librsb: A Shared Memory Parallel Sparse BLAS Implementation using the Recursive Sparse Blocks format

Michele Martone

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

Sparse Days’13
Toulouse, France
June 17–18, 2013
presentation outline

Intro
librsb: a Sparse BLAS implementation
A recursive layout

Performance measurements
SpMV, SpMV-T SymSpMV performance
COO to RSB conversion cost
Auto-tuning RSB for SpMV / SpMV-T / SymSpMV

Conclusions
Summing up
References
Extra: RSB vs MKL plots
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**: $x \leftarrow \alpha L^{-T} x$
librsb: a high performance Sparse BLAS implementation

- uses the **Recursive Sparse Blocks (RSB)** matrix layout
- provides (for true) a Sparse BLAS standard API
- most of its numerical kernels are *generated* (from GNU M4 templates)
- extensible to any *integral* C type
- pseudo-randomly generated testing suite
- get/set/extract/convert/... functions
- \(\approx 47\) KLOC of C (C99), \(\approx 20\) KLOC of M4, \(\approx 2\) KLOC of GNU Octave
- bindings to C and Fortran (ISO C Binding), as *OCT module* for GNU Octave
- LGPLv3 licensed
design constraints of the Recursive Sparse Blocks (RSB) format

- parallel, efficient $SpMV / SpSV / COO \rightarrow RSB$
- in-place $COO \rightarrow RSB \rightarrow COO$ conversion
- no oversized COO arrays / no fillin (e.g.: in contrast to BCSR)
- no need to pad $x, y$ vectors arrays
- architecture independent (only C code, POSIX)
- developed on/for shared memory cache based CPUs:
  - locality of memory references
  - coarse-grained workload partitioning

\(^1\text{e.g. in } SpMV\)
we propose:

- a *quad-tree* of sparse *leaf* submatrices
- outcome of recursive *partitioning* in *quadrants*
- leaf submatrices are stored by either *row oriented Compressed Sparse Rows* (CSR) or *Coordinates* (COO)
- an *unified* format for Sparse BLAS\(^2\) operations and variations (e.g.: diagonal implicit, one or zero based indices, transposition, complex types, stride, ...)
- partitioning with regards to both the underlying cache size and available threads
- leaf submatrices are *cache blocks*

---

\(^2\)Sparse Basic Linear Algebra Subprograms, e.g.: as in TOMS Algorithm 818 (Duff and Vömel, 2002). or the specification in [http://www.netlib.org/blas/blast-forum/chapter3.pdf](http://www.netlib.org/blas/blast-forum/chapter3.pdf)
Instance of an Information Retrieval matrix (573286 rows, 230401 columns, $41 \cdot 10^6$ nonzeroes):

sparse blocks layout:  
spy plot: 

(courtesy from Diego De Cao, Univ. Roma Tor Vergata)
Adaptivity to threads count

Figure: Matrix \textit{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* (1)

\[
A = \sum_i A_i
\]

\[
y \leftarrow y + \sum_i A_i \times x_i, \text{ with leaf submatrices } A_i
\]
Multi-threaded \textit{SpMV} (2)

\[ y \leftarrow y + \sum_i A_i \times x_i \]

Threads \( t \in \{1..T\} \) execute concurrently:

\[ y_{it} \leftarrow y_{it} + A_{it} \times x_{it} \]

we prevent \textit{race conditions} performing \textit{busy wait}\(^3\); we use

\begin{itemize}
  \item per-submatrix visit information
  \item per-thread current submatrix information
\end{itemize}

to lock each \( y_{it} \) and avoid visiting submatrices twice.

The symmetric variant locks two intervals of \( y \), corresponding to \( A_i \) and its transpose.

\(^3\)To be improved!
Pros/Cons of RSB’s operations in librsb

- + parallel $SpMV$ / $SpMV-T$ / $SymSpMV$
- + parallel $SpSV$ / $SpSV-T$ (though less scalable than $SpMV$)
- + many other common operations (e.g.: parallel matrix build algorithm)
- - a number of known cases (e.g.: unbalanced matrices) where parallelism is poor
- - some algorithms easy to express/implement for CSR are more complex for RSB
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) matrices⁴.

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.

⁴See next slide for a list.
## Matrices

<table>
<thead>
<tr>
<th>matrix</th>
<th>symm</th>
<th>nr</th>
<th>nc</th>
<th>nnz</th>
<th>nnz/nr</th>
</tr>
</thead>
<tbody>
<tr>
<td>arabic-2005</td>
<td>G</td>
<td>22744080</td>
<td>22744080</td>
<td>639999458</td>
<td>28.14</td>
</tr>
<tr>
<td>audikw.1</td>
<td>S</td>
<td>943695</td>
<td>943695</td>
<td>39297771</td>
<td>41.64</td>
</tr>
<tr>
<td>bone010</td>
<td>S</td>
<td>986703</td>
<td>986703</td>
<td>36326514</td>
<td>36.82</td>
</tr>
<tr>
<td>channel-500x100x100-b050</td>
<td>S</td>
<td>4802000</td>
<td>4802000</td>
<td>42681372</td>
<td>8.89</td>
</tr>
<tr>
<td>Cube_Coup_dt6</td>
<td>S</td>
<td>2164760</td>
<td>2164760</td>
<td>64685452</td>
<td>29.88</td>
</tr>
<tr>
<td>delauay_n24</td>
<td>S</td>
<td>16777216</td>
<td>16777216</td>
<td>50331601</td>
<td>3.00</td>
</tr>
<tr>
<td>dieFilterV3real</td>
<td>S</td>
<td>1102824</td>
<td>1102824</td>
<td>45204422</td>
<td>40.99</td>
</tr>
<tr>
<td>europe_osm</td>
<td>S</td>
<td>50912018</td>
<td>50912018</td>
<td>54054660</td>
<td>1.06</td>
</tr>
<tr>
<td>Flan_1565</td>
<td>S</td>
<td>1564794</td>
<td>1564794</td>
<td>59485419</td>
<td>38.01</td>
</tr>
<tr>
<td>Geo_1438</td>
<td>S</td>
<td>1437960</td>
<td>1437960</td>
<td>32297325</td>
<td>22.46</td>
</tr>
<tr>
<td>GL7d19</td>
<td>G</td>
<td>1911130</td>
<td>1955309</td>
<td>37322725</td>
<td>19.53</td>
</tr>
<tr>
<td>gsm_106857</td>
<td>S</td>
<td>589446</td>
<td>589446</td>
<td>11174185</td>
<td>18.96</td>
</tr>
<tr>
<td>hollywood-2009</td>
<td>S</td>
<td>1139905</td>
<td>1139905</td>
<td>57515616</td>
<td>50.46</td>
</tr>
<tr>
<td>Hook_1498</td>
<td>S</td>
<td>1498023</td>
<td>1498023</td>
<td>31207734</td>
<td>20.83</td>
</tr>
<tr>
<td>HV15R</td>
<td>G</td>
<td>2017169</td>
<td>2017169</td>
<td>283073458</td>
<td>140.33</td>
</tr>
<tr>
<td>indochina-2004</td>
<td>G</td>
<td>7414866</td>
<td>7414866</td>
<td>194109311</td>
<td>26.18</td>
</tr>
<tr>
<td>kron_g500-logn20</td>
<td>S</td>
<td>1048576</td>
<td>1048576</td>
<td>44620272</td>
<td>42.55</td>
</tr>
<tr>
<td>Long_Coup_dt6</td>
<td>S</td>
<td>1470152</td>
<td>1470152</td>
<td>44279572</td>
<td>30.12</td>
</tr>
<tr>
<td>nlpkkt120</td>
<td>S</td>
<td>3542400</td>
<td>3542400</td>
<td>50194096</td>
<td>14.17</td>
</tr>
<tr>
<td>nlpkkt160</td>
<td>S</td>
<td>8345600</td>
<td>8345600</td>
<td>11893185</td>
<td>14.25</td>
</tr>
<tr>
<td>nlpkkt200</td>
<td>S</td>
<td>16240000</td>
<td>16240000</td>
<td>232232816</td>
<td>14.30</td>
</tr>
<tr>
<td>nlpkkt240</td>
<td>S</td>
<td>27993600</td>
<td>27993600</td>
<td>401232976</td>
<td>14.33</td>
</tr>
<tr>
<td>relat9</td>
<td>G</td>
<td>12360060</td>
<td>549336</td>
<td>38955420</td>
<td>3.15</td>
</tr>
<tr>
<td>rgg_n_2_23_s0</td>
<td>S</td>
<td>8388608</td>
<td>8388608</td>
<td>63501393</td>
<td>7.57</td>
</tr>
<tr>
<td>rgg_n_2_24_s0</td>
<td>S</td>
<td>16777216</td>
<td>16777216</td>
<td>132557200</td>
<td>7.90</td>
</tr>
<tr>
<td>RM07R</td>
<td>G</td>
<td>381689</td>
<td>381689</td>
<td>37464962</td>
<td>98.16</td>
</tr>
<tr>
<td>road_usa</td>
<td>S</td>
<td>23947347</td>
<td>23947347</td>
<td>28854312</td>
<td>1.20</td>
</tr>
<tr>
<td>Serena</td>
<td>S</td>
<td>1391349</td>
<td>1391349</td>
<td>32961525</td>
<td>23.69</td>
</tr>
<tr>
<td>uk-2002</td>
<td>G</td>
<td>18520486</td>
<td>18520486</td>
<td>298113762</td>
<td>16.10</td>
</tr>
</tbody>
</table>

**Table:** Matrices used for our experiments: General (G), Symmetric (S).
Comparison to MKL, Unsymmetric $SpMV$ /$SpMV-T$
(presented at PMAA’12)

Summarizing:$^5$

- 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.: the tall relat9) are problematic
- $SpMV$ and $SpMV-T$ have almost same performance (unlike row or column biased formats)
- scales better than MKL

$^5$Plots in extra slide: performance in Fig. 6, scalability in Fig. 7.
Comparison to MKL, \textit{SymSpMV} (presented at PMAA’12)

Summarizing:\textsuperscript{6}

\begin{itemize}
\item speedups up to around 200\% in several cases; most exceeding 50\%
\item scales slightly less than MKL
\end{itemize}

\textsuperscript{6}Plots in extra slide: performance in Fig. 8, scalability in Fig. 9.
Performance observations (presented at PMAA’12)

Summarizing:

- no architecture specific optimization employed
- $SpMV/SpMV-T$ equally parallel
- $SpMV-T/SymSpMV$ much faster than CSR
- parallel assembly ($CooToRSB$)
- 20-50 iterations of $SpMV-T/SymSpMV$ to amortize conversion cost and start saving time w.r.t. MKL’s $mkl_dcsrmv^7$

---

$^7$Relative conversion times plots in extra slide: Fig. 10 for general matrices; Fig. 11 for symmetric matrices. Relative amortization times plots: Fig. 12 for general matrices; Fig. 13 for symmetric matrices.
An RSB matrix instance may be non optimal

- memory bandwidth may saturate using part of available threads
- an excessively coarse partitioning limits parallelism (contention in accessing the result vector!)

Figure: Left: max parallelism is 8; right, more parallelism, but also more indexing overhead.
Run-time Empirical Autotuning

Tuning algorithm or data structures at run time using benchmarking.

- notable sparse precursor: OSKI (Vuduc et al., 2005)
  - exposes tuning cost, support self-profiling
  - e.g.: best BCSR kernel(s)/decomposition choice
- sparse, planned: for CSB (sketched in Buluč et al., 2011)
  - tuning the operation data structures (not the matrix itself)
  - e.g.: multiple temporary result vectors
Simple Autotuning in RSB

The user specifies:

- matrix and $SpMV$ parameters (transposition, stride, number of right hand sides)
- optionally, time to spend in $SpMV$ and initial thread count

Starting with 1 thread, repeat until no more improvement:

- thread count is increased linearly (+1)
- the RSB matrix is re-partitioned as we had $(1/4, 1/2, \times 1, \times 2, \times 4)$ the current cache
- each (matrix instance, threads) combination performance is measured
- the best improving combination is kept

\[\text{To be released soon.}\]
**SpMV** auto-tuning

![Graph showing speedup of SpMV performance on Sandy Bridge, improvement over untuned 16 threaded RSB, general matrices.](image)

**Figure:** *SpMV* performance on Sandy Bridge, improvement over untuned 16 threaded RSB, general matrices.

- **arabic-2005:** 16 → 13 threads, 9088 → 2427 leaves, 2.64 idx.bytes/nnz
- **HV15R:** 16 → 11 threads, 2366 → 795 leaves, 2.6 idx.bytes/nnz
- **indochina-2004:** 16 → 13 threads, 2423 → 626 leaves, 2.43 → 2.40 idx.bytes/nnz
- **relat9:** 16 → 8 threads, 254 → 70 leaves, 7.9 → 7.0 idx.bytes/nnz
- **RM07R:** 16 → 9 threads, 430 leaves, 2.46 idx.bytes/nnz
- **uk-2002:** 16 → 11 threads, 4504 → 2233 leaves, 2.59 → 2.57 idx.bytes/nnz
Figure: *SpMV-T* performance on Sandy Bridge, improvement over untuned 16 threaded RSB, general matrices.

- **arabic-2005**: 16 → 9 threads, 2427 leaves, 2.64 idx.bytes/nnz
- **HV15R**: 16 → 12 threads, 2366 → 795 leaves, 2.6 idx.bytes/nnz
- **indochina-2004**: 16 → 15 threads, 2423 → 626 leaves, 2.43 → 2.40 idx.bytes/nnz
- **relat9**: 16 → 11 threads, 254 → 1078 leaves, 7.9 → 7.0 idx.bytes/nnz
- **RM07R**: 16 → 11 threads, 430 → 100 leaves, 2.46 → 2.59 idx.bytes/nnz
- **uk-2002**: 16 → 9 threads, 4504 → 2233 leaves, 2.59 → 2.57 idx.bytes/nnz
SymSpMV auto-tuning

Figure: SymSpMV performance on Sandy Bridge, improvement over untuned 16 threaded RSB, symmetric matrices.

- **europe_osm**: 16 → 6 threads, 532 → 2638 leaves, 8.0 → 4.5 \(\text{idx.bytes/nnz}\)
- **nlpkkt240**: 16 → 8 threads, 450 → 545 leaves, 3.17 → 3.10 \(\text{idx.bytes/nnz}\)
- **road_usa**: 16 → 10 threads, 237 → 2211 leaves, 8.0 → 5.48 \(\text{idx.bytes/nnz}\)
- ...
Auto-tuning: observations

- very sparse and symmetric matrices may need additional subdivisions
- \textit{SpMV} / \textit{SpMV-T} tuning strategies can differ significantly for non square matrices
- current strategy costs thousands of \textit{SpMV} ’s
Conclusions

- on large matrices, librsb competes with Intel’s highly optimized, proprietary CSR implementation
- many aspects can be improved; e.g.:
  - overcome busy wait based locking
  - optimize auto-tuning
  - ...
- a revision of Sparse BLAS to handle e.g.: auto-tuning semantics would be useful
Sparse BLAS:


librsb:

- [http://sourceforge.net/projects/librsb](http://sourceforge.net/projects/librsb)
Questions / discussion welcome!

Thanks for your attention.

Please consider testing librsb: spotting bugs/inefficiencies is essential for free software!
Comparison to MKL, SpMV & SpMV-T

**Figure:** Transposed/Non transposed SpMV performance on Sandy Bridge, versus MKL’s CSR, 16 threads, unsymmetric matrices.
Comparison to MKL, \(SpMV/SpMV-T\) scalability

**Figure:** Unsymmetric matrices. \(SpMV/SpMV-T\) parallel speedup (16 to 1 threads performance ratio).
Comparison to MKL, *SymSpMV*

![Graph showing SymSpMV performance on Sandy Bridge, versus MKL’s CSR, 16 threads (symmetric matrices).](image)

**Figure:** *SymSpMV* performance on Sandy Bridge, versus MKL’s CSR, 16 threads (symmetric matrices).
Comparison to MKL, SymSpMV scalability

Figure: Symmetric matrices. SymSpMV parallel speedup (16 to 1 threads performance ratio).

Back to page 4.
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

<table>
<thead>
<tr>
<th>audikw_1</th>
<th>bone010</th>
<th>channel−500x100x100−b050</th>
<th>Cube_Coup_dt6</th>
<th>delaunay_n24</th>
<th>dielFilterV3real</th>
<th>europe_osm</th>
<th>Flan_1565</th>
<th>Geo_1438</th>
<th>gsm_106857</th>
<th>hollywood</th>
<th>Hook_1498</th>
<th>kron_g500−logn20</th>
<th>Long_Coup_dt6</th>
<th>nlpkkt120</th>
<th>nlpkkt160</th>
<th>nlpkkt200</th>
<th>nlpkkt240</th>
<th>nlpkkt280</th>
<th>nlpkkt320</th>
<th>road_usa</th>
<th>Serena</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** Non transposed conversion-to-\(SpMV\) times ratio on Sandy Bridge, symmetric matrices, 1 and 16 threads.
Amortization of conversion cost, $SpMV$, $SpMV-T$

Figure: Unsymmetric matrices. Amount of $SpMV$/$SpMV-T$ executions with RSB necessary to amortize time of $CooToRSB$, and get advantage over MKL. 16 threads.

15 Back to page 1.
Amortization of conversion cost, *SymSpMV*

Figure: Symmetric matrices. Amount of *SymSpMV* executions with RSB necessary to amortize time of *CooToRSB*, and get advantage over MKL.16 threads.