
Subject: Reading columns of binary data

Posted by news.verizon.net on Mon, 07 Aug 2006 17:12:16 GMT

[View Forum Message](#) <> [Reply to Message](#)

This is probably more of a feature request than a question, though there is a chance the desired feature already exists within IDL.

A FITS binary table might plausibly consist of 500 columns and 500,000 rows of data in a fixed length binary format. To read the 32nd column there are 2 options:

(1) Loop over the 500,000 rows, extracting the scalar value for the 32nd column for each row, and construct the 500,000 element output array

(2) Read the entire 500,000 x 500 file into memory, and extract the 32nd column

(In practice, one probably would use a hybrid method of looping over an intermediate size buffer. Also note that an identical problem occurs when extracting every nth pixel from an extremely large image on disk.)

I understand that the extraction of a column will never be as fast as reading a row of data, because the bytes to be read are not contiguous.

But I am hoping that the heavy work can be done at a lower level than the IDL syntax.

Erin Sheldon has recently written a C routine `BINARY_READ` linked to IDL via a DLM to efficiently read a binary column (

http://cheops1.uchicago.edu/idlhelp/sdssidl/umich_idl.html#C_CODE).

(He also has routines `ASCII_READ` and `ASCII_WRITE` to do this for the less urgent problem of ASCII columns.) While I might adopt this routine, it would be nice for portability reasons if a DLM were not necessary. Say, a new keyword `SKIP` to `READU`

```
IDL> a = fltarr(200)
```

```
IDL> readu, 1, a, skip = 100
```

to indicate to skip 100 bytes before reading consecutive elements.

It appears that MATLAB already has a function `FREAD` to support reading columns of data.

--Wayne

Subject: Re: Reading columns of binary data

Posted by [JD Smith](#) on Tue, 08 Aug 2006 17:44:38 GMT

On Tue, 08 Aug 2006 16:57:41 +0200, Földy Lajos wrote:

```
>
> On Mon, 7 Aug 2006, JD Smith wrote:
>
>> I'd regard this as yet another example of why IDL needs a fast
>> compiled "side-loop" primitive for generic looping operations which
>> don't need the full conveniences of the interpreter loop (ability to
>> hit Control-C to interrupt, etc.). I should be able to say:
>>
>> a=fltarr(5000000,/NO_ZERO)
>> f=0.0
>> for_noblock i=0L,5000000L-1L do begin
>>   point_lun,un,i*100L
>>   readu,un,f
>>   a[i]=f
>> endfor
>>
>> and not have it be 100x slower than the equivalent in C.
>>
>> JD
>>
>
> Hi,
>
> I mentioned assoc yesterday, so let's try again:
>
> arr=assoc(un, [0.])
> for i=0L,5000000L-1L do a[i]=arr[25*i]
>
> it will do the point_lun/readu in one step, so it should be a little bit
> faster. But I think a faster loop here would not help much, as disk I/O
> is the limiting factor.
```

If that were the case then the IDL loop and a similar C loop (offsetting the file pointer and reading 4 bytes) should perform similarly. I haven't done the test (and would be happy to be proven wrong), but my guess is they wouldn't match by 1-2 orders of magnitude.

JD

Subject: Re: Reading columns of binary data
Posted by [Foldy Lajos](#) on Tue, 08 Aug 2006 18:35:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, 8 Aug 2006, JD Smith wrote:

> On Tue, 08 Aug 2006 16:57:41 +0200, FÖLDY Lajos wrote:

>

>>

>> On Mon, 7 Aug 2006, JD Smith wrote:

>>

>>> I'd regard this as yet another example of why IDL needs a fast
>>> compiled "side-loop" primitive for generic looping operations which
>>> don't need the full conveniences of the interpreter loop (ability to
>>> hit Control-C to interrupt, etc.). I should be able to say:

>>>

>>> a=fltarr(500000,/NO_ZERO)

>>> f=0.0

>>> for_noblock i=0L,500000L-1L do begin

>>> point_lun,un,i*100L

>>> readu,un,f

>>> a[i]=f

>>> endfor

>>>

>>> and not have it be 100x slower than the equivalent in C.

>>>

>>> JD

>>>

>>

>> Hi,

>>

>> I mentioned assoc yesterday, so let's try again:

>>

>> arr=assoc(un, [0.])

>> for i=0L,500000L-1L do a[i]=arr[25L*i]

>>

>> it will do the point_lun/readu in one step, so it should be a little bit
>> faster. But I think a faster loop here would not help much, as disk I/O
>> is the limiting factor.

>

> If that were the case then the IDL loop and a similar C loop (offsetting
> the file pointer and reading 4 bytes) should perform similarly. I haven't
> done the test (and would be happy to be proven wrong), but my guess is
> they wouldn't match by 1-2 orders of magnitude.

>

> JD

>

>

OK, I have written my homework :-)

IDL:

```

n=100000000l
openw, unit, 'test.dat', /get_lun
writeu, unit, findgen(n)
close, unit

```

```

k=n/100l
a=fltarr(k)
b=a

```

```

openr, unit, 'test.dat', /get_lun
f=0.
t=systime(1)
for j=0l,k-1l do begin
    point_lun, unit, j*400l
    readu, unit, f
    a[j]=f
endfor
print, 'point/read: ', systime(1)-t
close, unit

```

```

openr, unit, 'test.dat', /get_lun
arr=assoc(unit, [0.])
t=systime(1)
for j=0l,k-1l do b[j]=arr[100l*j]
print, 'assoc: ', systime(1)-t
close, unit

```

```

end

```

C (without any error checking, gcc-4.1.1 -O3, linux 32 bit):

```

#include <stdio.h>
#include <time.h>
#include <sys/time.h>

```

```

int main()
{
    struct timeval tval;
    struct timezone tzone;
    double t1, t2;
    int j;
    float* fp;

    fp=(float*)malloc(1000000l*sizeof(float));

```

```

FILE* file=fopen("test.dat", "r");

gettimeofday(&tval, &tzone);
t1=tval.tv_sec+1.0e-6*tval.tv_usec;

for (j=0; j<1000000; j++)
{ fseek(file, j*400, SEEK_SET);
  fread(fp+j, sizeof(float), 1, file);
}

gettimeofday(&tval, &tzone);
t2=tval.tv_sec+1.0e-6*tval.tv_usec;

printf("time: %f\n", t2-t1);
}

```

Best times:

```

IDL: point/read:  3.68
IDL: assoc:      3.05
C:              1.39

```

YMMV.

regards,
lajos

Subject: Re: Reading columns of binary data
 Posted by [JD Smith](#) on Tue, 08 Aug 2006 20:15:40 GMT
[View Forum Message](#) <> [Reply to Message](#)

```

>
> Best times:
>
> IDL: point/read:  3.68
> IDL: assoc:      3.05
> C:              1.39
>

```

Better than I thought, for sure... Wayne, maybe you don't need a DLM after all. One minor quibble: you include memory allocation in the times for C, but not the IDL versions. Should not change things (esp. if you use NO_ZERO).

Thanks for your homework ;).

