Subject: float function unexpectedly slow
Posted by timothyja123 on Wed, 12 Mar 2014 23:56:44 GMT
View Forum Message <> Reply to Message

Hi guys,

I've spent the last day or so squeezing every last big of performance out of one of my code paths. I've speant a large amount of time staring at IDL's profiler (with is great by the way) and I'm starting to get to a point were most time is speant inside IDL's own procedures which means there isn't much more room for me to tweak anything further.

Anyway to get to the point one thing I have discovered this morning is that the built in float() function seems to be unexpectedly slow.
For example my profiler shows the following:

float() calls (mostly stright string to float conversions)
86,640 = 117.13ms

at first glance this seems acceptable however when I then comapre this to strmatch calls things start to look like there is room for improvement in the float() function

strmatch() calls
85,215 = 34.46ms

So strmatch() is around 4x faster on average in my use cases.
Ofcourse the speed of strmatch() is dependend on the complexity of the regular expresion and the length of the string its searching but one would still assume it would always be slower than a float() call. Is this a reasonable assumption?
Is there any reason float() would be slower than something like strmatch()?

I'd like to get some opinions before I consider sending a support request about this.

Thanks,
Tim

Extra Notes:

I did a comparison between the IDL and Python function to see if my assumptions are resonable.

Results
Python - 86,000 calls = 31ms
IDL - 86,000 calls = 62ms (about 2x faster than what I'm seeing in my real program but still 1/2 the speed of Python)

Code used for comparison

Python

```
from datetime import datetime

tstart = datetime.now()

range_count = range(0, 86000)
float_str = '363.491'

for x in range_count:
 float_val = float(float_str)

tend = datetime.now()
print(tend - tstart)
```

IDL

```
-------
pro float_speed_test

  startTime = systime(/seconds)

  float_str = '363.491'

  for i=0, 86000 do float_val = float(float_str)

  finishTime = systime(/seconds)
  timeSpent = finishTime - startTime
  print, 'Time: ', timeSpent

end
```

Subject: Re: float function unexpectedly slow
Posted by Craig Markwardt on Thu, 13 Mar 2014 00:25:46 GMT
View Forum Message <> Reply to Message

On Wednesday, March 12, 2014 7:56:44 PM UTC-4, timoth...@gmail.com wrote:
> Anyway to get to the point one thing I have discovered this morning is that the built in float() function seems to be unexpectedly slow.

When I try your sample code on my iMac and Linux machine (both x86_64), both Python and IDL are about the same speed. In fact for me IDL is 3-10% faster!

Craig

Subject: Re: float function unexpectedly slow
Posted by Phillip Bitzer on Thu, 13 Mar 2014 00:29:12 GMT
View Forum Message <> Reply to Message

Hmm....I don't see quite the disparity you do. Using your sample routines, I get:

IDL->about 40 msec
Python->about 37 msec

(These are "mental" means from running each routine a few times.)

IDL/Python versions:
----------Python
import sys

sys.version
Out[4]: '2.7.6 |Anaconda 1.8.0 (x86_64)| (default, Jan 10 2014, 11:23:15) \n[GCC 4.0.1 (Apple Inc. build 5493)]'

----------IDL

IDL> help, !VERSION
** Structure !VERSION, 8 tags, length=104, data length=100:
   ARCH   STRING   'x86_64'
   OS   STRING   'darwin'
   OS_FAMILY   STRING   'unix'
   OS_NAME   STRING   'Mac OS X'
   RELEASE   STRING   '8.3'
   BUILD_DATE   STRING   'Nov 15 2013'
   MEMORY_BITS   INT   64
   FILE_OFFSET_BITS
                   INT   64

Subject: Re: float function unexpectedly slow
Posted by timothyja123 on Thu, 13 Mar 2014 00:31:19 GMT
View Forum Message <> Reply to Message

On Thursday, March 13, 2014 11:25:46 AM UTC+11, Craig Markwardt wrote:
> On Wednesday, March 12, 2014 7:56:44 PM UTC-4, timoth...@gmail.com wrote:
>
>> Anyway to get to the point one thing I have discovered this morning is that the built in float() function seems to be unexpectedly slow.
>
>
>
> When I try your sample code on my iMac and Linux machine (both x86_64), both Python and IDL are about the same speed. In fact for me IDL is 3-10% faster!
>
>
>
> Craig

Hmm interesting. Just out of curiosity what kind of CPU do you have and how many cores does it have?

The IDL documentation says: This routine is written to make use of IDL's thread pool, which can increase execution speed on systems with multiple CPUs.

I have 6 cores maybe its costing more time than its saving with that many cores.

The other question would be what version of python and idl are you using? I'm using IDL 8.2 and Python 3.2.2 on Windows 64-bit

---

Subject: Re: float function unexpectedly slow
Posted by timothyja123 on Thu, 13 Mar 2014 00:38:26 GMT
View Forum Message <> Reply to Message

Ok I've rerun is again in IDL with a fresh workspace and I'm seeing 48-50ms which is slightly closer to Python.

I guess my real challenge is reproducing the times I see in my application.

Thanks for the feedback by the way.

Subject: Re: float function unexpectedly slow
Posted by Craig Markwardt on Thu, 13 Mar 2014 00:45:19 GMT
View Forum Message <> Reply to Message

On Wednesday, March 12, 2014 8:31:19 PM UTC-4, timoth...@gmail.com wrote:
> On Thursday, March 13, 2014 11:25:46 AM UTC+11, Craig Markwardt wrote:
>
>> On Wednesday, March 12, 2014 7:56:44 PM UTC-4, timoth...@gmail.com wrote:
>
>>
>
>>> Anyway to get to the point one thing I have discovered this morning is that the built in float() function seems to be unexpectedly slow.
>
>>
>
>>
>
>>
>
>>
>
>> When I try your sample code on my iMac and Linux machine (both x86_64), both Python and IDL are about the same speed. In fact for me IDL is 3-10% faster!

>
>>
>
>>
>
>>
>
>> Craig
>
>
>
> Hmm interesting. Just out of curiosity what kind of CPU do you have and how many cores does it have?
>
>
>
> The IDL documentation says: This routine is written to make use of IDL's thread pool, which can increase execution speed on systems with multiple CPUs.
>
>
>
> I have 6 cores maybe its costing more time than its saving with that many cores.
>
>
>
> The other question would be what version of python and idl are you using? I'm using IDL 8.2 and Python 3.2.2 on Windows 64-bit

Mac:
IDL> print, !version, !cpu
{ x86_64 darwin unix Mac OS X 7.1 Apr 21 2009 64 64}{ 0
0 2 2 100000 0
}

Linux:
IDL> print, !cpu, !version
{ 0 0 4 4 100000
0}{ x86_64 linux unix linux 8.1 Mar 9 2011 64 64
}

Subject: Re: float function unexpectedly slow
Posted by chris_torrence@NOSPAM on Thu, 13 Mar 2014 03:50:37 GMT
View Forum Message <> Reply to Message

Hi Tim,

I'm seeing the same thing that Craig sees - on my older MacBook Pro, running IDL 8.3, IDL is about 15% faster than python 2.7.6.

Speaking as someone who knows the guts of the float function, I wouldn't bother filing a bug report. The float routine (and all the other conversion routines) are just about as fast as they are going to get. This is all "Dave Stern" code, which means it is dense but super efficient. There is just a fair amount of code involved in checking the input arguments, parsing the string & looking for decimal points, exponents, etc., and then finally creating the IDL_VARIABLE and filling in the value.

I think your time would be better spent in eliminating any loops in your programs. IDL's thread pool doesn't even begin to work until you have more than 100,000 elements in your input array, so you won't see any significant boost with multiple cores until you can pass in all of your values at once.

Hope this helps.

Cheers,
Chris
ExelisVIS

Subject: Re: float function unexpectedly slow
Posted by Heinz Stege on Mon, 31 Mar 2014 23:48:45 GMT
View Forum Message <> Reply to Message

Hi all,

I know, I'm a little late with this message. When I read Tim's posting, I had the idea to write my own system routine for converting a string to float. And this idea I didn't get out of my mind. So here is the story.

I'm not an expert in C. To tell the truth, I'm just a beginner. There may be things in my C-code, which are not good. (Comments, which help me learning, are welcome.) I'll place the code at the bottom of this posting.

First let me give the results. str_to_double(), this is the new system routine, makes about 50% more conversions per time than IDL's double().

The following commands
```
  str='363.491'
  t0=systime(1) &for i=0,9999999 do x=double(str) &print,systime(1)-t0
```
take 5.90 s. The same with x=str_to_double(str) instead of x=double(str) takes 3.81 s. I ran the commands several times in a changing order.

I think, Chris is absolutely right with his recommendation to
eliminate any loops in the IDL program. So I did another test with an
array of strings:
  str=strtrim(randomu(seed,1000000)*1000.,2)
  t0=systime(1) &for i=0,9 do x=double(str) &print,systime(1)-t0
This takes 5.35 s for double() and 3.60 s for str_to_double. This is
very similar to the result with the scalar string.

Interesting, that in this test the loop does not cost much CPU time.
The conversion of 1 million scalar strings do not need much more time
than one array with 1 milionen elements. However str_to_double() is
significantly faster than double().

The tests are done with version { x86 Win32 Windows Microsoft Windows
8.0.1 Oct 5 2010   32   64}.

Cheers, Heinz

Here ist the C code:

```
static IDL_VPTR str_to_double(int argc,IDL_VPTR *argv)
{
   IDL_VPTR result;

   if (argv[0]->type != IDL_TYP_STRING) {

 IDL_MessageFromBlock(msg_block,M_MSR_NO_STRING_TYPE,IDL_MSG_
LONGJMP,argv[0]- >type);
   }

   if (argv[0]->flags & IDL_V_ARR) {
      IDL_STRING *str;
      double *vector;
      IDL_MEMINT i;

      str=(IDL_STRING *) argv[0]->value.arr->data;
      vector=(double *) IDL_MakeTempArray(IDL_TYP_DOUBLE,
                                  argv[0]->value.arr->n_dim,

argv[0]->value.arr->dim,IDL_ARR_INI_NOP,&result);
      for (i=0; i<argv[0]->value.arr->n_elts; i++) {
         vector[i]=str[i].slen? atof(str[i].s) : 0.;
      }
   }
   else {
      double value;
```

```
    value=argv[0]->value.str.slen? atof(argv[0]->value.str.s) : 0.;
    result=IDL_GettmpDouble(value);
  }

  return result;
}
```