Subject: Re: C++ in IDL
Posted by JD Smith on Wed, 01 Jun 2005 17:00:34 GMT
View Forum Message <> Reply to Message

On Wed, 01 Jun 2005 10:10:34 -0600, David Fanning wrote:

> Michael Wallace writes:
>
>> There's that much of a difference? I thought sure that IDL would have
>> been designed for number crunching considering the market they are
>> trying to serve. Of course, if the IDL code wasn't optimized...
>
> I've seen this kind of comparison many times. On further
> investigation the IDL code is usually, uh, not well written. :-)
> I would take any comparisons not performed by J.D. with a
> grain of salt. :-)

I've been known to push on IDL to get as much performance as possible, and still, for many types of problems, a direct C-coded approach, compiled with optimization, will out-perform the best-optimized IDL version by a factor of 10 or more. Here's a good example from your site in which we collected and refined the best-optimized IDL algorithms from the experts (Craig, Wayne, etc.), and still I found a simple compiled C approach is 20x faster:

http://www.dfanning.com/code_tips/drizzling.html

IDL truly excels at basic large array operations (adding 1 to 1 million integers, etc.); for these types of problems you'll find comparable performance to compiled code (ballpark anyway). For more complex algorithms, you can make huge gains in performance with careful vectorization and other methods in IDL, but, in the end, may still (but not always) miss compiled performance by a non-negligible margin.

This is not actually surprising. It's part of the deal you make when you code in IDL at a high-level, vs., e.g., C or FORTRAN at a low level. Did you think you would get all of those many conveniences which IDL provides, like loose typing, dynamic variable content, optional and keyword arguments, variable type checking, etc., for free? A much fairer comparison would be between, e.g., Matlab's array-crunching performance and IDL's. Among the so-called 4G languages (including scripting extensions like Perl's PDL, etc.), IDL delivers among the best performance for basic array operations.

I might suggest prototyping everything in IDL, profiling to find the algorithm or sub-algorithm which is limiting performance, and then re-coding that in C as a DLM. Sometimes, as in the case referenced

above, the C version can be quite simple and compact, especially if all of the setup, display, analysis and data juggling is handled in IDL. Simple DLMs are actually not difficult to produce, and it's worth your while to learn how to make use of them. Their one real drawback is cross-platform compatibility, but with MAKE_DLL and auto glue, you can actually recover this to some degree.

JD
