Subject: Re: HASH makes too many temporaries
Posted by markb77 on Tue, 19 Mar 2013 08:55:10 GMT
View Forum Message <> Reply to Message

I would add to this that it is impossible to distinguish between scalars and arrays of HASH or LIST variables using the normal SIZE or ISA(/ARRAY) commands, as illustrated in the code below. Still waiting to hear from ITTVIS on the proper way to do this. Having to check the TYPENAME seems backwards.

```
pro test

   a = hash('first', 1, 'second', 2)
   b = hash('third', 3, 'fourth', 4, 'fifth', 5)

   print, 'Output of SIZE() for a HASH varible:'
   print, size(a, /STRUCTURE)

   c = [a,b]
   print, 'Output of SIZE() for an array of HASHes:'
   print, size(c, /STRUCTURE)

   print, 'Output of ISA(/ARRAY) for scalar hash:'
   print, isa(a, /array)

   print, 'Output of ISA(/ARRAY) for array of hashes:'
   print, isa(c, /array)

   help, a, output=ahelp
   help, c, output=chelp

   print, 'HELP for scalar hash:'
   print, ahelp

   print, 'HELP for array of hashes:'
   print, chelp

   print, 'TYPENAME for scalar hash:', typename(a)
   print, 'TYPENAME for array of hashes:', typename(c)

end

Output of SIZE() for a HASH varible:
{ OBJREF      11      0         0         2
1         2         0         0         0
0         0         0         0
}
```

Output of SIZE() for an array of HASHes:
{ OBJREF 11 0 0 2
1 2 0 0 0
0 0 0 0
}

Output of ISA(/ARRAY) for scalar hash:
1

Output of ISA(/ARRAY) for array of hashes:
1

HELP for scalar hash:
A HASH <ID=1 NELEMENTS=2>

HELP for array of hashes:
C OBJREF = Array[2]

TYPENAME for scalar hash:HASH

TYPENAME for array of hashes:OBJREF

Subject: Re: HASH makes too many temporaries
Posted by chris_torrence@NOSPAM on Tue, 19 Mar 2013 15:29:36 GMT
View Forum Message <> Reply to Message

On Monday, March 18, 2013 9:30:24 PM UTC-6, bobnn...@gmail.com wrote:
> I was really hoping that HASHes would be a nice replacement for structures. I'm quite adept at working with structures but it gets old rebuilding them all the time just to change the size of an array (or dealing with the ugly and error prone PTRs in structure tags). So HASHes seem like a nice change. But there are severe problems with them. I often work with structures with relatively small number of tags (say 10-25) but with some of these tags have large arrays. With HASHes this does not work well since they tend to force you to create lots of temporaries. Here is a very basic example that illustrates the problem:
>
>
>
> IDL> h = hash()
>
> IDL> h['arr'] = fltarr(100000000)
>
> IDL> help, h
>
> H HASH <ID=1 NELEMENTS=1>
>
>
>

> I've created a HASH and have a large array in it. Later I want to see the details of whats in the array (since the default help is not helpful for this, unlike a structure help):
>
>
>
> IDL> help, h['arr']
>
> <Expression> FLOAT = Array[100000000]
>
>
>
> Notice by the delay that this simple command caused a temporary of the large array to be formed. If you have more memory than I do on my laptop then make the array larger and you will notice it. This is crazy. The syntax h['arr'] should produce a reference to the array and NOT a new version of the array (this would allow it to be used on the left side of an = as well). If you want to do the simple call above without producing a temporary you have to use:
>
>
>
> IDL> arr = h.remove('arr')
>
> IDL> help, arr
>
> ARR FLOAT = Array[100000000]
>
> IDL> h['arr'] = temporary(arr)
>
>
>
> Arrgg! I think that HASHes and LISTs were not incorporated into IDL with sufficient thought.

Hi Bob,

There are a couple of points I would like to make:

1. If you use a structure, you run into the same problem with temporary copies:
s = {field: fltarr(100000000)}
void = memory() ; clear the highwater
help, /mem
help, s.field
help, /mem

IDL prints:
heap memory used: 401720825, max: 401720938, gets: 142642, frees: 141500
<Expression> FLOAT = Array[100000000]
heap memory used: 401720740, max: 801720947, gets: 142654, frees: 141512

2. Now, with structures, you can store into elements without making a copy. In the example

above, you can do:
s.field[5] = 3

With hashes, using multiple array indices, you can do the same thing:
h = hash('arr', fltarr(100000000))
h['arr', 5] = 3
print, h['arr', 5]

IDL prints:
      3.00000

Note that this doesn't make an extra copy, or use any extra memory. This behavior is documented in the HASH function docs, under the section "Access and Change Array Elements within a Hash". It also works for lists, and hashes within hashes, arrays with multiple dimensions, etc.

Perhaps a missing piece is structures within hashes? As pointed out in the other thread, there is currently no way to modify structure fields for a structure within a hash.

Anyway, hope this helps.
Cheers,
Chris
ExelisVIS

Subject: Re: HASH makes too many temporaries
Posted by chris_torrence@NOSPAM on Tue, 19 Mar 2013 15:36:23 GMT
View Forum Message <> Reply to Message

Hi superchromix,

Well, both LIST and HASH were really intended to be "scalar" items. The fact that they are implemented as objects, and that you can have arrays of objects, is just a loophole.

Since you can store arbitrary data inside of a list or hash, and you can have nested lists & hashes, theoretically, you shouldn't need to have an array of them.

But, if you really want to have an array of lists or hashes, then I would recommend using TYPENAME to determine whether you have a LIST/HASH. If TYPENAME returns LIST or HASH, then you have a scalar list/hash. If TYPENAME returns OBJREF, then you *know* that you either have a non-list/non-hash scalar object, or you have an array of objects, each of which could be *any* class (not just list or hash).

Cheers,
Chris
ExelisVIS

Subject: Re: HASH makes too many temporaries
Posted by Bob[4] on Tue, 19 Mar 2013 18:26:01 GMT
View Forum Message <> Reply to Message

On Tuesday, March 19, 2013 9:29:36 AM UTC-6, Chris Torrence wrote:
> On Monday, March 18, 2013 9:30:24 PM UTC-6, bobnn...@gmail.com wrote:
>
>> I was really hoping that HASHes would be a nice replacement for structures. I'm quite adept at working with structures but it gets old rebuilding them all the time just to change the size of an array (or dealing with the ugly and error prone PTRs in structure tags). So HASHes seem like a nice change. But there are severe problems with them. I often work with structures with relatively small number of tags (say 10-25) but with some of these tags have large arrays. With HASHes this does not work well since they tend to force you to create lots of temporaries. Here is a very basic example that illustrates the problem:
>
>
>> IDL> h = hash()
>
>> IDL> h['arr'] = fltarr(100000000)
>
>> IDL> help, h
>
>> H HASH <ID=1 NELEMENTS=1>
>
>>
>
>> I've created a HASH and have a large array in it. Later I want to see the details of whats in the array (since the default help is not helpful for this, unlike a structure help):
>
>>
>
>> IDL> help, h['arr']
>
>> <Expression> FLOAT = Array[100000000]
>
>> Notice by the delay that this simple command caused a temporary of the large array to be formed. If you have more memory than I do on my laptop then make the array larger and you will notice it. This is crazy. The syntax h['arr'] should produce a reference to the array and NOT a new version of the array (this would allow it to be used on the left side of an = as well). If you want to do the simple call above without producing a temporary you have to use:
>
>> IDL> arr = h.remove('arr')
>
>> IDL> help, arr
>
>> ARR FLOAT = Array[100000000]
>
>> IDL> h['arr'] = temporary(arr)
>

>> Arrgg! I think that HASHes and LISTs were not incorporated into IDL with sufficient thought.
>
>
> Hi Bob,
>
> There are a couple of points I would like to make:
>
> 1. If you use a structure, you run into the same problem with temporary copies:
>
> s = {field: fltarr(100000000)}
>
> void = memory() ; clear the highwater
>
> help, /mem
>
> help, s.field
>
> help, /mem
>
> IDL prints:
>
> heap memory used: 401720825, max: 401720938, gets: 142642, frees: 141500
>
> <Expression> FLOAT = Array[100000000]
>
> heap memory used: 401720740, max: 801720947, gets: 142654, frees: 141512

Yes, but at least you can find out what is in the structure without all the extra overhead:

help, s, /STRUCT

One cannot find out what is in a HASH or LIST without the overhead (unless one resorts to the trick I show above). I've written a hash_help procedure which prints out what is in the HASH in the format that help uses on a structure. This is a very handy routine. It was in doing this that I realized the problem with the temporary. I'm surprised HASHes and LISTs do not come with better help output by default (does anyone really work them and never want to see what is in them?).

> 2. Now, with structures, you can store into elements without making a copy. In the example above, you can do:
>
> s.field[5] = 3
>
> With hashes, using multiple array indices, you can do the same thing:
>
> h = hash('arr', fltarr(100000000))
>
> h['arr', 5] = 3
>

> print, h['arr', 5]
> IDL prints:
>
>   3.00000
>
> Note that this doesn't make an extra copy, or use any extra memory. This behavior is documented in the HASH function docs, under the section "Access and Change Array Elements within a Hash". It also works for lists, and hashes within hashes, arrays with multiple dimensions, etc.
>

Yes, I know about this but I find this syntax ugly. To try to get around this I tried to create the "DICT" class you mentioned in the post:

https://groups.google.com/d/msg/comp.lang.idl-pvwave/OmAS-0C 7iOU/6S0cJW3B_1UJ

I was able to overload the "." (getproperty and setproperty) but when I tried to overload bracket and use it along with getproperty then IDL throws an error. Thus, I would have to follow the same ugly method as above. Is it possible this restriction (using both {get,set}property with applyBrackets{Left,Right}Side) will be lifted in a future version of IDL? Have you thought more about a built in implementation of your DICT class that could get around these restrictions?

As I mentioned in the "!null" thread, I think IDL needs a reference type that would be similar to a PTR but would not need to be de-referenced to get at what it is pointing at. It could use de-referencing (or a function call) to set the reference but then would allow syntax like a normal variable. This would greatly simplify IDL programing and could perhaps be used to provide a way to access inside a HASH or LIST by having a function call return the reference to the item.

> Perhaps a missing piece is structures within hashes? As pointed out in the other thread, there is currently no way to modify structure fields for a structure within a hash.
>

Yes, not having this work for structures is a big problem.

Bob