

The Impact of Multicore on Math Software

Alfredo Buttari¹, Jack Dongarra^{1,2}, Jakub Kurzak¹, and Piotr Luszczek¹

¹ Innovative Computing Laboratory, University of Tennessee Knoxville, TN

² Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge National Laboratory, TN

It is difficult to estimate the magnitude of the discontinuity that the high performance computing (HPC) community is about to experience because of the emergence of the next generation of multi-core and heterogeneous processor designs [4]. For at least two decades, HPC programmers have taken for granted that each successive generation of microprocessors would, either immediately or after minor adjustments, make their old software run substantially faster. But three main factors are converging to bring this “free ride” to an end. First, system builders have encountered intractable physical barriers – too much heat, too much power consumption, and too much leaking voltage – to further increases in clock speeds. Second, physical limits on the number and bandwidth of pins on a single chip means that the gap between processor performance and memory performance, which was already bad, will get increasingly worse. Finally, the design trade-offs being made to address the previous two factors will render commodity processors, absent any further augmentation, inadequate for the purposes of tera- and peta-scale systems for advanced applications. This daunting combination of obstacles has forced the designers of new multi-core and hybrid systems, searching for more computing power, to explore architectures that software built on the old model are unable to effectively exploit without radical modification [5].

But despite the rapidly approaching obsolescence of familiar programming paradigms, there is currently no well understood alternative in whose viability the community can be confident. The essence of the problem is the dramatic increase in complexity that software developers will have to confront. Dual-core machines are already common, and the number of cores is expected to roughly double with each processor generation. But contrary to the assumptions of the old model, programmers will not be able to consider these cores independently (i.e. multi-core is *not* “the new SMP”) because they share on-chip resources in ways that separate processors do not. This situation is made even more complicated by the other non-standard components that future architectures are expected to deploy, including mixing different types of cores, hardware accelerators, and memory systems. Finally, the proliferation of widely divergent design ideas shows that the question of how to best combine all these new resources and components is largely unsettled. When combined, these changes produce a picture of a future in which programmers must overcome software design problems that are vastly more complex and challenging than in the past in order to take advantage of the much higher degrees of concurrency and greater computing power that new architectures will offer.

The work that we currently pursue is the *initial phase* of a larger project in *Parallel Linear Algebra for Scalable Multi-Core Architectures (PLASMA)* that aims to address this critical and highly disruptive situation. While PLASMA's ultimate goal is to create software frameworks that enable programmers to simplify the process of developing applications that can achieve both high performance and portability across a range of new architectures, the current high levels of disorder and uncertainty in the field processor design make it premature to attack this goal directly. More experimentation is needed with these new designs in order to see how prior techniques can be made useful by recombination or creative application and to discover what novel approaches can be developed into making our programming models sufficiently flexible and adaptive for the new regime.

Preliminary work we have already done on available multi-core and heterogeneous systems, such as the IBM CELL processor, shows that techniques for increasing parallelism and exploiting heterogeneity can dramatically accelerate application performance on these types of systems. Other researchers have already begun to utilize these results. Under this early PLASMA project, we are leveraging our initial work in the following three-pronged research effort:

- *Experiment with techniques* – Building on the model of large grain data flow analysis, we are exploring techniques that exploit dynamic and adaptive out-of-order execution patterns on multi-core and heterogeneous systems. Early experiences with matrix factorization techniques have already led us to the idea of dynamic look-ahead, and our preliminary experiments show that this technique can yield great improvements in performance.
- *Develop prototypes* – We are testing the most promising techniques through highly optimized (though neither flexible nor portable and thus not general enough) implementations that we, and other researchers in the community, can use to study their limits and gain insight into potential problems. These prototypes are also enabling us to assess how well suited these approaches are to dynamic adaptation and automated tuning.
- *Provide a design draft for the PLASMA framework* – An initial design plan for PLASMA frameworks for multi-core and hybrid architectures is being developed and, in combination with PLASMA software prototypes, will be distributed for community feedback.

We used the forgoing analysis of the problems of LAPACK/ScaLAPACK on multi-core systems as the basis of some preliminary tests of techniques for doing fast and efficient LA on multi-core. LA operations are usually performed as a sequence of smaller tasks; it is possible to represent the execution flow of an operation as a Directed Acyclic Graph (DAG) where the nodes represent the sub-tasks and the edges represent the dependencies among them. Whatever the execution order of the sub-tasks is, the result will be correct as long as these dependencies are not violated. This concept has been used in the past to define “look-ahead” techniques that have been extensively applied to the LU factorization. Such methods can be used to remedy the problem of synchronizations introduced by non-parallelizable tasks by overlapping their execution with

the execution of more efficient ones [1]. Although the traditional technique of look-ahead usually provides only a static definition of the execution flow that is hardwired in the source code, the idea of out-of-order execution it embodies can be extended to broader range of cases, where the execution flow is determined at run time in a fully dynamic fashion. With this dynamic approach, the subtasks that contribute to the result of the operation can be scheduled dynamically depending on the availability of resources and on the constraints defined by the dependencies among them (i.e., edges in the DAG).

Our recent work shows how the one-sided factorizations, LU, QR and Cholesky can benefit from the application of this technique [2]. Block formulations of these three factorizations, as well as many other one-sided transformations, follow a common scheme. In a single step of each algorithm, first operations are applied to a single block of rows or columns, referred to as the panel, then the result is applied to the remaining portion of the matrix. The panel operations are usually implemented with Level 1 and 2 BLAS and, in most cases, achieve the best performance when executed on a single processor or a small subset of all the processors used for the factorization.

Applying the idea of dynamic execution flow definition to LU factorization leads to the implementation of the left-looking variant of the algorithm, where the panel factorizations are performed as soon as possible, with the modification that if the panel factorization introduces a stall, then an update to a block of columns (or rows) of the right submatrix is performed instead. The updating continues only until next panel factorization is possible.

Experimental results show how the dynamic workflow technique is capable of improving the overall performance while providing an extremely high level of portability. By applying dynamic task scheduling to the QR, LU and Cholesky factorizations, it is possible to out perform a standard LAPACK implementation with threaded BLAS.

References

1. J. J. Dongarra, P. Luszczek, and A. Petit, *The LINPACK Benchmark: Past, Present, and Future*, Concurrency and Computation: Practice and Experience vol. 15, no. 9, pp. 803-820, August, 2003.
<http://www.netlib.org/benchmark/hpl/>.
2. J. Kurzak and J. J. Dongarra, *Implementation of Linear Algebra Routines with Lookahead - LU, Cholesky, QR*. In Workshop on State-of-the-Art in Scientific and Parallel Computing, Umea, Sweden, June, 2006.
3. D. E. Post and L. G. Votta, *Computational Science Demands a New Paradigm* Physics Today, vol. 58, no. 1, pp. 35-41, January, 2005.
4. Herb Sutter, *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*, Dr. Dobbs's Journal, 30(3), March 2005.
5. Krste Asanovic, et. al., *The Landscape of Parallel Computing Research: A View from Berkeley*, Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183, December 18, 2006.
<http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>.