Subject: Re: problems plotting LARGE amounts of 2D data?
Posted by thompson on Tue, 22 Feb 1994 17:22:48 GMT
View Forum Message <> Reply to Message

mcheng@dunlop.cs.wisc.edu (Michael Cheng) writes:

> Hi

> I am trying to find ways to plot LARGE amounts of 2D data,
> and I like to know what is currently the state of the art. For the
> sake of this posting, let's say "large" means much more data than
> fits in real memory. From my own experience, using the virtual memory
> of the workstation to store large amounts of data impedes performance
> due to excessive paging. Here is what I have been able to gather so far:

> 1) AVS: has 25,000 point limit. Everything is loaded into virtual memory.

> 2) Khoros 1: loads everything into virtual memory.
> Any updates from Khoros 2.0?

> 3)idl/pvwave: As far as I can tell from the short demo,
> loads everything into virtual memory?

> I'm posting this to the various Comp.graphics.*
> groups, hoping that I can get feedback from users of various software
> packages. I would appreciate comments on the current/future capabilities of
> the above packages. I would also like comments about other
> packages, such as SGI Explorer, IBM Data Explore, apE, or any other
> package. Thanks in advance.

> Mike

I think the way you're looking at this is all wrong. It's not AVS/Khoros/IDL which decide whether or not data are in real or virtual memory--it's the operating system. Operating systems such as Unix or VMS shield the program from knowing how much of the memory they're using is real or virtual. Generally speaking, you'll get as much of the real memory as possible and only the overflow will be stored in virtual memory. Any programs which are designed to run on a virtual memory operating system should behave the same way.

If a program such as IDL (or any of the others you mention) is putting things into virtual memory, it can only be because you don't have enough real memory available to you. The only thing you can do is buy more memory, or if your operating system supports memory usage quotas (such as VMS) then you need to increase your quotas.

Bill Thompson

Subject: Re: problems plotting LARGE amounts of 2D data?
Posted by mcheng on Tue, 22 Feb 1994 22:51:12 GMT
View Forum Message <> Reply to Message

In article <thompson.761937768@serts.gsfc.nasa.gov> thompson@serts.gsfc.nasa.gov (William Thompson) writes:

> If a program such as IDL (or any of the others you mention) is putting things
> into virtual memory, it can only be because you don't have enough real memory
> available to you. The only thing you can do is buy more memory, or if your
> operating system supports memory usage quotas (such as VMS) then you need to
> increase your quotas.

I argue that buying more memory is not always the best solution for the
following reasons:
1) Some of us are poor. (arguably a weak reason)
2) Some data are always larger than the largest amount of memory
reasonable amounts of money can buy.
3) Even a few medium sized data can overload real memory quickly.
For example, working with five 25-meg data sets already requires 125 megs
of memory.

There has always been a mistmatch between virtual memory policy and
the need to handle large amounts of data. This mismatch has been demonstrated
time and again in database systems. I feel that this issue will come
up again with respect to scientific data sets.

So we go back to my original question: is there any software package
for plotting large amounts of 2D data that does better than loading
everything into virtual memory?

Mike

Subject: Re: problems plotting LARGE amounts of 2D data?
Posted by peter on Wed, 23 Feb 1994 02:36:38 GMT
View Forum Message <> Reply to Message

Michael Cheng (mcheng@dunlop.cs.wisc.edu) wrote:

: Hi

: I am trying to find ways to plot LARGE amounts of 2D data,
: and I like to know what is currently the state of the art. For the
: sake of this posting, let's say "large" means much more data than
: fits in real memory. From my own experience, using the virtual memory
: of the workstation to store large amounts of data impedes performance
: due to excessive paging. Here is what I have been able to gather so far:

: 3)idl/pvwave: As far as I can tell from the short demo,
: loads everything into virtual memory?

IDL and PV-Wave have a 'associated' variable type, which lets you
map an array onto a disk file, and then step through the disk file
one slice at a time. For example,

```
 openr, lun, 'myfile.dat', /get_lun
 data = assoc(lun,bytarr(1024,1024))
```

associated the variable name data with the contents of file myfile.dat.
Nothing has been read yet. Then the statement

```
 slice = data(10)
```

will load the 11th bytarr(1024,1024) contained in the file into working
memory. A statement like

```
 data(10) = byte(fft(data(10)))
```

will read, process, and replace the 11th array in the disk file.

So, if you can access data in a nice order, so that you don't have to
go out to disk all the time, you can work with enormous data sets.

Something like this should also work for plotting a 2D data set.

Hope this helps.

 - Peter

Subject: Re: problems plotting LARGE amounts of 2D data?
Posted by thompson on Wed, 23 Feb 1994 20:20:02 GMT
View Forum Message <> Reply to Message

mcheng@dunlop.cs.wisc.edu (Michael Cheng) writes:

> In article <thompson.761937768@serts.gsfc.nasa.gov> thompson@serts.gsfc.nasa.gov
(William Thompson) writes:

>> If a program such as IDL (or any of the others you mention) is putting things
>> into virtual memory, it can only be because you don't have enough real memory
>> available to you. The only thing you can do is buy more memory, or if your
>> operating system supports memory usage quotas (such as VMS) then you need to
>> increase your quotas.

> I argue that buying more memory is not always the best solution for the
> following reasons:
> 1) Some of us are poor. (arguably a weak reason)
> 2) Some data are always larger than the largest amount of memory
> reasonable amounts of money can buy.
> 3) Even a few medium sized data can overload real memory quickly.
> For example, working with five 25-meg data sets already requires 125 megs
> of memory.

> There has always been a mistmatch between virtual memory policy and
> the need to handle large amounts of data. This mismatch has been demonstrated
> time and again in database systems. I feel that this issue will come
> up again with respect to scientific data sets.

> So we go back to my original question: is there any software package
> for plotting large amounts of 2D data that does better than loading
> everything into virtual memory?

> Mike

Somebody else mailed me privately that he thought that you were probably talking about database management techniques, rather than memory management once data had already been read into arrays. I don't know how the other packages you mentioned work, but I can comment on how this applies to IDL.

IDL doesn't incorporate any kind of database management scheme, neither relational nor network nor object-oriented. It only reads in what you tell it to read in. Generally speaking you have to write an IDL routine to read your data files, although there is built-in support nowadays for certain kinds of commonly-used file formats such as FITS, HDF, CDF and netCDF. So, in that sense IDL does *not* read everything into memory. It reads in what you tell it, whether that's an entire file all at once or piece by piece, because you've written an old-fashioned program to do just what you wanted to do. You have complete control and complete responsibility. :^)

On the other hand, if you wanted to read in a bunch of data and extract global properties from the data, then it would be easier to read the data into a single large array. For example if you had a 2048x2048 image that you wanted to display, then you would have to read that in as a single large array. Or if you wanted to read in 1000 X,Y traces of 500 points each and calculate the average trace, then it would in general be quicker to read those traces into a couple of big arrays than to process each trace individually in a loop. That's because loops are rather expensive in IDL.

I've certainly done things in IDL where I've handled large amounts of data by reading in the data bit by bit. For example, at one point I had a whole series of images and I wanted to get the mean and standard deviation as a function of pixel position. I didn't have enough memory to read all the data at once, so I

read the images one-by-one and did a running calculation. Other situations is where I wanted to apply the same routine to a bunch of files. I just constructed a program with a FOR loop that read each file in turn and processed it separately. In general, though, it's more efficient to use as much memory as you can.

Bill Thompson

Subject: Re: problems plotting LARGE amounts of 2D data?
Posted by jworley on Thu, 24 Feb 1994 21:51:56 GMT
View Forum Message <> Reply to Message

Hi

> michael> I am trying to find ways to plot LARGE amounts of 2D data,
> michael> and I like to know what is currently the state of the
> michael> art. For the sake of this posting, let's say "large" means
> michael> much more data than fits in real memory. From my own
> michael> experience, using the virtual memory of the workstation to
> michael> store large amounts of data impedes performance due to
> michael> excessive paging. Here is what I have been able to gather
> michael> so far:
>
> michael> 2) Khoros 1: loads everything into virtual memory. Any
> michael> updates from Khoros 2.0?

We are actively trying to solve the large data set problem. It is a technically challenging problem because the problems posed by large data sets present different problems in different applications. For example, in an isolated application, the problem is pretty much limited to how to get around the virtual memory limitation efficiently. However, in other environments, such as Cantata (the visual programming language in Khoros), large data sets also pose problems because intermediate stages in a processing pipeline can quickly chew up any temp space that may be available.

The first issue is being addressed by data services. Data services is a data abstraction that provides read and write data in many file formats via an application programmers interface (API). This API provides a means of storing and retrieving data in units that are convenient to your application area. Data services is responsible for managing memory by caching only the portions of the data set that you are processing. Thus, only a reasonable portion of your data set is in memory at any given time. Data processing programs (including graphical applications) that are distributed with Khoros 2.0 will be written to data services. People who use Khoros as a development platform will be strongly encouraged to write their applications to

data services as well.

The second issue, that of multiple copies is a byproduct of the intermediate stages of the data flow program. Each operator in the data flow program is written to read in an input set of data, perform some processing on the data, and then write out an output set of data. So, the problem of multiple copies is not really related to the data flow program, but rather related to the data flow operators. Ideally, if the data could be passed between operators in some serial fashion (such as streams or sockets), intermediate copies of the data would not be needed. Unfortunately, data flow operators often can not be written to accept serial input and produce serial output, but rather require the ability to access data non-sequentially (an N-dimensional FFT is an example). These operators require the ability to randomly access the data input and output. As you have pointed out, this presents a significant problem when operating on large data sets.

Our approach to addressing this problem is to provide functionality for automatically buffering streamed data so that it can be accessed in a non-serial fashion. By addressing this problem in the infrastructure, we can guarantee that only a minimal number of temporary copies are present at any time (typically this is two copies per data pathway).

There are probably some direct ways of "getting around" this problem. Forcing everyone to write stream-processing routines is one approach. However we don't see this as a reasonable solution since many algorithms don't lend themselves to this type of interaction. Our goal is to abstract low-level issues away from the users who will be creating their own modules so that they can focus on the problem of implementing their algorithms without having to worry about working around limitations in their hardware and operating system. In the context of large data sets and data flow environments such as Cantata, the cost of doing this is increased overhead in terms of both temporary storage and performance. The objective is to minimize these costs while also minimizing the complexity of the system.

Hope we have been helpful.

Jeremy Worley, Steve Kubica, and the Khoros Group

Jeremy Worley jworley@khoros.unm.edu
The Khoros Group (505)837-6500
--

Jeremy Worley jworley@khoros.unm.edu
The Khoros Group (505)837-6500