第三章 RISC-V汇编及其指令系统

- RISC-V概述
- RISC-V汇编语言
- RISC-V指令表示
- **实例分析**

第三章 RISC-V汇编及其指令系统

- 实例分析
  - 数组与指针
  - RISC-V冒泡排序
  - 编译器优化

两种数组清零方法的对比

用下标访问数组元素	用指针访问数组元素

clear1(long long int array[], int size){
  int i;
  for(i=0; i<size; i+=1)
  array[i]=0;
} //x10中存array首地址
	clear2(long long int *array, int size){
	  long long int *p;
	  for(p=&array[0]; p< &array[size];p=p+1)
	  ***p**=0;
	} //x10中存array首地址

addi t0,zero,0 #i=0
loop1: **slli t1,t0,3** #t1=i*8
add t2,x10,t1 #t2=array[i]地址
sd x0,0(t2) #array[i]=0
addi t0,t0,1 #i=i+1元素计数加1
blt t0,a1,loop1 #i<size, 循环
	addi t0, a0, 0 #t0=array[0]地址
	slli t1, a1,3 #t1=size*8
	add t2,a0,t1 #t2=最后一个元素的地址
	loop2:sd zero,0(t0) #memory[p]=0
	addi t0,t0,8 #p=p+8
	bltu t0,t2,loop2 #未到最后则循环

数组清零代码的分析

- **乘8——>左移3次（编译器优化）**
- 指针版本的循环体指令数少于数组版本
  - 数组版本i自增1后，重新计算下一个数组元素的地址（+8）
  - 指针版本的循环体内少了i*8和求新元素地址
  - 编译器优化——循环变量消除
- **高级语言编程推荐使用下标方式访问数组元素**
- **编译器优化生成的代码与手动使用指针一样有效**

第二章 RISC-V汇编及其指令系统

- **实例分析**
  - 数组与指针
  - **RISC-V冒泡排序**
  - 编译器优化

C冒泡排序示例

- **C冒泡排序函数的汇编指令示例**

- **swap过程**（**叶过程，**即不调用其他过程的过程）

```
void swap(long long int v[], long long int k)
{
    long long int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

- v保存在x10，k保存在x11，temp保存在x5

Swap汇编代码

已知：**v保存在x10，**
k保存在x11，
temp保存在x5

- **swap:**

```
slli x6, x11, 3     # x6 = k * 8
add x6, x10, x6  # x6 = v + (k * 8)
ld x5, 0(x6)       # x5= v[k] =x6
ld x7, 8(x6)       # x7 = v[k + 1]
sd x7, 0(x6)       # x6= x7
sd x5, 8(x6)       # x7 = x5
jalr x0, 0(x1)     # 返回调用函数
```

```
void swap(long long int v[], long long int k)
{
    long long int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

C排序的sort过程（黑书P95-P99）

- 排序（升序）过程（调用了swap）

```
void sort (long long int v[], int n) {
        int i, j;
        for (i = 0; i < n; i += 1) {
            for (j = i – 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
            swap(v, j); }
        }
}
```

- v保存在x10，n保存在x11，i保存在x19，j保存在x20

备注：冒泡思想是每趟排序轻的向上飘

保存寄存器

```
sort:
    addi sp, sp, -40        # 在栈中留出5个寄存器（双字）的空间
    sd x1, 32(sp)           # 保存x1的值（入栈）
    sd x22, 24(sp)          # 保存x22的值（入栈）
    sd x21, 16(sp)          # 保存x21的值（入栈）
    sd x20, 8(sp)           # 保存x20的值（入栈）
    sd x19, 0(sp)           # 保存x19的值（入栈）
```

v->x10， n->x11， i->x19， j->x20

过程体

移动参数

```
    mv x21, x10             # 复制x10中的值到x21
    mv x22, x11             # 复制x11中的值到x22
```

外循环

```
    li x19, 0               #  i = 0
for1tst:
    bge x19, x22, exit1     # 如果x19 ≥ x22（i ≥ n），跳转到exit1
```

内循环

```
    addi x20, x19, -1       #  j = i-1
for2tst:
    blt x20, x0, exit2      # 如果x20 < 0 (j < 0)，跳转到exit2
    slli x5, x20, 3         #  x5 = j * 8
    add x5, x21, x5         #  x5 = v + (j * 8)
    ld x6, 0(x5)            #  x6 = v[j]
    ld x7, 8(x5)            #  x7 = v[j + 1]
    ble x6, x7, exit2       #  如果x6 ≤ x7，跳转到exit2
```

参数传递和调用

```
    mv x10, x21             #  swap的第一个参数是v
    mv x11, x20             #  swap的第二个参数是j
    jal x1, swap            #  调用swap
```

内循环

```
    addi x20, x20, -1       #  j -= 1
    j for2tst               #  跳转到内层循环for2tst
```

外循环

```
exit2:
    addi x19, x19, 1        # i += 1
    j for1tst               # 跳转到外层循环的判断语句
```

恢复寄存器

```
exit1:
    ld x19, 0(sp)           # 恢复x19（出栈）
    ld x20, 8(sp)           # 恢复x20 （出栈）
    ld x21, 16(sp)          # 恢复x21 （出栈）
    ld x22, 24(sp)          # 恢复x22 （出栈）
    ld x1, 32(sp)           # 恢复x1（出栈）
    addi sp, sp, 40         # 恢复栈指针
```

过程返回

```
    jalr x0, 0(x1)          # 返回调用线程
```

保存寄存器

```
sort:
    addi sp, sp, -40        # 在栈中留出5个寄存器（双字）的空间
    sd x1, 32(sp)           # 保存x1的值（入栈）
    sd x22, 24(sp)          # 保存x22的值（入栈）
    sd x21, 16(sp)          # 保存x21的值（入栈）
    sd x20, 8(sp)           # 保存x20的值（入栈）
    sd x19, 0(sp)           # 保存x19的值（入栈）
```

过程体

移动参数

```
    mv x21, x10             # 复制x10中的值到x21
    mv x22, x11             # 复制x11中的值到x22
```

外循环

```
    li x19, 0               #  i = 0
for1tst:
    bge x19, x22, exit1     # 如果x19 ≥ x22（i ≥ n），跳转到exit1
```

内循环

```
    addi x20, x19, -1       #  j = i-1
for2tst:
    blt x20, x0, exit2      # 如果x20 < 0 (j < 0)，跳转到exit2
    slli x5, x20, 3         #  x5 = j * 8
    add x5, x21, x5         #  x5 = v + (j * 8)
    ld x6, 0(x5)            #  x6 = v[j]
    ld x7, 8(x5)            #  x7 = v[j + 1]
    ble x6, x7, exit2       #  如果x6 ≤ x7，跳转到exit2
```

参数传递和调用

```
    mv x10, x21             #  swap的第一个参数是v
    mv x11, x20             #  swap的第二个参数是j
    jal x1, swap            #  调用swap
```

内循环

```
    addi x20, x20, -1       #  j -= 1
    j for2tst               #  跳转到内层循环for2tst
```

v->x10， n->x11， i->x19， j->x20

外循环

```
exit2:
    addi x19, x19, 1        # i += 1
    j for1tst               # 跳转到外层循环的判断语句
```

恢复寄存器

```
exit1:
    ld x19, 0(sp)           # 恢复x19（出栈）
    ld x20, 8(sp)           # 恢复x20 （出栈）
    ld x21, 16(sp)          # 恢复x21 （出栈）
    ld x22, 24(sp)          # 恢复x22 （出栈）
    ld x1, 32(sp)           # 恢复x1（出栈）
    addi sp, sp, 40         # 恢复栈指针
```

过程返回

```
    jalr x0, 0(x1)          # 返回调用线程
```

```
void sort (long long int v[], int n)
{
    int i, j;
    for (i = 0; i < n; i += 1)
    {
        for (j = i – 1; j >= 0 &&
            v[j] > v[j + 1]; j -= 1)
        {
            swap(v, j);
        }
    }
}
```

v->x10， n->x11， i->x19， j->x20

过程体

```
移动参数    mv x21, x10          # 复制x10中的值到x21
            mv x22, x11          # 复制x11中的值到x22
外循环      li x19, 0            # i = 0
for1tst:
            bge x19, x22, exit1  # 如果x19 ≥ x22（i ≥ n），跳转到exit1
            addi x20, x19, -1    # j = i-1
for2tst:
内循环      blt x20, x0, exit2   # 如果x20 < 0 (j < 0)，跳转到exit2
            slli x5, x20, 3      # x5 = j * 8
            add x5, x21, x5      # x5 = v + (j * 8)
            ld x6, 0(x5)         # x6 = v[j]
            ld x7, 8(x5)         # x7 = v[j + 1]
            ble x6, x7, exit2    # 如果x6 ≤ x7，跳转到exit2
参数传递    mv x10, x21          # swap的第一个参数是v
和调用      mv x11, x20          # swap的第二个参数是j
            jal x1, swap         # 调用swap
内循环      addi x20, x20, -1    # j -= 1
            j for2tst            # 跳转到内层循环for2tst
外循环  exit2:
            addi x19, x19, 1     # i += 1
            j for1tst            # 跳转到外层循环的判断语句
```

```
void sort (long long int v[], int n)
{
    int i, j;
    for (i = 0; i < n; i += 1)
    {
        for (j = i – 1; j >= 0 &&
            v[j] > v[j + 1]; j -= 1)
        {
            swap(v, j);
        }
    }
}
```

v->x10， n->x11， i->x19， j->x20

过程体

```
移动参数    mv x21, x10          # 复制x10中的值到x21
            mv x22, x11          # 复制x11中的值到x22
外循环      li x19, 0            # i = 0
for1tst:
            bge x19, x22, exit1  # 如果x19 ≥ x22（i ≥ n），跳转到exit1
            addi x20, x19, -1    # j = i-1
for2tst:
内循环      blt x20, x0, exit2   # 如果x20 < 0 (j < 0)，跳转到exit2
            slli x5, x20, 3      # x5 = j * 8
            add x5, x21, x5      # x5 = v + (j * 8)
            ld x6, 0(x5)         # x6 = v[j]
            ld x7, 8(x5)         # x7 = v[j + 1]
            ble x6, x7, exit2    # 如果x6 ≤ x7，跳转到exit2
参数传递    mv x10, x21          # swap的第一个参数是v
和调用      mv x11, x20          # swap的第二个参数是j
            jal x1, swap         # 调用swap
内循环      addi x20, x20, -1    # j -= 1
            j for2tst            # 跳转到内层循环for2tst
exit2:
外循环      addi x19, x19, 1     # i += 1
            j for1tst            # 跳转到外层循环的判断语句
```

```
void sort (long long int v[], int n)
{
    int i, j;
    for (i = 0; i < n; i += 1)
    {
        for (j = i – 1; j >= 0 &&
            v[j] > v[j + 1]; j -= 1)
        {
            swap(v, j);
        }
    }
}
```

v->x10， n->x11， i->x19， j->x20

过程体

移动参数	mv x21, x10	# 复制x10中的值到x21
	mv x22, x11	# 复制x11中的值到x22
外循环	li x19, 0	# i = 0
	for1tst:	
	bge x19, x22, exit1	# 如果x19 > x22（i > n），跳转到exit1
内循环	addi x20, x19, -1	# j = i-1
	for2tst:	
	blt x20, x0, exit2	# 如果x20 < 0 (j < 0)，跳转到exit2
	slli x5, x20, 3	# x5 = j * 8
	add x5, x21, x5	# x5 = v + (j * 8)
	ld x6, 0(x5)	# x6 = v[j]
	ld x7, 8(x5)	# x7 = v[j + 1]
	ble x6, x7, exit2	# 如果x6 ≤ x7，跳转到exit2
参数传递和调用	mv x10, x21	# swap的第一个参数是v
	mv x11, x20	# swap的第二个参数是j
	jal x1, swap	# 调用swap
内循环	addi x20, x20, -1	# j -= 1
	j for2tst	# 跳转到内层循环for2tst
外循环	exit2:	
	addi x19, x19, 1	# i += 1
	j for1tst	# 跳转到外层循环的判断语句

```
void sort (long long int v[], int n)
{
    int i, j;
    for (i = 0; i < n; i += 1)
    {
        for (j = i – 1; j >= 0 &&
            v[j] > v[j + 1]; j -= 1)
        {
            swap(v, j);
        }
    }
}
```

v->x10， n->x11， i->x19， j->x20

过程体

```
移动参数
    mv x21, x10          # 复制x10中的值到x21
    mv x22, x11          # 复制x11中的值到x22
外循环
    li x19, 0            # i = 0
for1tst:
    bge x19, x22, exit1  # 如果x19 ≥ x22（i ≥ n），跳转到exit1
    addi x20, x19, -1    # j = i-1
for2tst:
    blt x20, x0, exit2   # 如果x20 < 0 (j < 0)，跳转到exit2
内循环
    slli x5, x20, 3      # x5 = j * 8
    add x5, x21, x5      # x5 = v + (j * 8)
    ld x6, 0(x5)         # x6 = v[j]
    ld x7, 8(x5)         # x7 = v[j + 1]
    ble x6, x7, exit2    # 如果x6 ≤ x7，跳转到exit2
参数传递
和调用
    mv x10, x21          # swap的第一个参数是v
    mv x11, x20          # swap的第二个参数是j
    jal x1, swap         # 调用swap
内循环
    addi x20, x20, -1    # j -= 1
    j for2tst            # 跳转到内层循环for2tst
外循环
exit2:
    addi x19, x19, 1     # i += 1
    j for1tst            # 跳转到外层循环的判断语句
```

保存寄存器

```
sort:
    addi sp, sp, -40        # 在栈中留出5个寄存器（双字）的空间
    sd x1, 32(sp)           # 保存x1的值（入栈）
    sd x22, 24(sp)          # 保存x22的值（入栈）
    sd x21, 16(sp)          # 保存x21的值（入栈）
    sd x20, 8(sp)           # 保存x20的值（入栈）
    sd x19, 0(sp)           # 保存x19的值（入栈）
```

过程体

v->x10， n->x11， i->x19， j->x20

移动参数

```
    mv x21, x10             # 复制x10中的值到x21
    mv x22, x11             # 复制x11中的值到x22
```

外循环

```
    li x19, 0               #  i = 0
for1tst:
    bge x19, x22, exit1     # 如果x19≥x22（i ≥ n），跳转到exit1
```

内循环

```
    addi x20, x19, -1       #  j = i-1
for2tst:
    blt x20, x0, exit2      # 如果x20 < 0 (j < 0)，跳转到exit2
    slli x5, x20, 3         #  x5 = j * 8
    add x5, x21, x5         #  x5 = v + (j * 8)
    ld x6, 0(x5)            #  x6 = v[j]
    ld x7, 8(x5)            #  x7 = v[j + 1]
    ble x6, x7, exit2       #  如果x6≤x7，跳转到exit2
```

参数传递和调用

```
    mv x10, x21             #  swap的第一个参数是v
    mv x11, x20             #  swap的第二个参数是j
    jal x1, swap            #  调用swap
```

内循环

```
    addi x20, x20, -1       #  j -= 1
    j for2tst               #  跳转到内层循环for2tst
```

外循环

```
exit2:
    addi x19, x19, 1        # i += 1
    j for1tst               # 跳转到外层循环的判断语句
```

恢复寄存器

```
exit1:
    ld x19, 0(sp)           # 恢复x19（出栈）
    ld x20, 8(sp)           # 恢复x20 （出栈）
    ld x21, 16(sp)          # 恢复x21 （出栈）
    ld x22, 24(sp)          # 恢复x22 （出栈）
    ld x1, 32(sp)           # 恢复x1（出栈）
    addi sp, sp, 40         # 恢复栈指针
```

过程返回

```
    jalr x0, 0(x1)          # 返回调用线程
```

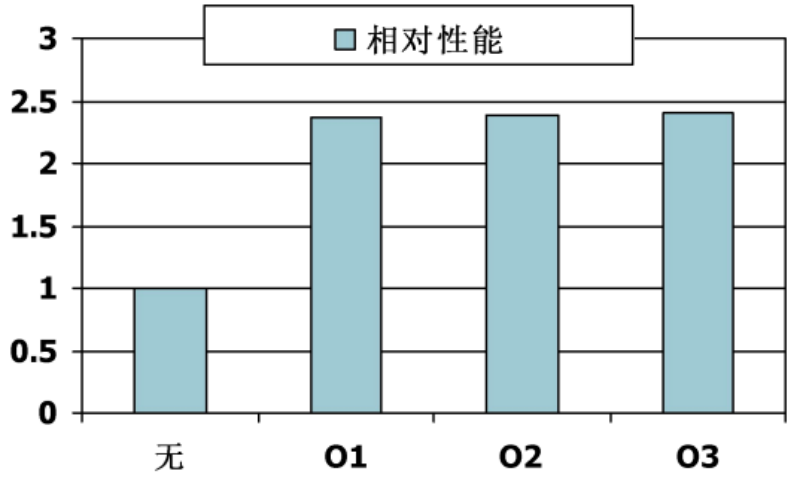
寄存器	助记符	注解
x0	**zero**	**固定值为0**
x1	ra	返回地址(Return Address)
x2	sp	栈指针(Stack Pointer)
x3	gp	全局指针(Global Pointer)
x4	tp	线程指针(Thread Pointer)
x5-x7	t0-t2	临时寄存器
x8	s0/fp	save寄存器/帧指针(Frame Pointer)
x9	s1	save寄存器
x10-x11	a0-a1	函数参数 / 函数返回值
x12-x17	a2-a7	函数参数
x18-x27	s2-s11	save寄存器
x28-x31	t3-6	临时寄存器

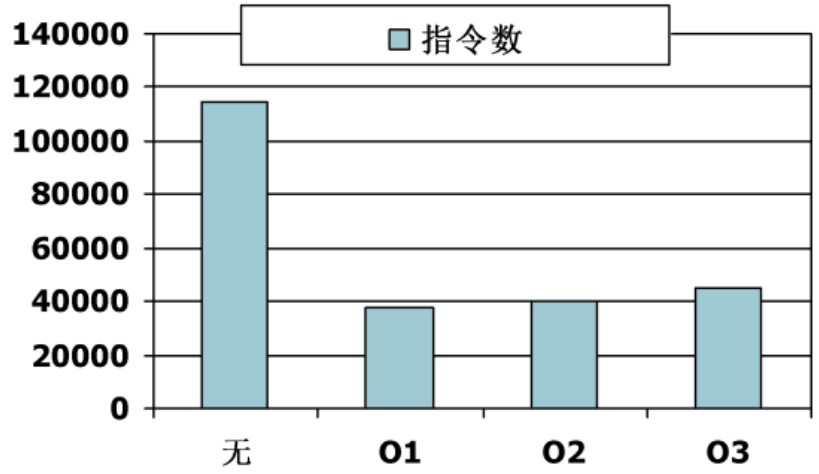
第二章 RISC-V汇编及其指令系统

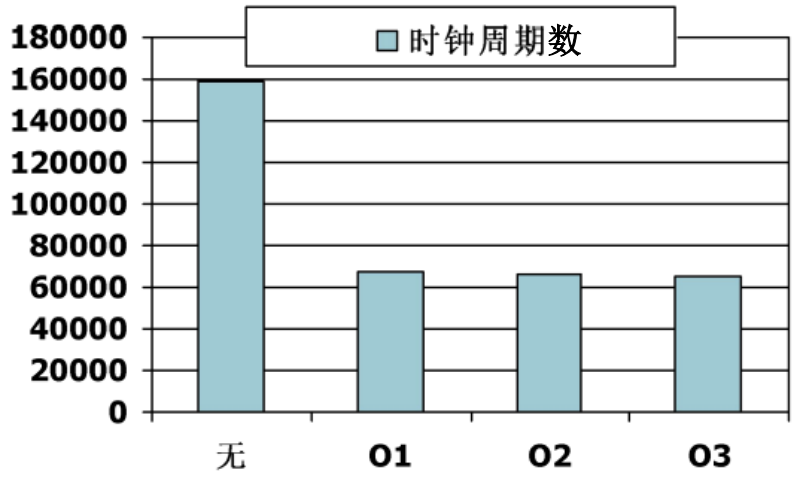
- **实例分析**
  - 数组与指针
  - RISC-V冒泡排序
  - **编译器优化**

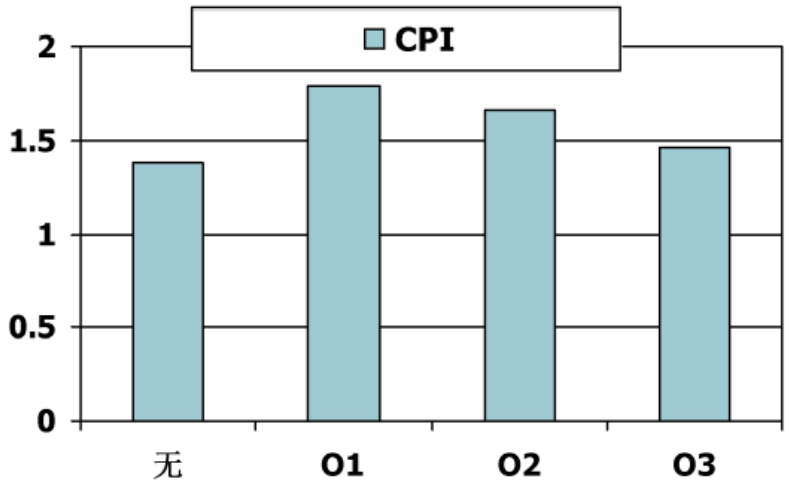
编译器优化的影响

用gcc for Pentium 4在Linux下编译

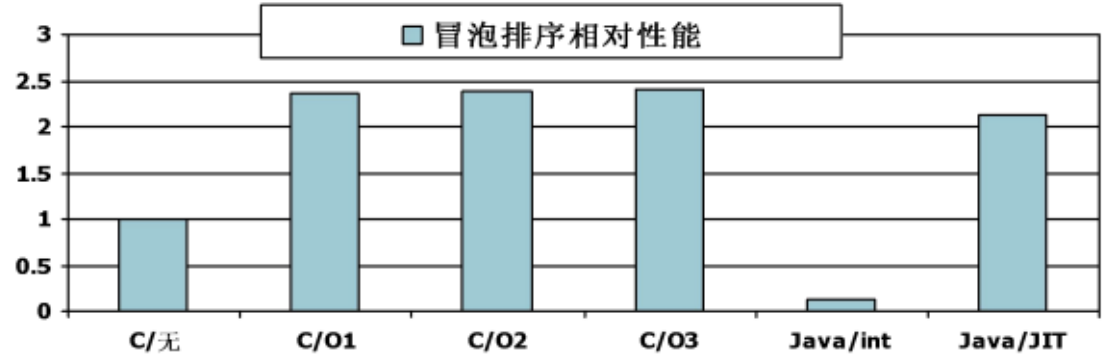


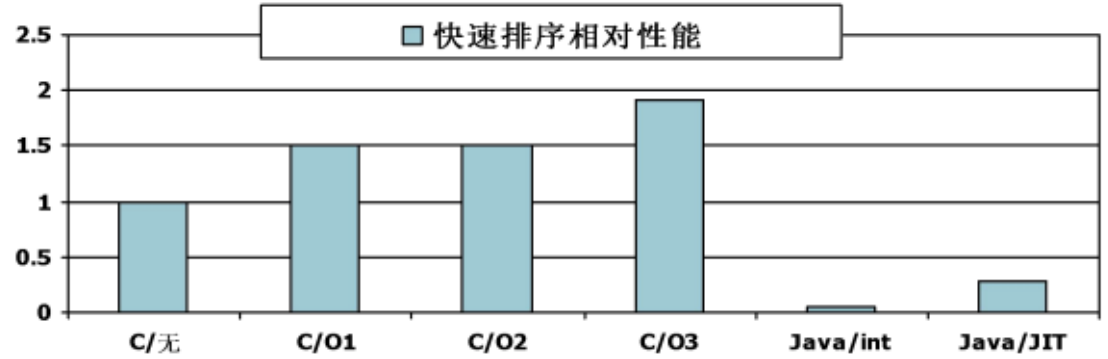


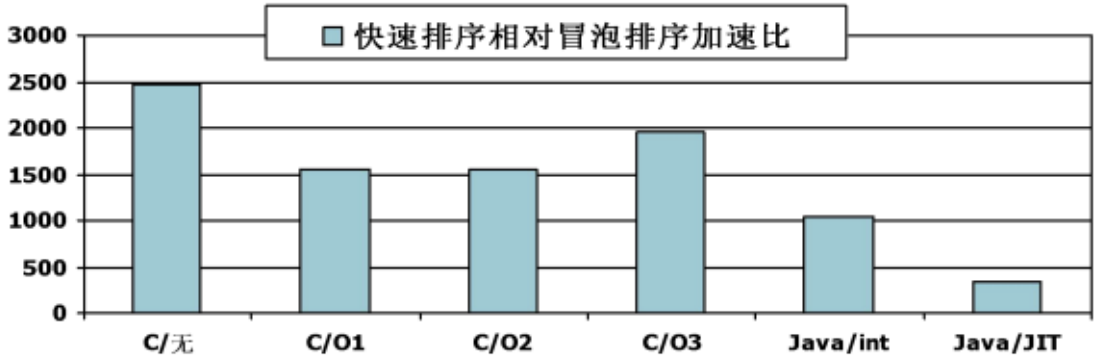




编程语言和算法的影响


冒泡排序相对性能
3
2.5
2
1.5
1
0.5
0
C/无
C/O1
C/O2
C/O3
Java/int
Java/JIT


快速排序相对性能
2.5
2
1.5
1
0.5
0
C/无
C/O1
C/O2
C/O3
Java/int
Java/JIT


快速排序相对冒泡排序加速比
3000
2500
2000
1500
1000
500
0
C/无
C/O1
C/O2
C/O3
Java/int
Java/JIT

总结

- 单看指令数或CPI都不是好的性能指标
- 编译器优化对算法敏感
- Java即时编译器生成的代码明显快于用JVM解释的代码
  - 有些情况下可以和优化过的C代码相媲美

备注：编译器优化对算法性能非常敏感。Java的JIT编译器通过动态优化技术，可以在运行时根据实际情况调整优化策略，从而提高程序的执行效率。而C语言的编译器优化则在编译时完成，虽然也提供了多种优化选项，但在灵活性上不如JIT编译器。因此，在某些特定场景下，Java的JIT编译器优化可以使算法的性能优于C语言的静态编译器优化。

- 没有什么可以弥补拙劣的算法！