

Towards Peak Performance on Hierarchical SMP Memory Architectures – New Recursive Blocked Data Formats and BLAS

Gustavson F.* Jonsson I.† Kågström B.† Ling P.†

1 GEMM-based and Superscalar GEMM-based Level 3 BLAS

As shown in [6, 7] it is possible to develop a portable and high-performance level 3 BLAS library [2, 3] mainly relying on a highly optimized GEMM, the routine for the general matrix multiply and add operation, $C \leftarrow \beta C + \alpha AB$, where C, A and B are matrices and α, β are scalars. With suitable partitioning, all the other level 3 BLAS can be defined in terms of GEMM and a small amount of level 1 and level 2 computations [6].

Kågström, Ling and Van Loan developed portable high performance model implementations in Fortran 77 of the GEMM-based level 3 BLAS [7, 8]. This software can handle all four data types and is designed to be easy to install and use on different platforms. Each of the GEMM-based routines has a only few system-dependent parameters for tuning the library to the systems memory and processor characteristics.

To approach peak performance on state-of-the-art superscalar microprocessors it is necessary to attain extensive register reuse. In general, multiple calls to the level 1 and level 2 BLAS routines prohibit an efficient register reuse.

In 1998, Kågström and Ling announced the first version of the Superscalar GEMM-based level 3 BLAS. The main difference in the design is that all calls to underlying level 1 and level 2 BLAS have been removed. As before, the dominating part of all floating point operations take place in calls to DGEMM. The remaining computations that take care of triangular diagonal blocks are handled by “in-line” code optimized for efficient register reuse. The Superscalar GEMM-based Level 3 BLAS has been adopted by many organizations including the ATLAS [9] and PHiPAC [1] projects.

2 Techniques for complex memory hierarchies

In our on-going development of the Superscalar GEMM-based level 3 BLAS we are investigating different techniques to reduce delays in cache and memory accesses. We focus on ways to handle memory hierarchies with multiple levels of cache memories efficiently. Furthermore, we are using threads for developing parallel versions of our routines. We have also implemented an assembly version of DGEMM which we use as a development tool. This implementation is targeted toward IBM PowerPC based symmetric multiprocessors (SMP).

*IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, U.S.A., gustav@watson.ibm.com

†Department of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden. {isak,bokg,pol}@cs.umu.se

2.1 Prefetching and algorithmic preloading

Generally, when data does not reside in the top levels of a memory hierarchy (registers or L1 cache), significant delays occur when the data is referenced. In [4] two techniques to overcome these delays are described, prefetching and a technique called algorithmic preloading.

Prefetching is commonly used in compilers. Unfortunately, not as well as one would desire, especially for complex memory hierarchies. In our assembly programs we use a “touch” instruction in assembly language to perform prefetching. It is currently not possible to invoke this instruction explicitly from Fortran or C. This instruction brings data up into L1 cache, essentially without disturbing the ongoing computations and data references.

Algorithmic preloading is a technique to reference data that is going to be used in the innermost loop, one or more iterations ahead of its use. Together with suitable unrolling of the loops, this technique can enable the innermost loop to run at close to peak performance. This technique can be implemented in high level languages such as Fortran or C and is also used successfully in some compilers. One of our goals is to find strong evidence for introducing support for these techniques in future compilers.

2.2 Hierarchical blocking

The purpose of the blocking structure of the currently released versions of the GEMM-based level 3 BLAS is to reuse the most frequently referenced matrix blocks in L1 cache as much as possible. However, for machines with several levels of cache memories a need for more advanced blocking strategies arises. One way to handle the memory hierarchy explicitly is to reorganize loops such that each loop set matches a specific level of the memory hierarchy. However, this is a tedious work and requires a lot of knowledge about the architecture characteristics. Another way is to use recursive blocking, formulated as a recursive divide and conquer algorithm. Hereby, we can automatically obtain an arbitrary number of blocking levels.

2.3 Recursive blocked data formats

In order to further utilize recursive blocked algorithms, recursive blocked data formats are introduced. In [5] we describe recursive blocked data formats and recursive blocked algorithms for level 3 BLAS. These recursive formats combine the efficiency of sub-blocks with fixed leading dimensions and the data reuse properties of the recursive algorithms. The fixed sized sub-blocks are tuned to match the size of the L1 cache, or a fixed fraction of it in order to work with several sub-blocks simultaneously.

The recursive storage of a rectangle is done by always dividing the largest dimension of the matrix in two halves, ensuring that the sub-blocks within each half are stored close to each other. This way, there will always be sets of data with enough data locality to match the cache sizes, at some level in the recursion. Together with the recursive blocked algorithm, this gives both temporal and spatial locality, a must for high performance computations on architectures with complex cache hierarchies. This leads to a blocking that is variable, automatic and “squarish”.

2.4 Recursive blocked GEMM

Our current implementation of DGEMM consists of three parts. Firstly, the scheduler which distributes the problem to the threads. Secondly, the recursive algorithm, which builds a *batch list* of the block multiplications that are needed to solve the thread-local problem.

$M = N = K$	ESSL-SMP		Recursive BLAS	
	GEMM	TRSM	GEMM	TRSM
208	1015	950	1664	840
320	1270	1071	1801	1156
400	1320	1146	1815	1454
512	1078	1036	1812	1552
640	1216	1235	1862	1638
800	1192	1286	1852	1775
1024	1020	1127	1837	1760
1280	1120	1170	1860	1833
1600	1156	1216	1830	1847
2048	991	1246	1806	1814

TABLE 1

Comparison of IBM’s ESSL-SMP library and the Recursive Superscalar GEMM-based Level 3 BLAS. Measurements are in Mflops/s, performed on a 4-way IBM PowerPC 604e, 332 MHz node. For all runs, the leading dimensions are equal to the matrix sizes. Time for data restructuring to recursive format is not included.

Thirdly, the computational kernel which perform GEMM operations on the subproblems in the batch list. The computational kernel is designed to hold six fixed size sub-blocks in L1 cache simultaneously. Three of them are involved in the current computation while the other three blocks are being prefetched using the “touch” instruction. When the current block computation is completed, a new computation starts involving the blocks just prefetched, and three new blocks are prefetched for the subsequent computations, and so on.

2.5 Parallelization of recursive blocked level 3 BLAS

Our distribution is dynamic, yet efficient. A virtual recursion tree is maintained throughout the execution. Each idle thread will read the number of operations left to be performed from the root node. This number is then divided by a number greater than the number of threads. The thread will then search in the tree for a subtree with less operations than the calculated fraction. This is similar to the OpenMP `SCHEDULE(GUIDED)` directive, but the tree algorithm divides the problem in several dimensions to keep the problem squarish. This way of scheduling enables good load balancing on non-dedicated machines, and the algorithm scales well. One reason is that the recursive task tree automatically keeps data references local to each thread. For TRSM, additional information is kept in the tree to maintain the critical path of execution.

3 Performance results

We show significant performance improvements using these techniques on SMP machines with complex memory hierarchies. We often see up to 50% better performance on a multi-processor node compared to other high performance implementations, see Table 1. The recursive algorithms and the recursive data storage format provide for exceptionally high levels of locality in the references, which leads to excellent scalability.

4 Acknowledgements

Development and measurements were performed at the facilities at HPC2N, Umeå, Sweden and UNI-C, Copenhagen, Denmark.

References

- [1] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel. Optimizing matrix multiply using PHiPAC: A portable, high performance, ANSI C coding methodology. In *Proceedings of the 11th International Conference on Supercomputing (ICS-97)*, pages 340–347, New York, July7–11 1997. ACM Press.
- [2] J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990.
- [3] J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling. Algorithm 679: A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs. *ACM Trans. Math. Softw.*, 16(1):18–28, March 1990.
- [4] F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling. Superscalar GEMM-based Level 3 BLAS – The On-going Evolution of a Portable and High-Performance Library. *To appear in Proceedings from PARA98*, Springer Verlag, 1998.
- [5] F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling. Recursive Blocked Data Formats and BLAS’s for Dense Linear Algebra Algorithms. *To appear in Proceedings from PARA98*, Springer Verlag, 1998.
- [6] B. Kågström and C. Van Loan. GEMM-Based Level-3 BLAS. Technical Report CTC91TR47, Department of Computer Science, Cornell University, Dec. 1989.
- [7] B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Software*, 1997. To appear.
- [8] B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: Portability and optimization issues. *ACM Trans. Math. Software*, 1997. To appear.
- [9] R. C. Whaley and J. J. Dongarra. Automatically tuned linear algebra software. Tech. Report TN 37996-1301, Computer Science Dept., Univ. of Tennessee, 1997.