Технічні науки

УДК 32.973.3

Чемерис Олександр Анатолійович
доктор технічних наук, старший науковий співробітник,
професор факультету інформатики та обчислювальної техніки
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Чемерис Александр Анатольевич
доктор технических наук, старший научный сотрудник,
профессор факультета информатики и вычислительной техники
Национальный технический университет Украины
«Киевский политехнический институт имени Игоря Сикорского»
Chemerys Oleksandr
D.Sc, Professor of the Faculty of Informatics and Computer Science
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"

Душабаєв Рустам Толкинбайович
студент факультету інформатики та обчислювальної техніки
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Душабаев Рустам Толкынбаевич
студент факультета информатики и вычислительной техники
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Dushabaiev Rustam
Student of the Faculty of Informatics and Computer Science of the
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"

ОПТИМІЗАЦІЯ РОЗМІРУ ТАЙЛУ ПРИ РОЗПАРАЛЕЛЮВАННІ ВКЛАДЕНИХ ЦИКЛІВ ЗА РАХУНОК ВИКОРИСТАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ

ОПТИМИЗАЦИЯ РАЗМЕРА ТАЙЛА ПРИ РАЗПАРАЛЛЕЛИВАНИИ ВЛОЖЕННЫХ ЦИКЛОВ ЗА СЧЕТ ИСПОЛЬЗОВАНИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

OPTIMIZATION OF THE TILE SIZE FOR PARALLELIZING NESTED LOOPS USING THE GENETIC ALGORITHM

Анотація. *Розглядається метод розбиття вкладених циклів на тайли та проблема пошуку оптимального розміру тайлу за допомогою генетичного алгоритму. Пропонується використання програмного пакету PLUTO – інструмент для трансформації вкладених циклів. Даний пакет не запроваджує механізми для пошуку оптимальних розмірів тайлів. В ході розробки системи оптимізації були розглянуті існуючі рішення даної проблеми. Приведено результати роботи системи на. В якості результатів роботи системи були представлені заміри часу виконання тестових програм до оптимізації та після. Програми були взяті на базі перевірочної колекції широко використовуваних алгоритмів різних класів PolyBench.*

Ключові слова: *тайлінг, генетичний алгоритм, PLUTO, PolyBench.*

Аннотация. *Рассматривается метод разбиения вложенных циклов в тайлы и проблема поиска оптимального размера тайла с помощью генетического алгоритма. Предлагается использование программного пакета PLUTO – инструмент для трансформации вложенных циклов. Данный пакет не вводит механизмы поиска оптимальных размеров тайлов. В ходе разработки системы оптимизации были рассмотрены существующие решения данной проблемы. Приведены результаты работы системы. В качестве результатов работы системы были представлены*

замеры времени выполнения тестовых программ до оптимизации и после. Программы были взяты на базе проверочной коллекции широко используемых алгоритмов разных классов PolyBench.

Ключевые слова: *тайлинг, генетический алгоритм, PLUTO, PolyBench*

Summary. *The method of dividing nested cycles into tiles and the problem of finding the optimal tile size using a genetic algorithm are considered. It is proposed to use the software package PLUTO - a tool for transforming nested loops. This package does not introduce mechanisms for finding the optimal tile size. During the development of the optimization system, the existing solutions to this problem were considered. The results of the system on. As a result of the system, measurements of the execution time of test programs before optimization and after were presented. The programs were taken on the basis of a test collection of widely used algorithms of different classes of PolyBench.*

Key words: *tailing, genetic algorithm, PLUTO, PolyBench.*

Вступ. Проблема автоматичного створення ефективних програм користувача є дуже актуальною. Найбільш гостро питання покращення ефективності виконання додатків користувача стоїть для вбудованих та мобільних систем, особливо для тих, що працюють на базі багатоядерних процесорів. На сьогоднішній день такі системи отримали широке розповсюдження завдяки розвитку програмного та апаратного забезпечення. Компілятори програмного забезпечення розвиваються та постійно покращують техніки оптимізації програмного коду, однак існує множина задач, що потребує додаткових оптимізацій. Для таких задач є алгоритмічне рішення, що потребує використання багатовимірних циклів. Одним із методів оптимізації таких частин алгоритму є метод розбиття вкладених циклів на тайли.

Особливістю даного методу являється покращення ефективності виконання програм користувача без необхідності залучення сторонніх бібліотек, а також не використовує паралелізм, тож він може бути застосований для програм, що будуть виконуватись на одноядерних процесорах.

Так як пошук оптимального розміру тайлу є NP повною задачею [1], то буде логічним використати певний нечіткий алгоритм. В даній роботі пропонується використати генетичний алгоритм. Хоча існують і інші погляди щодо вирішення даної задачі.

В [2] автори обраховують розмір тайлу для двох вкладених циклів в компіляторі IBM XL Fortran мінімізуючи функцію, що залежить від рядків кешу, що не перекривають одна одну та сторінок в буфері асоціативної трансляції, до яких звертається розбиті на тайли цикл. Для більших порядків вони використовували ітераційний підхід. Так вони знаходили розміри тайлів для зовнішніх циклів, а потім для двох останніх вирішувалась задача мінімізації.

Пізніше були створені фреймворки багатогранної автоматичні трансформації, такі як Pluto та PPCG, які ефективно вирішують задачу розбиття на тайли. Однак, вони не мають жодної моделі вибору розміру тайлу. Підходи, що були описані в [3; 4; 5] націлені на пошук розміру тайла для подальшого їх використання разом зі згаданими вище фреймворками. Однак, в них налічуються наступні недоліки:

- знаходять однакові розміри тайлів для всіх вимірів
- задають обмеження на простір пошуку розмірів тайлів

Існує ще пласт рішень вибору розміру тайлу для окремих випадків. Наприклад, такі бібліотеки як OpenBLAS, MKL, Eigen [6; 7; 8] покладаються на вручну оптимізовані реалізації, в яких розмір тайлів підлаштований інженерами під різний розмір об'єму оброблюваних даних. Існують спроби автоматизувати дані оптимізації, наприклад, LAPACK [9], PHiPAC [10],

ATLAS [11]. Вони ітеративно підбирають розмір тайла та порядок циклів намагаючись знайти варіант, який буде давати максимальну швидкодію. Всі ці бібліотеки мають високу швидкодію на великих масивах даних, але можуть давати гірші результати для невеликих матриць як показано в роботі [12].

Існують також заточені під конкретні задачі мови програмування та компілятор під конкретні задачі. Так, FLAME [13] розроблений з метою оптимізувати суто обрахунки лінійної алгебри. Нажаль він вимагає від програміста розбивати складні матричні операції на більш прості блоки, які потім замінюються на оптимальну бібліотечну реалізацію. До того ж відповідальність за те, щоб забезпечити оптимальний розмір тайлу та повністю використати потенціал даних бібліотек повністю лягає на програміста.

Оптимізація розміру тайлу

Метод тайлінгу заключається в тому, щоб фрагментувати цикли на блоки меншого розміру. Такий підхід забезпечує краще використання ресурсів обчислювальної техніки, а саме більш оптимальне застосування кеш пам'яті.

Аби краще зрозуміти проблему та суть методу, пропонується переглянути приклад двох вкладених циклів (рис. 1), що проходять по всім елементам масиву без використання будь яких оптимізацій.

```
for (i=1; i<NMAX; i++) {
  for (j=0; j<NMAX; j++) {
    for (k=0; k<i; k++) {
      b[j][k] += a[i][k] * b[j][i];
    }
  }
}
```

Рис. 1. Три вкладених цикли до розбиття на тайли

Джерело: авторська розробка

Результат роботи оптимізації методом тайлінгу (рис. 2) призводить до локалізації доступу до пам'яті, що покращує загальну ефективність виконання фрагменту коду.

```
for (t1=0;t1<=floord(NMAX-1,32);t1++) {
  for (t2=0;t2<=floord(NMAX-1,32);t2++) {
    for (t3=0;t3<=min(floord(NMAX-2,32),t2);t3++) {
      for (t4=32*t1;t4<=min(NMAX-1,32*t1+31);t4++) {
        for (t5=max(32*t2,32*t3+1);t5<=min(NMAX-1,32*t2+31);t5++) {
          for (t6=32*t3;t6<=min(32*t3+31,t5-1);t6++) {
            b[t4][t6] += a[t5][t6] * b[t4][t5];;
          }
        }
      }
    }
  }
}
```

Рис. 2. Фрагмент коду після оптимізації методом тайлингу

Джерело: авторська розробка

Даний фрагмент коду був згенерований засобами пакету PLUTO. PLUTO — це інструмент автоматичного розпаралелювання, заснований на багатогранній моделі. В термінах оптимізацій, що робить компілятор, даний метод запроваджує абстракцію для виконання високорівневих трансформацій, наприклад, оптимізацію вкладених циклів. За допомогою Pluto можна трансформувати код написаний на мові С. Трансформації в основному використовуються для ефективного розбиття циклів на тайли [14].

Для деяких алгоритмів PLUTO пропонує евристично підібрані розміри тайлів. За замовчуванням використовується розмір тайлу рівний 32 для кожного виміру. Форма приймається за прямокутник.

Був заміряний час виконання оригінального фрагменту коду та розбитого на тайли. Для того щоб оцінити відносний приріст в швидкості виконання, час за який виконується оригінальний код був прийнятий за

100%. Тоді відносний час виконання оптимізованого коду обраховується за наступною формулою:

$$T = \frac{T_{tiled}}{T_{orignal}} * 100\%$$

де T – відносний час виконання оптимізованого коду, T_{tiled} – час виконання оптимізованого коду, $T_{original}$ – час виконання оригінального фрагменту коду.

Провівши заміри часу виконання обох фрагментів коду (рис. 3) та порівнявши результати, можемо бачити значне покращення часу.

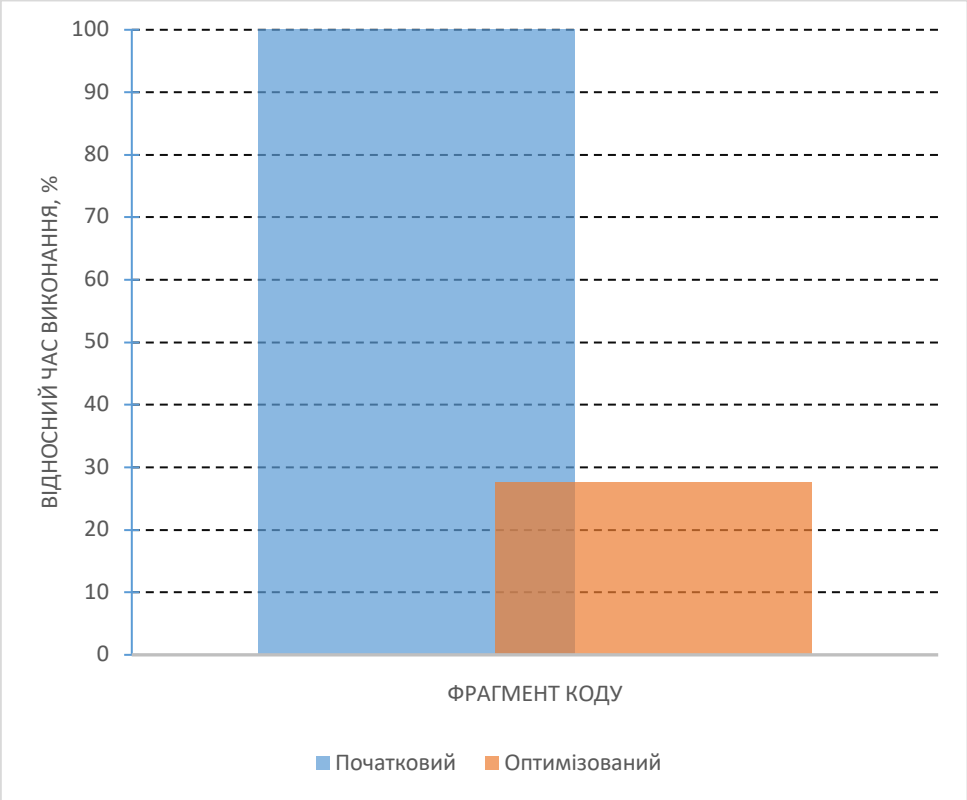

Рис. 3. Результати замірів роботи програм до і після оптимізації

Джерело: авторська розробка

Так як проблема пошуку оптимального розміру тайлу є NP-повною, то було б доречно використати один з еволюційних алгоритмів. В даному випадку був використаний генетичний алгоритм (рис. 4).

Хромосомою буде виступати розмір тайлу. Кодування хромосоми було виконано вектором $size = (a_1\ a_2\ \cdots a_n)$, де $n -$ це кількість вкладених циклів, тобто розмірність.

Схрещування була представлена у вигляді комбінування відповідних компонент хромосоми за наступною формулою [15]:

$$a_i = l_i * \lambda + r_i * (1 - \lambda)$$

де $a_i - i$ компонента вектору розміру тайлу, $l_i, r_i - i$ компонента векторів хромосом, що схрещуються, $\lambda -$ коефіцієнт, що обирається виподковим чином, при чому $\lambda \in [0; 1)$

Мутація міняє місцями дві випадкові компоненти.

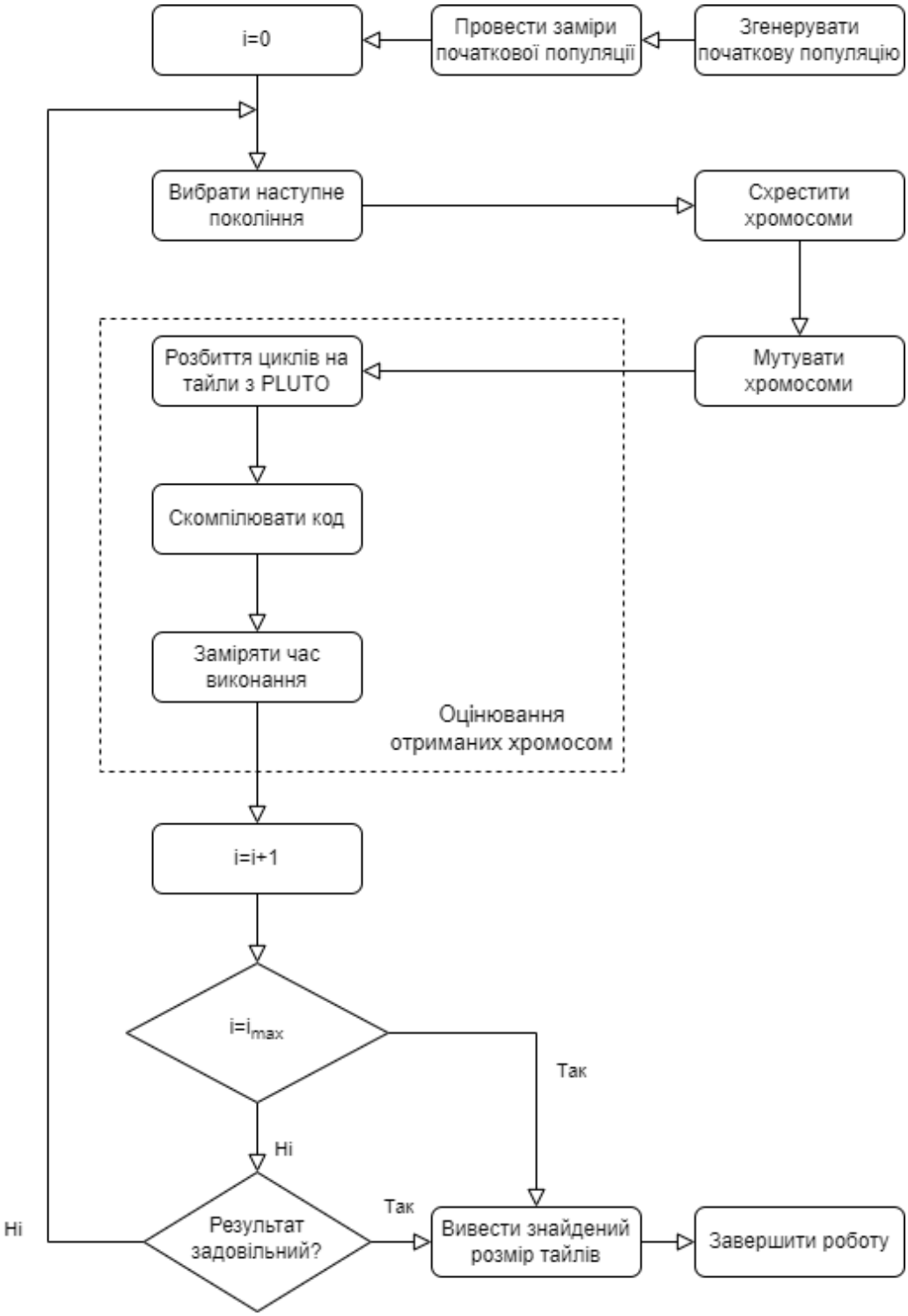

Рис. 4. Генетичний алгоритм

Джерело: авторська розробка

В результаті роботи алгоритму був згенерований остаточний варіант розбиття (рис. 5) із найліпшим знайденим розміром тайлів.

```
for (t1=0;t1<=floord(NMAX-1,237);t1++) {
  for (t2=0;t2<=floord(NMAX-1,134);t2++) {
    for (t3=0;t3<=min(floord(67*t2+66,80),floord(NMAX-2,160));t3++) {
      for (t4=237*t1;t4<=min(NMAX-1,237*t1+236);t4++) {
        for (t5=max(134*t2,160*t3+1);t5<=min(NMAX-1,134*t2+133);t5++) {
          for (t6=160*t3;t6<=min(160*t3+159,t5-1);t6++) {
            b[t4][t6] += a[t5][t6] * b[t4][t5];;
          }
        }
      }
    }
  }
}
```

Рис. 5. Зненерований фрагмент коду з більш оптимальним розміром тайлу

Джерело: авторська розробка

На графіку із замірами часу (рис. 6) видно покращення відносно випадкового розміру.

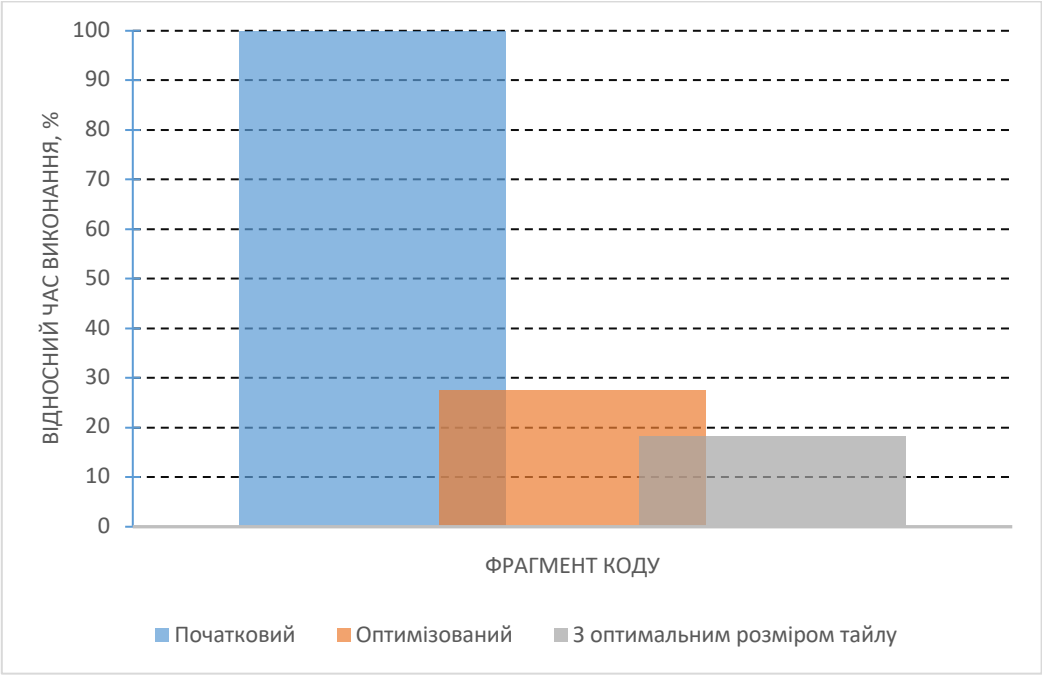

Рис. 6. Порівняння часу виконання оптимізованих фрагментів коду з розміром тайлу за замовчуванням та оптимальнім

Джерело: авторська розробка

Таблиця 1

Результати замірів у відсотковому співвідношенні

Без оптимізацій	Розмір тайлу за замовчуванням	Оптимальний розмір тайлу
100	**27.53**	**18.11**

Джерело: авторська розробка

З огляду на результати, можна дійти висновку, що генетичний алгоритм пошуку дозволив покращити ефективність роботи. Результати експериментів наведені в Таблиці 1.

Виходячи із специфіки проблеми і алгоритму, очевидно, що даний підхід не є вичерпним та можливі подальші оптимізації. Наразі алгоритм не змінює форму тайлу, щоб розширити множину пошуку. Це може збільшити час пошуку, але також може привести до кращих результатів.

Висновки. В даній науковій роботі запропоновано використання генетичного алгоритму для пошуку оптимального розміру тайлу при автоматичній трансформації та розпаралелюванні програм користувача для багатопроцесорних обчислювальних систем. Проведені експерименти, при яких визначався час виконання програми і проводився аналіз швидкодії програм до та після тайлінгу з порівнянням запропонованого методу оптимізованого тайлінгу, показують значний приріст швидкодії виконання програм користувача. Так, з послідовною програмою прискорення складає 5.5 разів, а в порівнянні зі звичайним тайліенгом – 1.5 рази.

Результати отримані в ході дослідження можуть використовуватися надалі у якості бази для подальших модифікацій та покращень методу розбиття на тайли.

Література

1. Boulet P., Dongarra D., Robert Y., Vivien F. Tiling for heterogeneous computing platforms. 1998 [Електронний ресурс].

2. Sarkar V., Megiddo N. An Analytical Model for Loop Tiling and Its Solution. 2000.
3. Sanyam Mehta, Gautham Beeraka, and Pen-Chung Yew. Tile Size Selection Revisited. 2013.
4. Shirako Jun, Sharma Kamal, Fauzia Naznin, Pouchet Louis-Noël, Ramanujam J., P. Sadayappan, Sarkar Vivek. Analytical Bounds for Optimal Tile Size Selection. 2012.
5. Tomofumi Yuki, Lakshminarayanan Renganarayanan, Sanjay Rajopadhye, Charles Anderson, Alexandre E. Eichenberger, and Kevin O'Brien. Automatic Creation of Tile Size Selection Models. 2010.
6. Eigen [n.d.]. A C++ template library for linear algebra. URL: http://eigen.tuxfamily.org/
7. MKL [n.d.]. Intel math kernel library (MKL). URL: http://software.intel.com/en-us/intel-mkl
8. Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. 2013. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs.
9. Anderson E., Bai Z., Dongarra J., Greenbaum A., McKenney A., Croz J. Du, Hammarling S., Demmel J., Bischof C., Sorensen D. LAPACK: A Portable Linear Algebra Library for High-performance Computers. 1990.
10. Bilmes Jeff, Asanovic Krste, Chin Chee-Whye, Demmel Jim. Optimizing Matrix Multiply Using PHiPAC: A Portable, Highperformance, ANSI C Coding Methodology. 2014.
11. Clint Whaley R., Petitet Antoine, Dongarra Jack J. Automated Empirical Optimization of Software and the ATLAS Project. 2000.
12. Shin Jaewook, Hall Mary, Chame Jacqueline, Chen Chun, Hovland Paul D. Autotuning and Specialization: Speeding up Matrix Multiply for Small Matrices with Compiler Technology. 2009.

13. Zee F. G. V., Chan E., Geijn R. A. v. d., Quintana-OrtÃŋ E. S., Quintana-OrtÃŋ G. The libflame Library for Dense Matrix Computations. 2009.
14. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model / Uday Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan. International Conference on Compiler Construction (ETAPS CC), Apr 2008, Budapest, Hungary.
15. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.