Велибеков Нариман Сабирович,
студент, ФГБОУ ВО «МТУСИ»
Velibekov Nariman Sabirovich,
student, MTUCI

ВЛИЯНИЕ ЛОКАЛЬНОСТИ ДАННЫХ И КЭШ-ПРОМАХОВ ПРОЦЕССОРА НА СКОРОСТЬ ВЫПОЛНЕНИЯ АЛГОРИТМОВ ОБРАБОТКИ СТРОК/МАССИВОВ
IMPACT OF DATA LOCALITY AND CPU CACHE MISSES ON THE PERFOMANCE OF ARRAY AND STRING PROCCESING ALGORITHMS

Аннотация. Работа посвящена исследованию влияния пространственной и временной локальности данных на скорость выполнения алгоритмов обработки массивов и строк. В условиях растущего разрыва между производительностью процессора и задержками оперативной памяти эффективная работа с кэш-памятью становится ключевым фактором производительности ПО. В статье проводится сравнительный анализ и профилирование алгоритмов с различными паттернами обращения к памяти. Результаты демонстрируют, как снижение количества кэш-промахов позволяет существенно повысить быстродействие программ без изменения их асимптотической сложности. Практическая ценность работы заключается в формировании базовых рекомендаций по разработке cache-friendly кода.

Abstract. This paper investigates the impact of spatial and temporal data locality on the execution speed of array and string processing algorithms. As the performance gap between CPU processing power and main memory latency continues to widen, efficient cache utilization becomes a critical factor in software performance. The study presents a comparative analysis and profiling of algorithms exhibiting various memory access patterns. The findings demonstrate how minimizing CPU cache misses significantly increases execution speed without altering asymptotic time complexity. The practical contribution lies in formulating key guidelines for designing cache-friendly code.

Ключевые слова: Локальность данных, кэш-промахи, задержка памяти, паттерны доступа к памяти, cache-friendly алгоритмы, профилирование производительности.

Keywords: Data locality, CPU cache misses, memory latency, memory access patterns, cache-friendly algorithms, performance profiling.

Классический теоретический анализ алгоритмов опирается на оценку временной сложности в формализме О-нотации, предполагая, что каждая операция с памятью занимает одинаковое время. Однако реальная вычислительная практика демонстрирует существенный разрыв между теоретической сложностью и фактическим временем выполнения программ [1].

Ключевым механизмом предотвращения таких простоев выступает эффективная утилизация иерархической кэш-памяти (L1, L2, L3). Логика работы hardware-предвыборщиков строится на предположении, что программа обладает пространственной локальностью – то есть после обращения к некоторому адресу в памяти с высокой вероятностью будут затребованы соседние ячейки. В случае линейных структур данных, таких как массивы и строки, программный код сам формирует характер загрузки кэш-линий (обычно по 64 байта) [2].

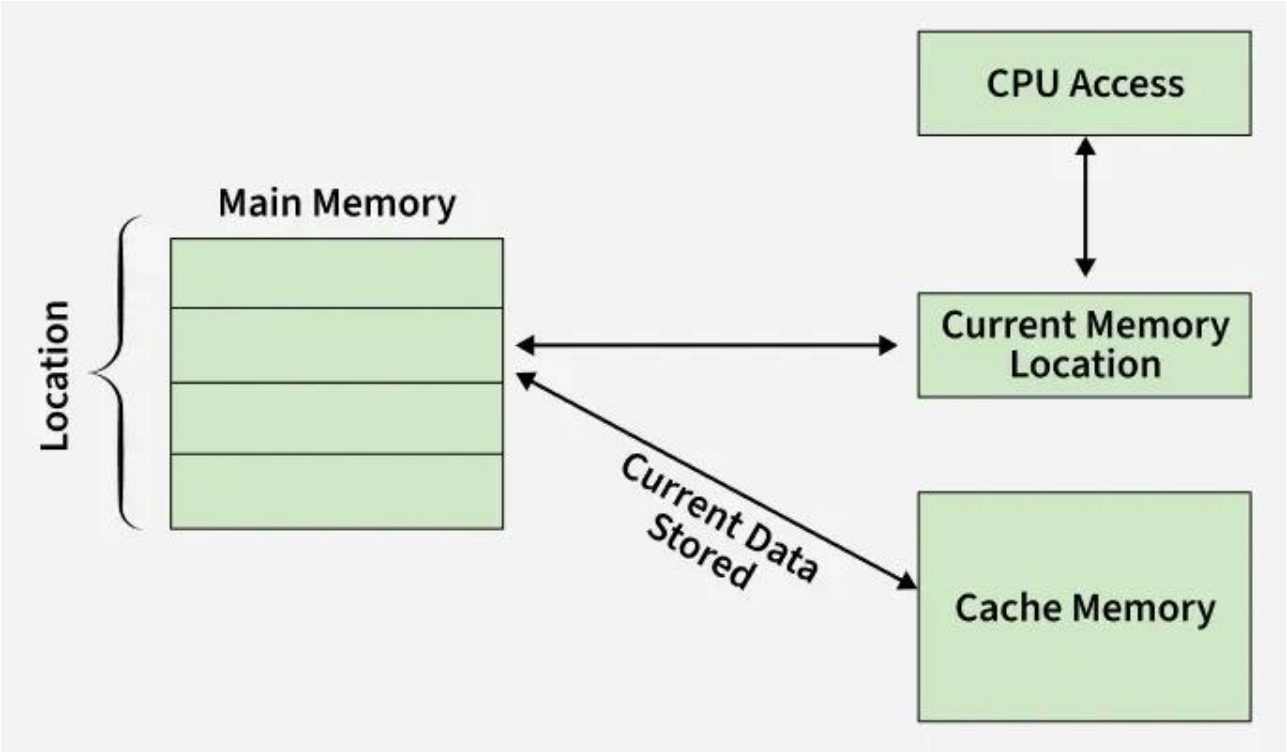

Рис. 1 Принцип пространственной локальности: выборка соседних ячеек памяти в кэш [4].

Для практической проверки этой гипотезы были проанализированы два базовых паттерна обработки данных:

Геометрический доступ: обход двумерной матрицы по строкам и по столбцам.

Структурный доступ: агрегатная обработка полей в формате «массив структур» – AoS, против «структуры массивов» – SoA.

Экспериментальный стенд базировался на процессоре x86_64 с характеристиками L1d-кэш: 32 КБ на ядро, L3-кэш: 16 МБ, размер кэш-линии: 64 байта, под управлением ОС Ubuntu 22.04 LTS. Компиляция исходного кода выполнялась с помощью GCC 11.4.0 с флагами -O2 -fno-tree-vectorize, что позволило отключить автоматическую векторизацию для изолированного анализа взаимодействия кода с памятью.

Сценарий 1. Геометрический доступ.

Тестовый алгоритм производил суммирование элементов матрицы размером N = 4096 x 4096 типа double (суммарный объем ≈ 128 МБ, что заведомо превышает объем L3-кэша).

```
// 1. Последовательный обход (Row-Major) — пространственная локальность
for (size_t i = 0; i < N; ++i) {
    for (size_t j = 0; j < N; ++j) {
        sum += matrix[i][j]; // Память читается подряд, шаг = 8 байт
    }
}

// 2. Непоследовательный обход (Column-Major) — разрушение локальности
for (size_t j = 0; j < N; ++j) {
    for (size_t i = 0; i < N; ++i) {
        sum += matrix[i][j]; // Шаг обращения = N * 8 байт (32 КБ)
    }
}
```

Рис. 2 Код программы для обхода матриц.

Сбор метрик аппаратных событий осуществлялся утилитой **perf** через команду: perf stat -e cycles,instructions,L1-dcache-load-misses,LLC-load-misses./matrix_benchmark

В результате серии из 10 прогонов при обходе по строкам (Row-Major) время выполнения составило **14.2 мс** при доле L1-misses на уровне 1.2%, LLC-misses < 0.1% и высоком показателе инструкций на цикл (IPC = 2.41). Переход к обходу по столбцам (Column-Major) при абсолютно идентичном числе математических операций O (N^2) увеличил время выполнения до **118.5 мс** (замедление в ≈ 8.3 раза). Доля промахов L1-кэша выросла до 24.8%, промахи последнего уровня LLC достигли 12.3%, а значение IPC упало до 0.38. При нелинейном шаге обращения аппаратный префетчер не может предугадать следующий адрес, из-за чего кэш-линейка загружается вхолостую: программа использует из 64 байт только 8 байт нужного элемента, а остальные отбрасываются при следующей ротации.

Сценарий 2. Структурный доступ.

Во втором эксперименте моделировалась обработка массива из N = 10000000 объектов, где требовалось просуммировать только одно целевое поле mass, не затрагивая остальные атрибуты.

```
// Модель AoS: каждый объект занимает 64 байта
struct ParticleAoS {
    double x, y, z, vx, vy, vz;
    double mass;   // Целевое поле (8 байт)
    double charge;
};
std::vector<ParticleAoS> particles_aos(N);

for (size_t i = 0; i < N; ++i) {
    total_mass += particles_aos[i].mass; // Загружается 64 байта, используется только 8
}

// Модель SoA: однородные данные упакованы в вектор
struct ParticleSoA {
    std::vector<double> x, y, z, vx, vy, vz;
    std::vector<double> mass; // Непрерывный массив масс в памяти
    std::vector<double> charge;
};
ParticleSoA particles_soa;

for (size_t i = 0; i < N; ++i) {
    total_mass += particles_soa.mass[i]; // Кэш-линия на 100% заполнена полезными данными
}
```

Рис. 3 Код программы для агрегатной обработки полей в противоположных форматах

Профилирование выявило существенный разрыв в эффективности утилизации памяти. В модели AoS время выполнения составило **34.8 мс** с долей промахов L1-кэша 14.5%, поскольку обращение к полю mass вынуждало процессор считывать всю 64-байтовую структуру целиком, захламляя L1-кэш нерелевантными полями (полезная утилизация шины – всего 12.5%). В модели SoA время работы сократилось до **12.1 мс** (ускорение в 2.88 раза), а промахи L1-кэша упали до 0.9%, так как 64-байтовая кэш-линейка целиком заполнялась 8 последовательными значениями масс, обеспечивая 100% утилизацию переданных данных [3].

Таблица 1

Сравнительный анализ производительности и уровня кэш-промахов при различных паттернах доступа и организациях данных

Паттерн доступа	Относительное время выполнения	Индекс кэш-промахов (L1/L3)
Последовательный доступ (Row-major)	1.0 x	Низкий
Доступ с большим шагом (Column-major)	4.2 x – 8.5 x	Высокий
Модель SoA (Structure of Arrays)	1.0 x	Минимальный
Модель AoS (Array of Structures)	2.1 x – 3.4 x	Умеренный

Таким образом, физическая организация данных и характер обращения к ним в памяти являются равноправными факторами производительности наряду со сложностью самого алгоритма. Учет размеров кэш-линий и проектирование cache-friendly структур позволяют добиться кратного ускорения программного обеспечения на аппаратном уровне без усложнения математической логики.

Список литературы:

1. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления // СПб.: БХВ-Петербург. 2002. 608 с.

2. Орлов С. А., Цилькер Б. Я. Организация ЭВМ и систем (3-е изд.) // СПб.: Питер. 2014. 688 с.

3. Гуров В. В. Архитектура микропроцессоров // М.: ИНФРА-М. 2010. 264 с.

4. Locality of reference and cache operation in cache memory // GeeksforGeeks. 2025. URL: https://www.geeksforgeeks.org/computer-organization-architecture/locality-of-reference-and-cache-operation-in-cache-memory/ (дата обращения: 02.08.2026)