Computer Methods in Biomechanics and Biomedical Engineering
Vol. 7, No. 4, August 2004, pp. 215–225

Taylor & Francis
Taylor & Francis Group

Evaluation of Parallel Decomposition Methods for Biomechanical Optimizations

BYUNG IL KOH[a], JEFFREY A. REINBOLT[b,c], BENJAMIN J. FREGLY[b,c,*] and ALAN D. GEORGE[a]

[a]*Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL 32611, USA;*
[b]*Department of Mechanical & Aerospace Engineering, University of Florida, Gainesville, FL 32611, USA;*
[c]*Department of Biomedical Engineering, University of Florida, Gainesville, FL 32611, USA*

(Received 15 December 2003; In final form 25 June 2004)

As the complexity of musculoskeletal models continues to increase, so will the computational demands of biomechanical optimizations. For this reason, parallel biomechanical optimizations are becoming more common. Most implementations parallelize the optimizer. In this study, an alternate approach is investigated that parallelizes the analysis function (i.e., a kinematic or dynamic simulation) called repeatedly by the optimizer to calculate the cost function and constraints. To evaluate this approach, a system identification problem involving a kinematic ankle joint model was solved using a gradient-based optimizer and three parallel decomposition methods: gradient calculation decomposition, analysis function decomposition, or both methods combined. For a given number of processors, analysis function decomposition exhibited the best performance despite the highest communication and synchronization overhead, while gradient calculation decomposition demonstrated the worst performance due to the fact that the necessary line searches were not performed in parallel. These findings suggest that the method of parallelization most commonly used for biomechanical optimizations may not be the most efficient, depending on the optimization algorithm used. In many applications, the best computational strategy may be to focus on parallelizing the analysis function.

Keywords: Biomechanical optimization; Parallel processing; Decomposition methods; Musculoskeletal models

INTRODUCTION

Optimization algorithms are frequently used to solve system identification or movement prediction problems utilizing complex musculoskeletal models [1–6]. To date, gradient-based, simplex, simulated annealing, genetic and particle swarm algorithms have been used for such applications [1–9]. For large-scale problems, these algorithms have a high computational cost since they evaluate the cost function and constraints iteratively to obtain a converged solution. Moreover, as the complexity of biomechanical systems increases [e.g. increased number of body segments, degrees-of-freedom (DOFs) or controlled muscle forces], the computational expense of simulating the human musculoskeletal system increases dramatically [2]. Even with fast converging optimization algorithms, the complexities of present-day biomechanical models can require thousands of function evaluations to achieve convergence [7,8]. Although the performance of single-processor computers has vastly increased in recent years, computation time can still be a limiting factor.

To circumvent this limitation, the computational load of biomechanical optimization problems can be decomposed and distributed to multiple processors in a parallel computer system. Two general approaches are possible to develop parallel algorithms for biomechanical optimization. The first takes advantage of data independence, an inherent characteristic of many optimization algorithms. Most optimization algorithms evaluate an analysis function (i.e., the cost function and constraints) iteratively to produce a converged solution. Some of these function evaluations have data independent characteristics that can be parallelized. For example, research has been done on gradient-based parallel optimization algorithms that seek to parallelize function evaluations for finite-difference gradient calculations [3–6,9,10]. Similarly, sampling within the search space by non-gradient-based

*Corresponding author. Address: Department of Mechanical & Aerospace Engineering, 231 MAE-A Building, Box 116250, University of Florida, Gainesville, FL 32611-6250, USA. Tel.: +1-352-392-8157. Fax: +1-352-392-7303. E-mail: fregly@ufl.edu

ISSN 1025-5842 print/ISSN 1476-8259 online
DOI: 10.1080/10255840412331290398

algorithms has also been parallelized [8–11]. The second approach seeks to parallelize the analysis function itself so that the workload is distributed to the processors with finer granularity. Together, the optimization algorithm and analysis function can be viewed as the upper and lower levels, respectively, of a two-level process, where either level can potentially be parallelized. To date, no biomechanical studies have sought to parallelize the lower level or both levels simultaneously.

In this paper, we describe three parallel algorithms for biomechanical optimization and present performance evaluations based on level of parallelization: upper level, lower level, and both levels. The concepts are demonstrated using a biomechanical system identification problem for a kinematic ankle joint model. Joint positions and orientations in the body segments that result in the best match to experimental movement data are determined using an unconstrained gradient-based optimization algorithm. The primary emphasis of this work is to demonstrate that lower-level parallelization can be one way to solve biomechanical optimization problems in a reasonable amount of time. Furthermore, it can be the best way to overcome the computational limitations of existing upper-level parallel optimization algorithms. We also show that two-level parallelization can use more available resources in parallel computer systems, even when the optimization problem has a small number of design variables.

COMPUTATIONAL METHODOLOGY

Parallel algorithms seek to distribute the workload evenly across available processors and then gather the results with minimal overhead. The overhead can be communication, synchronization, and other overhead caused by algorithm decomposition. The workload distribution determines the class of parallel algorithm in terms of granularity—coarse-grained or fine-grained. Coarse-grained parallel algorithms have less overhead than fine-grained algorithms but at the cost of frequent load imbalance. By contrast, fine-grained parallel algorithms are more evenly distributed and balanced but generally suffer from higher communication and synchronization overhead.

In this paper, three parallel algorithms are developed: one coarse-grained, one fine-grained, and one medium-grained. The coarse-grained parallel algorithm decomposes an upper-level gradient-based optimization algorithm. The fine-grained parallel algorithm decomposes a lower-level analysis function. The medium-grained parallel algorithm decomposes both levels simultaneously (combined approach of first two parallel algorithms).

The following sections describe the gradient-based optimization algorithm, an ankle joint system identification problem with corresponding analysis function, and the decomposition methods for the three parallel algorithms. A thorough understanding of the optimization algorithm and analysis function is necessary to develop parallel decomposition methods, since parallel algorithms depend heavily on the structure of the algorithms being decomposed. Each parallel algorithm is developed to achieve an evenly distributed workload.

Description of Gradient-based Optimization Algorithm

An unconstrained gradient-based optimization algorithm (Broydon–Fletcher–Goldfarb–Shanno, or BFGS) available in commercial software (VisualDOC, VanderPlaats R & D, Colorado Springs, CO, USA) [12] was used for the upper-level optimization in this study. The BFGS algorithm is a quasi-Newton method that creates an approximation of the inverse Hessian matrix. The Hessian matrix determines the direction used by the algorithm for line searches. The search direction S_k and updated design variables X_k are defined by Eqs. (1) and (2), respectively [12]:

$$S_k = -H^{-1}\nabla F(X_{k-1}) \tag{1}$$

$$X_k = X_{k-1} + \alpha_k S_k \tag{2}$$

where α_k is chosen to minimize $F(X_{k-1} + \alpha_k S_k)$, H the Hessian matrix, X the vector of design variables, $F(X)$ a function evaluation, and subscript k the iteration number. The Hessian matrix is updated by information from line searches and gradient calculations at each iteration [12].

The BFGS algorithm consists of three steps. First, the optimizer creates an approximation of the inverse Hessian matrix, where initially the Hessian matrix is set to the identity matrix. Second, the optimizer calculates gradients to determine the search direction. Third, the optimizer performs a line search to determine how far it can proceed along the search direction. The second and third steps determine the search direction and line search step size, respectively, through function evaluations. The function evaluations for gradient calculations are inherently parallelizable, since no data dependencies are involved.

Description of Analysis Function

A biomechanical system identification problem was used to evaluate three parallel decomposition methods. The problem involves determination of patient-specific parameter values that permit a kinematic ankle joint model to reproduce experimental movement data as closely as possible [2]. Twelve parameters (treated as design variables) specify the fixed positions and orientations of joint axes in adjacent body segments (i.e., shank, talus, and foot) within the three-dimensional, eight DOF kinematic ankle model (Fig. 1). The system identification problem is solved via a two-level optimization approach. Given the current guess for the 12 parameters (i.e., the model structure is fixed), the lower-level analysis function (or sub-optimization) adjusts the generalized coordinates in the model (i.e., the model configuration is varied) so as

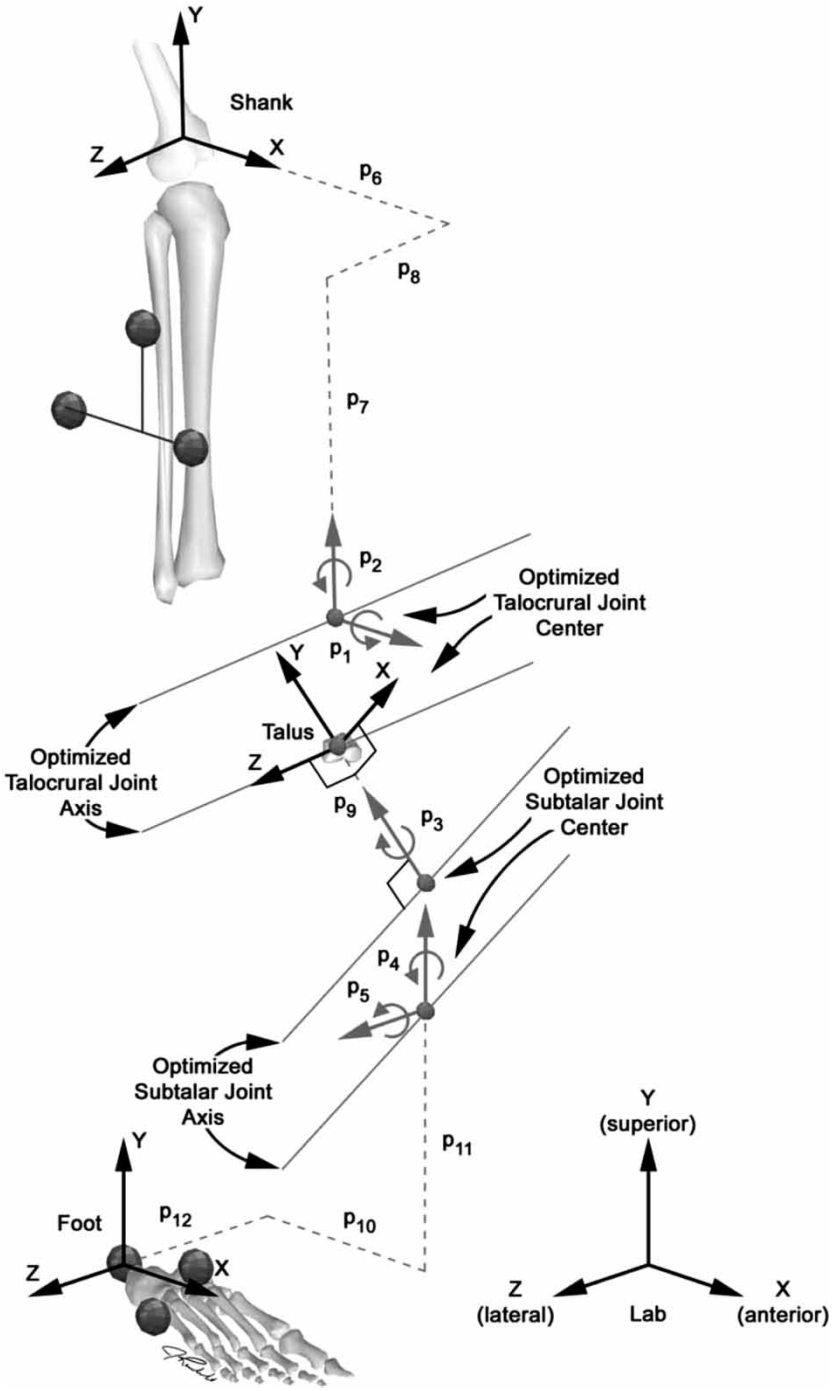

FIGURE 1 Kinematic ankle joint model used in the analysis function. The model is three dimensional, possesses eight DOFs (six for the position and orientation of the shank segment, and two for the talocrural and subtalar joints), and requires 12 parameters (p_1 through p_{12}) to define the positions and orientations of the joint axes in the shank, talus, and foot segment coordinate systems.

to minimize the 3D coordinate errors between modeled and experimental surface markers. During each lower-level optimization, the optimal solution from the previous time frame is used to seed the subsequent time frame. The upper-level optimization adjusts the 12 parameters defining the joint structure so as to minimize the cost function calculated by the lower-level optimization over all time frames. In the present study, this is achieved using the BFGS gradient-based optimizer.

In mathematical form, the above system identification optimization problem can be stated as follows:

$$f(x) = \min_{p} \sum_{t=1}^{nf} e(p,t) \tag{3}$$

with

$$e(p,t) = \min_{q} \sum_{i=1}^{nm} \sum_{j=1}^{3} (a_{ij}(t) - b_{ij}(p,q))^2 \tag{4}$$

where Eq. (3) is the cost function for the upper-level optimization. This equation is a function of patient-specific model parameters p and is evaluated for each of nf recorded time frames. Equation (4) represents the cost function for the lower-level optimization and uses a non-linear least square algorithm to adjust the model's DOFs q to minimize errors between experimentally measured marker coordinates a and model-predicted marker coordinates b, where nm is the number of markers whose three coordinates are used to calculate errors. Hence, the sum of the squares of 3D marker coordinate errors obtained from lower-level optimization of every time frame produces the cost function for the upper-level optimization.

Before developing parallel algorithms, performance of a sequential optimization was evaluated to investigate various possible decomposition methods. Fifty time frames of numerically generated surface marker data from previously reported isolated ankle motion experiments performed with three markers on the foot and shank were used as inputs to the optimization [9]. This approach made it possible to verify that the optimizer recovered the known correct optimal solution. An initial guess was used that was far from the known solution but still within anatomically realistic bounds. All performance results were measured on a Linux-based PC cluster (1.33 GHz Athlons each with 256 MB memory on a 100 Mbps switched Fast Ethernet network) in the High-performance Computing & Simulation Research Laboratory at the University of Florida. Total execution time for gradient-based optimization includes time for gradient calculation function evaluations, line search function evaluations, and other optimization processes (Hessian matrix updating, search direction determination) [13,14]. The number of function evaluations for gradient calculations, line searches, and total sequential execution time is shown in Table I. Sequential optimization for this problem requires over 850 function evaluations at a cost of 12 h of wall clock time, with over 76% of that time due to gradient calculation function evaluations.

TABLE I Number of function evaluations and total execution time (wall-clock time) for sequential optimization

	Gradient calculations	Line searches	Update Hessian matrix/ determine search direction	Total
Number of function evaluations	648	203	–	851
Execution time (s)	32,831	10,285	14	43,131

Execution time for function evaluations in gradient calculations and line searches takes the majority of execution time (99.97%) of sequential optimization. Function evaluations for line searches take 23.85% and gradient calculations take 76.12% of the total execution time.

Description of Decomposition Methods

Gradient Calculation Decomposition

A general approach for decomposing a gradient-based optimization is to distribute the gradient calculation function evaluations evenly to different processors. Since there are no data dependencies involved in these function evaluations, this method is commonly used for parallel gradient-based optimization algorithms [3–6,9,10]. The same decomposition method was implemented for the unconstrained BFGS algorithm using VisualDOC API functions [12] with the Message Passing Interface (MPI) [15]. A master–slave paradigm was used where the master processor had responsibility for running the optimizer and distributing function evaluations.

Gradient calculation decomposition can be visualized using a block diagram for a sample 4-processor system, where each processor performs a finite difference evaluation for 3 design variable sets (Fig. 2). Since the sample system identification problem has 12 design variables, 12 independent function evaluations can be distributed to the available processors. For each gradient calculation, the master processor broadcasts a design variable set (input data) and optimization settings to the slave processors and gathers the results upon completion using MPI send/receive functions. Broadcast and send/receive communications occur only once for each iteration of the upper-level optimization. After completion of the gradient calculations, only the master processor performs line searches while the slave processors sit idle. Since a line search is an inherently sequential process, it was not considered for parallelization in this decomposition method.

Expected parallel performance of this decomposition method depends heavily on the percentage of the time spent on line searches. For the problem analyzed here,

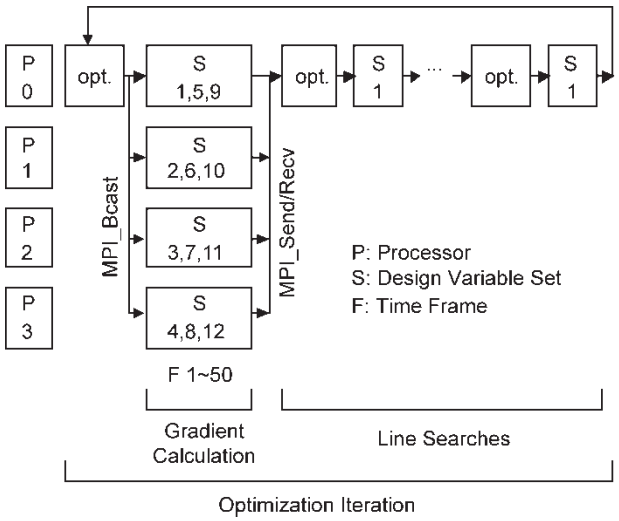

FIGURE 2 Block diagram for gradient calculation decomposition in a 4-processor system. Parallelization is performed only for the optimizer gradient calculations. Each processor evaluates the analysis function for all 50 time frames of data. For gradient calculations, each processor evaluates only 3 of 12 design variable sets, while for line searches, only the master processor (P0) repeatedly evaluates 1 design variable set.

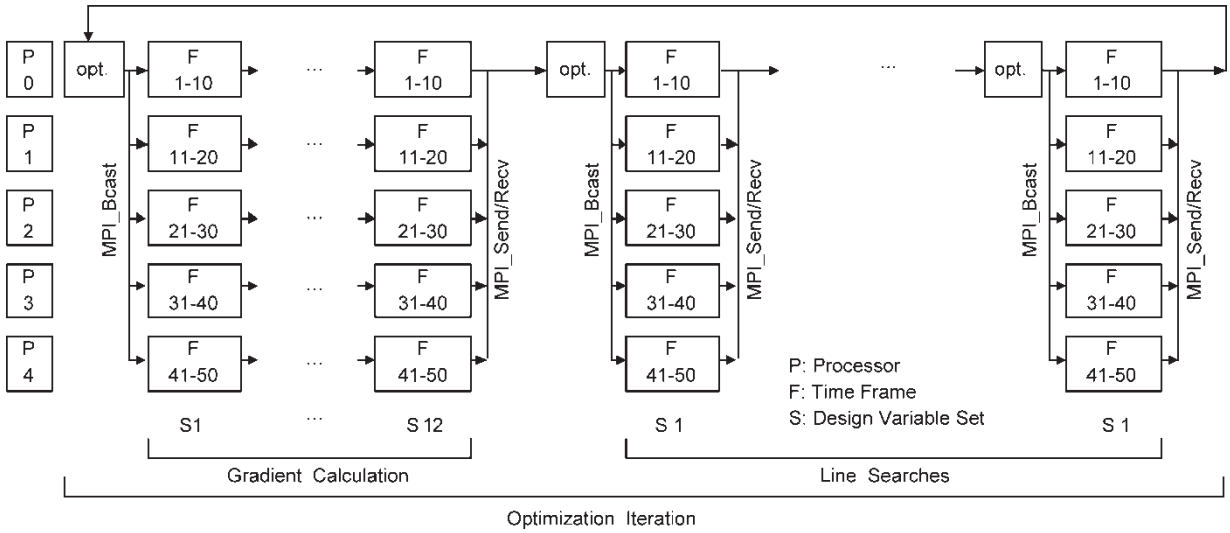

FIGURE 3 Block diagram for analysis function decomposition in a 5-processor system. Parallelization is performed only for the analysis function. Each processor evaluates the analysis function for only 10 of 50 time frames of data. For gradient calculations, each processor evaluates all 12 design variable sets, while for line searches, all 5 processors repeatedly evaluate 1 design variable set.

the parallel portion was not the entire upper-level optimization but only three quarters of it, since gradient calculations took 76% of sequential execution time. The percentage of time spent on gradient calculations would be different for other gradient-based optimization algorithms or other problems. Even though gradient calculation time fraction is expected to increase with the number of design variables, line searches are still required and cannot be parallelized easily with this decomposition method.

Analysis Function Decomposition

Analysis function decomposition is based on knowledge of the independent nature of the sub-optimizations. Execution time for function evaluations takes the majority ($>99\%$) of the total sequential execution time. According to Amdahl's law, performance of a parallel algorithm is limited by the portions of the algorithm that cannot be parallelized [16]. Since a larger percentage of analysis function computations can be parallelized than can optimizer computations, this decomposition method has the potential to produce better parallel performance.

Parallel decomposition at this level required minor modifications to the sample analysis function. As noted earlier, the analysis function performs a separate sub-optimization for each time frame of data, where the solution from one time frame is used to seed the initial guess for the next time frame. To create a parallel version of the same functionality, the analysis function was modified such that the seed for the sub-optimizations was re-initialized to zero after every five time frames. This modification permitted parallelization in blocks of five time frames, had little effect on computational speed, and produced no changes in optimal cost function or design variable values to within the precision of numerical truncation errors.

Analysis function decomposition can be visualized using a block diagram for a sample 5-processor system, where each processor performs sub-optimizations on 10 consecutive time frames (Fig. 3). During gradient calculations, the master processor broadcasts the same 12 design variable sets to all slave processors, and each slave processor executes lower-level optimizations on 2 consecutive sets of 5 time frames for each design variable set. To reduce communication frequency, the slave processors do not send their responses to the master processor after completion of each design variable set. Instead, each slave processor sequentially executes lower-level optimizations for all 12 design variable sets, stores the results, and then sends them as a group to the master processor using MPI send/receive functions. After receiving all responses from the slave processors, the master processor builds up the total response for each design variable set from the 10 time frames analyzed by each slave processor. This approach is used for line searches (1 design variable set) as well as gradient calculations (12 design variable sets).

Communication overhead is larger for this approach than with gradient calculation decomposition because of frequent communication and large message size. The communication frequency is approximately 6 times larger than that of gradient calculation decomposition, since in this case communication occurs before and after line searches as well as gradient calculations. The communication message size is 12 times larger when a 10-processor system is used for analysis function decomposition compared to a 12-processor system for gradient calculation decomposition. However, the benefit is finer granularity than with gradient calculation decomposition, since each processor executes only a portion of the analysis function rather than the entire function. As a result, every slave processor performs virtually the same amount of work for gradient calculations and line searches over the course of the solution.

Combined Decomposition

Combined decomposition is a hybrid approach that combines the benefits of gradient calculation and analysis function decomposition. This approach can be visualized using a block diagram for a sample 10-processor system, where each processor performs sub-optimizations on 10 consecutive time frames for 6 design variable sets (Fig. 4). During gradient calculations, available slave processors are divided into groups (e.g. 2 groups of 5 processors in a 10-processor system) for gradient calculation decomposition, where processors in each group execute assigned lower-level optimizations exactly as in analysis function decomposition. The number of groups depends on the number of design variables, and the number of processors in each group depends on the number of time frames used in the lower-level optimizations. However, during line searches, combined decomposition is identical to analysis function decomposition, since line searches were not parallelized. During that phase, only one group of processors is used and the remaining processors sit idle until the line search is completed.

This decomposition method also has larger communication overhead than does gradient calculation decomposition due to high communication frequency and large message size. However, it has less communication overhead than does analysis function decomposition because of relatively smaller message size and a reduced number of processors involved in communication during line searches. The communication frequency is the same as analysis function decomposition, as communication occurs before and after gradient calculations and line searches. The communication message size is half that of analysis function decomposition during gradient calculations, since the 12 design variable sets are divided into 2 groups. In addition, only half of the slave processors are involved in communication during line searches. This decomposition method has the advantage of using a large number of processors even though only a small number of design variables are used in the analysis function.

Description of Evaluation Metrics

To compare the performance of the three proposed decomposition methods, we performed sequential and parallel optimizations using the ankle joint system identification problem. The same initial guess and optimization parameters, such as finite difference step size and convergence criteria, were used in all optimizations.

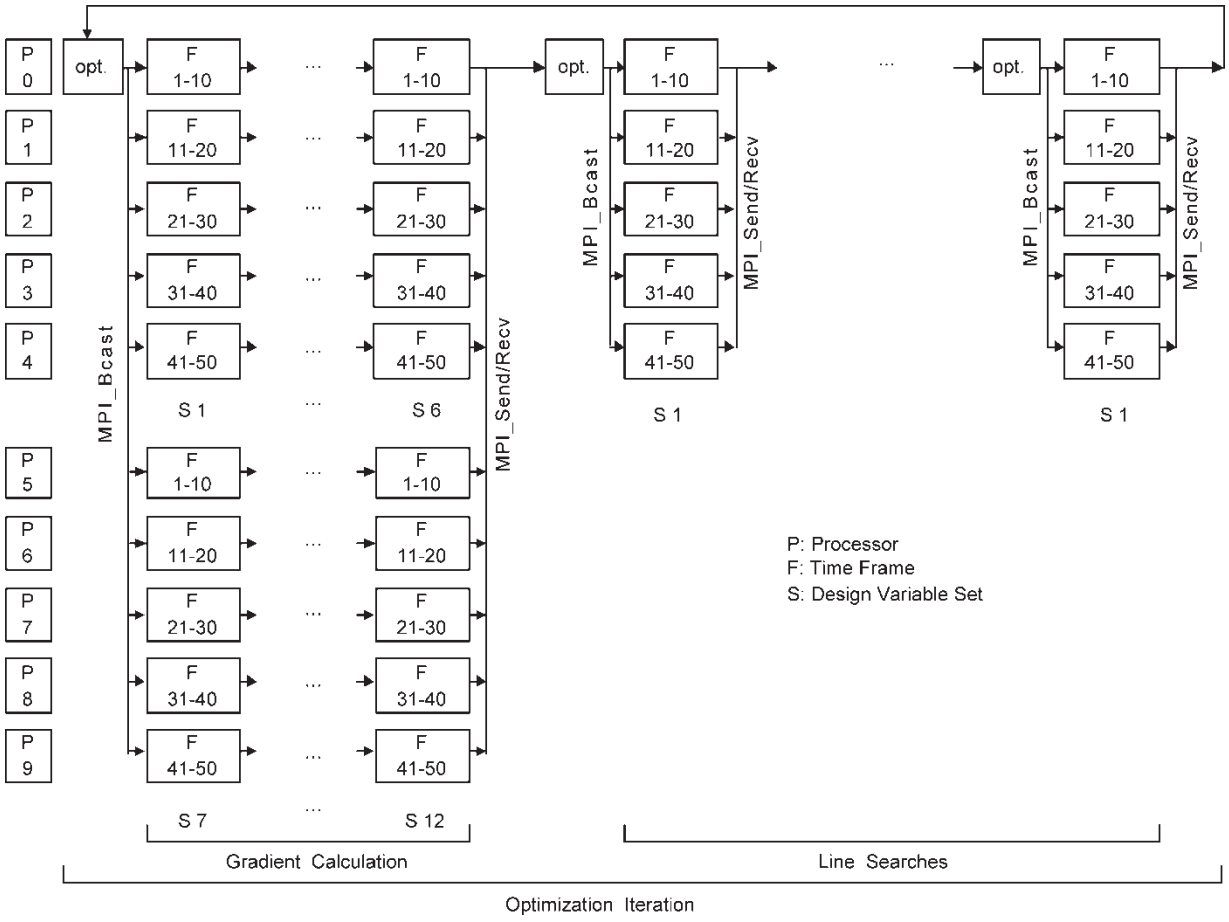

FIGURE 4 Block diagram for combined decomposition in a 10-processor system. Parallelization is performed for both the gradient calculations and the analysis function. Each processor evaluates the analysis function for only 10 of 50 time frames of data. For gradient calculations, each processor evaluates 6 of 12 design variable sets, while for line searches, only 5 of 10 processors repeatedly evaluate 1 design variable set.

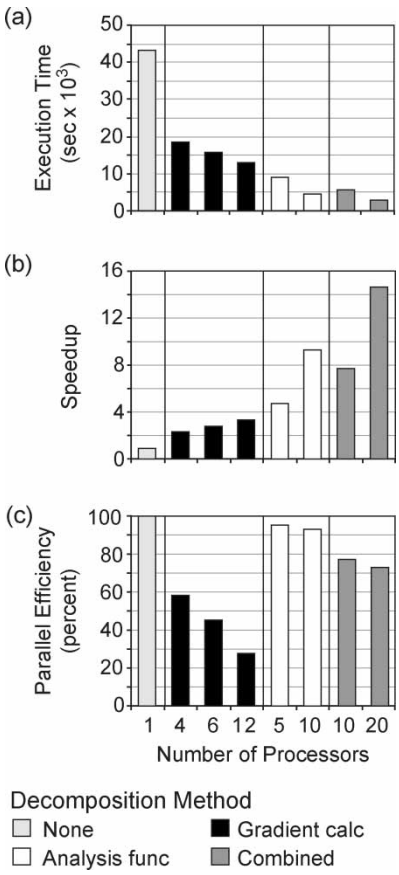

FIGURE 5 Performance metrics for the three parallel decomposition methods as a function of the number of processors. (a) Total execution time due to optimizer computations, communication and synchronization overhead, and analysis function computations. (b) Speedup—the ratio of sequential execution time to parallel execution time. (c) Parallel efficiency—the ratio of speedup to the number of processors. For a fixed number of processors, analysis function decomposition provided the best performance and gradient calculation decomposition the worst.

This method ensured that the sequential and parallel implementations produced identical results. For each decomposition method, the number of processors (≤ 20) was selected to achieve an even workload distribution: 4-, 6- and 12-processor systems for gradient calculation decomposition, 5- and 10-processor systems for analysis function decomposition, and 10- and 20-processor systems for combined decomposition. The maximum number of processors that can be used for gradient calculation decomposition is limited by the number of design variables, putting an upper bound on performance improvements with this method.

For each optimization approach, performance was quantified using three measures: total execution time, speedup (the ratio of sequential execution time to parallel execution time), and parallel efficiency (the ratio of speedup to the number of processors). Speedup and parallel efficiency are particularly important metrics. Ideally, speedup should equal the number of processors (i.e., use of n processors should decrease execution time by a factor of n). In practice, this outcome is almost never achieved, and how close the speedup is to the number of processors is an indication of the parallel efficiency. Parallel efficiencies that decrease rapidly as the number of processors increases indicate increasing overhead due to communication time, synchronization time, or the parallel algorithm itself. The best use of computational resources is achieved when the speedup is close to the number of processors and thus the parallel efficiency is close to 100%.

PERFORMANCE RESULTS

Parallel decomposition at one or both levels resulted in significant performance improvements compared to sequential optimization of the ankle joint system identification problem. Total execution time decreased as the number of processors increased for each of the parallel decomposition methods (Fig. 5a). This decrease was approximately linear with the number of processors for analysis function and combined decomposition but not for gradient calculation decomposition. For comparable numbers of processors, analysis function decomposition was the fastest and gradient calculation decomposition the slowest. Consistent with these observations, speedup was closest to the number of processors for analysis function decomposition and farthest from the number of processors for gradient calculation decomposition (Fig. 5b). Consequently, parallel efficiency was better than 90% for analysis function decomposition with 5 and 10 processors while it dropped from 59 to 28% for gradient calculation decomposition as the number of processors increased from 4 to 12 (Fig. 5c). All decomposition methods exhibited at least some decrease in parallel efficiency with increasing number of processors.

Breakdown of execution time revealed that analysis function computations were the major contributor (Table II). For all three decomposition methods, computation time varied in a manner similar to total execution time. Optimizer time, which is the time required by the master processor to perform optimization algorithm

TABLE II Optimizer, overhead, and computation time breakdown (in seconds) for the three parallel algorithms as a function of the number of processors

Parallel algorithms	Gradient calculation decomposition			Analysis function decomposition		Combined decomposition	
Number of processors	4	6	12	5	10	10	20
Optimizer (s)	14	14	14	14	14	14	14
Overhead (s)	25	29	86	77	155	119	198
Computation (s)	18,468	15,740	12,993	8966	4466	5481	2737

Overhead includes both communication and synchronization times.

tasks such as updating the Hessian matrix and determining the search direction, was negligible and constant across all methods. Overhead for communication and synchronization times increased with the number of processors and was largest for analysis function decomposition and smallest for gradient calculation decomposition. However, it was still small compared to computation time, never accounting for more than 7% of total execution time.

DISCUSSION

This study has presented three approaches for decomposing biomechanical optimization problems for parallel processing. The first decomposes optimizer gradient calculations for distribution to multiple processors, the second decomposes the analysis function (typically a kinematic or dynamic simulation) called repeatedly by the optimizer to calculate the cost function and constraints, and the third decomposes gradient calculations and the analysis function simultaneously. These approaches were demonstrated on an ankle joint system identification problem using a standard gradient-based optimizer. Though gradient calculation decomposition is the most common [3–6,9,10], analysis function decomposition resulted in the most computationally efficient parallel algorithm. Somewhat surprisingly, combined decomposition performed slightly worse than analysis function decomposition for a comparable number of processors, indicating that the additional work to parallelize the optimizer was not helpful for this particular problem. Since parallelization effort is typically spent on the optimizer rather than the analysis function, these results suggest that future parallel biomechanical optimization efforts should consider focusing on the analysis function instead.

The main reason gradient calculation decomposition performed the worst was our inability to parallelize line searches. During a line search, all slave processors sat idle while the master processor performed repeated analysis function calls to determine how far to move in the calculated search direction. This load imbalance grew as the number of processors increased. Since line searches required a significant amount of fixed execution time, total execution time could not decrease linearly with increasing number of processors, causing parallel efficiency to drop. Parallelization of line searches would improve performance significantly [17,18]. For analysis functions with a larger number of design variables, such as Anderson and Pandy's three-dimensional jumping simulation with 432 design variables [4], the effects of line search inefficiencies may be diminished relative to the gradient calculation computation time. Thus, even with gradient calculation decomposition, speedup and parallel efficiency may improve substantially for large-scale problems. The other limitation of this method is that the maximum number of processors is bounded by the number of design variables, which is why only 12 processors were used for this method. This limitation would also be eliminated for large-scale problems with hundreds of design variables, where low overhead due to communication and synchronization would become a more significant advantage.

Analysis function decomposition performed the best primarily because of its finer workload granularity as the number of processors increased. Finer granularity equates to lower likelihood of load imbalances that hurt parallel efficiency. This decomposition method is not limited by the characteristics of the optimization algorithm or the number of design variables and is able to parallelize the largest percentage of the total computations. These benefits outweighed the larger overhead caused by frequent communication and synchronization with large message sizes, since the overhead was still small compared to the required computation time. Despite these benefits, analysis function decomposition only makes sense if the analysis function involves computationally costly calculations such as sub-optimizations or solution of large systems of equations. Otherwise, communication and synchronization overhead may negate the performance benefits of lower-level parallelization. Furthermore, while gradient calculation decomposition can be reused without modification on different optimization problems, analysis function decomposition is specific to a particular problem.

Combined decomposition showed similar performance trends to analysis function decomposition since it includes this decomposition method in its formulation. However, it exhibited slightly worse performance than analysis function decomposition due to the presence of idle processors during line searches. Though an even workload distribution could be obtained by distributing 5 rather than 10 time frames to each processor during line searches, the extra effort to parallelize at both levels would not produce significant additional performance gains.

The results reported in our study are specific to gradient-based optimizers. Non-gradient methods, such as global genetic [8], simulated annealing [7,8] and particle swarm [9–11] algorithms, do not have line searches. For those methods, parallel decomposition of gradient calculations is replaced with parallel decomposition of sampling within the design space. Since more points in design space can be sampled than the number of design variables, the number of processors used for sampling decomposition is not limited by the number of design variables, though other factors may limit parallel efficiency. For example, since the particle swarm algorithm generally works best with 20 particles (i.e., 20 sample points in the design space) regardless of the number of design variables, use of more than 20 processors in a parallel implementation may not produce further performance gains [19]. The drawback of global optimizers is the significantly greater computation time required to obtain a solution. Thus, for problems where a good initial guess can be obtained, gradient-based optimizers still possess a distinct advantage in terms of computation time.

Our results are also specific to a particular kinematic analysis function. Parallel decomposition of our analysis

function was facilitated by the fact that sub-optimization of each time frame could be performed independently from other time frames. In addition to kinematic simulations, analysis functions involving inverse dynamics simulations possess this desirable characteristic. In contrast, forward dynamics simulations of human movement are difficult to parallelize since numerical integration is a sequential process. Wider availability of parallel numerical integrators would be valuable for such applications [20]. Alternatively, forward dynamic optimization problems can sometimes be reformulated as equivalent inverse dynamic optimization problems [21,22], thereby permitting parallelization of the analysis function directly (see Appendix). This concept opens up a variety of parallelization possibilities that have yet to be investigated in the biomechanics community. It is likely that other biomechanical optimization problems of interest could also be reformulated to permit analysis function decomposition.

CONCLUSIONS

In summary, we have presented three parallel algorithms for biomechanical optimization. The algorithms were applied to a biomechanical optimization problem involving system identification of a kinematic ankle joint model. A gradient-based optimizer was used with an analysis function that determined the optimal alignment of the model with experimental movement data given the current guess at the model parameters. Parallelization of optimizer gradient calculations resulted in the worst performance for a fixed number of processors while parallelization of the analysis function resulted in the best performance. Thus, parallelization of the optimizer may not always be the most computationally efficient choice. Significant performance gains for gradient-based optimizers would be obtained by parallelizing line searches as well. An interesting direction for future research would be to apply analysis function decomposition to other computationally intensive biomechanical optimization problems using gradient and non-gradient parallel optimization algorithms. To reduce idle processor time, dynamic load balancing or asynchronous parallel optimization algorithms could be explored to improve parallel performance in heterogeneous environments with different processor speeds. Furthermore, as the number of processors increases, so does the likelihood of processor failures. Thus, fault-tolerant algorithms for parallel optimization will need to be developed and implemented.

Acknowledgements

This study was funded by NIH National Library of Medicine (R03 LM07332) and Whitaker Foundation grants to B.J. Fregly. We thank Vladimir Balabanov at Vanderplaats R & D for valuable suggestions on VisualDOC parallelization.

References

[1] Pandy, M.G., Anderson, F.C. and Hull, D.G. (1992) "A parameter optimization approach for the optimal control of large-scale musculoskeletal systems", *Journal of Biomechanical Engineering* **114**(4), 450–460.

[2] Van den Bogert, A.J., Smith, G.D. and Nigg, B.M. (1994) "*In vivo* determination of the anatomical axes of the ankle joint complex: an optimization approach", *Journal of Biomechanics* **12**, 1477–1488.

[3] Anderson, F.C. and Pandy, M.G. (2001) "Dynamic optimization of human walking", *Journal of Biomechanical Engineering* **123**, 381–390.

[4] Anderson, F.C. and Pandy, M.G. (1999) "A dynamic optimization solution for vertical jumping in three dimensions", *Computer Methods in Biomechanics and Biomedical Engineering* **2**, 201–231.

[5] Anderson, F.C., Ziegler, J.M. and Pandy, M.G. (1995) "Application of high-performance computing to numerical simulation of human movement", *Journal of Biomechanical Engineering* **117**, 155–157.

[6] Pandy, M.G. (2001) "Computer modeling and simulation of human movement", *Annual Reviews of Biomedical Engineering* **3**, 245–273.

[7] Neptune, R.R. (1999) "Optimization algorithm performance in determining optimal controls in human movement analyses", *Journal of Biomechanical Engineering* **121**(2), 249–252.

[8] Knoek van Soest, A.J. and Richard Casius, L.J.R. (2003) "The merits of a parallel genetic algorithm in solving hard optimization problems", *Journal of Biomechanical Engineering* **125**(1), 141–146.

[9] Reinbolt, J.A., Schutte, J.F., Fregly, B.J., Koh, B.I., Haftka, R.T., George, A.G. and Mitchell, K.H. (2004) "Determination of patient-specific multi-joint kinematic models through two-level optimization", *Journal of Biomechanics*, in press.

[10] Schutte, J.F., Koh, B., Reinbolt, J.A., Haftka, R.T., George, A.D. and Fregly, B.J. (2003) "Scale-independent biomechanical optimization". *Proceedings of the Summer Bioengineering Conference*, Sonesta Beach Resort, Key Biscayne, FL, June 25–29.

[11] Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T. and George, A.G. (2004) "Parallel global optimization with the particle swarm algorithm", *International Journal for Numerical Methods in Engineering*, in press.

[12] (2001) *Reference Manual for VisualDOC C/C++ API*, Vanderplaats Research and Development, Inc., Colorado Springs, CO.

[13] Venter, G. and Watson, B.C. (2000) "Exploiting parallelism in general purpose optimization". *Proceedings of the 6th International Conference on Applications of High-performance Computers in Engineering*, Maui, HI, January 26–28.

[14] Venter, G. and Watson, B.C. (2000) "Efficient optimization algorithms for parallel application". *8th AIAA/USAF/NASA/ISSMO Symposium at Multidisciplinary Analysis and Optimization*, Long Beach, CA, September 6–8.

[15] Message Passing Interface Forum (1994) *MPI: A Message-passing Interface Standard*, Technical Report CS-94-230, Computer Science Department, University of Tennessee, April 1.

[16] Culler, D.E. and Singh, J.P. (1998) Parallel Computer Architecture: A Hardware/Software Approach (Morgan Kaufman Publishers Inc., San Francisco, CA).

[17] Eldred, M.S. and Schimel, B.D. (1999) "Extended parallelism models for optimization on massively parallel computers", Paper 16-POM-2 in *Proceedings of the 3rd World Congress of Structural and Multidisciplinary Optimization*, Buffalo, NY, May 17–21.

[18] Byrd, R.H., Schnabel, R.B. and Schultz, G.A. (1988) "Parallel quasi-Newton methods for unconstrained optimization", *Mathematical Programming* **42**, 273–306.

[19] Schutte, J.F. (2001) "Particle swarms in sizing and global optimization", Masters Thesis, University of Pretoria (South Africa).

[20] Jia, Z. and Leimkuhler, B. (2003) "A parallel multiple time-scale reversible integrator for dynamics simulation", *Future Generation Computer Systems* **19**, 415–424.

[21] Nagurka, M.L. and Yen, V. (1990) "Fourier-based optimal control of nonlinear dynamic systems", *Journal of Dynamic Systems, Measurement, and Control* **112**, 17–26.

[22] Lo, J., Huang, G. and Metaxas, D. (2002) "Human motion planning based on recursive dynamics and optimal control techniques", *Multibody System Dynamics* **8**, 433–458.

[23] Anderson, F.C. and Pandy, M.G. (2001) "Static and dynamic optimization solutions for gait are practically equivalent", *Journal of Biomechanics* **34**, 153–161.

APPENDIX

This appendix provides an example of how a forward dynamic optimization problem can be reformulated as an inverse dynamic optimization problem with parallelization of the analysis function to predict muscle activations and periodic motion when no experimental data are available. This is but one additional example of how biomechanical optimization problems can be re-cast in a form that permits analysis function parallelization. The reader is encouraged to think along similar lines for other biomechanical optimization problems of interest.

Consider a dynamic musculoskeletal model possessing n DOFs controlled by m muscles, where $F_j(t)$ is the force generated by any muscle j at time t and $a_j(t)$ is the corresponding muscle activation ($0 \le a_j(t) \le 1$). For simplicity, assume that $F_j(t)$ can be computed by scaling the muscle's peak isometric force F_j^o by its activation [23]:

$$F_j(t) = F_j^o a_j(t) \tag{A1}$$

The net muscle joint torque $T_k(t)$ at any joint k ($k = 1, \ldots, n$) is then a linear combination of the individual muscle forces:

$$T_k(t) = \sum_{j=1}^{m} r_{jk}(t) F_j(t) \tag{A2}$$

where $r_{jk}(t)$ is the moment arm of muscle j about joint k. In turn, the individual net muscle joint torques appear in the equations of motion of the multibody dynamic system (assumed to possess no closed loops for simplicity) as indicated below:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{T}(\mathbf{q}) + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}}) \tag{A3}$$

where $\mathbf{q}$ is the $n \times 1$ column matrix of generalized coordinates, $\mathbf{M}(\mathbf{q})$ the $n \times n$ system mass matrix, $\mathbf{T}(\mathbf{q})$ the $n \times 1$ column matrix due to net muscle joint torques, $\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}})$ the $n \times 1$ column matrix due to centripetal and Coriolis forces, $\mathbf{G}(\mathbf{q})$ the $n \times 1$ column matrix due to gravity forces, and $\mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$ the $n \times 1$ column matrix due to applied external forces and torques.

Thus, given the activation $a_j(t)$ for each muscle j at any time t, Eqs. (A1) and (A2) can be used to calculate $\mathbf{T}(\mathbf{q})$ in Eq. (A3). Based on this model structure, the following forward dynamic optimization problem could be formulated to predict muscle activations that would produce a periodic motion:

$$\text{minimize} \quad \sum_{i=1}^{f} \left(\sum_{j=1}^{m} a_j(t_i)^2 \right)$$

$$\text{subject to} \quad \ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}[\mathbf{T}(\mathbf{q}) + \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})]$$

$$\mathbf{q}(0) = \mathbf{q}(t_f)$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(t_f)$$

$$T_k(t) = \sum_{j=1}^{m} r_{jk}(t) F_j(t), k = 1, \ldots, n \tag{A4}$$

$$F_j(t) = F_j^o a_j(t),\ j = 1, \ldots, m$$

$$a_j(0) = a_j(t_f),\ j = 1, \ldots, m$$

$$0 \le a_j(t) \le 1,\ j = 1, \ldots, m$$

by varying

$$\mathbf{q}(0)$$

$$\dot{\mathbf{q}}(0)$$

$$a_j(t_l),\ j = 1, \ldots, m,\ l = 1, \ldots, p$$

where f is the number of time frames analyzed and p is the number of spline nodal points used to discretize each $a_j(t)$ curve (normally $p \ll f$). This cost function has been shown by Anderson and Pandy [23] to produce muscle force estimates that are similar to those found by minimization of metabolic energy expenditure. The constraints enforce a periodic (though not necessarily symmetric) motion and bound the amplitude of the muscle activations. The design variables are the initial conditions for each DOF along with muscle activation nodal points. For any specified set of design variables, the cost function and constraints in Eq. (A4) are evaluated by calling an analysis function that performs a forward dynamic simulation. Since there are typically more muscles than DOFs in the model, this formulation can result in forward dynamic optimization problems possessing hundreds of design variables [3,4,23].

The key concept for recasting this forward dynamic optimization problem as an inverse dynamic optimization problem is to swap the design variables with the predicted quantities. Instead of placing design variables on the muscle activations and predicting the resulting motion with forward dynamics, one places design variables on the motion and predicts the resulting muscle activations with inverse dynamics. The musculoskeletal model is the same regardless of which formulation is used. The resulting inverse dynamic optimization problem is formulated as follows:

$$\text{minimize} \quad \sum_{i=1}^{f} \left(\sum_{j=1}^{m} a_j(t_i)^2 \right)$$

$$\text{subject to} \quad \mathbf{T}(\mathbf{q}) = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) - \mathbf{F}(\mathbf{q}, \dot{\mathbf{q}})$$

$$\mathbf{q}(0) = \mathbf{q}(t_f)$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(t_f)$$

with suboptimization

$$\text{minimize} \quad \sum_{i=1}^{f} \sum_{j=1}^{m} a_j(t_i)^2 \tag{A5}$$

subject to

$$\sum_{j=1}^{m} r_{jk}(t)F_j(t) = T_k(t), \quad k = 1, \ldots, n$$

$$F_j(t) = F_j^o\, a_j(t), \quad j = 1, \ldots, m$$

$$a_j(0) = a_j(t_f), \quad j = 1, \ldots, m$$

$$0 \le a_j(t) \le 1, \quad j = 1, \ldots, m$$

by varying $\mathbf{q}(t_l),\ l = 1,\ \ldots,\ p$

In the analysis function for Eq. (A5), numerical integration of the equations of motion (forward dynamics) is replaced with repeated algebraic solution of the equations of motion (inverse dynamics) followed by a sub-optimization (quadratic programming) to predict muscle activations from net muscle joint torques. Rather than placing design variables on the muscle activations directly, design variables are placed on spline nodal points describing the time histories of the generalized coordinates. Once the $\mathbf{q}(t)$ nodal points are spline fit, $\dot{\mathbf{q}}(t)$, and $\ddot{\mathbf{q}}(t)$ can be computed at any time frame using the spline coefficients.

There are three advantages and one disadvantage of the inverse formulation compared to the forward formulation. The primary advantage is that the analysis function can be easily parallelized. Once $\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$ and $\ddot{\mathbf{q}}(t)$ are computed, each time frame in the remaining calculations is independent from all other time frames, making it easy to parallelize the inverse dynamics and sub-optimization steps across time frames. Furthermore, even the spline-fitting step can be parallelized across DOFs. A second advantage is that potential problems related to numerical integration—namely system instability and numerical stiffness—are completely eliminated. A third advantage is a reduction in the number of design variables. The forward formulation requires $mp + 2n$ design variables whereas the inverse formulation requires only np, where m is typically two to three times larger than n [3,4,23]. The one disadvantage is the computational cost of repeated sub-optimizations. However, seeding the initial guess for each time frame with the solution from the previous time frame significantly improves convergence speed. Overall computation time per function evaluation may still be less with the numerical integration depending on the number of processors available for the parallelization, the ability to obtain stable forward simulations, and the ability to integrate potentially stiff systems of equations. Note also that if net muscle joint torques rather than muscle activations are to be predicted, the sub-optimizations are eliminated from the inverse approach while the ability to parallelize the analysis function is retained, making this an attractive option for such problems [21,22].