Кадыров Тимур Валерьевич, группа 7308

Тема: «**Средства оптимизации программ на основе программной предвыборки данных**»

Цель: исследование средств оптимизации программ на основе программной предвыборки данных.

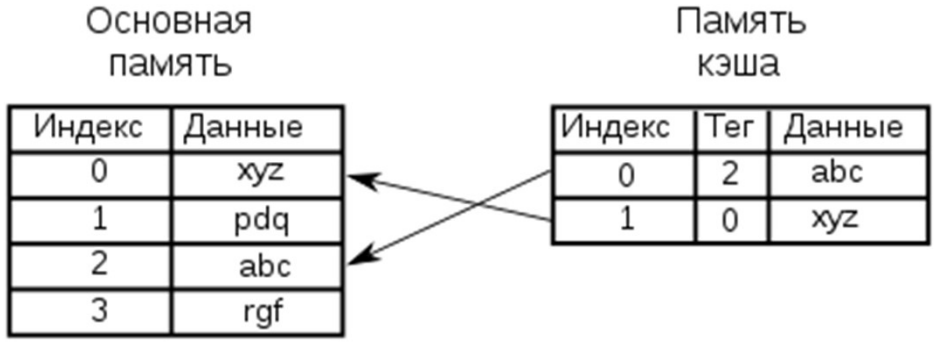
Задачи:

1) Анализ методов предвыборки данных
2) Анализ структуры LLVM
3) Анализ алгоритмов предвыборки данных
4) Тестирование алгоритмов предвыборки данных

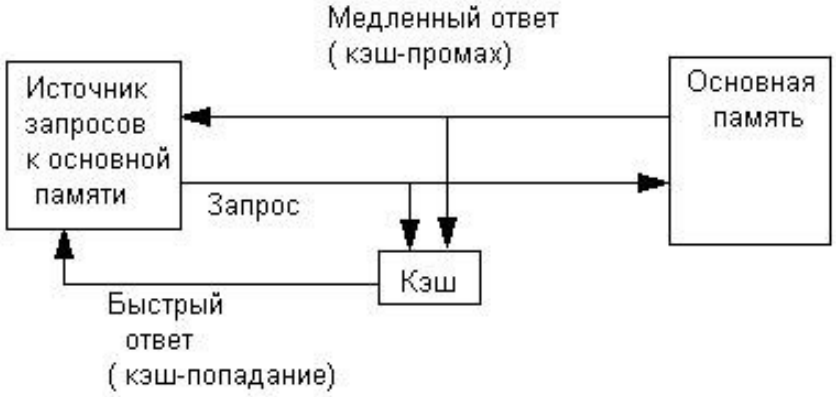
Иерархия памяти

Память в современных компьютерах имеет иерархическую структуру.

Кэш-память дублирует данные из основной памяти для ускорения доступа к ним.



При отсутствии данных в кэше происходит промах



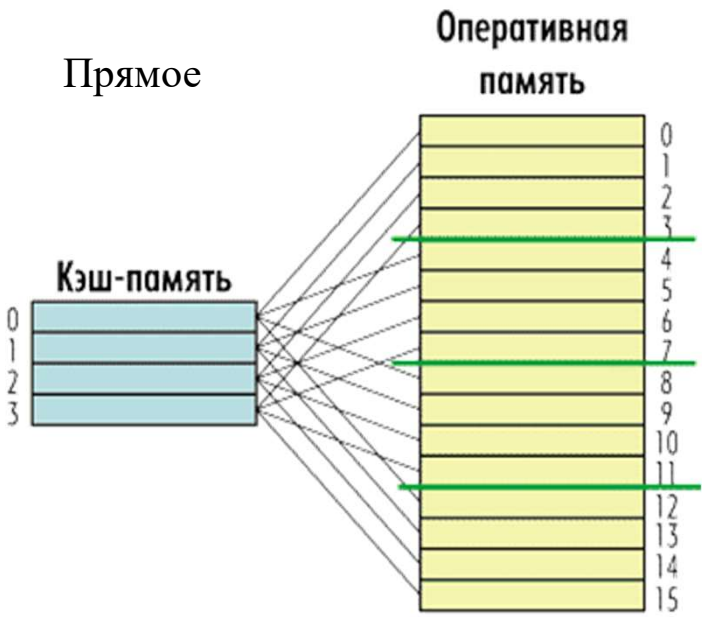
Структура кэш-памяти

Данные в кэш-памяти хранятся в виде блоков заданного размера, которые обычно называют кэш-линиями.

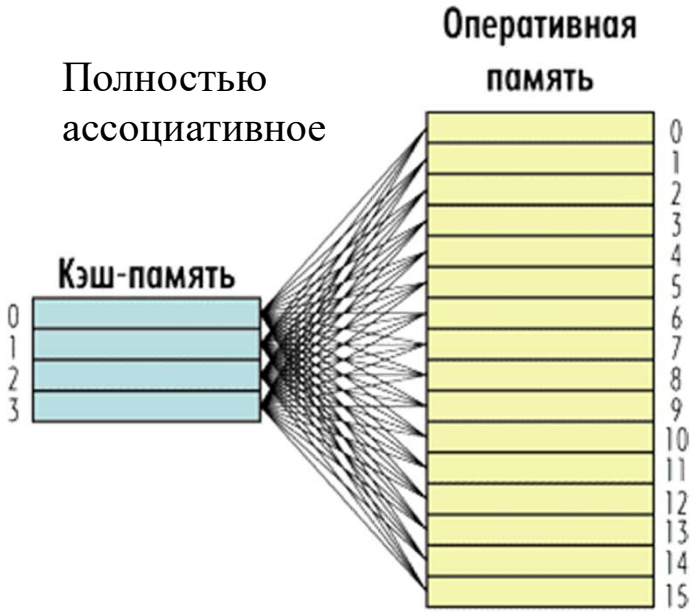
Для распределения данных по кэш-линиям применяют 3 основных алгоритма:

1) Прямое отображение (direct mapped cache)
2) Полностью ассоциативное отображение (fully associative cache)
3) Наборно-ассоциативное или множественно-ассоциативное или k-канальное ассоциативное (k-way set associative cache)

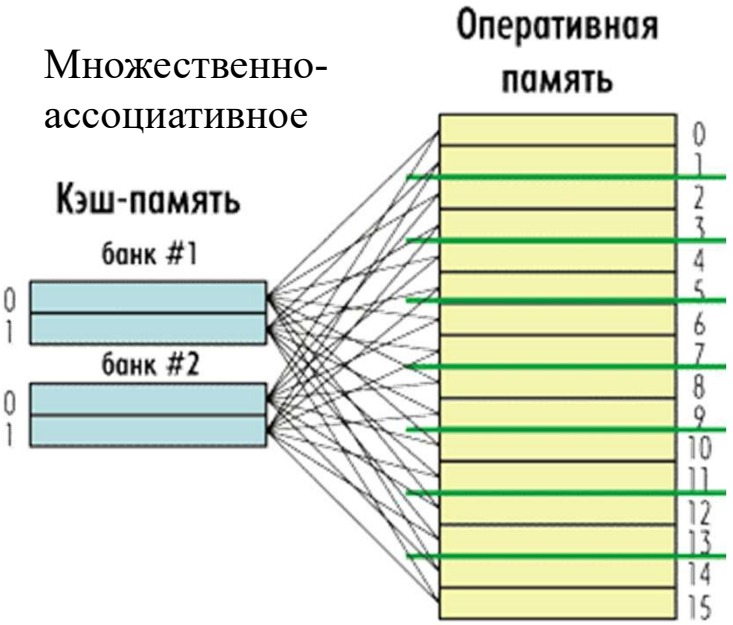
Прямое



Полностью ассоциативное

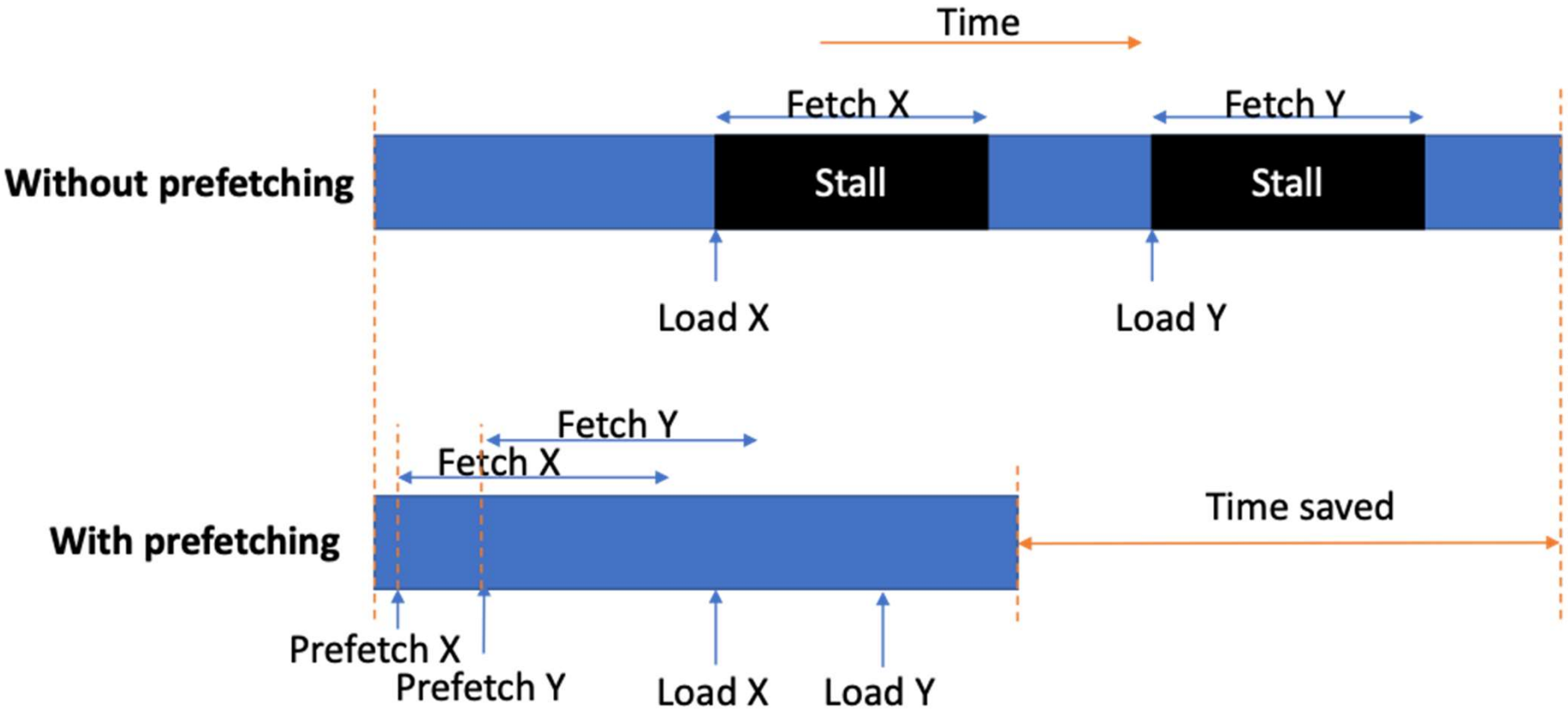


Множественно-ассоциативное



Предвыборка данных в кэш

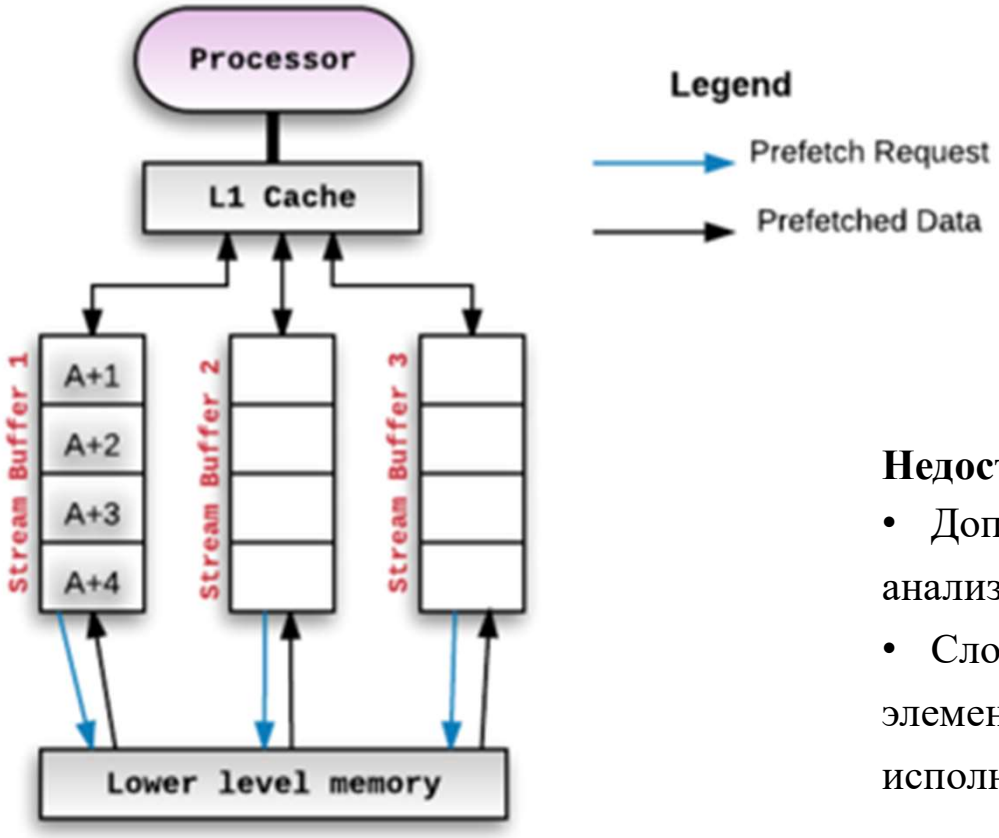
Предвыборка данных в кэш – это загрузка данных или команд из основной памяти в кэш заранее, чтобы избежать промаха кэша при обращении к этим данным. Может выполняться как аппаратными, так и программными методами.



Аппаратная предвыборка команд и данных

Аппаратная предвыборка данных осуществляется процессором с помощью встроенных алгоритмов. Существует множество методов, один из самых популярных – потоковый буфер.

Схема работы потокового буфера:



Преимущества аппаратной предвыборки:

- Процессор анализирует паттерны доступа к памяти прямо во время работы программы
- Отлично подходит для предвыборки команд и последовательных доступов к памяти (например, обход массива)

Недостатки аппаратной предвыборки:

- Дополнительные накладные расходы процессора на анализ паттернов
- Сложные паттерны (например, непрямой доступ к элементам массива) трудно предсказать во время исполнения программы

Программная предвыборка данных

Программная предвыборка данных выполняется программистом или компилятором путем вставки prefetch-инструкций. Главное преимущество в том, что можно проанализировать более сложные паттерны доступа к памяти.

```
int x[N]; int y[N];int z[N];
for(int j=0; j<N; j++)
{


  z[j] = 3*x[j]+y[j];



}
```

Код до добавления команд предвыборки

```
int x[N]; int y[N];int z[N];
for(int j=0; j<N/K; j++)
{
  PREFETCH(&x[j*K], K);
  PREFETCH(&y[j*K], K);
  for(int i=0; i<K; i++)
  {
    z[j*K+i] = 3*x[j*K+i]+y[j*K+i];
  }
}
```

Код со вставленными командами предвыборки

Алгоритмы программной предвыборки данных

Для массивов:

Original Loop

```
for (i = 0; i<100; i++)
   a[i] = 0;
```

Прямой доступ по индексу

Software Pipelined Loop (5 iterations ahead)

```
for (i = 0; i<5; i++)        /* Prolog */
   prefetch(&a[i]);

for (i = 0; i<95; i++) {     /* Steady State*/
   prefetch(&a[i+5]);
   a[i] = 0;
}

for (i = 95; i<100; i++)     /* Epilog */
   a[i] = 0;
```

Original Loop

```
for (i = 0; i<100; i++)
   sum += A[index[i]];
```

Непрямой доступ (с вычислением индекса)

Software Pipelined Loop (5 iterations ahead)

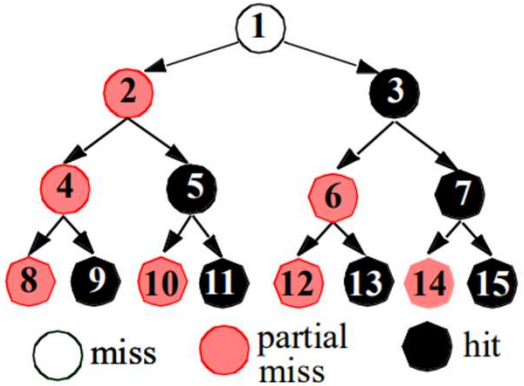
```
for (i = 0; i<5; i++)        /* Prolog 1 */
   prefetch(&index[i]);

for (i = 0; i<5; i++) {      /* Prolog 2 */
   prefetch(&index[i+5]);
   prefetch(&A[index[i]]);
}
for (i = 0; i<90; i++) {     /* Steady State*/
   prefetch(&index[i+10]);
   prefetch(&A[index[i+5]]);
   sum += A[index[i]];
}
for (i = 90; i<95; i++) {    /* Epilog 1 */
   prefetch(&A[index[i+5]]);
   sum += A[index[i]];
}
for (i = 95; i<100; i++)     /* Epilog 2 */
   sum += A[index[i]];
```

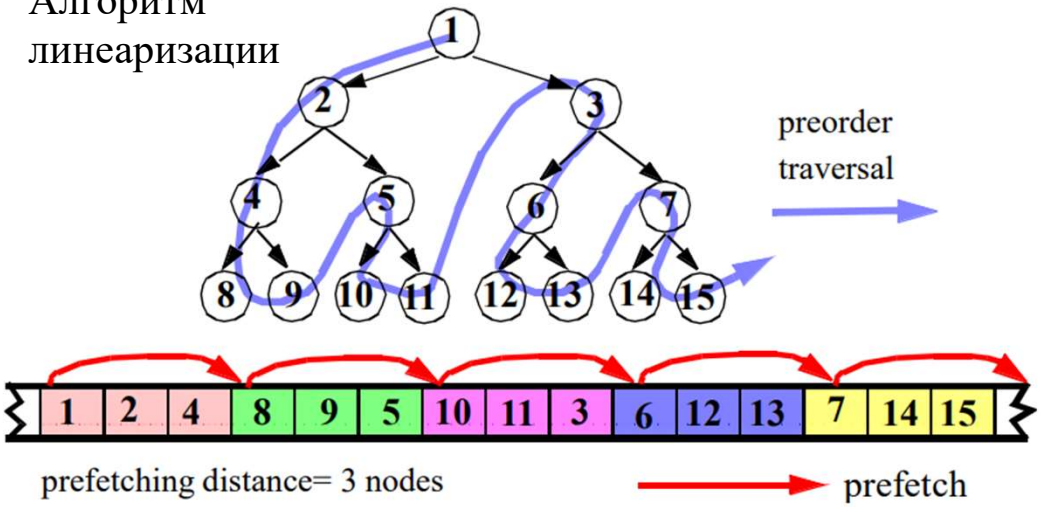
Для рекурсивных структур данных:

Жадный алгоритм

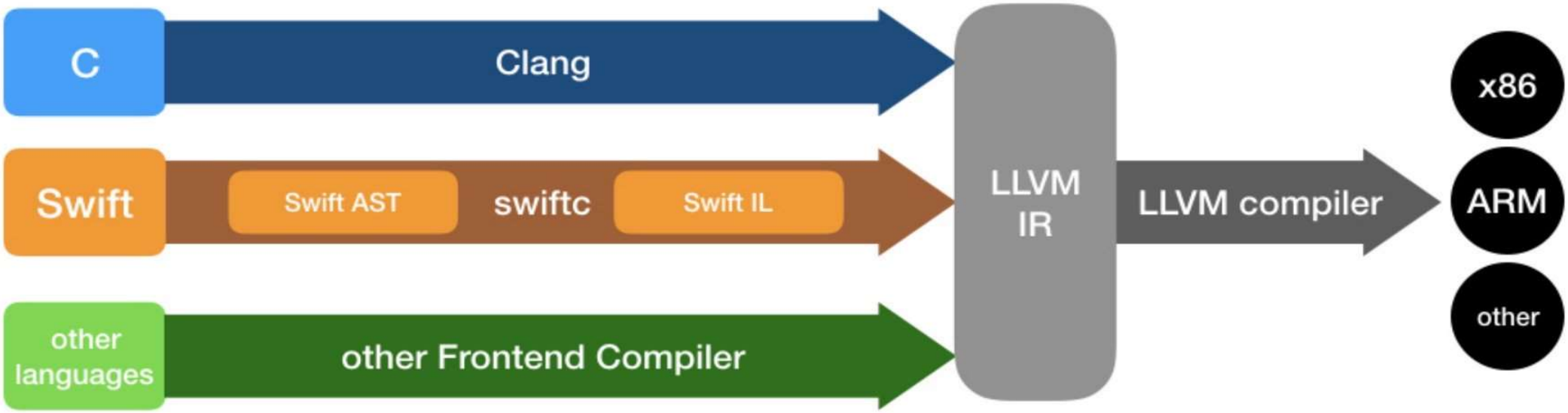
```
preorder(treeNode * t){
  if (t != NULL){
    pf(t->left);
    pf(t->right);
    process(t->data);
    preorder(t->left);
    preorder(t->right);
  }
}
```



Алгоритм линеаризации

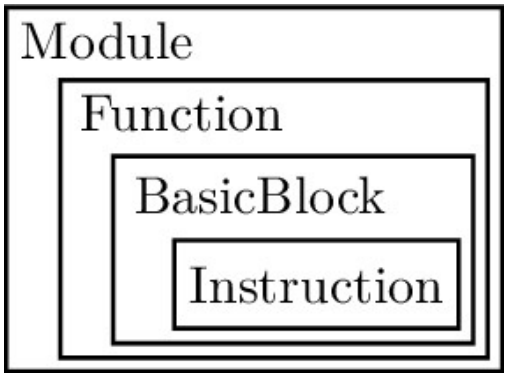


Инфраструктура для разработки компиляторов LLVM



Некоторые важные инструкции LLVM IR:

- **load и store** – обращение к памяти (запись и чтение)
- **getelementptr (gep)** – вычисление адреса
- **phi** – абстракция, определяющая значение переменной при различных переходах (например, в циклах)
- **alloca** – выделение памяти в стэк
- **icmp** – сравнение
- **select** – выбор из двух вариантов по условию



Реализация предвыборки для массивов с непрямым доступом к элементам

Для реализации был выбран данный тип массивов, так как такой паттерн трудно предугадывать процессору. Программа выполнена в виде прохода LLVM. В результате работы алгоритма получается такая трансформация:

LLVM-IR:

```
i = phi [ %inc, 0]
ptr1 = gep arr2, i
index = ld ptr1
ptr2 = gep arr, index
elem = ld i32, i32*
```

→

```
i = phi [ %inc, 0]
ptr1 = gep arr2, i
ptr1.pref  = gep arr2, i + 64
index = ld ptr1
ptr2 = gep arr, index
cond = icmp size2 < i + 32
index.1 = select cond, size2, i + 32
ptr1.1 = gep arr2, index.1
index.pref  = ld ptr1.1
ptr2.pref = gep arr, index.pref
elem = ld i32, i32*
prefetch ptr1.pref
prefetch ptr2.pref
```

аналог в Си:

```
arr[arr2[i]];
```

→

```
arr[arr2[i]];
prefetch(arr2[i+64]);
if (size < (i+32))
   prefetch(arr[arr2[i+32]);
else
   prefetch(arr[arr2[size]);
```

Описание алгоритма:

1) Находим непрямой доступ (в виде последовательного gep)
2) Ограничиваем дистанцию префетча, чтобы не выйти за границу массива индексов
3) Вставляем префетч на дистанцию, вычисленную по формуле

Формула дистанции:

$$k = \frac{c(t - (l - 1))}{t}$$

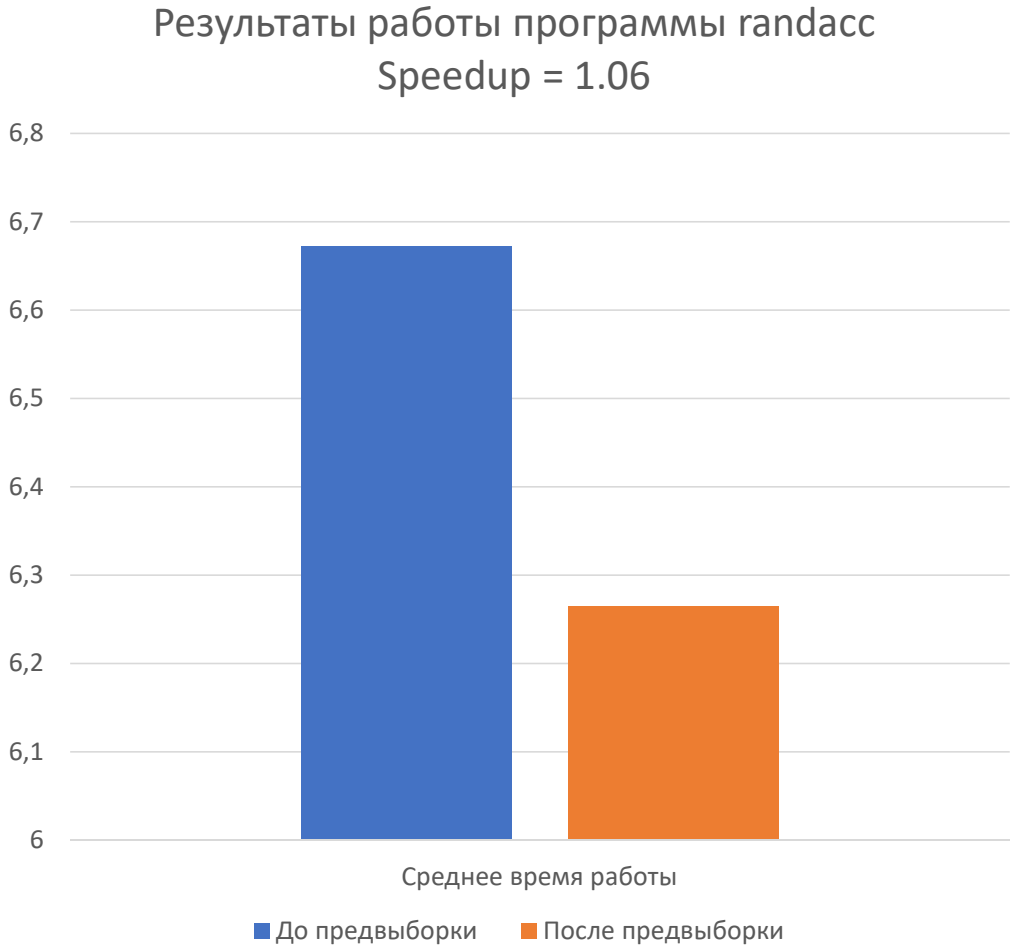
c = 64
t – число чтений
l – число чтений в префетче

Тестирование алгоритма

Для тестирования используется бенчмарк randacc, выполняющий множество доступов к большой матрице.

Результаты работы программы:

no prefetch	6.698	6.660	6.674	6.703	6.627
prefetch	6.286	6.275	6.254	6.255	6.256



Процесс сборки:

```
1. clang -S -emit-llvm -O1 -Xclang
-disable-O0-optnone -Xclang
-no-opaque-pointers randacc.c
2. opt -S -load-pass-
plugin=../libPrefetchIndirect.so
-passes=prefetch-indirect randacc.ll
-debug > randopt.ll
3. clang randopt.ll -o randopt
4. clang randacc.ll -o randacc
```

Выводы по работе

Результаты:

- исследованы методы и алгоритмы предвыборки данных;
- выполнена реализация алгоритма предвыборки данных;
- проведено тестирование алгоритма предвыборки данных.

Развитие работы:

- Больше экспериментов с различными бенчмарками на различных платформах.
- Реализация большего числа алгоритмов.
- Добавление алгоритмов в стандартную библиотеку LLVM.

Спасибо за внимание