Introduction To Computer Systems
Ex. 3 Pen and Paper

1 Overview

1.1 Confirmation of Concepts

1) Briefly describe the optimization methods presented below.

a) Code Motion

ANS: Move Code to reduce frequency with which computation performed.

b) Strength Reduction

ANS: Replace costly operation with simpler one.

c) Sharing of common subexpressions

ANS: Unify repeated expressions into one.

d) Remove Aliasing

ANS: Remove memory aliasing that causes code updates to occur every iteration.

e) Function Inlining

ANS: Make the process of replacing a subroutine or function call at the call site with the body of the subroutine or function being called.

f) Loop Unrolling

ANS: Removing unnecessary loops, or removing loops when it is faster not to iterate them.

2) Answer the following questions.

a) Why can't the compiler move the procedure in the loop for optimization? Give two reasons.

ANS:
1. The procedure may generate a side effect.
2. Different values can be returned even if the same arguments are delivered.

b) What are the limitations of optimizing the compiler? Give two things.

ANS:
1. When in doubt, the compiler must be conservative.
2. Most analysis is performed only within procedures

2 Code Optimization

2.1 Counting Instruction

1) Look at the given code and the corresponding assembly code, and answer the question.

*In the case of an instruction such as jge, it is considered to have been executed even if it is ignored because the conditions are not satisfied.

```
for(i=1;i<N+1;i++)
    rlt += i;
```

```
        movl    $1, -4(%rbp)
        jmp     .L2
.L3:
        movl    -4(%rbp), %eax
        addl    %eax, -8(%rbp)
        addl    $1, -4(%rbp)
.L2:
        movl    -20(%rbp), %eax
        cmpl    -4(%rbp), %eax
        jge     .L3
```

a) if N = 1, How many instructions are executed?

ANS: 11

b) if N = 3, How many instructions are executed?

ANS: 23

c) Find a function for N that calculates the number of instructions executed for the natural number N.

ANS: $F(n) = 6n + 5$

2) The following is a C code representing two algorithms with different complexities for printing all partial sums of a first-order array, and the corresponding assembly code. Answer the questions.

1. O(n^3) Algorithm

```
for(i=1;i<N+1;i++) {
    for(j=0;j<N+1-i;j++) {
        rlt = 0;
        for(k=0;k<i;k++) {
            rlt += arr[j+k];
        }
        //printf("%d\n", rlt);
    }
}
```

```
        movl    $1, -4(%rbp)
        jmp     .L2
.L7:
        movl    $0, -8(%rbp)
        jmp     .L3
.L6:
        movl    $0, -16(%rbp)
        movl    $0, -12(%rbp)
        jmp     .L4
.L5:
        movl    -8(%rbp), %edx
        movl    -12(%rbp), %eax
        addl    %edx, %eax
        cltq
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %eax
        addl    %eax, -16(%rbp)
        addl    $1, -12(%rbp)
.L4:
        movl    -12(%rbp), %eax
        cmpl    -4(%rbp), %eax
        jl      .L5
        addl    $1, -8(%rbp)
.L3:
        movl    -28(%rbp), %eax
        addl    $1, %eax
        subl    -4(%rbp), %eax
        cmpl    %eax, -8(%rbp)
        jl      .L6
        addl    $1, -4(%rbp)
.L2:
        movl    -28(%rbp), %eax
        cmpl    -4(%rbp), %eax
        jge     .L7
```

a) if N = 1, How many instructions are executed?

ANS: 41

b) if N = 3, How many instructions are executed?

ANS: 240

c) Find a function for N that calculates the number of instructions executed for the natural number N.

ANS: $F(N) = \frac{13}{6}N^3 + \frac{25}{2}N^2 + \frac{64}{3}N + 5$

2. O(n^2) Algorithm

```
prefix[0] = 0;
for (i = 1; i < N+1; i++) {
    prefix[i] = prefix[i - 1] + arr[i-1];
}
for (i = 0; i < N; i++) {
    for (j = i; j < N; j++) {
        rlt = prefix[j + 1] - prefix[i];
        //printf("%d\n", rlt);
    }
}
```

```
        movq    -32(%rbp), %rax
        movl    $0, (%rax)
        movl    $1, -4(%rbp)
        jmp     .L2
.L3:
        movl    -4(%rbp), %eax
        cltq
        salq    $2, %rax
        leaq    -4(%rax), %rdx
        movq    -32(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %ecx
        movl    -4(%rbp), %eax
        cltq
        salq    $2, %rax
        leaq    -4(%rax), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %edx
        movl    -4(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rsi
        movq    -32(%rbp), %rax
        addq    %rsi, %rax
        addl    %ecx, %edx
        movl    %edx, (%rax)
        addl    $1, -4(%rbp)
```

```
.L2:
        movl    -36(%rbp), %eax
        cmpl    -4(%rbp), %eax
        jge     .L3
        movl    $0, -4(%rbp)
        jmp     .L4
.L7:
        movl    -4(%rbp), %eax
        movl    %eax, -8(%rbp)
        jmp     .L5
.L6:
        movl    -8(%rbp), %eax
        cltq
        addq    $1, %rax
        leaq    0(,%rax,4), %rdx
        movq    -32(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %edx
        movl    -4(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rcx
        movq    -32(%rbp), %rax
        addq    %rcx, %rax
        movl    (%rax), %eax
        subl    %eax, %edx
        movl    %edx, -12(%rbp)
        addl    $1, -8(%rbp)
.L5:
        movl    -8(%rbp), %eax
        cmpl    -36(%rbp), %eax
        jl      .L6
        addl    $1, -4(%rbp)
.L4:
        movl    -4(%rbp), %eax
        cmpl    -36(%rbp), %eax
        jl      .L7
```

a) if N = 1, How many instructions are executed?

ANS: 66

b) if N = 3, How many instructions are executed?

ANS: 231

c) Find a function for N that calculates the number of instructions executed for the natural number N.

ANS: $F(N) = \frac{19}{2}N^2 + \frac{89}{2}N + 12$

d) Assuming that N is large enough, is an algorithm with complexity O(n^2) more efficient than an algorithm with complexity O(n^3) in terms of the number of instructions executed?

ANS: If N is large enough, the value of F(N) for the O(n^2) algorithm is much smaller than the value of F(N) for the O(n^3) algorithm. Thus, in terms of the number of instructions executed, the O(N^2) algorithm is much more efficient.

2.2 Strength Reduction

1) Answer the questions using the table below.
* assume that y is integer.

Operation	**Cycle**
Integer add	1
Integer Multiply	4
Integer Divide	36
Floating-point add	3
Floating-point multiply	5
Floating-point divide	38

Table 1

a) Which is faster, y / 10 or y * 0.1?

ANS: y * 0.1

b) Which is faster, y / 0.1 or y * 10?

ANS: y * 10

c) Calculate the total cycle to compute two expressions, (y / 0.1) + (y * 0.1) and (y * 10) + (y / 10). Compare and explain which is more efficient.

ANS:
(y / 0.1) + (y * 0.1) Total Cycle : 38 + 5 + 3 = 46
(y * 10) + (y / 10) Total Cycle : 4 + 36 + 1 = 41
Thus, (y * 10) + (y / 10) is more efficient.

2.3 Other Methods

1) Look at the code presented below and optimize it for the given optimization method.

1. Use Code Motion & Remove Aliasing

```
for (i=0;i<100;i++) {
    for (j=0;j<100;j++)
        arr[i] += 5*j + i*i;
}
```

ANS:

```
for(i=0;i<100;i++) {
    temp = i*i;
    int_val = 0;
    for(j=0;j<100;j++) {
        int_val += 5*j + temp;
    }
    arr[i] = int_val;
}
```

2. Use Function Inlining

```
int max(int a, int b) {
    return (a > b) ? a : b;
}
a = max(x,y);
```

ANS:

```
if (x > y)
    a = x;
else
    a = y;
```

3. Use Loop Unrolling

```
for (i=0;i<3;i++)
    ab[i] = i;
```

ANS:

```
ab[0] = 0;
ab[1] = 1;
ab[2] = 2;
```

4. Use Sharing of common subexpressions

```
a = b * c - c * d;
e = (b - d) * (b - d);
```

ANS:

```
temp = b - d;
a = temp * c;
e = temp * temp;
```