sparsersb: Accessing **librsb**'s Shared Memory Parallel Sparse BLAS Implementation from **GNU/Octave**

Michele Martone

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

OctConf'13
MOX @ Politecnico di Milano
Milano, Italy, June 24–26, 2013

Presentation outline

This presentation is about `sparsersb`

`sparsersb` is a command provided to GNU/Octave via an OCT-file, giving easy access to the `librsb` library

- defines the "`rsb sparse matrix`" type in the GNU/Octave language
- bits about the internals:
  - ≈2 KLoC of C++
  - `class octave_sparsersb_mtx:`
    `public octave_sparse_matrix`
  - `DEFUNOP (transpose, sparse_rsb_mtx)`
  - `DEFBINOP(op_mul, sparse_rsb_mtx, matrix)`
  - `DEFBINOP(op_trans_mul, sparse_rsb_mtx, matrix)`
  - `DEFBINOP(...)` # ... x 20
  - `DEFUNOP(...)` # ... x 6
  - `DEFASSIGNOP(...)` # ... x 4
  - ...
- GPLv3 licensed
- on Octave-Forge
- not aiming to replace `sparse` !

`sparsersb` is based on `librsb`

`librsb` is a library implementing multithreaded algorithms for the "RSB" sparse matrix format

- written with *iterative methods* in mind
- $\approx$ 80 KLoC of hand written C 99, GNU Octave, M4
- numerical kernels generated by M4
- correctness tests largely generated by Octave
- performant as/more than Intel MKL's CSR implementation
- LGPLv3 licensed

Motivations behind `sparsersb`

- experimenting/testing iterative methods
- testing `librsb` ideas
- contributing to GNU/Octave!

Motivations behind `librsb`

- my PhD research
- high performance for single node, shared memory (OpenMP)
- for iterative methods
- standalone or component of distributed memory (e.g.: MPI) applications
- optimized for large matrices ($\gg$ / $>$ outermost cache size)

Sparse Matrix Computations

- numerical matrices which are *large* and populated mostly by zeros
- ubiquitous in scientific/engineering computations (e.g.:PDE)
- the *performance* of sparse matrix codes computation on modern CPUs can be problematic (a fraction of peak)!
- there is no *"silver bullet"* for performance
- jargon: *performance=time efficiency*[1]

[1] Which so far turns out to be energy efficiency as well.

Which sparse matrix computations matter the most to us ?

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**) computation of (variants of) *Sparse Matrix-Vector Multiplication* and *Sparse Matrix-Vector Triangular Solve*:

- *SpMV* : "$y \leftarrow \beta\ y\ +\ \alpha\ A\ x$"
- *SpMV-T* : "$y \leftarrow \beta\ y\ +\ \alpha\ A^T\ x$"
- *SymSpMV*: "$y \leftarrow \beta y + \alpha\ A^T\ x,\ A = A^T$"
- *SpSV* : "$x \leftarrow \alpha\ L^{-1}\ x$"
- *SpSV-T* : "$x \leftarrow \alpha\ L^{-T}\ x$"

Shared memory parallel, cache based computers

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

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

The *Recursive Sparse Blocks* (RSB) matrix layout

- a *quad-tree* of sparse *leaf* submatrices
- outcome of recursive *partitioning* in *quadrants*
- submatrices are stored by either *row oriented Compressed Sparse Rows* (CSR) or *Coordinates* (COO)
- an *unified* format supporting many common operations and format variations (e.g.: diagonal implicit, one or zero based, transposition, complex types, ...)
- partitioning with regards to both the underlying cache size **and** available threads

sparsersb: warmup

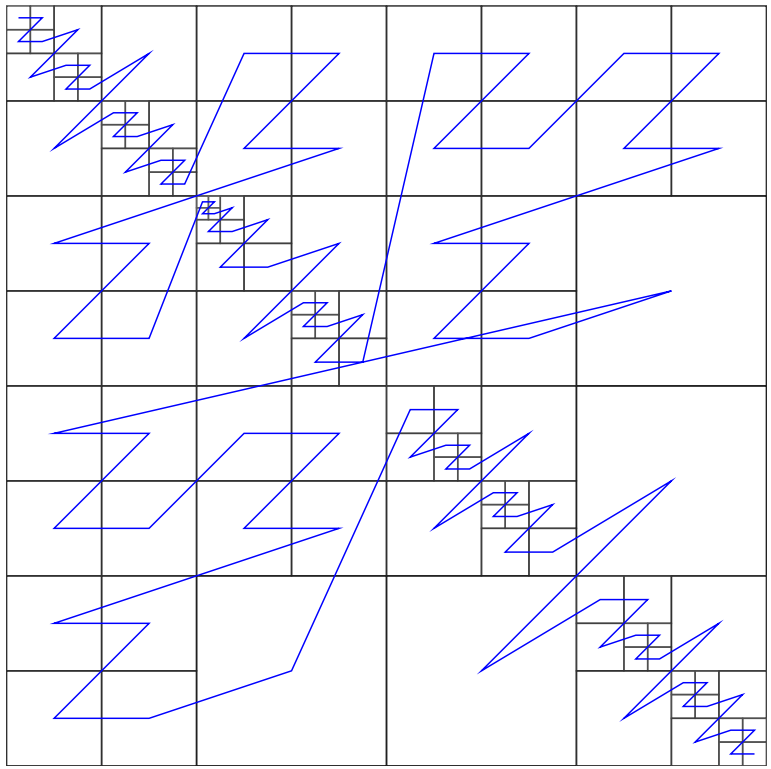
```
# Get matrix ASIC_320k from
# http://www.cise.ufl.edu/research/sparse/matrices/Sandia/ASIC_320k
#
octave:1> A=sparsersb("ASIC_320k.mtx.gz"); # load gzipped Matrix Market file
octave:2> typeinfo(A)
ans = rsb sparse matrix
octave:3> A
A =

Recursive Sparse Blocks  (rows = 321821, cols = 321821, nnz = 2635364 [0.0025%]

  (1, 1) -> 0.019
  (1, 12060) -> -1.5e-15
  (1, 12061) -> -5.8e-16
  (1, 23338) -> -0.0023
  (1, 23339) -> -1e-18
  (1, 23340) -> 0
  (1, 81086) -> -0.016
  (2, 2) -> 0.0074
  ...
```

Instance of *ASIC_320k* matrix ($3.2 \cdot 10^5$ rows, $3.2 \cdot 10^5$ columns, $2.6 \cdot 10^6$ nonzeroes):

```
octave:4> sparsersb(A,"renderb","ASIC_320k.eps");
```



Adaptivity to threads count

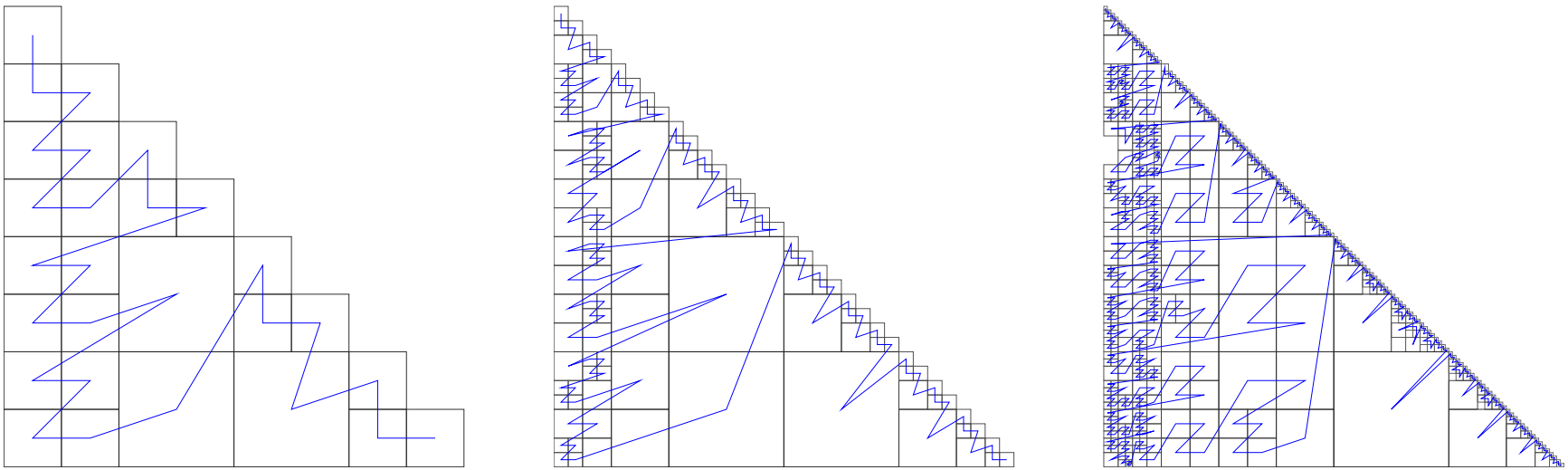


Figure: Matrix *audikw_1* (symmetric, 943695 rows, $3.9 \cdot 10^7$ nonzeroes) for 1, 4 and 16 threads on a Sandy Bridge.

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

Pros/Cons of our RSB implementation, w.r.t. performance

- + scalable parallel *SpMV* /*SpMV-T*
- + partially parallel *SpSV* /*SpSV-T*
- + many other common operations (e.g.: parallel matrix build algorithm)
- + native support for parallel symmetric multiply (*SymSpMV*)
- - a number of known cases (unbalanced matrices) where parallelism is poor
- - some algorithms easy to express/implement for CSR/CSC are more complex for RSB

sparsersb: matrix assembly

```
IA=[1,1,3]; # row    coordinates of non zero matrix coefficients
JA=[1,1,3]; # column coordinates of non zero matrix coefficients
VA=[1,10,33]; # non zero matrix coefficients

# Octave sparse matrix creation syntax:
OM=sparse   (IA,JA,VA,"summation"); # Octave's CSC
# Same for sparsersb:
RM=sparsersb(IA,JA,VA,"summation") # librsb's RSB

ans =

Recursive Sparse Blocks  (rows = 3, cols = 3, nnz = 2 [22%])

  (1, 1) -> 11
  (3, 3) -> 33
```

sparsersb: where to use it

sparsersb's performance lies in matrix-vector multiply:

```
alpha=1.0;
beta=-1.0;
X=[-1,-2,-3];
Y=[ 0, 0, 0];

# Octave matrix:
Y=alpha*OM*X+beta*Y;
# Same usage of the "*" operator:
Y=alpha*RM*X+beta*Y;
```

no tertiary operators for in place *SpMV*

Octave does at most provide binary operators for *SpMV* .

```
Y-...:
# The following:
Y+=RM.*X; # matrix-vector multiply
Y+=RM.'*X; # transposed matrix-vector multiply
# work like:
Y=Y+RM.*X; # matrix-vector multiply
Y=Y+RM.'*X; # transposed matrix-vector multiply
```

The result of the multiplication goes first to a temporary location, only then it is added to `Y`.

sparsersb matrix usage: some further operators

```
RM=sparsersb([1]) # conversion from dense
RM=sparsersb(OM)  # conversion from Octave sparse matrices
RM=transpose(RM)  # built-in: transpose, normest, trace, ...
RM * RM           # matrix-matrix multiplication
RM \ X            # triangular solve
RM * = alpha      # scaling
RM = -RM          # negation
```

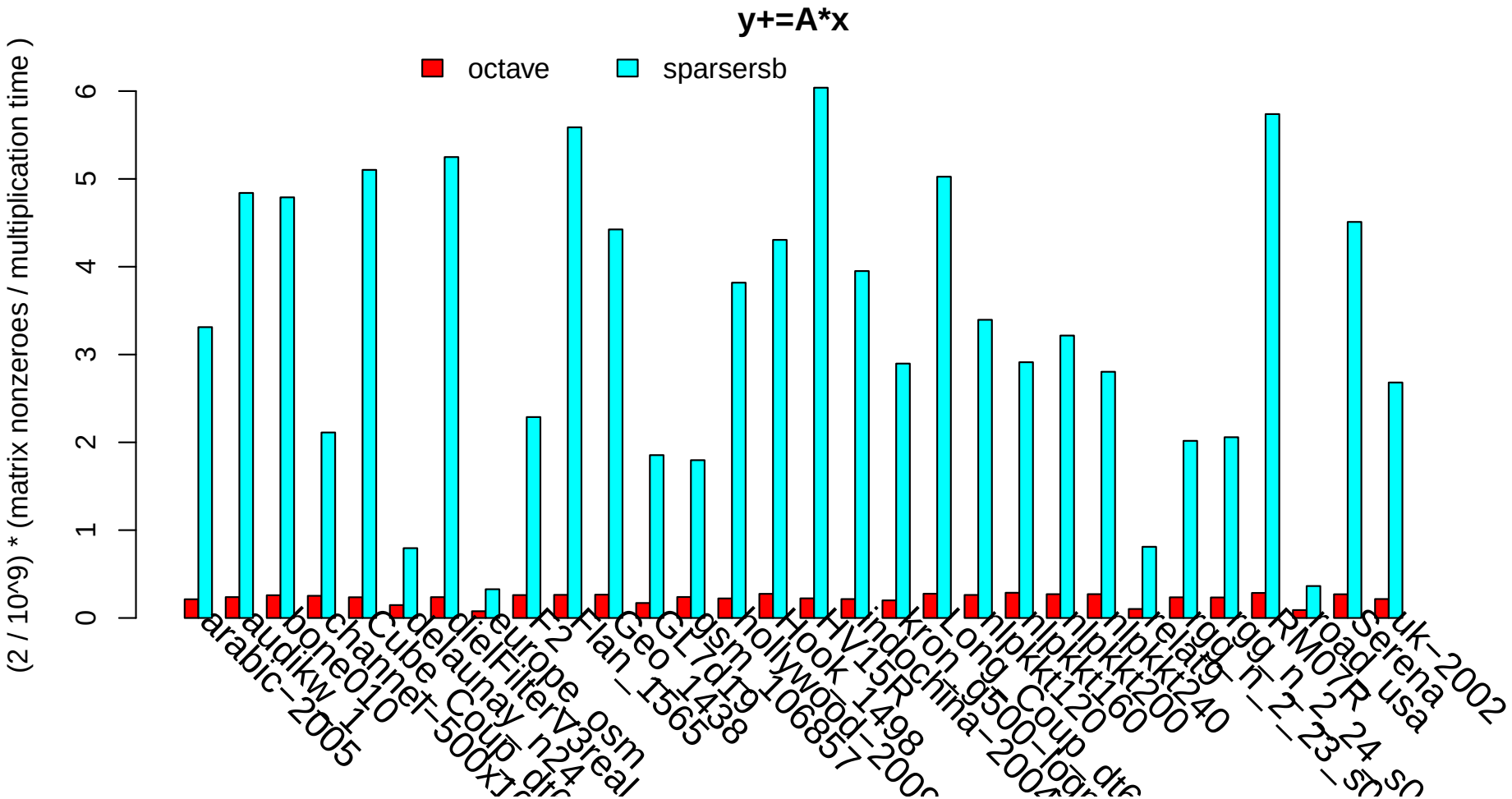
Performance Comparison to Octave

- running `octavebench.m` on a 2 x 8 core Intel Sandy Bridge "Romely-EP"
- Octave-3.6.4 with `CXXFLAGS=-O3 -pipe -march=native -mtune=native -mavx -pthread`
- `librsb/sparsersb` with `CFLAGS=-O3 -pipe -march=native -mtune=native -mavx -fPIC`
- compiled with GCC 4.7.2
- publicly available matrices from the Florida Sparse Matrix Collection

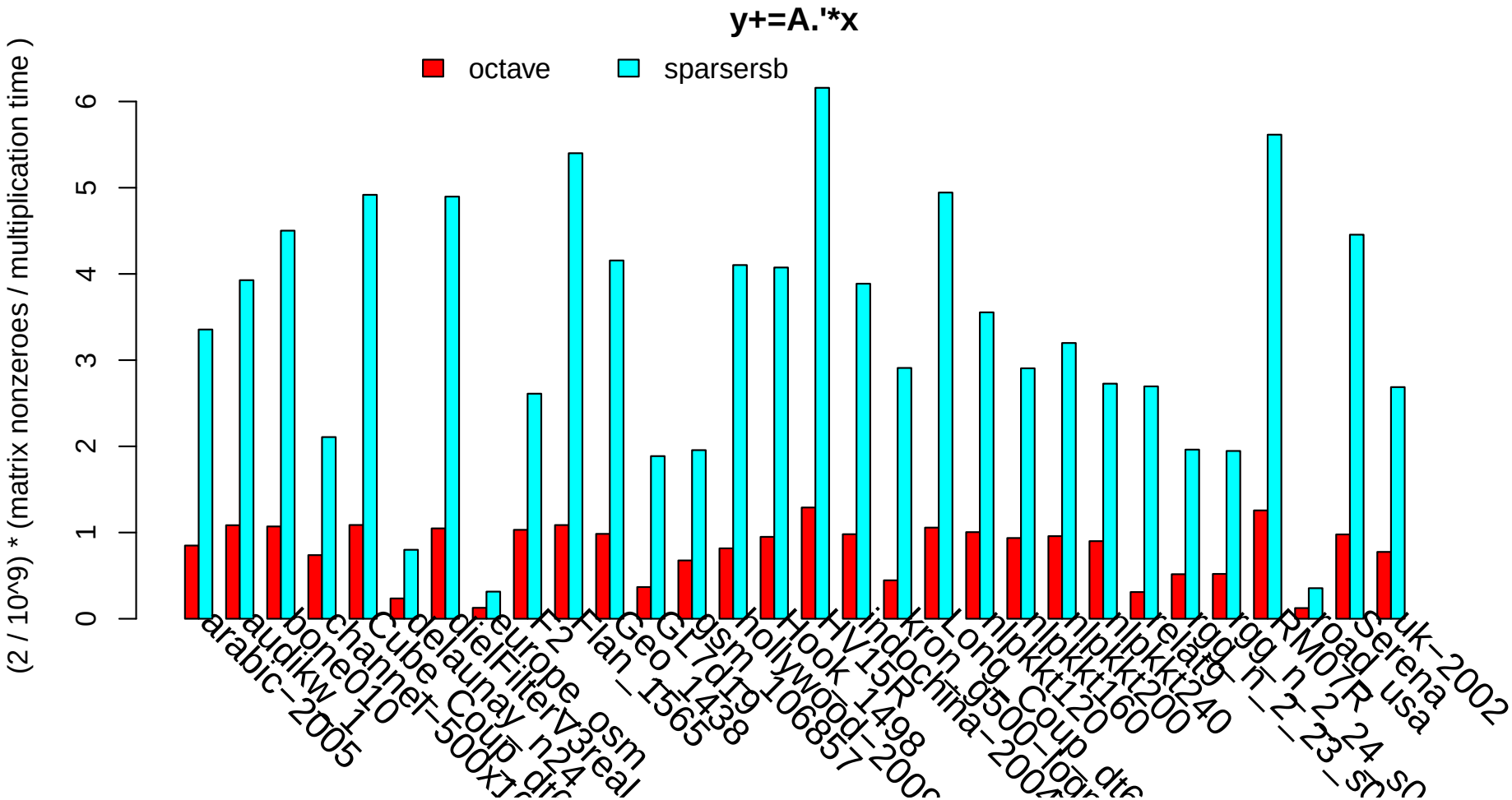
matrix	symm	nr	nc	nnz	nnz/nr
arabic-2005	G	22744080	22744080	639999458	28.14
audikw_1	S	943695	943695	39297771	41.64
bone010	S	986703	986703	36326514	36.82
channel-500x100x100-b050	S	4802000	4802000	42681372	8.89
Cube_Coup_dt6	S	2164760	2164760	64685452	29.88
delaunay_n24	S	16777216	16777216	50331601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

Table: Matrices used for our experiments. Symmetric are marked by S, general unsymmetric by G.

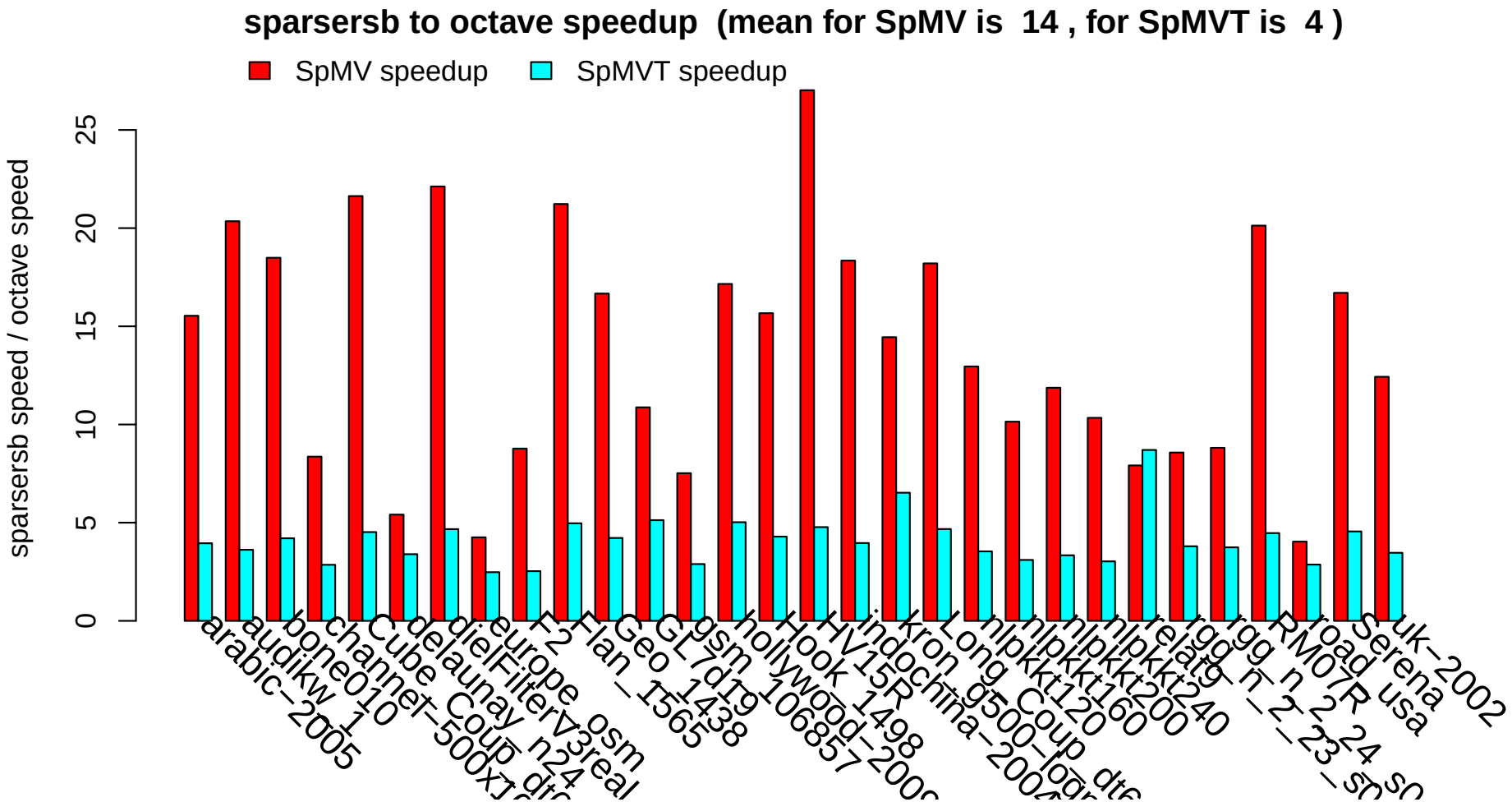
SpMV ("`Y+=A*B`") speed



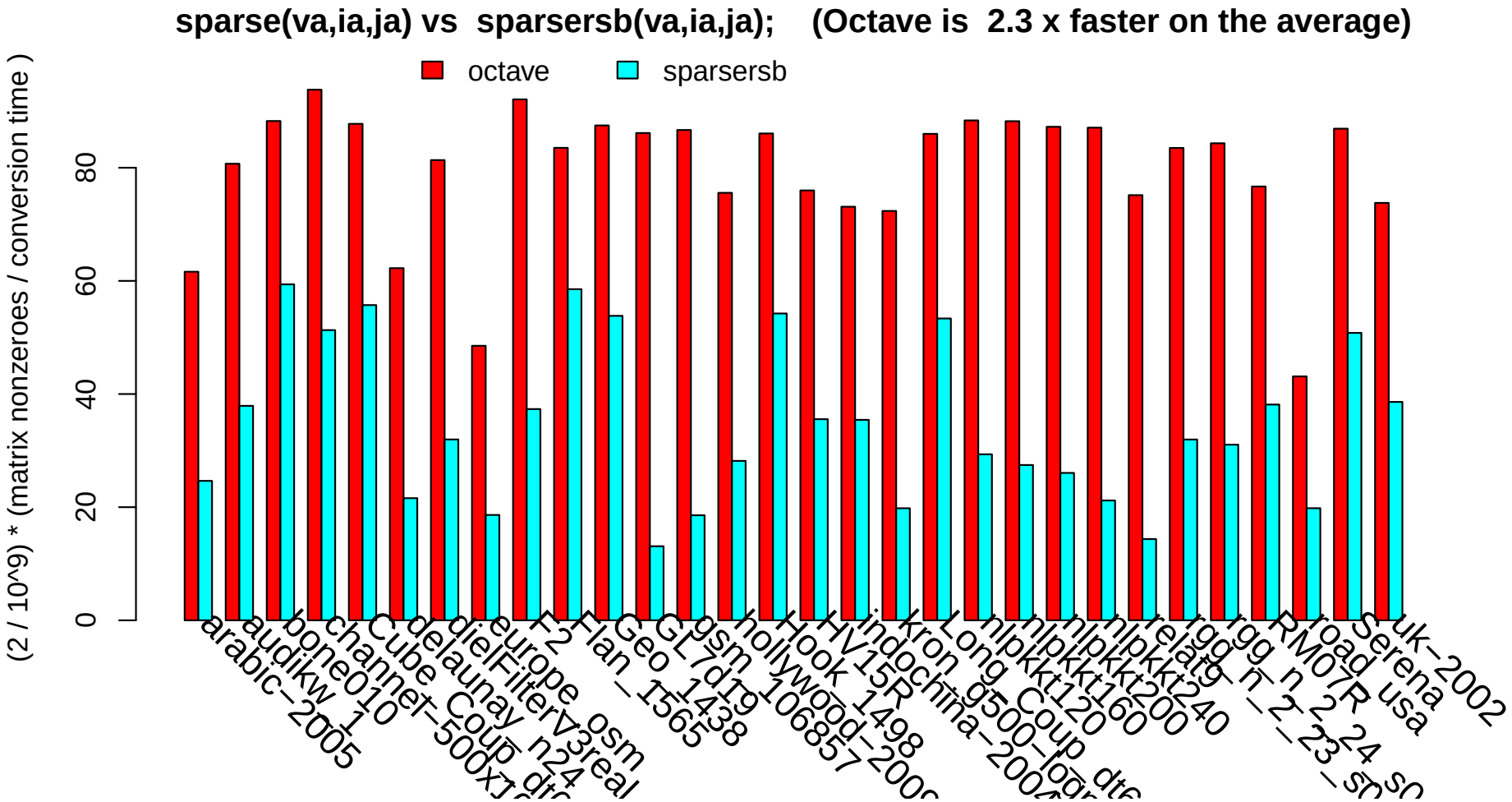
SpMV-T ("Y+=A.'*B") speed



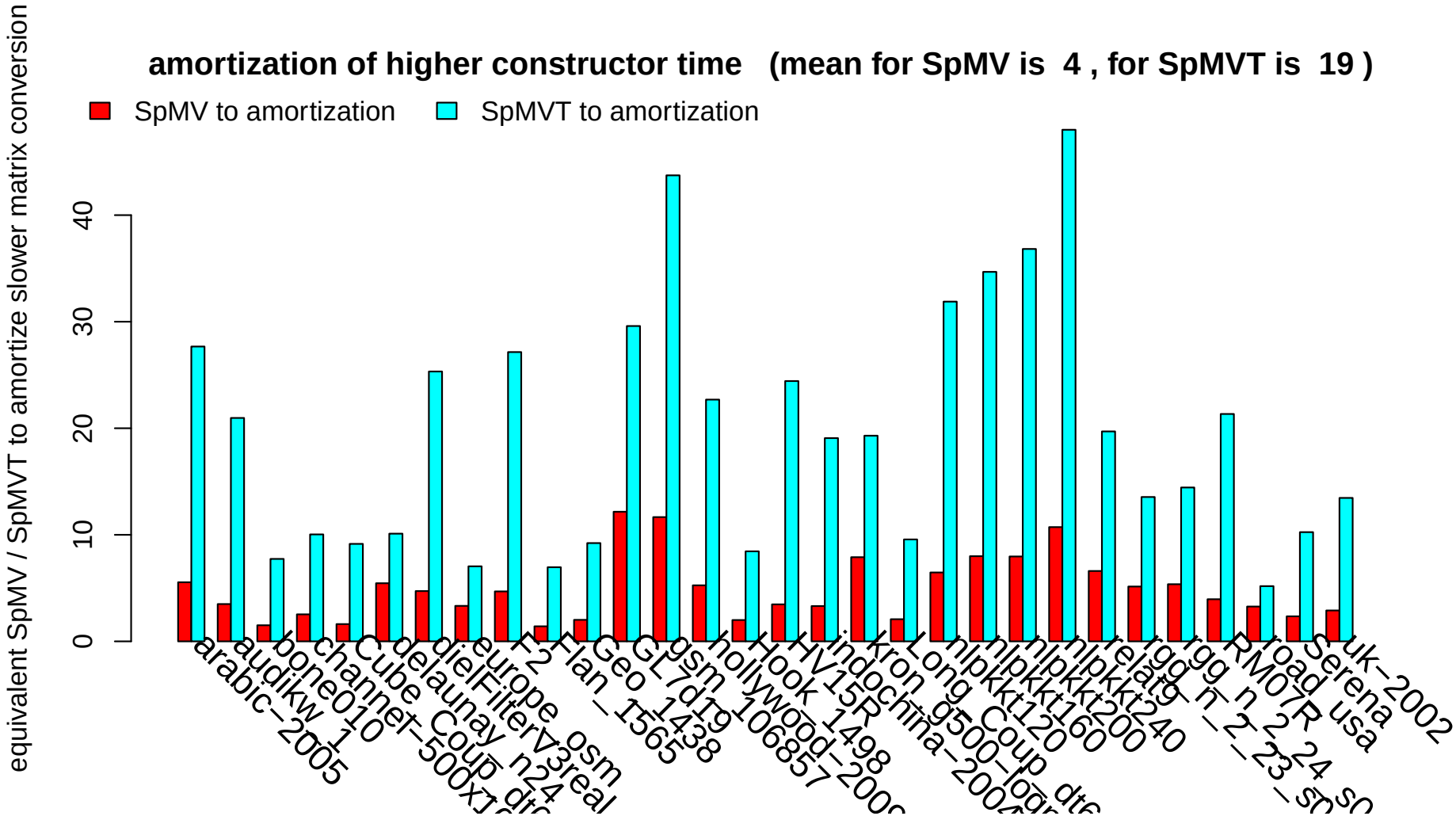
sparsersb speedup to Octave



Constructor ("sparsersb(ia,ja,va,nr,nj)") speed vs Octave's



Amortization of constructor in equivalent *SpMV* / *SpMV-T*



Performance observations

- $\approx 14\times$ faster than Octave's serial CSC in *SpMV*
- $\approx 4\times$ faster than Octave's serial CSC in *SpMV-T*
- $\approx 3\times$ slowdown w.r.t. Octave when assembling (from coordinate arrays)
- amortizes conversion overhead within a few dozen of multiplications

Low level introspection

```
A=sparsersb(sprand(3,3,0.5))
# A =
# Recursive Sparse Blocks  (rows = 3, cols = 3, nnz = 5 [56%])
#
#  (1, 2) -> 0.29
#  (2, 2) -> 0.21
#  (2, 3) -> 0.69
#  (3, 1) -> 0.63
#  (3, 2) -> 0.39
#
sparsersb(A,"get","RSB_MIF_MATRIX_INFO__TO__CHAR_P")
# ans = (3 x 3)[0x8586790] @ (0(0..0),0(0..0)) (5 nnz, 1.7 nnz/r)
# flags 0x2046186 (coo:1, csr:1), hw:1, storage: 40, subm: 3, symflags:
sparsersb(A,"get","RSB_MIF_LEAVES_COUNT__TO__RSB_BLK_INDEX_T")
# ans = 3
sparsersb(A,"get","RSB_MIF_INDEX_STORAGE_IN_BYTES_PER_NNZ__TO__RSB_REAL
# ans = 4.8
sparsersb(A,"get","RSB_MIF_TOTAL_SIZE__TO__SIZE_T")
# ans = 928
```

Setting librsb Options

Concept similar to that of Octave's spparms's.

```
sparsersb("set","RSB_IO_WANT_EXECUTING_THREADS","-1") # default
sparsersb("set","RSB_IO_WANT_EXECUTING_THREADS","1") # default m
sparsersb("set","RSB_IO_WANT_EXECUTING_THREADS","2")
sparsersb("set","RSB_IO_WANT_VERBOSE_ERRORS","7")
sparsersb("set","RSB_IO_WANT_MEMORY_HIERARCHY_INFO_STRING","")

# "get" will be available in version 1.1:
errlevel=sparsersb("get","RSB_IO_WANT_VERBOSE_ERRORS")
nthreads=sparsersb("get","RSB_IO_WANT_EXECUTING_THREADS")
```

Hand tuning the Subdivision Policy

```
A=sparsersb(ones(3000));
sparsersb(A,"get","RSB_MIF_LEAVES_COUNT__TO__RSB_BLK_INDEX_T")
# ans= 82

# A_20 will have less subdivisions
sparsersb("set","RSB_IO_WANT_SUBDIVISION_MULTIPLIER","2.0")
A_20=sparsersb(sparse(A));
sparsersb(A_20,"get","RSB_MIF_LEAVES_COUNT__TO__RSB_BLK_INDEX_T")
# ans= 1024

# A_05 will have more subdivisions
sparsersb("set","RSB_IO_WANT_SUBDIVISION_MULTIPLIER","0.5")
A_05=sparsersb(sparse(A));
sparsersb(A_05,"get","RSB_MIF_LEAVES_COUNT__TO__RSB_BLK_INDEX_T")
# ans= 256
```

These structures may expose different performance in usage.

Cache blocking policy

The exact RSB structure for a matrix depends on two parameters:

- thread count
- cache block size

These determine:

- parallelism
- memory bandwidth

Cache blocking trade-offs

Parallelism vs memory bandwidth:

- the finer the subdivision, the less contention among threads
- memory bandwidth the coarser the subdivision, less memory can be used to represent the matrix

Autotuning

Finds *best* performing matrix and threads count

- with respect to the specific *SpMV* variant:
  `sparsersb(A,"autotune",TRANSA,NRHS)`
- with respect to other parameters:
  `sparsersb(A,"autotune",TRANSA,NRHS,ITMAX,TMAX,TN,SF)`
- with defaults: `sparsersb(A,"autotune")`

Autotuning results

- Found to squeeze additional 10...60% performance
- (...at a cost of some thousands of *SpMV* ...)

sparsersb+librsb: In a nutshell

- bridge to `librsb` functionality for GNU/Octave
- on large matrices, `librsb` found to be faster than Intel's highly optimized, proprietary CSR implementation (see PMAA'12 results)
- portable, tunable code
- LGPLv3 licensed

Open questions and missing features

mainly

- non-standard operators e.g.: for row scaling: `A.+=v`
- missing operators
  - permutations; e.g.: `PA=A(IREN,JREN);`
  - linear access; e.g.: `A(1:nnz(A))`
- multi-operation benchmark/test suite (e.g.: shared with Octave)
- operators with more operands (e.g.: `Y=alpha*OM*X+beta*Y`)
- avoid *bitrot*

Future work

- stabilize usage interface and release
- implement missing operators
- make sure it works well with Octave's iterative solvers implemented in m-files (`gmres.m`, `pcg.m`, ...)
- compatibility with preconditioner functions (e.g.: ITSOL's, soon to come in Octave)
- operations for preconditioners; e.g.: *SpSV* ,*SpSV-T*

References

- Code and papers:
  `http://sourceforge.net/projects/librsb`
- `http://svn.code.sf.net/p/octave/code/trunk/octave-forge/main/sparsersb/src/sparsersb.cc`
- *Sparse Matrix Implementation in Octave*, 2006, by D. Bateman, A. Adler:
  `http://arxiv.org/abs/cs.MS/0604006`

Questions / discussion welcome!

Thanks for your attention.

Please consider testing `sparsersb`/`librsb`!
Spotting bugs/inefficiencies is essential for free software!

Extra: Autotuning functionality sample output (1/2)

```
Starting autotuning (10 x 3.000000 s stages, transa=T), -1 suggested starting threads.
* 1 threads: 1.106897 s.
  2 threads: 0.584075 s.
  3 threads: 0.397878 s.
Best threads choice is 3 (starting threads were 1).
Starting autotuner stage, with subdivision of 1.000000 (current threads=1, max threads = 1).
* 1 threads: 1.269518 s.
  2 threads: 0.605315 s.
  3 threads: 0.423535 s.
  4 threads: 0.327688 s.
  5 threads: 0.277200 s.
Best threads choice is 5 (starting threads were 3).
...
```

Extra: Autotuning functionality sample output (2/2)

```
...
Best threads choice is 12 (starting threads were 10).
For subdivision 4.000000 (2427 leaves, 2.6 bytes/nz, 12 threads), challenging 0.179623 s with 0.163669 s
Best performance obtained through subdivision multiplier of 4.000000 and 12 threads.
Gained a total speedup of 2.430991
In 1/10 tuner iterations, gained a total speedup of 2.430991
  10 threads: 0.172950 s.
  11 threads: 0.174151 s.
  12 threads: 0.166623 s.
...
Best threads choice is 12 (starting threads were 14).
For subdivision 16.000000 (9088 leaves, 2.6 bytes/nz, 12 threads), challenging 0.147328 s with 0.191135 s
Best performance obtained through subdivision multiplier of 4.000000 and 12 threads.
Gained a total speedup of 1.000000
In 7/10 tuner iterations, gained a total speedup of 2.719551
Auto tuning step 7 did not find a better configuration.
Skipping further auto tuning.
Estimated autotuner speedup: 2.719551 x
Setting autotuner-suggested thread count of 12 (will skip further thread number configurations!)
```

sparsersb.oct build

- Have a recent Octave installation; e.g.: version 3.6.3 [2]
- Checkout from Octave-Forge[3]
- Should compile `librsb` with -fPIC (any compiler)
- Should compile Octave with gfortran
- Build example: `./configure LIBRSB_CONFIG=/usr/local/bin/librsb-config; make; make tests;`

[2] `http://ftp.gnu.org/gnu/octave/octave-3.6.3.tar.bz2`

[3] `svn://svn.code.sf.net/p/octave/code/trunk/octave-forge/main/sparsers`