Comput Optim Appl (2014) 59:667–688
DOI 10.1007/s10589-014-9651-2

Efficient parallel solution of large-scale nonlinear dynamic optimization problems

Daniel P. Word · Jia Kang · Johan Akesson · Carl D. Laird

Received: 7 May 2013 / Published online: 19 April 2014

Abstract This paper presents a decomposition strategy applicable to DAE constrained optimization problems. A common solution method for such problems is to apply a direct transcription method and solve the resulting nonlinear program using an interior-point algorithm. For this approach, the time to solve the linearized KKT system at each iteration typically dominates the total solution time. In our proposed method, we exploit the structure of the KKT system resulting from a direct collocation scheme for approximating the DAE constraints in order to compute the necessary linear algebra operations on multiple processors. This approach is applied to find the optimal control profile of a combined cycle power plant with promising results on both distributed memory and shared memory computing architectures with speedups of over 50 times possible.

Keywords Dynamic optimization · Parallel computing · Collocation · Schur-complement decomposition · Parallel nonlinear optimization

D. P. Word · J. Kang
Artie McFerrin Department of Chemical Engineering, Texas A&M University, College Station, TX, USA

C. D. Laird (✉)
School of Chemical Engineering, Purdue University West Lafayette, IN, USA
e-mail: carllaird@purdue.edu

J. Akesson
Department of Automatic Control, Lund University, Lund, Sweden

J. Akesson
Modelon AB, Lund, Sweden

1 Introduction

Optimization of dynamic systems has proven to be an effective method for improving operation and profits in the chemical process industry [15,39,41,45,47]. The success of these methods has led to the continued growth of these systems to improve model rigor and increase the scope of the optimization problem, however the solution of very large-scale models remains challenging [22]. Concurrently, the advances in computing clock rates that we once took for granted have slowed dramatically. Computer chip design companies have instead focused on development of parallel computing architectures [49]. These new architectures require advanced algorithms to exploit their parallelism. Furthermore, the successful use of advanced solution approaches within industrial settings requires that these algorithms be interfaced with effective problem formulation tools. Modern object-oriented modeling languages allow for rapid creation of complex dynamic optimization problems and reduce the burden of model development, optimization problem formulation, and solver interfacing. These also ease the construction of complicated optimization problems, while making it easier to construct intractably large problems for serial algorithms. There is a need for the development of advanced parallel algorithms for dynamic optimization that can interface with modern modeling languages and utilize parallel computing architectures to efficiently solve these large-scale problems. This paper presents a decomposition approach for efficient, parallel solution of nonlinear dynamic optimization problems formulated using the Modelica-based JModelica.org platform [1] and the optimization package CasADi [4].

There are three common approaches for solution of dynamic optimization problems. With sequential techniques (also called control vector parameterization), the control variables are discretized, and the optimizer only sees these discretized degrees of freedom. An integrator then converges the model at each iteration with calculation of derivative information performed through various techniques including integrating sensitivity or adjoint equations. The integration of the model and calculation of the derivative information is often the dominant computational expense of this technique [23], and multiple approaches are being explored to improve solution times through parallelization of these computations [15,22,34,39]. On the other hand, simultaneous approaches discretize all problem variables, not just control variables, and include the discretized model as constraints in a large-scale optimization problem [7]. Pseudospectral methods discretize the model using global polynomials and collocation, with nodes calculated using Gaussian quadrature [5,16,19]. This approach may be suitable for problems with smooth solutions that can be reasonably approximated with polynomials of low degree [18]. For problems that have nonsmooth solutions or that are not well approximated with low degree global polynomials, orthogonal collocation on finite elements is a preferred discretization approach [13,28]. Here, the profiles are partitioned into finite elements and approximated using individual polynomials in each finite element. While the size of the optimization problem is significantly larger using the simultaneous approach, there is potential for improved performance since the simulation problem (represented by the discretized equality constraints) is solved simultaneously with the optimization problem. Even though such problems are very large, they are sparse and inherently structured as a result of the discretization.

Multiple shooting techniques combine aspects of both the sequential and simultaneous approaches. These techniques discretize time into multiple stages and the control variables are parameterized using a finite set of control parameters in each stage. The system is then solved on each stage, where each stage is an initial value problem. Equality constraints are added to enforce continuity between the stages [7].

In this paper, we consider efficient solution of dynamic optimization problems discretized using orthogonal collocation on finite elements. One hurdle in using simultaneous approaches in the past has been the burden on the modeler associated with the manual discretization of the model – a process that is typically tedious and error-prone. However, packages are available in modern modeling languages such as the Modelica-based JModelica.org platform [1,36], Pyomo [21], ACADO [24], and DynoPC [33] that allow straightforward declaration of dynamic equations and provide automatic discretization of these equations using direct collocation methods. This significantly reduces the burden on the modeler when using simultaneous solution approaches.

Nonlinear interior-point methods provide an excellent framework for specialized solution of these discretized dynamic optimization problems. These methods have proven to be effective tools to solve many very large-scale optimization problems [31,43,44,47]. The dominant computational expense in these methods usually lie in the solution of the augmented system, or KKT system, a linear system that results from the application of Newton's method to the primal-dual form of the KKT conditions of the barrier subproblem. The KKT system retains consistent structure from iteration to iteration, and various approaches can be utilized to reduce the time required to solve this system.

Amestoy et al. [2] developed a parallel distributed memory multifrontal approach for the solution of sparse linear equations that includes an asynchronous parallel algorithm for efficient numerical pivoting. This algorithm showed speedup of more than 7 on test problems, but algorithm performance was not evaluated using a large number of processors because suitable test problems were unavailable.

Scott [40] presented parallel general-purpose multi-frontal codes to solve large sparse systems of linear equations. These codes include the serial solution of a so-called interface problem, and the size and subsequent time for solution of this problem heavily influences the overall performance. This serial component limits the scalability of the approach to a relatively small number of processors, but the results still show a 5 times speedup on 8 processors compared with serial direct solvers.

Schenk and Gärtner [38] address issues to improve scalability and robustness of sparse direct factorization on shared memory multiprocessor architectures. To balance trade-off between performance and robustness, they employ block supernode diagonal pivoting. While this approach is not guaranteed to always be efficient, timing results are promising for some problems. Scale-up of this approach is limited by the fact that it is designed for shared memory architectures.

Kocak and Akay [29] explore a decomposition approach for solution of general linear systems using a Schur-complement. They investigate the use of an analysis step to determine problem structure that can be exploited using this decomposition and highlight that when exploiting problem structure for the Schur-complement decomposition, the amount of coupling between blocks makes a significant impact on algorithm performance. For problems lacking clear structure, determining how to effectively

decompose the problem is vital to maintain algorithm efficiency but can be computationally expensive. On the other hand, certain problem classes contain well-defined coupling by definition and offer an intuitive path for block structure decompositions.

An alternative to using parallel direct solvers that handle general problem structures is to focus on problem classes with a particular structure and develop algorithms that specifically exploit that structure. By requiring specific structure in problem formulations, the structure is predetermined so the cost of structural analysis can be avoided. This knowledge can then be exploited for problem level decomposition and parallel solution of these decomposed systems. Schur-complement methods have been used for many years to decompose and solve large-scale, structured, linear systems, and efficient parallel algorithms have been developed [29].

DeMiguel and Nogales [14] have established a theoretical relationship between bilevel decomposition algorithms and Schur-complement interior-point methods. This helps bridge the gap between the local convergence theory of bilevel decomposition algorithms and Schur interior-point methods. Additionally, this work shows how Schur-complement interior-point methods can be modified to allow solution of problems where the Schur-complement is not generally invertible [14]. [20] have applied robust optimization techniques to reparameterize linear discrete-time optimal state feedback problems as convex programs. A primal-dual interior-point solver is then used to solve this convex program. They highlight that it is straightforward to parallelize their algorithm if a Schur-complement decomposition is used to solve the linear KKT system at each iteration of the interior-point algorithm. This solution approach is shown to be efficient even though they do not implement their algorithm in parallel. [45] adapted the interior-point solver IPOPT to use a Schur-complement decomposition approach to solve the linear KKT system in parallel by exploiting block structure in multi-scenario parameter estimation problems. This approach has proven effective for solving several large-scale case studies [30,45]. [46] present a similar algorithm for optimal control under uncertainty. In [48] this algorithm is used to determine optimal control of a cryogenic air separation where demand and contractual obligation uncertainties are captured using a multi-scenario formulation. For these multi-scenario problems, individual scenarios can be decomposed into separate blocks where only variables that are common across multiple scenarios prevent the blocks from being fully independent. These coupling variables are permuted into the Schur-complement and dictate its size. The Schur-complement is solved in serial, and as the number of coupling variables and the size of the Schur-complement grows, this computational cost can become significant. However, for problems with a low number of coupling variables this approach shows significant performance improvements over full-space solution approaches [48].

In this paper we make use of the Modelica-based open source software JModelica.org [1] and CasADi [4], to transform high-level descriptions of nonlinear dynamic optimization problems into algebraic nonlinear programming problems through a direct collocation approach. This creates a block-banded structure in the KKT system where each finite element forms a block, and the system can be decoupled at finite element boundaries. The literature details a number of strategies exploiting this finite element block structure in serial [6,11,12,37]. When applying a nonlinear interior-point method to solve the optimization problem, the dominant computational expense

is the solution of the KKT system that must be solved at each iteration to produce the steps in the primal and dual variables. The block-banded structure is decomposed by forming a Schur-complement with respect to the state continuity equations. The size of the Schur-complement depends on the number of state variables and the number of processors used in the decomposition, making this approach most favorable for problems with significantly fewer state variables than algebraic variables.

Building on results in [32], we develop an interior-point algorithm for the solution of large-scale nonlinear dynamic optimization problems. We use a Schur-complement decomposition approach that exploits the block-structure inherent in the discretized optimization problem to allow solution of the problem in parallel. The performance of our decomposition algorithm is demonstrated on an optimal control problem for the start-up of a combined cycle power plant, where we achieve a speedup of over 50 times.

In Section 2 we describe the interior-point algorithm used to solve our optimization problems, and in Section 3 we describe the discretization strategy employed to transform dynamic problems into nonlinear programming problems. Section 4 shows how we decouple individual finite element blocks and use a Schur-complement decomposition to solve for the primal and dual variable steps in the interior-point algorithm. We describe our software implementation in Section 5 and outline the formulation of test problems in Section 6. We display the performance of our algorithm on these problems in Section 7. Conclusions and future work are summarized in Section 8.

2 Interior-point algorithm

The interior-point algorithm utilized in this work considers nonlinear problems of the form

$$\min \quad f(z) \tag{1}$$

$$\text{s.t.} \quad c(z) = 0 \tag{2}$$

$$\underline{z} \leq z \leq \overline{z}, \tag{3}$$

with n variables and m equality constraints, where $z \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are assumed to have continuous first and second derivatives. The vectors $\underline{z}$ and $\overline{z}$ are the set of lower and upper variable bounds for z respectively. This problem is solved using an interior-point method with a filter-based line-search based on that described in [42]. A detailed description of the algorithm and its derivation can be found there, and here we only describe the steps necessary to explain our parallel decomposition approach. As an interior-point method, variable bounds are removed from the constraints by adding a log penalty to the objective and forming the barrier subproblem,

$$\begin{aligned} \min \quad & f(z) - \mu \sum_{i=1}^{n} \ln(\overline{z}^{(i)} - z^{(i)}) - \mu \sum_{i=1}^{n} \ln(z^{(i)} - \underline{z}^{(i)}) \\ \text{s.t.} \quad & c(z) = 0, \end{aligned} \tag{4}$$

where μ is the barrier parameter for a single barrier iteration, and (i) denotes the i^{th} element of the vectors of length n.

The Lagrangian of the barrier subproblem (4) can then be written as,

$$\mathcal{L} = f(z) - \mu \sum_{i=1}^{n} \ln(\overline{z}^{(i)} - z^{(i)}) - \mu \sum_{j=1}^{n} \ln(z^{(j)} - \underline{z}^{(j)}) + \lambda^T c(z), \tag{5}$$

where λ is the vector of equality constraint multipliers. The first order optimality conditions are then,

$$\begin{aligned} \nabla_z \mathcal{L} = \nabla_z f(z) + \mu(\overline{Z})^{-1} e - \mu(\underline{Z})^{-1} e + \nabla_z c(z)\lambda = 0 \\ c(z) = 0, \end{aligned} \tag{6}$$

with $\overline{Z}$=diag$(\overline{z}-z)$ and $\underline{Z}$=diag$\big(z-\underline{z}\big)$. We form the primal-dual formulation by introducing the variables $\overline{d}$=$\mu[\overline{Z}]^{-1}e$ and $\underline{d}$=$\mu[\underline{Z}]^{-1}e$. The algorithm enforces $(\overline{z}-z)\geq 0$ and $(z-\underline{z})\geq 0$, which makes the new variables $\overline{d}, \underline{d} \geq 0$. Including these new variables gives the following system of equations:

$$\begin{aligned} \nabla_z \mathcal{L} = \nabla_z f(z) + \overline{d} - \underline{d} + \nabla_z c(z)\lambda = 0 \\ c(z) = 0 \\ \underline{Z}\,\underline{d} - \mu e = 0 \\ \overline{Z}\,\overline{d} - \mu e = 0. \end{aligned} \tag{7}$$

These equations are solved for a particular value of μ using a modified Newton's method. For each iteration k of the Newton's method, the following linear system must be solved:

$$\begin{bmatrix} \nabla^2_{zz}\mathcal{L}(z^k) & \nabla_z c(z^k) & -I & I \\ [\nabla_z c(z^k)]^T & 0 & 0 & 0 \\ \underline{D}^k & 0 & \underline{Z}^k & 0 \\ -\overline{D}^k & 0 & 0 & \overline{Z}^k \end{bmatrix} \begin{bmatrix} \Delta z^k \\ \Delta\lambda^k \\ \Delta\underline{d}^k \\ \Delta\overline{d}^k \end{bmatrix} = - \begin{bmatrix} \nabla_z f^k + \overline{d}^k - \underline{d}^k + \nabla_z c(z^k)\lambda \\ c(z^k) \\ \underline{Z}^k\underline{d}^k - \mu e \\ \overline{Z}^k\overline{d}^k - \mu e \end{bmatrix}. \tag{8}$$

Here, Δz^k, $\Delta\lambda^k$, $\Delta\underline{d}^k$, and $\Delta\overline{d}^k$ are the full steps for each of the respective variables, $\underline{D}^k$=diag$(\underline{d}^k)$, and $\overline{D}^k$=diag$(\overline{d}^k)$.

The augmented form, a symmetric system, is obtained by multiplying the third block row by $(\underline{Z}^k)^{-1}$, the fourth block row by $(-\overline{Z}^k)^{-1}$, and adding these rows to the first block row. This gives

$$\begin{bmatrix} H^k & \nabla_z c(z^k) \\ [\nabla_z c(z^k)]^T & 0 \end{bmatrix} \begin{bmatrix} \Delta z^k \\ \Delta\lambda^k \end{bmatrix} = - \begin{bmatrix} \tilde{r}_z^k \\ c(z^k), \end{bmatrix} \tag{9}$$

where,

$$H^k = \nabla^2_{zz}\mathcal{L}(z^k) + (\underline{Z}^k)^{-1}\underline{D}^k + (\overline{Z}^k)^{-1}\overline{D}^k \tag{10}$$

and

$$\tilde{r}^k_z = \nabla_z f^k + \nabla_z c(z^k)\lambda - (\underline{Z}^k)^{-1}\mu e + (\overline{Z}^k)^{-1}\mu e. \tag{11}$$

This interior-point algorithm employs a filter-based line-search strategy that requires the generated step to be a descent direction. This is ensured if the following inertia condition is satisfied [17],

$$\text{Inertia}(K) = (n, m, 0). \tag{12}$$

The inertia is the number of positive, negative, and zero eigenvalues of the matrix K, n is the number of variables, m is the number of equality constraints, and

$$K = \begin{bmatrix} H^k & \nabla_z c(z^k) \\ [\nabla_z c(z^k)]^T & 0 \end{bmatrix}. \tag{13}$$

We wish to use this algorithm to solve general non-convex NLPs. To ensure descent directions for these problems, we may need to modify the linear system (13) utilizing inertia correction. The modified linear system is

$$\begin{bmatrix} H^k + \delta_H I & \nabla_z c(z^k) \\ [\nabla_z c(z^k)]^T & -\delta_c I \end{bmatrix} \begin{bmatrix} \Delta z^k \\ \Delta\lambda^k \end{bmatrix} = - \begin{bmatrix} \tilde{r}^k_z \\ c(z^k) \end{bmatrix}. \tag{14}$$

Here, δ_H and δ_c are set to zero except when their values must be increased to satisfy the inertia condition. This system is solved for each interior-point iteration to calculate the full steps in z and λ. The necessary algebra is then performed to calculate the steps in $\Delta\overline{d}$ and $\Delta\underline{d}$, and a line-search is used to ensure that the steps taken in each of these variables are suitable.

In this algorithm, the most computationally expensive steps are solving the linear system (14) and calculating the residuals ($|c(z^k)|$), gradients ($\nabla_z f^k$ and $\nabla_z c(z^k)$), and Hessian H^k. For the dynamic problems addressed in this paper, inherent structure exists that allows for decomposition and efficient parallel solution of this linear system and parallel evaluation of these functions. In the next section, we describe the transcription approach we use for dynamic problems that induces the structure of these problems.

3 Dynamic model formulation & model transcription

The dynamic optimization problems considered in this paper are based on differential algebraic equation (DAE) models of the form

$$\min_u \int_{t_0}^{t_f} L(x, u, y)\, dt \tag{15}$$

s.t.

$$F(\dot{x}, x, u, y) = 0, \tag{16}$$

$$x(t_0) = x_0 \tag{17}$$

$$\underline{x} \leq x \leq \overline{x} \tag{18}$$

$$\underline{u} \leq u \leq \overline{u} \tag{19}$$

$$\underline{y} \leq y \leq \overline{y} \tag{20}$$

where $\dot{x} \in R^{n_x}$ are the state derivative variables, $x \in R^{n_x}$ are the state variables, $u \in R^{n_u}$ are the control input variables, $y \in R^{n_y}$ are the algebraic variables, and all variables may have lower and upper bounds represented by underlines and overlines respectively. It is assumed that the DAE satisfies the relationship $\left[\frac{\partial F}{\partial \dot{x}}, \frac{\partial F}{\partial y}\right] \neq 0$ (i.e. the system is non-singular). This condition is consistent with the requirements for an index 1 DAE as shown in [9]. Index reduction techniques can be used to meet this assumption [35].

The optimization problem is discretized using a simultaneous transcription method based on finite elements, with Radau collocation points. See, e.g. [8] for a recent monograph. Lagrange polynomials are used to approximate the state, algebraic and control input profiles.

The optimization mesh is defined by n_e finite elements, with normalized lengths h_i, i=1, ..., n_e, where $\sum_{i=1}^{n_e} h_i = 1$. The left boundary of each finite element is then given by

$$t_i = t_0 + (t_f - t_0) \sum_{k=1}^{i-1} h_k \quad \forall\, i = 1, ..., n_e, \tag{21}$$

and the collocation points are given by

$$t_{i,j} = t_0 + (t_f - t_0)\left(\sum_{k=1}^{i-1} h_k + \tau_j h_i\right) \quad \forall\, i = 1, ..., n_e, \quad j = 1, ..., n_c, \tag{22}$$

where $\tau_j \in (0, 1]$, j=1, ..., n_c are the Radau collocation points. In this work we use a 5th order Radau method and therefore $n_c = 3$.

At each collocation point, the discretized variable vectors, $\dot{x}_{i,j}, x_{i,j}, u_{i,j}$, and $y_{i,j}$ for i=1, ..., n_e and j=1, ..., n_c, are introduced. In addition, state variables at the beginning of each finite element, $x_{i,0}$ for i=1, ..., n_e, are introduced. In each finite element, the differentiated variables are approximated by

$$x(t) = \sum_{k=0}^{n_c} x_{ik} L_k^{n_c+1}\left(\frac{t - t_i}{(t_f - t_0)h_i}\right), \quad t \in [t_i, t_{i+1}] \tag{23}$$

where $L_j^{n_c+1}(\tau)$ are Lagrange polynomials of order n_c which are computed based on the points τ_0, ..., τ_{n_c}, with $\tau_0 = 0$. Accordingly, the expressions for the derivatives $\dot{x}_{i,j}$ are given by

$$\dot{x}_{i,j} = \frac{1}{(t_f - t_0)h_i} \sum_{k=0}^{n_c} x_{i,k} \dot{L}_k^{n_c+1}(\tau_j), \;\; i = 1, ..., n_e, \;\; j = 1, ..., n_c. \tag{24}$$

At each collocation point, $t_{i,j}$, the DAE relation (16)

$$F(\dot{x}_{i,j}, x_{i,j}, u_{i,j}, y_{i,j}) = 0, \;\; i = 1, ..., n_e, \;\; j = 1, ..., n_c \tag{25}$$

holds. In addition, continuity constraints for the state variable profiles are enforced

$$x_{i-1,n_c} = x_{i,0}, \;\; i = 2, ..., n_e \tag{26}$$

as well as the initial conditions $x_{1,0}$=x_0. Notice that the relation (26) holds since for Radau collocation points, τ_{n_c}=1.

We denote the equations (24) and (25) in residual form for each finite element i by $R_i = R(z_i)$=0, where $z_i^T = [x_{i,0}^T \dot{x}_{i,1}^T, x_{i,1}^T, u_{i,1}, y_{i,1}^T, \ldots, \dot{x}_{i,n_c}^T, x_{i,n_c}^T, u_{i,n_c}, y_{i,n_c}^T]$.

The cost function (15) is discretized using a quadrature formula

$$\sum_{i=1}^{n_e} \sum_{j=1}^{n_c} w_j L(x_{i,j}, u_{i,j}, y_{i,j}) = f(z) \tag{27}$$

where w_j are the Radau quadrature weights.

The discretized optimal control problem can now be written in the form

$$\min_z \;\; f(z) \tag{28a}$$

$$\text{s.t.} \;\; c(z) = 0 \tag{28b}$$

$$\underline{z} \le z \le \overline{z} \tag{28c}$$

where

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_{n_e} \end{bmatrix}, \qquad z_i = \begin{bmatrix} x_{i,0} \\ \dot{x}_{i,1} \\ x_{i,1} \\ u_{i,1} \\ y_{i,1} \\ \vdots \\ \dot{x}_{i,n_c} \\ x_{i,n_c} \\ u_{i,n_c} \\ y_{i,n_c} \end{bmatrix} \forall \; i = 1, ..., n_e, \quad \text{and} \quad c(z) = \begin{bmatrix} \overline{G} z_1 - x_0 \\ R(z_1) \\ \underline{G} z_1 + \overline{G} z_2 \\ \vdots \\ \underline{G} z_{n_e-1} + \overline{G} z_{n_e} \\ R(z_{n_e}) \end{bmatrix}. \tag{29}$$

Here, $\underline{z}$ and $\overline{z}$ are respectively the lower and upper bounds on z, n_c is the number of collocation points, n_e is the number of finite elements, and element coupling matrices (arising from the continuity equations) are given by

$$\overline{G} = \begin{bmatrix} I & 0 & \dots & 0 \end{bmatrix}, \quad \underline{G} = \begin{bmatrix} 0 & \dots & 0 & -I & 0 & 0 \end{bmatrix}. \tag{30}$$

The coupling constraints $\underline{G}z_{i-1} + \overline{G}z_i = 0$ link individual finite elements in time. It is important to note that only the state variables are temporally coupled between elements, not the algebraic variables. In other words, the $-I$ block in $\underline{G}$ corresponds only to x_{i,n_c}. Therefore, the dimension of these constraints, (i.e., the number of rows in $\underline{G}$ and $\overline{G}$) is dependent on the number of state variables only. It is this property that will be exploited to decompose the problem and develop an efficient parallel solution approach.

4 Parallel solution of the dynamic optimization problem

Introducing decoupling variables, $q_i = x_{i,n_c}$, $i = 1..n_e - 1$, we may rewrite the NLP resulting from collocation as

$$\min_z \quad f(z) \tag{31a}$$

$$\text{s.t.} \quad c(z, q) = 0 \tag{31b}$$

$$\underline{z} \le z \le \overline{z} \tag{31c}$$

where $\underline{z}$ and $\overline{z}$ are respectively the lower and upper bounds on z, $z^T = [z_1^T, \dots, z_{n_e}^T]$, $q^T = [q_1^T, \dots, q_{n_e-1}]$, and

$$c(z, q) = \begin{bmatrix} \overline{G}z_1 - x_0 \\ R(z_1) \\ \underline{G}z_1 + q_1 \\ \overline{G}z_2 - q_1 \\ R(z_2) \\ \underline{G}z_2 + q_2 \\ \vdots \\ \overline{G}z_{n_e} - q_{n_e-1} \\ R(z_{n_e}) \end{bmatrix}. \tag{32}$$

Solution of this large-scale nonlinear programming problem is possible with a number of potential algorithms. The dominant cost of an interior-point algorithm is the solution of the linear KKT system at each iteration to find the full step in the primal and dual variables. This linear system can be solved with direct factorization methods appropriate for symmetric indefinite systems, however, the structure of the optimal control problem induces structure in the KKT system that can be exploited with a parallel decomposition algorithm. This linear system can be decomposed by selecting

break-points in time and performing a Schur-complement decomposition with respect to the coupling constraints (or corresponding multipliers) and variables.

Here, we describe our decomposition algorithm assuming that decoupling variables q and coupling constraints are added between each finite element. This gives a system where each finite element represents one decomposed block. However, our actual implementation is able to decompose the problem with multiple finite elements per block, where decoupling variables and coupling constraints are not added between every finite element. For example, a problem with 128 finite elements can be separated into two blocks of 64 finite elements each, 4 blocks of 32 finite elements each, and so forth. The number of blocks formed should be determined by the number of available processors.

The linear KKT system solved at each iteration of the interior-point optimization algorithm (specified in (14)) can be written in the following block-bordered structure. Again, for simplicity of notation, the structure is written with a decoupling variables introduced between every finite element.

$$\begin{bmatrix} K_1 & & & A_1^T \\ & K_2 & & A_2^T \\ & & \ddots & \vdots \\ & & K_{n_e} & A_{n_e}^T \\ A_1 & A_2 & \dots & A_{n_e} \quad Q \end{bmatrix} \begin{bmatrix} \Delta v_1 \\ \Delta v_2 \\ \vdots \\ \Delta v_{n_e} \\ \Delta v_s \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n_e} \\ r_s \end{bmatrix} \tag{33}$$

where

$$K_i = \begin{bmatrix} H_i + \delta_H I & \overline{G}^T & \nabla_{z_i} R(z_i) \\ \overline{G} & -\delta_c I & \\ \nabla_{z_i} R(z_i)^T & & -\delta_c I \end{bmatrix}, \quad \Delta v_i = \begin{bmatrix} \Delta z_i \\ \Delta \lambda_{\overline{G},i} \\ \Delta \lambda_{R,i} \end{bmatrix}, \quad r_i = \begin{bmatrix} -\tilde{r}_{z_i} \\ -\overline{G} z_i + q_{i-1} \\ -R_i \end{bmatrix}$$
$$\forall\, i = 1, \dots, n_e, \tag{34}$$

$$A_1 = \begin{bmatrix} 0 & 0 & 0 \\ \underline{G} & 0 & 0 \\ 0 & 0 & 0 \\ & \vdots & \end{bmatrix}$$

$$A_i = \begin{bmatrix} 0 & 0 & 0 \\ & \vdots & \\ 0 & -I & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \underline{G} & 0 & \\ 0 & 0 & 0 \\ & \vdots & \end{bmatrix} \forall\, i = 2, \dots, n_e - 1 \tag{35}$$

$$A_{n_e} = \begin{bmatrix} 0 & 0 & 0 \\ & \vdots & \\ 0 & 0 & 0 \\ 0 & -I & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$Q = \begin{bmatrix} \delta_H I & I & & & & & \\ I & -\delta_c I & & & & & \\ & & \delta_H I & I & & & \\ & & I & -\delta_c I & & & \\ & & & & \ddots & & \\ & & & & & \delta_H I & I \\ & & & & & I & -\delta_c I \end{bmatrix}, \tag{36}$$

$$\Delta v_s = \begin{bmatrix} \Delta q_1 \\ \Delta \lambda_{\underline{G},1} \\ \vdots \\ \Delta q_{n_e-1} \\ \Delta \lambda_{\underline{G},n_e-1} \end{bmatrix}, \text{ and } r_s = \begin{bmatrix} -\nabla_q \tilde{r}_1 \\ -\underline{G} z_1 - q_1 \\ \vdots \\ -\nabla_q \tilde{r}_{n_e-1} \\ -\underline{G} z_{n_e-1} - q_{n_e-1} \end{bmatrix}. \tag{37}$$

Here, H_i is the modified Hessian described in (10), and δ_H and δ_c may be zero or positive depending on the need of the algorithm to handle non-convexity and/or singlularity in the Jacobian. The Δv_i vectors include the primal and dual variables for element i, and Δv_s contains the dual variables for the coupling constraints. In this permutation, the coupling constraints, (i.e., the Jacobian matrices $\underline{G}$ and $\overline{G}$), and their corresponding dual variables have been permuted to the borders of the KKT system.

The step in the variables v_s can be decoupled from the remaining variables by eliminating the A_i matrices, resulting in the following Schur-complement decomposition,

$$\left[Q - \sum_i A_i K_i^{-1} A_i^T \right] \Delta v_s = r_s - \sum_i A_i K_i^{-1} r_i. \tag{38}$$

This decomposition allows solution of the KKT system using the following algorithm.

Algorithm: Schur-Complement Solve of KKT System

1: for each i in 1, ..., n_e
  1.1: factor K_i (using MA27 from Harwell Subroutine Library)
2: Initialize S by letting $S = Q$ (shown in (36))
3: let $r_{sc} = r_s$
4: for each i in 1, ..., n_e
  4.1: for each nonzero column j in A_i^T
    4.1.1: solve the system $K_i s_i^{<j>} = [A_i^T]^{<j>}$ for $s_i^{<j>}$
    4.1.2: let $S^{<j>} = S^{<j>} - A_i s_i^{<j>}$
  4.2: solve the system $K_i p_i = r_i$ for p_i
  4.3: let $r_{sc} = r_{sc} - A_i p_i$
5: solve $S \Delta v_s = r_{sc}$ for Δv_s

6: for each i in 1, ..., n_e
6.1: solve $K_i \Delta v_i = r_i - A_i^T \Delta v_s$ for Δv_i

There are several levels of parallelism that can be exploited in this algorithm. If there is one processor available for each element, then Steps 1, 4, and 6 can all be parallelized by utilizing one processor for each i in n_e. Furthermore, if more processors are available, individual column backsolves in Step 4.1 can be parallelized. In the traditional Schur-complement decomposition approach, the number of columns in A_i^T is typically dependent on the overall number of coupling or first-stage variables. However, with this problem structure, the number of nonzero columns in A_i^T is dependent on the number of state variables only. Therefore, when solving the system in parallel, the size of the Schur-complement grows with the number of processors, but the number of backsolves required for Step 4.1 does not.

In previous work we have shown excellent parallel scalability using this strategy for problems with complicating variables [45,47] where the size of the Schur-complement is determined by the number of complicating variables only. In the approach described here, the size of the Schur-complement increases with the number of states and the number of processors used. As the size of the Schur-complement grows, the increased cost of solving the Schur-complement system (Step 5) will erode parallel speedup. In general, the Schur-complement is dense and the cost of solution with a dense linear solver would increase cubically with the size of the Schur-complement. However, due to the block structure induced on the A_k blocks by the collocation and decomposition, the Schur-complement here can be quite sparse. Figure 1 shows the block structure of the Schur-complement formed using our approach. While the dimension of the Schur-complement grows quadratically with the number of A_k blocks and is equal to $(2N-2)^2$, where N is the number of blocks, the number of nonzero blocks in the Schur-complement only grows linearly and is equal to $4+6(N-2)$. Note that N must be greater or equal to 2 for this decomposition to be utilized. The Schur-complement

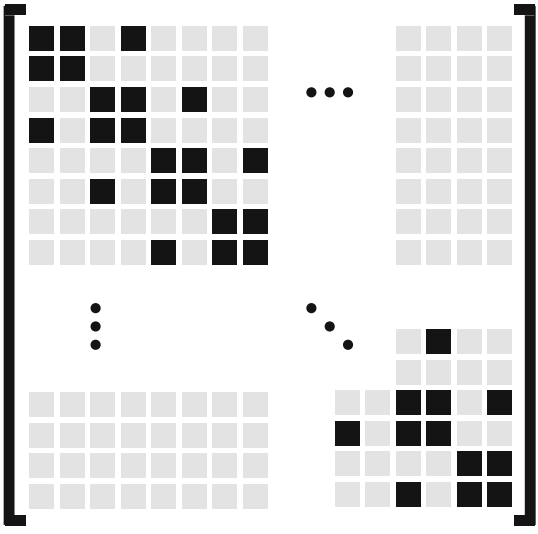

Fig. 1 The Schur-complement sparsity pattern. The dark boxes represent blocks that can be dense and contain nonzeros, and the light boxes represent blocks that can only contain zeros.

can be quite sparse when a large number of processors is used, and using efficient sparse linear solvers can dramatically reduce the computational time required to solve the Schur-complement, improving parallel scalability. In this work we use the sparse linear solver MA27 [25].

5 Implementation

In this paper we make use of the JModelica.org [1] modeling framework that is based on Modelica, Optimica, and Python, to transform high-level descriptions of dynamic optimization problems into algebraic nonlinear programming problems through a direct collocation approach. JModelica.org is a comprehensive modeling and optimization tool for large-scale dynamic optimization problems. The platform employs compiler technology, symbolic manipulation, and code generation to transform high-level Modelica and Optimica descriptions into efficient executables suitable for linking with numerical solvers. In addition, XML files containing DAE-constrained optimization problems can be generated from Modelica and Optimica descriptions. We interface with the open-source symbolic framework for automatic differentiation and optimal control, CasADi [4], for efficient automatic differentiation. CasADi supports import of model XML files compliant with the format used by JModelica.org, which enables a seamless integration between the tools, [3].

The interior-point algorithm is implemented in C++ using an object-oriented design where the core interior-point algorithm is independent of the specific problem representation and linear algebra routines. This design eases the development of custom decomposition approaches to exploit specific problem structure since the fundamental algorithm does not require change. This design also allows for straightforward parallelization since only the interfaces to the linear algebra routines are exposed to the fundamental algorithm, and the underlying operations function identically whether executed in serial or parallel. The algorithm requires several vector, vector-vector, and matrix-vector linear algebra operations (e.g. dot product, norms, matrix-vector multiplication), and in the parallel implementation these operations are parallelized using MPI routines.

6 Benchmark problem

To demonstrate the performance characteristics of our algorithm we solve a problem to find the optimal control profile for the start-up of a combined cycle power plant. The power plant model is encoded in the object-oriented modeling language Modelica, and is based on a library consisting of model classes representing the model components. The model object diagram is shown in Figure 2. The key limiting factor with regards to start-up speed is the thermal stress in the steam turbine. The stress in the turbine axis is proportional to the temperature gradient, which in turn results when hot steam is exposed to the cold axis surface. Apart from the heat transfer in the steam turbine axis, the main model elements are the boiler, an economizer and a super heater. The control input of the model is the load of the gas turbine. The model has ten differential states, 127 algebraic variables and one control input.

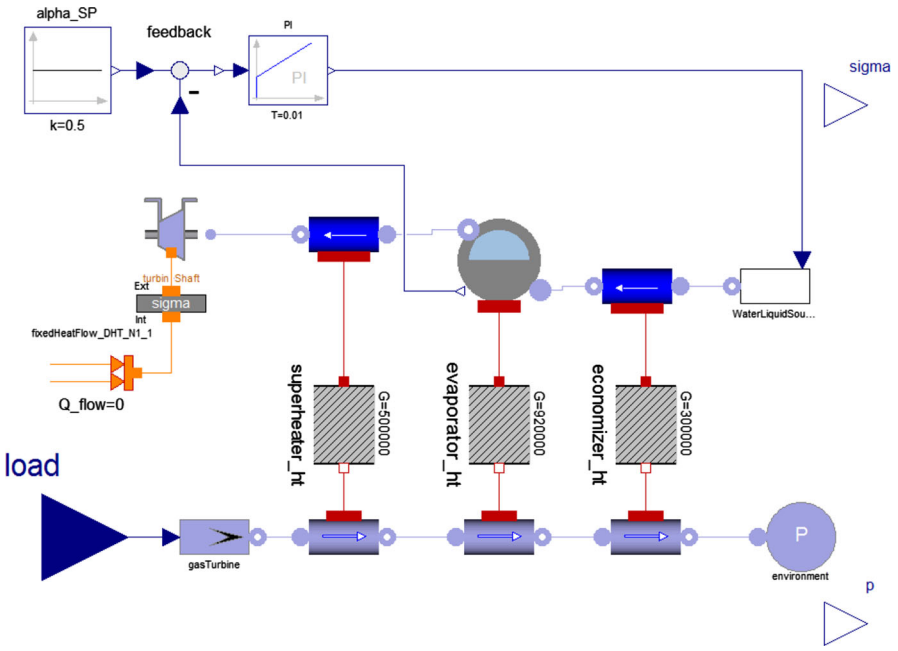

Fig. 2 Object diagram of the combined cycle power plant.

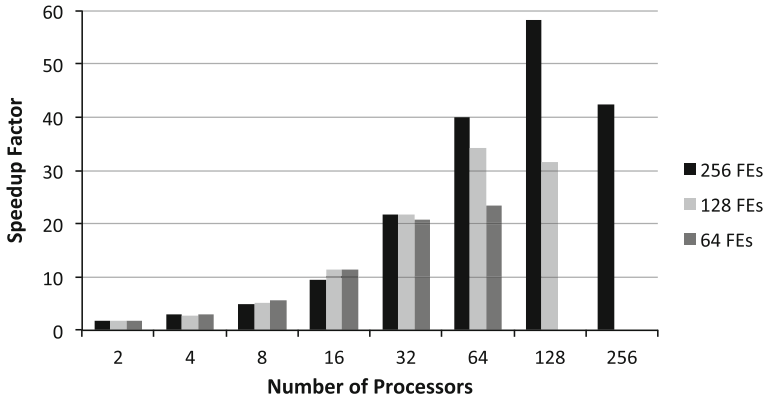

Fig. 3 Speedups comparing the parallel Schur-complement linear solver with the serial Schur-complement linear solver for problems with 64, 128, and 256 finite elements.

The optimal control problem seeks to determine a trajectory for the gas turbine load which drives the pressure in the boiler to a target value, corresponding to full production, while respecting a bound on the thermal stress in the steam turbine axis. In addition, there is a rate limit on the control input. For details on the models and the optimal control formulation, we refer to [10]. Due to its large size, the complete model is not included herein, however it is available as the CombinedCycleStartup example in the JModelica.org repository [26,27].

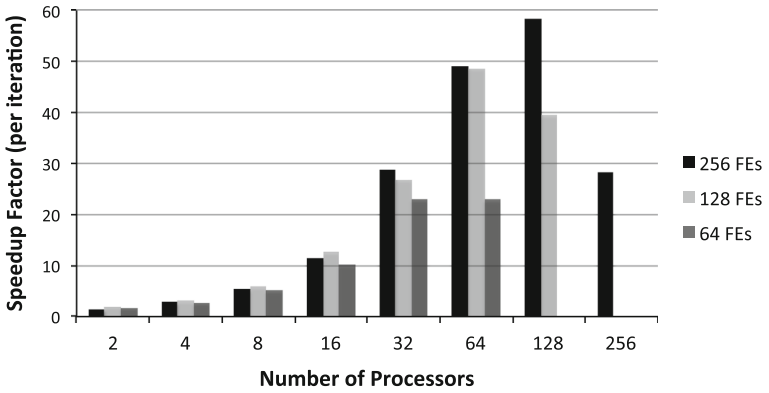

Fig. 4 Speedups comparing the parallel Schur-complement linear solver with the full-space linear solver MA27 for problems with 64, 128, and 256 finite elements.

7 Performance results

The computational cost of the approach described in Section 4 is a function of the number of state variables, (i.e. the dimension of the coupling constraints), and the number of processors used in the decomposition. As we increase the number of processors, we have the potential for greater parallelization, however, the size of the Schur-complement (and hence the cost of Step 5) also increases. Increased parallelization also adds computational time required for inter-processor communication. Depending upon problem size, this additional communication burden can be significant.

We test the speedup of our algorithm on 3 test problems of different sizes with 64, 128, and 256 finite elements resulting in problems with 29878, 59766, and 119542 variables and 29686, 59382, and 118774 constraints respectively. Figure 3 shows the speedup of our parallel decomposition code when compared with the same decomposition performed in serial. In this example, a speedup of over 50 times was possible for the largest problem. Figure 4 shows the speedup for our parallel decomposition when compared with the serial full-space approach where the entire KKT system is solved directly with MA27. Here, we see that our parallel decomposition is still capable of appreciable speedup (over 50 times faster for the largest problem). Note that the speedups comparing the decomposition and full-space algorithms are per iteration of the interior-point algorithm rather than for the overall runtime. Recall that our approach introduces additional coupling variables when decomposing the problem for parallel solution. To provide a fair comparison, these coupling variables are not introduced when solving the problem using the full-space approach since these variables are not necessary and their inclusion would add additional computational overhead to the method. Speedup per iteration is shown because the full-space and decomposition algorithms are solving different systems and may therefore take slightly different steps, which may require a different but comparable number of steps (and therefore interior-point iterations). In our experience neither algorithm consistently requires fewer iterations.

While significant speedup is observed, as expected, the speedup of our parallel algorithm deteriorates when we increase to larger numbers of processors. This is due to two aspects. First, as the number of processors is increased, the computational time required by each processor decreases, while the communication overhead increases. For larger problems, where the computational burden per processor is higher, this effect is reduced. For this reason, we expect that our algorithm would demonstrate better speedup on larger problems (for the same number of state variables). Second, the time required to solve the Schur-complement (Step 5) increases with the number of blocks, and, while this time remains small, it makes an increasingly significant contribution to the solution time.

Table 1 details the times per iteration required in the computationally expensive steps of the linear solve. Both the serial and parallel times are shown for our test problem with varying numbers of finite elements and blocks. Increasing the number of blocks makes each individual block smaller, and factorizing and performing backsolves with these smaller blocks can be significantly faster. This is why as the number of blocks increases the times required for Steps 1 and 6 can actually decrease even when performing these calculations in serial. However, while the number of operations in Step 4 stays the same for each block, increasing the number of blocks increases the total number of operations, so even though individual backsolves may be faster, the time required for the serial algorithm increases. For the parallel algorithm, each block is distributed on separate processors so the time required for Step 4 does not increase until the inter-processor communication overhead becomes significant. This table also shows the increase in computational time required to solve the Schur-complement. For example, for the 256 finite element case Step 5 requires less than 0.005% of the linear solver time when using 2 blocks but more than 33% of the linear solver time when using 256 blocks.

All timing results were obtained using the Red Mesa supercomputing cluster at Sandia National Lab. This cluster is made up of computing nodes, each with two, 2.93 GHz quad-core, Nehalem X5570-processors, giving 8 computing cores per node. Each node has 12 GB of DDR3 RAM. For the results shown in Figs. 3 and 4 only one computing core was used per computing node. However, computing clusters with distributed memory are typically much more expensive and less common than the shared memory architectures of the standard desktop computer. Computer manufactures are increasing the number of computing cores available on the standard desktop computer, and it is important to understand the performance that can be expected on these common computer architectures. Figure 5 compares the speedups achieved using distributed memory and shared memory architectures. The shared memory speedups come from solving the combined cycle powerplant start-up problem with 256 finite elements using an increasing number of computing cores on a single computing node. Note that as the number of processors increases, the speedup does not suffer.

8 Conclusions and future work

This paper presents a decomposition approach that is applicable for nonlinear dynamic optimization problems formulated using the simultaneous approach. A Schur-

Table 1 The table shows in seconds the average time per iteration required in the steps of the Schur-complement algorithm for the example problem as the number of finite elements and blocks is changed. Both serial and parallel times are shown.

Schur-Complement Times									
# of FEs	# of Blocks	Step 1		Step 4		Step 5		Step 6	
		Serial SC	Parallel SC	Serial SC	Parallel SC	Serial SC	Parallel SC	Serial SC	Parallel SC
64	2	0.35	0.197	0.10	0.05	7.1E-05	5.9E-05	1.2E-02	4.9E-03
	4	0.37	0.115	0.13	0.05	1.6E-04	1.5E-04	1.1E-02	3.0E-03
	8	0.35	0.058	0.14	0.03	3.2E-04	3.2E-04	1.1E-02	1.6E-03
	16	0.34	0.030	0.16	0.01	6.6E-04	6.7E-04	1.0E-02	5.8E-04
	32	0.22	0.010	0.17	0.01	1.4E-03	1.4E-03	8.5E-03	2.7E-04
	64	0.21	0.005	0.23	0.01	3.0E-03	3.1E-03	8.0E-03	3.0E-04
128	2	0.80	0.472	0.28	0.15	6.5E-05	7.6E-05	2.8E-02	1.4E-02
	4	0.83	0.280	0.31	0.13	1.7E-04	1.7E-04	2.5E-02	7.8E-03
	8	0.75	0.126	0.34	0.09	3.5E-04	3.7E-04	2.3E-02	4.4E-03
	16	0.73	0.062	0.37	0.03	7.3E-04	7.7E-04	2.2E-02	1.8E-03
	32	0.61	0.029	0.37	0.01	1.4E-03	1.4E-03	2.0E-02	7.1E-04
	64	0.39	0.009	0.45	0.01	2.9E-03	2.9E-03	1.7E-02	4.4E-04
	128	0.26	0.001	0.69	0.01	6.2E-03	5.4E-03	1.5E-02	7.6E-04
256	2	3.05	1.853	0.74	0.41	1.4E-04	1.1E-04	6.9E-02	3.5E-02
	4	2.28	0.754	0.90	0.34	1.6E-04	2.1E-04	6.1E-02	1.8E-02
	8	2.00	0.361	0.79	0.21	3.5E-04	3.5E-04	5.2E-02	9.6E-03
	16	1.78	0.174	0.71	0.09	6.6E-04	7.3E-04	4.6E-02	4.4E-03
	32	1.43	0.064	0.79	0.04	1.4E-03	1.5E-03	4.2E-02	2.4E-03
	64	1.40	0.034	1.00	0.02	2.9E-03	3.0E-03	4.1E-02	1.0E-03
	128	1.21	0.014	1.79	0.02	7.8E-03	8.1E-03	3.7E-02	1.2E-03
	256	0.84	0.005	3.66	0.03	1.9E-02	2.2E-02	3.4E-02	2.1E-03

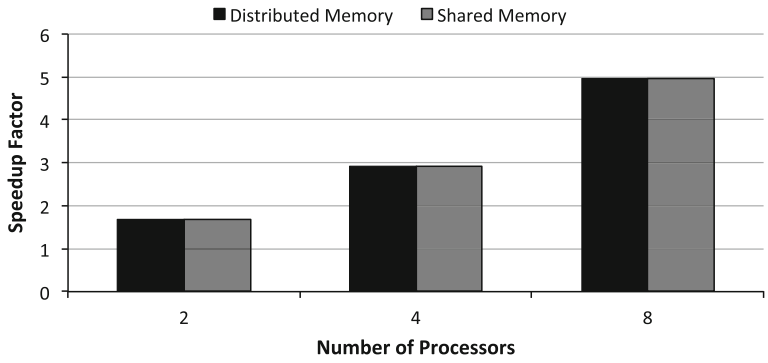

Fig. 5 Speedups comparing shared-memory and distributed-memory architectures. These speedups compare our solution approach using the parallel Schur-complement algorithm over the serial Schur-complement algorithm on shared-memory and distributed-memory architectures.

complement algorithm is used for parallel solution of the linear systems resulting from an interior-point solution of these optimization problems. The dominant costs of the Schur-complement algorithm are Step 1 (factoring the K blocks), Step 4 (forming the Schur-complement), and Step 5 (solving the Schur-complement), however, Steps 1 and 4 can be efficiently parallelized, and for our test problem the computational cost of Step 5 remained small.

We interfaced our solution approach with the JModelica.org modeling framework that transforms high-level descriptions of dynamic optimization problems into algebraic nonlinear programming problems through a direct collocation approach [1]. This modeling framework is integrated with CasADi [4] to provide efficient automatic differentiation [3].

For problems with few states and many algebraics, this solution approach has the potential for significant speedup. As the size of the Schur-complement increases (more states or processors), parallel speedup is eroded, and with an increasing number of processors the cost of inter-processor communication can become significant. This is especially true with relatively small problems that solve quickly, since the time required for communication can easily become a significant cost of the algorithm. Nevertheless, this approach can still be highly efficient. For blocks A_k of arbitrary structure, the Schur-complement may indeed be dense, but for dynamic optimization problems studied here, the structure of the A_k blocks is such that the resulting Schur-complement is both block structured and relatively sparse making it ideally suited for solution using sparse linear solvers. In this work, the efficient, serial sparse linear solver MA27 was used to solve the Schur-complement with promising results, but it could be possible to see further algorithm improvements if an efficient, parallel sparse linear solver was utilized.

The combined cycle power plant problem shows that this parallel decomposition approach can be coupled with a parallel interior-point algorithm for efficient solution of real optimal control problems. The case study displayed the potential for speedups of more than 50 over full-space approaches using distributed memory computing architectures. Additional results compared the speedup of our algorithm using shared

memory and distributed memory computing architectures, and for the computing hardware used in our tests no differences were observed.

Acknowledgments The authors thank Francesco Casella for providing the combined cycle power plant model used in this work and Joel Andersson for his assistance with interfacing our software with CasADi. The authors gratefully acknowledge partial financial support for Daniel Word provided by Sandia National Laboratories and the Office of Advanced Scientific Computing Research within the DOE Office of Science as part of the Applied Mathematics program. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U. S. Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000. Thanks is also extended for financial support for Jia Kang provided by the National Science Foundation Cyber-Enabled Discovery and Innovation (CDI)-Type II. The authors gratefully acknowledge partial financial support for Carl Laird and Daniel Word from the National Science Foundation (CAREER Grant CBET# 0955205). The authors gratefully acknowledges financial support for Johan Akesson from the Swedish Science Foundation through the grant *Lund Center for Control of Complex Engineering Systems (LCCC)*.

References

1. Åkesson, J., Årzén, K.E., Gäfvert, M., Bergdahl, T., Tummescheit, H.: Modeling and optimization with Optimica and JModelica.org–languages and tools for solving large-scale dynamic optimization problem. Comput. Chem. Eng. **34**(11), 1737–1749 (2010). doi:10.1016/j.compchemeng.2009.11.011
2. Amestoy, P., Duff, I., L'Excellent, J.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods Appl. Mech. Eng. **184**(2), 501–520 (2000)
3. Andersson, J., Åkesson, J., Casella, F., Diehl, M.: (2011, March). Integration of CasADi and JModelica.org. In 8th International Modelica Conference 2011, Dresden, Germany
4. Andersson, J., Åkesson, J., Diehl, M.: (2012) CasADi: A symbolic package for automatic differentiation and optimal control. In Recent Advances in Algorithmic, Differentiation. Springer 297–307
5. Benson, D.A., Huntington, G.T., Thorvaldsen, T.P., Rao, A.V.: Direct trajectory optimization and costate estimation via an orthogonal collocation method. J. Guid. Control. Dyn. **29**(6), 1435–1440 (2006)
6. Biegler, L., Cervantes, A., Wächter, A.: Advances in simultaneous strategies for dynamic process optimization. Chem. Eng. Sci. **57**(4), 575–593 (2002)
7. Biegler, L., Grossmann, I.: Retrospective on optimization. Comput. Chem. Eng. **28**(8), 1169–1192 (2004)
8. Biegler, L. T.: (2010). Nonlinear programming: concepts, algorithms, and applications to chemical processes, Vol. 10. SIAM
9. Brenan, K. E., Campbell, S. L.-V., Petzold, L. R.: (1989). Numerical solution of initial-value problems in differential-algebraic equations, Vol. 14. SIAM
10. Casella, F., Donida, F., Åkesson, J.: (2011, August) Object-oriented modeling and optimal control: A case study in power plant start-up. In 18th IFAC World Congress, Milano, Italy
11. Cervantes, A., Biegler, L.: A stable elemental decomposition for dynamic process optimization. J. Comput. Appl. Math. **120**(1), 41–57 (2000)
12. Cervantes, A., Wächter, A., Tütüncü, R., Biegler, L.: A reduced space interior point strategy for optimization of differential algebraic systems. Comput. Chem. Eng. **24**(1), 39–51 (2000)
13. Darby, C.L., Hager, W.W., Rao, A.V.: Direct trajectory optimization using a variable low-order adaptive pseudospectral method. J. Spacecr. Rockets **48**, 433–445 (2011)
14. DeMiguel, V., Nogales, F.: On decomposition methods for a class of partially separable nonlinear programs. Math. Oper. Res. **33**(1), 119–139 (2008)
15. Diehl, M., Bock, H., Schlöder, J., Findeisen, R., Nagy, Z., Allgöwer, F.: Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. J. Process Control **12**(4), 577–585 (2002)
16. Fornberg, B.: A practical guide to pseudospectral methods, vol. 1. Cambridge University Press, Cambridge (1998)
17. Forsgren, A., Gill, P.E., Wright, M.H.: Interior methods for nonlinear optimization. SIAM review **44**(4), 525–597 (2002)

18. Garg, D., Patterson, M., Hager, W.W., Rao, A.V., Benson, D.A., Huntington, G.T.: A unified framework for the numerical solution of optimal control problems using pseudospectral methods. Automatica **46**(11), 1843–1851 (2010)
19. Garg, D., Patterson, M.A., Francolin, C., Darby, C.L., Huntington, G.T., Hager, W.W., Rao, A.V.: Direct trajectory optimization and costate estimation of finite-horizon and infinite-horizon optimal control problems using a Radau pseudospectral method. Comput. Optim. Appl. **49**(2), 335–358 (2011)
20. Goulart, P., Kerrigan, E., Ralph, D.: Efficient robust optimization for robust control with constraints. Math. Progr. **114**(1), 115–147 (2008)
21. Hart, W., Laird, C., Watson, J., Woodruff, D.: Pyomo-optimization modeling in Python, vol. 67. Springer, New York (2012)
22. Hartwich, A., Marquardt, W.: Dynamic optimization of the load change of a large-scale chemical plant by adaptive single shooting. Comput. Chem. Eng. **34**(11), 1873–1889 (2010)
23. Hartwich, A., Stockmann, K., Terboven, C., Feuerriegel, S., Marquardt, W.: Parallel sensitivity analysis for efficient large-scale dynamic optimization. Optim. Eng. **12**(4), 489–508 (2011)
24. Houska, B., Ferreau, H.J., Diehl, M.: ACADO toolkit-An open-source framework for automatic control and dynamic optimization. Optim. Control Appl. Methods **32**(3), 298–312 (2011)
25. HSL (2011) A collection of Fortran codes for large scale scientific computation. HSL. http://www.hsl.rl.ac.uk
26. JModelica.org (2012a). CombinedCycle.mo. https://svn.jmodelica.org/trunk/Python/src/pyjmi/examples/files. [Revision 4090]
27. JModelica.org (2012b). CombinedCycleStartup.mop. https://svn.jmodelica.org/trunk/Python/src/pyjmi/examples/files. [Revision 4090]
28. Kameswaran, S., Biegler, L.T.: Convergence rates for direct transcription of optimal control problems using collocation at Radau points. Comput. Optim. Appl. **41**(1), 81–126 (2008)
29. Kocak, S., Akay, H.: Parallel Schur complement method for large-scale systems on distributed memory computers. Appl. Math. Model. **25**(10), 873–886 (2001)
30. Laird, C., Biegler, L.: Large-scale Nonlinear Programming for Multi-scenario Optimization. In: Bock, H.G., Kostina, E., Phu, H.X., Ranacher, R. (eds.) Modeling, simulation and optimization of complex processes, pp. 323–326. Springer, New York (2008)
31. Laird, C., Biegler, L., van Bloemen Waanders, B., Bartlett, R.: Contamination source determination for water networks. J. Water Res. Plan. Manag. **131**(2), 125–134 (2005)
32. Laird, C., Wong, A., Akesson, J.: (2011) Parallel solution of large-scale dynamic optimization problems. In 21st European Symposium on Computer Aided Process Engineering-ESCAPE, Vol. 21
33. Lang, Y.-D., Biegler, L.: A software environment for simultaneous dynamic optimization. Comput. Chem. Eng. **31**(8), 931–942 (2007)
34. Leineweber, D., Bauer, I., Bock, H., Schlöder, J.: An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. part 1: theoretical aspects. Comput. Chem. Eng. **27**(2), 157–166 (2003)
35. Mattsson, S., Söderlind, G.: Index reduction in differential-algebraic equations using dummy derivatives. SIAM J. Sci. Comput. **14**(3), 677–692 (1993)
36. Modelica Association (2007) The Modelica language specification version 3.0
37. Rao, C.V., Wright, S.J., Rawlings, J.B.: Application of interior-point methods to model predictive control. J. Optim. Theory Appl. **99**(3), 723–757 (1998)
38. Schenk, O., Gärtner, K.: Solving unsymmetric sparse systems of linear equations with PARDISO. Future Gener. Comput Syst **20**(3), 475–487 (2004)
39. Scheu, H., Marquardt, W.: Sensitivity-based coordination in distributed model predictive control. J. Process Control **21**(5), 715–728 (2011)
40. Scott, J.: Parallel frontal solvers for large sparse linear systems. ACM Trans. Math. Softw. (TOMS) **29**(4), 395–417 (2003)
41. Tanaka, R., Martins, C.: Parallel dynamic optimization of steel risers. J. Offshore Mech. Arct. Eng. **133**(1), 011302–011309 (2011)
42. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Progr. **106**(1), 25–58 (2006)
43. Word, D., Cummings, D., Burke, D., Iamsirithaworn, S., Laird, C.: A nonlinear programming approach for estimation of transmission parameters in childhood infectious disease using a continuous time model. J. R. Soc. Interface **9**(73), 1983–1997 (2012)

44. Zavala, V., Biegler, L.: Large-scale parameter estimation in low-density polyethylene tubular reactors. Ind. Eng. Chem. Res. **45**(23), 7867–7881 (2006)
45. Zavala, V., Laird, C., Biegler, L.T.: Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. Chem. Eng. Sci. **63**(19), 4834–4845 (2008)
46. Zhu, Y. Laird, C.: (2008) A parallel algorithm for structured nonlinear programming. In Proceeding of 5th International Conference on Foundations of Computer-Aided Process Operation, FOCAPO, pp. 345–348
47. Zhu, Y., Legg, S., Laird, C.: (2009) Optimal design of cryogenic air separation columns under uncertainty. Computers & Chemical Engineering 34. Selected papers from the 7th International Conference on the Foundations of Computer-Aided Process Design (FOCAPD)
48. Zhu, Y., Legg, S., Laird, C.: Optimal operation of cryogenic air separation systems with demand uncertainty and contractual obligations. Chem. Eng. Sci. **66**(5), 953–963 (2011)
49. Zhu, Y., Word, D., Siirola, J., Laird, C.: Exploiting modern computing architectures for efficient large-scale nonlinear programming. Comput. Aided Chem. Eng. **27**, 783–788 (2009)