OpenEuler | Compiler SIG

630高性能版本特性介绍

单位：华为技术有限公司

汇报人：王鼎 李彦成 伍明川 林达 丁光亚

指针压缩优化

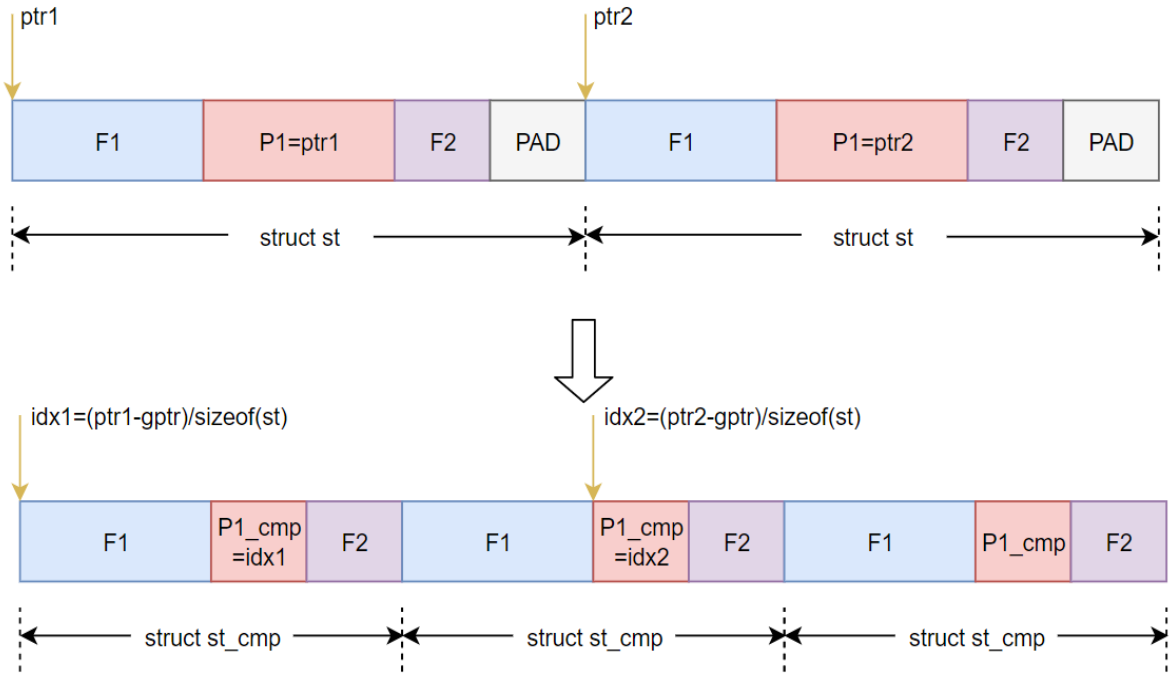
OpenEuler Compiler SIG

```
Struct st
{
  long f1;
  st *p1;
  int f2;
};
```

```
Struct st_ptrcmp
{
  long f1;
  uint8 p1_ptrcmp;
  int f2;
};
```

```
Struct st_ptrcmp
{
  long f1;
  uint16 p1_ptrcmp;
  int f2;
};
```

```
Struct st_ptrcmp
{
  long f1;
  uint32 p1_ptrcmp;
  int f2;
};
```



场景：

- 连续访问大型结构体数组可能使访存成为瓶颈
- 通过将结构体指针转换为结构体相对于内存池中首个元素的偏移，可以将指针从64 bits压缩至32、16或8bit
- 压缩后的结构体体积更小，能够有效降低内存使用量，缓解带宽压力，提升cache命中率
- 以SPECCPU2017 intrate 505.mcf为例，性能提升**18%**

限制：

- 指针压缩要求目标结构体类型的所有元素都来自于同一个内存池
- 用户需要保证压缩后的指针大小合法，在不同压缩等级下，结构体数组的大小的最大值分别为：8 bit(2^8-1)， 16 bit(2^16-1)， 32 bit(2^32-1)，

冗余成员消除优化

OpenEuler Compiler SIG

优化背景：

- 部分程序存在消除无用结构体成员（DFE）的优化机会。
- 优化场景：结构体域成员出现冗余成员。

  =>内存空间多余消耗，造成页表刷新频繁、内存访问延时，导致程序性能不佳。

优化效果：

- 目标：分析并消除dead field，缩小结构体大小。
- 效果：消除没有引用的冗余成员，缩小结构体内存布局大小，以减少程序所分配空间大小，提升访存连续性。
- 配合效果：与指针压缩等结构体相关优化配合，能产生更好的性能加速效果。以SPECCPU2017 intrate 505.mcf为例。

  DFE(+3%) + 指针压缩(+4%) => **+18%**
- 使用限制：DFE优化会破坏内存布局。

  不建议在使用指针偏移访问的场景里启用该优化。

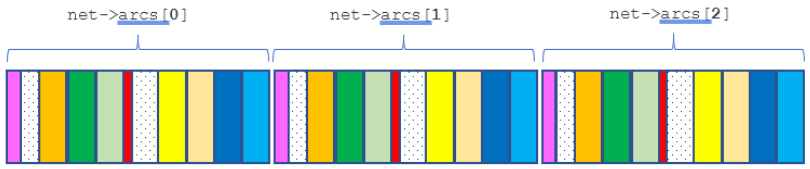
```
typedef struct {int f_a; double f_b; int f_c;} str;
str *arr = (str*)malloc(N * sizeof(str));
for (int i = 0; i < N; i++)
  … = arr[i].f_a;
for (int i = 0; i < N; i++)
  … = arr[i].f_c;
```

Before dead field elimination.

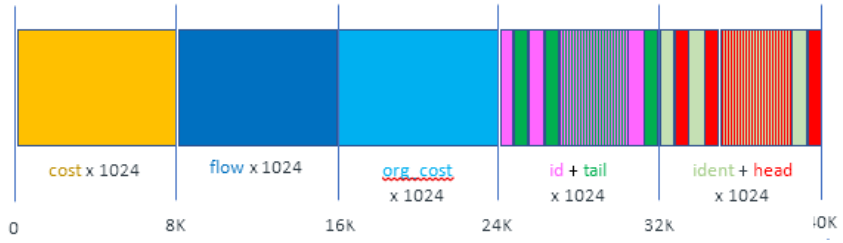
```
typedef struct {int f_a; int f_c;} str;
str *arr = (str*)malloc(N * sizeof(str));
for (int i = 0; i < N; i++)
  … = arr[i].f_a;
for (int i = 0; i < N; i++)
  … = arr[i].f_c;
```

After dead field elimination.

结构体layout优化



before relayout



after relayout
8KB对齐

```
struct arc
{
  int id;
  cost_t cost;
  uint16_t tail, head;
  short ident;
  flow_t flow;
  cost_t org_cost;
};
```

reorder fields

```
struct arc
{
  cost_t cost;      // 8 Byte
  flow_t flow;
  cost_t org_cost
  int id;       // 4 Byte
  uint16_t tail;    // 2 Byte
  uint16_t head;
  short ident;
};
```

内存分配改写

1. calloc -> aligned_alloc，并在第一个8KB的内存里记录分配空间的大小；
2. 将返回的指针偏移8KB，作为保存结构体数据的起始地址，将结构体地址改写其第一个成员的实际地址；
3. realloc的改写根据记录的size，调用aligned_alloc，memcpy，free，以确保重新分配的内存8K对齐；
4. 根据8K对齐，判断指针加减法是否跨越bucket，以计算正确的地址；

成员访问改写

Before relayout	After relayout
a->cost	*(cost_t *)(a)
a->flow	*(flow_t *)(a + 8192)
a->org_cost	*(cost_t *)(a + 16384)
a->id	*(int *)(a + 24576)
a->tail	*(uint16_t *)(a + 24580)
a->head	*(uint16_t *)(a + 24582)
a->ident	*(uint16_t *)(a + 32768)

OpenEuler
Compiler SIG

适用场景：

1. 分配在堆里的结构体数组
2. 频繁访问某几个成员
3. 有限次的内存分配

使用限制：

1. realloc确保分配的空间大于原先分配的空间大小
2. relayout会浪费部分内存空间，内存吃紧的场景不建议使用

优化效果：

以SPECCPU2017 intrate 505.mcf为例，性能提升**12%**

Reference:
https://gcc.gnu.org/wiki/GCCSpec2017/mcf

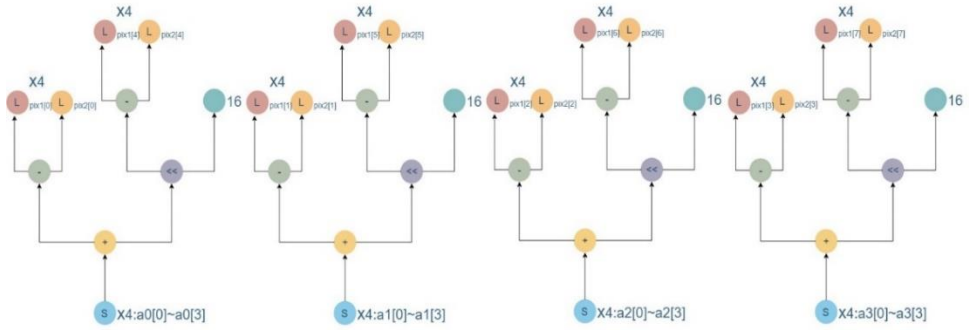
循环拆分-矢量化优化

```
int x264_pixel_satd_8x4( uint8_t *pix1, int i_pix1, uint8_t *pix2, int i_pix2 )
{
    uint32_t tmp[4][4];
    uint32_t a0, a1, a2, a3;
    int sum = 0;
    for( int i = 0; i < 4; i++, pix1 += i_pix1, pix2 += i_pix2 )
    {
        a0 = (pix1[0] - pix2[0]) + ((pix1[4] - pix2[4]) << 16);
        a1 = (pix1[1] - pix2[1]) + ((pix1[5] - pix2[5]) << 16);
        a2 = (pix1[2] - pix2[2]) + ((pix1[6] - pix2[6]) << 16);
        a3 = (pix1[3] - pix2[3]) + ((pix1[7] - pix2[7]) << 16);
        int t0 = a0 + a1;
        int t1 = a0 - a1;
        int t2 = a2 + a3;
        int t3 = a2 - a3;
        tmp[i][0] = t0 + t2;
        tmp[i][2] = t0 - t2;
        tmp[i][1] = t1 + t3;
        tmp[i][3] = t1 - t3;
    }
    ...
}
```

STEP1：loop-distribution

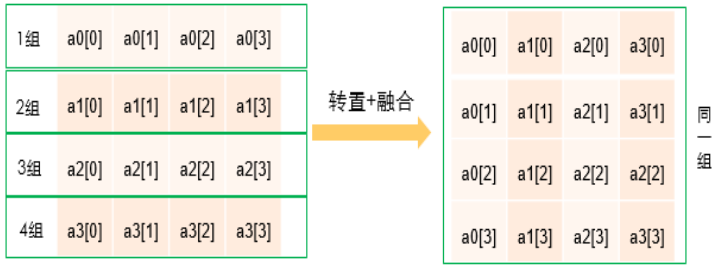
```
loop1:
for( int i = 0; i < 4; i++, pix1 += i_pix1, pix2 += i_pix2 )
    {
        a00[i] = (pix1[0] - pix2[0]) + ((pix1[4] - pix2[4]) << 16);
        a11[i] = (pix1[1] - pix2[1]) + ((pix1[5] - pix2[5]) << 16);
        a22[i] = (pix1[2] - pix2[2]) + ((pix1[6] - pix2[6]) << 16);
        a33[i] = (pix1[3] - pix2[3]) + ((pix1[7] - pix2[7]) << 16);
    }

loop2:
for( int i = 0; i < 4; i++)
    {
        int t0 = a0[i] + a1[i];
        int t1 = a0[i] - a1[i];
        int t2 = a2[i] + a3[i];
        int t3 = a2[i] - a3[i];
        tmp[i][0] = t0 + t2;
        tmp[i][2] = t0 - t2;
        tmp[i][1] = t1 + t3;
        tmp[i][3] = t1 - t3;
    }
```



slp 分析过程

STEP2: transposed slp analysis



OpenEuler Compiler SIG

主要思想：

1. 分析循环中grouped_storeload，分析同构的计算形式，插入临时数组；
2. 矢量化时，将相似的grouped_store进行转置，并融合为一组进行slp分析；
3. 成功矢量化后，依据融合的改写，将数据写入正确的内存；

优化效果：

以SPECCPU2017 intrate

525.x264_r为例，性能提升**5%**

array-widen-compare优化

OpenEuler Compiler SIG

```
#define my_min(x, y) ((x) < (y) ? (x) : (y))
uint32_t
func (uint32_t n0, uint32_t n1, const uint32_t limit, const uint8_t * a,
const uint8_t * b)
{
    uint32_t n = my_min(n0, n1);
    while (++n != limit)
        if (a[n] != b[n])
            break;
    return n;
}
```

```
#define my_min(x, y) ((x) < (y) ? (x) : (y))
uint32_t
func (uint32_t n0, uint32_t n1, const uint32_t limit, const uint8_t *
a, const uint8_t * b)
{
    uint32_t n = my_min(n0, n1);
    for (++n; n + sizeof(uint64_t) <= limit; n += sizeof(uint64_t))
    {
        uint64_t k1 = *((uint64_t*)(a+n));
        uint64_t k2 = *((uint64_t*)(b+n));
        if(k1 != k2)
        {
            int lz = __builtin_ctzll(k1 ^ k2);
            n += lz/8;
            return n;
        }
    }
    for (;n != limit; ++n)
    {
        if (a[n] != b[n])
            break;
    }
    return n;
}
```

主要思想：
用宽数据类型对原数组指针（指向的数组元素为窄类型）解引用，达到一次比较多个元素的效果

优化效果：
SPECCPU2017子项557提升约**7%**

适用场景：
循环存在多出口、数组比较的出口条件为等于或不等于、循环形式简单等

ccmp优化

OpenEuler Compiler SIG

```
int func(int a, int b, int c)
{
  while(1)
  {
     if(a--==0 || b>=c)
     {
         return 1;
      }
   }
}
```

编译命令：
gcc –O1 –S –o test.s test.c

```
func:
.LFB0:
        .cfi_startproc
.L2:
        cmp     w1, w2
        cset    w3, ge
        cmp     w0, 0
        cset    w4, eq
        orr     w3, w3, w4
        sub     w0, w0, #1
        cbz     w3, .L2
        mov     w0, 1
        ret
        .cfi_endproc
.LFE0:
```

优化背景：
在arm64支持ccmp指令，对于上述场景是存在使用ccmp指令的机会

优化效果：
从右图可以看出，经过优化之后在arm64平台上能够使能ccmp指令。SPECCPU2017子项557提升约**1%**

```
func:
.LFB0:
        .cfi_startproc
        b       .L2
.L3:
        mov     w0, w3
.L2:
        sub     w3, w0, #1
        cmp     w0, 0
        ccmp    w1, w2, 0, ne
        blt     .L3
        mov     w0, 1
        ret
```

适用场景：
仅支持arm64


OpenEuler | Compiler SIG