Liya Xu
lx2hy
Section 103
2014/4/8

In-Lab 9

Optimized Code

For the optimization comparison question, I wrote the following C++ code snippet. It uses a loop to calculate the product of x and y, which is defined in the main. The function is called in the main.

```
#include <iostream>
using namespace std;

int product(int x, int y){
    int result = 0;
    for (int i = 0; i < y; i++){
        result += x;
    }

    return result;
}

int main() {
    int x = 5;
    int y = 8;
    int z = product(x, y);
    cout << z << endl;
}
```

Non-optimized code:

```
_Z7productii:
.LFB966:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        sub     esp, 16
        mov     DWORD PTR [ebp-8], 0
        mov     DWORD PTR [ebp-4], 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [ebp+8]
        add     DWORD PTR [ebp-8], eax
```

```
        add     DWORD PTR [ebp-4], 1
.L2:
        mov     eax, DWORD PTR [ebp-4]
        cmp     eax, DWORD PTR [ebp+12]
        setl    al
        test    al, al
        jne     .L3
        mov     eax, DWORD PTR [ebp-8]
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE966:
        .size   _Z7productii, .-_Z7productii
        .globl  main
        .type   main, @function
main:
.LFB967:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        and     esp, -16
        sub     esp, 32
        mov     DWORD PTR [esp+20], 5
        mov     DWORD PTR [esp+24], 8
        mov     eax, DWORD PTR [esp+24]
        mov     DWORD PTR [esp+4], eax
        mov     eax, DWORD PTR [esp+20]
        mov     DWORD PTR [esp], eax
        call    _Z7productii
        mov     DWORD PTR [esp+28], eax
        mov     eax, DWORD PTR [esp+28]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZNSolsEi
        mov     DWORD PTR [esp+4], OFFSET
FLAT:_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        mov     DWORD PTR [esp], eax
        call    _ZNSolsEPFRSoS_E
        mov     eax, 0
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE967:
```

```
        .size   main, .-main
        .type   _Z41__static_initialization_and_destruction_0ii, @function
```

Optimized code:

```
_Z7productii:
.LFB1006:
        .cfi_startproc
        mov     edx, DWORD PTR [esp+8]
        xor     eax, eax
        mov     ecx, DWORD PTR [esp+4]
        imul    ecx, edx
        test    edx, edx
        cmovg   eax, ecx
        ret
        .cfi_endproc
.LFE1006:
        .size   _Z7productii, .-_Z7productii
        .section.text.startup,"ax",@progbits
        .p2align 4,,15
        .globl  main
        .type   main, @function
main:
.LFB1007:
        .cfi_startproc
        push    ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        mov     ebp, esp
        .cfi_def_cfa_register 5
        and     esp, -16
        sub     esp, 16
        mov     DWORD PTR [esp+4], 40
        mov     DWORD PTR [esp], OFFSET FLAT:_ZSt4cout
        call    _ZNSolsEi
        mov     DWORD PTR [esp], eax
        call    _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
        xor     eax, eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1007:
        .size   main, .-main
        .p2align 4,,15
        .type   _GLOBAL__sub_I__Z7productii, @function
```

1. The optimized code looks cleaner and more concise. For the loop optimization, it makes use of more registers other than offsets of ebp. I guess it allows for faster access if using more registers.
2. In the code, I did not use the multiplication to directly multiply x and y together, but rather did it in a loop to add x y times. I also noticed that in the non-optimized code, it involves a lot of jumping around, which is the implementation for the for-loop. In the optimized code, it simplifies the code by directly multiply the two parameters.
3. In the main of the optimized code, it created 16 bytes for the local variables, while in the “normal code it created 32 bytes. I guess this is because in the non-optimized code, it has to create room for the two local variables while in the optimized code it only has one.
4. Comparing the “normal” main code with the optimized main code, I found out that in the optimized code it didn’t call the function at all. It moved the value of the product directly into [esp+4]. In my opinion, this is a kind of optimization that optimizes the program’s execution speed.