A Tale of Two Assembly Codes

Mike Bailey
mjb@cs.oregonstate.edu

* With apologies to Charles Dickens

```c
int NumInThreadTeam;
int NumAtBarrier;
int NumGone;

void InitBarrier( int n )
{
    NumInThreadTeam = n;
    NumAtBarrier = 0;
}

void WaitBarrier( )
{
    omp_set_lock( &Lock );
    {
        NumAtBarrier++;
        if( NumAtBarrier == NumInThreadTeam )
        {
            NumGone = 0;
            NumAtBarrier = 0;
            // let all other threads return before this one unlocks:
            while( NumGone != NumInThreadTeam-1 );
            omp_unset_lock( &Lock );
            return;
        }
    }
    omp_unset_lock( &Lock );
    while( NumAtBarrier != 0 );
    #pragma omp atomic
    NumGone++;
}
```

---

From the Parallel Background Notes:

Tip #4 -- Sending a Message to the Optimizer:

The `volatile` Keyword

The `volatile` keyword is used to let the compiler know that another thread might be changing a variable "in the background", so don't make any assumptions about what can be optimized away.

```c
int val = 0;
```...

```c
volatile int val = 0;
```...

A good compiler optimizer will eliminate this code because it "knows" that, for all time, `val == 0`.

The `volatile` keyword tells the compiler optimizer that it cannot count on `val` being `== 0` here.

---

Assembly Code with -O3

```assembly
subq $8, %rsp
```

---

```assembly
volatile int NumInThreadTeam;
volatile int NumAtBarrier;
volatile int NumGone;
```
Assembly Code with volatile and –O3

```
subq $8, %rsp
movl $Lock, %edi
call omp_set_lock
movl NumAtBarrier(%rip), %eax
addl $1, %eax
movl %eax, NumAtBarrier(%rip)
movl NumAtBarrier(%rip), %edx
movl NumInThreadTeam(%rip), %eax
cmpl %eax, %edx
je .L128
movl $Lock, %edi
call omp_unset_lock
.L126:
movl NumAtBarrier(%rip), %eax
testl %eax, %eax
jne .L126
lock addl $1, NumGone(%rip)
addq $8, %rsp
ret
.L128:
movl $0, NumGone(%rip)
movl $0, NumAtBarrier(%rip)
.L124:
movl NumInThreadTeam(%rip), %edx
movl NumGone(%rip), %eax
subl $1, %edx
cmpl %eax, %edx
jne .L124
movl $Lock, %edi
addq $8, %rsp
jmp omp_unset_lock
```

Moral of the Story

I should read my own notes more often... 😊