УДК 004.021, 519.685 DOI: 10.14529/cmse200105

АНАЛИТИЧЕСКОЕ МОДЕЛИРОВАНИЕ МАТРИЧНО-ВЕКТОРНОГО ПРОИЗВЕДЕНИЯ НА МНОГОЯДЕРНЫХ ПРОЦЕССОРАХ*

© 2020 Е.Н. Акимова[1,2], Р.А. Гареев[1]

[1] *Уральский федеральный университет им. Б.Н. Ельцина (620002 Екатеринбург, ул. Мира, д. 19),*

[2] *Институт математики и механики им. Н.Н. Красовского УрО РАН (620990 Екатеринбург, ул. Софьи Ковалевской, д. 16)*

E-mail: aen15@yandex.ru, gareevroman@gmail.com

Поступила в редакцию: 31.01.2020

Эффективная реализация матрично-векторного произведения имеет существенную практическую значимость в областях машинного обучения, интеллектуального анализа данных, квантовой химии, математической физики, численных методов линейной алгебры, высокопроизводительных вычислений и др. В данной работе представлен алгоритм автоматизированной оптимизации матрично-векторного произведения по времени выполнения, использующийся на этапе компиляции без ручной настройки и автонастройки. Алгоритм основан на моделировании вычислений на гипотетическом многоядерном процессоре, предложенном авторами, с применением полиэдрального представления. В отличие от подходов, основанных на ручной настройке и автонастройке, алгоритм может применяться для создания новых оптимизированных реализаций матрично-векторного произведения в условиях недоступности целевой архитектуры и ограниченности времени выполнения. Алгоритм использован для оптимизации программного кода, реализующего решение структурной обратной задачи гравиметрии о нахождении поверхности раздела сред методом Левенберга—Марквардта. Проведено сравнение производительности полученной реализации с реализациями на основе оптимизированных библиотек линейной алгебры Intel MKL, BLIS, OpenBLAS. Результаты численных экспериментов показывают сравнимость предложенного алгоритма по эффективности с подходами, созданными с использованием ручной настройки при доступе к целевым архитектурам процессоров.

Ключевые слова: компиляторы, линейная алгебра, матрично-векторные операции, аналитическое моделирование, обратная задача гравиметрии.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

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.

Введение

Матрично-векторное произведение (matrix-vector multiplication, MVM) и матричное произведение (matrix-matrix multiplication, MMM) широко используются для решения прикладных задач. В частности, MVM вычисляется в процессе обрезки весов (weight pruning) в глубинных нейронных сетях (Deep Neural Networks, DNN), используемых в машинном обучении [1]. MVM применяется в интеллектуальном анализе графов [2]. MVM вычисляется при определении коэффициента скорости реакции в квантовой химии [3]. MVM и MMM используются для решения широкого ряда задач математической физики и, в частности, математической геофизики, численными методами [4]. Минимизация времени выполнения MVM является отдельным предметом изучения высокопроизводительных вычислений [5, 6].

*Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии (ПаВТ) 2020»

Современные подходы, используемые для оптимизации MVM по времени выполнения, основаны на эмпирическом поиске. Их можно разделить на два вида: ручная настройка (manual-tuning) и автонастройка (auto-tuning) [7]. Для выполнения ручной настройки для отдельно рассматриваемой архитектуры компьютера специалист, обладающий знаниями особенностей используемых аппаратных средств, создает новую реализацию требуемой операции. В случае автонастройки для каждой отдельно рассматриваемой компьютерной архитектуры выполняется перебор значений параметров заранее созданной реализации для нахождения их оптимальных значений. Это делает невозможным их применение в процессе кросс-компиляции и в других условиях недоступности целевой архитектуры процессора. Выполнение ручной настройки и автонастройки может потребовать значительных временных затрат. Вследствие этого ручная настройка и автонастройка не могут использоваться, когда время выполнения ограничено. Таким примером являются оптимизации, применяемые компилятором по умолчанию. С целью возможности их частого использования на этапе разработки и компиляции программ предпочтительным являются подходы, требующие наименьшего времени выполнения.

Результаты [7] в области моделирования MMM на гипотетическом процессоре показывают, что эмпирический поиск необязателен для достижения высокой производительности. Рассматриваемые подходы для оптимизации MMM не могут быть использованы для MVM из-за меньшей доли вычислений и большей доли обращений к памяти [8].

В работах [9, 10] представлены алгоритмы оптимизации по времени выполнения обобщенных MMM и сверток тензоров (tensor contraction, TC) в процессе компиляции.

В данной работе предложен алгоритм оптимизации по времени выполнения MVM без эмпирического поиска. Оптимизация выполняется с использованием полиэдрального представления. Для определения значений параметров оптимизации применяется моделирование вычисления MVM на гипотетическом многоядерном процессоре. Алгоритм использован для оптимизации программы, реализующей решение обратной структурной задачи гравиметрии о нахождении поверхности раздела сред методом Левенберга—Марквардта.

Полученный программный код сравнивается по производительности с программными кодами, оптимизированными с помощью библиотек линейной алгебры, созданных с использованием ручной настройки при доступе к целевым архитектурам процессоров (Intel MKL [11], BLIS [12], OpenBLAS [13]). Преимущество предложенного алгоритма состоит в возможности его применения для создания новых оптимизированных реализаций MVM в условиях недоступности целевой архитектуры и ограниченности времени выполнения.

Численные эксперименты проводились на узле с двумя 18-ядерными процессорами Intel Xeon E5-2697 v4 суперкомпьютера «Уран» (Институт математики и механики им. Н.Н. Красовского УрО РАН, Екатеринбург, Россия). Показана сравнимость нашего подхода по производительности с подходами, основанными на эмпирическом поиске.

Статья организована следующим образом. Раздел 1 посвящен описанию алгоритма автоматизированной оптимизации матрично-векторного произведения. В разделе 2 выполнена постановка структурной обратной задачи гравиметрии, описан алгоритм её решения, приведены результаты сравнения производительности полученной реализации решения задачи гравиметрии с реализациями на основе оптимизированных библиотек линейной алгебры. В заключении содержатся выводы и направления дальнейших исследований.

1. Автоматизированная оптимизация MVM

В данном разделе описывается алгоритм автоматизированной оптимизации (ААО) MVM, использующий аналитическое моделирование и оптимизацию полиэдрального представления. Описываемые методы могут использоваться во время кросс-компиляции, выполняемой промышленными компиляторами, и в других случаях, когда время выполнения компиляции ограничено либо архитектура целевого процессора недоступна.

Ручная оптимизация широко применяется для MVM и других операций линейной алгебры, выполняемых экспертами в области ее приложений [5]. Для того чтобы упростить разработку новых высокопроизводительных реализаций под новые архитектуры процессоров, специалистами была предложена автонастройка и аналитическое моделирование [7]. Данные подходы используются для нахождения значений параметров созданных реализаций, обеспечивающих наилучшую производительность программ.

В случае автонастройки время выполнения рассматриваемой реализации измеряется для каждого набора значений параметров. Аналитическое моделирование позволяет смоделировать потребление ресурсов гипотетического процессора (ГП) [7] с целью установления зависимостей между производительностью и значениями параметров рассматриваемой реализации. Для определения наилучших значений параметров реализации MVM в работе предложена модификация аналитической модели, которая использовалась в библиотеке BLIS для MMM.

1.1. Модель гипотетического процессора

ГП описывается следующим образом [7].

- **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 $N_{\text{L}_i} = 1$.

Авторами предложена модификация ГП, которая определяет инструкции предвыборки, помещающие данные в кэш первого уровня.

- **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}.

Помимо инструкций предвыборки модификация определяет минимальное количество тактов L_{VLOAD}, которое должно быть совершено перед началом выполнения новой зависимой по данным векторной инструкции загрузки данных на векторный регистр процессора.

Вычислительная сложность MVM составляет $O(N^2)$, что в N раз меньше, чем вычислительная сложность MMM. Вследствие этого каждый элемент умножаемой матрицы используется только $O(1)$ раз, а не $O(N)$ как в случае MMM. Следовательно, стоимость загрузки элементов матрицы из памяти на регистры процессора преобладает над количеством выполняемых операций. Предвыборка данных, загружающая элементы матрицы в кэш первого уровня до момента их использования, может уменьшить количество промахов кэша и, следовательно, уменьшить стоимость загрузки элементов матрицы. Как следствие, эффективная предвыборка необходима в случае MVM.

1.2. Алгоритмы вычисления MVM

Алгоритм 1 вычисления MVM для случая транспонированной матрицы имеет следующий вид:

Алгоритм 1: Вычисление MVM. Случай транспонированной матрицы

```
1  begin
2      for j_c = 0...N step N_c do
3          for i_c = 0...M step M_c do
4              for j_b = 0...N_c step N_b do
5                  Выполняем предвыборку M_c × N_b элементов матрицы A с шагом D
6                  for i_b = 0...M_c step 1 do
7                      I = i_c + i_b
8                      acc(0:N_r − 1) = x(I)
9                      for j_r = 0...N_b step N_r do
10                         𝕁 = j_c + j_b + j_r : j_c + j_b + j_r + N_r − 1
11                         y(𝕁) += A(I, 𝕁) · acc(0 : N_r − 1)
12                     end
13                 end
14             end
15         end
16     end
17 end
```

Алгоритм 2 вычисления MVM для случая нетранспонированной матрицы имеет следующий вид:

Алгоритм 2: Вычисление MVM. Случай нетранспонированной матрицы

```
1  begin
2      for j_c = 0...N step N_c do
3          for i_c = 0...M step M_c do
4              for j_r = 0...N_c step N_r do
5                  if j_r mod 4C_L1/S_DATA == 0 then
6                      Предвыборка M_c × 4C_L1/S_DATA элементов A с шагом D
7                      Предвыборка 4C_L1/S_DATA элементов x с шагом D
8                  end
9              end
10             J = j_c + j_r : j_c + j_r + N_r − 1
11             acc(0, 0 : N_r − 1) += A(i_c, J) · x(J)
12             ...
13             acc(M_c − 1, 0 : N_r − 1) += A(i_c + M_c − 1, J) · x(J)
14         end
15         y(i_c) += Σ_{i=0}^{N_r−1} acc(0, i)
16         ...
17         y(i_c + M_c − 1) += Σ_{i=0}^{N_r−1} acc(M_c − 1, i)
18     end
19 end
```

В процессе выполнения алгоритмов 1 и 2 матрицы и векторы разбиваются на блоки. Это влияет на выполняемую предвыборку данных, загружающую элементы матрицы в кэш первого уровня, а также позволяет использовать элементы блоков, хранящихся в кэш-памяти, повторно. Размеры блоков и размер шага предвыборки являются параметрами алгоритма. Их оптимальные значения определяются с помощью гипотетического процессора, описанного в разделе 1.1.

В случае транспонированной матрицы формулы для нахождения значений параметров имеют следующий вид:

$$N_b = \min\left\{ \max\left\{N_{\text{VFMA}} L_{\text{VFMA}} N_{\text{VEC}}, (N_{\text{REG}} - 2) N_{\text{VEC}}\right\}, \left\lceil \frac{N_{L_1} C_{L_1}}{N_{\text{VEC}} S_{\text{DATA}}} \right\rceil N_{\text{VEC}} \right\},$$

$$N_r = N_{\text{VEC}}, \quad M_c = \min\left\{W_{L_2} - 1, W_{L_1}\right\}, \quad N_c = N_{L_2} C_{L_2} / S_{\text{DATA}},$$

$$D = \left\lceil \frac{L_{\text{prefetch}}}{\lceil M_c \lceil N_b S_{\text{DATA}} / C_{L_1} \rceil / N_{\text{prefetch}} \rceil + M_c \left(\lceil N_b / (N_{\text{VEC}} N_{\text{VFMA}}) \rceil + L_{\text{VLOAD}} \right)} \right\rceil.$$

В случае нетранспонированной матрицы формулы для нахождения значений параметров имеют следующий вид:

$$N_c = N_{L_2} C_{L_2} / S_{\text{DATA}}, \quad N_r = N_{\text{VEC}} \left\lceil \gamma / \min\left\{W_{L_1}, W_{L_2} - 1\right\} \right\rceil, \quad M_c = \lceil \gamma N_{\text{VEC}} / N_r \rceil,$$
$$\gamma = N_{\text{VFMA}} L_{\text{VFMA}},$$

$$D = \left\lceil \frac{L_{\text{prefetch}}}{\lceil 4(M_c+1)/N_{\text{prefetch}} \rceil + 4C_{L_1}\left(\lceil M_c N_r / (N_{\text{VFMA}} N_{\text{VEC}}) \rceil + L_{\text{VLOAD}} \right) / (S_{\text{DATA}} N_r)} \right\rceil .$$

Значение параметров N_{REG}, N_{VEC}, S_{DATA}, N_{VFMA}, L_{VFMA}, N_{L_i}, C_{L_i}, W_{L_i}, L_{prefetch}, N_{prefetch}, L_{VLOAD} являются параметрами модели гипотетического процессора, описанной в разделе 1.1.

В реальных случаях параметры определяются техническими характеристиками целевого процессора.

1.3. Комплекс программ

Для автоматизированного распознавания и оптимизации MVM в промышленных компиляторах применяется полиэдральная компиляция. Основой такого подхода служит полиэдральное представление программы [14].

Определение. Полиэдральная модель — это математическое представление (фреймворк) для моделирования и оптимизации доступа к памяти, производимого в группах вложенных циклов, не рассматривая отдельные вычисления.

Инфраструктура для полиэдральной компиляции выполняет построение и оптимизацию полиэдрального представления программы, используя промежуточное представление программного кода, генерируемого компилятором. Промежуточное представление создается фронтендом компилятора с целью дальнейшей оптимизации и генерации кода для выбранной архитектуры [15].

В работах [9, 10] авторами показано, что аналитическое моделирование MMM, выполняемое BLIS, и автоматизированная оптимизация и распараллеливание, выполняемые над полиэдральным представлением, могут быть использованы для достижения производительности реализаций, созданных для случая свертки тензоров TC. Отметим, что MVM является важным частным случаем TC.

На рис. 1 изображен комплекс автоматизированной оптимизации обобщенных тензорных операций (AOOTO), представленный в работах [9, 10]. Комплекс AOOTO имеет три основные части: фронтенд, выполняющий компиляцию программы и построение промежуточного представления; инфраструктуру для построения полиэдрального представления и его модификаций, например, для распознавания TC и ее оптимизации (на рис. 1 выделено цветом); бэкенд, выполняющий оптимизацию промежуточного кода программы, а также генерацию ассемблерного кода для целевой архитектуры. Такая структура позволяет обеспечить тестируемость описанных частей и их взаимозаменяемость.

В качестве фронтенда, выполняющего компиляцию в комплексе AOOTO, использован фронтенд Clang [16]. Комплекс оптимизаций Polly [15] применен в качестве инфраструктуры для построения полиэдрального представления, а также в качестве основы для реализации распознавания TC и его оптимизации. Внешняя библиотека LLVM Core [16] использована в качестве бэкенда, выполняющего оптимизацию промежуточного представления программы, а также генерацию ассемблерного кода для целевой архитектуры.

Существует множество подходов для оптимизации TC по времени выполнения. В частности, реализация TC может быть оптимизирована, используя последовательности оптимизаций циклов [17]. Для оптимизации TC могут быть использованы высокопроизводительные готовые реализации. Фреймворки GETT (The general matrix multiplication (GEMM)-like

Tensor-Tensor multiplication) [18] и TBLIS (Tensor-Based Library Instantiation Software) [19] используют микро-ядра МММ. Микро-ядра являются частями готовой реализации МММ, содержащие векторные инструкции. Фреймворк Polly использует микро-ядра МММ и генерирует их автоматизировано.

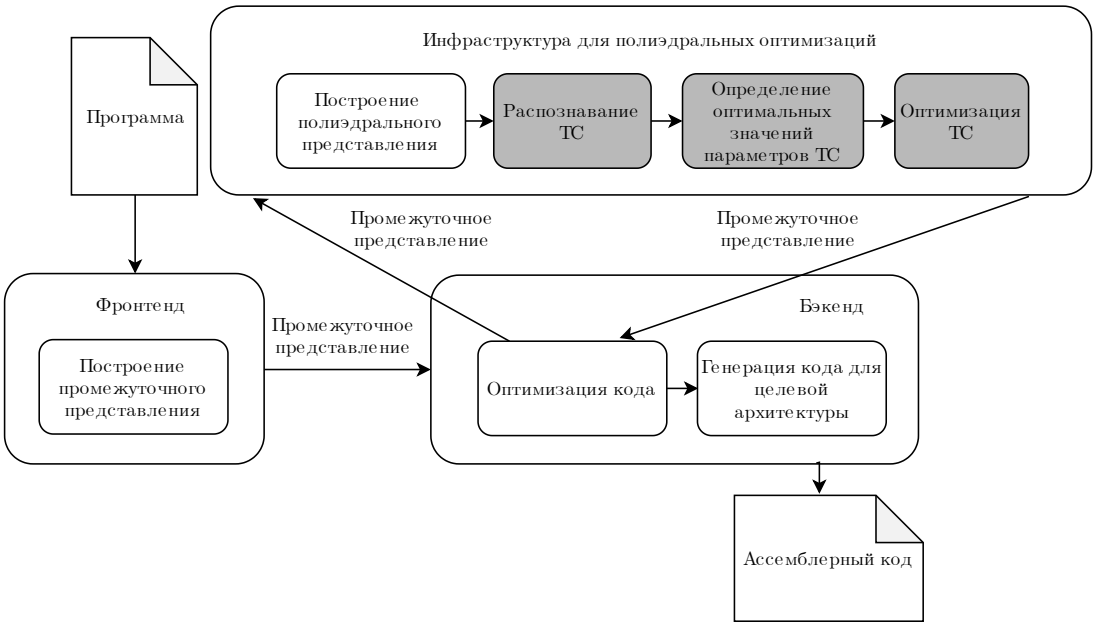

Рис. 1. Комплекс программ

Polly не поддерживает оптимизацию MVM. В ближайшее время предполагается внедрение описанного алгоритма ААО MVM в Polly. Новые инструкции полиэдрального представления могут быть сгенерированы для выполнения предвыборки данных в кэш первого уровня выбранных итераций циклов. Для нетранспонированных матриц в MVM нужно сгенерировать и векторизовать редукцию элементов вычисляемого вектора [5, 20].

2. Численные эксперименты

2.1. Постановка обратной задачи гравиметрии и метод решения

Рассмотрим трехмерную структурную обратную задачу гравиметрии о восстановлении поверхности раздела между средами.

Задача состоит в нахождении функции $\zeta = \zeta(x, y)$, описывающей поверхность раздела сред постоянной плотности σ_1 и σ_2 по гравитационному полю $\Delta g(x, y, 0)$, измеренному на некоторой площади земной поверхности, и скачку плотности на границе раздела $\Delta\sigma = \sigma_1 - \sigma_2$. Пусть $z = h$ — асимптотическая плоскость для данной поверхности раздела ζ такая, что $\lim\limits_{x,y\to\pm\infty} \zeta(x, y) = h$.

Задача описывается двумерным нелинейным интегральным уравнением первого рода [21]:

$$f\Delta\sigma \int\limits_{-\infty}^{\infty}\int\limits_{-\infty}^{\infty} \left\{ \frac{1}{\sqrt{((x-x')^2 + (y-y')^2 + \zeta^2(x', y'))}} - \right.$$
$$\left. - \frac{1}{\sqrt{((x-x')^2 + (y-y')^2 + h^2)}} \right\} dx'dy' = \Delta g(x, y, 0), \tag{1}$$

где f — гравитационная постоянная.

Предположим, что значения поля вне некоторой прямоугольной области $\Pi = \{(x, y) : a \leqslant x \leqslant b, c \leqslant y \leqslant d\}$ близки к нулю. Проведем дискретизацию области Π на сетке $N \times M$: $(x_i, y_j), \quad i = 1, ..., N, \quad j = 1, \ldots, M$ с шагами Δx и Δy.

После аппроксимации интегрального оператора по квадратурным формулам получим уравнение

$$\begin{aligned} & f\Delta\sigma\Delta x\Delta y\times \\ & \times\sum_{i=1}^{N}\sum_{j=1}^{M}\left[\frac{1}{\sqrt{(x_i-x_u)^2+(y_j-y_v)^2+\zeta^2(x_i,y_j)}}-\right. \\ & \left.-\frac{1}{\sqrt{(x_i-x_u)^2+(y_j-y_v)^2+h^2}}\right]=\Delta g(x_u,y_v,0), \\ & u=1,...,M, \quad v=1,...,N. \end{aligned} \tag{1a}$$

Правую часть $\Delta g(x_u, y_v, 0)$ представим в виде вектора

$$\begin{aligned} & F = \{F_{(v-1)M+u}\} = \{\Delta g(x_u, y_v, 0)\}, \\ & u = 1, ..., M, \quad v = 1, ..., N \end{aligned}$$

и искомую поверхность представим в виде вектора

$$z = \{z_{(j-1)M+i}\} = \{\zeta(x_i, y_j), \quad i = 1, ..., M, \quad j = 1, ..., N\}.$$

Тогда систему (1a) запишем в операторном виде

$$A(z) = F. \tag{2}$$

Для решения системы (2) будем использовать метод Левенберга—Марквардта [22]

$$z^{k+1} = z^k - \gamma\left[\left(A'(z^k)\right)^{\intercal} A'(z^k) + \alpha I\right]^{-1} \times A'(z^k)^{\intercal}(A'(z^k) - F), \tag{3}$$

где z^k — приближенное решение на k-ой итерации, $A'(z^k)$ — производная Фреше оператора A в точке u^k, I — единичный оператор, γ — демпфирующий множитель, α — параметр регуляризации.

В качестве начального приближения используется $z^0 \equiv h$. Критерием останова является условие $\left\|A(z^k) - F\right\| / \|F\| \leqslant \varepsilon_1$ при некотором $\varepsilon_1 < 0$.

Вместо выполнения трудоемкой процедуры обращения матрицы воспользуемся следующим приемом. На каждом шаге итерационного метода Левенберга—Марквардта [22] решаем систему линейных алгебраических уравнений

$$Bz = b, \tag{4}$$

где

$$B = \left(A'(z^k)\right)^{\intercal} A'(z^k) + \alpha I, \quad b = \left[\left(A'(z^k)\right)^{\intercal} A'(z^k) + \alpha I\right] z^k - \gamma A'(z^k)^{\intercal}(A'(z^k) - F).$$

Для решения (4) используется метод минимальных невязок [22]

$$z^{l+1} = z^l - \frac{\langle B(Bz^l - b), Bz^l - b\rangle}{\|B(Bz^l - b)\|^2}(Bz^l - b), \tag{5}$$

где z^l — приближенное решение на l-ой итерации метода минимальных невязок.

В качестве начального приближения используется $z^0 \equiv 0$. Критерием останова является условие $\left\|Bz^l - b\right\|/\|b\| \leqslant \varepsilon_2$ при некотором $\varepsilon_2 < 0$.

Замечание. Метод Левенберга—Марквардта (3) и метод минимальных невязок (5) на каждом шаге итерационного процесса (3) требуют выполнения значительного количества операций MVM. Поэтому предложенный алгоритм автоматической оптимизации MVM может быть использован для автоматизированной и эффективной оптимизации соответствующей программы.

2.2. Результаты экспериментов

В данном разделе проведено сравнение производительности нескольких реализаций программного кода для решения обратной задачи гравиметрии (раздел 2.1) с использованием библиотек линейной алгебры Intel MKL, BLIS, OpenBLAS с реализацией, полученной с использованием предлагаемого алгоритма AAO MVM (см. раздел 1).

Для выполнения экспериментов использовались два 18-ядерных процессора Intel Xeon E5-2697 v4 суперкомпьютера «Уран». Для распараллеливания программы применялись директивы стандарта OpenMP и группы из 36 потоков.

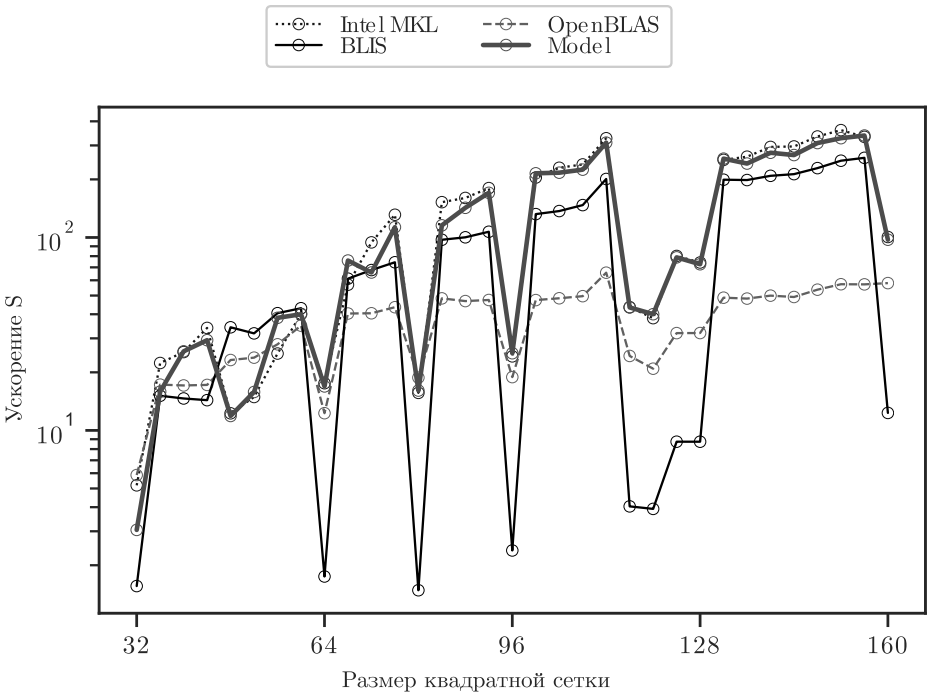

Рис. 2. Ускорение, полученное для обратной задачи гравиметрии

Определим ускорение $S = T_{serial}/T_{parallel}$, где T_{serial} – время выполнения последовательного кода программы с применением стандартных оптимизаций компилятора третьего уровня (O3), $T_{parallel}$ – время выполнения параллельного кода программы с использованием библиотечных процедур либо кода, оптимизированного с помощью предлагаемого алгоритма.

На рис. 2 для 36 ядер процессора Intel Xeon E5-2697 v4 суперкомпьютера «Уран» изображены коэффициенты ускорения для решения обратной задачи гравиметрии методом Левенберга—Марквардта на квадратных сетках с размерами, изменяющимися от 32 до

160 с шагом 4. Представленные коэффициенты ускорения получены для разработанного алгоритма AAO MVM, а также для библиотек BLIS, OpenBLAS и Intel MKL.

В большинстве случаев наилучшие результаты получены с использованием библиотеки Intel MKL. Результаты алгоритма AAO MVM сравнимы с библиотечными для всех рассмотренных размеров сеток. В среднем они отличаются не более чем на 1% и превосходят результаты библиотек OpenBLAS и BLIS.

На основании результатов численных экспериментов можно сделать вывод, что реализация программы, использующей описанный метод оптимизации MVM, сравнима по производительности с реализациями, использующими библиотеки BLIS, OpenBLAS и Intel MKL, оптимизированные вручную. Результаты численных экспериментов показывают, что значения параметров реализации, полученные с использованием аналитического моделирования (раздел 1), не зависят от размерности задачи.

Заключение

В данной работе представлен алгоритм автоматизированной оптимизации матрично-векторного произведения по времени выполнения, использующийся на этапе компиляции. Алгоритм AAO MVM основан на оптимизации полиэдрального представления программы и моделировании вычислений на гипотетическом многоядерном процессоре. В отличие от подходов, основанных на ручной настройке и автонастройке, описанный алгоритм может применяться для создания новых реализаций матрично-векторного произведения в условиях недоступности целевой архитектуры процессора и ограниченности времени выполнения.

Алгоритм использован для оптимизации программного кода, реализующего решение структурной обратной задачи гравиметрии о нахождении поверхности раздела сред методом Левенберга—Марквардта. Проведено сравнение производительности полученной реализации с реализациями на основе оптимизированных библиотек линейной алгебры Intel MKL, BLIS, OpenBLAS. Результаты численных экспериментов показывают сравнимость предложенного алгоритма по эффективности с подходами, созданными с использованием эмпирического поиска при доступе к целевым архитектурам процессоров. В дальнейшем планируется внедрение описанного алгоритма в комплекс оптимизаций Polly с целью использования в процессе компиляции, выполняемой фронтендом Clang.

Перспективным направлением будущих исследований является применение аналитического моделирования для оптимизации матрично-векторного произведения матриц специального вида (матрица Теплица и др.). В основе таких подходов может использоваться симметрия элементов и другие свойства матриц.

Литература

1. Yu J., Lukefahr A., Palframan D., *et al.* Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism // SIGARCH Computer Architecture News. 2017. Vol. 45, no. 2, P. 548–560. DOI: 10.1145/3140659.3080215.

2. 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.

3. Kaushik D., Gropp W., Minkoff M., *et al.* Improving the Performance of Tensor Matrix Vector Multiplication in Cumulative Reaction Probability Based Quantum Chemistry Codes // High Performance Computing (HiPC 2008). Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

P. 120–130. DOI: 10.2172/928654.

4. Martyshko P.S., Akimova E.N., Misilov V.E. Solving the structural inverse gravity problem by the modified gradient methods // Izvestiya, Physics of the Solid Earth. 2016. Vol. 52, no. 5. P. 704–708. DOI: 10.1134/S1069351316050098.
5. Hassan S.A., Mahmoud M.M., Hemeida A., *et al.* Effective Implementation of Matrix-Vector 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.
6. 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.
7. 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.
8. Frison G. Algorithms and Methods for High-Performance Model Predictive Control // Technical University of Denmark. 2016. 345 c.
9. Akimova E.N., Gareev R.A. Algorithm of Automatic Parallelization of Generalized Matrix Multiplication // CEUR Workshop Proceedings. 2017. Vol. 2005. P. 1–10.
10. Gareev R., Grosser T., Kruse M. 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.
11. Intel. Intel Math Kernel Library (Intel MKL). URL: `https://software.intel.com/en-us/mkl` (дата обращения: 27.10.2019).
12. Van Zee F.G., Van De Geijn R.A. 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.
13. Xianyi Z., Qian W., Yunquan Z. Model-driven level 3 BLAS performance optimization on Loongson 3A processor // Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on Parallel and Distributed Systems, IEEE. 2012. P. 684–691. DOI: 10.1109/icpads.2012.97.
14. Feautrier P., Lengauer C. Polyhedron Model // Encyclopedia of Parallel Computing. Springer US, Boston, MA. 2011. P. 1581–1592. DOI: 10.1007/978-0-387-09766-4_502.
15. Grosser T., Größlinger A., Lengauer C. Polly—Performing polyhedral optimizations on a low-level intermediate representation // Parallel Process. Lett. 2012. Vol. 22. no. 4. DOI: 10.1142/S0129626412500107.
16. Lattner C. LLVM: An Infrastructure for Multi-Stage Optimization // Master's Thesis. 2002. URL: `http://llvm.cs.uiuc.edu` (дата обращения: 27.10.2019).
17. Apra E., Klemm M., Kowalski K. Efficient Implementation of Many-Body Quantum Chemical Methods on the Intel®Xeon Phi™Coprocessor // International Conference for High Performance Computing, Networking, Storage and Analysis. 2014. P. 674–684. DOI: 10.1109/SC.2014.60
18. Springer P., Bientinesi P. Design of a high-performance GEMM-like tensor-tensor multiplication // ACM Trans. Math. Softw. 2018. Vol. 44, no. 3, Article 28. DOI: 10.1145/3157733.

19. Matthews D. High-performance tensor contraction without BLAS // SIAM Journal on Scientific Computing. 2016. Vol. 40. DOI: 10.1137/16M108968X

20. Doerfert J., Streit K., Hack S., *et al.* Polly's polyhedral scheduling in the presence of reductions // arXiv preprint. 2015. arXiv:1505.07716.

21. Numerov B.V. Interpretation of gravitational observations in the case of one contact surface // Dokl. Akad. Nauk SSSR. 1930. P. 569–574.

22. Vasin V.V., Eremin I.I. Operators and iterative processes of Fejér type: theory and applications // Walter de Gruyter. 2009. Vol. 53. 155 p. DOI: 10.1515/9783110218190.

Акимова Елена Николаевна, д.ф.-м.н., доцент, ведущий научный сотрудник Института математики и механики им. Н.Н. Красовского Уральского отделения РАН (Екатеринбург, Российская Федерация); профессор кафедры информационных технологий и систем управления, Институт радиоэлектроники и информационных технологий-РтФ, УрФУ имени первого Президента России Б.Н. Ельцина (Екатеринбург, Российская Федерация)

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

DOI: 10.14529/cmse200105

APPLICATION OF ANALYTICAL MODELING OF MATRIX-VECTOR MULTIPLICATION ON MULTICORE PROCESSORS

© 2020 E.N. Akimova[1,2], R.A. Gareev[1]

[1] *Ural Federal University*
(Mira 19, Yekaterinburg, 620002 Russia),
[2] *N.N. Krasovskii Institute of Mathematic sand Mechanics, UrB RAS*
(S. Kovalevskaya 16, Yekaterinburg, 620990 Russia)
E-mail: aen15@yandex.ru, gareevroman@gmail.com
Received: 31.01.2020

Many areas of study and their applications such as machine learning, data mining, quantum chemistry, mathematical physics, and high-performance computing require effective implementation of matrix-vector multiplication. In this work, we present an overview of the algorithm for automatic optimization of matrix-vector multiplication. The algorithm models computations on the hypothetical multicore processor, which is introduced by the authors, and applies polyhedral modeling. In comparison with methods, which rely on manual tuning and auto-tuning, the algorithm can be utilized when the execution time is a critical factor and the target platform is not accessible. We apply the approach to optimize an implementation of the solution of the inverse gravimetry problem of finding an interface between the layers, which uses the iterative Levenberg—Marquardt method. The performance of the obtained implementation is compared with the performances produced by implementations, which are based on MKL, BLIS, and OpenBLAS. Results of the experimental evaluation show that the considered approach is comparable with the approaches, which are created on target architectures using manual tuning.

Keywords: compilers, linear algebra, matrix-vector operations, analytical modeling, inverse gravimetry problem.

FOR CITATION

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.

References

1. Yu J., Lukefahr A., Palframan D., *et al.* Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. SIGARCH Computer Architecture News. 2017. Vol. 45, no. 2. P. 548–560. DOI: 10.1145/3140659.3080215.

2. 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.

3. Kaushik D., Gropp W., Minkoff M., *et al.* Improving the Performance of Tensor Matrix Vector Multiplication in Cumulative Reaction Probability Based Quantum Chemistry Codes. High Performance Computing (HiPC 2008). Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. P. 120–130. DOI: 10.2172/928654.

4. Martyshko P.S., Akimova E.N., Misilov V.E. Solving the structural inverse gravity problem by the modified gradient methods. Izvestiya, Physics of the Solid Earth. 2016. Vol. 52, no. 5. P. 704–708. DOI: 10.1134/S1069351316050098.

5. Hassan S.A., Mahmoud M.M., Hemeida A., *et al.* Effective Implementation of Matrix-Vector 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.

6. 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.

7. 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.

8. Frison G. Algorithms and Methods for High-Performance Model Predictive Control. Technical University of Denmark. 2016. 345 p.

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

10. Gareev R., Grosser T., Kruse M. 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.

11. Intel. Intel Math Kernel Library (Intel MKL). Available at: `https://software.intel.com/en-us/mkl` (accessed: 27.10.2019).

12. Van Zee F.G., Van De Geijn R.A. 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.

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

14. Feautrier P., Lengauer C. Polyhedron Model. Encyclopedia of Parallel Computing. Springer US, Boston, MA. 2011. P. 1581–1592. DOI: 10.1007/978-0 387-09766-4_502.

15. Grosser T., Größlinger A., Lengauer C. Polly—Performing polyhedral optimizations on a low-level intermediate representation. Parallel Process. Lett. 2012. Vol. 22, no. 4. DOI: 10.1142/S0129626412500107.

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

17. Apra E., Klemm M., Kowalski K. Efficient Implementation of Many-Body Quantum Chemical Methods on the Intel®Xeon Phi™Coprocessor. International Conference for High Performance Computing, Networking, Storage and Analysis. 2014. P. 674–684. DOI: 10.1109/SC.2014.60

18. Springer P., Bientinesi P. Design of a high-performance GEMM-like tensor-tensor multiplication. ACM Trans. Math. Softw. 2018. Vol. 44, no. 3, Article 28. DOI: 10.1145/3157733.

19. Matthews D. High-performance tensor contraction without BLAS. SIAM Journal on Scientific Computing. 2016. Vol. 40. DOI: 10.1137/16M108968X.

20. Doerfert J., Streit K., Hack S., *et al.* Polly's polyhedral scheduling in the presence of reductions. arXiv preprint. 2015. arXiv:1505.07716.

21. Numerov B.V. Interpretation of gravitational observations in the case of one contact surface. Dokl. Akad. Nauk SSSR. 1930. P. 569–574.

22. Vasin V.V., Eremin I.I. Operators and iterative processes of Fejér type: theory and applications. Walter de Gruyter. 2009. Vol. 53. 155 p. DOI: 10.1515/9783110218190.