Subject: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Sidney Cadot on Wed, 17 Mar 2004 09:47:13 GMT
View Forum Message <> Reply to Message

Hi all,

For a system we're making, a rather big IDL file is generated containing well over 12,000 function definitions, accompanied by a selector function (see below for a rationale).

What we're seeing is that in command-line IDL, this works like a charm: compilation of the file takes about 4--5 seconds on a reasonably fast machine, which is acceptable.

However, when this file is compiled from within IDLDE, this takes well over three minutes-- rougly a factor 60 increase(!)

Does anybody know what causes this, and perhaps a solution?

We tried pre-compiling the functions using a SAV file; this yields a significant increase both in IDL (cmd line version): 3 sec, and IDLDE (used time down to 87 seconds), but the relative difference is still quite puzzling.

Best regards,

Sidney Cadot
Science and Technology Corp., The Netherlands

P.S. the reason we're doing this is that we need to implement a string-based map with optiomal performance, like this:

```
FUNCTION f_tom
  RETURN, 123
END

FUNCTION f_dick
  RETURN, 456
END

FUNCTION f_harry
  RETURN, 789
END
```

```
FUNCTION f, name
  CATCH, error_status
  IF error_status EQ 0 THEN RETURN, -1
  RETURN, call_function("f_" + name)
END
```

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by mvukovic on Wed, 17 Mar 2004 20:16:02 GMT
View Forum Message <> Reply to Message

Sidney Cadot <sidney@jigsaw.nl> wrote in message news:<1079516867.600179@euler.servers.luna.net>...
> Hi all,
>
> For a system we're making, a rather big IDL file is generated containing
> well over 12,000 function definitions, accompanied by a selector
> function (see below for a rationale).
>
> What we're seeing is that in command-line IDL, this works like a charm:
> compilation of the file takes about 4--5 seconds on a reasonably fast
> machine, which is acceptable.
>
> However, when this file is compiled from within IDLDE, this takes well
> over three minutes-- rougly a factor 60 increase(!)
>
> Does anybody know what causes this, and perhaps a solution?
>
> We tried pre-compiling the functions using a SAV file; this yields a
> significant increase both in IDL (cmd line version): 3 sec, and IDLDE
> (used time down to 87 seconds), but the relative difference is still
> quite puzzling.
>
> Best regards,
>
> Sidney Cadot
> Science and Technology Corp., The Netherlands
>
>
>
>
> P.S. the reason we're doing this is that we need to implement a
> string-based map with optiomal performance, like this:
>
> FUNCTION f_tom
> RETURN, 123
> END

>
> FUNCTION f_dick
>   RETURN, 456
> END
>
> FUNCTION f_harry
>   RETURN, 789
> END
>
> FUNCTION f, name
>   CATCH, error_status
>   IF error_status EQ 0 THEN RETURN, -1
>   RETURN, call_function("f_" + name)
> END

Out of curiosity, would a structure work here:
a={f_tom:123,f_dick:456,f_harry:789...} ?

It could be created using create_struct.

Retrieve info using

a=str.f_dick

Curious minds want to know :-)

(And never mind about ``curiosity kills the cat'')

Mirko

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by andrew.cool on Wed, 17 Mar 2004 22:02:21 GMT
View Forum Message <> Reply to Message

Sidney Cadot <sidney@jigsaw.nl> wrote in message
news:<1079516867.600179@euler.servers.luna.net>...
> Hi all,
>
> For a system we're making, a rather big IDL file is generated containing
> well over 12,000 function definitions, accompanied by a selector
> function (see below for a rationale).
>
> What we're seeing is that in command-line IDL, this works like a charm:
> compilation of the file takes about 4--5 seconds on a reasonably fast
> machine, which is acceptable.
>

> However, when this file is compiled from within IDLDE, this takes well
> over three minutes-- rougly a factor 60 increase(!)
>
> Does anybody know what causes this, and perhaps a solution?
>
> We tried pre-compiling the functions using a SAV file; this yields a
> significant increase both in IDL (cmd line version): 3 sec, and IDLDE
> (used time down to 87 seconds), but the relative difference is still
> quite puzzling.
>
> Best regards,
>
> Sidney Cadot
> Science and Technology Corp., The Netherlands
>
>
>
>
> P.S. the reason we're doing this is that we need to implement a
> string-based map with optiomal performance, like this:
>
> FUNCTION f_tom
> RETURN, 123
> END
>
> FUNCTION f_dick
> RETURN, 456
> END
>
> FUNCTION f_harry
> RETURN, 789
> END
>
> FUNCTION f, name
> CATCH, error_status
> IF error_status EQ 0 THEN RETURN, -1
> RETURN, call_function("f_" + name)
> END

OK, me dumb bunny - me no know what a string based map is.
But based on your example above, how about this?

```
map_array = Strarr(12000,2)

map_array(0,1) = string(indgen(12000),form='(i5.5)')
map_array(5000,0) ='dick'
```

```
t = Systime(1)
found_index = Where(map_array(*,0) EQ 'dick')
print,'Time taken = ',Systime(1) - t,' seconds'
print,'Found Index = ',found_index
ret_value = map_array(found_index,1)
print,'Returned Value = ',ret_value
```

Now on my PC, Time taken = 0.00000000 seconds,
which I'd call pretty close to "optiomal".
Do you really need 12000 function definitions?

Andrew Cool
DSTO, Adelaide, South Australia

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by justspam03 on Thu, 18 Mar 2004 09:13:32 GMT
View Forum Message <> Reply to Message

Hi,

your example is just generic for the kind of problem you want
to solve, I assume. Otherwise why not use a hash? A very
simple implementation (unsorted arrays) on a Pentium IV,
2.6 GHz, IDL 6.0 yields

Setting 12000 random values (key: string, value: integer):
  mean 0.15 ms per entry (total ~2 seconds)
Random access of 12000 values from this set:
  mean 0.3 ms per access, (total ~3.5 s)

Is the access via call_function much faster?
Cheers
 Oliver

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Sidney Cadot on Thu, 18 Mar 2004 20:32:23 GMT
View Forum Message <> Reply to Message

Mirko Vukovic wrote:
>> P.S. the reason we're doing this is that we need to implement a
>> string-based map with optiomal performance, like this:
>>

```
>> FUNCTION f_tom
>>    RETURN, 123
>> END
>>
>> FUNCTION f_dick
>>    RETURN, 456
>> END
>>
>> FUNCTION f_harry
>>    RETURN, 789
>> END
>>
>> FUNCTION f, name
>>    CATCH, error_status
>>    IF error_status EQ 0 THEN RETURN, -1
>>    RETURN, call_function("f_" + name)
>> END
>
>
>  Out of curiosity, would a structure work here:
>  a={f_tom:123,f_dick:456,f_harry:789...}  ?
>
>  It could be created using create_struct.
>
>  Retrieve info using
>
>  a=str.f_dick
>
>  Curious minds want to know :-)
```

Your idea is sound, but I am not aware of a way to retrieve the index of a tag-name based on its name.

You assume that "f_dick" is available at compile time, whereas I need to resolve the string at runtime. Something like this would work:

```
i = TAG_INDEX("f_dick", str)
value = str.(i)
```

... But only if functionality to get a tag index can be retrieved from a struct (anyone knows how to do this?) and if its fast, i.e. if IDL implements it via a hash table or similar.

Best regards,

Sidney

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Sidney Cadot on Thu, 18 Mar 2004 20:36:00 GMT
View Forum Message <> Reply to Message

Andrew Cool wrote:

> OK, me dumb bunny - me no know what a string based map is.
> But based on your example above, how about this?
>
> map_array = Strarr(12000,2)
>
> map_array(0,1) = string(indgen(12000),form='(i5.5)')
> map_array(5000,0) ='dick'
>
> t = Systime(1)
> found_index = Where(map_array(*,0) EQ 'dick')
> print,'Time taken = ',Systime(1) - t,' seconds'
> print,'Found Index = ',found_index
> ret_value = map_array(found_index,1)
> print,'Returned Value = ',ret_value

This uses a linear search. This is a lot slower than IDL's internal hashing, which is basically O(1).

> Now on my PC, Time taken = 0.00000000 seconds,
> which I'd call pretty close to "optiomal".

Try calling it a million times and compare it to my solution; you will see a significant difference.

> Do you really need 12000 function definitions?

I have yet to see a better solution.

Best regards,

Sidney

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Sidney Cadot on Thu, 18 Mar 2004 20:40:38 GMT
View Forum Message <> Reply to Message

Oliver Thilmann wrote:

> Hi,

>
> your example is just generic for the kind of problem you want
> to solve, I assume. Otherwise why not use a hash? A very
> simple implementation (unsorted arrays) on a Pentium IV,
> 2.6 GHz, IDL 6.0 yields
>
> Setting 12000 random values (key: string, value: integer):
> mean 0.15 ms per entry (total ~2 seconds)
> Random access of 12000 values from this set:
> mean 0.3 ms per access, (total ~3.5 s)
>
> Is the access via call_function much faster?

I'm afraid to sound terribly stupid here, but is there support for hashing in IDL? I haven't been able to find it.

My laborious trick is nothing more than circumventing the lack of hashing as a standard feature in IDL (by piggybacking on the internal hashing IDL uses for function names). If you know a better way, I would be very much interested!

Best regards, Sidney

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by btt on Thu, 18 Mar 2004 20:50:56 GMT
View Forum Message <> Reply to Message

Sidney Cadot wrote:
> Mirko Vukovic wrote:
>
>>> P.S. the reason we're doing this is that we need to implement a
>>> string-based map with optiomal performance, like this:
>>>
>>> FUNCTION f_tom
>>> RETURN, 123
>>> END
>>>
>>> FUNCTION f_dick
>>> RETURN, 456
>>> END
>>>
>>> FUNCTION f_harry
>>> RETURN, 789
>>> END
>>>
>>> FUNCTION f, name

```
>>>    CATCH, error_status
>>>    IF error_status EQ 0 THEN RETURN, -1
>>>    RETURN, call_function("f_" + name)
>>>  END
>>
>>
>>
>>  Out of curiosity, would a structure work here:
>>  a={f_tom:123,f_dick:456,f_harry:789...}  ?
>>
>>  It could be created using create_struct.
>>
>>  Retrieve info using
>>  a=str.f_dick
>>
>>  Curious minds want to know :-)
>
>
>  Your idea is sound, but I am not aware of a way to retrieve the index of
>  a tag-name based on its name.
>
>  You assume that "f_dick" is available at compile time, whereas I need to
>   resolve the string at runtime. Something like this would work:
>
>  i = TAG_INDEX("f_dick", str)
>  value = str.(i)
>
>  ... But only if functionality to get a tag index can be retrieved from a
>  struct (anyone knows how to do this?) and if its fast, i.e. if IDL
>  implements it via a hash table or similar.
>
```

How about this ?

```
IDL> print, tag_index(['color', 'clip', 'banana'], !P, count = count) &
print, count
          4          3         -1
          2
```

```
;-----start
FUNCTION TAG_INDEX, tagName, str, count = count

tags = TAG_NAMES(str)

n = n_elements(tagName)
```

```
tag_index = LonArr(n, /noZero)

Count = 0L
For i = 0L, n-1 Do begin
 tag_index[i] = (WHERE(tags EQ StrUpCase(tagName[i]), cnt))[0]
 Count = Count + (cnt GT 0)
EndFor

Return, tag_index
END
;-----finish
```

Ben

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Sidney Cadot on Thu, 18 Mar 2004 22:48:09 GMT
View Forum Message <> Reply to Message

Ben Tupper wrote:

```
>  Sidney Cadot wrote:
>
>>  Mirko Vukovic wrote:
>>
>>>>  P.S. the reason we're doing this is that we need to implement a
>>>>  string-based map with optiomal performance, like this:
>>>>
>>>>  FUNCTION f_tom
>>>>    RETURN, 123
>>>>  END
>>>>
>>>>  FUNCTION f_dick
>>>>    RETURN, 456
>>>>  END
>>>>
>>>>  FUNCTION f_harry
>>>>    RETURN, 789
>>>>  END
>>>>
>>>>  FUNCTION f, name
>>>>    CATCH, error_status
>>>>    IF error_status EQ 0 THEN RETURN, -1
>>>>    RETURN, call_function("f_" + name)
>>>>  END
>>>
```

>>>
>>>
>>>
>>> Out of curiosity, would a structure work here:
>>> a={f_tom:123,f_dick:456,f_harry:789...} ?
>>>
>>> It could be created using create_struct.
>>>
>>> Retrieve info using
>>> a=str.f_dick
>>>
>>> Curious minds want to know :-)
>>
>>
>>
>> Your idea is sound, but I am not aware of a way to retrieve the index
>> of a tag-name based on its name.
>>
>> You assume that "f_dick" is available at compile time, whereas I need
>> to resolve the string at runtime. Something like this would work:
>>
>> i = TAG_INDEX("f_dick", str)
>> value = str.(i)
>>
>> ... But only if functionality to get a tag index can be retrieved from
>> a struct (anyone knows how to do this?) and if its fast, i.e. if IDL
>> implements it via a hash table or similar.
>>
>
> How about this ? [[code snipped]]

Thanks for the effort, but this sort of defeats the purpose of the whole exercise, which is to have a fast mapping function. Your solution is linear search (the TAG_NAMES and WHERE functions), which is too slow for out application.

Best regards,

Sidney

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by JD Smith on Fri, 19 Mar 2004 01:11:11 GMT
View Forum Message <> Reply to Message

On Thu, 18 Mar 2004 21:40:38 +0100, Sidney Cadot wrote:

> Oliver Thilmann wrote:
>
>> Hi,
>>
>> your example is just generic for the kind of problem you want
>> to solve, I assume. Otherwise why not use a hash? A very
>> simple implementation (unsorted arrays) on a Pentium IV,
>> 2.6 GHz, IDL 6.0 yields
>>
>> Setting 12000 random values (key: string, value: integer):
>> mean 0.15 ms per entry (total ~2 seconds)
>> Random access of 12000 values from this set:
>> mean 0.3 ms per access, (total ~3.5 s)
>>
>> Is the access via call_function much faster?
>
> I'm afraid to sound terribly stupid here, but is there support for
> hashing in IDL? I haven't been able to find it.
>
> My laborious trick is nothing more than circumventing the lack of
> hashing as a standard feature in IDL (by piggybacking on the internal
> hashing IDL uses for function names). If you know a better way, I would
> be very much interested!

I think he means just using linear search, ala WHERE. This technically is a form of hashing: it just happens to utilize just one hash bucket (alright, a useless form ;).

Anyway, that's some clever use of function name searching for free hashing. IDL does not expose any internal hashing functionality, but Craig wrote a hash object which works reasonably well. I don't find it on his site, but perhaps he'd be willing to share.

What your method fails to offer that a real hash would is the ability to create new hash entries at run-time: all of your hash strings are fixed at runtime, which means you could transform them beforehand to integers (e.g. just use the sort index), and index a large static array instead, which would be orders of magnitude faster. Strings of course may be more convenient, but in the fixed string-space case they aren't technically necessary. Another variant on this would be akin to a C macro: just create a batch file for input using named variables like:

```
;; map_include.pro
dick=0
frank=1
harry=2
tom=3
...
```

```
zappa=11999

map=[456,222,789,123,...,777]


;; my map-using routine
pro mymaproutine
  @map_include
  print,map[zappa], ' is not as good as ',map[frank]
end
```

That will take a bit to compile, but once it compiles it should fly, and of course allows arrays, etc. If you insist on having a string mapper function, you can cheat:

```
function f, name
  @map_include
  ind=routine_names(name,FETCH=0)
  return,map[ind]
end
```

This doesn't really solve the real problem, which would include the ability to add more string keys at runtime, but it may be as much as you need. In the meantime, write your friendly IDL support techs and request a decent internal hash type, or at least a front end to a hash interface!

JD

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by justspam03 on Fri, 19 Mar 2004 10:06:02 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@as.arizona.edu> wrote in message news:<pan.2004.03.19.01.11.11.181250@as.arizona.edu>...

```
>>>
>>>  Setting 12000 random values (key: string, value: integer):
>>>    mean 0.15 ms per entry (total  ~2 seconds)
>>>  Random access of 12000 values from this set:
>>>    mean 0.3 ms per access, (total ~3.5 s)
>>>
>
>  I think he means just using linear search, ala WHERE.  This
>  technically is a form of hashing: it just happens to utilize just one
>  hash bucket (alright, a useless form ;).
```

>
> Anyway, that's some clever use of function name searching for free
> hashing. IDL does not expose any internal hashing functionality, but
> Craig wrote a hash object which works reasonably well. I don't find
> it on his site, but perhaps he'd be willing to share.
>

JD is right of course in that this not a hash, but an associative
array and the linear execution time ('set' has to check for existing
keys) is due to the use of 'where'.

Performance was good enough for me, though (with only a handful of
keys per array), so I didn't care any further about using hashes.

You can find a hash_table implementation on the RSI user contribution
site. Quick performance test for 12000 sets/gets:

#hashes	set/get per entry (ms)
13	6
101	0.8
1001	0.15
12000	0.08/0.05

Talking about it:
How would you calculate a hash value from a string? In C I would
base it on the ASCII value of the chars, but in IDL? Above mentioned
implementation converts the string via byte() and then loops over
the resulting array. Is there a faster way (loops always take so long)?
Cheers
Oliver

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by mwvogel on Fri, 19 Mar 2004 11:30:54 GMT

View Forum Message <> Reply to Message

> You can find a hash_table implementation on the RSI user contribution
> site. Quick performance test for 12000 sets/gets:
> #hashes set/get per entry (ms)
> 13 6
> 101 0.8
> 1001 0.15
> 12000 0.08/0.05
>
> Talking about it:
> How would you calculate a hash value from a string? In C I would
> base it on the ASCII value of the chars, but in IDL? Above mentioned
> implementation converts the string via byte() and then loops over

> the resulting array. Is there a faster way (loops always take so long)?

For real short strings (up to approx 11 chars) one could replace

```
        ascii = ulong(byte(key))
        total = 0UL
        for i = 0, n_elements(ascii) - 1 do begin
           total = total * 37UL + ascii[i]
              ; 37UL is a magic number suggested by the literature
        endfor
        return, total
with
        ascii = ULONG(BYTE(key))
        total = ULONG(TOTAL(ascii *
ULONG(37D^(N_ELEMENTS(ascii)-FINDGEN(N_ELEMENTS(ascii)) -1)), /DOUBLE))
        return, total
```

However, TOTAL() produces a double, and is therefore prone to roundoff errors, possibly reducing the hashing efficiency. Also, I am not so sure that the for loop is that much slower for small arrays. At least the original code is easier to read :-)

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Craig Markwardt on Fri, 19 Mar 2004 17:26:21 GMT
View Forum Message <> Reply to Message

justspam03@yahoo.de (Oliver Thilmann) writes:
...
> How would you calculate a hash value from a string? In C I would
> base it on the ASCII value of the chars, but in IDL? Above mentioned
> implementation converts the string via byte() and then loops over
> the resulting array. Is there a faster way (loops always take so long)?

As JD says, I don't have my own hashtable class on my web page, but I can send it upon request for now. As Vogel suggest, I avoid looping by using TOTAL and a precalculated array of hash values. I modeled some of the interface after the container class.

Craig

--

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

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by richard.martin on Mon, 22 Mar 2004 13:49:23 GMT
View Forum Message <> Reply to Message

The threads seem to have got off topic again. So why is IDLDE so rubbish? Don't know! But then I guess like all the sain people round I use emacs-IDLWAVE precisely because IDLDE is so bad.

As for your programing task, "Searching in string spaces", it looks very similar to the WHERE_AINB problem, which we all know should be done by the
king of IDL routines VALUE_LOCATE (Controversial! Heresy even!) as in

```
ord=VALUE_LOCATE(nameindex, name)
result=(nameindex(ord) ne name) ? -1 : namevalue(ord)
```

The {nameindex} must be sorted.

Your "Multi-Function" method is still quicker than this (but only by a factor of two), but the above can be vectorised very easily.

And finaly, for all of you out there that leave the complete history of the thread in your replies, the <DELETE>-key can normaly be found just above the
<CR>-key, top right-hand corner of the QWERTY bit, end of the row with all the numbers on it ;-).

Rich.

Subject: Re: Compiling file with many functions: huge performance difference between IDL and IDLDE
Posted by Mark Hadfield on Mon, 22 Mar 2004 23:45:44 GMT
View Forum Message <> Reply to Message

RM wrote:
> So why is IDLDE so
> rubbish? Don't know!

Because it won't let you compile 3000 functions per second?

I think the OP's idea is very clever, but it's asking the system to do something that was never anticipated by the designers. (Except, perhaps, in a nightmare.)

I must admit I'm curious as to why IDLDE addds this overhead, and whether it does it on all platforms. I'm sorry but I don't consider

"it's so rubbish" to be an explanation.

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)