

# Program Performance



## Example 1

```
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];  
}
```

# Code Motion

```
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];
}
```



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

# Compiler-Generated Code Motion (-O1)

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

```
set_row:
    testq   %rcx, %rcx           # Test n
    jle    .L1                  # If 0, goto done
    imulq  %rcx, %rdx           # ni = n*i
    leaq   (%rdi,%rdx,8), %rdx  # rowp = A + ni*8
    movl   $0, %eax            # j = 0
.L3:
    movsd  (%rsi,%rax,8), %xmm0 # t = b[j]
    movsd  %xmm0, (%rdx,%rax,8) # M[A+ni*8 + j*8] = t
    addq   $1, %rax             # j++
    cmpq   %rcx, %rax          # j:n
    jne    .L3                 # if !=, goto loop
.L1:
    rep ; ret                   # done:
```

## Example 2

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

## Strength Reduction

```
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;  
}
```

## Example 3

```
/* 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;
```

```
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
```

## Share Subexpressions

```
/* 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
```