* + - 1. Рабочая среда: компьютер 64 разрядной ОС   
         Процессор: AMD FX(tm)-8350 Eight-Core Processor 4.01 GHz.

1. Версии компиляторов: GCC – 64-7.1.0, Clang –V6.0.0 , Visual C – VS2015 x64.

1) Сравнение ехе файлов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компилятор | | Visual C | GCC | Clang |
| Без оптимизации | Время мкс | 178,51 | 551,91 | 154,45 |
| Размер Кб | 114,00 | 56,07 | 97,00 |
| С оптимизацией по скорости работы | Время мкс | 99,56 | 105,36 | 87,97 |
| Размер Кб | 113,00 | 55,17 | 76,50 |
| С оптимизацией по объёму | Время мкс | 117,00 | 128,78 | 127,56 |
| Размер Кб | 111,00 | 55,07 | 73,55 |

Как видно из таблицы Сlang лучше всего оптимизирует по времени, в то время как GCC – по памяти

1. 2) Сравнение кода ассемблера

|  |  |  |  |
| --- | --- | --- | --- |
| **Visual C** |  |  |  |
| Код на С | Код ассемблера | Код ассемблера (опт по врем) | Пояснения |
| Размножение констант и копий | | | |
| j4 = 2; | ; Line 58 | ; Line 62 | Игнорирование заведомо ложных условий и совместных с ними, а так же лишних безусловных присваиваний |
| if( i2 < j4 && i4 < j4 ) | mov DWORD PTR j4, 2 | mov eax, DWORD PTR k5 |
| i2 = 2; | ; Line 59 | mov DWORD PTR j4, eax |
| j4 = k5; | mov eax, DWORD PTR j4 |  |
| if( i2 < j4 && i4 < j4 ) | cmp DWORD PTR i2, eax |  |
| i5 = 3; | jge SHORT $LN14@main |  |
|  | mov eax, DWORD PTR j4 |  |
|  | cmp DWORD PTR i4, eax |  |
|  | jge SHORT $LN14@main |  |
|  | ; Line 60 |  |
|  | mov DWORD PTR i2, 2 |  |
|  | $LN14@main: |  |
|  | ; Line 62 |  |
|  | mov eax, DWORD PTR k5 |  |
|  | mov DWORD PTR j4, eax |  |
|  | ; Line 63 |  |
|  | mov eax, DWORD PTR j4 |  |
|  | cmp DWORD PTR i2, eax |  |
|  | jge SHORT $LN15@main |  |
|  | mov eax, DWORD PTR j4 |  |
|  | cmp DWORD PTR i4, eax |  |
|  | jge SHORT $LN15@main |  |
|  | ; Line 64 |  |
|  | mov DWORD PTR i5, 3 |  |
| Свертка констант, арифметические тождества и излишние операции загрузки/сохранения | | | |
| i3 = 1 + 2; | ; Line 71 | ; Line 72 | сумма 1 и 2 заменяется сразу на 3, замена бесполезных операций таких как +0 или /1 на обычное присваивание, копии операций также убираются |
| flt\_1 = 2.4 + 6.3; | mov DWORD PTR i3, 3 | movsd xmm0, QWORD PTR \_\_real@4021666666666666 |
| i2 = 5; | ; Line 72 | xorps xmm3, xmm3 |
| j2 = i + 0; | movsd xmm0, QWORD PTR \_\_real@4021666666666666 | ; Line 85 |
| k2 = i / 1; | movsd QWORD PTR flt\_1, xmm0 | movsd xmm4, QWORD PTR \_\_real@3ff0000000000000 |
| i4 = i \* 1; | ; Line 73 | xor edi, edi |
| i5 = i \* 0; | mov DWORD PTR i2, 5 | ; Line 90 |
| #ifndef NO\_ZERO\_DIVIDE | ; Line 74 | movsd xmm2, QWORD PTR \_\_real@4003333333333333 |
| i2 = i / 0; | mov eax, DWORD PTR i | ; Line 107 |
| flt\_2 = flt\_1 / 0.0; | mov DWORD PTR j2, eax | mov ecx, edi |
| #else | ; Line 75 | movsd xmm1, QWORD PTR flt\_6 |
| printf( "This compiler handles divide-by-zero as \an error\n"); | mov eax, DWORD PTR i | mov eax, DWORD PTR Imov DWORD PTR j2, eax |
| #endif | mov DWORD PTR k2, eax | mov DWORD PTR i4, eax |
| flt\_3 = 2.4 / 1.0; | ; Line 76 | divsd xmm4, xmm3 |
| flt\_4 = 1.0 + 0.0000001; | mov eax, DWORD PTR i | movsxd r10, eax |
| flt\_5 = flt\_6 \* 0.0; | mov DWORD PTR i4, eax | mov rdx, rsi |
| flt\_6 = flt\_2 \* flt\_3; | ; Line 77 | mov eax, DWORD PTR j5 |
|  | imul eax, DWORD PTR i, 0 | mov DWORD PTR i3, 3 |
|  | mov DWORD PTR i5, eax | mov DWORD PTR i2, r10d |
|  | ; Line 73 | lea eax, DWORD PTR [rax\*4] |
|  | mov DWORD PTR i2, 5 | mov DWORD PTR k2, eax |
|  | ; Line 74 | mulsd xmm1, xmm3 |
|  | mov eax, DWORD PTR i | mulsd xmm4, xmm0 |
|  | mov DWORD PTR j2, eax | movsd QWORD PTR flt\_1, xmm0 |
|  | ; Line 75 | movsd xmm0, QWORD PTR \_\_real@3ff000001ad7f29b |
|  | mov eax, DWORD PTR i | movsd QWORD PTR flt\_3, xmm2 |
|  | mov DWORD PTR k2, eax | movsd QWORD PTR flt\_2, xmm4 |
|  | ; Line 76 | mulsd xmm4, xmm2 |
|  | mov eax, DWORD PTR i | movsd QWORD PTR flt\_4, xmm0 |
|  | mov DWORD PTR i4, eax | movsd QWORD PTR flt\_5, xmm1 |
|  | ; Line 77 | movsd QWORD PTR flt\_6, xmm4 |
|  | imul eax, DWORD PTR i, 0 | npad 6 |
|  | mov DWORD PTR i5, eax |  |
|  | ; Line 84 |  |
|  | mov eax, DWORD PTR i |  |
|  | cdq |  |
|  | xor ecx, ecx |  |
|  | idiv ecx |  |
|  | mov DWORD PTR i2, eax |  |
|  | ; Line 85 |  |
|  | movsd xmm0, QWORD PTR flt\_1 |  |
|  | divsd xmm0, QWORD PTR \_\_real@0000000000000000 |  |
|  | movsd QWORD PTR flt\_2, xmm0 |  |
|  | ; Line 90 |  |
|  | movsd xmm0, QWORD PTR \_\_real@4003333333333333 |  |
|  | movsd QWORD PTR flt\_3, xmm0 |  |
|  | ; Line 91 |  |
|  | movsd xmm0, QWORD PTR \_\_real@3ff000001ad7f29b |  |
|  | movsd QWORD PTR flt\_4, xmm0 |  |
|  | ; Line 92 |  |
|  | movsd xmm0, QWORD PTR flt\_6 |  |
|  | mulsd xmm0, QWORD PTR \_\_real@0000000000000000 |  |
|  | movsd QWORD PTR flt\_5, xmm0 |  |
|  | ; Line 93 |  |
|  | movsd xmm0, QWORD PTR flt\_2 |  |
|  | mulsd xmm0, QWORD PTR flt\_3 |  |
|  | movsd QWORD PTR flt\_6, xmm0 |  |
|  |  |  |
| Лишнее присваивание | | | |
| k3 = 1; | ; Line 99 | mov DWORD PTR k3, 1 | Игнорирование лишнего присваивания, присваивание напрямую через регистр |
| k3 = 1; | mov DWORD PTR k3, 1 |
|  | ; Line 100 |
|  | mov DWORD PTR k3, 1 |
|  |  |
|  |  |
|  |  |
|  |  |
| Снижение мощности | | | |
| k2 = 4 \* j5; | ; Line 106 | lea rsi, OFFSET FLAT:ivector4 | вместо переменных используются регистры |
| for( i = 0; i <= 5; i++ ) ivector4[ i ] = i \* 2; | mov eax, DWORD PTR j5 | mov rdx, rsi |
|  | shl eax, 2 | lea eax, DWORD PTR [rax\*4] |
|  | mov DWORD PTR k2, eax | mov DWORD PTR k2, eax |
|  | ; Line 107 | $LL4@main: |
|  | mov DWORD PTR i, 0 | ; Line 108 |
|  | jmp SHORT $LN4@main | movzx eax, cx |
|  | $LN2@main: | lea rdx, QWORD PTR [rdx+2] |
|  | mov eax, DWORD PTR i | add ax, ax |
|  | inc eax | inc ecx |
|  | mov DWORD PTR i, eax | mov WORD PTR [rdx-2], ax |
|  | $LN4@main: | cmp ecx, 5 |
|  | cmp DWORD PTR i, 5 | jle SHORT $LL4@main |
|  | jg SHORT $LN3@main |  |
|  | ; Line 108 |  |
|  | mov eax, DWORD PTR i |  |
|  | add eax, eax |  |
|  | movsxd rcx, DWORD PTR i |  |
|  | lea rdx, OFFSET FLAT:ivector4 |  |
|  | mov WORD PTR [rdx+rcx\*2], ax |  |
|  | jmp SHORT $LN2@main |  |
|  |  |  |
| Простой цикл | | | |
| j5 = 0; | ; Line 114 | ; Line 114 | Избавление от лишнего присваивания в переменные во время вычисления, вычисления проходят в регистрах, некоторые не оптимальное умножение (умножение на переменную изменившуюся на 1) заменено на сложение, так же как деление на вычитание |
| k5 = 10000; | mov DWORD PTR j5, 0 | mov r9d, 10000 |
| do { | ; Line 115 | mov ecx, 30000 |
| k5 = k5 - 1; | mov DWORD PTR k5, 10000 ; 00002710H | mov DWORD PTR j5, r9d |
| j5 = j5 + 1; | $LN7@main: | mov r8d, edi |
| i5 = (k5 \* 3) / (j5 \* constant5); | ; Line 117 | npad 6 |
| } while ( k5 > 0 ); | mov eax, DWORD PTR k5 | $LL7@main: |
|  | dec eax | ; Line 117 |
|  | mov DWORD PTR k5, eax | sub ecx, 3 |
|  | ; Line 118 | ; Line 118 |
|  | mov eax, DWORD PTR j5 | add r8d, 5 |
|  | inc eax | ; Line 119 |
|  | mov DWORD PTR j5, eax | mov eax, ecx |
|  | ; Line 119 | cdq |
|  | imul eax, DWORD PTR k5, 3 | idiv r8d |
|  | imul ecx, DWORD PTR j5, 5 | mov DWORD PTR i5, eax |
|  | cdq | ; Line 120 |
|  | idiv ecx | test ecx, ecx |
|  | mov DWORD PTR i5, eax | jg SHORT $LL7@main |
|  | ; Line 120 |  |
|  | cmp DWORD PTR k5, 0 |  |
|  | jg SHORT $LN7@main |  |
|  |  |  |
| Управление переменной индукции цикла | | | |
| for( i = 0; i < 100; i++ ) ivector5[ i \* 2 + 3 ] = 5; | ; Line 125 | ; Line 125 | Избавление от вычисления заведомо известного шага внутри цикла |
| mov DWORD PTR i, 0 | mov edx, 100 |
| jmp SHORT $LN10@main | lea rcx, OFFSET FLAT:ivector5+12 |
| $LN8@main: | mov DWORD PTR i, edx |
| mov eax, DWORD PTR i | mov ebx, 5 |
| inc eax | npad 2 |
| mov DWORD PTR i, eax | $LL10@main: |
| $LN10@main: | ; Line 126 |
| cmp DWORD PTR i, 100jge SHORT $LN9@main | mov DWORD PTR [rcx], ebx |
| ; Line 126 | lea rcx, QWORD PTR [rcx+8] |
| mov eax, DWORD PTR i | sub rdx, 1 |
| lea eax, DWORD PTR [rax+rax+3] | jne SHORT $LL10@main |
| cdqe |  |
| lea rcx, OFFSET FLAT:ivector5 |  |
| mov DWORD PTR [rcx+rax\*4], 5 |  |
| jmp SHORT $LN8@main |  |
|  |  |
| Глубокие подвыражения | | | |
| if( i < 10 ) | ; Line 132 | mov DWORD PTR i2, r10d | избавление от заведомо ложных условий, и примыкающих к ним операций |
| j5 = i5 + i2; | cmp DWORD PTR i, 10 | add eax, r10d |
| else | jge SHORT $LN16@main | mov DWORD PTR k5, eax |
| k5 = i5 + i2; | ; Line 133 |  |
|  | mov eax, DWORD PTR i2 |  |
|  | mov ecx, DWORD PTR i5 |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | mov DWORD PTR j5, eax |  |
|  | jmp SHORT $LN17@main |  |
|  | $LN16@main: |  |
|  | ; Line 135 |  |
|  | mov eax, DWORD PTR i2 |  |
|  | mov ecx, DWORD PTR i5 |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | mov DWORD PTR k5, eax |  |
|  |  |  |
| Проверка того, как компилятор генерирует адреспеременной с константным индексом, размножает копии и регистры | | | |
| ivector[ 0 ] = 1 ivector[ i2 ] = 2; ivector[ i2 ] = 2; ivector[ 2 ] = 3; | ; Line 143 | ; Line 144 | избавление от рассчета значений адресов ни коим образом не изменяющих начальный адрес, более оптимальный рассчет адреса изменяемого, использование регистра напрямую вместо переменной |
| mov eax, 4 | lea rcx, OFFSET FLAT:ivector |
| imul rax, rax, 0 | mov DWORD PTR ivector, 1 |
| lea rcx, OFFSET FLAT:ivector | ; Line 145 |
| mov DWORD PTR [rcx+rax], 1 | mov DWORD PTR [rcx+r10\*4], 2 |
| ; Line 144 | ; Line 153 |
| movsxd rax, DWORD PTR i2 | mov DWORD PTR ivector+8, 3 |
| lea rcx, OFFSET FLAT:ivector |  |
| mov DWORD PTR [rcx+rax\*4], 2 |  |
| ; Line 145 |  |
| movsxd rax, DWORD PTR i2 |  |
| lea rcx, OFFSET FLAT:ivector |  |
| mov DWORD PTR [rcx+rax\*4], 2 |  |
| ; Line 146 |  |
| mov eax, 4 |  |
| imul rax, rax, 2 |  |
| lea rcx, OFFSET FLAT:ivector |  |
| mov DWORD PTR [rcx+rax], 3 |  |
| Удаление общих подвыражений | | | |
| if(( h3 + k3 ) < 0 || ( h3 + k3 ) > 5 ) printf("Common subexpression elimination\n"); | ; Line 153 | ; Line 153 | то же избавление от заведомо ложных условий присваивание сначала в переменную g3 а после в m3 ибо так оптимальнее |
| else { | mov eax, DWORD PTR k3 | mov ecx, DWORD PTR h3 |
| m3 = ( h3 + k3 ) / i3; | mov ecx, DWORD PTR h3 | inc ecx |
| g3 = i3 + (h3 + k3); | add ecx, eax | cmp ecx, ebx |
|  | mov eax, ecx | ja SHORT $LN20@main |
|  | test eax, eax | ; Line 156 |
|  | jl SHORT $LN20@main | mov eax, 1431655766 ; 55555556H |
|  | mov eax, DWORD PTR k3 | imul ecx |
|  | mov ecx, DWORD PTR h3 | mov eax, edx |
|  | add ecx, eax | shr eax, 31 |
|  | mov eax, ecx | add edx, eax |
|  | cmp eax, 5 | ; Line 157 |
|  | jle SHORT $LN18@main | lea eax, DWORD PTR [rcx+3] |
|  | $LN20@main: | mov DWORD PTR g3, eax |
|  | ; Line 154 | mov DWORD PTR m3, edx |
|  | lea rcx, OFFSET FLAT:$SG5360 | jmp SHORT $LN19@main |
|  | call printf | $LN20@main: |
|  | jmp SHORT $LN19@main | ; Line 154 |
|  | $LN18@main: | lea rcx, OFFSET FLAT:??\_C@\_0CC @DDFJGIGO @Common? 5subexpression ?5elimination@ |
|  | ; Line 156 | call printf |
|  | mov eax, DWORD PTR k3 |  |
|  | mov ecx, DWORD PTR h3 |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | cdq |  |
|  | idiv DWORD PTR i3 |  |
|  | mov DWORD PTR m3, eax |  |
|  | ; Line 157 |  |
|  | mov eax, DWORD PTR k3 |  |
|  | mov ecx, DWORD PTR h3 |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | mov ecx, DWORD PTR i3 |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | mov DWORD PTR g3, eax |  |
| Вынесение инвариантного кода (j \* k) может быть вынесено из цикла | | | |
| for( i4 = 0; i4 <= max\_vector; i4++) | ; Line 165 | ; Line 181 | j\*k - код который выполняется на каждой интерации цикла выносится из него и считается 1 раз вне цикла, после происходит три присваивания вместо цикла |
| ivector2[ i4 ]= j\*k; | mov DWORD PTR i4, 0 | movzx eax, BYTE PTR k |
|  | jmp SHORT $LN13@main | movzx ecx, BYTE PTR j |
|  | $LN11@main: | imul ecx, eax |
|  | mov eax, DWORD PTR i4 | mov DWORD PTR i4, 3 |
|  | inc eax | mov BYTE PTR ivector2, cl |
|  | mov DWORD PTR i4, eax | mov BYTE PTR ivector2+1, cl |
|  | $LN13@main: | mov BYTE PTR ivector2+2, cl |
|  | cmp DWORD PTR i4, 2 | npad 5 |
|  | jg SHORT $LN12@main |  |
|  | ; Line 166 |  |
|  | mov eax, DWORD PTR j |  |
|  | imul eax, DWORD PTR k |  |
|  | movsxd rcx, DWORD PTR i4 |  |
|  | lea rdx, OFFSET FLAT:ivector2 |  |
|  | mov BYTE PTR [rdx+rcx], al |  |
|  | jmp SHORT $LN11@main |  |
|  |  |  |
| Функция: dead\_code | | | |
| void dead\_code( a, b ) | dead\_code PROC | dead\_code PROC ; COMDAT | Так как этот участок кода не несет в себе никакой смысловой нагрузки для выполнения всей программы в целом, то он игнорируется компилятором |
| int a; | ; File c:\users\Sanek\desktop\Папка10\optbench.c | ;File c:\users\Sanek\desktop\Папка10\optbench.c |
| char \*b; | ; Line 197 | ; Line 203 |
| { | $LN4: | ret 0 |
| int idead\_store; idead\_store = a; if( 0 ) | mov QWORD PTR [rsp+16], rdx | dead\_code ENDP |
| printf( "%s\n", b ); | mov DWORD PTR [rsp+8], ecx |  |
| } | sub rsp, 56 ; 00000038H |  |
|  | ; Line 200 |  |
|  | mov eax, DWORD PTR a$[rsp] |  |
|  | mov DWORD PTR idead\_store$[rsp], eax |  |
|  | ; Line 201 |  |
|  | xor eax, eax |  |
|  | test eax, eax |  |
|  | je SHORT $LN2@dead\_code |  |
|  | ; Line 202 |  |
|  | mov rdx, QWORD PTR b$[rsp] |  |
|  | lea rcx, OFFSET FLAT:$SG5368 |  |
|  | call printf |  |
|  | $LN2@dead\_code: |  |
|  | ; Line 203 |  |
|  | add rsp, 56 ; 00000038H |  |
|  | ret 0 |  |
|  | dead\_code ENDP |  |
|  |  |  |
| Функция: unnecessary\_loop | | | |
| void unnecessary\_loop() |  | unnecessary\_loop PROC ; COMDAT | В каждой итерации цикла происходит одно и то же, а посему этот цикл убирается, в переменные кладутся лишь те значения которые были бы в этих переменных в конце последней интерации |
| { | unnecessary\_loop PROC | ; File c:\users\Sanek\desktop\Папка10\optbench.c |
| int x; | ; File c:\users\Sanek\desktop\Папка10\optbench.c | ; Line 220 |
| x = 0; | ; Line 214 | mov eax, DWORD PTR j5 |
| for( i = 0; i < 5; i++ ) k5 = x + j5; | $LN6: | mov DWORD PTR k5, eax |
| } | sub rsp, 24 | mov DWORD PTR i, 5 |
|  | ; Line 217 | ; Line 221 |
|  | mov DWORD PTR x$[rsp], 0 | ret 0 |
|  | ; Line 218 | unnecessary\_loop ENDP |
|  | mov DWORD PTR i, 0 |  |
|  | jmp SHORT $LN4@unnecessar |  |
|  | $LN2@unnecessar: |  |
|  | mov eax, DWORD PTR i |  |
|  | inc eax |  |
|  | mov DWORD PTR i, eax |  |
|  | $LN4@unnecessar: |  |
|  | cmp DWORD PTR i, 5 |  |
|  | jge SHORT $LN3@unnecessar |  |
|  | ; Line 220 |  |
|  | mov eax, DWORD PTR j5 |  |
|  | mov ecx, DWORD PTR x$[rsp] |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | mov DWORD PTR k5, eax |  |
|  | jmp SHORT $LN2@unnecessar |  |
|  | $LN3@unnecessar: |  |
|  | ; Line 221 |  |
|  | add rsp, 24 |  |
|  | ret 0 |  |
|  | unnecessary\_loop ENDP |  |
|  |  |  |
| Функция: loop\_jamming | | | |
| void loop\_jamming( x ) | loop\_jamming PROC | loop\_jamming PROC ; COMDAT | Циклы с одинаковыми условиями и бегущими переменным сливаются в один цикл |
| int x; | ; File c:\users\Sanek\desktop\Папка10\optbench.c | ; File c:\users\Sanek\desktop\Папка10\optbench.c |
| { | ; Line 232 | ; Line 233 |
| for( i = 0; i < 5; i++ ) | mov DWORD PTR [rsp+8], ecx | mov r10d, DWORD PTR j5 |
| k5 = x + j5 \* i; | ; Line 233 | mov r8d, 5 |
| for( i = 0; i < 5; i++ ) | mov DWORD PTR i, 0 | mov r9d, r8d |
| i5 = x \* k5 \* i; | jmp SHORT $LN4@loop\_jammi | mov eax, ecx |
| } | $LN2@loop\_jammi: | npad 14 |
|  | mov eax, DWORD PTR i | $LL4@loop\_jammi: |
|  | inc eax | ; Line 234 |
|  | mov DWORD PTR i, eax | mov edx, eax |
|  | $LN4@loop\_jammi: | mov DWORD PTR k5, eax |
|  | cmp DWORD PTR i, 5 | add eax, r10d |
|  | jge SHORT $LN3@loop\_jammi | sub r9, 1 |
|  | ; Line 234 | jne SHORT $LL4@loop\_jammi |
|  | mov eax, DWORD PTR j5 | ; Line 235 |
|  | imul eax, DWORD PTR i | xor eax, eax |
|  | mov ecx, DWORD PTR x$[rsp] | mov DWORD PTR i, r8d |
|  | add ecx, eax | imul edx, ecx |
|  | mov eax, ecx | npad 3 |
|  | mov DWORD PTR k5, eax | $LL7@loop\_jammi: |
|  | jmp SHORT $LN2@loop\_jammi | ; Line 236 |
|  | $LN3@loop\_jammi: | mov DWORD PTR i5, eax |
|  | mov DWORD PTR i, 0 | add eax, edx |
|  | jmp SHORT $LN7@loop\_jammi | sub r8, 1 |
|  | $LN5@loop\_jammi: | jne SHORT $LL7@loop\_jammi |
|  | mov eax, DWORD PTR i | ; Line 237 |
|  | inc eax | ret 0 |
|  | mov DWORD PTR i, eax | loop\_jamming ENDP |
|  | $LN7@loop\_jammi: |  |
|  | cmp DWORD PTR i, 5 |  |
|  | jge SHORT $LN6@loop\_jammi |  |
|  | ; Line 236 |  |
|  | mov eax, DWORD PTR x$[rsp] |  |
|  | imul eax, DWORD PTR k5 |  |
|  | imul eax, DWORD PTR i |  |
|  | mov DWORD PTR i5, eax |  |
|  | jmp SHORT $LN5@loop\_jammi |  |
|  | $LN6@loop\_jammi: |  |
|  | ; Line 237 |  |
|  | ret 0 |  |
|  | loop\_jamming ENDP |  |
| Функция: loop\_unrolling | | | |
| void loop\_unrolling( x ) | loop\_unrolling PROC | loop\_unrolling PROC ; COMDAT | оптимизации нет |
| int x; | ; File c:\users\Sanek\desktop\Папка10\optbench.c | ; File c:\users\Sanek\desktop\Папка10\optbench.c |
| { | ; Line 250 | ; Line 250 |
| for( i = 0; i < 6; i++ ) ivector4[ i ] = 0; | $LN8: | $LN14: |
| } | mov DWORD PTR [rsp+8], ecx | sub rsp, 40 ; 00000028H |
|  | sub rsp, 56 ; 00000038H | ; Line 251 |
|  | ; Line 251 | xor edx, edx |
|  | mov DWORD PTR i, 0 | lea r8, OFFSET FLAT:ivector4 |
|  | jmp SHORT $LN4@loop\_unrol | mov ecx, edx |
|  | $LN2@loop\_unrol: | mov DWORD PTR i, edx |
|  | mov eax, DWORD PTR i | npad 11 |
|  | inc eax | $LL4@loop\_unrol: |
|  | mov DWORD PTR i, eax | ; Line 252 |
|  | $LN4@loop\_unrol: | movsxd rax, ecx |
|  | cmp DWORD PTR i, 6 | add rax, rax |
|  | jge SHORT $LN3@loop\_unrol | cmp rax, 12 |
|  | ; Line 252 | jae SHORT $LN13@loop\_unrol |
|  | movsxd rax, DWORD PTR i | inc ecx |
|  | shl rax, 1 | mov WORD PTR [rax+r8], dx |
|  | mov QWORD PTR $T1[rsp], rax | mov DWORD PTR i, ecx |
|  | cmp QWORD PTR $T1[rsp], 12 | cmp ecx, 6 |
|  | jae SHORT $LN6@loop\_unrol | jl SHORT $LL4@loop\_unrol |
|  | jmp SHORT $LN7@loop\_unrol | ; Line 253 |
|  | $LN6@loop\_unrol: | add rsp, 40 ; 00000028H |
|  | call \_\_report\_rangecheckfailure | ret 0 |
|  | $LN7@loop\_unrol: | $LN13@loop\_unrol: |
|  | lea rax, OFFSET FLAT:ivector4 | ; Line 252 |
|  | xor ecx, ecx | call \_\_report\_rangecheckfailure |
|  | mov rdx, QWORD PTR $T1[rsp] | int 3 |
|  | mov WORD PTR [rax+rdx], cx | $LN12@loop\_unrol: |
|  | jmp SHORT $LN2@loop\_unrol | loop\_unrolling ENDP |
|  | $LN3@loop\_unrol: |  |
|  | $LN5@loop\_unrol: |  |
|  | ; Line 253 |  |
|  | add rsp, 56 ; 00000038H |  |
|  | ret 0 |  |
|  | loop\_unrolling ENDP |  |
|  |  |  |
| Функция: jump\_compression | | | |
| int jump\_compression( i, j, k, l, m ) | jump\_compression PROC | jump\_compression PROC ; COMDAT | более простое сжатие условий |
| int i, j, k, l, m; | ; File c:\users\Sanek\desktop\Папка10\optbench.c | ; File c:\users\Sanek\desktop\Папка10\optbench.c |
| { | ; Line 264 | ; Line 266 |
| beg\_1: | mov DWORD PTR [rsp+32], r9d | mov r10d, DWORD PTR m$[rsp] |
| if( i < j ) if( j < k ) if( k < l ) if( l < m ) l += m; else goto end\_1; else k += l; else { j += k; | mov DWORD PTR [rsp+24], r8d | $end\_1$18: |
| end\_1: goto beg\_1; } | mov DWORD PTR [rsp+16], edx | cmp ecx, edx |
| else | mov DWORD PTR [rsp+8], ecx | jge SHORT $LN15@jump\_compr |
| i += j; return( i + j + k + l + m ); | $beg\_1$11: | npad 7 |
| } | ; Line 266 | $beg\_1$19: |
|  | mov eax, DWORD PTR j$[rsp] | ; Line 267 |
|  | cmp DWORD PTR i$[rsp], eax | cmp edx, r8d |
|  | jge SHORT $LN2@jump\_compr | jge SHORT $LN4@jump\_compr |
|  | ; Line 267 | ; Line 268 |
|  | mov eax, DWORD PTR k$[rsp] | cmp r8d, r9d |
|  | cmp DWORD PTR j$[rsp], eax | jge SHORT $LN6@jump\_compr |
|  | jge SHORT $LN4@jump\_compr | ; Line 269 |
|  | ; Line 268 | cmp r9d, r10d |
|  | mov eax, DWORD PTR l$[rsp] | jge SHORT $end\_1$18 |
|  | cmp DWORD PTR k$[rsp], eax | ; Line 270 |
|  | jge SHORT $LN6@jump\_compr | add r9d, r10d |
|  | ; Line 269 | ; Line 282 |
|  | mov eax, DWORD PTR m$[rsp] | lea eax, DWORD PTR [rcx+rdx] |
|  | cmp DWORD PTR l$[rsp], eax | add eax, r8d |
|  | jge SHORT $LN8@jump\_compr | add eax, r9d |
|  | ; Line 270 | add eax, r10d |
|  | mov eax, DWORD PTR m$[rsp] | ; Line 283 |
|  | mov ecx, DWORD PTR l$[rsp] | ret 0 |
|  | add ecx, eax | $LN4@jump\_compr: |
|  | mov eax, ecx | ; Line 276 |
|  | mov DWORD PTR l$[rsp], eax | add edx, r8d |
|  | jmp SHORT $LN9@jump\_compr | ; Line 266 |
|  | $LN8@jump\_compr: | jmp SHORT $end\_1$18 |
|  | ; Line 272 | $LN6@jump\_compr: |
|  | jmp SHORT $end\_1$12 | ; Line 274 |
|  | $LN9@jump\_compr: | add r8d, r9d |
|  | jmp SHORT $LN7@jump\_compr | ; Line 282 |
|  | $LN6@jump\_compr: | lea eax, DWORD PTR [rcx+rdx] |
|  | ; Line 274 | add eax, r8d |
|  | mov eax, DWORD PTR l$[rsp] | add eax, r9d |
|  | mov ecx, DWORD PTR k$[rsp] | add eax, r10d |
|  | add ecx, eax | ; Line 283 |
|  | mov eax, ecx | ret 0 |
|  | mov DWORD PTR k$[rsp], eax | $LN15@jump\_compr: |
|  | $LN7@jump\_compr: | ; Line 281 |
|  | jmp SHORT $LN5@jump\_compr | add ecx, edx |
|  | $LN4@jump\_compr: | ; Line 282 |
|  | ; Line 276 | lea eax, DWORD PTR [rcx+rdx] |
|  | mov eax, DWORD PTR k$[rsp] | add eax, r8d |
|  | mov ecx, DWORD PTR j$[rsp] | add eax, r9d |
|  | add ecx, eax | add eax, r10d |
|  | mov eax, ecx | ; Line 283 |
|  | mov DWORD PTR j$[rsp], eax | ret 0 |
|  | $end\_1$12: | jump\_compression ENDP |
|  | ; Line 278 |  |
|  | jmp SHORT $beg\_1$11 |  |
|  | $LN5@jump\_compr: |  |
|  | ; Line 279 |  |
|  | jmp SHORT $LN3@jump\_compr |  |
|  | $LN2@jump\_compr: |  |
|  | ; Line 281 |  |
|  | mov eax, DWORD PTR j$[rsp] |  |
|  | mov ecx, DWORD PTR i$[rsp] |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | mov DWORD PTR i$[rsp], eax |  |
|  | $LN3@jump\_compr: |  |
|  | ; Line 282 |  |
|  | mov eax, DWORD PTR j$[rsp] |  |
|  | mov ecx, DWORD PTR i$[rsp] |  |
|  | add ecx, eax |  |
|  | mov eax, ecx |  |
|  | add eax, DWORD PTR k$[rsp] |  |
|  | add eax, DWORD PTR l$[rsp] |  |
|  | add eax, DWORD PTR m$[rsp] |  |
|  | ; Line 283 |  |
|  | ret 0 |  |
|  | jump\_compression ENDP |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **GCC** |  |  |  |
| Код на С | Код ассемблера | Код ассемблера (опт по врем) | Пояснения |
| Размножение констант и копий | | | |
| j4 = 2; | leaq j4(%rip), %rax | movl k5(%rip), %eax | Игнорирование заведомо ложных условий и совместных с ними а так же лишних безусловных присваиваний |
| if( i2 < j4 && i4 < j4 ) | movl $2, (%rax) | movl %eax, j4(%rip) |
| i2 = 2; | leaq i2(%rip), %rax |  |
| j4 = k5; | movl (%rax), %edx |  |
| if( i2 < j4 && i4 < j4 ) | leaq j4(%rip), %rax |  |
| i5 = 3; | movl (%rax), %eax |  |
|  | cmpl %eax, %edx |  |
|  | jge .L2 |  |
|  | leaq i4(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq j4(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl %eax, %edx |  |
|  | jge .L2 |  |
|  | leaq i2(%rip), %rax |  |
|  | movl $2, (%rax) |  |
|  | .L2: |  |
|  | leaq k5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq j4(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq i2(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq j4(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl %eax, %edx |  |
|  | jge .L3 |  |
|  | leaq i4(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq j4(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl %eax, %edx |  |
|  | jge .L3 |  |
|  | leaq i5(%rip), %rax |  |
|  | movl $3, (%rax) |  |
|  |  |  |
| Свертка констант, арифметические тождества и излишние операции загрузки/сохранения | | |  |
| i3 = 1 + 2; | leaq i3(%rip), %rax | xorpd %xmm2, %xmm2 | сумма 1 и 2 заменяется сразу на 3, замена бесполезных операций таких как +0 или /1 на обычное присваивание, копии операций также убираются |
| flt\_1 = 2.4 + 6.3; | movl $3, (%rax) | movsd .LC0(%rip), %xmm1 |
| i2 = 5; | leaq flt\_1(%rip), %rdx | xorl %ecx, %ecx |
| j2 = i + 0; | movabs $4621087282649523814, %rax | movsd .LC2(%rip), %xmm0 |
| k2 = i / 1; | movq %rax, (%rdx) | movl $3, i3(%rip) |
| i4 = i \* 1; | leaq i2(%rip), %rax | movsd %xmm1, flt\_1(%rip) |
| i5 = i \* 0; | movl $5, (%rax) | divsd %xmm2, %xmm1 |
| #ifndef NO\_ZERO\_DIVIDE | leaq i(%rip), %rax | movl i(%rip), %eax |
| i2 = i / 0; | movl (%rax), %edx | movsd %xmm0, flt\_3(%rip) |
| flt\_2 = flt\_1 / 0.0; | leaq j2(%rip), %rax | movl %eax, %edx |
| #else | movl %edx, (%rax) | movl %eax, j2(%rip) |
| printf( "This compiler handles divide-by-zero as \an error\n"); | leaq i(%rip), %rax | movl %eax, i4(%rip) |
| #endif | movl (%rax), %edx | sarl $31, %edx |
| flt\_3 = 2.4 / 1.0; | leaq k2(%rip), %rax | idivl %ecx |
| flt\_4 = 1.0 + 0.0000001; | movl %edx, (%rax) | mulsd flt\_6(%rip), %xmm2 |
| flt\_5 = flt\_6 \* 0.0; | leaq i(%rip), %rax | mulsd %xmm1, %xmm0 |
| flt\_6 = flt\_2 \* flt\_3; | movl (%rax), %edx | movsd %xmm1, flt\_2(%rip) |
|  | leaq i4(%rip), %rax | movslq %eax, %r9 |
|  | movl %edx, (%rax) | movabs $4607182419250377371, %rax |
|  | leaq i5(%rip), %rax | movq %rax, flt\_4(%rip) |
|  | movl $0, (%rax) | movsd %xmm2, flt\_5(%rip) |
|  | leaq i(%rip), %rax | movsd %xmm0, flt\_6(%rip) |
|  | movl (%rax), %eax | movl %r9d, i2(%rip) |
|  | movl $0, -4(%rbp) |  |
|  | movl %eax, %edx |  |
|  | sarl $31, %edx |  |
|  | idivl -4(%rbp) |  |
|  | movl %eax, %edx |  |
|  | leaq i2(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq flt\_1(%rip), %rax |  |
|  | movsd (%rax), %xmm0 |  |
|  | xorpd %xmm1, %xmm1 |  |
|  | divsd %xmm1, %xmm0 |  |
|  | leaq flt\_2(%rip), %rax |  |
|  | movsd %xmm0, (%rax) |  |
|  | leaq flt\_3(%rip), %rdx |  |
|  | movabsq $4612586738352862003, %rax |  |
|  | movq %rax, (%rdx) |  |
|  | leaq flt\_4(%rip), %rdx |  |
|  | movabsq $4607182419250377371, %rax |  |
|  | movq %rax, (%rdx) |  |
|  | leaq flt\_6(%rip), %rax |  |
|  | movsd (%rax), %xmm1 |  |
|  | xorpd %xmm0, %xmm0 |  |
|  | mulsd %xmm1, %xmm0 |  |
|  | leaq flt\_5(%rip), %rax |  |
|  | movsd %xmm0, (%rax) |  |
|  | leaq flt\_2(%rip), %rax |  |
|  | movsd (%rax), %xmm1 |  |
|  | leaq flt\_3(%rip), %rax |  |
|  | movsd (%rax), %xmm0 |  |
|  | mulsd %xmm1, %xmm0 |  |
|  | leaq flt\_6(%rip), %rax |  |
|  | movsd %xmm0, (%rax) |  |
|  |  |  |
| Лишнее присваивание | | | |
| k3 = 1; | leaq k3(%rip), %rax | movl $1, k3(%rip) | Игнорирование лишнего присваивания, присваивание напрямую через регистр |
| k3 = 1; | movl $1, (%rax) |
|  | leaq k3(%rip), %rax |
|  | movl $1, (%rax) |
| Снижение мощности | | | |
| k2 = 4 \* j5; | leaq j5(%rip), %rax | movw $0, ivector4(%rip) | Цикл заменен на 5 присваиваний, значение i в конечную интерацию сразу присваивается своей переменной, побитовый сдвиг вместо умножения на числа кратные 2 |
| for( i = 0; i <= 5; i++ ) | movl (%rax), %eax | movw $2, 2+ivector4(%rip) |
| ivector4[ i ] = i \* 2; | leal 0(,%rax,4), %edx | movw $4, 4+ivector4(%rip) |
|  | leaq k2(%rip), %rax | movw $6, 6+ivector4(%rip) |
|  | movl %edx, (%rax) | movw $8, 8+ivector4(%rip) |
|  | leaq i(%rip), %rax | movw $10, 10+ivector4(%rip) |
|  | movl $0, (%rax) | movl $6, i(%rip) |
|  | jmp .L4 | movl j5(%rip), %eax |
|  | .L5: | sall $2, %eax |
|  | leaq i(%rip), %rax | movl %eax, k2(%rip) |
|  | movl (%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | addl %eax, %eax |  |
|  | movl %eax, %ecx |  |
|  | leaq ivector4(%rip), %rax |  |
|  | movslq %edx, %rdx |  |
|  | movw %cx, (%rax,%rdx,2) |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal 1(%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | .L4: |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl $5, %eax |  |
|  | jle .L5 |  |
|  |  |  |
| Простой цикл | | | |
| j5 = 0; | leaq j5(%rip), %rax | L6: | Избавление от лишнего присваивания в переменные во время вычисления, вычисления проходят в регистрах, некоторые не оптимальное умножение (умножение на переменную изменившуюся на 1) заменено на сложение, так же как деление на вычитание |
| k5 = 10000; | movl $0, (%rax) | movl %ecx, %edx |
| do { | leaq k5(%rip), %rax | movl %ecx, %eax |
| k5 = k5 - 1; | movl $10000, (%rax) | subl $3, %ecx |
| j5 = j5 + 1; | .L6: | sarl $31, %edx |
| i5 = (k5 \* 3) / (j5 \* constant5); | leaq k5(%rip), %rax | idivl %r8d |
| } while ( k5 > 0 ); | movl (%rax), %eax | addl $5, %r8d |
|  | leal -1(%rax), %edx | cmpl $-3, %ecx |
|  | leaq k5(%rip), %rax | jne .L6 |
|  | movl %edx, (%rax) | movl $0, k5(%rip) |
|  | leaq j5(%rip), %rax | movl $10000, j5(%rip) |
|  | movl (%rax), %eax | movl %eax, i5(%rip) |
|  | leal 1(%rax), %edx | movl $10000, %edx |
|  | leaq j5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq k5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | movl %edx, %eax |  |
|  | addl %eax, %eax |  |
|  | leal (%rax,%rdx), %ecx |  |
|  | leaq j5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | movl %edx, %eax |  |
|  | sall $2, %eax |  |
|  | addl %edx, %eax |  |
|  | movl %eax, -4(%rbp) |  |
|  | movl %ecx, %eax |  |
|  | movl %eax, %edx |  |
|  | sarl $31, %edx |  |
|  | idivl -4(%rbp) |  |
|  | movl %eax, %edx |  |
|  | leaq i5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq k5(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | testl %eax, %eax |  |
|  | jg .L6 |  |
|  |  |  |
| Управление переменной индукции цикла | | | |
| for( i = 0; i < 100; i++ ) ivector5[ i \* 2 + 3 ] = 5; | leaq i(%rip), %rax | Leaq 12+ivector5(%rip), %rcx | Избавление от вычисления заведомо известного шага внутри цикла |
| movl $0, (%rax) | Leaq 812+ivector5(%rip), %r8 |
| jmp .L7 | .L7: |
| .L8: | movl $5, (%rcx) |
| leaq i(%rip), %rax | addq $8, %rcx |
| movl (%rax), %eax | cmpq %r8, %rcx |
| addl %eax, %eax | jne .L7 |
| leal 3(%rax), %edx | movl $100, i(%rip) |
| leaq ivector5(%rip), %rax |  |
| movslq %edx, %rdx |  |
| movl $5, (%rax,%rdx,4) |  |
| leaq i(%rip), %rax |  |
| movl (%rax), %eax |  |
| leal 1(%rax), %edx |  |
| leaq i(%rip), %rax |  |
| movl %edx, (%rax) |  |
| .L7: |  |
| leaq i(%rip), %rax |  |
| movl (%rax), %eax |  |
| cmpl $99, %eax |  |
| jle .L8 |  |
|  |  |
| Глубокие подвыражения | | | |
| if( i < 10 ) | leaq i(%rip), %rax | movl %r9d, i2(%rip) | избавление от заведомо ложных условий, и примыкающих к ним операций |
| j5 = i5 + i2; | movl $0, (%rax) | movl %eax, i5(%rip) |
| else | leaq i(%rip), %rax | addl %r9d, %eax |
| k5 = i5 + i2; | movl (%rax), %eax | movl %eax, k5(%rip) |
|  | cmpl $9, %eax |  |
|  | jg .L9 |  |
|  | leaq i5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq i2(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | addl %eax, %edx |  |
|  | leaq j5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | jmp .L10 |  |
|  | .L9: |  |
|  | leaq i5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq i2(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | addl %eax, %edx |  |
|  | leaq k5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  |  |  |
| Проверка того, как компилятор генерирует адреспеременной с константным индексом, размножает копии и регистры | | | |
| ivector[ 0 ] = 1 ivector[ i2 ] = 2; ivector[ i2 ] = 2; ivector[ 2 ] = 3; | leaq ivector(%rip), %rax | Leaq ivector(%rip), %rdx | избавление от рассчета значений адресов ни коим образом не изменяющих начальный адрес, более оптимальный рассчет адреса изменяемого, использование регистра напрямую вместо переменной |
| movl $1, (%rax) | movl $1, ivector(%rip) |
| leaq i2(%rip), %rax | movl $2, (%rdx,%r9,4) |
| movl (%rax), %edx | movl $3, 8+ivector(%rip) |
| leaq ivector(%rip), %rax |  |
| movslq %edx, %rdx |  |
| movl $2, (%rax,%rdx,4) |  |
| leaq i2(%rip), %rax |  |
| movl (%rax), %edx |  |
| leaq ivector(%rip), %rax |  |
| movslq %edx, %rdx |  |
| movl $2, (%rax,%rdx,4) |  |
| leaq ivector(%rip), %rax |  |
| movl $3, 8(%rax) |  |
| Удаление общих подвыражений | | | |
| if(( h3 + k3 ) < 0 || ( h3 + k3 ) > 5 ) printf("Common subexpression elimination\n"); | leaq h3(%rip), %rax | movl h3(%rip), %r8d | то же избавление от заведомо ложных условий присваивание сначала в переменную g3 а после в m3 ибо так оптимальнее |
| else { | movl (%rax), %edx | leal 1(%r8), %ecx |
| m3 = ( h3 + k3 ) / i3; | leaq k3(%rip), %rax | cmpl $5, %ecx |
| g3 = i3 + (h3 + k3); | movl (%rax), %eax | ja .L13 |
|  | addl %edx, %eax | movl %ecx, %eax |
|  | testl %eax, %eax | movl $1431655766, %edx |
|  | js .L11 | sarl $31, %ecx |
|  | leaq h3(%rip), %rax | imull %edx |
|  | movl (%rax), %edx | addl $4, %r8d |
|  | leaq k3(%rip), %rax | movl %r8d, g3(%rip) |
|  | movl (%rax), %eax | subl %ecx, %edx |
|  | addl %edx, %eax | movl %edx, m3(%rip) |
|  | cmpl $5, %eax | .L13: |
|  | jle .L12 | leaq .LC4(%rip), %rcx |
|  | .L11: | call puts |
|  | leaq .LC4(%rip), %rcx |  |
|  | call puts |  |
|  | jmp .L13 |  |
|  | .L12: |  |
|  | leaq h3(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq k3(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal (%rdx,%rax), %ecx |  |
|  | leaq i3(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | movl %eax, -4(%rbp) |  |
|  | movl %ecx, %eax |  |
|  | movl %eax, %edx |  |
|  | sarl $31, %edx |  |
|  | idivl -4(%rbp) |  |
|  | movl %eax, %edx |  |
|  | leaq m3(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq h3(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq k3(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | addl %eax, %edx |  |
|  | leaq i3(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | addl %eax, %edx |  |
|  | leaq g3(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  |  |  |
| Вынесение инвариантного кода (j \* k) может быть вынесено из цикла | | | |
| for( i4 = 0; i4 <= max\_vector; i4++) | leaq i4(%rip), %rax | movzbl j(%rip), %eax | j\*k - код который выполняется на каждой интерации цикла выносится из него и считается 1 раз вне цикла |
| ivector2[ i4 ]= j\*k; | movl $0, (%rax) | mulb k(%rip) |
|  | jmp .L14 | movb %al, ivector2(%rip) |
|  | .L15: | movb %al, 1+ivector2(%rip) |
|  | leaq i4(%rip), %rax | movb %al, 2+ivector2(%rip) |
|  | movl (%rax), %r8d |  |
|  | leaq j(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | movl %eax, %ecx |  |
|  | leaq k(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | movl %eax, %edx |  |
|  | movl %ecx, %eax |  |
|  | imull %edx, %eax |  |
|  | movl %eax, %ecx |  |
|  | leaq ivector2(%rip), %rdx |  |
|  | movslq %r8d, %rax |  |
|  | movb %cl, (%rdx,%rax) |  |
|  | leaq i4(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal 1(%rax), %edx |  |
|  | leaq i4(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | .L14: |  |
|  | leaq i4(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl $2, %eax |  |
|  |  |  |
|  |  |  |
| Функция: dead\_code | | | |
| void dead\_code( a, b ) | dead\_code: | dead\_code: | Так как этот участок кода не несет в себе никакой смысловой нагрузки для выполнения всей программы в целом, то он игнорируется компилятором |
| int a; | pushq %rbp | .seh\_endprologue |
| char \*b; | .seh\_pushreg %rbp | ret |
| { | movq %rsp, %rbp | .seh\_endproc |
| int idead\_store; idead\_store = a; if( 0 ) | subq $16, %rsp |  |
| printf( "%s\n", b ); | .seh\_stackalloc 16 |  |
| } | .seh\_setframe %rbp, 16 |  |
|  | .seh\_endprologue |  |
|  | movl %ecx, 16(%rbp) |  |
|  | movq %rdx, 24(%rbp) |  |
|  | movl 16(%rbp), %eax |  |
|  | movl %eax, -4(%rbp) |  |
|  | addq $16, %rsp |  |
|  | popq %rbp |  |
|  | ret |  |
|  | .seh\_endproc |  |
|  | .globl unnecessary\_loop.def unnecessary\_loop; .scl 2; .type 32; .endef |  |
|  | .seh\_proc unnecessary\_ |  |
| Функция: unnecessary\_loop | | | |
| void unnecessary\_loop() | .globl unnecessary\_loop | .globl unnecessary\_loop | В каждой итерации цикла происходит одно и то же, а посему этот цикл убирается, в переменные кладутся лишь те значения которые были бы в этих переменных в конце последней интерации |
| { | .def unnecessary\_loop .scl 2; .type 32; .endef | .def unnecessary\_loop; .scl 2; .type 32; .endef |
| int x; | .seh\_proc unnecessary\_loop | .seh\_proc unnecessary\_loop |
| x = 0; | unnecessary\_loop: | unnecessary\_loop: |
| for( i = 0; i < 5; i++ ) k5 = x + j5; | pushq %rbp | .seh\_endprologue |
| } | .seh\_pushreg %rbp | movl j5(%rip), %eax |
|  | movq %rsp, %rbp | movl $5, i(%rip) |
|  | subq $16, %rsp | movl %eax, k5(%rip) |
|  | .seh\_stackalloc 16 | ret |
|  | .seh\_setframe %rbp, 16 | .seh\_endproc |
|  | .seh\_endprologue |  |
|  | movl $0, -4(%rbp) |  |
|  | leaq i(%rip), %rax |  |
|  | movl $0, (%rax) |  |
|  | jmp .L18 |  |
|  | .L19: |  |
|  | leaq j5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | movl -4(%rbp), %eax |  |
|  | addl %eax, %edx |  |
|  | leaq k5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal 1(%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | .L18: |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl $4, %eax |  |
|  | jle .L19 |  |
|  | addq $16, %rsp |  |
|  | popq %rbp |  |
|  | ret |  |
|  | .seh\_endproc |  |
| Функция: loop\_jamming | | | |
| void loop\_jamming( x ) | .globl loop\_jamming | .globl loop\_jamming | Циклы с одинаковыми условиями и бегущими переменным сливаются в один цикл |
| int x; | .def loop\_jamming; .scl 2; .type 32; .endef | .def loop\_jamming; .scl 2; .type 32; .endef |
| { | .seh\_proc loop\_jamming | .seh\_proc loop\_jamming |
| for( i = 0; i < 5; i++ ) | loop\_jamming: | loop\_jamming: |
| k5 = x + j5 \* i; | pushq %rbp | .seh\_endprologue |
| for( i = 0; i < 5; i++ ) | .seh\_pushreg %rbp | movl j5(%rip), %eax |
| i5 = x \* k5 \* i; | movq %rsp, %rbp | movl $5, i(%rip) |
| } | .seh\_setframe %rbp, 0 | leal (%rcx,%rax,4), %eax |
|  | .seh\_endprologue | movl %eax, k5(%rip) |
|  | movl %ecx, 16(%rbp) | imull %ecx, %eax |
|  | leaq i(%rip), %rax | sall $2, %eax |
|  | movl $0, (%rax) | movl %eax, i5(%rip) |
|  | jmp .L21 | ret |
|  | L22: | .seh\_endproc |
|  | leaq j5(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | imull %eax, %edx |  |
|  | movl 16(%rbp), %eax |  |
|  | addl %eax, %edx |  |
|  | leaq k5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal 1(%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | L21: |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl $4, %eax |  |
|  | jle .L22 |  |
|  | leaq i(%rip), %rax |  |
|  | movl $0, (%rax) |  |
|  | jmp .L23 |  |
|  | L24: |  |
|  | leaq k5(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | movl %eax, %edx |  |
|  | imull 16(%rbp), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | imull %eax, %edx |  |
|  | leaq i5(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal 1(%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | L23: |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl $4, %eax |  |
|  | jle .L24 |  |
|  | popq %rbp |  |
|  | ret |  |
|  | .seh\_endproc |  |
| Функция: loop\_unrolling | | | |
| void loop\_unrolling( x ) | .globl loop\_unrolling | .seh\_proc loop\_unrolling | простой цикл (в коем выполняется по одной операции не зависящей от остальной части кода всей программы) раскидывается на множество соответствующих присваиваний, в бегущей переменной значение которое лежало бы в ней за последнюю интерацию |
| int x; | .def loop\_unrolling; .scl 2; .type 32; .endef | loop\_unrolling: |
| { | .seh\_proc loop\_unrolling | .seh\_endprologue |
| for( i = 0; i < 6; i++ ) ivector4[ i ] = 0; | loop\_unrolling: | movw $0, ivector4(%rip) |
| } | pushq %rbp | movw $0, 2+ivector4(%rip) |
|  | .seh\_pushreg %rbp | movw $0, 4+ivector4(%rip) |
|  | movq %rsp, %rbp | movw $0, 6+ivector4(%rip) |
|  | .seh\_setframe %rbp, 0 | movw $0, 8+ivector4(%rip) |
|  | .seh\_endprologue | movw $0, 10+ivector4(%rip) |
|  | movl %ecx, 16(%rbp) | movl $6, i(%rip) |
|  | leaq i(%rip), %rax | ret |
|  | movl $0, (%rax) | .seh\_endproc |
|  | jmp .L26 |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %edx |  |
|  | leaq ivector4(%rip), %rax |  |
|  | movslq %edx, %rdx |  |
|  | movw $0, (%rax,%rdx,2) |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | leal 1(%rax), %edx |  |
|  | leaq i(%rip), %rax |  |
|  | movl %edx, (%rax) |  |
|  | .L26: |  |
|  | leaq i(%rip), %rax |  |
|  | movl (%rax), %eax |  |
|  | cmpl $5, %eax |  |
|  | jle .L27 |  |
|  | popq %rbp |  |
|  | ret |  |
|  | .seh\_endproc |  |
| Функция: jump\_compression | | | |
| int jump\_compression( i, j, k, l, m ) | .globl jump\_compression | globl jump\_compression | Сжатие множества сравнений в более простые |
| int i, j, k, l, m; | def jump\_compression; .scl 2; .type 32; .endef | .def jump\_compression; .scl 2; .type 32; .endef |
| { | .seh\_proc jump\_compression | .seh\_proc |
| beg\_1: | jump\_compression: | jump\_compression |
| if( i < j ) if( j < k ) if( k < l ) if( l < m ) l += m; else goto end\_1; else k += l; else { j += k; | pushq %rbp | jump\_compression: |
| end\_1: goto beg\_1; } | .seh\_pushreg %rbp | .seh\_endprologue |
| else | movq %rsp, %rbp | movl 40(%rsp), %eax |
| i += j; return( i + j + k + l + m ); | .seh\_setframe %rbp, 0 | cmpl %ecx, %edx |
| } | .seh\_endprologue | jg .L23 |
|  | movl %ecx, 16(%rbp) | jmp .L17 |
|  | movl %edx, 24(%rbp) | .p2align 4,,10 |
|  | movl %r8d, 32(%rbp) | L26: |
|  | movl %r9d, 40(%rbp) | cmpl %r9d, %r8d |
|  | .L29: | jge .L20 |
|  | movl 16(%rbp), %eax | cmpl %eax, %r9d |
|  | cmpl 24(%rbp), %eax | .p2align 4,,5 |
|  | jge .L30 | jl .L25 |
|  | movl 24(%rbp), %eax | cmpl %edx, %ecx |
|  | cmpl 32(%rbp), %eax | .p2align 4,,3 |
|  | jge .L31 | jge .L17 |
|  | movl 32(%rbp), %eax | L23: |
|  | cmpl 40(%rbp), %eax | cmpl %edx, %r8d |
|  | jge .L32 | .p2align 4,,3 |
|  | movl 40(%rbp), %eax | jg .L26 |
|  | cmpl 48(%rbp), %eax | addl %r8d, %edx |
|  | jge .L37 | cmpl %edx, %ecx |
|  | movl 48(%rbp), %eax | jl .L23 |
|  | addl %eax, 40(%rbp) | L17: |
|  | jmp .L34 | addl %edx, %ecx |
|  | movl 40(%rbp), %eax | addl %ecx, %edx |
|  | addl %eax, 32(%rbp) | addl %edx, %r8d |
|  | jmp .L34 | addl %r8d, %r9d |
|  | movl 32(%rbp), %eax | addl %r9d, %eax |
|  | addl %eax, 24(%rbp) | ret |
|  | jmp .L29 | .p2align 4,,10 |
|  | nop | L20: |
|  | jmp .L29 | addl %r9d, %r8d |
|  | L30: | addl %ecx, %edx |
|  | movl 24(%rbp), %eax | addl %edx, %r8d |
|  | addl %eax, 16(%rbp) | addl %r8d, %r9d |
|  | movl 24(%rbp), %eax | addl %r9d, %eax |
|  | movl 16(%rbp), %edx | ret |
|  | addl %eax, %edx | .p2align 4,,10 |
|  | movl 32(%rbp), %eax | L25: |
|  | addl %eax, %edx | addl %ecx, %edx |
|  | movl 40(%rbp), %eax | addl %eax, %r9d |
|  | addl %eax, %edx | addl %edx, %r8d |
|  | movl 48(%rbp), %eax | addl %r8d, %r9d |
|  | addl %edx, %eax | addl %r9d, %eax |
|  | popq %rbp | ret |
|  | ret | .seh\_endproc |
|  | .seh\_endproc |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Clang** |  |  |  |
| Код на С | Код ассемблера | Код ассемблера (опт по врем) | Пояснения |
| Размножение констант и копий | | | |
| j4 = 2; | pushl %ebp | # BB#0: # %entry | в отличае от предыдущего компилятора здесь все условия выполняются, некоторые переменные берутся из регистров |
| if( i2 < j4 && i4 < j4 ) | movl %esp, %ebp | movl \_i2, %eax |
| i2 = 2; | pushl %edi | movl \_i4, %ecx |
| j4 = k5; | pushl %esi | movl $2, \_j4 |
| if( i2 < j4 && i4 < j4 ) | subl $80, %esp | cmpl $1, %eax |
| i5 = 3; | movl 12(%ebp), %eax | jg LBB0\_3 |
|  | movl 8(%ebp), %ecx | # BB#1: # %entry |
|  | movl $0, -12(%ebp) | cmpl $1, %ecx |
|  | movl $2, \_j4 | jg LBB0\_3 |
|  | movl \_i2, %edx | # BB#2: # %if.then |
|  | cmpl \_j4, %edx | movl $2, %eax |
|  | movl %eax, -16(%ebp) # 4-byte Spill | movl $2, \_i2 |
|  | movl %ecx, -20(%ebp) # 4-byte Spill | LBB0\_3: # %if.end |
|  | jge LBB4\_3 | movl \_k5, %edx |
|  | # BB#1: # %land.lhs.true | cmpl %edx, %ecx |
|  | movl \_i4, %eax | movl %edx, \_j4 |
|  | cmpl \_j4, %eax | jge LBB0\_6 |
|  | jge LBB4\_3 | # BB#4: # %if.end |
|  | # BB#2: # %if.then | cmpl %edx, %eax |
|  | movl $2, \_i2 | jge LBB0\_6 |
|  | LBB4\_3: # %if.end | # BB#5: # %if.then5 |
|  | movl \_k5, %eax | movl $3, \_i5 |
|  | movl %eax, \_j4 | LBB0\_6: # %if.end6 |
|  | movl \_i2, %eax |  |
|  | cmpl \_j4, %eax |  |
|  | jge LBB4\_6 |  |
|  | # BB#4: # %land.lhs.true3 |  |
|  | movl \_i4, %eax |  |
|  | cmpl \_j4, %eax |  |
|  | jge LBB4\_6 |  |
|  | # BB#5: # %if.then5 |  |
|  | movl $3, \_i5 |  |
|  | LBB4\_6: # %if.end6 |  |
|  |  |  |
| Свертка констант, арифметические тождества и излишние операции загрузки/сохранения | | | |
| i3 = 1 + 2; | leal "??\_C@\_0EG@ FMOENJDP@This? 5compiler?5handles? 5divide?9by? 9@", %eax | pushl %esi | #ССЫЛКА! |
| flt\_1 = 2.4 + 6.3; | movl $1, %ecx | movl \_i, %eax |
| i2 = 5; | movsd \_\_real@4021666666666666, %xmm0 # xmm0 = mem[0],zero | movl $3, \_i3 |
| j2 = i + 0; | movl $3, \_i3 | movl $1075930726, \_flt\_1+4 # imm = 0x40216666 |
| k2 = i / 1; | movsd %xmm0, \_flt\_1 | movl $1717986918, \_flt\_1 # imm = 0x66666666 |
| i4 = i \* 1; | movl $5, \_i2 | movl $5, \_i2 |
| i5 = i \* 0; | movl \_i, %edx | movl $0, \_i5 |
| #ifndef NO\_ZERO\_DIVIDE | addl $0, %edx | movl %eax, \_j2 |
| i2 = i / 0; | movl %edx, \_j2 | movl %eax, \_k2 |
| flt\_2 = flt\_1 / 0.0; | movl \_i, %edx | movl %eax, \_i4 |
| #else | movl %eax, -24(%ebp) # 4-byte Spill | pushl $L\_str |
| printf( "This compiler handles divide-by-zero as \an error\n"); | movl %edx, %eax | calll \_puts |
| #endif | cltd | addl $4, %esp |
| flt\_3 = 2.4 / 1.0; | idivl %ecx | movsd \_flt\_2, %xmm1 # xmm1 = mem[0],zero |
| flt\_4 = 1.0 + 0.0000001; | movl %eax, \_k2 | xorpd %xmm0, %xmm0 |
| flt\_5 = flt\_6 \* 0.0; | movl \_i, %eax | movl $1073951539, \_flt\_3+4 # imm = 0x40033333 |
| flt\_6 = flt\_2 \* flt\_3; | shll $0, %eax | movl $858993459, \_flt\_3 # imm = 0x33333333 |
|  | movl %eax, \_i4 | movl $1072693248, \_flt\_4+4 # imm = 0x3FF00000 |
|  | imull $0, \_i, %eax | movl $450359963, \_flt\_4 # imm = 0x1AD7F29B |
|  | movl %eax, \_i5 | .loc 1 92 27 # optbench.c:92:27 |
|  | movl -24(%ebp), %eax # 4-byte Reload | mulsd \_flt\_6, %xmm0 |
|  | movl %eax, (%esp) | mulsd \_\_real@4003333333333333, %xmm1 |
|  | calll \_printf | movsd %xmm0, \_flt\_5 |
|  | xorps %xmm0, %xmm0 | movsd %xmm1, \_flt\_6 |
|  | movsd \_\_real@3ff000001ad7f29b, %xmm1 # xmm1 = mem[0],zero |  |
|  | movsd \_\_real@4003333333333333, %xmm2 # xmm2 = mem[0],zero |  |
|  | movsd %xmm2, \_flt\_3 |  |
|  | movsd %xmm1, \_flt\_4 |  |
|  | mulsd \_flt\_6, %xmm0 |  |
|  | movsd %xmm0, \_flt\_5 |  |
|  | movsd \_flt\_2, %xmm0 # xmm0 = mem[0],zero |  |
|  | mulsd \_flt\_3, %xmm0 |  |
|  | movsd %xmm0, \_flt\_6 |  |
|  |  |  |
| Лишнее присваивание | | | |
| k3 = 1; | movl $1, \_k3 | movl $1, \_k3 | Игнорирование лишнего присваивания, присваивание напрямую через регистр |
| k3 = 1; | movl $1, \_k3 |
|  |  |
| Снижение мощности | | | |
| k2 = 4 \* j5; | movl \_j5, %ecx | movl \_j5, %eax | Цикл заменен на 5 присваиваний, значение i в конечную интерацию сразу присваивается своей переменной, побитовый сдвиг вместо умножения на числа кратные 2 |
| for( i = 0; i <= 5; i++ ) ivector4[ i ] = i \* 2; | shll $2, %ecx | movl $131072, \_ivector4 # imm = 0x20000 |
|  | movl %ecx, \_k2 | movl $393220, \_ivector4+4 # imm = 0x60004 |
|  | movl $0, \_i | movl $655368, \_ivector4+8 # imm = 0xA0008 |
|  | movl %eax, -28(%ebp) # 4-byte Spill | shll $2, %eax |
|  | LBB4\_7: # %for.cond | movl %eax, \_k2 |
|  | # =>This Inner Loop Header: Depth=1 |  |
|  | cmpl $5, \_i |  |
|  | jg LBB4\_10 |  |
|  | movl \_i, %eax |  |
|  | shll $1, %eax |  |
|  | movw %ax, %cx |  |
|  | movl \_i, %eax |  |
|  | movw %cx, \_ivector4(,%eax,2) |  |
|  | movl \_i, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_i |  |
|  | jmp LBB4\_7 |  |
| Простой цикл | | | |
| j5 = 0; | LBB4\_10: # %for.end | movl $0, \_k5 | Игнорирование бессмысленного цикла, присваивание конечных значений |
| k5 = 10000; | movl $0, \_j5 | movl $10000, \_j5 # imm = 0x2710 |
| do { | movl $10000, \_k5 # imm = 0x2710 | movl $0, \_i5 |
| k5 = k5 - 1; | LBB4\_11: # %do.body |  |
| j5 = j5 + 1; | # =>This Inner Loop Header: Depth=1 |  |
| i5 = (k5 \* 3) / (j5 \* constant5); | movl \_k5, %eax |  |
| } while ( k5 > 0 ); | subl $1, %eax |  |
|  | movl %eax, \_k5 |  |
|  | movl \_j5, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_j5 |  |
|  | imull $3, \_k5, %eax |  |
|  | imull $5, \_j5, %ecx |  |
|  | cltd |  |
|  | idivl %ecx |  |
|  | movl %eax, \_i5 |  |
|  | # BB#12: # %do.cond |  |
|  | # in Loop: Header=BB4\_11 Depth=1 |  |
|  | cmpl $0, \_k5 |  |
|  | jg LBB4\_11 |  |
|  |  |  |
| Управление переменной индукции цикла | | | |
| for( i = 0; i < 100; i++ ) ivector5[ i \* 2 + 3 ] = 5; | LBB4\_14: # %for.cond19 | LBB0\_7: | Избавление от вычисления заведомо известного шага внутри цикла |
| cmpl $100, \_i | movl $5, \_ivector5+812(%eax) |
| jge LBB4\_17 | addl $8, %eax |
| # BB#15: # %for.body22 | jne LBB0\_7 |
| movl \_i, %eax |  |
| shll $1, %eax |  |
| movl $5, \_ivector5+12(,%eax,4) |  |
| # BB#16: # %for.inc26 |  |
| movl \_i, %eax |  |
| addl $1, %eax |  |
| movl %eax, \_i |  |
| jmp LBB4\_14 |  |
| LBB4\_17: |  |
|  |  |
| Глубокие подвыражения | | | |
| if( i < 10 ) | cmpl $10, \_i | movl \_j5, %eax | избавление от заведомо ложных условий, и примыкающих к ним операций |
| j5 = i5 + i2; | jge LBB4\_19 | movl %eax, \_k5 |
| else | # BB#18: # %if.then31 |  |
| k5 = i5 + i2; | movl \_i5, %eax |  |
|  | addl \_i2, %eax |  |
|  | movl %eax, \_j5 |  |
|  | jmp LBB4\_20 |  |
|  | LBB4\_19: # %if.else |  |
|  | movl \_i5, %eax |  |
|  | addl \_i2, %eax |  |
|  | movl %eax, \_k5 |  |
|  |  |  |
| Проверка того, как компилятор генерирует адреспеременной с константным индексом, размножает копии и регистры | | | |
| ivector[ 0 ] = 1 ivector[ i2 ] = 2; ivector[ i2 ] = 2; ivector[ 2 ] = 3; | movl $1, \_ivector | movl $1, \_ivector | избавление от рассчета значений адресов ни коим образом не изменяющих начальный адрес, более оптимальный рассчет адреса изменяемого, использование регистра напрямую вместо переменной |
| movl \_i2, %eax | movl $2, \_ivector(,%eax,4) |
| movl $2, \_ivector(,%eax,4) | movl $3, \_ivector+8 |
| movl \_i2, %eax |  |
| movl $2, \_ivector(,%eax,4) |  |
| movl $3, \_ivector+8 |  |
| Удаление общих подвыражений | | | |
| if(( h3 + k3 ) < 0 || ( h3 + k3 ) > 5 ) printf("Common subexpression elimination\n"); | movl \_h3, %eax | movl \_h3, %ecx | то же избавление от заведомо ложных условий присваивание сначала в переменную g3 а после в m3 ибо так оптимальнее |
| else { | addl \_k3, %eax | incl %ecx |
| m3 = ( h3 + k3 ) / i3; | cmpl $5, %eax | cmpl $6, %ecx |
| g3 = i3 + (h3 + k3); | jle LBB4\_23 | jb LBB0\_10 |
|  | LBB4\_22: # %if.then43 | # BB#9: # %if.then43 |
|  | leal "??\_C@\_0CC@ DDFJGIGO @Common? 5subexpression ?5elimination@", %eax | pushl $L\_str.1 |
|  | movl %eax, (%esp) | calll \_puts |
|  | calll \_printf | LBB0\_10: # %if.else45 |
|  | movl %eax, -32(%ebp) # 4-byte Spill | movl \_i3, %esi |
|  | jmp LBB4\_24 | movl %ecx, %eax |
|  | LBB4\_23: # %if.else45 | cltd |
|  | movl \_h3, %eax | idivl %esi |
|  | addl \_k3, %eax | addl %ecx, %esi |
|  | cltd | movl %eax, \_m3 |
|  | idivl \_i3 | movl %esi, \_g3 |
|  | movl %eax, \_m3 |  |
|  | movl \_i3, %eax |  |
|  | movl \_h3, %ecx |  |
|  | addl \_k3, %ecx |  |
|  | addl %ecx, %eax |  |
|  | movl %eax, \_g3 |  |
|  |  |  |
| Вынесение инвариантного кода (j \* k) может быть вынесено из цикла | | | |
| for( i4 = 0; i4 <= max\_vector; i4++) | movl $0, \_i4 | movl \_k, %ecx | j\*k - код который выполняется на каждой интерации цикла выносится из него и считается 1 раз вне цикла |
| ivector2[ i4 ]= j\*k; | LBB4\_25: # %for.cond51 | movl $3, \_i4 |
|  | cmpl $2, \_i4 | imull \_j, %ecx |
|  | jg LBB4\_28 | movb %cl, \_ivector2+2 |
|  | # BB#26: # %for.body54 | movzbl %cl, %ecx |
|  | movl \_j, %eax | movl %ecx, %edx |
|  | imull \_k, %eax | shll $8, %edx |
|  | movb %al, %cl | orl %ecx, %edx |
|  | movl \_i4, %eax | movw %dx, \_ivector2 |
|  | movb %cl, \_ivector2(,%eax) |  |
|  | # BB#27: # %for.inc58 |  |
|  | movl \_i4, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_i4 |  |
|  | jmp LBB4\_25 |  |
|  |  |  |
| Функция: dead\_code | | | |
| void dead\_code( a, b ) | \_dead\_code: | \_dead\_code: # @dead\_code | Данный код не выполняется |
| int a; | pushl %ebp | Lfunc\_begin1: |
| char \*b; | movl %esp, %ebp | # BB#0: # %entry |
| { | subl $12, %esp | retl |
| int idead\_store; idead\_store = a; | movl 12(%ebp), %eax | Lfunc\_end1: |
| if( 0 ) | movl 8(%ebp), %ecx |  |
| printf( "%s\n", b ); | movl 8(%ebp), %edx |  |
| } | movl %edx, -4(%ebp) |  |
|  | movl %eax, -8(%ebp) |  |
|  | movl %ecx, -12(%ebp) |  |
|  | addl $12, %esp |  |
|  | popl %ebp |  |
|  | retl |  |
| Функция: unnecessary\_loop | | | |
| void unnecessary\_loop() | \_unnecessary\_loop: # @unnecessary\_loop | .globl \_unnecessary\_loop # -- Begin function unnecessary\_loop | В каждой итерации цикла происходит одно и то же, а посему этот цикл убирается, в переменные кладутся лишь те значения которые были бы в этих переменных в конце последней интерации |
| { | # BB#0: # %entry | .p2align 4, 0x90 |
| int x; | pushl %ebp | \_unnecessary\_loop: # @unnecessary\_loop |
| x = 0; | movl %esp, %ebp | Lfunc\_begin2: |
| for( i = 0; i < 5; i++ ) k5 = x + j5; | pushl %eax | # BB#0: # %entry |
| } | movl $0, -4(%ebp) | movl \_j5, %eax |
|  | movl $0, \_i | movl $5, \_i |
|  | LBB7\_1: # %for.cond | movl %eax, \_k5 |
|  | # =>This Inner Loop Header: Depth=1 | retl |
|  | cmpl $5, \_i | Lfunc\_end2: |
|  | jge LBB7\_4 |  |
|  | # BB#2: # %for.body |  |
|  | # in Loop: Header=BB7\_1 Depth=1 |  |
|  | movl -4(%ebp), %eax |  |
|  | addl \_j5, %eax |  |
|  | movl %eax, \_k5 |  |
|  | # BB#3: # %for.inc |  |
|  | movl \_i, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_i |  |
|  | jmp LBB7\_1 |  |
|  | LBB7\_4: # %for.end |  |
|  | addl $4, %esp |  |
|  | popl %ebp |  |
|  | retl |  |
| Функция: loop\_jamming | | | |
| void loop\_jamming( x ) | pushl %ebp | .globl \_loop\_jamming # -- Begin function loop\_jamming | Циклы с одинаковыми условиями и бегущими переменным сливаются в один цикл |
| int x; | movl %esp, %ebp | .p2align 4, 0x90 |
| { | pushl %eax | \_loop\_jamming: # @loop\_jamming |
| for( i = 0; i < 5; i++ ) | movl 8(%ebp), %eax | Lfunc\_begin3: |
| k5 = x + j5 \* i; | movl $0, \_i | # BB#0: # %entry |
| for( i = 0; i < 5; i++ ) | movl %eax, -4(%ebp) # 4-byte Spill | movl 4(%esp), %eax |
| i5 = x \* k5 \* i; | LBB8\_1: # %for.cond | movl \_j5, %ecx |
| } | # =>This Inner Loop Header: Depth=1 | movl $5, \_i |
|  | cmpl $5, \_i | leal (%eax,%ecx,4), %ecx |
|  | jge LBB8\_4 | imull %ecx, %eax |
|  | # BB#2: # %for.body | movl %ecx, \_k5 |
|  | # in Loop: Header=BB8\_1 Depth=1 | shll $2, %eax |
|  | movl 8(%ebp), %eax | movl %eax, \_i5 |
|  | movl \_j5, %ecx | retl |
|  | imull \_i, %ecx | Lfunc\_end3: |
|  | addl %ecx, %eax |  |
|  | movl %eax, \_k5 |  |
|  | # BB#3: # %for.inc |  |
|  | # in Loop: Header=BB8\_1 Depth=1 |  |
|  | movl \_i, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_i |  |
|  | jmp LBB8\_1 |  |
|  | LBB8\_4: # %for.end |  |
|  | movl $0, \_i |  |
|  | LBB8\_5: # %for.cond1 |  |
|  | cmpl $5, \_i |  |
|  | jge LBB8\_8 |  |
|  | # BB#6: # %for.body3 |  |
|  |  |  |
|  | movl 8(%ebp), %eax |  |
|  | imull \_k5, %eax |  |
|  | imull \_i, %eax |  |
|  | movl %eax, \_i5 |  |
|  | # BB#7: # %for.inc6 |  |
|  | movl \_i, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_i |  |
|  | jmp LBB8\_5 |  |
|  | LBB8\_8: # %for.end8 |  |
|  | addl $4, %esp |  |
|  | popl %ebp |  |
|  | retl |  |
|  |  |  |
|  |  |  |
| Функция: loop\_unrolling | | | |
| void loop\_unrolling( x ) | pushl %ebp | . .globl \_loop\_unrolling # -- Begin function loop\_unrolling | Замена цикла присваиваниями |
| int x; | movl %esp, %ebp | .p2align 4, 0x90 |
| { | pushl %eax | loop\_unrolling: # |
| for( i = 0; i < 6; i++ ) | movl 8(%ebp), %eax | # @loop\_unrolling |
| ivector4[ i ] = 0; | movl $0, \_i | Lfunc\_begin4: |
| } | movl %eax, -4(%ebp) # 4-byte Spill | # BB#0: # |
|  | LBB9\_1: # %for.cond | %entry |
|  | # =>This Inner Loop Header: Depth=1 | movl $0, \_ivector4+4 |
|  | cmpl $6, \_i | movl $0, \_ivector4+4 |
|  | jge LBB9\_4 | movl $0, \_ivector4 |
|  | # BB#2: # %for.body | movl $0, \_ivector4+8 |
|  | # in Loop: Header=BB9\_1 Depth=1 |  |
|  | movl \_i, %eax |  |
|  | movw $0, \_ivector4(,%eax,2) |  |
|  | # BB#3: # %for.inc |  |
|  | # in Loop: Header=BB9\_1 Depth=1 |  |
|  | movl \_i, %eax |  |
|  | addl $1, %eax |  |
|  | movl %eax, \_i |  |
|  | jmp LBB9\_1 |  |
|  | LBB9\_4: # %for.end |  |
|  | addl $4, %esp |  |
|  | popl %ebp |  |
|  | retl |  |
|  |  |  |
| Функция: jump\_compression | | | |
| int jump\_compression( i, j, k, l, m ) | pushl %ebp | .globl \_jump\_compression # -- Begin function jump\_compression | Сжатие множества сравнений в более простые |
| int i, j, k, l, m; | movl %esp, %ebp | .p2align 4, 0x90 |
| { | pushl %edi | \_jump\_compression: # @jump\_compression |
| beg\_1: | pushl %esi | Lfunc\_begin5: |
| if( i < j ) if( j < k ) if( k < l ) if( l < m ) l += m; else goto end\_1; else k += l; else { j += k; | subl $20, %esp | .loc 1 264 0 # optbench.c:264:0 |
| end\_1: goto beg\_1; } | movl 24(%ebp), %eax | # BB#0: # %entry |
| else | movl 20(%ebp), %ecx | pushl %edi |
| i += j; return( i + j + k + l + m ); | movl 16(%ebp), %edx | pushl %esi |
| } | movl 12(%ebp), %esi | movl 28(%esp), %edi |
|  | movl 8(%ebp), %edi | movl 24(%esp), %esi |
|  | movl %eax, -12(%ebp) # 4-byte Spill | movl 20(%esp), %ecx |
|  | movl %ecx, -16(%ebp) # 4-byte Spill | movl 16(%esp), %eax |
|  | movl %edx, -20(%ebp) # 4-byte Spill | movl 12(%esp), %edx |
|  | movl %esi, -24(%ebp) # 4-byte Spill | jmp LBB5\_1 |
|  | movl %edi, -28(%ebp) # 4-byte Spill | .p2align 4, 0x90 |
|  | LBB10\_1: # %beg\_1 | LBB5\_7: # %if.else10 |
|  | # =>This Inner Loop Header: Depth=1 | addl %ecx, %eax |
|  | movl 8(%ebp), %eax | LBB5\_1: # %entry |
|  | cmpl 12(%ebp), %eax | cmpl %edx, %eax |
|  | jge LBB10\_13 | jle LBB5\_8 |
|  | # BB#2: # %if.then | # BB#2: # %if.then |
|  | # in Loop: Header=BB10\_1 Depth=1 | cmpl %ecx, %eax |
|  | movl 12(%ebp), %eax | jge LBB5\_7 |
|  | cmpl 16(%ebp), %eax | # BB#3: # %if.then2 |
|  | jge LBB10\_10 | cmpl %esi, %ecx |
|  | # BB#3: # %if.then2 | jge LBB5\_6 |
|  | # in Loop: Header=BB10\_1 Depth=1 | # BB#4: # %if.then4 |
|  | movl 16(%ebp), %eax | cmpl %edi, %esi |
|  | cmpl 20(%ebp), %eax | jge LBB5\_1 |
|  | jge LBB10\_8 | # BB#5: # %if.then6 |
|  | # BB#4: # %if.then4 | addl %edi, %esi |
|  | # in Loop: Header=BB10\_1 Depth=1 | jmp LBB5\_9 |
|  | movl 20(%ebp), %eax | LBB5\_8: # %if.else13 |
|  | cmpl 24(%ebp), %eax | addl %eax, %edx |
|  | jge LBB10\_6 | LBB5\_9: # %if.end15 |
|  | # BB#5: # %if.then6 | addl %edi, %eax |
|  | movl 24(%ebp), %eax | addl %esi, %eax |
|  | addl 20(%ebp), %eax | addl %ecx, %eax |
|  | movl %eax, 20(%ebp) | addl %edx, %eax |
|  | jmp LBB10\_7 | popl %esi |
|  | LBB10\_6: # %if.else | popl %edi |
|  | # in Loop: Header=BB10\_1 Depth=1 | retl |
|  | jmp LBB10\_11 | LBB5\_6: # %if.else7 |
|  | LBB10\_7: # %if.end | addl %esi, %ecx |
|  | jmp LBB10\_9 | jmp LBB5\_9 |
|  | LBB10\_8: # %if.else7 | Lfunc\_end5: |
|  | movl 20(%ebp), %eax |  |
|  | addl 16(%ebp), %eax |  |
|  | movl %eax, 16(%ebp) |  |
|  | LBB10\_9: # %if.end9 |  |
|  | jmp LBB10\_12 |  |
|  | LBB10\_10: # %if.else10 |  |
|  | # in Loop: Header=BB10\_1 Depth=1 |  |
|  | movl 16(%ebp), %eax |  |
|  | addl 12(%ebp), %eax |  |
|  | movl %eax, 12(%ebp) |  |
|  | LBB10\_11: # %end\_1 |  |
|  | # in Loop: Header=BB10\_1 Depth=1 |  |
|  | jmp LBB10\_1 |  |
|  | LBB10\_12: # %if.end12 |  |
|  | jmp LBB10\_14 |  |
|  | LBB10\_13: # %if.else13 |  |
|  | movl 12(%ebp), %eax |  |
|  | addl 8(%ebp), %eax |  |
|  | movl %eax, 8(%ebp) |  |
|  | LBB10\_14: # %if.end15 |  |
|  | movl 8(%ebp), %eax |  |
|  | addl 12(%ebp), %eax |  |
|  | addl 16(%ebp), %eax |  |
|  | addl 20(%ebp), %eax |  |
|  | addl 24(%ebp), %eax |  |
|  | addl $20, %esp |  |
|  | popl %esi |  |
|  | popl %edi |  |
|  | popl %ebp |  |
|  | retl |  |

3) Сравнение оптимизации трёх компиляторов

|  |  |  |  |
| --- | --- | --- | --- |
| **Сравнение трех компиляторов** | | | |
| VisualC | GCC | Clang | Вывод |
| Размножение констант и копий | | | |
| ; Line 62 | movl k5(%rip), %eax | movl \_i2, %eax | Оптимизация идентична |
| mov eax, DWORD PTR k5 | movl %eax, j4(%rip) | movl \_i4, %ecx |
| mov DWORD PTR j4, eax |  | movl $2, \_j4 |
| Свертка констант, арифметические тождества и излишние операции загрузки/сохранения | | | |
| ; Line 72 | xorpd %xmm2, %xmm2 | pushl %esi | Оптимизация идентична |
| movsd xmm0, QWORD PTR \_\_real@4021666666666666 | movsd .LC0(%rip), %xmm1 | movl \_i, %eax |
| xorps xmm3, xmm3 | xorl %ecx, %ecx | movl $3, \_i3 |
| ; Line 85 | movsd .LC2(%rip), %xmm0 | movl $1075930726, \_flt\_1+4 # imm = 0x40216666 |
| movsd xmm4, QWORD PTR \_\_real@3ff0000000000000 | movl $3, i3(%rip) | movl $1717986918, \_flt\_1 # imm = 0x66666666 |
| xor edi, edi | movsd %xmm1, flt\_1(%rip) | movl $5, \_i2 |
| ; Line 90 | divsd %xmm2, %xmm1 | movl $0, \_i5 |
| movsd xmm2, QWORD PTR \_\_real@4003333333333333 | movl i(%rip), %eax | movl %eax, \_j2 |
| ; Line 107 | movsd %xmm0, flt\_3(%rip) | movl %eax, \_k2 |
| mov ecx, edi | movl %eax, %edx | movl %eax, \_i4 |
| movsd xmm1, QWORD PTR flt\_6 | movl %eax, j2(%rip) | pushl $L\_str |
| mov eax, DWORD PTR Imov DWORD PTR j2, eax | movl %eax, i4(%rip) | calll \_puts |
| mov DWORD PTR i4, eax | sarl $31, %edx | addl $4, %esp |
| divsd xmm4, xmm3 | idivl %ecx | movsd \_flt\_2, %xmm1 # xmm1 = mem[0],zero |
| movsxd r10, eax | mulsd flt\_6(%rip), %xmm2 | xorpd %xmm0, %xmm0 |
| mov rdx, rsi | mulsd %xmm1, %xmm0 | movl $1073951539, \_flt\_3+4 # imm = 0x40033333 |
| mov eax, DWORD PTR j5 | movsd %xmm1, flt\_2(%rip) | movl $858993459, \_flt\_3 # imm = 0x33333333 |
| mov DWORD PTR i3, 3 | movslq %eax, %r9 | movl $1072693248, \_flt\_4+4 # imm = 0x3FF00000 |
| mov DWORD PTR i2, r10d | movabs $4607182419250377371, %rax | movl $450359963, \_flt\_4 # imm = 0x1AD7F29B |
| lea eax, DWORD PTR [rax\*4] | movq %rax, flt\_4(%rip) | .loc 1 92 27 # optbench.c:92:27 |
| mov DWORD PTR k2, eax | movsd %xmm2, flt\_5(%rip) | mulsd \_flt\_6, %xmm0 |
| mulsd xmm1, xmm3 | movsd %xmm0, flt\_6(%rip) | mulsd \_\_real@4003333333333333, %xmm1 |
| mulsd xmm4, xmm0 | movl %r9d, i2(%rip) | movsd %xmm0, \_flt\_5 |
| movsd QWORD PTR flt\_1, xmm0 |  | movsd %xmm1, \_flt\_6 |
| movsd xmm0, QWORD PTR \_\_real@3ff000001ad7f29b |  |  |
| movsd QWORD PTR flt\_3, xmm2 |  |  |
| movsd QWORD PTR flt\_2, xmm4 |  |  |
| mulsd xmm4, xmm2 |  |  |
| movsd QWORD PTR flt\_4, xmm0 |  |  |
| movsd QWORD PTR flt\_5, xmm1 |  |  |
| movsd QWORD PTR flt\_6, xmm4 |  |  |
| npad 6 |  |  |
| Лишнее присваивание | | | |
| mov DWORD PTR k3, 1 | movl $1, k3(%rip) | movl $1, \_k3 | Оптимизация идентична |
|  |  |  |
| Снижение мощности | | | |
| lea rsi, OFFSET FLAT:ivector4 | movw $0, ivector4(%rip) | movl \_j5, %eax | VisualC оставила цикл в то время как остальные заменили на присваивание |
| mov rdx, rsi | movw $2, 2+ivector4(%rip) | movl $131072, \_ivector4 # imm = 0x20000 |
| lea eax, DWORD PTR [rax\*4] | movw $4, 4+ivector4(%rip) | movl $393220, \_ivector4+4 # imm = 0x60004 |
| mov DWORD PTR k2, eax | movw $6, 6+ivector4(%rip) | movl $655368, \_ivector4+8 # imm = 0xA0008 |
| $LL4@main: | movw $8, 8+ivector4(%rip) | shll $2, %eax |
| ; Line 108 | movw $10, 10+ivector4(%rip) | movl %eax, \_k2 |
| movzx eax, cx | movl $6, i(%rip) |  |
| lea rdx, QWORD PTR [rdx+2] | movl j5(%rip), %eax |  |
| add ax, ax | sall $2, %eax |  |
| inc ecx | movl %eax, k2(%rip) |  |
| mov WORD PTR [rdx-2], ax |  |  |
| cmp ecx, 5 |  |  |
| jle SHORT $LL4@main |  |  |
|  |  |  |
|  |  |  |
| Простой цикл | | | |
| ; Line 114 | L6: | movl $0, \_k5 | Clang обошел цикл и сразу приравнял переменные |
| mov r9d, 10000 | movl %ecx, %edx | movl $10000, \_j5 # imm = 0x2710 |
| mov ecx, 30000 | movl %ecx, %eax | movl $0, \_i5 |
| mov DWORD PTR j5, r9d | subl $3, %ecx |  |
| mov r8d, edi | sarl $31, %edx |  |
| npad 6 | idivl %r8d |  |
| $LL7@main: | addl $5, %r8d |  |
| ; Line 117 | cmpl $-3, %ecx |  |
| sub ecx, 3 | jne .L6 |  |
| ; Line 118 | movl $0, k5(%rip) |  |
| add r8d, 5 | movl $10000, j5(%rip) |  |
| ; Line 119 | movl %eax, i5(%rip) |  |
| mov eax, ecx | movl $10000, %edx |  |
| cdq |  |  |
| idiv r8d |  |  |
| mov DWORD PTR i5, eax |  |  |
| ; Line 120 |  |  |
| test ecx, ecx |  |  |
| jg SHORT $LL7@main |  |  |
|  |  |  |
| Управление переменной индукции цикла | | | |
| ; Line 125 | Leaq 12+ivector5(%rip), %rcx | LBB0\_7: | Оптимизация идентична |
| mov edx, 100 | Leaq 812+ivector5(%rip), %r8 | movl $5, \_ivector5+812(%eax) |
| lea rcx, OFFSET FLAT:ivector5+12 | .L7: | addl $8, %eax |
| mov DWORD PTR i, edx | movl $5, (%rcx) | jne LBB0\_7 |
| mov ebx, 5 | addq $8, %rcx |  |
| npad 2 | cmpq %r8, %rcx |  |
| $LL10@main: | jne .L7 |  |
| ; Line 126 | movl $100, i(%rip) |  |
| mov DWORD PTR [rcx], ebx |  |  |
| lea rcx, QWORD PTR [rcx+8] |  |  |
| sub rdx, 1 |  |  |
| jne SHORT $LL10@main |  |  |
|  |  |  |
| Глубокие подвыражения | | | |
| mov DWORD PTR i2, r10d | movl %r9d, i2(%rip) | movl \_j5, %eax | Оптимизация идентична |
| add eax, r10d | movl %eax, i5(%rip) | movl %eax, \_k5 |
| mov DWORD PTR k5, eax | addl %r9d, %eax |  |
|  | movl %eax, k5(%rip) |  |
|  |  |  |
| Проверка того, как компилятор генерирует адреспеременной с константным индексом, размножает копии и регистры | | | |
| ; Line 144 | Leaq ivector(%rip), %rdx | movl $1, \_ivector | Оптимизация идентична |
| lea rcx, OFFSET FLAT:ivector | movl $1, ivector(%rip) | movl $2, \_ivector(,%eax,4) |
| mov DWORD PTR ivector, 1 | movl $2, (%rdx,%r9,4) | movl $3, \_ivector+8 |
| ; Line 145 | movl $3, 8+ivector(%rip) |  |
| mov DWORD PTR [rcx+r10\*4], 2 |  |  |
| ; Line 153 |  |  |
| mov DWORD PTR ivector+8, 3 |  |  |
|  |  |  |
| Удаление общих подвыражений | | | |
| ; Line 153 | movl h3(%rip), %r8d | movl \_h3, %ecx | Оптимизация идентична |
| mov ecx, DWORD PTR h3 | leal 1(%r8), %ecx | incl %ecx |
| inc ecx | cmpl $5, %ecx | cmpl $6, %ecx |
| cmp ecx, ebx | ja .L13 | jb LBB0\_10 |
| ja SHORT $LN20@main | movl %ecx, %eax | # BB#9: # %if.then43 |
| ; Line 156 | movl $1431655766, %edx | pushl $L\_str.1 |
| mov eax, 1431655766 ; 55555556H | sarl $31, %ecx | calll \_puts |
| imul ecx | imull %edx | LBB0\_10: # %if.else45 |
| mov eax, edx | addl $4, %r8d | movl \_i3, %esi |
| shr eax, 31 | movl %r8d, g3(%rip) | movl %ecx, %eax |
| add edx, eax | subl %ecx, %edx | cltd |
| ; Line 157 | movl %edx, m3(%rip) | idivl %esi |
| lea eax, DWORD PTR [rcx+3] | .L13: | addl %ecx, %esi |
| mov DWORD PTR g3, eax | leaq .LC4(%rip), %rcx | movl %eax, \_m3 |
| mov DWORD PTR m3, edx | call puts | movl %esi, \_g3 |
| jmp SHORT $LN19@main |  |  |
| $LN20@main: |  |  |
| ; Line 154 |  |  |
| lea rcx, OFFSET FLAT:??\_C@\_0CC @DDFJGIGO @Common? 5subexpression ?5elimination@ |  |  |
| call printf |  |  |
|  |  |  |
| Вынесение инвариантного кода (j \* k) может быть вынесено из цикла | | | |
| ; Line 181 | movzbl j(%rip), %eax | movl \_k, %ecx | Оптимизация идентична |
| movzx eax, BYTE PTR k | mulb k(%rip) | movl $3, \_i4 |
| movzx ecx, BYTE PTR j | movb %al, ivector2(%rip) | imull \_j, %ecx |
| imul ecx, eax | movb %al, 1+ivector2(%rip) | movb %cl, \_ivector2+2 |
| mov DWORD PTR i4, 3 | movb %al, 2+ivector2(%rip) | movzbl %cl, %ecx |
| mov BYTE PTR ivector2, cl |  | movl %ecx, %edx |
| mov BYTE PTR ivector2+1, cl |  | shll $8, %edx |
| mov BYTE PTR ivector2+2, cl |  | orl %ecx, %edx |
| npad 5 |  | movw %dx, \_ivector2 |
|  |  |  |
| Функция: dead\_code | | | |
| dead\_code PROC ; COMDAT | dead\_code: | \_dead\_code: # @dead\_code | Оптимизация идентична |
| ;File c:\users\Sanek\desktop\Папка10\optbench.c | .seh\_endprologue | Lfunc\_begin1: |
| ; Line 203 | ret | # BB#0: # %entry |
| ret 0 | .seh\_endproc | retl |
| dead\_code ENDP |  | Lfunc\_end1: |
|  |  |  |
| Функция: unnecessary\_loop | | | |
| unnecessary\_loop PROC ; COMDAT | .globl unnecessary\_loop | .globl \_unnecessary\_loop # -- Begin function unnecessary\_loop | Оптимизация идентична |
| ; File c:\users\Sanek\desktop\Папка10\optbench.c | .def unnecessary\_loop; .scl 2; .type 32; .endef | .p2align 4, 0x90 |
| ; Line 220 | .seh\_proc unnecessary\_loop | \_unnecessary\_loop: # @unnecessary\_loop |
| mov eax, DWORD PTR j5 | unnecessary\_loop: | Lfunc\_begin2: |
| mov DWORD PTR k5, eax | .seh\_endprologue | # BB#0: # %entry |
| mov DWORD PTR i, 5 | movl j5(%rip), %eax | movl \_j5, %eax |
| ; Line 221 | movl $5, i(%rip) | movl $5, \_i |
| ret 0 | movl %eax, k5(%rip) | movl %eax, \_k5 |
| unnecessary\_loop ENDP | ret | retl |
|  | .seh\_endproc | Lfunc\_end2: |
| Функция: loop\_jamming | | | |
| loop\_jamming PROC ; COMDAT | .globl loop\_jamming | .globl \_loop\_jamming # -- Begin function loop\_jamming | Оптимизация идентична |
| ; File c:\users\Sanek\desktop\Папка10\optbench.c | .def loop\_jamming; .scl 2; .type 32; .endef | .p2align 4, 0x90 |
| ; Line 233 | .seh\_proc loop\_jamming | \_loop\_jamming: # @loop\_jamming |
| mov r10d, DWORD PTR j5 | loop\_jamming: | Lfunc\_begin3: |
| mov r8d, 5 | .seh\_endprologue | # BB#0: # %entry |
| mov r9d, r8d | movl j5(%rip), %eax | movl 4(%esp), %eax |
| mov eax, ecx | movl $5, i(%rip) | movl \_j5, %ecx |
| npad 14 | leal (%rcx,%rax,4), %eax | movl $5, \_i |
| $LL4@loop\_jammi: | movl %eax, k5(%rip) | leal (%eax,%ecx,4), %ecx |
| ; Line 234 | imull %ecx, %eax | imull %ecx, %eax |
| mov edx, eax | sall $2, %eax | movl %ecx, \_k5 |
| mov DWORD PTR k5, eax | movl %eax, i5(%rip) | shll $2, %eax |
| add eax, r10d | ret | movl %eax, \_i5 |
| sub r9, 1 | .seh\_endproc | retl |
| jne SHORT $LL4@loop\_jammi |  | Lfunc\_end3: |
| ; Line 235 |  |  |
| xor eax, eax |  |  |
| mov DWORD PTR i, r8d |  |  |
| imul edx, ecx |  |  |
| npad 3 |  |  |
| $LL7@loop\_jammi: |  |  |
| ; Line 236 |  |  |
| mov DWORD PTR i5, eax |  |  |
| add eax, edx |  |  |
| sub r8, 1 |  |  |
| jne SHORT $LL7@loop\_jammi |  |  |
| ; Line 237 |  |  |
| ret 0 |  |  |
| loop\_jamming ENDP |  |  |
|  |  |  |
| Функция: loop\_unrolling | | | |
| loop\_unrolling PROC ; COMDAT | .seh\_proc loop\_unrolling | . .globl \_loop\_unrolling # -- Begin function loop\_unrolling | VisualC использует цикл в то время как остальные - присваивание |
| ; File c:\users\Sanek\desktop\Папка10\optbench.c | loop\_unrolling: | .p2align 4, 0x90 |
| ; Line 250 | .seh\_endprologue | loop\_unrolling: # @loop\_unrolling |
| $LN14: | movw $0, ivector4(%rip) | Lfunc\_begin4: |
| sub rsp, 40 ; 00000028H | movw $0, 2+ivector4(%rip) | # BB#0: # %entry |
| ; Line 251 | movw $0, 4+ivector4(%rip) | movl $0, \_ivector4+4 |
| xor edx, edx | movw $0, 6+ivector4(%rip) | movl $0, \_ivector4 |
| lea r8, OFFSET FLAT:ivector4 | movw $0, 8+ivector4(%rip) | movl $0, \_ivector4+8 |
| mov ecx, edx | movw $0, 10+ivector4(%rip) | movl $6, \_i |
| mov DWORD PTR i, edx | movl $6, i(%rip) | retl |
| npad 11 | ret | Lfunc\_end4: |
| $LL4@loop\_unrol: | .seh\_endproc |  |
| ; Line 252 |  |  |
| movsxd rax, ecx |  |  |
| add rax, rax |  |  |
| cmp rax, 12 |  |  |
| jae SHORT $LN13@loop\_unrol |  |  |
| inc ecx |  |  |
| mov WORD PTR [rax+r8], dx |  |  |
| mov DWORD PTR i, ecx |  |  |
| cmp ecx, 6 |  |  |
| jl SHORT $LL4@loop\_unrol |  |  |
| ; Line 253 |  |  |
| add rsp, 40 ; 00000028H |  |  |
| ret 0 |  |  |
| $LN13@loop\_unrol: |  |  |
| ; Line 252 |  |  |
| call \_\_report\_rangecheckfailure |  |  |
| int 3 |  |  |
| $LN12@loop\_unrol: |  |  |
| loop\_unrolling ENDP |  |  |
|  |  |  |
| Функция: jump\_compression | | | |
| jump\_compression PROC ; COMDAT | globl jump\_compression | .globl \_jump\_compression # -- Begin function jump\_compression | Оптимизация идентична |
| ; File c:\users\Sanek\desktop\Папка10\optbench.c | .def jump\_compression; .scl 2; .type 32; .endef | .p2align 4, 0x90 |
| ; Line 266 | .seh\_proc | \_jump\_compression: # @jump\_compression |
| mov r10d, DWORD PTR m$[rsp] | jump\_compression | Lfunc\_begin5: |
| $end\_1$18: | jump\_compression: | .loc 1 264 0 # optbench.c:264:0 |
| cmp ecx, edx | .seh\_endprologue | # BB#0: # %entry |
| jge SHORT $LN15@jump\_compr | movl 40(%rsp), %eax | pushl %edi |
| npad 7 | cmpl %ecx, %edx | pushl %esi |
| $beg\_1$19: | jg .L23 | movl 28(%esp), %edi |
| ; Line 267 | jmp .L17 | movl 24(%esp), %esi |
| cmp edx, r8d | .p2align 4,,10 | movl 20(%esp), %ecx |
| jge SHORT $LN4@jump\_compr | L26: | movl 16(%esp), %eax |
| ; Line 268 | cmpl %r9d, %r8d | movl 12(%esp), %edx |
| cmp r8d, r9d | jge .L20 | jmp LBB5\_1 |
| jge SHORT $LN6@jump\_compr | cmpl %eax, %r9d | .p2align 4, 0x90 |
| ; Line 269 | .p2align 4,,5 | LBB5\_7: # %if.else10 |
| cmp r9d, r10d | jl .L25 | addl %ecx, %eax |
| jge SHORT $end\_1$18 | cmpl %edx, %ecx | LBB5\_1: # %entry |
| ; Line 270 | .p2align 4,,3 | cmpl %edx, %eax |
| add r9d, r10d | jge .L17 | jle LBB5\_8 |
| ; Line 282 | L23: | # BB#2: # %if.then |
| lea eax, DWORD PTR [rcx+rdx] | cmpl %edx, %r8d | cmpl %ecx, %eax |
| add eax, r8d | .p2align 4,,3 | jge LBB5\_7 |
| add eax, r9d | jg .L26 | # BB#3: # %if.then2 |
| add eax, r10d | addl %r8d, %edx | cmpl %esi, %ecx |
| ; Line 283 | cmpl %edx, %ecx | jge LBB5\_6 |
| ret 0 | jl .L23 | # BB#4: # %if.then4 |
| $LN4@jump\_compr: | L17: | cmpl %edi, %esi |
| ; Line 276 | addl %edx, %ecx | jge LBB5\_1 |
| add edx, r8d | addl %ecx, %edx | # BB#5: # %if.then6 |
| ; Line 266 | addl %edx, %r8d | addl %edi, %esi |
| jmp SHORT $end\_1$18 | addl %r8d, %r9d | jmp LBB5\_9 |
| $LN6@jump\_compr: | addl %r9d, %eax | LBB5\_8: # |
| ; Line 274 | ret | addl %eax, %edx |
| add r8d, r9d | .p2align 4,,10 | LBB5\_9: # |
| ; Line 282 | L20: | addl %edi, %eax |
| lea eax, DWORD PTR [rcx+rdx] | addl %r9d, %r8d | addl %esi, %eax |
| add eax, r8d | addl %ecx, %edx | addl %ecx, %eax |
| add eax, r9d | addl %edx, %r8d | addl %edx, %eax |
| add eax, r10d | addl %r8d, %r9d | popl %esi |
| ; Line 283 | addl %r9d, %eax | popl %edi |
| ret 0 | ret | retl |
| $LN15@jump\_compr: | .p2align 4,,10 | LBB5\_6: # |
| ; Line 281 | L25: | addl %esi, %ecx |
| add ecx, edx | addl %ecx, %edx | jmp LBB5\_9 |
| ; Line 282 | addl %eax, %r9d | Lfunc\_end5: |
| lea eax, DWORD PTR [rcx+rdx] | addl %edx, %r8d |  |
| add eax, r8d | addl %r8d, %r9d |  |
| add eax, r9d | addl %r9d, %eax |  |
| add eax, r10d | ret |  |
| ; Line 283 | .seh\_endproc |  |
| ret 0 |  |  |
| jump\_compression ENDP |  |  |
|  |  |  |

На ряду с остальными двумя, Clang является наилучшим в оптимизации алгебраических выражений и условий; GCC – больше специализируется на удалении переменных в тех местах, где можно использовать регистры напрямую, поэтому в основном оптимизация достигается за счет этого, и поэтому же, GCC является лучшим в оптимизации по памяти; в то время как VisualC является наиболее слабым оптимизирующим компилятором в этих же областях.