Performance Improvement of LIBRSB-1.3 Sparse BLAS on EuroHPC-class Systems

Michele MARTONE* **(michele.martone@lrz.de)**

Leibniz Supercomputing Centre, Garching bei München, Germany

Context: Project LyNcs

project "LyNcs": *"Linear Algebra, Krylov-subspace methods, and multi-grid solvers for the discovery of New Physics"*

subproject of *PRACE-6IP Grant agreement ID: 823767*, Work Package 8 (*"Forward-looking software solutions towards Exascale"*)

synergy of expertises and codes spanning the entire software stack:

- LQCD, The Cyprus Institute: DDalphaAMG, LyNcs-API
- numerical linear algebra, Inria Bordeaux (France): Fabulous
- **portable performance kernels**, LRZ: LIBRSB

this poster: main LIBRSB achievements during the project

Numerical Techniques of Interest to LyNcs

- iterative methods: *block Krylov*
- require efficient Sparse Matrix-Matrix multiplication aka SpMM

SpMM in matrix form (m aka $NRHS$ aka *number of right hand sides*):

$$\overbrace{\begin{bmatrix} c_{11} & \dots & c_{1m} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nm} \end{bmatrix}}^{updated\ dense\ C} \leftarrow \beta \overbrace{\begin{bmatrix} c_{11} & \dots & c_{1m} \\ \vdots & \ddots & \vdots \\ c_{n1} & \dots & c_{nm} \end{bmatrix}}^{dense\ C} + \alpha \overbrace{\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}}^{sparse\ A} \overbrace{\begin{bmatrix} b_{11} & \dots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nm} \end{bmatrix}}^{dense\ B}$$

More on the methods of our interest:

R. B. Morgan. *Restarted block GMRES with deflation of eigenvalues*, Appl. Numer. Math., 54(2):222–236, 2005. ▪ M. Robbe and M. Sadkane. *Exact and inexact breakdowns in the block GMRES method*, Linear Algebra Appl., 419:265–285, 2006. ▪ E. Agullo, L. Giraud, and Y.-F. Jing. *Block GMRES method with inexact breakdowns and deflated restarting*, SIAM J. Matrix Anal. Appl., 35(4):1625–1651, 2014.

LIBRSB

project page: `http://librsb.sf.net`

- >100KLOC of C99, OpenMP, and modern templated C++
- node-level **shared-memory**-parallel operations, for:
  - Sparse BLAS: matrix assembly/destroy, **SpMM**, triangular solve
  - *interactive* applications (update, sparse-sparse ops, conversions)
  - *distributed-memory applications* (block extract, update, etc.)
- with own interface in C/C++ and Fortran
  - with Sparse BLAS interface (BLAS Technical Forum Standard)
  - with interfaces for GNU Octave and Python interpreters
- **LGPLv3-licensed** *free software*, available via Spack, Guix-HPC, EasyBuild, and on Debian, Ubuntu, OpenSUSE, Windows,...

New in LIBRSB-1.3: Modern C++ interface

```
#include <rsb.hpp>
#include <vector>
#include <array>
using namespace ::rsb;

int main() {
  RsbLib rsblib;
  const int nnzA { 7 }, nrhs { 2 };
  const int nrA { 6 }, ncA { 6 };
  const std::vector<int>    IA {0,1,2,3,4,5,1};
  const int                 JA [] = {0,1,2,3,4,5,0};
  const std::vector<double> VA {1,1,1,1,1,2}, B(nrhs * ncA,1);
  std::array<double,nrhs * nrA> C;
  const double alpha {2}, beta {1};

  RsbMatrix<double> mtx(IA,JA,VA,nnzA);// Notice above declarations of IA,JA,VA

  mtx.spmm(RSB_TRANSPOSITION_N, alpha, nrhs, RSB_FLAG_WANT_ROW_MAJOR_ORDER, B,
     beta, C); // ditto for B and C
}
```

Figure 1: Program using the new *modern C++* interface to create a matrix and multiply it by a vector. Usage of C++20's `std::span` in the declaration of `RsbMatrix()` and `spmm()` ensures type safety while not prescribing usage of any specific *array* type.

GNU Octave + liboctave + LIBRSB = SparseRSB

- GNU Octave: a MATLAB-like interactive numerical language
- liboctave: access Octave via C++

```
octave:1> A = sparsersb ( sparse (rand (3) > .6) )
A =

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

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

Figure 2: The `sparsersb` keyword allows transparent usage of LIBRSB in the same way as the `sparse` built-in. Part of Octave Forge.

Python + Cython + LIBRSB = PyRSB

- SciPy: popular Python scientific computing API
- Cython: *optimising static compiler* for C extensions to Python

```
import numpy
import scipy
from rsb           import rsb_matrix
V=[ 11.,12.,22.]
I=[   0,   0,   1]
J=[   0,   1,   1]
a = rsb_matrix((V, (I, J)), [3,3])
 ...
y = y + a * x;
```

Figure 3: PyRSB's usage is styled after SciPy's `csr_matrix`.

Recursive Sparse Blocks (RSB) Layout

- for *large* matrices (uses *cache locality*, *coarse thread parallelism*)
- supports *autotuning* (*layout* adjusted to maximize performance)

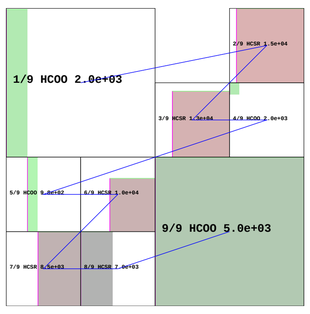

Figure 4: Instance of classical test matrix *bayer02* ($14k \times 14k, 64k$ nonzeroes). Black-bordered boxes are *sparse blocks*, and are Z-ordered. Greener have fewer nnz than average, redder have more. Either in *"Coordinate"* format (COO) or *"Compressed Sparse Rows"* (CSR). Blocks rows (LHS) and columns (RHS) ranges evidenced (left and top side).

Improved in LIBRSB-1.3: SpMM + Fortran Layout

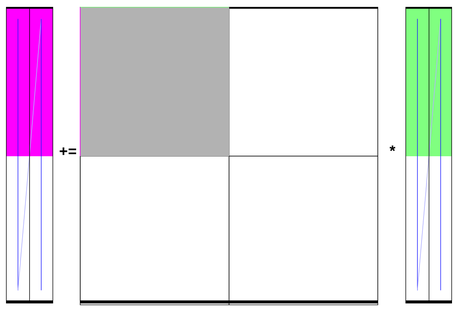

Figure 5: A matrix and its two SpMM operands stored in **column-major** (also known as Fortran) order. The matrix consists of four *sparse blocks*, of which one is highlighted. Each operand is stored one column after the other, with memory following the blue line. Consequently, each operand's two column portions pertaining a given sparse block **are not contiguous**. LIBRSB-1.2 would multiply each sparse block by each contiguous subvector. LIBRSB-1.3 can now perform this in one pass only.

Improved in LIBRSB-1.3: SpMM + C Layout

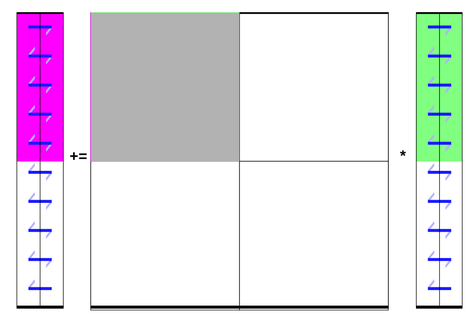

Figure 6: A matrix and its two SpMM operands stored in **row-major** (also known as C) order. The matrix consists of four *sparse blocks*, of which one is highlighted. Each operand stores one row after the other, with memory following the blue line. Consequently, the two column portions operands pertaining a given sparse blocks **are contiguous**. Here too LIBRSB-1.2 kernels would multiply each sparse block repeatedly. LIBRSB-1.3 can perform this in one pass only. Note that for NRHS=1, by-rows and by-columns are equivalent.

Setup of SpMM Performance Experiment

- Five machines with different architectures
  - `amd64`: smng (Intel Skylake), rome (AMD Rome), coop (Intel Cooper Lake)
  - `arm64`: thx (Marvell ThunderX2), a64fx (Fujitsu A64FX)
- Same input data and parameter ranges
  - 24 threads, autotuning on
  - 1,2,4 NRHS (*right hand sides* operands to SpMM)
  - Operands layouts: by-rows and by-columns
  - The four BLAS numerical types
  - A selection of 44 matrices (symmetric and unsymmetric)
- Compilation flags
  - smng,rome: `icc -ipo -O3 -no-prec-div -fp-model fast=2 -xHost`
  - `gcc -Ofast -march=native -mtune=native` elsewhere

Speedup of by-rows Layout over by-columns Layout

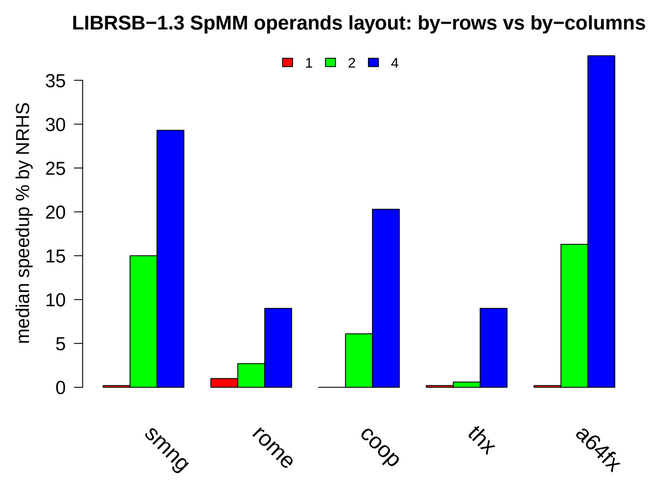

Figure 7: For each combination of matrix and numerical type, by-rows-layout SpMM performance is divided by by-columns-layout performance, and here grouped by machine and NRHS. For SpMM (NRHS>1), by-rows layout is now the recommended layout (in LIBRSB-1.2, by-columns was the optimal one). With NRHS=2 it improves up to 16%; with NRHS=4, by 9–38%, depending on the architecture.

Speedup of version 1.3 over version 1.2

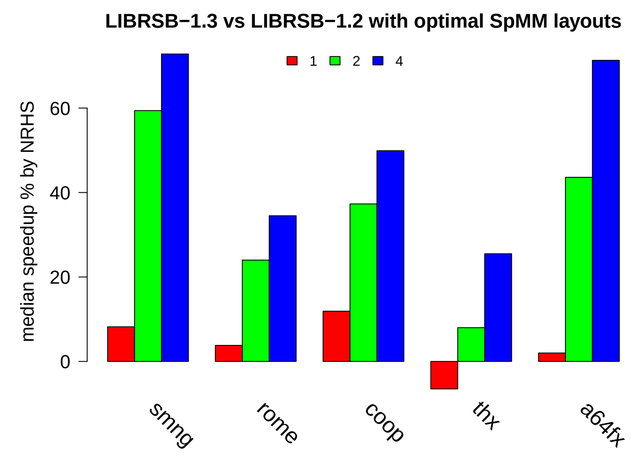

Figure 8: With the exception of SpMV on thx, 1.3-to-1.2 ratios median values indicate an SpMM improvement across all architectures. The SpMV (NRHS=1) speedup median improvement amounts to a few percentage points (for improvements obviously unrelated to the operands layout). If using the per-version recommended operands' layout, SpMM with NRHS=2 improves by 8–59%, SpMM with NRHS=4 by 25–73%, depending on the architecture.

Conclusions

The LyNcs project allowed improving LIBRSB in:

- Performance of SpMM (*by-rows* is optimal operands layout now)
- Usability via C++, GNU Octave, Python
- Many aspects of robustness and overall quality (e.g. tests, coverage, *continuous integration* — details out of the scope of this poster)

Acknowledgements

The Implementation Phase of PRACE receives funding from the EU's Horizon 2020 Research and Innovation Programme (2014–2020) under grant agreement 823767.

The author gratefully acknowledges the Gauss Centre for Supercomputing e.V. (`https://www.gauss-centre.eu`) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at the Leibniz Supercomputing Centre (`https://www.lrz.de`). Part of the performance results have been obtained on systems in the test environment BEAST (Bavarian Energy Architecture & Software Testbed) also at LRZ.

The EuroHPC Summit Week conference series receives funding from the European Union's Horizon 2020 research and innovation programme under grant agreements 823767 (PRACE-6IP) and 824151 (HPC-GIG).

#EHPCSW