Subject: Use of Temporary( ) vs an Optimised Compiler
Posted by Martin Downing on Mon, 26 Nov 2001 15:38:10 GMT
View Forum Message <> Reply to Message

Here's a thought for the day:
we have all had to get used to using the temporary function to enable memory efficient code. Some of us less effectively than others!
ie: instead of
    a = 2*a + b/a
write
    a = 2*a + b/TEMPORARY(a)

Personally although good practice I find it makes code hard to read. Who agrees that this could and should be dealt with at the compilation stage, obviously if A is being reassigned then the previous contents are lost so the compiler could reuse A when processing the last copy of A on the right hand side. Would that be so hard for RSI to implement?

Martin

Martin Downing,
Clinical Research Physicist,
Grampian Orthopaedic RSA Research Centre,
Woodend Hospital, Aberdeen, AB15 6LS.

Subject: Re: Use of Temporary( ) vs an Optimised Compiler
Posted by Craig Markwardt on Tue, 27 Nov 2001 05:42:47 GMT
View Forum Message <> Reply to Message

"Martin Downing" <martin.downing@ntlworld.com> writes:

> Here's a thought for the day:
> we have all had to get used to using the temporary function to enable memory
> efficient code. Some of us less effectively than others!
> ie: instead of
>   a = 2*a + b/a
> write
>   a = 2*a + b/TEMPORARY(a)
>
> Personally although good practice I find it makes code hard to read. Who
> agrees that this could and should be dealt with at the compilation stage,
> obviously if A is being reassigned then the previous contents are lost so
> the compiler could reuse A when processing the last copy of A on the right
> hand side. Would that be so hard for RSI to implement?

I agree, Martin. A compiler writer would know exactly when a variable

on the right hand side is being reassigned. However, your example
points out at least one of the problems.

>   a = 2*a + b/TEMPORARY(a)

Since A appears twice on the right hand side, the compiler would need
to be smart enough to not overwrite A after its first appearance. In
fact, I am not sure that IDL makes any guarantees about order of
evaluation and side effects. Isn't it possible that the TEMPORARY()
gets called before the first A is evaluated? [Not sure on this, but
that's why I avoid the situation.]

Second of all, if the compiler automatically TEMPORARY()'ed every
variable that was reassigned, it makes debugging harder. What if your
expression was:

>   a = 2*a + F(B)

If F(B) crashed after 2*A was evaluated, then there may be no way to
recover the original value of A. So, there would probably need to be
a "debug" vs. "performance" compilation flag.

Craig

--
 --- --------------
Craig B. Markwardt, Ph.D.   EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
 --- --------------

Subject: Re: Use of Temporary( ) vs an Optimised Compiler
Posted by Martin Downing on Tue, 27 Nov 2001 10:42:10 GMT
View Forum Message <> Reply to Message

"Craig Markwardt" <craigmnet@cow.physics.wisc.edu> wrote in message
news:onwv0cahwo.fsf@cow.physics.wisc.edu...
>
> I agree, Martin. A compiler writer would know exactly when a variable
> on the right hand side is being reassigned. However, your example
> points out at least one of the problems.
>
>>   a = 2*a + b/TEMPORARY(a)
>
> Since A appears twice on the right hand side, the compiler would need
> to be smart enough to not overwrite A after its first appearance. In
> fact, I am not sure that IDL makes any guarantees about order of

> evaluation and side effects. Isn't it possible that the TEMPORARY()
> gets called before the first A is evaluated? [Not sure on this, but
> that's why I avoid the situation.]
>
Hi Craig

Oh boy, I had thought IDL followed left to right calculation with usual precidence rules, which I reassured myself of by running the above statement to see whether it crashed! (a good proof or what!) Come to think of it I'm not sure I have found such claims anywhere in the manuals! Dont worry I'd use brackets where it matters anyway. Either way I would have thought the compiler could count the number of instances of the left hand variable on the right and release the memory on the last one. I agree a flag would be good for debugging.

Martin

> Second of all, if the compiler automatically TEMPORARY()'ed every
> variable that was reassigned, it makes debugging harder. What if your
> expression was:
>
>> a = 2*a + F(B)
>
> If F(B) crashed after 2*A was evaluated, then there may be no way to
> recover the original value of A. So, there would probably need to be
> a "debug" vs. "performance" compilation flag.
>
Agreed

Subject: Re: Use of Temporary( ) vs an Optimised Compiler
Posted by Pavel A. Romashkin on Tue, 27 Nov 2001 17:24:01 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
>> a = 2*a + b/TEMPORARY(a)
>
> Since A appears twice on the right hand side, the compiler would need
> to be smart enough to not overwrite A after its first appearance. In
> fact, I am not sure that IDL makes any guarantees about order of
> evaluation and side effects. Isn't it possible that the TEMPORARY()
> gets called before the first A is evaluated?

I recently had to think about memory allocation in IDL for the first time as I had to use large arays (well, some 4E+7 points, astronomy guys don't laugh).
In my case it appeared faster (and sometimes the only way possible to

avoid infufficient memory errors) to split the above expression

a = 2*a +b/temporary(a)

into

a = 2*temporary(a)
a = 2*b/temporary(a)

It takes a while even to do math on xE+7 points, and allocating arrays
further slows things down.
In this simple case, this is easy to do, but some expressions were quite
hard to split like this. There are probably better ways.
Cheers,
Pavel

Subject: Re: Use of Temporary( ) vs an Optimised Compiler
Posted by Paul van Delst on Tue, 27 Nov 2001 18:46:02 GMT
View Forum Message <> Reply to Message

"Pavel A. Romashkin" wrote:
>
> Craig Markwardt wrote:
>>
>>> a = 2*a + b/TEMPORARY(a)
>>
>> Since A appears twice on the right hand side, the compiler would need
>> to be smart enough to not overwrite A after its first appearance. In
>> fact, I am not sure that IDL makes any guarantees about order of
>> evaluation and side effects. Isn't it possible that the TEMPORARY()
>> gets called before the first A is evaluated?
>
> I recently had to think about memory allocation in IDL for the first
> time as I had to use large arays (well, some 4E+7 points, astronomy guys
> don't laugh).
> In my case it appeared faster (and sometimes the only way possible to
> avoid infufficient memory errors) to split the above expression
>
> a = 2*a +b/temporary(a)
>
> into
>
> a = 2*temporary(a)
> a = 2*b/temporary(a)
>
> It takes a while even to do math on xE+7 points, and allocating arrays
> further slows things down.

> In this simple case, this is easy to do, but some expressions were quite
> hard to split like this. There are probably better ways.

How about,

a = SQRT(2*TEMPORARY(a)^2 + b)

Would the use of the exponentiation operator and SQRT() function be slower than the two step process above for supa-big arrays?

paulv

--
Paul van Delst Religious and cultural
CIMSS @ NOAA/NCEP purity is a fundamentalist
Ph: (301)763-8000 x7274 fantasy
Fax:(301)763-8545 V.S.Naipaul

Subject: Re: Use of Temporary( ) vs an Optimised Compiler
Posted by Liam E. Gumley on Tue, 27 Nov 2001 19:13:39 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
[stuff deleted]
> In fact, I am not sure that IDL makes any guarantees about order of
> evaluation and side effects.

For an explanation of operator precedence:

IDL> ? operator precedence

In short, it's not the order of evaluation that matters: it's the precedence of the operators in the expression. The order matters only when you have operators of equal precedence, in which case the expression is evaluated from left to right.

As the documentation says, "when in doubt, parenthesize".

Cheers,
Liam.
Practical IDL Programming
http://www.gumley.com/
