

# The Relevance of New Data Structure Approaches for Dense Linear Algebra in the new Multi-Core / Many Core Environments

Fred G. Gustavson

IBM T.J. Watson Research Center,  
Yorktown Heights, NY 10598, USA  
email: fg2@us.ibm.com

**Abstract.** For about ten years now, Bo Kågström's Group in Umea, Sweden, Jerzy Waśniewski's Team at Danish Technical University in Lyngby, Denmark, and I at IBM Research in Yorktown Heights have been applying recursion and New Data Structures (NDS) to increase the performance of Dense Linear Algebra (DLA) factorization algorithms. Later, John Gunnels, and later still, Jim Sexton, both now at IBM Research also began working in this area. For about three years now almost all computer manufacturers have dramatically changed their computer architectures which they call Multi-Core, (MC). It turns out that these new designs give poor performance for the traditional designs of DLA libraries such as LAPACK and ScaLAPACK. Recent results of Jack Dongarra's group at the Innovative Computing Laboratory in Knoxville, Tennessee have shown how to obtain high performance for DLA factorization algorithms on the Cell architecture, an example of an MC processor, but only when they used NDS. In this talk we will give some reasons why this is so.

## 1 Introduction

Multi-core/Many Core (MC) can be considered a revolution in Computing. In many of my papers I have talked about the fundamental triangle of Algorithms, Architectures and Compilers [7] or the Algorithms and Architecture approach [1, 12, 11]. MC is revolution in Architectures. The fundamental triangle concept says that all three areas are inter-related. This means Compilers and Algorithms must change and probably in a radical way. The LAPACK and ScaLAPACK projects under the direction of Jim Demmel and Jack Dongarra started an enhancement of LAPACK and ScaLAPACK projects in late 2004. Then in 2006, Jack Dongarra started another project, called PLASMA, which was directed at the effect MC would have on LAPACK. His initial findings were that traditional BLAS based LAPACK will need substantial changes. So, what appeared at first to be enhancements of these libraries now appears to be directed at more basic structural changes.

For about 10 years now the work of Bo Kågström's Group in Umea, Sweden, Jerzy Waśniewski's Team at Danish Technical University in Lyngby, Denmark,

and I at IBM Research in Yorktown Heights have been applying recursion and New Data Structures (NDS) to increase the performance of Dense Linear Algebra (DLA) factorization algorithms. More recently, John Gunnels and Jim Sexton of IBM research have also become somewhat involved. It turns out that the results of our researches are also relevant to MC processors. A useful result of our research was the introduction of new data structures, (NDS) [9, 12, 10, 2, 3, 15, 14].

The essence of MC is many cores on a single chip. The Cell BBE (broad band engine) is an example. Cell is a heterogeneous chip consisting of a single traditional PPE (power PC processor) and 8 SPE's (Synergistic Processing Element) and a novel memory system interconnect. Each SPE core can be thought of as a processor and a "cache memory". Because of this, "cache blocking" is still very important. Cache blocking was first invented by my Group at IBM in 1985 [16] and the Cedar project at the University of Illinois.

The advent of the Cray 2 was a reason for the introduction of level 3 BLAS [6] followed by the introduction of the LAPACK [4] library in the early 1990's. Now, according to some preliminary results of the PLASMA project, this level 3 BLAS approach is no longer adequate to produce high performance LAPACK codes for MC. Nonetheless, it can be argued that the broad idea of "cache blocking" is still mandatory as data in the form of matrix elements must be fed to the SPE's so they can be processed. And, equally important, is the arrangement in memory of the matrices that are being processed. So, this is what we will call "cache blocking" here.

My invited lecture at PPAM07 addressed "cache blocking" as it relates to dense linear algebra factorization algorithms (DLAFA). It repeated some of my early work on this subject and it sketched a proof that DLAFA could be viewed as just doing matrix multiply by adopting the linear transformation approach of applying equivalence transformations to a set of matrix equations  $Ax = b$  to produce an equivalent (simpler) form of these equations  $Cx = d$ . Examples of  $C$  are  $LU = PA$ , for Gaussian elimination,  $LL^T = A$ , for Cholesky Factorization, and  $QR = A$ , for Householder's factorization. I adopted this view to show a general way to produce a whole collection of DLAFA as opposed to the commonly accepted way of describing the same collection as set of distinct algorithms [8]. A second reason was to indicate that for each linear transformation I performed I was invoking the definition of matrix multiplication. Here is the gist of the proof as it applies to  $LU = PA$ .

1. Perform  $n = \lceil N/NB \rceil$  rank NB linear transformations on  $A$  to get  $U$ .
2. Each of these  $n$  composed NB linear transformations is matrix multiply by definition.
3. By the principle of equivalence we have  $Ax = b$  if and only if  $Ux = L^{-1}Pb$ .

Matrix multiplication clearly involves "cache blocking". Around the mid 1990's I noticed, see page 739 of [9], that the API for Level 3 BLAS GEMM could hurt performance. In fact, this API is also the API for 2-D arrays in Fortran and C. LAPACK and ScaLAPACK also use this API for full arrays. On the other hand, high performance implementations of GEMM do not use this API as doing

so would lead to sub-optimal performance. In fact, some amount of data copy is usually done by most high performance **GEMM** implementations. Now, level 3 BLAS are called multiple times by DLAFAs. This means that multiple data copy will usually occur in DLAFAs that use standard level 3 BLAS. The full NDS are storage formats that are good for **GEMM**. This was another main message of my invited talk at PPAM07.

DLAFA algorithms can be expressed in terms of scalar elements  $a_{i,j}$  which are one by one block matrices. Alternatively, they can be expressed in terms of partitioned submatrices,  $A(I : I + \text{NB} - 1, J : J : \text{NB} - 1)$  of order  $\text{NB}$ . See [8] for a definition of colon notation. The algorithms are almost identical. However, the later description automatically incorporates "cache blocking" into a DLAFA. Take the scalar statement  $c_{i,j} = c_{i,j} - a_{i,k}b_{k,j}$  representing matrix multiply as a fused multiply-add. The corresponding statement for partitioned submatrices becomes a kernel routine for level 3 BLAS **GEMM**. However, it is imperative to store the order  $\text{NB}$  SB's as contiguous blocks of matrix data, as this is what level 3 BLAS **GEMM** does internally. We remark that this is not possible with the standard Fortran and C API. This was another main message of my talk at PPAM07 which emphasized the importance of storing the submatrices of DLAFAs as contiguous blocks of storage. An essence of NDS for full matrices is to store their submatrices as contiguous blocks of storage.

The simple format of full NDS has each rectangular block (RB) or square block (SB) as being in standard column major (CM) or standard row major (RM) format, see [12, 11] for more details. These formats are the same as block data layout (BDL). BDL is described in [17] and this paper shows that BDL is the format that leads to minimal L1, L2, TLB misses for matrix operations that treat rows and columns equally. This result is true because the cache mapping of a SB is the identity mapping for most cache designs.

The last part of my invited lecture spoke about the recent results of the PLASMA project as it related to the Linpack benchmark  $LU = PA$  when running on the Cell processor. According to Dongarra's Team it was crucial that NDS be used as the matrix format. Using the standard API did not yield good performance results.

I close this extended abstract of my invited lecture by mentioning recent results obtained by considering the IBM new Blue Gene/L computers [5]. The simple format of full NDS needs to be re-arranged internally to take into account "cache blocking" for the L0 cache. The L0 cache is a new term defined in [14] and it refers to the register file of the FPU that is attached to the L1 cache. Full details are given in [14].

## References

1. R. C. Agarwal, F. G. Gustavson, M. Zubair. Exploiting functional parallelism of POWER2 to design high-performance numerical algorithms. *IBM Journal of Research and Development*, Vol. 38, No. 5, Sep. 1994, pp. 563,576.

2. B. S. Andersen, F. G. Gustavson, and J. Waśniewski. A Recursive Formulation of Cholesky Factorization of a Matrix in Packed Storage. *ACM TOMS*, Vol. 27, No. 2 June 2001, pp. 214-244.
3. B. S. Andersen, J. A. Gunnels, F. G. Gustavson, J. K. Reid and J. Waśniewski. A Fully Portable High Performance Minimal Storage Hybrid Cholesky Algorithm. *ACM TOMS*, Vol. 31, No. 2 June 2005, pp. 201-227.
4. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide Release 3.0*, SIAM, Philadelphia, 1999, ([http://www.netlib.org/lapack/lug/lapack\\_lug.html](http://www.netlib.org/lapack/lug/lapack_lug.html)).
5. S. Chatterjee et. al. Design and Exploitation of a High-performance SIMD Floating-point Unit for Blue Gene/L. *IBM Journal of Research and Development*, Vol. 49, No. 2-3, March-May 2005, pp. 377-391.
6. J. J. Dongarra and J. Du Croz, S. Hammarling and I. Duff. A Set of Level 3 Basic Linear Algebra Subprograms, *TOMS*, Vol. 16, No. 1, Mar. 1990, pp. 1-17.
7. E. Elmroth, F. G. Gustavson, I. Jonsson, and B. Kågström. Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software. *SIAM Review*, Vol. 46, No. 1, Mar. 2004, pp. 3,45.
8. G. Golub, C. VanLoan. Matrix Computations. Book, *John Hopkins Press*, Baltimore and London, 3rd. 1996.
9. F. G. Gustavson. Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms. *IBM Journal of Research and Development*, Vol. 41, No. 6, Nov. 1997, pp. 737,755.
10. F. G. Gustavson and I. Jonsson. Minimal Storage High Performance Cholesky via Blocking and Recursion *IBM Journal of Research and Development*, Vol. 44, No. 6, Nov. 2000, pp. 823,849.
11. F. G. Gustavson New Generalized Data Structures for Matrices Lead to a Variety of High Performance Linear Algebra Algorithms. *Computational Science - PPAM 2001*, R. Wyrzykowski, J. Dongarra, M. Paprzycki, J. Waśniewski eds., Lecture Notes in Computer Science 2328. Springer-Verlag, pp. 418-436, 2001.
12. F. G. Gustavson High Performance Linear Algebra Algorithms using New Generalized Data Structures for Matrices. *IBM Journal of Research and Development*, Vol. 47, No. 1, Jan. 2003, pp. 31,55.
13. F. G. Gustavson. New Generalized Data Structures for Matrices Lead to a Variety of High performance Dense Linear Algorithms. *Computational Science - Para 2004*, J. J. Dongarra, K. A. Madsen, J. Waśniewski, eds., Lecture Notes in Computer Science 3732. Springer-Verlag, pp. 11-20, 2006.
14. F. G. Gustavson, J. Gunnels, J. Sexton. Minimal Data Copy For Dense Linear Algebra Factorization. *Computational Science - Para 2006*, B. Kågström, E. Elmroth, J. Dongarra, J. Waśniewski eds., Lecture Notes in Computer Science 4699. Springer-Verlag, pp. 540-549, 2007.
15. F. G. Gustavson, J. Waśniewski. LAPACK Cholesky routines in rectangular full packed format. *Computational Science - Para 2006*, B. Kågström, E. Elmroth, J. Dongarra, J. Waśniewski eds., Lecture Notes in Computer Science 4699. Springer-Verlag, pp. 570-579, 2007.
16. IBM. IBM Engineering and Scientific Subroutine Library for AIX Version 3, Release 3. *IBM Pub. No. SA22-7272-00* Feb. 1986.
17. N. Park, B. Hong, V. Prasanna. Tiling, Block Data Layout, and Memory Hierarchy Performance. *IEEE Trans. Parallel and Distributed Systems*, 14(7):640-654, 2003.