ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ТЕЛЕКОММУНИКАЦИИ
INFORMATION TECHNOLOGY AND TELECOMMUNICATIONS

УДК 004. 021, 004.42

DOI: 10.21822/2073-6185-2024-51-3-60-71 Оригинальная статья/Original article

Технологии кеширования данных в современных микропроцессорах

В.А. Егунов, В.А. Шабаловский
Волгоградский государственный технический университет,
400005, г.Волгоград, пр. им.Ленина, 28, Россия

Резюме. Цель. Исследование, представленное в статье, направлено на изучение методов повышения эффективности программного обеспечения в современных вычислительных системах с иерархической структурой памяти. **Метод**. Исследование основано на технологиях кеширования данных в микропроцессорах. **Результат.** Представлены результаты анализа различных подходов к разработке эффективного программного обеспечения с учетом характеристик подсистемы памяти вычислительной системы, что позволило доказать важность кеш-памяти в улучшении производительности и взаимодействии компонентов компьютера. **Вывод.** Кеш-память является критически важным элементом в архитектуре микропроцессоров, играющим ключевую роль в определении производительности вычислительной системы. Оптимизация использования кеша может заметно улучшить время доступа к данным и, как следствие, общую производительность системы. Разработчикам программного обеспечения необходимо уделять особое внимание характеристикам подсистемы памяти при проектировании и реализации решений.

Ключевые слова: кеш-память, оптимизация, эффективность программного обеспечения, кеш-промах, кеш-попадание, ассоциативность кеш-памяти

Для цитирования: В.А. Егунов, В.А. Шабаловский. Технологии кеширования данных в современных микропроцессорах. Вестник Дагестанского государственного технического университета. Технические науки. 2024; 51(3):60-71. DOI:10.21822/2073-6185-2024-51-3-60-71

Data caching technologies in modern microprocessors

V.A. Egunov, V.A. Shabalovsky
Volgograd State Technical University,
28 Lenin Ave., Volgograd 400005, Russia

Abstract. Objective. The study presented in the paper is aimed at studying the methods for improving the efficiency of software in modern computing systems with a hierarchical memory structure. **Method.** The study is based on data caching technologies in microprocessors. **Result.** The article presents the results of the analysis of various approaches to the development of efficient software taking into account the characteristics of the memory subsystem of the computing system, which made it possible to prove the importance of cache memory in improving the performance and interaction of computer components. **Conclusion.** Cache memory is a critical element in the architecture of microprocessors, playing a key role in determining the performance of the computing system. Optimizing the use of cache can significantly improve data access time and, as a result, overall system performance. Software developers need to pay special attention to the characteristics of the memory subsystem when designing and implementing solutions.

Keywords: cache memory, optimization, software efficiency, cache miss, cache hit, associativity of cache memory

For citation: V.A. Egunov, V.A. Shabalovsky. Data caching technologies in modern microprocessors. Herald of Daghestan State Technical University. Technical Sciences. 2024; 51(3):60-71. DOI:10.21822/2073-6185-2024-51-3-60-71

Введение. Кеширование данных в современных микропроцессорах (МП) представляет собой ключевой элемент оптимизации работы вычислительных систем. Эта технология направлена на улучшение производительности путем ускорения доступа к данным, которые МП использует наиболее часто [1, 2]. Практически каждое современное ядро МП, от чипов со сверхнизким энергопотреблением, таких как ARM Cortex-A5 [3], до процессоров Intel Core i9 [4], использует кеш-память. Значимость кеш-памяти в компьютерных системах сложно переоценить. Несмотря на то, что производительность компьютера часто зависит от целого ряда характеристик, таких как частота или объем оперативной памяти, кеш-память играет ключевую роль в определении, насколько эффективно эти компоненты могут взаимодействовать. Кеш-память значительно снижает время доступа МП к основной памяти, сохраняя часто используемые данные и инструкции. Это приводит к ускорению доступа к данным и заметному увеличению общей производительности системы.

Постановка задачи. В статье рассматриваются принципы организации кеширования данных в современных МП, анализируются различные подходы и методы оптимизации, а также влияние этих технологий на общую производительность вычислительных систем. Постоянное совершенствование МП и подсистем памяти, в т.ч. кеш-памяти, требуют поиска новых и использования существующих подходов к разработке эффективного программного обеспечения, в полной мере учитывающего возможности современных вычислительных систем.

Для достижения цели были поставлены следующие задачи:

1. Рассмотреть современные технологии кеширования данных в микропроцессорах.
2. Провести обзор перспективных исследований и разработок в данной области.
3. Проанализировать подходы к разработке эффективного программного обеспечения с учетом характеристик подсистемы памяти вычислительной системы.

Методы исследования. Аналитически обобщим современные технологии кеширования данных в микропроцессорах.

Кеширование – техника, используемая для повышения производительности компьютерных систем за счет хранения копий данных или результатов вычислений во временной области хранения, известной как кеш-память. Это позволяет быстрее получать доступ к часто используемой информации, сокращая время и ресурсы, необходимые для извлечения данных из более медленных слоев хранения [5]. Данная технология, во-первых, повышает скорость доступа к данным, так как извлечение информации из кеша занимает значительно меньше времени по сравнению с основными местами хранения, такими как основная память, жесткие диски или сетевые соединения.

Во-вторых, использование кеша повышает эффективность системы, поскольку снижает нагрузку на основные хранилища и сеть, что в конечном итоге улучшает общую производительность. Наконец, использование кеша также обеспечивает экономическую выгоду, т.к. позволяет оптимизировать использование ресурсов и обрабатывать больше задач с меньшим количеством оборудования, что приводит к снижению затрат.

Кеш-память процессора используется для уменьшения времени доступа к основной памяти, являясь одним из верхних уровней иерархии памяти. Кеш использует небольшую, быструю память (типа SRAM или SDRAM), которая хранит копии часто используемых данных из основной памяти. Если большая часть запросов в память будет обрабатываться кеш-памятью, средняя задержка обращения к памяти будет приближаться к задержкам работы кеша. Самая маленькая единица данных, которая может быть передана между кешем и основной памятью, называется кеш-линия или строка кеша. При этом оптимизация процесса чтения и записи данных осуществляется за счет работы с блоками данных, а не с отдельными байтами [6]. Использование кеш-памяти для повышения эффективности программного обеспечения связано с понятиями пространственной и временной локальности [7]. Пространственная локальность предполагает, что обращение к определенным местам хранения часто сопровождается обобщениями к близлежащим местам, что используется кешами для извлечения и хранения блоков данных в предвидении будущего доступа.

Другими словами, если произошло обращение к данным, расположенным по адресу i, велика вероятность, что следующим обращение будет осуществлено по адресу (i + 1)/ Временная локальность, в свою очередь, указывает на то, что если к определенным данным был доступ недавно, вероятно, к ним снова будет осуществлен доступ в ближайшем будущем.

В настоящее время используются различные технологии, связанные с кешированием данных, применяемые с целью повышения эффективности вычислительных систем. К основным технологиям, используемым в подсистемах памяти современных МП, можно отнести следующие.

Многоуровневая кеш-память. Современные МП часто используют многоуровневую кеш-систему, включающую кеши L1, L2 и L3. В некоторых системах также могут присутствовать кеши более высоких уровней, например L4, но такие архитектуры используются в настоящий момент редко [8]. Основной принцип работы многоуровневого кеша – минимизировать задержку при обменах между МП и памятью, получая при этом больший объем буферной памяти для хранения данных.

Кеш первого уровня самый маленький и быстрый. Его размер небольшой, обычно от 2 до 64 КБ. Кеш L1 делится на две части: одна для хранения инструкций L1i и другая для хранения данных L1d. Это разделение поддерживает различную полосу пропускания выборки, используемую МП, поскольку большинство программ обычно требует больше кеша для данных, чем для инструкций. Кеш L2 больше по размеру и немного медленнее по скорости по сравнению с L1. L2 может совместно использоваться несколькими ядрами или только одним, в зависимости от архитектуры ЦП. Размер L2 в современных МП от 64 КБ до 512 КБ. Кеш L3 больше, чем L1 и L2, но гораздо более медленный, чаще всего разделяется всеми ядрами. Кеш L3 играет важную роль в совместном использовании данных и в организации связи между ядрами МП. Размер L3 составляет от 1 МБ до 62 МБ в зависимости от типа МП. Многоуровневая кеш-память является ключевым компонентом архитектуры современных МП, позволяющим значительно уменьшить задержки при обращении к основной памяти и повысить общую производительность системы.

Ассоциативная организация кеша задает способ отображения строк основной памяти на строки кеш-памяти, который определяет способ хранения и поиска данных в кеше. Данная характеристика кеш-памяти оказывает большое влияние на скорость доступа к данным и эффективность кеширования [10].

Выделяют три основных типа ассоциативности в кеш-памяти:

Кеш-память с прямым отображением (рис.1).

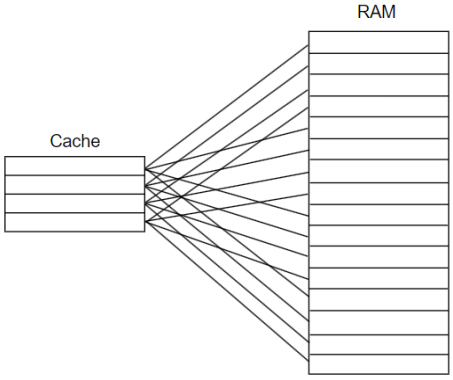

Рис. 1. Кеш-память с прямым отображением
Fig. 1. Direct Mapped Cache

Это самый быстрый подход к отображению памяти, который используется для копирования блока основной памяти в доступную строку кеша. Данный алгоритм назначает блок памяти определенной строке кеша напрямую, без дополнительных промежуточных процессов [11]. Если строка кеша уже занята другим блоком памяти, то предыдущий блок удаляется для загрузки в него нового. При этом адрес ячейки в памяти компьютера разбивается на три составляющих: смещение в строке кеша; номер строки кеша, в которую помещается строка основной памяти, содержащая требуемые данные; тег, используемый кеш-памятью для иден-

тификации строки. По номеру строки в основной памяти кеш-контроллер однозначно определяет сроку кеша, в которую надо поместить загружаемые данные.

Полностью ассоциативная кеш-память (рис. 2). При таком варианте отображения любая строка основной памяти может быть размещена в любой строке кеш-памяти [12]. Данный подход характеризуется высокой эффективностью, гибкостью и более высокой скоростью по сравнению с прямым отображением.

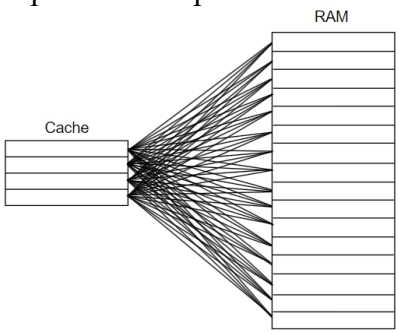

Рис. 2. Полностью ассоциативное отображение
Fig. 2. Fully associative mapping

Кеш-память с множественно – ассоциативным отображением (рис.3). Данный тип кеш-памяти сочетает в себе достоинства двух предыдущих вариантов, представляя собой компромиссный вариант. В данном случае кеш-память делится на множества, при этом отображения на множества осуществляются в «прямом» режиме, размещение внутри множества – полностью ассоциативное.

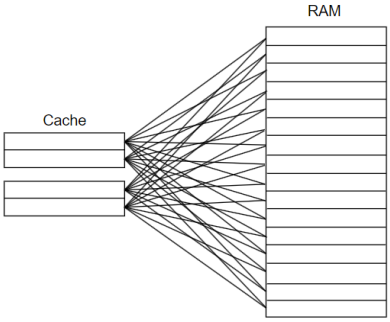

Рис. 3. Кеш-память с множественно – ассоциативным отображением
Fig. 3. Cache memory with multiple-associative mapping

В табл. 1 приведены оценки вероятности промахов кеша в различных конфигурациях. Вероятность промаха классифицируется как «Низкая», «Средняя», «Высокая» на основе общих принципов, поскольку конкретные задачи могут значительно различаться в зависимости от рабочей нагрузки, размера кеша и архитектуры системы.

Табл.1 отображает, как ассоциативность кеша влияет на различные типы и общую вероятность промахов. Обязательные промахи остаются неизменными во всех вариантах, поскольку они зависят от первого доступа к данным.

На вероятность обязательных промахов оказывает влияние размер кеш-линии и предвыборка данных. Промахи по емкости также остаются довольно стабильными, так как они в большей степени зависят от размера кеша, чем от его ассоциативности.

Ассоциативность в данном случае может оказать влияние только в случае использования данных, не представляющих собой непрерывный массив. Конфликтные промахи уменьшаются с увеличением ассоциативности, ведущей к снижению вероятности промахов в кеше.

Таким образом, выбор типа ассоциативности кеша зависит от конкретных требований к производительности и стоимости системы. Баланс между скоростью, стоимостью и сложностью реализации определяет, какой тип ассоциативности будет выбран для конкретной архитектуры. В подавляющем большинстве современных МП используются множественно-ассоциативные кеши.

Таблица 1. Оценка вероятности кеш-промахов
Table 1. Estimation of the probability of cache misses

Тип кеша Cache type	**Обязательные промахи Mandatory misses**	**Промахи по емкости Misses by capacity**	**Конфликтные промахи Conflict misses**	**Оценка общей вероятности промахов Estimation of the overall probability of misses**
Прямое отображение Direct display	Тип кеш-памяти не оказывает влияния Cache type has no effect	Тип кеш-памяти не оказывает влияния Cache type has no effect	Высокая High	Высокая High
2-way Set Associative			Средняя Average	Средне-высокая Medium-high
4-way Set Associative			Низкая Low	Средняя Average
8-way Set Associative			Очень низкая Very low	Низко-средняя Low-medium
Полностью ассоциативный Fully associative			Отсутствует Absent	Низкая Low

Предвыборка данных (cache prefetching). Данная техника повышения производительности заключается в асинхронной загрузке данных из основной памяти в кеш-память заранее, до того, как эти данные будут запрошены МП [13]. Это позволяет снизить задержки при обращении к данным, которые не находятся в кеше, увеличивая общую производительность системы и эффективность программного обеспечения. В большинстве современных МП реализована автоматическая аппаратная предвыборка кода и данных. Предвыборка кода тесно связана с предсказанием ветвлений и выходит за рамки настоящего исследования. Предвыборка данных позволяет МП заранее загружать данные, которые могут понадобиться программе на следующих этапах работы. МП анализирует, по каким адресам и в каком порядке обращается программа и пытается предсказать, какие данные понадобятся в ближайшем будущем, осуществляет их автоматическую предвыборку в кеш-память. Самым простым и наиболее часто используемым методом предсказания здесь является метод загрузки следующей кеш-линии из основной памяти. Данный метод оказывается очень эффективным при последовательном доступе к памяти. При случайном доступе в память предвыборка не оказывает должного эффекта, при этом может даже оказывать отрицательное влияние, засоряя кеш-память ненужными данными. Существует также возможность программной предвыборки данных, для этого в МП существуют специальные команды, которые позволяют загрузить данные в необходимый уровень кеш-памяти. Часто данные команды вставляются в код программы компиляторами в процессе оптимизации кода. Данные команды также могут использоваться непосредственно программистами, однако чаще используются соответствующие интринсики. В частности, интринсик `_mm_prefetch` является частью расширения SIMD (Single Instruction, Multiple Data) от Intel [14]. SIMD-инструкции позволяют выполнять операции над несколькими данными одной командой МП, улучшая тем самым, производительность обработки данных. `_mm_prefetch` предназначена для предвыборки данных в кеш, позволяя программисту указать, какие данные скоро будут нужны, чтобы они могли быть загружены заранее.

Пример использования _mm_prefetch.

```
#include <xmmintrin.h>
void processLargeArray(float* array, int size) {
        for (int i = 0; i < size; i++) {
                // Предвыборка следующего блока данных в кэш
        _mm_prefetch((const char*)&array[i + 16], _MM_HINT_T0);
        }
}
```

В приведенном примере предвыборка используется для загрузки данных из большого массива `float` в кеш перед их обработкой. Аргумент `_MM_HINT_T0` указывает на уровень кеша, в который следует загрузить данные (в данном случае — кеш L1). Следует отметить, что для получения прироста производительности необходимо правильно подби-

рать шаг предвыборки. Однако, при правильном использовании данная методика является достаточно мощным инструментом оптимизации программного обеспечения.

Раздельный кеш данных и инструкций. Во многих МП кеш данных и инструкции разделяются. Это связано с тем, что в процессе обработки данных объем считываемых инструкций и данных может сильно отличаться. Кроме того, подобный подход обладает рядом преимуществ. Так, например, разделенный кеш может иметь вдвое большую пропускную способность по сравнению с объединенным кешем. Использование разделенного кеша оказывает сильное влияние на производительность МП, имеющих конвейер команд, т.к. доступ к инструкциям и данным можно организовать в одном и том же цикле на различных этапах конвейера. Есть и чисто технологические преимущества от такого подхода, которые заключаются в том, что кеш инструкций можно разместить рядом с блоком выборки инструкций, а кеш данных - рядом с блоком памяти. Такой подход может сократить задержки обоих блоков. Кроме того, т.к. объем считываемых инструкций и данных в подавляющем числе случаев сильно отличается, инструкции и данные не будут конфликтовать друг с другом в общем кеше. Стоит отметить, что разделенная технология используется в современных МП для реализации кеша L1. Раздельные кеши данных и инструкций широко используются в современных МП, включая такие архитектуры как x86 и ARM. Производители, такие как Intel, AMD и Apple, в своих процессорах используют этот подход для оптимизации производительности.

Обсуждение результатов. Подходы к разработке эффективного программного обеспечения с учетом характеристик подсистемы памяти вычислительной системы. В настоящее время ведутся исследования и разработки для повышения производительности и масштабируемости в современных вычислительных системах [15].

Рассмотрим основные направления: интеллектуальное кеширование (Smart Caching), 3D-стекирование памяти, оптимизация для специализированных задач, гибридные кеш-системы, уменьшение электропотребления.

Интеллектуальное кеширование использует алгоритмы машинного обучения и искусственного интеллекта для предсказания того, какие данные будут запрошены приложениями в ближайшем будущем, чтобы заранее загрузить эти данные в кеш. Это позволяет сократить время доступа к данным и увеличить общую производительность системы [16]. Примером могут служить алгоритмы, основанные на нейронных сетях. Они могут анализировать образцы доступа к данным и оптимизировать стратегии кеширования на лету. По сути, данная технология представляет собой «умную предвыборку» и может стать существенным фактором в дальнейшем совершенствовании технологии кеширования данных в МП.

3D-стекирование памяти позволяет увеличить плотность хранения данных и пропускную способность, стекируя несколько слоев памяти друг на друга вертикально. Данная технология уменьшает физическое расстояние между процессором и памятью, сокращая задержки и повышая скорость передачи данных.

Примером служит HBM (High Bandwidth Memory), которая используется в графических процессорах и устройствах для высокопроизводительных вычислений, обеспечивая значительно большую пропускную способность по сравнению с традиционной памятью. Стоит отметить, что данная технология разрабатывается многими ведущими IT-компаниями, в частности, Intel и AMD. Гибридные кеш-системы комбинируют различные типы кэш-памяти (например, SRAM, DRAM, Flash) в одной системе, используя их преимущества для повышения эффективности хранения и доступа к данным [17]. Примером таких систем является использование быстрой SRAM для кеширования наиболее часто используемых данных и более медленной, но емкой Flash-памяти для редко используемых данных. Разработка энергоэффективных архитектур кеша включают в себя использование технологий, минимизирующих энергопотребление при доступе к данным и их хранении, например: оптимизированные алгоритмы кеширования, минимизирующие число промахов кеша, что снижает количество энергозатратных операций доступа

к основной памяти; применение низкоэнергетических технологий памяти, например, LPDDR для кеша низких уровней. Примером служит технология Intel's Speed Shift, которая позволяет процессору более динамично управлять частотой и напряжением, оптимизируя энергопотребление в зависимости от текущей нагрузки. Оптимизация программного обеспечения с учётом характеристик подсистемы памяти вычислительной системы является ключевым аспектом достижения максимальной производительности. Необходимо руководствоваться определенными принципами программирования и оптимизации, чтобы увеличить эффективность программного обеспечения. Вот некоторые из них:

1. Локальность данных. Кеш-память работает эффективнее, кода программа проявляет высокую степень локальности данных. Существуют два типа локальности. Временная локальность предполагает, что после того, как программа получила доступ к определенному фрагменту данных, вероятно, она будет обращаться к нему снова в ближайшем будущем. Чтобы использовать временную локальность, достаточно повторно использовать данные, к которым осуществлялось обращение, и которые были помещены в кеш-память [18]. Пространственная локальность предполагает организацию доступа к данным, которые расположены рядом с данными, к которым недавно также осуществлялся доступ. Чтобы воспользоваться свойствами пространственной локальности, необходимо обеспечить доступ к данным, размещенным в памяти последовательно [18]. Кроме того, из принципа пространственной локальности вытекает необходимость обработки данных в той последовательности, в которой они расположены в памяти. Например, C хранит массивы построчно, в то время как Fortran хранит в порядке столбцов. Примеры на языке C приведены ниже.

Пространственная локальность

```
for (int i = 0; i < rows; ++i) {
      for (int j = 0; j < cols; ++j) {
      // Обработка элемента matrix[i][j]
      }
  }
```

Временная локальность

```
int sumArray(int* array, int size) {
   int sum = 0;
//Повторное использование //переменной sum
for (int i = 0; i < size; ++i) {
       sum += array[i];
   }
return sum;
}
```

Оптимизация доступа к памяти достигается за счет приведенных выше методов, хотя стоит отметить, что использование свойств временной локальности оказывается гораздо более выгодным в случае обращения к блокам данных, а не к отдельным переменным. Например, вместо многократного обращения к отдельной переменой, стоит стремиться к многократному использованию блока данных, ранее уже помещенного в кеш-память. В частности, изменение порядка обработки строк и столбцов матрицы может улучшить использование кеш-памяти. Ниже приведен классический код умножения матрицы на языке C, включающий в себя три цикла: цикл по строкам, цикл по столбцам и цикл вычисления скалярного произведения.

```
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
   for (int k = 0; k < n; ++k) {
result[i][j] += a[i][k] * b[k][j];
    }
  }
 }
}
```

Данную реализацию назовем ijk по порядку следования циклов. Однако, циклы можно менять местами, серьезно изменяя итоговую производительность. Результат исследования таких перестановок приведен в табл. 2. Замеры времени осуществлялись на одной и той же вычислительной системе для матриц одинаковой размерности.

Из табл. 2 видно, что циклы с порядками ikj и kij являются наиболее быстрыми. Такая скорость достигается за счёт того, что при указанном порядке выполнения циклов обращение к данным происходит в пределах одной в кеш-линии, что обеспечивает кеш-

попадания. Теоретически, порядок ikj должен быть самым быстрым вариантом за счёт того, что при таком порядке обращение к разным строкам матриц происходит реже всего, что увеличивает кэш-попадания. Однако порядок kij показывает аналогичный результат.

Таблица 2. Исследование программы умножения матриц
Table 2. Study of the matrix multiplication program

Вид цикла Cycle type	Время, с Time, s
ijk	1.1693
ikj	0.4675
jik	1.11
jki	2.7319
kij	0.4777
kji	2.7470

2. Блочная реализация алгоритмов. Размеры блоков подбираются таким образом, чтобы целиком помещаться в кеш-память определенного уровня, чаще всего оптимизация осуществляется под L1. В данном случае в соответствии с принципами пространственной и временной локальности количество кеш-промахов минимизируется. Пример блочного разделения матриц приведен на рис. 4, там же приводится алгоритм блочного умножения матриц.

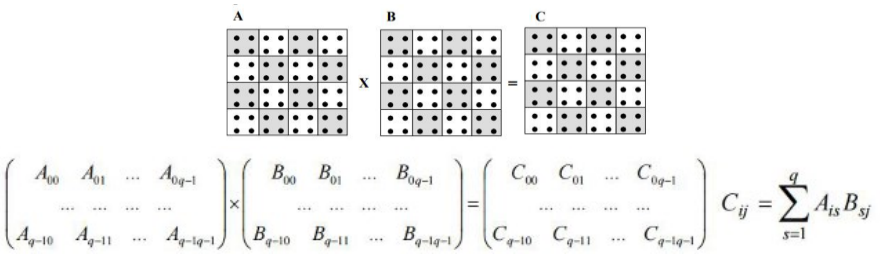

Рис. 4. Блочный алгоритм умножения матриц
Fig. 4. Block matrix multiplication algorithm

Размеры блоков могут варьироваться в зависимости от типа МП в соответствии с размером кеша L1. Классическим примером в этой области является блочное умножение матриц, при котором исходные матрицы делится на блоки меньшего размера [19].

```
void multiplyMatricesBlock(int** a, int** b, int** result, int n, int blockSize) {
  for (int i = 0; i < n; i += blockSize) {
    for (int j = 0; j < n; j += blockSize) {
      for (int k = 0; k < n; k += blockSize) {
        for (int ii = i; ii < i + blockSize; ii++) {
          for (int kk = k; kk < k + blockSize; kk++) {
            for (int jj = j; jj < j + blockSize; jj++) {
              result[ii][jj] += a[ii][kk] * b[kk][jj];
            }
          }
        }
      }
    }
  }
}
```

При оптимальных размерах блоков эффективность программы значительно возрастает.

3. Оптимизация под длину строки кеша. Необходимо учитывать длину строки кеш-памяти при проектировании структур данных. Каждый МП имеет фиксированную длину строки кеша, которую несложно узнать, в т.ч. и программным путем.

Для уменьшения числа загрузок кеш-строк и снижения вероятности кеш-промахов, лучше выравнивать данные по границам кеш-строк.

Данной цели может помочь добиться выделение памяти под массивы данных размерами, кратными длине кеш-строк, а также выравнивание статически определяемых и динамически выделяемых массивов данных по границам строк (например, функция _mm_malloc()).

```
#define CACHE_LINE_SIZE 64
typedef struct {
// Размер структуры кратен размеру строки кеша
char data[CACHE_LINE_SIZE];
} CacheLineData;
void processCacheAlignedData(CacheLineData* data) {
// Обработка данных, выровненных по строке кеша
}
```

4. Компилятор и флаги компилятора. Современные компиляторы могут оптимизировать код с целью повышения эффективности использования кеш-памяти.

Для использования этой опции необходимо использовать соответствующие флаги компилятора, такие как -O2 или -O3 в GCC. Кроме того, некоторые компиляторы имеют возможность оптимизации кода под целевую архитектуру вычислительной системы, например, Intel C++ Compiler. Примером использования флагов могут служить флаги компиляции для GCC [21].

```
gcc -O2 myprogram.c -o myprogram
```

5. Профилирование и бенчмаркинг. Для изучения взаимодействия программы с кеш-памятью можно воспользоваться инструментами профилирования. Программы, такие как perf в Linux, встроенные анализаторы производительности в средах разработки (IDE) или специализированные приложения, такие как Intel VTune Amplifier, помогут выявить потенциальные проблемы, связанные с кеш-памятью. Использование инструмента «perf» [22] для анализа кеш-промахов представлен ниже.

```
perf stat -e cache-misses,cache-references ./myprogram
```

6. Привязка потоков. При работе с многопоточными программами используются методы привязки потоков к конкретным ядрам МП. Данная техника позволяет уменьшить вероятность конфликтов в кеше между потоками. В приведенном примере происходит привязка потока к конкретному ядру МП с использованием pthreads.

```
int main() {
pthread_t thread;
cpu_set_t cpuset;
pthread_create(&thread, NULL, threadFunction, NULL);
CPU_ZERO(&cpuset);
CPU_SET(0, &cpuset);
pthread_setaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
pthread_join(thread, NULL);
return 0;
}
```

7. Кеш-оптимизация отдельных задач. Задача кеш-оптимизации привлекает широкое внимание исследователей в т.ч., потому, что данная методика позволяет существенно повысить эффективность получаемого программного обеспечения. Существует большое число публикаций, посвященных оптимизации программных реализаций конкретных вычислительных задач. В качестве примера можно привести многочисленные исследования в области оптимизаций процедур BLAS (Basic Linear Algebra Subprograms) и LAPACK(Linear Algebra PACKage). Авторы также публиковали результаты своих исследований в этой области [23, 24], а также в области улучшения стратегии кеширования данных для вычислительных систем с иерархической структурой памяти [25, 26].

Вывод. Полученные результаты наглядно демонстрируют необходимость кеш-оптимизации, а также возможность получения значительного прироста производительности программного обеспечения.

Результаты анализа подходов к разработке эффективного программного обеспечения с учетом характеристик подсистемы памяти вычислительной системы выявил важность оптимизации программ для улучшения взаимодействия с кеш-памятью.

В целом можно сказать, что внимательное отношение к характеристикам кеш-памяти и осознанное использование доступных инструментов и методик оптимизации

позволяет значительно улучшить производительность программного обеспечения, уменьшить время выполнения задач и повысить общую эффективность вычислительных систем.

Анализ различных подходов к разработке эффективного программного обеспечения с учетом характеристик подсистемы памяти вычислительной системы позволил выявить важность кеш-памяти в улучшении производительности и взаимодействии компонентов компьютера.

Разработка различных методов оптимизации использования кеш-памяти, включая локальность данных, оптимальное кеширование структур данных, учет длины строки кеша, использование флагов компилятора для оптимизации, профилирование и бенчмаркинг, а также привязку потоков к конкретным ядрам МП, открыло новые возможности для разработчиков программного обеспечения в направлении повышения эффективности их продуктов. Кеш-память является критически важным элементом в архитектуре микропроцессоров, играющим ключевую роль в определении производительности вычислительной системы.

Оптимизация использования кеша может заметно улучшить время доступа к данным и, как следствие, общую производительность системы. Разработчикам программного обеспечения необходимо уделять особое внимание характеристикам подсистемы памяти при проектировании и реализации своих решений, используя современные подходы и технологии кеширования для достижения наилучших результатов.

Библиографический список:

1. Bhat, Subrahmanya and Bhat, Subrahmanya and Kamath, K. R, Cache Hierarchy in Modern Processors and Its Impact on Computing (May 11, 2017). International Journal of Management, IT and Engineering (IJMIE), Volume 5, Issue 7, pp. 248-253, ISSN: 2249-0558, July 2015, Proceedings of National Conference “Recent Advances in IT, Management and Social Sciences”, Manegma – 2015, Mangalore on 23rd April, 2015, ISBN No. 978-81-929306-6-4, Available at SSRN: https://ssrn.com/abstract=2966616
2. Alexander von Bülow, Jürgen Stohr, and Georg Färber Towards an Efficient Use of Caches in State of the Art Processors for Real-Time Systems // Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems. — Catania, Italy:Steve Goddard, 2004. — С. 5-9.
3. Cortex-A5 // developer.arm URL: https://developer.arm.com/Processors/Cortex-A5
4. Processeurs Intel® Core™ de 14e génération pour PC de bureau // intel.fr URL: https://www.intel.fr /content/www/fr/fr/products/docs/processors/core/core-14th-gen-desktop-brief.html
5. Антонов А.А., Ключев А.О., Комар М.С., Кустарев П.В., Кучерявый Е.А., Молчанов Д.А., Петров В.И., Платунов А.Е. Разработка протокола множественного доступа для процессоров с многоуровневым кэшированием // Научно-технический вестник информационных технологий, механики и оптики. 2015. №3. URL: https://cyberleninka.ru/article/n/razrabotka-protokola-mnozhestvennogo-dostupa-dlya-protsessorov-s-mnogourovnevym-keshirovaniem.
6. Measuring the size of the cache line empirically // lemire URL: https://lemire.me/blog/ 2023/12/12/measuring-the-size-of-the-cache-line-empirically/
7. Bruce Jacob, Spencer W. Ng, David T. Wang, CHAPTER 1 - An Overview of Cache Principles, Editor(s): Memory Systems,Morgan Kaufmann, 2008, Pages 57-77, ISBN 9780123797513.
8. Eze, Val & Eze, Martin & Edozie, Enerst & Eze, Esther. (2023). Design and Development of Effective Multi-Level Cache Memory Model. International Journal of Recent Technology and Applied Science (IJORTAS). 5. 54-64. 10.36079/lamintang.ijortas-0502.515.
9. IBM's New System Z CPU Offers 40 Percent More Performance per Socket, Integrated AI // extremetech URL: https://www.extremetech.com/computing/326402-ibms-new-system-z-cpu-offers-40-percent-more-performance-per-socket-integrated-ai
10. Cache Memory in Computer Organization // geeksforgeeks URL: https://www.geeksforgeeks.org/cache-memory-in-computer-organization/.
11. Jouppi, Norman. (1998). Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache Prefetch Buffers.. Conference Proceedings - Annual Symposium on Computer Architecture. 18. 388-397. 10.1109/ISCA.1990.134547.
12. Garzón, Esteban & Hanhan, Robert & Lanuzza, Marco & Teman, Adam & Yavits, Leonid. (2024). FASTA: Revisiting Fully Associative Memories in Computer Microarchitecture. IEEE Access. PP. 10.1109/ACCESS.2024.3355961.
13. Guocong Quan, Atilla Eryilmaz, Jian Tan, Ness Shroff,Prefetching and caching for minimizing service costs: Optimal and approximation strategies, Performance Evaluation, 2021;145:102149, ISSN 0166-5316,

14. Function core::arch::x86_64::_mm_prefetch // doc.rust-lang URL: https://doc.rust-lang.org/beta/core/arch/x86_64/fn._mm_prefetch.html.
15. Филисов Д.А. Стратегии оптимизации для высоконагруженныхприложений: повышение общей производительности // Вестник науки. 2023. №7 https://cyberleninka.ru/article/n/strategii-optimizatsii-dlya-vysokonagruzhennyh-prilozheniy-povyshenie-obschey-proizvoditelnosti.
16. Wu, HT., Cho, HH., Wang, SJ. *et al.* Intelligent data cache based on content popularity and user location for Content Centric Networks. *Hum. Cent. Comput. Inf. Sci.* 2019;(9)44. https://doi.org/10.1186/s13673-019-0206-5.
17. Аль-згуль Мосаб Басам Гибридные алгоритмы в системах кэширования объектов // Advanced Engineering Research (Rostov-on-Don). 2008. №4-39. URL: https://cyberleninka.ru/article/n/gibridnye-algoritmy-v-sistemah-keshirovaniya-obektov.
18. Locality of Reference and Cache Operation in Cache Memory // turbopages URL: https://www.geeksforgeeks.org/ locality-of-reference-and-cache-operation-in-cache-memory/.
19. Юрушкин М. В., Семионов С. Г. Переразмещение матриц к блочному виду с минимизацией использования дополнительной памяти // Известия вузов. Северо-Кавказский регион. Серия: Технические науки. 2017. №3 (195). URL: https://cyberleninka.ru/article/n/pererazmeschenie-matrits-k-blochnomu-vidu-s-minimizatsiey-ispolzovaniya-dopolnitelnoy-pamyati.
20. LRU Cache — A Cache Data Structure // medium URL: https://ogroetz.medium.com/lru-cache-a-cache-data-structure-1fab0d948e94.
21. GCC, the GNU Compiler Collection // gcc.gnu URL: https://gcc.gnu.org/.
22. Chapter 25. Profiling memory accesses with perf mem // access.redhat URL: https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/8/html/monitoring_and_managing_system_status_and_performance/
23. Егунов В.А. Кэш-оптимизация процесса вычисления собственных значений на параллельных вычислительных системах// Прикаспийский журнал: управление и высокие технологии. - 2019. - № 1 (45). - С. 154-163.
24. Егунов В.А. О влиянии кэш-памяти на эффективность программной реализации базовых операций линейной алгебры // Прикаспийский журнал: управление и высокие технологии. - 2018. - № 3. - С. 88-96.
25. Егунов, В.А. Метод улучшения стратегии кеширования для вычислительных систем с общей памятью / В.А. Егунов, А.Г. Кравец // Программная инженерия. - 2023. - Т. 14, № 7. - С. 329-338. - DOI: 10.17587/prin.14.329-338.
26. Kravec A.G., Egunov V.A. The Software Cache Optimization-Based Method for Decreasing Energy Consumption of Computational Clusters// Energies. 2022;15(20):16 (October-2) [Special issue «Smart Energy and Sustainable Environment»]. Article 7509. 16p. DOI: https://doi.org/10.3390/en15207509.

References:

1. Bhat, Subrahmanya and Bhat, Subrahmanya and Kamath, K. R, Cache Hierarchy in Modern Processors and Its Impact on Computing (May 11, 2017). International Journal of Management, IT and Engineering (IJMIE), Volume 5, Issue 7, Pp. 248-253, ISSN: 2249-0558, July 2015, Proceedings of National Conference “Recent Advances in IT, Management and Social Sciences”, Manegma – 2015, Mangalore on 23rd April, 2015, ISBN No. 978-81-929306-6-4, Available at SSRN: https://ssrn.com/abstract=2966616
2. Alexander von Bülow, Jürgen Stohr, and Georg Färber Towards an Efficient Use of Caches in State of the Art Processors for Real-Time Systems. Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems. Catania, Italy:Steve Goddard, 2004;5-9.
3. Cortex-A5. developer.arm URL: https://developer.arm.com/Processors/Cortex-A5
4. Processeurs Intel® Core™ de 14e génération pour PC de bureau // intel.fr URL: https://www.intel.fr/content/www/fr/fr/products/docs/processors/core/core-14th-gen-desktop-brief.html
5. Antonov A.A., Klyuchev A.O., Komar M.S., Kustarev P.V., Kucheryavyj E.A., Molchanov D.A., Petrov V.I., Platunov A.E. Development of a multiple access protocol for processors with multi-level caching // Scientific and Technical Bulletin of Information Technologies, Mechanics and Optics. 2015;3. URL: https://cyberleninka.ru/article/n/razrabotka-protokola-mnozhestvennogo-dostupa-dlya-protsessorov-s-mnogourovnevym-keshirovaniem. (In Russ)
6. Measuring the size of the cache line empirically // lemire URL: https://lemire.me/blog/2023/12/12/measuring-the-size-of-the-cache-line-empirically/
7. Bruce Jacob, Spencer W. Ng, David T. Wang, CHAPTER 1 - An Overview of Cache Principles, Editor(s): Memory Systems,Morgan Kaufmann, 2008, Pages 57-77, ISBN 9780123797513.
8. Eze, Val & Eze, Martin & Edozie, Enerst & Eze, Esther. (2023). Design and Development of Effective Multi-Level Cache Memory Model. International Journal of Recent Technology and Applied Science (IJORTAS). 5. 54-64. 10.36079/lamintang.ijortas-0502.515.
9. IBM's New System Z CPU Offers 40 Percent More Performance per Socket, Integrated AI // extremetech URL: https://www.extremetech.com/computing/326402-ibms-new-system-z-cpu-offers-40-percent-more-performance-per-socket-integrated-ai

10. Cache Memory in Computer Organization // geeksforgeeks URL: https://www.geeksforgeeks.org/cache-memory-in-computer-organization/.
11. Jouppi, Norman. (1998). Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache Prefetch Buffers.. Conference Proceedings - Annual Symposium on Computer Architecture. 18. 388-397. 10.1109/ISCA.1990.134547.
12. Garzón, Esteban & Hanhan, Robert & Lanuzza, Marco & Teman, Adam & Yavits, Leonid. (2024). FASTA: Revisiting Fully Associative Memories in Computer Microarchitecture. IEEE Access. PP. 10.1109/ACCESS.2024.3355961.
13. Guocong Quan, Atilla Eryilmaz, Jian Tan, Ness Shroff. Prefetching and caching for minimizing service costs: Optimal and approximation strategies, Performance Evaluation, 2021;145:102149, ISSN 0166-5316
14. Function core::arch::x86_64::_mm_prefetch //doc.rust-lang URL: https://doc.rust-lang.org/beta/core/arch/x86_64/fn._mm_prefetch.html.
15. Filisov D.A. Optimization strategies for high-load applications: improving overall performance // Bulletin of Science. 2023. №7 https://cyberleninka.ru/article/n/strategii-optimizatsii-dlya-vysokonagruzhennyh-prilozheniy-povyshenie-obschey-proizvoditelnosti. (In Russ)
16. Wu, HT., Cho, HH., Wang, SJ. *et al.* Intelligent data cache based on content popularity and user location for Content Centric Networks. *Hum. Cent. Comput. Inf. Sci.* **9**, 44 (2019). https://doi.org/10.1186/s13673-019-0206-5.
17. Al'-zgul' Mosab Basam. Hybrid algorithms in object caching systems. *Advanced Engineering Research* (Rostov-on-Don). 2008. №4-39. URL: https://cyberleninka.ru/article/n/gibridnye-algoritmy-v-sistemah-keshirovaniya-obektov. (In Russ)
18. Locality of Reference and Cache Operation in Cache Memory//turbopages URL: https://www.geeksforgeeks.org/locality-of-reference-and-cache-operation-in-cache-memory/.
19. YUrushkin M. V., Semionov S. G. Repositioning matrices to a block view while minimizing the use of additional memory . *News of universities. The North Caucasus region. Series: Technical Sciences*. 2017. №3 (195). URL: https://cyberleninka.ru/article/n/pererazmeschenie-matrits-k-blochnomu-vidu-s-minimizatsiey-ispolzovaniya-dopolnitelnoy-pamyati. (In Russ)
20. LRU Cache — A Cache Data Structure // medium URL: https://ogroetz.medium.com/lru-cache-a-cache-data-structure-1fab0d948e94.
21. GCC, the GNU Compiler Collection // gcc.gnu URL: https://gcc.gnu.org/.
22. Chapter 25. Profiling memory accesses with perf mem//access.redhat URL: https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/8/html/monitoring_and_managing_system_status_and_performance/
23. Egunov V.A. Cache optimization of the process of calculating eigenvalues on parallel computing systems. *Caspian Journal: Management and High Technologies*. 2019;1 (45):154-163. (In Russ)
24. Egunov V.A. On the effect of cache memory on the effectiveness of software implementation of basic linear algebra operations. *Caspian Journal: Management and High Technologies*. 2018;3: 88-96.
25. Egunov V.A., Kravec A.G. A method for improving the caching strategy for computing systems with shared memory. *Software Engineering.* 2023; 14(7):329-338. - DOI: 10.17587/prin.14.329-338. (In Russ)
26. Kravec A.G., Egunov V.A. The Software Cache Optimization-Based Method for Decreasing Energy Consumption of Computational Clusters. *Energies.* 2022;15(20):16 (October-2) [*Special issue «Smart Energy and Sustainable Environment»].* Article 7509. DOI: https://doi.org/10.3390/en15207509. (In Russ)

Сведения об авторах:

Егунов Виталий Алексеевич, кандидат технических наук, доцент, кафедра ЭВМ и систем; vegunov@mail.ru

Шабаловский Владимир Андреевич, магистрант, кафедра ЭВМ и систем; shabalovsky.v@mail.ru

Information about authors:

Vitaly A. Egunov**,** Cand. Sci. (Eng), Assoc.Prof., Computers and Systems Department; vegunov@mail.ru

Vladimir A. Shabalovsky, Master Student, Computers and Systems Department; shabalovsky.v@mail.ru

Конфликт интересов/Conflict of interest.

Авторы заявляют об отсутствии конфликта интересов/The authors declare no conflict of interest.

Поступила в редакцию/Received 05.03.2024.

Одобрена после рецензирования/ Reviced 09.04.2024.

Принята в печать/Accepted for publication 09.04.2024.