ArjunAir: updating and parallelizing an existing time domain electromagnetic inversion program

Patrick Belliveau, Colin Farquharson, Ronald Haynes, Memorial University of Newfoundland*

SUMMARY

Results from ongoing work to parallelize the existing 2.5D airborne electromagnetic inversion program ArjunAir are presented here. ArjunAir is the only code known to the authors to see extended use in the mineral exploration industry for the rigorous inversion of time domain airborne electromagnetic (EM) data using a two-dimensional (2D) conductivity model. This study sought to increase the efficiency of the code by re-implementing the most computationally expensive calculations with modern high-performance routines, employing parallel algorithms wherever possible. Distributed memory and shared memory versions of the ArjunAir forward solver have been developed. Speedups as high as 23.7 for the distributed memory code and 15 for the shared memory workstation version, relative to the original code, have been achieved. Ongoing work is focused on developing a hybrid MPI/OpenMP forward solver, and on building a minimum structure inversion code using the new implementation of the forward solver. This will replace the existing inversion algorithm, which is based on a non-linear damped least-squares fit to the data.

INTRODUCTION

ArjunAir was originally developed at the Australian Commonwealth Science and Industrial Research organization (CSIRO). For an overview of the program, see Wilson et al. (2006). The code allows for rigorous inversion of frequency and time domain airborne EM survey data using a 2D earth model. The magnetic dipole sources used in airborne EM surveys generate electromagnetic fields that vary in three dimensions but when a 2D conductivity model is assumed, the full behaviour of the fields along a 2D section can be modelled by decomposing the problem into a set of 2D subproblems. This technique is known as 2.5D modelling (e.g. Hohmann, 1987; Key and Ovall, 2011).

ArjunAir was made open-source in 2010. That presented an opportunity to examine the source code and look for areas that could be improved, with the goal of creating a new, more efficient version of the program that retains the full functionality of the original code and its ability to interface with the commercial electromagnetic processing and modelling software Maxwell (Wilson et al., 2006). This has led to a new version of ArjunAir that is able to invert, in a practical length of time, much bigger datasets on denser meshes than was feasible with the original version. A practical length of time is loosely defined here as less than a full day, the same definition used by the original developers (Wilson et al., 2006). This project has also shown that it is possible to use modern optimized software libraries and parallel programming tools to significantly improve the performance of a piece of legacy geophysics software and adapt it to take advantage of modern high performance computing hardware.

ArjunAir's inversion algorithm is based on the Levenberg-Marquardt algorithm. An overview of the algorithm is given by Raiche et al. (1985) and is based on the work of Jupp and Vozoff (1975). The inversion scheme is iterative. The conductivity model is updated at each iteration by finding the update vector that minimizes a linearized expression for the data misfit in a damped least-squares sense. Computing the model update is based on the damped generalized inverse of the Jacobian matrix of the linearized discrete forward modelling operator. Computing the generalized inverse requires taking the singular value decomposition of the Jacobian, which is dense and stored in full form. The entries of the Jacobian are calculated using the adjoint operator method, described by McGillivray et al. (1994).

The most time consuming calculations in the inversion are the computation of the data misfit, the entries of the Jacobian matrix, and the singular value decomposition (SVD) of the Jacobian. Computing the misfit requires a forward modelling to compute the response of the current earth model to each EM source used in the inversion. Computing the Jacobian requires a forward modelling using adjoint transmitter locations, as well as the numerical evaluation of the integral of the product of the actual and adjoint electric fields over each cell in the 2D mesh.

It is clear from the last paragraph that an efficient forward modelling routine is an essential prerequisite of a fast inversion algorithm. To that end this project has focused on improving the performance of the ArjunAir forward solver. This goal has been largely accomplished and a summary of results is presented below. Current work is focused on using the ArjunAir forward solver within a minimum structure inversion algorithm, similar to the approaches taken in, e.g., (Farquharson et al., 2003; Key, 2012). Such an approach avoids having to compute and store the SVD of the Jacobian and should be much less sensitive to the initial conductivity model input to the inversion program.

FORWARD MODELLING

ArjunAir models the response of a 2D earth model to a time domain EM system by converting the time domain forward problem to a set of frequency domain problems. The frequency domain problem is solved at a user customizable range of logarithmically spaced frequencies. Time domain behaviour is then recovered by inverse Fourier transformation. ArjunAir uses a primary/secondary field separation, where the primary field is the free-space magnetic dipole field. Using a free-space primary field implies that the anomalous conductivity, σ_a is equal to the total conductivity, σ. The frequency domain equations for the secondary electric and magnetic fields $\mathbf{E}^s$ and $\mathbf{H}^s$ can then be written

$$\nabla \times \mathbf{E}^s = -i\mu\omega\mathbf{H}^s, \quad \nabla \times \mathbf{H}^s = \sigma(\mathbf{E}^s + \mathbf{E}^p) \qquad (1)$$

where μ is magnetic permeability, ω is the angular frequency of the fields, and $\mathbf{E}^p$ is the primary electric field. Electrical conductivity is assumed to be isotropic and to vary spatially only in the *x-z* Cartesian plane. Decomposing equations 1 into their Cartesian components and Fourier transforming with respect to the along-strike coordinate leads to a set of two coupled scalar partial differential equations for the along strike components of the electric and magnetic fields in the wavenumber domain, where the fields are functions of x, z, ω and k_y, where k_y is the along-strike wavenumber. This set of equations is solved on a 2D mesh at 21 values of the along-strike wavenumber, using an iso-parametric finite-element method, described by Sugeng et al. (1993). The source behaviour is encompassed by the primary field, eliminating the need to explicitly include the magnetic dipole sources in the finite element scheme. The other components of the wavenumber domain secondary fields are known in terms of spatial derivatives of the along-strike fields and can be computed by numerical differentiation. The frequency domain magnetic fields at the receiver locations are recovered by inverse Fourier transformation from the wavenumber domain.

Thus, the forward problem is composed of solving 21 independent wavenumber domain 2D problems and converting the resulting magnetic fields at the observation locations to the time domain. The independence of these wavenumber domain problems presents an obvious opportunity for parallelization. In a cluster or grid environment, solving multiple wavenumber domain problems concurrently proved to be a very successful strategy but on a single workstation, in the context of inversion where the Jacobian matrix must be stored concurrently with all data required for the forward solve, each wavenumber domain problem still requires too much memory for more than one to be run concurrently. Since developing a code that is efficient on a multi-core workstation was a key goal of this project, it was necessary to look for opportunities for parallelization and calculations that could be performed using faster algorithms within each 2D problem.

The two main bottlenecks in the 2D problem are the solution of the finite element system of equations and the computation of the primary electric field in the wavenumber domain. Since the inversion algorithm requires only the magnetic fields at the observation locations, only those fields need to be inverse Fourier transformed from the wavenumber domain and thus that transformation is a negligible part of the overall forward modelling run-time.

Efforts to increase the performance of the forward solver were focused almost exclusively on these two bottleneck calculations. It was suspected that the solution of the finite element equations would be by far the most time consuming component of the forward modelling calculation but profiling revealed the primary field computation to be another significant bottleneck. Together, these two computations make up approximately 99% of the forward modelling run-time on large problems. The number is slightly misleading because the assembly and solution of the finite element equations are interspersed in the original version of ArjunAir. Computation of the primary electric field in the wavenumber domain will be discussed first.

Primary electric field

The electric field due to a harmonic magnetic dipole of frequency ω and unit magnetization, oriented in the *x-z* plane at an angle θ from the *z* axis, is given in Cartesian coordinates by the closed form expression

$$\mathbf{E} = \frac{i\omega\mu}{4\pi r^3}\left(y\cos\theta\hat{\mathbf{x}} + (z\sin\theta - x\cos\theta)\hat{\mathbf{y}} + y\sin\theta\hat{\mathbf{z}}\right) \tag{2}$$

where $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are unit vectors and $r = \sqrt{x^2+y^2+z^2}$. A closed form expression for the free-space magnetic dipole field in the wavenumber domain does not exist. It must be computed by numerical Fourier transformation of equation 2. The digital filtering technique (e.g., Anderson, 1982; Christensen, 1990), applied more often in electromagnetic geophysics for the computation of Hankel transforms, provides an efficient method for computing Fourier transforms when the sample points in the wavenumber domain are logarithmically spaced, as in ArjunAir. The ArjunAir developers wrote their own digital filtering routine using filter coefficients computed using software developed by Christensen (1990). The transform of the primary electric field must be computed for each transmitter location and frequency, at every node in the finite element mesh.

If this is done by explicit use of digital filtering at every node, frequency and transmitter, a time domain problem using 100 transmitter locations and a mesh of 10,000 elements would require $\sim 10^8$ calls to the digital filtering routine for each 2D subproblem and $\sim 10^9$ calls for a full forward modelling. This was the method used in the original version of ArjunAir. However, the Fourier integrals required to compute each transform do not depend on frequency, and spatially they depend only on the quantity $\rho = x^2 + z^2$, not on x and z separately. For example, the y component of the wavenumber domain primary field, $\tilde{E}_y$, is given by

$$\tilde{E}_y = \frac{i\omega\mu}{4\pi}(z\sin\theta - x\cos\theta)\int_{-\infty}^{\infty}\frac{\mathrm{e}^{-ik_y y}}{(\rho^2+y^2)^{3/2}}\mathrm{d}y. \tag{3}$$

The expressions for $\tilde{E}_x$ and $\tilde{E}_z$ are similar. The value of the integrals for all three components vary smoothly as a function of ρ. Therefore it should be possible to compute the integrals for a representative set of values of ρ and interpolate between these values to find the value of the integrals at the 2D mesh node points for each transmitter location. This strategy proved successful and reduced the primary field computation run-time by 92-93% on medium and large problems. Figure 1 shows run-time as a function of the number of transmitter locations for a VTEM style survey on a mesh with 71221 nodes (23432 elements). Fourier integrals were computed using digital filtering at a set of linearly spaced values of ρ. For each transmitter location, cubic spline routines from ArjunAir were used to interpolate the values of the Fourier integrals from the sampled values of ρ to the mesh node points less than a given cutoff distance from transmitter. The primary field was set to zero for nodes outside the cutoff distance. Using a lookup table with integrals evaluated at 2231 values of ρ and a cutoff distance of 2000 m provided sufficient accuracy to introduce negligible error in the final results on all models tested.

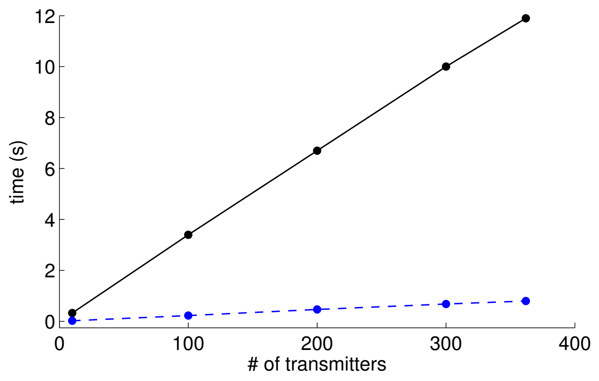


Figure 1: Primary field computation run-times. The solid black line shows results using explicit Fourier transformation at each node. The dashed blue line shows the lookup table results.

Solving the finite element equations

ArjunAir allows the user to customize the range of frequencies used to approximate the time domain behaviour of the EM fields. By default, 28 frequencies are used. Each frequency domain problem is in turn decomposed into 21 2D wavenumber domain problems. Thus in a typical forward problem, 588 2D problems need to be solved. This leads to 588 finite-element coefficient matrices that each need to be solved against n_t right hand sides (RHS), where n_t is the number of transmitters being modelled. When using an iterative solver, solving against each RHS represents a separate problem, and run-time scales linearly with the number of transmitters. However, if one is able to factor the coefficient matrices, solving the factored systems against multiple transmitters is very fast, with solve time per transmitter being 1-2% of the factorization time (Wilson et al., 2006; Oldenburg et al., 2013).

Developing an efficient sparse direct solver is a very specialized task. Several excellent high performance parallel sparse direct solvers are now available (Gould et al., 2007), which was not the case when ArjunAir was originally developed. It uses a custom made sparse direct solver, the developers' own implementation of the frontal method of sparse matrix factorization, due to Irons (1970). Although impressive considering it was written by geophysicists and not experts in numerical linear algebra, ArjunAir's frontal solver is naive and inefficient compared to modern software packages that take advantage of advanced matrix reordering techniques and highly optimized dense linear algebra library routines.

Two such packages, MuMPS (Amestoy et al., 2001) and Pardiso (Schenk et al., 2000) were tested in this study. The version of Pardiso included in release 10.3.6 of the Intel Math Kernel Library was used. MuMPS is a fully scalable distributed memory MPI solver that uses a multifrontal factorization technique. Pardiso is a shared memory code parallelized with OpenMP that uses a supernodal factorization algorithm. Figure 2 compares the run-times of each solver run in sequential mode on an assortment of ArjunAir forward problems, showing how performance varies as a function of the order of the finite element coefficient matrix (twice the number of nodes in the mesh) and as a function of the number of transmitters for a fixed matrix size. These plots show that even without parallel speedup, the MuMPS and Pardiso solvers show substantial improvement over the original ArjunAir code, with Pardiso offering superior performance to MuMPS.

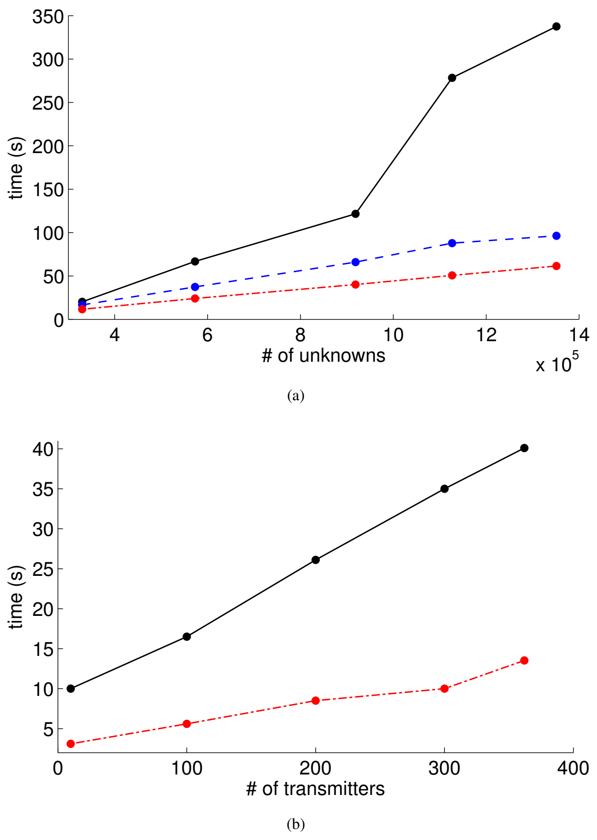


Figure 2: Time to assemble, factor and solve the finite element equations plotted against a) the number of unknowns for a fixed number of RHS, and b) the number of RHS for a fixed mesh size. Original ArjunAir solver times are plotted on the solid black line. MuMPS run-times are on the dashed blue line and Pardiso times are on the dot-dashed red line.

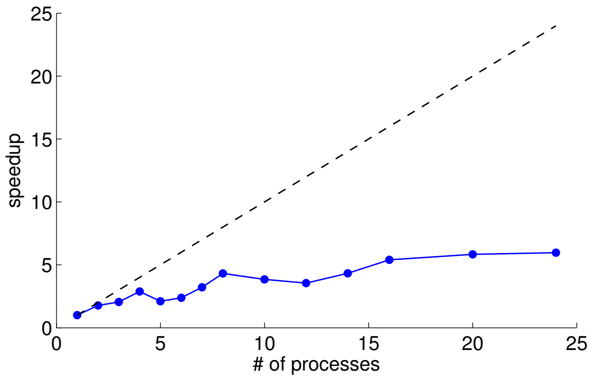
MuMPS was chosen for its ability to scale to a large number of processors and work directly with unassembled finite-element matrices. We tested it across multiple nodes of a Linux cluster. Each node on the cluster has two six core Intel Xeon CPUs. Limited benefit was derived from running MuMPS with enough MPI processes to require multiple nodes. Scaling was acceptable only with a small number of processes running on a single node. The main metric used to evaluate parallel scaling performance was speedup, defined as the run-time for a sequential algorithm, divided by the run-time for the parallel version of that algorithm on the same problem. Figures 3(a) and 3(b) show scaling results for MuMPS on a 16 km long mesh, with 100 transmitters, spaced approximately every 170 m along the mesh. This tests the factorization performance of the solvers on large matrices with a relatively small number of RHS for the length of the line. In a more typical large scale scenario, e.g. a 4 km mesh with 20 m transmitter location spacing (362 transmitters), solving against the multiple RHS dominates the computation time compared to factorization. Better speedup is observed in this scenario, as shown in Figure 3(c). Speedup in factorization is limited by the analysis and pre-processing of the matrix and by its sparsity structure. Solution of the system by back-substitution can be performed independently for each RHS and is therefore a trivially parallel calculation. This is reflected in Figure 3(c), where speedup in the solve phase leads to almost linear speedup in the entire calculation, for up to five threads. However, poorer scaling in the analysis and factorization of the matrix limit the speedup beyond five threads. Similar factors limit the scaling of MuMPS and in addition, as a distributed memory code, it likely has greater overheads for communication and data movement than Pardiso.

Parallelization over full 2D solves

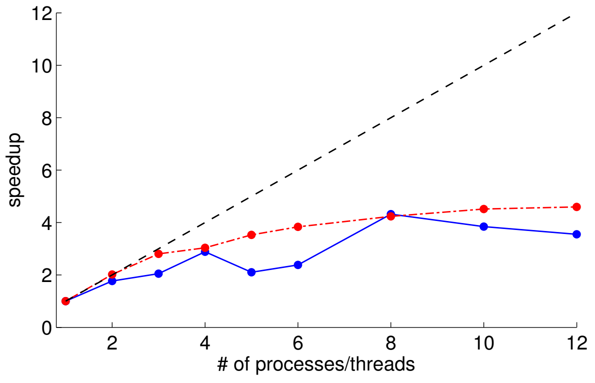
The excellent absolute performance of Pardiso and its scaling over a small number of threads, the poor scaling performance of both Pardiso and MuMPS on larger numbers of threads or processes, and the independent nature of the 2D wavenumber domain subproblems in 2.5D modelling argue for a hybrid approach when adequate memory resources are available to solve multiple subproblems concurrently. In a modern cluster setting, each node is often a multi-core computer. On such a cluster it makes sense to use MPI to split a computation into several large tasks and assign each one to a separate node. Each task can then be further parallelized using shared memory tools such as OpenMP. In the case of ArjunAir, MPI was used to assign one 2D problem to each node and OpenMP was used to parallelize the solution of the 2D problems within the nodes. When using only one thread per MPI process, almost linear speedup was achieved, with a maximum speedup of 8 on 10 nodes, the largest number of nodes tested. The discrepancy between the observed and ideal speedup is due to imperfect load balancing. Unfortunately results using multiple threads per process have not yet been obtained. Work on this front is ongoing.

CONCLUSIONS AND FUTURE WORK

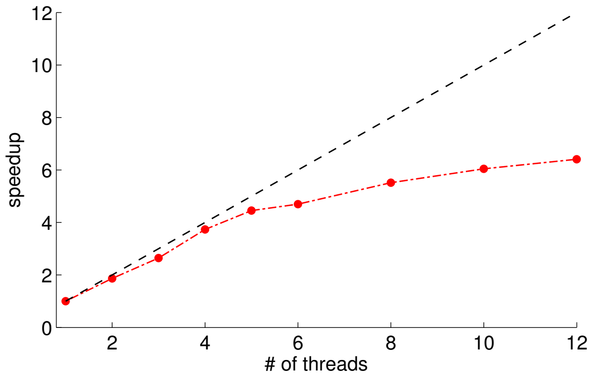
Improvements in the forward modelling routine have significantly decreased ArjunAir inversion run-times. The improvements have also resulted in a small decrease in the code's memory footprint but memory constraints remain an impediment to running inversions on larger datasets and meshes. Storing the Jacobian matrix and its SVD is the biggest memory sink in ArjunAir. It is a large dense matrix and must be formed in the spatial wavenumber domain, with copies for all wavenumbers needing to be stored simultaneously. Current work on implementing a minimum structure inversion using the ArjunAir forward solver will eliminate the need to compute and store the SVD of the Jacobian. Using a sparse representation of the Jacobian is also being explored. When current work is completed, it is expected that speedups relative to the original code of 20-30 on a 12 core workstation and over 100 on a cluster will be achieved for full inversions.



(a)



(b)



(c)

Figure 3: MuMPS scaling results, with ideal speedup shown on the dashed black line, MuMPS observed speedup on the solid blue line and Pardiso observed speedup on the dot-dashed red line. a) MuMPS speedup with 572872 unknowns and 93 RHS on up to 2 full nodes (24 MPI processes). b) MuMPS and Pardiso speedups with 572872 unknowns and 93 RHS on a single node. c) Pardiso speedup with 142442 unknowns and 362 transmitters on a single node.