СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ
СибГУТИ

Кафедра вычислительных систем

Сравнительный анализ эффективности алгоритмов поиска в непрерывно размещенных в памяти массивах

Терентьев Андрей Викторович, Коротков Кирилл Константинович

студенты 2 курса

Институт информатики и вычислительной техники
Сибирский государственный университет телекоммуникаций и информатики

andrey.terent1ev@mail.ru

Научный руководитель — д.т.н. профессор Курносов М.Г.

Российская научно-техническая конференция «Обработка информации и математическое моделирование»
Новосибирск, СибГУТИ, 24 апреля 2025

Задачи

Реализация алгоритмов поиска

Двоичный поиск (BS/BL/Prefetch)

Поиск – Eytzinger (Eyt/EytBL)

Микроархитектурная оптимизация

Устранение условных переходов

Явная предвыборка данных

Эксперимент

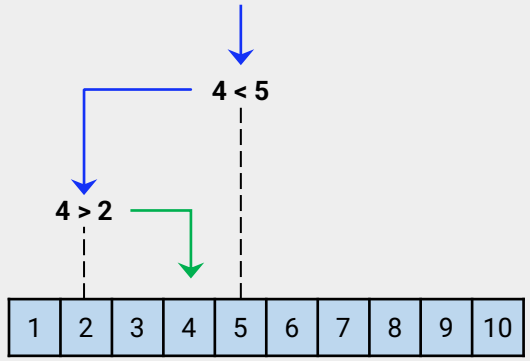
Тесты эффективности алгоритмов на **x86-64** и **RISC-V**

Алгоритмы поиска

BS	BL	Prefetch	Eyt	EytBL
Двоичный поиск с условными переходами	Двоичный поиск с устранением условных переходов	Двоичный поиск с устранением условных переходов и явной предвыборкой данных	Поиск в размещении элементов по алгоритму Eytzinger с условными переходами	Поиск в размещении элементов по алгоритму Eytzinger с устранением условных переходов

Двоичный поиск

Двоичный поиск с условными переходами



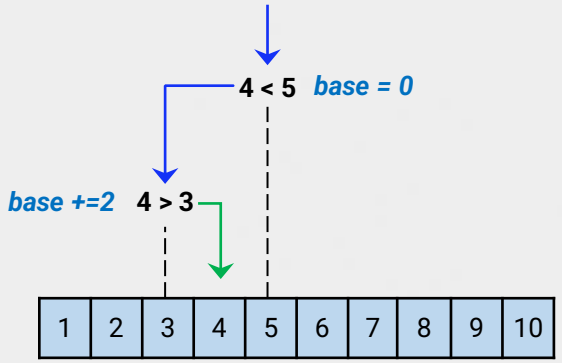

```
IdxType branchy_search(ElemType x) const{
     IdxType l = 0; IdxType r = n;
     while (l < r) {
        IdxType m = (l + r) / 2;
        if (x < a[m])r = m; // ветвления
        else if (x > a[m])l = m + 1;
        else return m;
     }
     return r;
}
```

Проблемы: условные переходы, ошибки предсказания ветвлений.

Условные переходы

Предсказуемость выполнения

Двоичный поиск с устранением условных переходов




```
IdxType branchfree_search(ElemType x) const {
   const ElemType *base = a; IdxType n = this->n;
   while (n > 1) {
     const IdxType half = n / 2;
     base = (base[half] < x) ? &base[half] : base;
     n -= half;
   }
   return (*base < x) + base - a;
          (*base < x) = 1, base - a = 2
 }
```

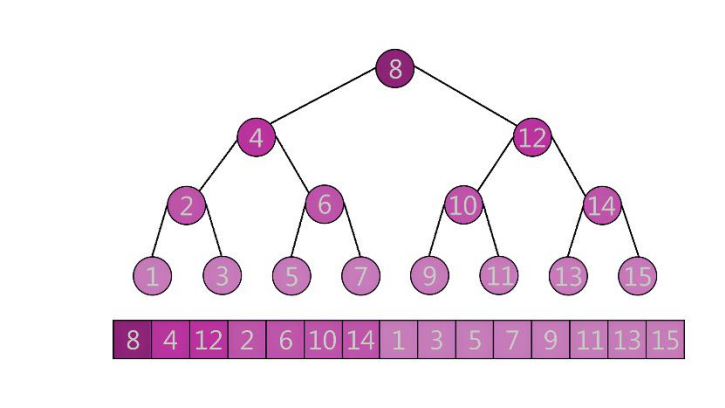
Постоянное число итераций $\log_2 n$ и отсутствие досрочного выхода улучшают предсказуемость.

Поиск – Eytzinger

Размещение элементов Eytzinger

1. Построение завершенного двоичного дерева поиска (complete binary tree).

2. Обход «в ширину» (breadth-first-search) по уровням дерева.



Алгоритм поиска

Начинаем с корня и двигаемся по дереву:

- если x меньше текущего элемента – переходим к левому дочернему узлу с индексом $2i + 1$;
- если x больше текущего элемента – идем к правому дочернему узлу $2i + 2$.

Поиск по алгоритму Eytzinger с условными переходами

```
IdxType eytz_branchy_search(ElemType x) const {
        IdxType i = 0;
        while (i < n) {
            if (x < a[i]) i = 2*i + 1;
            else if (x > a[i]) i = 2*i + 2;
            else return i;
        }
        IdxType j = (i+1) >> builtin_ffs(~(i+1));
        return (j == 0) ? n : j-1;
}
```

Поиск по алгоритму Eytzinger с устранением условных переходов

```
IdxType eytz_branchless_search(ElemType x) const {
        IdxType i = 0;
        while (i < n) {
          i = (x <= a[i]) ? (2*i + 1) : (2*i + 2);
        }
        IdxType j = (i+1) >> __builtin_ffs(~(i+1));
        return (j == 0) ? n : j-1;
}
```

Версия без условных переходов не гарантирует постоянного числа итераций: их количество зависит от глубины найденного элемента в виртуальном дереве.

Технические характеристики

Процессор	Максимальное кол-во int элементов			Опции компиляции
	L1	L2	L3	
Xeon Gold 5218N	2^{13}	2^{18}	2^{22}	–O4 –march=native
Xeon Gold 6230N	2^{13}	2^{18}	2^{23}	–O4 –march=native
SpacemiT K1	2^{13}	2^{17}	-	–O2 –mcpu=spacemit-x60 –march=rv64gc_zba_zbb_zbs

Методика проведения тестов

В качестве показателя эффективности измерялось время выполнения поиска ключа в массиве.

```
void measure_time(int n, int *index, int repeat, int max_pow) {
     srand(0);
     for (int i = 1; i <= max_pow; i++) {
        n = pow2(i); int* a = new int[n]; int x = rand()% n; // Искомый элемент
        fill_array(a, n);         // Заполнение массива псевдослучайными числами
        sorted_array<int, int> sa(a, n);
        z = std::chrono::seconds(0);
        for (int run = 0; run < repeat; run++) {
            auto start = chrono::steady_clock::now();
            *index = sa.search(x);
            auto end = chrono::steady_clock::now();
            chrono::duration<double> elapsed = end - start;
            if (run == 0) firstrun = elapsed;
            else if (run == repeat - 1) lastrun = elapsed;
            z += elapsed;
        }
        z = z / repeat      // Время алгоритма — среднее по repeat запусков
     }
}
```

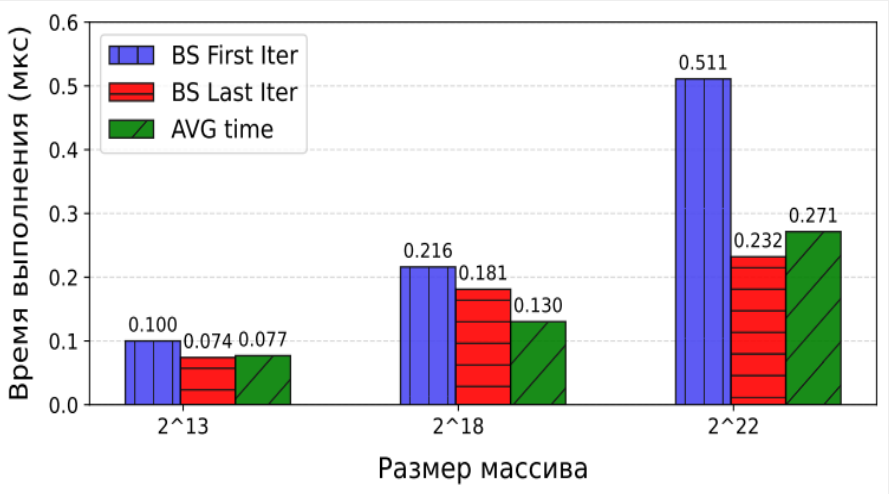
Для каждого размера массива алгоритм поиска выполнялся 10^6 раз.

На каждой итерации менялся ключ поиска, заполнение массива оставалось неизменным.

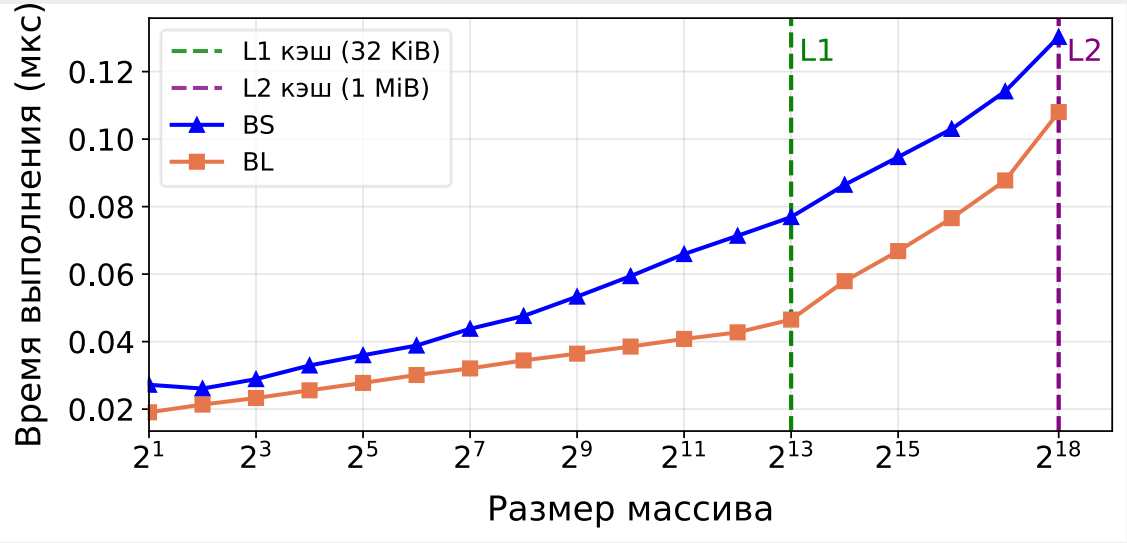
Результат измерения:

- среднее время выполнения алгоритма,
- время выполнения алгоритма на первой итерации («холодный» кэш),
- время выполнения алгоритма в последней итерации («горячий» кэш).

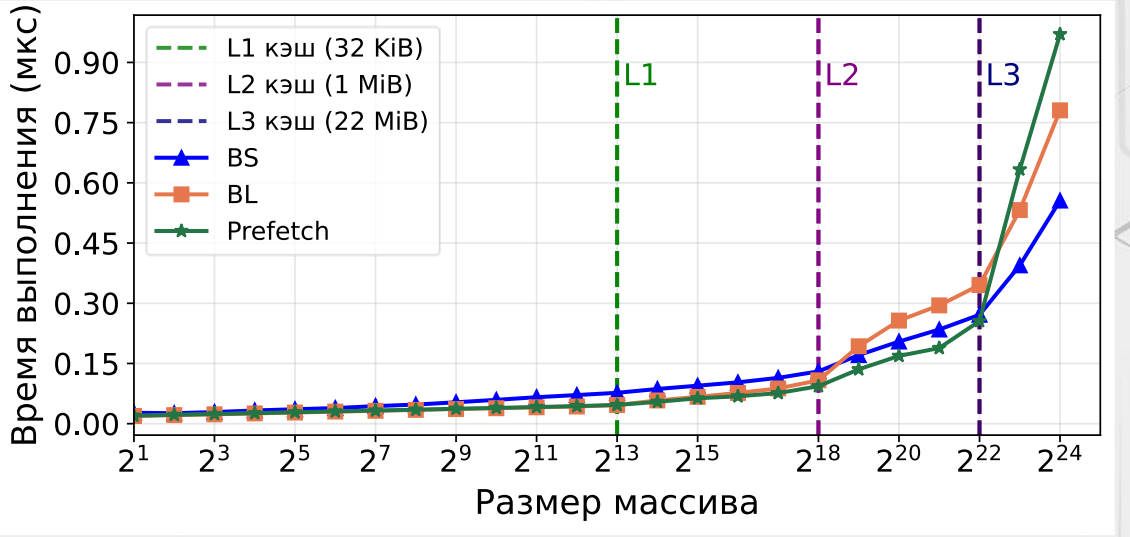
Пример измерения первой и последней итерации



Результаты исследования двоичного поиска с классическим размещением данных в массиве



На значениях до 2^{18} (уровни L1/L2) поиск без условных переходов быстрее в 1.5-1.75 раза.

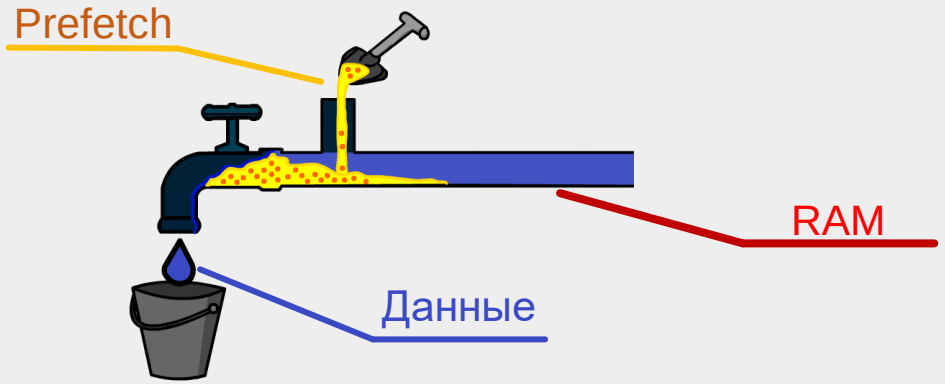


После границы кэша L3 (2^{22}) начинается резкое увеличение времени выполнения алгоритма на процессоре 6230N.

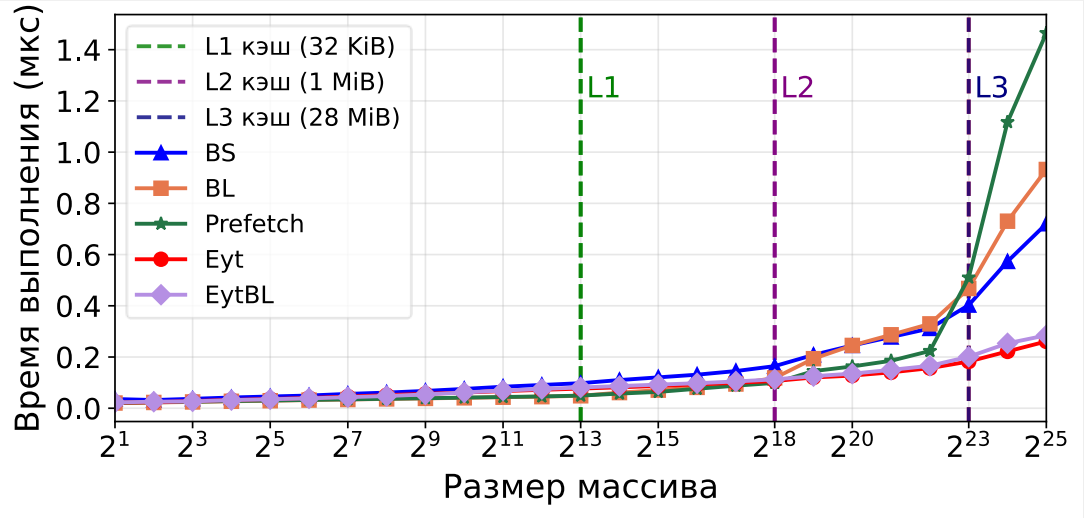
Двоичный поиск с устранением условных переходов и явной предвыборкой

```
IdxType branchfree_pref_search(ElemType x) const {
        const ElemType *base = a; IdxType n = this->n;
        while (n > 1) {
            const IdxType half = n / 2;
            builtin_prefetch(&base[half/2]);
            builtin_prefetch(&base[half + half/2]);
            base = (base[half] < x)? &base[half]: base;
            n -= half;
        }
        return (*base < x) + base - a;
}
```

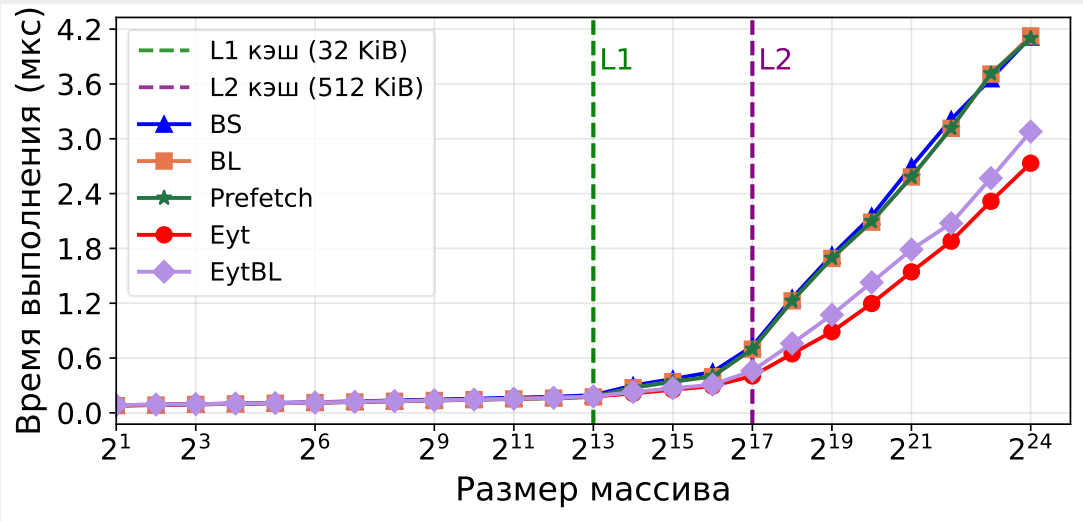
Prefetch запросы вытесняют нужные данные в очереди после уровня L3.



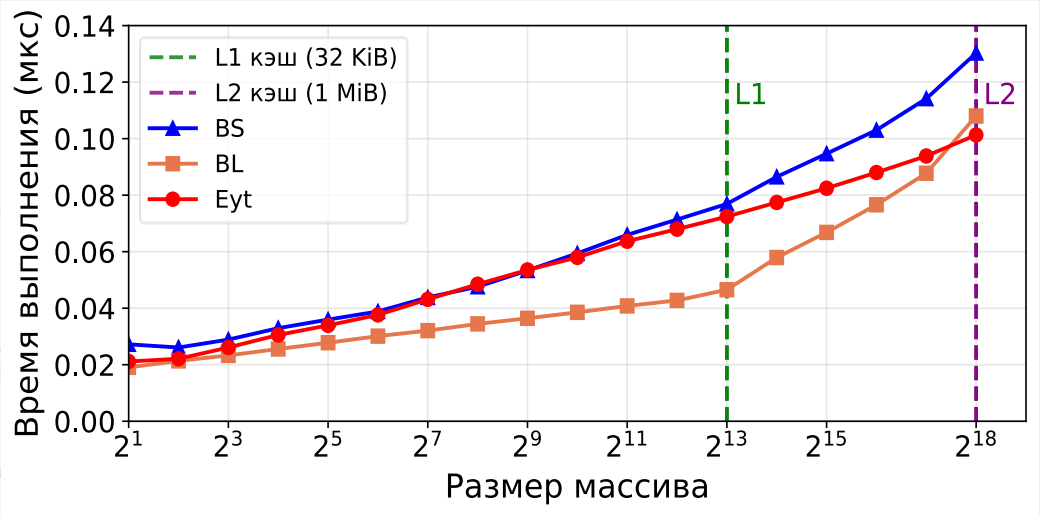
Результаты исследования всех реализованных алгоритмов



6230N



SpacemiT K1



5218N

На RISC-V Eytzinger теряет преимущество из-за отсутствия L3 кэша.

Branchless в 1.5 - 1.75 раза быстрее на малых массивах (L1/L2).

Eytzinger быстрее в 1.75 - 2 раза на средних/больших массивах (L2/L3+).

Заключение

Зависимость времени поиска от размера данных

Версии алгоритмов без условных переходов демонстрируют преимущество в 1.5–1.75 раза на малых и средних массивах, которые помещаются в кэш L1/L2.

Для больших массивов (L3+), поиск в размещении Eytzinger значительно эффективнее стандартных алгоритмов, за счет лучшей локальности данных и уменьшения количества промахов кэша.

Влияние микроархитектуры

x86-64 (Xeon Gold)
Версия без условных переходов показывает максимальную эффективность на небольших и средних массивах, но уступает другим методам при работе с большими объемами данных.

RISC-V (SpacemiT K1)
Разница менее выражена из-за отсутствия L3 кэша и особенностей предсказания ветвлений.

Направления дальнейшей работы

- ❑ Оптимизация под современные процессоры (влияние инструкций AMX, AVX-512)
- ❑ Оптимизация под специализированные задачи (поиск в сжатых данных)
- ❑ Исследование энергоэффективности алгоритмов (RISC-V, ARM Cortex).

Ссылки на литературу

1. Array Layouts for Comparison-Based Searching, 2015. URL: https://arxiv.org/abs/1509.05053
2. Algorithms for Modern Hardware, 2015. URL: https://en.algorithmica.org/hpc/
3. Clifford Stein Introduction to Algorithms. Third Edition, 2021.

СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ
СибГУТИ

Кафедра вычислительных систем

Спасибо! Вопросы?

Терентьев Андрей Викторович, Коротков Кирилл Константинович

студенты 2 курса

Институт информатики и вычислительной техники
Сибирский государственный университет телекоммуникаций и информатики

andrey.terent1ev@mail.ru

Научный руководитель — д.т.н. профессор Курносов М.Г.

Российская научно-техническая конференция «Обработка информации и математическое моделирование»
Новосибирск, СибГУТИ, 24 апреля 2025