УДК 004.021, 004.023 **DOI: 10.14529/cmse200202**

МЕТОДЫ ОПТИМИЗАЦИИ ОБОБЩЕННЫХ ТЕНЗОРНЫХ СВЕРТОК

© 2020 Р.А. Гареев

Уральский федеральный университет им. Б.Н. Ельцина
(620002 Екатеринбург, ул. Мира, д. 19)
E-mail: gareevroman@gmail.com
Поступила в редакцию: 18.03.2020

Свертка тензоров является одной из основных операций «Тензорного исчисления» — отдельного раздела математики, ставшего основным языком для описания фундаментальных законов таких областей науки, как теория относительности, механика, электродинамика и физика твердого тела. Эффективность выполнения свертки тензоров и ее обобщений имеет существенную практическую значимость для таких областей, как решение задач математической физики, машинного обучения, в спектральных методах, в квантовой химии, при интеллектуальном анализе данных, в высокопроизводительных вычислениях на многопроцессорных системах и др. В последние двадцать лет количество методов оптимизации тензорных сверток значительно увеличилось и продолжает возрастать. В статье представлен обзор активно используемых подходов к оптимизации свертки тензоров, применяемых при решении прикладных задач на однопроцессорных и многопроцессорных вычислительных системах с распределенной памятью. В работе представлены методы оптимизации важных частных случаев свертки тензоров — матричного и матрично-векторного произведения, использующихся для большинства оптимизаций сверток тензоров. Описанные оптимизации могут применяться в процессе компиляции программ, выполняемой промышленными компиляторами. Представленная информация может помочь при систематизации уже имеющихся знаний.

Ключевые слова: свертка тензоров, линейная алгебра, высокопроизводительные вычисления.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Гареев Р.А. Методы оптимизации обобщенных тензорных сверток // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 2. С. 19–39. DOI: 10.14529/cmse200202.

Введение

Развитие вычислений с тензорами привели к созданию «Тензорные исчисления», отдельного активно развивающегося раздела математики, использующегося почти во всех областях механики, теоретической физики и прикладной математики. Можно выделить восемь основных операций, выполняемых над тензорами: сложение и вычитание, умножение на число, умножение тензоров, свертка, перестановка индексов, симметрирование и альтернирование. Каждая из перечисленных операций имеет широкое применение в различных научных дисциплинах [1].

Операция свертки тензоров (tensor contraction, TC) применяется в алгоритмах машинного обучения, таких как обучение и логический вывод в глубинных нейронных сетях (Deep Neural Networks, DNN) [2]. Квантовая химия широко использует TC в методах Хартри—Фока, а также при решении других задач, имеющих существенную практическую знчимость [3]. TC применяется при моделировании потоков несжимаемой жидкости на основе спектральных методов [4]. TC может быть использована для вычисления многомерного преобразования Фурье. В частности, трехмерное дискретное преобразование Фурье может быть вычислено через TC тензоров, полученных в результате применения двумерных дискретных преобразований Фурье [3].

Умножение матриц (matrix-matrix multiplication, MMM) и умножение матрицы на вектор (matrix-vector multiplication, MVM) могут рассматриваться как частные случаи ТС, широко применяющиеся при решении прикладных задач. Примером использования MMM служит шифрование баз данных с последующим интеллектуальным анализом данных (Data Mining) [5]. MMM вычисляется во время обрезки весов (weight pruning), использующимся в машинном обучении [6]. Энергия корреляции молекул, рассматриваемая в квантовой химии, рассчитывается с использованием MMM [7]. MMM служит основой для большинства операций с матрицами, изучаемых в высокопроизводительных вычислениях [9]. В качестве примеров применения MVM можно привести обрезку весов, используемую в машинном обучении [10]; анализ графов [11]; расчет коэффициентов скорости реакции [12]. Большая часть операций с векторами реализована с использованием MVM [13–15]. MVM применяется при решении задач математической физики численными методами, в частности, при решении задач математической геофизики. Например, MVM успешно используется при решении задач гравиметрии или магнитометрии о нахождении поверхностей раздела сред по гравитационным или магнитным данным методом Левенберга—Марквардта [8].

ТС может быть обобщена на произвольные полукольца матриц для решения задач о путях [16] (нахождение кратчайших путей, построение минимального остовного дерева, задача о самом широком пути и др.). Необходимость в решении таких задач возникает, в частности, в математических методах исследования операций, робототехнике, планировании размещения предприятий, транспортировке товаров, проектировании сверхбольших интегральных схем (VLSI design), при нахождении путей в картографических сервисах (MapQuest, Google Maps и др.).

В данной статье выполнен обзор методов оптимизации времени выполнения ТС. В разделе 1 дана классификация подходов к оптимизации MMM и MVM, использующимся в качестве основы для оптимизаций ТС. В разделе 2 описываются методы оптимизации ТС. В заключении приводится обобщение результатов, полученных в ходе исследования.

1. Оптимизация MMM и MVM

В силу широкой распространенности и практической значимости оптимизация MMM и MVM является отдельным предметом изучения высокопроизводительных вычислений [9]. Оптимизированные MMM и MVM могут применяться для оптимизации ТС для тензоров большей размерности (Раздел 2). В данном разделе рассматриваются основные подходы к оптимизации MMM и MVM.

Специалисты в области приложений линейной алгебры одними из первых предложили использовать единый интерфейс к высокопроизводительным фундаментальным операциям, реализованным как стороннее программное обеспечение [17]. Более сорока лет таким интерфейсом являются базовые подпрограммы линейной алгебры (Basic Linear Algebra Subprograms, BLAS) [13–15]. Описываемые подпрограммы разделяются на три уровня. Первый уровень описывает операции над векторами, такие как, например, скалярное произведение и векторные нормы. На втором уровне интерфейса BLAS содержится описание операций над матрицами и векторами, таких как матрично-векторное произведение, а также описание подпрограмм для нахождения решения системы алгебраических уравнений с треугольной матрицей. Третьим уровнем описывается матричное произведение для различных типов матриц, построение симметричной матрицы и другие операции над матрицами. Интерфейс BLAS продолжает развиваться. BLAS++ [18], обновленная версия интерфейса

BLAS, используется в проектах, направленных на достижение экзафлопсной производительности.

Применение BLAS требует новой реализации фундаментальных операций для каждой новой архитектуры процессора. В большинстве случаев они создаются экспертами в области высокопроизводительных вычислений, что требует значительных временных затрат. Реализации фундаментальных операций могут распространяться как открытое программное обеспечение (GotoBLAS [9, 19], OpenBLAS [20], BLIS [21]) и как проприетарные библиотеки, выпускаемые производителями аппаратного обеспечения (Intel's MKL [22], IBM's ESSL [23], ArmPL [24]).

Для сокращения времени разработки высокопроизводительных реализаций BLAS авторы библиотек ATLAS [25] и PHiPAC [26] предложили использование автонастройки (autotuninig). В ходе ее работы выполняется перебор всех возможных значений параметров рассматриваемой программы с целью получения реализации, имеющей наименьшее время выполнения. Как и в случае ручной настройки (manual-tuning), для выполнения автонастройки необходим доступ к целевой архитектуре, а также потенциально большие временные затраты. Было показано, что в случае ATLAS и MMM автонастройка может приводить к неоптимальным результатам [9].

Алгоритм 1: Вычисление матричного произведения

```
1  begin
2      for j = 0...N step N_c do
3          for p = 0...K step K_c do
4              Упаковываем B(p:p + K_c − 1, j:j + N_c − 1) в массив B_c
5              for i = 0...M step M_c do
6                  Упаковываем A(i:i + M_c − 1, p:p + K_c − 1) в массив A_c
7                  for j_c = 0...N_c step N_r do
8                      for i_c = 0...M_c step M_r do
9                          for p_c = 0...K_c do
10                             𝕀 = i_c : i_c + M_r − 1
11                             𝕁 = j_c : j_c + N_r − 1
12                             C_c(𝕀, 𝕁) += A_c(𝕀, p_c) · B_c(p_c, 𝕁)
13                         end
14                     end
15                 end
16             end
17         end
18     end
19 end
```

В качестве примера рассмотрим параметризованный алгоритм 1, применяемый для выполнения матричного умножения в фреймворке BLIS [17]. Перемножаемые матрицы A и B имеют размерность $M \times K$ и $K \times N$, соответсвенно. Матрица C размерности $M \times N$ содержит результат выполнения матричного произведения. В процессе выполнения алгоритма матрицы A и B разбиваются на блоки (рис. 1). Это позволяет многократно использовать элементы блоков, хранящихся в кэш-памяти. Размеры блоков N_c, K_c, M_c, N_r и M_r являются

параметрами алгоритма. Их оптимальные значения могут определяться с помощью автонастройки и ручной настройки. С целью обеспечения последовательного доступа к операндам МММ используется упаковка данных. В процессе выполнения упаковки данных создаются временные выравненные в памяти массивы A_c и B_c, хранящие элементы перемножаемых матриц. Внутренний цикл вместе со своим содержимым называется микро-ядром МММ. Оно может генерироваться автоматически в процессе компиляции или реализовываться вручную в виде отдельной процедуры, написанной на языке ассемблера для целевой архитектуры [27, 28].

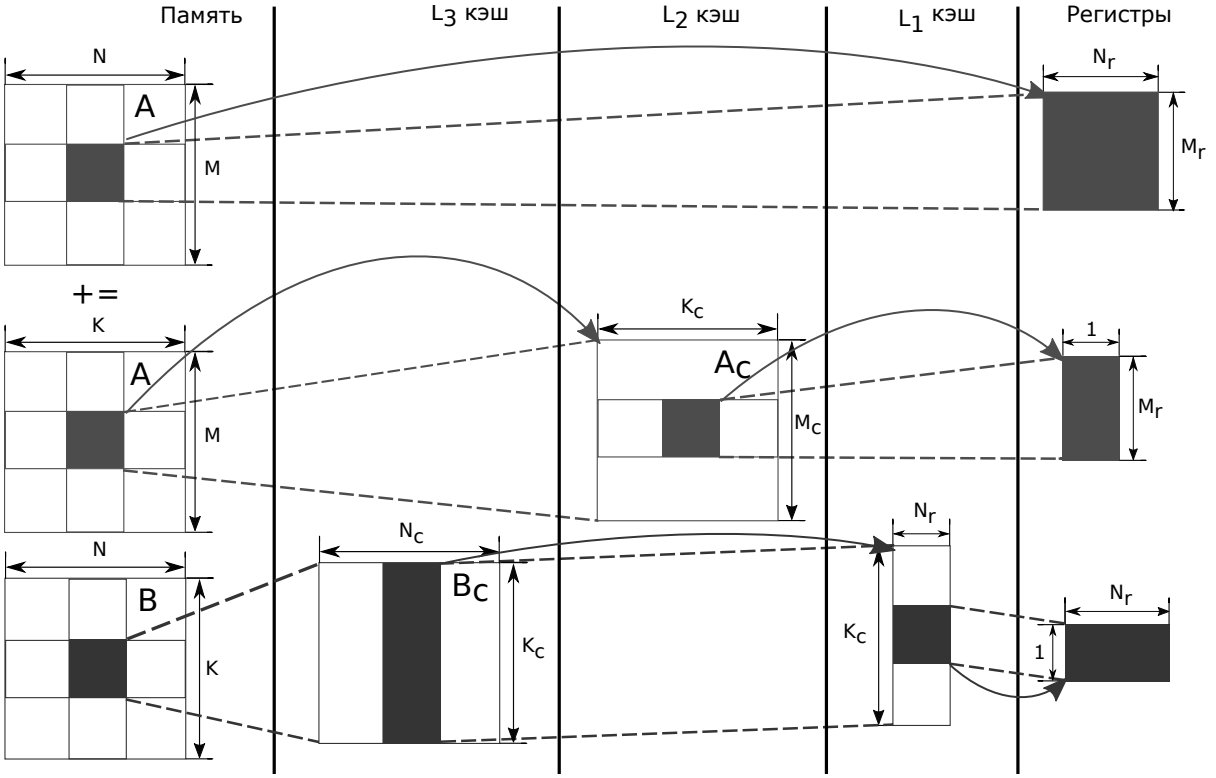

Рис. 1. Разбиение матриц A, B и C на блоки

Автонастройка и аналитические техники могут применяться совместно с целью определения наилучшей оптимизации. Такой подход используется в компиляторах LGen [29] и Build to Order BLAS (BTO) [30]. LGen — это компилятор для базовых операций линейной алгебры с операндами малой размерности, известной на этапе компиляции. Для выполнения разбиений циклов, векторизации и слияния циклов LGen использует специализированный язык, позволяющий построить математическое представление программы для ее дальнейшей оптимизации. Результатом работы LGen являются функции на языке C, содержащие SIMD инструкции. Пример выражения, подаваемого на вход компилятору LGen, приведен на рис. 2. BTO компилирует последовательность выражений, содержащих матрицы и векторы, написанную на языке MATLAB, в оптимизированный код на языке C++. В процессе оптимизации BTO выполняет слияние циклов, основываясь на аналитических моделях и эмпирических методах. На рис. 3 показан пример программы на языке MATLAB, компилируемой BTO.

Фреймворк AUGEM [31] использует автонастройку совместно с эвристическими алгоритмами векторизации, распределения регистров и планирования команд. С помощью библиотеки POET [32] фреймворк AUGEM распознает части абстрактного синтаксического

дерева программы на языке C, содержащие заранее определенный набор операций с матрицами, для их дальнейшей оптимизации. Пример операции с матрицами, распознаваемой фреймворком AUGEM показан на рис. 4.

```
alpha   =  Scalar()
beta    =  Scalar()
A       =  Matrix(6, 11)
x       =  Matrix(11, 1)
y       =  Matrix(6,  1)
z       =  Matrix(6,  1)
Generate(beta = (A*x+alpha*y)^T*z + Beta, opts)
```

Рис. 2. Пример выражения, подаваемого на вход компилятору LGen

```
GEMVER
in v1 : vector, v2 : vector,
   u1 : vector, u2 : vector,
   alpha : scalar, beta : scalar,
   x : vector, y : vector
inout A : dense column matrix
out z : vector, w : vector {
  A = A + v1 * u1' + v2 * u2'
  z = beta * (A' * x) + y
  w = alpha * (A * z)
}
```

Рис. 3. Пример выражения, подаваемого на вход компилятору BTO

```
mmCOMP(A, idx1, B, idx2, res):
tmp0=A[idx1]
tmp1=B[idx2]
tmp2=tmp0*tmp1
res=res+tmp2
```

Рис. 4. Пример операции с матрицами, распознаваемой фреймворком AUGEM

С целью сокращения времени автонастройки множество значений исследуемых параметров может быть уменьшено за счет использования дополнительных измерений, выполняемых на целевой архитектуре. Такой подход называется итеративной компиляцией (iterative compilation) [33]. В ходе ее работы алгоритм оценивает параметры целевой архитектуры, такие как, например, количество промахов кэша для данной реализации. В дальнейшем полученная информация применяется для определения значений оцениваемых параметров. Улучшенная автонастройка (auto-fine-tuning) [34] представляет собой отдельную модификацию автонастройки. Ее отличие в использовании аналитических моделей, уменьшающих пространство поиска. Такие модели могут быть созданы с применением знаний о современных архитектурах процессоров и оптимизуруемой операции. В общем случае указанные улучшения не гарантируют устранения проблем автонастройки.

Можно избежать перебора значений параметров программы за счет моделирования потребления ресурсов используемой архитектуры. Для этого применяются модели гипотети-

ческих процессоров, описывающих такие характеристики целевого процессора, как векторные регистры и векторные инструкции, иерархию кэш памяти [17, 34]. Аналитическое моделирование позволяет установить зависимости между потреблением ресурсов процессора и производительностью оптимизируемой программы. На момент написания данной работы не существует автоматического способа выполнения аналитического моделирования, позволяющего достичь высокой производительности оптимизируемой программы. Вследствие этого, как и в случае ручной настройки, аналитическое моделирование должно выполняться вручную специалистом.

Нахождение значений параметров программы, для которой выполнено аналитическое моделирование, и известны технические характеристики целевого процессора, выполняется за постоянное время и, вследствие этого, может использоваться в условиях ограниченного времени [28, 35]. Примером такой ситуации являются оптимизации, выполняемые промышленными компиляторами по умолчанию. С целью применения в процессе разработки и при компиляции готовых пакетов программ используемые оптимизации должны выполняться за наименьшее возможное время. Совместное применение автоматического распознавания кода, аналитических подходов и автоматической генерации кода позволяет автоматически получать высокопроизводительные реализации программ в процессе оптимизации, выполняемой компиляторами [28, 35].

В качестве примера модели гипотетического процессора рассмотрим модель, используемую в фреймворке BLIS для нахождения значений параметров реализации ММ [17]. Она может быть описана следующим образом [8]:

- «**Архитектура загрузки/сохранения и векторные регистры:** данные должны быть загружены в регистры процессора перед тем как с ними могут быть выполнены вычисления. Имеется N_{REG} векторных регистров. Каждый из них может содержать N_{VEC} значений размера S_{DATA}. Если N_{REG} равно 0, N_{VEC} присваивается значение 1. Предполагается, что команды для работы с памятью могут выполняться одновременно с командами арифметики с плавающей точкой».
- «**Векторные инструкции:** пропускная способность процессора составляет N_{VFMA} векторных операций смешанного сложения и умножения (vector fused multiply-add instruction, VFMA) за один такт. Отдельная VFMA выполняет N_{VEC} умножений и сложений, составляющих VFMA. Минимальное количество тактов L_{VFMA}, которое должно быть совершено перед началом выполнения новой зависимой по данным VFMA-инструкции, определяет задержку VFMA-инструкции».
- «**Кэш-память:** весь кэш данных — это множественно-ассоциативный кэш с политикой вытеснения последнего по времени использования (Least Recently Used, LRU). Каждый уровень кэша L_i характеризуется следующими параметрами: размер линии кэша C_{L_i}, степень ассоциативности W_{L_i}, количество множеств N_{L_i}, размер S_{L_i}, где $S_{L_i} = N_{L_i} C_{L_i} W_{L_i}$. В случае полностью ассоциативного кэша $N_{\mathrm{L}_i} = 1$».

Аналитическое моделирование может использоваться для минимизации перемещения данных и, как следствие, уменьшения времени работы приложения и его энергопотребления. Рассматриваемый алгоритм может быть изменен для достижения нижней асимптотической границы на перемещение данных в иерархии памяти, а также между процессорами [36, 37]. В частности могут изменяться формулы для нахождения значений параметров блочного алгоритма. Описанный подход обладает преимуществом в случае распределенных архитектур, имеющих большое количество отдельных узлов.

Кэш-независимые алгоритмы (cache-oblivious algorithms) [38] — это отдельный класс алгоритмов, позволяющих асимптотически оптимально использовать кэш-память, игнорируя ее отдельные параметры, такие как количество уровней кэша, длина кэш-линий, ассоциативность и т.д. Такие алгоритмы основываются на методе разделяй-и-властвуй (divide-and-conquer), разделяющем задачу на подзадачи до тех пор, пока данные отдельной подзадачи не смогут поместиться в каком-нибудь уровне кэш-памяти. Практическая оценка кэш-независимых алгоритмов показывает, что отсутствие в них таких механизмов, как предвыборка данных, затрудняет достижение производительности кода, созданного экспертами [39].

Несмотря на то, что MVM может рассматриваться как частный случай MMM, отношение количества вычислений, выполняемых MVM, к количеству перемещений данных из памяти меньше чем в случае MMM. Вследствие этого, методы для оптимизации MMM должны быть модифицированы для оптимизации MVM. В частности, из-за недопустимых издержек невозможно использование таких техник, как упаковка данных. В то же время возрастает необходимость в эффективной предвыборке данных, позволяющей уменьшить издержки чтения данных из памяти. Большинство подходов, используемых для оптимизации MVM, основаны на ручной настройке [9, 19–24, 40, 41] и автонастройке [25, 26]. Как и в случае MMM, аналитическое моделирование может применяться для оптимизации MVM с целью достижения производительности кода, оптимизированного вручную [8, 42]. Модификация представленной раннее гипотетической модели процессора может включать следующие дополнения, описывающее инструкции предвыборки [8]:

- «**Инструкции предвыборки:** за один такт процессора может быть выполнено N_{prefetch} инструкций. Задержка каждой инструкции составляет L_{prefetch} тактов процессора. Каждая инструкция может загрузить данные, размер которых равен C_{L_i}».

Особенности целевых архитектур процессоров и размерность рассматриваемой задачи могут использоваться совместно для автоматического получения оптимизированных реализаций BLAS. Так, например, библиотека LIBXSMM [43] генерирует низкоуровневые высокопроизводительные реализации MMM для некоторых архитектур от Intel и матриц, содержащих менее 80×80 элементов. Для этих целей используется специальный алгоритм предвыборки данных и модифицированный алгоритм векторизации.

Фреймворк POCA [27] позволят получить высокопроизводительные микро-ядра, части MMM, содержащие векторные инструкции. На рис. 5 представлен пример микро-ядра MMM для процессоров с микроархитектурой Sandy Bridge [27]. POCA использует эвристический алгоритм для распределения регистров процессора на основе анализа графа зависимостей и особенностей целевой архитектуры, таких как количество векторных регистров и количество тактов, требуемых для выполнения отдельных векторных инструкций микро-ядра.

2. Оптимизация ТС

За последние двадцать лет прогресс во многих научных дисциплинах, таких как квантовая химия и общая теория относительности, основывался на моделировании физических систем, использующем ТС. Несмотря на это, количество подходов к оптимизации ТС намного меньше чем количество подходов к оптимизации MMM и MVM [44]. В данном разделе описываются основные методы оптимизации времени выполнения ТС.

```
1 A0 = vload AP                  // <a0, a1, a2, a3>
2 B0 = vload BP                  // <b0, b1, b2, b3>
3 B1 = vshuffle B0 <1,0,3,2>     // <b1, b0, b3, b2>
4 B2 = vshuffle B1 <2,3,0,1>     // <b3, b2, b1, b0>
5 B3 = vshuffle B2 <1,0,3,2>     // <b2, b3, b0, b1>
6 M0 = fmul A0, B0               //
7 C0 = fadd C0, M0               // <c00, c11, c22, c33>
8 M1 = fmul A0, B1               //
9 C1 = fadd C1, M1               // <c01, c10, c23, c32>
10 M2 = fmul A0, B2              //
11 C2 = fadd C2, M2              // <c03, c12, c21, c30>
12 M3 = fmul A0, B3              //
13 C3 = fadd C3, M3              // <c02, c13, c20, c31>
14 AP = add AP, 4                // AP += 4
15 BP = add BP, 4                // BP += 4
16 prefetch_0 (BP+64)            // 8 * 64 = 512(bytes)
17 prefetch_0 AN                 // next A block
18 AN = add AN, 32               // Mr * 8 = 32(bytes)
```

Рис. 5. Пример микро-ядра MMM для процессора с микроархитектурой Sandy Bridge

Рассмотрим определения d-мерного тензора и TC, позволяющие свести оптимизацию этой операции к оптимизации MMM и MVM:

Определение 1. d-мерный тензор $\mathcal{T} \in \mathbb{R}^{n_{u_0} \times \ldots \times n_{u_{d-1}}}$ может быть определен как множество элементов из $\mathbb{R}$, описываемое следующим образом [45, 46]

$$\mathcal{T} \equiv \{A_{u_0 \ldots u_{d-1}} \in \mathbb{R} | (u_0, \ldots, u_{d-1}) \in n_{u_0} \times \ldots \times n_{u_{d-1}}\}.$$

В рамках данной работы будем предполагать, что элементы d-мерного тензора $\mathcal{T} \in \mathbb{R}^{n_{u_0} \times \ldots \times n_{u_{d-1}}}$ представлены многомерным массивом размерности $n_{u_0} \times \ldots \times n_{u_{d-1}}$. В общем случае это утверждение не всегда верно.

Определение 2. Пусть $\mathcal{A}$, $\mathcal{B}$, и $\mathcal{C}$ — это d_A-, d_B-, и d_C- мерные тензоры, соответственно. Сворачиваемые индексы $\mathcal{A}$ и $\mathcal{B}$ описываются кортежем $P = p_0 \ldots p_{t-1}$. Индексы $\mathcal{C}$, а также свободные индексы $\mathcal{A}$ и $\mathcal{B}$ описываются кортежами $I = i_0 \ldots i_{r-1}$ и $J = j_0 \ldots j_{s-1}$, соответственно. Операция свертывания или свертки $\mathcal{A}$ и $\mathcal{B}$ определяется как $\mathcal{C}_{\pi_C(IJ)} = \sum_P \alpha \cdot \mathcal{A}_{\pi_A(IP)} \cdot \mathcal{B}_{\pi_B(PJ)} + \beta \cdot \mathcal{C}_{\pi_C(IJ)}$, где $\sum_P = \sum_{p_0=0}^{n_{p_0}-1} \ldots \sum_{p_{t-1}=0}^{n_{p_{t-1}}-1}$, $\pi_C(IJ)$, $\pi_A(IP)$, и $\pi_B(PJ)$ — это перестановки индексов, α, $\beta \in \mathbb{R}$ [45, 46].

С целью оптимизации TC может быть использован программный код оптимизированных MMM и MVM, полученный в результате выполнения алгоритмов, описанных в предыдущем разделе. Так, например, метод Transpose-Transpose-GEMM-Transpose (TTGT) [47], применяемый в различных прикладных библиотеках (MATLAB Tensor Toolbox [48], NumPy [49], Eigen [50] и др.), выполняет перестановку индексов тензоров с целью их представления в виде матриц и использования MMM. Для выполнения таких перестановок требуются дополнительное время и память.

Метод Loops-over-GEMMS (LoG) [44] представляет сворачиваемые тензоры как наборы матриц, для которых выполняется MMM. Производительность LoG зависит от размера

таких матриц, а также расположения тензоров в памяти [44]. Так, например, в случае свертки 3-мерного тензора $\mathcal{A} \in \mathbb{R}^{m\times n\times k}$ и 2-мерного тензора $\mathcal{B} \in \mathbb{R}^{k\times l}$ в 3-мерный тензор $\mathcal{C} \in \mathbb{R}^{m\times n\times l}$, имеющей следующий вид $\mathcal{C}_{\alpha\beta\rho} = \sum\limits_{\delta=0}^{k-1} \mathcal{A}_{\alpha\beta\delta} \cdot \mathcal{B}_{\delta\rho} + \mathcal{C}_{\alpha\beta\rho}$, LoG вычислит МММ для каждого из n значений индекса β.

Для решения задач в таких научных областях, как, например, квантовая химия, созданы модификации метода TTGT для распределенных вычислений. Такие подходы применяют разбиение тензоров на блоки и другие техники для уменьшения передачи данных в иерархии памяти, а также между процессорами. Примерами библиотек, содержащих реализации описанных методов, служат Tensor Contraction Engine [47], Cyclops Tensor Framework [51], libtensor [52], TiledArray [53, 54] и др.

Для сокращения издержек, связанных с перестановкой индексов тензоров, можно использовать только микро-ядра, части реализации оптимизированного МММ, содержащие векторные инструкции на языке ассемблера. Такой подход применяется в методах GEMM-like Tensor-Tensor multiplication (GETT) [46] и Tensor-Based Library Instantiation Software (TBLIS) [45]. Как и в случае алгоритма 1 для вычисления МММ подходы GETT и TBLIS упаковывают элементы операндов ТС во временные одномерные массивы с целью дальнейшего использования кода микро-ядра МММ.

В отличии от GETT, TBLIS представляет тензоры в виде матриц, что позволяет явно выполнять разбиение матриц на блоки и другие трансформации, применяемые в алгоритме 1. В качестве примера применения такого представления рассмотрим d-мерный тензор $\mathcal{T} \in \mathbb{R}^{n_{u_0}\times\ldots\times n_{u_{d-1}}}$. $A_{u_0\ldots u_{d-1}}$, элемент тензора $\mathcal{T}$, расположен в памяти со смещением $\sum\limits_{k=0}^{d-1} u_k \prod\limits_{l=k+1}^{d-1} n_{u_l}$. Пусть свободные и сворачиваемые индексы тензора $\mathcal{T}$ описываются кортежами $I = i_0 \ldots i_{r-1}$ и $P = p_0 \ldots p_{s-1}$, соответсвенно. Тогда смещение может быть представлено в виде $\sum\limits_{k=0}^{d_I-1} i_k \left(\prod\limits_{l=k+1}^{d_I-1} n_{i_l}\right)\left(\prod\limits_{l=0}^{d_P-1} n_{p_l}\right) + \sum\limits_{k=0}^{d_P-1} p_k \prod\limits_{l=k+1}^{d_P-1} n_{p_l} = \left(\sum\limits_{k=0}^{d_I-1} i_k \prod\limits_{l=k+1}^{d_I-1} n_{i_l}\right) n_P + \sum\limits_{k=0}^{d_P-1} p_k \prod\limits_{l=k+1}^{d_P-1} n_{p_l} = \overline{I} n_P + \overline{P}$. Если рассмотреть смещение в памяти $iN + j$ элемента M_{ij} некой матрицы размерности M×N, то можно вывести формулы, позволяющие логически представить тензор $\mathcal{T}$ в виде матрицы размерности $n_I \times n_P$:

$$\overline{I} = \sum_{k=0}^{d_I-1} i_k \prod_{l=k+1}^{d_I-1} n_{i_l},$$

$$n_P = \prod_{l=0}^{d_P-1} n_{p_l}, \quad n_I = \prod_{l=0}^{d_I-1} n_{i_l},$$

$$\overline{P} = \sum_{k=0}^{d_P-1} p_k \prod_{l=k+1}^{d_P-1} n_{p_l}.$$

Другое отличие GETT в применении автонастройки для нахождения наилучших значений параметров разбиения и способов выполнить упаковку элементов тензора. Использование автонастройки позволяет повысить производительность получаемого кода, но делает невозможным применение GETT в условиях недоступности целевой архитектуры и ограниченности времени выполнения. Кроме этого GETT не поддерживает тензоры, размер которых неизвестен на этапе компиляции.

Подход, используемый в TBLIS, может быть объединен с аналитическим моделированием для устранения необходимости в коде MMM, оптимизированном вручную. Полученный метод совместно с алгоритмами для распознавания кода TC может применяться автоматически в процессе оптимизации, выполняемой промышленным компилятором по умолчанию во время компиляции и кросс-компиляции. На рис. 6 изображен комплекс автоматизированной оптимизации обобщенных тензорных операций (AOOTO) [8, 28]. В процессе работы AOOTO фронтенд Clang [55] строит промежуточное представление программы, используемое комплексом оптимизаций Polly [56]. Polly создает специальное математическое представление программы с целью распознавания TC и ее оптимизации с применением аналитического моделирования и метода, предложенного в TBLIS. Библиотека LLVM Core [55] использована для генерации ассемблерного кода целевой архитектуры процессора.

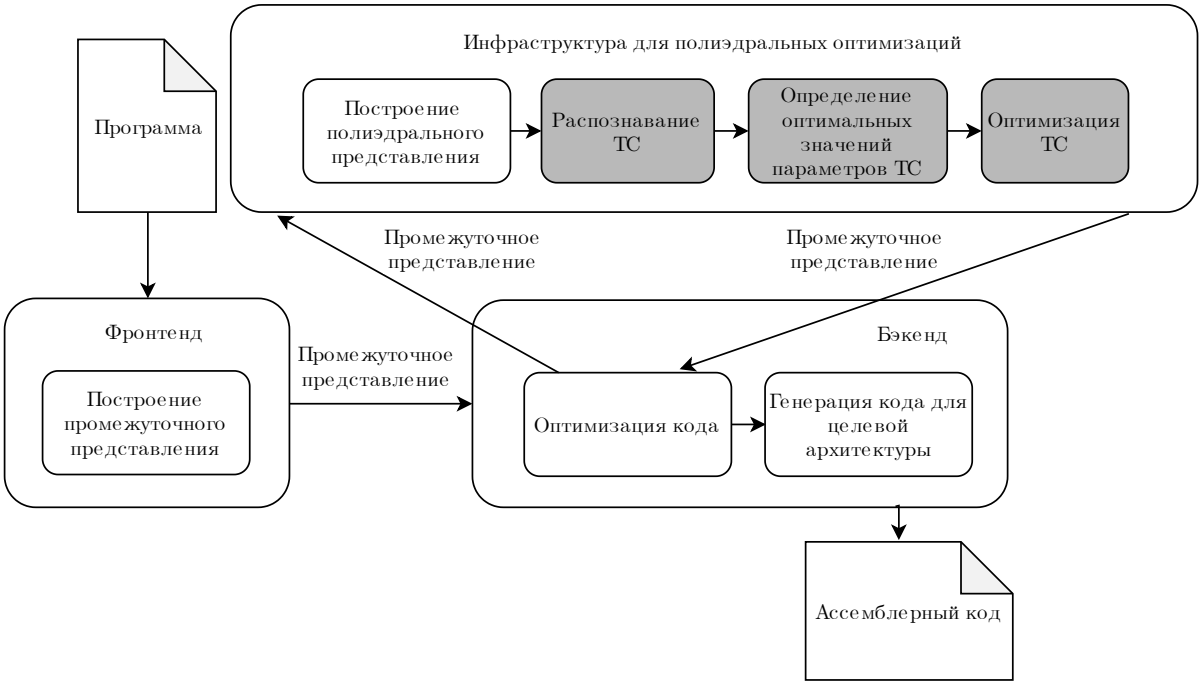

Рис. 6. Комплекс программ AOOTO

Для оптимизации TC могут применяться последовательности стандартных оптимизаций циклов (перестановка циклов, разбиение циклов на блоки, размотка циклов, векторизация и др.). Такие подходы зависят от размерности тензоров и их расположения в памяти [57]. В частности, производительность TC для задач малой размерности может быть улучшена за счет векторизации циклов [58]. Для задач большой размерности может применяться генератор кода циклов для графических ускорителей [59].

Заключение

TC является операцией «Тензорного исчисления», отдельного раздела математики, созданного для решения теоретических вопросов, требующих более общего исчисления чем векторное. TC, обобщение TC на произвольные полукольца матриц, а также MMM и MVM, частные случаи TC, широко применяются как в теории, так и на практике. Вследствие этого, эффективное выполнение TC имеет существенную практическую значимость.

В статье представлен обзор походов к оптимизации TC. Большинство таких методов основывается на преобразовании тензоров к матричной форме с целью применения оп-

тимизаций MMM и MVM, или использования отдельных частей оптимизированного кода MMM и MVM. Описывается представление тензоров, позволяющее свести оптимизацию ТС к оптимизации MMM и MVM.

Рассмотрены подходы к оптимизации MMM и MVM, реализованные в виде проприетарных библиотек и свободного программного обеспечения. Большая их часть основывается на ручной настройке и самонастройке, осложняющих использование в процессе кросс-компиляции, выполняемой промышленными компиляторами, и других условиях ограниченного времени и отсутствия доступа к целевой архитектуре. Время работы таких методов может быть уменьшено за счет применения аналитического моделирования, устанавливающего зависимость между использованием ресурсов гипотетического процессора и производительностью программы. В случае распределенных архитектур время выполнения MMM и MVM может быть уменьшено достижением нижней асимптотической границы на перемещение данных в иерархии памяти, а также между процессорами.

Литература

1. Коренев Г.В. Тензорное исчисление. Москва: Изд-во МФТИ, 2000. 240 с.
2. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. URL: `https://www.tensorflow.org/` (дата обращения: 02.02.2020).
3. Pekurovsky D. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions // SIAM Journal on Scientific Computing. 2012. Vol. 34, no. 4. P. 192–209. DOI: 10.1137/11082748X.
4. Deville M.O., Fischer P.F., Mund E.H High-Order Methods for Incompressible Fluid Flow. Cambridge University Press, 2002. 489 p.
5. Jayachandran S., Venkatachalam T. A Secure Scheme for Privacy Preserving Data Mining Using Matrix Encoding // Australian Journal of Basic and Applied Sciences (AJBAS). 2015. URL: `http://www.ajbasweb.com/old/ajbas/2015/July/741-744.pdf` (дата обращения: 17.03.2020).
6. Cong J., Xiao B. Minimizing Computation in Convolutional Neural Networks // Artificial Neural Networks and Machine Learning – ICANN. Springer International Publishing, Cham. 2014. P. 281–290. DOI: 10.1007/978-3-319-11179-7_36.
7. Watson M., Olivares-Amaya R., Edgar R.G., Aspuru-Guzik A. Accelerating Correlated Quantum Chemistry Calculations Using Graphical Processing Units // Computing in Science Engineering. 2010. Vol. 12, no. 4. P. 40–51. DOI: 10.1021/ct900543q.
8. Акимова Е.Н., Гареев Р.А. Аналитическое моделирование матрично-векторного произведения на многоядерных процессорах // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 1. С. 69–82. DOI: 10.14529/cmse200105.
9. Goto K., Geijn R. Anatomy of High-performance Matrix Multiplication // ACM Transactions on Mathematical Software. 2008. Vol. 34, no. 3. P. 1–25. DOI: 10.1145/1356052.1356053.
10. Yu J., Lukefahr A., Palframan D., Dasika G., Das R., Mahlke S. Scalpel: Customizing DNN pruning to the underlying hardware parallelism // 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (Ontario, Canada, June, 24–28, 2017). New York, ACM. 2017. P. 548–560. DOI: 10.1145/3079856.3080215.

11. Yang X., Parthasarathy S., Sadayappan P. Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining // Proceedings of the VLDB Endowment. 2011. Vol. 4, no. 4. P. 231–242. DOI: 10.14778/1938545.1938548.

12. Kaushik D., Gropp W., Minkoff M., Smith B. Improving the Performance of Tensor Matrix Vector Multiplication in Cumulative Reaction Probability Based Quantum Chemistry Codes // High Performance Computing – HiPC 2008 (Bangalore, India, December, 17–20, 2008). Heidelberg, Springer-Verlag. 2008. P. 120–130. DOI: 10.1007/978-3-540-89894-8_14.

13. Lawson C.L., Hanson R.J., Kincaid D.R., Krogh F.T. Basic Linear Algebra Subprograms for Fortran Usage // ACM Transactions on Mathematical Software. 1979. Vol. 5. no. 3. P. 308–323. DOI: 10.1145/355841.355847.

14. Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J. An Extended Set of FORTRAN Basic Linear Algebra Subprograms // ACM Transactions on Mathematical Software. 1988. Vol. 14, no. 1. P. 1–17. DOI: 10.1145/42288.42291.

15. Dongarra J.J., Du Croz J., Hammarling S., Duff I.S. A Set of Level 3 Basic Linear Algebra Subprograms // ACM Transactions on Mathematical Software. 1990. Vol. 16, no. 1. P. 1–17. DOI: 10.1145/77626.79170.

16. Sedukhin S.G., Paprzycki M. Generalizing Matrix Multiplication for Efficient Computations on Modern Computers // Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics – Volume Part I (Reykjavik, Iceland, September, 9–13, 2006). Heidelberg, Springer-Verlag. 2006. P. 225–234. DOI: 10.1007/978-3-642-31464-3_23.

17. Low T.M., Igual F.D., Smith T.M., Quintana-Orti E.S. Analytical Modeling Is Enough for High-Performance BLIS // ACM Transactions on Mathematical Software. 2016. Vol. 43, no. 2. P. 12:1–12:18. DOI: 10.1145/2925987.

18. Gates M., Luszczek P., Abdelfattah A., Kurzak J., Dongarra J., Arturov K., Cecka C., Freitag C. C++ API for BLAS and LAPACK. SLATE Working Notes. 2017. Vol. 2. P. 1–45. URL: https://www.icl.utk.edu/files/publications/2017/icl-utk-1031-2017.pdf (дата обращения: 17.03.2020).

19. Goto K., Van De Geijn R. High-performance Implementation of the Level-3 BLAS // ACM Transactions on Mathematical Software. 2008. Vol. 35. no. 1. P. 4:1–4:14. DOI: 10.1145/1377603.1377607.

20. Xianyi Z., Qian W., Yunquan Z. Model-driven level 3 BLAS performance optimization on Loongson 3A processor // 2012 IEEE 18th International Conference on Parallel and Distributed Systems (Singapore, December, 17–19, 2012). Massachusetts Ave. 2012. P. 684–691. DOI: 10.1109/ICPADS.2012.97.

21. Van Zee F., Van De Geijn R. BLIS: A Framework for Rapidly Instantiating BLAS Functionality // ACM Transactions on Mathematical Software. 2015. Vol. 41, no. 3. P. 14:1–14:33. DOI: 10.1145/2764454.

22. Intel Math Kernel Library (Intel MKL) URL: https://software.intel.com/ru-ru/intel-mkl/?cid=sem43700011401059448&intel_term=intel+mkl&gclid=CIjbtvqaqM8CFSoNcwodDPUAbw&gclsrc=aw.ds (дата обращения: 02.02.2020).

23. IBM Engineering and Scientific Subroutine Library Intel Math Kernel Library (Intel MKL). URL: https://www.ibm.com/support/knowledgecenter/SSFHY8_6.1/reference/essl_reference_pdf.pdf (дата обращения: 02.02.2020).

24. ARM Performance Libraries Reference Manual. URL: `https://ds.arm.com/media/uploads/arm_performance_libraries_reference_manual_arm-ecm-0493690.pdf` (дата обращения: 02.02.2020).

25. Whaley R.C., Dongarra J.J. Automatically Tuned Linear Algebra Software // Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (Montreal, Canada, November, 7, 1998). New York, ACM. 1998. P. 1–27. DOI: 10.5555/509058.509096.

26. Bilmes J., Asanovic K., Chin C., Demmel J. Optimizing Matrix Multiply Using PHiPAC: A Portable, High-performance, ANSI C Coding Methodology // Proceedings of the 11th International Conference on Supercomputing (Vienna, Austria, July, 7–11, 1997). New York, ACM. 1997. P. 340–347. DOI: 10.1145/2591635.2667174.

27. Su X., Liao X., Xue J. Automatic Generation of Fast BLAS3-GEMM: A Portable Compiler Approach // Proceedings of the 2017 International Symposium on Code Generation and Optimization (Austin, TX, USA, February, 4–8, 2017). IEEE Press. 2017. P. 122–133. DOI: 10.1109/CGO.2017.7863734.

28. Gareev R., Grosser T., Michael K. High-Performance Generalized Tensor Operations: A Compiler-Oriented Approach // ACM Transactions on Architecture and Code Optimization. 2018. Vol. 15, no. 3. P. 34:1–34:27. DOI: 10.1145/3235029.

29. Spampinato D.G., Markus P. A Basic Linear Algebra Compiler // International Symposium on Code Generation and Optimization (Orlando, FL, USA, February, 15–19, 2014). New York, ACM. 2014. P. 23–32. DOI: 10.1145/2581122.2544155.

30. Belter G., Jessup E.R., Karlin I., Siek J.G. Automating the Generation of Composed Linear Algebra Kernels // Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (Portland, OR, USA, November, 14–20, 2009). New York, ACM. 2009. P. 59:1-59:12. DOI: 10.1145/1654059.1654119.

31. Wang Q., Zhang X., Zhang Y., Yi Q. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs // Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (Denver, Colorado, USA, November, 17–22, 2013). New York, ACM. 2013. P. 25:1–25:12. DOI: 10.1145/2503210.2503219.

32. Yi Q., Wang Q., Cui H. Specializing Compiler Optimizations Through Programmable Composition for Dense Matrix Computations // Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (Cambridge, UK, December, 13–17, 2014). Washington, IEEE Computer Society. 2014. P. 596–608. DOI: 10.1109/MICRO.2014.14.

33. Knijnenburg P.M., Kisuki T., O'Boyle M.F.P. Iterative Compilation // Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation (SAMOS). Springer Berlin Heidelberg, 2002. P. 171–187. DOI: 10.1007/3-540-45874-3_10.

34. Yotov K., Xiaoming L., Gang R., Garzaran M.J.S., Padua D., Pingali K., Stodghill P. Is Search Really Necessary to Generate High-Performance BLAS? // Proceedings of the IEEE. 2005. Vol. 93, no. 2. P. 358–386. DOI: 10.1109/JPROC.2004.840444.

35. Akimova E.N., Gareev R.A. Algorithm of Automatic Parallelization of Generalized Matrix Multiplication // CEUR Workshop Proceedings. 2017. Vol. 2005. P. 1–10.

36. Demmel J. Communication Avoiding Algorithms // Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis (Salt Lake City, UT, USA,

November, 10–16, 2012). Massachusetts Ave., IEEE Computer Society. 2012. P. 1942–2000. DOI: 10.1109/SC.Companion.2012.351.

37. Ballard G., Carson E., Demmel J., Hoemmen M., Knight N., Schwartz O. Communication lower bounds and optimal algorithms for numerical linear algebra // Acta Numerica. 2014. Vol. 23. P. 1–155. DOI: 10.1017/S0962492914000038.

38. Frigo M., Leiserson C.E., Prokop H., Ramachandran S. Cache-Oblivious Algorithms // Proceedings of the 40th Annual Symposium on Foundations of Computer Science (New York City, New York, USA, October, 17–19, 1999). Massachusetts Ave., IEEE Computer Society. 2012. P. 285–298. DOI: 10.5555/795665.796479.

39. Yotov K., Roeder T., Pingali K., Gunnels J., Gustavson F. An Experimental Comparison of Cache-oblivious and Cache-conscious Programs // Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures (San Diego, California, USA, June, 9–11, 2007). New York, ACM. 2007. P. 93–104. DOI: 10.1145/1248377.1248394.

40. Liang J., Zhang Y. Optimization of GEMV on Intel AVX processor // International Journal of Database Theory and Application. 2016. Vol. 9. P. 47–60. DOI: 10.14257/ijdta.2016.9.2.06.

41. Hassan S.A., Mahmoud M.M.M., Hemeida A.M., Saber M.A. Effective Implementation of MatrixVector Multiplication on Intel's AVX Multicore Processor // Computer Languages, Systems & Structures. 2018. Vol. 51. P. 158–175. DOI: 10.1016/j.cl.2017.06.003.

42. Akimova E.N., Gareev R.A., Misilov V.E. Analytical Modeling of Matrix-Vector Multiplication on Multicore Processors: Solving Inverse Gravimetry Problem // 2019 International Multi-Conference on Engineering, Computer and Information Sciences (Novosibirsk, Russia, October, 21–27, 2019). Massachusetts, IEEE Xplore Digital Library. 2019. P. 0823–0827. DOI: 10.1109/SIBIRCON48586.2019.8958103.

43. Heinecke A., Pabst H., Henry G. LIBXSMM: A high performance library for small matrix multiplications // HPC transforms (Austin, TX, USA, November, 15–20, 2015). New York, ACM. 2015. P. 1–1. DOI: 10.1109/SC.2016.83.

44. Napoli E.D., Fabregat-Traver D., Quintana-Ortí G., Bientinesi P. Towards an efficient use of the BLAS library for multilinear tensor contractions // Applied Mathematics and Computation. 2014. Vol. 235. P. 454–468. DOI: 10.1016/j.amc.2014.02.051.

45. Matthews D. High-Performance Tensor Contraction without BLAS // SIAM Journal on Scientific Computing. 2018. Vol. 40, no. 1. P. 1–24. DOI: 10.1137/16m108968x.

46. Springer P., Bientinesi P. Design of a High-Performance GEMM-like Tensor-Tensor Multiplication // ACM Transactions on Mathematical Software. 2018. Vol. 44, no. 3. P. 28:1–28:29. DOI: 10.1145/3157733.

47. Hirata S. Tensor Contraction Engine: Abstraction and Automated Parallel Implementation of Configuration-Interaction, Coupled-Cluster, and Many-Body Perturbation Theories // The Journal of Physical Chemistry A. 2003. Vol. 107, no. 46. P. 9887–9897. DOI: 10.1021/jp034596z.

48. Bader B.W., Kolda G.T. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping // ACM Transactions on Mathematical Software. 2006. Vol. 32. P. 635–653. DOI: 10.1145/1186785.1186794.

49. Walt S., Colbert C., Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation // Computing in Science & Engineering. 2011. Vol. 13. P. 22–30. DOI: 10.1109/MCSE.2011.37.

50. Eigen v3. URL: `http://eigen.tuxfamily.org` (дата обращения: 02.02.2020).

51. Solomonik E., Matthews D., Hammond J., Stanton J.F., Demmel J. A massively parallel tensor contraction framework for coupled-cluster computations // Journal of Parallel and Distributed Computing. 2014. Vol. 74, no. 12. P. 3176–3190. DOI: 10.1016/j.jpdc.2014.06.002.

52. Epifanovsky E., Wormit M., Ku T., Landau A., Zuev D., Khistyaev K., Manohar P., Kaliman I., Dreuw A., Krylov A. New implementation of high-level correlated methods using a general block tensor library for high-performance electronic structure calculations // Journal of computational chemistry. 2013. Vol. 34. P. 2293–2309. DOI: 10.1002/jcc.23377.

53. Calvin J., Lewis C.A., Valeev E.F. Scalable Task-Based Algorithm for Multiplication of Block-Rank-Sparse Matrices // Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms (Austin, Texas, USA, November, 15, 2015). New York, ACM. 2015. P. 4:1–4:8. DOI: 10.1145/2833179.2833186.

54. Calvin J., Valeev E.F. Task-Based Algorithm for Matrix Multiplication: A Step Towards Block-Sparse Tensor Computing // Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms (Austin, Texas, USA, November, 15, 2015). New York, ACM. 2015. P. 1–9.

55. Lattner C. LLVM: An Infrastructure for Multi-Stage Optimization // Master's Thesis. 2002. URL: `http://llvm.cs.uiuc.edu` (дата обращения: 27.10.2019).

56. Grosser T., Größlinger A., Lengauer C. Polly—Performing polyhedral optimizations on a low-level intermediate representation // Parallel Processing Letters. 2012. Vol. 22. No. 4. DOI: 10.1142/S0129626412500107.

57. Apra E., Klemm M., Kowalski K. Efficient Implementation of Many-Body Quantum Chemical Methods on the Intel E; Xeon Phi Coprocessor // International Conference for High Performance Computing, Networking, Storage and Analysis (New Orleans, LA, USA, November, 16–21, 2014). Massachusetts Ave., IEEE Computer Society. 2014. P. 674–684. DOI: 10.1109/SC.2014.60.

58. Stock K., Henretty T., Murugandi I., Sadayappan P., Harrison R. Model-Driven SIMD Code Generation for a Multi-resolution Tensor Kernel // Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (New Orleans, LA, USA, May, 16–20, 2011). Massachusetts Ave., IEEE Computer Society. 2014. P. 1058–1067. DOI: 10.1109/IPDPS.2011.101.

59. Ma W., Krishnamoorthy S., Villa O., Kowalski K. GPU-Based Implementations of the Noniterative Regularized-CCSD(T) Corrections: Applications to Strongly Correlated Systems // Journal of Chemical Theory and Computation. 2011. Vol. 7, no. 5. P. 1316–1327. DOI: 10.1021/ct1007247.

Гареев Роман Альбертович, аспирант, Институт радиоэлектроники и информационных технологий-РтФ, УрФУ имени первого Президента России Б.Н. Ельцина (Екатеринбург, Российская Федерация)

DOI: 10.14529/cmse200202

OPTIMIZATION METHODS FOR GENERALIZED TENSOR CONTRACTION

© 2020 R.A. Gareev

Ural Federal University (Mira 19, Yekaterinburg, 620002 Russia)

E-mail: gareevroman@gmail.com

Received: 18.03.2020

Tensor contraction is one of major operations defined in tensor calculus, a separate branch of mathematics, which become a fundamental language of theory of relativity, mechanics, electrodynamics, and solid state physics. Effective implementation of tensor contraction is of considerable practical significance for such areas as solving of mathematical physics problems, machine learning, spectral element methods, quantum chemistry, data mining, and high performance computing. In the last twenty years, the number of optimization methods for tensor contraction has increased and continues to grow. In this article, the author reviews widespread approaches for optimization of tensor contraction, which are used on single processor as well as multiprocessor systems with distributed memory. The review contains the description of methods for optimization of matrix and matrix-vector multiplications, important particular cases of tensor contraction, which are used as a base for the most tensor contraction optimizations. The described optimizations can be applied during program compilation performed by production compilers. The information provided in this work could be useful for systematizing knowledge.

Keywords: tensor contraction, linear algebra, high-performance computing.

FOR CITATION

Gareev R.A. Optimization Methods for Generalized Tensor Contraction. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2020. Vol. 9, no. 2. P. 19–39. (in Russian) DOI: 10.14529/cmse200202.

References

1. Korenev G.V. Tensor analysis. Moscow, MIPT, 2000. 240 p.
2. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Available at: `https://www.tensorflow.org/` (accessed: 02.02.2020).
3. Pekurovsky D. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. SIAM Journal on Scientific Computing. 2012. Vol. 34, no. 4. P. 192–209. DOI: 10.1137/11082748X.
4. Deville M.O., Fischer P.F., Mund E.H High-Order Methods for Incompressible Fluid Flow. Cambridge University Press, 2002. 489 p.
5. Jayachandran S., Venkatachalam T. A Secure Scheme for Privacy Preserving Data Mining Using Matrix Encoding. Australian Journal of Basic and Applied Sciences (AJBAS). 2015. Available at: `http://www.ajbasweb.com/old/ajbas/2015/July/741-744.pdf` (accessed: 17.03.2020).
6. Cong J., Xiao B. Minimizing Computation in Convolutional Neural Networks. Artificial Neural Networks and Machine Learning – ICANN. Springer International Publishing, 2014. P. 281–290. DOI: 10.1007/978-3-319-11179-7_36.

7. Watson M., Olivares-Amaya R., Edgar R.G., Aspuru-Guzik A. Accelerating Correlated Quantum Chemistry Calculations Using Graphical Processing Units. Computing in Science Engineering. 2010. Vol. 12, no. 4. P. 40–51. DOI: 10.1021/ct900543q.

8. Akimova E.N., Gareev R.A. Application of Analytical Modeling of Matrix-Vector Multiplication on Multicore Processors. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2020. Vol. 9, no. 1. P. 69–82. (in Russian) DOI: 10.14529/cmse200105.

9. Goto K., Geijn R. Anatomy of High-performance Matrix Multiplication. ACM Transactions on Mathematical Software. 2008. Vol. 34, no. 3. P. 1–25. DOI: 10.1145/1356052.1356053.

10. Yu J., Lukefahr A., Palframan D., Dasika G., Das R., Mahlke S. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (Ontario, Canada, June, 24–28, 2017). New York, ACM. 2017. P. 548–560. DOI: 10.1145/3079856.3080215.

11. Yang X., Parthasarathy S., Sadayappan P. Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining. Proceedings of the VLDB Endowment. 2011. Vol. 4, no. 4. P. 231–242. DOI: 10.14778/1938545.1938548.

12. Kaushik D., Gropp W., Minkoff M., Smith B. Improving the Performance of Tensor Matrix Vector Multiplication in Cumulative Reaction Probability Based Quantum Chemistry Codes. High Performance Computing – HiPC 2008 (Bangalore, India, December, 17–20, 2008). Heidelberg, Springer-Verlag. 2008. P. 120–130. DOI: 10.1007/978-3-540-89894-8_14.

13. Lawson C.L., Hanson R.J., Kincaid D.R., Krogh F.T. Basic Linear Algebra Subprograms for Fortran Usage. ACM Transactions on Mathematical Software. 1979. Vol. 5. no. 3. P. 308–323. DOI: 10.1145/355841.355847.

14. Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J. An Extended Set of FORTRAN Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software. 1988. Vol. 14, no. 1. P. 1–17. DOI: 10.1145/42288.42291.

15. Dongarra J.J., Du Croz J., Hammarling S., Duff I.S. A Set of Level 3 Basic Linear Algebra Subprograms. ACM Transactions on Mathematical Software. 1990. Vol. 16, no. 1. P. 1–17. DOI: 10.1145/77626.79170.

16. Sedukhin S.G., Paprzycki M. Generalizing Matrix Multiplication for Efficient Computations on Modern Computers. Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics – Volume Part I (Reykjavik, Iceland, September, 9–13, 2006). Heidelberg, Springer-Verlag. 2006. P. 225–234. DOI: 10.1007/978-3-642-31464-3_23.

17. Low T.M., Igual F.D., Smith T.M., Quintana-Orti E.S. Analytical Modeling Is Enough for High-Performance BLIS. ACM Transactions on Mathematical Software. 2016. Vol. 43, no. 2. P. 12:1–12:18. DOI: 10.1145/2925987.

18. Gates M., Luszczek P., Abdelfattah A., Kurzak J., Dongarra J., Arturov K., Cecka C., Freitag C. C++ API for BLAS and LAPACK. SLATE Working Notes. 2017. Vol. 2. P. 1–45. Available at: `https://www.icl.utk.edu/files/publications/2017/icl-utk-1031-2017.pdf` (accessed: 17.03.2020).

19. Goto K., Van De Geijn R. High-performance Implementation of the Level-3 BLAS. ACM Transactions on Mathematical Software. 2008. Vol. 35. no. 1. P. 4:1–4:14. DOI: 10.1145/1377603.1377607.

20. Xianyi Z., Qian W., Yunquan Z. Model-driven level 3 BLAS performance optimization on Loongson 3A processor. 2012 IEEE 18th International Conference on Parallel and Distributed Systems (Singapore, December, 17–19, 2012). Massachusetts Ave. 2012. P. 684–691. DOI: 10.1109/ICPADS.2012.97.

21. Van Zee F., Van De Geijn R. BLIS: A Framework for Rapidly Instantiating BLAS Functionality. ACM Transactions on Mathematical Software. 2015. Vol. 41, no. 3. P. 14:1–14:33. DOI: 10.1145/2764454.

22. Intel Math Kernel Library (Intel MKL) Available at: `https://software.intel.com/ru-ru/intel-mkl/?cid=sem43700011401059448&intel_term=intel+mkl&gclid=CIjbtvqaqM8CFSoNcwodDPUAbw&gclsrc=aw.ds` (accessed: 02.02.2020).

23. IBM Engineering and Scientific Subroutine Library Intel Math Kernel Library (Intel MKL). Available at: `https://www.ibm.com/support/knowledgecenter/SSFHY8_6.1/reference/essl_reference_pdf.pdf` (accessed: 02.02.2020).

24. ARM Performance Libraries Reference Manual. Available at: `https://ds.arm.com/media/uploads/arm_performance_libraries_reference_manual_arm-ecm-0493690.pdf` (accessed: 02.02.2020).

25. Whaley R.C., Dongarra J.J. Automatically Tuned Linear Algebra Software. Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (Montreal, Canada, November, 7, 1998). New York, ACM. 1998. P. 1–27. DOI: 10.5555/509058.509096.

26. Bilmes J., Asanovic K., Chin C., Demmel J. Optimizing Matrix Multiply Using PHiPAC: A Portable, High-performance, ANSI C Coding Methodology. Proceedings of the 11th International Conference on Supercomputing (Vienna, Austria, July, 7–11, 1997). New York, ACM. 1997. P. 340–347. DOI: 10.1145/2591635.2667174.

27. Su X., Liao X., Xue J. Automatic Generation of Fast BLAS3-GEMM: A Portable Compiler Approach. Proceedings of the 2017 International Symposium on Code Generation and Optimization (Austin, TX, USA, February, 4–8, 2017). IEEE Press. 2017. P. 122–133. DOI: 10.1109/CGO.2017.7863734.

28. Gareev R., Grosser T., Michael K. High-Performance Generalized Tensor Operations: A Compiler-Oriented Approach. ACM Transactions on Architecture and Code Optimization. 2018. Vol. 15, no. 3. P. 34:1–34:27. DOI: 10.1145/3235029.

29. Spampinato D.G., Markus P. A Basic Linear Algebra Compiler. International Symposium on Code Generation and Optimization (Orlando, FL, USA, February, 15–19, 2014). New York, ACM. 2014. P. 23–32. DOI: 10.1145/2581122.2544155.

30. Belter G., Jessup E.R., Karlin I., Siek J.G. Automating the Generation of Composed Linear Algebra Kernels. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (Portland, OR, USA, November, 14–20, 2009). New York, ACM. 2009. P. 59:1-59:12. DOI: 10.1145/1654059.1654119.

31. Yi Q., Wang Q., Cui H. Specializing Compiler Optimizations Through Programmable Composition for Dense Matrix Computations. Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (Cambridge, UK, December, 13–17, 2014). Washington, IEEE Computer Society. 2014. P. 596–608. DOI: 10.1109/MICRO.2014.14.

32. Wang Q., Zhang X., Zhang Y., Yi Q. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. Proceedings of the International

Conference on High Performance Computing, Networking, Storage and Analysis (Denver, Colorado, USA, November, 17–22, 2013). New York, ACM. 2013. P. 25:1–25:12. DOI: 10.1145/2503210.2503219.

33. Knijnenburg P.M., Kisuki T., O'Boyle M.F.P. Iterative Compilation. Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation (SAMOS). Springer Berlin Heidelberg, 2002. P. 171–187. DOI: 10.1007/3-540-45874-3_10.

34. Yotov K., Xiaoming L., Gang R., Garzaran M.J.S., Padua D., Pingali K., Stodghill P. Is Search Really Necessary to Generate High-Performance BLAS? Proceedings of the IEEE. 2005. Vol. 93, no. 2. P. 358–386. DOI: 10.1109/JPROC.2004.840444.

35. Akimova E.N., Gareev R.A. Algorithm of Automatic Parallelization of Generalized Matrix Multiplication. CEUR Workshop Proceedings. 2017. Vol. 2005. P. 1–10.

36. Demmel J. Communication Avoiding Algorithms. Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis (Salt Lake City, UT, USA, November, 10–16, 2012). Massachusetts Ave., IEEE Computer Society. 2012. P. 1942–2000. DOI: 10.1109/SC.Companion.2012.351.

37. Ballard G., Carson E., Demmel J., Hoemmen M., Knight N., Schwartz O. Communication lower bounds and optimal algorithms for numerical linear algebra. Acta Numerica. 2014. Vol. 23. P. 1–155. DOI: 10.1017/S0962492914000038.

38. Frigo M., Leiserson C.E., Prokop H., Ramachandran S. Cache-Oblivious Algorithms. Proceedings of the 40th Annual Symposium on Foundations of Computer Science (New York City, New York, USA, October, 17–19, 1999). Massachusetts Ave., IEEE Computer Society. 2012. P. 285–298. DOI: 10.5555/795665.796479.

39. Yotov K., Roeder T., Pingali K., Gunnels J., Gustavson F. An Experimental Comparison of Cache-oblivious and Cache-conscious Programs. Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures (San Diego, California, USA, June, 9–11, 2007). New York, ACM. 2007. P. 93–104. DOI: 10.1145/1248377.1248394.

40. Liang J., Zhang Y. Optimization of GEMV on Intel AVX processor. International Journal of Database Theory and Application. 2016. Vol. 9. P. 47–60. DOI: 10.14257/ijdta.2016.9.2.06.

41. Hassan S.A., Mahmoud M.M.M., Hemeida A.M., Saber M.A. Effective Implementation of MatrixVector Multiplication on Intel's AVX Multicore Processor. Computer Languages, Systems & Structures. 2018. Vol. 51. P. 158–175. DOI: 10.1016/j.cl.2017.06.003.

42. Akimova E.N., Gareev R.A., Misilov V.E. Analytical Modeling of Matrix-Vector Multiplication on Multicore Processors: Solving Inverse Gravimetry Problem. 2019 International Multi-Conference on Engineering, Computer and Information Sciences (Novosibirsk, Russia, October, 21–27, 2019). Massachusetts, IEEE Xplore Digital Library. 2019. P. 0823–0827. DOI: 10.1109/SIBIRCON48586.2019.8958103.

43. Heinecke A., Pabst H., Henry G. LIBXSMM: A high performance library for small matrix multiplications. HPC transforms (Austin, TX, USA, November, 15–20, 2015). New York, ACM. 2015. P. 1–1. DOI: 10.1109/SC.2016.83.

44. Napoli E.D., Fabregat-Traver D., Quintana-Ortí G., Bientinesi P. Towards an efficient use of the BLAS library for multilinear tensor contractions. Applied Mathematics and Computation. 2014. Vol. 235. P. 454–468. DOI: 10.1016/j.amc.2014.02.051.

45. Matthews D. High-Performance Tensor Contraction without BLAS. SIAM Journal on Scientific Computing. 2018. Vol. 40, no. 1. P. 1–24. DOI: 10.1137/16m108968x.

46. Springer P., Bientinesi P. Design of a High-Performance GEMM-like Tensor-Tensor Multiplication. ACM Transactions on Mathematical Software. 2018. Vol. 44, no. 3. P. 28:1–28:29. DOI: 10.1145/3157733.

47. Hirata S. Tensor Contraction Engine: Abstraction and Automated Parallel Implementation of Configuration-Interaction, Coupled-Cluster, and Many-Body Perturbation Theories. The Journal of Physical Chemistry A. 2003. Vol. 107, no. 46. P. 9887–9897. DOI: 10.1021/jp034596z.

48. Bader B.W., Kolda G.T. Algorithm 862: MATLAB tensor classes for fast algorithm prototyping. ACM Transactions on Mathematical Software. 2006. Vol. 32. P. 635–653. DOI: 10.1145/1186785.1186794.

49. Walt S., Colbert C., Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering. 2011. Vol. 13. P. 22–30. DOI: 10.1109/MCSE.2011.37.

50. Eigen v3. Available at: `http://eigen.tuxfamily.org` (accessed: 02.02.2020).

51. Solomonik E., Matthews D., Hammond J., Stanton J.F., Demmel J. A massively parallel tensor contraction framework for coupled-cluster computations. Journal of Parallel and Distributed Computing. 2014. Vol. 74, no. 12. P. 3176–3190. DOI: 10.1016/j.jpdc.2014.06.002.

52. Epifanovsky E., Wormit M., Ku T., Landau A., Zuev D., Khistyaev K., Manohar P., Kaliman I., Dreuw A., Krylov A. New implementation of high-level correlated methods using a general block tensor library for high-performance electronic structure calculations. Journal of computational chemistry. 2013. Vol. 34. P. 2293–2309. DOI: 10.1002/jcc.23377.

53. Calvin J., Lewis C.A., Valeev E.F. Scalable Task-Based Algorithm for Multiplication of Block-Rank-Sparse Matrices. Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms (Austin, Texas, USA, November, 15, 2015). New York, ACM. 2015. P. 4:1–4:8. DOI: 10.1145/2833179.2833186.

54. Calvin J., Valeev E.F. Task-Based Algorithm for Matrix Multiplication: A Step Towards Block-Sparse Tensor Computing. Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms (Austin, Texas, USA, November, 15, 2015). New York, ACM. 2015. P. 1–9.

55. Lattner C. LLVM: An Infrastructure for Multi-Stage Optimization. Master's Thesis. 2002. Available at: `http://llvm.cs.uiuc.edu` (accessed: 27.10.2019).

56. Grosser T., Größlinger A., Lengauer C. Polly—Performing polyhedral optimizations on a low-level intermediate representation. Parallel Processing Letters. 2012. Vol. 22. No. 4. DOI: 10.1142/S0129626412500107.

57. Apra E., Klemm M., Kowalski K. Efficient Implementation of Many-Body Quantum Chemical Methods on the Intel E; Xeon Phi Coprocessor. International Conference for High Performance Computing, Networking, Storage and Analysis (New Orleans, LA, USA, November, 16–21, 2014). Massachusetts Ave., IEEE Computer Society. 2014. P. 674–684. DOI: 10.1109/SC.2014.60.

58. Stock K., Henretty T., Murugandi I., Sadayappan P., Harrison R. Model-Driven SIMD Code Generation for a Multi-resolution Tensor Kernel. Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (New Orleans, LA, USA, May, 16–20, 2011). Massachusetts Ave., IEEE Computer Society. 2014. P. 1058–1067. DOI: 10.1109/IPDPS.2011.101.

59. Ma W., Krishnamoorthy S., Villa O., Kowalski K. GPU-Based Implementations of the Noniterative Regularized-CCSD(T) Corrections: Applications to Strongly Correlated Systems. Journal of Chemical Theory and Computation. 2011. Vol. 7, no. 5. P. 1316–1327. DOI: 10.1021/ct1007247.