EuroHPC Joint Undertaking

PRACEdays21

EuroHPC Summit Week

#PRACEdays

lrz

Portable and Efficient Sparse Matrices Computations: from Interactive Notebook to PRACE Tier-0 Systems

PRACE

Michele MARTONE

Leibniz Supercomputing Centre, Garching bei München, Germany

Project LyNcs: HPC APIs for LQCD

Lattice Quantum Chromodynamics (LQCD): subatomic physics

a major fraction of HPC usage

€ PRACE-6IP Grant agreement ID: 823767, Project LyNcs:

synergy of expertises and codes

- numerical linear algebra, Inria Bordeaux (France): FABULOUS
- LQCD: The Cyprus Institute: DDALPHAAMG, LYNCS-API
- **portable performance kernels**, LRZ: LIBRSB (**this poster**)

Numerical Techniques of Interest

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

SpMM in matrix form:

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

More on the methods of our interest:

Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003 ▪ R.W. Freund, M. Malhotra, *A block QMR algorithm for non-Hermitian linear systems with multiple right-hand sides*, Linear Algebra Appl. 254 (1997) 119–157. ▪ 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: A Sparse BLAS Library

- *Recursive Sparse Blocks* (RSB) layout
- operations for *distributed-memory applications*
  (e.g. **shared-memory** SpMM, triangular solve, ...), but also
- operations for *interactive* applications
  (e.g. matrix update, sparse-sparse sum/multiplication, conversions)
- dual API:
  - own interface in C/C++ and FORTRAN
  - Sparse BLAS (BLAS Technical Forum Standard)
- portable **LGPLv3-licensed** *free software*

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

```cpp
#include <rsb.hpp>
#include <vector>
#include <iostream>
auto main() -> int {
  RsbLib rsblib;
  const rsb_nnz_idx_t nnzA { 7 };
  const rsb_coo_idx_t nrA { 6 }, ncA { 6 }, nrhs { 2 };
  std::vector<rsb_coo_idx_t> IA {0,1,2,3,4,5,1}, JA {0,1,2,3,4,5,0};
  std::vector<double> VA {1,1,1,1,1,1,2}, X(nrhs*ncA,1);
  std::vector<double> Y(nrhs*nrA,0);
  const double alpha {2}, beta {1};
  rsb_int_t tn {0};
  rsb_real_t sf {0}; // speedup factor
  const rsb_flags_t order {RSB_FLAG_WANT_COLUMN_MAJOR_ORDER};

  RsbMatrix<double> mtx(IA,JA,VA);

  mtx.tune_spmm(sf,RSB_TRANSPOSITION_N,alpha,nrhs,order,X,beta,Y);
  std::cout << " expected tuning speedup of " << sf << "x\n";
  mtx.spmm(RSB_TRANSPOSITION_N,alpha,nrhs,order,X,beta,Y);
}
```

Figure 1: Program using the new C++20 interface to create a matrix, *tune it* with respect to specific operation parameters, and multiply it by a vector. Here error handling is *exception*-based.

RSB Layout and Algorithms

- >100KLOC of C99 + OPENMP + generated specialized kernels
- intended for *large* matrices, where it favours:
  + cache locality
  + coarse thread parallelism
- supports *online empirical autotuning*

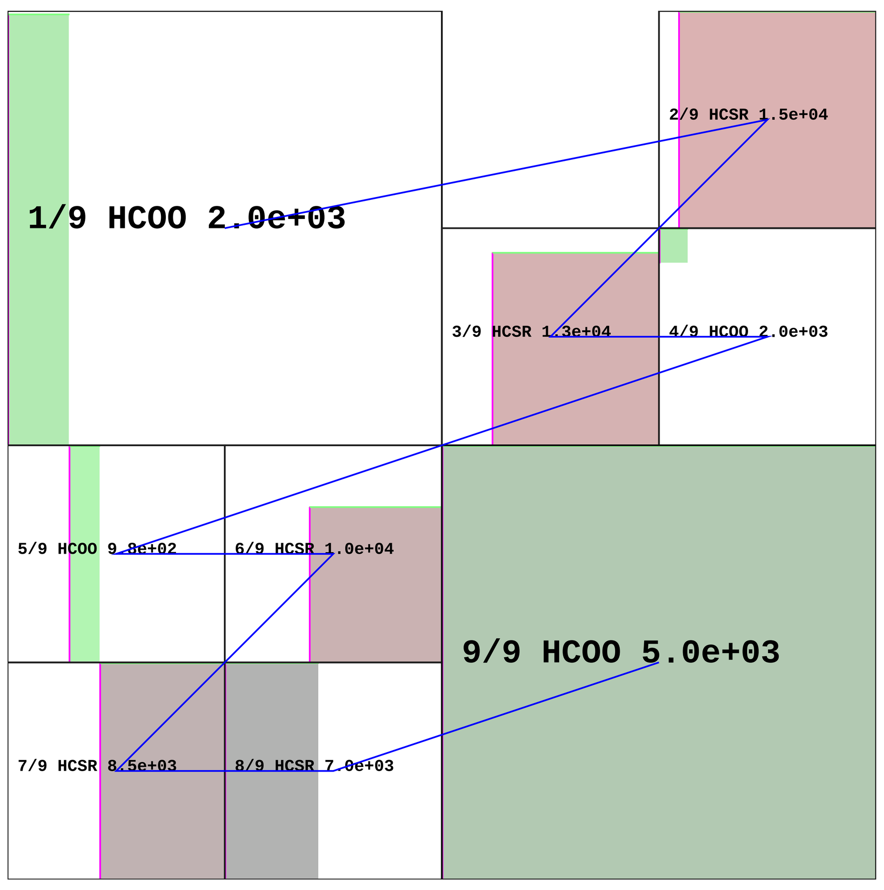

Figure 2: RSB instance of classical test matrix *bayer02* ($14k \times 14k$, $64k$ nonzeroes). Black-bordered boxes are *sparse blocks*, and are Z-ordered. Greener have fewer nnz than average, redder have more. Blocks' rows (LHS) and columns (RHS) ranges evidenced (left and top side).

Matrices of Interest

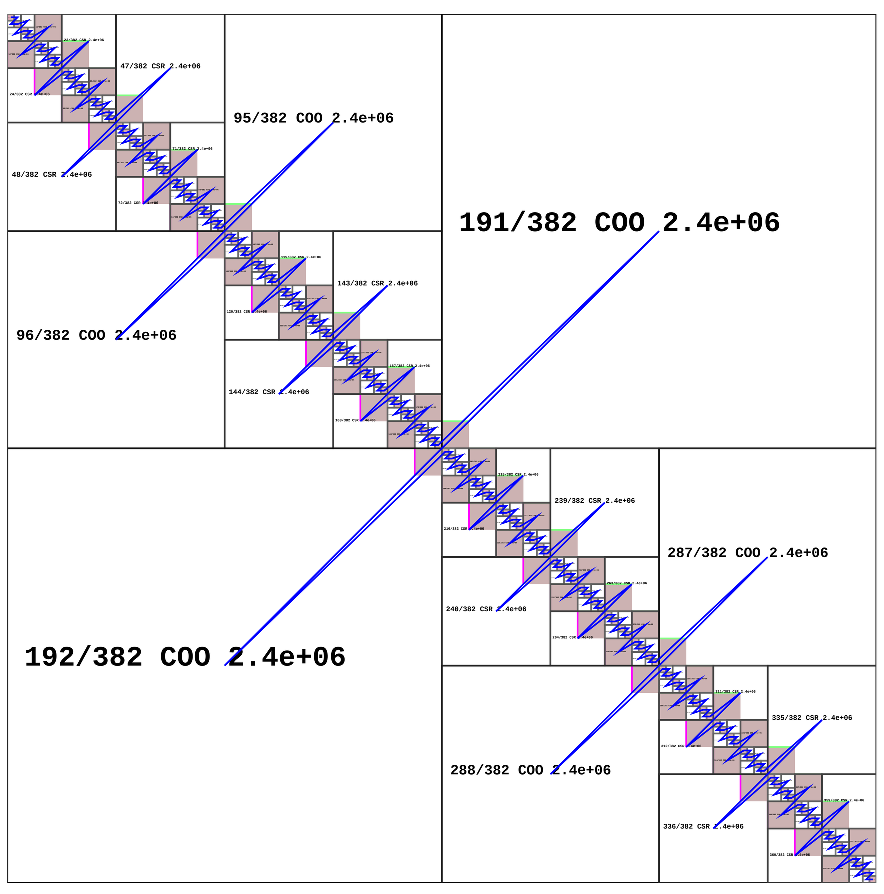

Figure 3: Instance of matrix representing the Wilson Dirac Operator. Used in Lattice QCD to simulate quarks. Square, $12M$ equations, $597M$ nonzeroes. Matrix: courtesy of Dr. Jacob Finkenrath.

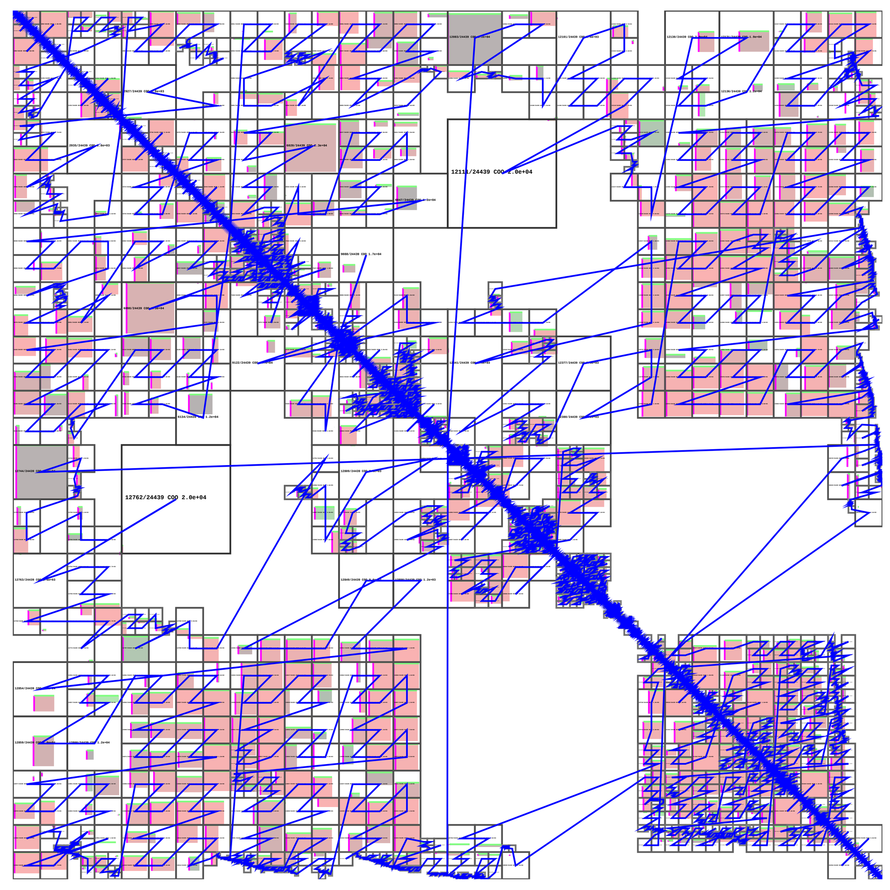

Figure 4: Nanoscale light-matter interactions equations matrix. Square complex unsymmetric, with $2M$ equations and $281M$ nonzeroes. Here, as 1934 COO and 22479 CSR sparse blocks. This instance occurs during autotuning's *search* for a well-performing blocking on an AMD ROME EPYC 7742, with 16 threads spread across the cores. Matrix: courtesy Dr. Luc Giraud and Dr. Stéphane Lanteri, Inria.

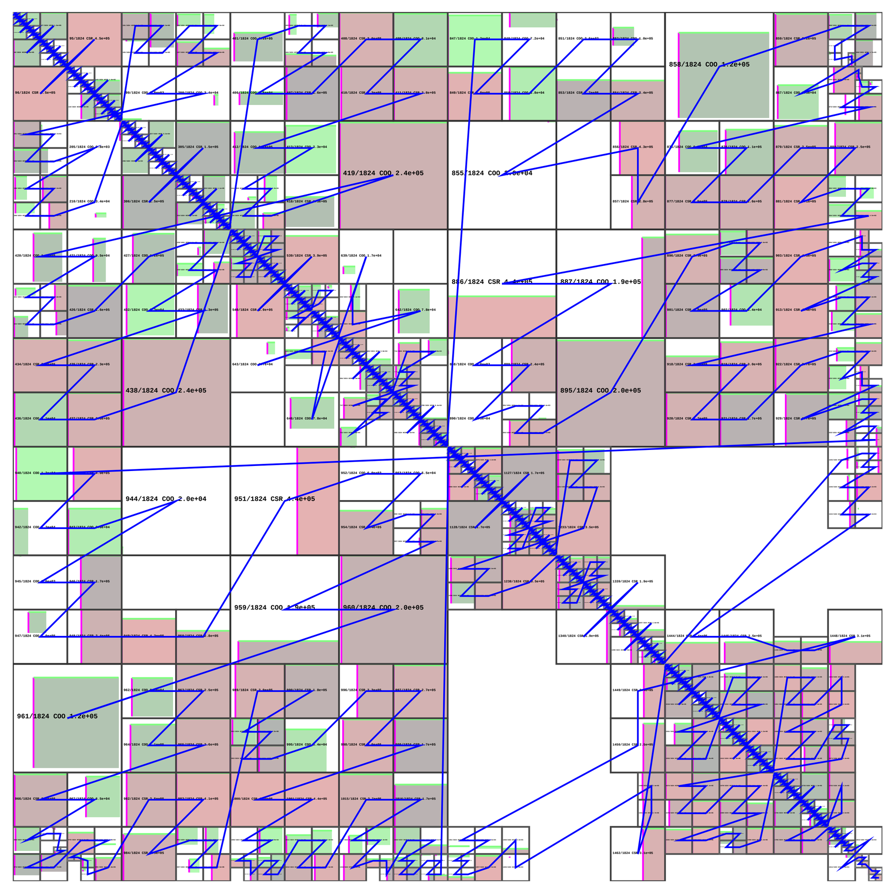

Figure 5: Same matrix. *Best* autotuning-determined layout for SpMM with 2 RHS: 170 COO and 1530 CSR blocks. Using ***sh**ort* indices (therefore **H**COO, **H**CSR): indexing averages to only 2.64 bytes per nnz. Performed SpMM 40% faster than an initial "first guess" structure with 7173 blocks. Autotuning took less than a minute. SpMM operation time slightly (12%) faster than CSR with INTEL MKL "20200822".

Current Development Focus

- performance improvement of SpMM kernels
- partners' use cases, novel architectures (LRZ and Inria testbeds)
- quality improvement of internals (bugfixes, CI/CD) and interfaces

One Library, many Interfaces

- original: `rsb.h`
- machine-translated:
  - FORTRAN *module*, via script
  - Sparse BLAS headers and module, via GNU M4
  - PYTHON: via script + CYTHON
- GNU OCTAVE: written in C++ using `rsb.h` and `liboctave`

Features access:

- all: C/C++
- most: FORTRAN
- many: PYRSB, OCTAVE-SPARSERSB
- limited: Sparse BLAS API

Performance:

- in C and FORTRAN, **zero overhead**
- **optimal** if used natively compiled (likely auto-vectorization) (also available on SPACK and GUIX-HPC)
- **portable** if used precompiled (e.g. `# apt install librsb0 octave-sparsersb`)

PYRSB vs SCIPY CSR and SPARSERSB vs OCTAVE CSC:

- $1 \approx 10\times$ faster (on large matrices)
- SCIPY CSR and OCTAVE CSC both serial
- PYRSB and SPARSERSB have intrinsic *copy overheads*

For JULIA users: interface by D. C. Jones
`https://github.com/dcjones/RecursiveSparseBlocks.jl`

GNU Octave + LIBRSB = SparseRSB

- GNU OCTAVE: a MATLAB-like interactive numerical language
- SPARSERSB: *package* to access LIBRSB **transparently**

```
octave:1> R = ( rand (3) > .6 )
R =

   0   0   0
   0   0   0
   1   0   1

octave:2> A_octave = sparse (R)
A_octave =

Compressed Column Sparse (rows = 3, cols = 3, nnz = 2 [22%])

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

octave:3> A_librsb = sparsersb (R)
A_librsb =

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

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

Figure 6: Package with only one new keyword. Most arithmetical and conversion operators and builtins acting on `sparse` work on `sparsersb` object as well.

`http://octave.sourceforge.net/sparsersb/`

Python + Cython + LIBRSB = PyRSB

SciPy.org Cython

- SCIPY: popular Python scientific computing API
- CYTHON: *optimising static compiler* for C extensions to PYTHON
- PYRSB: extension module to access LIBRSB **transparently**

```python
import numpy
import scipy
from scipy.sparse import csr_matrix
from rsb           import rsb_matrix

V=[ 11.,12.,22.]
I=[   0,   0,   1]
J=[   0,   1,   1]

c = csr_matrix((V, (I, J)))

a = rsb_matrix((V, (I, J)))
a = rsb_matrix((V, (I, J)), [3,3])
a = rsb_matrix((V,  I, J))

y = y + a * x; # equivalent to y = y + c * x
```

Figure 7: `rsb_matrix` usage is styled after SciPY's `csr_matrix`.

`https://github.com/michelemartone/pyrsb`

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

#EHPCSW