**The amazing Duff’s Device by Tom Duff!**

Subject: Re: Explanation, please!

Summary: Original citation

From: td@alice.UUCP (Tom Duff)

Organization: AT&T Bell Laboratories, Murray Hill NJ

Date: 29 Aug 88 20:33:51 GMT

Message-ID: <8144@alice.UUCP>

I normally do not read comp.lang.c, but Jim McKie told me that ``Duff's

device'' had come up in comp.lang.c again. I have lost the version that was

sent to netnews in May 1984, but I have reproduced below the note in which I

originally proposed the device. (If anybody has a copy of the netnews version,

I would gratefully receive a copy at research!td or td@research.att.com.)

To clear up a few points:

1. The point of the device is to express general loop unrolling directly in

C. People who have posted saying `just use memcpy' have missed the point, as

have those who have criticized it using various machine-dependent memcpy

implementations as support. In fact, the example in the message is not

implementable as memcpy, nor is any computer likely to have an memcpy-like

idiom that implements it.

2. Somebody claimed that while the device was named for me, I probably

didn't invent it. I almost certainly did invent it. I had definitely not seen

or heard of it when I came upon it, and nobody has ever even claimed prior

knowledge, let alone provided dates and times. Note the headers on the message

below: apparently I invented the device on November 9, 1983, and was proud (or

disgusted) enough to send mail to dmr. Please note that I do not claim to have

invented loop unrolling, merely this particular expression of it in C.

3. The device is legal dpANS C. I cannot quote chapter and verse, but Larry

Rosler, who was chairman of the language subcommittee (I think), has assured me

that X3J11 considered it carefully and decided that it was legal. Somewhere I

have a note from dmr certifying that all the compilers that he believes in

accept it. Of course, the device is also legal C++, since Bjarne uses it in

his book.

4. Somebody invoked (or more properly, banished) the `false god of

efficiency.' Careful reading of my original note will put this slur to rest.

The alternative to genuflecting before the god of code-bumming is finding a

better algorithm. It should be clear that none such was available. If your

code is too slow, you must make it faster. If no better algorithm is

available, you must trim cycles.

5. The same person claimed that the device wouldn't exhibit the desired

speed-up. The argument was flawed in two regards: first, it didn't address

the performance of the device, but rather the performance of one of its few

uses (implementing memcpy) for which many machines have a high-performance

idiom. Second, the poster made his claims in the absence of timing data, which

renders his assertion suspect. A second poster tried the test, but botched the

implementation, proving only that with diligence it is possible to make

anything run slowly.

6. Even Henry Spencer, who hit every other nail square on the end with the

flat round thing stuck to it, made a mistake (albeit a trivial one). Here is

Henry replying to bill@proxftl.UUCP (T. William Wells):

>>... Dollars to doughnuts this

>>was written on a RISC machine.

>Nope. Bell Labs Research uses VAXen and 68Ks, mostly.

I was at Lucasfilm when I invented the device.

7. Transformations like this can only be justified by measuring the

resulting code. Be careful when you use this thing that you don't unwind the

loop so much that you overflow your machine's instruction cache. Don't try to

be smarter than an over-clever C compiler that recognizes loops that implement

block move or block clear and compiles them into machine idioms.

Here then, is the original document describing Duff's device:

From research!ucbvax!dagobah!td Sun Nov 13 07:35:46 1983

Received: by ucbvax.ARPA (4.16/4.13) id AA18997; Sun, 13 Nov 83 07:35:46 pst

Received: by dagobah.LFL (4.6/4.6b) id AA01034; Thu, 10 Nov 83 17:57:56 PST

Date: Thu, 10 Nov 83 17:57:56 PST

From: ucbvax!dagobah!td (Tom Duff)

Message-Id: <8311110157.AA01034@dagobah.LFL>

To: ucbvax!decvax!hcr!rrg, ucbvax!ihnp4!hcr!rrg, ucbvax!research!dmr, ucbvax!research!rob

Consider the following routine, abstracted from code which copies an array of

shorts into the Programmed IO data register of an Evans & Sutherland Picture

System II:

send(to, from, count)

register short \*to, \*from;

register count;

{

do

\*to = \*from++;

while(--count>0);

}

(Obviously, this fails if the count is zero.)

The VAX C compiler compiles the loop into 2 instructions (a movw and a sobleq,

I think.) As it turns out, this loop was the bottleneck in a real-time

animation playback program which ran too slowly by about 50%. The standard way

to get more speed out of something like this is to unwind the loop a few times,

decreasing the number of sobleqs. When you do that, you wind up with a

leftover partial loop. I usually handle this in C with a switch that indexes a

list of copies of the original loop body. Of course, if I were writing

assembly language code, I'd just jump into the middle of the unwound loop to

deal with the leftovers. Thinking about this yesterday, the following

implementation occurred to me:

send(to, from, count)

register short \*to, \*from;

register count;

{

register n=(count+7)/8;

switch(count%8){

case 0: do{ \*to = \*from++;

case 7: \*to = \*from++;

case 6: \*to = \*from++;

case 5: \*to = \*from++;

case 4: \*to = \*from++;

case 3: \*to = \*from++;

case 2: \*to = \*from++;

case 1: \*to = \*from++;

}while(--n>0);

}

}

Disgusting, no? But it compiles and runs just fine. I feel a combination of

pride and revulsion at this discovery. If no one's thought of it before, I

think I'll name it after myself.

It amazes me that after 10 years of writing C there are still little corners

that I haven't explored fully. (Actually, I have another revolting way to use

switches to implement interrupt driven state machines but it's too horrid to go

into.)

Many people (even bwk?) have said that the worst feature of C is that switches

don't break automatically before each case label. This code forms some sort of

argument in that debate, but I'm not sure whether it's for or against.

yrs trly

Tom