High-Performance Implementation of the Level-3 BLAS

Kazushige Goto
Texas Advanced Computing Center
The University of Texas at Austin
Austin, TX 78712
kgoto@tacc.utexas.edu

Robert van de Geijn
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
rvdg@cs.utexas.edu

FLAME Working Note #20

May 6, 2006

Abstract

A simple but highly effective approach for transforming high-performance implementations on cache-based architectures of matrix-matrix multiplication into implementations of other commonly used matrix-matrix computations (the level-3 BLAS) is presented. Exceptional performance is demonstrated on various architectures.

1 Introduction

Attaining high performance for matrix-matrix operations such as *symmetric matrix-matrix multiply* (SYMM), *symmetric rank-k update* (SYRK), *symmetric rank-2k update* (SYR2K), *triangular matrix-matrix multiply* (TRMM), and *triangular solve with multiple right-hand sides* (TRSM) by casting the bulk of computation in terms of a *general matrix-matrix multiply* (GEMM) has become a generally accepted practice [7]. Variants on this theme include loop-based algorithms and recursive algorithms, as well as hybrids that incorporate both of these [4]. In this paper we show that better performance can be attained by specializing a high-performance GEMM kernel [5] so that it computes the desired operation. For the busy reader the results are previewed in Fig. 1.

This paper is organized as follows: In Section 2 we review the basic techniques behind a high-performance matrix-matrix multiplication implementation. More traditional techniques for implementing level-3 BLAS are reviewed in Section 3. These alternative techniques are then used to obtain highly optimized implementations of SYMM, SYRK, SYR2K, TRMM, and TRSM, in Sections 4–7. Concluding remarks are given in the final section.

2 High-Performance Implementation of Matrix-Matrix Multiplication

To understand how to convert a high-performance matrix-matrix multiplication (GEMM) implementation into a fast implementation for one of the other matrix-matrix operations that are part of the level-3 Basic Linear Algebra Subprograms (BLAS) [3], one has to first review the state-of-the-art of high-performance implementation of the GEMM operation. In this section we give a minimal description, referring the interested reader to [5].

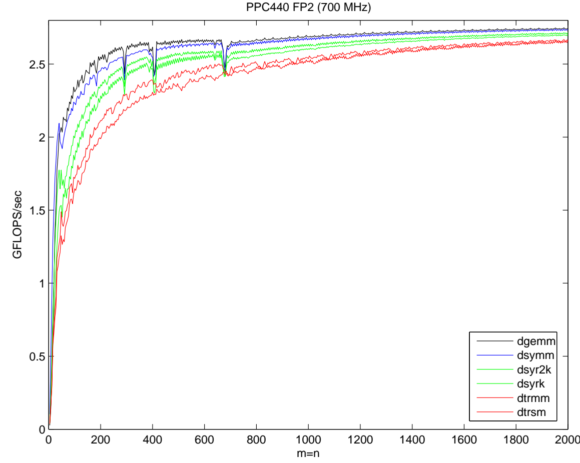


Figure 1: Performance of all level-3 BLAS on the IBM PPC440 FP2 (700 MHz). (Curves in the figure appear, from top to bottom, in the order indicated in the legend.)

Figure 2: Left: Partitioning of A and B. Right: Blocking for one individual panel-panel multiplication (GEPP) operation, $C := A_j B_j + C$.

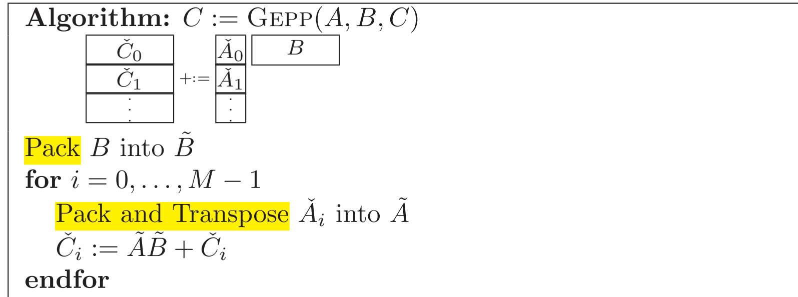


Figure 3: Outline of optimized implementation of GEPP.

Consider the computation $C := AB + C$, where C, A, and B are $m \times n$, $m \times k$, and $k \times n$ matrices, respectively. Assume for simplicity that $m = b_m M$, $n = b_n N$, and $k = b_k K$, where M, N, K, b_m, b_n, and b_k are all integers. Partition as in Fig. 2(left):

$$A \rightarrow \left(\begin{array}{c|c|c|c} A_0 & A_1 & \cdots & A_{K-1} \end{array} \right) \quad \text{and} \quad B \rightarrow \left(\begin{array}{c} \check{B}_0 \\ \hline \check{B}_1 \\ \hline \vdots \\ \hline \check{B}_{K-1} \end{array} \right),$$

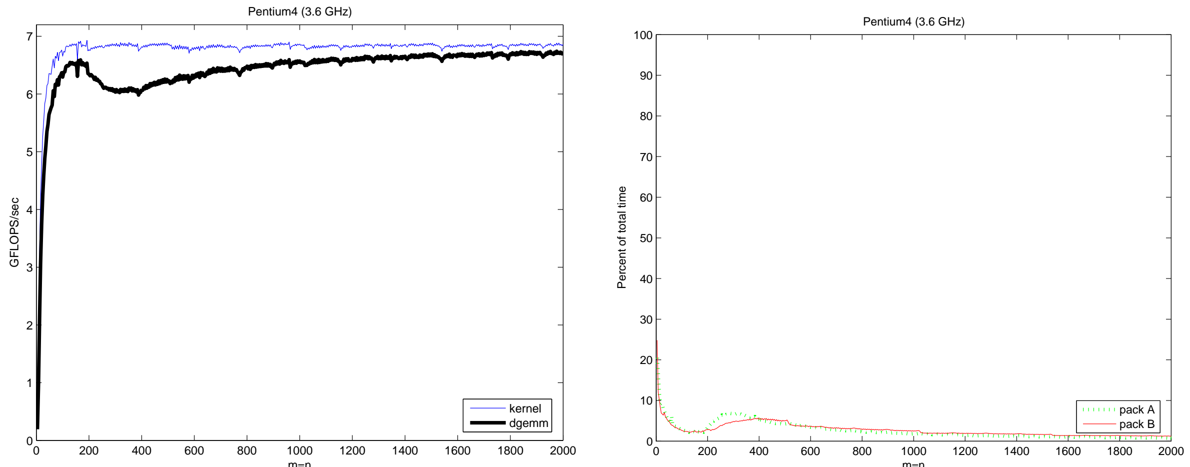


Figure 4: Performance of GEMM on Pentium4 (3.6 GHz). Left: The performance of the GEBP kernel routine and GEMM is given by the curves labeled "Kernel" and"dgemm". Right: The curves labeled "Pack A" and "Pack B" indicate the percent of total time spent in each of these operations.

where A_p and $\check{B}_p$ contain b_k columns and rows, respectively[1]. Then

$$C := A_0\check{B}_0 + A_1\check{B}_1 + \cdots + A_{K-1}\check{B}_{K-1} + C.$$

A typical high-performance implementation of GEMM will focus on making each update $C := A_p\check{B}_p + C$, which we will call a *panel-panel multiplication* (GEPP), as fast as possible. The overall performance of GEMM is essentially equal to that of each individual GEPP with panel width equal to an optimal size b_k.

Figure 3 gives a high-performance algorithm for the GEPP operation, $C := AB + C$, where the "k" dimension is b_k. The algorithm requires three highly optimized components:

- **Pack B:** A routine for packing B into a contiguous buffer. On some architectures this routine may also reorganize the data for specialized instructions used by the GEBP kernel routine described below.
- **Pack and transpose $\check{A}_i$:** A routine for packing $\check{A}_i$ into a contiguous buffer. Often this routine also transposes the matrix to improve the order in which it is accessed by the GEBP kernel routine.
- GEBP **kernel routine:** This routine computes $\check{C}_i := \tilde{A}\tilde{B} + \check{C}_i$ using the packed buffers. GEBP stands for General block-times-panel multiply.

On current architectures the size of $\check{A}_i$ is chosen to fill about half of the L2 cache (or the memory addressable by the TLB), as explained in [5]. Considerable effort is required to tune each of these components, especially the GEBP kernel routine. In subsequent sections we will show how other level-3 BLAS can be implemented in such a way that this effort can be amortized.

In Fig. 4 the performance and overhead of the various kernels is reported for the high-performance implementation of DGEMM (double-precision GEMM) from [5]. It is this implementation upon which the remainder of this paper is based. In Fig. 5 we compare the performance of this DGEMM implementation with those of the vendor implementations (MKL and ESSL) and ATLAS.

Throughout the paper performance is presented for double precision (64-bit) computation of the target operation on a number of architectures:

[1]The ˇ is used to indicate a partitioning by rows in this paper.

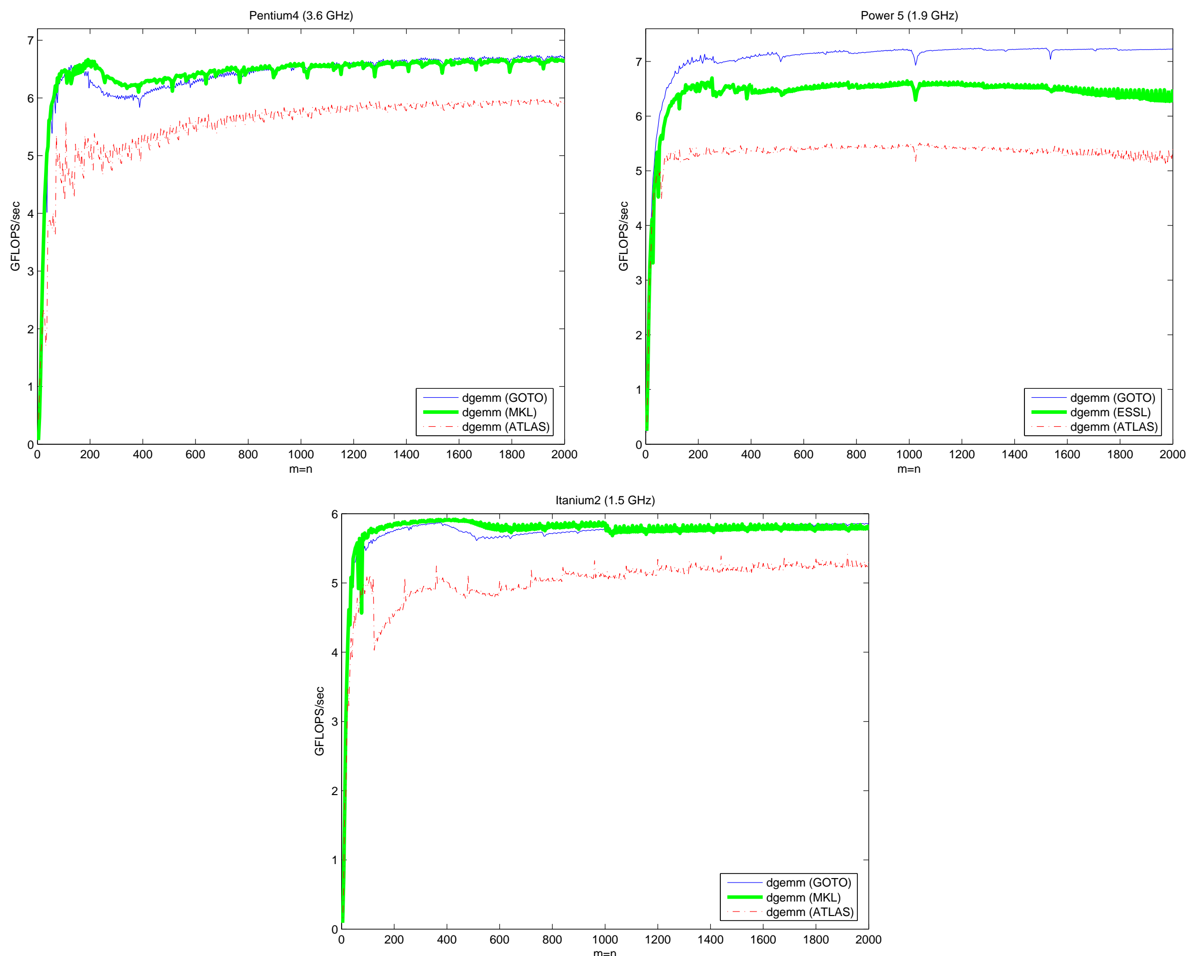


Figure 5: Performance of various implementations of DGEMM.

Architecture	Clock speed	Peak Performance	blocking size		Vendor library
			b_m	b_k	
Intel (R) Pentium4 (R)	3.6 GHz	7.2 GFLOPS/sec	768	192	MKL 8.0.1
IBM Power 5	1.9 GHz	7.6 GFLOPS/sec	256	256	ESSL 4.2.0
IBM PPC440 FP2	700 MHz	2.8 GFLOPS/sec	128	3072	not available to us
Intel (R) Itanium2 (R)	1.5 GHz	6 GFLOPS/sec	128	1024	MKL 8.0.1

We also compare against ATLAS 3.7.11, a public-domain implementation of the BLAS [8], except for the Itanium2 system, on which ATLAS 3.7.8 attained better performance. In our graphs that report the rate of execution (GFLOPS/sec) the top line always represents the theoretical peak of the processor. The blocking sizes b_m and b_k are as indicated in the above table.

Remark 2.1 *Key insights from [5] are that (1) the submatrix $\hat{A}_i$ is typically non-square, (2) the cost of packing $\hat{A}_i$ is significant, which means that the column dimension of B should be large, and (3) the cost of packing B is significant and should therefore be amortized over as many blocks of A as possible and repacking should be avoided.*

3 Traditional Approaches for Implementing the Level-3 BLAS

We use the symmetric matrix-matrix multiplication (SYMM), $C := AB + C$ where A is symmetric, as an example of how traditional approaches to implementing the Level-3 BLAS proceed. We will assume that only the lower triangular part of A is stored (in the lower triangular part of the array that stores A).

3.1 Loop-based approach

In Fig. 6(left) we show a typical computation SYMM as a loop that traverses the matrices a block of rows and/or columns at a time. We believe the notation used in that figure, which has been developed as part of the FLAME project, to be sufficiently intuitive not to require further explanation [2, 6]. Notice that the bulk of the computation is cast in terms of GEPP. So far the size of A_{11} in each iteration is chosen to equal the optimal b_k discussed in Section 2 so that the GEPP updates $A_{01}^T B_0$ and $A_{21} B_2$ attain near-optimal performance. Since typically b_k is small relative to m (the dimension of A) the update $C_1 := A_{11} B_1 + C_1$, which we will call a *symmetric block-matrix multiplication* (SYBM) requires relatively few operations. Letting n equal the column dimension of C, the total operation count of a SYMM operation is $2m^2 n$ floating point operations (flops). A total of $2(m/b_k)b_k^2 n = 2mb_k n$ flops are in the SYBM computations and $2(m - b_k)mn$ in the GEPP operations. Even if the performance attained by the SYBM operations is less than that of a GEPP, the overall performance degrades only moderately if $m >> b_k$. In our case, SYBM is implemented by copying A_{11} into a temporary matrix, making it "general" by copying the lower triangular part to the upper triangular part, and calling GEMM.

There are two sources of complications and/or inefficiencies in this approach:

- The three operations ($C_0 := A_{10}^T B_1 + C_0$, $C_1 := A_{11} B_1 + C_1$, and $C_2 := A_{21} B_1 + C_2$) are typically treated as three totally separate operations, meaning that B_1 must be packed redundantly for each of the three operations. In Fig. 4 it is shown that the packing of B_1 is a source of overhead for an individual GEPP operation that cannot be neglected.

- Since the shape and size of of C_1 and the $b_k \times b_k$ block of A_{11} in the SYBM operation do not match those of the corresponding submatrices in the GEPP operation, optimization of the SYBM operation is not a matter of making minor modifications to the kernel GEBP operation. A high-performance implementation would require a redesign of this kernel.

These problems are most noticeable when A is relatively small.

3.2 Recursive algorithms

Partition

$$C \rightarrow \left(\frac{C_T}{C_B} \right), \quad A \rightarrow \left(\begin{array}{c|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array} \right), \quad \text{and} \quad B \rightarrow \left(\frac{B_T}{B_B} \right),$$

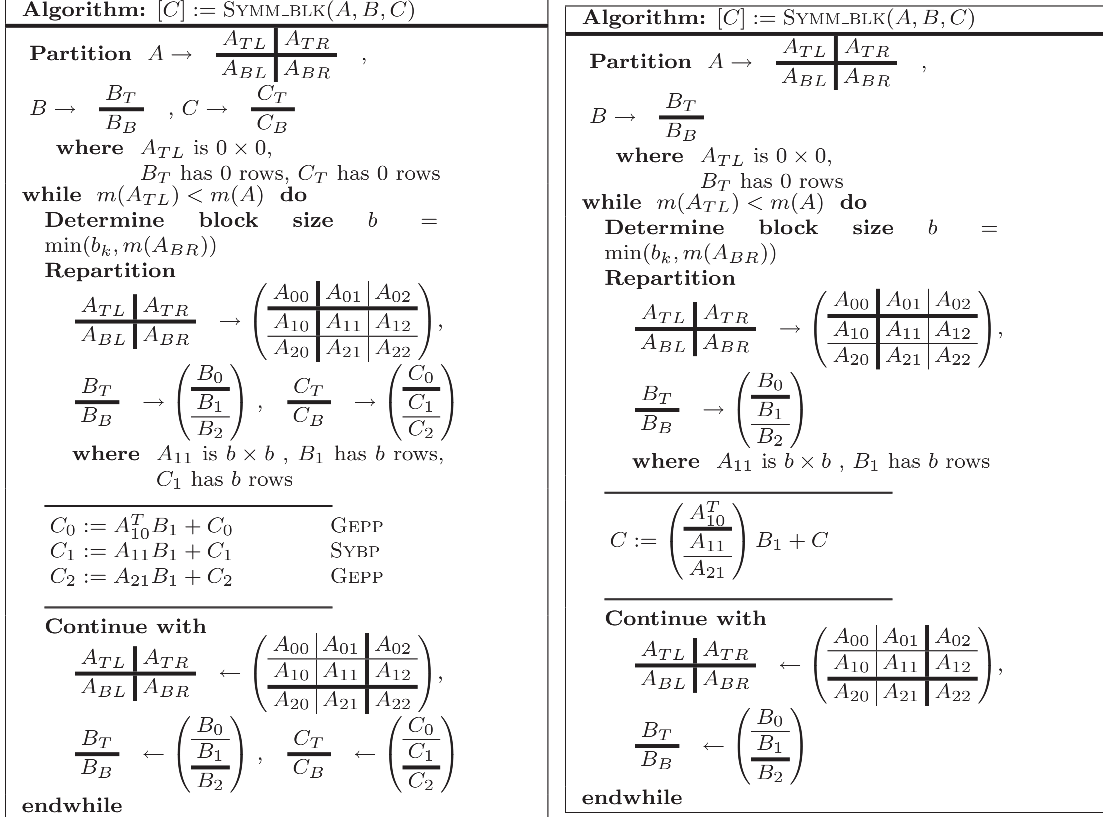


Figure 6: Algorithms for computing SYMM. Left: Typical loop-based algorithm. Right: Casting in terms of a single GEPP.

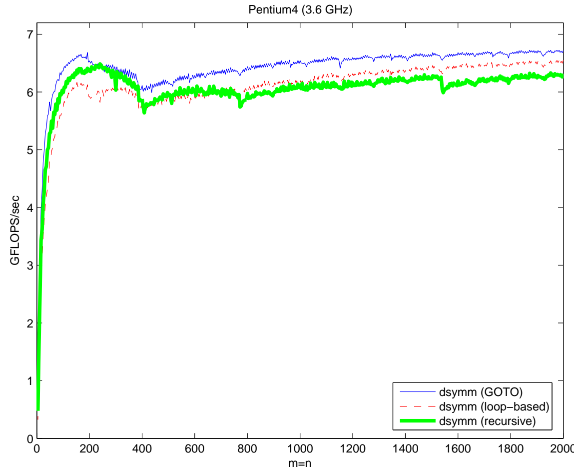


Figure 7: Performance of different implementations of Symm: The algorithm put forth later in this paper, a loop-based algorithm, and a recursive algorithm.

where A_{TL} is a $k \times k$ matrix and B_T and C_T are $k \times n$. Then $C := AB + C$ yields

$$\left(\frac{C_T}{C_B}\right) = \left(\begin{array}{c|c} A_{TL} & A_{BL}^T \\ \hline A_{BL} & A_{BR} \end{array}\right) \left(\frac{B_T}{B_B}\right) + \left(\frac{C_T}{C_B}\right) = \left(\frac{A_{TL}B_T + A_{BL}^T B_B + C_T}{A_{BL}B_T + A_{BR}B_B + C_B}\right).$$

The terms $A_{TL}B_T + C_T$ and $A_{BR}B_B + C_B$ can be achieved by recursive calls to Symm. This time the bulk of the computation is cast in terms of Gemm: $A_{BL}^T B_B$ and $A_{BL}B_T$. Typically the recursion stops when matrix A is relatively small, at which point A may be copied into a general matrix, after which Gemm can be employed for this small problem.

There are two sources of complications and/or inefficiencies in this recursive approach:

- The same panels of B will be packed multiple times as part of the individual calls to Gemm, which itself is cast in terms of Gepp operations.
- Unless care is taken the recursion will not create subproblems of sizes that are integer multiples of b_k, which causes the Gemm operations to attain less than optimal performance.

3.3 Performance

The performance of the two traditional approaches described in this section are reported in Fig. 7. In that graph we also report the performance attained by the approach delineated later in this paper. The block size for the loop-based algorithm was taken to equal 192 while the recursion terminated when the size of the subproblem was less then or equal to 192.

4 Symm

The alternative approach to implementing Symm professed by this paper is ridiculously simple to describe: Execute exactly the same algorithm as was employed for Gepp by modifying the routine that copies sub-

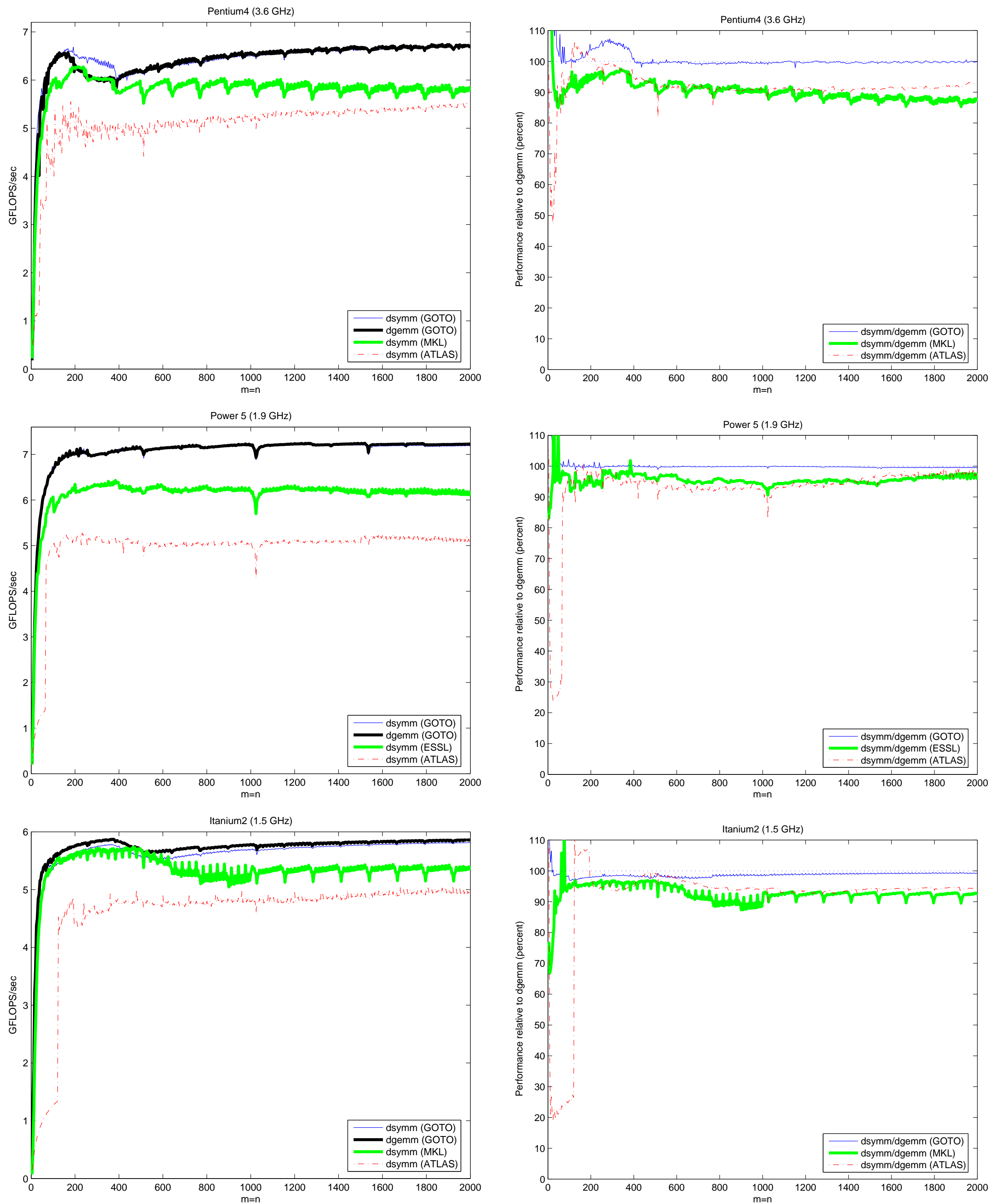
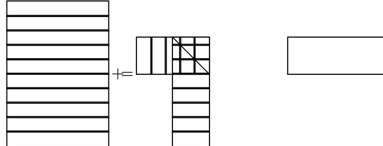


Figure 8: Performance of Symm relative to Gemm. (For the Power5 architecture the line for `dsymm` and `dgemm` are almost coincident. The `dsymm` line lies ever so slightly above that of `dgemm`.)

matrices of matrix A into packed form to accommodate the symmetric nature of matrix A.

To understand this fully, first consider the algorithm in Fig. 6(right). Notice that *if* A_{10}^T, A_{11}, and A_{21} are copied into a single panel of columns of width b_k, then the GEPP algorithm in Fig. 3 can be executed. This approach is inefficient in the sense that these submatrices are first copied and then subsequently packed as part the GEPP algorithm. This suggests that instead of first copying the three parts into a single column panel, the GEPP algorithm should be modified so that the copying is done as needed, a block of $b_m \times b_k$ at a time, as illustrated by



While simple, the method has a number of immediate benefits:

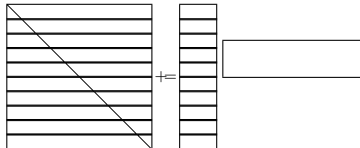
- The packing of the block of rows of B is amortized over all computation with A_{10}, A_{11}, and A_{21}.
- The routine for packing submatrices of A needs only be modified slightly.
- The exact same kernel GEBP routine as for implementing GEPP can be used.

Interestingly enough, the approach yields performance that often *exceeds* that of GEMM, as shown in Fig. 8.

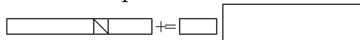
5 SYRK and SYR2K

Next we discuss the symmetric rank-k (SYRK) and symmetric rank-2k (SYR2K) updates: $C := AA^T + C$ and $C := AB^T + BA^T + C$, where C is an $m \times m$ symmetric matrix and A and B are $m \times k$. We will assume that only the lower triangular part of C is stored.

Let us focus on SYRK first. As for the GEMM and SYMM operations it is important to understand how one panel-panel multiply is optimized: the case where A contains b_k columns ($k = b_k$) Mimicking the GEPP implementation yields



The idea now is that the computation of each row panel of C is modified to take advantage of the fact that only the part that lies at or below the diagonal needs to be updated. One straightforward way to accomplish this is to break each such row panel into three parts:



The kernel GEBP routine can be used to update the left part. A special kernel updates the lower triangular block on the diagonal, and the right part is not updated at all.

The implementation of SYR2K is a simple extension of this, where a slight optimization is that each row panel of C is updated with the appropriate part of both AB^T and BA^T since this keeps the panel of C in the L3 cache, if present. Again, the performance is impressive, as illustrated in Figs. 9 and 10.

6 TRMM

We will examine the specific case of the triangular matrix-matrix multiply (TRMM) $B := LB$, where L is a lower triangular $m \times m$ matrix and B is $m \times n$.

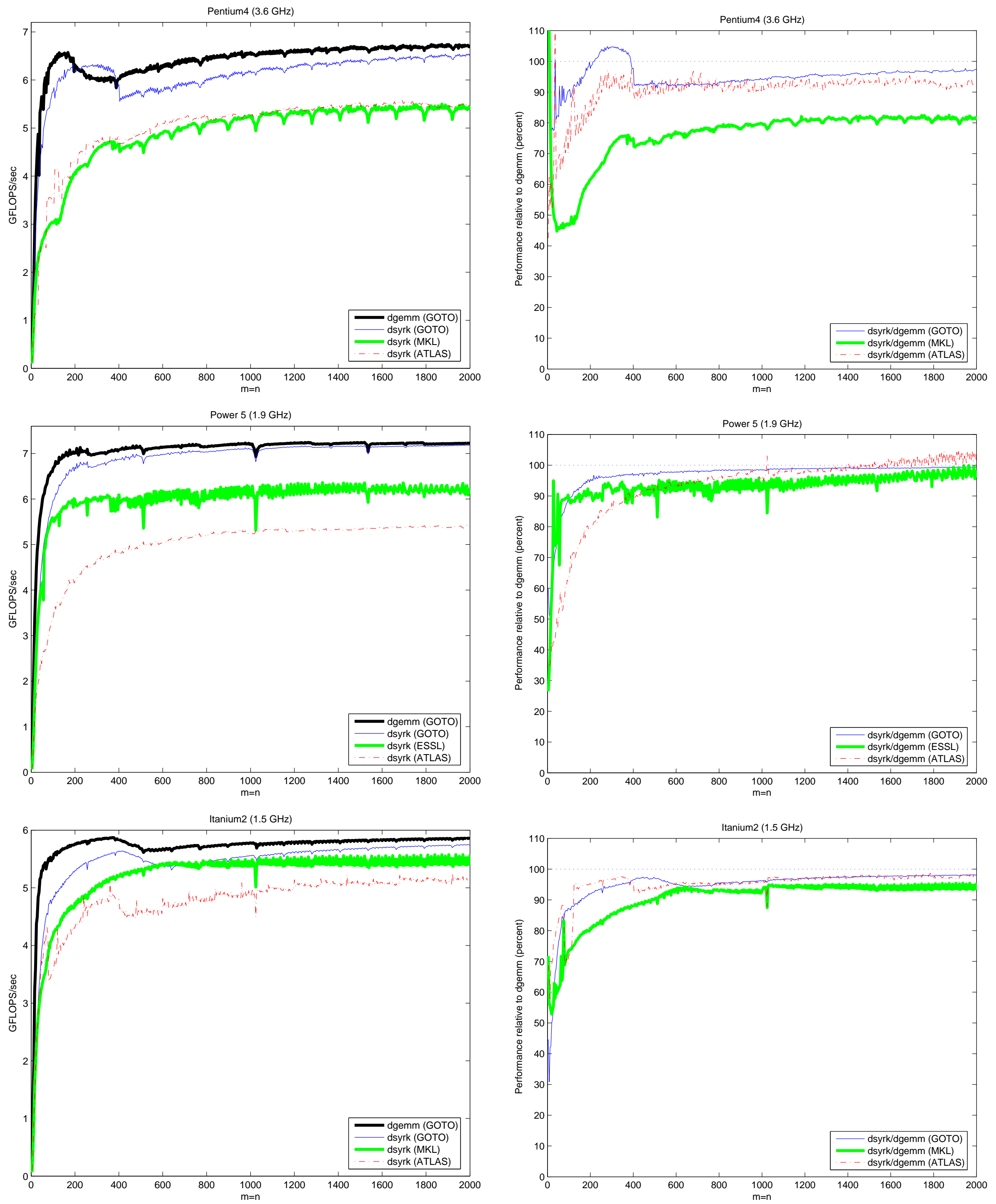


Figure 9: Performance of Syrk relative to Gemm.

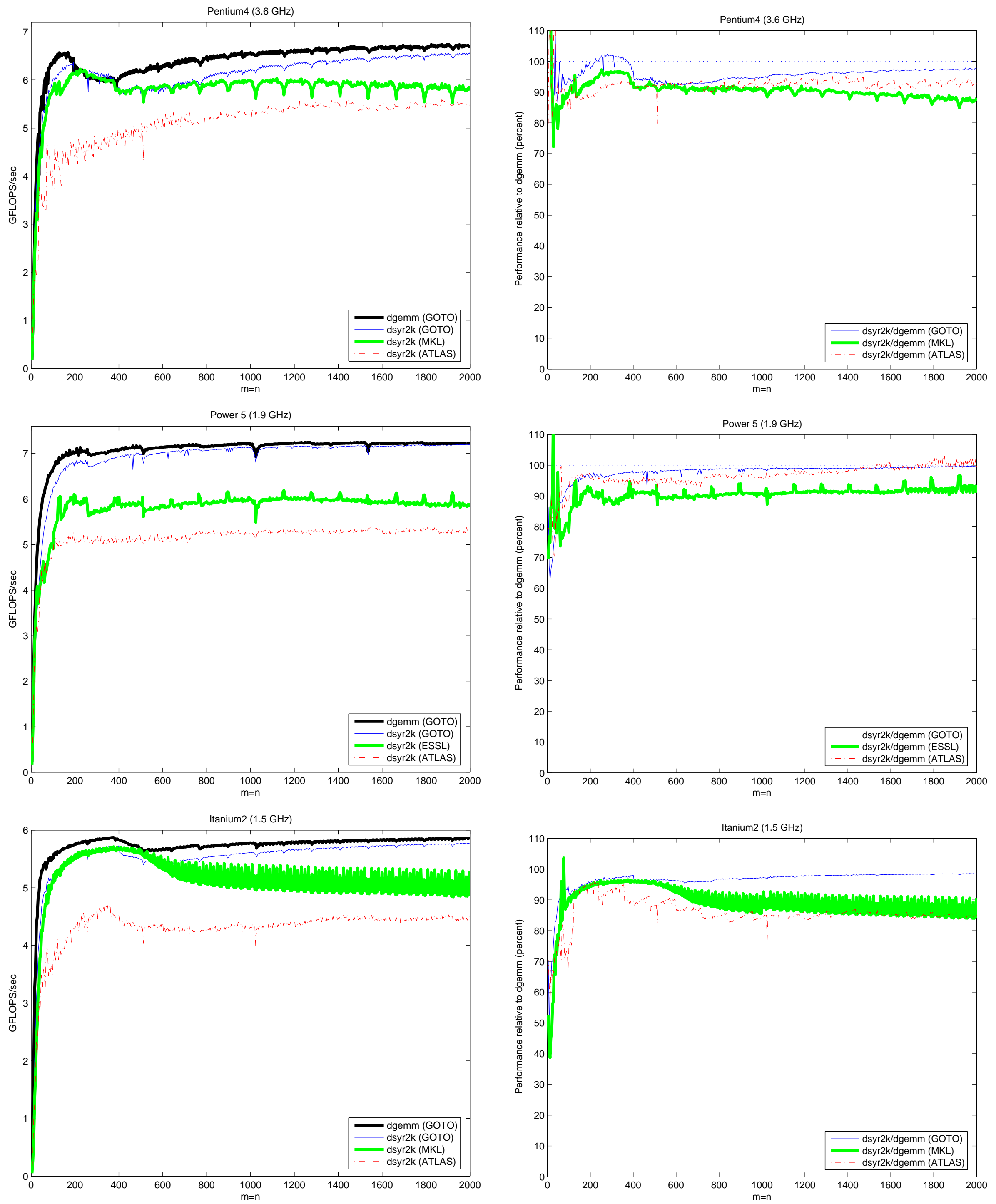


Figure 10: Performance of SYR2K relative to GEMM.

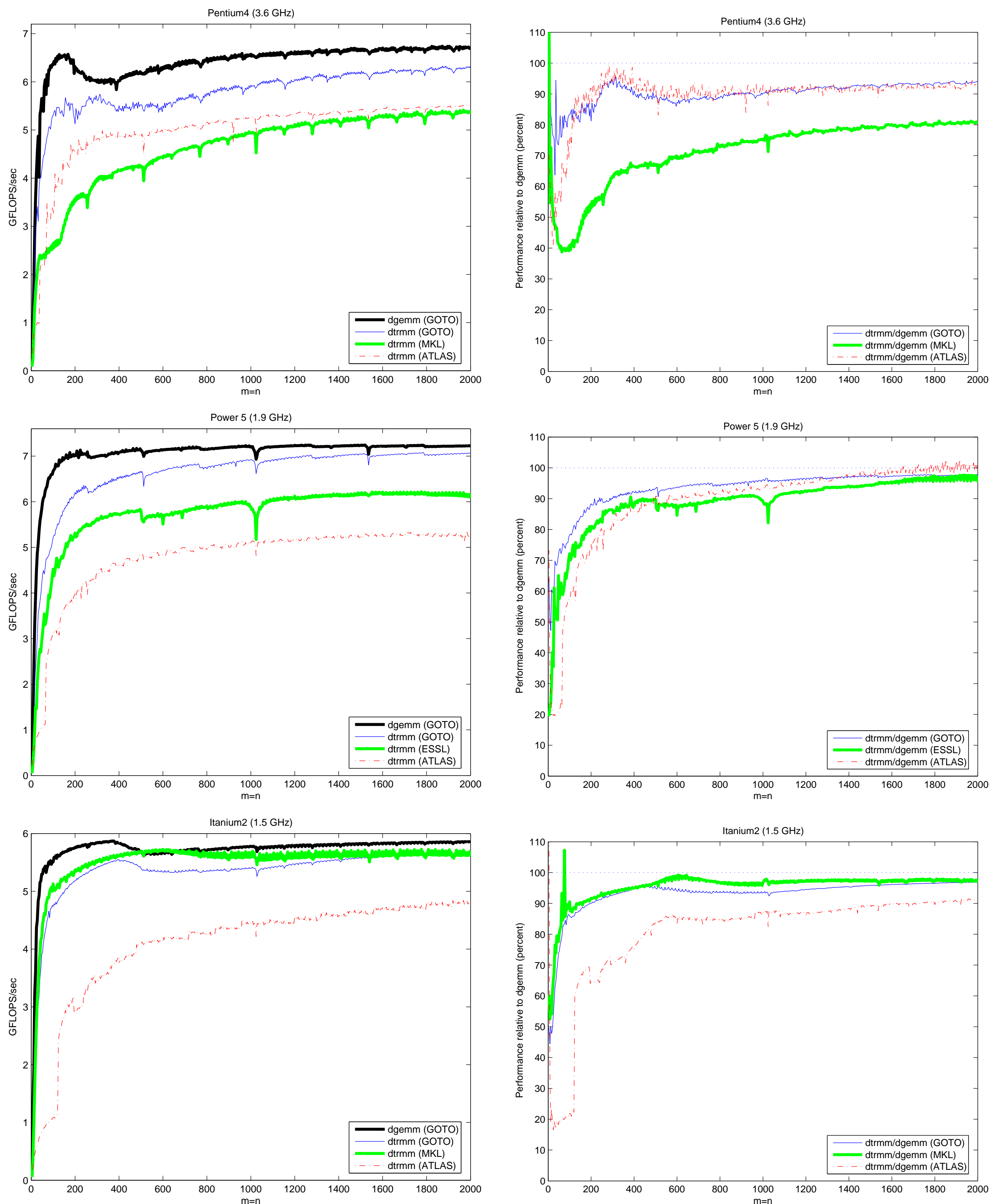
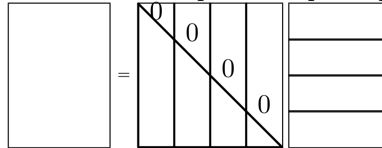
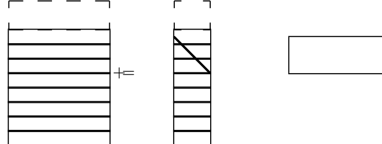


Figure 11: Performance of Trmm relative to Gemm.

Again, this operation can be cast in terms of a sequence of panel-panel multiplies:



An examination of how the GEPP algorithm can be modified for the special needs of TRMM yields



One notices that again most of the computation can be cast in terms of the kernel GEBP routine, except for the computation with the blocks that contain part of the diagonal. There are a number of ways of dealing with those special blocks:

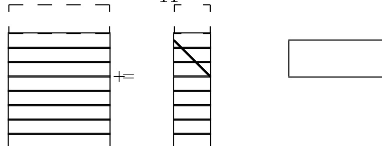
- As the block is packed and transposed the elements in from the upper triangular part can be set to zero after which the kernel GEBP routine can be used without modification. The advantage is that only the packing routine needs to be modified slightly. The disadvantage is that considerable computation is performed with elements that equal zero.
- Modify the kernel GEBP routine so that it does not compute with elements that lie above the diagonal. Conceptually this means changing a few loop-bounds. In practice there is loop-unrolling that is incorporated in the kernel GEBP routine that makes this somewhat more complex. One possibility for overcoming this while making only slight changes to the kernel routine is to set those elements that lie in a region covered by the loop-unrolling to zero and to compute with those but not other elements that lie above the diagonal. This can then be accomplished by only modifying loop-bounds in the kernel routine without disturbing code related to loop-unrolling.

We favor the second solution in our implementations. The performance of the TRMM routine is demonstrated in Fig. 11.

7 TRSM

We will examine the specific case of the triangular solve with multiple right-hand sides (TRSM) $B := L^{-1}B$, where L is a lower triangular $m \times m$ matrix and B is $m \times n$. An algorithm that casts most computation in terms of GEPP is given in Fig. 13.

Let us examine the combined updates $B_1 := L_{11}^{-1} B_1$ and $B_2 := B_2 - L_{21} B_1$:



It is in how to deal with the blocks that contain the diagonal that complications occur. For these blocks

- B_1 will have been copied into a packed array $\tilde{B}$.
- The current row panel will have b_m rows. Let us denote this row panel by the matrix C.

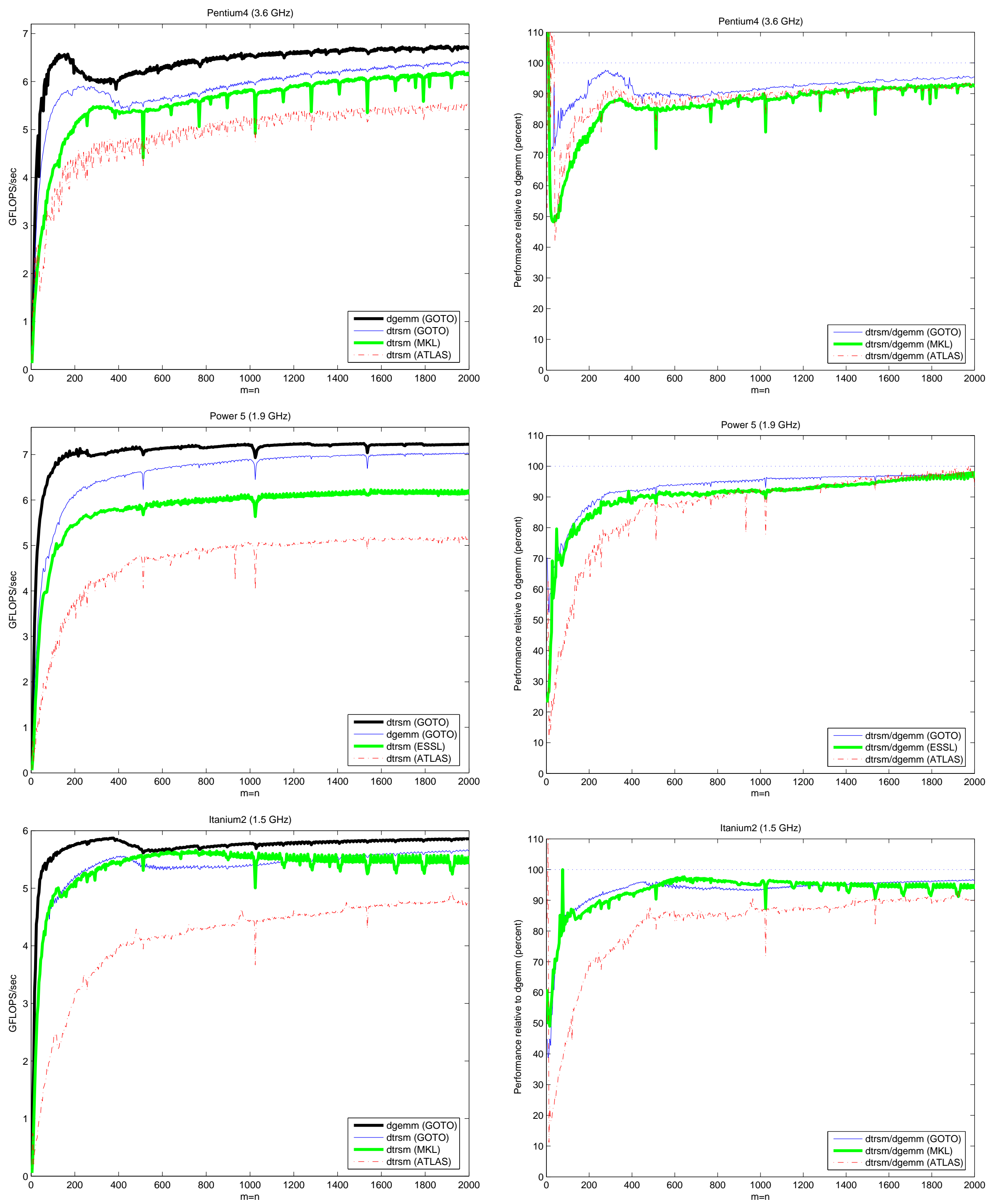


Figure 12: Performance of Trsm relative to Gemm.

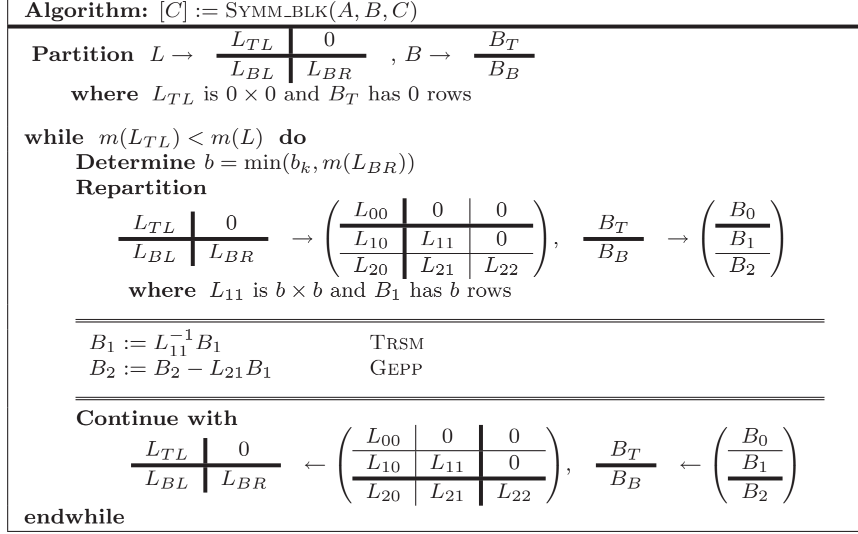
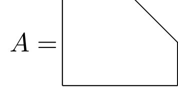
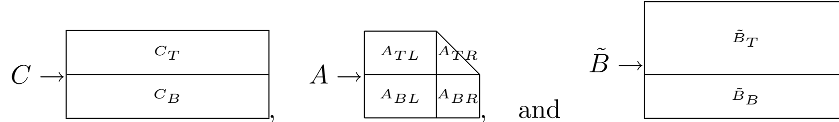


Figure 13: Algorithm for computing TRSM that casts most computation in terms of GEPP.

- A typical block of L_{11}, A, that contains the diagonal will have the shape



- Partitioning C, A, and $\tilde{B}$ as



  the operation to be performed can then given by

  - $C_T := A_{TR}^{-1}(C_B - A_{TL} C_T)$.
    The data in C_T coincides with the part of B that was copied into $\tilde{B}$. Thus the result in C_T needs also be updated in the corresponding part of $\tilde{B}$.
  - $C_B := C_B - A_{BL} \tilde{B}_T - A_{BR} \tilde{B}_B$.
    Again, the data in C_B coincides with the part of B that was copied into $\tilde{B}$. Thus the result in C_B needs also be updated in the corresponding part of $\tilde{B}$.

Clearly, the kernel that implements this requires considerable care and cannot be simply derived from the kernel GEBP routine. Performance of our implementation is reported in Fig. 12.

8 Conclusion

In this paper, we have presented a simple yet highly effective approach to implementing level-3 BLAS routines by modifying the currently most effective technique for implementing matrix-matrix multiplication.

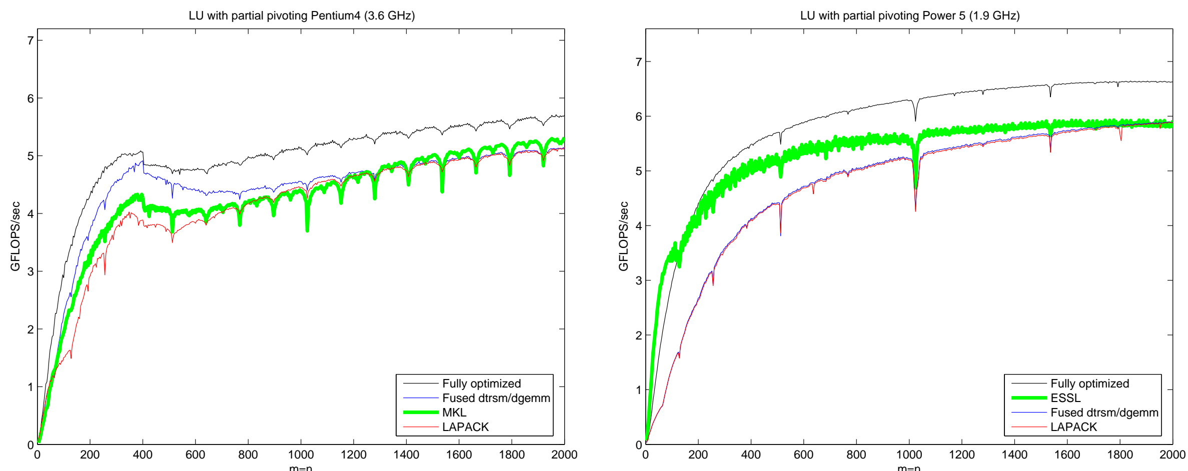


Figure 14: Performance of LU factorization with partial pivoting for different levels of optimization. For the graph on the right, the "Fused dtrsm/dgemm" curve is almost indistinguishable from the "Reference" curve.

The methodology inherently avoids unnecessary recopying of data into packed format. It suggests that routines like those that pack and kernel routines be exposed as building blocks for libraries.

The performance comparison with the MKL library on the Itanium2 architecture may appear to present a counterexample to the techniques advocated by this paper, since for some operations the MKL implementation outperforms our implementations. We note that their implementations require substantially more effort than those supported by our work.

There are a number of other situations in which exposing these building blocks will become advantageous if not necessary.

- A typical LU factorization (with or without pivoting) performs a Trsm operation with a matrix that subsequently becomes an operand in a Gepp. This could allow a new packing of that data to be avoided if the packed array used in the implementation of the Trsm is saved. The benefits are illustrated in Fig. 14. The curve labeled "LAPACK" corresponds to the LAPACK implementation of LU with partial pivoting [1], with an optimized blocking size of 64 on both the Pentium4 and the Power 5 architectures. This implementation makes separate calls to dtrsm and dgemm, requiring the "B" matrix to be repacked. The curve labeled "Fused dtrsm/dgemm" fuses the dtrsm and dgemm calls so that the packed "B" matrix can be reused, while keeping the blocking size the same. Notice that the improvement is worthwhile on the Pentium4 but not on the Power 5.

  A fully optimized implementation of LU with partial pivoting, which optimizes the swapping of multiple rows (dlaswp), adds recursion to the "factorization of the current panel", and increases the blocking size to 192 and 256 on the Pentium4 and Power 5 architectures, respectively, is labeled by "Fully optimized".

- Implementation of level-3 BLAS on SMP or multi-core platforms could easily incur redundant packing operations by the different threads. Exposing the building blocks could avoid this, improving performance considerably.

We believe this suggests that the standardization of interfaces to such building blocks is in order.

Additional information

Implementations of the described techniques are available for essentially all current architectures. Libraries can be obtained from `http://www.tacc.utexas.edu/resources/software/`.

Acknowledgments

This research was sponsored in part by NSF Grant CCF-0540926. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The Itanium2 server used in this research was donation by Hewlett-Packard. Access to the IBM PPC440 FP2 was arranged by Lawrence Livermore National Laboratory. The Texas Advanced Computer Center provided access to the other architectures. We would like to thank Dr. John Gunnels and members of the FLAME team for their comments on an earlier draft of this paper.

References

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide (third ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[2] Paolo Bientinesi and Robert van de Geijn. Representing dense linear algebra algorithms: A farewell to indices. FLAPACK Working Note #17 TR-2006-10, The University of Texas at Austin, Department of Computer Sciences, 2006.

[3] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[4] Erik Elmroth, Fred Gustavson, Isak Jonsson, and Bo Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):3–45, 2004.

[5] Kazushige Goto and Robert van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Soft.* in review.

[6] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.

[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. Soft.*, 24(3):268–302, 1998.

[8] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. In *Proceedings of SC'98*, 1998.