ПРИМЕНЕНИЕ ТЕХНИКИ РАЗРЕЗАНИЯ МАССИВА СТРУКТУР ДЛЯ СЛУЧАЯ ПЕРЕВЫДЕЛЕНИЯ ПАМЯТИ

Выполнил:
студент группы М01-813 В. Е. Шампаров

Научный руководитель:
к. ф.-м. н. М. И. Нейман-заде

Консультант:
А. Л. Маркин

Москва, 2020

Основная проблема
Неэффективность использования кэша

```
struct InnerStruct { float val; int id[2]; InnerStruct *other; };
struct BigStruct { int size; InnerStruct *data; };
```

```
void UseStruct( BigStruct embr_struct, int size, float add ) {
    int i;
    for ( i = 0; i < size; i++ )
        embr_struct.data[i].val += add;
}
```

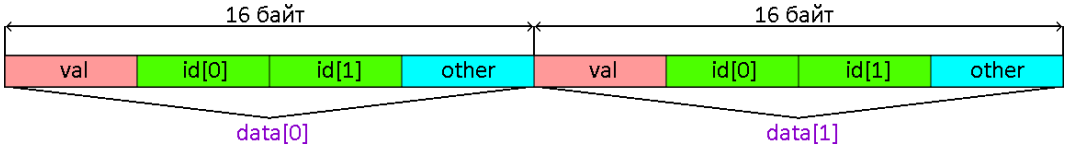


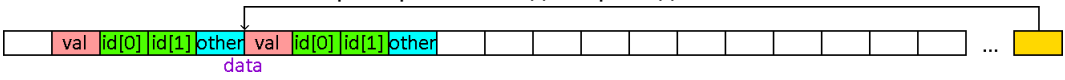
Рис. 1: Данные в кэш-линии на первой итерации цикла

При работе только с одним полем элемента вложенного массива в цикле часть загруженных данных не используется, что приводит к неэффективному использованию кэша

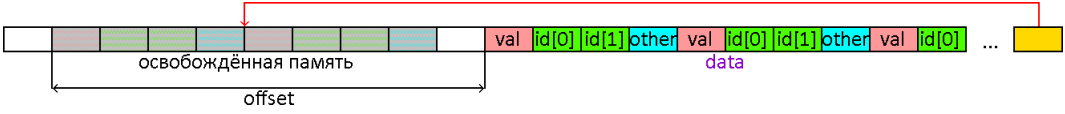
Сопутствующая проблема
Перевыделение памяти

Дополнительная сложность при решении проблемы неэффективного использования кэша – перевыделение памяти с кодом восстановления корректности указателей. Нужно оставить этот код работающим.

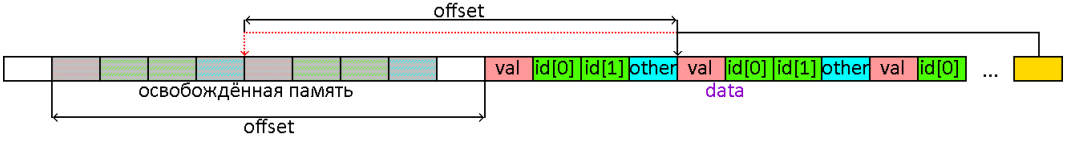
Например, память до перевыделения:



Память после перевыделения:



Память после восстановления корректности указателей:



Существующее решение проблемы неэффективного использования кэша

В компиляторе lcc для процессоров АО «МЦСТ» реализована оптимизация structpeel, решающая похожую проблему неэффективного использования кэша:

- Декомпозирует вложенный массив структур в несколько массивов, каждый из которых соответствует полю структуры:

```
typedef struct {
    int size;
    struct { float val; int id; } *data;
} BigStruct;
```

```
typedef struct {
    int size;
    float *data_val;
    int *data_id;
} BigStruct;
```

- Неприменима для случаев, когда существуют внешние указатели на элементы декомпозируемых массивов;
  - В частности, неприменима для случаев, когда к массиву применяется алгоритм восстановления корректности указателей на элементы массива.

Принцип решения проблемы неэффективного использования кэша

Разрезать вложенный массив структур на два связанных указателями:

1. массив из часто используемых полей (красные поля на рисунке):
   - все внешние указатели на элементы массива перенаправить на него;
   - из каждого элемента этого массива указатель ведёт на соответствующий элемент второго массива;
2. массив из редко используемых полей (зелёные и голубые поля на рисунке).

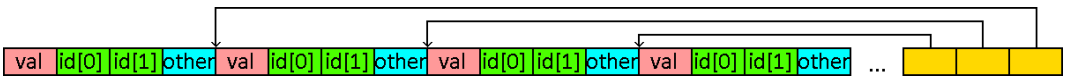


Рис. 2: Пример расположения данных в массиве и внешних указателей

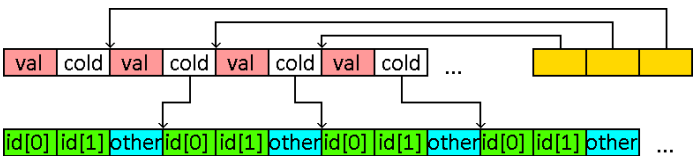


Рис. 3: Пример преобразования массива структур и внешних указателей

Цель работы

Разработать оптимизацию разрезания structsplit для компилятора lcc, преобразующую вложенный массив структур в пару из двух массивов, в первом из которых в каждом элементе есть указатель на соответствующий элемент второго.

Например:

```
typedef struct { float val; int id[2]; InnerStruct *other; } InnerStruct;
typedef struct { int size; InnerStruct *data; } BigStruct;
```

преобразуется в:

```
typedef struct { float val; InnerStruct_cold *cold_part; } InnerStruct_hot;
typedef struct { int id[2]; InnerStruct_hot *other; } InnerStruct_cold;
typedef struct { int size; InnerStruct_hot *data_hot;
                           InnerStruct_cold *data_cold;
} strspl_BigStruct;
```

Известные решения

Оптимизации Structure Splitting для компиляторов GCC 4.3-4.8 и Open64

- Для GCC:
  - Не обрабатывает перевыделение памяти;
- Для Open64:
  - Не позволяет выделить в отдельные массивы более одного поля;
  - Не обрабатывает перевыделение памяти;

Для программ на языке Java оптимизация Structure Splitting для научного компилятора Vortex:

- Использует вычисленные из профильной информации счётчики обращений к всем полям и структурам компилируемой программы.
- Не обрабатывает перевыделение памяти;

Реализованное решение

Реализована оптимизация structsplit для C и Fortran в составе компилятора lcc. Алгоритм:

1. этап анализа - поиск структур для оптимизации, основанный на алгоритме для Vortex;
2. этап преобразования:
   - 2.1 коррекция типов;
   - 2.2 коррекция типов объектов;
   - 2.3 изменение всех обращений к скорректированным объектам;

Принцип разрезания:

1. первый массив («горячий») – из часто используемых полей элемента массива;
2. второй массив («холодный») – из редко используемых полей элемента массива;

Этап анализа

Для каждого вложенного массива структур производится вычисление счётчиков обращений к полям элемента массива и анализ, состоящий из:

1. поиска максимального счётчика M_j обращений к полю элемента массива;
2. разделения всех полей элемента массива по критерию $\frac{M_j}{c_{i,j}} \leq C$, C – «граничная константа», задаваемая пользователем, $c_{i,j}$ – счётчик обращений к конкретному полю; удовлетворяющие условию поля считаются «горячими», остальные – «холодные»;
3. принятия решения: если есть и «горячие», и «холодные» поля, то массив структур можно разрезать.

Все массивы структур, к которым можно применить оптимизацию, записываются в специальную таблицу FOT вместе с наборами «горячих» и «холодных» полей и охватывающей структурой.

Этап преобразования
Коррекция типов

Все найденные массивы структур из таблицы FOT модифицируются:

1. из «горячих» и «холодных» полей создаются соответствующие структуры;
2. в охватывающей структуре указатель на начало массива заменяется на пару указателей на начала массивов из «горячих» и «холодных» структур.

Во всех прочих структурах и типах функций: указатели с типом разрезанных массивов структур заменяются на указатели с типом соответствующих «горячих» структур.

Например:

```
typedef struct { float val; int id[2]; InnerStruct *other; } InnerStruct;
typedef struct { int size; InnerStruct *data; } BigStruct;
```

преобразуется в:

```
typedef struct { float val; InnerStruct_cold *cold_part; } InnerStruct_hot;
typedef struct { int id[2]; InnerStruct_hot *other; } InnerStruct_cold;
typedef struct { int size; InnerStruct_hot *data_hot;
                           InnerStruct_cold *data_cold; } strspl_BigStruct;
```

Этап преобразования
Коррекция типов объектов

1. Заменяются типы всех объектов в соответствии с модифицированными на предыдущем этапе;
2. Все указатели на элементы разрезанных массивов заменяются на указатели на соответствующие элементы «горячих» массивов.

Например:

```
BigStruct b;
InnerStruct *ptr;
```

преобразуется в:

```
strspl_BigStruct b;
InnerStruct_hot *ptr;
```

Этап преобразования
Замена обращений

Производятся преобразования для разрезаемых массивов:

- обращение к полю элемента разрезаемого массива с помощью охватывающей структуры заменяется на обращение к соответствующему полю в «горячем» или «холодном» массиве;

`b->data[i].id`	`b->data_hot[i].id`
`b->data[i].val`	`b->data_cold[i].val`

- обращение к «холодному» полю элемента массива с помощью указателя на элемент заменяется на чтение указателя на «холодную» структуру в «горячей» и обращение к соответствующему полю в «холодной» структуре;

`ptr->id`	`hot_ptr->id`
`ptr->val`	`hot_ptr->cold_part->val`

Этап преобразования
Пример замены выделения памяти

Выделение или перевыделение памяти для массива заменяется на специальный код для выделения памяти и сохранения связности созданных массивов.

Пример замены:

```
BigStruct *big; big.data = calloc( size, sizeof(InnerStruct));
```

```
BigStruct *big;
size_t new_size = size * sizeof(InnerStruct_hot)
    + size * sizeof(InnerStruct_cold)
    + alignof(InnerStruct_cold) - alignof(InnerStruct_hot);
big.data_hot = calloc( new_size, 1 );
size_t new_cold_ptr = (size_t)big.data_hot + size * sizeof(InnerStruct_hot)
    + alignof(InnerStruct_cold) - alignof(InnerStruct_hot);
big.data_cold = (InnerStruct_cold*) new_cold_ptr;
for( int i = 0; i < size; i++ ) {
    big.data_hot[i].cold_part = &(big.data_cold[i]);
}
```

Измерения и анализ

Измерения проведены на компьютерах с процессорами «Эльбрус-4С», «Эльбрус-8С» и «Эльбрус-8СВ».
Измерено ускорение исполнения тестов в зависимости от константы разделения «горячих» и «холодных» полей – «граничной константы». Выбрана «граничная константа», обеспечивающая максимальное среднее ускорение.
Измерено изменение количества блокировок конвейера по считыванию данных при применении оптимизации с выбранной константой.
Контекст для применения оптимизации найден в задачах из пакетов SPEC CPU2000, SPEC CPU2006 и SPEC CPU2017:

1. 181.mcf, 429.mcf, 505.mcf;
2. 464.h264ref, 525.x264 – подходящие структуры малозначимы, так что результаты околонулевые;
3. 482.sphinx3 – подходящие структуры малозначимы, так что результаты околонулевые.

Измерения и анализ
Среднее ускорение

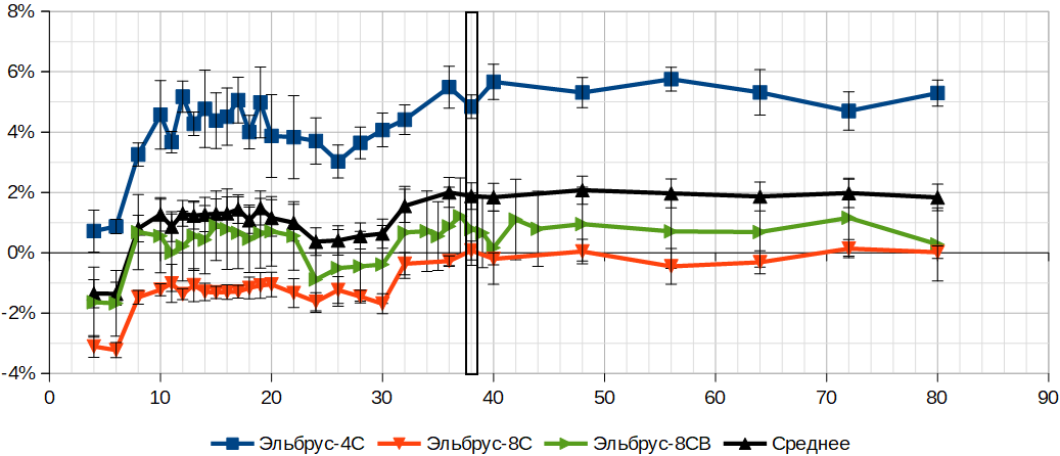


Рис. 4: Среднее геометрическое ускорение по 6 тестам в зависимости от «граничной константы»; обведено выбранное значение «граничной константы»

Максимальное ускорение и блокировки по памяти

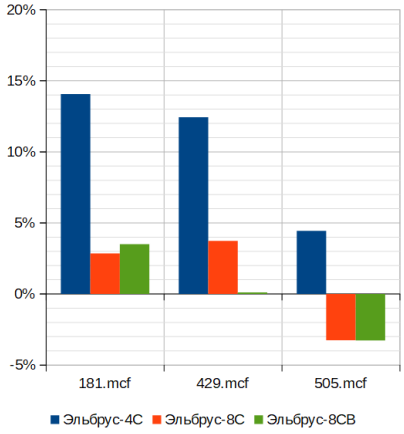


Рис. 5: Ускорение по задачам при «граничной константе», обеспечивающей максимальное среднее ускорение

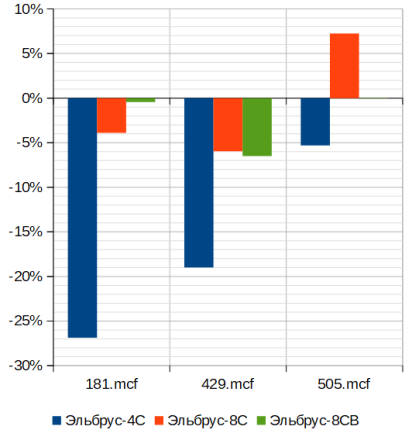


Рис. 6: Изменение количества блокировок конвейера по считыванию данных при применении оптимизации с выбранной константой

Измерения и анализ

Выводы

- Множества значений «граничной константы», при которых достигается максимальное среднее ускорение, для различных процессоров почти совпадают;
- Достигнутое в результате оптимизации ускорение сильно зависит от характеристик кэша процессора: наибольшее ускорение достигается на «Эльбрус-4С», имеющем двухуровневый кэш;
- Для процессоров «Эльбрус-4С» и «Эльбрус-8С» ускорение и падение количества блокировок по памяти друг другу соответствуют; на «Эльбрус-8СВ» установить данное соответствие не удалось;
- Так как оптимизация может и ускорить выполнение теста, и замедлить, оптимизация не включена в набор оптимизаций по умолчанию, а включается опцией -fstruct-reorg.
- Наибольшие ускорения:
  - среднее геометрическое ускорение на 6 задачах: 2,1% $\pm$ 0,5%;
  - ускорение на одной задаче при выбранной «граничной константе»: 14% $\pm$ 2% (задача 181.mcf на «Эльбрус-4С»).

Результаты

- Реализована оптимизация structsplit, разрезающая массив вложенных структур и обеспечивающая работоспособность имеющегося кода восстановления корректности указателей на элементы массива.
- Оптимизация прошла проверку на некоторых задачах пакетов тестирования SPEC CPU на различных процессорах. Зафиксировано среднее ускорение выполнения на 6 задачах вплоть до 2,1% $\pm$ 0,5% и максимальное ускорение на 26% $\pm$ 3% в задаче 181.mcf на компьютере с процессором «Эльбрус-4С».
- Дальнейшие направления работы:
  - добавление в structsplit возможности оптимизации программ на языке C++;
  - поиск и применение иных преобразований расположения данных компилируемых программ для улучшения работы с кэш-памятью.

Проблема
Перевыделение памяти (код)

Код перевыделения памяти и восстановления корректности указателей на примере указателей из одних элементов массива в другие элементы того же массива:

```
void Realloc( InnerStruct **data, int old_size, int new_size ) {
    int i;
    InnerStruct *old_data = *data;
    *data = realloc( *data, new_size * sizeof(InnerStruct) );
    long offset = (long)*data - (long)old_data;
    for ( i = 0; i < old_size; i++ )
        *data[i].other = (InnerStruct*)((long)*data[i].other + offset);
}
```