Control-Flow based Optimizations

Gadi Haber

1

Code Reordering

Original code:

```
CMP R8, R9
jl    L1
....
THEN PART
....
L1: CONTINUE PART
....
```

Reordered code:

```
CMP R8, R9
jge   L2
L1: CONTINUE PART
....
L2: ....
THEN PART
....
jmp   L1
```

Improving:

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

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

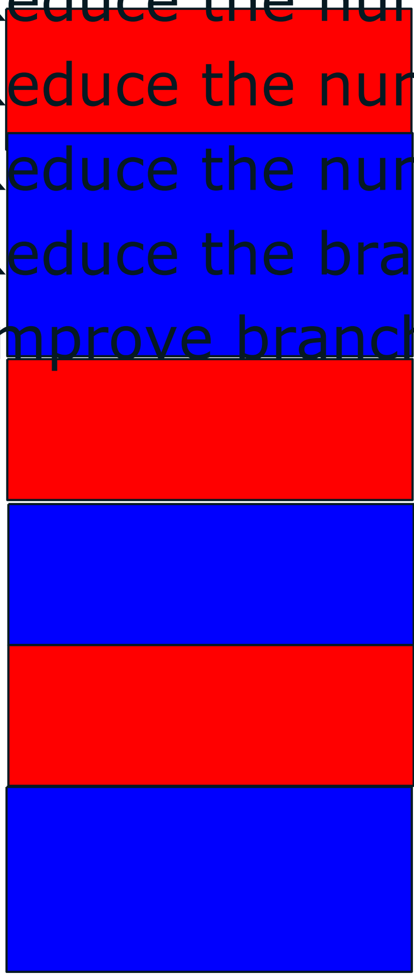
Reduce the number of I-cache misses

Reduce the number of I-TLB misses

Reduce the number of page faults

Reduce the branch penalty

Improve branch prediction



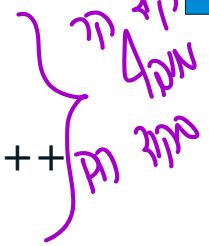
בפרקטיק קודמים נתנו לזיכרון להתפזר

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++;
     }
  }
}
```

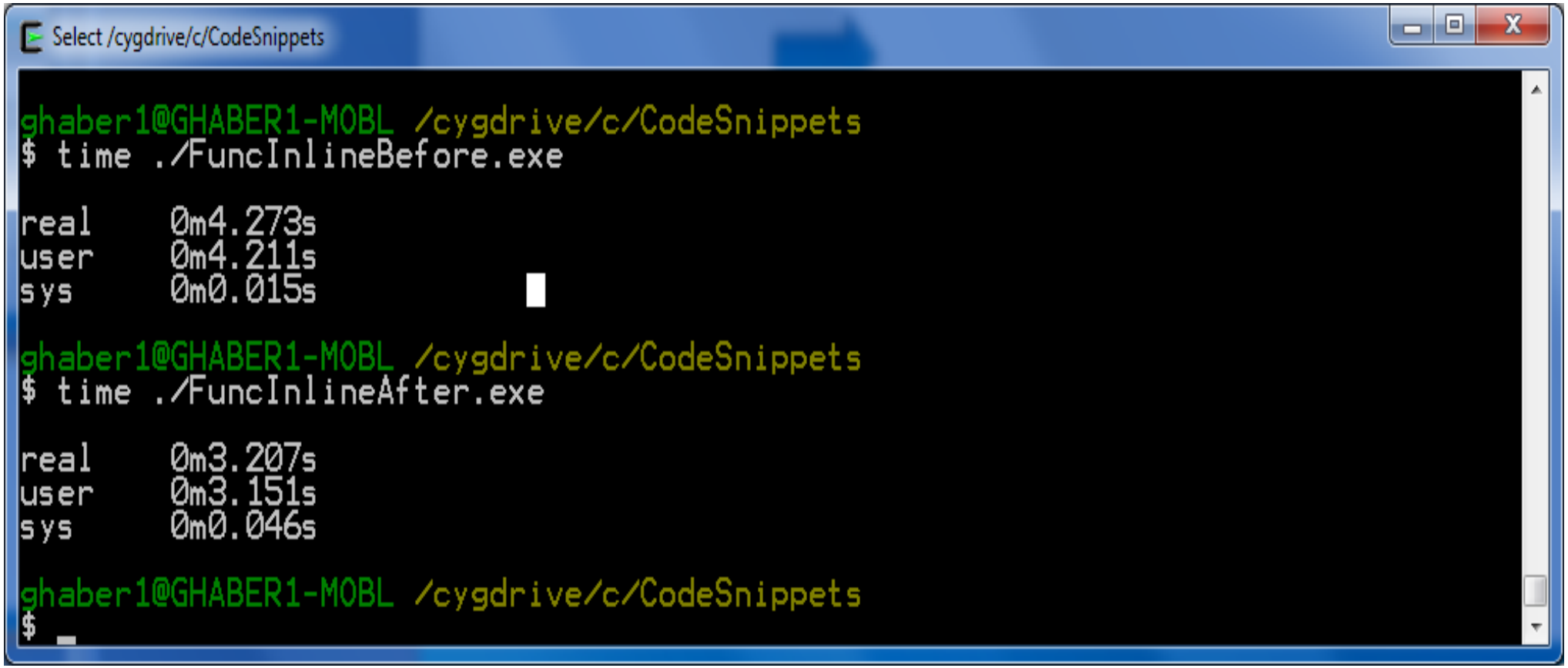
Code Reordering Example – cont'd

```
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
```



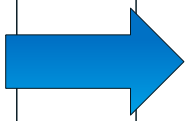
```
L2:
      cmpl      $999999999, -4(%ebp)
      jg    L1
      cmpl      $1000000000, -4(%ebp)
      jg    L5

      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
```

Code Reordering Example - results



Loop Unroll Example – Before and After

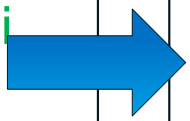
```
#include <stdio.h>

#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

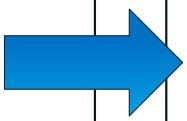
```
L2:
        leal-12(%ebp), %eax
        incl(%eax)
        leal-16(%ebp), %eax
        incl(%eax)
        leal-20(%ebp), %eax
        incl(%eax)
        leal-4(%ebp), %eax // loop var i
        incl(%eax)
        cmpl$999999999, -4(%ebp)
        Jle L2
L3:
```



```
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:
```

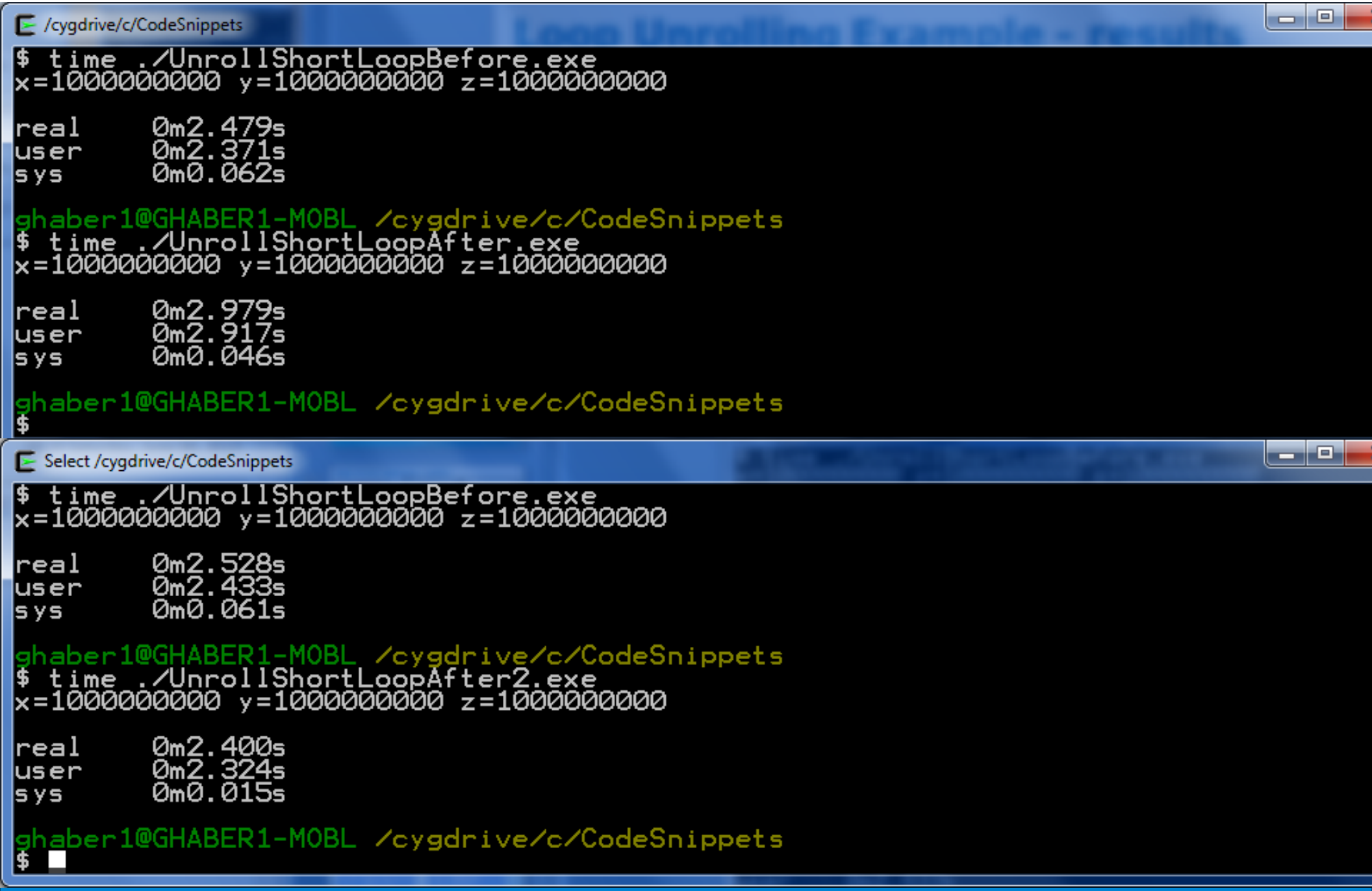
ב4 הלולאה המקורי כדי שנוכל לפרוס ל4 אולי →

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



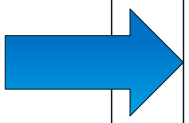
Determining Possible Candidates for Loop Unrolling

Loop Unrolling Pattern:

- Loop is reducable,
  - 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
- **Loop induction variable** is updated by a constant stride

Function Inline Example – Before and After

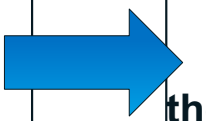
```
#include <stdio.h>

#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();
  }
}
```



```
#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++;
       }
   }
}
```

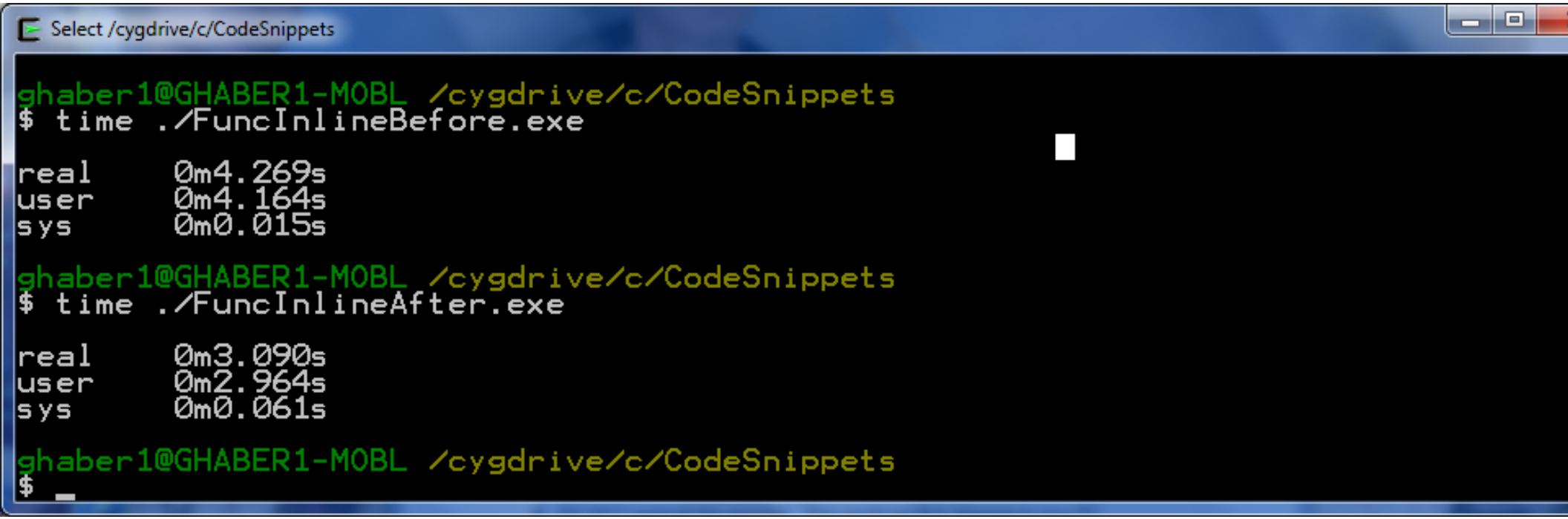
Function Inline Example – cont'd

```
_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

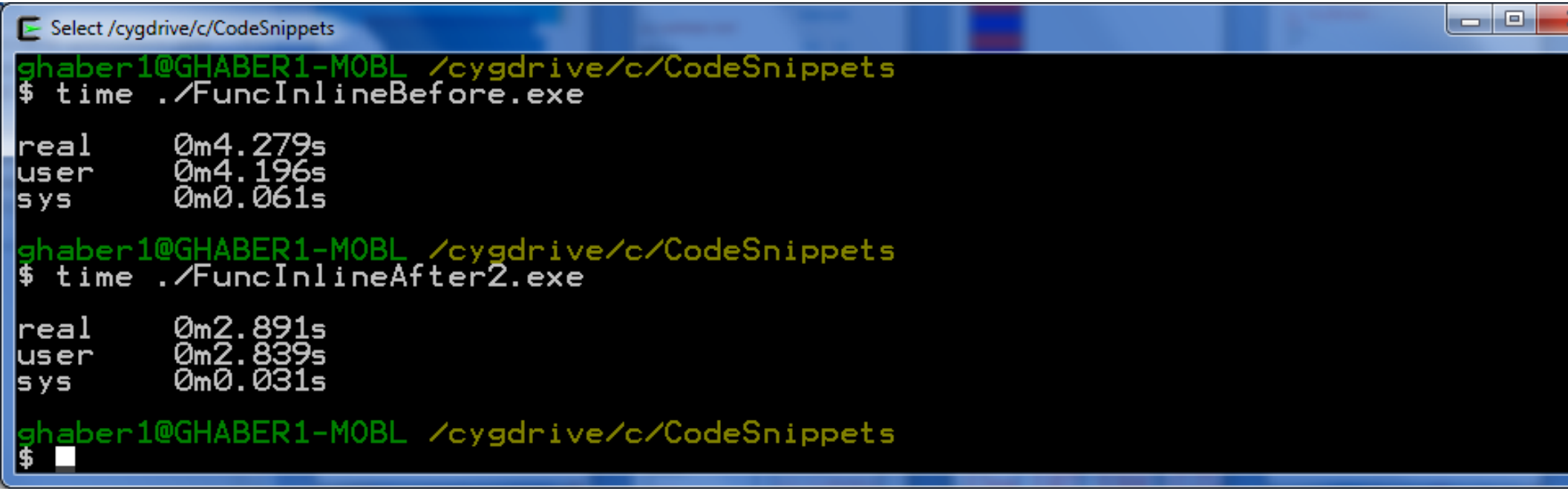


Function Inline Example – cont'd

- 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



Function Inline 2nd Example – Original code

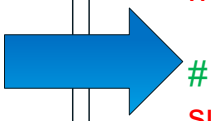
```
#include <stdio.h>

#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
```

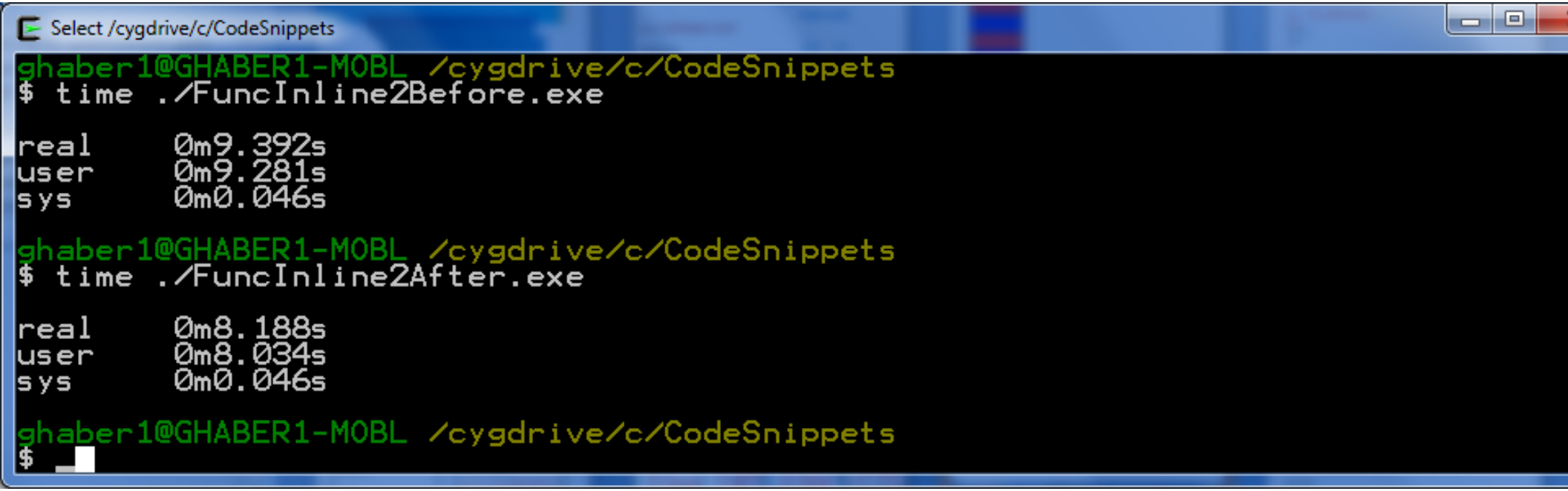
```
_foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    $0, -4(%ebp)
        movl    $0, -8(%ebp)
        leal    -4(%ebp), %eax
        incl    (%eax)
        leal    -8(%ebp), %eax
        incl    (%eax)
        leave
        ret
_bar:
        pushl   %ebp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



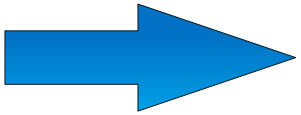
Function Inline Example - constrains

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 ואז reordering בשביל זה