Mohammad Iqbal

CSC 210

12/02/2024

**Report: Assembly Code Optimization for sort Function**

**1. Overview of Changes**

The optimization process for the sort function focused on reducing stack memory usage by substituting stack-based local variables with register-based variables. This results in fewer memory accesses, making the code more efficient.

**2. Key Changes and Explanations**

**2.1 Variable Storage**

* **Original Code:**
  + Local variables like -24(%rbp), -28(%rbp), -8(%rbp), and -4(%rbp) were stored on the stack. Every read or write required accessing memory via the base pointer register (%rbp).
* **Optimized Code:**
  + The variables are now held in registers:
    - %r12 holds the pointer to the array (%rdi in original code).
    - %r13d holds the size of the array (%esi in original code).
    - %r8d replaces -8(%rbp) for the outer loop counter.
    - %r9d replaces -4(%rbp) for the inner loop counter.

**Benefit**: Registers offer faster access compared to stack memory.

**2.2 Elimination of Stack Access**

* **Original Code:**
  + Each iteration of the loop involves frequent stack access:

movl -8(%rbp), %eax

subl $1, %eax

movl %eax, -4(%rbp)

* + The values are written back to memory (-4(%rbp)) after every computation.
* **Optimized Code:**
  + Loop counters are directly manipulated in registers:

movl %r8d, %eax

subl $1, %eax

movl %eax, %r9d

* + No memory write-back is needed, as everything stays in the registers.

**Benefit**: Reduces latency caused by memory accesses.

**2.3 Instruction Simplifications**

* **Original Code:**
  + Redundant stack allocations:

subq $32, %rsp

movq %rdi, -24(%rbp)

movl %esi, -28(%rbp)

movl $0, -8(%rbp)

* **Optimized Code:**
  + Registers directly replace the stack frame setup:

movq %rdi, %r12

movl %esi, %r13d

movl $0, %r8d

* **Benefit**: Simplifies code and removes unnecessary stack management.

**2.4 Loop Logic**

* **Original Code:**
  + Inner loop logic repeatedly accesses stack variables to retrieve indices and compare array values.
* **Optimized Code:**
  + Indices and array pointers are maintained in registers, reducing the overhead of memory access for conditions and updates.

**Benefit**: Improves inner loop performance, critical for nested loops.

**2.5 Consistency in Array Access**

* Both versions use similar assembly instructions for accessing array elements:

leaq 0(,%rax,4), %rdx

addq %rdx, %rax

movl (%rax), %edx

However, the optimized version uses register variables (%r12, %r9d) to replace stack variables.

**3. Performance Benefits**

* **Reduced Memory Latency**: Moving from stack to registers minimizes memory bottlenecks.
* **Improved Instruction Throughput**: Register operations are faster, allowing for better CPU utilization.
* **Cleaner Control Flow**: Reducing stack operations simplifies the generated assembly, making it easier for the CPU pipeline to process.

**4. Conclusion**

The optimized assembly version eliminates unnecessary stack usage by leveraging registers for loop counters, array pointers, and intermediate variables. These changes significantly reduce the overhead caused by memory accesses, leading to faster execution and more efficient use of CPU resources.

**Original vs Optimized sort Assembly code:**

\_Z4sortPii:

.LFB2277:

.cfi\_startproc

endbr64

pushq %rbp

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.cfi\_def\_cfa\_register 6

subq $32, %rsp

movq %rdi, %r12

movl %esi, %r13d

movl $0, %r8d

jmp .L6

.L10:

movl %r8d, %eax

subl $1, %eax

movl %eax, %r9d

jmp .L7

.L9:

movl %r9d, %edx

movq %r12, %rax

movl %edx, %esi

movq %rax, %rdi

call \_Z4swapPii

subl $1, %r9d

.L7:

cmpl $0, %r9d

js .L8

movl %r9d, %eax

cltq

leaq 0(,%rax,4), %rdx

movq %r12, %rax

addq %rdx, %rax

movl (%rax), %edx

movl %r9d, %eax

cltq

addq $1, %rax

leaq 0(,%rax,4), %rcx

movq %r12, %rax

addq %rcx, %rax

movl (%rax), %eax

cmpl %eax, %edx

jg .L9

.L8:

addl $1, %r8d

.L6:

movl %r8d, %eax

cmpl %r13d, %eax

jl .L10

nop

nop

leave

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

\_Z4sortPii:

.LFB2277:

.cfi\_startproc

endbr64

pushq %rbp

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.cfi\_def\_cfa\_register 6

subq $32, %rsp

movq %rdi, -24(%rbp)

movl %esi, -28(%rbp)

movl $0, -8(%rbp)

jmp .L6

.L10:

movl -8(%rbp), %eax

subl $1, %eax

movl %eax, -4(%rbp)

jmp .L7

.L9:

movl -4(%rbp), %edx

movq -24(%rbp), %rax

movl %edx, %esi

movq %rax, %rdi

call \_Z4swapPii

subl $1, -4(%rbp)

.L7:

cmpl $0, -4(%rbp)

js .L8

movl -4(%rbp), %eax

cltq

leaq 0(,%rax,4), %rdx

movq -24(%rbp), %rax

addq %rdx, %rax

movl (%rax), %edx

movl -4(%rbp), %eax

cltq

addq $1, %rax

leaq 0(,%rax,4), %rcx

movq -24(%rbp), %rax

addq %rcx, %rax

movl (%rax), %eax

cmpl %eax, %edx

jg .L9

.L8:

addl $1, -8(%rbp)

.L6:

movl -8(%rbp), %eax

cmpl -28(%rbp), %eax

jl .L10

nop

nop

leave

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

**Performance:**

|  |  |  |
| --- | --- | --- |
| Elements | Computer Generated Assembly | Modified Assembly |
| 0 | 0 | 0 |
| 1000 | 0.008947 | 0.002627 |
| 2000 | 0.020353 | 0.008948 |
| 3000 | 0.035199 | 0.015271 |
| 4000 | 0.043024 | 0.021787 |
| 5000 | 0.060948 | 0.038498 |
| 6000 | 0.081005 | 0.054723 |
| 7000 | 0.105568 | 0.076777 |
| 8000 | 0.129923 | 0.097555 |
| 9000 | 0.148859 | 0.110624 |
| 10000 | 0.18518 | 0.134205 |