Software Engineering Conference Russia
November 14-15, 2019. Saint-Petersburg

Structure Splitting для компилятора для микропроцессоров Эльбрус

В.Е. Шампаров[1,2] А.Л. Маркин[1,2]

[1]АО «МЦСТ»

[2]МФТИ

Проблема
Что решает Structure Splitting

```
typedef struct arc_t arc; /* sizeof(arc) = 32 */
struct arc_t{ ...; int cost; ...; arc *next; };
```

```
void UseStruct( arc *arcs, int size, int add) {
    for( int i = 0; i < size; i++ )
        arcs[i].cost += add;
}
```

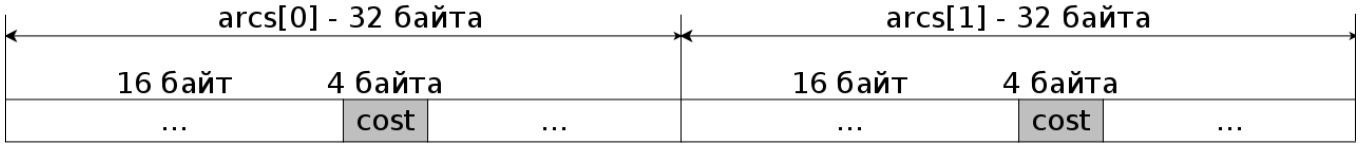

Рис. 1: Загружаемые в кэш данные на первой итерации

Часть загруженных данных не используется – неэффективное использование кэша

Идея Structure Splitting
Суть

Поделим структуру на две структуры меньшего размера:

- горячую – с часто используемыми полями;
- холодную – с редко используемыми полями.

```
typedef struct hot_arc_t
                hot_arc;
struct hot_arc_t {
    ...;
    int cost;
    cold_arc *this_cold;
};
```

```
typedef struct cold_arc_t
                cold_arc;
struct cold_arc_t {
    ...;
    hot_arc *next;
};
```

Далее будем работать с горячей структурой как со всей, а к холодной получим доступ через поле `this_cold` горячей.

Идея Structure Splitting
Обоснование

Плюсы:

- меньше промахов кэша при доступе к горячим полям из-за уменьшения размера элемента горячего массива;

Минусы:

- больше операций при доступе к холодным полям;
- при копировании структуры нужно корректно обработать холодную структуру;

Реализация
Перед всем

Как включить автоматическое применение:

Компилятор	Необходимые опции
GCC версий 4.3 – 4.7 или ветка `struct-reorg-branch`	`-fipa-struct-reorg` `-fwhole-program -combine`
LLVM	`-fstruct-layout=[1,2,3] -flto`
Intel C++ Compiler	Включено по умолчанию
LCC для архитектур «Эльбрус» и SPARC (будет доступна с версии 1.25)	`--true=struct_split` `-fwhole` `--true=struct_access_count`

Реализация
Сбор данных

Нужно знать счётчики обращений к полям.

Примеры для ручного режима:

- в C++ все обращения к полям структур заменить на вызовы сеттеров/геттеров и получить профиль с отключённой inline-оптимизацией;
- с помощью профиля вручную оценить счётчики.

Важно! Если в качестве данных не уверены, лучше не делать. Программа может замедлиться.

Реализация
Анализ

Алгоритм:

1. находим для каждой рассматриваемой структуры S_j максимальный счётчик поля M_j;
2. маркируем все поля в элементе массива как горячие или холодные:
   - поле $f_{i,j}$ (в структуре S_j) со счётчиком $c_{i,j}$ горячо, если $\frac{M_j}{c_{i,j}} \leq C$, C - константа;
   - иначе поле холодное;
3. убираем из рассмотрения все структуры, у которых все поля горячие;
4. для оставшихся структур при ручном применении можно оценить эффект оптимизации.

Реализация
Применение к типам

Было:

```
typedef struct arc_t arc; /* sizeof(arc) = 32 */
struct arc_t{ ...; int cost; ...; arc *next; };
```

Стало:

```
typedef struct hot_arc_t
                hot_arc;
struct hot_arc_t {
    ...;
    int cost;
    cold_arc *this_cold;
};
```

```
typedef struct cold_arc_t
                cold_arc;
struct cold_arc_t {
    ...;
    hot_arc *next;
};
```

При ручном применении лучше горячую структуру сделать с тем же именем, с которым была старая структура.

Реализация
Применение к переменным

Замена типов всех переменных типа старой структуры на тип горячей структуры. При ручном применении нужна, только если старая и горячая структуры имеют разные названия.

Было	Стало
`arc *val;`	`hot_arc *val;`

Исключение: сами массивы структур. В этом случае вместо старого массива создаём два: горячий и холодный.

Было	Стало
`arc *arr;`	`hot_arc *hot_arr;` `cold_arc *cold_arr;`

Реализация
Применение к обращениям

Для полей горячих структур можно ничего не менять.
Для полей холодных структур:

- если обращение к полю было через элемент массива с индексом, то заменяем на обращение через элемент холодного массива с индексом;
- иначе добавляем чтение указателя на холодную структуру.

Было	Стало
`arr[i].cost` `arr[i].next`	`hot_arr[i].cost` `cold_arr[i].next`

`arc *arr_elem_ptr;`
`arr_elem_ptr->cost`
`arr_elem_ptr->next`
	`hot_arc *arr_elem_ptr;`
	`arr_elem_ptr->cost`
	`arr_elem_ptr->this_cold->next`

Реализация
Ручное применение к выделению памяти

Все вызовы функций выделения памяти типа `malloc` или операторов `new` заменяем на пары вызовов для горячих и холодных массивов. Затем проставляем указатели на холодные элементы в горячих.

Было	`arr = calloc( new_size, sizeof(arc) );`

Стало	`hot_arr = calloc( new_size,`
	`  sizeof(hot_arc) );`
	`cold_arr = calloc( new_size,`
	`  sizeof(cold_arc) );`
	`for ( i = 0; i < new_size; i++ ) {`
	`  hot_arr[i].this_cold = &(cold_arr[i]);`
	`}`

Реализация

Автоматическое применение к выделению памяти

Отличие от ручного – выделение памяти для обоих массивов одним вызовом функции и вычислением внутри выделенной памяти адреса холодного массива.

Было	`arr = calloc( new_size, sizeof(arc) );`

Стало	`hot_arr = calloc( new_size,`
	`  sizeof(hot_arc) +`
	`  sizeof(cold_arc) );`
	`cold_arr = (cold_arc*)(hot_arr + new_size);`
	`for ( i = 0; i < new_size; i++ ) {`
	`  hot_arr[i].this_cold = &(cold_arr[i]);`
	`}`

Результаты

Измерительный стенд: компьютер с процессором «Эльбрус-8С» 1300 МГц с архитектурой системы команд типа VLIW.

Наборы тестов: SPEC CPU2000 и SPEC CPU2006.

Применение	Ручное	Автоматическое
181.mcf		+19%
429.mcf	+20%	+12%

Применение

В любой программе, где:

- нужна высокая производительность;
- часто используемые данные хранятся в массивах структур;
- другие способы повышения производительности уже использованы.

Пример – gamedev.

Контакты

Докладчик:

Виктор Шампаров
АО «МЦСТ», МФТИ
victor.shamparov@yandex.ru
Профиль LinkedIn