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

Michele Martone

High Level Support Team @ MPG-IPP’s RZG

Garching — October 24, 2011
numerical matrices which are *large* and populated mostly by zeros
ubiquitous in scientific/engineering computations (e.g.: PDE)
used in *information retrieval* and *document ranking*
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: \textit{performance}=\textit{time efficiency}
Focus of librsb

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):

- **SpMV**: “$y ← A x$”
- **SpMV-T**: “$y ← A^T x$”
- **SpSV**: “$x ← L^{-1} x$”
- **SpSV-T**: “$x ← L^{-T} x$”
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 *row oriented Compressed Sparse Rows* (CSR) or *Coordinate* (COO)
- an *unified* format for Sparse BLAS\(^1\) operations
- partitioning with regards to both the underlying cache size and available threads

\(^1\)Basic Linear Algebra Subprograms
Instance of an Information Retrieval matrix (573286 rows, 230401 columns, $41 \cdot 10^6$ nonzeroes):

Courtesy of Diego De Cao.
We compute $y_1$ in the first thread, $y_2$ in the second:

\[
\begin{vmatrix}
  y_1 \\
  y_2
\end{vmatrix} = A \begin{vmatrix}
  x_1 \\
  x_2
\end{vmatrix} = \begin{vmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{vmatrix} \begin{vmatrix}
  x_1 \\
  x_2
\end{vmatrix}
\]

\[
= \begin{vmatrix}
  A_{11} & A_{12} \\
  0 & 0
\end{vmatrix} \begin{vmatrix}
  x_1 \\
  x_2
\end{vmatrix} + \begin{vmatrix}
  0 & 0 \\
  A_{21} & A_{22}
\end{vmatrix} \begin{vmatrix}
  x_1 \\
  x_2
\end{vmatrix}
\]

\[
= \begin{vmatrix}
  A_{11}x_1 + A_{12}x_2 \\
  0
\end{vmatrix} + \begin{vmatrix}
  0 \\
  A_{21}x_1 + A_{22}x_2
\end{vmatrix}
\]

Recursion continues on each thread
Single threaded recursive $SpSV$

$Lx = b \Rightarrow \begin{vmatrix} L_1 & 0 & |x_1 | \\ M & L_2 & |x_2 | \\ \end{vmatrix} = \begin{vmatrix} b_1 \\ b_2 \end{vmatrix}$

$x = \begin{vmatrix} x_1 \\ x_2 \end{vmatrix} = L^{-1} b = \begin{vmatrix} L_1 & 0 \\ M & L_2 \end{vmatrix}^{-1} \begin{vmatrix} b_1 \\ b_2 \end{vmatrix} = \begin{vmatrix} L_1^{-1} b_1 \\ L_2^{-1} (b_2 - Mx_1) \end{vmatrix}$

This computation is executed recursively.
Pros/Cons of RSB

- + scalable parallel $SpMV/SpMV-T$
- + scalable parallel $SpSV/SpSV-T$ (of course, less than $SpMV$!)
- + many other common operations (e.g.: parallel matrix build algorithm)
- + native support for symmetric matrices
- - a number of known cases (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.10.3-0) sparse matrix routines.

We report here results on an Intel Xeon 5670 and publicly available matrices.
Figure: Transposed/Non transposed SpMV performance on M4, versus MKL, 12 threads, large unsymmetric matrices.
Comparison to MKL, Symmetric SPMV

Figure: SpMV performance on M4, versus MKL, 12 threads, symmetric matrices.
Comparison to MKL, SPSV

Figure: Transposed/Non transposed SpSV performance on M4, versus MKL, single thread.

(MKL SpSV is not multithreaded)
In short

A shared memory parallel Sparse BLAS library implementation:

- on large matrices, better performance than Intel’s highly optimized, proprietary CSR implementation
- provides primitives for sparse solvers
- usable from within the open source PSBLAS library
- may be further tuned
library usage

Configuration & build:

./configure
make
make tests
make install
An example program: Hello RSB (C)

```c
#include <rsb.h>
int main(const int argc, char * const argv[]) {
  struct rsb_matrix_t *matrix=NULL;
  rsb_err_t errval=RSB_ERR_NO_ERROR;
  const rsb_coo_index_t m=3,k=3;
  const rsb_nnz_index_t nnz=3;
  double VA[]={11,22,33},X[m],B[k];
  const double one=1,zero=0;
  rsb_coo_index_t IA[]={ 0, 1, 2 },JA[]={ 0, 1, 2 };
  errval|=rsb_init(RSB_NULL_INIT_OPTIONS);
  matrix=rsb_allocate_rsb_sparse_matrix_const(VA,IA,JA,nnz,
                                             RSB_NUMERICAL_TYPE_DOUBLE,m,k,1,1,RSB_FLAG_NOFLAGS,NULL);
  errval|=rsb_spmv(RSB_TRANSPOSITION_N,&one,matrix,B,1,&zero,X,1);
  rsb_free_sparse_matrix(matrix);
  errval|=rsb_exit();
  if(errval==RSB_ERR_NO_ERROR) return 0; else return -1;
}
```
Linking example

cc -c `librsb-config --I_opts` hello.c
cc -o hello hello.o `librsb-config --static --libs --ldflags`
./hello
Octave + librsb = sparsersb.oct

Experimental interfacing, with a new Octave function, sparsersb

- librsb’s functionality and performance
- Octave’s ease of use
- a work in progress, but ready for testing
Installation tips:

- Tested with Octave 3.2.4 and 3.4.2
- Should compile librsb with -fPIC (any compiler)
- Should compile Octave with gfortran
- Plugin compilation:
  ./configure ; make ; make tests
alpha=1.0, beta=-1.0
IA=[ 1, 2, 1, 3] % matrix row indices
JA=[ 1, 2, 3, 3] % matrix column indices
VA=[11,22,13,33] % 4 coefficients for a 3 x 3 matrix
% Same matrix declaration syntax:
OM=sparse (IA,JA,VA,"summation") % Octave
RM=sparersb(IA,JA,VA,"summation") % RSB
X=[-1,-2,-3]’
% Same usage for matrix-vector product:
Y=[ 0, 0, 0]’; Y=alpha*(OM*X)+beta*Y
Y=[ 0, 0, 0]’; Y=alpha*RM*X+beta*Y
% And so on for many operations, like:
TM=transpose(RM) % transposition
RM*=alpha % scaling
% Some constructs are not yet allowed, like:
% RM(1,1)=111; MM=RM*RM;
# built under 3.4.2:

# for 64 bit systems:
octave --path \
/afs/ipp-garching.mpg.de/home/m/mima/Public/lib64/

# for 32 bit systems:
octave --path \
/afs/ipp-garching.mpg.de/home/m/mima/Public/lib32/

# now 'sparsersb' is a Octave function
Questions welcome!

Thanks for your attention.

Please consider testing librsb spotting last bugs/inefficiencies is essential for a quality public release!
#!/bin/sh
mkdir $HOME/local
tar xvzsf /afs/ipp-garching.mpg.de/home/m/mima/
/Public/src/librsb-0.0.1444M.tar.gz
cd librsb-0.0.1444M
./configure --disable-debug \ 
   --with-memhinfo=L2:4/64/512K,L1:8/64/24K \ 
   --prefix=/opt/librsb-debug/ \ 
   --with-matrix-types=double,double\ complex \ 
   CFLAGS=-O3\ -fPIC --prefix=$HOME/local
make
make tests
make install
#!/bin/sh
# look for updates on ftp://ftp.gnu.org/gnu/octave/
mkdir $HOME/local
tar xjf /afs/ipp-garching.mpg.de/home/m/mima/Public/src/octave-3.4.2.tar.bz2
cd octave-3.4.2
# Uncomment if building on Ubuntu 9 (with libpcre headers):
# ./configure --prefix=$HOME/local F77=gfortran
# Uncomment if building on HPC-FF:
# tar xjf /afs/rzg/home/m/mima/Public/src/pcre-8.13.tar.gz
# cd pcre-8.13 && ./configure --prefix=$HOME/local
# make && make install && cd -
# ./configure --prefix=$HOME/local \
#   CPPFLAGS=-I$HOME/local/include/ \
#   LDFLAGS=-L$HOME/local/lib --without-umfpack \
#   --without-amd \
#   --without-camd --without-ccolamd --without-colamd
make && make check && make install
sh autogen.sh
./configure MKOCTFILE=mkoctfile-3.4.2
#
# if not in $PATH, you can pass LIBRSB_CONFIG also..
#
# Tip: for the compilation of sparsersb, delete temporarily
# the line
# typedef OCTAVE_IDX_TYPE octave_idx_type
# in Octave’s config.h, which gets included and pollutes our
# namespace.
# You have to do this trick right after the configure.
# e.g.:
vim $HOME/local/include/octave-3.4.2/octave/config.h
make
# Then you can restore Octave’s config.h
make tests
# Enjoy!
Extra: benchmarking (only) MATLAB

# e.g.:
./matlabbench.sh matrix.mtx
Extra: benchmarking sparsersb versus Octave

# e.g.:
ex 
"L2:4/64/512K,L1:8/64/24K"
OMP_NUM_THREADS=1 ./octavebench.m matrix.mtx
OMP_NUM_THREADS=2 ./octavebench.m matrix.mtx
OMP_NUM_THREADS=4 ./octavebench.m matrix.mtx
OMP_NUM_THREADS=8 ./octavebench.m matrix.mtx