Scaling the EIT Problem

Alistair Boyle, Andy Adler, Andrea Borsic

Abstract

There are a number of interesting problems that could be tackled if the computing capacity of current EIT systems can be improved. We examined the performance of two such systems and found that a noticeable portion of the compute time is spent in finding the solution of sparse matrices. We developed and used a new sparse matrix testbench, Meagre-Crowd, to evaluate a selection of these sparse matrix solvers and found that there are definite performance gains available.

I. INTRODUCTION

THE procedure used to solve inverse problems in areas such as Electrical Impedance Tomography (EIT) involves numerous steps to determine a solution. Software such as EIDORS [1] implement these algorithms using the numerical linear algebra tools in MatLab to manipulate sparse matrices that result from a Finite Element Method (FEM) formulation.

Computing power has increased according to Moore's law for 40 years, approximately doubling every 18 months [2]. Until recently, this has been achieved by increasing processor clock frequencies and a corresponding improvement in instructions-per-second throughput and memory bandwidth. Taking advantage of these improvements has involved little software cost. Purchasing a new processor gains one the advantages of the new hardware when the software is recompiled. In recent years, the benefits of increasing clock frequency have been limited by power consumption. Manufacturers have turned to System-on-Chip (SoC) with multiple processor cores to maintain Moore's Law for commodity processors. Simultaneously, the cost of computing power has dropped appreciably.

This change has fundamentally altered the computing landscape. To take advantage of the continuing growth in computing power, and thus, the ability to solve ever more complicated problems, software developers are now being forced to deal with issues previously limited to those developing High Performance Computing (HPC) applications. They are venturing into the world of symmetric multiprocessing (SMP) and distributed memory computing with the widely used OpenMP and MPI standards.

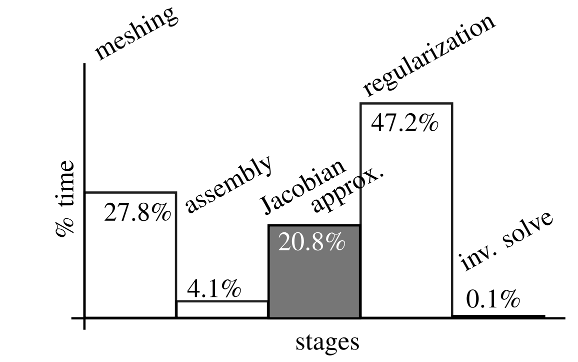
For EIT, increases in problem size, in terms of required computing power, come from a number of sources: the number of nodes and elements in a discretization of the domain, the dimensionality (handling of two- versus three-dimensional domains), processing of time series data, and correlating other modalities such as CT with EIT, amongst others.

There are many ways to increase the capacity of the EIT data processing system by taking advantage of SMP and distributed resources. A quick analysis of EIDORS's use of compute time for a sample problem (Figure 1a, Section II) showed that a significant amount of time was spent in the determination of the Jacobian matrix. This step primarily employed the MatLab numerical linear algebra solver. (Where the complexities are tucked away behind the backslash $A\backslash b$ operator.) This is a relatively easy component to swap out since the solver is treated as a black box and there are a number of alternatives available.

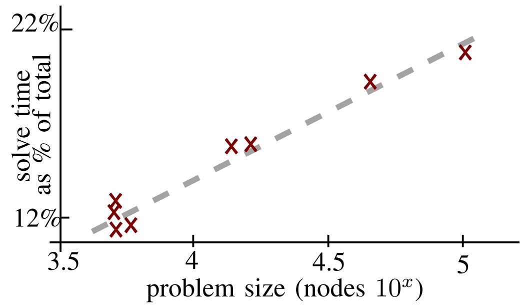
In this work, we examine the performance of EIDORS as the size of EIT problem scales and show preliminary performance results for a selection of alternative sparse matrix solvers using an independent testbench we have developed, Meagre-Crowd.

A. Boyle and A. Adler are with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON e-mail: boyle@sce.carleton.ca

A. Borsic is with the Thayer School of Engineering at Dartmouth, Hanover, NH



(a) Profiling EIDORS; a 101421 node 3D difference EIT problem



(b) Ratio of Jacobian approximation to total time as node density increased

Fig. 1. Performance of EIDORS for a 3D difference EIT problem; 8 core, 64GB, 2.66GHz Intel Xeon X5550

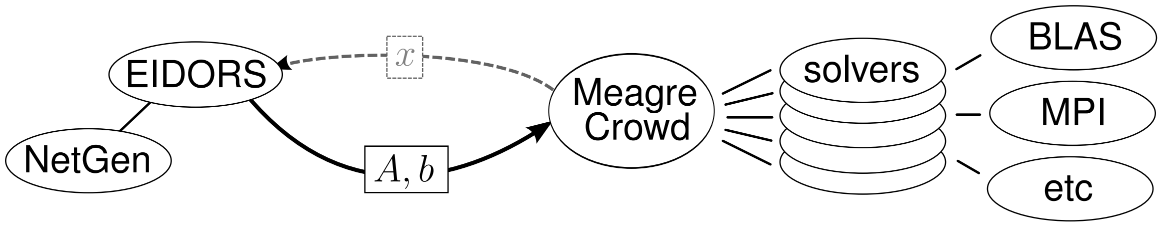


Fig. 2. Test bench dependencies: EIDORS, NetGen, Meagre-Crowd and assorted solvers and their dependencies (BLAS, MPI, etc.)

II. Performance Profiling EIDORS

EIDORS is a testbed for algorithm development running in MatLab, so it is certainly not optimized for maximum performance. None the less, it is representative of the computational requirements of the EIT inverse problem. The procedure for finding a linearized solution can be roughly divided into five stages: mesh generation, system matrix assembly, Jacobian approximation, inverse regularization, and finding the inverse solution. In the mesh generation phase, a domain is divided into, for example, triangles or tetrahedra, for use in the FEM formulation. This mesh is then assembled into a forward system matrix that contains a system of equations describing both the properties of the domain and the boundary conditions. A Jacobian is approximated using this forward problem by adjusting conductivities and estimating their effect on the measurements. The Jacobian is regularized (including calculating a dense matrix inverse) to form an inverse problem that is tractable. Finally, the inverse problem is solved by multiplying with difference measurement data to determine a conductivity distribution that could have resulted in the measurements. Finding the Jacobian involves numerous solutions using the sparse forward system matrices and therefore, requires an efficient sparse solver to achieve reasonable performance as the problem size scales.

A difference EIT problem was selected for profiling. (3D, 3 planes of 15 electrodes, 101421 nodes, 548299 elements, single-step difference EIT, EIDORS 3.4, NetGen 4.9.11, MatLab 7.11.0.584 (R2010b containing UMFPACK 5.4.0), 8 core Intel Xeon X5550 @ 2.67GHz SMP system, 64GB memory.) It was found that meshing took 169.12 seconds, model assembly took 24.95 seconds, the Jacobian approximation took 126.324 seconds, regularization took 287.272 seconds, and the final solution was determined in 0.82 seconds (Figure 1a). The ratio of time spent in the Jacobian approximation stage compared to the total working time to obtain a solution was found to grow logarithmically with the node density (Figure 1b).

Similar initial profiling results were obtained using the New Dartmouth Reconstructor Matlab (NDRM) EIT code [3], solving absolute EIT reconstructions with a non-linear Tikhonov regularized Gauss-Newton solver.

III. Alternative Solvers

The requirement for a sparse solver used in general EIT problems is that for real-valued system matrices, the solver can handle symmetric matrices, while for complex-valued system matrices (generally, not symmetric or Hermitian), the solver must be able to handle an unsymmetric system. Since the EIT problem can be structured as a square system matrix, solvers capable of QR decomposition are not required. In general, symmetric solvers perform in nearly the same manner as unsymmetric solvers (using similar algorithms) but must handle twice as many values. Therefore, symmetric solvers can be expected to perform twice as fast as unsymmetric solvers. For this reason, we chose to focus on unsymmetric solvers as a more general case. Similarly, we focused on real valued solvers though, in principle, these results can be extended to complex systems because the algorithms are the same with twice the numeric throughput.

Meagre-Crowd 0.4.5 [4] was used to test the performance of the sparse matrix solvers: UMFPACK 5.5.0 [5], MUMPS 4.9.2 [6], WSMP 11.01.19 [7], Pardiso 4.1.1 [8], TAUCS 2.2 [9], SuperLU_DIST 2.5 [10] and CHOLMOD 1.7.1 [11]. We developed Meagre-Crowd as a new open source project that integrates sparse solvers in a common framework to benchmark sparse linear algebra performance. Code was released under the GPL. The benchmark used a shared set of dependencies so that various versions of MPI, BLAS and LAPACK (optimized to varying degrees) did not affect the results. It should be emphasized that choosing an implementation of MPI and BLAS optimized for the current platform does have a significant effect on the run-time but does not affect inter-solver comparisons within the Meagre-Crowd testbench.

A set of test matrices were generated using EIDORS 3.4 as described in the previous section. These matrices were 5041 to 101465 on a side containing 35301 to 769590 non-zero entries so that there were 5041 to 101465 unknowns solved 45 times (45 orthogonol stimulation patterns). The maximum size of the generated matrices was limited by MatLab's memory usage (64GB). Matching right-hand sides were generated to give 45 orthogonal solutions for use in the Jacobian approximation. In addition, the expected solution was stored and compared at the conclusion of each testbench run as a check of solution validity. All problems were real-valued and consequently symmetric, but the (full) unsymmetric sparse matrices were used in these tests. (CHOLMOD was a symmetric-only solver used as a comparison versus the unsymmetric solvers.) These matrices were loaded into the Meagre-Crowd testbench running on an Intel Core2 Duo T9550 at 2.66GHz with 3GB of memory. Memory usage for most solvers remained below 1GB with the exception of UMFPACK on the largest problem at 2.1GB, indicating that memory capacity was not a factor in the sparse solvers' ability to find a solution.

Solution time was defined as the time to proceed through the sparse solver phases: re-ordering, symbolic and numeric factorization, substitution and iterative refinement. The two solvers capable of multicore operation on unsymmetric matrices (WSMP and MUMPS) were compared against the field of solvers in Figure 3. The figure shows solver performance benchmarked against UMFPACK's so that, for example, at 100,000 nodes CHOLMOD was nearly seven times faster than UMFPACK.

IV. Observations

Absolute solver performance in terms of time-to-solution is directly related to the volume of data that is being handled. Matrices found in typical FEM formulations are stored in a sparse format that does not record the zero entries in the matrix. The number of non-zeros in the sparse matrix structure directly determine the quantity of data that must be moved over the network, in memory, and through caches. During the solution process, the sparsity and specific structure of the matrix determines the degree of fill-in, the expansion in quantity of data to be stored as a solution is derived. Solution times for UMFPACK ranged from 563ms to 7 minutes and 41 seconds; though, solution times on the smallest problems varied by a factor of two or more.

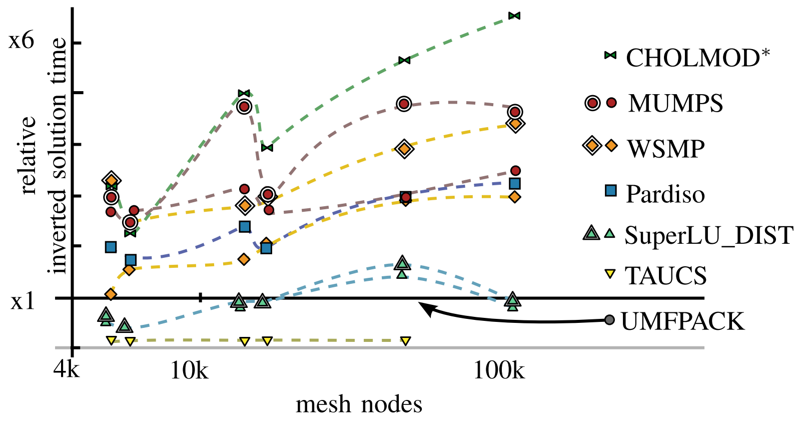


Fig. 3. Solver Scaling: Mesh Size and Compute Nodes, solver performance for a range of mesh sizes. (Inverted solution time was measured relative to UMFPACK, where $N = t_{\text{UMFPACK}}/t_{\text{solver}}$ so that a faster solver was plotted as N times faster than UMFPACK. For WSMP and MUMPS, results for two-cores have a double-symbol. *Note that CHOLMOD is a symmetric sparse matrix solver while the others are handling unsymmetric matrices.) [Preliminary results.]

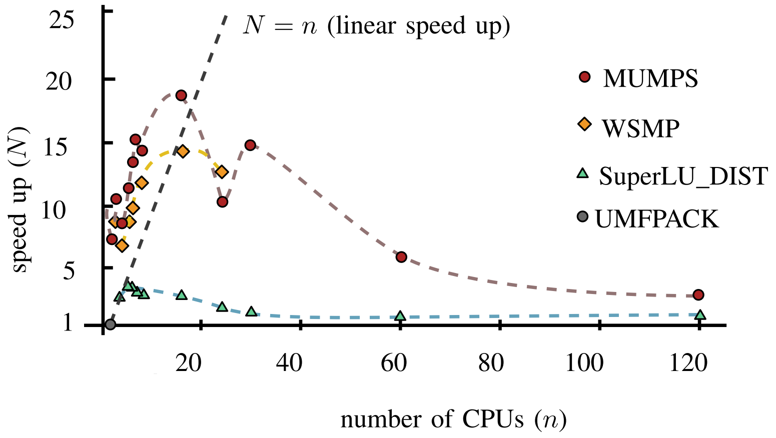


Fig. 4. Solver Scaling: Compute Nodes, solver performance for a mesh of 45289 nodes. (Inverted solution time was measured relative to UMFPACK, where $N = t_{\text{UMFPACK}}/t_{\text{solver}}$ so that a faster solver was plotted as N times faster than UMFPACK. [Preliminary results.]

In Figure 3, we observed a noticeable performance benefit for large matrices with both MUMPS and WSMP when compared to UMFPACK, particularly when multicore capabilities were used. CHOLMOD, the symmetric solver, showed the benefits of a reduced quantity of data. As problem size was increased, CHOLMOD increased its performance over the other solvers. The solver TAUCS, an Out-of-core (OOC) solver, was very slow due to the overhead of disk access but never exceeded more than 100MB of memory for the largest problem size. Presumably, the largest problems could be handled by a solver such as TAUCS that's performance will not be limited by memory capacity.

V. Discussion

Observations regarding the performance of EIDORS (and consequently, MatLab) as the size of EIT problem scales show the limitations of the system as memory consumption and solution time grow to the capacity of the available resources. Using an independent testbench, Meagre-Crowd, we show preliminary performance results for a selection of alternative sparse matrix solvers. Solvers examined in the testbench were not optimized, but with preliminary default settings, respectable improvements are possible. Performance that grows with problem size is possible in the sparse solver portion of the EIT solution process.

Other opportunities for higher-level parallelism have not been explored here: pipe-lining of tasks and splitting of tasks such as the Jacobian calculations. Combining these higher-level algorithmic changes with the latest in sparse solvers could lead to appreciable improvements in the capacity of current EIT systems.

Acknowledgements

This work was supported by a grant from NSERC Canada.

References

[1] A. Adler and W. R. B. Lionheart, "Uses and abuses of EIDORS: An extensible software base for EIT," *Physiol. Meas.*, vol. 27, no. 5, pp. S25–S42, May 2006.

[2] R. Schaller, "Moore's law: past, present and future," *IEEE Spectrum*, vol. 34, no. 6, pp. 52–59, Jun. 1997.

[3] A. Borsic, A. Hartov, K. Paulsen, and P. Manwaring, "3d electric impedance tomography reconstruction on multi-core computing platforms," *Proceedings IEEE EMBC'08, Vancouver*, Aug. 2008.

[4] A. Boyle, "Meagre-crowd: A sparse solver testbench," Mar. 2011. [Online]. Available: https://github.com/boyle/meagre-crowd

[5] T. Davis, "Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software*, vol. 30, no. 2, pp. 196–199, 2004.

[6] P. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Parallel Computing*, vol. 32, no. 2, pp. 136–156, 2006.

[7] A. Gupta, G. Karypis, and V. Kumar, "A highly scalable parallel algorithm for sparse matrix factorization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 5, pp. 502–520, May 1997.

[8] O. Schenk and K. G "Solving unsymmetric sparse systems of linear equations with pardiso."

[9] S. Toledo, D. Chen, and V. Rotkin, "Taucs: A library of sparse linear solvers," vol. 2.2, 2003. [Online]. Available: http://www.tau.ac.il/~stoledo/taucs/

[10] X. S. Li and J. Demmel, "Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Transactions on Mathematical Software*, vol. 29, no. 2, pp. 110–140, 2003.

[11] Y. Chen, T. Davis, W. Hager, and S. Rajamanickam, "Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate," *ACM Trans. Math. Software*, vol. 35, no. 3, Oct. 2008.