Replicated Computational Results (RCR) Report for "A distributed-memory package for dense Hierarchically Semi-Separable matrix computations using randomization"

Dominic Meiser, Tech-X Corporation, 5621 Arapahoe Avenue, Boulder CO, 80303, USA.

In this report we replicate a subset of the performance results in the paper "A distributed-memory package for dense Hierarchically Semi-Separable matrix computations using randomization".

1. INTRODUCTION

Rouet et al. present details of a newly developed library, STRUMPACK, for the construction of hierarchically semi-separable matrices and for arithmetic with these matrices [3]. Compression of dense matrices is achieved by means of a low rank approximation for sub blocks of the matrix. STRUMPACK provides functions to factor a hierarchical matrix into a form suitable for efficient solution and thus provides a means to create pre-conditioners for dense matrix problems. Among other things, STRUMPACK provides facilities to convert ordinary dense matrices into a hierarchical matrix format. All of this functionality is available in a distributed memory architecture based on MPI.

In this report we replicate a representative subset of the computational results presented by Rouet et al. We focus on compression, factorization, and solver performance reported in Table III of [3]. We also replicate the strong scaling results reported in Fig. 10 in that paper.

Specifically, we carry out the following steps:

(1) Download the STRUMPACK distribution package.
(2) Build STRUMPACK on a laptop.
(3) Build STRUMPACK on Edison at NERSC.
(4) Run STRUMPACK on Edison to replicate performance measurements for QChem Toeplitz matrix and to check strong scaling for a dense matrix from a boundary element formulation of an electromagnetic scattering problem.

The following sections provide details on these steps.

2. BUILDING STRUMPACK ON A LAPTOP

To verify that we could build the package, we first get it to run on a laptop. This allows us to iron out any issues ahead of time in a controlled environment before attempting to do the same on an expensive computer like Edison. Additionally, this provides confidence that STRUMPACK can be used on generic computers, and not just in the specific environment used by Rouet et al.

We obtain the sources from the STRUMPACK distribution website,

```
STRUMPACK_URL=http://portal.nersc.gov/project/sparse/strumpack \
  TARBALL_NAME=STRUMPACK-Dense-1.1.1.tar.gz \
  wget $STRUMPACK_URL/$TARBALL_NAME
tar xf STRUMPACK-Dense-1.1.1.tar.gz
```

The source distribution contains a 20 page manual with introduction, installation instructions, background on the algorithms, and an API reference. A readme file provides concise instructions on how to build STRUMPACK.

STRUMPACK's third party library requirements are rather minimal: MPI, BLAS, LAPACK, and ScaLAPACK. For our laptop builds We satisfy these requirements as follows:

— For MPI, we use mpich version 3.1.4.

— For BLAS and LAPACK, we use the binary distribution on the centos 6 linux distribution (rpm packages `blas-3.2.1-4.el6.x86_64`, `blas-devel-3.2.1-4.el6.x86_64`, `lapack-3.2.1-4.el6.x86_64`, and `lapack-devel-3.2.1-4.el6.x86_64`).
— For ScaLAPACK, we use version 2.0.2 from netlib [2].

All code is compiled with gcc version 4.9.3.

STRUMPACK's build system consists of gnu makefiles. The makefiles are customized by writing a Makefile.inc that defines system specific variables. Examples of Makefile.inc files are provided for gnu based linux systems, Edison, and Hopper. To build on our laptop we modify Makefile.gnu to point to our MPI, BLAS, LAPACK, and ScaLAPACK installation. With these modifications we successfully build the C++, C, and Fortran examples. We don't verify building of the matlab bindings because our laptop does not have matlab installed. We are able to execute all example binaries and the output from them looks reasonable.

3. BUILDING ON EDISON

The majority of the performance results reported in [3] were obtained on Edison [1] at NERSC. To build STRUMPACK on Edison, we use the default intel programming environment, `PrgEnv-intel/5.2.56` with version 15.0.1 20141023 of the intel compilers, and with module `cray-mpich/7.3.1` for MPI support. The template `Makefile.edison` provided with the source distribution of STRUMPACK works without modifications with these modules. The Makefile.edison file compiles all sources with `-O3`.

4. PERFORMANCE ON EDISON

We use the example `solve.cpp` to reproduce the results for the QChem Toeplitz matrix in Table III in [3]. Currently there is no mechanism to control the example program via command line options. Thus we modify the `solve.cpp` source code and rebuild for every set of parameters. We set the problem size to `n=80,000` and change the number of right hand sides to `nrhs=1`. Details of the computation are controlled via options of the computational object `sdp` of type `StrumpackDensePackage`. After discussion with Rouet et al., we configure `sdp` as follows to match the settings used in [3]:

```
sdp.use_HSS=true;
sdp.tol_HSS=1e-8;
sdp.levels_HSS=13;
sdp.min_rand_HSS=250;
sdp.lim_rand_HSS=5;
sdp.inc_rand_HSS=10;
sdp.max_rand_HSS=250;
sdp.steps_IR=10;
sdp.tol_IR=1e-8;
```

The setting `sdp.use_HSS=true` specifies that HSS compression be used. By setting this parameter to false it is possible to test the solution of the system via ScaLAPACK instead of STRUMPACK. The parameter `sdp.tol_HSS` sets ϵ in Table III in [3]. The parameter `sdp.levels_HSS` specifies the maximum number of levels in the HSS tree. The parameters `sdp.min_rand_HSS`, `sdp.lim_rand_HSS`, `sdp.inc_rand_HSS`, and `sdp.max_rand_HSS` configure the adaptive sampling of random vectors used for the construction of the HSS matrix in STRUMPACK. By setting minimum and maximum numbers of random vectors equal to one another, `sdp.min_rand_HSS==sdp.max_rand_HSS`, we effectively disable the adaptive sampling algorithm and use a fixed number of vectors, 250 in our example. The number of sampling vectors is adjusted for the different tolerances ϵ since the maximum rank depends on the precision of the reduced rank representation. The number of sampling vectors is indicated in table I for each ϵ. The parameter `sdp.steps_IR` determines the maximum

Table I. Summary of performance measurements. All numbers indicate times in seconds. The timings from [3] are indicated in parenthesis. The column labeled # Rand. Vecs indicates the number of sampling vectors used for the compression, Max. rank indicates the maximum rank for any sub-block estimated by STRUMPACK for the given level of accuracy, Compression lists the times consumed for HSS compression, Factorization the time for computing the ULV factorization of the compressed matrices, and Solution + IR is the time for solution and iterative refinement. All times are in seconds.

ϵ	# Rand. Vecs	Max. rank	Compression	Factorization	Solution + IR
1.0×10^{-8}	250	171 (169)	19.0 (19.0)	0.04 (0.04)	0.2 (1.5)
1.0×10^{-6}	200	147 (147)	18.2 (17.6)	0.07 (0.03)	0.7 (2.0)
1.0×10^{-4}	150	121 (120)	16.9 (16.7)	0.04 (0.02)	0.4 (5.6)

number of iterations used for iterative refinement and `sdp.tol_IR` specifies the precision to which the linear system is being solved. Reporting of performance numbers relies on the function `sdp.print_statistics()`.

Initially, we run the `solve` driver in a mode where the precision of the compressed matrix is verified. The verification of the compression is a very memory intensive operation and thus requires more compute nodes than were used in [3]. After successfully verifying the accuracy of the compression, we disable the verification feature and reduce the number of nodes to the configuration used in [3]. Specifically, we use four compute nodes with 16 MPI processes on each of them.

Our performance results are shown in table I along with the results reported in [3] for comparison. The timings we obtain for the compression step are very close to the results reported in [3]. The timings for factorization and solution vary a bit. But because these stages are only fractions of a second long they are much more susceptible to fluctuations. In our estimation the timing results are close enough to consider the results of [3] replicated.

Besides the timing results we also made sure that the maximum rank reported by STRUMPACK agrees with [3]. We find that in our runs the maximum rank is within 1% of the ranks reported in [3]. The maximum rank obtained depends on the random sampling vectors. As the sampling vectors are not identical from run to run it is not surprising that there are small variations.

We see much shorter times for solution and iterative refinement than what is reported by Rouet et al. This is because we are using much larger solution tolerances, `sdp.tol_IR=sdp.tol_HSS`. Rouet et al. used `sdp.tol_IR=1.0e-10`. The timing difference in the solution step can reasonably be explained by this difference in requested solution accuracy.

5. REPLICATION OF STRONG SCALING RESULTS

To replicate the strong scaling performance reported in Fig. 10 in [3] we use the program, `driver.cpp`, that was used by Rouet et al for that study. This executable is not part of the release distribution of [3]. We build it on Edison using

```
CC driver.cpp -I ${STRUMPACK_SRC_DIR}
```

The directory `STRUMPACK_SRC_DIR` is the directory in which the STRUMPACK sources are located. The driver program loads the matrices from Rouet's scratch directory on Edison. The matrices are stored in a proprietary binary format.

The timing results for the different stages of the solution process are shown in Fig. 1. For this run we use 6000 random vectors for the random sampling space and the depth of the HSS tree is six. The equations are solved to a precision of $\epsilon = 1.0 \times 10^{-4}$ We use the timing measurements as reported by `driver.cpp`. The compression, factorization, and matrix vector multiplication strong scale well to 2048 MPI process. The solution kernel does not scale well which is not surprising given the small amount of parallelism that can be exploited by this computation. However, this is usually only a relatively small fraction

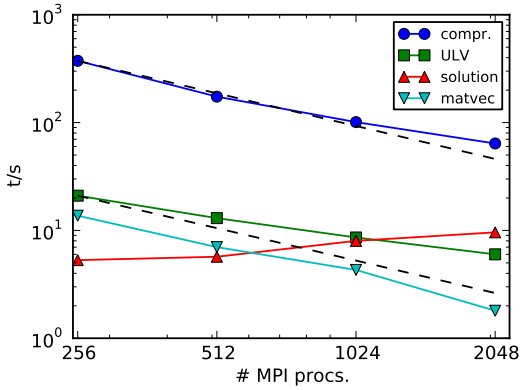

Fig. 1. Strong scaling of various stages of the solution of an electromagnetic boundary element problem with STRUMPACK. "compr." refers to the HSS compression, "ULV" to the factorization, and "solution" and "matvec" are statistics for the solution stage and matrix vector products. The dashed lines indicate ideal scaling.

of the total solution time. Overall we find that STRUMPACKis slightly faster in our runs than the timings reported in [3]. We find good qualitative agreement in the scaling behavior between our results and Rouet et al.

6. CONCLUSION

We replicate the results reported in [3]. The STRUMPACK library is easy to build and contains sufficient documentation for others to use the library. We build the library on a laptop and on Edison. We reproduce the timing results reported for a synthetic matrix as well as strong scaling results for a matrix originating from an electromagnetic boundary element problem.

7. ACKNOWLEDGEMENTS

We would like to thank Franọis-Henry Rouet and Xiaoye S. Li for valuable feedback and discussion and M. Heroux for guidance. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U. S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

Edison. http://www.nersc.gov/users/computational-systems/edison/.

netlib. http://www.netlib.org/.

F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov. A distributed-memory package for dense Hierarchically Semi-Separable matrix computations using randomization. *ACM Transaction on Mathematical Software*, ??:??, 2014.