

# ICS Homework 5

April 4, 2020

## 1 Organization

Usually we use the following representation of polynomials in math:

$$f(x) = a_n * x^n + a_{n-1} * x^{n-1} + \dots + a_1 * x + a_0$$

But this form is not suitable for computation in computer. Instead, we use the following representation:

$$f(x) = a_0 + x * (a_1 + x * (a_2 + \dots + x * (a_{n-1} + x a_n)))$$

### 1.1

Please explain why the latter representation is faster. (HINT: Consider the number of computation primitive used)

Calculate  $x^n$  is expensive. Even we calculate it iteratively, there is still one more multiplication to get  $a_i * x^i$  in each iteration.

### 1.2

We have the following code to evaluate the polynomial on a given x, but it's very slow. Please optimize it using machine-independent optimization.

```
1 struct coefficient {
2     int a;
3     struct coefficient *next;
4 };
5
6
7 int get_n(struct coefficient *alist) {
8     int n = 0;
9     while (alist) {
10         n++;
11         alist = alist->next;
12     }
13     return n;
14 }
```

```

1
2  int get_ai(struct coefficient *alist, int i) {
3      int current = get_n(alist);
4      while (current != i) {
5          alist = alist->next;
6          current--;
7      }
8      return alist->a;
9  }
10
11 int cal(struct coefficient *alist, int x) {
12     int result = get_ai(alist, n);
13     int n = 0;
14     for (int i = get_n(alist) - 1; i >= 0; i --)
15         result = result * x +
16             get_ai(alist, i);
17     return result;
18 }

```

ANS:

```

1  int calculate(struct coefficient *alist, int x) {
2      int result = alist->a;
3      list = alist->next;
4      while (alist) {
5          result = result * x + alist->a;
6          alist = alist->next;
7      }
8      return result;
9  }

```

### 1.3

Here is the array version of the function. We place  $a_i$  in `a[i]` now.

```

1  int cal(int *a, int n, int x) {
2      int result = a[n];
3      for (int i = n - 1; i >= 0; i--)
4          result = result * x + a[i];
5  }

```

And the asm code looks like this:

```

1 loop:
2     testl %ebx, %ebx
3     jge done
4     imull %r13d, %edx
5     movl (%r14, %ebx, 4), %eax
6     addl %eax, %edx
7     subl $1, %ebx
8     jmp loop
9 done:

```

Can you use multiple accumulators to optimize this program? How or Why?  
 No. Each iteration must use the result generated in the last iteration.

## 2 System Software

Linux provides an explicit mechanism for blocking signals. Read the C program and answer questions below.

```

1  #include <stdio.h>
2  #include <signal.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5
6  void sig_han(int sig)
7  {
8      printf("signal_handled\n");
9  }
10
11 int main()
12 {
13     sigset_t set;
14     int i;
15     signal(SIGINT, sig_han);
16     sigemptyset(&set);
17     sigaddset(&set, SIGINT);
18     sigprocmask(SIG_BLOCK, &set, NULL);
19
20     for (i = 0; i < 3; i++) {
21         printf("send_signal\n");
22         kill(getpid(), SIGINT);
23     }
24
25     sigprocmask(SIG_UNBLOCK, &set, NULL);
26     return 0;

```

27 | }

What is the output of this program? Please explain your answer.

ANS

send signal

send signal

send signal

signal handled

The pending signal is not delivered to the process when the signal is blocked. So the control flow will not change to the signal handler until the signal is unblocked. The signal will not be delivered multiple times because pending signals are not queued.