Методы оптимизации выполнения тензорных операций на многоядерных процессорах

05.13.11 - математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени кандидата физико-математических наук

Роман Альбертович Гареев

Научный руководитель:
АКИМОВА Елена Николаевна,
д.ф.-м.н., вед. н.с. ИММ УрО РАН,
профессор УрФУ

Цель диссертационной работы

Цель работы: данной работы является разработка методов оптимизации времени выполнения тензорных операций без ручной настройки и автонастройки, а также создание программной системы автоматического выполнения оптимизаций и их автоматического распараллеливания во время компиляции на многоядерных процессорах общего назначения.

Основные задачи

1. Разработать модель гипотетического процессора, которая позволяет сократить время выполнения матрично-векторных операций и их обобщений на замкнутые полукольца с элементами из множества вещественных чисел.
2. Разработать новые алгоритмы выполнения тензорных операций константной сложности относительно размерности тензоров, уменьшающие время выполнения таких операций.
3. Разработать программную систему для автоматической оптимизации времени выполнения тензорных операций и их автоматического распараллеливания при компиляции программ для многоядерных процессоров общего назначения.
4. Провести вычислительные эксперименты, подтверждающие эффективность разработанной программной системы по сравнению с аналогами, использующими ручную настройку и автонастройку.

Работы по теме диссертации

1	Goto K., Geijn R.A. van de. Anatomy of High-Performance Matrix Multiplication // ACM Transactions on Mathematical Software. 2008. Vol. 34, no. 3. P. 12:1—12:25. DOI: 10.1145/1356052.1356053.	Алгоритм вычисления матричного произведения (MMM), реализованный во многих оптимизированных библиотеках.
2	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.	Модель гипотетического процессора (ГП), позволяющая вывести оптимальные параметры алгоритма MMM в фреймворке BLIS.
3	Matthews D. High-Performance Tensor Contraction without BLAS // SIAM Journal on Scientific Computing. 2018. Vol. 40, no. 1. P. C1–C24. DOI: 110.1137/16m108968x.	Алгоритм сведения оптимизации свертки тензоров (TC) к алгоритмам оптимизации MMM и MVM.
4	Hassan S.A., Mahmoud M.M.M., Hemeida A.M., Saber M.A. 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.	Алгоритм вычисления матрично-векторного произведения (MVM).
5	Sedukhin S., Miyazaki T., Kuroda K. Orbital Systolic Algorithms and Array Processors for Solution of the Algebraic Path Problem // IEICE Transactions. 2010. Vol. 93-D, no. 3. P. 534–541. DOI: 10.1587/transinf.E93.D.534.	Обобщения MMM на замкнутые полукольца матриц.

Оптимизация ТС

Определение. d-мерный тензор $\mathcal{T} \in \mathbb{R}^{n_{u_0} \times \cdots \times n_{u_{d-1}}}$ может быть определен как:

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

Определение. Пусть $\mathcal{A}$, $\mathcal{B}$, и $\mathcal{C}$ — 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 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 $\pi_B(PJ)$ — перестановки индексов, $\alpha, \beta \in \mathbb{R}$.

Matthews D. High-Performance Tensor Contraction without BLAS // SIAM Journal on Scientific Computing. 2018. Vol. 40, no. 1. P. C1 – C24. DOI: 110.1137/16m108968x.

Модификация модели ГП (отображается на реальный процессор)

- **Архитектура загрузки/сохранения и векторные регистры:** данные должны быть загружены в регистры процессора перед тем, как с ними могут быть выполнены вычисления. N_{REG} – кол-во векторных регистров. N_{VEC} – кол-во значений размерности S_{DATA}, которые может содержать каждый из векторных регистров.
- **Векторные инструкции:** N_{VMMA} – кол-во операций VMMA, вычисляемых процессором за один такт. Отдельная VMMA выполняет N_{VEC} невекторных умножений и сложений, составляющих MMA. L_{VMMA} – минимальное кол-во тактов, которое должно быть совершено перед началом выполнения новой зависимой по данным VMMA.
- **Кэш память**: весь кэш данных – это множественно-ассоциативный кэш. Каждый уровень кэша L_i характеризуется следующими параметрами: C_{L_i} – размер линии кэша, W_{L_i} – степень ассоциативности, N_{L_i} – кол-во множеств, S_{L_i} – размер уровня кэша L_i, где $S_{L_i} = N_{L_i} C_{L_i} W_{L_i}$.
- **Инструкции предвыборки:** N_{prefetch} – кол-во инструкций предвыборки, которое может быть выполнено за один такт. L_{prefetch} – кол-во тактов, составляющих задержку каждой инструкции. Каждая инструкция может загрузить данные, размер которых равен C_{L_i}.

Красным цветом выделены разработки автора.

Операция ММА[$\otimes$, $\oplus$] и её приложения

Определение. ММА[$\otimes$, $\oplus$] - операция вида $C \leftarrow C \,\overline{\oplus}\, A \,\overline{\otimes}\, B$, где $\overline{\oplus}$ и $\overline{\otimes}$ - операции из замкнутого полукольца матриц $\{S^{N\times N}, \overline{\oplus}, \overline{\otimes}, \bar{O}, \bar{I}\}$, определенного над замкнутым полукольцом $\{S, \oplus, \otimes, 0, 1\}$.

- **Транзитивное замыкание бинарного отношения:**

ММА[$\wedge$, $\vee$] = $c_{ij} \leftarrow c_{ij} \vee \left\{\vee_k \{a_{ik} \wedge b_{kj}\}\right\}$

- **Задача о кратчайшем пути:**

ММА[$+$, min] = $c_{ij} \leftarrow min\left\{c_{ij}, \min_k\{a_{ik} + b_{kj}\}\right\}$

- **Задача о самом широком пути:**

ММА[$+$, max] = $c_{ij} \leftarrow max\left\{c_{ij}, \max_k\{a_{ik} + b_{kj}\}\right\}$

- **Задача о пути с наибольшей надежностью**:

ММА[$\times$, max] = $c_{ij} \leftarrow max\left\{c_{ij}, \max_k\{a_{ik} \times b_{kj}\}\right\}$

Задача о пути с наименьшей надежностью:

ММА[$\times$, min] = $c_{ij} \leftarrow min\left\{c_{ij}, \min_k\{a_{ik} \times b_{kj}\}\right\}$

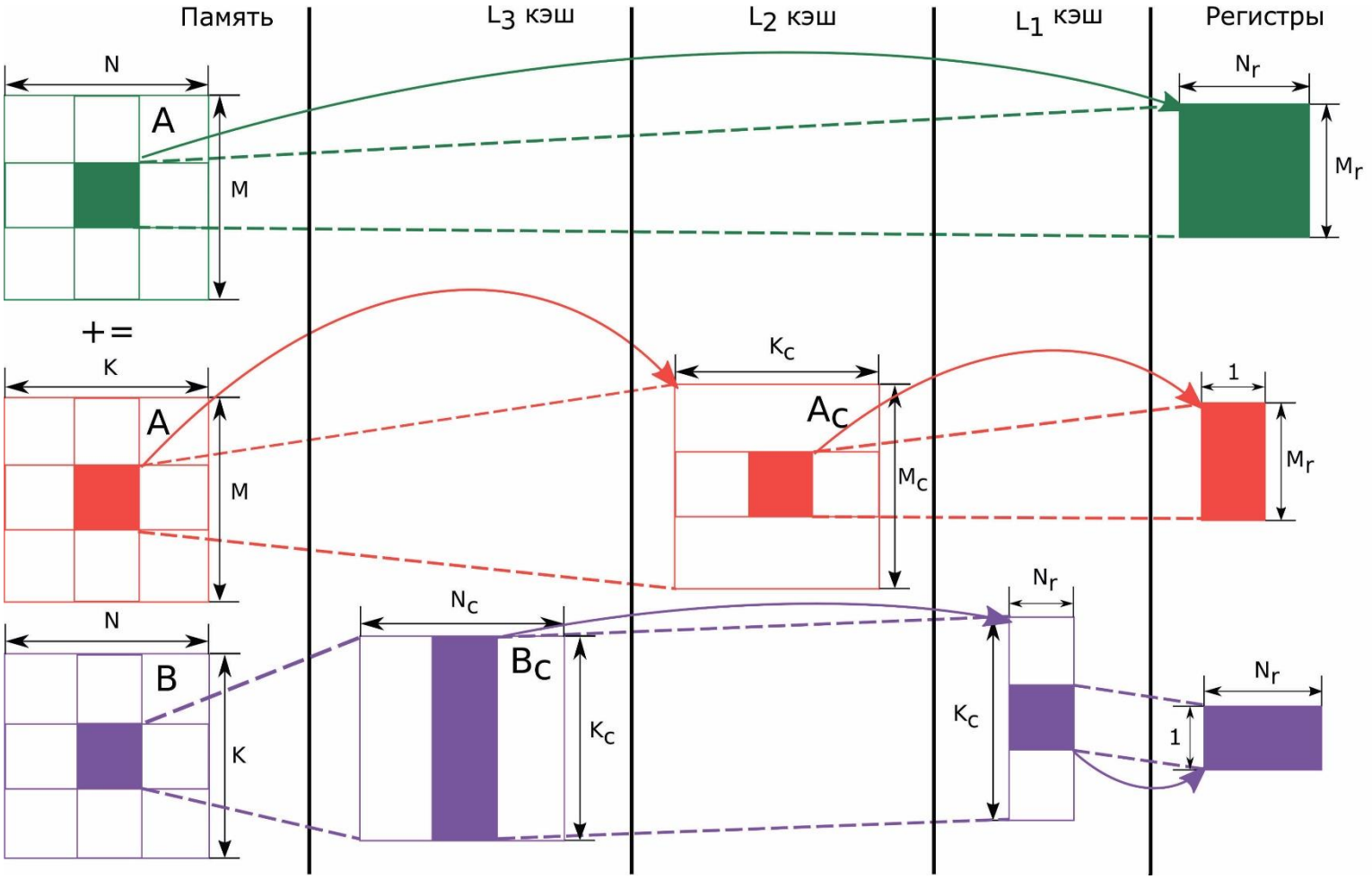
- **Нахождение путей наибольшей вместимости**:

ММА[min, max] = $c_{ij} \leftarrow max\left\{c_{ij}, \max_k\{min(a_{ik}, b_{kj})\}\right\}$

Вычисление обобщенного ММ (алгоритм 1)

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

Использование памяти во время выполнения алгоритма 1



Вычисление значений параметров алгоритма 1

$$N_r = \lceil \sqrt{\gamma} / N_{VEC} \rceil N_{VEC}, \quad K_c = \left\lfloor \frac{W_{L1} - 1}{1 + N_r / M_r} \right\rfloor N_{L_1} C_{L_1} / M_r S_{DATA}, \quad M_r = \lceil \gamma / N_r \rceil,$$

$$N_c = \left\lfloor C_{B_c} / K_c S_{DATA} N_r \right\rfloor N_r, \quad M_c = (W_{L_2} - 2) S_{L_2} / K_c S_{DATA} W_{L_2},$$

где $\gamma = N_{VEC} L_{VMMA} N_{VMMA}$, $C_{B_c} -$ количество байтов доступных для B_c.

Формулы для N_r и M_r обеспечивают отсутствие простоя конвейера векторных инструкций ГП в процессе выполнения тела внутреннего цикла алгоритма 1. Формулы для K_c, M_c и N_c основаны на том, что элементы матриц, используемые чаще остальных, должны оставаться в кэш-памяти наименьшего из доступных уровней как можно дольше.

Утверждение 1. Если данные могут быть мгновенно загружены из памяти на векторные регистры, то существуют значения параметров N_r и M_r алгоритма 1, при которых отсутствует простаивание конвейера векторных инструкций ГП в процессе выполнения алгоритма 1.

Вычисление обобщенного MVM. Случай транспонированной матрицы (алгоритм 2)

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

Вычисление значений параметров алгоритма 2

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

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

$$D = \lceil L_{prefetch}/\delta \rceil,$$

где $\delta = \lceil M_c \lceil N_b S_{DATA}/C_{L_1} \rceil / N_{prefetch} \rceil + M_c(\lceil N_b/(N_{VEC}N_{VFMA}) \rceil + L_{VLOAD})$

Формулы для N_b и N_r обеспечивают отсутствие простоя конвейера векторных инструкций ГП в процессе выполнения внутреннего цикла алгоритма 2 и отсутствие вытеснения элементов матрицы A, предвыбираемой в L_1, учитывая ограничение на количество векторных регистров. Формулы для M_c, N_c и D основаны на том, что элементы матриц, используемые чаще остальных, должны оставаться в кэш-памяти наименьшего из доступных уровней как можно дольше.

Вычисление обобщенного MVM. Случай не транспонированной матрицы (алгоритм 3)

1 **begin**
2 **for** $j_c = 0 \ldots \mathrm{N}$ **с шагом** N_c **do**
3 **for** $i_c = 0 \ldots \mathrm{M}$ **с шагом** M_c **do**
4 $I = i_c M_c$, $acc[M_c][N_r]$
5 Выполняем предвыборку элементов A и X с шагом D
6 $\mathbb{J} = j_c N_c + j_r : j_c N_c + j_r + N_r - 1$
7 $acc(0, 0 : N_r - 1) \oplus = A(I, \mathbb{J}) \otimes X(\mathbb{J})$
8 ...
9 $acc(M_c - 1, 0 : N_r - 1) \oplus = A(I + M_c - 1, \mathbb{J}) \otimes X(\mathbb{J})$
10 **end**
11 $Y(i_c) \oplus = \bigoplus_{i=0}^{N_r - 1} acc(0, i)$
12 ...
13 $Y(i_c + M_c - 1) \oplus = \bigoplus_{i=0}^{N_r - 1} acc(M_c - 1, i)$
14 **end**
15 **end**

Вычисление значений параметров алгоритма 3

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

$$M_c = \lceil \gamma/N_r \rceil,\quad \gamma = N_{VMMA} L_{VMMA} N_{VEC},$$

$$D = \lceil L_{\text{prefetch}}/(\left\lceil 4(M_c + 1)/N_{\text{prefetch}}\right\rceil + 4\mathbb{C}_{\text{L}_1}(\left\lceil (M_c N_r)/(N_{\text{VFMA}} N_{\text{VEC}})\right\rceil + L_{\text{VLOAD}})/N_r)\rceil.$$

Формулы для M_c и N_r обеспечивают отсутствие простоя конвейера векторных инструкций ГП в процессе выполнения внутреннего цикла алгоритма 3 и отсутствие вытеснения элементов матрицы A, предвыбираемой в L_1. Формулы для M_c, N_c и D основаны на том, что элементы матриц, используемые чаще остальных, должны оставаться в кэш-памяти наименьшего из доступных уровней как можно дольше.

Утверждение 2. Если данные могут быть мгновенно загружены из памяти на векторные регистры, то существуют значения параметров N_r, N_b и значения параметров M_c и N_r, позволяющие избежать простаивания конвейера векторных инструкций ГП во время выполнения алгоритма 2 и цикла с индуктивной переменной i_c алгоритма 3, соответственно.

Полиэдральное представление программы

В ходе выполнения алгоритма используются оптимизации циклов, которые применяются к полиэдральному представлению программы, предложенному в работе Feautrier P. Полиэдральное представление используется для моделирования и оптимизации доступа к памяти, производимого в гнездах циклов, не рассматривая отдельные вычисления. Для определения компонент полиэдрального представления программы используются целочисленные многогранники и отношения Пресбургера.

```
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    for (p = 0; p < K; p++)
S:    C[i][j] += A[i][p] * B[p][j];
```

- **Области итерирования**

$\{S(i,j,p) | 0 \leq i \leq M \wedge 0 \leq j \leq N \wedge 0 \leq p \leq K\}$

- **Аффинные планы**

$\{S(i,j,p) \rightarrow (i,j,p)\}$

- **Функции доступа к памяти**

$\{S(i,j,p) \rightarrow A(i,p)\}$
$\{S(i,j,p) \rightarrow B(p,j)\}$
$\{S(i,j,p) \rightarrow C(i,j)\}$
$\{S(i,j,p) \rightarrow C(i,j)\}$

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

TC-подобное ядро

Определение. TC-подобное ядро (англ. Tensor Contraction like kernel, сокр. *TC-like kernel*) - множество полностью вложенных циклов.

- Ядро удовлетворяет требованиям полиэдральной модели.
- Без ограничения общности TC-подобное ядро содержит три непустых множества циклов, имеющих одну индуктивную переменную и единичный шаг. Данные множества образуют три непустых набора $I = i_0 \dots i_{r-1}, J = j_0 \dots j_{s-1}$ и $P = p_0 \dots p_{t-1}$.
- Тело цикла TC-подобного ядра, имеющего наибольшую глубину, м. б. представлено в виде C $_{\pi_C(IJ)}$ = E(A $_{\pi_A(IP)}$, B $_{\pi_B(PJ)}$), где A $_{\pi_A(IP)}$, B $_{\pi_B(PJ)}$, C $_{\pi_C(IJ)}$ – обращения к тензорам $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, соответственно; $\pi_C(IJ)$, $\pi_A(IP)$, $\pi_B(PJ)$ – перестановки индексов; E – выражение, содержащее чтения из тензоров $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ и произвольное кол-во чтений из констант.

Оптимизация TC-подобного ядра
алгоритм 4

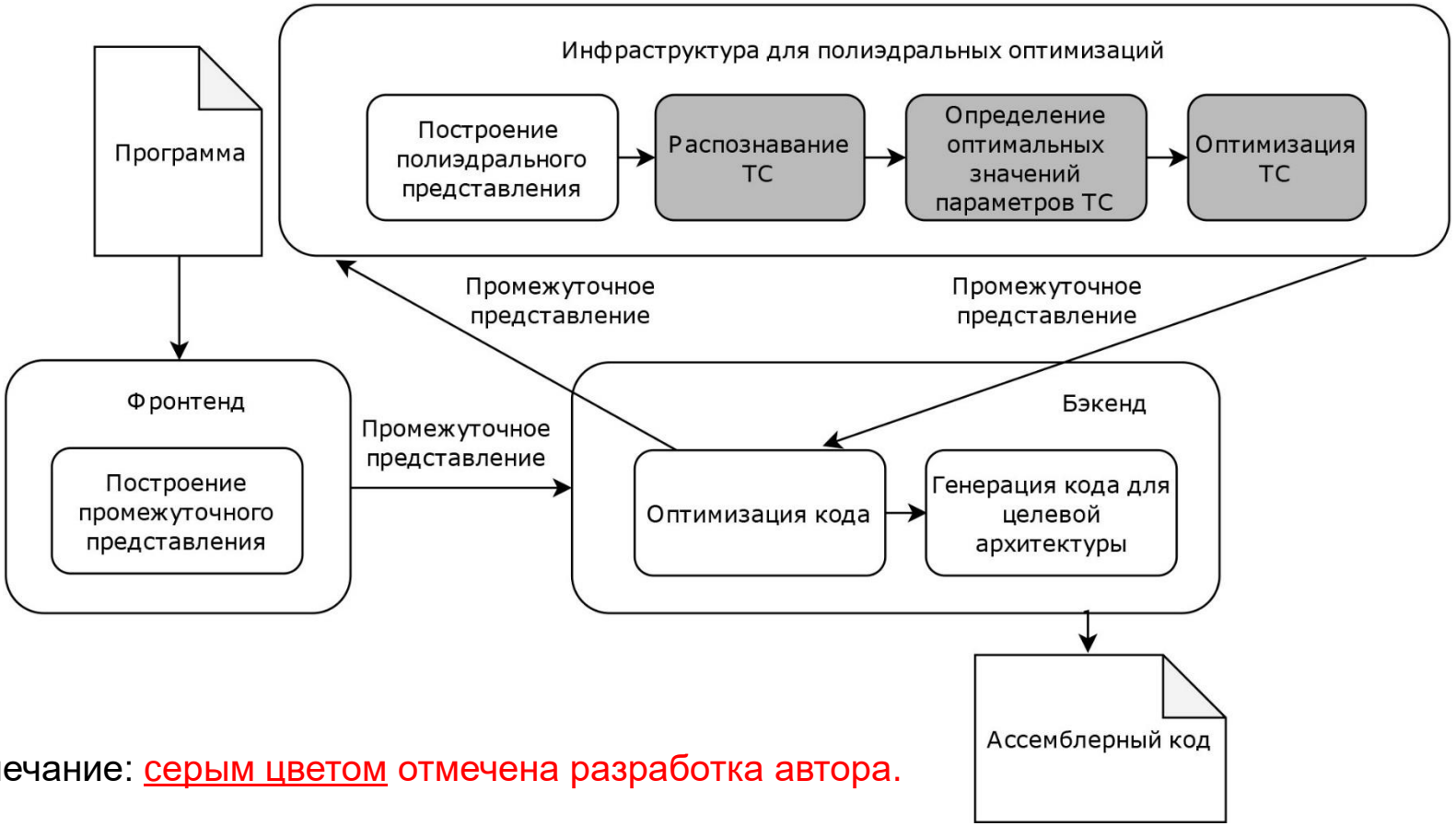
Входные данные:

Утверждение S полиэдрального представления программы. S является частью TC-подобного ядра, и, вследствие этого, представимо в виде C[i][j] = E(A[i][p], B[p][j], C[i][j]), где i, j, p — индукционные переменные циклов; E — выражение, содержащее операции чтения из матриц A, B, C; A[i][p], B[p][j], C[i][j] — обращения к матрицам A, B, C, соответственно.

1. Отождествить каждый цикл с его индукционной переменной.
2. Переставить циклы так, чтобы i, j, p приобрели наибольшую глубину и следующий порядок: j, p, и i, где i имеет наименьшую глубину.
3. Разбить i, j, p на блоки размера M_c, N_c, и K_c, соответственно, получить циклы i_c, j_c и p_c, переставить i_c и p_c.
4. Разбить i_c, j_c, p_c на блоки размера M_r, N_r, и 1, соответственно, получить циклы i_r, j_r, и p_r, удалить p_r.
5. Выделить безусловные области итерирования i_r и j_r и выполнить их размотку (loop unroll).
6. Выполнить упаковку и векторизовать код в p_c.

Выходные данные: Оптимизированный код

Архитектура ПС АОТО



Примечание: серым цветом отмечена разработка автора.

Требуемая информация: W_{L_i}, S_{L_i}, L_{prefetch}, N_{prefetch}, N_{VMMA}, L_{VMMA}

Реализация архитектуры ПС АОТО

- **Фронтенд Clang**: генерация промежуточного представления.

- **Фреймворк Polly**: применен для построения полиэдрального представления, а также в качестве основы для реализации распознавания ТС-подобного ядра и его оптимизации.

- **Внешняя библиотека LLVM Core**: оптимизация промежуточного кода; генерация ассемблерного кода.

Для случая обобщенных ММ указанная оптимизация автора была внедрена в основной код Polly проекта LLVM (Low Level Virtual Machine). Проект победил в Google Summer of Code 2016.

Gareev R., Grosser T. (ETH Zürich, Switzerland), Kruse M. (Argonne National Laboratory, USA). High-Performance Generalized Tensor Operations: A Compiler-Oriented Approach // ACM Transactions on Architecture and Code Optimization (TACO). 2018. Vol. 15, no. 3. P. 34:1–34:27.

Автоматическое распараллеливание ТС

Утверждение 3. Если цикл L содержится в группе полностью вложенных циклов, то значение многопоточной производительности программы может быть вычислено как $F(L)C_R$, где

$$F(L) = N_O/(C_{PAR} + C_W C_R),$$

L – оцениваемый цикл; N_O – кол-во операций с плавающей запятой, вычисляемых циклом L; C_{PAR} – кол-во тактов процессора, требуемых для создания группы из $N_{THREADS}$ потоков; C_W – кол-во сек. требуемых для выполнения цикла L после распараллеливания, C_R – тактовая частота процессора.

Автоматическое распараллеливание программ на основе представленной формулы планируется внедрить в ПС АОТО.

Постановка обратной задачи гравиметрии

Рассматривается задача о восстановлении поверхности раздела между средами по известному скачку плотности $\Delta\sigma$ и гравитационному полю $\Delta\, g$(x,y,0), измеренному на некоторой площади земной поверхности. Предположим, что $z = h$ – асимптотическая плоскость для данной поверхности раздела ζ такая, что $\lim_{x,y\to\pm\infty} \zeta(x,y) = h$. Функция $\zeta = \zeta(x,y)$, описывающая искомую поверхность раздела, удовлетворяет нелинейному двумерному интегральному уравнению Фредгольма первого рода:

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

где f – гравитационная постоянная.

Метод решения обратной задачи гравиметрии

Используя дискретизацию уравнения на сетке $n = M \times N$, где задана правая часть $\Delta g(x, y, 0)$, а также аппроксимацию интегрального оператора A по квадратурным формулам, получаем $A(z) = F$.

Применим итеративно регуляризованный метод Левенберга–Марквардта:

$$z^{k+1} = z^k - \gamma \left[A'\left(z^k\right)^T A'\left(z^k\right) + \alpha I\right]^{-1} \times A'\left(z^k\right)^T \left(A'(z^k) - F\right).$$

$$B\left(z^k\right) z^{k+1} \equiv \left[A'\left(z^k\right)^T A'\left(z^k\right) + \alpha I\right] z^{k+1} = b,$$

где $b \equiv \left[A'\left(z^k\right)^T A'\left(z^k\right) + \alpha I\right] z^k - \gamma A'\left(z^k\right)^T \left(A'\left(z^k\right) - F\right)$.

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

где z^l – приближенное решение на l-й итерации метода минимальных невязок.
В качестве начального приближения используется $\mathrm{z}^0 \equiv 0$. Критерием останова является условие $\|\mathrm{Bz} - \mathrm{b}\|/\|\mathrm{b}\| \leqslant \varepsilon$ при некотором $\varepsilon > 0$.

Vasin V.V., Eremin I.I. Operators and iterative processes of Fejer type: theory and applications // Berlin, Germany, Walter de Gruyter. Vol. 53. 2009. 155 p.

Сравнение реализаций решения задачи по *T*

Сравним время выполнения реализации решения задачи на основе алгоритма 3 для вычисления MVM в случае нетранспонированных матриц с временем выполнения реализаций на основе кода библиотек Intel MKL, OpenBLAS, и BLIS на квадратных сетках с размером от 96 до 160 и с шагом 4.

Два 18-ядерных Intel Xeon E5-2697 v4 с частотой 2.3 ГГц. S_{L_1} = 32 Кбайт, S_{L_2} = 256 Кбайт, S_{L_3} = 30 Мбайт, $W_{L_{1,2}}$ = 8, W_{L_3} = 20
(Суперкомпьютер “Уран” ИММ УрО РАН).

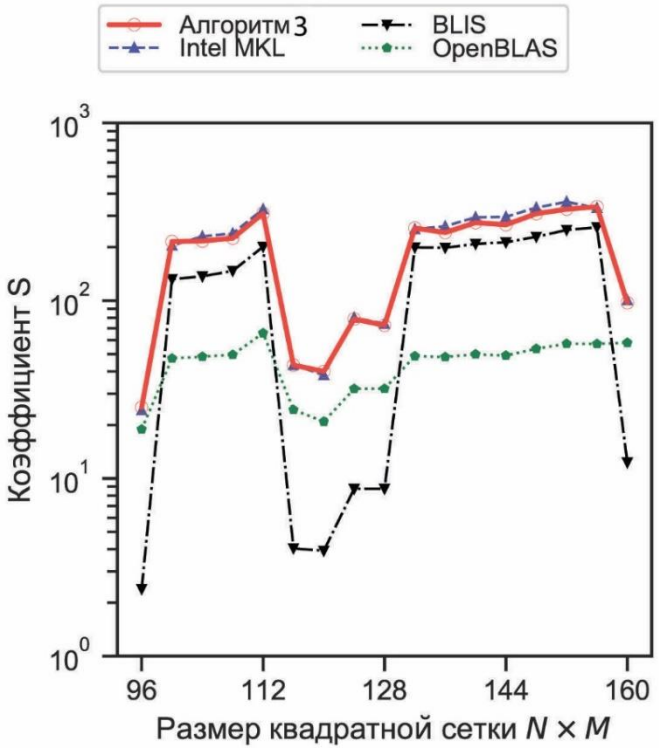
Определим ускорение $S = T_{serial}/T_{parallel}$, где T_{serial} – время выполнения последовательного кода программы с применением стандартных оптимизаций компилятора третьего уровня (O3), где $T_{parallel}$ - время выполнения параллельного кода программы.

Сравнение реализаций решения задачи по *T*

Условие останова: $\|Bz - b\| / \|b\| \leqslant \varepsilon$.

Таблица 1: время решения в минутах для обратной задачи гравиметрии на сетках размерности M × N.

Реализация \ M = N	96	128	160
Алгоритм 3	**0.573**	**1.22**	**3.62**
Intel MKL	0.59	1.19	3.5
OpenBLAS	0.762	2.78	6.068
BLIS	6.029	10.16	28.58
Без оптимизации	14.39	88.76	351.99



На разных сетках разное число итераций, поэтому время счета различно.

Результаты: реализация решения задачи гравиметрии на основе алгоритма 3 сравнима с реализациями на основе кода библиотек Intel MKL, OpenBLAS и BLIS для всех рассмотренных размеров сеток. Результаты алгоритма 3 отличаются на 1% от реализации на основе Intel MKL и превосходят реализации на основе библиотек OpenBLAS и BLIS.

Сравнение реализаций MMM по производительности

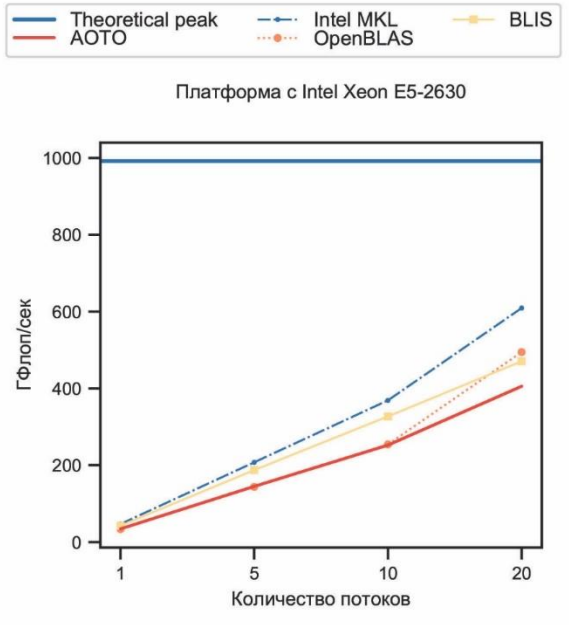
Сравним производительность ПС AOTO с производительностью библиотек, содержащих оптимизированную реализацию MMM (Intel MKL, ARMPL, OpenBLAS, BLIS), и производительностью компиляторов (Clang, GCC, IBM XLC, ICC).

Замечание: Для ICC использовалась -qno-opt-matmul, чтобы предотвратить распознавание и замену MMM на вызов реализации, доступной в Intel MKL.

Пример: Два 10-ядерных Intel Xeon E5-2630 v4 с частотой 2.2 ГГц. S_{L_1} = 32 Кбайт, S_{L_2} = 256 Кбайт, S_{L_3} = 25 Мбайт, $W_{L_{1,2}}$ = 8, W_{L_3} = 20 (ETH Zürich).

Результаты:

- ПС AOTO достигает 1.63-кратного ускорения по сравнению с ICC (без Intel MKL) и 20-кратного ускорения по сравнению с другими компиляторами (Clang, GCC, IBM XLC).
- ПС AOTO достигает 83.33% производительности рассмотренных библиотек, созданных вручную под конкретный процессор (Intel MKL, ARMPL, OpenBLAS, BLIS).



Сравнение реализаций МММ по производительности

Производительность: $P = N_O/T$, где N_O – количество операций с плавающей запятой, вычисляемых программой, T – время выполнения программы в секундах.

Таблица 2: время вычисления в секундах для МММ, плотных квадратных матриц размерности 8000×8000 и разного количества потоков.

Реализация \ Количество потоков	1	5	10	20
AOTO	**29.83**	**7.07**	**4.06**	**2.73**
Intel MKL	21.84	4.93	2.78	1.68
OpenBLAS	30.33	7.13	4.03	2.07
BLIS	24.06	5.46	3.13	2.18

Результат: ПС AOTO сопоставима по производительности скомпилированного кода с кодом созданных вручную библиотек, реализующих МММ.

Сравнение реализаций решения задач о путях

Рассмотрим нахождение путей наименьшей и наибольшей надежности, путей наибольшей вместимости, путей наибольшей стоимости, кратчайших путей. Используем алгоритм последовательного возведения матрицы весов в квадрат.

Пример: Рассматривались плотные квадратные матрицы, содержащие элементы типа double, и четыре потока. Размерность матриц изменялась от 32 до 4000 с шагом 2. 4-ядерный Intel Core i7-3820 с частотой 3.6 ГГц. S_{L_1} = 32 Кбайт, S_{L_2} = 256 Кбайт, S_{L_3} = 10 Мбайт, $W_{L_{1,2}}$ = 8, W_{L_3} = 20. (Процессор УрФУ)

Таблица 3: время выполнения решений задач о путях в секундах.

Задача	Размеры	ICC	AOTO
Нахождение кратчайших путей	1024	0.427	0.429
	2048	3.75	3.59
	4000	28.99	28.19
Нахождение путей наибольшей надежности	1024	0.41	0.31
	2048	3.59	2.51
	4000	28.36	20.17

Результаты: ПС AOTO достигает 1.4-кратного ускорения по сравнению с компилятором ICC и 151-кратное ускорение по сравнению с компиляторами Clang и GCC. Библиотеки, реализующие интерфейс BLAS, не могут быть использованы для оптимизации решения задач о путях.

Сравнение реализаций MVM по производительности

Сравним производительность реализаций алгоритмов 2 и 3 с производительностью библиотек (Intel MKL, OpenBLAS, BLIS), содержащих реализации MVM вида $A^{\mathrm{T}}b$ и $\mathrm{A}b$ для случая транспонированной и не транспонированной матриц.

Пример: Рассматривались плотные квадратные матрицы размерности 25600× 25600 и разное количество потоков**.** Два 18-ядерных Intel Xeon E5-2697 v4 с частотой 2.3 ГГц. S_{L_1} = 32 Кбайт, S_{L_2} = 256 Кбайт, S_{L_3} = 30 Мбайт, $W_{L_{1,2}}$ = 8, W_{L_3} = 20 (Суперкомпьютер “Уран” ИММ УрО РАН).

Таблица 4: время вычисления MVM вида $A^{\mathrm{T}}b$ в сек.

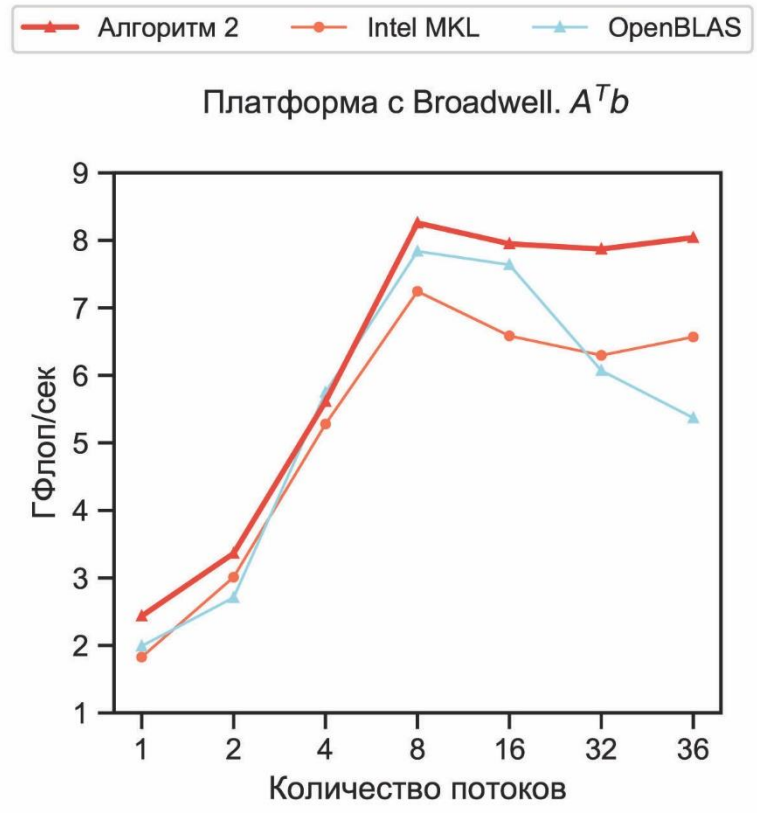
Количество потоков \ Реализация	1	2	4	8	16	32	36
Алгоритм 2	0.54	0.39	0.23	0.158	0.165	0.17	0.163
Intel MKL	0.72	0.44	0.25	0.18	0.199	0.208	0.199
OpenBLAS	0.66	0.48	0.23	0.167	0.171	0.22	0.24
BLIS	0.63	0.74	0.65	0.74	0.73	0.696	0.68

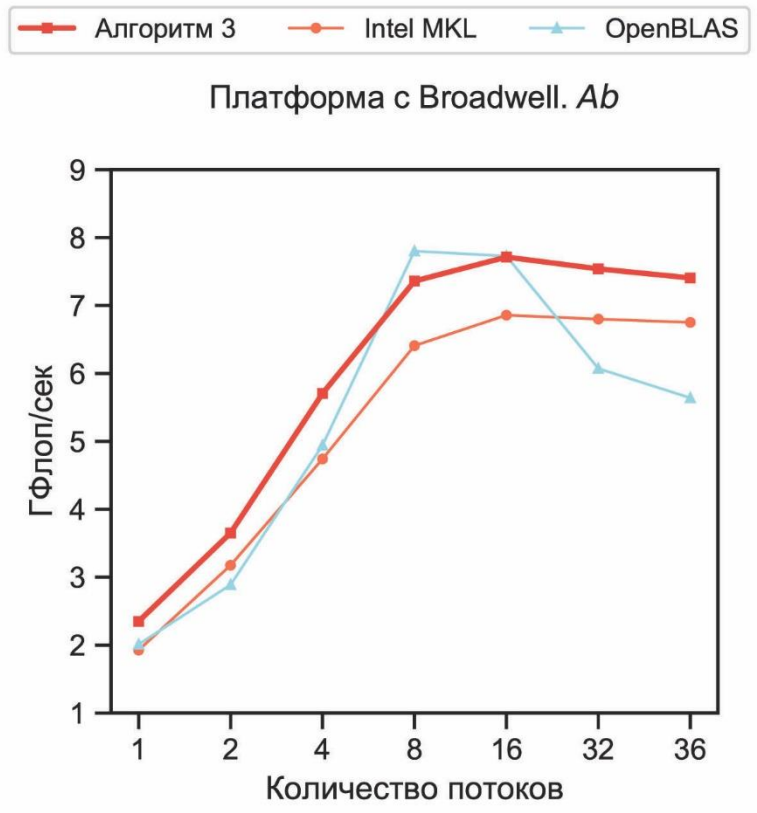
Таблица 5: время вычисления MVM вида $\mathrm{A}b$ в сек.

Количество потоков \ Реализация	1	2	4	8	16	32	36
Алгоритм 3	0.56	0.36	0.23	0.18	0.169	0.173	0.176
Intel MKL	0.68	0.41	0.28	0.2	0.191	0.193	0.194
OpenBLAS	0.65	0.45	0.27	0.168	0.169	0.22	0.23
BLIS	0.64	0.68	0.74	0.8	0.79	0.7	0.695

Результаты: достигается производительность рассмотренных библиотек

Сравнение реализаций MVM по производительности





Результат: ПС АОТО достигает производительности скомпилированного кода созданных вручную библиотек, реализующих MVM.

Сравнение реализаций ТС по производительности

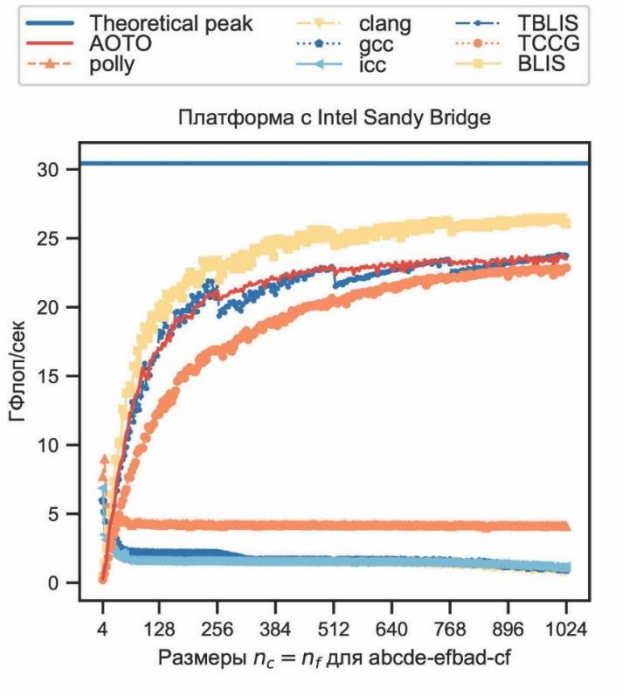
Сравним производительность ПС AOTO с производительностью (TCCG, TBLIS), позволяющих получить оптимизированную реализацию ТС, и с производительностью (Clang, GCC, ICC).

Замечание: Для ICC использовалась -qopt-matmul, чтобы обеспечить распознавание и замену MMM на вызов реализации, доступной в Intel MKL.

Пример: Intel Core i7-3820 с частотой 3.6 ГГц. S_{L_1} = 32 Кбайт, S_{L_2} = 256 Кбайт, S_{L_3} = 10 Мбайт, $W_{L_{1,2}}$ = 8, W_{L_3} = 20. (Процессор УрФУ)

Результаты:

- ПС AOTO достигает 80-кратного ускорения по сравнению с рассмотренными компиляторами (Clang, GCC, ICC).
- ПС AOTO достигает 86.12% производительности рассмотренных фреймворков (TCCG, TBLIS).



Сравнение реализаций TC по производительности

Таблица 6: время вычисления в секундах TC вида $\mathcal{C}_{abcde} = \sum_0^{n_f} \mathcal{A}_{efbad} \cdot \mathcal{B}_{cf} + \mathcal{C}_{abcde}$ где $n_a = n_b = 8$ и $n_d = n_e = 4$.

Реализация \ $n_c = n_f$	256	512	768	1024
AOTO	**0.006**	**0.023**	**0.053**	**0.089**
BLIS	0.0057	0.022	0.048	0.078
TBLIS	0.006	0.024	0.054	0.087
TCCG	0.0079	0.026	0.055	0.09
Polly	0.032	0.13	0.29	0.5
ICC	0.085	0.35	0.82	1.87
GCC	0.063	0.34	0.84	2.001
Clang	0.065	0.343	0.85	2.14

Результат: ПС AOTO сопоставима по производительности скомпилированного кода с кодом фреймворков, реализующих TC (TCCG, TBLIS).

Основные результаты, выносимые на защиту

1. Разработана новая модель гипотетического процессора, которая позволяет сократить время выполнения матрично-векторных операций и их обобщений на замкнутые полукольца с элементами из множества вещественных чисел.
2. Разработаны новые алгоритмы выполнения тензорных операций константной сложности относительно размерности тензоров, уменьшающие время выполнения таких операций. Выведены формулы, позволяющие получить значения параметров алгоритмов выполнения тензорных операций в зависимости от характеристик многоядерных процессоров общего назначения для архитектур x86-64, x86, ppc64le, aarch64.
3. Разработана ПС AOTO для автоматической оптимизации времени выполнения тензорных операций и их автоматического распараллеливания при компиляции программ для многоядерных процессоров общего назначения. Получена оценка производительности многопоточной программы, представленной группой полностью вложенных циклов. Автоматическая оптимизация времени выполнения обобщения матричного произведения внедрена в основной код Polly проекта LLVM.
4. С помощью экспериментов при решении обратной задачи гравиметрии, общей задачи о путях, оптимизации матрично-векторных операций и тензорных сверток подтверждена применимость ПС AOTO для оптимизации времени выполнения тензорных операций. Показано, что ПС AOTO сопоставима по производительности скомпилированного кода с кодом библиотек Intel MKL, OpenBLAS, BLIS, реализующих матричные и матрично-векторные произведения; с фреймворками TCCG и TBLIS, реализующими свертки тензоров; со специализированным компилятором ICC исущественно превосходит компиляторы общего назначения Clang и GCC.

Преимущества ПС АОТО

- Использует формулы, позволяющие автоматически получить значения параметров алгоритмов выполнения тензорных операций в зависимости от характеристик многоядерных процессоров общего назначения.

- Автоматически сокращает время выполнения ТС, не описываемой интерфейсом BLAS. Интерфейс BLAS неприменим для реализации новых методов сокращения времени выполнения ТС, реализованных в ПС АОТО, а также в фреймворках TCCG и TBLIS, поддерживающих малое количество архитектур.

- Автоматически сокращает время выполнения ММА, не описываемого интерфейсом BLAS, с целью сокращения времени выполнения решений общей задачи о путях для замкнутых полуколец с элементами из множества вещественных чисел.

- Применяется для автоматического получения высокопроизводительных реализаций ММТ и MVM для различных архитектур и типов данных, включая не описываемые интерфейсом BLAS целочисленные типы данных.

Задачи о путях

$G = (V, E, w),\ V = \{1,2,\ldots,N\},\ E \subseteq V \times V,\ \ w: E \to S$ над $\{S, \oplus, \otimes, *, 0, 1\}$.

$$w(p) = w(i, k_1) \otimes w(k_1, k_2) \otimes \ldots \otimes w(k_m, j),$$

где $p = \langle i, k_1, k_2, \ldots, k_m, j \rangle$.

Задача APP (Algebraic Path Problem) состоит в нахождении d_{ij} для всех i и j, где d_{ij} — сумма весов всех путей из вершины i в вершину j.

Алгоритм вычисления MMM

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

Goto K., Geijn R.A. van de. Anatomy of High-Performance Matrix Multiplication // ACM Transactions on Mathematical Software. 2008. Vol. 34, no. 3. P. 12:1—12:25. DOI: 10.1145/1356052.1356053.

Модель ГП

- **Архитектура загрузки/сохранения и векторные регистры:** данные должны быть загружены в регистры процессора перед тем, как с ними могут быть выполнены вычисления. N_{REG} – кол-во векторных регистров. N_{VEC} – кол-во значений размерности S_{DATA}, которые может содержать каждый из векторных регистров.
- **Векторные инструкции:** N_{VFMA} – кол-во операций VFMA, вычисляемых процессором за один такт. Отдельная VFMA выполняет N_{VEC} невекторных умножений и сложений, составляющих FMA. L_{VFMA} – минимальное кол-во тактов, которое должно быть совершено перед началом выполнения новой зависимой по данным VMMA.
- **Кэш память**: весь кэш данных – это множественно-ассоциативный кэш с политикой вытеснения последнего по времени использования (англ. *Least Recently Used*, сокр. *LRU*). Каждый уровень кэша L_i характеризуется следующими параметрами: C_{L_i} – размер линии кэша, W_{L_i} – степень ассоциативности, N_{L_i} – кол-во множеств, S_{L_i} – размер уровня кэша L_i, где $S_{L_i} = N_{L_i} C_{L_i} W_{L_i}$.

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.

Блочная модификация алгоритма Флойда–Уоршелла

```
1  begin
2      for k = 0...N step B do
3          K = {k, k+1, ..., k+B-1}
4          Вычисляем A^(k+B)_(K,K), используя стандартный алгоритм Флойда-Уоршелла
5          for i = 0, ..., k-1, k+B, ..., N step B do
6              I = {i, i+1, ..., i+B-1}, A^(k+B)_(I,K) = A^(k)_(I,K) ⊗̄ A^(k+B)_(K,K)
7          end
8          for i = 0, ..., k-1, k+B, ..., N step B do
9              I = {i, i+1, ..., i+B-1}
10             for j = 0, ..., k-1, k+B, ..., Nstep B do
11                 J = {j, j+1, ..., j+B-1}, A^(k+B)_(I,J) = A^(k)_(I,J) ⊕̄ A^(k+B)_(I,K) ⊗̄ A^(k)_(K,J)
12             end
13         end
14         for i = 0, ..., k-1, k+B, ..., N step B do
15             J = {j, j+1, ..., j+B-1}, A^(k+B)_(K,J) = A^(k+B)_(K,K) ⊗̄ A^(k)_(K,J)
16         end
17     end
18 end
```

Akihito T., Sedukhin S. Parallel Blocked Algorithm for Solving the Algebraic Path Problem on a Matrix Processor // Lecture Notes in Computer Science. 2005. Vol. 3726. P. 786–795. DOI: 10.1007/11557654_89.

Рассмотренные случаи ТС

- abcdef-gfbc-dega
- abcdef-gfac-degb
- abcdef-gfab-degc
- abcdef-gebc-dfga
- abcdef-geac-dfgb
- abcdef-geab-dfgc
- abcdef-gdbc-efga
- abcdef-gdac-efgb
- abcdef-gdab-efgc
- abcdef-efgc-gdab
- abcdef-efgb-gdac
- abcdef-efga-gdbc
- abcdef-dfgc-geab
- abcdef-dfgb-geac
- abcdef-dfga-gebc
- abc-acd-db

- abcdef-gfbc-dega
- abcdef-degc-gfab
- abcdef-degb-gfac
- abcdef-dega-gfbc
- abcde-efcad-bf
- abcde-efbad-cf
- abcde-ecbfa-fd
- abcd-ec-abed
- abcd-eb-aecd
- abcd-ebad-ce
- abcd-eafd-fbec
- abcd-eafc-bfde
- abcd-eafb-fdec
- abcd-ea-ebcd
- abcd-deca-be
- abcd-dbea-ec

- abc-dca-bd
- abcd-aefc-fbed
- abcd-aefb-fdce
- abcd-aedf-fbec
- abcd-aedf-bfce
- abcd-aecf-fbed
- abcd-aecf-bfde
- abcd-aebf-fdec
- abcd-aebf-dfce
- abc-bda-dc
- abc-adc-db
- abc-adec-ebd
- ab-cad-dcb
- abc-adc-bd
- abc-ad-bdc
- ab-acd-dbc

Апробация работы

1. 46-ая международная молодежная школа-конференция <<Современные проблемы математики и ее приложений>> (СоПроМат-2015). Екатеринбург, 25-31 января 2015 г.

2. 2nd International Workshop on Radio Electronics & Information Technologies (REIT'2017). Yekaterinburg, November 15, 2017.

3. 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON). Yekaterinburg, October 25-27, 2019.

4. Национальный Суперкомпьютерный Форум (НСКФ-2019). Переславль-Залесский, 26–29 ноября 2019 г.

5. XIV международная конференция <<Параллельные вычислительные технологии>> (ПаВТ'2020). Пермь, 31 марта – 2 апреля 2020 г.

Публикации автора по теме диссертации

Статьи в изданиях из перечня ВАК:

1. 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.
2. Гареев Р.А. Методы оптимизации обобщенных тензорных сверток // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 2. С. 19–39.

Статьи в изданиях, индексируемых в SCOPUS, Web of Science:

3. Gareev R., Grosser T., Kruse M. High-Performance Generalized Tensor Operations: A Compiler-Oriented Approach // ACM Transactions on Architecture and Code Optimization (TACO). 2018. Vol. 15, no. 3. P. 34:1–34:27.
4. Akimova E.N., Gareev R.A. Algorithm of Automatic Parallelization of Generalized Matrix Multiplication // Proceedings of the 2nd International Workshop on Radio Electronics & Information Technologies. (Ekaterinburg, November 15, 2017). CEUR Workshop Proceedings. 2017. Vol. 2005. P. 1–10.

Публикации автора по теме диссертации

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

Статья в издании, индексируемом в РИНЦ:

6. Гареев Р.А. Сравнение средств генерации абстрактного синтаксического дерева из полиэдральной модели в библиотеках CLooG и ISL // Труды 46-й Международной молодежной школы-конференции ”Современные проблемы математики и ее приложений - 2015". (Екатеринбург, Ноябрь, 26–Ноябрь, 29, 2015). Институт математики и механики УрО РАН им. Н.Н. Красовского. 2015. С. 200–202.