An efficient sparse matrix storage scheme for shared memory parallel **Sparse BLAS** operations

Michele Martone
michele.martone@ipp.mpg.de

High Level Support Team
Max Planck Society Institute for Plasma Physics
Garching, Germany

PMAA'12
London, UK
June 28, 2012

Context: Sparse Matrix Computations

- numerical matrices which are *large* and populated mostly by zeros
- ubiquitous in scientific/engineering computations (e.g.:PDE, *information retrieval*, *document ranking*)
- the *performance* of sparse matrix codes computation on modern CPUs can be problematic (e.g.: fraction of peak (FLOP) performance)!
- our jargon: *performance=time efficiency*

Core of Sparse BLAS [1] operations: *Level 2*

The numerical solution of **linear systems** of the form $Ax = b$ (with A a sparse matrix, x, y dense vectors) using **iterative methods** requires repeated (and thus, **fast**) application of *Sparse Matrix-Vector Multiplication* (*SpMV*) operation

- **SpMV:** "$y \leftarrow \beta\ y\ +\ \alpha\ A\ x$"
- **SpMV-T:** "$y \leftarrow \beta\ y\ +\ \alpha\ A^T\ x$"

as well as the Sparse Triangular Solve operation, and variations.

[1] Sparse Basic Linear Algebra Subprograms, e.g.: as in TOMS Algorithm 818 (Duff and Vömel, 2002).

Context: shared memory parallel, cache based

high performance programming cache based, shared memory parallel computers requires:

- *locality of memory references*—for the memory hierarchy has:
  - limited memory bandwidth
  - memory access latency
- programming multiple cores for coarse-grained *workload partitioning*
  - high synchronization and cache-coherence costs

A recursive matrix storage: *Recursive Sparse Blocks* (RSB)

we propose:

- a *quad-tree* of sparse *leaf* submatrices
- outcome of recursive *partitioning* in *quadrants*
- submatrices are stored as *row oriented Compressed Sparse Rows* (CSR) or *Coordinate* (COO)
- an *unified* format for Sparse **BLAS** operations
- partitioning with regards to both the underlying cache size **and** available threads

Example of matrix partitioned for RSB

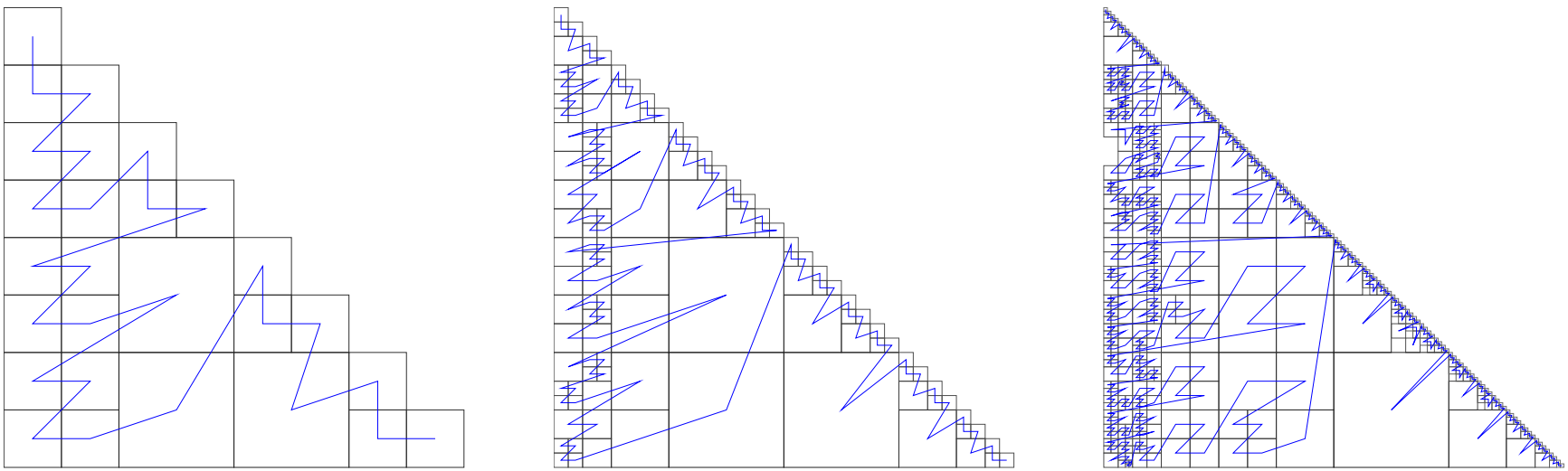
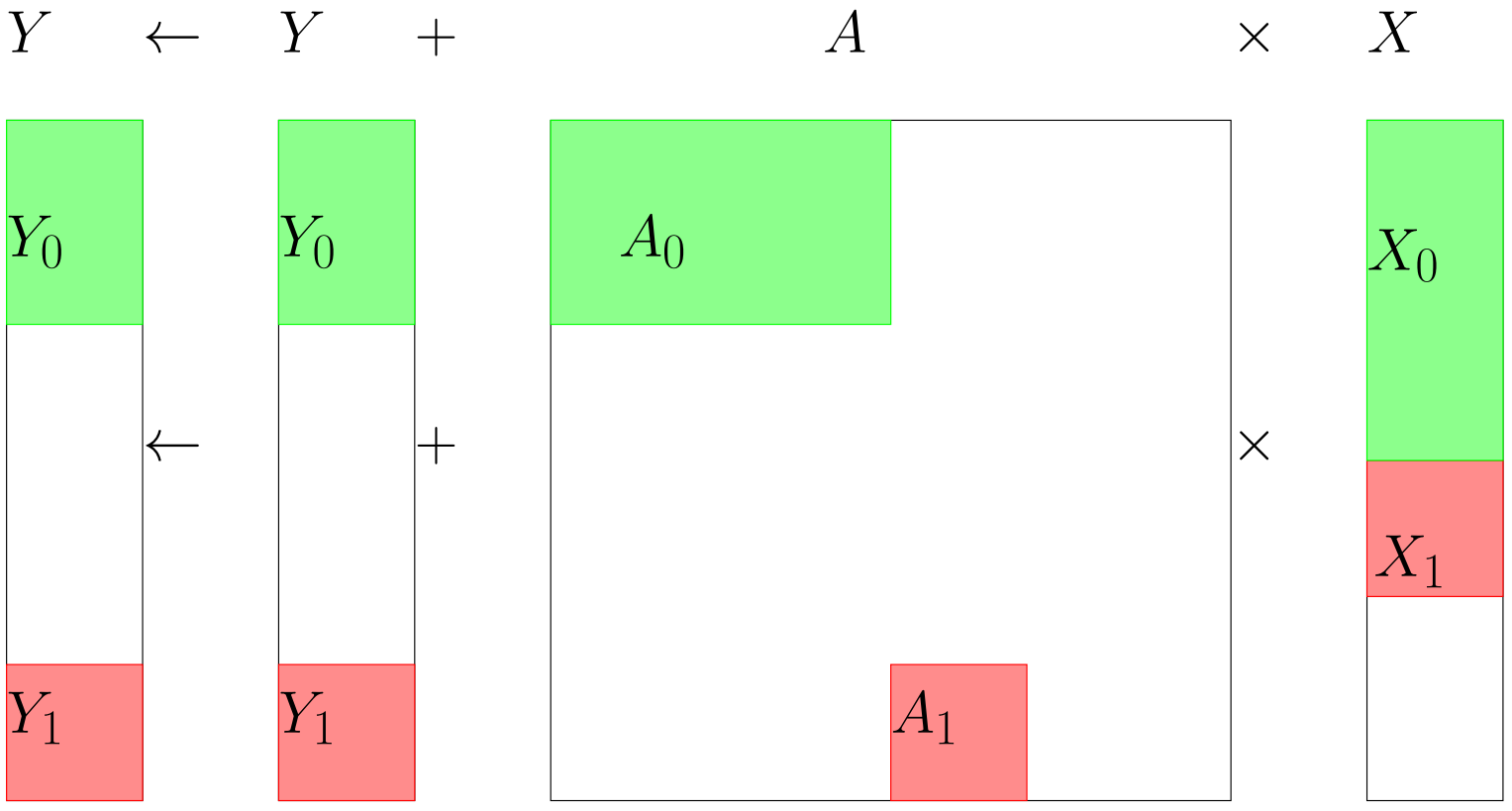


Figure: Matrix *audikw_1* (symmetric, 943695 rows, $3.9 \cdot 10^7$ nonzeroes) for 1, 4 and 16 threads on a Sandy Bridge.

Matrix layout described in (Martone et al., 2010).

Multi-threaded *SpMV*



$y \leftarrow y + \sum_i A_i \times x_i$, with leaves A_i; $A = \sum_i A_i$

Multi-threaded *SpMV*

$$y \leftarrow y + \sum_i A_i \times x_i$$

Threads $t \in \{1..T\}$ execute concurrently[2]:

$$y_{i_t} \leftarrow y_{i_t} + A_{i_t} \times x_{i_t}$$

we prevent *race conditions* by performing locking (and *busy wait*) if needed; we use[3]:

- per-submatrix visit information
- per-thread current submatrix information

[2] Using OpenMP

[3] See extra: slide 19

Pros/Cons of RSB

- + scalable parallel *SpMV*/*SpMV-T*
- + parallel *SpSV*/*SpSV-T* (of course, less scalable than *SpMV*!)
- + many other common operations (e.g.: parallel matrix build algorithm)
- + native support for symmetric matrices *SpMV*
- - a number of known cases where parallelism is poor (unbalanced matrices)
- - some algorithms easy to express/implement for CSR may be problematic for RSB

SpMV: RSB's vs MKL's CSR on Intel Sandy Bridge

Experimental time efficiency comparison of our RSB prototype to the proprietary, highly optimized Intel's *Math Kernels Library* (MKL r.10.3-7) sparse matrix routines (`mkl_dcsrmv` – `double` precision case).

We report here results on a double *"Intel Xeon E5-2680 0 @ 2.70GHz"* (2×8 cores) and publicly available large ($> 10^7$ nonzeroes) matrices[4].

We compiled our code with the *"Intel C 64 Compiler XE, Version 12.1.1.256 Build 20111011"* using `CFLAGS="-O3 -xAVX -fPIC -openmp"` flags.

[4] See extra slide slide 20 for a list.

Comparison to MKL, Unsymmetric *SpMV*

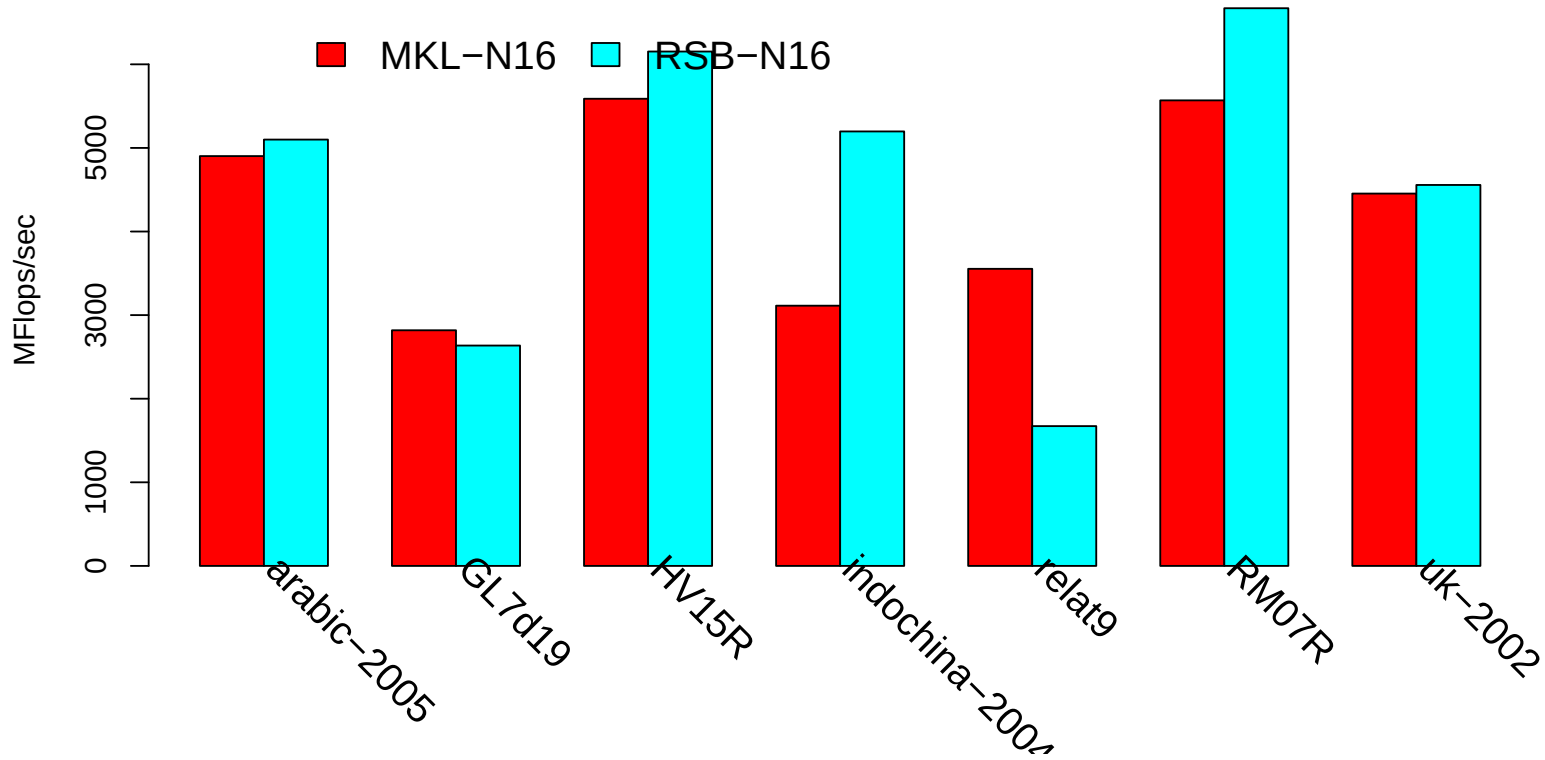


Figure: **N**on transposed *SpMV* performance on Sandy Bridge versus MKL's CSR, 16 threads, unsymmetric matrices.

Comparison to MKL, Unsymmetric *SpMV*

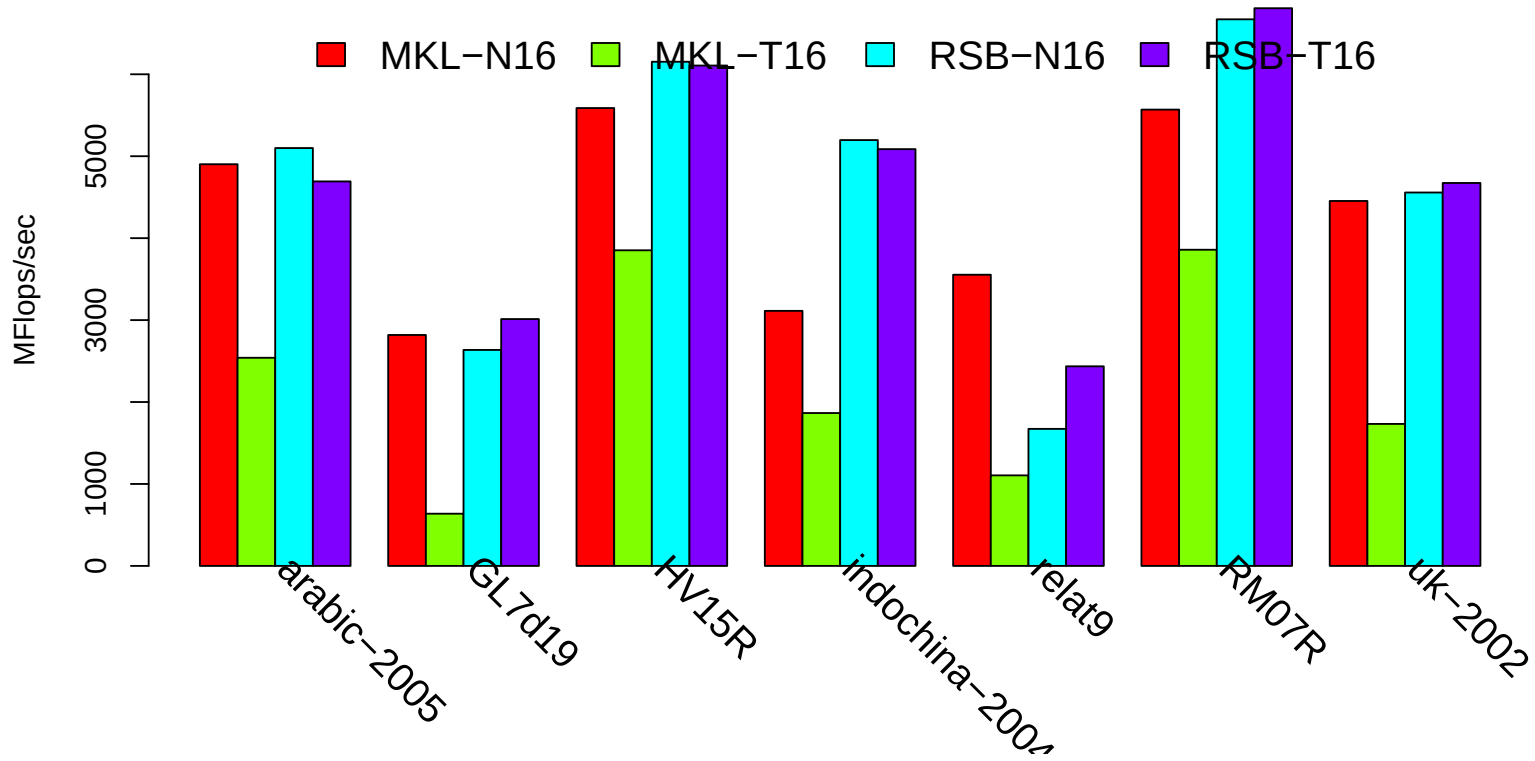


Figure: **T**ransposed/**N**on transposed *SpMV* performance on Sandy Bridge versus MKL's CSR, 16 threads, unsymmetric matrices.

Comparison to MKL, Unsymmetric *SpMV*

- untransposed *SpMV* even 60 % faster than MKL's CSR
- *SpMV-T* even 4 times faster (on *GL7d19*: here MKL does not scale)
- some matrices (e.g.: *tall relat9*) are problematic
- transposed and untransposed *SpMV* have almost same performance

Comparison to MKL, Symmetric *SpMV*

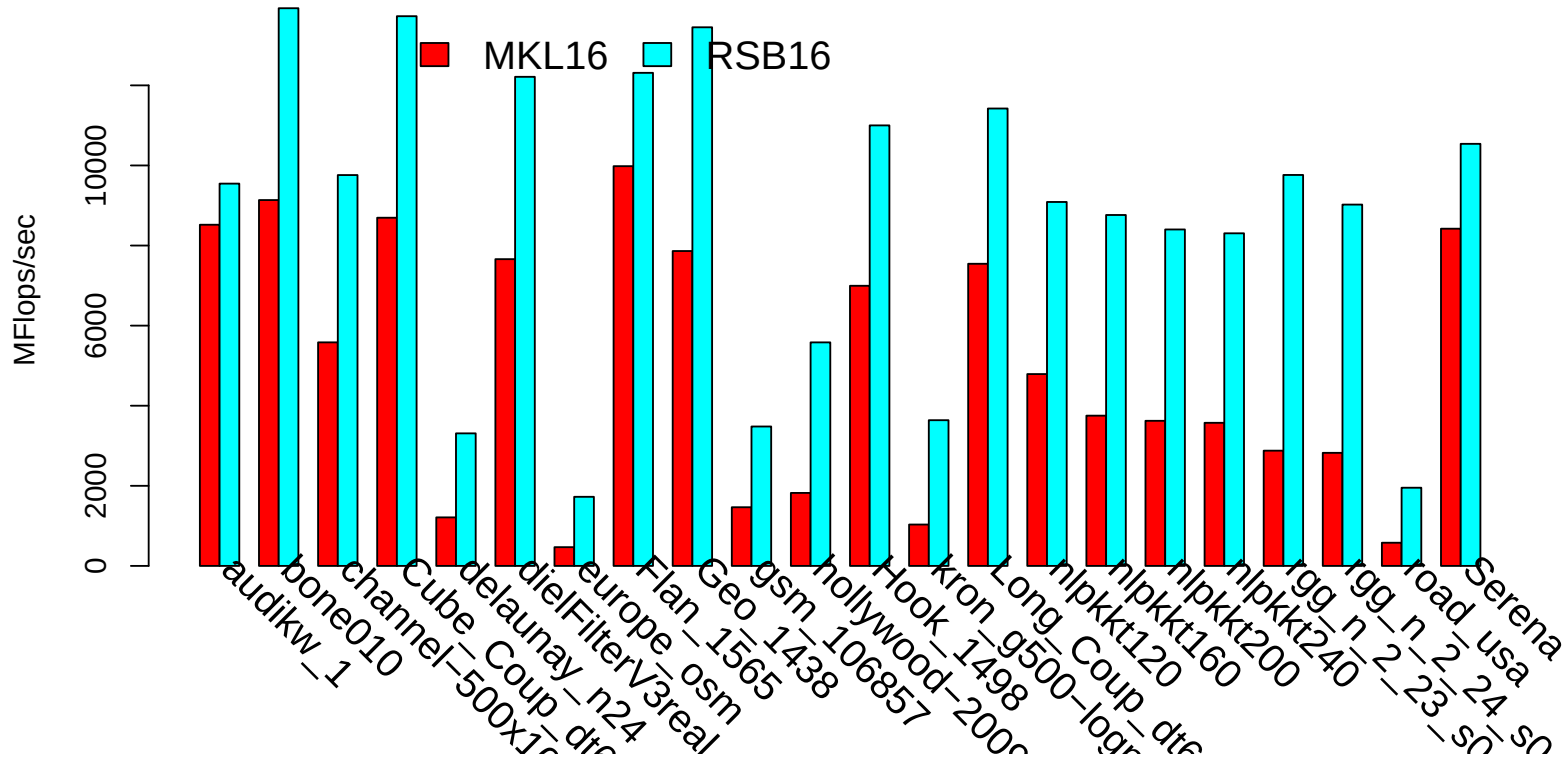


Figure: *SpMV* performance on Sandy Bridge versus MKL's CSR, 16 threads, symmetric matrices.

Comparison to MKL, Symmetric *SpMV*

- speedups up to around 200% in several cases
- RSB has large advantage over MKL's CSR (symmetric locking is easier)

Relative Cost of (row sorted) COO to RSB conversion

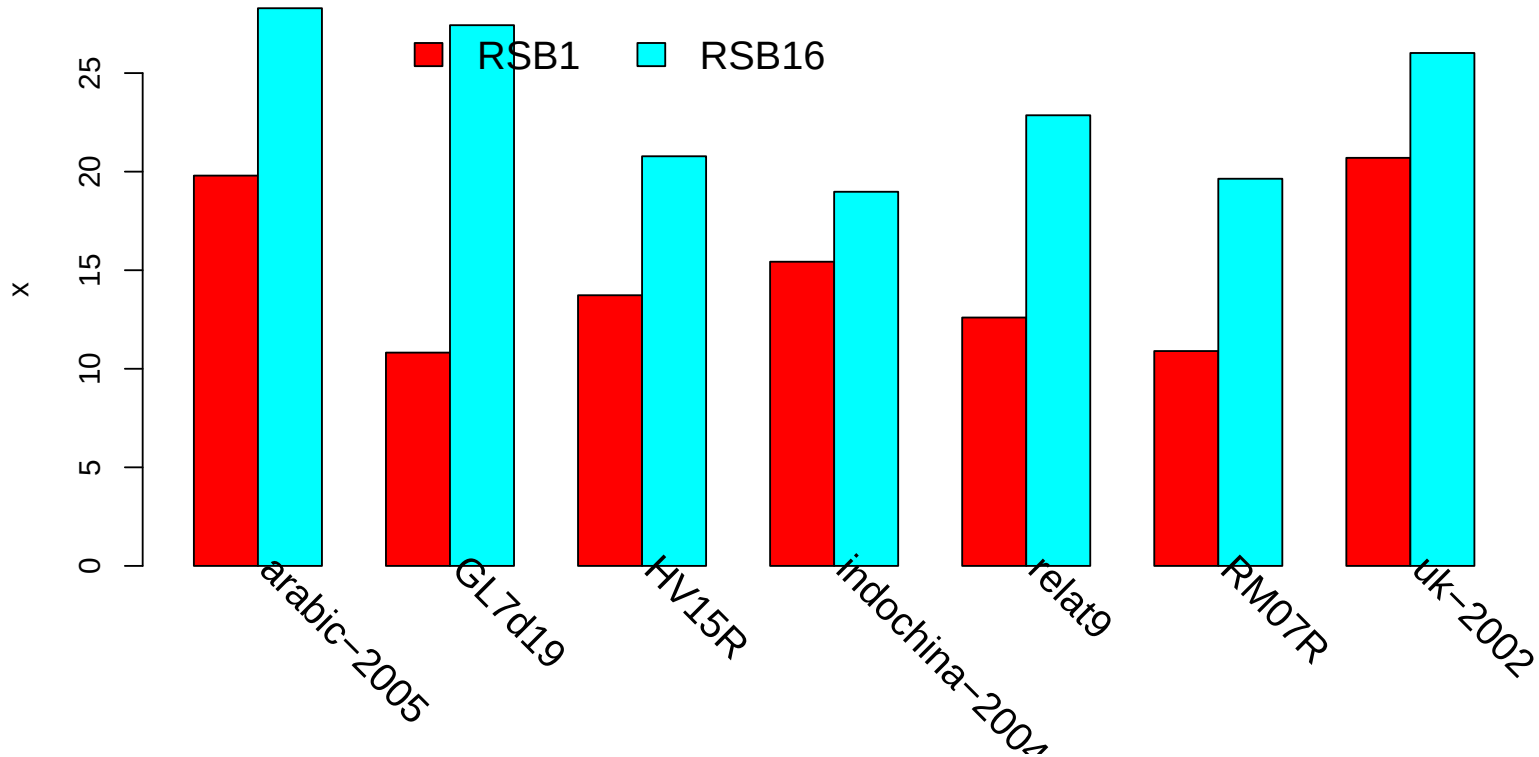


Figure: Non transposed conversion-to-*SpMV* times ratio on Sandy Bridge, unsymmetric matrices, 1 and 16 threads.

Relative Cost of (row sorted) COO to RSB conversion

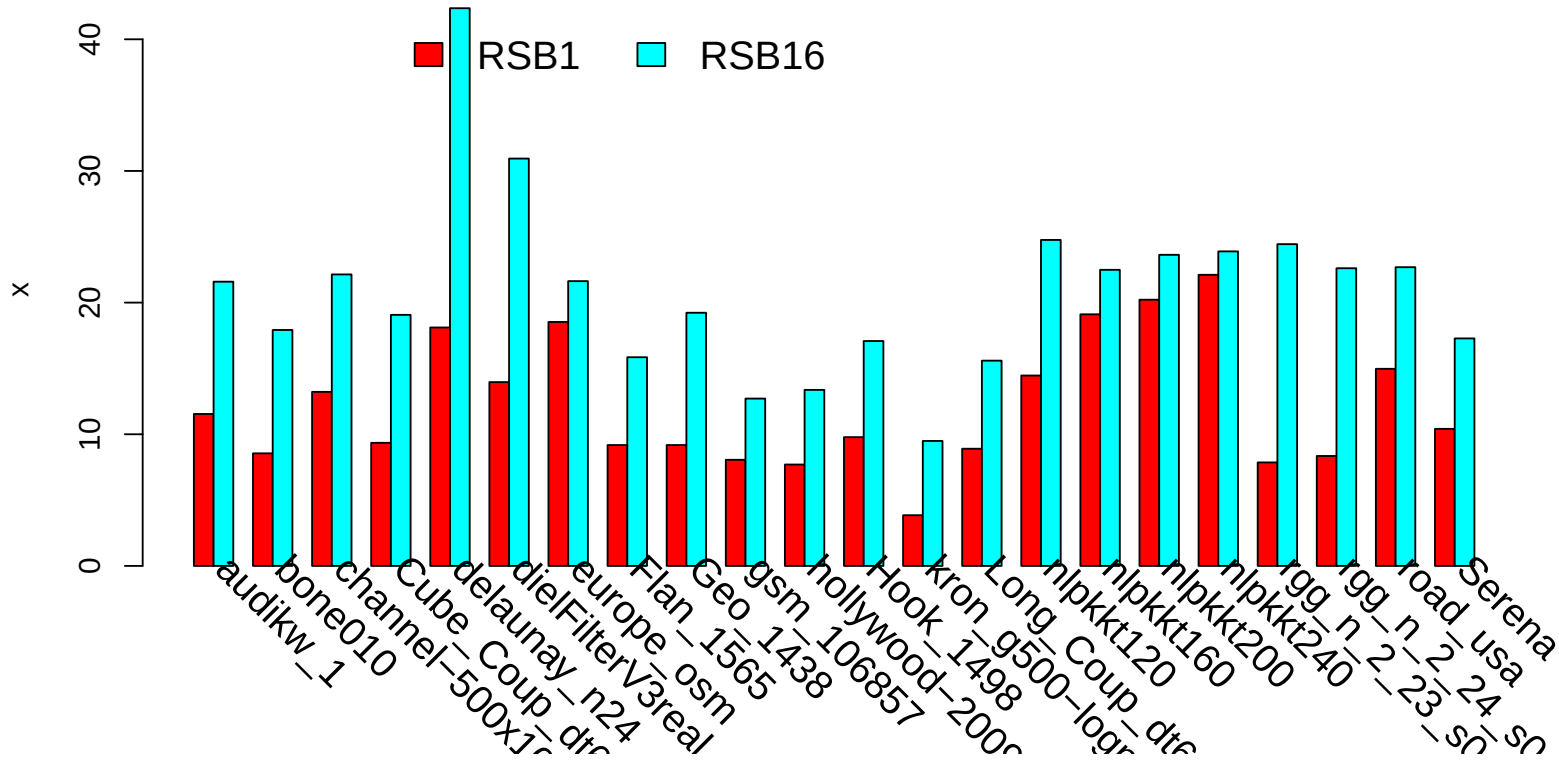


Figure: Non transposed conversion-to-*SpMV* times ratio on Sandy Bridge, symmetric matrices, 1 and 16 threads.

Comments about row sorted COO to RSB conversion

- conversion to RSB can be costly (e.g.: 10 to 20 *SpMV*'s)
- scales less than *SpMV*
- many *SpMV* iterations amortize this cost
- reasonable memory requirements, COO/CSR interoperability/array reuse

Conclusions

- a *cache friendly* format for BLAS operations
- *SpMV* and *SpMV-T* both scalable
- on large matrices, competitive with Intel's highly optimized, proprietary CSR implementation; especially with *SpMV-T* and symmetric matrices
- core functionality for a distributed memory parallel Sparse BLAS library

Further

Source code for RSB is already freely available online in form of:

- a C library (with Fortran bindings): `librsb`
- a module for `GNU Octave`: `sparsersb`
- documentation:
  `http://www.ipp.mpg.de/~mima/librsb/html/`

Questions welcome!

Thanks for your attention.

Please consider testing `librsb`:
`http://www.ipp.mpg.de/~mima/`

References

- Michele Martone, Salvatore Filippone, Salvatore Tucci, Marcin Paprzycki, and Maria Ganzha. *Utilizing recursive storage in sparse matrix-vector multiplication - preliminary considerations.* In Thomas Philips, editor, CATA, pages 300-305. ISCA, 2010.
- Iain S. Duff and Christof Vömel. *Algorithm 818: A reference model implementation of the Sparse BLAS in Fortran 95* In ACM Trans. on Math. Softw., n. 2, vol. 28, pages 268–283, ACM, 2002

Multi-threaded *SpMV*

$$y \leftarrow y + \sum_i A_i \times x_i$$

Concurrently on threads $t \in \{1..T\}$:

$$y_{i_t} \leftarrow y_{i_t} + A_{i_t} \times x_{i_t}$$

For threads (t, u), no (y_{i_t}, y_{i_u}) shall intersect at a given time — using lock+usage bitmap to prevent *race conditions*

Matrices

matrix	symm	r	c	nnz	nnz/r
arabic-2005.mtx	G	22744080	22744080	639999458	28.14
audikw_1.mtx	S	943695	943695	39297771	41.64
bone010.mtx	S	986703	986703	36326514	36.82
channel-500x100x100-b050.mtx	S	4802000	4802000	42681372	8.89
Cube_Coup_dt6.mtx	S	2164760	2164760	64685452	29.88
delaunay_n24.mtx	S	16777216	16777216	50331601	3.00
dielFilterV3real.mtx	S	1102824	1102824	45204422	40.99
europe_osm.mtx	S	50912018	50912018	54054660	1.06
Flan_1565.mtx	S	1564794	1564794	59485419	38.01
Geo_1438.mtx	S	1437960	1437960	32297325	22.46
GL7d19.mtx	G	1911130	1955309	37322725	19.53
gsm_106857.mtx	S	589446	589446	11174185	18.96
hollywood-2009.mtx	S	1139905	1139905	57515616	50.46
Hook_1498.mtx	S	1498023	1498023	31207734	20.83
HV15R.mtx	G	2017169	2017169	283073458	140.33
indochina-2004.mtx	G	7414866	7414866	194109311	26.18
kron_g500-logn20.mtx	S	1048576	1048576	44620272	42.55
Long_Coup_dt6.mtx	S	1470152	1470152	44279572	30.12
nlpkkt120.mtx	S	3542400	3542400	50194096	14.17
nlpkkt160.mtx	S	8345600	8345600	118931856	14.25
nlpkkt200.mtx	S	16240000	16240000	232232816	14.30
nlpkkt240.mtx	S	27993600	27993600	401232976	14.33
relat9.mtx	G	12360060	549336	38955420	3.15
rgg_n_2_23_s0.mtx	S	8388608	8388608	63501393	7.57
rgg_n_2_24_s0.mtx	S	16777216	16777216	132557200	7.90
RM07R.mtx	G	381689	381689	37464962	98.16
road_usa.mtx	S	23947347	23947347	28854312	1.20
Serena.mtx	S	1391349	1391349	32961525	23.69
uk-2002.mtx	G	18520486	18520486	298113762	16.10

Table: Matrices used for our experiments.