Subject: Re: IDL and Beowolf ?
Posted by George N. White III on Wed, 25 Jul 2001 13:34:45 GMT
View Forum Message <> Reply to Message

On Mon, 23 Jul 2001, Steve Smith<steven_smith> wrote:

> On Mon, 23 Jul 2001 16:08:32 GMT, dmarshall@ivory.trentu.ca
> <dmarshall@ivory.trentu.ca> wrote:
>> I just finished reading The Do-It-Yourself Supercomputer by William W.
>> Hargrove, Forrest M. Hoffman and Thomas Sterling in this months Scientific
>> American
>> http://www.sciam.com/2001/0801issue/0801hargrove.html
>> And wondered if anyone had any thoughts/experience with running IDL
>> applications on such a cluster.
>>
>> Dave.
>
> I inquired about this with RSI few years ago: the answer I got was that
> since IDL is an interpretive language, it could not be parallelized. As
> I'm sure you've learned by your reading, the key to using distributed
> computing is that your job must be cast in such a way that it can be
> parallelized. I'm not guru, but that is what I was told by RSI, FYI!

My experience has been that data sets are expanding in size even more
rapidly than CPU speed, so RSI needs to looking for approaches that help
with larger data sets. Data flow is probably more critical than having
lots of CPUS doing f.p. math.

IDL and Matlab face similar issues for parallelization, so the following
may provide some insight:

http://www.mathworks.com/company/newsletter/pdf/spr95cleve.p df "Why there
isn't a parallel matlab"

but then see:
http://www.rtexpress.com/ "We have a parallel matlab"

There are many different parallel system designs, so no single approach
will be effective for all machines. For linear algebra, the BLAS
provide a useful set of functions that can be provided via libraries
that have been written to support particular hardware.

IDL could certainly implement parallel versions of some key functions, but
it is difficult to know whether the non-trivial effort needed to make
this work would actually produce useful speedups for "real-world"
problems.

Parallel processing could be implemented for problems where the same

expensive calculation is performed on each element of a large array, but with no interaction between elements. Two very different approaches to parallelizing such problems are:

1. partition the array into "tiles", and process the tiles in parallel.
Just as there are libraries that support space-efficient manipulation of sparse arrays, one could have libraries to support parallel manipulation of tiled arrays on a Beowulf machine.

Tiled arrays have other benefits -- consider a time series of images and suppose you want to process data from a small ROI over time. There could be significant savings if you only need to load data for the tiles containing the ROI, rather than loading complete images.

2. loop unrolling, e.g.,

```
  for i=0,n do B[i]=expensive_function(A[i])
```

becomes:

```
  for i=0,n,k do begin
    B[i]=expensive_function(A[i])
    B[i+1]=expensive_function(A[i+1])
    ...
    B[i+k-1]=expensive_function(A[i+k-1])
  end
```

where the "expensive_function" calls are run in parallel. Depending on just how expensive the functions are, this may require tighter coupling of processors, more like large SGI machines than a Beowulf, to get useful speedups.

--
George N. White III <gnw3@acm.org> Bedford Institute of Oceanography