如何使程序运行得更快？

①编译器友好的代码　②针对特定处理器优化.

*使用gcc作为编译器.

编译器的优化限制.

①不会改变程序行为(除非程序使用了非标准命令).

②可能会被指令与代码风格混淆的代码.

③只有在程序过程(procedure)之中进行优化。

④只对静态信息作优化。

若有置疑，则不进行优化.

普遍性的优化技巧

代码移动(Code Motion)

通过将重复的代码块进行

移动以减少计算量

Code Motion

- Reduce frequency with which computation performed
  - If it will always produce same result
  - Especially moving code out of loop

```
void set_row(double *a, double *b,
   long i, long n)
{
    long j;
    for (j = 0; j < n; j++)
        a[n*i+j] = b[j];
}
```

→

```
    long j;
    int ni = n*i;
    for (j = 0; j < n; j++)
        a[ni+j] = b[j];
```

减少计算量(Reduction in Strength).

将消耗资源多的操作改为消耗少的

使用移位命令是乘/除.

设计出一系列乘法

```
for (i = 0; i < n; i++) {
  int ni = n*i;
  for (j = 0; j < n; j++)
    a[ni + j] = b[j];
}
```

→

```
int ni = 0;
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++)
    a[ni + j] = b[j];
  ni += n;
}
```

共享子表述式(Share Common Subexpressions)

重用表述式的值.

```
/* Sum neighbors of i,j */
up =    val[(i-1)*n + j  ];
down =  val[(i+1)*n + j  ];
left =  val[i*n     + j-1];
right = val[i*n     + j+1];
sum = up + down + left + right;
```

3 multiplications: i*n, (i–1)*n, (i+1)*n

```
leaq   1(%rsi), %rax  # i+1
leaq   -1(%rsi), %r8  # i-1
imulq  %rcx, %rsi     # i*n
imulq  %rcx, %rax     # (i+1)*n
imulq  %rcx, %r8      # (i-1)*n
addq   %rdx, %rsi     # i*n+j
addq   %rdx, %rax     # (i+1)*n+j
addq   %rdx, %r8      # (i-1)*n+j
```

```
long inj = i*n + j;
up =    val[inj - n];
down =  val[inj + n];
left =  val[inj - 1];
right = val[inj + 1];
sum = up + down + left + right;
```

1 multiplication: i*n

```
imulq    %rcx, %rsi  # i*n
addq     %rdx, %rsi  # i*n+j
movq     %rsi, %rax  # i*n+j
subq     %rcx, %rax  # i*n+j-n
leaq     (%rsi,%rcx), %rcx # i*n+j+n
```

造成优化的因素：

1. 过程调用.

例：

```
void lower(char *s)
{
  size_t i;
  for (i = 0; i < strlen(s); i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

每次循环都会调用 strlen，

strlen：O(n).

因此导致程序时间为：O(n²)

原因①过程调用可能带来副作用 ②编译器不知道函数的具体实现

2. 内存别名

定义：程序中不同的部分指向了内存中的同一区域。

编译器假设内存区域可能发生变量，因此会反复读/写内存。

指令级别的并行（乱序执行）：

可以通过改变计算的顺序，来利用流水线的并行能力.

处理器优化：

- AVX2 指令集.
- 分支预测