Scalable Algorithms in Optimization: Computational Experiments

Steven J. Benson[*], Lois McInnes[*], Jorge J. Moré[*], and Jason Sarich[*]

Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

We survey techniques in the Toolkit for Advanced Optimization (TAO) for developing scalable algorithms for mesh-based optimization problems on distributed architectures. We discuss the distribution of the mesh, the computation of the gradient and the Hessian matrix, and the use of preconditioners. We show that these techniques, together with mesh sequencing, can produce results that scale with mesh size.

I. Introduction

We describe strategies in the Toolkit for Advanced Optimization (TAO)[1, 2] for the solution of optimization problems of the form

$$\min \left\{ f(x) : x_l \le x \le x_u \right\}, \tag{1}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ is twice continuously differentiable and the vectors x_l and x_u are bounds on the variables x. TAO focuses on large-scale problems in which the solution of (1) requires high-performance (parallel) advanced architectures, that is, problems where time or memory requirements make the use of a distributed architecture mandatory.

Problems of the form (1) often arise as discretization of infinite-dimensional problems of the form

$$\min \left\{ f(v) : v_l \le v \le v_u \right\}, \tag{2}$$

where $v : \mathbb{R}^p \mapsto \mathbb{R}^q$ is a function defined on some domain $\mathcal{D} \subset \mathbb{R}^p$, and v_l and v_u are, respectively, the lower and upper (pointwise) bounds on v. In these cases, discretization of the domain $\mathcal{D}$ leads to a large optimization problem of the form (1) where the variables x represent the values of v at the grid points of the discretization of $\mathcal{D}$. If there are m grid points, then the number of variables $n = mq$ in (1). We use the term *mesh-based* optimization problem to refer to problems of the form (2) that require the determination of a function v on a mesh.

We explore scalable algorithms for mesh-based optimization problems in the sense that the number of operations required to solve the problem grow linearly with the number of variables. We also require that the computing time grow linearly with the number of variables; this is an essential requirement from a computational viewpoint.

Scalable algorithms for mesh-based problems generally require mesh-invariance in the sense that the number of iterations is independent of the mesh spacing. In theory, mesh-invariance can be obtained with Newton's method,[3] although in practice numerous obstacles arise.

Developing scalable algorithms in a single-processor environment can be difficult, but these difficulties are significantly magnified in a distributed processor environment because the overhead of communication between processors must be balanced with other performance considerations. Relatively little optimization

[*]9700 S. Cass Avenue, Argonne, IL 60439, (`{benson, mcinnes,more,sarich}@mcs.anl.gov`)

software exists for parallel environments, so practitioners are left to assemble their own methods. We show that TAO can be used to obtain scalable performance on distributed architectures.

The focus of this paper is on recent computational experiments with the optimization algorithms in TAO for mesh-based problems on distributed architectures. Section II reviews the Newton method in TAO and some of the tools used in the solution of mesh-based problems. We discuss the distribution of the mesh, the computation of the gradient and the Hessian matrix, and the use of appropriate preconditioners. The tools for the distribution of the mesh and for the preconditioners are provided by PETSc,[4,5] whereas the gradient is computed with ADIC.[6,7] These tools reduce many of the barriers associated with solving these problems on distributed architectures.

Section III presents computational experiments with the TAO algorithms. We use an obstacle problem for a minimal surface problem to illustrate the performance behavior. Our results show that the use of mesh sequencing overcomes convergence difficulties from poor initial guesses and that we obtain scalable performance on distributed architectures.

II. Toolkit for Advanced Optimization

The main obstacles in developing scalable algorithms for the solution of mesh-based optimization problems with Newton's method are the distribution of the mesh, the computation of the gradient ∇f and the Hessian matrix $\nabla^2 f$, and the use of appropriate preconditioners. In this section we discuss the TAO tools for overcoming these obstacles.

The Newton algorithm in TAO is based on the TRON code of Lin and Moré.[8,9] The major computational requirements of TRON are to compute the quadratic

$$q_k(w) = \langle \nabla f(x_k), w \rangle + \tfrac{1}{2}\langle w, \nabla^2 f(x_k) w \rangle,$$

which approximates the reduction $f(x_k + w) - f(x_k)$, and to compute an approximate solution of the trust region subproblem

$$\min \left\{ q_k(w) : x_k + w \in \Omega,\ \|w\| \le \Delta_k \right\},$$

where Ω is the feasible region

$$\Omega = \left\{ x \in \mathbb{R}^n : x_l \le x \le x_u \right\}.$$

TRON uses a gradient projection method to identify a Cauchy point x_k^C in the feasible region Ω and explores the face which contains the Cauchy point using a preconditioned conjugate gradient method. The efficiency of TRON can be partly be attributed to the fact that only a few faces are explored in the typical optimization problem. Convergence of TRON holds under very general conditions that include degeneracy.

The version of TRON in TAO follows this general outline but with significant and important differences. A major difference is that the TAO version was designed for a distributed environment, whereas the original version is restricted to single-processor environments. Another major difference is that the original version of TRON is restricted to the use of an incomplete Cholesky factorization,[10] whereas the TAO version can use appropriate preconditioners in PETSc.

A. Benchmarks

We use an obstacle problem for a minimal surface problem to benchmark TAO. This problem can be formulated in terms of finding a function $v : \mathcal{D} \mapsto \mathbb{R}$ that solves the optimization problem

$$\min \left\{ \int_{\mathcal{D}} \left(1 + \|\nabla v(x)\|^2\right)^{1/2} dx : v \in K \right\}, \tag{3}$$

where $\mathcal{D}$ is a domain in $\mathbb{R}^2$, the constraint set K is defined by

$$K = \left\{ v : v(x) = v_B(x),\ x \in \partial\mathcal{D} \text{ and } v(x) \ge v_L(x),\ x \in \mathcal{D} \right\},$$

$v_B : \mathcal{D} \mapsto \mathbb{R}$ is the boundary data, and $v_L : \mathcal{D} \mapsto \mathbb{R}$ is the (lower) obstacle. This is a mesh-based optimization problem where the function

$$f(v) = \int_{\mathcal{D}} \left(1 + \|\nabla v(x)\|^2\right)^{1/2} \, dx \tag{4}$$

is convex. There is a unique solution v for any boundary data v_B and obstacle v_L. The unique solution for the boundary data

$$v_B(\xi_1, \xi_2) = \begin{cases} 1 - (2\xi_1 - 1)^2 & \text{if} \quad \xi_2 \in \{0, 1\} \\ 0 & \text{otherwise} \end{cases}$$

and obstacle

$$v_L(x) = \begin{cases} 1 & \text{if} \quad x \in [\frac{1}{4}, \frac{3}{4}] \times [\frac{1}{4}, \frac{3}{4}] \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

is shown on the left of Figure 1. The behavior of the solution v at the boundary of

$$\mathcal{A}(v) = \{x \in \mathcal{D} : v(x) = v_L(x)\}$$

is of interest. We extend standard optimization terminology and refer to $\mathcal{A}(v)$ as the active set of v. For this problem the solution v is not continuously differentiable at the boundary of $\mathcal{A}(v)$, and thus $v \notin C^1(\mathcal{D})$. This is clear from Figure 1. For general obstacle problems[11] we can expect only that $v \in C^1(\mathcal{D})$.

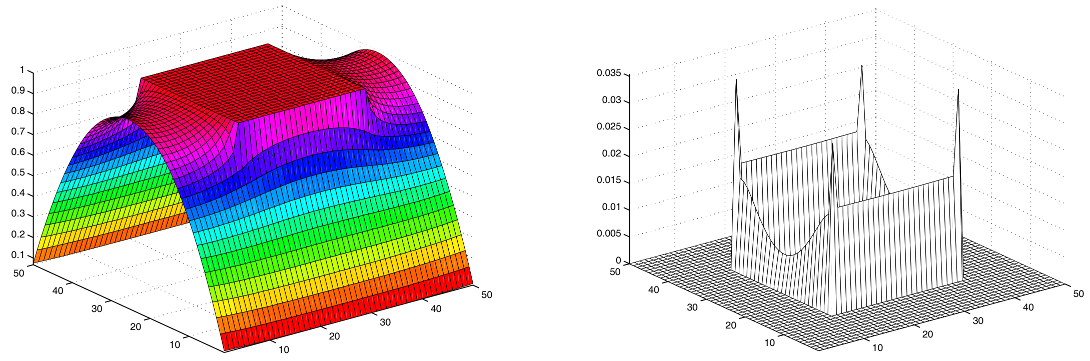

Figure 1. Solution v of the obstacle problem (3) on $\mathcal{D} = (0,1) \times (0,1)$ for an obstacle of unit height is shown on the left. The multiplier function for v is on the right.

The solution v of this obstacle problem is degenerate from an optimization viewpoint because most of the multipliers are zero at the solution. For this problem the multiplier function $\lambda : \mathcal{D} \mapsto \mathbb{R}$ is plotted on the right of Figure 1. Note that $\lambda(x) = 0$ outside the active set. In general, we have $\lambda(x) \geq 0$ on $\mathcal{A}(v)$, but in this case we have $\lambda(x) = 0$ on the interior of the active set. The discontinuity of λ on the boundary of the active set can be troublesome for primal/dual optimization algorithms that also seek convergence of the multipliers. See[12] for additional discussion on this point.

An approximation to the obstacle problem can be obtained by letting $v_{i,j}$ be the value of v at $z_{i,j}$, where $z_{i,j} \in \mathbb{R}^2$ are the vertices of a partition of $\mathcal{D}$ into rectangular elements of size $h_x \times h_y$. The function in the obstacle problem (4) can be approximated by a function

$$f(v) = \sum f_{i,j}(v),$$

where the element function $f_{i,j}$ are defined by

$$f_{i,j}(v) = h_x h_y \left\{ 1 + \left(\frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 + \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\}^{1/2}.$$

The element function $f_{i,j}$ is an approximation to the integral in (4) over a typical rectangular element. Thus, f is the sum of element functions $f_{i,j}$ that depend only on the four mesh values associated with that element.

B. Distributed Mesh

In a distributed framework we can assign to each processor a subset of the vertices in the mesh for the domain $\mathcal{D}$ so that each vertex lies on exactly one processor. Each processor computes the functions $f_{i,j}$ over a subdomain of $\mathcal{D}$ corresponding to the vertices assigned to this subdomain, but special consideration must be taken where adjacent vertices lie on different processors. Points that are adjacent to the local mesh but lying on a different processors are called *ghost points*. In order to compute the function in these regions, ghost points first must be passed between processors before the computations can occur.

PETSc[4,5] provides several utilities for managing data associated with structured and unstructured meshes. In particular, the distributed array (DA) object provides facilities for managing the field associated with a single structured mesh. Distributed arrays are intended for use with logically rectangular meshes when communication of nonlocal data is needed before local computations can occur. Support includes the exchange of data associated with ghost points and more generalized gather-scatter operations.

Once the ghost points have been passed, each processor can compute the function on its local domain by using techniques identical to those used with a single processor. In particular, each processor can sum the functions $f_{i,j}$ on each element to generate the local function value. An MPI routine call adds the function values over the subdomains. The function value over the entire mesh is found by adding the function over each of the subdomains.

Thus, in parallel, the computation of f can be separated into three stages. The first stage scatters and gathers ghost points among the processors with the PETSc DA. The second stage computes the function in parallel, each processor working on its own subdomain. The final stage uses an MPI call to add the local function values.

C. Derivatives

Newton's algorithm for the optimization problem (1) requires the gradient ∇f and the Hessian matrix $\nabla^2 f$ of the function f. Writing code that evaluates the function f can be difficult and prone to error. Developing correct parallel code for computing the derivatives magnifies these difficulties.

Automatic differentiation[13,14] is a technique for generating code that computes gradients and higher-order derivatives directly from code that evaluates only the function. ADIC[6,7] is a tool for the automatic differentiation of ANSI C programs. Given code that computes a function f, ADIC generates additional code that computes f and the gradient ∇f of f.

We apply automatic differentiation only to the element function $f_{i,j}$. This function of four variables may be written entirely in ANSI C, without calls to other subroutines. The application of ADIC to this function is much easier than applying ADIC to the entire function and all of its subroutines. This approach, however, still requires the application to gather the ghost points from neighboring processors in the first stage, loop over all the elements to evaluate the gradient of the each function, and then scatter the gradient values of the ghostpoint back to the processors on which the variables reside. While the first and third stages involve library calls to external libraries, the second stage involves a simple function evaluation in ANSI C. Thus, the gradient of our objective can be computed in parallel by separating the computational part of the work from the communication side of it using ADIC.

Since second derivatives are not currently available from ADIC, we use finite differences to compute the Hessian matrix. By coloring the the columns of the matrix such that columns of the same color do not share any common rows, we reduce the number of gradient evalations required to approximate the Hessian. The regular structure of the meshes allows us to color the columns using the Curtis-Powell-Reed[17] scheme. By perturbing all columns of the same color simultaneously, only nine gradient evaluations are needed to approximate the Hessian.

This description of the use of automatic differentiation tools and finite differences in optimization is brief; see[15, 16] for additional information on the generation of first and second order information for optimization applications in a distributed environment.

D. Scalable Preconditioners

The performance of optimization solvers depends heavily on the performance of the linear solver used in the algorithm and on the performance of the application in evaluating the function f and its derivatives. PETSc[4, 5] integrates a hierarchy of components that range from low-level distributed data structures for vectors and matrices through high-level linear solvers. This approach promotes reuse and flexibility, and in many cases, helps to decouple the problems of parallelism from algorithmic choices. PETSc provides implementations of basic objects, such as matrices, vectors, and linear solvers. The linear solvers include Krylov methods with a variety of preconditioners, such as incomplete LU factorizations, and additive Schwartz methods.

The impact of preconditioner performance on optimization algorithms has been examined[18] in the context of the gradient-projection conjugate gradient algorithm in TAO for solving bound-constrained convex quadratic problems. We analyzed the performance in terms of the number of variables, the number of free variables, and the preconditioner. Our results showed that there is a complex performance behavior for optimization problems of the form (1) and that performance results that focus only on scalability can be deceiving if the total computing time is not taken into account.

In particular, for our benchmark problems we showed that a block Jacobi preconditioner with one block per processor, where each subproblem is solved with a standard sparse ILU(2) factorization, is faster than a variant with ILU(0). We also showed that both block Jacobi variants are faster than a simple point Jacobi method, although the point Jacobi preconditioner exhibits better scalability.

E. Mesh Sequencing

Mesh sequencing is a technique for solving mesh-based problems in which an interpolated coarse mesh solution is used as the initial starting point for a finer mesh. This is a standard technique for solving systems of nonlinear (partial differential) equations, but with few exceptions it does not tend to get used for solving mesh-based optimization problems. See Bank, Gill and Marcia[12]) for a recent exception.

For systems of equations the solution v tends to be smooth, so the use of mesh sequencing is likely to produce highly accurate starting points. The solution v for constrained (mesh-based) optimization problems is not usually twice continuously differentiable, however, so the accuracy is likely to be lower. Indeed, as noted in Section II, the solution to problem (3) with obstacle (5) is only once continuously differentiable.

The lack of smoothness of v raises questions on the order of convergence of the solution v_h with mesh size h. In the next section we give results showing the improvements obtained from mesh sequencing.

III. Computational Experiments

The computational results in this section were obtained on Jazz, a 350-node Linux cluster located at Argonne's Laboratory Computing Resource Center. Each node has a 2.4 GHz Pentium Xeon, 175 nodes with 2 GB of RAM, and 175 nodes with 1 GB of RAM.

We present the results of computational experiments with the Newton algorithm in TAO to solve the obstacle problem described in Section II. The domain $\mathcal{D}$ was set to the unit square $(0,1) \times (0,1)$, and $\mathcal{D}$ was partitioned into rectangular elements of size $h_x \times h_y$. If n_x and n_y are, respectively, the number of mesh points in each coordinate, then this partition gives rise to a bound-constrained optimization problem (1) with $n = n_x n_y$ variables.

We present results for the obstacle problem with $n = 4481^2 = 20,079,361$ variables. The starting point for Newton's method is the function

$$v(\xi_1, \xi_2) = 1 - (2\xi_1 - 1)^2 \tag{6}$$

evaluated at the mesh points. Figure 1 shows that this is a good approximation to the solution of the obstacle problem outside the active set. We require that the final iterate satisfy the convergence condition

$$\|r(x)\|_2 \leq \tau = 10^{-7}, \tag{7}$$

where $r(x)$ is the projected gradient defined by

$$r(x) = \left\{ \begin{array}{lll} 0 & \text{if} & x_k = l_k,\ x_k = u_k \\ \min(0, \partial_k f(x)) & \text{if} & x_k = l_k \\ \max(0, \partial_k f(x)) & \text{if} & x_k = u_k \\ \partial_k f(x) & \text{if} & x_k \in (l_k, u_k). \end{array} \right\}$$

This is a standard convergence condition for bound-constrained problems. The choice of $\tau = 10^{-10}$ guarantees good accuracy on this problem.

Table 1 presents the results obtained with Newton's method when DA is used to distribute the mesh and ADIC is used to compute the gradient and the Hessian matrix as described in Section II, but no mesh sequencing is used. For these results we used a Jacobi preconditioner.

Table 1. Performance results without mesh sequencing on 140 nodes. The symbol † is used if there is no convergence after 100 iterations.

Mesh	n_{iters}	Time (s)
71 × 71	6	0.58
141 × 141	8	1.45
281 × 281	10	2.85
561 × 561	21	9.34
1121 × 1121	†	†
2241 × 2241	†	†
4481 × 4481	†	†

The results in Table 1 show that the number of iterations grows as the mesh is refined but that for the finest meshes we terminate Newton's method after $n_{\text{iters}} = 100$ iterations. Although Newton's method is mesh invariant, the starting point is assumed to be in the region of quadratic convergence. A more careful analysis of these results, however, shows that the starting point defined by function (6) is not in the region of quadratic convergence for any of the grids.

We now consider the use of mesh sequencing. We start with a mesh with 71×71 grid points and, at each stage, use linear interpolation on the coarse mesh solution to obtain the starting point for the fine mesh. Thus, an $n_x \times n_y$ coarse mesh becomes a fine $(2n_x - 1) \times (2n_y - 1)$ mesh. We use the same termination condition (7) at each level of refinement.

The results in Table 2 show that, after the solution is obtained on the coarsest mesh, the number of iterations per level is either two or three. This is the desired behavior for mesh sequencing. The solution times per level also grow, but only linearly.

Table 2. Performance results with mesh sequencing on 140 nodes.

Mesh	n_{iters}	Time (s)	Cum. Time (s)
71 × 71	6	0.58	0.58
141 × 141	3	0.44	1.02
281 × 281	2	0.52	1.54
561 × 561	2	1.31	1.85
1121 × 1121	2	5.51	7.36
2241 × 2241	2	19.5	26.9
4481 × 4481	2	189	216

The results in this section are interesting because they contradict the folklore that convex problems are easy. For the obstacle problem we have shown that we cannot obtain good performance if we neglect the mesh structure. With the use of mesh sequencing, performance improves dramatically, and we obtain scalable results.

IV. Concluding Remarks

We have presented preliminary results for TAO on mesh-based problems for an obstacle problem for a minimal surface problem. A more extensive study of the performance of TAO algorithms on mesh-based problems is planned.

Version 1.6 of TAO includes all the features that we have discussed, in particular, mesh sequencing via Distributed Arrays (DA), and generation of gradients of mesh functions with ADIC. The TAO software[2] includes source code, documentation, tutorials, and sample problems.

Acknowledgments

Work supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1]Benson, S., McInnes, L. C., Moré, J. J., and Sarich, J., "TAO Users Manual," Technical Memorandum ANL/MCS-TM-242 (Revision 1.4), Argonne National Laboratory, Argonne, Illinois, 2002.

[2]"TAO," See `www.mcs.anl.gov/tao`.

[3]Dontchev, A. L., Hager, W. W., and Veliov, V. M., "Uniform convergence and mesh independence of Newton's method for discretized variational problems," *SIAM J. Control Optim.*, Vol. 39, 2000, pp. 961–980.

[4]Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F., "PETSc 2.0 Users Manual," Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, Argonne, Illinois, 1998.

[5]"PETSc," See `www.mcs.anl.gov/petsc`.

[6]Bischof, C., Roh, L., and Mauer, A., "ADIC: An Extensible Automatic Differentiation Tool for ANSI-C," *Software — Practice and Experience*, Vol. 27, No. 12, 1997, pp. 1427–1456.

[7]"ADIC," See `www.mcs.anl.gov/adic`.

[8]Lin, C.-J. and Moré, J. J., "Newton's method for large bound-constrained optimization problems," *SIAM J. Optim.*, Vol. 9, 1999, pp. 1100–1127.

[9]"TRON," See `www.mcs.anl.gov/~more/tron`.

[10]Lin, C.-J. and Moré, J. J., "Incomplete Cholesky factorizations with limited memory," *SIAM J. Sci. Comput.*, Vol. 21, 1999, pp. 24–45.

[11]Rodrigues, J.-F., *Obstacle problems in mathematical physics*, North-Holland Publishing Co., Amsterdam, 1987, Notas de Matemática [Mathematical Notes], 114.

[12]Bank, R. E., Gill, P. E., and Marcia, R. F., "Interior point methods for a class of elliptic variational inequalities," *High Performance Algorithms and Software for Nonlinear Optimization*, edited by L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. Van Bloemen Waanders, Springer-Verlag, 2003, pp. 218–235.

[13]Griewank, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, 2000.

[14]Corliss, G., Faure, C., Griewank, A., Hascoët, L., and Naumann, U., editors, *Automatic Differentiation: From Simulation to Optimization*, Computer and Information Science, Springer, New York, 2001.

[15]Abate, J., Benson, S., Grignon, L., Hovland, P. D., McInnes, L. C., and Norris, B., "Integrating AD with Object-Oriented Toolkits for High-performance Scientific Computing," *Automatic Differentiation: From Simulation to Optimization*, edited by G. Corliss, C. Faure, A. Griewank, L. Hascoët, and U. Naumann, Computer and Information Science, Springer, New York, 2002, pp. 173–178.

[16]Hovland, P., Lee, S., McInnes, L., Norris, B., and Smith, B., "Challenges and opportunities in using automatic differentiation with object-oriented toolkits for scientific computing," *High Performance Algorithms and Software for Nonlinear Optimization*, edited by L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. Van Bloemen Waanders, Springer-Verlag, 2003, pp. 133–147.

[17]Curtis, A. R., Powell, M. J. D., and Reid, J. K., "On the Estimation of Sparse Jacobian Matrices," *J. Inst. Math. Appl.*, Vol. 13, 1974, pp. 117–119.

[18]Benson, S., McInnes, L. C., and Moré, J. J., "A case study in the performance and scalability of optimization algorithms," *ACM Transactions on Mathematical Software*, Vol. 27, 2001, pp. 361–376.