Space-filling curves in SpMV multiplication

Albert-Jan Yzelman (ExaScience Lab / KU Leuven)
Dirk Roose (KU Leuven)

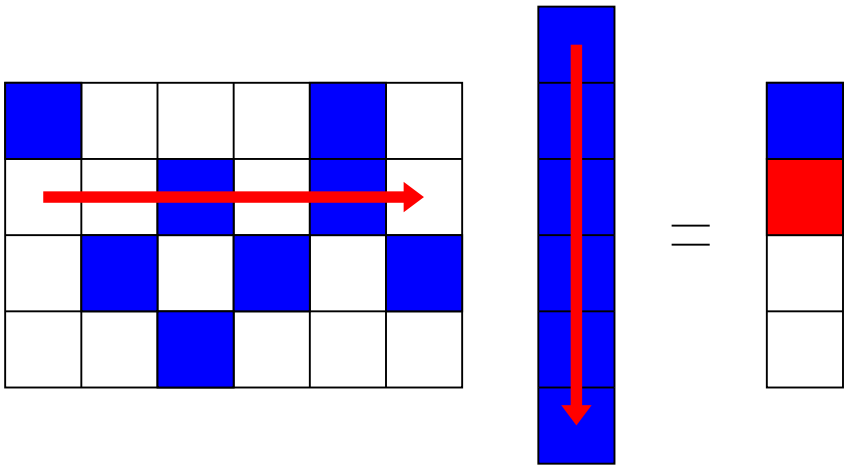
September 2013

Introduction

Given a *sparse* $m \times n$ matrix A and an $n \times 1$ input vector x.

We consider both sequential and parallel computation of

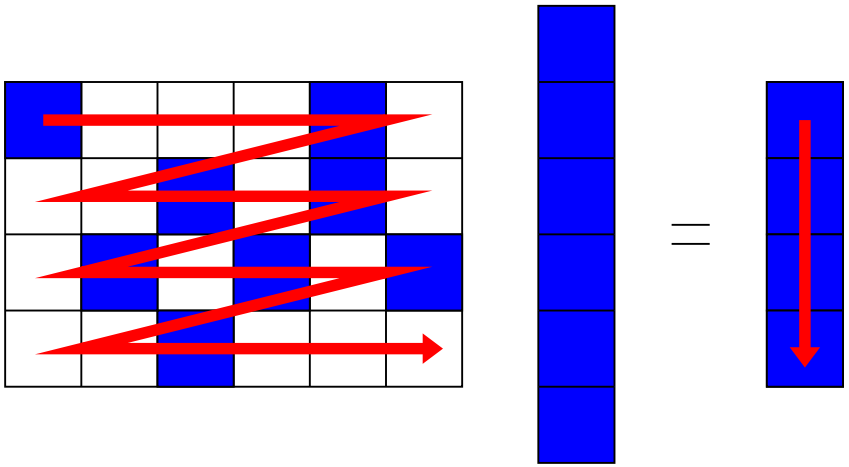
$$Ax = y :$$



We utilise space-filling curves to offset inefficient cache use.

Introduction

Curves have always been used in sparse computations:



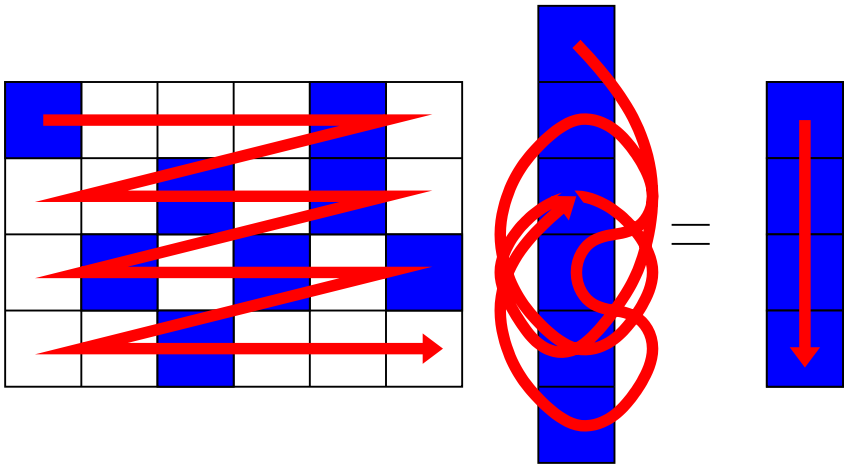
Compressed Row Storage (CRS)

A row-major ordering of the matrix nonzeroes is imposed by the above curve.

- This causes a linear access of the output vector y;
- but causes irregular access of the input vector x.

Introduction

Curves have always been used in sparse computations:



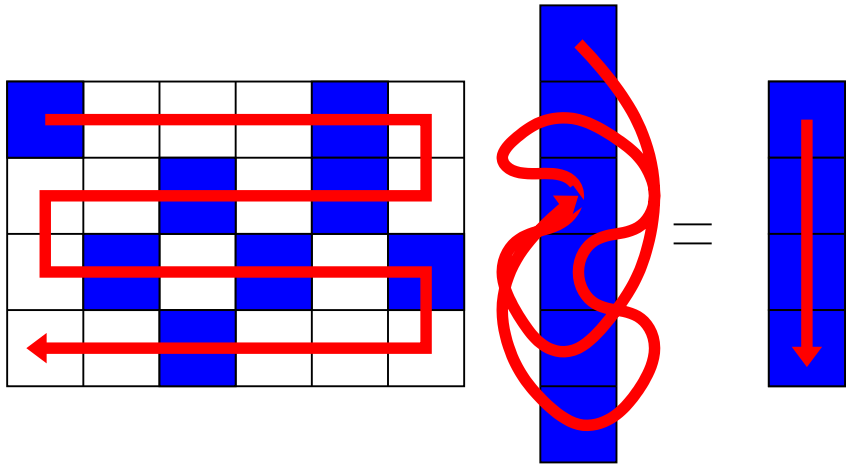
Compressed Row Storage (CRS)

A row-major ordering of the matrix nonzeroes is imposed by the above curve.

- This causes a linear access of the output vector y;
- but causes irregular access of the input vector x.

Introduction

Ideas for improvement:



Zig-zag CRS

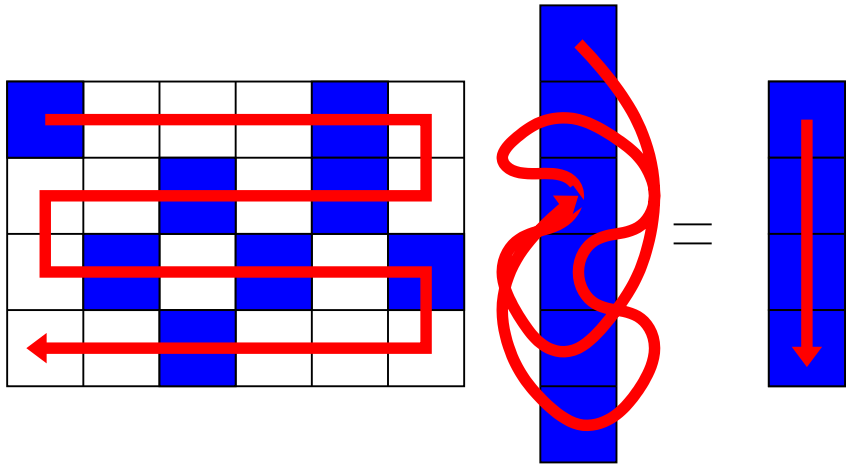
Alternating ascending-descending row-major ordering.

- Retains linear access of the output vector y;
- imposes a bit more ($\mathcal{O}(m)$) locality.

Ref.: A. N. Yzelman and Rob H. Bisseling, "Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods", SIAM Journal of Scientific Computation 31(4), pp. 3128-3154 (2009).

Introduction

Ideas for improvement:



Zig-zag CRS

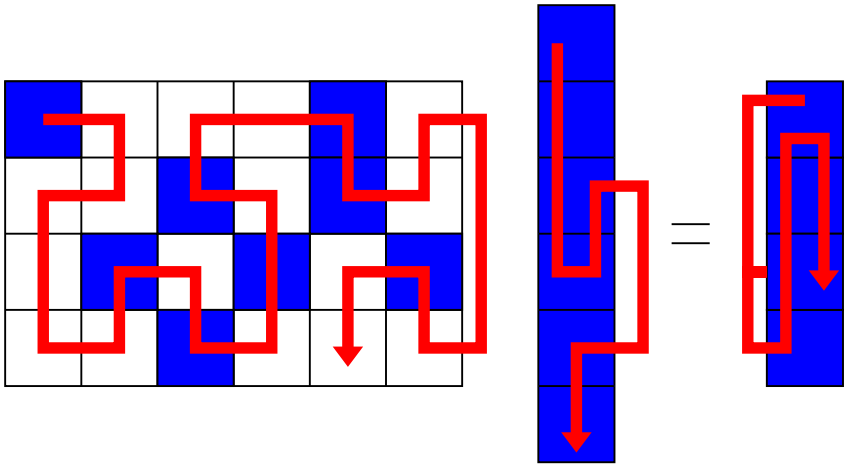
Alternating ascending-descending row-major ordering.

- Retains linear access of the output vector y;
- imposes a bit more ($\mathcal{O}(m)$) locality.

Ref.: A. N. Yzelman and Rob H. Bisseling, "Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods", SIAM Journal of Scientific Computation 31(4), pp. 3128-3154 (2009).

Introduction

Ideas for improvement: why not space-filling curves?



Fractal storage using the coordinate format (COO)

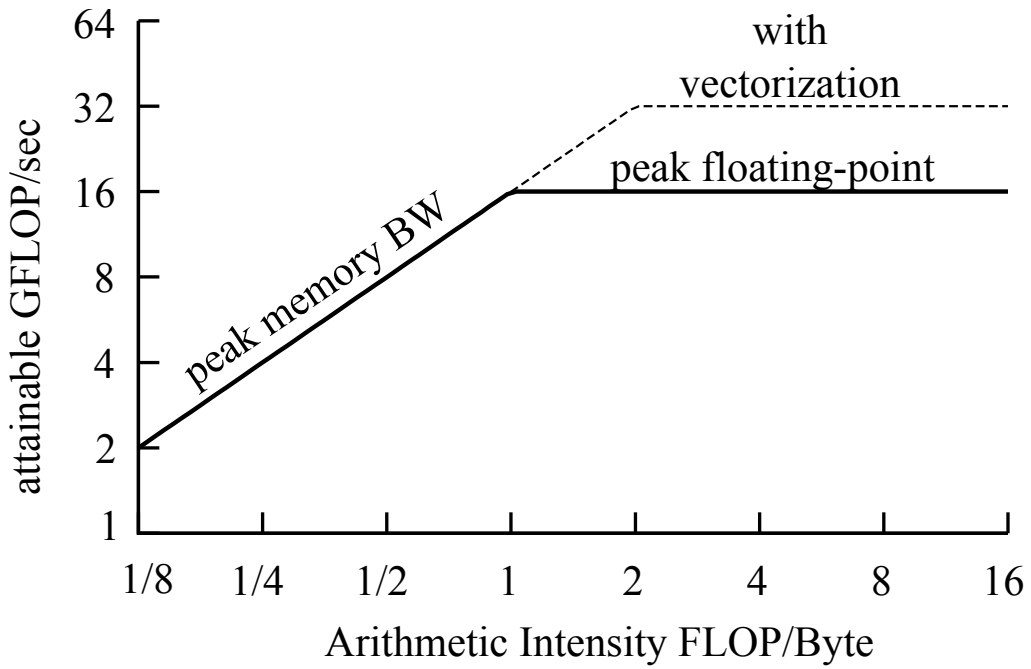
Nonzero ordered according to the Hilbert curve.

- No longer linear access of the output vector y, but
- accesses on both x and y now have temporal locality.

Ref.: Haase, Liebmann and Plank, "A Hilbert-Order Multiplication Scheme for Unstructured Sparse Matrices", International Journal of Parallel, Emergent and Distributed Systems 22(4), pp. 213-220 (2007).

Sequential SpMV

Space-filling curves avoid **inefficient cache use**, but that is not the only problem:



- SpMV has **low arithmetic intensity**: bandwidth issues arise. Compression is mandatory!

(Image courtesy of Prof. Wim Vanroose, UA)

Sequential SpMV

Assuming a row-major order of nonzeroes:

$$A = \begin{pmatrix} 4 & 1 & 3 & 0 \\ 0 & 0 & 2 & 3 \\ 1 & 0 & 0 & 2 \\ 7 & 0 & 1 & 1 \end{pmatrix}$$

CRS:

$$A = \begin{cases} V & [4\ 1\ 3\ 2\ 3\ 1\ 2\ 7\ 1\ 1] \\ J & [0\ 1\ 2\ 2\ 3\ 0\ 3\ 0\ 2\ 3] \\ \hat{I} & [0\ 3\ 5\ 7\ 10] \end{cases}$$

Storage requirements:

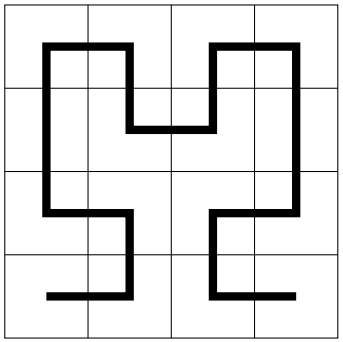
$$\Theta(2nz + m + 1),$$

where nz is the number of nonzeroes in A.

Sequential SpMV

Assuming a Hilbert order of nonzeroes:

$$A = \begin{pmatrix} 4 & 1 & 3 & 0 \\ 0 & 0 & 2 & 3 \\ 1 & 0 & 0 & 2 \\ 7 & 0 & 1 & 1 \end{pmatrix}$$



COO:

$$A = \begin{cases} V & [7\ 1\ 4\ 1\ 2\ 3\ 3\ 2\ 1\ 1] \\ J & [0\ 0\ 0\ 1\ 2\ 2\ 3\ 3\ 3\ 2] \\ I & [3\ 2\ 0\ 0\ 1\ 0\ 1\ 2\ 3\ 3] \end{cases}$$

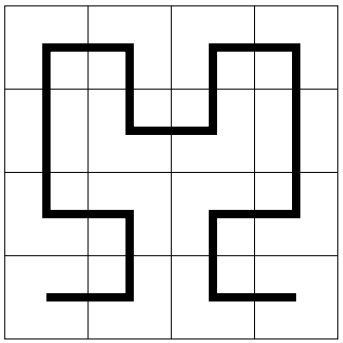
Storage requirements:

$$\Theta(3nz).$$

This extra data movement is **prohibitive**.

Sequential SpMV

$$A = \begin{pmatrix} 4 & 1 & 3 & 0 \\ 0 & 0 & 2 & 3 \\ 1 & 0 & 0 & 2 \\ 7 & 0 & 1 & 1 \end{pmatrix}$$



BICRS:

$$A = \begin{cases} V & [7\ 1\ 4\ 1\ 2\ 3\ 3\ 2\ 1\ 1] \\ \Delta J & [0\ 4\ 4\ 1\ 5\ 4\ 5\ 4\ 3\ 1] \\ \Delta I & [3\ \text{-}1\ \text{-}2\ 1\ \text{-}1\ 1\ 1\ 1] \end{cases}$$

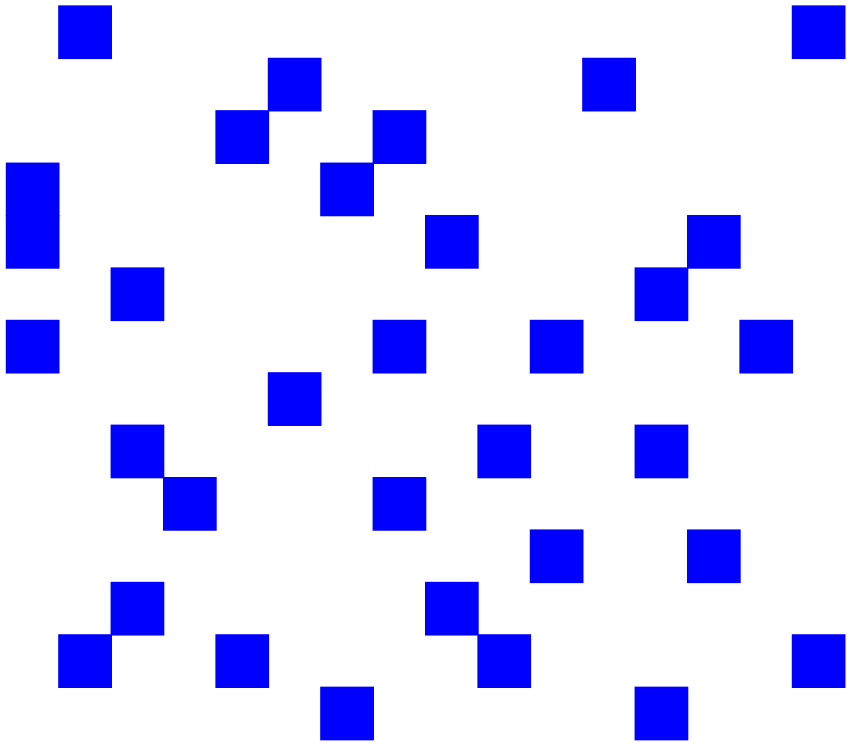
Storage requirements:

$$\Theta(2nz + row\_jumps + 1).$$

Ref.: Yzelman and Bisseling, "A cache-oblivious sparse matrix–vector multiplication scheme based on the Hilbert curve", Progress in Industrial Mathematics at ECMI 2010, pp. 627-634 (2012).

Sequential SpMV

Is cache-obliviousness on the level of nonzeroes required?

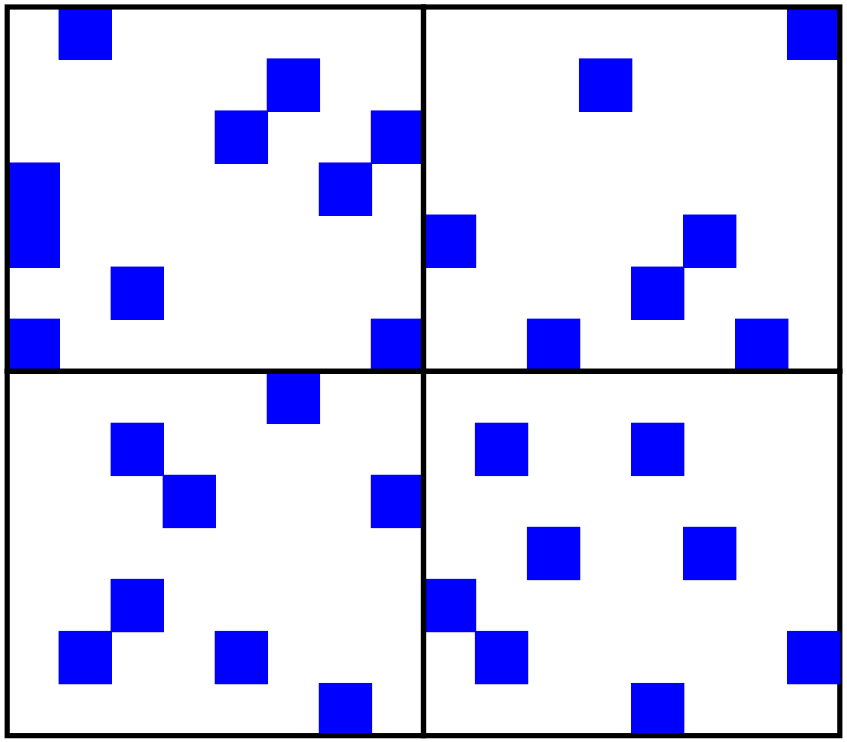


Sparse blocking may have advantages:

- corresponding vector elements will fit into cache,
- may apply low-level optimisations within blocks.

Sequential SpMV

Is cache-obliviousness on the level of nonzeroes required?

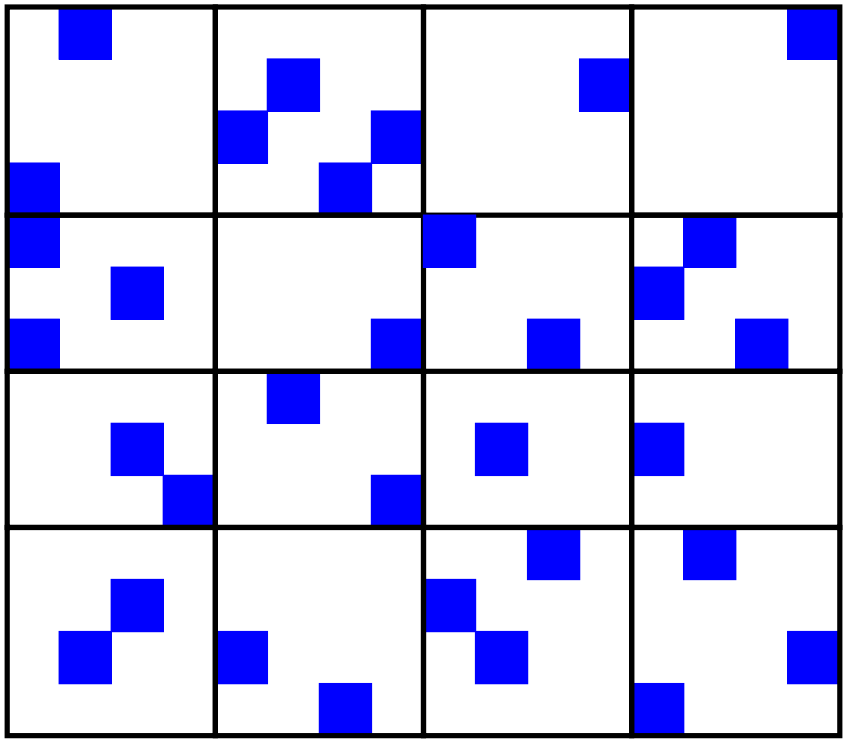


Sparse blocking may have advantages:

- corresponding vector elements will fit into cache,
- may apply low-level optimisations within blocks.

Sequential SpMV

Is cache-obliviousness on the level of nonzeroes required?

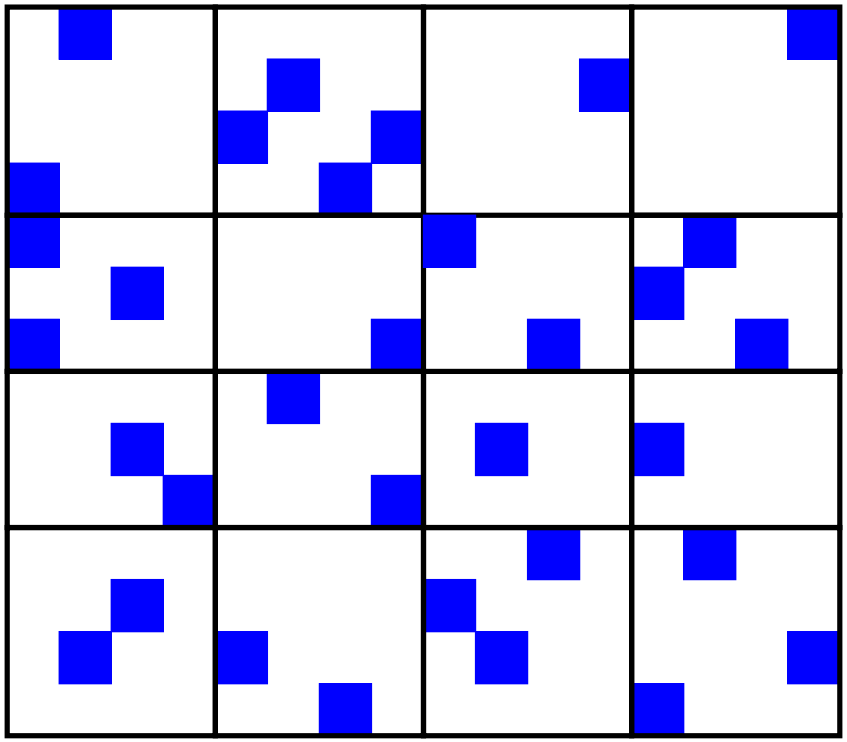


Sparse blocking may have advantages:

- corresponding vector elements will fit into cache,
- may apply low-level optimisations within blocks.

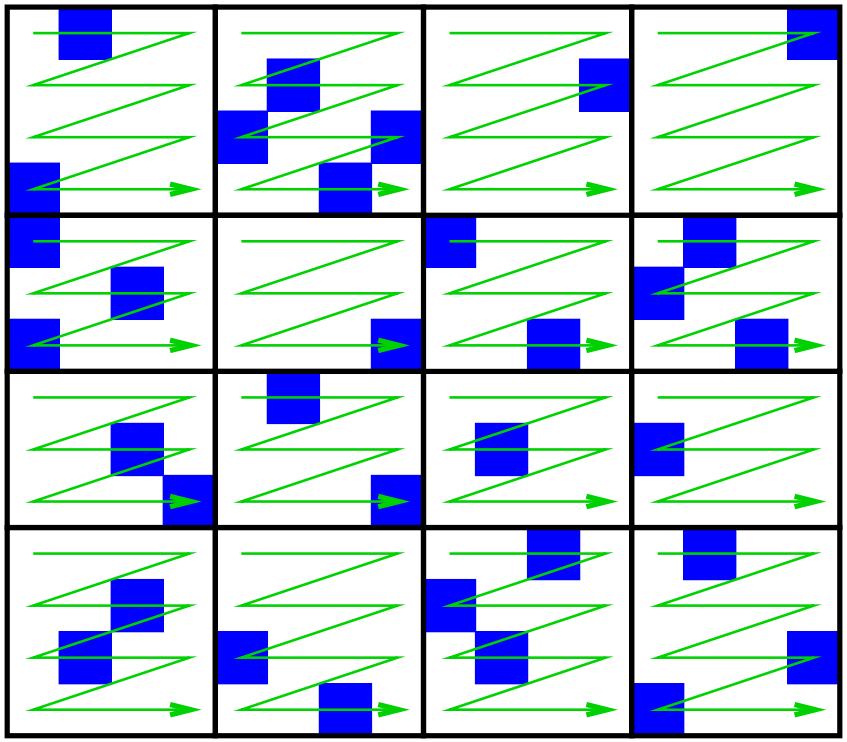
Sequential SpMV

Space-filling curves on top, full cache-obliviousness:



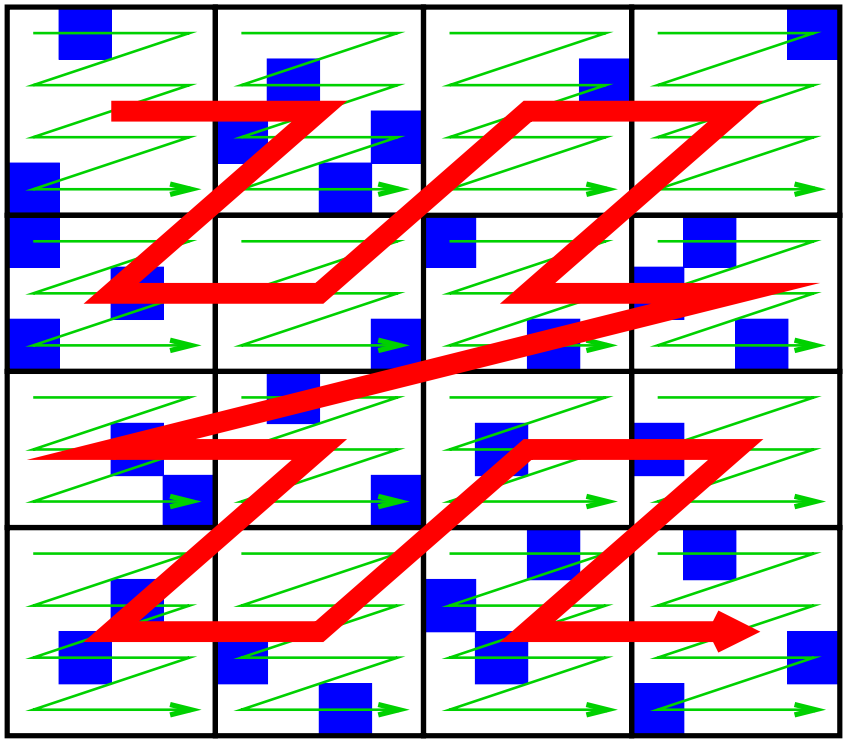
Sequential SpMV

Space-filling curves on top, full cache-obliviousness:



Sequential SpMV

Space-filling curves on top, full cache-obliviousness:

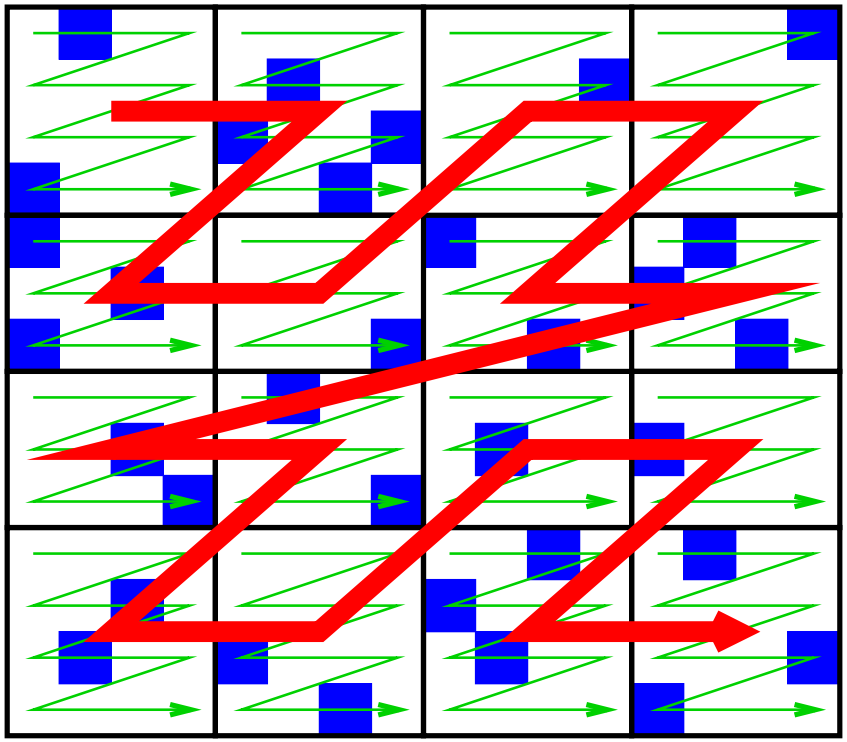


(Using the Z-curve and dense BLAS)

Ref.: Lorton and Wise, "Analyzing block locality in Morton-order and Morton-hybrid matrices", SIGARCH Computer Architecture News, 35(4), pp. 6-12 (2007).

Sequential SpMV

Space-filling curves on top, full cache-obliviousness:

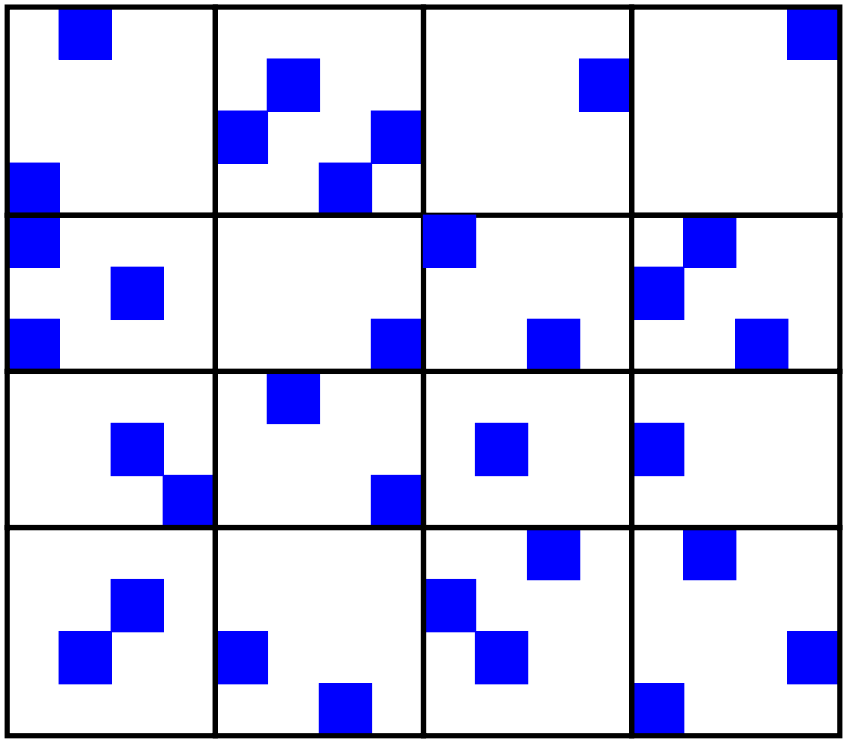


(Using the Z-curve, a quad-tree, and CRS within blocks)

Ref.: Martone, Filippone, Tucci, Paprzycki, and Ganzha, "Utilizing recursive storage in sparse matrix-vector multiplication - preliminary considerations", Proceedings of the ISCA 25th International Conference on Computers and Their Applications (CATA), pp 300-305 (2010).

Sequential SpMV

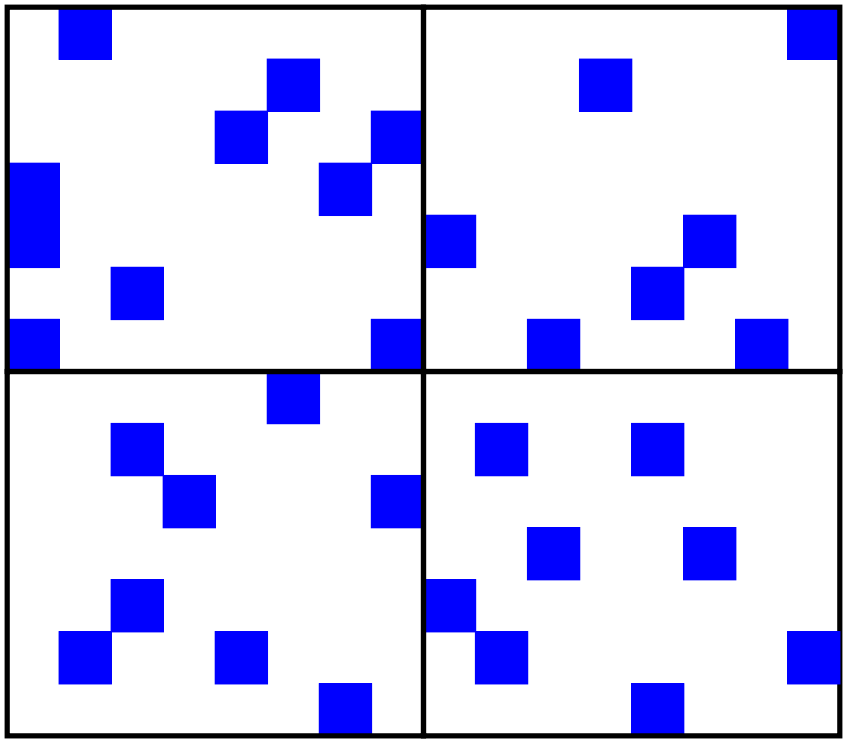
Space-filling curves on top, full cache-obliviousness:



But how much storage does CRS within blocks require?

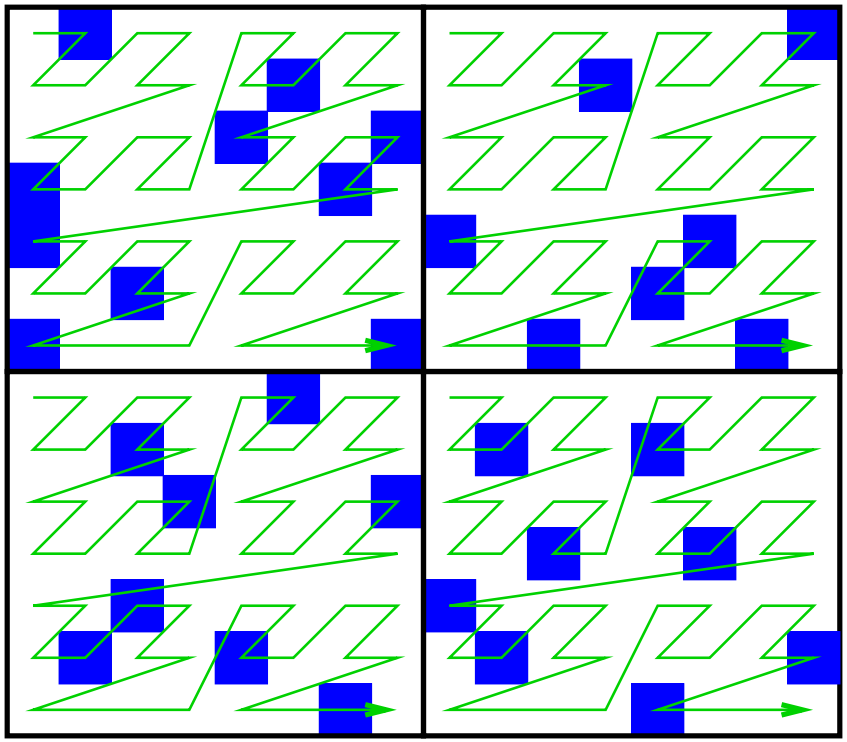
Sequential SpMV

Space-filling curves within can be stored efficiently:



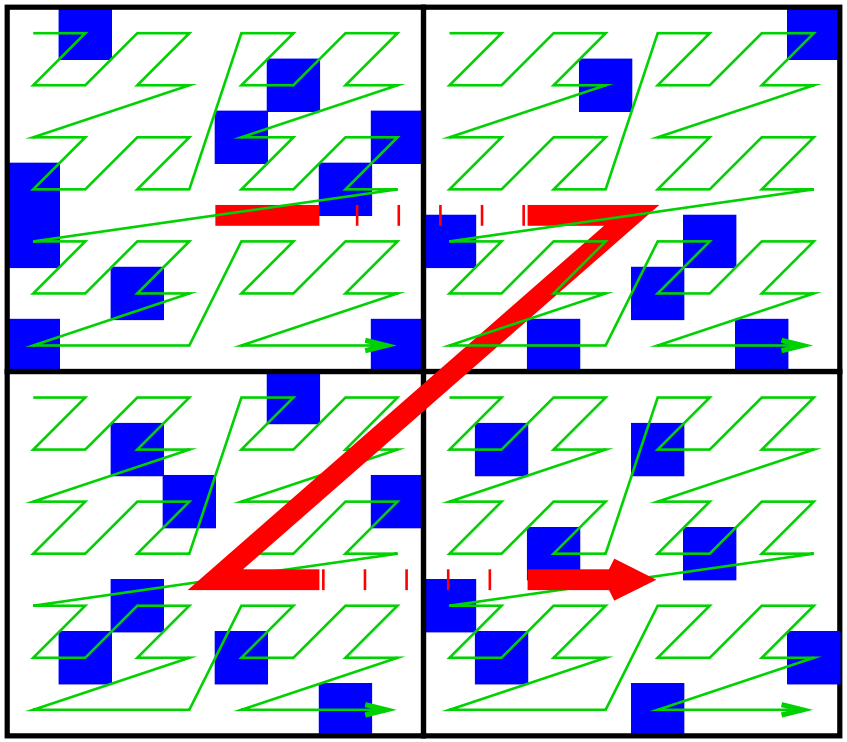
Sequential SpMV

Space-filling curves within can be stored efficiently:



Sequential SpMV

Space-filling curves within can be stored efficiently:

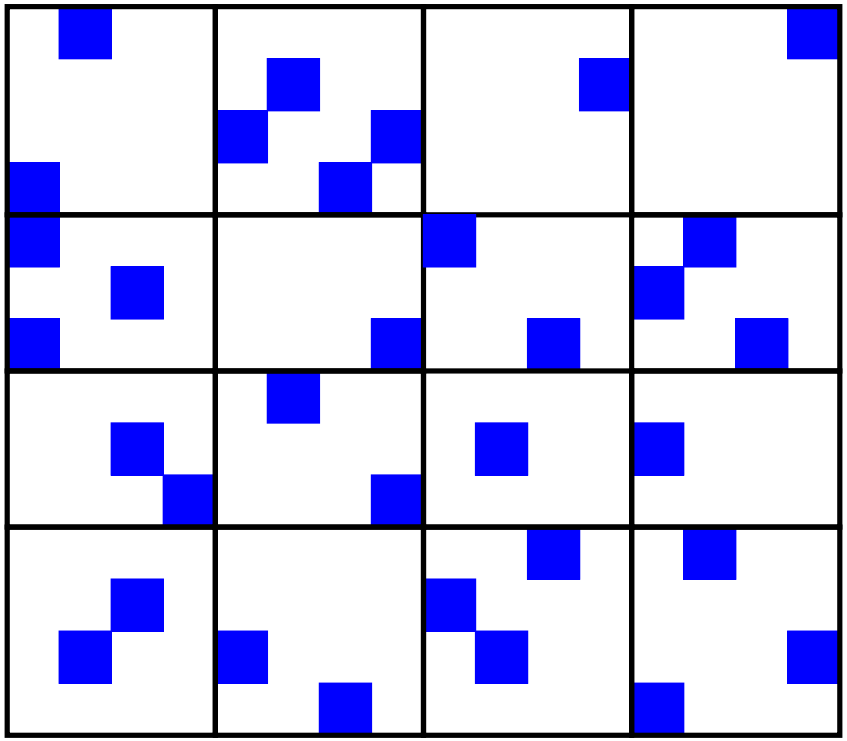


(Stored using Compressed Sparse Blocks, CSB)

Ref.: Buluç, Williams, Oliker, and Demmel, "Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication", Proc. of the Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, pp. 721-733 (2011).

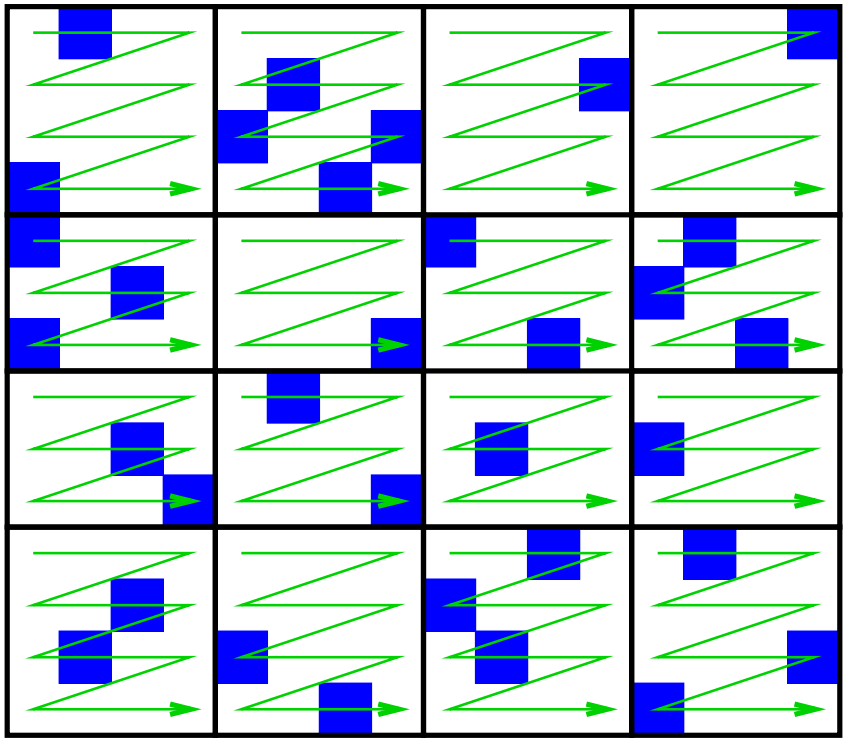
Sequential SpMV

Efficient storage with full cache-obliviousness:



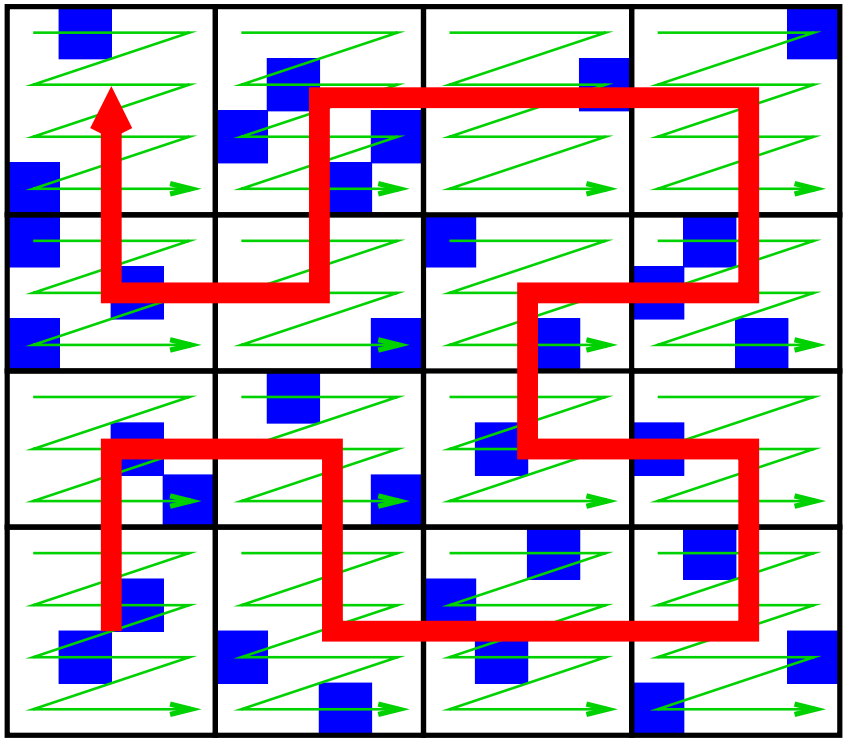
Sequential SpMV

Efficient storage with full cache-obliviousness:



Sequential SpMV

Efficient storage with full cache-obliviousness:



(Using compressed BICRS, CBICRS)

Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Transactions on Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Sequential SpMV

Overview of sparse matrix data structures:

Name	Additional storage compared to **flat** CRS
blocked CRS	$\mathcal{O}(m^2)$
COO	$\mathcal{O}(nz - m)$
BICRS	$\mathcal{O}(row\_jumps - m)$
CSB[1]	0
CBICRS[2]	$\mathcal{O}(n + m - nz)$

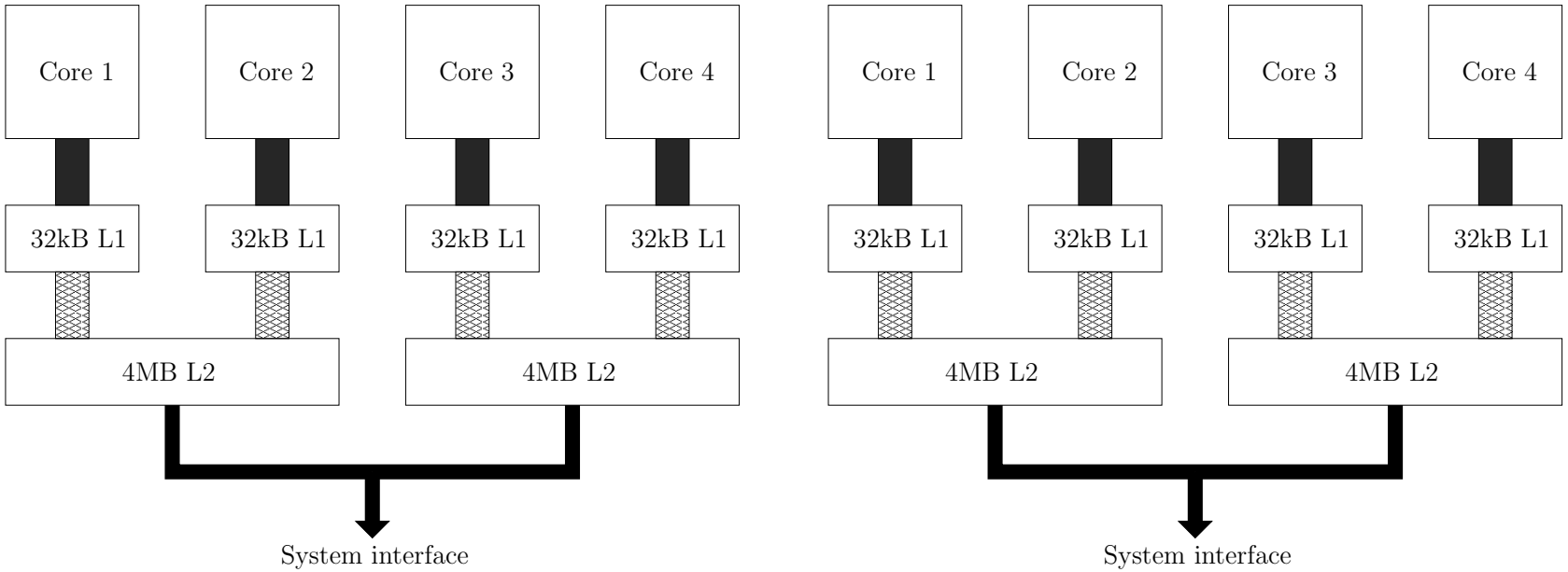
(1: for appropriately chosen blocking size)

Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Ref.: Buluç, Williams, Oliker, and Demmel, "Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication", Proc. of the Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, pp. 721-733 (2011).

Parallel SpMV

Non-uniform memory access becomes a problem:

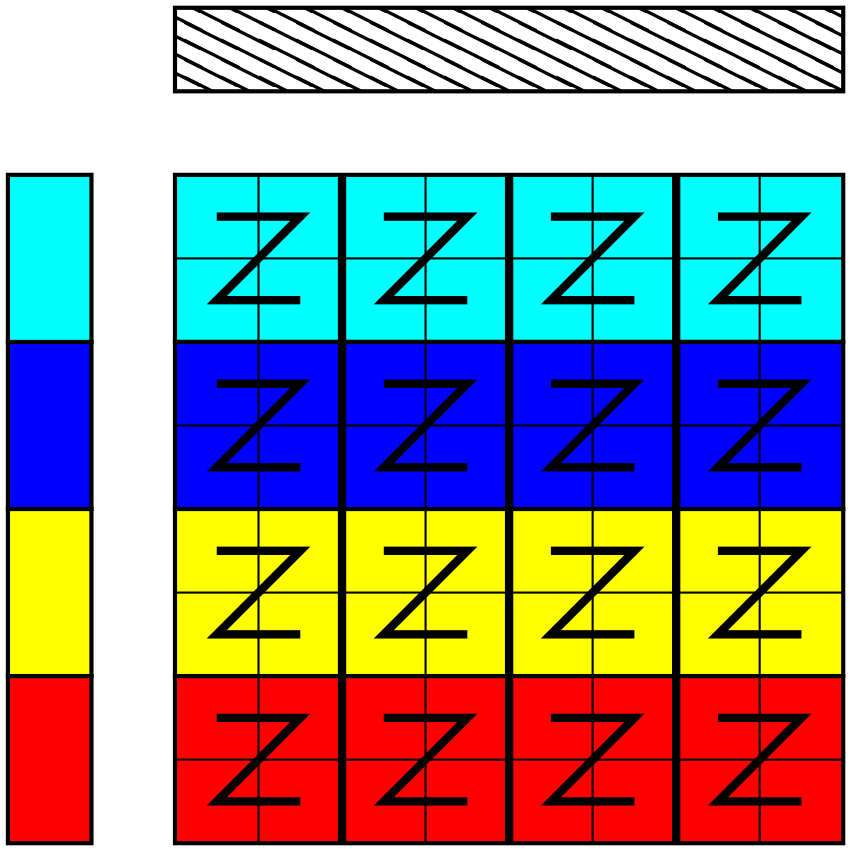


We had cache-oblivious SpMVs;

do **core-oblivious** methods exist?

Parallel SpMV

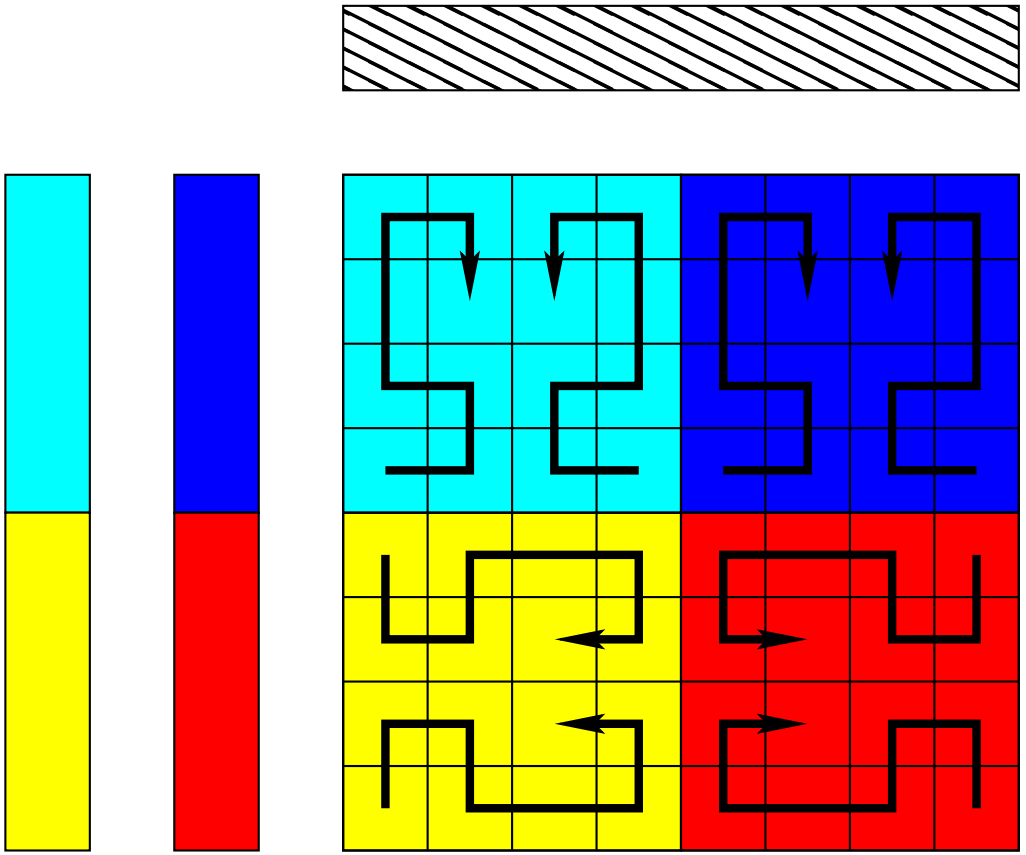
'Core-oblivious', for parallelisation (Cilk CSB):



Ref.: Buluç, Williams, Oliker, and Demmel, "Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication", Proc. of the Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International, pp. 721-733 (2011).

Parallel SpMV

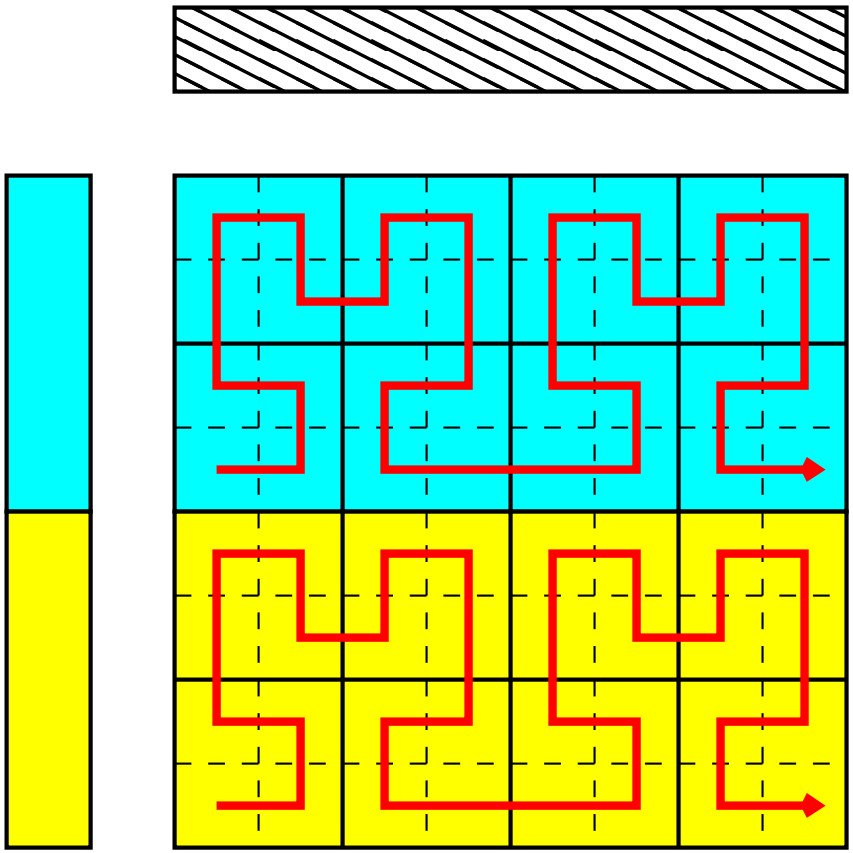
'Core-oblivious', for cache-efficiency (PThreads 0D):



Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Parallel SpMV

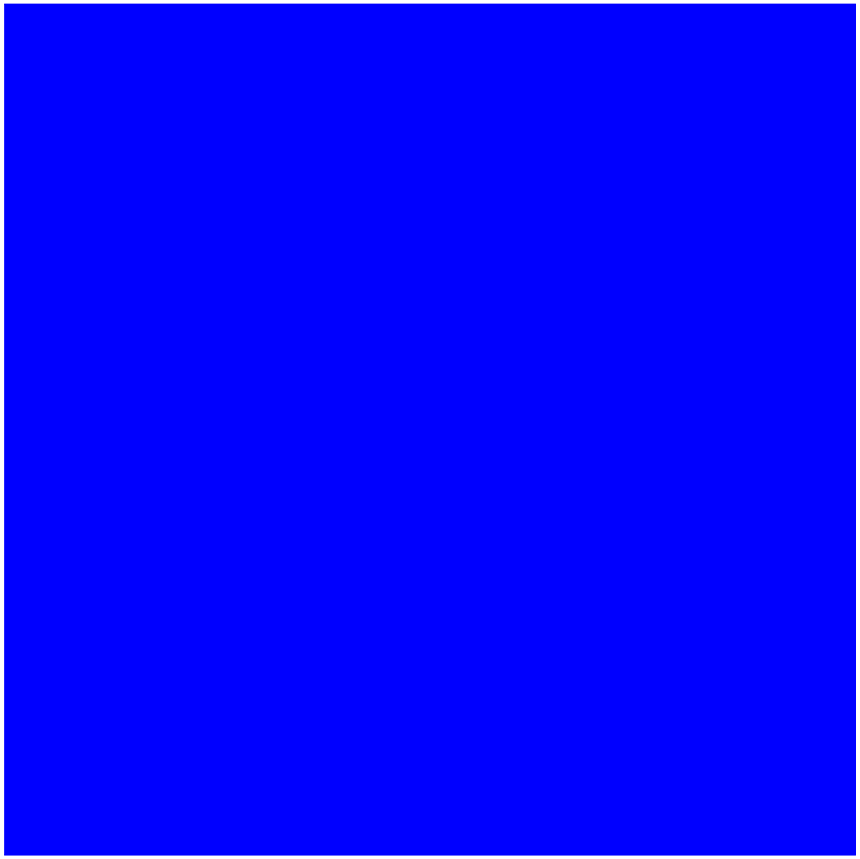
'Core-oblivious', good caching and parallelisation (PThr. 1D):



Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Parallel SpMV

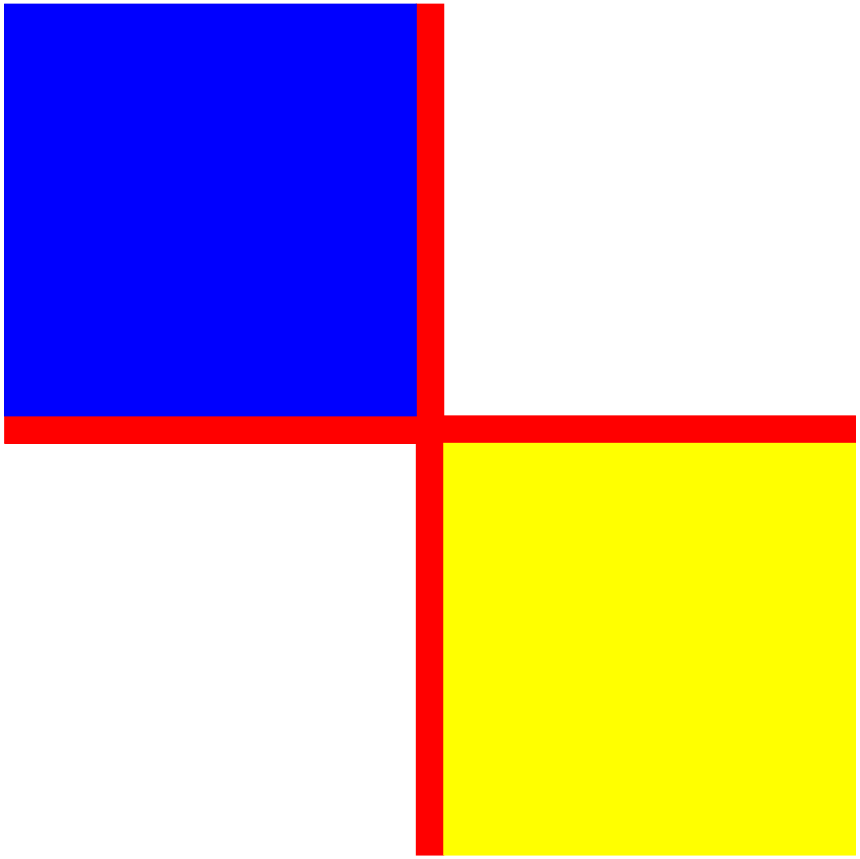
Using **partitioning and reordering** (BSP 2D):



Ref.: Yzelman and Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication", Parallel Computing 37(12), pp. 806-819 (2011).

Parallel SpMV

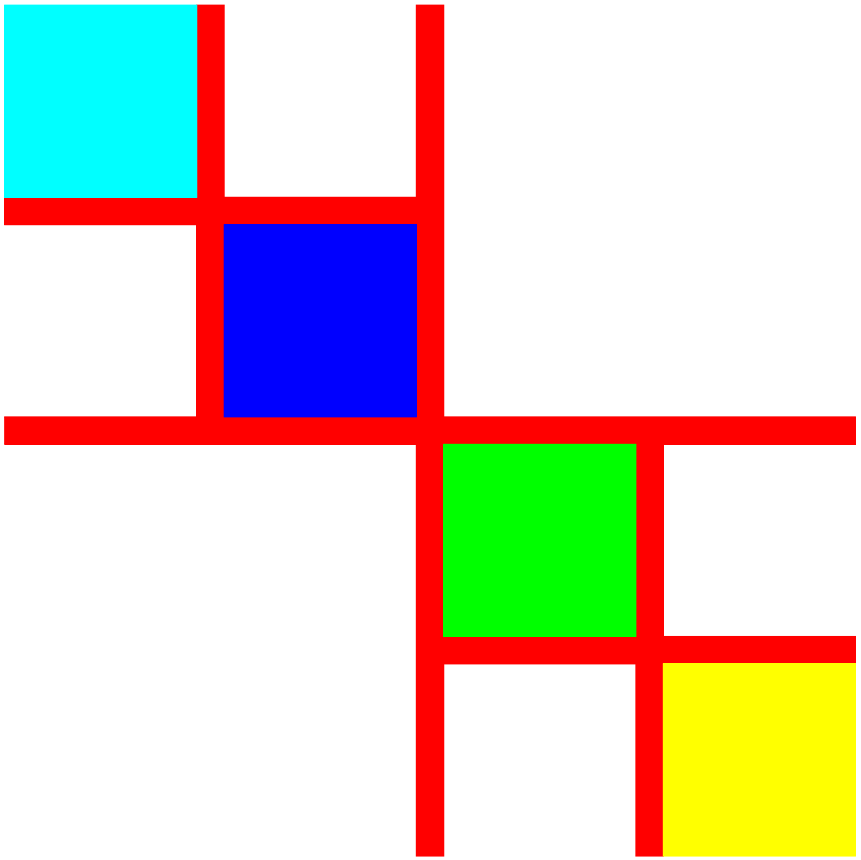
Using **partitioning and reordering** (BSP 2D):



Ref.: Yzelman and Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication", Parallel Computing 37(12), pp. 806-819 (2011).

Parallel SpMV

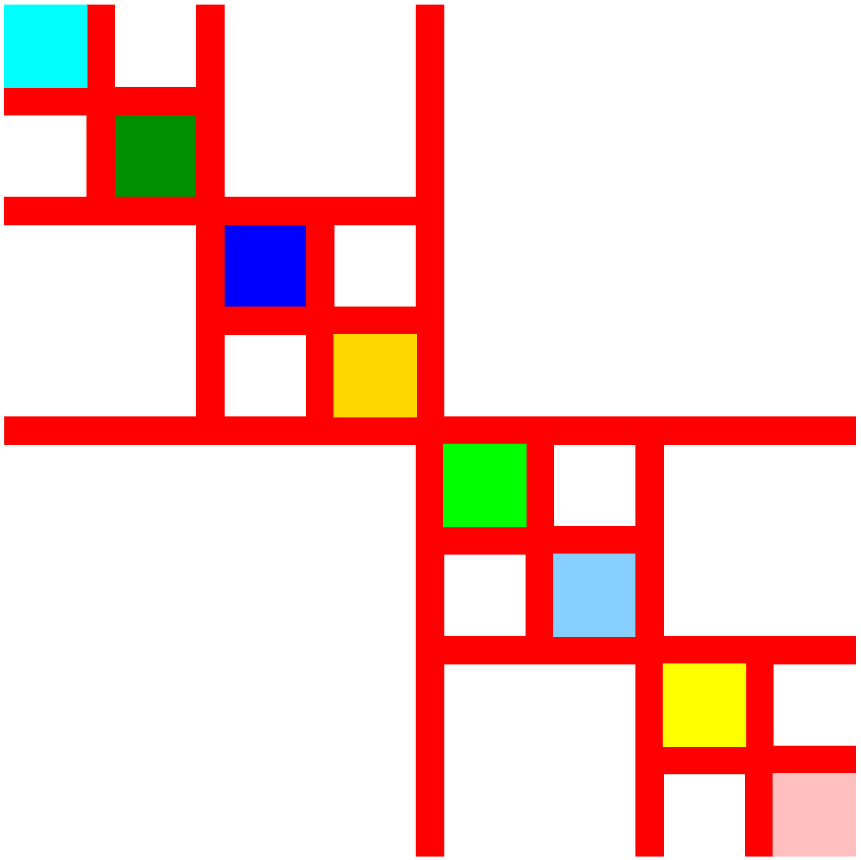
Using **partitioning and reordering** (BSP 2D):



Ref.: Yzelman and Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication", Parallel Computing 37(12), pp. 806-819 (2011).

Parallel SpMV

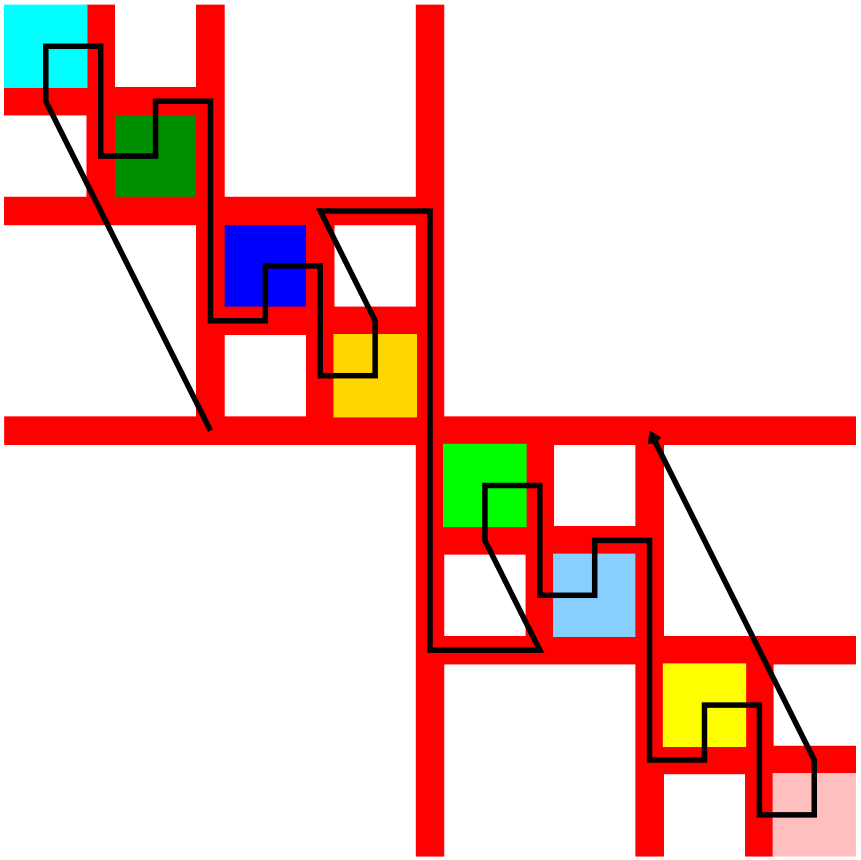
Using **partitioning and reordering** (BSP 2D):



Ref.: Yzelman and Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication", Parallel Computing 37(12), pp. 806-819 (2011).

Parallel SpMV

Using partitioning, reordering, & '**space-avoiding**' curves:

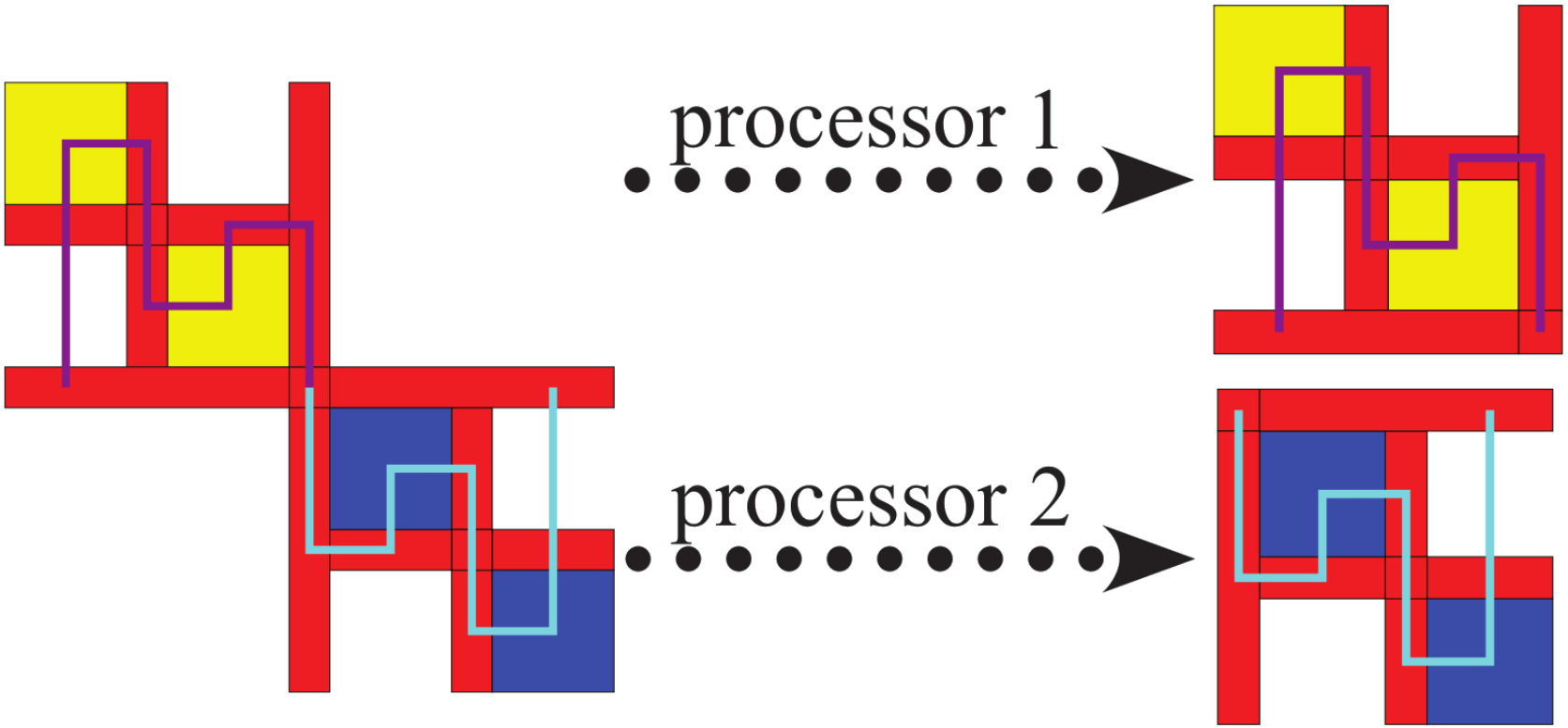


(In sequential, this is fully cache-oblivious)

Ref.: Yzelman and Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication", Parallel Computing 37(12), pp. 806-819 (2011).

Parallel SpMV

Leveraging the partitioning for coarse-grained parallelism:



(Unlike the previous methods, this method scales fully)

Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Parallel SpMV

Using **bulk synchronous parallel** (BSP), this 2D SpMV strategy is easily implemented. Each processor executes:

for each local $a_{ij} \neq 0$ with non-local x_j **do**
    bsp_get x_j from remote process
bsp_sync

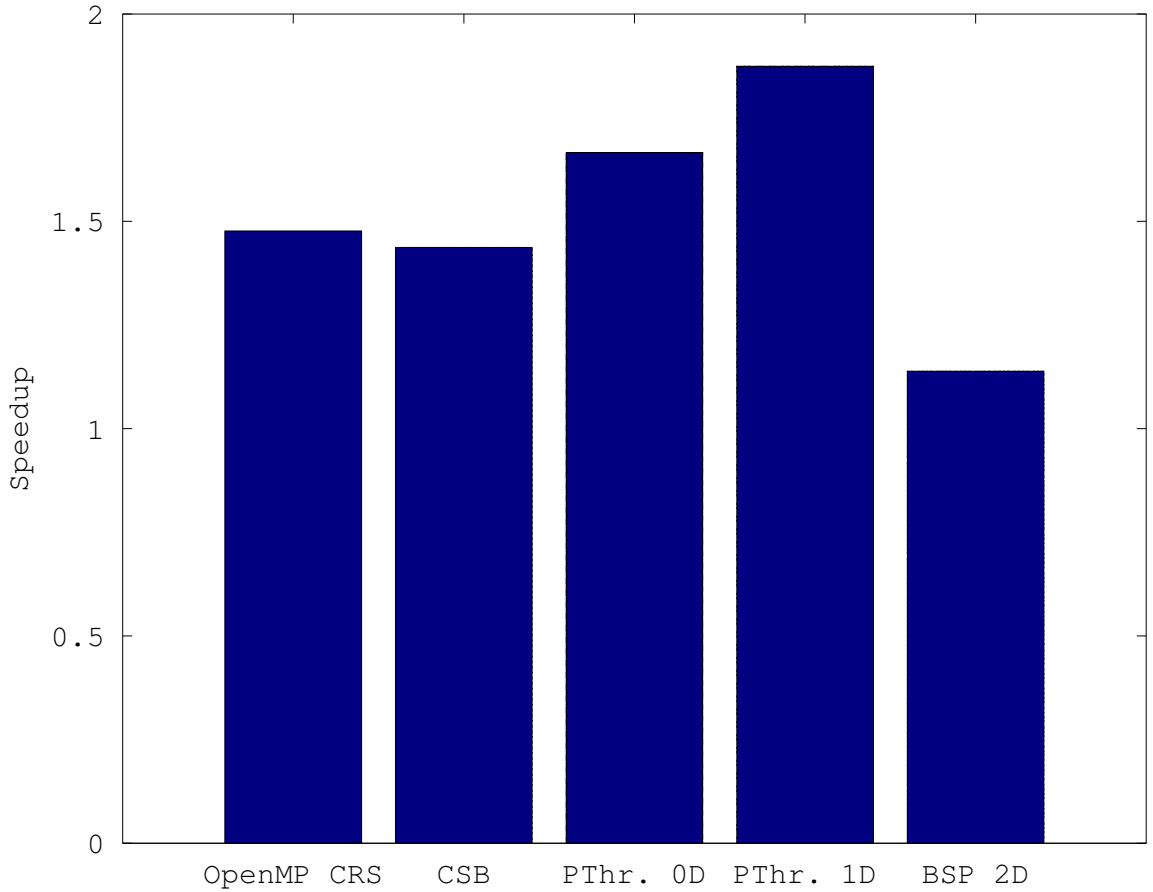
multiply $y = Ax$ using local nonzeroes only
for each local $a_{ij} \neq 0$ with non-local y_i **do**
    bsp_send (i, y_i) to remote process
bsp_sync

while I have messages in queue ($bsp\_qsize > 0$) **do**
    bsp_move (i, α) from queue and add α to local y_i

Ref.: Yzelman, Bisseling, Roose, and Meerbergen, "MulticoreBSP for C: a high-performance library for shared-memory parallel programming", International Journal of Parallel Programming, doi: 10.1007/s10766-013-0262-9, in press (2013).

Experiments

Results on one matrix (ldoor) on a quad-core processor:



(Speedups are relative to sequential CRS.)

Experiments

We use a small subset of matrices to compare methods:

Matrix	Size	Nonzeroes	Origin
Freescale1	3 428 755	17 052 626	Semiconductor industry
adaptive	6 815 744	27 248 640	Numerical simulation
ldoor	952 203	42 493 817	Structural engineering
wiki2007	3 566 907	45 030 389	Link matrix
road_usa	23 947 347	57 708 624	Road network

We run each method on 900 times and obtain the average execution time, on both a 40-core and a 64-core machine.

Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Ref.: Yzelman, Bisseling, Roose, and Meerbergen, "MulticoreBSP for C: a high-performance library for shared-memory parallel programming", International Journal of Parallel Programming, doi: 10.1007/s10766-013-0262-9, in press (2013).

Experiments

We report average speedups relative to sequential CRS:

	4 x 10	8 x 8
OpenMP CRS	8.8	7.2
PThread 0D	3.5	3.2
PThread 1D	13.6	20.0
Cilk CSB	22.9	26.9
BSP 2D	21.3	30.8

4 x 10: HP DL-580, 4 sockets, 10 core Intel Xeon E7-4870
8 x 8 : HP DL-980, 8 sockets, 8 core Intel Xeon E7-2830

Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).

Ref.: Yzelman, Bisseling, Roose, and Meerbergen, "MulticoreBSP for C: a high-performance library for shared-memory parallel programming", International Journal of Parallel Programming, doi: 10.1007/s10766-013-0262-9, in press (2013).

Conclusion

We demonstrated the use of space-filling curves within sparse matrix computations.

Thank you for your attention!

For software, see:

- The Sparse Library (sparse matrix datastructures):
  `http://albert-jan.yzelman.net/software/#SL`
- The MulticoreBSP library:
  `http://www.multicorebsp.com`
- The BSP 2D code:
  `http://albert-jan.yzelman.net/software/#HPBSP`

Sequential SpMV

Overview of **flat** data structures (all values are in bits).
Here, $\tilde{m}$ is the number of nonempty rows in A.

Name	Extra storage requirement relative to CRS	
COO	$\lceil \log_2 n \rceil \, (nz - m) \gg 0$, typically	
BICRS	$\lceil \log_2 n \rceil \, (row\_jumps - m) \leq$ COO	
CRS	0	
BICRS	$\lceil \log_2 n \rceil \, (\tilde{m} - m) \leq 0$	(note[1])

(In practice, $\lceil \log_2 n \rceil$ is rounded up to $8, 16, 32,$ or 64 bits.)

[1]: if the nonzeroes are in row-major order.

Ref.: Yzelman and Bisseling, "Two-dimensional cache-oblivious sparse matrix-vector multiplication", Parallel Computing 37(12), pp. 806-819 (2011).

Sequential SpMV

Overview of **blocked** data structures (values are in bits):

Name	Extra storage requirement relative to **flat** CRS	
CRS	$\lceil \log_2 n \rceil \, m^2/\sqrt{n} = \mathcal{O}(m^2)$	(note[1,2,3])
COO	$\lceil \log_2 n \rceil \, (nz - m) \gg 0$, typically	
BICRS	$\lceil \log_2 n \rceil \, (row\_jumps - m) \leq$ COO	(can be < 0)
CSB	0	(note[1,2])
CBICRS	$\frac{1}{2} \lceil \log_2 n \rceil \, (n + \tilde{m} - nz) < 0$, typically	(note[1,3])

(In practice, $\lceil \log_2 n \rceil$ is rounded up to a multiple of 8.)

[1]: if the block size is $\sqrt{n}$.
[2]: if the blocks are in row-major order.
[3]: if the nonzeroes within blocks are in row-major order.

Ref.: Yzelman and Roose, "High-Level Strategies for Parallel Shared-Memory Sparse Matrix–Vector Multiplication", IEEE Trans. Parallel and Distributed Systems, doi: 10.1109/TPDS.2013.31, in press (2013).