Cache Oblivious Dense and Sparse Matrix Multiplication Based on Peano Curves

Michael Bader and Alexander Heinecke

Institut für Informatik, Technische Universitüt München, Germany

Abstract. *Cache oblivious* algorithms are designed to benefit from any existing cache hierarchy—regardless of cache size or architecture. In matrix computations, cache-oblivious approaches are usually obtained from block-recursive approaches (see [5, 4], e.g.). In this article, we extent an existing cache-oblivious approach for matrix operations, which is based on Peano space-filling curves, for multiplication of sparse and dense matrices (sparse-dense, dense-sparse, sparse-sparse). We present the respective block-oriented data structure to store sparse matrices, and give first performance results on multicore platforms.

1 Block-recursive Matrix Multiplication Based on Peano Curves

In previous works [1, 2], we introduced a cache-oblivious algorithm for matrix multiplication that uses Peano space-filling curves to store the elements in memory, and to derive a block-recursive scheme for matrix multiplication.

Figure 1 illustrates the recursively defined linear order to store the matrix elements. It is based on so-called *iterations* of a Peano curve. Four different block numbering patterns marked as P, Q, R, and S are combined in a way to ensure a contiguous numbering of the matrix elements.

A blockwise matrix multiplication of such matrices is illustrated in equation (1). Each matrix block is named with respect to its numbering scheme and indexed with the name of the global matrix and the position within the storage scheme:

$$\underbrace{\begin{pmatrix} P_{A0} & R_{A5} & P_{A6} \\ Q_{A1} & S_{A4} & Q_{A7} \\ P_{A2} & R_{A3} & P_{A8} \end{pmatrix}}_{=:A} \underbrace{\begin{pmatrix} P_{B0} & R_{B5} & P_{B6} \\ Q_{B1} & S_{B4} & Q_{B7} \\ P_{B2} & R_{B3} & P_{B8} \end{pmatrix}}_{=:B} = \underbrace{\begin{pmatrix} P_{C0} & R_{C5} & P_{C6} \\ Q_{C1} & S_{C4} & Q_{C7} \\ P_{C2} & R_{C3} & P_{C8} \end{pmatrix}}_{=:C}. \tag{1}$$

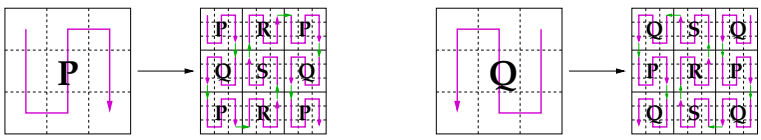

Fig. 1. Recursive construction of a Peano curve (for patterns P and Q, only)

In [1], the following inherently cache-efficient execution order was presented for the resulting block multiplications:

$$
\begin{array}{cccc}
P_0 \mathrel{+}= P_0P_0 & R_5 \mathrel{+}= P_6R_3 & \longrightarrow R_5 \mathrel{+}= R_5S_4 & P_6 \mathrel{+}= R_5Q_7 \longrightarrow P_6 \mathrel{+}= P_6P_8 \\
\downarrow & \uparrow & \downarrow & \uparrow \qquad \downarrow \\
Q_1 \mathrel{+}= Q_1P_0 & S_4 \mathrel{+}= Q_7R_3 & S_4 \mathrel{+}= S_4S_4 & Q_7 \mathrel{+}= S_4Q_7 \qquad Q_7 \mathrel{+}= Q_7P_8 \\
\downarrow & \uparrow & \downarrow & \uparrow \qquad \downarrow \\
P_2 \mathrel{+}= P_2P_0 & R_3 \mathrel{+}= P_8R_3 & R_3 \mathrel{+}= R_3S_4 & P_8 \mathrel{+}= R_3Q_7 \qquad P_8 \mathrel{+}= P_8P_8 \\
\downarrow & \uparrow & \downarrow & \uparrow \\
P_2 \mathrel{+}= R_3Q_1 & P_2 \mathrel{+}= P_8P_2 & R_3 \mathrel{+}= P_2R_5 & P_8 \mathrel{+}= P_2P_6 \\
\downarrow & \uparrow & \downarrow & \uparrow \\
Q_1 \mathrel{+}= S_4Q_1 & Q_1 \mathrel{+}= Q_7P_2 & S_4 \mathrel{+}= Q_1R_5 & Q_7 \mathrel{+}= Q_1P_6 \\
\downarrow & \uparrow & \downarrow & \uparrow \\
P_0 \mathrel{+}= R_5Q_1 & \longrightarrow P_0 \mathrel{+}= P_6P_2 & R_5 \mathrel{+}= P_0R_5 & \longrightarrow P_6 \mathrel{+}= P_0P_6
\end{array}
\tag{2}
$$

(indices A,B,C are left out to improve readability). Recursive extension of this blockwise multiplication leads to a cache-oblivious algorithm, which gains excellent locality properties from the underlying Peano curve.

2 Efficient Implementation on Multicore Platforms

In [2], we described how our block-recursive algorithm can be combined with hardware-oriented, highly efficient multiplication kernels. For that purpose, recursion in memory layout and block multiplication is stopped on small so-called *L1 blocks*, whose size are chosen depending on the size of the available level-1 cache (two L1 blocks should fit into the L1 cache). In a recent paper [3], we demonstrated that the resulting implementation, called TifaMMy, is competitive with up-to-date BLAS implementations (GotoBLAS and Intel's MKL). Moreover, we presented a shared-memory parallelisation of our algorithm based on OpenMP, and tested the performance of our approach on several multicore platforms. Figure 2 shows the average and maximum performance for double precision on a Xeon server with four quad-core Xeon processors (Tigerton, 2.93 GHz). In this test, TifaMMy outran both GotoBLAS and MKL in terms of absolute

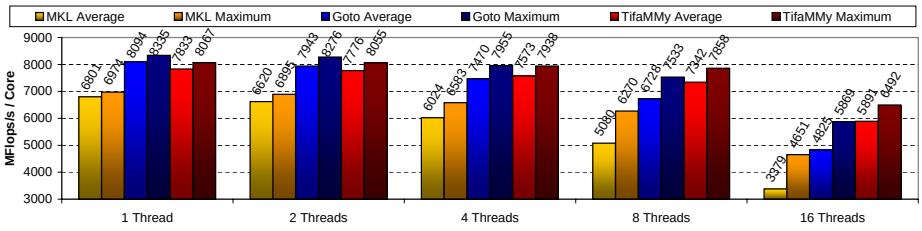

Fig. 2. Relative performance of TifaMMy, GotoBLAS and MKL on a Xeon server using up to 16 cores (four times quad-core). Performance is given in MFlop/s *per core*.

performance, if 8 or all 16 cores were used.

3 A Block Recursive Data Structure for Sparse Matrices

To extend our approach for sparse matrix multiplications, we first need to modify our block-recursive data structure to allow efficient storage of sparse matrices. Inspired by an approach used by Herrero and Navarro [6], we allow, in a first step, that each L1 block of the matrix can be either a zero block, a dense block stored in row-major order (as already existent), or a sparse matrix block stored according to the CSR (compressed sparse row) data structure. The respective data structure is split into two parts: one part that contains all the management information (in particular, the types of L1 blocks), and a contiguous data stream that holds, in Peano order, the element data required to store the dense and sparse L1 blocks.

In a second step, we then need to allow to stop the block recursion in the numbering scheme, once a block as in equation 1 contains only zero elements. The respective, adaptive block recursion is best described by a tree structure, where each node of the tree has 9 children (according to the Peano blocking), and where the leaves of the tree are either zero blocks (of any size), or L1 blocks that can be sparse, dense, or zero. For storing this tree (in the management part of our data structure), we propose a linearised storage of this tree according to a mixed depth-first/breadth-first traversal.

The blockwise multiplication is performed according to the Peano scheme given in equation (2). Multiplications that involve zero blocks are, of course, not performed. Due to the missing block operations, the Peano multiplication schemes loses its strict locality properties [1], the more zero blocks are present. However, we still expect a positive effect on cache performance.

4 Performance, Conclusion

We did first performance test of our approach on Xeon workstation that holds two quadcore processors (Clovertown, 2,66 GHz). We compared TifaMMy's implementation of a sparse-dense matrix multiplication with the one provided by Intel's MKL. For comparison, we used a 9-diagonal sparse matrix of size $n^2 \times n^2$, where each row i of the matrix contains non-zero elements a_{ij}, if $j \in \{i, i \pm 1, i \pm n, i \pm n \pm 1\}$. For the single-thread test, TifaMMy gave a performance gain of approximately 25 % compared to MKL. On eight cores, TifaMMy was also able to outrun MKL by 10–20 %, though the sparse-dense multiplication is still work in progress, and TifaMMy's sparse-dense multiplication kernels is not yet fully optimised. The meagre speedup when using eight cores instead of one also illustrates the stronger memory-boundedness of the sparse-dense matrix multiplication.

Still, our first results are encouraging, and motivate further research on the use of Peano-based, cache oblivious approaches in the context of sparse and sparse-dense matrix multiplication. One particular application we have in mind, is the computation of the exponential function of sparse, recursively structured

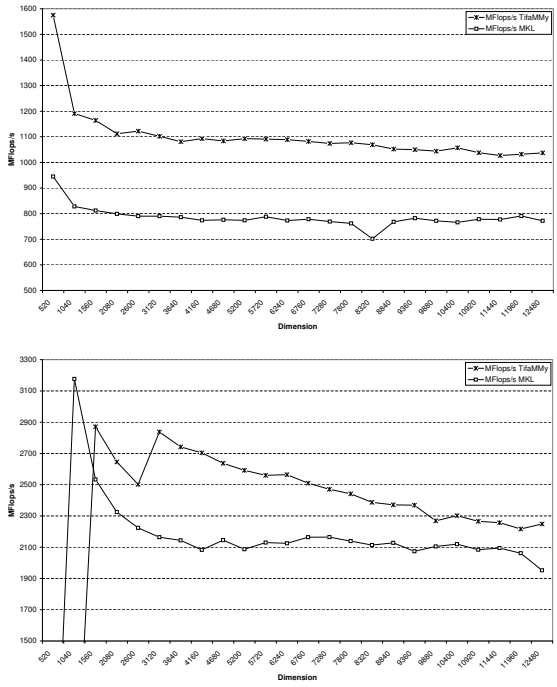

Fig. 3. Performance of sparse-dense matrix multiplication on 1 (top plot) or 8 (bottom plot) cores of a Xeon workstation with two quadcore processors.

matrices, where the exponential function is approximated by Chebyshev polynomials. In the respective, modified Horner scheme to evaluate the matrix polynomials, sparse-dense matrix multiplications are the time-critical subtask.

References

1. Bader, M., Zenger, C.: Cache oblivious matrix multiplication using an element ordering based on a Peano curve. Linear Algebra Appl. 417 (2–3), 2006.
2. Bader, M., Franz, R., Guenther, S., Heinecke, A.: Hardware-oriented Implementation of Cache Oblivious Matrix Operations Based on Space-filling Curves. LNCS 4967, 2008, in print.
3. Bader, M., Heinecke, A.: Parallel Matrix Multiplication based on Space-filling Curves on Shared Memory Multicore Platforms. Proc. of the ACM Int. Conf. on Computing Frontiers 2008, accepted.
4. Elmroth, E., Gustavson, F., Jonsson, I., and Kågström, B.: Recursive blocked algorithms and hybrid data structures for dense matrix library software. SIAM Review 46 (1), 2004.
5. Gustavson, F. G.: Recursion leads to automatic variable blocking for dense linear-algebra algorithms. IBM Journal of Research and Development 41 (6), 1999.
6. Herrero, J.R., Navarro, J.J.: Adapting Linear Algebra Codes to the Memory Hierarchy Using a Hypermatrix Scheme. LNCS 3911, 2006, pp. 1058-1065.