World Academy of Science, Engineering and Technology
International Journal of Mechanical and Mechatronics Engineering
Vol:5, No:3, 2011

Performance Boundaries for Interactive Finite Element Applications

Jaewon Jang[1] and Gregory R. Miller[2]

Abstract—This paper presents work characterizing finite element performance boundaries within which live, interactive finite element modeling is feasible on current and emerging systems. These results are based on wide-ranging tests performed using a prototype finite element program implemented specifically for this study, thereby enabling the unified investigation of numerous direct and iterative solver strategies and implementations in a variety of modeling contexts. The results are intended to be useful for researchers interested in interactive analysis by providing baseline performance estimates, to give guidance in matching solution strategies to problem domains, and to spur further work addressing the challenge of extending the present boundaries.

Keywords—Finite Elements, Interactive Modeling, Numerical Analysis.

I. Introduction

MOST research and development in the general area of computer-based structural analysis has focused on classes of problems pushing the feasibility boundaries of whatever contemporary hardware has been available at the time. Research of this nature has been and continues to be extremely useful, but in this paper the focus is on a different boundary: the boundary between problems that can be solved and visualized using *live modeling* and those that cannot. Live modeling can be defined as a computational approach in which one interacts with an analytical model directly via controls or mouse gestures and observes the results of these actions immediately in an animated, quasi-real time fashion. In effect, the system being modeled appears "live" in the sense that it responds immediately and realistically to input from its environment. Clearly, the feasible live-modeling boundary encompasses a much smaller set of problems than the simply feasible boundary, but both boundaries have been expanding and continue to expand on pace with the general increase in computing power. The purpose of the work presented here, is to characterize the current and emerging feasible boundary for live modeling in a range of structural mechanics contexts.

An obvious fundamental issue is to define a baseline metric for interactive analysis. There can be a significant amount of variation in solution times necessary to achieve suitable interaction rates for live modeling in different contexts. For cases involving haptic as well as visual feedback, refresh rates on the order of hundreds of Hertz are needed [1], standard NTSC video operates at about 30 frames-per-second, while in other instances involving complex visual display in evolving systems, $O(1Hz)$ can be more than adequate. To provide a standardized performance target in this paper, an arbitrary 1-frame-per-second cut-off value is chosen to define the outer limit of the live modeling boundary.

In regards to scope, practical live modeling boundaries will be considered here in terms of two primary issues: (1) problem size; and (2) problem complexity. Problem size refers to the number of degrees freedom of the structural model in question, while problem complexity refers to topology of the structural model (such as sparsity pattern or bandwidth of the system stiffness), conditioning quality of the linear system stiffness, analysis type (whether linear or nonlinear), and so on.

Because of the extremely wide range of problems that can arise in regards to complexity, it is necessary to identify some suitably compact set of cases to enable a reasonably focused study to be completed. At the simplest level of complexity lie problems that are purely linear, with varying load/generalized constraint conditions such that interactive solution is essentially reduced to forward-backward substitution. At the other end of the spectrum lie problems with fully nonlinear geometric and material behavior, evolving topology and loading, and dynamic effects. The principal focus of this work is on linear problems in which load, support, topology, and member properties are all assumed to be variable. Thus, interactivity during these kinds of modifications in general requires the full solution of a standard linear structural analysis problem within the 1-second boundary described above. This scope is of practical relevance in common design situations, and because of the central role such linear solutions play as the key building block in most nonlinear solution schemes, it can provide useful predictive information for these more general contexts, as well.

II. Approach

The overall approach of this study has been to construct a unified platform to test a broad set of solution strategies and algorithms, to verify that the performance of the platform implementations are representative of contemporary high performance solver capabilities, and then to undertake performance testing across several classes of structural modeling contexts. This section provides an overview of these various components of the work.

A. Algorithms and Implementations

The solution strategies used in this study involved both direct and iterative methods, and included Gauss Elimination, Jacobi, Gauss-Seidel, Successive Over-Relaxation, Symmetric

[1]Wolfram Research, 100 Trade Center Drive, Champaign, IL 61820-7237, USA
[1]Department of Civil & Environmental Engineering, University of Washington, Box 352700, Seattle, WA 98195-2700.

Successive Over-Relaxation, Block Jacobi, Incomplete LU Relaxation, Conjugate Gradient, and geometric Multigrid [2], [3], [4], [5], [6]. Various matrix ordering techniques were also used, specifically Reverse Cuthill-Mckee [7], (Approximate) Minimum Degree Ordering [8], and Nested Dissection Ordering [9]. The solution algorithms were implemented and assembled into a single framework for purposes of absolute and relative performance comparison and investigation of suitability for interactive use. By introducing a single implementation framework, the number of variables affecting performance are reduced so that performance tests can be conducted mainly focused on key variables. The framework itself was developed within the broader context of a prototype interactive modeling environment, and so the full range of live modeling issues could be considered.

With few exceptions, implementations were developed essentially from scratch using a variety of sparse storage schemes. The primary departure of this work from existing implementations was to use tensor and vector-based data as the fundamental units of computation. This approach has been used by the authors previously in more limited contexts, and full descriptions of the motivations and implementation issues can be found in [10], [11]. Among other things, this approach can lead to exploitation of inherent natural data blocking, which can improve run-time efficiency, and enables the direct use of coordinate-free element formulations [12], which can simplify and streamline code development.

B. Performance Verifications

To ensure that the performance of the testing platform would be representative of contemporary high-performance implementations, it was compared to existing high performance libraries. External libraries used for this study included `ATLAS` [13], `GotoBLAS` [14], `NIST Sparse BLAS` [15], [16], `LAPACK` [17], `METIS` [9], `AMD` [18], `CHOLMOD` [19], `UMFPACK` [20], `SuperLU` [21], and `PETSc` [22]. These libraries are optimized via special compiler options and inlining technique or machine-specifically tuned using benchmark test suites during compilation. `ATLAS` , `GotoBLAS` , `NIST Sparse BLAS` , and `LAPACK` were used to measure the performance of basic operations while `METIS` , `AMD` , `CHOLMOD` , `UMFPACK` , `SuperLU` , and `PETSc` were used to compare the performance of direct and iterative solvers. Since high performance solver implementations typically combine several layers of libraries—`CHOLMOD` , `UMFPACK` , and `SuperLU` are built on top of `ATLAS` or `GotoBLAS` , and `PETSc` is built on top of `ATLAS` and `LAPACK` —the libraries used here can be considered representative because solvers using similar foundational libraries tend to exhibit similar performance.

In addition to competitive benchmark tests relative to existing the-state-of-art solvers, further benchmarking for identifying performance level of individual solution methods that have been incorporated into the prototype was also performed. Although the primary class of problems considered in this study are those that can be solved within the real-time boundary, large scale problems were also taken into account so as to examine the general scaling behavior of solution algorithms and consider possible usage of iterative algorithms in live-modeling contexts.

Performance comparisons were carried out on a variety of platforms using multiple compilers and a range of problem types and sizes. A partial set of results of these performance comparisons are presented in [10], and more recent and complete results can be found in [23], along with further details of the testing procedures. Because the performance of any solver implementations relies on many factors such as compilers, algorithms [24], CPU speed, cache size, memory hierarchy, front side bus characteristics, and so on [25], performance results are best interpreted as indicative rather than definitive measures. This having been said, the significant outcome of these comparisons for the purposes of this paper is that the implementations developed for this study were found to be competitive with existing high-performance solution packages, and thus can be considered representative of current technologies. This was especially true for problems in the range of sizes amenable to interactive modeling, in which the prototype implementation was frequently found to exhibit the best performance (performance differences between implementations of similar algorithms typically were on the order of 10-15%, with no single implementation being superior in all contexts).

C. 1-Second Performance Boundary Testing

Using the arbitrary 1-second target solution time discussed above as the demarcation limit of interactive analysis, a broad series of testing was performed to investigate the problem size limits for which interactive analysis is feasible. This testing again included the use of numerous solution methods on various platforms in the context of a range of problem types. Because the 1-second boundary tests ultimately could be run on only a limited number of machines, SPEC benchmark [26] results, which are available for hundreds of machines, were investigated as a means to extend the direct test results for a broader class of machines. The outcomes of this investigation will be discussed further in the results section of the paper.

Identification of 1-second boundaries was achieved in a conceptually straightforward manner: problems of a given class were set up with parameterized size variables, and then solutions were obtained for a series of problems of increasing size (in terms of total number of degrees of freedom). When the solution time exceeded one second, a simple linear interpolation was used on the problem sizes bracketing the 1-second boundary, and this was considered the crossover problem size. Because of the inherent granularity of the discrete problem size increases associated with the problem size parameterizations, the crossover problem size values determined in this manner did not actually correspond exactly to an actual mesh or configuration from the problem class in question. The associated “errors” of such an identification scheme, however, are well within the ranges of variability associated with any such performance testing involving multiple compilers, processors, and solution algorithms.

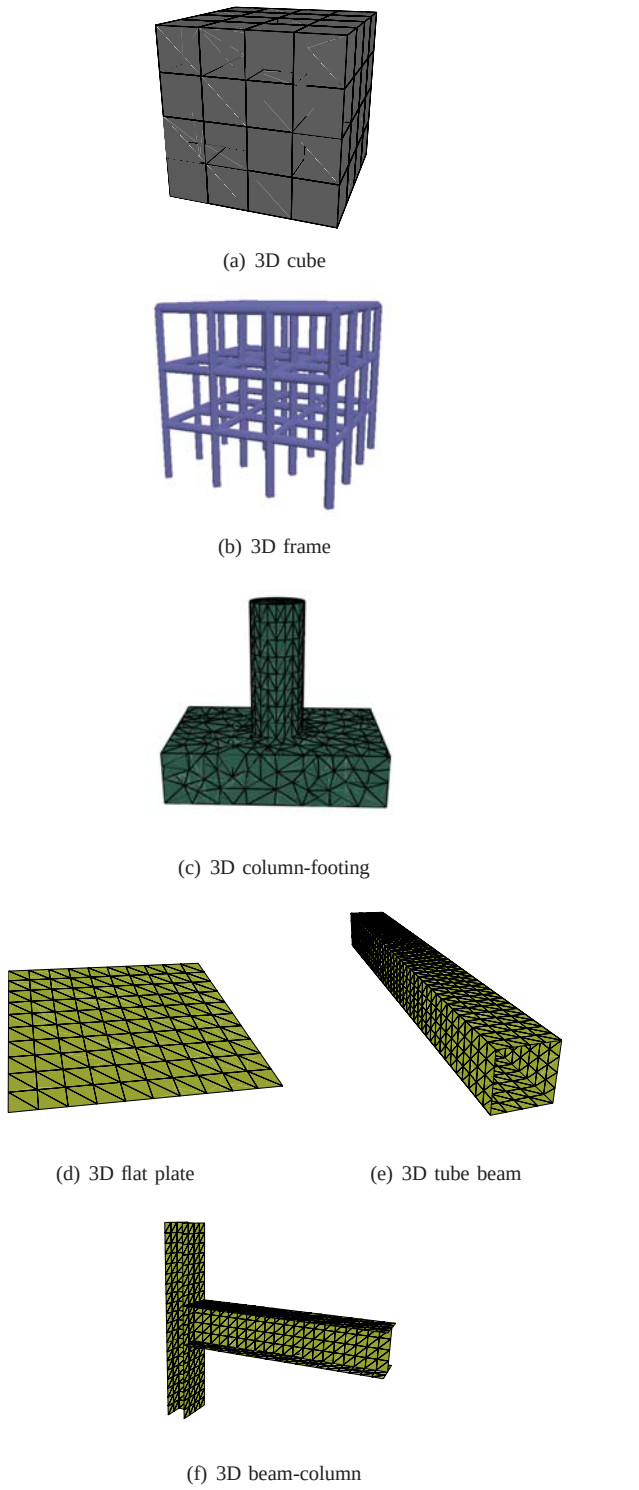
III. 1-Second Boundary Performance Results

In this section, the maximum number of degrees of freedom that can be solved within a nominal 1-second boundary for various structure classes of models using various solvers are identified. The classes of structural models considered are depicted in Figure III. The solvers employed for locating 1-second boundaries in this section are symmetric LU decomposition (LU), Cholesky factorization (CHOL), 2×2 block-based symmetric LU decomposition (2×2LU), incomplete LU preconditioned Conjugate Gradient (ILUPCG), and geometric Multigrid (MG) (see [23] for full implementation details). For the iterative solver, various tolerance levels ($10^{-4} \sim 10^{-12}$) are used because a different level of tolerance can be selected depending on differing levels of accuracy that might be suitable for a particular application. By exploiting this additional level of flexibility of iterative solvers, one can increase the number of degrees of freedom amenable to interactive analysis for cases in which reduced accuracy is acceptable for a given application. An iterative solver with a tolerance of 10^{-12} generates solutions that are close to solutions generated by direct solvers, while reduced accuracy is obtained with a tolerance of 10^{-4}.

The run-time measures presented here include the time for stiffness matrix assembly, stiffness matrix ordering (when appropriate), and linear system solution. For iterative solvers, time for converting data structures into appropriate forms is included (the test implementation uses a linked-list structure to represent stiffness for assembly and some direct solution algorithms, but converts to array-based storage for iterative methods), and the mesh hierarchy set-up time is also included for the Multigrid solver. Model display time is not included because model display time varies depending on the quality of the graphics. The particular test results reported in this section were run on a Pentium Mobile 1.73GHz (64KB L1, 2MB L2, 533MHz FSB, and 1024MB RAM). The GNU gcc 3.4.4 (cygming special) compiler was used with the '-O3' optimization option. Machine-specific optimization options such as '-march=pentium-m' reduces solution time slightly in some cases, but the performance gains were marginal, and thus they were not used here. Although the results presented in the following sections represent a subset of the overall results, they can be considered as representative of the general trends seen in the broader testing.

The particular algorithm/ordering combinations used in the following sections are those that were determined to exhibit the best performance for the case in question. Virtually all reasonable combinations of solvers, ordering schemes, and data structures were considered in identifying the particular top performers presented below. More details of these investigations can be found in [23].

Because the focus of this study was on solution times rather than modeling accuracy, the elements used in the various models were generally standard types, the basic nodal connectivity being the most significant factor in the overall testing. This having been said, in the case of shell elements it is relatively difficult to identify a "standard" type—this study used a shell element based on the work in [27].



(a) 3D cube

(b) 3D frame

(c) 3D column-footing

(d) 3D flat plate

(e) 3D tube beam

(f) 3D beam-column

Fig. 1. Various structural models used to locate 1-second boundaries †

† For all models, elastic modulus (E) of 2.9×10^4 ksi is used. The Poisson's ratio (ν) of model (a), (b), and (c) is 0.3 while the other models, (d), (e), and (f), have ν of 0.25.

A. 3D cube

The solvers employed for detecting 1-second boundaries for the 3D cube model are symmetric LU decomposition (LU), Cholesky factorization (CHOL), 2×2 block-based symmetric LU decomposition (2×2LU), and incomplete LU preconditioned Conjugate Gradient with zero level of fill-in[1] (ILU(0)PCG). The cube is composed of identical numbers of 8-node brick elements in all directions as illustrated in Figure 1(a). The boundary conditions are such that the bottom surface is fully constrained while the opposite surface is loaded vertically. For direct solvers, Nested Dissection Ordering [9], which introduces the smallest number of fill-in elements and the fastest solution time as well, is used to order stiffness matrices, while naturally banded matrices are used for ILU(0)PCG, since natural ordering produces the best ILU preconditioner for this model.

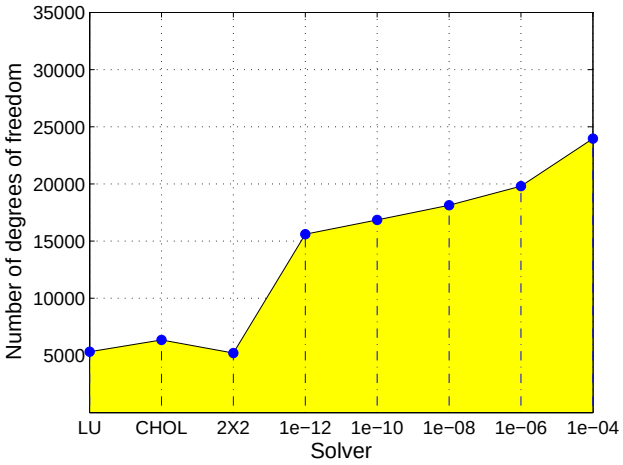


Fig. 2 1-second boundary of the 3D cube model for various direct solvers and varying convergence tolerances for the ILU(0)PCG iterative solver

Figure 2 shows the maximum number of degrees of freedom that can be solved within the 1-second boundary for the 3D cube model. As shown in Figure 2, the iterative solver accommodates a larger number of degrees of freedom than direct solvers for all tolerance levels from 10^{-4} to 10^{-12}. This is because the model has only translational degrees of freedom and the boundary conditions are relatively simple, resulting in well conditioned system stiffness matrices. For example, the 375-degree-of-freedom 3D cube shown in Figure 1(a) with material properties of $E = 2.9 \times 10^4$ ksi and $\nu = 0.3$, and a dimension of 200 in $\times$200 in $\times$200 in has a condition number of 1.31×10^4, while other models shown in Figure III have condition numbers two or three orders of magnitude larger.

B. 3D frame

For the 1-second boundary of the 3D frame model shown in Figure 1(b), Minimum Degree Ordering is used for direct solvers while naturally banded system matrices are used for ILU(0)PCG. The boundary conditions are such that all base nodes are clamped and forces are applied laterally at the top floor. An 8-in diameter circular cross section is used and the length of each member is 50 in, and so these members are relatively stocky.

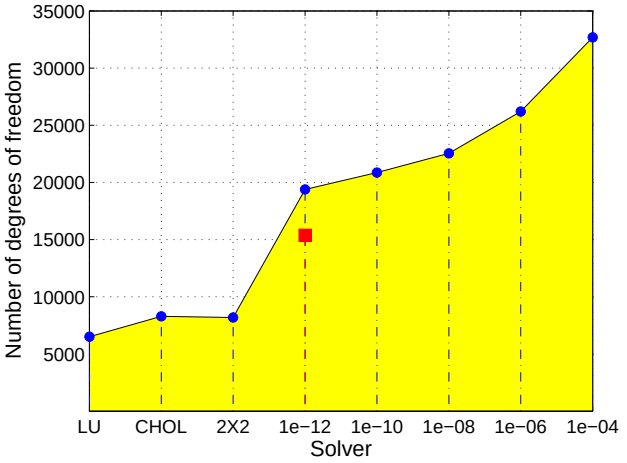


Fig. 3 1-second boundary of the 3D frame model for various direct solvers and varying convergence tolerances for the ILU(0)PCG iterative solver. The inset square corresponds to the use of more slender members, which results in a reduction of the 1-second boundary for the iterative solver

As shown in Figure 3, the 3D frame has the same general trend as in the 3D cube problem: among the direct solvers, CHOL has the largest 1-second boundary, and the iterative solver outperforms direct solvers such that the 1-second boundary of the iterative solver for all tolerance levels is larger than those of the direct solvers. The number of degrees of freedom at the 1-second boundary for ILU(0)PCG with a 10^{-12} tolerance is approximately 19K for the tested frame. However, if a 4-in diameter circular cross section is used, the condition number of the system increases, and the feasible number of degrees of freedom decreases to 15K as illustrated in Figure 3. This illustrates the somewhat fickle nature of iterative solution schemes.

C. 3D column-footing

A 3D column-footing mesh composed of tetrahedron elements shown in Figure 1(c) was generated using an open-source library, Tetgen [28], which uses Delaunay triangulation to generate conforming 3D solid or surface meshes. The boundary conditions of the 3D column-footing are such that all nodes on the top surface of the column are vertically loaded and the bottom surface of the footing is completely constrained.

Two solvers, CHOL and ILU(0)PCG, are used with Nested Dissection Ordering and Reverse-Cuthill Mckee Ordering, respectively. For this example, the block-based direct solvers are not employed due to the problematic sparsity pattern such as shown Figure 4 (an upper triangular part of 3D column-footing matrix with a size of 3,186 3D vector degrees of freedom generated using Tetgen is shown), which is caused by recursively refining certain regions of the 3D column-footing

[1]Zero level fill-in means that the factorization proceeds without introducing any additional entries to the system.

domain. In this case, block-based approaches end up including a large number of zero entries during the block setup phase, and those zero elements add additional computational time. The overall performance of block-based solution algorithms deteriorates, even though their Mflop rate increases.

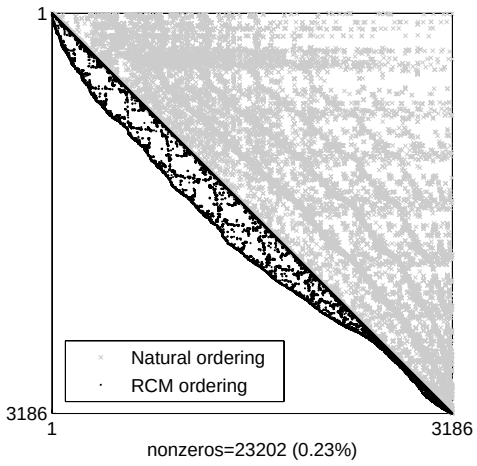


Fig. 4 Sparsity pattern for the 3D column-footing model with a size of 3, 186 3D tensor degrees of freedom generated by recursively refining a model domain

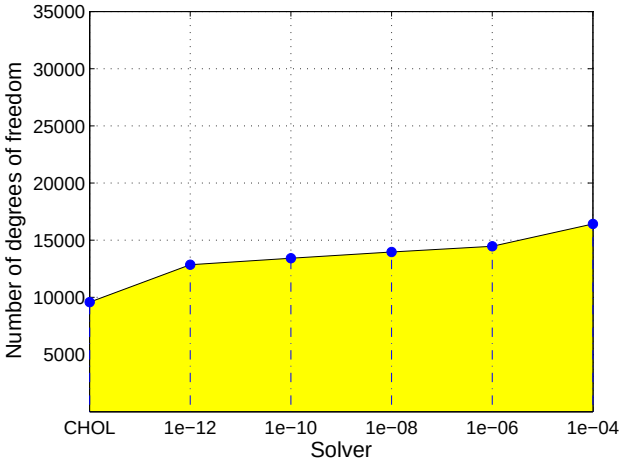


Fig. 5. 1-second boundary of the 3D column-footing model for the Cholesky direct solver and varying convergence tolerances for the ILU(0)PCG iterative solver.

Test results for the 1-second boundaries for the 3D column-footing is shown in Figure 5. As in the 3D cube and the 3D frame model, the iterative solver exhibits better performance than direct solvers for all tolerance levels, but the performance differences between tolerance levels are less dramatic than the previously tested two models. This is due to the difference in convergence history between models. For example, the convergence history of ILU(0)PCG for the 3D cube model is relatively uniform compared to the convergence of ILU(0)PCG for the 3D column-footing, and in general the shape of the 1-second boundary for iterative solvers using on various tolerance levels is very similar to the shape of the convergence history.

D. 3D flat plate

A 3D flat plate model with a ratio of length to thickness of 40.0 in$/1.0$ in$= 40$ such as shown in Figure 1(d) was generated and the 1-second boundaries were located in order to show the effect of loading conditions on 3D structures composed of shell elements. As illustrated in Figure 1(d), the same number of triangular shell elements are used in two directions for the 3D flat plate. The boundary conditions are such that the nodes on one side of the plate are fully constrained, and one of the end nodes of the other side of the plate is loaded with two different loading conditions: one introduces membrane action only and the other also produces out-of-plane bending action. Minimum Degree Ordering is used to renumber matrices for Cholesky factorization while natural ordering is used for the ILU preconditioned Conjugate Gradient $\big($ILU(0)PCG$\big)$.

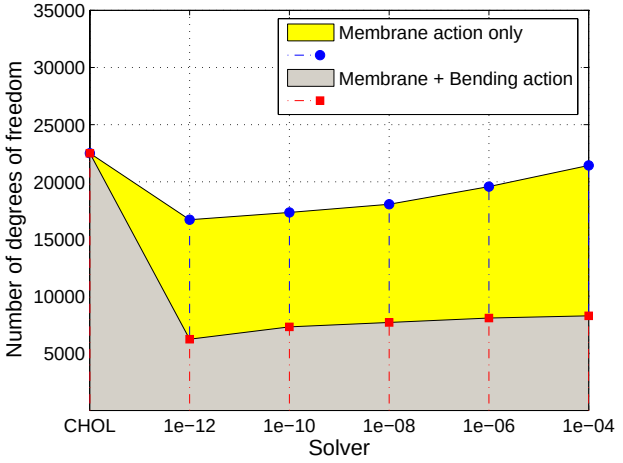


Fig. 6. 1-second boundary of the 3D plate model for the Cholesky direct solvers and varying convergence tolerances for the ILU(0)PCG iterative solver.

Test results are shown in Figure 6. As shown, the maximum number of degrees of freedom for direct solvers are identical for the two loading cases because direct solvers are not affected by conditioning of the stiffness matrix or the complexity of the loading conditions—this is a good example of the robust behavior of direct solution algorithms when they are applied to symmetric positive definite system. The bounding number of degrees of freedom for ILU(0)PCG with a tolerance of 10^{-12} are $16,677$ for the case that introduces only membrane action and $6,235$ for membrane plus bending action. Therefore, it can be said that the performance of iterative solvers for 3D structures comprised of shell elements can be significantly affected by the load condition. For this class of problems the direct solver (CHOL) outperforms the iterative solver $\big($ILU(0)PCG$\big)$ for all tolerance levels.

E. 3D beam-like geometry

In finite element-based civil engineering applications, many problems have beam-like geometric configurations that are

typically modeled using either line elements or shell elements. In both cases, the corresponding system matrices have rotational degrees of freedom that significantly deteriorate the performance of iterative solvers in case the loading conditions are relatively complex.

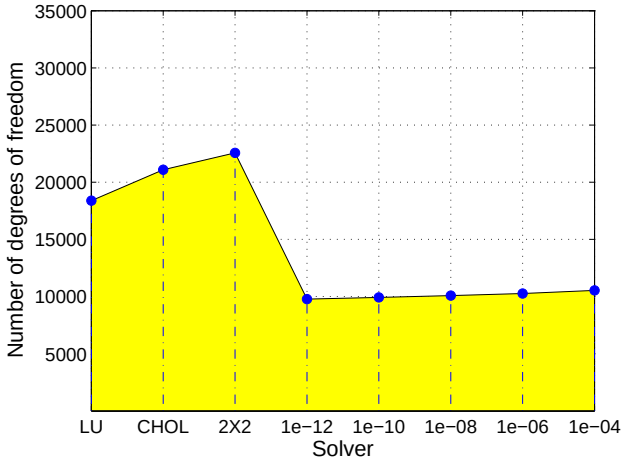
Direct solvers are well suited for this beam-like geometry because the system stiffness typically has a small number of nonzero elements compared to other 3D domains that have approximately the same number of elements in all directions, and because direct solvers are not significantly affected by loading conditions. Iterative solvers typically show relatively poor performance compared to direct solvers for these 3D beam-like problems if a high level of accuracy is required, but they might be employed when a relatively low level of accuracy is acceptable.

A 3D tube cantilever shown in Figure 1(e) and a 3D beam-column shown in Figure 1(f) were used to locate 1-second boundaries for beam-like geometry problems. Dimensions of the 3D beam-column are such that the member lengths are 40 inches, the depths and widths are 8 inches, and the thicknesses are 1 inch, so these are again relatively stocky members, which tends to play to iterative solver strengths. The ratio of length to depth for the 3D square tube cantilever is $80/8 = 10$ with 1-inch wall thickness. For the 3D beam-column, both bottom and top column surfaces are fixed and loads are applied at the tip of the beam, while for the 3D tube cantilever one end is fixed and the other end is loaded vertically. For the two problems, four solvers are used; LU, CHOL, $2\times2LU$, and ILU(0)PCG. Minimum Degree Ordering is used with the direct solvers and natural ordering is used with ILU(0)PCG.

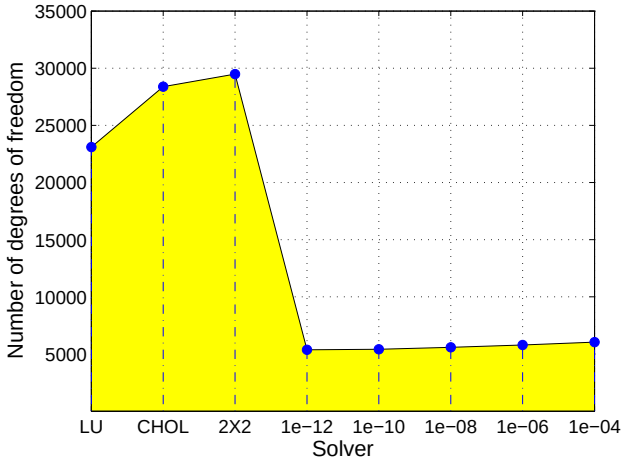
The test results are shown in Figure 7(a) and 7(b). As mentioned, direct solvers outperform the iterative solver for both problems. Among direct solvers, 2×2 block-based symmetric LU decomposition ($2 \times 2LU$) results in a larger 1-second boundary than Cholesky factorization (CHOL) because it is suitable for beam-like geometry (i.e., each node has six degrees of freedom which can be naturally represented using a 2×2 tensor block and the number of nonzeros are relatively smaller than for 3D domains).

An observation made from Figure 7 is that 2×2 LU handles a larger number of degrees of freedom for the 3D beam-column than the 3D tube (i.e., $29,481$ unknowns compared to $22,569$ unknowns), while ILU(0)PCG results in a larger number of degrees of freedom for the 3D tube than the 3D beam-column (i.e., $9,778$ unknowns compared to $5,363$ unknowns). The performance difference between 2×2 LU in Figure 7(a) and Figure 7(b) results from the relative dimensions of the two models. The dimension of the 3D beam-column is roughly equivalent to 80×24, while the dimension of the 3D cube is approximately equal to 80×32—the 3D beam-column consists of six 40×8 plates and the 3D tube is composed of four 8×80 plates. The performance difference using ILU(0)PCG for the two models is a result of the different quality in ILU preconditioner.

So far, the 1-second boundaries presented for iterative solvers have been based on ILUPCG rather than MG even though PCG is more susceptible to loading conditions than



(a) 1-second boundaries of 3D tube



(b) 1-second boundaries of 3D beam-column

Fig. 7. 1-second boundaries of 3D beam-like geometric models

MG, and the required iteration number of PCG for convergence increases with problem size, while that of MG remains relatively constant regardless of the size of a given problem. In spite of these favorable characteristics of MG over PCG, PCG is still dominantly used rather than MG because of its black-box like feature—PCG only requires two inputs, a stiffness matrix and a right hand side vector, and returns a solution vector. MG on the other hand, requires a sequence of meshes which makes it less separable from the problem details. For these reason, PCG is usually employed as a default iterative solver for symmetric positive definite problems for many iterative solver library implementations [29].

In case MG is used to solve beam-like geometry problems, the 1-second boundaries can be increased such as shown in Figure 8, in which the increased 1-second boundaries using MG applied to the 3D cube cantilever is shown. Note that the MG 1-second boundary increases along with the larger tolerance level. It is almost a straight line, since the convergence history of MG is uniform, thus MG could be effective in special purpose modeling in which the method's

coupling with problem details could be accommodated.

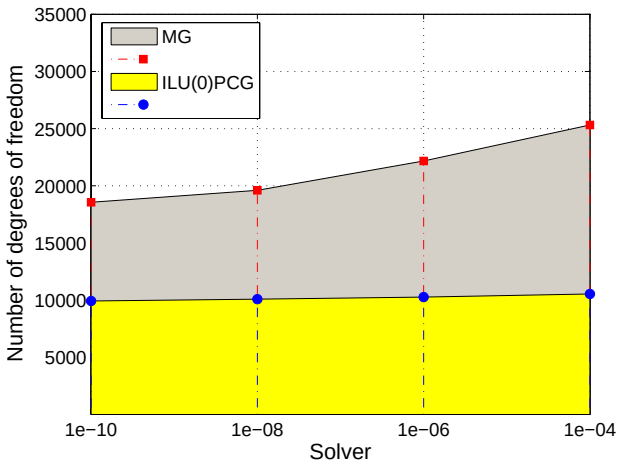


Fig. 8. Increased 1-second boundaries using Multigrid applied to the 3D tube cantilever shown in Figure 7(a)

IV. Factors Affecting 1-second Boundaries

The solution time for the same number of degrees of freedom varies with various factors, and particular factors that govern the performance of direct and iterative solvers typically are different. The four most important factors that govern the performance of solution algorithms are (i) the number of nonzero elements, (ii) the sparsity patterns of the stiffness matrices, (iii) the conditioning of the stiffness matrices, and (iv) the loading condition.

First, the number of nonzeros affects the solution time of both direct and iterative solvers such that the possible number of degrees of freedom for models having a small number of nonzeros are larger than models with large number of nonzeros. For iterative solvers, the amount of work for solving a linear system is roughly equivalent to the number of nonzeros times the iteration number—that is the required amount of work for n times of matrix-vector multiplications.

Second, as mentioned, the sparsity pattern of the stiffness matrix governs the performance of direct and iterative solvers. In general, the sparsity structure of the system matrix is rearranged in order to reduce the number of fill-ins during matrix factorization or in order to increase the quality of the preconditioner and possibly spacial locality of data.

Third, the performance of direct solvers is essentially unaffected by conditioning of the stiffness matrix, while that of iterative solvers is highly sensitive to this quantity. A numerical example used to illustrate significance of conditioning of the system matrix is a 3D tube such as shown in Figure 1(e), which has $80 \times 8 = 640$ elements for each plate. Two models are generated such that the one has an identical elastic modulus of 29000 for all triangular shell elements, and the other has a variety of elastic moduli between 1 to 32767. The iteration number for ILU(0)PCG to converge up to the tolerance of 10^{-12} is 292 for the identical elastic modulus model and 319 for the random elastic modulus model. The corresponding condition number for the two cases are 4.677×10^5 and 6.454×10^5 showing that the 3D tube with various elastic modulus is relatively poorly conditioned.

Fourth, as already shown in the 3D flat plate example, loading conditions can significantly deteriorate the performance of iterative solvers. Experimental tests performed for this study indicate that structural models having only translational degree of freedom usually are not strongly affected by loading conditions, but models with rotational degrees of freedom may significantly affected.

In summary, among the four factors that govern the performance of solution methods, sometimes significantly and sometimes insignificantly, the number of nonzero elements affects the performance of both direct and iterative solvers. The sparsity pattern seriously influences direct solvers but has relatively little impact on iterative solvers. Conditioning of stiffness and loading condition generally only affect iterative solvers. Based on these observations, it can be concluded that in some instances, iterative solvers could allow for significant expansion in the size of problems suitable for interactive analysis. However, these instances would be limited to cases in which the range of models was restricted (e.g., special-purpose contexts with preset configurations). For general purpose interactive modeling, direct solvers are likely to be preferable.

V. Performance Prediction on Various Machines

Although the trends and observations presented above are fairly general, the numerical 1-second boundary test results presented in the previous section are strictly valid only for a particular processor in a particular machine. It is clearly desirable to expand the test results for a variety of computer systems. Although multiple machines were considered in this study, performing truly comprehensive timing test on all currently available machines is not realistic. As an alternative, one might predict the performance on different machines by using a simple extrapolation based on a simple metric like processor clock speed. However, this would not be reliable because the performance of the computer system is not solely dependent on CPU clock speed, but rather is determined by complex interactions between various factors such as CPU speed, cache sizes, and a front side bus speed for floating point intensive scientific computations [25].

A better alternative to predict approximate performance on diverse computer systems is to use reliable external benchmark results. SPEC (Standard Performance Evaluation Corporation) CPU2000 has been chosen in this regard [26]. Since the CPU2000 benchmark only measures the performance of computer systems based on CPU, memory architecture, and compiler (it does not measure the performance of I/O, graphics, networking, and so on), it is very similar to the performance tests that have been conducted for this study. In addition, there are more than $4,000$ benchmark test results available and all results are verified by SPEC team. The CPU2000 benchmark suite is composed of two groups of tests, SPECint and SPECfp for integer and floating point operation, respectively. Since finite element applications are dominated by floating point operations, the floating point benchmark

suite, SPECfp, is chosen to estimate the 1-second boundaries on other machines. Among the 14 sets of benchmark tests in the SPECfp suite, 172.mgrid and 191.fma3d are selected due to their similarity to problems used in the previous section. The 172.mgrid is based on a 3D potential field problem with a simple Multigrid solver [30], and the 191.fma3d simulates the inelastic dynamic response of a 3D solid subjected to impulsively applied forces [31].

A small subset of SPECfp benchmark scores are reproduced in Table I with relevant hardware (CPU speed, cache sizes, and front side bus speed), operating system, and compiler information. As part of the present study, performance of iterative solvers was compared to an external reference, PETSc , using two machines, a Pentium III 1GHz and IBM eServer pSeries 630 1.45GHz. The SPECfp benchmark results for these exact machines are also included in Table I. As listed in Table I, the SPECfp performance ratio of IBM to Pentium is $769/215 = 3.58$ for 172.mgrid, and $792/290 = 2.73$ for 191.fma3d. In our testing the average calculated performance factor using incomplete Cholesky preconditioned Conjugate Gradient (ICCPCG) between IBM and Pentium were 4.09 for the 2D 4-node case and 2.56 for the 3D 8-node case. The matrices used in the 172.mgrid benchmark represent a scalar field of an constant coefficient equation discretized using a finite difference method [30]. Therefore, their sparsity patterns and conditioning of matrices are similar to 2D 4-node finite element matrices. On the other hand, the matrices of the 191.fma3d benchmark are generated from finite element discretization using various solid elements [31], hence they are approximately equivalent to 3D 8-node matrices. Comparing the appropriate predicted performance ratio to that actually observed via direct testing in this representative example illustrates that SPECfp benchmark estimates of 1-second boundaries on different hardware can be obtained that generally are within 15% of actual values (12% ($3.58/4.09 = 0.88$) and 6% ($2.73/2.56 = 1.06$) in this particular case).

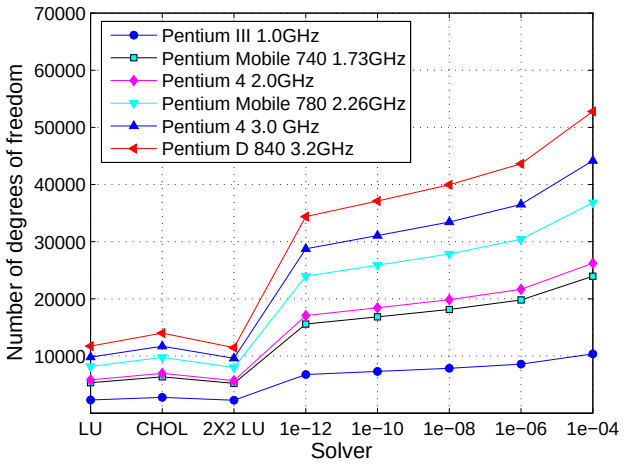


Fig. 9. Estimated number of degrees of freedom within 1 second boundary on a variety of CPUs for the 3D cube shown in Figure 1(a)

Based on the scaling estimation described above, extrapolated estimates for 1-second boundary for 3D cube models on various machines using SPECfp ratios are shown selectively in Figure 9. Based on the results shown in Figure 9, it can be seen that using the fastest current uniprocessor machines it is possible to perform interactive linear analyses on 3D problems ranging from $10,000$ to close to $65,000$ degrees of freedom.

The results in Figure 9 are generally based on best-case scenarios, and so they are best interpreted as representing upper (i.e., optimistic) bounds on interactive analysis. It is worth noting that to the degree single-processor CPU speed appears to have plateaued in recent years, these single cpu values could be valid for some time. These values also can be used to estimate upper bound performance limits for multiprocessor systems, simply by scaling by the number of processors. Because true order-N speedup is generally not possible to achieve, especially for smaller problems, a scaling factor other 1.0 would be appropriate. However, for a given problem type/size combination, one can use the estimates in Figure 9 to get at least a rough idea of how many processors one might need to bring solution times to within the interactive range.

Table II presents a summary overview of interactive problem size ranges for the fundamental classes of linear problems considered here. The table shows problem size limits for which 1-second solution times were achievable using the best combination of solver and data structure, and includes an indication of which class of algorithm corresponds to the given result. The tested processor values correspond to those obtained using the reference machine, and SPEC-based estimated limits are provided, as well. As discussed earlier in this paper, there can be significant variability in any such performance figures, so these results should be considered indicative rather than definitive. This having been said, the tested results provide reasonable bounds for performance prediction.

Several conclusions can be drawn from Table II:

- In general, the best direct solvers outperform the best iterative solvers for classes of problems in which conditioning of the system stiffness is not ideal. This most notably involves shell models (e.g. 3D tube cantilever).
- Direct solvers show better performance than iterative solvers in most cases for the problems having beam-like geometric configuration (relatively long one or two directions than other direction) (e.g. 3D tube cantilever and 3D beam-column).
- For domains having similar numbers of elements in all directions, iterative solvers generally show the better performance even for problems that can be solved within the 1-second boundary (e.g. 3D cube and 3D frame).

These conclusions are similar to one would expect for general finite element computing contexts.

VI. Summary and Conclusions

This paper has identified and characterized interactive performance boundaries for a range of structural models and solution algorithms. The key benchmark driving the study was a solution-time limit of 1 second, which was used to define and

TABLE I
SPECFP2000 SCORES FOR VARIOUS MACHINES (EXCERPTED FROM [26])

		172.mgrid	191.fma3d
Pentium III 1.0GHz:	(16+16)KB L1, 256KB L2, 133MHz Bus, Windows 2000, Intel 5.0	215	290
Pentium 4 1.7GHz:	(12+8)KB L1, 256KB L2, 400MHz Bus, Windows XP, Intel 6.0	647	659
Pentium 4 2.0GHz:	(12+8)KB L1, 256KB L2, 400MHz Bus, Windows XP, Intel 6.0	715	733
Pentium Mobile 755 2.0GHz:	(32+32)KB L1, 2MB L2, 400MHz Bus, Windows XP, Intel 8.0	768	812
Pentium Mobile 780 2.26GHz:	(32+32)KB L1, 2MB L2, 533MHz Bus, Windows XP, Intel 9.0	1122	1029
Pentium 4 3.0GHz:	(12+8)KB L1, 512KB L2, 800MHz Bus, Windows XP, Intel 7.1	1197	1235
Pentium D 840 3.2GHz	(12+16)KB L1, 1MB L2, 800MHz Bus, Windows XP, Intel 9.0	1443	1476
IBM eServer 630 1.45GHz:	(64+32)KB L1, 1536KB L2, 8MB L3, AIX 5.2 64-bit, xlc 8.1.0	769	792

TABLE II
TESTED AND ESTIMATED 1-SECOND BOUNDARIES FOR CURRENT REPRESENTATIVE MACHINES.

Model	Pentium Mobile 1.73GHz		Pentium D 840 3.2GHz	
	# dofs	Solver	# dofs (estimated)	Solver
3D cube	6.35K	CHOL	14.00K	CHOL
	15.60K	ILUPCG	34.47K	ILUPCG
3D frame	8.28K	CHOL	18.30K	CHOL
	15.37K	ILUPCG	33.87K	ILUPCG
3D column-footing	9.59K	CHOL	21.12K	ILUPCG
	12.85K	ILUPCG	28.31K	ILUPCG
3D flat plate	22.50K	CHOL	49.57K	CHOL
	16.68K	ILUPCG	36.74K	ILUPCG
3D tube cantilever	22.57K	2×2 LU	49.72K	2×2 LU
	9.78K	ILUPCG	21.54K	ILUPCG
	18.55K	MG	40.87K	MG
3D beam-column	29.48K	2×2 LU	64.95K	2×2 LU
	5.36K	ILUPCG	11.81K	ILUPCG

investigate interactive performance. Structural models ranging from line elements to 2D and 3D continuum and shell elements were considered, with a primary focus on the fundamental task of solving linear problems in contexts for which full solutions are necessary. Solution algorithms included a broad set of direct and iterative implementations, virtually all of which were written from the ground up to support tensor-based computations and to function well in interactive contexts. Single-CPU architectures were the primary computing environment considered, using a combination of direct testing and SPECfp2000-based estimation to cover the current range of processors in typical desktop use.

The results show that the 1-second solution boundary ranges over problem sizes of about 10,000-60,000 degrees of freedom depending on problem type and solution algorithm. Increasing these bounds via simple increases in processor speeds is no longer likely to occur as in the past—rather, techniques suitable for multicore CPUs and related architectures will need to be exploited.

REFERENCES

[1] A. Lindblad, "Increasing the functionality of finite element based surgical suturing simulators," Ph.D. dissertation, University of Washington, 2006.
[2] A. George and J. W.-H. Liu, *Computer solution of large sparse positive definite systems*. Prentice Hall, Inc., 1981.
[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the solution of linear systems: Building blocks for iterative methods*, 2nd ed. Siam, 1994.
[4] J. W. Demmel, *Applied numerical linear algebra*. Siam, 1997.
[5] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*, 2nd ed. Siam, 2000.
[6] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed. Siam, 2003.
[7] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proc. 24th Nat. Conf. ACM*, 1969, pp. 157–172.
[8] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *Siam J. Matrix Anal. Appl.*, vol. 17, no. 4, pp. 886–905, 1996.
[9] G. Karypis, *METIS*, http://glaros.dtc.umn.edu/gkhome/metis/metis/overview, 2006.
[10] G. R. Miller, P. Arduino, J. Jang, and C. Choi, "Localized tensor-based solvers for interactive finite element applications using C++ and Java," *Comput. Struct.*, vol. 81, no. 7, pp. 423–437, 2003.
[11] M. D. Rucki and G. R. Miller, "An algorithmic framework for flexible finite element-based structural modeling," *Comp. Meth. Appl. Mech. Engrg.*, vol. 36, no. 3-4, pp. 363–384, 1996.
[12] G. R. Miller, "Coordinate-free isoparametric elements," *Comput. Struct.*, vol. 49, pp. 1027–1035, 1993.
[13] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimizations of software and the ATLAS project," *Parallel Comput.*, vol. 27, no. 1-2, pp. 3–35, 2001.
[14] K. Goto, *GotoBLAS*, Texas Advanced Computing Center, Univ. of Texas at Austin, 2006, http://www.tacc.utexas.edu/resources/software/software.php.
[15] K. A. Remington and R. Pozo, *NIST Sparse BLAS*, National Institute of Standards and Technology, 1996, http://math.nist.gov/spblas.
[16] J. J. Dongarra, *Basic linear algebra subprograms technical (BLAST) forum standard*, Univ. of Tennessee, Knoxville, Aug. 2001, http://www.netlib.org/blas/blast-forum/.
[17] E. Anderson, Z. Bai, C. Bischof, S. Blackford, and J. Demmel, *LAPACK users' guide*, 3rd ed., Siam, 1999, http://www.netlib.org/lapack/.
[18] P. R. Amestoy, T. A. Davis, and I. S. Duff, *AMD version 2.0 user guide*, Dept. of Computer and Information Science, Univ. of Florida, 2006, http://www.cise.ufl.edu/research/sparse/amd.
[19] T. A. Davis and W. W. Hager, "Dynamic supernodes in sparse cholesky update/downdate and triangular solves," CISE Dept, Univ. of Florida, Tech. Rep. TR-2006-004, 2006, http://www.cise.ufl.edu/research/sparse/cholmod.
[20] T. A. Davis, *UMFPACK version 5.0 user guide*, Dept. of Computer and Information Science, Univ. of Florida, May. 2006, http://www.cise.ufl.edu/research/sparse/umfpack.
[21] J. W. Demmel, J. R. Gilbert, and X. S. Li, *SuperLU user's guide*, Univ. of California at Berkeley, 1999, http://www.cs.berkeley.edu/ demmel/SuperLU.html.
[22] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik,

M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 2.1.5, 2004.
[23] J. W. Jang, "Characterization of live modeling performance boundaries for computational structural mechanics," Ph.D. dissertation, University of Washington, 2007.
[24] S. S. Skiena, *The algorithm design manual*. Springer-Verlag New York, Inc., 1998.
[25] D. A. Patterson and J. L. Hennessy, *Computer organization and design: The hardware/software interface*, 3rd ed. Morgan Kaufmann Publishers, 2005.
[26] SPEC Team, *SPEC CPU2000 v1.3 documentation*, Standard Performance Evaluation Corporation, 2001, http://www.spec.org/cpu2000.
[27] C. A. Felippa, "A study of optimal membrane triangles with drilling freedoms," University of Colorado, Tech. Rep. CU-CAS-03-02, 2003.
[28] H. Si, *TetGen: A quality tetrahedral mesh generator and three-dimensional delaunay triangulator version 1.4*, Jan. 2006, http://tetgen.berlios.de/.
[29] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2001, http://www.mcs.anl.gov/petsc.
[30] D. Bailey, T. Harris, and W. Saphir, "The NAS parallel benchmark 2.0," NASA Ames Research Center, Tech. Rep. NAS-95-020, 1995.
[31] S. W. Key and C. C. Hoff, "An improved constant membrane and bending stress shell element for explicit transient dynamics," *Comp. Meth. Appl. Mech. Engrg.*, vol. 124, pp. 33–47, 1995.