ECE565 Homework1 Report

Yisong Zou
NetId: yz558

1. (30 points) Code Optimization – “Beat the Compiler”.

(a)

Results when using gcc -O2 -o loop_performance loop_performance.c

gcc -O2

Ten Runs	Argument(10000000)(seven 0s)	Argument(100000000)(eight 0s)
1st(milliseconds)	27.322	268.833
2nd(milliseconds)	26.391	258.334
3rd(milliseconds)	26.624	258.541
4th(milliseconds)	26.996	268.294
5th(milliseconds)	27.273	259.977
6th(milliseconds)	28.181	266.166
7th(milliseconds)	26.739	264.944
8th(milliseconds)	26.963	266.627
9th(milliseconds)	26.789	264.117
10th(milliseconds)	26.486	267.082
Shortest Time	**26.391**	**258.334**

Results when using gcc -O3 -o loop_performance loop_performance.c

gcc -O3

Ten Runs	Argument(10000000)(seven 0s)	Argument(100000000)(eight 0s)
1st(milliseconds)	21.923	214.149
2nd(milliseconds)	23.527	208.560
3rd(milliseconds)	22.039	216.123
4th(milliseconds)	22.158	207.204
5th(milliseconds)	21.986	217.395
6th(milliseconds)	21.906	212.907
7th(milliseconds)	21.450	216.722
8th(milliseconds)	21.212	212.022
9th(milliseconds)	21.891	207.504
10th(milliseconds)	21.978	215.260
Shortest Time	**21.212**	**207.204**

(b) Describe the type of machine and environment you are running on

1) processor architecture: x86-64
2) CPU frequency: CPU MHz: 2300.000
3) OS type: Ubuntu 18.04.5 LTS
4)It is inside duke VM

(c) Study the code in the do_loops() function.

Loop Unrolling

Code:

```
void do_loops(int *a, int *b, int *c, int N) {
  int i;
  for (i = N - 1; i >= 1; i -= 9) {
    a[i] = a[i] + 1;
    a[i - 1] = a[i - 1] + 1;
    a[i - 2] = a[i - 2] + 1;
    a[i - 3] = a[i - 3] + 1;
    a[i - 4] = a[i - 4] + 1;
    a[i - 5] = a[i - 5] + 1;
    a[i - 6] = a[i - 6] + 1;
    a[i - 7] = a[i - 7] + 1;
    a[i - 8] = a[i - 8] + 1;
  }
  for (i = 1; i < N; i += 9) {
    b[i] = a[i + 1] + 3;
    b[i + 1] = a[i + 2] + 3;
    b[i + 2] = a[i + 3] + 3;
    b[i + 3] = a[i + 4] + 3;
    b[i + 4] = a[i + 5] + 3;
    b[i + 5] = a[i + 6] + 3;
    b[i + 6] = a[i + 7] + 3;
    b[i + 7] = a[i + 8] + 3;
    b[i + 8] = a[i + 9] + 3;
  }
  for (i = 1; i < N; i += 9) {
    c[i] = b[i - 1] + 2;
    c[i + 1] = b[i] + 2;
    c[i + 2] = b[i + 1] + 2;
    c[i + 3] = b[i + 2] + 2;
    c[i + 4] = b[i + 3] + 2;
```

```
    c[i + 5] = b[i + 4] + 2;
    c[i + 6] = b[i + 5] + 2;
    c[i + 7] = b[i + 6] + 2;
    c[i + 8] = b[i + 7] + 2;
  }
}
```

gcc -O2 with Loop Unrolling

Ten Runs	Argument(10000000)(seven 0s)	Argument(100000000)(eight 0s)
1st(milliseconds)	23.937	235.889
2nd(milliseconds)	24.946	238.672
3rd(milliseconds)	23.590	226.480
4th(milliseconds)	24.058	222.157
5th(milliseconds)	23.796	236.752
6th(milliseconds)	23.746	227.602
7th(milliseconds)	24.344	225.330
8th(milliseconds)	24.459	226.794
9th(milliseconds)	24.585	232.023
10th(milliseconds)	24.009	227.108
Shortest Time	23.590	222.157

gcc -O3 with Loop Unrolling

Ten Runs	Argument(10000000)(seven 0s)	Argument(100000000)(eight 0s)
1st(milliseconds)	24.180	226.786
2nd(milliseconds)	23.978	227.899
3rd(milliseconds)	24.409	226.480
4th(milliseconds)	24.450	226.634
5th(milliseconds)	23.796	227.372
6th(milliseconds)	23.746	227.602
7th(milliseconds)	24.176	227.249
8th(milliseconds)	23.175	227.116
9th(milliseconds)	24.262	226.546
10th(milliseconds)	23.702	227.108
Shortest Time	23.175	226.480

Discuss: (Does it match your expectation? What reason(s) might explain the result?)

According to the output of objdump as the follows, the unrolled version ahs more instructions done in a single loop and this will result in fewer loop management instructions. This matches my expectation, as unrolling reduce instruction count by make fewer loops, then there will be less loop management

instructions, it will run faster in gcc -O2. However, as -O3 has already have this optimization, it does not show improvement, and seems make it a little worse.

objdump results for original code(-O2)(Each color block represents a loop)

```
a75:   7e 2d                    jle    aa4 <do_loops+0x34>
a77:   48 98                    cltq
a79:   44 8d 49 fe              lea    -0x2(%rcx),%r9d
a7d:   4c 8d 04 85 00 00 00     lea    0x0(,%rax,4),%r8
a84:   00
a85:   49 c1 e1 02              shl    $0x2,%r9
a89:   4a 8d 04 07              lea    (%rdi,%r8,1),%rax
a8d:   4e 8d 44 07 fc           lea    -0x4(%rdi,%r8,1),%r8
a92:   4d 29 c8                 sub    %r9,%r8
a95:   0f 1f 00                 nopl   (%rax)
a98:   83 00 01                 addl   $0x1,(%rax)
a9b:   48 83 e8 04              sub    $0x4,%rax
a9f:   4c 39 c0                 cmp    %r8,%rax
aa2:   75 f4                    jne    a98 <do_loops+0x28>
aa4:   83 f9 01                 cmp    $0x1,%ecx
aa7:   7e 4a                    jle    af3 <do_loops+0x83>
aa9:   8d 41 fe                 lea    -0x2(%rcx),%eax
aac:   4c 8d 04 85 04 00 00     lea    0x4(,%rax,4),%r8
ab3:   00
ab4:   31 c0                    xor    %eax,%eax
ab6:   66 2e 0f 1f 84 00 00     nopw   %cs:0x0(%rax,%rax,1)
abd:   00 00 00
ac0:   8b 4c 07 08              mov    0x8(%rdi,%rax,1),%ecx
ac4:   83 c1 03                 add    $0x3,%ecx
ac7:   89 4c 06 04              mov    %ecx,0x4(%rsi,%rax,1)
acb:   48 83 c0 04              add    $0x4,%rax
acf:   49 39 c0                 cmp    %rax,%r8
ad2:   75 ec                    jne    ac0 <do_loops+0x50>
ad4:   31 c9                    xor    %ecx,%ecx
ad6:   66 2e 0f 1f 84 00 00     nopw   %cs:0x0(%rax,%rax,1)
add:   00 00 00
ae0:   8b 3c 0e                 mov    (%rsi,%rcx,1),%edi
ae3:   83 c7 02                 add    $0x2,%edi
ae6:   89 7c 0a 04              mov    %edi,0x4(%rdx,%rcx,1)
aea:   48 83 c1 04              add    $0x4,%rcx
aee:   48 39 c1                 cmp    %rax,%rcx
af1:   75 ed                    jne    ae0 <do_loops+0x70>
```

objdump results for unrolling code(-O2) (Each color block represents a loop)

```
a88:   41 83 e8 09              sub    $0x9,%r8d
a8c:   83 40 20 01              addl   $0x1,0x20(%rax)
a90:   83 40 1c 01              addl   $0x1,0x1c(%rax)
a94:   83 40 18 01              addl   $0x1,0x18(%rax)
a98:   83 40 14 01              addl   $0x1,0x14(%rax)
a9c:   83 40 10 01              addl   $0x1,0x10(%rax)
aa0:   83 40 0c 01              addl   $0x1,0xc(%rax)
aa4:   83 40 08 01              addl   $0x1,0x8(%rax)
aa8:   83 40 04 01              addl   $0x1,0x4(%rax)
aac:   83 00 01                 addl   $0x1,(%rax)
aaf:   48 83 e8 24              sub    $0x24,%rax
ab3:   45 85 c0                 test   %r8d,%r8d
ab6:   7f d0                    jg     a88 <do_loops+0x18>
ab8:   83 f9 01                 cmp    $0x1,%ecx
abb:   0f 8e f6 00 00 00        jle    bb7 <do_loops+0x147>
ac1:   48 8d 46 04              lea    0x4(%rsi),%rax
ac5:   48 83 c7 08              add    $0x8,%rdi
ac9:   41 b9 01 00 00 00        mov    $0x1,%r9d
acf:   90                       nop
ad0:   44 8b 17                 mov    (%rdi),%r10d
ad3:   41 83 c1 09              add    $0x9,%r9d
ad7:   48 83 c7 24              add    $0x24,%rdi
adb:   48 83 c0 24              add    $0x24,%rax
adf:   45 8d 42 03              lea    0x3(%r10),%r8d
ae3:   44 89 40 dc              mov    %r8d,-0x24(%rax)
ae7:   44 8b 5f e0              mov    -0x20(%rdi),%r11d
aeb:   45 8d 43 03              lea    0x3(%r11),%r8d
aef:   44 89 40 e0              mov    %r8d,-0x20(%rax)
af3:   44 8b 57 e4              mov    -0x1c(%rdi),%r10d
af7:   45 8d 42 03              lea    0x3(%r10),%r8d
afb:   44 89 40 e4              mov    %r8d,-0x1c(%rax)
aff:   44 8b 5f e8              mov    -0x18(%rdi),%r11d
b03:   45 8d 43 03              lea    0x3(%r11),%r8d
b07:   44 89 40 e8              mov    %r8d,-0x18(%rax)
b0b:   44 8b 57 ec              mov    -0x14(%rdi),%r10d
b0f:   45 8d 42 03              lea    0x3(%r10),%r8d
b13:   44 89 40 ec              mov    %r8d,-0x14(%rax)
b17:   44 8b 5f f0              mov    -0x10(%rdi),%r11d
b1b:   45 8d 43 03              lea    0x3(%r11),%r8d
b1f:   44 89 40 f0              mov    %r8d,-0x10(%rax)
b23:   44 8b 57 f4              mov    -0xc(%rdi),%r10d
b27:   45 8d 42 03              lea    0x3(%r10),%r8d
b2b:   44 89 40 f4              mov    %r8d,-0xc(%rax)
```

```
b2f:    44 8b 5f f8             mov    -0x8(%rdi),%r11d
b33:    45 8d 43 03             lea    0x3(%r11),%r8d
b37:    44 89 40 f8             mov    %r8d,-0x8(%rax)
b3b:    44 8b 57 fc             mov    -0x4(%rdi),%r10d
b3f:    45 8d 42 03             lea    0x3(%r10),%r8d
b43:    44 89 40 fc             mov    %r8d,-0x4(%rax)
b47:    44 39 c9                cmp    %r9d,%ecx
b4a:    7f 84                   jg     ad0 <do_loops+0x60>
b4c:    48 83 c2 04             add    $0x4,%rdx
b50:    bf 01 00 00 00          mov    $0x1,%edi
b55:    0f 1f 00                nopl   (%rax)
b58:    8b 06                   mov    (%rsi),%eax
b5a:    83 c7 09                add    $0x9,%edi
b5d:    48 83 c6 24             add    $0x24,%rsi
b61:    48 83 c2 24             add    $0x24,%rdx
b65:    83 c0 02                add    $0x2,%eax
b68:    89 42 dc                mov    %eax,-0x24(%rdx)
b6b:    8b 46 e0                mov    -0x20(%rsi),%eax
b6e:    83 c0 02                add    $0x2,%eax
b71:    89 42 e0                mov    %eax,-0x20(%rdx)
b74:    8b 46 e4                mov    -0x1c(%rsi),%eax
b77:    83 c0 02                add    $0x2,%eax
b7a:    89 42 e4                mov    %eax,-0x1c(%rdx)
b7d:    8b 46 e8                mov    -0x18(%rsi),%eax
b80:    83 c0 02                add    $0x2,%eax
b83:    89 42 e8                mov    %eax,-0x18(%rdx)
b86:    8b 46 ec                mov    -0x14(%rsi),%eax
b89:    83 c0 02                add    $0x2,%eax
b8c:    89 42 ec                mov    %eax,-0x14(%rdx)
b8f:    8b 46 f0                mov    -0x10(%rsi),%eax
b92:    83 c0 02                add    $0x2,%eax
b95:    89 42 f0                mov    %eax,-0x10(%rdx)
b98:    8b 46 f4                mov    -0xc(%rsi),%eax
b9b:    83 c0 02                add    $0x2,%eax
b9e:    89 42 f4                mov    %eax,-0xc(%rdx)
ba1:    8b 46 f8                mov    -0x8(%rsi),%eax
ba4:    83 c0 02                add    $0x2,%eax
ba7:    89 42 f8                mov    %eax,-0x8(%rdx)
baa:    8b 46 fc                mov    -0x4(%rsi),%eax
bad:    83 c0 02                add    $0x2,%eax
bb0:    89 42 fc                mov    %eax,-0x4(%rdx)
bb3:    39 f9                   cmp    %edi,%ecx
bb5:    7f a1                   jg     b58 <do_loops+0xe8>
```

Loop Fusion

Code:

```
void do_loops(int *a, int *b, int *c, int N) {
  int i;
  for (i = N - 1; i >= 1; i--) {
    a[i] = a[i] + 1;
  }
  for (i = 1; i < N; i++) {
    b[i] = a[i + 1] + 3;
    c[i] = b[i - 1] + 2;
  }
}
```

gcc -O2 with Loop Fusion

Argument(10000000)(seven 0s)(Shortest 25.574)

```
Time=25.574000 milliseconds
Time=26.018000 milliseconds
Time=25.936000 milliseconds
Time=26.117000 milliseconds
Time=26.006000 milliseconds
Time=25.869000 milliseconds
Time=25.853000 milliseconds
Time=26.084000 milliseconds
Time=25.948000 milliseconds
Time=25.891000 milliseconds
```

gcc -O2 with Loop Fusion

Argument(100000000)(eight 0s) (Shortest 263.761)

```
Time=272.514000 milliseconds
Time=266.591000 milliseconds
Time=263.761000 milliseconds
Time=265.740000 milliseconds
Time=268.265000 milliseconds
Time=268.219000 milliseconds
Time=265.682000 milliseconds
Time=275.816000 milliseconds
Time=264.860000 milliseconds
Time=274.635000 milliseconds
```

gcc -O3 with Loop Fusion
Argument(10000000)(seven 0s)(Shortest 31.054)

```
Time=31.582000 milliseconds
Time=32.287000 milliseconds
Time=31.580000 milliseconds
Time=31.054000 milliseconds
Time=31.968000 milliseconds
Time=31.666000 milliseconds
Time=32.605000 milliseconds
Time=31.840000 milliseconds
Time=31.577000 milliseconds
Time=32.022000 milliseconds
```

gcc -O3 with Loop Fusion
Argument(100000000)(eight 0s)(Shortest 306.348)

```
Time=331.303000 milliseconds
Time=314.896000 milliseconds
Time=306.348000 milliseconds
Time=313.078000 milliseconds
Time=331.573000 milliseconds
Time=312.558000 milliseconds
Time=323.960000 milliseconds
Time=317.125000 milliseconds
Time=310.286000 milliseconds
Time=314.238000 milliseconds
```

Discuss: (Does it match your expectation? What reason(s) might explain the result?)

According to the objdump results as the following, this matches my expectation, as fusion reduce instruction count by make fewer loop management instructions, it should run faster. For -O2 it is faster and for -O3 it is much slower, I think this might because -O3 has better optimization than fusion or it already has fusion.

objdump results for fusion code(-O2) (Each color block represents a loop)

```
0000000000000a70 <do_loops>:
 a70:   8d 41 ff                lea    -0x1(%rcx),%eax
 a73:   85 c0                   test   %eax,%eax
 a75:   7e 2d                   jle    aa4 <do_loops+0x34>
 a77:   48 98                   cltq
 a79:   44 8d 49 fe             lea    -0x2(%rcx),%r9d
 a7d:   4c 8d 04 85 00 00 00    lea    0x0(,%rax,4),%r8
```

```
a84:    00
a85:    49 c1 e1 02             shl    $0x2,%r9
a89:    4a 8d 04 07             lea    (%rdi,%r8,1),%rax
a8d:    4e 8d 44 07 fc          lea    -0x4(%rdi,%r8,1),%r8
a92:    4d 29 c8                sub    %r9,%r8
a95:    0f 1f 00                nopl   (%rax)
a98:    83 00 01                addl   $0x1,(%rax)
a9b:    48 83 e8 04             sub    $0x4,%rax
a9f:    4c 39 c0                cmp    %r8,%rax
aa2:    75 f4                   jne    a98 <do_loops+0x28>
aa4:    83 f9 01                cmp    $0x1,%ecx
aa7:    7e 34                   jle    add <do_loops+0x6d>
aa9:    8d 41 fe                lea    -0x2(%rcx),%eax
aac:    4c 8d 04 85 08 00 00    lea    0x8(,%rax,4),%r8
ab3:    00
ab4:    b8 04 00 00 00          mov    $0x4,%eax
ab9:    0f 1f 80 00 00 00 00    nopl   0x0(%rax)
ac0:    8b 4c 07 04             mov    0x4(%rdi,%rax,1),%ecx
ac4:    83 c1 03                add    $0x3,%ecx
ac7:    89 0c 06                mov    %ecx,(%rsi,%rax,1)
aca:    8b 4c 06 fc             mov    -0x4(%rsi,%rax,1),%ecx
ace:    83 c1 02                add    $0x2,%ecx
ad1:    89 0c 02                mov    %ecx,(%rdx,%rax,1)
ad4:    48 83 c0 04             add    $0x4,%rax
ad8:    4c 39 c0                cmp    %r8,%rax
adb:    75 e3                   jne    ac0 <do_loops+0x50>
add:    f3 c3                   repz retq
adf:    90                      nop
```

Loop Strip Mining

Code:

```c
void do_loops(int *a, int *b, int *c, int N) {
  int i;
  int j;
  for (j = N - 1; j >= 1; j -= 9) {
    for (i = j; i > j - 9; i--) {
      a[i] = a[i] + 1;
    }
  }
  for (j = 1; j < N; j += 9) {
    for (i = j; i < j + 9; i++) {
```

```
      b[i] = a[i + 1] + 3;
    }
  }
  for (j = 1; j < N; j += 9) {
    for (i = j; i < j + 9; i++) {
      c[i] = b[i - 1] + 2;
    }
  }
}
```

gcc -O2 with Loop Strip Mining
Argument(10000000)(seven 0s)(Shortest 31.956)

```
Time=32.337000 milliseconds
Time=32.175000 milliseconds
Time=32.877000 milliseconds
Time=32.511000 milliseconds
Time=32.330000 milliseconds
Time=32.358000 milliseconds
Time=32.070000 milliseconds
Time=32.057000 milliseconds
Time=32.152000 milliseconds
Time=31.956000 milliseconds
```

gcc -O2 with Loop Strip Mining
Argument(100000000)(eight 0s)(Shortest 313.814)

```
Time=319.009000 milliseconds
Time=319.301000 milliseconds
Time=329.217000 milliseconds
Time=315.710000 milliseconds
Time=313.814000 milliseconds
Time=321.578000 milliseconds
Time=317.843000 milliseconds
Time=316.670000 milliseconds
Time=315.294000 milliseconds
Time=314.180000 milliseconds
```

gcc -O3 with Loop Strip Mining
Argument(10000000)(seven 0s)(Shortest 22.863)

```
Time=23.165000 milliseconds
```

```
Time=23.554000 milliseconds
Time=23.611000 milliseconds
Time=23.617000 milliseconds
Time=23.122000 milliseconds
Time=24.354000 milliseconds
Time=22.863000 milliseconds
Time=27.051000 milliseconds
Time=23.004000 milliseconds
Time=23.087000 milliseconds
```

gcc -O3 with Loop Strip Mining
Argument(100000000)(eight 0s)(Shortest 222.123)

```
Time=230.581000 milliseconds
Time=235.131000 milliseconds
Time=233.066000 milliseconds
Time=233.635000 milliseconds
Time=223.609000 milliseconds
Time=222.737000 milliseconds
Time=222.123000 milliseconds
Time=225.214000 milliseconds
Time=222.586000 milliseconds
Time=223.465000 milliseconds
```

Discuss: (Does it match your expectation? What reason(s) might explain the result?)

According to the objdump result(too long so I will just describe it here), this matches my expectation, as under this circumstance, mining cannot benefit from vectorization, it only adds many useless loop management instructions and more level of useless loops. For -O3 it is a little slower and for -O2 it is much slower, this is because this mining lead to more overhead of loop management instructions.

(d) Discuss whether you can beat the compiler

From the (C) part, I can beat the computer only in -O2 with loop unrolling, the statistics are as the following.

My Code Results with Unrolling

gcc -O2 with Loop Unrolling

Ten Runs	Argument(10000000)(seven 0s)	Argument(100000000)(eight 0s)
1st(milliseconds)	23.937	235.889
2nd(milliseconds)	24.946	238.672

3rd(milliseconds)	23.590	226.480
4th(milliseconds)	24.058	222.157
5th(milliseconds)	23.796	236.752
6th(milliseconds)	23.746	227.602
7th(milliseconds)	24.344	225.330
8th(milliseconds)	24.459	226.794
9th(milliseconds)	24.585	232.023
10th(milliseconds)	24.009	227.108
Shortest Time	**23.590**	**222.157**

Original Code Results

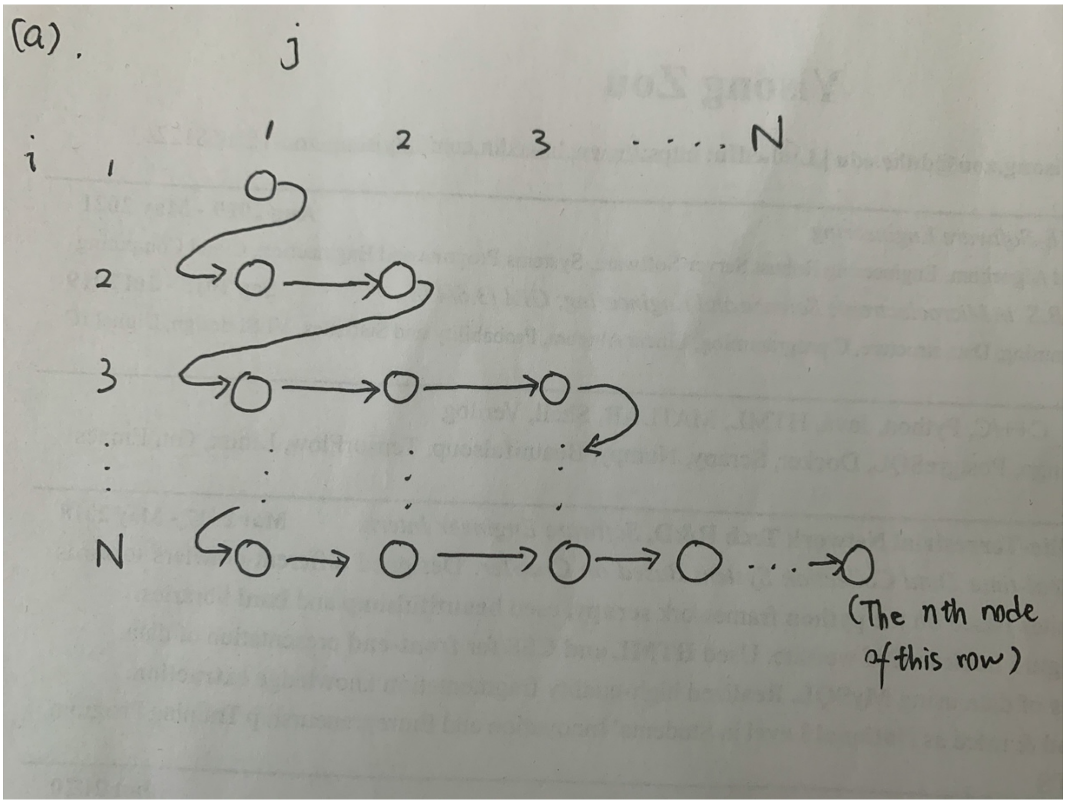
gcc -O2

Ten Runs	Argument(10000000)(seven 0s)	Argument(100000000)(eight 0s)
1st(milliseconds)	27.322	268.833
2nd(milliseconds)	26.391	258.334
3rd(milliseconds)	26.624	258.541
4th(milliseconds)	26.996	268.294
5th(milliseconds)	27.273	259.977
6th(milliseconds)	28.181	266.166
7th(milliseconds)	26.739	264.944
8th(milliseconds)	26.963	266.627
9th(milliseconds)	26.789	264.117
10th(milliseconds)	26.486	267.082
Shortest Time	**26.391**	**258.334**

From the above statistics, my code have beaten the computer in -O2 option.

2. (20 *points*) **Dependence Analysis.**

(a) Draw ITG



(b) List all the dependencies

(b). Dependences

$S_1[i,j] \Rightarrow_T S_2[i+1, j+1]$ (loop carried)

$S_1[i,j] \Rightarrow_T S_3[i,j]$ (loop independent)

$S_1[i-1,j+1] \Rightarrow_A S_1[i,j]$ (loop carried)

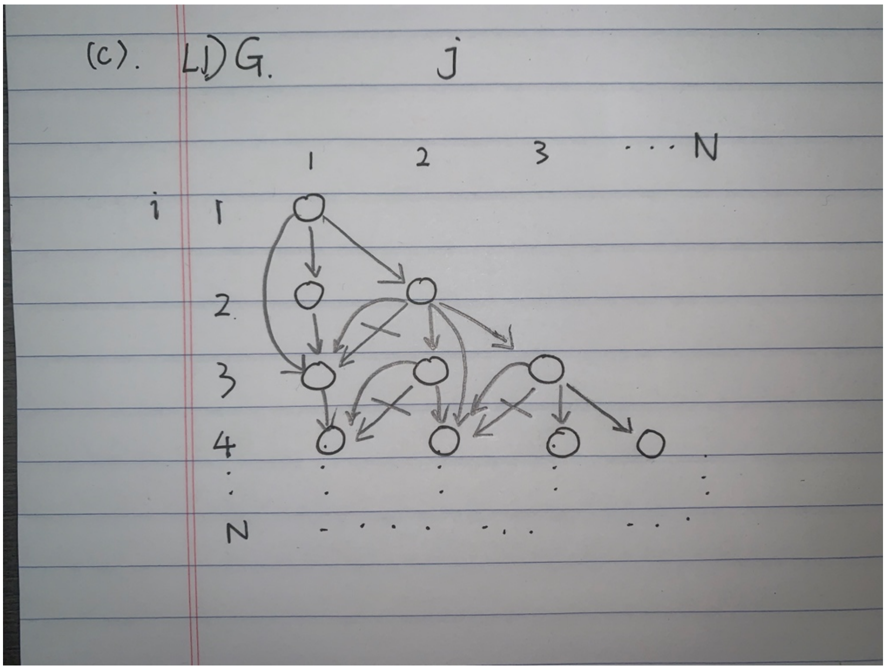
$S_1[i,j] \Rightarrow_A S_2[i,j]$ (loop independent)

$S_3[i-1,j] \Rightarrow_T S_1[i,j]$ (loop carried)

$S_3[i-1,j] \Rightarrow_T S_2[i+1,j]$ (loop carried)

$S_4[i,j] \Rightarrow_T S_4[i+1,j-1]$ (loop carried)

(c) Draw the LDG



3. (20 *points*) Function In-lining and performance

(a).

Without Inlining Function Code:

```
__attribute__((noinline)) int add(int a, int b) { return (a + b); }
```

Without inlining Original code:

Argument(10000000)(seven 0s)(Shortest 28.311)

```
1.  Time=28.357 milliseconds
2.  Time=29.208 milliseconds
3.  Time=28.336 milliseconds
4.  Time=28.42 milliseconds
5.  Time=28.405 milliseconds
6.  Time=28.311 milliseconds
7.  Time=28.441 milliseconds
8.  Time=28.571 milliseconds
9.  Time=28.46 milliseconds
10. Time=28.375 milliseconds
```

Without inlining Original code:

Argument(100000000)(eight 0s)(Shortest 281.478)

```
1.  Time=282.653 milliseconds
2.  Time=283.631 milliseconds
3.  Time=282.651 milliseconds
4.  Time=281.774 milliseconds
5.  Time=282.116 milliseconds
6.  Time=282.202 milliseconds
7.  Time=282.153 milliseconds
8.  Time=284.597 milliseconds
9.  Time=281.881 milliseconds
10. Time=281.478 milliseconds
```

Inlining Function Code:

```
inline __attribute__((always_inline)) int add(int a, int b) { return (a + b)
; }
```

With inlining:

Argument(10000000)(seven 0s)(Shortest 10.169)

```
Time=10.193 milliseconds
Time=10.169 milliseconds
Time=10.281 milliseconds
Time=10.281 milliseconds
Time=10.609 milliseconds
Time=10.291 milliseconds
Time=10.782 milliseconds
Time=10.484 milliseconds
Time=10.268 milliseconds
Time=10.314 milliseconds
```

With inlining:

Argument(100000000)(eight 0s)(Shortest 103.273)

```
Time=104.732 milliseconds
Time=104.525 milliseconds
Time=104.213 milliseconds
Time=105.15 milliseconds
Time=104.727 milliseconds
Time=103.273 milliseconds
Time=104.606 milliseconds
Time=104.956 milliseconds
Time=104.669 milliseconds
Time=104.098 milliseconds
```

(b)

We can see from the following two different assemblies that inline will cause the main function to directly contain the lines of the function while no_inline version will make a call to the function.

Assembly without inlining

```
e12:        e8 19 fd ff ff          callq  b30 <gettimeofday@plt>
e17:        44 89 e0                mov    %r12d,%eax
e1a:        4c 8b 54 24 50          mov    0x50(%rsp),%r10
e1f:        4c 8b 4c 24 30          mov    0x30(%rsp),%r9
e24:        4c 8b 44 24 70          mov    0x70(%rsp),%r8
e29:        48 8d 0c 85 04 00 00    lea    0x4(,%rax,4),%rcx
e30:        00
e31:        31 d2                   xoqr    %edx,%edx
e33:        0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
e38:        41 8b 34 12             mov    (%r10,%rdx,1),%esi
e3c:        41 8b 3c 11             mov    (%r9,%rdx,1),%edi
e40:        e8 eb 04 00 00          callq  1330 <_Z3addii>
```

```
e45:        41 89 04 10               mov    %eax,(%r8,%rdx,1)
e49:        48 83 c2 04               add    $0x4,%rdx
e4d:        48 39 ca                  cmp    %rcx,%rdx
e50:        75 e6                     jne    e38 <main+0x2c8>
e52:        31 f6                     xor    %esi,%esi
e54:        48 89 ef                  mov    %rbp,%rdi
e57:        e8 d4 fc ff ff            callq  b30 <gettimeofday@plt>
```

Assembly with inlining

```
e12:        e8 19 fd ff ff            callq  b30 <gettimeofday@plt>
    e17:        48 8b 7c 24 50            mov    0x50(%rsp),%rdi
    e1c:        48 8b 74 24 70            mov    0x70(%rsp),%rsi
    e21:        4c 8b 44 24 30            mov    0x30(%rsp),%r8
    e26:        48 8d 47 10               lea    0x10(%rdi),%rax
    e2a:        48 8d 4e 10               lea    0x10(%rsi),%rcx
    e2e:        48 39 c6                  cmp    %rax,%rsi
    e31:        0f 93 c2                  setae  %dl
    e34:        48 39 cf                  cmp    %rcx,%rdi
    e37:        0f 93 c0                  setae  %al
    e3a:        09 c2                     or     %eax,%edx
    e3c:        49 8d 40 10               lea    0x10(%r8),%rax
    e40:        48 39 c6                  cmp    %rax,%rsi
    e43:        0f 93 c0                  setae  %al
    e46:        49 39 c8                  cmp    %rcx,%r8
    e49:        0f 93 c1                  setae  %cl
    e4c:        09 c8                     or     %ecx,%eax
    e4e:        84 c2                     test   %al,%dl
    e50:        0f 84 7e 03 00 00         je     11d4 <main+0x664>
    e56:        83 fb 08                  cmp    $0x8,%ebx
    e59:        0f 86 75 03 00 00         jbe    11d4 <main+0x664>
    e5f:        48 89 f9                  mov    %rdi,%rcx
    e62:        48 c1 e9 02               shr    $0x2,%rcx
    e66:        48 f7 d9                  neg    %rcx
    e69:        83 e1 03                  and    $0x3,%ecx
    e6c:        0f 84 5a 03 00 00         je     11cc <main+0x65c>
    e72:        41 8b 00                  mov    (%r8),%eax
    e75:        03 07                     add    (%rdi),%eax
    e77:        83 f9 01                  cmp    $0x1,%ecx
    e7a:        89 06                     mov    %eax,(%rsi)
    e7c:        0f 84 b0 03 00 00         je     1232 <main+0x6c2>
    e82:        41 8b 40 04               mov    0x4(%r8),%eax
    e86:        03 47 04                  add    0x4(%rdi),%eax
```

```
e89:	83 f9 02             	cmp    $0x2,%ecx
e8c:	89 46 04             	mov    %eax,0x4(%rsi)
e8f:	0f 84 dc 03 00 00    	je     1271 <main+0x701>
e95:	41 8b 40 08          	mov    0x8(%r8),%eax
e99:	03 47 08             	add    0x8(%rdi),%eax
e9c:	41 b9 03 00 00 00    	mov    $0x3,%r9d
ea2:	89 46 08             	mov    %eax,0x8(%rsi)
ea5:	41 89 db             	mov    %ebx,%r11d
ea8:	31 c0                	xor    %eax,%eax
eaa:	31 d2                	xor    %edx,%edx
eac:	41 29 cb             	sub    %ecx,%r11d
eaf:	89 c9                	mov    %ecx,%ecx
eb1:	48 c1 e1 02          	shl    $0x2,%rcx
eb5:	45 89 da             	mov    %r11d,%r10d
eb8:	4c 8d 34 0f          	lea    (%rdi,%rcx,1),%r14
ebc:	4d 8d 2c 08          	lea    (%r8,%rcx,1),%r13
ec0:	41 c1 ea 02          	shr    $0x2,%r10d
ec4:	48 01 f1             	add    %rsi,%rcx
ec7:	66 0f 1f 84 00 00 00 	nopw   0x0(%rax,%rax,1)
ece:	00 00 
ed0:	f3 41 0f 6f 44 05 00 	movdqu 0x0(%r13,%rax,1),%xmm0
ed7:	83 c2 01             	add    $0x1,%edx
eda:	66 41 0f fe 04 06    	paddd  (%r14,%rax,1),%xmm0
ee0:	0f 11 04 01          	movups %xmm0,(%rcx,%rax,1)
ee4:	48 83 c0 10          	add    $0x10,%rax
ee8:	41 39 d2             	cmp    %edx,%r10d
eeb:	77 e3                	ja     ed0 <main+0x360>
eed:	44 89 da             	mov    %r11d,%edx
ef0:	83 e2 fc             	and    $0xfffffffc,%edx
ef3:	41 39 d3             	cmp    %edx,%r11d
ef6:	42 8d 04 0a          	lea    (%rdx,%r9,1),%eax
efa:	74 70                	je     f6c <main+0x3fc>
efc:	48 63 d0             	movslq %eax,%rdx
eff:	41 8b 0c 90          	mov    (%r8,%rdx,4),%ecx
f03:	03 0c 97             	add    (%rdi,%rdx,4),%ecx
f06:	89 0c 96             	mov    %ecx,(%rsi,%rdx,4)
f09:	8d 50 01             	lea    0x1(%rax),%edx
f0c:	39 d3                	cmp    %edx,%ebx
f0e:	7e 5c                	jle    f6c <main+0x3fc>
f10:	48 63 d2             	movslq %edx,%rdx
f13:	41 8b 0c 90          	mov    (%r8,%rdx,4),%ecx
f17:	03 0c 97             	add    (%rdi,%rdx,4),%ecx
f1a:	89 0c 96             	mov    %ecx,(%rsi,%rdx,4)
f1d:	8d 50 02             	lea    0x2(%rax),%edx
```

```
    f20:        39 d3                   cmp    %edx,%ebx
    f22:        7e 48                   jle    f6c <main+0x3fc>
    f24:        48 63 d2                movslq %edx,%rdx
    f27:        41 8b 0c 90             mov    (%r8,%rdx,4),%ecx
    f2b:        03 0c 97                add    (%rdi,%rdx,4),%ecx
    f2e:        89 0c 96                mov    %ecx,(%rsi,%rdx,4)
    f31:        8d 50 03                lea    0x3(%rax),%edx
    f34:        39 d3                   cmp    %edx,%ebx
    f36:        7e 34                   jle    f6c <main+0x3fc>
    f38:        48 63 d2                movslq %edx,%rdx
    f3b:        41 8b 0c 90             mov    (%r8,%rdx,4),%ecx
    f3f:        03 0c 97                add    (%rdi,%rdx,4),%ecx
    f42:        89 0c 96                mov    %ecx,(%rsi,%rdx,4)
    f45:        8d 50 04                lea    0x4(%rax),%edx
    f48:        39 d3                   cmp    %edx,%ebx
    f4a:        7e 20                   jle    f6c <main+0x3fc>
    f4c:        48 63 d2                movslq %edx,%rdx
    f4f:        83 c0 05                add    $0x5,%eax
    f52:        41 8b 0c 90             mov    (%r8,%rdx,4),%ecx
    f56:        03 0c 97                add    (%rdi,%rdx,4),%ecx
    f59:        39 c3                   cmp    %eax,%ebx
    f5b:        89 0c 96                mov    %ecx,(%rsi,%rdx,4)
    f5e:        7e 0c                   jle    f6c <main+0x3fc>
    f60:        48 98                   cltq
    f62:        41 8b 14 80             mov    (%r8,%rax,4),%edx
    f66:        03 14 87                add    (%rdi,%rax,4),%edx
    f69:        89 14 86                mov    %edx,(%rsi,%rax,4)
    f6c:        31 f6                   xor    %esi,%esi
    f6e:        48 89 ef                mov    %rbp,%rdi
    f71:        e8 ba fb ff ff          callq  b30 <gettimeofday@plt>
```

(c)

The performance matches my expectation. There is a great improvement on the speed of the function because it avoids the function call & return instructions and will allow the compiler to better optimize the code.

(d)

The performance of the original code:

Argument(10000000)(seven 0s)(Shortest 11.101)

```
Time=11.459 milliseconds
Time=11.826 milliseconds
```

```
3.  Time=11.486 milliseconds
4.  Time=12.155 milliseconds
5.  Time=11.531 milliseconds
6.  Time=11.983 milliseconds
7.  Time=11.517 milliseconds
8.  Time=12.101 milliseconds
9.  Time=11.541 milliseconds
10. Time=12.048 milliseconds
```

Argument(100000000)(eight 0s)(Shortest 114.973)

```
1.  Time=117.608 milliseconds
2.  Time=115.398 milliseconds
3.  Time=115.349 milliseconds
4.  Time=114.973 milliseconds
5.  Time=116.951 milliseconds
6.  Time=117.818 milliseconds
7.  Time=117.138 milliseconds
8.  Time=117.996 milliseconds
9.  Time=115.649 milliseconds
10. Time=116.269 milliseconds
```

The original code's performance is nearly the same as the in-lining code, I think it is because -O3 compiler has already done the in-lining by default, so there is not such a large difference.

4. (15 *points*) Loop Transformations

Original Code:

```
...
int a[N][4];
int rand_number = rand();
for (i=0; i<4; i++) {
threshold = 2.0 * rand_number;
for (j=0; j<N; j++) {
if (threshold < 4) {
sum = sum + a[j][i];
} else {
sum = sum + a[j][i] + 1;
}
}
} ...
```

After Loop Invariant Hoisting

```
...
int a[N][4];
int rand_number = rand();
threshold = 2.0 * rand_number;
for (i=0; i<4; i++) {
for (j=0; j<N; j++) {
if (threshold < 4) {
sum = sum + a[j][i];
} else {
sum = sum + a[j][i] + 1;
}
}
} ...
```

After Loop Unroll and Jam

```
...
int a[N][4];
int rand_number = rand();
threshold = 2.0 * rand_number;
for (i=0; i<4; i+=4) {
for (j=0; j<N; j++) {
if (threshold < 4) {
sum = sum + a[j][i];
```

```
sum = sum + a[j][i + 1];
sum = sum + a[j][i + 2];
sum = sum + a[j][i + 3];
} else {
sum = sum + a[j][i] + 1;
sum = sum + a[j][i + 1] + 1;
sum = sum + a[j][i + 2] + 1;
sum = sum + a[j][i + 3] + 1;
}
}
} ...
```

After Loop Unswitching

```
...
int a[N][4];
int rand_number = rand();
threshold = 2.0 * rand_number;
if (threshold < 4) {
for (i=0; i<4; i+=4) {
for (j=0; j<N; j++) {
sum = sum + a[j][i];
sum = sum + a[j][i + 1];
sum = sum + a[j][i + 2];
sum = sum + a[j][i + 3];
}
}
} else {
for (i=0; i<4; i+=4) {
for (j=0; j<N; j++) {
sum = sum + a[j][i] + 1;
sum = sum + a[j][i + 1] + 1;
sum = sum + a[j][i + 2] + 1;
sum = sum + a[j][i + 3] + 1;
}
}
} ...
```

After Loop Interchange

```
...
int a[N][4];
int rand_number = rand();
threshold = 2.0 * rand_number;
if (threshold < 4) {
```

```
for (j=0; j<N; j++) {
for (i=0; i<4; i+=4) {
sum = sum + a[j][i];
sum = sum + a[j][i + 1];
sum = sum + a[j][i + 2];
sum = sum + a[j][i + 3];
}
}
} else {
for (j=0; j<N; j++) {
for (i=0; i<4; i+=4) {
sum = sum + a[j][i] + 1;
sum = sum + a[j][i + 1] + 1;
sum = sum + a[j][i + 2] + 1;
sum = sum + a[j][i + 3] + 1;
}
}
} ...
```

5. (15 *points*) **Loop Transformations**

(a)

Unsafe because originally there is **loop carried output dependence from S1[i] to S3[i – 1]** and **loop carried anti dependence from S2[i] to S3[i – 1] ,** however after the loop fusion, the dependences has become **loop carried output dependence from S3[i – 1] to S1[i]** and **loop carried true dependence from S3[i-1] to S2[i] .**

(b)

Unsafe because in the example of loop interchange, the **outermost loop carries a anti dependence from [i-1 , j+1] to [i , j].** So here i< i' and j > j', so it is unsafe.

(c)Safe.