A Splitting Approach for the Parallel Solution of Large Linear Systems on GPU Cards

[Extended Abstract]

Ang Li
Electrical and Computer Engineering
University of Wisconsin-Madison
1513 University Avenue
Madison, WI-53706
ali28@wisc.edu

Radu Serban
Mechanical Engineering
University of Wisconsin-Madison
1513 University Avenue
Madison, WI-53706
serban@wisc.edu

Dan Negrut
Mechanical Engineering
University of Wisconsin-Madison
1513 University Avenue
Madison, WI-53706
negrut@wisc.edu

ABSTRACT

We discuss a "split and parallelize" approach for solving linear systems $\mathbf{Ax} = \mathbf{b}$, sparse or dense banded, on a Graphics Processing Unit (GPU) card. The matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is possibly nonsymmetric, sparse, and moderately large; i.e., $10\,000 \leq N \leq 500\,000$. In a comparison against Intel's `MKL`, the `SaP::GPU` solver fares well when used to solve dense banded systems that are close to being diagonally dominant. We also compare `SaP` against three sparse direct solvers: `PARDISO`, `SuperLU`, and `MUMPS`, in terms of two metrics: robustness and time to solution. `SaP::GPU` is distributed as open source under a permissive BSD3 license.

Categories and Subject Descriptors

G.1.3 [**Numerical Linear Algebra**]: Linear systems (direct and iterative methods); Sparse, structured, and very large systems (direct and iterative methods).

General Terms

Algorithms.

`SaP` can be used to solve linear systems $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} = (a_{ij}), 1 \leq i, j \leq N$, is dense banded or sparse. In the former case, the half-bandwidth is denoted K and the matrix is assumed mildly diagonally dominant with coefficient d:

$$|a_{ii}| \geq d \sum_{j \neq i} |a_{ij}| \ , \forall i = 1, \ldots, N \ .$$

`SaP` pursues one of four solution paths denoted `SaP::GPU-`$\mathrm{D_d}$, `SaP::GPU-`$\mathrm{C_d}$, `SaP::GPU-`$\mathrm{D_s}$, and `SaP::GPU-`$\mathrm{C_s}$, see Fig. 1 [3]. When handling dense matrices, `SaP::GPU-`$\mathrm{D_d}$ splits the matrix $\mathbf{A}$ in P partitions $\mathbf{A}_i$ that are independently LU-factored. The coupling between sub-blocks is ignored. On the other hand, `SaP::GPU-`$\mathrm{C_d}$ captures this coupling, in a fashion inspired by the SPIKE approach [4]. When handling sparse matrices, the implementation first pre-processes the coefficient matrix $\mathbf{A}$ to transform it from sparse to dense banded with heavy diagonal entries. To this end, we employ two reorderings: a diagonal boosting, and a bandwidth reduction. After potentially dropping off entries, the resulting matrix is treated as dense banded and solved pursuing one of two paths: with and without coupling between the diagonal blocks $\mathbf{A}_i$. Note that none of the four strategies leads to a direct solver owing to approximations embedded in the methodology. Consequently, solving $\mathbf{Ax} = \mathbf{b}$ in `SaP` draws on a Krylov-subspace iterative solver that is preconditioned using the specific strategies associated with `SaP::GPU-`$\mathrm{D_d}$, `SaP::GPU-`$\mathrm{C_d}$, etc. The entire solution approach is GPU-parallel, with the exception of the matrix reorderings, which are implemented using a hybrid CPU-GPU approach [3].

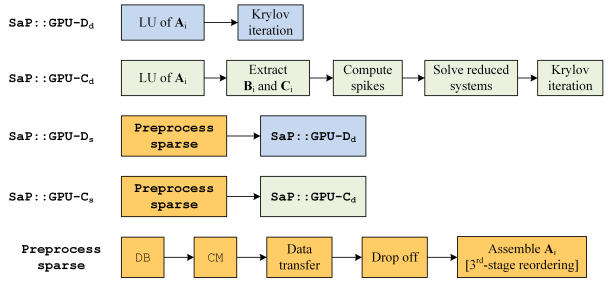

Figure 1: Computational flow for `SaP::GPU`.

Numerical Results, Dense Banded Linear Systems. We conducted two studies to demonstrate that `SaP` time to solution is virtually independent of P for $P > 30$ and of d when $d > 0.06$. A set 60 test were run using randomly generated dense banded matrices with $N \in \{1000, 5000, 10000, 20000, 50000, 100000, 200000, 500000, 1000000\}$ and $K \in \{10, 20, 50, 100, 200, 500\}$. The boxplot statistical results in Fig. 2 indicate that the median of the speedup of `SaP` over MKL is approximately two. In other words, of the tests run, half run in `SaP` at least two times faster than they run in MKL. The boxplot indicates that there are several outliers which run

as fast and eight times faster in SaP. Of 60 tests attempted, three failed due to "out of memory" circumstances, and two ran marginally faster in MKL [3].

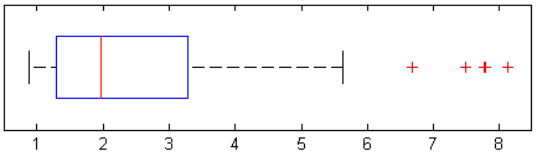

Figure 2: Statistical analysis of SaP speedup over Intel's MKL solution.

Numerical Results, Sparse Linear Systems. Figure 3 employs a median-quartile method to measure the statistical spread of the 114 matrices used to assess the performance of SaP relative to that of PARDISO [5], SuperLU [2], and MUMPS [1]. In terms of size, N is between 8192 and 4 690 002. In terms of nonzeros, nnz is between 41 746 and 46 522 475. The median for N is 71 328. The median for nnz is 1 167 967.

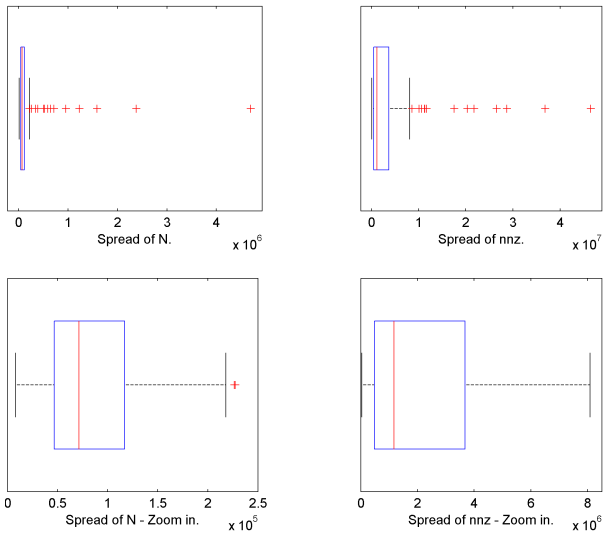

Figure 3: Statistical information regarding the dimension N and number of nonzeros nnz for the 114 coefficient matrices used to compare SaP::GPU, PARDISO, SuperLU, and MUMPS.

SaP::GPU failed to solve 28 linear systems. In 23 cases, SaP ran out of GPU global memory. In the remaining five cases, SaP::GPU failed to converge. The rest of the solvers failed as follows: PARDISO 40 times, SuperLU 22 times, and MUMPS 35 times. The Tesla K20X GPU card used had 6 GB of GDDR5-type memory. The other three solvers ran on a desktop with two 3GHz, 25 MB last level cache, Intel Xeon E5-2690v2 processors that tapped into 64 GB of system memory.

For the 114 linear systems considered there was a perfect negative correlation between speed and robustness. PARDISO was the fastest, followed by MUMPS, then SaP, and finally SuperLU. Of the 57 linear systems solved both by SaP and PARDISO, SaP was faster 20 times. Of the 71 linear systems solved both by SaP and SuperLU, SaP was faster 38 times. Of the 60 linear systems solved both by SaP and MUMPS, SaP was faster 27 times. Of the 60 linear systems solved both by PARDISO and SuperLU, PARDISO was faster 60 times. Of the 57 linear systems solved both by SaP and MUMPS, PARDISO was faster 57 times. And finally, of the 64 linear systems solved both by SuperLU and MUMPS, SuperLU was faster 24 times.

The four sparse solvers were also compared using a median-quartile method to measure statistical spread. Assume that $T_\alpha^{\texttt{SaP}}$ and $T_\alpha^{\texttt{PARDISO}}$ represent the times required by SaP::GPU and PARDISO, respectively, to finish test α. A relative speedup is computed like

$$\mathcal{S}_\alpha^{\texttt{SaP}-\texttt{PARDISO}} = \log_2 \frac{T_\alpha^{\texttt{PARDISO}}}{T_\alpha^{\texttt{SaP}}}.$$

with $\mathcal{S}_\alpha^{\texttt{SaP}-\texttt{MUMPS}}$ and $\mathcal{S}_\alpha^{\texttt{SaP}-\texttt{SuperLU}}$ similarly defined. These values are subsequently used to generate Fig. 4. The results suggest that when it finishes, PARDISO can be expected to be about two times faster than SaP. MUMPS is marginally faster than SaP, which on average can be expected to be slightly faster than SuperLU.

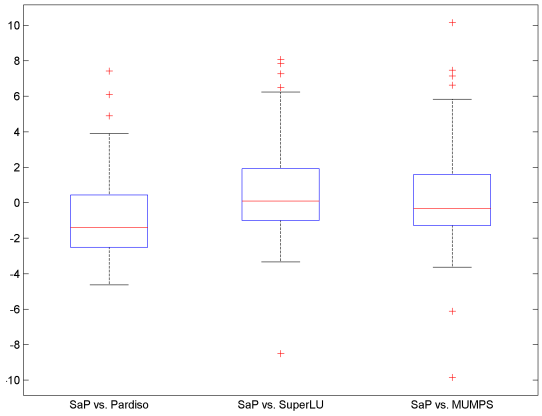

Figure 4: Statistical spread for SaP::GPU's performance relative to that of PARDISO, SuperLU, and MUMPS.

Finally, we compared SaP::GPU against the cuSOLVER GPU solver. Out of 114 tested systems, cuSOLVER successfully solved 45 and there were only three linear systems (ex11, ABACUS_shell_ud, and jan99jac120) which were successfully solved by cuSOLVER but not by SaP::GPU. Of the 42 systems solved by both SaP::GPU and cuSOLVER, cuSOLVER was faster only in five cases. In all 69 systems cuSOLVER failed to solve, the implementation ran out of memory.

1. REFERENCES

[1] MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. http://mumps.enseeiht.fr, 2015.

[2] J. W. Demmel. SuperLU Users' Guide. *Lawrence Berkeley National Laboratory*, 2011.

[3] A. Li, R. Serban, and D. Negrut. Analysis of a splitting approach for the parallel solution of linear systems on GPU cards – submitted. *SIAM Journal of Scientific Computing*, 00(0):0, 2015.

[4] E. Polizzi and A. Sameh. A parallel hybrid banded system solver: the SPIKE algorithm. *Parallel Computing*, 32(2):177–194, 2006.

[5] O. Schenk, K. Gartner, W. Fichtner, and A. Stricker. PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Generation Computer Systems*, 18(1):69–78, 2001.