Parallel SpMV multiplication

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

December 2013

Acknowledgements

This presentation outlines the paper

A. N. Yzelman and D. Roose, *"High-level strategies for parallel shared-memory sparse matrx–vector multiplication"*, IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS), in press: `http://dx.doi.org/10.1109/TPDS.2013.31`

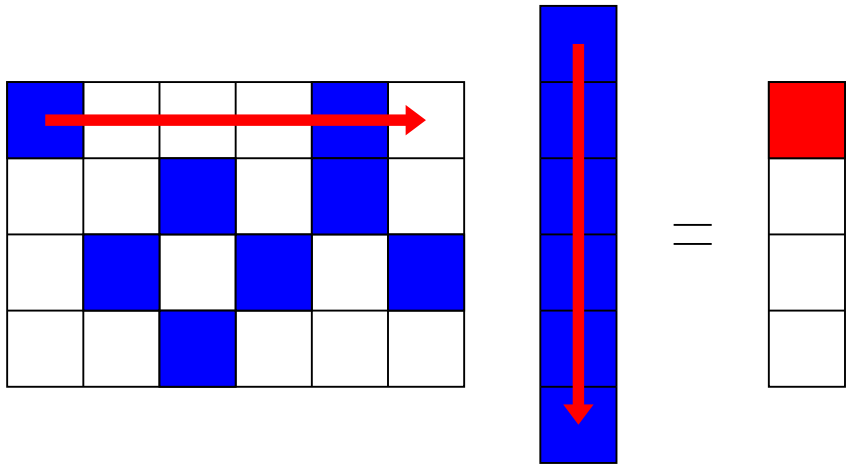
This work is funded by Intel and by the Institute for the Promotion of Innovation through Science and Technology (IWT), in the framework of the Flanders ExaScience Lab, part of Intel Labs Europe.

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

$$Ax = y :$$

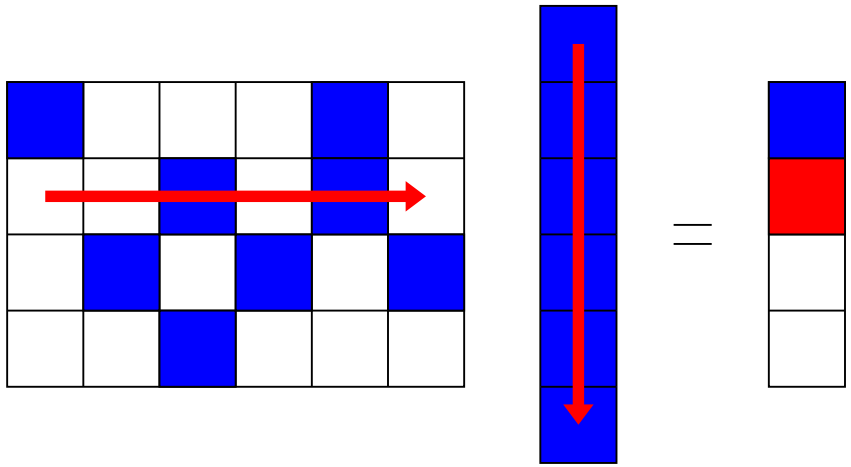


Introduction

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

We consider both sequential and parallel computation of

$$Ax = y :$$

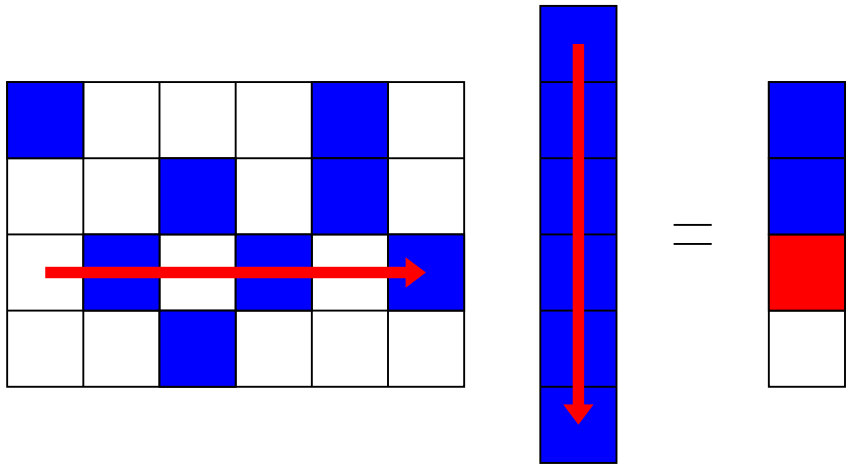


Introduction

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

We consider both sequential and parallel computation of

$$Ax = y :$$

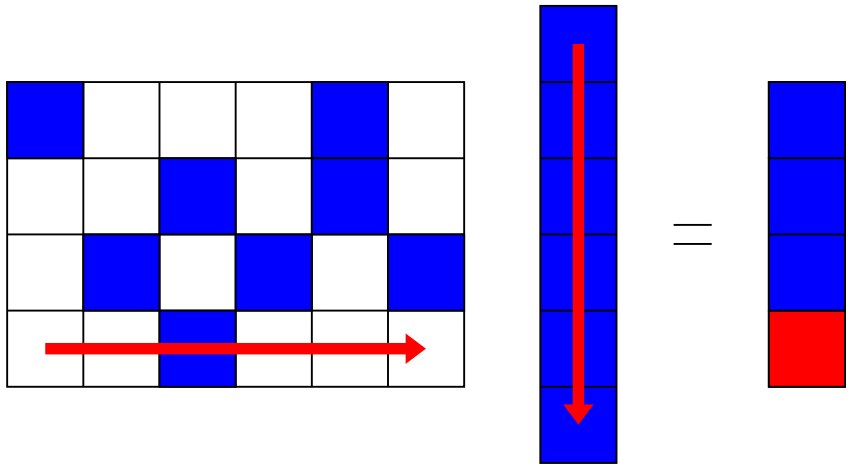


Introduction

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

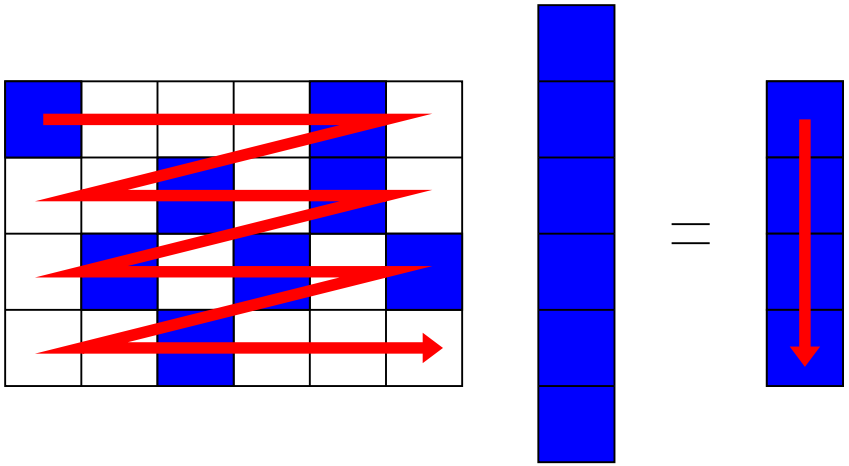
We consider both sequential and parallel computation of

$$Ax = y :$$



Introduction

First obstacle: inefficient cache use

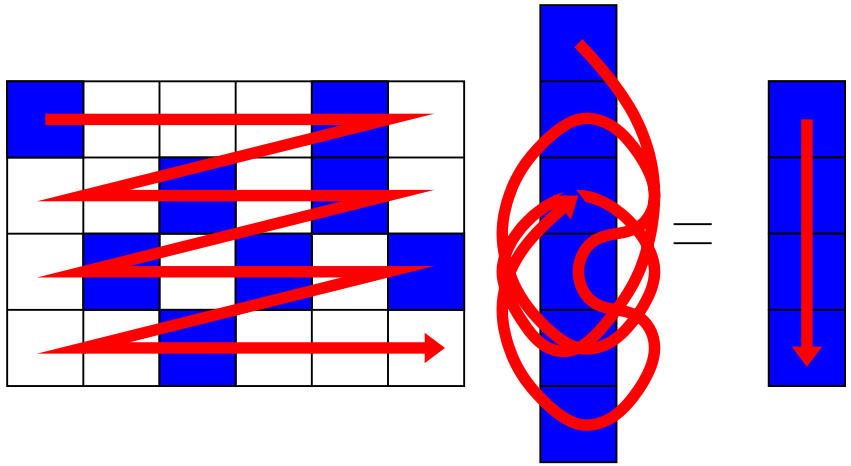


Row-major ordering of nonzeroes:

- linear access of the output vector y;
- ...

Introduction

First obstacle: inefficient cache use

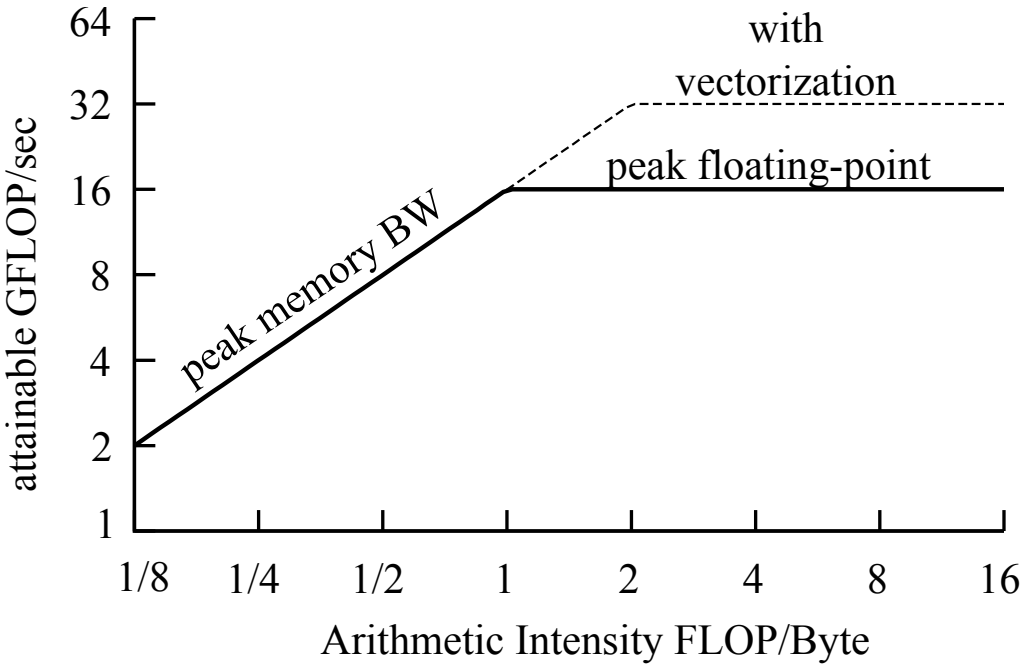


Row-major ordering of nonzeroes:

- linear access of the output vector y;
- irregular access of the input vector x.

Introduction

Second obstacle: the SpMV is bandwidth bound

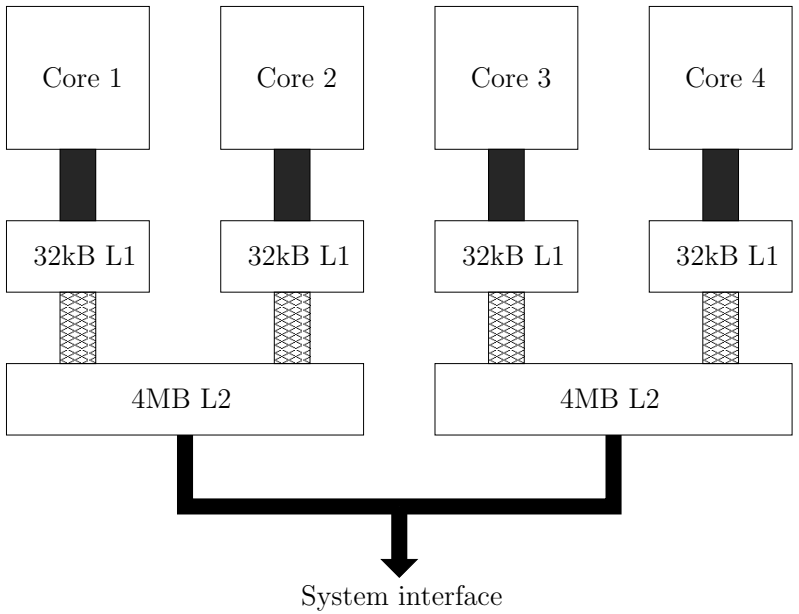


SpMV has **low arithmetic intensity**: 0.2–0.25.
Compression is mandatory!

(Image courtesy of Prof. Wim Vanroose, UA)

Introduction

Third obstacle: NUMA architectures

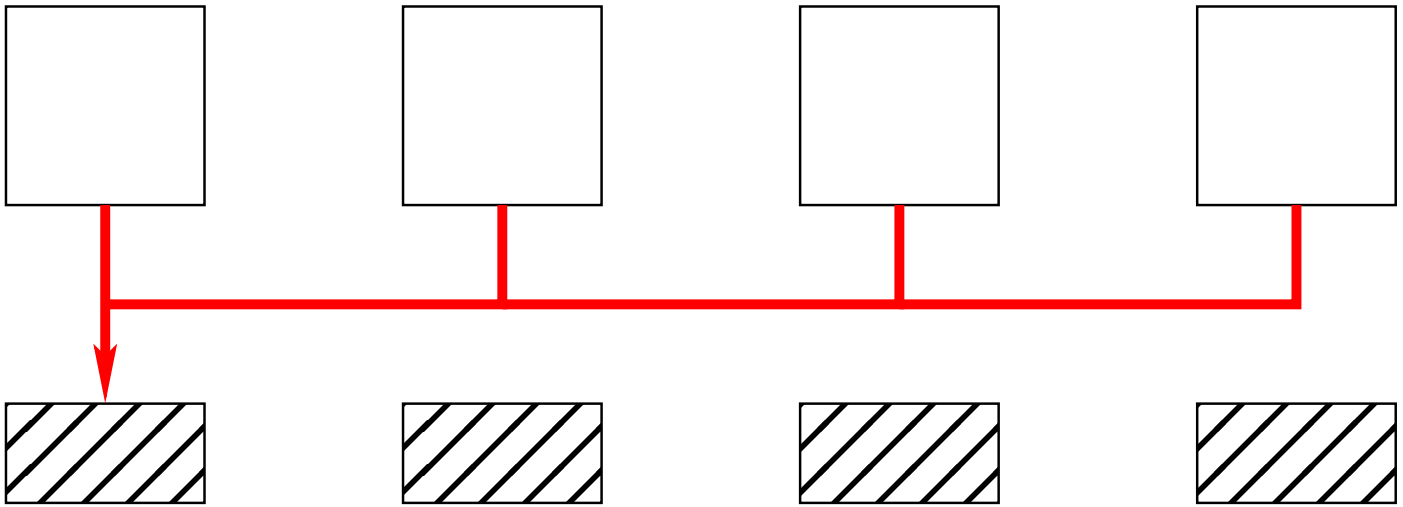


Processor-level NUMAness affects cache behaviour.

Share data from x or y between by neighbouring cores.

Introduction

Third obstacle: NUMA architectures



Socket-level NUMAness affects the effective bandwidth.

If each processor moves data to (and from) the same memory bank, we are bound by the bandwidth of that single memory bank.

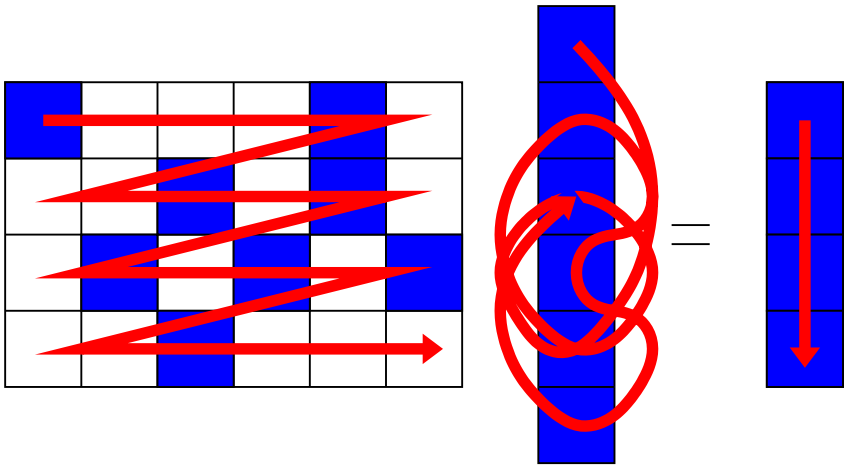
Introduction: summary

Three factors impede creating an efficient shared-memory parallel SpMV multiplication kernel:

1. inefficient cache use,
2. limited memory bandwidth, and
3. non-uniform memory access (NUMA).

Increasing cache-efficiency

Adapt the nonzero order:

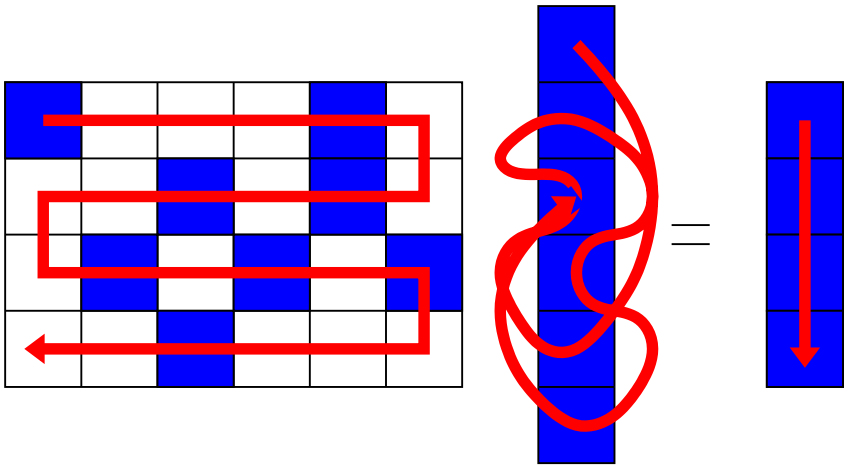


Original situation

- linear access of the output vector y;
- irregular access of the input vector x.

Increasing cache-efficiency

Adapt the nonzero order:



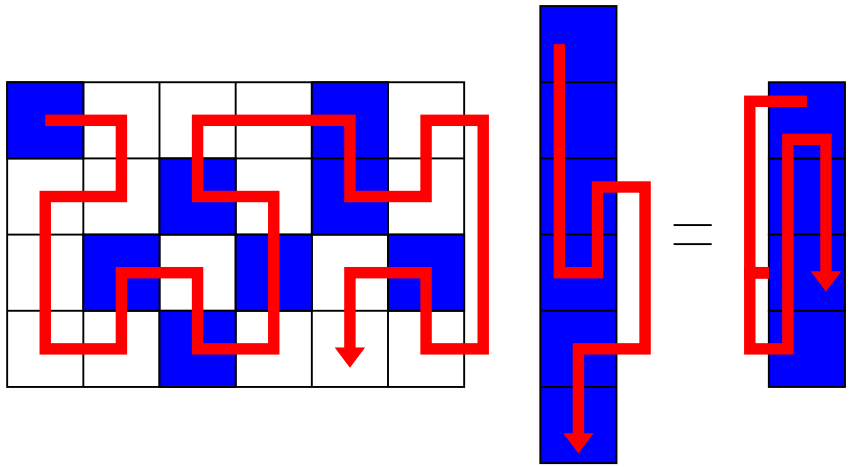
Zig-zag CRS

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

Increasing cache-efficiency

Adapt the nonzero order using space-filling curves:



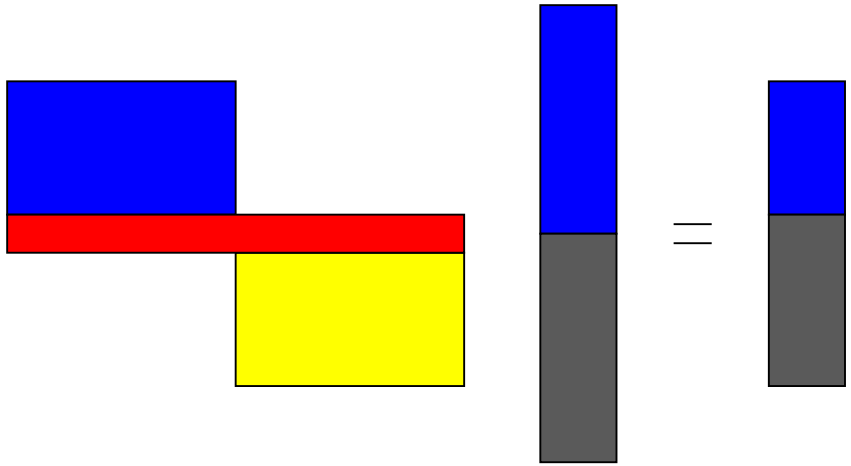
Fractal storage using the coordinate format, COO

- no linear access of y, but
- better combined locality on x and y.

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).

Increasing cache-efficiency

Or reordering matrix rows and columns:



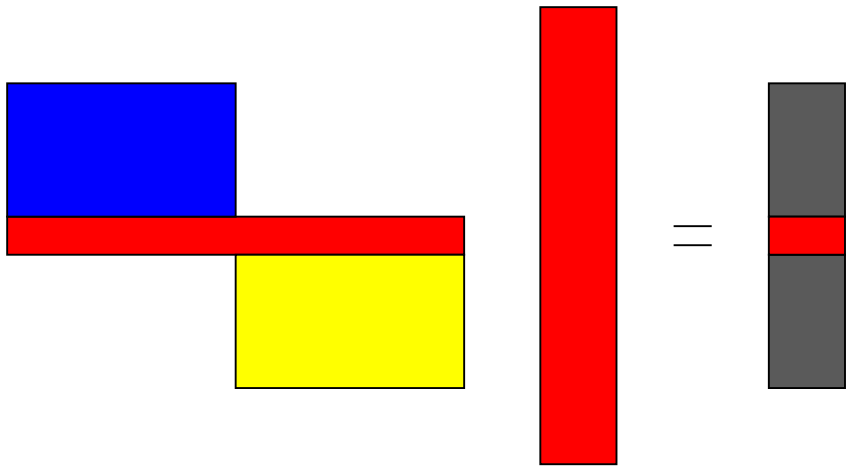
Reordering based on sparse matrix partitioning

- combines with adapting the nonzero order,
- models upper-bound on cache misses (with ZZ-CRS).

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).

Increasing cache-efficiency

Or reordering matrix rows and columns:



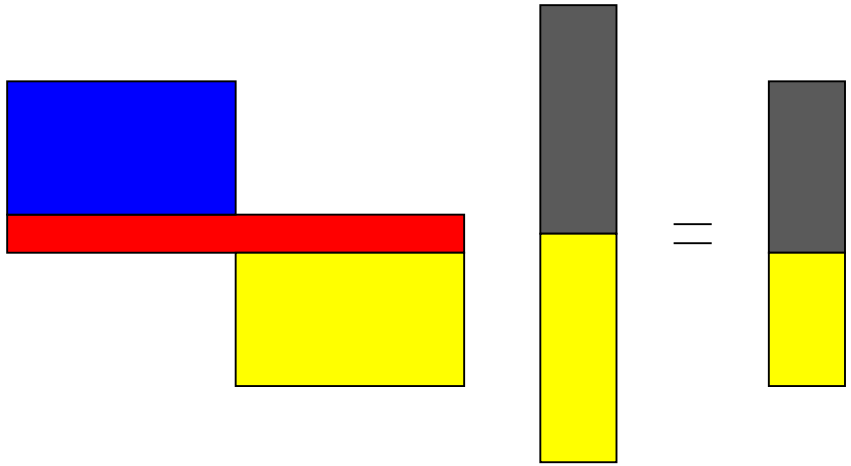
Reordering based on sparse matrix partitioning

- combines with adapting the nonzero order,
- models upper-bound on cache misses (with ZZ-CRS).

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).

Increasing cache-efficiency

Or reordering matrix rows and columns:



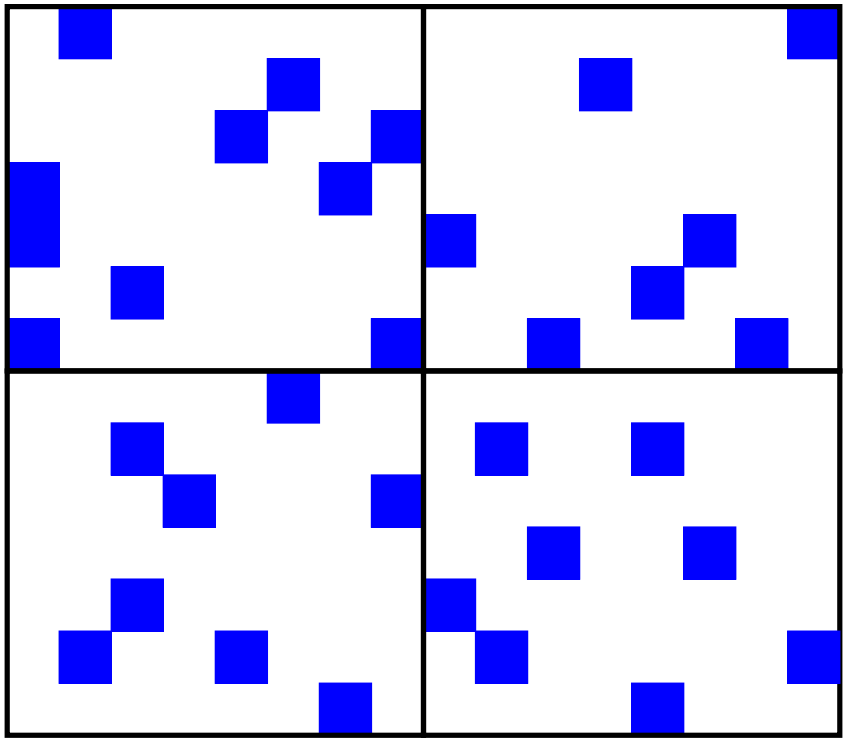
Reordering based on sparse matrix partitioning

- combines with adapting the nonzero order,
- models upper-bound on cache misses (with ZZ-CRS).

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).

Increasing cache-efficiency

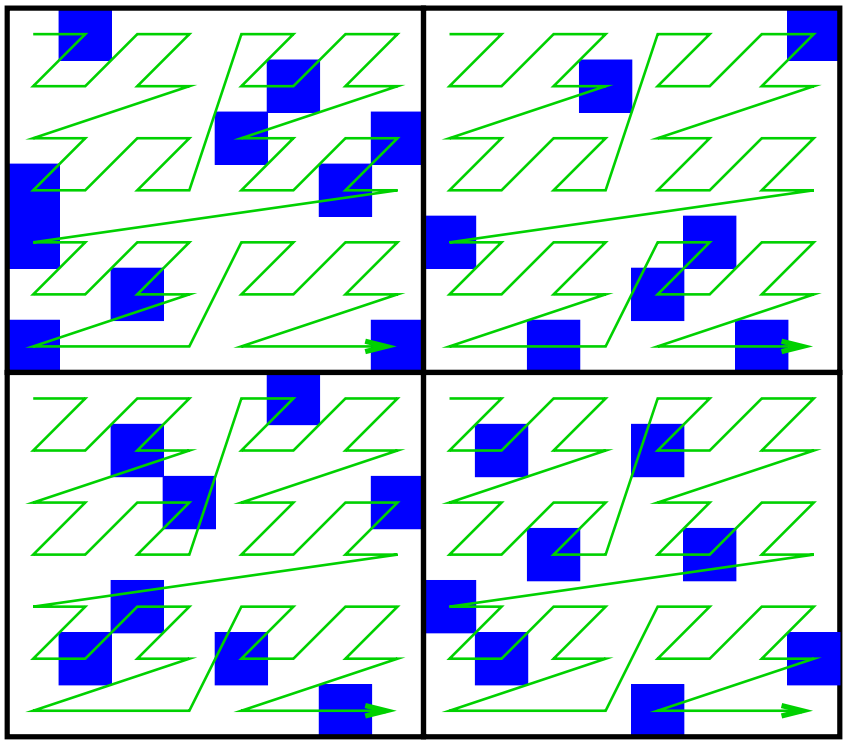
Sparse blocking enhances reordering:



- corresponding vector elements will fit into cache,
- block-wise reordering is faster.

Increasing cache-efficiency

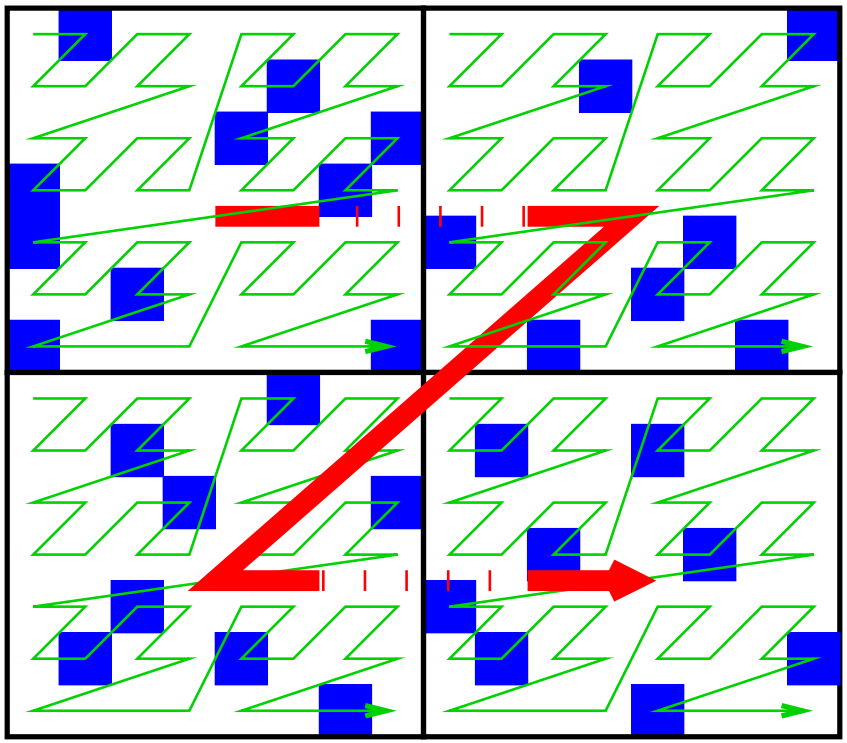
Two options: space-filling curves **within** or without:



(Compressed Sparse Blocks, CSB)

Increasing cache-efficiency

Two options: space-filling curves **within** or without:

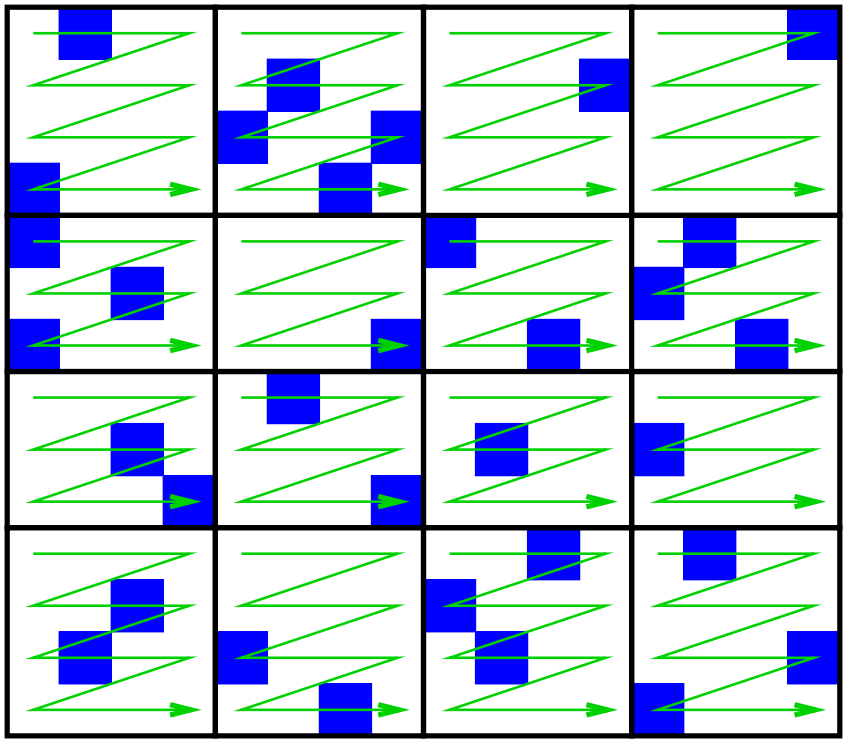


(Compressed Sparse Blocks, CSB)

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

Increasing cache-efficiency

Two options: space-filling curves within or **without**:

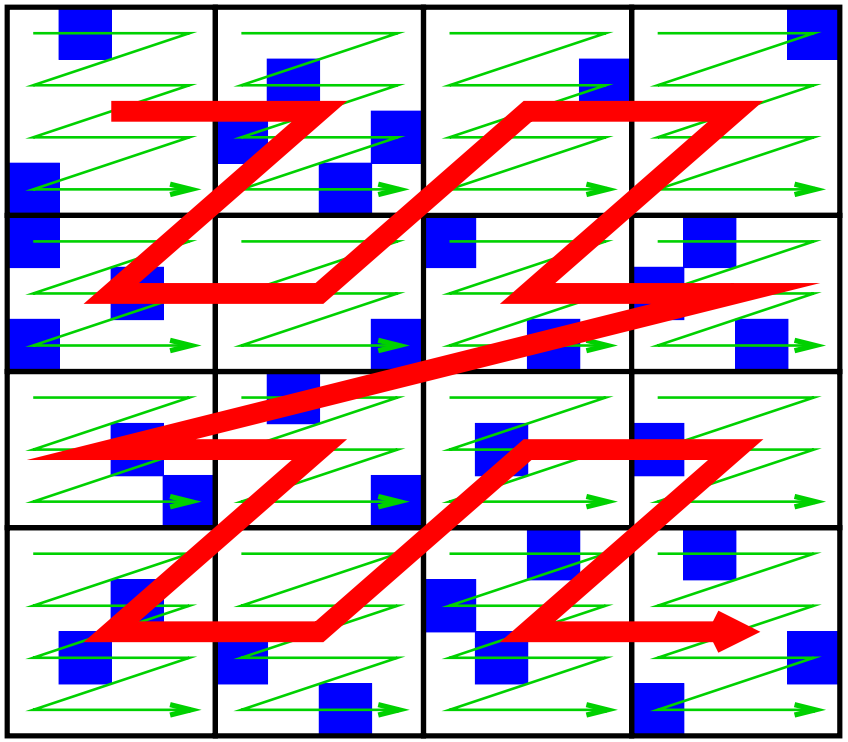


Ref.: Lorton and Wise, "Analyzing block locality in Morton-order and Morton-hybrid matrices", SIGARCH Computer Architecture News, 35(4), pp. 6-12 (2007).

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).

Increasing cache-efficiency

Two options: space-filling curves within or **without**:



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

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).

Increasing cache-efficiency

Techniques:

- adapting the nonzero order,
- adapting the row and column order, and
- sparse blocking.

(These are orthogonal approaches.)

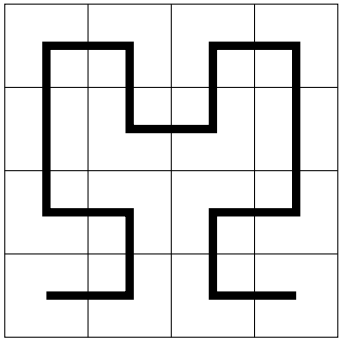
Other techniques:

- cache-aware, matrix-aware blocking:
  e.g., Sparsity and OSKI.

Compression

Assuming a Hilbert order of nonzeroes:

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



The coordinate format (COO):

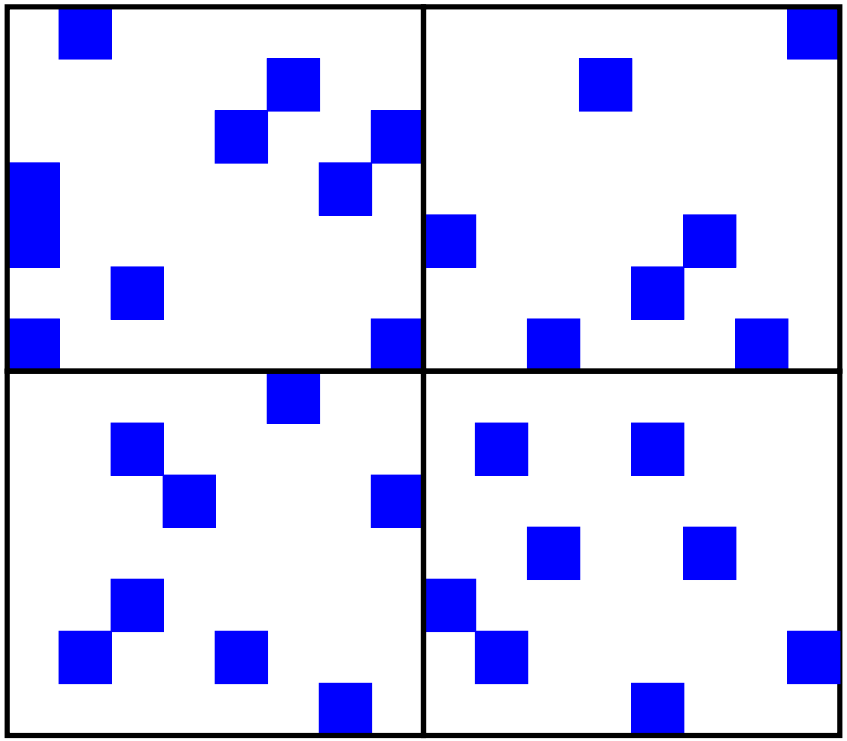
$$A = \begin{cases} V & [7\ 1\ 4\ 1\ 2\ 3\ 3\ 2\ 1\ 1] \\ J & [0\ 0\ 0\ 1\ 2\ 2\ 3\ 3\ 3\ 2] \\ I & [3\ 2\ 0\ 0\ 1\ 0\ 1\ 2\ 3\ 3] \end{cases}$$

Storage requirements:

$$\Theta(3nz).$$

Compression

Compressed COO:



Use COO to store each $\beta \times \beta$ block. Elements of I, J then require only $\log_2 \beta$ bits, for a total storage of $\mathcal{O}(2nz + m)$.

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

Compression

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:

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

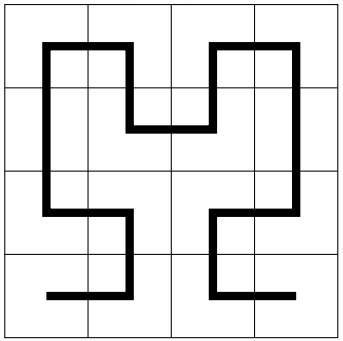
Storage requirements:

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

(nz is the number of nonzeroes in A.)

Compression

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



Bi-directional incremental CRS (BICRS):

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

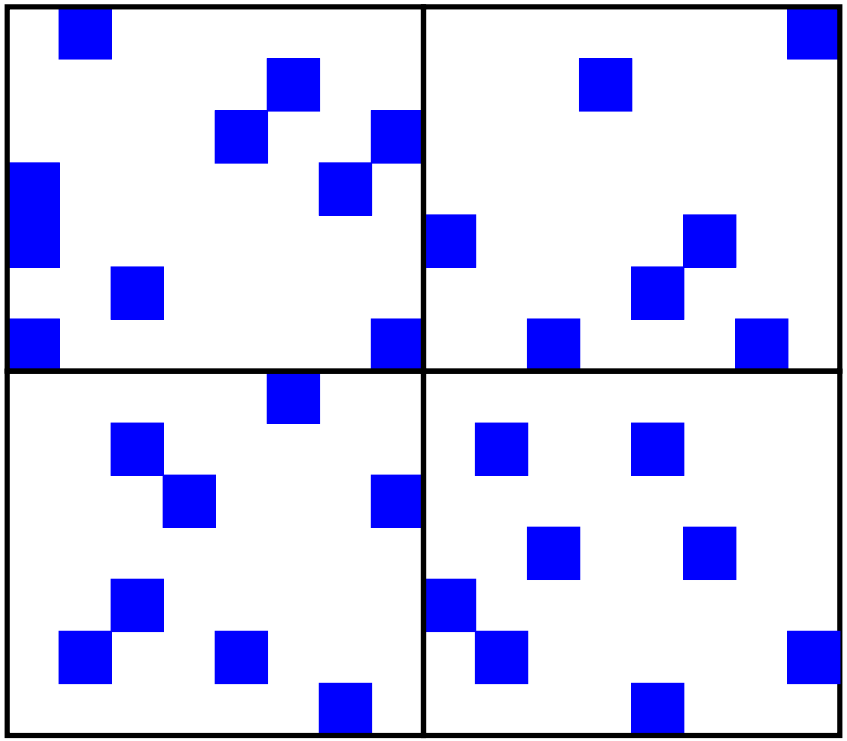
Storage requirements, allowing **arbitrary traversals**:

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

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).

Compression

Compressed BICRS:

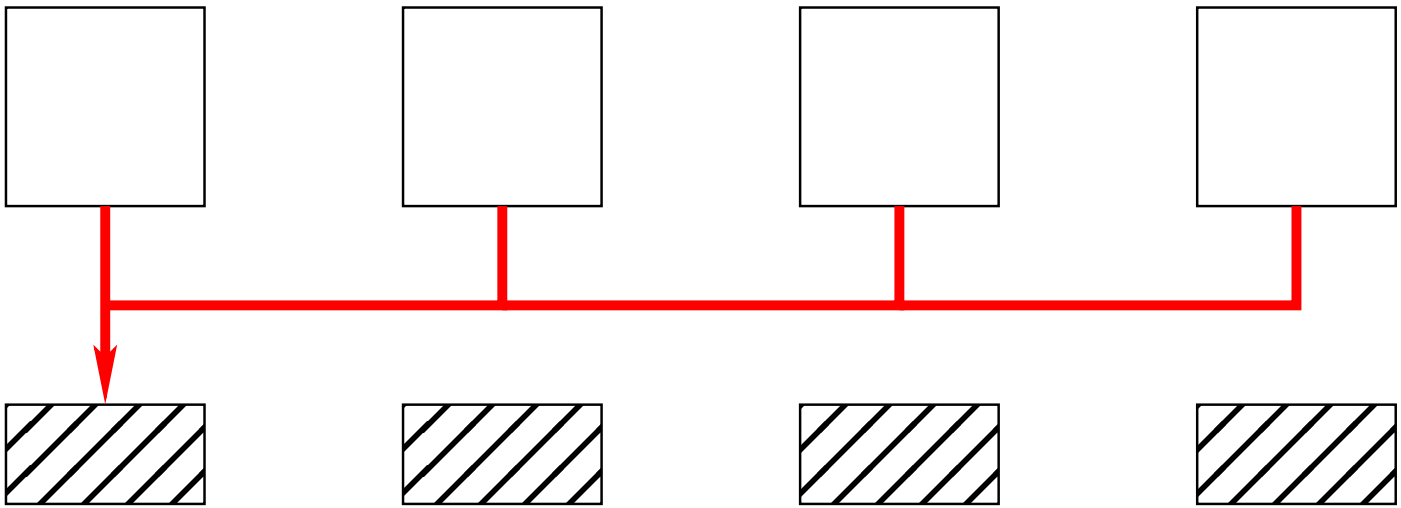


Using **BICRS** to store each $\beta \times \beta$ block, compressing $\Delta I, \Delta J$. Total storage may be **less** than $\mathcal{O}(2nz + m)$.

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).

Distributions and NUMA

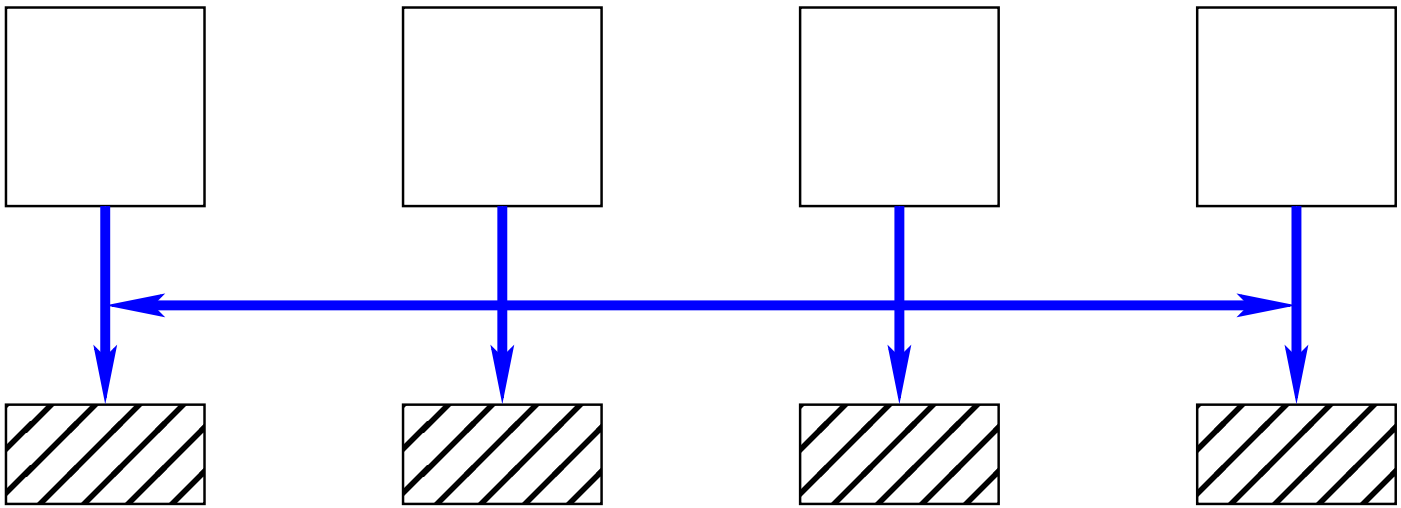
Implicit distribution, centralised local allocation:



If each processor moves data to the same single memory element, the **bandwidth is limited** by a single controller.

Distributions and NUMA

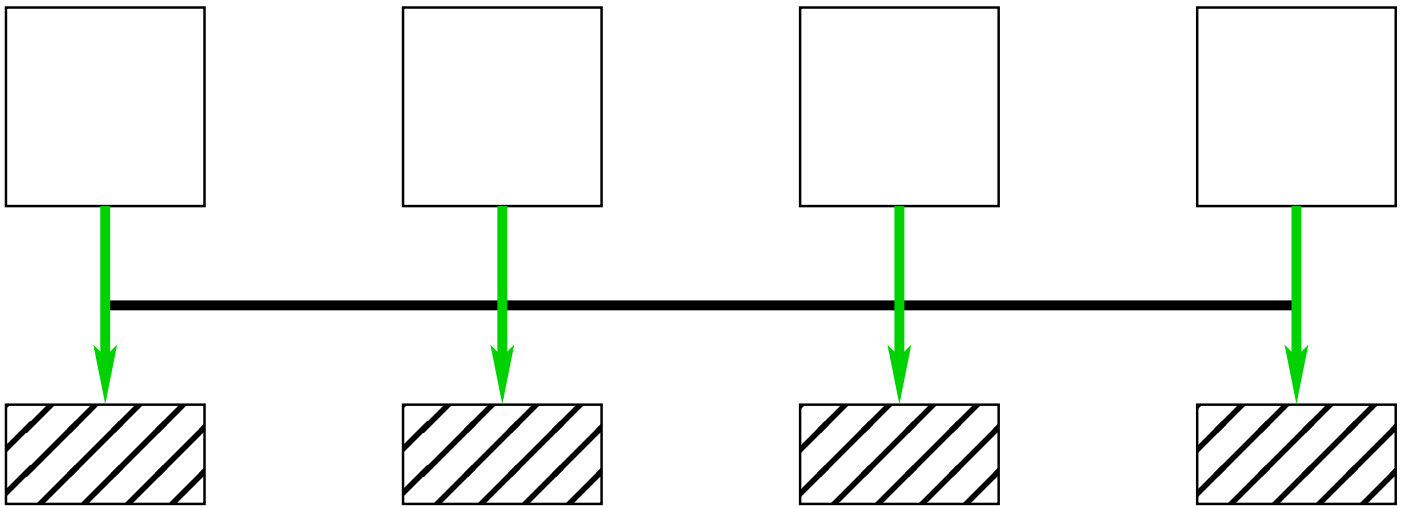
Implicit distribution, centralised interleaved allocation:



If each processor moves data from all memory elements, the bandwidth multiplies **if accesses are uniformly random**.

Distributions and NUMA

Explicit distribution, distributed local allocation:



If each processor moves data from and to its own unique memory element, the **bandwidth multiplies**.

Distributions and NUMA

Use interleaved allocation for everything:

- no bounds on non-local data movement!

Explicitly distribute everything:

- only local data. Explicit inter-process data movement.

Mixed approach y and A explicit, x interleaved:

- no bounds on non-local data movement for x.

Choices

A state-of-the-art SpMV strategy has to choose a

- cache-efficient strategy (aware, oblivious, ...),
- competitive data structure (compression),
- NUMA-efficient distribution strategy.

Fine-grained examples:

- OpenMP CRS: regular CRS, interleaved;

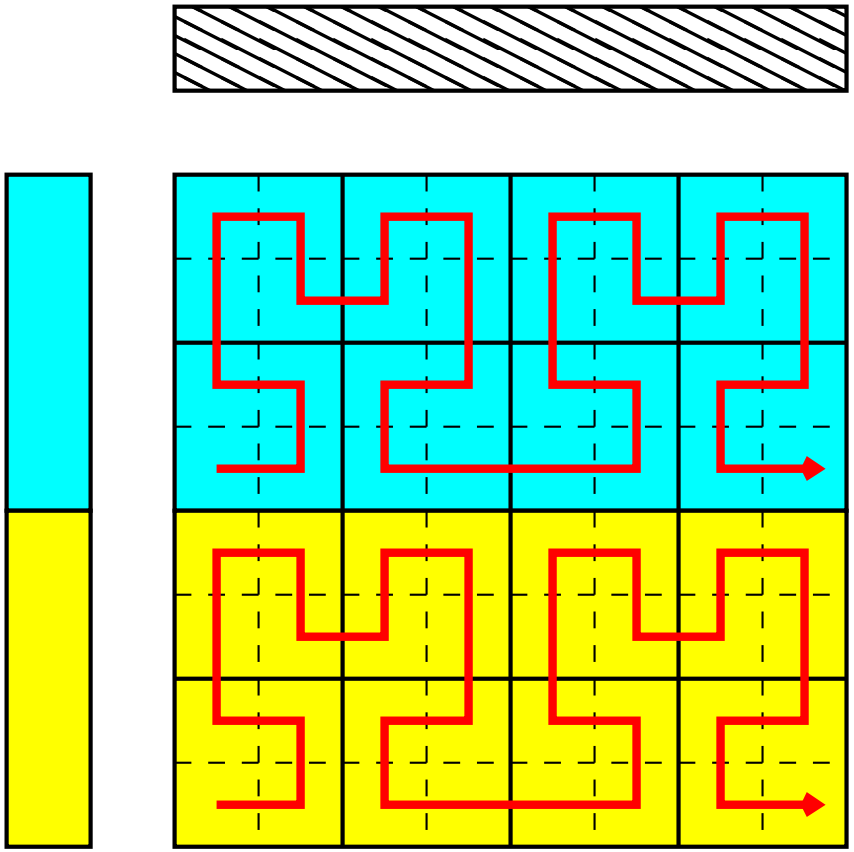
```
#omp parallel for private( i, k ) schedule( dynamic, 8 )
  for i = 0 to m − 1 do
    for k = Î_i to Î_{i+1} − 1 do
      add V_k · x_{J_k} to y_i
```

- CSB: Z-curves, compressed (blocked) COO, interleaved.

When fine-grained, is interleaving everything mandatory?

Coarse-grained SpMVs

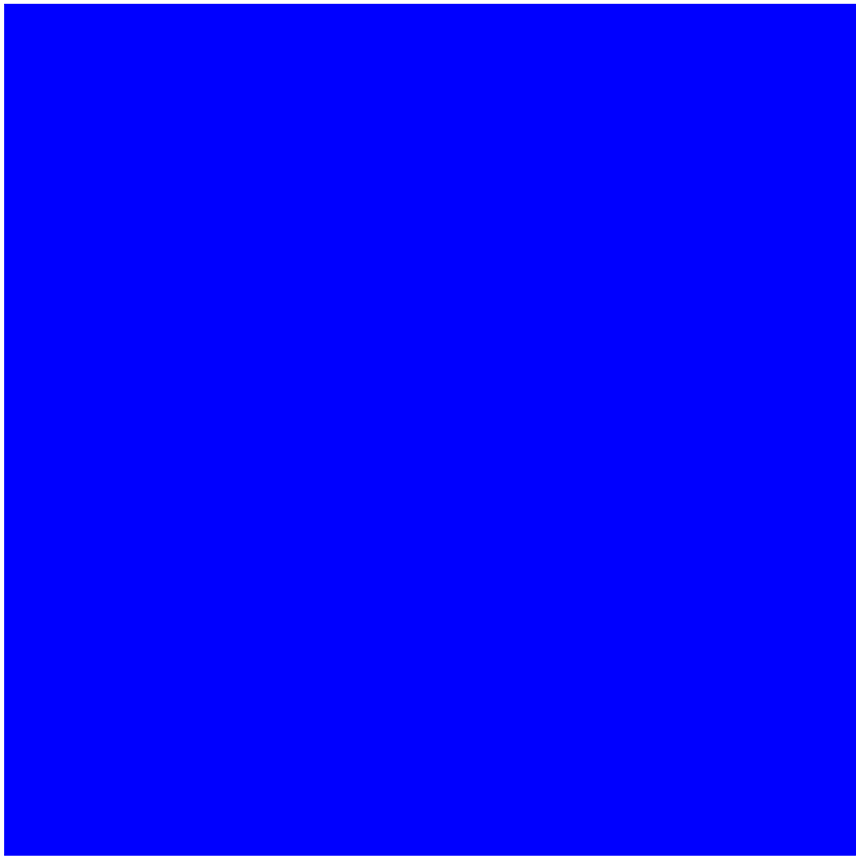
1D: Hilbert curve, compressed (blocked) BICRS, explicit allocation of y and A.



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).

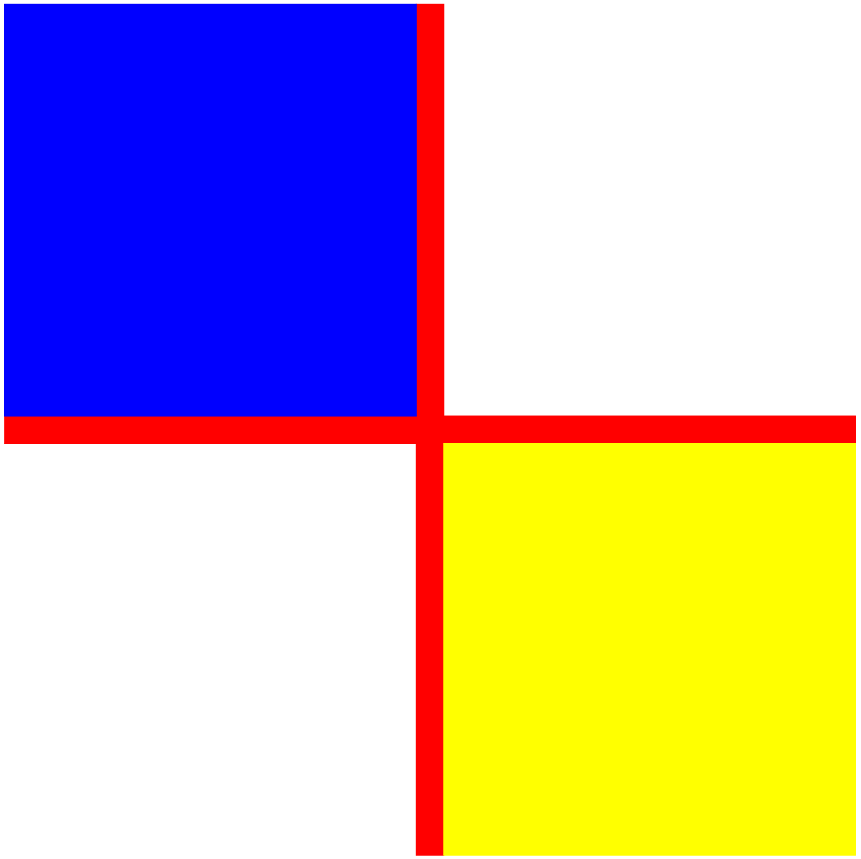
Coarse-grained SpMVs

2D: using **partitioning and reordering**:



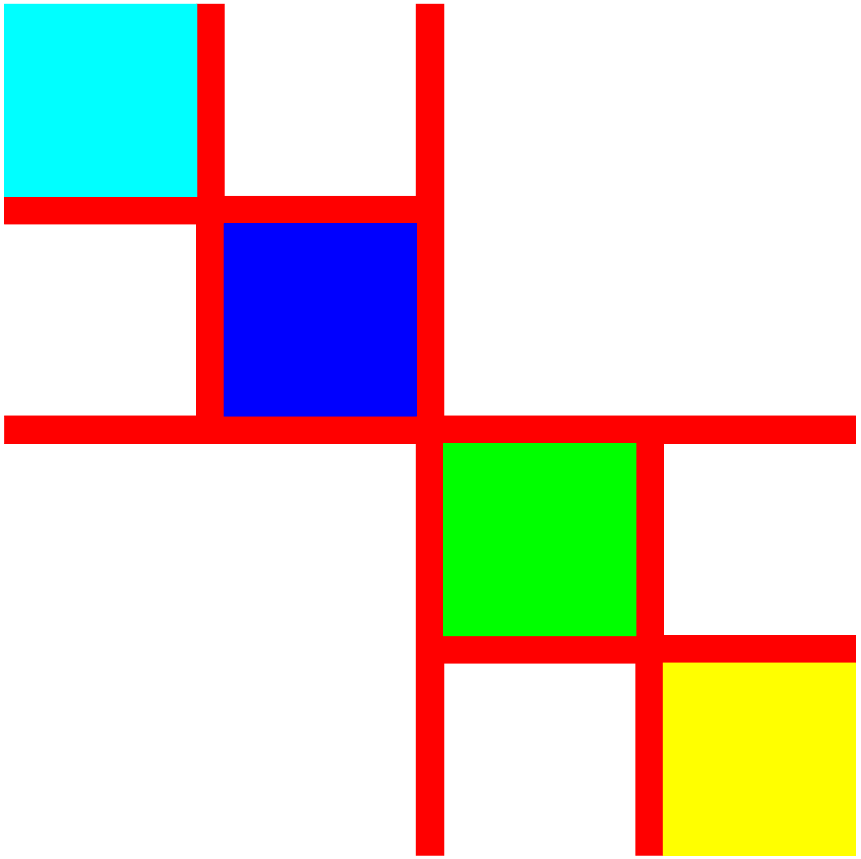
Coarse-grained SpMVs

2D: using **partitioning and reordering**:



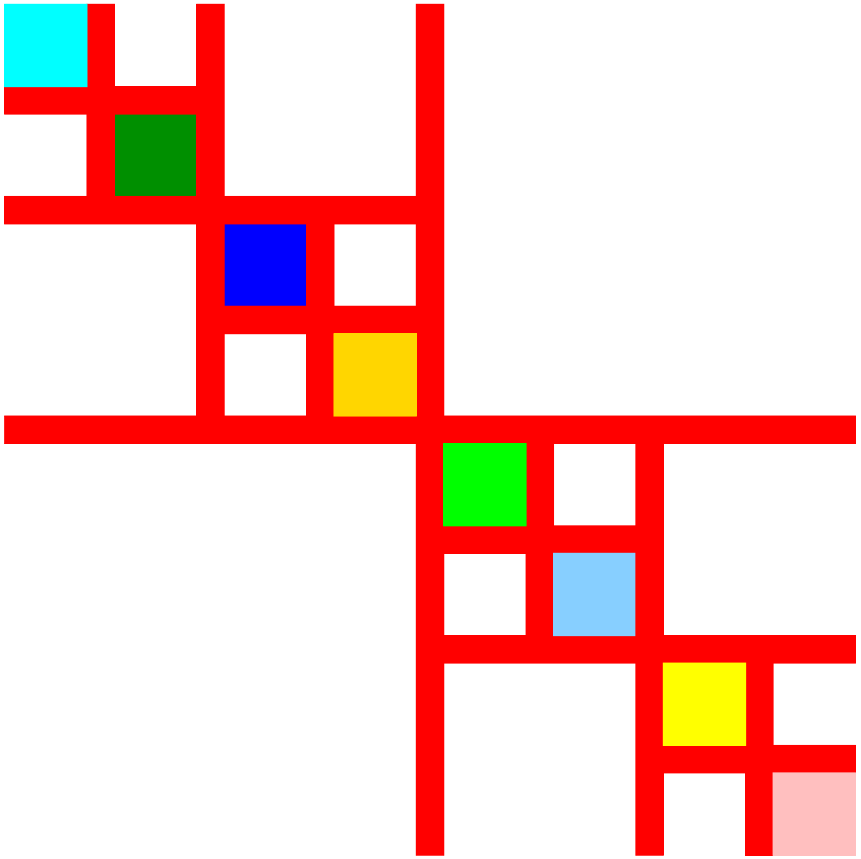
Coarse-grained SpMVs

2D: using **partitioning and reordering**:



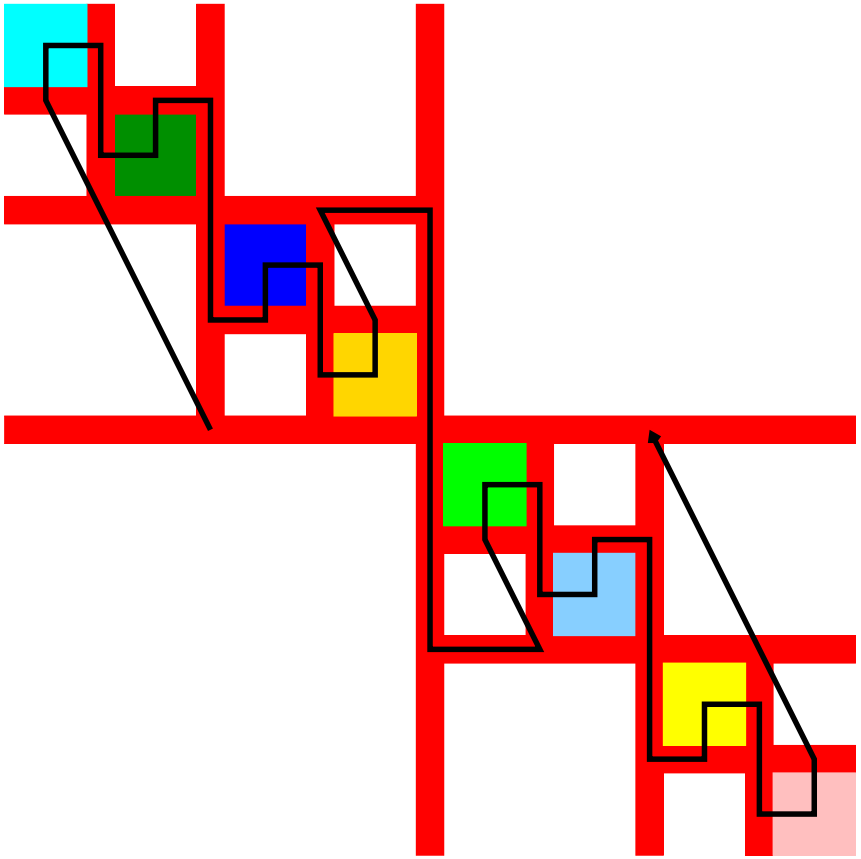
Coarse-grained SpMVs

2D: using **partitioning and reordering**:



Coarse-grained SpMVs

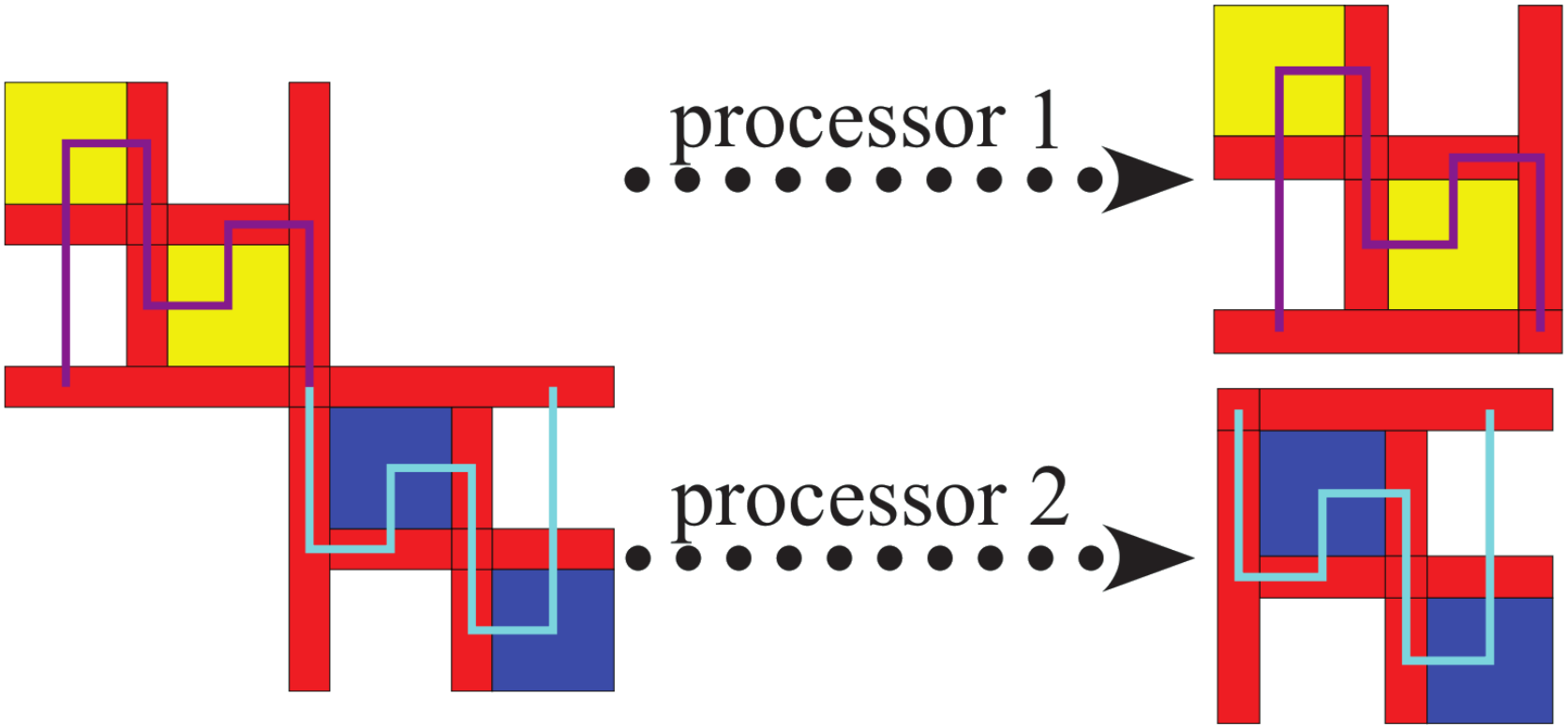
2D: using **partitioning and reordering**:



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

Coarse-grained SpMVs

2D: using **partitioning and reordering**:



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).

BSP 2D IMPLEMENTATION

for each local $a_{ij} \neq 0$ **do**
  if x_j is not local **then**
    bsp_get x_j from remote process
bsp_sync

multiply $y = Ax$ (using local nonzeroes only)

for each local $a_{ij} \neq 0$ **do**
  if y_i is non-local **then**
    bsp_send (i, y_i) to remote process
bsp_sync

while $bsp\_qsize() > 0$ **do**
  bsp_move (i, α) from queue
  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: CPUs

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

Average speedups relative to sequential CRS:

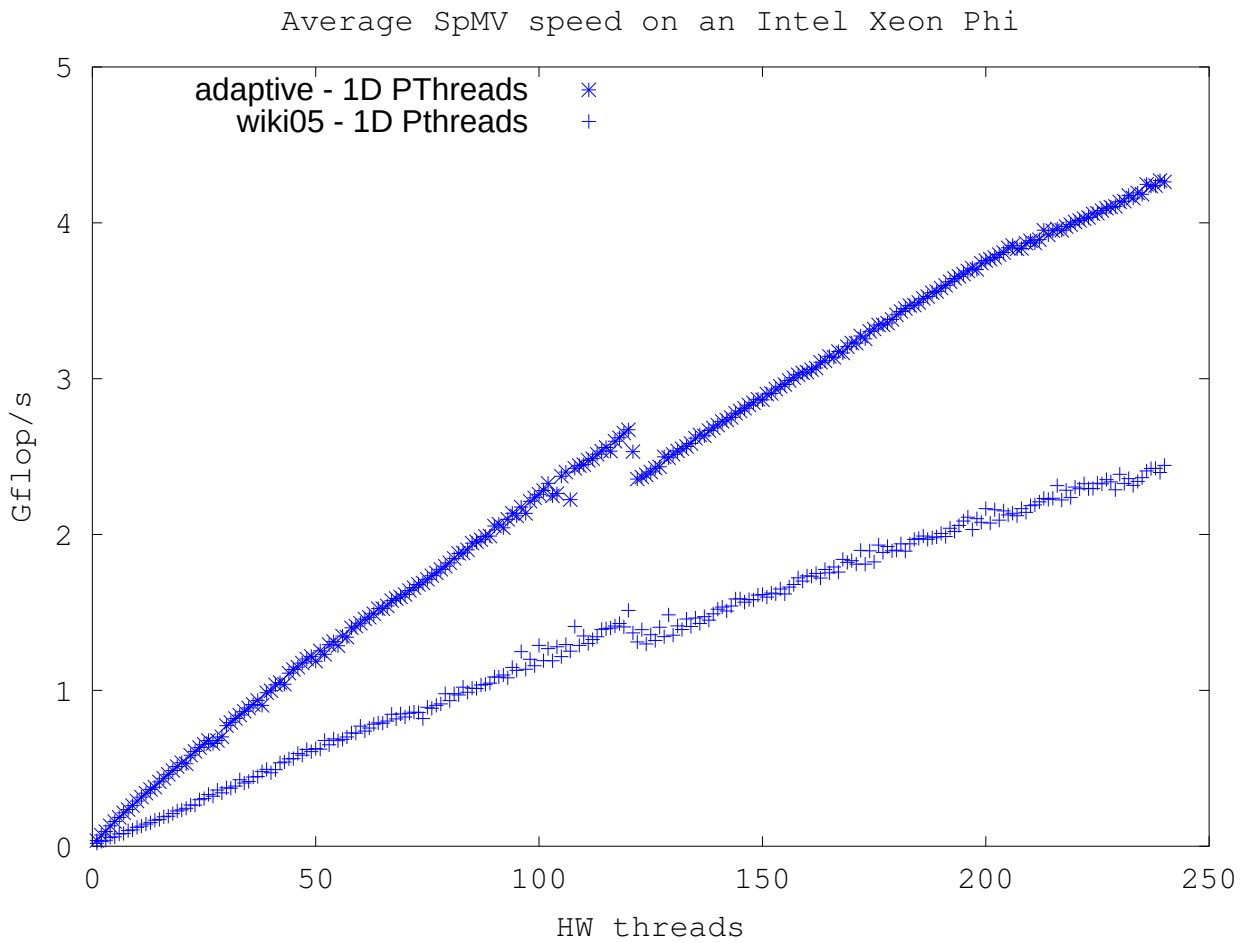
	4 x 10	8 x 8
OpenMP CRS	8.8	7.2
PThread 1D	13.6	20.0
Cilk CSB	22.9	26.9
BSP 2D	21.3	30.8

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, 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).

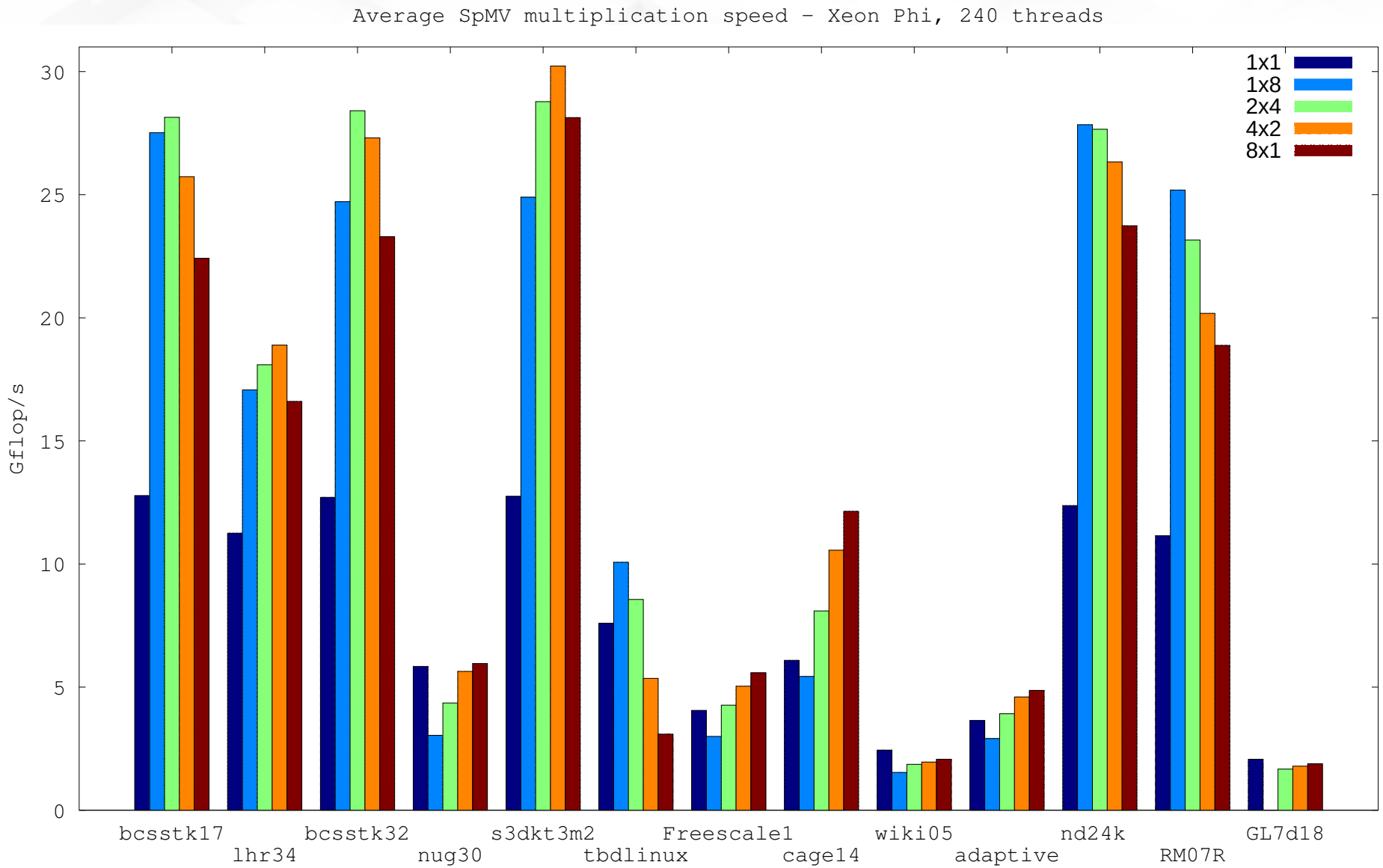
Experiments: Xeon Phi

60 cores, 4 HW threads per core, scales 'too slowly':



Latency bound; add more threads, or employ **vectorisation**

Experiments: Xeon Phi



Conclusion

We have seen techniques for

- cache-oblivious SpMV multiplication, and
- sparse matrix compression.

For new and upcoming architectures:

- highly NUMA architectures require 2D partitioning,
- vectorisation can help latency hiding (even within a bandwidth-bound computation).

`http://www.multicorebsp.com`

`people.cs.kuleuven.be/~albert-jan.yzelman/software.php`