конференции. — 2020. https://doi.org/10.20948/abrau-2020-24.

[4] *Катаев Н.А., Колганов А.С.* Автоматизированное распараллеливание программ для гетерогенных кластеров с помощью системы SAPFOR / Колганов А.С. Катаев Н.А. // Вычислительные методы и программирование. — No. 4. — 2022. https://num-meth.ru/index.php/journal/article/view/1246/1214.

[5] *А.С., Колганов.* Опыт применения механизма областей для поэтапного распараллеливания программных комплексов с помощью системы SAPFOR / Колганов А.С. // Вычислительные методы и программирование. — No. 4. — 2020. https://num-meth.ru/index.php/journal/article/view/1073/1131.

[6] Библиотека Sage++. http://www.extreme.indiana.edu/sage/.

[7] *Колганов, А.С.* Автоматизация распараллеливания Фортран-программ для гетерогенных кластеров / А.С. Колганов. http://TODO.

[8] NAS Parallel Benchmarks. https://www.nas.nasa.gov/software/npb.html.

[9] *Крюков В. А. Колганов А. С., Кочармин М. Д.* Автоматизированное распараллеливание Фортран-программ на общую память / Кочармин М. Д. Крюков В. А., Колганов А. С. // Ломоносовские чтения. — 2024.

https://conf.msu.ru/file/event/8752/eid8752_attach_
a7f03100cf01a40d0a1bc490c7691661e89e5edc.pdf.