

Why New Data Structures for Matrices are Necessary for Dense Linear Algebra in the new Multi-Core / Many Core Environments

Fred G. Gustavson

Adjunct Professor at Umea University and Emeritus at IBM Research

Umea, Sweden and Yorktown Heights, NY, USA

phone: 1-914-373-4607

email: fg2935@hotmail.com

July 6, 2009

Abstract

A matrix is clearly a two dimensional object. Any array in the Fortran and C Programming Languages is one dimensional. The Fortran and C programming languages provide users with “two, three and higher dimensional arrays” for use in implementing their algorithms. However, we repeat for emphasis that all of these arrays have a one dimensional layout. The fundamental result of Dimension Theory [1] states that it is *impossible* to preserve data locality in a neighborhood of a point p of a D -dimensional object when one uses a d -dimensional layout of the neighborhood of point p when $D > d$. Also, for emphasis, I will put quotes about two and higher dimensional Fortran and C arrays to indicate the apparent *paradox* in the nomenclature used by these two universal programming languages.

The traditional designs of DLA libraries such as LAPACK and ScaLAPACK write their code in Fortran and C and store their matrices (API) in “two dimensional” arrays. In these libraries, performance portability is gained by using level-3 BLAS whose API for matrices is also the Fortran “two dimensional” array.

Over a decade ago many researchers began using New Data Structures (NDS) for matrices to improve the performance of their Dense Linear Algebra (DLA) algorithms. Before that these NDS were used to improve the performance of level-3 BLAS. BLAS contributions included vendor implementations, Phipac, ATLAS, and later Goto BLAS. A lot of the former efforts were described in a March 2004 SIAM review article on the use of Recursion and NDS by the author and Umea colleagues.

The multi-core era (MC) began in late 2005 and is now firmly established. Recent results of Jack Dongarra’s group at the Innovative Computing Laboratory in Knoxville, Tennessee and many other researchers have shown how to obtain high performance for DLA factorization algorithms on MC processors, but only when they used NDS. In this talk we will give some reasons why this is so. We concentrate on the unsolved problem of transforming in-place between NDS and the two standard data structures of DLA. They are Column Major (CM) or Row Major (RM) order (note the implication of 1-D in CM and RM) for rectangular array formats and packed order (n 1-D vectors concatenated together) for symmetric and triangular array formats. We show that fast solutions to this problem exist. The *importance* of this work allows existing and current level three LAPACK and ScaLAPACK codes to obtain the benefits of the new DLA codes being developed for MC processors.

In the talk we will briefly describe the fundamental result of Dimension Theory and its related research that saw the introduction of space filling curves bearing the names of Hilbert and Peano. Recently such curves are again of interest for DLA and they partly constitute NDS. Basically, these curves (being 1-D) better approximates a 2-D matrix locally and hence 2-D submatrix processing obtains higher performance for all cache based machines. We will demonstrate that Fortran and C has a 1-D layout of any matrix A . This will imply that one cannot transpose a submatrix B of A in-place. Hence, out-of-place matrix transposition of B is universally used today.

Matrix multiplication is required to do DLA factorization. Matrix transposition is an essential ingredient of matrix multiplication algorithms. There are two types of matrix transposition algorithms. They are called out-of-place and in-place algorithms. We will demonstrate that a 1-D layout of A causes both B and B^T to be non-contiguous in A . It follows that in-place transposition algorithms *fragments* memory when one uses a cycle following permutation

paradigm to implement an in-place matrix transposition algorithm [2]. This introduces a *severe* performance problem which is commonly called cache thrashing.

We then describe how to overcome the cache thrashing problem by “*cache blocking*” our in-place matrix transposition algorithm [3, 4]. We do this by passing over matrix A multiple times instead of only once! By using NDS we can introduce both vector and sub-matrix (block) forms of in-place transformation. We also show that vector in-place transposition is identical to transforming a block column of A in CM format to rectangular block (RB) format and vice versa.

Existing in-place transposition algorithms have higher complexity than $O(mn)$. We briefly describe new work with Lars Karlsson and Bo Kagstrom that produces an $O(mn)$ algorithm for this problem.

References

- [1] H. Tietze. Three Dimensions–Higher Dimensions. *Famous Problems of Mathematics*, book, 1965, Graylock Press, pp. 106-120.
- [2] F. G. Gustavson, T. Swirszcz. In-Place Transposition of Rectangular Matrices. *Computational Science - Para 2006*, B. Kågström, E. Elmroth, J. Dongarra, J. Waśniewski eds., Lecture Notes in Computer Science 4699. Springer-Verlag, pp. 560-569, 2007.
- [3] F. G. Gustavson The Relevance of New Data Structure Approaches for Dense Linear Algebra in the new Multicore / Manycore Environments. *Computational Science - Para 2008*, A. Elster, J. Dongarra, J. Waśniewski eds., Lecture Notes in Computer Science xxxx. Springer-Verlag, pp. aaa-aaa+9, 2009. Also see IBM RC report 24599, pp. 1-10, 7/08/08.
- [4] L. Karlsson Blocked and Scalable Matrix Computations- Packed Cholesky, In-Place Transposition, and Two Sided Transformations *Licentiate Thesis* Dept. Of Computer Science, Umea, Sweden, pp. 63-91, 2009