Albert-Jan N. Yzelman
Department of Computer Science, KU Leuven
ExaScience Lab (Intel Labs Europe)

ExaScience Lab
Intel Labs Europe
EXASCALE COMPUTING

Strategies for

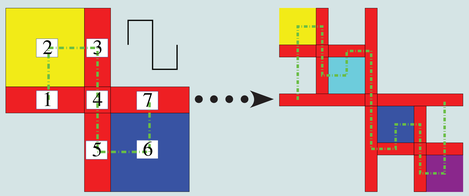
Shared-Memory Parallel Sparse Matrix–Vector Multiplication

Sequential stategies

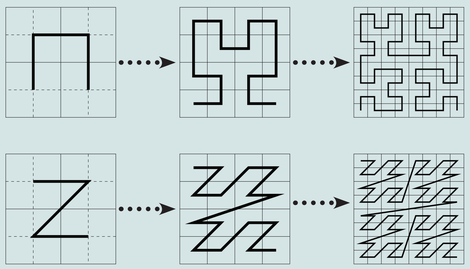
A parallel SpMV multiplication can be derived from an efficient sequential cache-oblivious scheme. An example of the latter is reordering the sparse matrix into the *Doubly Separated Block Diagonal* (SBD) form, followed by a standard multiply:

$$PAQ = \begin{pmatrix} A_1 & S_1 & \\ S_2 & S_3 & S_4 \\ & S_5 & A_2 \end{pmatrix}.$$

Recursion and blockwise multiplication are then visualised as follows:



These blocks are ordered according to a cache-friendly curve. It is also possible to use space-filling curves on the nonzeroes of the original matrix; both the Hilbert curve (top) and the Morton curve (bottom) have been used this way.



[HLP07] Gundolf Haase, Manfred Liebmann, and Gernot Plank. A Hilbert-order multiplication scheme for unstructured sparse matrices. *International Journal of Parallel, Emergent and Distributed Systems*, 22(4):213–220, 2007.

[YB09] A. N. Yzelman and Rob H. Bisseling. Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods. *SIAM Journal on Scientific Computing*, 31(4):3128–3154, 2009.

[YB11] A. N. Yzelman and Rob H. Bisseling. Two-dimensional cache-oblivious sparse matrix–vector multiplication. *Parallel Computing*, 37(12):806 – 819, 2011.

[YB12] A. N. Yzelman and Rob H. Bisseling. A cache-oblivious sparse matrix–vector multiplication scheme based on the Hilbert curve. In M. Günther, A. Barlet, M. Brunk, S. Schöps, and M. Striebel, editors, *Progress in Industrial Mathematics at ECMI 2010*, volume 17 of *Mathematics in Industry*.

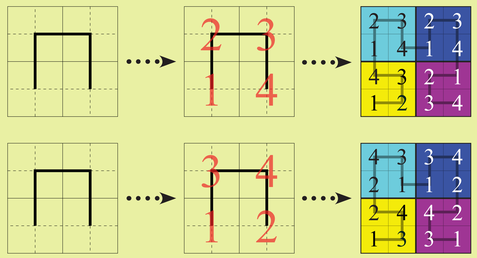
Introduction

The sparse matrix–vector (SpMV) multiplication $y = Ax$, with A an $m \times n$ sparse matrix and x, y dense vectors of appropriate sizes, is a widely used kernel. It performs two floating point operations on three data elements (x_j, y_i, a_{ij}) per nonzero, with additional index data requirements dependent on how A is stored. This results in a bandwidth-limited algorithm, which, when parallelised, multiplies the bandwidth requirements even further.

Nonzeroes are accessed only once during multiplication, whereas elements from x and y are, in general, reused. Optimisation thus focuses on optimising these reuse patterns. Techniques include adapting the nonzero structure of sparse matrices or adapting the storage method of the matrix, while keeping bandwidth requirements in check.

Non-distributed strategies

Extending the cache-oblivious algorithms to shared-memory multicore architectures means accounting for the possibility of shared caches. Changing the regular ordering induced by the Hilbert curve to an interleaved order yields a 'core-oblivious' scheme; compare the regular access (top) and this interleaved access (bottom), and their effect on parallelisation.



Note how processors interchange between the start and end of their curves, thus using the data brought into shared caches by neighbouring processors.

Problem: submatrices assigned to cores have overlapping rows, the exact number of which varies per matrix due to load-balancing. To prevent data races, each core allocates a local output vector of size $\Theta(m)$, worst case. Postprocessing then combines the local results into a global output vector.

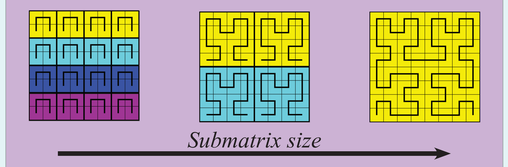
This does not scale, however; in contrast to the 1D and 2D distributed strategies presented on the right.

Using the Morton curve, implicit distribution, block-level replication and fine-grained parallelism

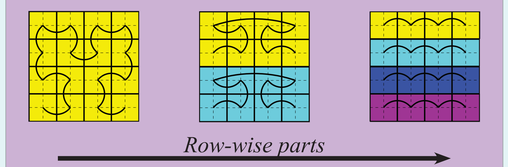
Morton curve, implicit, coarse-grained

1D strategies

Strategies based on space-filling curves do not have to apply reordering on the nonzero level. Instead, A can be subdivided into disjoint blocks. A block-based strategy can apply the Hilbert curve on nonzeroes within these blocks, while processing the blocks themselves in the standard Compressed Row Storage (CRS) order. Assigning rows of blocks to processors implicitly yields a 1D strategy:



Alternatively, the Hilbert curve can define the order in which blocks are processed, while nonzeroes within blocks are stored in CRS format. An *explicit* rowwise distribution yields the strategy illustrated below, for 2×2 blocks. Both of these 1D strategies are fully scalable.



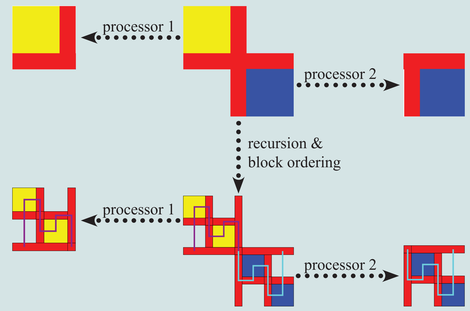
[MFT+10] M. Martone, S. Filippone, S. Tucci, M. Paprzycki, and M. Ganzha. Utilizing recursive storage in sparse matrix-vector multiplication. In *proceedings of the ISCA 25th International CATA Conference*, 2010.

[BFF+09] A. Buluç, J. Fineman, M. Frigo, J. Gilbert, and C. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *proceedings of the 21st SPAA*, 2009.

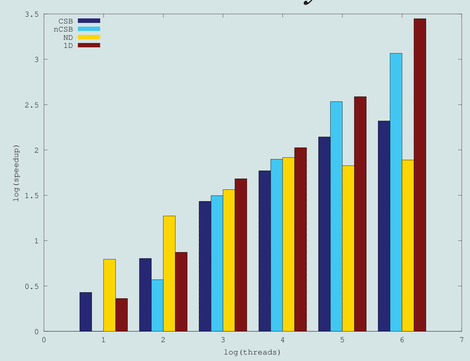
2D strategies

Explicit rowwise distributions using local output vectors help to attain good performance on NUMA (e.g., multi-socket) architectures, as writing to the output vector is done via the memory banks closest to the executing core.

Explicitly distributing the input vector as well, so that cores both have local input and output vectors, brings us closer to distributed-memory partitioning techniques. Combined with the cache-oblivious SBD form, an efficient two-dimensional strategy arises:



Preliminary results



Experiments on the wikipedia '06 matrix on an 8-socket 8-core machine. Speedups are relative to a sequential CRS kernel.

KATHOLIEKE UNIVERSITEIT LEUVEN

albert-jan.yzelman@cs.kuleuven.be