Using nonlinear array layouts in dense matrix operations*

José R. Herrero, Juan J. Navarro

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

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

1 A bottom-up approach

We have studied two data structures for dense matrix computations: a Hypermatrix data structure [1] and a Square Block Format [2]. 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 accordance. 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 using 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.

Inner kernel based on our Small Matrix Library (SML) In previous papers [3, 4] we presented our work on the creation of a Small Matrix Library (SML): a set of routines 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.

* This work was supported by the Ministerio de Ciencia y Tecnología of Spain (TIN2004-07739-C02-01)

Hypermatrix structure We have used a data structure based on a hypermatrix (HM) scheme [1], 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. 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 [5]. We have used a HM on dense Cholesky factorization and matrix multiplication with encouraging results. In [6] we showed that the use of orthogonal blocks 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 [7]. For these reasons we have also experimented with a Square Block Format.

Square Block Format The overhead of a dense code based on hypermatrices due to the recursivity and indexing, together with the difficulties to produce efficient parallel codes based on this data structure, has led us to experiment with a different data structure. We use a simple Square Block Format (*SB*) [2]. It corresponds to a 2D data layout of submatrices stored in column-major order (see figure 1). The shaded area represents padding introduced to force data alignment.

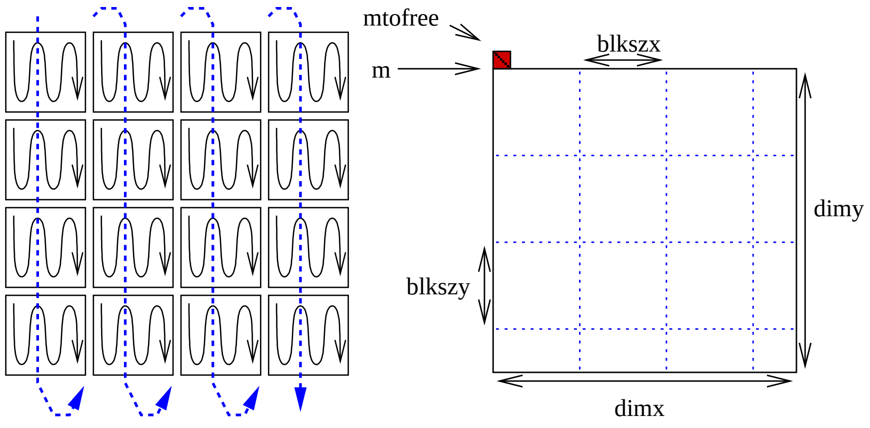

Fig. 1. Square Block Format.

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

2 Results

We present results for matrix multiplication on three platforms. The matrix multiplication used is $C = C - A^T \times B$. Each of the following figures shows the results of DGEMM in ATLAS, Goto or the vendor BLAS, and SB using our SML. Goto BLAS [8] are known to obtain excellent performance. 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. As can be seen on the plots SB outperforms HM.

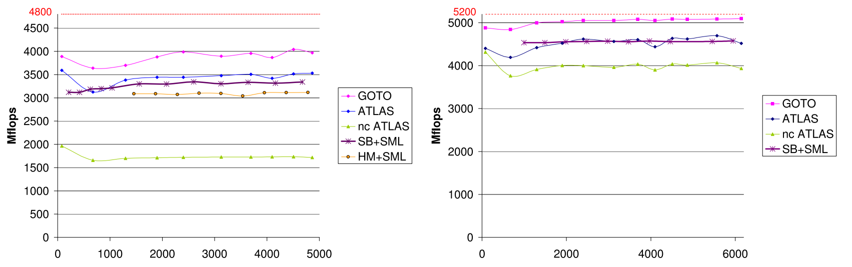

Fig. 2. Performance of dense matrix multiplication on an Intel Pentium 4 Xeon and an Itanium 2 processor.

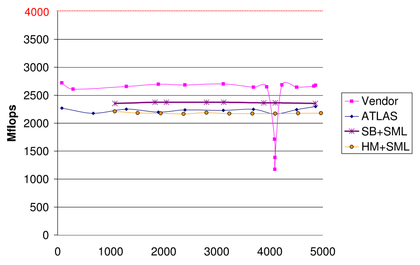

Fig. 3. Performance of dense matrix multiplication on a Power 4 processor.

For the Intel machines (figure 2) 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[1]. We refer to this code in the graphs as 'nc ATLAS'. We can observe that in both cases ATLAS performance drops heavily. SB with

[1] 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

SML kernels obtain performance close to that of ATLAS on the Pentium 4 Xeon, similar to ATLAS on the Itanium2, and better than ATLAS on the Power4. For the latter we show the Mflops obtained by the vendor DGEMM routine which outperform both ATLAS and SB (figure 3). We can see that even highly optimized routines provided by the vendor can fail 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 [9]. These problems can be avoided using nonlinear array layouts.

3 Recursive+HM vs Iterative+SB: Conclusions

The results obtained with an iterative code working on a Square Block Format outperform the recursive code which uses a hypermatrix. Our results agree with those presented in [10]. We would like to implement a dense Cholesky factorization using an iterative approach and a Square Blocked Lower (or Upper) Packed Format [2]. We plan to do it straightaway.

References

1. 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
2. Gustavson, F.G.: New generalized data structures for matrices lead to a variety of high performance algorithms. In: PPAM. (2001) 418–436
3. 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
4. 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
5. Wise, D.S.: Representing matrices as quadtrees for parallel processors. Information Processing Letters **20** (1985) 195–199
6. 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)
7. 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)
8. 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)
9. 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
10. 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

inner kernel. The use of such hand-made inner kernels has improved significantly the overall performance of ATLAS subroutines on some platforms.