

# Using non-canonical array layouts in dense matrix operations<sup>\*</sup>

José R. Herrero, Juan J. Navarro

Computer Architecture Dept., Univ. Politècnica de Catalunya  
Barcelona, (Spain)  
{josepr,juanjo}@ac.upc.edu,

**Abstract.** We present two implementations of dense matrix multiplication based on two different non-canonical array layouts: one based on a hypermatrix data structure (HM) where data submatrices are stored using a recursive layout; the other based on a simple block data layout with square blocks (SB) where blocks are arranged in column-major order. We show that the iterative code using SB outperforms a recursive code using HM and obtains competitive results on a variety of platforms.

## 1 Introduction

A matrix representation is a method used by a computer language to store matrices of more than one dimension in memory. Fortran and C use different schemes. Fortran uses "Column Major", in which all the elements for a given column are stored contiguously in memory. C uses "Row Major", which stores all the elements for a given row contiguously in memory. These two schemes are considered canonical storage. The default column/row-major order used by programming languages such as Fortran and C limits locality to a single dimension.

As processor speed continues to increase relative to memory speed, locality optimizations get to be a significant performance issue for algorithms operating on large matrices. Data has to be reused in cache as effectively as possible: locality has to be exploited. In low associativity caches conflicts should be avoided or reduced. A considerable amount of research has been conducted towards achieving an effective use of memory hierarchies. Next, we provide an overview of such work. Although we tried to be complete we are sure we missed some references. Performance can only be obtained by matching the algorithm to the architecture and vice-versa [1]. Conventional techniques such as *tiling* [2–4], *precopying* [5] and *padding* [5, 6] have been used extensively. In addition, alternative storage formats have been proposed to address the issue of locality.

### 1.1 Serial dense codes using non-canonical array layouts

A submatrix storage was proposed in [7] with the purpose of minimizing the page faulting occurring in a paged memory system. The authors partition matrices

---

<sup>\*</sup> This work was supported by the Ministerio de Educación y Ciencia of Spain (TIN2004-07739-C02-01)

<http://people.ac.upc.edu/josepr/>

into square submatrices, keeping one submatrix per page, obtaining orders of magnitude improvement in the number of page faults for several common matrix operations.

In the last ten years there have been several studies on the application of non-canonical array layouts in uniprocessor environments. Recursivity has been introduced into linear algebra codes. Block recursive codes for dense linear algebra computations appear to be well-suited for execution on machines with deep memory hierarchies because they are effectively blocked for all levels of the hierarchy [8, 9]. Unfortunately, block recursive algorithms do not interact well with the TLB [10]. This has led to the eruption of new storage formats [11–16].

Different authors refer to a given data layout using different names. A data layout where matrices are stored as submatrices which are in turn stored by columns has been named as Submatrix storage in [7], BC in [11, 17], SB in [14, 18–20], 4D in [21], BDL in [22], and TDL in [23]. In this document we will refer to such data layout as SB (Square Block Format). This name reflects the square nature of the submatrices and is the one used most extensively in the literature.

Some studies have focused on the use of quadrees or Space Filling Curves (SFC) for serial dense codes. In [24], a recursive matrix multiplication algorithm with quadrees was presented. This algorithm uses recursion down to the level of single array elements which causes a dramatic loss of performance. Later, the same authors improved performance by stopping recursion at  $8 \times 8$  blocks [25]. In [26] the authors have experimented with five different SFCs (U, X, Z, Gray and Hilbert) on the matrix multiplication algorithm. The performance reported was similar for all five. Morton (Z) order has relative simplicity in calculating block addresses compared to the other orderings and is often the order of choice. In spite of its relative simplicity compared to other layouts, calculation of addresses in Morton layout is expensive. There are several indexing techniques which differ in their structure, but which all induce Morton order: Morton, level-order, and Ahnentafel indexing. These indexing schemes require bit manipulation unless a lookup table is precomputed [21]. Bit masks can be used when dimensions are powers of two [27]. However, this requires padding.

Tiling can also be applied to non-canonical data layouts. In [22] the authors show that improved cache and TLB performance can be achieved when tiling is applied to both Block Data Layout (BDL) and Morton layout. In their experiments matrix multiplication with an iterative code using BDL was often faster than a recursive code using Morton layout. As we will comment below, our results agree with this: our iterative tiled algorithm working on SB outperforms the recursive code operating on hypermatrices. Authors have also investigated on tile size selection for non-canonical array layouts [28, 22, 29] and have come to similar conclusions to the case of canonical storage: blocks should target the level 1 cache.

## 2 A bottom-up approach

We have studied two data structures for dense matrix computations: a Hypermatrix data structure [30] and a Square Block Format [14]. We present them in

section 3. In both cases we drive the creation of the structure from the bottom: the inner kernel fixes the size of the data submatrices. Then the rest of the data structure is produced in conformance. We do this because the performance of the inner kernel has a dramatic influence in the overall performance of the algorithm. Thus, our first priority is to use the best inner kernel at hand. Afterwards, we can adapt the rest of the data structure (in case hypermatrices are used) and/or the computations.

### 2.1 Inner kernel based on our Small Matrix Library (SML)

In previous papers [31,23] we presented our work on the creation of a Small Matrix Library (SML): a set of routines, written in Fortran, specialized in the efficient operation on matrices which fit in the first level cache. The advantage of our method lies in the ability to generate very efficient inner kernels by means of a good compiler. Working on regular codes for small matrices, most of the compilers we have used in different platforms create very efficient inner kernels for matrix multiplication. We use the matrix multiplication routine within our SML as the inner kernel of our general matrix multiplication codes.

## 3 Non-canonical array layouts

In this section we briefly describe two non-canonical data layouts: a hypermatrix scheme and a simple square block format.

### 3.1 Hypermatrix structure

We have used a data structure based on a hypermatrix (HM) scheme [30], in which a matrix is partitioned recursively into blocks of different sizes. The HM structure consists of  $N$  levels of submatrices, where  $N$  is an arbitrary number. In order to have a simple HM data structure which is easy to traverse we have chosen to have blocks at each level which are multiples of the lower levels. The top  $N-1$  levels hold pointer matrices which point to the next lower level submatrices. Only the last (bottom) level holds data matrices (see Figure 1). Data matrices are stored as dense matrices and operated on as such. Hypermatrices can be seen as a generalization of quadtrees. The latter partition each matrix precisely into four submatrices [32].

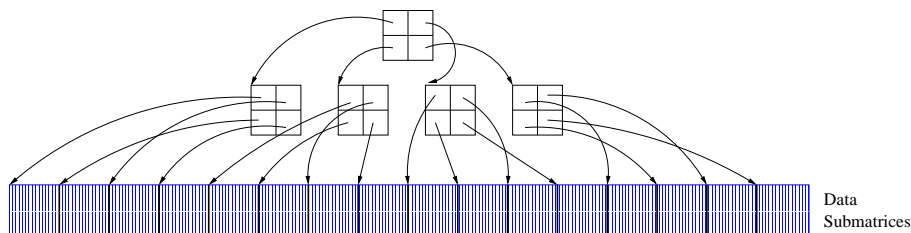
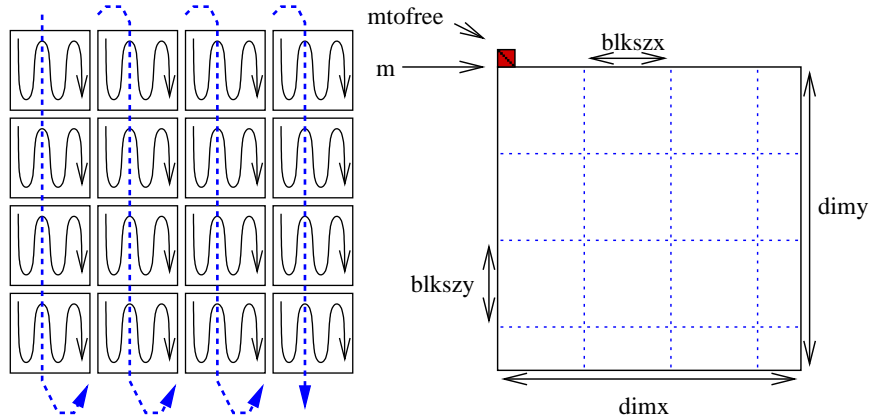


Fig. 1. Hypermatrix: example with two level of pointer matrices.

We have used a HM on dense Cholesky factorization and matrix multiplication with encouraging results. In [33] we showed that the use of orthogonal blocks [34] was beneficial to obtain performance. However, this approach presents some overhead following pointers and recursing down to the data submatrix level. There are also difficulties in the parallelization [35]. For these reasons we have also experimented with a Square Block Format.

### 3.2 Square Block Format

The overhead of a dense code based on recursion and hypermatrices together with the difficulties to produce efficient parallel code based on this data structure, has led us to experiment with a different data structure. We use a simple Square Block Format (*SB*) [14] stored as a 4D array. It corresponds to a 2D data layout of submatrices kept in column-major order as can be seen in the left part of Figure 2. We use a simple data structure which is described graphically in the right part of Figure 2. The shaded area at the top represents padding introduced to force data alignment.



**Fig. 2.** Square Block Format.

Using this data structure we were able to improve the performance of our matrix multiplication code, obtaining very competitive results. Our code implements tiling and we use a code generator to create different loop orders. We present the results obtained with the best loop order found.

## 4 Results

We present results for matrix multiplication on three platforms. The matrix multiplication performed is  $C = C - A^T \times B$ . Each of the following figures shows the results of DGEMM in ATLAS [36], Goto [37] or the vendor BLAS, and our code based on SB format and our SML. Goto BLAS are known to obtain excellent performance on Intel platforms. They are coded in assembler

and targeted to each particular platform. The dashed line at the top of each plot shows the theoretical peak performance of the processor. Some plots show the performance obtained with the dense codes based on the hypermatrix (HM) scheme. It can be seen on the plots that SB outperforms HM.

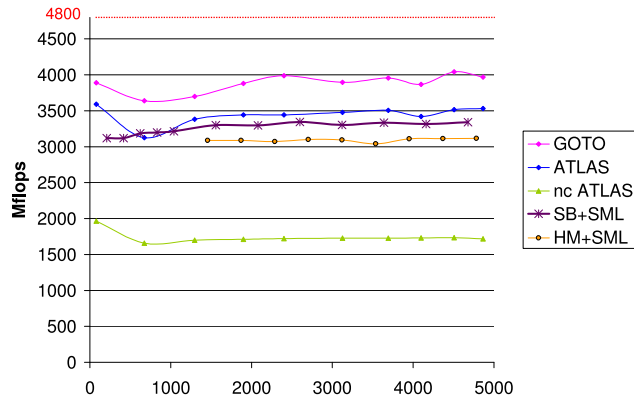


Fig. 3. Performance of DGEMM on Intel Pentium 4 Xeon.

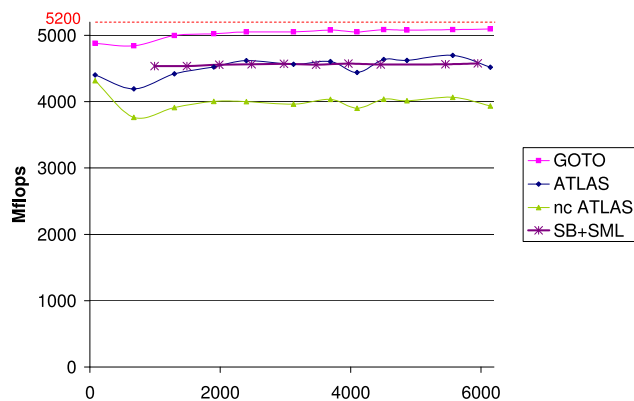


Fig. 4. Performance of DGEMM on Intel Itanium 2.

For the Intel machines (figures 3 and 4) we have included the Mflops obtained with a version of the ATLAS library where the hand-made codes were not enabled at ATLAS installation time<sup>1</sup>. We refer to this code in the graphs

<sup>1</sup> Directory `tune/blas/gemm/CASES` within the ATLAS distribution contains about 90 files which are, in most cases, written in assembler, or use some instructions written in assembler to do data prefetching. Often, one (or more) of these codes outperform the automatically generated codes. The best code is (automatically) selected as the inner kernel. The use of such hand-made inner kernels has improved significantly the overall performance of ATLAS subroutines on some platforms.

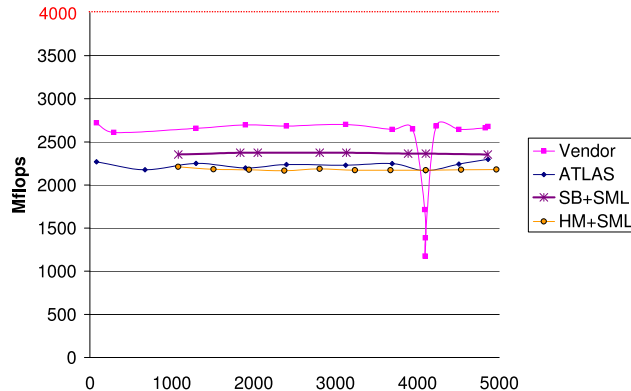


Fig. 5. Performance of DGEMM on Power 4.

as 'nc ATLAS'. We can observe that in both cases ATLAS performance drops heavily. SB with SML kernels obtain performance close to that of ATLAS on the Pentium 4 Xeon, similar to ATLAS on the Itanium 2, and better than ATLAS on the Power4 (Figure 5). For the latter we show the Mflops obtained by the vendor DGEMM routine which outperforms both ATLAS and our code based on SB. We can see that even highly optimized routines provided by the vendor may be significantly slower under certain circumstances. For instance, some large leading dimensions can be particularly harmful and produce lots of TLB misses if data is not precopied. At the same time, data precopying must be performed selectively due to the overhead incurred at execution time [6]. These problems can be avoided using non-canonical array layouts.

## 5 Conclusions

The results obtained with an iterative code working on a Square Block Format surpass those obtained with a recursive code which uses a hypermatrix. This happens even when the upper levels of the hypermatrix are defined so that the upper levels of the memory hierarchy are properly exploited. This is due to the overhead caused by recursion. We conclude that a simple Square Block format provides a good way to exploit locality and iterative codes can outperform recursive codes. Our results agree with those presented in [22, 38].

## References

1. Agarwal, R.C., Gustavson, F.G., Zubair, M.: Exploiting functional parallelism of POWER2 to design high-performance numerical algorithms. *IBM J. Res. Dev.* **38** (1994) 563–576
2. Gallivan, K., Jalby, W., Meier, U., Sameh, A.: Impact of hierarchical memory systems on linear algebra algorithm design. *Int. J. of Supercomputer Appl.* **2** (1988) 12–48

3. Irigoin, F., Triolet, R.: Supernode partitioning. In: POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, New York, NY, USA, ACM Press (1988) 319–329
4. Wolfe, M.: More iteration space tiling. In ACM, ed.: Proceedings, Supercomputing '89: November 13–17, 1989, Reno, Nevada, ACM Press (1989) 655–664
5. Lam, M., Rothberg, E., Wolf, M.: The cache performance and optimizations of blocked algorithms. In: Proceedings of ASPLOS'91. (1991) 67–74
6. Temam, O., Granston, E.D., Jalby, W.: To copy or not to copy: a compile-time technique for assessing when data copying should be used to eliminate cache conflicts. In: Supercomputing. (1993) 410–419
7. McKellar, A.C., E. G. Coffman, J.: Organizing matrices and matrix operations for paged memory systems. *Communications of the ACM* **12** (1969) 153–165
8. Gustavson, F.G.: Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM J. Res. Dev.* **41** (1997) 737–756
9. Toledo, S.: Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.* **18** (1997) 1065–1081
10. Ahmed, N., Pingali, K.: Automatic generation of block-recursive codes. In: Euro-Par 2000, LNCS1900. (2000) 368–378
11. Gustavson, F., Henriksson, A., Jonsson, I., Kågström, B.: Recursive blocked data formats and BLAS's for dense linear algebra algorithms. *LNCS* **1541** (1998) 195–206
12. Andersen, B.S., Gustavson, F.G., Karaivanov, A., Marinova, M., Wasniewski, J., Yalamov, P.Y.: LAWRA: Linear algebra with recursive algorithms. In Sørøvik, T., Manne, F., Moe, R., Gebremedhin, A.H., eds.: *PARA*. Volume 1947 of *Lecture Notes in Computer Science.*, Springer (2000) 38–51
13. Andersen, B.S., Wasniewski, J., Gustavson, F.G.: A recursive formulation of Cholesky factorization of a matrix in packed storage. *ACM Transactions on Mathematical Software (TOMS)* **27** (2001) 214–244
14. Gustavson, F.G.: New generalized data structures for matrices lead to a variety of high-performance algorithms. In Engquist, B., ed.: *Simulation and visualization on the grid: Paralleldatorcentrum, Kungl. Tekniska Högskolan, 7th annual conference*, Stockholm, Sweden, December 1999: proceedings. Volume 13 of *Lecture Notes in Computational Science and Engineering.*, Springer-Verlag Inc. (2000) 46–61
15. Andersen, B.S., Gunnels, J.A., Gustavson, F., Wasniewski, J.: A recursive formulation of the inversion of symmetric positive definite matrices in packed storage data format. In Juha Fagerholm, Juha Haataja, Jari Jarvinen, Mikko Lyly, Peter Raback, Savolainen, V., eds.: *PARA'02*. Volume 2367 of *LNCS.*, Heidelberg, Springer - Verlag (2002) 287–296
16. Andersen, B.S., Gunnels, J.A., Gustavson, F.G., Reid, J.K., Wasniewski, J.: A fully portable high performance minimal storage hybrid format Cholesky algorithm. *ACM Transactions on Mathematical Software* **31** (2005) 201–227
17. Elmroth, E., Gustavson, F., Jonsson, I., Kågström, B.: Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review* **46** (2004) 3–45
18. Gustavson, F.G.: High-performance linear algebra algorithms using new generalized data structures for matrices. *IBM J. Res. Dev.* **47** (2003) 31–55
19. Gustavson, F.G.: New generalized data structures for matrices lead to a variety of high performance dense linear algebra algorithms. In Dongarra, J., Madsen, K., Wasniewski, J., eds.: *PARA'04*. Volume 3732 of *LNCS.*, Springer (2004) 11–20
20. Gustavson, F.G.: Algorithm Compiler Architecture Interaction Relative to Dense Linear Algebra. Technical Report RC23715 (W0509-039), IBM, T.J. Watson (2005)

21. Chatterjee, S., Jain, V.V., Lebeck, A.R., Mundhra, S., Thottethodi, M.: Nonlinear array layouts for hierarchical memory systems. In: Proceedings of the 13th international conference on Supercomputing, ACM Press (1999) 444–453
22. Park, N., Hong, B., Prasanna, V.K.: Tiling, block data layout, and memory hierarchy performance. *IEEE Trans. Parallel and Distrib. Systems* **14** (2003) 640–654
23. Herrero, J.R., Navarro, J.J.: Compiler-optimized kernels: An efficient alternative to hand-coded inner kernels. In: Proceedings of the International Conference on Computational Science and its Applications (ICCSA). LNCS 3984. (2006) 762–771
24. Frens, J.D., Wise, D.S.: Auto-blocking matrix-multiplication or tracking BLAS3 performance from source code. *Proc. 6th ACM SIGPLAN Symp. on Principles and Practice of Parallel Program.*, SIGPLAN Notices **32** (1997) 206–216
25. Wise, D.S., Frens, J.D.: Morton-order matrices deserve compilers' support. Technical Report TR 533, Computer Science Department, Indiana University (1999)
26. Chatterjee, S., Lebeck, A.R., Patnala, P.K., Thottethodi, M.: Recursive array layouts and fast parallel matrix multiplication. In: Proc. of the 11th annual ACM symposium on Parallel algorithms and architectures, ACM Press (1999) 222–231
27. Athanasaki, E., Koziris, N.: Fast indexing for blocked array layouts to improve multi-level cache locality. In: Interaction between Compilers and Computer Architectures. (2004) 109–119
28. Valsalam, V., Skjellum, A.: A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and optimized low-level kernels. *Concurrency and Computation: Practice and Experience* **14** (2002) 805–839
29. Athanasaki, E., Koziris, N., Tsanakas, P.: A tile size selection analysis for blocked array layouts. In: Interaction between Compilers and Computer Architectures. (2005) 70–80
30. Fuchs, G., Roy, J., Schrem, E.: Hypermatrix solution of large sets of symmetric positive-definite linear equations. *Comp. Meth. Appl. Mech. Eng.* **1** (1972) 197–216
31. Herrero, J.R., Navarro, J.J.: Automatic benchmarking and optimization of codes: an experience with numerical kernels. In: *Int. Conf. on Software Engineering Research and Practice*, CSREA Press (2003) 701–706
32. Wise, D.S.: Representing matrices as quadtrees for parallel processors. *Information Processing Letters* **20** (1985) 195–199
33. Herrero, J.R., Navarro, J.J.: Adapting linear algebra codes to the memory hierarchy using a hypermatrix scheme. In: *Int. Conf. on Parallel Processing and Applied Mathematics*. LNCS 3911. (2005) 1058–1065
34. Navarro, J.J., Juan, A., Lang, T.: MOB forms: A class of Multilevel Block Algorithms for dense linear algebra operations. In: Proceedings of the 8th International Conference on Supercomputing, ACM Press (1994) 354–363
35. Herrero, J.R., Navarro, J.J.: A study on load imbalance in parallel hypermatrix multiplication using OpenMP. In: *Int. Conf. on Parallel Processing and Applied Mathematics*. LNCS 3911. (2005) 124–131
36. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: *Supercomputing '98*, IEEE Computer Society (1998) 211–217
37. Goto, K., van de Geijn, R.: On reducing TLB misses in matrix multiplication. Technical Report CS-TR-02-55, Univ. of Texas at Austin (2002)
38. Yotov, K., Roeder, T., Pingali, K., Gunnels, J., Gustavson, F.: Is cache oblivious DGEMM a viable alternative? In: *PARA'06*, Springer-Verlag (2006) To appear