Assignment 6 Solutions

Friday, November 18, 2022 19:59

2. [40] Suppose we've got a procedure that computes the inner product of two arrays u and v. Consider the following C code:

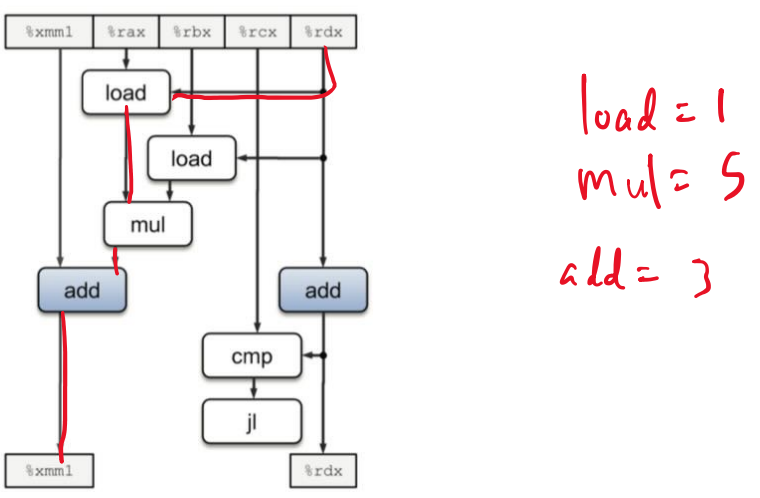
```
void inner(float *u, float *v, int length, float *dest) {
    int i;
    float sum = 0.0f;
    for (i = 0; i < length; ++i) {
        sum += u[i] * v[i];
    }
    *dest = sum;
}
```

The x86-64 assembly code for the inner loop is as follows:

```
# u in %rbx, v in %rax, length in %rcx, i in %rdx, sum in %xmm1
.L87:
    movss (%rbx, %rdx, 4), %xmm0 # Get u[i]

   mulss (%rax, %rdx, 4), %xmm0 # Multiply by v[i]
   adds %xmm0, %xmm1            # Add to sum
   addq $1, %rdx                # Increment i
   cmpq %rcx, %rdx              # Compare i to length
   jl .L87                      # If <, keep looping
```

Also consider this data-flow diagram for the above procedure:



load = 1
mul = 5
add = 3

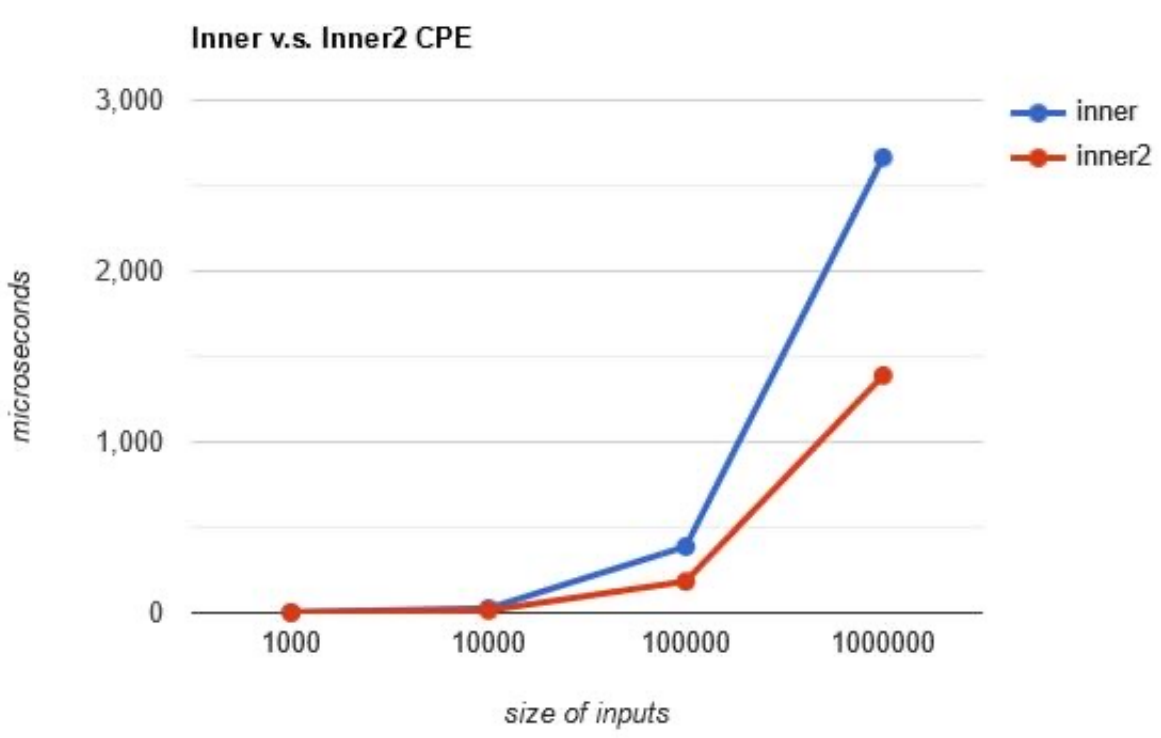
a. (10) Which operation(s) in the data-flow diagram above can NOT be parallelized? Hint: these will be the operation(s) that depend on the result of that operation from the previous loop iteration. Hint: see discussion around Figures 5.14 and 5.15. Write your answers in your solutions document.

Both adding the sum (adds %xmm0, %xmm1), and adding 1 to the iterator (addq $1, %rdx) cannot be parallelized. This is because sum and the iterator must first be calculated in the previous loop before being operated on due to the +=.

b. (10) Given your answer from part a, what is the best-case CPE for the loop as currently written? Assume that *float* addition has a latency of 3 cycles, *float* multiplication has a latency of 5 cycles, and all integer operations have a latency of 1 cycle. Hint: the best-case CPE will be latency of the slowest of the operation(s) you identified in part a. Write your answers in your solutions document.

5 Cycles per Execution. Take longest path in data-flow diagram, and observe that the multiplication is the most time consuming, and every operation up to it will finish.

d. (10) Using your code from part c, collect data on the execution times of *inner* and *inner2* with varying array lengths. Summarize your findings and argue whether *inner* or *inner2* is more efficient than the other (or not). Create a graph using appropriate data points to support your argument. Include your summary and graph in your solutions document. Compile with -Og.



Based on my findings it appears that inner 2 is generally faster by a factor of 2. Using loop unrolling, we can calculate 4 sums instead of one per execution. It is not improved by a factor of 4, rather by a factor of 2. This is likely due to hardware limitations, such as a lack of available registers for all computations in the execution.