

Control-Flow based Optimizations

Gadi Haber

הערה נוספת - יש לטחון על חומרת המחיר ובעיות חסות
הפסדות למחיר ניקן למקום cache miss כשהפסדות קוד לא
למציאת שם. נשאר למצוא כמה שיותר קוד מיותר באי
control flow transfer כדי למנוע cache miss



Code Reordering

קוד חק - מתבצע החן שלחיק ויש יתרון
זמנה באופטימיזציה של

Original code:

CMP R8, R9

jl L1

....

THEN PART

....

L1: CONTINUE PART

....
Improving:

1. Cache ratio - ("Hot" code grouped together)
2. Branch Penalty - (the "BT L2" is rarely taken)

Reordered code:

CMP R8, R9

jge L2

L1: CONTINUE PART

....

L2:

THEN PART

....

jmp L1

נוסיד קפיצה
כי חא הדאמור אהובי אחרק החק

אפשר אורזו לבנות לו' בדיקה שלחן שלחן
ה- trace אסני אחר

Example of loop unroll with code reorder

Original code:

Loop_label:

```
....  
    CMP R8, R9  
    jl    L1  
....  
    THEN PART  
....  
L1: CONTINUE PART  
....  
    inc r11  
    cmp r11, 0x9999  
    jle loop_label
```

Unrolled x2 code:

Loop_label:

```
....  
    CMP R8, R9  
    jl    L1  
....  
    THEN PART  
....  
L1: CONTINUE PART  
....  
    inc r11  
    cmp r11, 0x9999  
    jg outside_loop_label // jle ? jg  
....  
    CMP R8, R9  
    jl    L1  
....  
    THEN PART  
....  
L1: CONTINUE PART  
....  
    inc r11  
    cmp r11, 0x9999  
    jle loop_label  
outside_loop_label:
```

Unrolled x2 + reordered code:

Loop_label:

```
....  
    CMP R8, R9  
    jge    OUTSIDE_L1_1 // jl ? jge  
L1_1: CONTINUE PART  
....  
    inc r11  
    cmp r11, 0x9999  
    jg outside_loop_label // jle ? jg  
....  
    CMP R8, R9  
    jge    OUTSIDE_L1_2 // jl ? jge  
L1_2: CONTINUE PART  
....  
    inc r11  
    cmp r11, 0x9999  
    jle loop_label  
outside_loop_label:  
....  
OUTSIDE_L1_1:  
....  
    THEN PART  
    jmp l1_1  
OUTSIDE_L1_2:  
....  
    THEN PART  
    jmp l1_2
```

Code reordering optimization

Reduce number of branch instructions

Reduce the number of I-cache misses

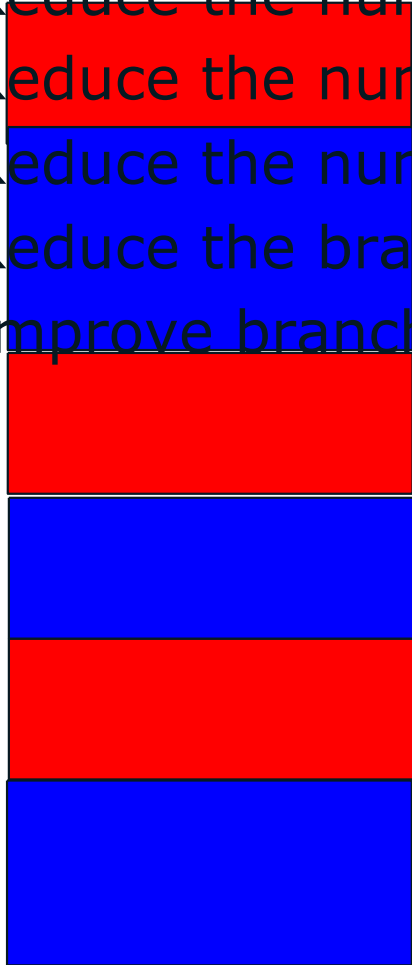
Reduce the number of I-TLB misses

Reduce the number of page faults

Reduce the branch penalty

Improve branch prediction

בסחיקט קודאים נתן ליפול אחסור



Code Reordering Example – Original code

```
#include <stdio.h>
```

```
#define MAX_ITERATIONS 1000000000
```

```
void main()  
{
```

```
    int i,j,x =0 ,y=0,z=0;
```

```
    for (i=0; i < MAX_ITERATIONS; i++) {
```

```
        if (i <= MAX_ITERATIONS) {
```

```
            x++;
```

```
        } else {
```

```
            x++;
```

```
        }
```

```
    }
```

```
}
```

Code Reordering Example – cont'd

L2:

```
cmpl $999999999, -4(%ebp)
jg L1
cmpl $1000000000, -4(%ebp)
jg L5
```

```
leal -12(%ebp), %eax # x++
incl(%eax)
jmp L4
```

L5:

```
leal -12(%ebp), %eax # x++
incl(%eax)
```

L4:

```
leal -4(%ebp), %eax # i++
incl(%eax)
jmp L2
```

L1:

```
leave
ret
```

L2:

```
cmpl $999999999, -4(%ebp)
jg L1
cmpl $1000000000, -4(%ebp)
jg L5
```

```
leal -12(%ebp), %eax # x++
incl(%eax)
# jmp L4
```

L4:

```
leal -4(%ebp), %eax # i++
incl(%eax)
jmp L2
```

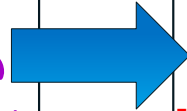
L1:

```
leave
ret
```

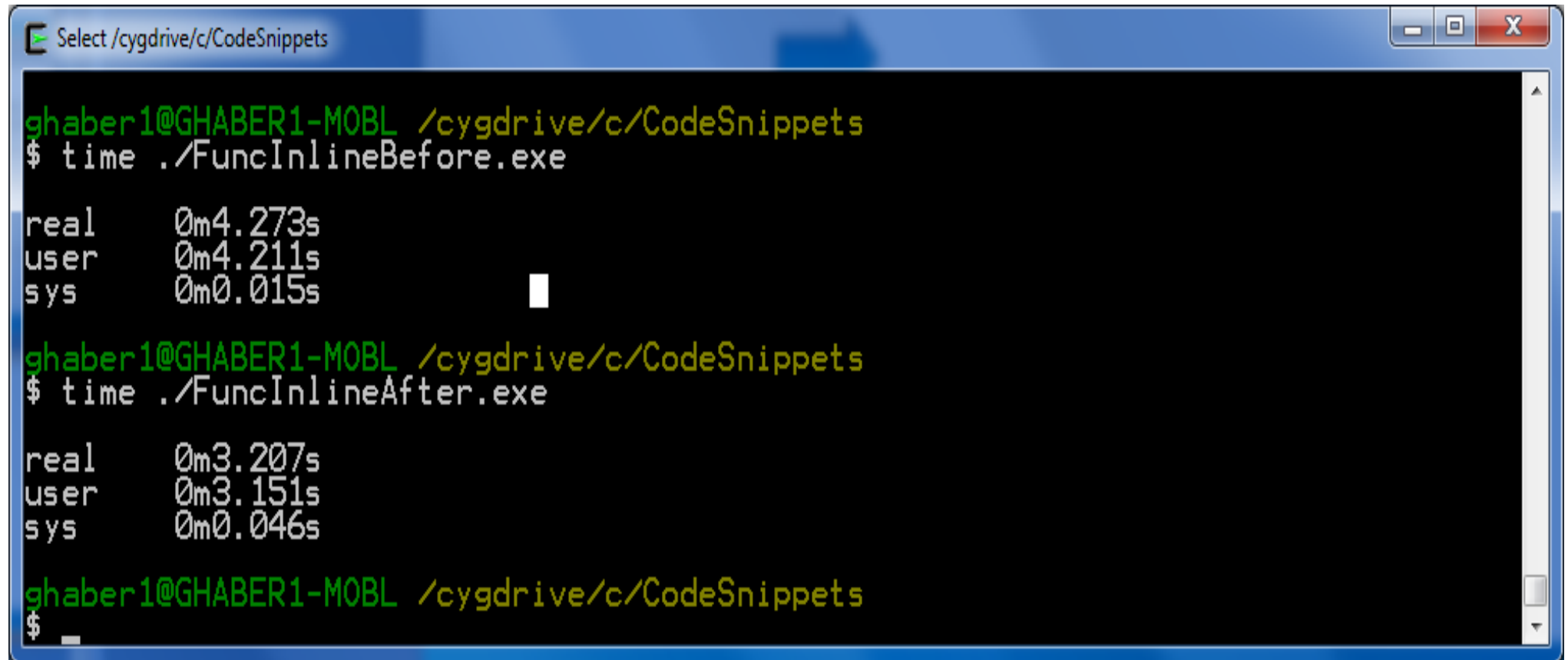
L5:

```
leal -12(%ebp), %eax # x++
incl(%eax)
jmp L4
```

Handwritten notes:
A purple bracket groups the L5 code block in the original code. Next to it, the text "4 down" is written, indicating a four-line downward shift in the original code's layout.



Code Reordering Example - results



The screenshot shows a Windows command prompt window titled "Select /cygdrive/c/CodeSnippets". The prompt is "ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets". The first command executed is "\$ time ./FuncInlineBefore.exe", which produces the following output:

Category	Time
real	0m4.273s
user	0m4.211s
sys	0m0.015s

The second command executed is "\$ time ./FuncInlineAfter.exe", which produces the following output:

Category	Time
real	0m3.207s
user	0m3.151s
sys	0m0.046s

The prompt then returns to "ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets" and "\$ _".

Loop Unroll Example – Before and After

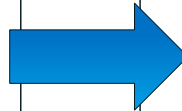
```
#include <stdio.h>

#define MAX_ITERATIONS 1000000000

void main()
{
    int i,j,x =0 ,y=0,z=0;

    for (i=0; i < MAX_ITERATIONS; i++)
    {
        x++;
        y++;
        z++;
    }

    printf("x=%d y=%d z=%d\n", x,y,z);
}
```



```
#include <stdio.h>

// provided MAX_ITERATIONS is even:
#define MAX_ITERATIONS 1000000000

void main()
{
    int i,j,x =0 ,y=0,z=0;

    for (i=0; i < MAX_ITERATIONS; i+=2) {
        x++;
        y++;
        z++;

        x++;
        y++;
        z++;
    }

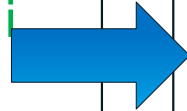
    printf("x=%d y=%d z=%d\n", x,y,z);
}
```


Loop Unroll Example1 – cont'd

L2:

```
leal-12(%ebp), %eax
incl(%eax)
leal-16(%ebp), %eax
incl(%eax)
leal-20(%ebp), %eax
incl(%eax)
leal-4(%ebp), %eax // loop var i
incl(%eax)
cmpl$999999999, -4(%ebp)
Jle L2
```

L3:



L2:

```
leal-12(%ebp), %eax
incl(%eax)
leal-16(%ebp), %eax
incl(%eax)
leal-20(%ebp), %eax
incl(%eax)
leal-4(%ebp), %eax
incl(%eax)
cmpl$999999999, -4(%ebp)
jg L3 // reverse the cond. jump
```

```
leal-12(%ebp), %eax
incl(%eax)
leal-16(%ebp), %eax
incl(%eax)
leal-20(%ebp), %eax
incl(%eax)
leal-4(%ebp), %eax
incl(%eax)
cmpl$999999999, -4(%ebp)
Jle L2
```

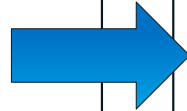
L3:

Loop Unroll Example2 – cont'd

L2:

```
cmpl$999999999, -4(%ebp)
jgL3
leal-12(%ebp), %eax
incl(%eax)
leal-16(%ebp), %eax
incl(%eax)
leal-20(%ebp), %eax
incl(%eax)
leal-4(%ebp), %eax // loop var i
incl(%eax)
jmp L2
```

L3:



L2:

```
cmpl$999999999, -4(%ebp)
jgL3
leal-12(%ebp), %eax
incl(%eax)
leal-16(%ebp), %eax
incl(%eax)
leal-20(%ebp), %eax
incl(%eax)
leal-4(%ebp), %eax
incl(%eax)
# jmpL2
```

```
cmpl$999999999, -4(%ebp)
jgL3
leal-12(%ebp), %eax
incl(%eax)
leal-16(%ebp), %eax
incl(%eax)
leal-20(%ebp), %eax
incl(%eax)
leal-4(%ebp), %eax
incl(%eax)
```

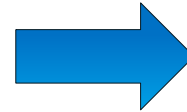
jmpL2

L3:

Loop Unroll Example2 – cont'd

```
L2:
    cmpl$999999999, -4(%ebp)
    jg L3
    leal-12(%ebp), %eax
    incl(%eax)
    leal-16(%ebp), %eax
    incl(%eax)
    leal-20(%ebp), %eax
    incl(%eax)
    leal-4(%ebp), %eax
    incl(%eax)
    jmp L2
```

L3:



```
L2:
    cmpl$999999999-1, -4(%ebp) // fix compare
    jg L2_Tail
```

```
    leal-12(%ebp), %eax
    incl(%eax)
    leal-16(%ebp), %eax
    incl(%eax)
    leal-20(%ebp), %eax
    incl(%eax)
    leal-4(%ebp), %eax
    incl(%eax)
    # jmp L2
```

```
#cmpl$999999999, -4(%ebp)
#jg L3
    leal-12(%ebp), %eax
    incl(%eax)
    leal-16(%ebp), %eax
    incl(%eax)
    leal-20(%ebp), %eax
    incl(%eax)
    leal-4(%ebp), %eax
    incl(%eax)
    jmpL2
```

L2_Tail:


```
    cmpl$999999999, -4(%ebp)
    jgL3
    leal-12(%ebp), %eax
    incl(%eax)
    leal-16(%ebp), %eax
    incl(%eax)
    leal-20(%ebp), %eax
    incl(%eax)
    leal-4(%ebp), %eax
    incl(%eax)
    jmp L2_Tail
```

L3:

הוא הולך להקריב כזו שנים לזמן 4

Loop Unrolling in Pseudo C - cont'd

```
//orig loop:  
for (i=0; i < N; i++) {  
    <body>  
}
```



```
// Unroll loop by X:  
if (N < X)  
    goto remainder_label;  
  
for (i=0; i < N-X; ) {  
    <body>; i++; // unroll 1  
    <body>; i++; // unroll 2  
    ...  
    <body>; i++; // unroll X  
}  
remainder_label:  
for (; i < N; i++) {  
    <body>  
}
```

Loop Unrolling Example - results

```
/cygdrive/c/CodeSnippets
$ time ./UnrollShortLoopBefore.exe
x=1000000000 y=1000000000 z=1000000000

real    0m2.479s
user    0m2.371s
sys     0m0.062s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./UnrollShortLoopAfter.exe
x=1000000000 y=1000000000 z=1000000000

real    0m2.979s
user    0m2.917s
sys     0m0.046s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$

Select /cygdrive/c/CodeSnippets
$ time ./UnrollShortLoopBefore.exe
x=1000000000 y=1000000000 z=1000000000

real    0m2.528s
user    0m2.433s
sys     0m0.061s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./UnrollShortLoopAfter2.exe
x=1000000000 y=1000000000 z=1000000000

real    0m2.400s
user    0m2.324s
sys     0m0.015s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$
```

Determining Possible Candidates for Loop Unrolling

Loop Unrolling Pattern:

- Loop is reducible,
 - i.e. does not contain direct jumps to middle of it
- Loop terminates with a conditional backward jump to beginning of loop
 - Sets the locations of the **end loop** and **loop beginning**
- Conditional backward jump is preceded by a **Check instruction** that sets RFLAGS
- **Check instruction** consists of 2 operands:
 - operand1: **Loop induction variable/reg** - Written in loop **only once** before the Check
 - Operand2: **Compare Variable/Reg** - Read-only variable/reg in the loop
- **Loop induction variable** is written to only once in a dominating location according to the Control Flow Graph (CFG) of the loop

14 Loop induction variable is updated by a constant stride

Function Inline Example – Before and After

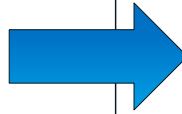
```
#include <stdio.h>
```

```
#define MAX_ITERATIONS  
1000000000
```

```
void foo()  
{  
    int x=0, y=0;  
    x++;  
    y++;  
}
```

```
void main()  
{  
    int i,j,x =0 ,y=0,z=0;  
  
    for (i=0; i < MAX_ITERATIONS; i++) {  
        foo();  
    }  
}
```

רק פה אפשר inlining
אזכור אחד פשוט ויש פונקציה פה



```
#include <stdio.h>
```

```
#define MAX_ITERATIONS 1000000000
```

```
void foo()  
{  
    int x=0, y=0;  
    x++;  
    y++;  
}
```

```
void main()  
{  
    int i,j,x =0 ,y=0,z=0;
```

```
    for (i=0; i < MAX_ITERATIONS; i++) {  
        // body of foo():  
        int x=0, y=0;  
        x++;  
        y++;  
    }  
}
```

Function Inline Example – cont'd

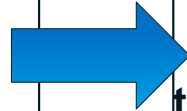
_foo:

```
pushl %ebp
movl %esp, %ebp
subl $8, %esp
movl $0, -4(%ebp)
movl $0, -8(%ebp)
leal -4(%ebp), %eax
incl (%eax)
leal -8(%ebp), %eax
incl (%eax)
leave
ret
```

_main:

L3:

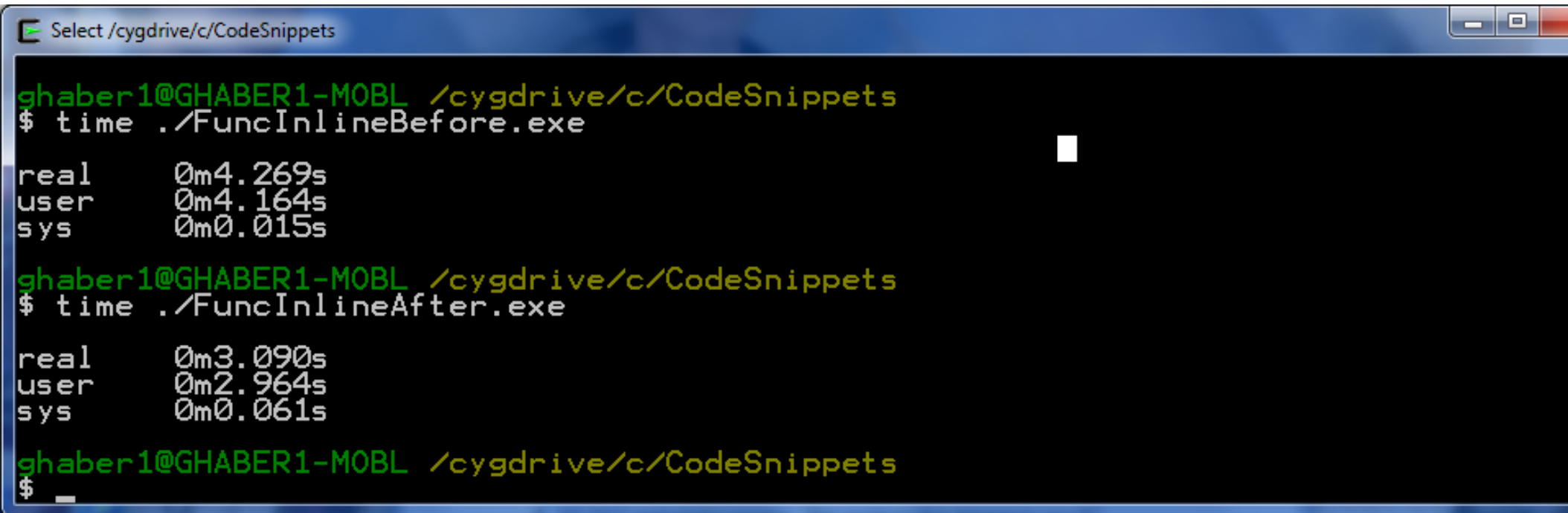
```
cmpl $999999999, -4(%ebp)
jg L2
call _foo
leal -4(%ebp), %eax
incl (%eax)
jmp L3
```



L3:

```
cmpl $999999999, -4(%ebp)
jg L2
# call _foo
pushl %ebp
leal -4(%esp), %ebp # -4(%esp) is because the
call instruction reduces the esp by 4
# and places the return address on the stack
# movl %esp, %ebp
# subl $8, %esp
subl $12, %esp # = subl $8+4, %esp (to keep
the same esp value here as in original __foo)
movl $0, -4(%ebp)
movl $0, -8(%ebp)
leal -4(%ebp), %eax
incl (%eax)
leal -8(%ebp), %eax
incl (%eax)
# leave
# ret
addl $12, %esp # addl $8+4, %esp (restore
%esp)
popl %ebp
leal -4(%ebp), %eax
incl (%eax)
jmp L3
```


Function Inline Example - results



A terminal window titled "Select /cygdrive/c/CodeSnippets" displays the execution times for two programs. The first program, FuncInlineBefore.exe, has a real time of 0m4.269s, user time of 0m4.164s, and system time of 0m0.015s. The second program, FuncInlineAfter.exe, has a real time of 0m3.090s, user time of 0m2.964s, and system time of 0m0.061s. The terminal prompt is ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets.

```
Select /cygdrive/c/CodeSnippets

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncInlineBefore.exe

real    0m4.269s
user    0m4.164s
sys     0m0.015s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncInlineAfter.exe

real    0m3.090s
user    0m2.964s
sys     0m0.061s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$
```

Function Inline Example – cont'd

- If we can verify that the function `__foo()` does not reference the `%esp` register after saving it in the prolog and that there are no internal function calls from within `__foo()`, then we can further improve performance.
- The following transformation improved performance by 32% by removing the need
- to modify the `%esp` reg and use the
`“leal -4(%esp), %ebp”`

L3:

`cmpl $999999999, -4(%ebp)`

`jg L2`

`# call __foo`

`pushl %ebp`

`leal -4(%esp), %ebp`

`# movl %esp, %ebp`

`# subl $8, %esp`

`movl $0, -4(%ebp)` # (need to make sure that the body of `_foo` is not referencing `%esp` and there are no function calls from within `_foo`)

`movl $0, -8(%ebp)`

`leal -4(%ebp), %eax`

`incl (%eax)`

`leal -8(%ebp), %eax`

`incl (%eax)`

`# leave`

`# ret`

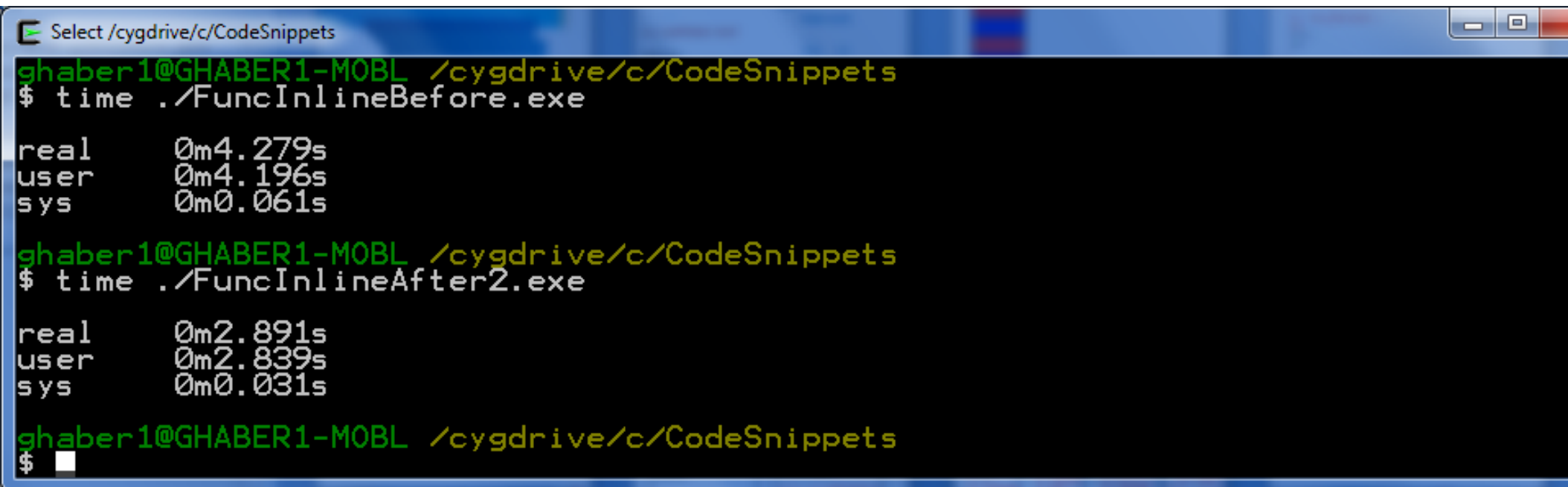
`popl %ebp`

`leal -4(%ebp), %eax`

`incl (%eax)`

`jmp L3`

Function Inline cont'd Example - results



The screenshot shows a Cygwin terminal window with the title bar 'Select /cygdrive/c/CodeSnippets'. The terminal displays the following commands and their execution times:

```
ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncInlineBefore.exe

real    0m4.279s
user    0m4.196s
sys     0m0.061s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncInlineAfter2.exe

real    0m2.891s
user    0m2.839s
sys     0m0.031s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$
```

Program	real	user	sys
./FuncInlineBefore.exe	0m4.279s	0m4.196s	0m0.061s
./FuncInlineAfter2.exe	0m2.891s	0m2.839s	0m0.031s

Function Inline 2nd Example – Original code

```
#include <stdio.h>
```

```
#define MAX_ITERATIONS  
1000000000
```

```
void foo()  
{  
    int x=0, y=0;  
    x++;  
    y++;  
}
```

```
int bar (int i, int j)  
{  
    int x=0, y=0;  
    x++;  
    y++;  
  
    return i+j;  
}
```

```
void main()
```

```
{
```

```
    int i,j,x =0 ,y=0,z=0;
```

```
    for (i=0; i < MAX_ITERATIONS; i++) {
```

```
        foo();
```

```
        y += bar(i,x);
```

```
    }
```

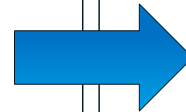
```
}
```

Functions Inline 2nd Example – Original code

```
_main:
...
L4:
    cmpl    $999999999, -
4(%ebp)
    jg      L3
    call    _foo
    movl    -12(%ebp), %eax
    movl    %eax, 4(%esp)
    movl    -4(%ebp), %eax
    movl    %eax, (%esp)
    call    _bar
    movl    %eax, %edx
    leal    -16(%ebp), %eax
    addl    %edx, (%eax)
    movl    -16(%ebp), %eax
    movl    %eax, 4(%esp)
    movl    $LC0, (%esp)
    leal    -4(%ebp), %eax
    incl    (%eax)
    jmp     L4
```

```
_foo:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $0, -4(%ebp)
    movl    $0, -8(%ebp)
    leal    -4(%ebp), %eax
    incl    (%eax)
    leal    -8(%ebp), %eax
    incl    (%eax)
    leave
    ret

_bar:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp
    movl    $0, -4(%ebp)
    movl    $0, -8(%ebp)
    leal    -4(%ebp), %eax
    incl    (%eax)
    leal    -8(%ebp), %eax
    incl    (%eax)
    movl    12(%ebp), %eax
    addl    8(%ebp), %eax
    leave
    ret
```



```
_main:
L4:
    cmpl    $999999999, -
4(%ebp)
    jg      L3
    call    _foo
    movl    -12(%ebp), %eax
    movl    %eax, 4(%esp)
    movl    -4(%ebp), %eax
    movl    %eax, (%esp)

# call _bar
    subl    $4, %esp
```

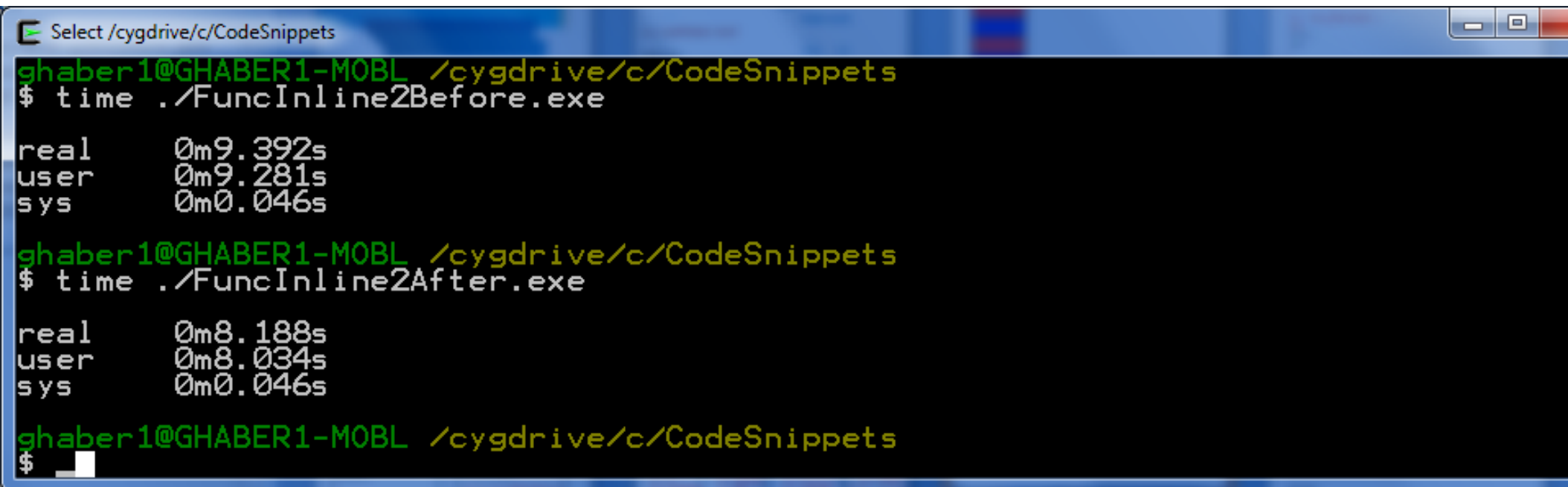
```
CallBar:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $8, %esp

    movl    $0, -4(%ebp)
    movl    $0, -8(%ebp)
    leal    -4(%ebp), %eax
    incl    (%eax)
    leal    -8(%ebp), %eax
    incl    (%eax)
    movl    12(%ebp), %eax
    addl    8(%ebp), %eax
    #leave
    movl    %ebp, %esp
    popl    %ebp
    # ret
    addl    $4, %esp
```

```
RetBar:

    movl    %eax, %edx
    leal    -16(%ebp), %eax
    addl    %edx, (%eax)
    movl    -16(%ebp),
%eax
    movl    %eax, 4(%esp)
    movl    $LC0, (%esp)
    leal    -4(%ebp), %eax
    incl    (%eax)
    jmp     L4
```

Function Inline 2nd Example - results



A terminal window titled "Select /cygdrive/c/CodeSnippets" displays the execution times for two programs. The first program, `FuncInline2Before.exe`, has a real time of 0m9.392s, user time of 0m9.281s, and system time of 0m0.046s. The second program, `FuncInline2After.exe`, has a real time of 0m8.188s, user time of 0m8.034s, and system time of 0m0.046s. The terminal prompt is `ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets`.

```
Select /cygdrive/c/CodeSnippets
ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncInline2Before.exe

real    0m9.392s
user    0m9.281s
sys     0m0.046s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncInline2After.exe

real    0m8.188s
user    0m8.034s
sys     0m0.046s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$
```

Function Inline Example - constrains

problematic function candidates for Inlining:

- Function Call is not to the beginning of a valid function
- Inlined function does not end with a ret instr
- Inlined function has more than one ret instrs
- Inlined function uses indirect calls/jumps
- Inlined function contains direct jumps outside the scope of the function
- Inlined function uses negative displacement from RSP or positive displacement from RBP.
- Stack pointer in the inlined function is not balanced
- More..

Function Specialization Example

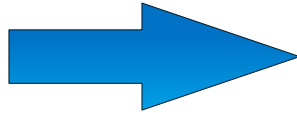
```
int foo(int choice)
{
    switch (choice)
    {
        case 0:
            return 0;

        case 1:
            return 1;

        case 2:
            return 2;
    }
}
```

```
void main()
{
    int i,j = 2,x =0 ,y=0,z=0;

    for (i=0; i < MAX_ITERATIONS; i++) {
        foo(j);
    }
    printf("x=%d y=%d z=%d\n", x,y,z);
}
```



```
int foo_2()
{
    return 2;
}
```

```
int foo(int choice)
{
    switch (choice)
    {
        case 0:
            return 0;

        case 1:
            return 1;

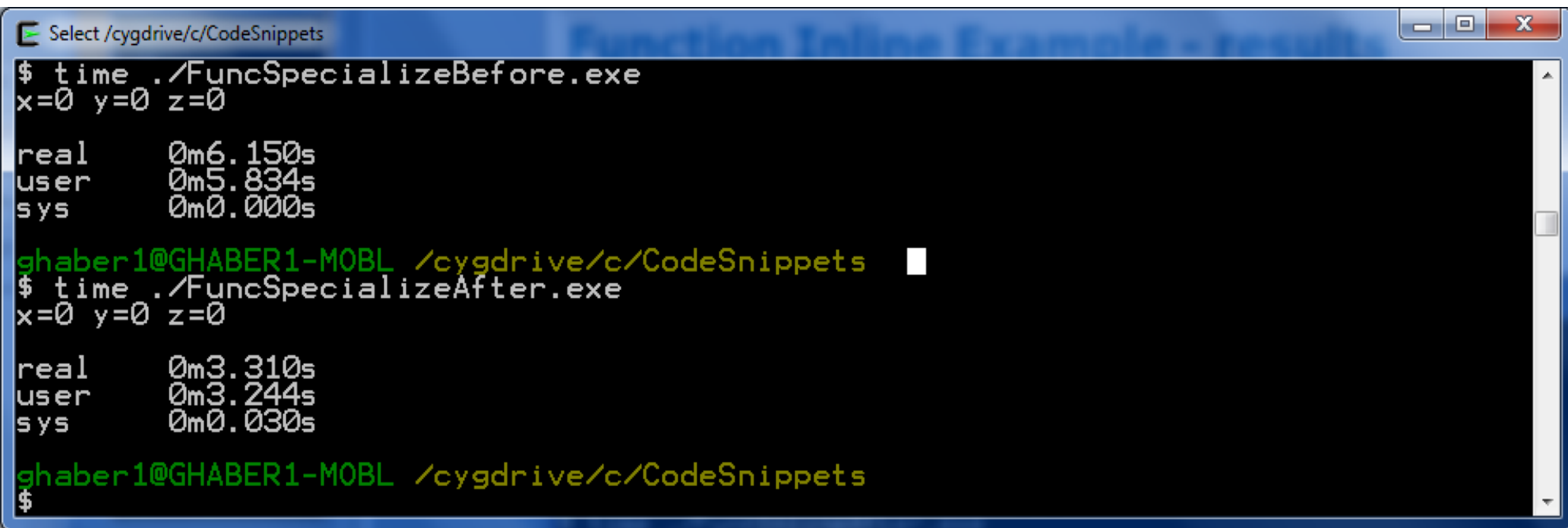
        case 2:
            return 2;
    }
}
```

```
void main()
{
    int i,j = 2,x =0 ,y=0,z=0;

    for (i=0; i < MAX_ITERATIONS; i++) {
        if (j == 2)
            foo_2();
        else
            foo(j);
    }
    printf("x=%d y=%d z=%d\n", x,y,z);
}
```


Function Specialization Example - results

inlining + reordering prob 4x



A terminal window titled "Select /cygdrive/c/CodeSnippets" displays the execution times for two programs. The first program, FuncSpecializeBefore.exe, has a real time of 0m6.150s, user time of 0m5.834s, and system time of 0m0.000s. The second program, FuncSpecializeAfter.exe, has a real time of 0m3.310s, user time of 0m3.244s, and system time of 0m0.030s. The terminal prompt is ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets.

```
$ time ./FuncSpecializeBefore.exe
x=0 y=0 z=0

real    0m6.150s
user    0m5.834s
sys     0m0.000s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$ time ./FuncSpecializeAfter.exe
x=0 y=0 z=0

real    0m3.310s
user    0m3.244s
sys     0m0.030s

ghaber1@GHABER1-MOBL /cygdrive/c/CodeSnippets
$
```