

ARE YOU READY FOR MULTI-MULTI-CORE SYSTEMS?

By Rob Meyer, CEO, Numerical Algorithms Group

It's not news to most in the developer community that "the business model for HPC-specific applications software has all but evaporated in the last decade". The same U.S. Government-sponsored survey¹ by the Council on Competitiveness that made this statement went on to report, "There is a lack of readiness for Petascale Systems. Three-quarters (74%) of the ISV applications are 'legacy applications' that are more than five years old, and seven out of eight (87%) are at least three years old. Fewer than half (46%) of the ISV applications scale even to hundreds of processors today, and 40% of the applications have no immediate plans to scale to this level. Very few codes scale to thousands of processors today or are being aimed at this level of scalability. If current development timeframes continue, the majority of ISV codes will not be able to take full advantage of petascale systems until three to five years after they are introduced."

The exact lag time from where the HPC market is today to where it needs to go is unknown. But while crystal balls that clarify the rate at which market demand for HPC products will mature are admittedly in short supply, the hardware trends that will smooth the transition to a mature HPC market are easy to see. The 2006 version of technical computing "reality" is an inexpensive dual-core processor from AMD or Intel on a desktop system or a dual or quad core RISC processor from Sun or IBM running on a server. In 2007 we should expect to see inexpensive quad-core processors from AMD and Intel and 8 or more cores per processor in 2008. These small SMP systems will be a far cry from the proprietary \$500,000+ SMP systems of a few years ago. What does this mean to the dynamics of HPC market maturity? The bottom line is that 4-8 core systems costing a few thousand dollars will be sitting on the desk of every technical software user in the relatively near future.

If your organization is hoping for a long life expectancy for the applications being developed today, it's time to take a step back and see if your software development organization is ready for the multi-multi-core systems becoming widely available *soon*.

The emergence of clusters with multi-core nodes offers several potential scenarios: in one, the system could treat each core on a node as a "virtual" processor with its own memory, potentially allowing a user's existing message passing-based code to run without modification. In another, a modified compiler could generate code such that a cluster consisting of multi-core processor nodes look like a "virtual monolithic SMP machine" to the application. This could be a real advantage for applications currently written for SMP architectures or serial applications that could be readily modified to call SMP components. The final, and most complex, scenario is one in which the user application is "aware" of a heterogeneous environment and incorporates code to utilize both message passing and SMP techniques to gain the maximum performance. Of course

¹ Council on Competitiveness "Study of ISVs Serving the High Performance Computing Market: Part A-Current Market Dynamics", July 2005

there are numerous other variants but these establish the point that there are few simple answers to gaining significant additional performance from existing code in a cluster environment with multi-core nodes.

However your organization chooses to meet the emerging demands for HPC-compatibility, these emerging multi-multi-core systems underline the importance of having a well-considered software migration strategy. New applications or updates to existing ones need to have portability and quality assurance factored into their design if they are to cope with coming hardware changes. At the Numerical Algorithms Group we have been creating, porting, and supporting numerical software for thirty-five years, and still consider our methods a work-in-progress. A cornerstone of our development processes has been to spare no effort to ensure that our components work as promised on each succeeding generation of hardware with the performance, robustness, and accuracy users expect. It is our expectation that applications will be ported multiple times to new chip architectures and operating system versions. And, we expect that even seemingly straightforward upgrades have potential to create errors in the ported application because of multiple changes in operating systems, compiler options, underlying math libraries, and so on.

With creating long-life applications that can withstand these hardware evolutions in mind, our development scheme unfolds as follows—

Our process begins with algorithm selection. Often, there are several methods available for solving the same mathematical or statistical problems we face but one may have advantages over another. For example, one method might incorporate its own error estimator. This feature would help the user in understanding results and would be very useful in assuring accuracy as the routine is ported to another platform. In addition, since speed and accuracy are often in competition with one another, it's important to choose an approach that will maintain sufficient accuracy over a broad range of data as the code is ported. Our internal shorthand for keeping this balance in perspective is the question "*Just how fast do you want the wrong answer?*"

Another aspect of our process is concerned with language features. In coding, we maintain strict standards on coding techniques and shy away from non-standard language extensions. While the use of certain language features or coding techniques may save some code or improve performance, the advantage is frequently negated when the code needs to be ported.

At each step of the process we emphasize design for portability and carry out expert peer review. Of particular importance for our numerical code is the creation of stringent test code for each method that exercises all areas and determines conditions under which the code becomes unstable or produces inaccurate results. This code is generally larger than the code being tested. The results from this stringent test code help us to ensure that the code behaves as documented and helps us select compiler options such that the code does not lose an unacceptable amount of accuracy in porting.

Complementing the stringent test code above are example test programs to exercise the code once installed with data and results from a verified installation. Finally, there is documentation – lots of documentation, including both for installation and for routine usage. Documentation, as much as functionality, helps insure a long life for software.

In summary, our approach to creating accurate, robust and portable code is a comprehensive process – careful selection of methods, strict coding standards for portability, expert peer review, stringent test programs, example programs, validation of compiler options and thorough documentation.

This might sound exhausting at first and not all of it may apply to your application. But if you consider the real time horizons for your applications and the likelihood that your application will need to be ported repeatedly, it is well worth taking comparable steps or using code—whether open source, commercial or internally developed- -that has similarly been developed with any eye towards outliving the current hardware. This is especially the case if your organization expects to use third party HPC code components to revamp applications for the next generation of multi-core systems.

Rob Meyer is CEO of the Numerical Algorithms Group (www.nag.com), an organization dedicated to making cross-platform mathematical, statistical and data mining components and tools for developers as well as 3D visualization application development environments. NAG operates worldwide with offices in Chicago (USA), Oxford (UK), and Tokyo (Japan). Today it serves over 10,000 sites and has created a worldwide collaborative network of the world's best mathematical experts. Questions can be forwarded to Rmeyer@nag.com.

Formatted: Font: Italic

Formatted: Font: Times New Roman, Italic

Source: Ubiquity -- Volume 7, Issue 36
(September 19, 2006 - September 25, 2006)
<http://www.acm.org/ubiquity/>