LE11.3.1 Loop optimizations

0.0/1.0 point (ungraded)

In block structured languages such as C or Java, the scope of a variable declared locally within a block extends only over that block, i.e., the value of the local variable cannot be accessed outside the block. Conceptually, storage is allocated for the variable when the block is entered and deallocated when the block is exited. In many cases, this means the compiler is free to use a register to hold the value of the local variable instead of a memory location.

Consider the following C fragment:

```
int sum = 0;
{ int i;
  for (i = 0; i < 10; i = i+1) sum += i;
}
```

If this loop is compiled such that intermediate results are stored in memory, then the resulting compiled code would look like this:

```
      ST(R31,sum)
      ST(R31,i)
_L7:
      LD(sum,R0)
      LD(i,R1)
      ADD(R0,R1,R0)
      ST(R0,sum)
      ADDC(R1,1,R1)
      ST(R1,i)
      CMPLTC(R1,10,R0)
      BT(R0,_L7)
```

If instead the compilation of this code is optimized by using registers to hold the values of the local variables i and sum, then the compiled code would look like this:

```
      MOVE(R31,R2)          // R2 holds sum
      ST(R2,sum)
      MOVE(R31,R1)          // R1 holds i
_L5:
      ADD(R2,R1,R2)
      ADDC(R1,1,R1)
      CMPLTC(R1,10,R0)
      BT(R0,_L5)
      ST(R2,sum)
```

Define a memory access as any access to memory, i.e., instruction fetch, data read (LD), or data write (ST). Compare the total number of memory accesses generated by executing the optimized loop with the total number of memory access for the unoptimized loop.

Unoptimized loop:

Number of instructions in loop:

8 Answer: 8

Number of data accesses per loop iteration:

4 Answer: 4

Optimized loop:

Number of instructions in loop:

4 Answer: 4

Number of data accesses per loop iteration:

0 Answer: 0

Given that the loop is executed 10 times, what is the total number of memory accesses in the unoptimized loop versus the optimized loop?

Unoptimized loop: Total memory accesses:

120 Answer: 120

Optimized loop: Total memory accesses:

40 Answer: 40

Explanation
The unoptimized loop has 8 instructions in its loop plus 4 data accesses (2 loads and 2 stores). This means that across 10 iterations of the loop, there are a total of 120 memory accesses made.
In the optimized loop, the loop is reduced to 4 instructions with 0 data accesses within the loop. This means that across 10 iterations of this loop, there are a total of 40 memory accesses.

Some optimizing compilers "unroll" small loops to amortize the overhead of each loop iteration over more instructions in the body of the loop. For example, one unrolling of the loop above would be equivalent to rewriting the program as:

```
int sum = 0;
{ int i;
  for (i = 0; i < 10; i = i+2) { sum += i; sum += i+1; }
}
```

The hand-compiled unrolled loop looks like this:

```
     MOVE(R31,R2)          // R2 holds sum
     ST(R2,sum)
     MOVE(R31,R1)          // R1 holds i
_L5:
     ADD(R2,R1,R2)
     ADDC(R1,1,R0)
     ADD(R2,R0,R2)
     ADDC(R1,2,R1)
     CMPLTC(R1,10,R0)
     BT(R0,_L5)
     ST(R2,sum)
```

Unrolled loop:
Number of instructions in loop:

Answer: 6

Number of data accesses per loop iteration:

Answer: 0

Unrolled loop: Total memory accesses:

Answer: 30

Explanation
The unrolled loop has 6 instructions in its loop plus 0 data accesses. Note that in this loop, since two values of i are handled in each iteration of the loop, i is incremented by 2 each time through the loop, so the total number of loop iterations is 5. This means that the 6*5 = 30 total memory accesses are made when using this unrolled loop. This demonstrates that as a compiler performs a larger number of optimizations, the resulting number of memory accesses decreases.

Submit

Discussion

Hide Discussion

Topic: 11. Compilers / LE11.3

Add a Post

All Posts

Super Confused on Optimized loop

discussion posted 8 years ago by **apiper0770**

The course in my opinion is not very straight forward sometimes or the definitions I'm not very familiar with yet.

Can some one clarify what is the difference between an instruction and a data access in regards to the optimized loop.

In the unoptimized loop I counted the # of lines after (_L7) to get # of instructions and number of data accesses I counted was by counting # of Stores & Loads. This is how I got 8 and 4; however, using this same method doesn't work for optimized, so I must have done something wrong. Can someone help me to understand what I am doing wrong?

Thank you

This post is visible to everyone.

Add a Response

1 response

apiper0770
8 years ago

I think it would have been more beneficial to explain how the number of instructions and data access was obtained, rather then on explaining how to do the simple multiplication of one loops memory access by the # of loops in order to get total memory accesses.

The second part is rather obvious compared to the first, at least in my opinion.

Aww daaaaahhhh..... the loop ended at the branch instruction BT() so the store afterwards is not part of the loop. Now I understand.

posted 8 years ago by **apiper0770**

I had a similar conversation with myself :)

posted 8 years ago by **rhodesd**

must be some sort of special effect

posted 8 years ago by **wojd**

Add a comment

Showing all responses

Add a response:

Preview

Submit