Addressing the Computational Cost of Large EIT Solutions

Alistair Boyle[1], Andrea Borsic[2] and Andy Adler[1]

[1]: Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada
[2]: Thayer School of Engineering, Dartmouth College, Hanover, NH, U.S.A.

E-mail: boyle@sce.carleton.ca, andrea.borsic@dartmouth.edu, adler@sce.carleton.ca

Abstract. Electrical Impedance Tomography (EIT) is a soft field tomography modality based on the application of electric current to a body and measurement of voltages through electrodes at the boundary. The interior conductivity is reconstructed on a discrete representation of the domain using a FEM mesh and a parametrization of that domain. The reconstruction requires a sequence of numerically intensive calculations. There is strong interest in reducing the cost of these calculations.

An improvement in the compute time for current problems would encourage further exploration of computationally challenging problems such as the incorporation of time series data, wide-spread adoption of three-dimensional simulations, and correlation of other modalities such as CT and ultrasound. Multicore processors offer an opportunity to reduce EIT computation times but may require some restructuring of the underlying algorithms to maximize the use of available resources.

This work profiles two EIT software packages (EIDORS and NDRM) to experimentally determine where the computational costs arise in EIT as problems scale. Sparse matrix solvers, a key component for the FEM forward problem and sensitivity estimates in the inverse problem, are shown to take a considerable portion of the total compute time in these packages. A sparse matrix solver performance measurement tool, Meagre-Crowd, is developed to interface with a variety of solvers and compare their performance over a range of two- and three-dimensional problems of increasing node density. Results show that distributed sparse matrix solvers that operate on multiple cores are advantageous up to a limit that increases as the node density increases. We recommend a selection procedure to find a solver and hardware arrangement matched to the problem and provide guidance and tools to perform that selection.

Keywords: Electrical Impedance Tomography, Finite Element Method, Distributed Computing, Sparse Linear Algebra Submitted to: *Physiol. Meas.*

1. Introduction

Electrical Impedance Tomography (EIT) is a soft field imaging modality that finds applications in process monitoring, in medical diagnosis and in geotechnical surveying. The technique is based on the application of electric currents to the object under investigation through a number of electrodes located on the boundary. Voltages

resulting at the electrodes from applied currents are measured and are used to estimate the electrical conductivity distribution within the object. EIT conductivity reconstruction can be formulated as a discretized forward problem using the Finite Element Method (FEM). The corresponding inverse problem of estimating the conductivity is determined by fitting a parametrized model to the data. The inverse problem is ill-posed in the sense that small changes in the measurements can correspond to drastically different conductivity distributions. This ill-posed-ness is dealt with by applying regularization to find a numerically stable solution. Finding an acceptable conductivity solution poses computational challenges as the FEM node density increases and the corresponding number of numeric calculations grows. The computational challenges can be addressed by applying restrictive assumptions on the reconstructions or by applying greater computing resources.

Since the 1960s, many numeric codes have required nothing more than recompilation to take advantage of updates to commodity processors. Improvements in instructions-per-second performance and memory bandwidth have been due to increases in processor frequency and architectural refinements. In more recent times (c.2005), a major shift in the design of commodity computing resources has occurred which has been driven by the growing power consumption of high frequency single-core processors. The move toward multicore processors is a means to maintain Moore's Law while limiting the exponential growth of power consumption (Parkhurst et al., 2006). (Where Moore's Law over the last 40 years has come to refer to the doubling of computing power every 18 months (Schaller, 1997).) The application of the familiar EIT tools requires some degree of reconsideration with the growing availability of commodity multicore processors.

In this paper, we are motivated by the desire to work with larger EIT problems. An example is the move toward three-dimensional reconstructions, rather than two-dimensional reconstructions common at this time. Three-dimensional EIT is desirable because electric current flows throughout the domain unlike similar tomographic modalities such as x-ray Computed Tomography (CT) where the x-rays are relatively confined to a two-dimensional plane. Meshing the three-dimensional domain results in a significantly larger number of FEM mesh nodes compared to the two-dimensional domain when the final conductivity image is to retain the same resolution. Additional computational demands also result from working with complex valued data, time series data, and correlating other modalities (CT, ultrasound) with EIT.

To tackle these larger EIT problems we would like to make use of distributed computational techniques. The conversion of once serialized operations to a parallel multiprocessing environment can be a complicated task that has an upper bound on possible efficiency gains set by Amdahl's Law (Amdahl, 1967).

$$\text{Speed-up}_A = \frac{1}{f + \frac{1-f}{N}} \tag{1}$$

where f is the fraction of a program that cannot be made parallel and N is the number of processors available. Amdahl's arguments, though somewhat simplistic (Suna and Gustafsona, 1991; Hill and Marty, 2008), are a fundamental reason why much of the design effort for processor hardware has focused on single processor performance. Amdahl's Law suggests why disappointing performance can result after considerable work to make portions of a code run in parallel. Gustafson's Law has been suggested as a more appropriate model that claims the parallel portion of problems will scale to

the available computing resources giving (Gustafson, 1988)

$$\text{Speed-up}_G = f + N(1 - f) \tag{2}$$

In EIT, while there are serialization points in the solution, there are no significant portions of the problem that fundamentally require serial processing. The intricacy of both the techniques and problems that have been tackled has grown as more capable compute resources have become available. The broad availability of multicore processors should, thus, motivate development of the tools to make use of these new resources.

The communication cost between the computational units is a critical design consideration for these parallel algorithms that is not considered by Amdahl's Law. In multicore processors, a typical architecture shares a common memory and coherent caching infrastructure between processing cores. This is in contrast to a distributed memory architecture where each unit has independent memory and shared data must be communicated between the units. Most commodity multicore processors today are based on a Symmetric Multi-Processing (SMP) architecture using shared memory and identical cores which limits the communication cost and complexity of an initial transition to parallel code.

In this paper, we address the question of 1) where computational costs exist in the numerical solution of EIT problems, and 2) specifically look at the behaviour of a variety of sparse matrix solvers when applied to EIT problems. We examine the methods used for solving the EIT problem using a FEM and show that the numeric aspects of more finely meshed problems drive a requirement for distributed computational methods. We investigate specific remedies in the form of distributed sparse solvers to understand their usefulness and limitations in the context of EIT.

2. Overview

This work is divided into two portions. The first constructs a view of the computational costs by building a profile of the performance of the EIT problem solving toolkit. The second explores the specific choices available for improving FEM simulation speed and introduces software designed to compare these codes.

2.1. Profiling

Two EIT codes were available for performance profiling: EIDORS and the New Dartmouth Reconstructor MATLAB (NDRM). EIDORS is principally written and executed within the MATLAB environment (Adler and Lionheart, 2006). It implements a FEM solution to the EIT problem with numerous choices for solution approaches. EIDORS can use a number of mesh generators including distmesh and NETGEN. NDRM is principally written in MATLAB as well (Borsic et al., 2008). It implements a FEM solution to the EIT problem using a Gauss-Newton iterative solver with C-language .mex files used for a few computationally intensive routines.

In both of these codes, the general procedure for solving the problem is to 1) obtain measurements according to a schedule of drive and measurement patterns, 2) generate meshes, 3) estimate sensitivities to build a Jacobian, and 4) use the Jacobian in combination with some regularization scheme to solve the inverse problem according to some minimization criteria. Iterative solutions return to step 3) to estimate a new Jacobian using updated conductivities. Methods that refine the mesh depending on the solution would return to step 2) to generate a mesh update. (Figure 1)

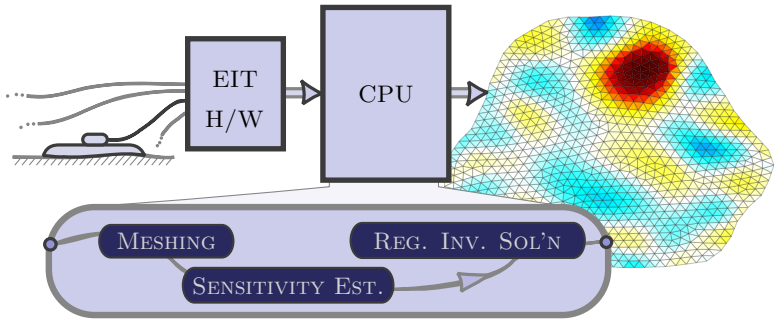

Figure 1: A generic EIT system; The object under investigation has electrodes attached. Measurements are obtained from an EIT system through these electrodes. On the processor 1) meshes are generated, 2) the sensitivity matrix is estimated, and 3) a regularized inverse solution is determined. This result is visualized (in this case) as a 2D slice.

Measurements are obtained from an EIT system and frequently connected to a general purpose computer to perform the heavier numeric computations. The time cost of obtaining these measurements is directly related to the stimulation and measurement protocol and communication throughput of the EIT equipment. For "real" measurements, this time can be considered a fixed constant that is instrumentation dependent and not explored further in this work. When performing simulations, rather than using a physical system to obtain measurements, the measurements are constructed by using a forward simulation of the EIT problem. This can be achieved with a FEM using the desired conductivity distribution. The stimulation and measurement protocol are then applied to the FEM to determine the measurements.

Mesh generation for arbitrary domains is a challenging computational problem but has received considerable attention due to its applicability to most FEM related problem spaces. In general, algorithms exist for meshing two- and three-dimensional domains in $O(n \log n)$ time with $O(n)$ space for an n element mesh. In two dimensions, these algorithms are the Delauney triangulation via plane sweep and divide-and-conquer (Fortune, 1987; Guibas and Stolfi, 1985). For three-dimensional domains, octree, advancing front, and Delauney techniques are available (Shephard and Georges, 1991; Löhner and Parikh, 1988; Watson, 1981). Additional speedups are possible in multiprocessing environments through domain decomposition. The qualities of the mesh can have a direct effect on the quality of the FEM calculations since the FEM provides guarantees only as to the globally correct approximation which leaves room for local errors (Strang and Fix, 1987). Choosing a regular structure such as a two-dimensional circle or a three-dimensional box make the meshing relatively cheap to generate, but this is not a practical option for most cases because, in EIT, it has been shown that a correct boundary shape when comparing the forward and inverse problems can be critical to getting reasonable results (Grychtol et al., 2011). A uniform mesh is also not appropriate because a higher localized mesh density is desirable in regions, such as near electrodes, where stronger electric fields exist. If the shape is treated as constant, the mesh generation cost can be treated as a one-time expense. In practical applications, this can become infeasible when the boundary is matched

to specific patients through, for example, using a segmented CT or MRI to get an approximately correct patient-specific boundary (Tizzard et al., 2005). Other scenarios where mesh generation should be counted in the cost of an EIT simulation are when adaptive mesh refinement is required or in algorithmic development where the mesh is frequently adjusted as the problem is built up.

The computation of the sensitivity matrix (the Jacobian) requires numerous calculations of the forward problem. The most straightforward and intuitive approach is the perturbation method where each element of conductivity in the mesh to be used for the inverse problem's solution is perturbed in turn and the effect upon the measurements is recorded. The matrix of these perturbation experiments forms the Jacobian matrix. More efficient algorithms to estimate the Jacobian exist such as the self-adjoint method (Breckon, 1990; Polydorides and Lionheart, 2002). Another such trick is to remember that the Jacobian only considers the effect at the electrodes, so a significant reduction in computation time can be achieved by applying a QR decomposition to the set of all unit stimulations forming the right-hand side. Applying this $\mathbf{Q}$ and particularly the new $\mathbf{R}$, results in a system that only solves for the necessary node voltages on the boundary

$$\mathbf{AX} = \mathbf{B} = \mathbf{QR} \tag{3}$$

$$\mathbf{X} = (\mathbf{A}^{-1}\mathbf{Q})\mathbf{R} \tag{4}$$

where if the columns of $\mathbf{Q}$ that will eventually be zeroed by the lower part of $\mathbf{R}$ are dropped, the required computations can be significantly reduced.

The EIT forward problem forms a sparse matrix that can be solved through either direct or iterative methods. A direct solution is possible when solving a linearization of the EIT problem. In the direct solution, the factorization formed by analyzing and manipulating the FEM matrix can be reused for many different right-hand sides. Small perturbations require recalculation of a portion of the factorization. Iterative methods exchange some numerical accuracy for speed. When sparse matrix problems become large, iterative methods are often the only option. Perturbations used in an iterative solution can use a prior solution to converge quickly on the new solution but cannot calculate multiple right-hand sides efficiently. In either case, it is possible to perform these calculations using multiple processors by distributing the work.

Once determined, the Jacobian is used in combination with some regularization scheme. The Jacobian itself is a dense matrix, and therefore, the result of the application of a regularization scheme will result in a new dense matrix which in turn can be solved given a set of measurements. There are two regularization implementations available: the Tikhonov form and the Wiener filter form.

$$\Delta\sigma = (\mathbf{J}^\intercal\mathbf{W}\mathbf{J} + \lambda^2\mathbf{R})^{-1}\mathbf{J}^\intercal\mathbf{W}\Delta\mathbf{v} \qquad \text{Tikhonov} \tag{5}$$

$$\Delta\sigma = \mathbf{R}^{-1}\mathbf{J}^\intercal(\mathbf{J}\mathbf{R}^{-1}\mathbf{J}^\intercal + \lambda^2\mathbf{W}^{-1})^{-1}\Delta\mathbf{v} \qquad \text{Wiener filter} \tag{6}$$

where $\mathbf{J}$ is the Jacobian, $\mathbf{R}$ is some regularization matrix, and $\mathbf{W}$ is a measurement noise covariance term for measurement update $\Delta\mathbf{v}^{(k)} = \mathbf{v}_{meas} - \mathbf{v}_{est}^{(k)}$ and conductivity update $\Delta\sigma^{(k)}$. The Tikhonov form scales with the size of the parametrized model (the coarse model) while the Wiener filter form scales with the number of measurements. The choice of implementation can have a significant impact on the computational cost of these dense calculations. The Wiener filter form is used here. The dense matrix calculations can be accelerated with multiprocessing or through the use of repurposed graphics processor offload engines (Purcell et al., 2002).

To compare these three later computational components of EIT (meshing, sensitivity and regularized solution calculations), simulations were run using increasingly more detailed meshes with the two EIT codes for both two- and three-dimensional problems. In two dimensions, triangular elements were used so that a n node mesh resulted in T elements

$$T = n_b + 2n_i + 2h - 2 \qquad \text{2D triangular} \tag{7}$$

with n_b boundary nodes, n_i internal nodes and h internal holes contained in the mesh. In three dimensions, there is no direct relation between nodes and elements. Tetrahedral elements were used so that a n node mesh resulted in T elements

$$T = \frac{1}{3}e_b + e_i - n_i + H - h - 1 \qquad \text{3D tetrahedral} \tag{8}$$

with e_b boundary edges, e_i internal edges and H through holes (Ewing et al., 1970). In general for meshes without holes and a large number of elements, these can be approximated as $T \simeq 2n$ for two dimensions and $T \simeq 6n$ for three dimensions. Therefore, three-dimensional problems tend to have many more non-zeros in their FEM system matrices and require more computational "horsepower" as a result.

For each run, two meshes are generated: a fine mesh for forward calculations and a coarser mesh for inverse calculations. The fine mesh is used to achieve numerical accuracies that correspond with instrumentation noise levels. The coarser mesh is used for the inverse portion of the problem because, as the mesh elements become smaller, their effect on the measurements is reduced. This results in an increased condition number for the Jacobian matrices and requires greater regularization which cancels much of the potential resolution benefit of the smaller elements. A geometric mapping between the two takes advantage of the benefits of both.

2.2. Sparse Solvers

Meshing and the dense matrix calculations were not explored in further detail for this work as we chose to focus on the problem of improving the sensitivity and forward problem calculation times. The forward solution is required in simulation of stimulation and measurement patterns, as well as the sensitivity estimation for the inverse problem. These require numerous sparse matrix calculations on the FEM system.

Sparse solvers take advantage of the fact that a matrix such as the FEM matrix largely contain zeros. Treating these matrices as dense incurs a significant computational cost as the node density increases. The dense matrix calculation rate becomes limited by memory bandwidth in transferring data that is largely known to be zero or very nearly zero. An alternate approach, where only the non-zero matrix elements are stored, is appropriate. Due to the requirements for accurate, and therefore, irregular boundaries, the FEM matrices used in EIT tend to take on a sparse but not precisely uniform structure. This structure requires a level of indirection to store efficiently. A first step is storage as coordinate pair and value (a format commonly called COO), but a further optimization orders the data so that one coordinate of the coordinate pair is implicit in the storage format and thus, "compressed." This storage scheme is called the Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC) format depending on which coordinate has been compressed. If MATLAB integration is a goal, careful consideration of sparse solver storage formats should be undertaken. The conversion between formats was not included in the performance

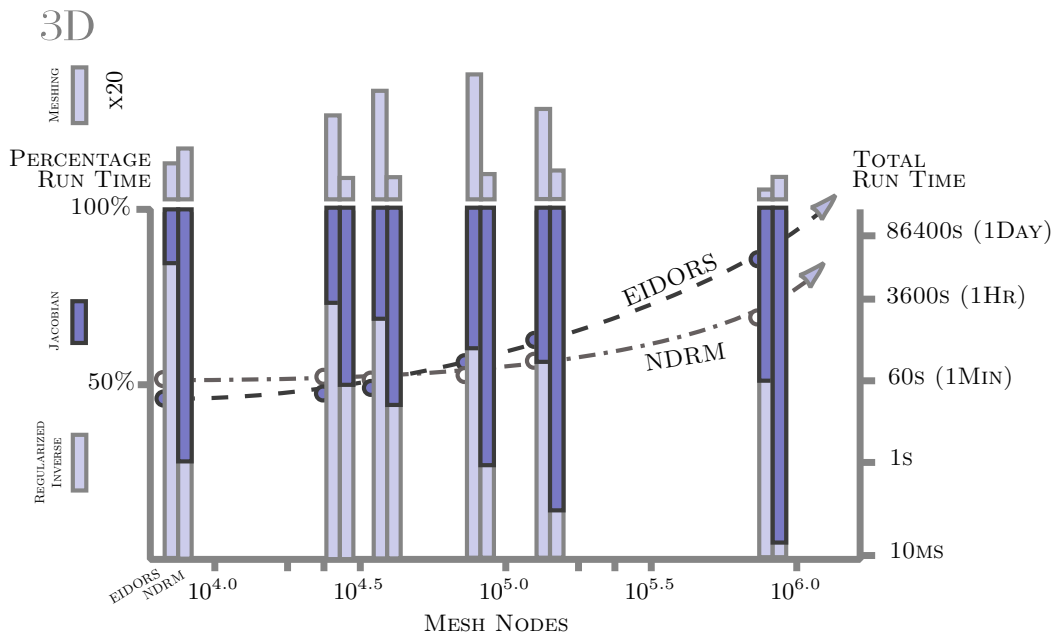

Figure 2: Profiling EIDORS and NDRM for 3D; Total run time as node density increased and shows percentage of run time for sensitivity estimation (Jacobian), and regularizing the inverse and solving. Meshing time is shown as a multiple of the two (scaled down by a factor of twenty). For each node density, EIDORS is shown on the left and NDRM on the right. Total run time is shown against the right-hand side logarithmic scale while percentage run time is shown against the left-hand side linear scale. The percentage of run time devoted to sensitivity estimation as well as total run time increased with node density.

results presented in this work but could be a significant performance factor for a solver integrated with MATLAB via a `.mex` interface. Further storage efficiencies can be achieved by recognizing symmetric and Hermitian conditions on the matrix to reduce storage requirements by half. In general, a symmetric matrix can be solved directly with a Cholesky decomposition while an unsymmetric matrix can be solved with an LU decomposition. A QR decomposition is not necessary because the matrices are square. For an n-by-n sparse matrix, the LU decomposition generally requires approximately twice as much computational time as the Cholesky decomposition due to the larger number of non-zero entries in the matrix. For EIT, a strictly real valued conductivity problem is symmetric, but complex valued problems are not Hermitian (but are symmetric) and cannot typically take advantage of sparse symmetric solvers (Bunch and Kaufman, 1977). The complex valued problem requires roughly twice the storage and four times the number of multiplications when compared to a real valued problem using the same decomposition due strictly to the handling of the complex numbers.

For direct solvers, if the matrices were dense, the next step would be to factorize the matrix and then solve with forward and backward substitution to arrive at a

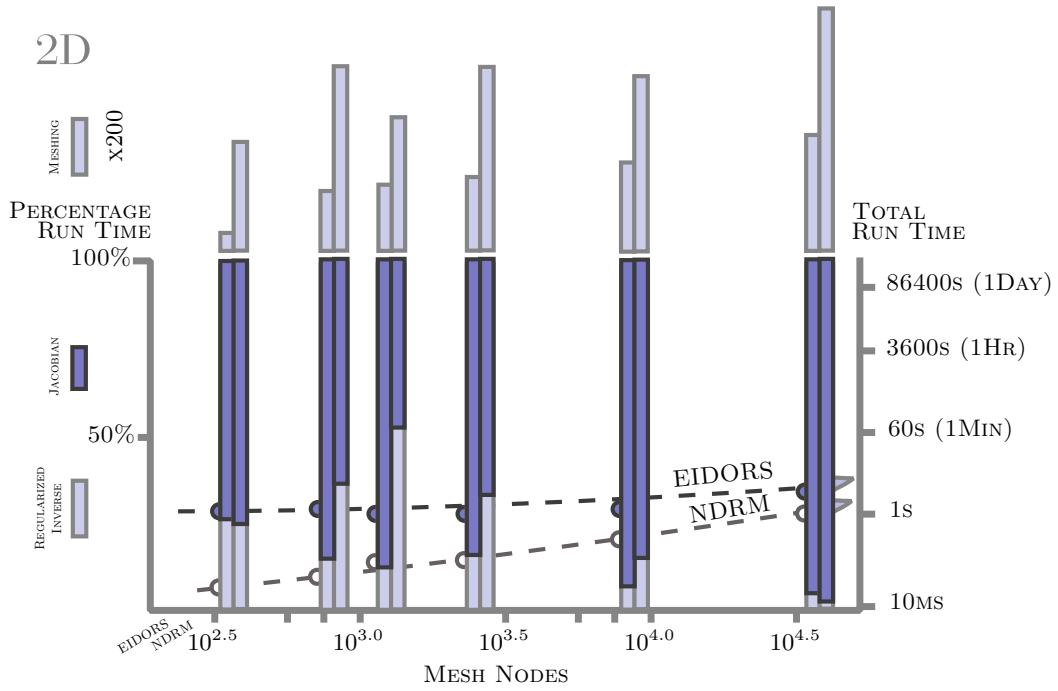

Figure 3: Profiling EIDORS and NDRM for 2D; Total run time as node density increased and shows percentage of run time for sensitivity estimation (Jacobian), and regularizing the inverse and solving. Meshing time is shown as a multiple of the two (scaled down by a factor of two hundred). For each node density, EIDORS is shown on the left and NDRM on the right. Total run time is shown against the right-hand side logarithmic scale while percentage run time is shown against the left-hand side linear scale. The percentage of run time devoted to sensitivity estimation as well as total run time increased with node density.

solution. The factorization requires the majority of the computational effort while forward and backward substitution is relatively quick. For sparse matrices, the factorization is somewhat complicated by the effect of fill-in on the factorization. (Fill-in is where previously zero entries in the matrix become non-zero as the factorization proceeds which leads to an increase in the number of computations and storage space required.) This can be an expensive computational problem so that heuristics are required to select a "good" factorization strategy without wasting too much time considering alternatives and thus, cancelling the potential time benefit that might be achieved by finding a better factorization.

Using multiple processors holds out the possibility of faster computations, but the advantage of additional worker CPUs is counterbalanced by the need to communicate between these units and the components of the process that require serialization. The direct sparse matrix solution process is commonly split into four phases. An initial symbolic factorization is found based on the location of non-zero entries in the matrix. This factorization is then implemented using the numeric values in the matrix. The right-hand sides are then used with forward and backward substitution to find specific

solutions. Finally, depending on numerical accuracy concerns, iterative refinement can be used to correct for any numeric errors introduced in the prior steps. Each of these stages can be completed in parallel, but in general, at each step some degree of synchronization between the workers limits the maximum achievable parallelization speedup.

The most computationally expensive components of the sparse calculations are in the symbolic and numeric factorizations. Since a poor symbolic factorization leads to an expensive numeric factorization, symbolic factorization has received considerable attention in the literature (Liu, 1989; Duff et al., 1990; Davis et al., 2004).

Iterative sparse solvers require a set of matrix addition and multiplication operations and generally, use a direct solution to a much smaller matrix as part of the process. Examples of this are the Conjugate Gradient and Generalized Minimum Residual (GMRES) methods. Iterative solutions are also required for non-linear EIT problems. In this work, we chose to focus on linearized difference EIT and the direct solvers that provide an efficient mechanism for obtaining a solution to these problems because they also provide a core component for large scale iterative solvers.

To test the performance of sparse direct solvers, we developed a common testing platform called Meagre-Crowd. (Meagre-Crowd is open-source software released under the GNU GPL version 3 license and available at `http://github.com/boyle/meagre-crowd`.) Meagre-Crowd addresses the challenge of integrating a wide range of matrix solvers in a uniform framework and provides assistance with compiling and linking these dependencies. This platform is a mechanism for measuring the parallel sparse matrix solution performance of a number of integrated solvers. These solvers include single core solvers (UMFPACK), out-of-core file-based solvers (TAUCS), symmetric solvers (CHOLMOD), and multicore based solvers (WSMP, MUMPS, Pardiso, SuperLU_DIST). Meagre-Crowd ensures an apples-to-apples comparison by using a common set of underlying numerical libraries for all solvers. Meagre-Crowd provides conversion functionality between the various matrix storage formats, so that each solver can use its native format and allows loading from common file formats such as MatrixMarket. It also provides an autotools based configuration and build system to facilitate ease-of-use and a test suite to confirm correct operation.

3. Procedure

Profiling of EIDORS and NDRM was carried out by timing the components of the solution process. Coarse and fine meshes were generated with Netgen and the geometric mapping between the two was calculated. This phase was counted as "meshing." NDRM uses offline mesh generation so common meshes between EIDORS and NDRM were used. The meshes were used to build FEM models including Complete Electrode Model (CEM) boundary conditions. Forward simulations were run using unit stimulations according to the stimulation and measurement protocol, and these were used to estimate the Jacobian via the self-adjoint method. This stage was counted as the "Jacobian" time. The FEM system matrices A, node voltages x and boundary conditions b were saved for later use in the sparse solver comparison. Finally, the Jacobian was used in combination with Tikhonov regularization using Laplacian smoothing to build the inverse problem and solved with a single-step Gauss-Newton iteration. This final step was counted as the "regularized inverse." The time required for each of these steps was recorded, and the results were plotted in Figure 2 and Figure 3.

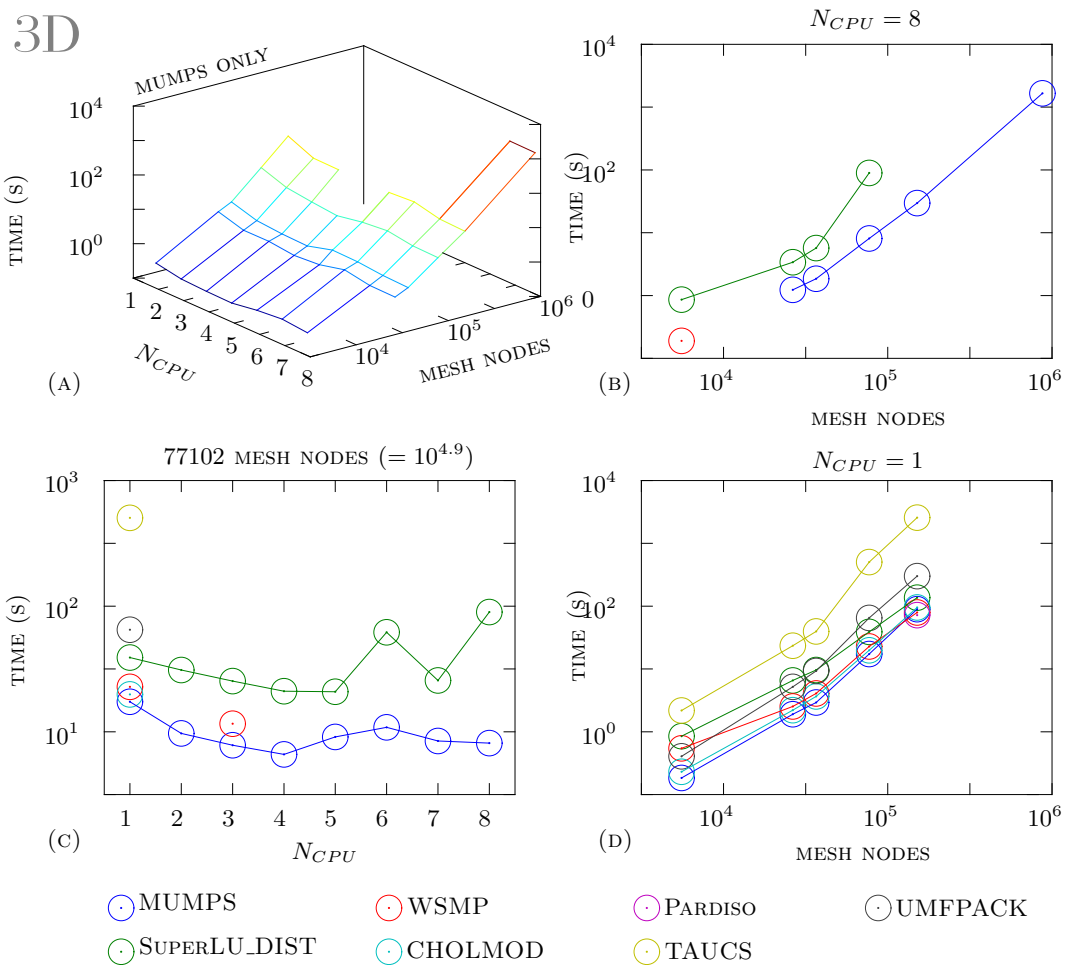

Figure 4: Performance Tests in Meagre-Crowd for 3D EIT; Solution time plotted as a function of number of FEM nodes and number of processing cores. MUMPS performs the best on these test problems and shows a significant processing advantage when up to four processors are assigned to the task. Note that CHOLMOD times are normalized by multiplying by two because it is a symmetric solver and required to do approximately half the work.

Two meshes were used to solve the inverse problem. A fine mesh was required for accuracy on the forward simulations. A coarse mesh was required to manage the ill-conditioning of the inverse problem. The fine mesh was used to estimate a Jacobian based on boundary voltage changes due to small perturbations in the most recent estimate of conductivity. To solve the inverse problem, the Jacobian, a regularization scheme and boundary voltage measurements were used to estimate the conductivity. Small elements (for example from the fine mesh) have little affect on the boundary elements and significantly contribute to ill-conditioning of the matrices.

The geometric mapping between coarse and fine meshes can be an expensive

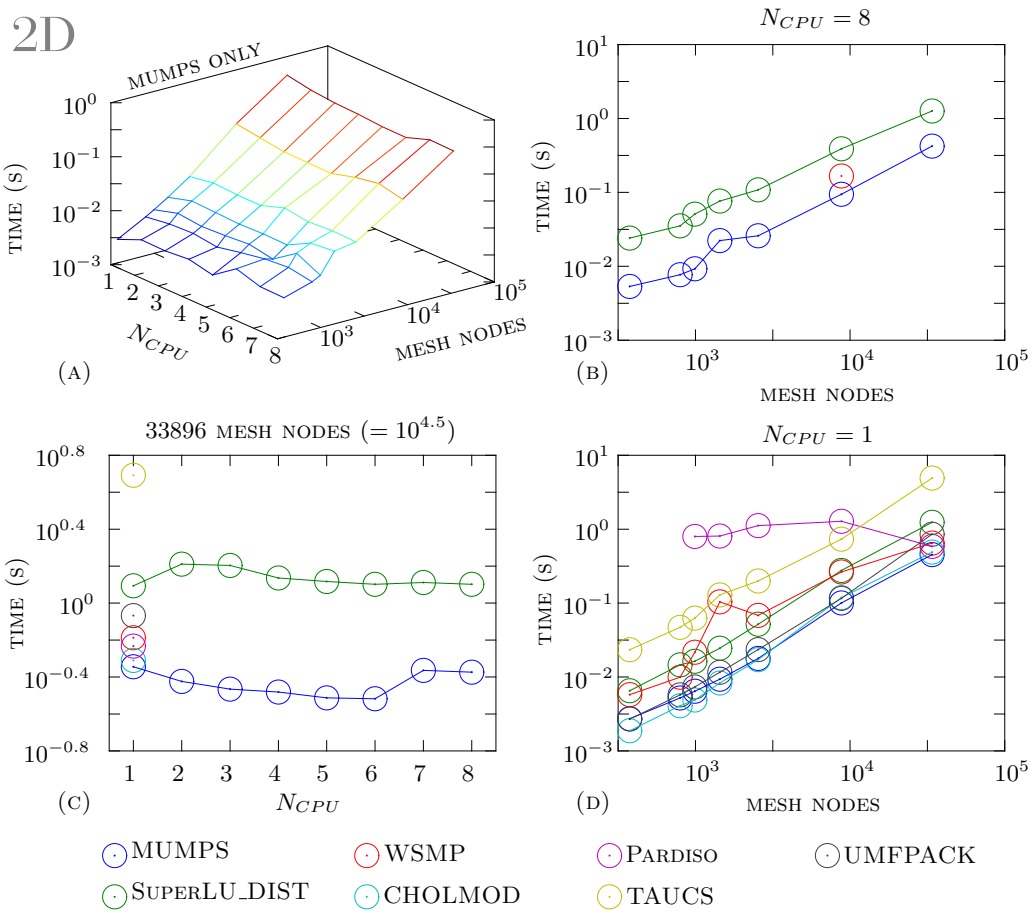

Figure 5: Performance Tests in Meagre-Crowd for 2D EIT; Solution time plotted as a function of number of FEM nodes and number of processing cores. No advantage for the multiprocessing options is observed on these test problems. Note that CHOLMOD times are normalized by multiplying by two because it is a symmetric solver and required to do approximately half the work.

operation depending on the algorithm choice. NDRM finds the elements of the fine mesh most closely related to each coarse mesh element while EIDORS calculates geometric overlap fraction from the coarse to fine mesh. The EIDORS procedure is more accurate but more computationally intensive for large numbers of elements. The NDRM algorithm is most appropriate when the coarse mesh has elements much larger than the fine mesh so that the error introduced in the mapping is small. The cost of doing these computation with either algorithm scales as the number of elements grows.

EIDORS 3.5 was run on an 8-core Intel Xeon X5550 SMP system at 2.67GHz with 64GB memory using NETGEN 4.9.11 and MATLAB 7.13.0.564 (R2011b containing UMFPACK 5.4.0). NDRM was executed using a 4-core Intel Core i5-750 system at 2.67GHz with 12GB memory using the same version of MATLAB. For the purposes

of this comparison these processors were equivalent because they were using the same processor microarchitecture, the same cache size and running at the same clock frequency. The processes were restricted to a single processing core and were not memory bound.

The profiling exercise was done over a range of fine-mesh densities ranging from 5510 to 873805 nodes (22994 to 4792380 elements) in three dimensions and 362 to 33882 nodes (632 to 67018 elements) in two dimensions. This gave comparable element resolution on the electrode plane between two- and three-dimensional problems. The maximum size of the meshes was limited by EIDORS's memory consumption (64GB). Three electrode rings of fifteen electrodes each were used for the three-dimensional problems. A single ring of electrodes was used in the two-dimensional problems. Element refinement in the regions near the electrodes was set to have node densities five times that of the central region. The coarse mesh was designed to have one quarter the node density of the fine mesh but was somewhat constrained by the problem geometry and refinement near the electrodes. Circular and cylindrical geometries were modelled.

The sparse matrix performance measurements using Meagre-Crowd were performed on the same hardware as the EIDORS profiling: an 8-core system. For each of the two- and three-dimensional FEM system matrices, the system was solved for all corresponding right-hand sides on a range of one to eight simultaneous processors. A common EIT FEM matrix was used at each mesh node density for all solvers. For three-dimensional problems, 45 orthogonal right-hand sides resulted from the electrode stimulation protocol and 15 in two dimensions. The expected solution was stored from EIDORS 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. These measurements address the sparse matrix solution phase but not the sparse multiplication required to complete the calculation of the Jacobian in the self-adjoint method. Meagre-Crowd 0.4.6 (Boyle et al., 2011) was used to test the performance of the sparse matrix solvers: UMFPACK 5.5.0 (Davis, 2004), MUMPS 4.9.2 (Amestoy et al., 2006), WSMP 11.01.19 (Gupta et al., 1997), Pardiso 4.1.1 (Schenk and Gärtner, 2004), TAUCS 2.2 (Toledo et al., 2003), SuperLU_DIST 2.5 (Li and Demmel, 2003) and CHOLMOD 1.7.1 (Chen et al., 2008). (CHOLMOD is a symmetric, positive-definite solver used as a comparison versus these unsymmetric solvers.) These solvers shared a common set of dependencies: Netlib BLAS update Apr 19 2011, LAPACK 3.3.0, ScaLAPACK 2.0. Choosing an implementation of MPI and BLAS optimized for the current platform did have a significant effect on the run-time but did not affect inter-solver comparisons within the Meagre-Crowd testbench. Where possible the sparse matrix ordering codes were also common: AMD 2.2.1, CAMD 2.2.1, CCOLAMD 2.7.2, COLAMD 2.7.2, ParMETIS 3.1.1, and Scotch 5.1.10b. The solver configurations were left at their default (generally automated and heuristically optimized) settings rather than having these controls adjusted to achieve optimal performance for specific matrices.

4. Results

Figure 2 shows profiling results for EIDORS on three-dimensional problems. A dashed line with half circle markers shows the exponential increase in run time as a function of mesh nodes. The percentage of run time is plotted as 100% of what would be a single step iteration of a Gauss-Newton type solver. The meshing time (scaled) is plotted as

an additional percentage of this inner iteration time. In EIDORS, as the mesh node density increased the relative amount of time required for meshing also increased but then fell for higher density meshes. This was a function of the coarse-to-fine mesh mapping which dominated run time for mid-density meshes. The figure also shows profiling performed using NDRM for the same three-dimensional problems. Meshing took a constant amount of time relative to the rest of the process due in large part to the simpler coarse-to-fine mapping algorithm employed there. Actual mesh generation times between EIDORS and NDRM were identical as a result of having NETGEN in common. With increased mesh node density, NDRM consumed the majority of its time in the sparse Jacobian calculations. Run time overall was less than EIDORS for larger problems.

For two-dimensional problems, Figure 3 shows that meshing time dominated the rest of the problem for equivalent element resolution on the central electrode plane of the three-dimensional problem. Jacobian calculations represented a majority of the remaining time, but these times were quite short.

The profiling results showed that the sparse calculations consumed a major portion of the time for more finely meshed three-dimensional problems. The performance results from the Meagre-Crowd testbench showed that, for a given problem size, these sparse solvers are consistently spread over a range of 1.5 orders of magnitude in time. UMFPACK, similar to the version used in MATLAB, is in the middle of this group. (Figure 4, Figure 5) For three-dimensional problems, up to four cores provided speed improvements. Applying more cores was detrimental to performance. For two-dimensional problems, up to five cores provided speed improvements for the larger problems. For larger matrices, the run times were much faster than for their three-dimensional equivalent problems. The smallest two-dimensional problems experienced slow downs when solved with multiple cores.

Some locations in the sparse solver performance plots were left blank. This was for a number of reasons including the following: some solvers only supported single-core operation; some solvers failed to factorize the system matrix; some solvers exhibited memory management problems; some shortcomings in the integration of solvers into Meagre-Crowd; and other missing data points represented failed heuristics in the front ends of these codes that tried to select an appropriate configuration automatically with occasional spectacular failures.

Memory usage for most solvers remained below 1GB with the exception of UMFPACK on the largest problem at 2.1GB which indicates that neither memory capacity nor swapping were a factor in the sparse solvers' performance.

5. Discussion

In this work, we explored the computational cost of solving EIT inverse problems. A profile of the performance of components used in the solution of two- and three-dimensional EIT was constructed, and it was shown that a large portion of the time was spent in sparse matrix calculations as the mesh density grew for two- and three-dimensional problems. The sparse matrix performance of a selection of solvers was compared against mesh density, processing cores and two- versus three-dimensional problems using a testbench (Meagre-Crowd) that allowed an apples-to-apples comparison of sparse solvers. It was found that some reasonable benefit in computation time can be gained for three-dimensional problems with the best of these solvers for a restricted number of processing cores but that the benefits are challenging

to realize due to the complexity of building and integrating some solvers (Scott and Hu, 2007). Larger problems magnified the benefits and extended the number of cores which could be profitably used.

Our general guidance regarding the use of parallel sparse matrix solvers with respect to EIT is that their performance is closely related to the problem. They are not necessarily "just better." To obtain a satisfactory comparison an appropriate procedure is to first determine a range of mesh densities of interest, both coarse and fine, that satisfy requirements. Second, select a set of candidate solvers and ensure that they will all be using the same dependencies. Third, iterate over the solvers, number of processors and range of mesh node densities to find a solver which has satisfactory performance over the expected range. In considering the merits of a solver, the quality of the software, whether the source is available, and whether the software build and integration are "easy" might be further criteria to narrow the field of candidates. In order to assist in this process we have created a tutorial with an example profiling problem as an electronic supplement and released with EIDORS.

The EIT toolkits examined in this work (EIDORS and NDRM) are somewhat restricted in the domain they can explore by memory usage limitations. Work on reducing the memory consumption of these codes in MATLAB would be beneficial if they are to continue to be efficient research platforms for large EIT problems.

This work does not use MATLAB's parallel processing feature set which relies on a multi-threaded BLAS implementation rather than parallel processing sparse solvers. Without moving to MATLAB's Parallel Computing Toolbox, the scalability of MATLAB based solutions is restricted to symmetric (homogeneous) shared memory systems. The size of problems that can be handled in MATLAB appears to match this general restriction. Sparse solver development to date has been based on the assumption that optimal resource usage cannot be achieved by low-level parallelization alone (Duff et al., 2002). To build larger scaled EIT systems appears to require specific and careful memory handling and close integration with the other tools: meshing, sparse and dense matrix manipulations. The multicore sparse solvers measured in this comparison largely use MPI for communication which allows extension to distributed memory systems. This comes at the cost of some software complexity and setup overhead.

This work has helped shed some light on the computational costs of EIT problems and what bottlenecks might be expected in the future as we try to take more complete advantage of available multicore processors.

The open-source Meagre-Crowd sparse matrix performance testbench software is available at `http://github.com/boyle/meagre-crowd`

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

A. Adler and W. R. B. Lionheart. Uses and abuses of EIDORS: An extensible software base for EIT. *Physiol. Meas.*, 27(5):S25–S42, May 2006. ISSN 0967-3334. doi: 10.1088/0967-3334/27/5/S03.

G. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings AFIPS '67*, Apr 1967.

P. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

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

A. Boyle, A. Adler, and A. Borsic. Scaling the EIT problem. In *12th Conf. Electrical Impedance Tomography*, Univ. of Bath, Bath, UK, May 2011.

W. R. Breckon. *Image reconstruction in electrical impedance tomography.* PhD thesis, 1990.

J. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, January 1977.

Y. Chen, T. Davis, W. Hager, and S. Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Software*, 35(3), October 2008.

T. Davis. Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method. *ACM Trans. on Math. Software*, 30(2):196–199, 2004.

T. Davis, J. Gilbert, S. Larimore, and E. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Software*, 30(3), 2004.

I. Duff, N. Gould, J. Reid, A. Scott, and K. Turner. The factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11(2):181–204, 1990.

I. Duff, M. Heroux, and R. Pozo. An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum. *ACM Trans. Math. Software*, 28(2), June 2002.

D. Ewing, , A. Fawkes, and J. Griffiths. Rules governing the numbers of nodes and elements in a finite element mesh. *International Journal for Numerical Methods in Engineering*, 2 (4):597–600, December 1970.

S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1–4):153–174, 1987.

B. Grychtol, W. R. B. Lionheart, G. K. Wolf, M. Bodenstein, and A. Adler. The importance of shape: thorax models for GREIT. In *Conf EIT 2011*, Bath, UK, May 2011.

L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi. *ACM Trans. Graphics*, 4(2), April 1985.

A. Gupta, G. Karypis, and V. Kumar. A highly scalable parallel algorithm for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, May 1997.

J. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5), May 1988.

M. Hill and M. Marty. Amdahl's Law in the multicore era. *IEEE Computer*, 41(7):33–38, July 2008.

X. S. Li and J. Demmel. Superlu_dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Software*, 29(2):110–140, 2003.

J. Liu. The role of elimination trees in sparse factorization. *SIAM. J. Matrix Anal. & Appl.*, 11(1):134–172, 1989.

R. Löhner and P. Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *Int. J. Numerical Methods in Fluids*, 8(10):1135–1149, October 1988.

J. Parkhurst, J. Darringer, and B. Grundmann. From single core to multi-core: preparing for a new exponential. In *ACM ICCAD '06*, 2006.

N. Polydorides and W. R. B. Lionheart. A Matlab toolkit for three-dimensional electrical impedance tomography: a contribution to the Electrical Impedance and Diffuse Optical Reconstruction Software project. *Meas. Sci. and Tech.*, 13(12):1871–1883, 2002.

T. Purcell, I. Buck, W. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *ACM SIGGRAPH '02*, volume 21, 2002.

R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997.

O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004.

J. Scott and Y. Hu. Experiences of sparse direct symmetric solvers. *ACM Trans. Math. Software*, 33(3), August 2007.

M. Shephard and M. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *Int. J. Numerical Methods in Eng.*, 32(4):709–749, 1991.

G. Strang and G. Fix. *An Analysis of the Finite Element Method*. Wellesley Cambridge Press, August 1987.

X. Suna and J. Gustafsona. Benchmarking of high performance supercomputers. *Parallel Computing*, 17(10-11):1093–1109, December 1991.

A. Tizzard, L. Horesh, R. Yerworth, D. Holder, and R. Bayford. Generating accurate finite element meshes for the forward model of the human head in EIT. *Physiol. Meas.*, 26: 251–261, 2005.

S. Toledo, D. Chen, and V. Rotkin. Taucs: A library of sparse linear solvers. 2.2, 2003. URL `http://www.tau.ac.il/~stoledo/taucs/`.

D. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.