G. Haase and M. Liebmann

A Hilbert-Order Multiplication Scheme for Unstructured Sparse Matrices

Report No. 3/2005 October, 2005

Institute for Mathematics D,
Graz University of Technology
Steyrergasse 30
A-8010 Graz, Austria

Institute for Mathematics and
Scientific Computing,
University of Graz
Heinrichstrasse 36
A-8010 Graz, Austria

A Hilbert-Order Multiplication Scheme for Unstructured Sparse Matrices*

Gundolf Haase† Manfred Liebmann‡

Abstract

We investigate a new storage format for unstructured sparse matrices, based on the space filling Hilbert curve. Numerical tests with matrix-vector multiplication show the potential of the fractal storage format (FS) in comparison to the traditional compressed row storage format (CRS). The FS format outperforms the CRS format by up to 50% for matrix-vector multiplications with multiple right hand sides.

1 Introduction

The simulation of stationary processes often involves elliptic partial differential equations (PDEs). Solving these problems, that is determining the state variables from given source and boundary data, with the finite element method (FEM) is standard in numerical analysis today. This situation changes for time-dependent nonlinear optimization problems and related inverse problems arising from different practical applications. Here the numerical analysis leads to a sequence of linear equations and it is often necessary to solve multiple large-scale linear systems in order to solve the whole numerical problem. Therefore, improvements in linear solvers will reduce the time required for the solution of the practical problem significantly.

*This research has been partially supported by the bm:bwk (Federal Ministry for Education, Science and Culture) in the "Austrian Grid"

†Institute for Mathematics and Scientific Computing, Karl-Franzens University Graz, Heinrichstr. 36, A–8010 Graz, Austria, (gundolf.haase@uni-graz.at).

‡Institute for Analysis and Computational Mathematics, Johannes Kepler University Linz, Altenberger Str. 69, A–4040 Linz, Austria, (manfred.liebmann@uni-graz.at).

We focus in this paper on symmetric positive definite scalar problems in 3D, discretized with the finite element method on an unstructured mesh. The resulting system of equations

$$A\underline{u} = \underline{f} \tag{1}$$

is unstructured and sparse. The matrix A with n columns and rows typically has $\mathcal{O}(n)$ non-zero entries.

Direct solvers for solving (1) are suboptimal because of their storage requirements and related time complexity for the solution process. Instead, iterative methods like a conjugate gradient solver together with a good problem-specific preconditioner give optimal results. For the problem class at hand multigrid (MG) techniques especially algebraic multigrid (AMG) [BHM00, HL02] methods are a good choice. A profiling of the *Pebbles* [PEB02] AMG solver showed, that 75% of the time required for the solution of the linear system is spent in kernel routines, like matrix-vector multiplication and closely related routines like Gauss-Seidel smoothers in the AMG preconditioner. Sparse matrix-vector multiplications are also used in the intergrid transfer operators for the multigrid cycle. Aside from the reduction of arithmetic operations [HR05] and parallelization [Haa00, DHL03] the effective usage of the processor caches is key for high throughput application. On modern processor architectures cache misses can cost hundreds of valuable clock cycles.

There are several ways to improve the cache hit rate. Focusing on many smoothing sweeps as the group in Erlangen [WKRK00, KW03] showed results in high floating point throughput, but the method is restricted to tensor product and similar grids. Similar techniques for unstructured grids have been tested by the Kentucky group [Hu00], see also [DHH$^+$00a, DHH$^+$00b]. Techniques based on space-filling curves have been implemented recently by the group in München [Die05] for adaptive multilevel schemes and resulted in nearly optimal cache performance [Meh05, Zen05]. These results where achieved only on structured grids.

This paper introduces a new fractal storage format to improve the cache performance of matrix-vector multiplication compared to standard compressed row storage format, with matrices derived from unstructured grids. The basic idea of the fractal storage format is derived from the space-filling Hilbert curve. Where some algorithmic details are adapted for a simple software implementation, while preserving the intrinsic locality properties of the space-filling Hilbert curve.

2 Matrix formats and matrix-vector multiplications

2.1 The compressed row storage format

We introduce the well-known classical compressed row storage (CRS) format for completeness and for introducing the notations therein. The CRS format for sparse matrices uses two vectors of length nnz named `val` for storing a non-zero matrix entries $A_{i,j}$ and `col_idx` for storing the appropriate column indices j. An entry `row_ptr`(i) of the vector `row_ptr` points to the beginning of the matrix row i in the two vectors above see Figure 1. Assuming a memory

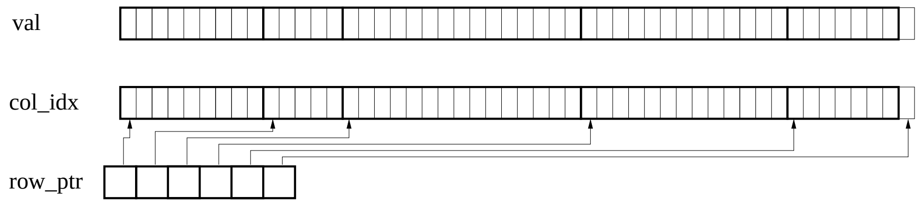

Figure 1: Matrix in CRS storage.

consumption of 8 Byte (double precision) for storing the matrix entries and 4 Byte for column and row indices we end up with total memory costs of $\mathcal{M} = 12 * nnz + 4 * (n+1)$. The matrix-vector multiplication is as follows. A multiplication with the transposed matrix requires a different subroutine

Algorithm 1 Matrix-vector multiplication in CRS: $\underline{d} \longleftarrow \underline{d} + A * \underline{w}$

```
void CRS_Mult(const double val[], const int col_idx[], const int row_ptr[],
                                        int n, const double w[], double d[])
for all i=0,...,n-1 do
  tmp = 0.0;
  for all jp=row_ptr[i],...,row_ptr[i+1]-1 do
    tmp += val[jp] * w[col_idx[jp]];
  end for
  d[i] = tmp;
end for
return;
```

with worse memory access patterns.

2.2 The fractal storage format

Space filling curves are used as a tool in computer graphics and computer science. They are typically applied to regular structures like tensor product meshes. But instead of applying the space filling Hilbert curve to the nodes of a finite element mesh, we pick up an idea from computer graphics hardware design [MWM01] to store the sparse system matrix in Hilbert-order. Thus enabling a matrix-vector multiplication scheme, that takes advantage of the intrinsic locality of the Hilbert curve to improve the cache access patterns for the multiplication process. Finally the Hilbert-order access pattern is locally modified to enable a simpler software implementation of the coordinate mapping between Hilbert-order and standard coordinate format. This modification does not change the overall properties of the algorithm. We call the above defined storage format the fractal storage (FS) format.

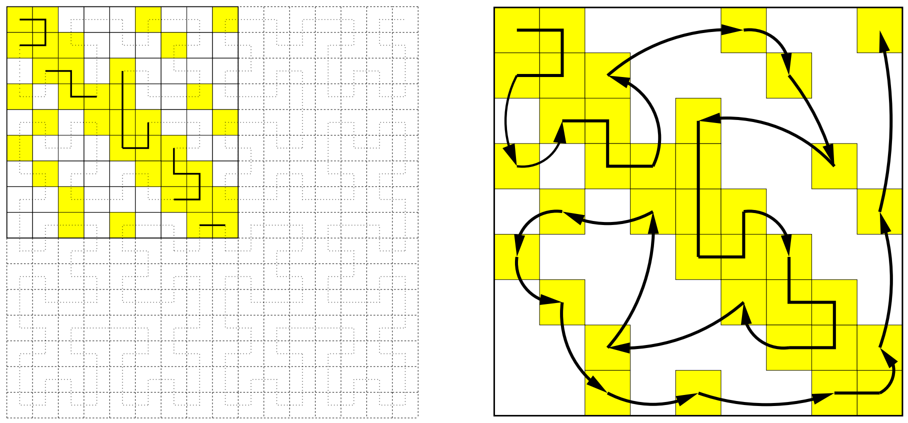

Figure 2: Hilbert curve through an extented domain and fractal curve with non-zero matrix elements

If we think of all $n \times n$ matrix entries of A as pixels, we can draw a Hilbert curve across an extended $2^k \times 2^k$ square domain, see left side in Fig. 2. Now, starting at the upper left corner ($A_{1,1}$) we follow the Hilbert curve and store all non-zero matrix elements along the way in Hilbert-order in a linear sequence, Fig. 2. Due to the fact, that the fractal curve changes rows and columns we have to store row and column indices for all non-zero matrix entries, in a structure like

```
struct{ int    row; int    col;  double val; } FS_Entry;
```

The resulting fractal storage format of the sparse matrix is simply an array `FS_Entry a[]` depicted in Fig. 3. The matrix-vector multiplication for the

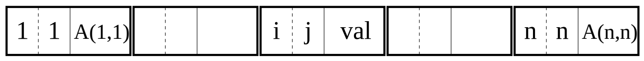

Figure 3: Matrix in FS format with a 16 Byte structure for the entries

FS format contains only a simple loop. Swapping the row with the column

Algorithm 2 Matrix-vector multiplication in FS format: $\underline{d} \longleftarrow \underline{d} + A * \underline{w}$

```
void FS_Mult(const SF_Entry a[], int nnz, const double w[], double d[])
for all i=0,...,nnz-1 do
    const FS_Entry &ai = a[i];
    d[ai.row] += ai.val * w[ai.col];
end for
return;
```

index in the above algorithm results immediately in a multiplication with the transposed matrix, while the memory access patterns are essentially the same. For the sake of a simpler software implementation, we slightly modify the algorithm and locally rotate the arcs of the Hilbert curve as noted above.

3 Numerical Results

We used the discretized potential equations in a simulation of arryhthmia induction in a rabbit ventricular model for our numerical tests, see Fig. 4. This involves solving a stationary problem embedded in a time dependent simulation [PLKV05]. The first test, named *TBunnyC2*, is a small 111589×111589 sparse matrix with 1444052 non-zero elements. The second example, *TBunnyC*, is a 862515×862515 sparse matrix with 12775719 non-zero elements.

Simulations are carried out for the fractal multiplication scheme named *FS*, for an SSE2 optimized version of the fractal multiplication scheme: *FS/SSE2* for the classical compressed row storage multiplication scheme *CRS* and the transposed version of the compressed row storage scheme: *CRST*. All test cases, that is *FS*, *FS/SSE2*, *CRS* and *CRST* are carried out with up to eight right hand sides, that are stored in an interleaved way to improve data

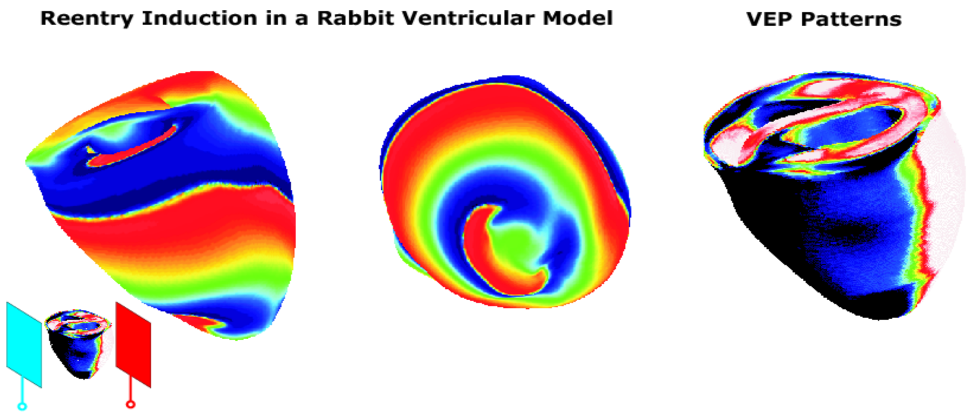

Figure 4: Electrical potential in the rabbit heart

locality. RHS denotes the number of right hand sides and the performance is measured in $MFLOPS$.

The performance data was collected over typically 16 to 32 iterations over the same matrix with an initial startup iteration to avoid memory management related performance issues. The average in million floating point operations per second (MFLOPS) for the various test cases is displayed below. The test machines are a Pentium4 1.3GHz processor with 256KB L2-cache and 800MHz RDRAM and a Pentium4 3.4GHz processor with 1MB L2-cache and 533MHz DDR2 SDRAM.

The first test was carried out on the smaller example $TBunnyC2$ with 111589 non-zero elements. We see that FS is always better on the CPU with the small cache and the same holds for more than two right hand sides on the larger cache computer. The use of the SSE unit improves the performance by 23% and 15%, respectively.

Performance data for $TBunnyC$ with 862515 nodes and 12775719 non-zero elements. Again, FS is always better on the CPU with the small cache but the same holds only for eight right hand sides on the larger cache computer. The use of the SSE unit improves the performance by 25% and 17%, respectively.

Table 1: TBunnyC2

P4 1.3GHz/RHS	8	4	2	1
FS	153.808	159.069	140.883	106.107
FS/SSE2	189.773	192.34	147.635	99.2689
CRS	116.986	118.715	107.34	97.0791
CRST	124.094	117.172	88.7368	73.8174
P4 3.4GHz/RHS	8	4	2	1
FS	534.602	487.701	400.952	288.81
FS/SSE2	614.592	556.324	446.47	295.742
CRS	407.92	430.107	455.268	394.112
CRST	346.627	375.498	363.856	303.512

4 Conclusions

The direct comparison of the fractal multiplication scheme FS derived from the idea of a Hilbert curve with the traditional compressed row storage scheme CRS shows an overall performance advantage on the low end Pentium4 machine with a small 256K L2 cache in almost all cases.

On the high end Pentium4 machine with 1MB of L2 cache the traditional scheme has an advantage for one and two right hand sides, but the fractal multiplication scheme gives always the best peak performance typically achieved with eight simultaneous right hand sides.

Using compiler intrinsics to optimize the inner loop with SSE2 instructions to perform two simultaneous multiplications or additions improves the performance of the fractal multiplication scheme by about $15 - 20\%$ points.

Finally it should be noted that the performance of the fractal multiplication scheme is not sensible to matrix transposition due to the intrinsic locality of the data access pattern. While in the case of the compressed row storage format the data access pattern in the transposed case is suboptimal and this is reflected in the performance data. In the transposed case the fractal multiplication scheme is at least equal and in many cases superior to the traditional scheme. The same will be valid for symmetric matrices where a simple change in Alg. 3 reduces the amount of memory for storing the matrix while the MFLOP rate will remain the same. Doing the same for a CRS-matrix will result in a performance between the test CRS and $CRST$.

Table 2: TBunnyC

P4 1.3GHz/RHS	8	4	2	1
FS	166.969	180.237	158.551	114.999
FS/SSE2	208.943	216.566	157.634	107.983
CRS	109.814	120.508	115.962	116.974
CRST	117.562	119.635	97.5478	90.1285
P4 3.4GHz/RHS	8	4	2	1
FS	584.268	531.932	428.031	310.538
FS/SSE2	680.167	613.848	489.096	311.484
CRS	445.114	533.276	538.102	422.011
CRST	375.121	454.563	428.031	314.298

References

[BHM00] W. L. Briggs, V. E. Henson, and S. McCormick. *A Multigrid Tutorial.* SIAM, second edition, 2000.

[DHH$^+$00a] C. Douglas, G. Haase, J. Hu, W. Karl, M. Kowarschik, U. Rüde, and C. Weiss. Portable memory hierarchy techniques for pde solvers, part I. *SIAM News*, 33(5):1, 8–9, 2000.

[DHH$^+$00b] C. Douglas, G. Haase, J. Hu, W. Karl, M. Kowarschik, U. Rüde, and C. Weiss. Portable memory hierarchy techniques for pde solvers, part II. *SIAM News*, 33(6):1, 10–11, 16, 2000.

[DHL03] Craig C. Douglas, Gundolf Haase, and Ulrich Langer. *A Tutorial on Elliptic PDE Solvers and Their Parallelization.* Software, Environments, and Tools,. SIAM, Philadelphia, 2003. ISBN 0-89871-541-5.

[Die05] Nadine Dieminger. Kriterien für die selbstadaption cache-effizienter mehrgitteralgorithmen. Diplomarbeit, Fakultät für Informatik, Technische Universität München, 2005.

[Haa00] Gundolf Haase. A parallel AMG for overlapping and non-overlapping domain decomposition. *Electronic Transactions on Numerical Analysis (ETNA)*, 10:41–55, 2000.

[HL02] Gundolf Haase and Ulrich Langer. *Modern Methods in Scientific Computing and Applications*, volume 75 of *NATO Science Series II. Mathematics, Physics and ChemistryD*, chapter Multigrid Methods: From Geometrical to Algebraic Versions, pages 103–154. Kluwer Academic Press, Dordrecht, 2002.

[HR05] Gundolf Haase and Stefan Reitzinger. Cache issues of algebraic multigrid methods for linear systems with multiple right-hand sides. *SIAM J. Sci. Comput.*, 27(1):1–18, 2005.

[Hu00] J. Hu. *Cache Based Multigrid on Unstructured Grids in Two and Three Dimensions.* PhD thesis, University of Kentucky, Department of Mathematics, Lexington, KY, 2000.

[KW03] M. Kowarschik and C. Weiß. An overview of cache optimization techniques and cache-aware numerical algorithms. In *Proceedings of the GI-Dagstuhl Forschungseminar: Algorithms for Memory Hierarchies*, volume 2625 of *(LNCS)*. Springer, 2003.

[Meh05] Miriam Mehl. A cache-oblivious self-adapted full multigrid method. Talk at the Copper Mountain Conference on Multigrid Methods, April 2005.

[MWM01] Michael D. McCool, Chris Wales, and Kevin Moule. Incremental and hierarchical hilbert order edge equation polygon rasterizatione. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, volume 119, pages 65–72, New York, 2001. ACM Press.

[PEB02] PEBBLES. *User's Guide.* Johannes Kepler University Linz, SFB "Numerical and Symbolic Scientific Computing", 2002. http://www.numa.uni-linz.ac.at/Research/Projects/pebbles.html.

[PLKV05] G. Plank, L.J. Leon, S. Kimber, and E.J. Vigmond. Defibrillation depends on conductivity fluctuations and the degree of disorganization in reentry patterns. *J. Cardiovasc. Electrophysiol*, 16(2):205–216, 2005.

[WKRK00] C. Weiss, M. Kowarschik, U. Rüde, and W. Karl. Cache-aware multigrid methods for solving poisson's equation in two dimensions. *Computing*, 64(4):381–399, 2000.

[Zen05] Christoph Zenger. A cache-oblivious adaptive parallel multilevel implementation of the finite element method using space-filling curves. Talk at the Oberwolfach Seminar on Fast Solvers for PDEs, May 2005.