

New Data Structures for Matrices and Specialized Inner Kernels: Low overhead for High Performance*

José R. Herrero

Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
josepr@ac.upc.edu

Abstract. Dense linear algebra codes are often expressed and coded in terms of BLAS calls. This approach, however, achieves suboptimal performance due to the overheads associated to such calls.

Taking as an example the dense Cholesky factorization of a symmetric positive definite matrix we show that the potential of non-canonical data structures for dense linear algebra can be better exploited with the use of specialized inner kernels. The use of non-canonical data structures together with specialized inner kernels has low overhead and can produce excellent performance.

1 Introduction

Linear Algebra codes often make use of Level 3 Basic Linear Algebra Subroutines (BLAS) [1]. Performance portability is achieved through high performance libraries such as ATLAS [2], Goto BLAS [3] or those provided by machine vendors. Operations are often expressed in terms of matrix-matrix multiplication operations (calls to routine GEMM) [4]. Thus, a great effort is devoted to the implementation of this routine [5–11].

Data precopying [12] can be very useful to exploit locality and facilitate data streaming. However, such copies introduce overhead [13] since they are repeated in every call to BLAS routines [9, 14].

Figure 1 shows the performance of several codes using version 3.7.11 of the ATLAS library¹. We can observe that the performance obtained by the matrix multiplication routine (DGEMM) is substantially higher than that obtained by the Cholesky factorization routine (DPOTRF). This is mainly due to the overhead mentioned above.

* This work was supported by the Ministerio de Educación y Ciencia of Spain with grants TIN2004-07739-C02-01 and TIN2007-60625.

¹ Experiments have been conducted on an Intel Itanium 2 processor running at 1.3 GHz with a theoretical peak performance of 5.2 GFlops (indicated by the dashed line at the top of some graphs).

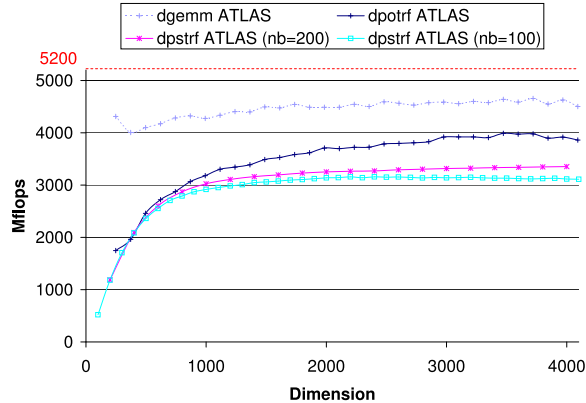


Fig. 1. Performance of matrix multiplication and Cholesky factorization using ATLAS.

New Data Structures (NDS) have been introduced in dense linear algebra codes as an alternative to canonical storage [15–25]. The main goals are improving locality, avoiding data copies, and obtaining reduced (or even minimum) storage requirements for triangular and symmetric matrices while achieving performance similar to codes which work on full storage.

When NDS are used data copies can be avoided: instead of calling BLAS routines, simpler kernels can be defined and used. Several papers reflect the need and/or the creation of such kernels [10, 14, 19, 21, 23, 24, 26–31]. If BLAS calls are used, however, then unnecessary overhead is paid. The two curves at the bottom of Figure 1 show the performance of a Cholesky factorization routine (DPSTRF) based on a Square Blocked Packed Format (SBPF) [19] for two block sizes ($nb=200$ and $nb=100$) when calls to ATLAS BLAS routines are performed. The lower matrix is used in our implementation. We can observe that performance drops for the smaller block size. This happens because the overhead is multiplied due to the larger number of calls.

For several years Goto’s BLAS library has become the reference library due to its great performance. Recently, changes have been introduced in Goto’s library to produce even higher performance for BLAS3 operations [3]. The authors have modified the copying procedure to avoid redundant packing and, therefore, reduce the overhead and increase performance for operations such as the Symmetric Rank-K (SYRK) and Triangular Solve (TRSM) amongst others. Since the Cholesky operation is implemented with calls to these operations, together with DGEMM, the performance of both DPOTRF and DPSTRF is excellent. Figure 2 shows the performance obtained with these two routines. We have used Goto’s BLAS version 1.15 in our experiments. The curve at the top corresponds to a direct call to DPOTRF, while the rest correspond to calls to DPSTRF with different block sizes. Although performance drops for smaller block sizes, performance is remarkable. Thus, the avoidance of redundant packing together

with the high performance matrix multiplication kernels [11] are very effective and seem good examples to follow.

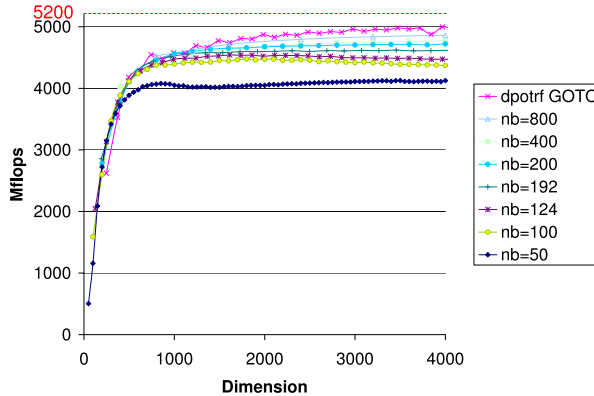


Fig. 2. Performance of Cholesky factorization for several block sizes using Goto's library.

Our goal is to analyze the potential of NDS combined with specialized inner kernels. Thus, first we need to determine the best candidate for our study. Recursivity in dense linear algebra has been explored for several years [23, 32–34], but recently it has been shown that iterative codes with proper block sizes can achieve better performance than recursive codes [35–37]. For this reason we focus our study on a dense Cholesky factorization using an iterative approach. We have chosen to work with a Square Blocked Lower Packed Format [19] with the TRANS parameter fixed to 'T'. Although this format exceeds the minimum storage provided by other formats such as a Hybrid Full Packed (HFP) [22] format, it is rather simple, it allows for data accesses in the inner kernels with stride one and it is easy to adapt to our inner kernels. This allows us to obtain upper bounds on the performance of this approach. In the same direction, we compare only with the best combinations for DGEMM ($A^T \times B$) and DPOTRF (UPLO='U') assuming Fortran column-major storage.

2 Specialized Inner Kernels

In this section we briefly state our approach to the creation of high performance specialized kernels.

Profiling Optimization efforts must be applied to those parts of code which take up most computation time. In this case, for instance, this means focusing on the optimization of the matrix-matrix multiplication routine first.

A bottom-up approach We drive the creation of the structure from the bottom: the inner kernel fixes the size of the data submatrices [37]. 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 and/or the computations.

Specialization Code specialization is commonly used to optimize generic routines which are otherwise difficult to optimize. Specialization can simplify the code and, at the same time, allow the usage of other optimization techniques as, for instance, memoization [38]. Simple codes are easier to optimize, both automatically and manually.

Inner kernel based on our Small Matrix Library (SML) In previous papers [29, 31] 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 regular codes such as matrix-matrix multiplication.

Applying this approach to the Cholesky factorization we first create the inner kernel for the matrix multiplication operation (GEMM). This results in a block size $nb = 124$ for the Itanium 2 machine. Once the block size is already fixed we apply the same approach to the other operations (TRSM, SYRK, and POTRF). We use the resulting routines, which we store within the SML, as the inner kernels of our general NDS linear algebra codes.

3 Results

We present performance of the Cholesky factorization of a matrix into an upper triangular matrix using routine DPOTRF in both Goto's library and ATLAS; and DPSTRF working on a lower triangular matrix in SBP format which calls the inner kernels in SML. We use the DPSTRF routine in a similar way as Gustavson does in [21] with the only differences that we use a block size of 124×124 and SML routines are called in our case. We also present results for matrix multiplication. The matrix multiplication performed is $C = C - A^T \times B$. We show results of DGEMM corresponding to ATLAS [2], Goto [9] and our code based on SB format and the SML [37]. Goto BLAS are known to obtain excellent performance on Intel platforms. They are coded in assembler and targeted to each particular platform. Figure 3 shows the performance of these six codes on an Intel Itanium 2. The dashed line at the top of the plot shows the theoretical peak performance of the processor.

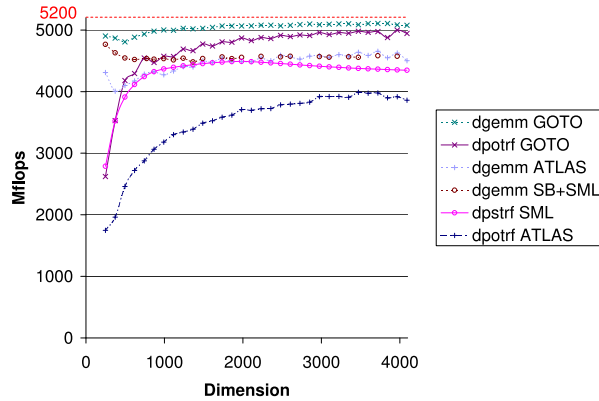


Fig. 3. Performance of several matrix multiplication and Cholesky factorization codes.

We can observe that the performance of our codes based on NDS and SML are similar to ATLAS for matrix multiplication and outperform ATLAS for Cholesky factorization. Goto's codes are in all cases the best. The merit of the combination of NDS and SML codes lays in the fact that they are the only ones which are not based on codes written in assembly language.

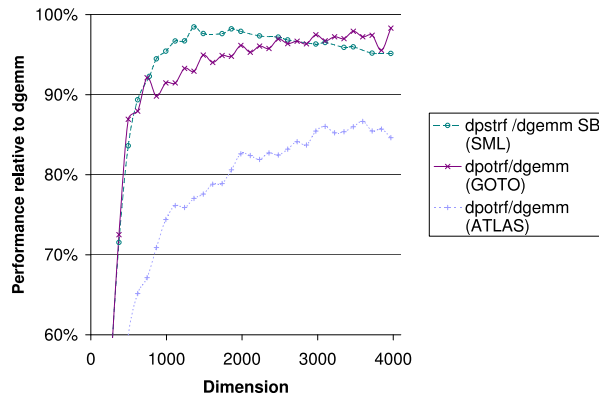


Fig. 4. Performance of Cholesky factorization relative to matrix multiply.

We have mentioned in the introduction that calls to BLAS routines usually introduce some overhead (although it has been reduced in Goto's library). This can be better observed in Figure 4 which shows the performance of Cholesky factorization relative to the corresponding matrix multiplication subroutine. ATLAS represents the traditional approach to implementing linear algebra codes in terms of BLAS routines. This results in poor performance of DPOTRF rela-

tive to DGEMM. Goto’s library however has a considerably higher ratio due to the modifications commented in section 1. We can observe that the ratio DPSTRF/DGEMM SB is very high. This is due to the low overhead present in the calls to SML routines from the NDS code. We note that the relative performance drops for matrix dimensions $\gtrsim 2000$. The reason for this is that we have implemented multilevel orthogonal block forms (MOB) [5, 37] in our matrix multiplication code based on the SB format. However, we have not used any additional blocking technique on the SBPF Cholesky. Thus, locality is not fully exploited for larger matrices within the DPSTRF routine.

In [24] the authors provide some performance results for Packed Recursive (PR) and Packed Hybrid (PH) formats. In order to have an approximate hint of their performance we have taken the values taken on an Itanium 2 running at 1 GHz. The theoretical peak performance of that machine is 4 GFlops. In order to compare results we present performance relative to the theoretical peak. Although this is not exactly the same as running the program on the same machine it can give us a hint on the relative performance of these codes. We present the case where the upper matrix and a block size $nb = 200$ are used. Figure 5 presents these results. The curves labeled with a final + sign include the time necessary to perform the rearrangement of data from canonical storage into the NDS tested. The cost of the rearrangement is $O(N^2)$ [14, 31, 39]. We can observe that the SBP format with calls to SML routines provides higher performance even when the block size is smaller $nb = 124$.

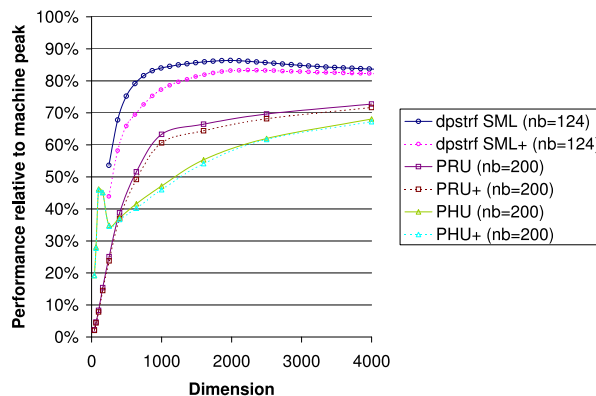


Fig. 5. Performance of several NDS Cholesky factorization codes with and without rearrangement.

Finally, Figure 6 presents the performance of all variants of Cholesky factorization tested. This figure includes the combination of DPSTRF with calls to BLAS from Goto’s library with a block size $nb = 124$. We can observe that the performance obtained is slightly better than that corresponding to the case

where SML routines are called. Both cases indicate that it is possible to have reduced storage and very high performance simultaneously.

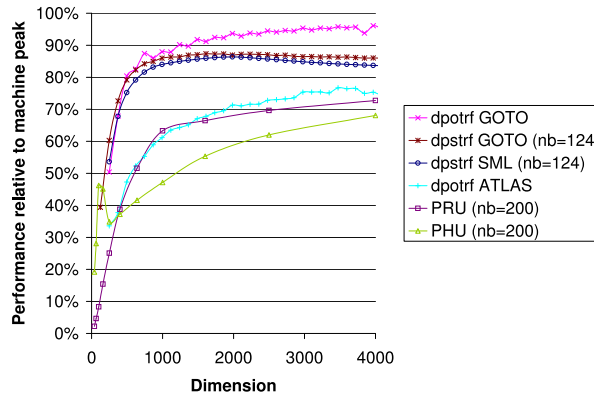


Fig. 6. Performance of Cholesky factorization codes.

4 Conclusions

The specialization of the inner kernels avoids performing unnecessary operations repetitively. In addition, it simplifies the code allowing for more opportunities for automatic optimization. Working with simple square blocks it is possible to produce efficient inner kernels with the help of an optimizing compiler.

When these kernels are called directly from linear algebra codes which store matrices using non-canonical data structures the overhead is very low. This happens because there are no costs associated to copying data and checking certain parameters. The performance obtained from the resulting Cholesky factorization is better than that of several previous implementations and approaches that of a hand-optimized implementation in which most representative parts of the code are written in assembly code and data is packed for efficient use of the register file (Goto BLAS).

References

1. Dongarra, J., Croz, J.D., Duff, I., Hammarling, S.: A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.* **16** (1990) 1–17
2. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: *Supercomputing '98*, IEEE Computer Society (1998) 211–217
3. Goto, K., Geijn, R.V.D.: High-performance implementation of the level-3 BLAS. *ACM Transactions on Mathematical Software (TOMS)* (To appear)
4. Kågström, B., Ling, P., van Loan, C.: GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark. *ACM Transactions on Mathematical Software (TOMS)* **24** (1998) 268–302

5. 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
6. 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
7. Navarro, J.J., García, E., Herrero, J.R.: Data prefetching and multilevel blocking for linear algebra operations. In: Proceedings of the 10th international conference on Supercomputing, ACM Press (1996) 109–116
8. Gunnels, J.A., Henry, G., van de Geijn, R.A.: A family of high-performance matrix multiplication algorithms. In: International Conference on Computational Science (1). (2001) 51–60
9. 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)
10. Chatterjee, S., Bacheega, L.R., Bergner, P., Dockser, K.A., Gunnels, J.A., Gupta, M., Gustavson, F.G., Lapkowski, C.A., Liu, G.K., Mendell, M., Nair, R., Wait, C.D., Ward, T.J.C., Wu, P.: Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L. *IBM Journal of Research and Development* **49** (2005) 377–391
11. Goto, K., van de Geijn, R.A.: Anatomy of a high-performance matrix multiplication. *ACM Transactions on Mathematical Software* **34** (2007)
12. Lam, M., Rothberg, E., Wolf, M.: The cache performance and optimizations of blocked algorithms. In: Proceedings of ASPLOS'91. (1991) 67–74
13. 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
14. Gustavson, F.G., Gunnels, J.A., Sexton, J.C.: Minimal data copy for dense linear algebra factorization. In: PARA'06. Volume 4699 of Lecture Notes in Computer Science., Springer (2006) 540–549
15. McKellar, A.C., E. G. Coffman, J.: Organizing matrices and matrix operations for paged memory systems. *Communications of the ACM* **12** (1969) 153–165
16. 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
17. 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
18. 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
19. 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
20. 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
21. Gustavson, F.G.: High-performance linear algebra algorithms using new generalized data structures for matrices. *IBM J. Res. Dev.* **47** (2003) 31–55

22. Gunnels, J.A., Gustavson, F.G.: A new array format for symmetric and triangular matrices. In Dongarra, J., Madsen, K., Wasniewski, J., eds.: *PARA*. Volume 3732 of *Lecture Notes in Computer Science.*, Springer (2004) 247–255
23. 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
24. Andersen, B.S., Gunnels, J.A., Gustavson, F.G., Reid, J.K., Waśniewski, J.: A fully portable high performance minimal storage hybrid format Cholesky algorithm. *ACM Transactions on Mathematical Software* **31** (2005) 201–227
25. Bader, M., Mayer, C.: Cache oblivious matrix operations using Peano curves. In: *PARA'06*. Volume 4699 of *Lecture Notes in Computer Science.*, Springer-Verlag (2006) 521–530
26. Siek, J.G., Lumsdaine, A.: A rational approach to portable high performance: The basic linear algebra instruction set (BLAIS) and the fixed algorithm size template (FAST) library. In: *Object-Oriented Technology, ECOOP'98 Workshop Reader*. Volume 1543 of *Lecture Notes in Computer Science.*, Springer (1998) 468–469
27. Wise, D.S., Frens, J.D.: Morton-order matrices deserve compilers' support. Technical Report TR 533, Computer Science Department, Indiana University (1999)
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. 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
30. Kågström, B.: Management of deep memory hierarchies - recursive blocked algorithms and hybrid data structures for dense matrix computations. In Dongarra, J., Madsen, K., Wasniewski, J., eds.: *PARA*. Volume 3732 of *Lecture Notes in Computer Science.*, Springer (2004) 21–32
31. 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
32. Gustavson, F.G.: Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM J. Res. Dev.* **41** (1997) 737–756
33. Toledo, S.: Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.* **18** (1997) 1065–1081
34. 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
35. 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
36. Gunnels, J., Gustavson, F., Pingali, K., Yotov, K.: Is cache-oblivious DGEMM viable? In: *PARA'06*. Volume 4699 of *Lecture Notes in Computer Science.*, Springer-Verlag (2006) 919–928
37. Herrero, J.R., Navarro, J.J.: Using non-canonical array layouts in dense matrix operations. In: *PARA'06*. Volume 4699 of *Lecture Notes in Computer Science.*, Springer-Verlag (2006) 580–588
38. Michie, D.: Memo functions and machine learning. *Nature* **218** (1968) 19–22
39. Gustavson, F.G.: Algorithm Compiler Architecture Interaction Relative to Dense Linear Algebra. Technical Report RC23715 (W0509-039), IBM, T.J. Watson (2005)