Fast Regridding using LSQR on Graphics Hardware

G. Buchgraber[1], F. Knoll[2], M. Freiberger[2], C. Clason[3], M. Grabner[1], and R. Stollberger[2]
[1]Institute for Computer Graphics and Vision, Graz University of Technology, Graz, Austria, [2]Institute of Medical Engineering, Graz University of Technology, Graz, Austria, [3]Institute of Mathematics and Scientific Computing, University of Graz, Graz, Austria

Introduction: Iterative image reconstruction methods have become increasingly popular for parallel imaging or constrained reconstruction methods, but the main drawback is the long reconstruction time. In the case of non-Cartesian imaging, resampling of *k*-space data between Cartesian and non-Cartesian grids has to be performed in each iteration step. Therefore the gridding procedure tends to be the time limiting step in these reconstruction strategies. With the upcoming parallel computing toolkits (such as CUDA [1]) for graphics processing units (GPUs) image reconstruction can be accelerated in a tremendous way [2,3]. In this work, we present a fast GPU based gridding method and a corresponding inverse-gridding procedure by reformulating the gridding procedure as a linear problem with a sparse system matrix (see **Fig.1**), similar to the approach in [4].

Methods: In MR literature the term "gridding" is often used as a synonym for a convolution interpolation. This process can be easily formulated as a problem of solving a set of linear equations

$$A\vec{x} = \vec{b}\,, \qquad (1)$$

where x and b are vectors containing the uniform and non-uniform data samples, and A is a matrix describing the relationship between them. Because of exclusive local relationships between non-uniform and uniform data samples, the matrix A is large and sparse, and therefore we use LSQR [5] as a numerical method for solving this set of linear equations. LSQR is specifically tailored to sparse matrices and has, in comparison to other iterative solvers like PCG (preconditioned conjugate gradient method), advantages regarding complexity and numerical stability in fixed-point arithmetic. During its iterations, LSQR performs several basic linear algebra operations, including sparse matrix-vector multiplication, vector addition and scaling, as well as scalar multiplication of two vectors. In our implementation all of these operations are done in parallel on the GPU. For a fast and efficient parallel computation of sparse matrix-vector products, the matrix A is arranged in a special memory layout, considering coalesced thread access and memory bandwidth, as recommended in [6]. For optimized GPU implementations, special care must been taken to ensure that the GPU's texture cache is used efficiently because the sample positions are usually irregularly distributed (because they lie on e.g. radial trajectories). We propose to solve this by reorganisation of the samples in vector b, so that samples that are close to each other will be processed in parallel or at least consecutively. Several two-dimensional reordering strategies with a good locality preserving behaviour were investigated, such as space-filling curves like the Hilbert curve [7] or the Z-order curve [8] and some line-by-line methods shown in **Fig.2**.

LSQR is used for the gridding step (computing vector x) and the inverse-gridding procedure (computing vector b) uses a sparse matrix-vector multiplication. The matrix A can be defined arbitrarily and represents the used regridding kernel. The well known Kaiser-Bessel kernel and the simple strategy of bilinear interpolation were used in this work. In the case of bilinear interpolation, the sparse matrix-vector multiplication, utilized in the inverse-gridding step, can be replaced by a native GPU texture interpolation which additionally benefits from the optimized hardware implementation. Concerning LSQR, the total number of iterations depends on the chosen stopping criteria, which are usually the absolute or relative tolerance, given by a threshold for the L2-norm of the residual, and the maximum number of iterations, in case the desired accuracy could not be achieved.

Results and Discussion: Performance tests were carried out using Cartesian complex k-space data of different image size, with a sparse matrix A that represented simple bilinear interpolation. Inverse Gridding with the proposed method was used to interpolate data on radial trajectories. The following matrix sized were investigated: Cartesian: 256x256 → Radial: 300 spokes with 256 sample points; Cartesian: 512x512 → Radial: 512 spokes with 512 sample points; Cartesian: 1024x1024 → Radial: 1024 spokes with 1024 sample points. The GPU implementation (using single precision) of the proposed gridding algorithm with LSQR outperforms the built-in Matlab (The MathWorks, Natick, MA) CPU implementation by factors of 25 to 31. With the used memory layout for A, even matrices with size of $1024^2 \times 1024^2$ could be applied without decreases in speedup. Considering the inverse-gridding procedure, the use of native texture interpolation accelerated this step with an additional speedup of approximately 10 in comparison to sparse matrix-vector multiplication. For the 1024x1024 matrix, GPU inverse gridding computation times were 4.2ms with sparse matrix vector multiplication and 0.409ms with native texture interpolation. However, it should be noted that for applications in iterative image reconstruction methods, the regridding and the inverse gridding step have to be performed in each iteration. With our method, the time limiting computation is the LSQR based regridding step which takes roughly 1000 times longer (6s in comparison to 4.2ms for the case of inverse gridding with sparse matrix vector multiplication). The effect of reorganisation strategies (**Fig.2**) was evaluated with Cartesian k-space test data of size 1024x1024. Different amounts of randomly distributed positions within the boundaries of the source data set were selected. Interpolation was then carried out before and after reordering of interpolation positions. The experimental results of the proposed non-uniform sample reorganisation strategies, evaluated on an Nvidia GeForce 280, are displayed in **Fig.3**. Speedups from 2 to 7 were observed in comparison to conventional, unordered data positions in b.

In conclusion, the proposed GPU regridding approach yields significant speedups in computation time. This will be of great benefit in iterative image reconstruction methods for parallel imaging or constrained reconstruction strategies.

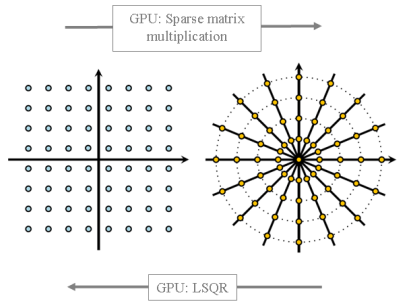

Fig. 1: *Position of acquired data samples on Cartesian (a) or radial (b) sampling trajectories and definitions of corresponding GPU regridding operations.*

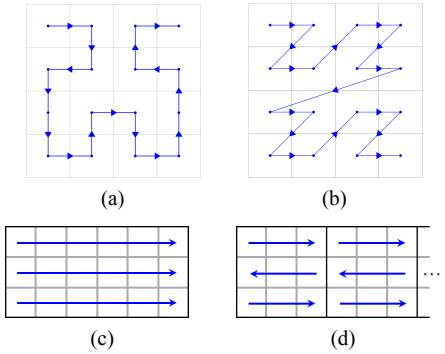

Fig. 2: *Reordering of non-uniform samples along:*
(a) Hilbert curve (2^{nd} order)
(b) Z-order curve (2^{nd} order)
(c) Line-By-Line
(d) Blocked Line-By-Line with alternating row directions

Matrix size		Computation time		
Radial	Cartesian	CPU	GPU	Speedup
300[256]	256×256	7.897s	0.313s	25
512[512]	512×512	39.634s	1.249s	32
1024[1024]	1024×1024	184.456s	6.024s	31

Tab. 1: *Regridding computation times: Matlab LSQR (CPU) vs. GPU LSQR performance results (abs. tolerance: 1e-5, max. iterations: 1000) evaluated on Nvidia GeForce 8800 GTX connected to a host x86/Linux system.*

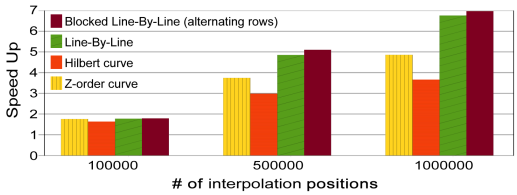

Fig. 3: *Evaluation of GPU texture 2D interpolation performance. The interpolation time of randomly distributed sample positions is compared with the same positions reordered along the patterns suggested in Fig.2.*

References: [1] NVIDIA Corporation, NVIDIA CUDA – Programming Guide (2009), [2] Hansen et al., MRM 59: 463-468 (2008), [3] Sorensen et al., IEEE Trans. Med. Imag. 27(4): 538-547 (2008), [4] D. Rosenfeld, MRM 40(1):14-23 (1998), [5] Paige et al., ACM Trans. Math. Softw. 8:43-71 (1982), [6] M. Liebmann, Efficient PDE Solvers on Modern Hardware with Applications in Medical and Technical Sciences, PhD Thesis (University of Graz, 2009), [7] Hilbert, Ueber die stetige Abbildung einer Line auf ein Flaechenstueck, Mathematische Annalen 38:459-460 (1891), [8] Morton, A computer oriented geodetic data base; and a new technique in file sequencing. IBM technical report, IBM Ltd., (1966)