Massachusetts Institute of Technology

18.337 Parallel Computing

Parallel PDE-constrained Optimization

Chad Lieberman

May 16, 2008

1 Introduction

The efficient solution of large-scale systems resulting from the discretization of partial differential equations (PDE) are of interest to engineers in many fields. Over the past several decades, as processors became faster, the time-to-solution for a given large-scale system has decreased thereby permitting the solution of larger and larger systems. It is human nature to desire the solution of systems larger than those we can reasonably solve today. Indeed, engineers will likely continue to pose simulation, analysis, or control problems over increasingly resolved and larger domains. The result is gigantic linear systems which must be solved by a machine.

While model reduction algorithms have been invented for solving linear systems resulting from PDEs, one can always invent a computational domain or set of physics for which the reduced-order model is too large to solve. Nevertheless, model reduction is one method by which large-scale systems can be solved faster. Indeed, they are employed in both real-time and multi-query settings. However, some model reduction algorithms consist of two phases: an offline phase consists of obtaining a basis for projection (often by collecting snapshots of the full-order system) *a priori* and projecting the full-order system; and an online phase where the reduced-order model is deployed. The offline phase can be very expensive since it involves the full-order solution of the large-scale dynamical system.

Proper orthogonal decomposition (POD) model reduction utilizes the method of snapshots to build the basis for projection. It is well known that the applicability of the reduced-order model depends highly on the samples, i.e. the snapshots. How to sample the parameter-state input-output map of the system to build these bases has no best answer. For that reason, model reducers implement heuristics to decide how to sample.

One heuristic method is the greedy algorithm originally proposed in [29]. The greedy algorithm chooses the sample points by solving a PDE-constrained optimization problem. At each iteration we obtain a sample

by choosing the parameter for which the error in output between the full-order and current reduced-order models is (locally) maximal. Although we hope to find the global maximum over parameter space, the objective function is rarely convex; therefore, we can only obtain a locally maximal solution at every iteration. This algorithm for sampling to produce reduced-order models has been successfully implemented in [2, 9].

Parallelization of the resulting PDE-constrained optimization problem is the topic of the current work. While ongoing research in this field is conducted, we focus specifically on one particular problem implementation through which many of the important topics in parallelism arise. In the next section, we introduce the application of interest and formulate the problem. Section 3 discusses the opportunities for parallelism throughout the application of the greedy sampling algorithm to the problem of Section 2. While the discussion is specific to the present problem, we highlight the generality of the proclamations and conclusions.

2 Formulation

In this section, we formulate the model reduction problem using the greedy-based POD technique. First, we introduce the governing equations of porous media flow, the application of interest. Then, proper orthogonal decomposition (POD) is described in the context of projection-based model reduction techniques. Finally, the greedy sampling algorithm is proposed for obtaining efficient samples from parameter space.

2.1 Application

Let Ω be the Lipschitz continuous domain with boundary Γ consisting of one part Γ_{N} on which Neumann (natural) boundary conditions are applied and one Γ_{D} on which Dirichlet (essential) boundary conditions are imposed. We required that $\Gamma_{\mathrm{N}} \cup \Gamma_{\mathrm{D}} = \Gamma$ but that they do not intersect, i.e. $\Gamma_{\mathrm{N}} \cap \Gamma_{\mathrm{D}} = \emptyset$. We call $u(\mathbf{x}; K)$ the pressure head in the domain associated with hydraulic conductivity $K(\mathbf{x})$. Let $f(\mathbf{x})$ be the source, $h(\mathbf{x})$ be the flux on the Neumann boundary, and $g(\mathbf{x})$ be the prescribed pressure head on the Dirichlet boundary. Then, for a given parameter field K, the pressure head is obtained by solving

$$\begin{aligned} -\nabla K \cdot (\nabla u) &= f \quad \text{in } \Omega, \\ K \nabla u \cdot n &= h \quad \text{on } \Gamma_{\mathrm{N}}, \\ u &= g \quad \text{on } \Gamma_{\mathrm{D}} \end{aligned}$$

where n is the outward-pointing unit normal.

For ease of explanation, we study only the problem with homogeneous essential boundary conditions $g = 0$ in this work. We note, however, that a novel approach to the model reduction algorithm for

inhomogeneous Dirichlet conditions is developed in [21]. Under these assumptions and given an interpolation scheme, the governing equations may be written in variational form and discretized resulting in the following linear system

$$\mathbf{A}(K)\mathbf{u} = \mathbf{f} \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the stiffness matrix, $\mathbf{f} \in \mathbb{R}^N$ is the load vector, and $\mathbf{u} \in \mathbb{R}^N$ is a vector of unknown nodal pressure head values. As mentioned, engineers are interested in the solution of systems like (1) for which $N > 10^6$.

While such systems can be solved with a single processor today, reduced-order models are required in the real-time and multi-query settings. For real-time application, the need is obvious: such systems simply cannot be solved quickly enough. The multi-query setting often pertains to some probabilistic systems analysis, e.g. Bayesian inference. In that case, solution of the model is required for many different values of the parameter, and therefore utilization of the full-order model is too slow.

2.2 POD model reduction

Proper orthogonal decomposition (POD) model reduction is a projection-based technique for reducing the dimensionality of the linear system (1). By identifying a subspace of the parameter-state input-output map, we are able to project the full-order system onto a lower-dimensional space. There, we solve the system for modal coefficients of the basis vectors and reconstruct the full-order solution by the associated linear combination. Let the parameter basis $\mathbf{P}$ and the state basis $\mathbf{V}$ be given, then the reduced-order model

$$\mathbf{A}_r\mathbf{u}_r = \mathbf{f}_r \tag{2}$$

is obtained by Galerkin projection. Here, $\mathbf{A}_r = \mathbf{V}^T\mathbf{A}(\mathbf{P}\mathbf{p}_r)\mathbf{V} \in \mathbb{R}^{n\times n}$ is the reduced stiffness matrix, $\mathbf{u}_r \in \mathbb{R}^n$ is the reduced state, and $\mathbf{f}_r = \mathbf{V}^T\mathbf{f} \in \mathbb{R}^n$ is the projected load vector where $n \ll N$. The small dense linear system (2) is solved for the modal coefficients $\mathbf{u}_r$ and then the approximation to the full-order solution is obtained $\tilde{\mathbf{u}} = \mathbf{V}\mathbf{u}_r$. The argument of the stiffness matrix is the restriction of the full parameter $\mathbf{p}$ to the subspace span($\mathbf{P}$). In this case, we employ simultaneous parameter-state model reduction; that is, since the parameter K resides on the mesh, it is very high dimensional. Since the goal is the solution of an optimization problem (inverse, control, etc.) over the feasible space of the parameter, we reduce the parameter as well.

2.3 Tensor vector product

One computation of the discrete governing equations deserves further explanation. The stiffness matrix $\mathbf{A}$ is a matrix-valued function of the parameter $\mathbf{p}$ through what I call the tensor-vector product. Without

derivation, the weak form of the governing PDE is

$$\int_\Omega K\nabla u \cdot \nabla v \, d\Omega = \int_\Omega f v \, d\Omega + \int_{\Gamma_\mathrm{N}} h v \, d\Gamma, \quad \forall v \in H_0^1(\Omega). \tag{3}$$

When the left hand side is discretized by interpolating the test and trial functions with linear nodal basis ϕ_i $(i = 1, 2, \ldots, N)$, the resulting stiffness matrix becomes a linear function of the coefficients of the parameter vector $\mathbf{p} = \mathbf{K}$. It is constructed via the tensor-vector product $\mathbf{A}(\mathbf{p}) = \mathbf{T} \otimes \mathbf{p}$ where

$$\mathbf{T}_{i,j,k} = \int_\Omega \phi_k \nabla\phi_i \cdot \nabla\phi_j \, d\Omega \tag{4}$$

and

$$\mathbf{T} \otimes \mathbf{p} = \sum_{k=1}^{N} p_k \mathbf{T}_{:,:,k}. \tag{5}$$

That is, the tensor vector product is the linear matrix expansion in the elements of the vector. Note that the tensor $\mathbf{T} \in \mathbb{R}^{N\times N\times N}$ is very sparse because of the compact support of the basis functions. The structure of the tensor is leveraged in the serial tensor-vector product code, but other work-arounds are required for parallel implementation. This computation is at the heart of the PDE-constrained optimization problem we pose in the next section. Current progress on parallel algorithms for its computation are expounded in Section 4.

2.4 Greedy sampling

We now describe one method for obtaining the bases $\mathbf{P}$ and $\mathbf{V}$ for projection-based model reduction. We use the method of snapshots to construct the bases. However, the question remains: how do we choose the samples? One method is the greedy algorithm. We pose it here in the residual formulation described in [4]. While obtaining the parameter field which maximizes the residual is not the same as maximizing the error in output between the full- and reduced-order models, it has the advantage of not depending on a solution of the full-order system. Instead, only the full-order stiffness matrix-vector product is required. Since the greedy sampling algorithm requires at each iteration the solution of the PDE-constrained optimization problem, it is a complex optimization problem whose solution time requires speed-up.

At each iteration of the algorithm, we must solve the following optimization problem for $\mathbf{p}^*$

$$\mathbf{p}^* = -\frac{1}{2}\arg\min \|\mathbf{r}\|_2^2 \;=\; -\frac{1}{2}\|\mathbf{C}(\mathbf{A}(\mathbf{p})\mathbf{V}\mathbf{u}_r - \mathbf{f})\|_2^2 \tag{6}$$

$$\text{subject to} \quad \mathbf{P}^T\mathbf{M}\mathbf{P}\mathbf{p}_r \;=\; \mathbf{P}\mathbf{M}\mathbf{p} \tag{7}$$

$$\mathbf{A}_r\mathbf{u}_r \;=\; \mathbf{f}_r. \tag{8}$$

where $\mathbf{C}$ is a matrix relating the residual to the output of interest. The constraints require that the reduced

parameter is the same as the full parameter in some mass-matrix weighted integrated sense and the residual measures the extent to which the solution approximated by the reduced-order model satisfies the full-order linear system.

We solve the optimization problem (6) by forming the Lagrangian and utilizing adjoints to compute the gradient of the objective function with respect to the entries in the parameter $\mathbf{p}$. These gradients (and the objective function evaluation) are passed to a suitable solver in MATLAB, e.g. using `fmincon`. Although the analytical Hessian is not suitable for computation in the same routine, we supply it by finite differences in order to utilize `fmincon`'s large-scale algorithm: trust-region interior-reflective Newton. It is the computation of the objective function, the gradient evaluation by adjoints, and the finite differencing of the Hessian where we hope to gain speed-up. Since the optimization routine requires repeated calls of these functions, and since the optimization must take place dozens of times to construct the reduced bases, speed-up of this routine is required for timely assessment of the resulting reduced-order model.

2.4.1 Objective function

The objective function needs to be evaluated for different parameter vectors $\mathbf{p}$ during the course of optimization. Given $\mathbf{p}$, the current $\mathbf{P}$ and $\mathbf{V}$, $\mathbf{f}$ and a method to compute $\mathbf{A}(\cdot)$, we must carry out the following computations: (i) project $\mathbf{p}$ onto the reduced-parameter subspace span($\mathbf{P}$) to obtain $\mathbf{p}_r$; (ii) assemble the stiffness matrix $\mathbf{A}(\mathbf{P}\mathbf{p}_r)$; (iii) construct the reduced-order model $\mathbf{A}_r\mathbf{u}_r = \mathbf{f}_r$; (iv) solve for $\mathbf{u}_r$; and (v) evaluate the 2-norm of the residual. In serial this might be done with Algorithm 1.

Algorithm 1 Serial Objective Function Evaluation $F = \texttt{objfun}(\mathbf{T}, \mathbf{M}, \mathbf{P}, \mathbf{V}, \mathbf{p}, \mathbf{f}, \mathbf{C})$

$\mathbf{p}_r = (\mathbf{P}^T\mathbf{M}\mathbf{P})^{-1}(\mathbf{P}^T\mathbf{M}\mathbf{p})$;
$\mathbf{A} = \mathbf{T} \otimes \mathbf{p}$;
$\mathbf{A}_r = \mathbf{V}^T(\mathbf{T} \otimes (\mathbf{P}\mathbf{p}_r))\mathbf{V}$;
$\mathbf{f}_r = \mathbf{V}^T\mathbf{f}$;
$\mathbf{u}_r = \mathbf{A}_r^{-1}\mathbf{f}_r$;
$\mathbf{r} = \mathbf{C}(\mathbf{A}\mathbf{V}\mathbf{u}_r - \mathbf{f})$;
$F = -\frac{1}{2}\|\mathbf{r}\|_2^2$;

2.4.2 Gradient

Consider now computing the gradient of the objective function with respect to the parameter $\mathbf{p}$, i.e. $\nabla_{\mathbf{p}} F \in \mathbb{R}^N$. The gradient is computed via adjoints and elimination. The somewhat lengthy derivation is omitted here. We just present the algorithm.

Algorithm 2 Serial Gradient Evaluation $\mathbf{g} = \texttt{grad}(\mathbf{T}, \mathbf{M}, \mathbf{P}, \mathbf{V}, \mathbf{A}_r, \mathbf{u}_r, \mathbf{C})$

$\lambda_h = \mathbf{A}_r^{-1}(\mathbf{C}\mathbf{V}^T\mathbf{A}\mathbf{r})$;

$\mathbf{f}_k = -\lambda_h(\mathbf{T} \otimes \mathbf{P})\mathbf{V}\mathbf{u}_r;$
$\lambda_k = (\mathbf{P}^T\mathbf{M}\mathbf{P})^{-1}\mathbf{f}_k;$
$\mathbf{Z} = (\mathbf{T} \otimes \mathbf{I})\mathbf{u}_r;$
$\mathbf{g} = -\mathbf{C}\mathbf{Z}\mathbf{r} + \mathbf{M}\mathbf{P}\lambda_k;$

2.4.3 Hessian

As mentioned before, the analytical Hessian is far too complicated to be worth deriving. In fact, it might be much more computationally intensive to compute directly (given the analytic formula) than to get a good approximation by finite differences. We provide a routine to `fmincon` which computes the Hessian-vector product. Then, the large-scale algorithm can be utilized. Here, we write the algorithm in one-sided difference form. Note that a central-difference would yield a more accurate result, but may not be worth the expense.

Algorithm 3 Serial Hessian-vector Product Evaluation $\mathbf{h} = \texttt{hessvec}(\mathbf{v})$

$\mathbf{h} = \mathbf{0}$;
for i=1:N **do**
  $\tilde{\mathbf{p}} = \mathbf{p} + \epsilon\mathbf{e}_i$;
  $\tilde{\mathbf{g}} = \texttt{grad}(\tilde{\mathbf{p}})$
  $\mathbf{h} = \mathbf{h} + (\mathbf{v}(i)/\epsilon)(\tilde{\mathbf{g}} - \mathbf{g})$;
end for

3 Parallelism

Enter parallel computing. We use Star-P, a software packge developed by Interactive Supercomputing, Inc. which allows users to perform parallel computations on an HPC server from a MATLAB front end. In this section, we highlight the opportunities in Algorithms 1–3 to be taken advantage of on multi-processor clusters.

3.1 Concurrency

We first investigate the concurrency in the algorithms. *Concurrency* exists in an algorithm where operations are data and task independent. In particular, to find concurrency, we must identify the data dependence in each algorithm.

Consider first Algorithm 1. Computation of the residual requires three sources of information: (i) input data ($\mathbf{f}$ and $\mathbf{C}$); (ii) full-order stiffness matrix $\mathbf{T} \otimes \mathbf{p}$; and (iii) reduced-order solution $\mathbf{u}_r$. Each of these components can be computed independent of the others. That is, we envision three paths from start to end. At the start, all of the data comes in. Along each branch we compute (or transfer data). And at

the end, the norm of the residual is computed. This would amount to the so-called multi-input multi-data (MIMD) program. While MIMD programs represent maximum efficiency for many algorithms by taking advantage of all concurrency, they are notoriously difficult to implement.

In Algorithm 2 we see some similar aspects. The computations of the two adjoints λ_h and λ_k are independent, as is the computation of $\mathbf{Z}$. They are all collected in the final line where the gradient is constructed form these components.

Algorithm 3 demonstrates vulnerable concurrency. We can take advantage of the concurrency disguised within this for loop to compute the Hessian-vector product. Each of the iterations of the for loop are completely independent from each other. Therefore, we have true task parallelism, i.e. if we had N processors we could imagine computing one part of $\mathbf{h}$ on each processor and summing the result across processors.

3.2 Data parallelism

There is plenty of data parallelism to be taken advantage of in these algorithms. For example, notice that Algorithms 1–2 both require the computation of the tensor-vector product $\mathbf{T} \otimes (\cdot)$. For large problems, the tensor $\mathbf{T}$ will not fit on most clients; therefore, at the very least it must reside on the server.

There are many methods by which the generic tensor-vector product and tensor-matrix product of the algorithms could be computed. However, we must consider the unfortunate fact that neither Matlab nor Star-P provide support for multi-dimensional sparse data structures. In serial, the tensor can be stored and utilized efficiently in a modified column-compressed format because the nonzero pattern of the result of any tensor-vector product is known *a priori*. The format is constructed so that the computation of any nonzero in the result amounts to a dot product over data which already resides next to each other in memory. In parallel, the algorithm, and even the storage of the tensor must be reconsidered. See Section 4. In addition, the matrix-vector products, etc. can also be implemented in parallel, but the speedup will not be great for the size of the class of problems we consider. We recognize, however, that we may have to do this to minimize communication.

Once we store the tensor on the server, it is in our best interest to compute on the server until a small result can be passed back to the client. This is one important consideration of communication.

3.3 Communication

Now we consider the communication between the client and the server, and between the different processors of the server. Based on preliminary experimentation, it is the former which is the most expensive as the data is sent over the MIT network. For this reason, we want to limit the size of data and the frequency of transmissions of data communicated between the client and the server. While the latter communication we keep in the back of our heads, it is not the dominant consideration here.

There seem to be two approaches for implementation here, each requiring different communication. Suppose, for example, that each call to `fmincon` could be run on the server. Then, in principle, the data required for one call to fmincon could be passed to the server, the optimization problem would be solved with communication only among the processors on the server, and then the solution returned back to the client. As a first try, however, it may be worthwhile to view the computation of the objective function, gradient, and hessian-vector product as an inner loop to be computed on the server. That is, as `fmincon` requires, these three components can be computed individually on the server.[1]

The data required to compute the objective function evaluation would be instantiated on the server. If the objective function is required, it is computed on the server, and then passed back to the client (where `fmincon` is running). The communication required to do this is the outgoing message of $\mathbf{p} \in \mathbb{R}^N$ and the incoming message $F \in \mathbb{R}$.

If the gradient is requested, the required data already resides on the server (from the objective function evaluation). Therefore, the gradient must be computed on the server and then transferred to the client. This requires another message of size N. Likewise, the hessian-vector product results in a vector $\mathbf{h} \in \mathbb{R}^N$ – another N-element message.

Note that if `fmincon` runs on the server, the only message required is the solution back to the client – one message of length N.

4 pp_STIRNCG

Instead of hacking the more complicated `fmincon` code, I modified a freely-available optimizer called `STIRNCG` developed by Tan Bui-Thanh for his doctoral thesis at MIT in 2006–2007. `STIRNCG` is the Cadillac of optimizers for large-scale bound-constrained problems. In optimization, the quality of a code can often be discerned from the number of adjectives accompany it. In the case of `STIRNCG`, the adjectives all appear in the name. In particular, `STIRNCG` is a Newton solver, i.e. it uses the Hessian directly (either by finite difference or analytic user-supplied) and not an approximation thereof (e.g. with BFGS update). The optimizer also uses a subspace method to solve for the Newton direction in a lower-dimensional manifold, thereby saving computation on that portion of the inner loop. In addition, `STIRNCG` employs inexact conjugate gradients (CG) to solve $Hp = -g$ for the descent direction p in order to avoid oversolving. Once the descent direction is ascertained, a trust-region subproblem is solved in the subspace. If the step provides sufficient reduction, it is taken; if not, the trust-region radius is decreased and the trust-region subproblem is solved once again. Once a step is taken, if it leads outside of the bound constraints, it is reflected back by an interior-reflective scheme. In summary, `STIRNCG` is a subspace, trust-region, interior-reflective, Newton

[1] Note that `fmincon`, or any suitable optimization routine, requires either the objective function, the objective function and the gradient, or the objective function, gradient, and hessian. For example, it never requires the hessian, and not the objective function.

solver using inexact CG to solve for the descent direction.

`STIRNCG` also follows the conventions of `fmincon` in that a user specifies two separate functions: one for the objective function and gradient computation and one for the Hessian-vector product. One may also pass a preconditioning routine, but I do not take advantage of that feature here. However, the main point is that `STIRNCG` maintains the ease of use of `fmincon` – a consistency I extend to star-p. In effect, I instrumented `STIRNCG` to run in parallel using star-p. That is, a user of the `STIRNCG` optimizer does not have to make any modifications to his/her code in order to run in parallel. Instead of calling `STIRNCG`, one calls `pp_STIRNCG` instead and provides a few additional arguments to control data passing between modules of the optimizer. In the future, I will utilize some more basic data passing structures/concepts to avoid the need for these additional arguments. Even they can be eliminated! That is, to say, with some streamlining and refactoring, `pp_STIRNCG` can be substituted directly into a `STIRNCG` application and the user will have parallel performance immediately. This convenience eliminates the need for extended periods of code development complete with all the associated headaches and frustrations.

A typical `STIRNCG` optimization iteration is embodied in the flowchart in Figure 1. The algorithm

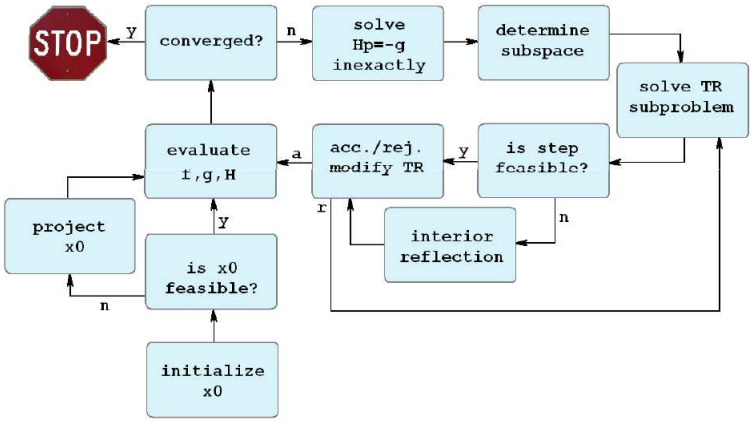

Figure 1: The flowchart for a trust-region based optimization algorithm (e.g. STIRNCG).

begins with an initial iterate. If the location is not in the feasible set, it is projected there. Then, at each iteration, the objective function f and gradient g are computed. The convergence criteria are tested; if not satisfied, a new search direction is computed by solving $Hp = -g$ where H is the Hessian matrix of second derivatives and p is the search direction. In `STIRNCG`, this linear system is solved inexactly with CG. Note that it is not necessary, in particular when we are far from the solution, to solve this system exactly. We just require a direction of sufficient descent. After the search direction is obtained, a trust-region subproblem is constructed along that direction. This subproblem is an inequality constrained quadratic program (QP) based on the first three terms of a Taylor series expansion of f about the current iterate. Once the trust-region subproblem is solved, we check that the step provides sufficient decrease of f. If it does, we accept the step and expand the trust-region. If not, we reject the step, decrease the trust-region

and resolve the trust-region subproblem with tighter constraints. The process is repeated until convergence is achieved.

In PDE-constrained optimization, the objective function, gradient, and Hessian-vector product computations are the bottleneck. Generally, these evaluations require at least one simulation which could take months to solve (even in parallel!). In comparison, the mechanics of the optimization solver require little or no cost. Therefore, the focus in developing `pp_STIRNCG` was directed toward parallelizing these three aspects: objective function, gradient, and Hessian-vector product evaluations. This is carried out using star-p.

We also, however, want everything to happen in parallel (as mentioned before). The current version of `fmincon` available from ISC is `ppfmincon` which computes gradients and Hessians by finite-differencing instantiates `fmincon` on the client and continually passes data back and forth between the client and server. For slow connections (e.g. to starp.csail.mit.edu), such an implementation makes the parallelism useless except for extremely large problems. Instead, I take a different approach. The concept is that the optimization routine requires some amount of input data. Then, once it has that data, it can compute away until it obtains the solution, and should not be talking to the client during any of that period. Therefore, in the wrapper to `pp_STIRNCG`, the necessary data is first placed on the server. From there, all computations take place on the back end and only the solution is returned to the client. So we pay a cost upfront to transfer the data, but eliminate the constant (and unnecessary) client/server communication. In fact, if the data is pre-computed it can be transferred to the back end directly by FTP, so even this overhead can be dramatically reduced.

5 Results

The parallel `STIRNCG` code I wrote was tested on several model problems. We consider the spatially and temporally discrete system

$$Ax = Bu \quad y = Cx \tag{9}$$

where $A \in \mathbb{R}^{N\times N}$, $x \in \mathbb{R}^N$ is the state, $B \in \mathbb{R}^{N\times q}$ is the input transition matrix, $u \in \mathbb{R}^q$ are the inputs. The output equation contains the outputs $y \in \mathbb{R}^d$ and the output matrix $C \in \mathbb{R}^{d\times N}$.

We wish to find the inputs u such that the output matches some data we have. That is, we wish to solve the optimization problem

$$\begin{aligned} u^* = \arg\min \mathcal{J} &= \frac{1}{2}\|y - y_D\|^2 + \frac{1}{2}\beta\|u\|^2 \\ \texttt{subject to} \quad Ax &= Bu \\ y &= Cx \end{aligned}$$

where y_D are the target data and β is a regularization parameter necessary to make the problem well-posed. In this case the regularization term penalizes high-energy inputs. That is, we wish to match the outputs but penalize those inputs requiring a lot of power (e.g. in an optimal control problem).

The details of the solution of this optimization problem are not the focus here. Instead, we are interested in the scalability of `pp_STIRNCG` with the size of the model N and number of processors np. To that end, a parameter study was completed over problem sizes $N = 1K, 2K, 3K, 4K$ and processors $np = 1, 4, 8, 12$. All of the runs were completed on the star-p cluster at CSAIL (`starp.csail.mit.edu`) and there was no other traffic during the runs.

In Figure 2, the speedups are plotted per iterate of the optimization. For each problem, the same initial conditions were used, and indeed, the same steps were taken by the optimizer. Therefore, the speedup we see should be isolated to the contribution of parallelization.

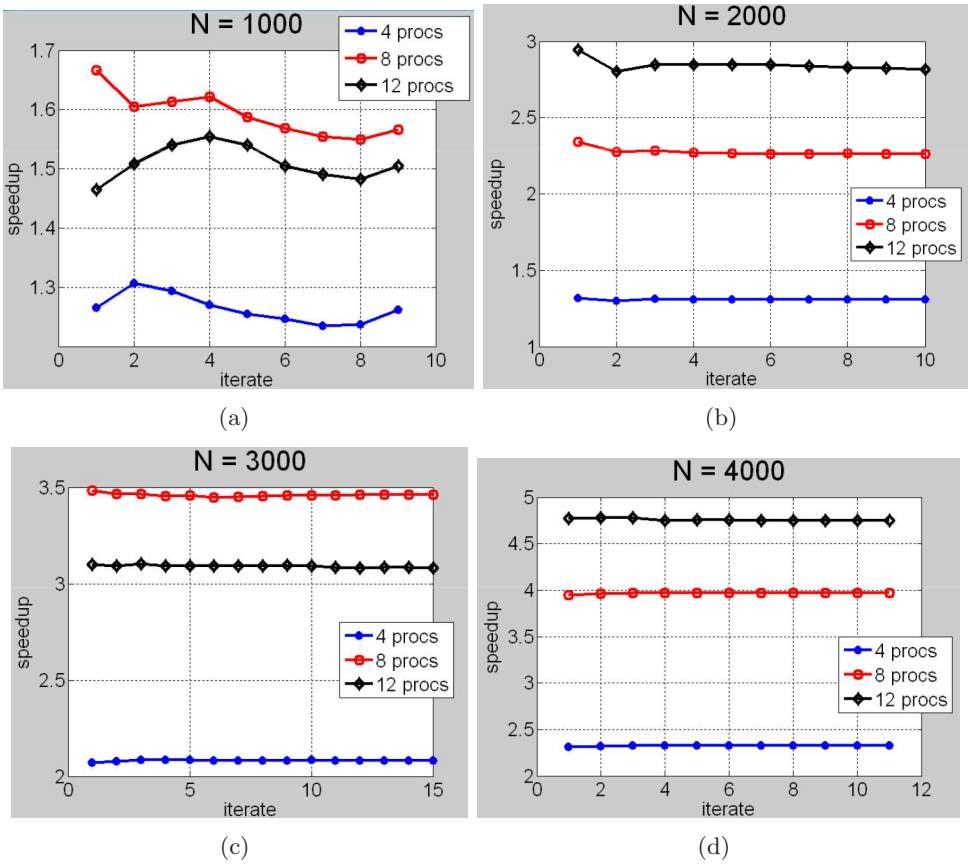

Figure 2: Speedup results for the model problems for $np = 4, 8, 12$.

The speedups are computed by dividing the serial ($np = 1$) time by the parallel time in each case. It is first important to note that we do achieve speedups exceeding unity – that is, parallelization has helped. Then, we consider the scalability. For embarassingly parallel implementations, the speedup should go linearly with the number of processors. This should be taken as the upper-limit, the best we can do. In

our case, we do not obtain linear speedup, but we do get good results. Perhaps more importantly, however, is that the speedup increases as we increase the problem size. Although time and storage limitations forced small model problems for testing, we are already seeing that the parallelism will become better and better as we increase N – this is exactly what we want.

There are two oddities in the results which are worth pointing out. For problems of size $N = 1000$ and $N = 3000$, we see that using 12 processors actually gives us decreased performance from that which used 8 processors. I cannot explain this aberration in the data. This is one disadvantage of the star-p mindset – that everything is abstracted from the user – but the easy implementation and portability compensate entirely for that.

In summary, we have achieved almost $np/2$ speedup for the model problem $N = 4000$. The author believes that is a significant speedup for a computation as complex as an optimization routine. Not only do we achieve parallel speedup, but this can be taken advantage of by anyone with existing Matlab codes used with some of the optimizers in the Optimization toolbox (e.g. `fmincon`). All that needs to be supplied is some additional arguments to the optimizer call. Building a parallel optimization toolbox on top of star-p is the goal – as star-p grows and becomes better and better, the toolbox automatically does as well.

6 Conclusion

We have presented and put into perspective the task of PDE-constrained optimization. Now that the simulation community has a good handle on numerical solution of PDEs, the logical next step is to optimize with these simulations in the inner loop. Many engineering companies are now doing exactly that. The optimization requires the solution of large-scale PDE simulations and their respective adjoints throughout the iteration. These solutions are very expensive; therefore, they were targeted for parallelization. A freely available optimizer tuned for such problems is `STIRNCG` developed by Tan Bui-Thanh (MIT). I instrumented this optimizer to compute everything in parallel; that is, to eliminate communication between the client and server, an aspect of ISC's `ppfmincon` which slows down performance dramatically. The result is a completely portable parallel optimizer for bound-constrained problems.

References

[1] V. Akcelik, G. Biros, O. Ghattas, J. Hill, D. Keyes, and B. van Bloemen Waanders, "Parallel algorithms for PDE-constrained optimization," Frontiers of Parallel Computing, SIAM 2006.

[2] O. Bashir, K. Willcox, O. Ghattas, B. van Bloemen Waanders, and J. Hill, "Hessian-based model reduction for large-scale systems with initial-condition inputs," Int. J. Numer. Meth. Engng 2008; 73:844–868.

[3] M. Bergmann, L. Cordier, and J-P Brancher, "Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model," Phys. Fluids 2005; 17-097101.

[4] T. Bui-Thanh, K. Willcox, and O. Ghattas, "Model reduction for large-scale systems with high-dimensional parametric input space," Journal of Scientific Computing 2008.

[5] J. Burkardt, M. Gunzburger, and H-C Lee, "POD and CVT-based reduced-order modeling of Navier-Stokes flows," Comput. Methods Appl. Mech. Engrg. 2006; 196:337–355.

[6] L. Daniel, O.C. Siong, L.S. Chay, K.H. Lee, and J. White, "A multiparameter moment-matching model-reduction approach for generating geometrically parameterized interconnect performance models." IEEE Transactions on CAD of Int. Circ. Systems, May 2004; 23(5):678–693.

[7] A.E. Deane, I.G. Kevrekidis, G.E. Karniadakis, and S.A. Orszag, "Low-dimensional models for complex geometry flows: Application to grooved channels and circular cylinders," Phys. Fluids, 1991; 3(10):2337–2354.

[8] P. Feldmann and R.W. Freund, "Efficient Linear Circuit Analysis by Pade Approximation via the Lanczos Process," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1995; 14:639–649.

[9] D. Galbally, "Nonlinear model reduction for uncertainty quantification in large-scale inverse problems: application to nonlinear CDR equation," Master's thesis, Dept. of Aero. & Astro., Massachusetts Institute of Technology, 2008.

[10] K. Gallivan, E. Grimme, and P. Van Dooren, "Pade Approximation of Large-Scale Dynamic Systems with Lanczos Methods," Proceedings of the 33rd IEEE Conference on Decision and Control, December 1994.

[11] E. Grimme, "Krylov Projection Methods for Model Reduction," PhD thesis, Coordinated-Science Laboratory, University of Illinois at Urbana-Champaign, 1997.

[12] S. Gugercin and A. Antoulas, "A survey of model reduction by balanced truncation and some new results," International Journal of Control, 2004; 77:748–766.

[13] M. Gunzburger, J. Peterson, and J. Shadid, "Reduced-order modeling of time-dependent PDEs with multiple parameters in the boundary data," Comput. Methods Appl. Mech. Engrg. 2007; 196:1030–1047.

[14] P. Holmes, J.L. Lumley, and G. Berkooz, "Turbulence, Coherent Structures, Dynamical Systems and Symmetry," Cambridge University Press, Cambridge, UK, 1996.

[15] K. Ito and S. Ravindran, "A reduced-order method for simulation and control of fluid flows," J. Comp. Phys. 1998; 143:403–425.

[16] G. Kerschen, J-C Golinval, A. Vakakis, and L. Bergmann, "The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: an overview," Nonlinear Dynamics 2005; 41:147–169.

[17] P. Krysl, S. Lall, and J.E. Marsden, "Dimensional model reduction in non-linear finite element dynamics of solids and structures," Int. J. Numer. Meth. Engng. 2001; 51:479–504.

[18] S. Lall, P. Krysl, and J.E. Marsden, "Structure-preserving model reduction for mechanical systems," Physica D: Nonlinear Phenomena 2003; 184:304–318.

[19] S. Lall, J.E. Marsden, and S. Glavaski, "A subspace approach to balanced truncation for model reduction of nonlinear control systems," Int. J. Robust and Nonlinear Control 2002; 12:519–535.

[20] J. Li and J. White, "Low rank solution of Lyapunov equations," SIAM Journal on Matrix Analysis and Applications, 2002; 24(1):260–280.

[21] C. Lieberman and K. Willcox, "Model reduction for large-scale systems with inverse nonlinearities and inhomogeneous essential boundary conditions," Work-in-progress, 2008.

[22] T. Penzl, "Algorithms for model reduction of large dynamical systems," Linear Algebra and its Applications, June 2006; 415(23):322–343.

[23] S. Ravindran, "A reduced-order approach for optimal control of fluids using proper orthogonal decomposition," Int. J. Num. Meth. Fluids 2000; 34:435–448.

[24] C. Rowley, T. Colonius, and R. Murray, "Model reduction for compressible flows using POD and Galerkin projection," Physica D: Nonlinear Phenomena 2004; 189:115–129.

[25] D. Ryckelynck, L. Hermanns, F. Chinesta, and E. Alercon, "An efficient *a priori* model reduction for boundary element models," Engineering Analysis with Boundary Elements 2005; 29:796–801.

[26] L. Silveira, M. Kamon, I. Elfadel, and J. White, "A coordinate-transformed Arnoldi algorithm for generating guaranteed stable reduced-order models of RLC circuits," Comp. Meth. in Appl. Mech. Engnrg., 199; 169:377–389.

[27] L. Sirovich, "Turbulence and the dynamics of coherent structures. Part 1: Coherent structures," Quarterly of Applied Mathematics, October 1987; 45(3):561–571.

[28] D.C. Sorensen and A.C. Antoulas, "The Sylvester equation and approximate balanced reduction," Linear Algebra and its Applications, 2002; 351352:671700.

[29] K. Veroy, C. Prudhomme, D. Rovas, and A. Patera, "A posteriori error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations," AIAA Paper 2003-3847, Proceedings of the 16th AIAA Computational Fluid Dynamics Conference, Orlando, FL, 2003.

[30] Q. Yu, J. Wang, E.S. Kuh, "Passive multipoint moment-matching model order reduction algorithm on multiport distributed interconnect networks," IEEE Transactions on Circuits and Systems–I: Fund. Theory and Appl. 1999; 46(1):140–160.