A Deflated Assembly-Free Approach to Large-Scale Implicit Structural Dynamics

Amir M. Mirzendehdel

Krishnan Suresh

suresh@engr.wisc.edu

Department of Mechanical Engineering
UW-Madison, Madison, Wisconsin 53706, USA

ABSTRACT

The primary computational bottle-neck in implicit structural dynamics is the repeated inversion of the underlying stiffness matrix. In this paper, a fast inversion technique is proposed by merging four distinct but complementary concepts: (1) voxelization with adaptive local refinement, (2) assembly-free (a.k.a. matrix-free or element-by-element) finite element analysis, (3) assembly-free deflated conjugate gradient, and (4) multi-core parallelization. In particular, we apply these concepts to the well-known Newmark-beta method, and the resulting *assembly-free deflated conjugate gradient* (AF-DCG) is well-suited for large-scale problems. It can be easily ported to many-core CPU and multi-core GPU architectures, as demonstrated through numerical experiments.

1. INTRODUCTION

The focus of this paper is on large-scale structural dynamics, where one is interested in transient analysis of flexible and geometrically complex elastic bodies such as the one in Figure 1. Specifically, given an external force that varies over time, the objective is to find the displacements, stresses, etc. within the body, as a function of time. Transient analysis is critical, for example, in impact studies, predicting fatigue-life, crack-propagation studies, etc.

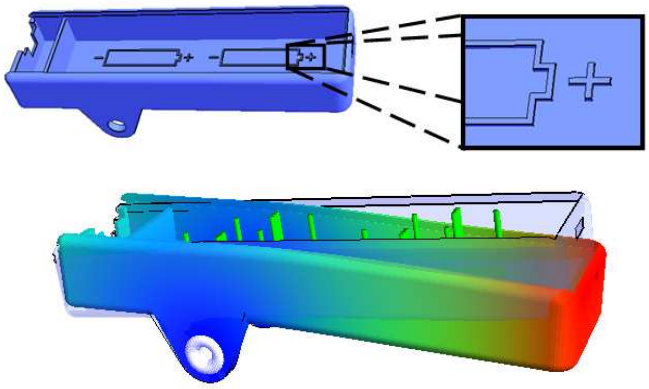


Figure 1: An example of transient analysis of a thin elastic structure.

A standard approach to structural dynamics of flexible bodies is to discretize the geometry via finite elements; this results in a system of second order differential equations in time [1]

$$M\ddot{u} + C\dot{u} + Ku = f^{ext} \tag{1.1}$$

where:

$$\begin{aligned} &n : \text{Number of degrees of freedom} \\ &M_{n\times n} : \text{Mass matrix} \\ &C_{n\times n} : \text{Damping matrix} \\ &K_{n\times n} : \text{Stiffness matrix} \\ &f^{ext}_{n\times 1} : \text{External Force vector} \\ &u_{n\times 1} : \text{Displacement field} \\ &\dot{u}_{n\times 1} : \text{Velocity field} \\ &\ddot{u}_{n\times 1} : \text{Acceleration field} \end{aligned} \tag{1.2}$$

In this paper, without loss of generality, we shall assume proportional damping:

$$C = \alpha^d M + \beta^d K \tag{1.3}$$

where α^d & β^d are the damping coefficients.

Equation (1.1) is typically solved through time-stepping, either via an explicit or an implicit method. In explicit methods, the solution at time t is used to obtain the solution at $t+\Delta t$. This, as it turns out, entails the inversion of the mass matrix M [2] (by inversion, we mean solving linear systems of equations governed by the underlying matrix). Since M can often be diagonalized, its inversion is trivial, leading to rapid time-stepping. However, explicit methods are unstable for large time steps Δt.

On the other hand, implicit methods such as the Newmark-beta method are unconditionally stable, but require the 'inversion' of an effective stiffness matrix [1], a computationally demanding task. In this paper, we explore a new method for fast 'inversion' of this stiffness matrix.

In Section 2 the Newmark-beta is summarized, followed by a review of the literature relevant to this paper. In Section 3, the proposed method is discussed, together with its CPU and GPU implementation. Numerical experiments are presented in Section 4. Conclusions and future work are covered in Section 5.

2. LITERATURE REVIEW

2.1 Newmark-beta Method

One of the most popular implicit methods in structural dynamics is the Newmark-beta method. In 1959, Newmark formulated this method [3] by introducing two numerical unit-less scalar parameters β and γ $\left(0 \le \beta \le 0.5\ ,\ 0 \le \gamma \le 1\right)$. The desired quantities at time $t+\Delta t$ are modeled as a function of the quantities at time t as follows:

$$u_{t+\Delta t} = u_t + \Delta t \dot{u}_t + \frac{\left(\Delta t\right)^2}{2}\left[\left(1-2\beta\right)\ddot{u}_t + 2\beta\ddot{u}_{t+\Delta t}\right] \tag{2.1}$$

$$\dot{u}_{t+\Delta t} = \dot{u}_t + \Delta t\left[\left(1-\gamma\right)\ddot{u}_t + \gamma\ddot{u}_{t+\Delta t}\right] \tag{2.2}$$

$$\ddot{u}_{t+\Delta t} = \frac{1}{\beta\left(\Delta t\right)^2}\left(u_{t+\Delta t} - u_t\right) - \frac{1}{\beta\Delta t}\dot{u}_t - \frac{1-2\beta}{2\beta}\ddot{u}_t \tag{2.3}$$

By substituting these into Equation (1.1), we obtain a simple linear system:

$$K^{eff} u_{t+\Delta t} = f^{eff}{}_{t+\Delta t} \tag{2.4}$$

where K^{eff} and f^{eff} are effective stiffness matrix and effective force vector, respectively, where:

$$K^{eff} = K + \frac{\gamma}{\beta\Delta t}C + \frac{1}{\beta\left(\Delta t\right)^2}M \tag{2.5}$$

$$\begin{aligned} f^{eff}{}_{t+\Delta t} &= f^{ext}{}_{t+\Delta t} + f^C + f^M \\ f^C &= C\left(\frac{\gamma}{\beta\Delta t}u_t + \frac{\gamma-\beta}{\beta}\dot{u}_t + \frac{\Delta t\left(\gamma-2\beta\right)}{2\beta}\ddot{u}_t\right) \\ f^M &= M\left(\frac{1}{\beta\left(\Delta t\right)^2}u_t + \frac{1}{\beta\Delta t}\dot{u}_t + \frac{1-2\beta}{2\beta}\ddot{u}_t\right) \end{aligned} \tag{2.6}$$

As is typical, the effective stiffness matrix is assumed to be symmetric positive-definite. In linear elasticity, the stiffness and mass matrices remain constant throughout the analysis, and the effective stiffness matrix needs to be computed only once. However, the effective force vector must be updated at each time step since it depends on the displacement, velocity, and acceleration fields. In large-deformation models and in elasto-plasticity, the effective stiffness matrix can change over time.

2.2 Direct and Iterative Solvers: Tradeoff

Computationally, the most intensive task in the Newmark-beta method is solving the linear system in Equation (2.4). Direct solvers are robust, and rely on factoring the matrix, for example, into a Cholesky decomposition:

$$K^{eff} = LL^T \tag{2.7}$$

This is followed by a triangular solve:

$$u_{t+\Delta t} = L^{-T}(L^{-1})f^{eff}{}_{t+\Delta t} \tag{2.8}$$

In transient analysis, direct methods are particularly favorable since the factorization needs to be carried just once.

However, due to the explicit factorization, direct solvers are memory intensive. For example, to quote from the ANSYS manual [4], "[sparse direct solver] *is the most robust solver in ANSYS, but it is also compute- and I/O-intensive*". Specifically, for a matrix with *one million degrees of freedom* [4]:

- *Approximately 1 GB of memory is needed for assembly.*
- *An additional 10 to 20 GB memory is needed for factorization.*

Since memory-access is often the bottle-neck in modern computer architecture [5], this directly translates into increased clock time. In other words, *reducing memory usage is crucial for large-scale problems.*

Iterative solvers have low foot-print; they do not factorize the stiffness matrix, but compute the solution iteratively. When the stiffness matrix is symmetric and positive definite, the most common iterative solver is the conjugate gradient [6]. In iterative solvers:

1. The number of iterations must be minimized; this typically achieved through an efficient preconditioner and/or through multi-grid/deflation techniques. In this paper, we consider a particular deflation technique proposed in [7].
2. Equally important is an efficient implementation of sparse matrix-vector multiplication (SpMV). SpMV has drawn considerable attention from several researchers. For example, see [8] for an implementation of SpMV on graphics-programmable units (GPUs). In this paper, we consider *an assembly-free implementation of SpMV*.

In summary, one can conclude that, for large-scale implicit structural dynamics:

- Iterative solvers scale better than direct methods; this is illustrated through a numerical experiment in Section 4.
- Preconditioning and/or multi-grid/deflation is important in iterative techniques.
- Efficient SpMV and reducing memory foot-print will reduce the computational cost per iteration.
- Exploiting multi-core architecture shows promise, but hinges on building parallelization-friendly algorithms.

2.3 Assembly Free Finite Element Analysis

In classic FEA, the element matrices are typically assembled into global matrices K & M. In this paper, we will apply assembly-free FEA where neither K nor M are assembled/stored. Instead, the fundamental matrix operations such as the sparse matrix-vector multiplication (SpMV) are performed in an assembly-free elemental level, i.e., an SpMV operation of Ku is interpreted as follows:

$$Ku = \sum_e \left(K_e u_e\right) \tag{2.9}$$

In other words, instead of assembling the global matrix, and then carrying out SpMV, an element-vector multiplication is carried, and then the results are assembled. This idea was first proposed in 1983 [9], but has resurfaced due to the surge in fine-grain parallelization.

Current research on assembly-free FEA can be grouped into three categories: (1) developing effective preconditioners, (2) extending the concept to a wider-class of problems, and (3) its efficient implementation, for example, on multi-core graphics-programmable units (GPU).

In the first category, Augarde et al. [10] developed an element based displacement pre-conditioner for linear elasticity. To accelerate convergence, Arbenz et al. [5] introduced a scalable multi-level pre-conditioner for microstructural FEA. The effectiveness of this preconditioner was demonstrated on finite element models with millions of elements. Bellavia et al. (2013) [11] introduced a matrix-free pre-conditioner for symmetric positive definite systems that relies on partial Cholesky factorization with deflation techniques. This was used for solving sequences linear systems with different right-hand sides.

In the second category, Yadav et al. [12] used an assembly-free method to perform large-scale modal analysis, and also discussed its implementation on the GPU.

In the third category, Mueller et al. [13], presented a matrix-free GPU implementation of a preconditioned CG solver for anisotropic elliptic partial differential equations. In [14], the performance of the GPU implementation of assembly-free FEA using trilinear hexahedral elements, was compared against the corresponding serial version run on a conventional processor for various mesh sizes and sparse matrix storage schemes.

In this paper, we extend assembly free FEA to transient analysis, propose an efficient deflation (preconditioning) technique, and discuss its implementation on the GPU.

3. PROPOSED STRATEGY

In the present paper, a deflated implicit structural dynamics method is developed by implementing and merging four distinct but complementary concepts (see Figure 2). We shall discuss each of these concepts in the following sections, but briefly:

1. Voxelization with Adaptive Refinement: Voxelization is a special form of finite element discretization where all elements are identical (hexahedral elements); the most important benefit of voxelization is meshing-robustness in that voxelization rarely fails unlike classic meshing. In addition, in FEA, voxelization significantly reduces memory foot-print since the element stiffness matrices are all identical; this directly translates into increased speed of analysis. However, a well-known challenge with voxelization is reduced accuracy, especially in stress prediction. This is addressed here through adaptive refinement where the mesh is refined in specific locations as needed. A unique advantage of this in transient simulation is that since the stress locations can change over time, the mesh only need to be refined in specific locations as needed, without sacrificing on speed.
2. Assembly-Free: As stated earlier, we shall pursue assembly-free analysis due to its inherent fine-grain parallelism. The voxel mesh is particularly well-suited for fast assembly-free analysis since a single element matrix is sufficient. However, one of the usual challenges in assembly-free iterative analysis is preconditioning/ deflation technique; here we rely on assembly-free deflation discussed next.
3. Deflation: Deflation is a powerful acceleration technique for conjugate gradient [15], and is more amenable to an assembly-free implementation, than classic preconditioners such as incomplete Cholesky. The particular method of deflation exploited in this paper is based on rigid-body agglomeration.
4. Parallelization: Finally, given the above infrastructure, fine-grain parallelization is achieved in this paper on multi-core CPUs using OpenMP, and on many-core GPUs, using NVIDIA's CUDA language. Each of the above concepts is discussed in the following sections.

Note that the above concepts are independent of each other; for example, one can apply deflation without imposing assembly-free analysis. By combining all four complementary concepts, maximum computational benefits can be gained.

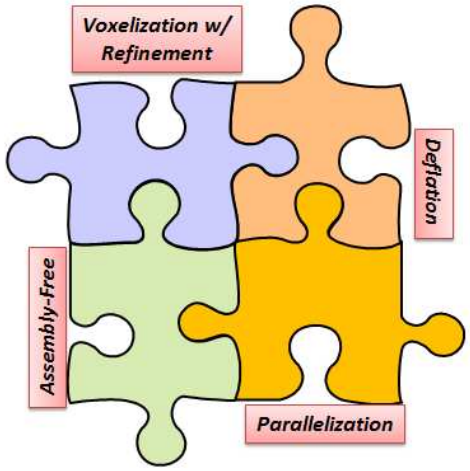


Figure 2: An overview of the proposed method.

3.1 Voxelization with Adaptive Refinement

As stated earlier, in this paper, we consider a simple finite element discretization, where the geometry is approximated via uniform hexahedral elements or 'voxels'; the voxel-approach has gained significant popularity recently due to its robustness and low memory foot-print [16]. The voxelization of the geometry in Figure 1 is illustrated in Figure 3; it has over 300,000 elements. Fortunately, even such a large-size problem is easily handled via the proposed method.

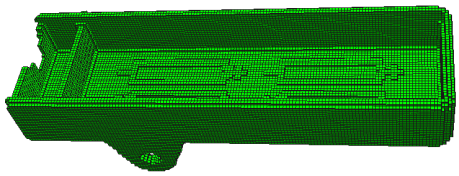


Figure 3: Brute-force voxelization of the structure in Figure 1.

The voxelization of a triangulated CAD model, also referred to as 3-D scan conversion, is straight-forward, and is discussed, for example, in [17]. The most significant benefits of voxelization are: (1) it is robust in that it rarely fails (unlike traditional meshing), (2) the mesh storage is compact, (3) the cost of voxelization is usually negligible and is relatively insensitive to geometric complexity, and (4) it promotes assembly-free-FEA.

Typically, the downside of voxelization is that the stresses tend to be less accurate. We mitigate this through two strategies described below.

Given a voxelization, one can choose a variety of hexahedral finite element shape functions. The simplest is the set of tri-linear shape functions $N_i(i=1,\ldots,8)$ described in [18], where each node-based shape function is of the form:

$$N_i = 0.125(1+\xi_i\xi)(1+\eta_i\eta)(1+\zeta_i\zeta); i=1\ldots8 \quad (3.1)$$

where ξ_i, η_i, and ζ_i $(i=1,\ldots,8)$ are nodal coordinates in the reference domain and ξ, η, and ζ are respective coordinates of Gaussian quadrature points used to numerically evaluate shape functions $N_i(i=1,\ldots,8)$.

However, the resulting 8-noded elements are 'stiff', and convergence is slow. One could use 20-node or 27-node

elements, but this increases the memory requirements significantly.

Instead we use the *Wilson* element endowed with three additional bubble-functions P of the form of [19], [20]:

$$\begin{aligned} P_1(\xi,\eta,\zeta) &= (1-\xi^2) \\ P_2(\xi,\eta,\zeta) &= (1-\eta^2) \\ P_3(\xi,\eta,\zeta) &= (1-\zeta^2) \end{aligned} \tag{3.2}$$

The resulting element stiffness matrix over domain Ω are of the form:

$$\bar{K}_e = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \tag{3.3}$$

where

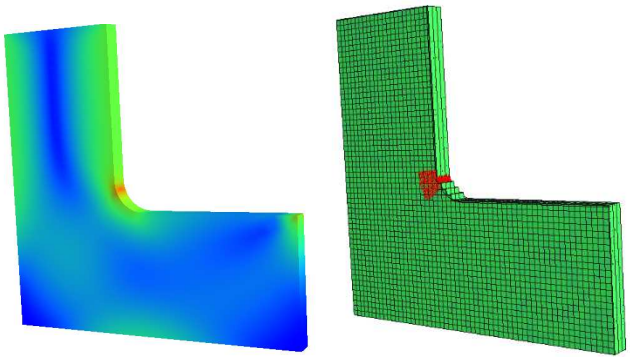
$$\begin{aligned} K_{11} &= \int \nabla N^T D \nabla N d\Omega \\ K_{12} &= \int \nabla N^T D \nabla P d\Omega \\ K_{21} &= \int \nabla P^T D \nabla N d\Omega \\ K_{22} &= \int \nabla P^T D \nabla P d\Omega \end{aligned} \tag{3.4}$$

One can condense out the bubble degrees of freedom, resulting again in a reduced 24 degrees of freedom element stiffness matrix [20]:

$$K_e = K_{11} - K_{12}(K_{22} \setminus K_{21}) \tag{3.5}$$

This significantly improves the stress predictions without penalizing the computation since Equation (3.5) needs to be carried out once (for a single element). Similar condensation can be carried out for the mass element matrix.

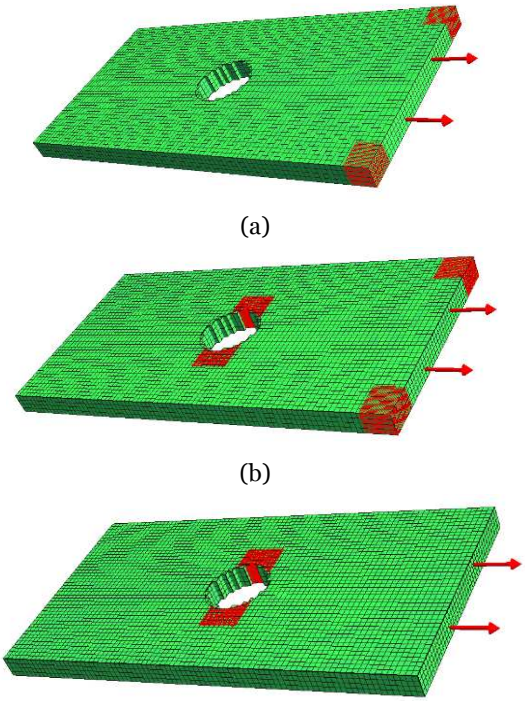
A second strategy that we adopt here is adaptive refinement through sub-modeling. Sub-modeling [2] (or local mesh refinement) is a classic idea in FEA where after a global problem is solved, one creates a higher-resolution mesh around regions of stress concentrations. The solution from the global mesh is enforced as "Dirichlet boundary condition" on the periphery of the local mesh, and a local analysis is carried out. As is well known, this simple strategy avoids the high cost of fine resolution at a global level, but delivers high accuracy. The same strategy is adopted here in that the mesh is refined near regions of stress concentrations (see Figure 4).



(a) (b)

Figure 4: (a) Stress concentration, and (b) local refinement.

In particular, during transient analysis, the region of stress concentration can change over time (see Figure 5). To this end, at each time step, we find the critical locations, create the local fine mesh, and solve the local problems. Thus, the mesh only need to be refined in specific locations as needed, without sacrificing on speed.



(a) (b) (c)

Figure 5: The regions of stress concentration can change during transient analysis. (a) t = 1e-7 s (b) t = 1e-6 (c) t = 1e-5

3.2 Assembly-Free FEA

The advantages of a matrix-free analysis are: (1) memory requirements are obviously reduced, and therefore fine resolution transient analysis can be carried out, (2) memory reduction indirectly translates into increased computational speed [21], and (3) matrix-free multiplication is well suited for parallelization on multi-core architectures [12]. Melanz et al. (2013) [22] developed a matrix-free method to solve systems of very stiff systems of kinematically constrained problems using implicit integration with absolute nodal coordinates.

Since our final goal here is to solve Equation (2.4), observe in Equation (1.3) that K^{eff} is a linear combination of K and M, therefore the element effective stiffness matrix K_e^{eff} can be computed as follows:

$$K_e^{eff} = \left(1 + \frac{\gamma\beta^d}{\beta h}\right) K_e + \left(\frac{1}{\beta \Delta t^2} + \frac{\gamma\alpha^d}{\beta \Delta t}\right) M_e \tag{3.6}$$

Considering Equation (2.6) and Equation (1.3), one can see that f^{eff} can be expressed in terms of Ku, $K\dot{u}$, $K\ddot{u}$, Mu, $M\dot{u}$, and $M\ddot{u}$ as follows:

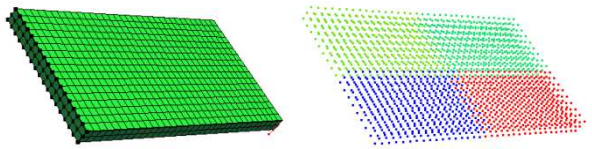
$$
\begin{aligned}
f^{eff}_{t+\Delta t} &= f^{external}_{t+\Delta t} + \frac{\gamma\beta^d}{\beta\Delta t} K u_t + \\
&\left(\frac{\gamma\beta^d}{\beta} - \beta^d\right) K \dot{u}_t + \left(\frac{\gamma\beta^d \Delta t}{2\beta} - \beta^d \Delta t\right) K \ddot{u}_t + \\
&\left(\frac{1}{\beta\Delta t^2} + \frac{\gamma\alpha^d}{\beta\Delta t}\right) M u_t + \left(\frac{1}{\beta\Delta t} + \frac{\gamma\alpha^d}{\beta} - \alpha^d\right) M \dot{u}_t + \\
&\left(\frac{\gamma\alpha^d \Delta t + 1}{2\beta} - \alpha^d \Delta t - 1\right) M \ddot{u}_t
\end{aligned}
\tag{3.7}
$$

Thus to compute the effective force at each time-step, one must carry out several sparse matrix-vector multiplications; these can be carried out in an assembly-free manner.

3.3 Deflated Conjugate Gradient

Deflation is a popular method for accelerating iterative methods such as conjugate gradient. The concept behind deflation [15] is to construct a matrix W, referred to as the *deflation space*, whose columns 'approximately' span the low eigen-vectors of the (effective) stiffness matrix.

Since computing the eigen-vectors is obviously expensive, Adams and others [7], [23] suggested a simple *agglomeration* technique where finite element nodes are collected into small number of groups. For example, Figure 6 illustrates agglomeration of the finite element nodes into four groups.



(a) (b)

Figure 6: (a) Finite element mesh, (b) agglomeration of mesh nodes into four groups.

As a step towards constructing the W matrix, nodes within each group are collectively treated as a rigid body. The motivation is that these agglomerated rigid body modes mimic the low-order eigen-modes. Then displacement of each node within a group is expressed as:

$$
\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{pmatrix} \lambda_g \tag{3.8}
$$

where

$$
\lambda_g = \left\{u_0, v_0, w_0, \theta_x, \theta_y, \theta_z\right\}^T \tag{3.9}
$$

are the six unknown rigid body motions associated with the group, and (x, y, z) are the relative coordinates of the node with respect to the geometric center of the group. Observe that Equation (3.8) is essentially a restriction operation similar to that of multi-grid [24]. Once the mapping in Equation (3.8) is constructed for all the nodes, they can be 'assembled' to result in a deflation matrix W :

$$
d = W\lambda \tag{3.10}
$$

where d is the 3N degrees of freedom (N: number of Nodes), λ is the 6G degrees of freedom associated with the groups (G: number of Groups). One can now exploit the W matrix to create the *deflated conjugate gradient* (DCG) algorithm described below:

Algorithm: Deflated CG (DCG); solve $Kd = f$

1. Construct the deflation space W
2. Choose d_0 where $W^T r_0 = 0$ & $r_0 = f - Kd_0$
3. Solve $W^T K W \mu_0 = W^T K r_0$; $p_0 = r_0 - W\mu_0$
4. $j = 1$
5. While $\left\|r_{j-1}\right\| > \varepsilon$, do:
6. $\alpha_{j-1} = \dfrac{r_{j-1}^T r_{j-1}}{p_{j-1}^T K p_{j-1}}$
7. $d_j = d_{j-1} + \alpha_{j-1} p_{j-1}$
8. $r_j = r_{j-1} - \alpha_{j-1} K p_{j-1}$
9. $\beta_{j-1} = \dfrac{r_j^T r_j}{r_{j-1}^T r_{j-1}}$
10. Solve $W^T K W \mu_j = W^T K r_j$ for μ
11. $p_j = \beta_{j-1} p_{j-1} + r_j - W\mu_j$
12. $j = j + 1$
13. End-Do While

When N >> G, i.e., when the number of mesh nodes far exceeds the number of groups:

- The primary computation is the sparse matrix-vector multiplication (SpMV) Kx in steps 5 and 9.
- Additional computations include the restriction operation $W^T x$ in step 9, the prolongation $W\mu$ in step 10, and the solution of the linear system $(W^T K W)\mu = y$ in step 9.

The one-time coarse matrix $W^T K W$ construction in step 3 can be viewed as a series of SpMV, followed by a series of restriction operations. Observe that the deflation matrix W is also sparse.

3.4 Newmark Algorithm

The algorithm is quite similar to classic Newmark-beta method for transient elasticity problems, and proceeds as follows.

Algorithm: Assembly-Free Newmark

For solving $M\ddot{u} + C\dot{u} + Ku = f$

1. Model the geometry, voxelize, and set initial conditions
2. Set material properties
3. Compute K_e and M_e
4. Set Newmark coefficients $\left(\beta \;\&\; \gamma\right)$

   Set damping coefficients $\left(\alpha^d \;\&\; \beta^d\right)$

   Set duration and step size $\left(T \;\&\; \Delta t\right)$
5. $K_e^{eff} = \left(1 + \dfrac{\gamma\beta^d}{\beta h}\right) K_e + \left(\dfrac{1}{\beta\Delta t^2} + \dfrac{\gamma\alpha^d}{\beta\Delta t}\right) M_e$
6. While $t_n \le T$, do:
   a. Update $f^{eff}{}_{t+\Delta t} = f^{ext}{}_{t+\Delta t} + f^C + f^M$
   b. Solve $K^{eff} u_{t+\Delta t} = f^{eff}{}_{t+\Delta t}$ using DCG (section 3.3)

c. $\ddot{u}_{t+\Delta t} = \frac{1}{\beta(\Delta t)^2}\left(u_{t+\Delta t} - u_t\right) - \frac{1}{\beta \Delta t}\dot{u}_t - \frac{1-2\beta}{2\beta}\ddot{u}_t$

d. $\dot{u}_{t+\Delta t} = \dot{u}_t + \Delta t\left[(1-\gamma)\ddot{u}_t + \gamma \ddot{u}_{t+\Delta t}\right]$

e. Compute strains and stresses; optionally, use a refined mesh for improved stress estimates

7. End-While

3.5 SpMV: CPU and GPU Parallelization

In all our numerical studies, almost 80% of the computation time is spent in executing the assembly-free sparse matrix-vector multiplication (SpMV). We therefore focused our efforts on parallelization effort of SpMV. The parallelization algorithm is describe below

Parallelization of y = K*x (SpMV)

1. Compute a single element stiffness matrix K_e
2. Assign a thread to each node
3. For N = 1,2, …, #Nodes:
   a. Set y(3*N:3N+2) = 0
   b. For each element 'e' connected to node, Do
      i. Fetch the element vector x_e
      ii. Perform $y_e = K_e * x_e$
      iii. Accumulate y_e into y(3*N:3N+2)
   c. End-Do
4. End-Do nodes

On the CPU, assignment of threads was achieved through OpenMP commands (www.openmp.org). On the GPU, SpMV was parallelized using NVidia CUDA [25]. Other modules such as 'vector dot-product', etc., were also accelerated through OpenMP and CUDA commands.

For larger problems, SpMV parallelization could perhaps be extended through message passing interface (MPI), but this was not explored in this work.

4. NUMERICAL EXPERIMENTS

In this Section, we present results from numerical experiments based on the proposed algorithm. All experiments were conducted on a Windows 7 64-bit machine with the following hardware: Intel Core i7 CPU running at 3.4GHz with 8 GB of memory, and a graphics card of GeForce GTX-760; parallelization on CPU and GPU were implemented through OpenMP and CUDA, respectively.

For all experiments, the Newmark coefficients were:

$$\gamma = 0.5 \qquad \beta = 0.25 \tag{4.1}$$

The material properties are those of steel:

$$E = 2.1e11(N/m^2) \qquad \nu = 0.28 \qquad \rho = 7700(kg/m^3) \tag{4.2}$$

4.1 Impact of Assembly-free Analysis on Speed

In the first experiment, we compare the proposed assembly-free deflated conjugate gradient (AF-DCG) against the popular commercial finite element software, ANSYS. The geometry is a steel cantilever beam of dimension $0.5 \times 0.02 \times 0.05$ (meters). A tip-force of one Newton is applied at $t = 0$ (and maintained thereafter) as illustrated in Figure 7. Two different mesh-sizes were used as described below. In ANSYS, the 'Brick 8 node 185' element was used, while the AF-DCG relies on the Wilson element described earlier.

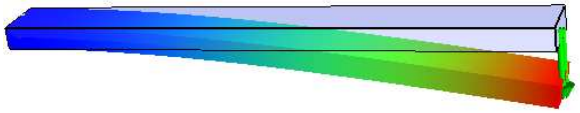


Figure 7: Deflected cantilever beam.

The damping coefficients are $\alpha^d = 0(s^{-1}); \beta^d = 5E-4(s)$; the analysis time is 0.2 seconds, with $\Delta t = 0.0005(s)$.

In both implementations, with 8000 elements, the maximum deflection was reached at around 0.01 seconds, where:

$$D_{\max} = 5.88E-6 \text{ (m)} \quad \text{(ANSYS)} \qquad D_{\max} = 5.92E-6 \text{ (m)} \quad \text{(AF-DCG)} \tag{4.3}$$

The slight difference can be attributed to the difference in the two shape functions used. It was confirmed through mesh refinement that the Wilson element used in this paper is more accurate.

Figure 8 illustrates the transient response of the normalized tip displacement in ANSYS $\left(D_{\max} = 5.88E-6 \text{ (m)}\right)$.

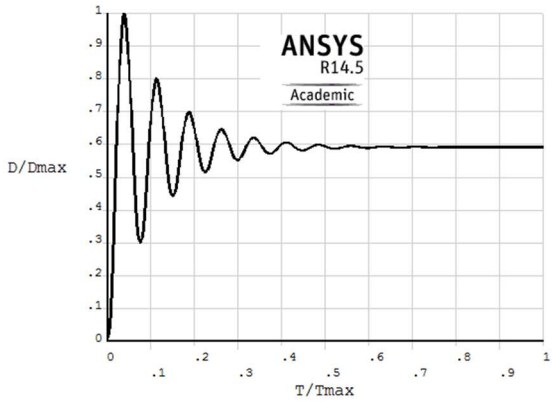


Figure 8: Relative tip displacement (ANSYS)

Figure 9 illustrates the corresponding displacement in AF-DCG. Both methods converged to the static deflection as expected $\left(D_{\max} = 5.92E-6 \text{ (m)}\right)$.

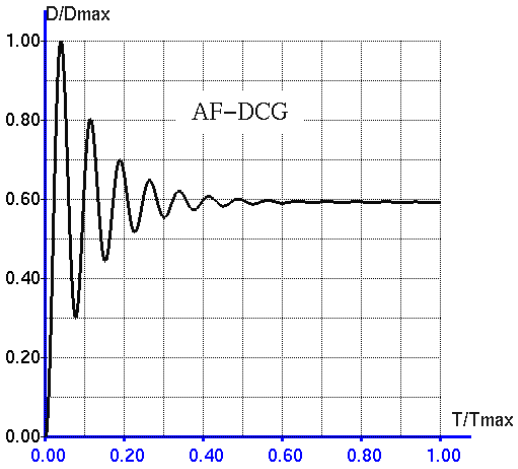


Figure 9: Relative tip displacement (AF-DCG)

To compare the computational costs, the geometry was discretized using two different mesh sizes: 8000 elements and 25000 elements. With each mesh size, four different solvers were considered (1) ANSYS-direct, (2) ANSYS-pre-conditioned conjugate gradient (PCG), (3) proposed AF-DCG on the CPU, and (4) proposed AF-DCG on the GPU (where SpMV is implemented on the GPU).

Figure 10 compares the computational times. As one can observe, AF-DCG implementations are about seven times faster than ANSYS for the smaller mesh size, and about fifteen times faster for the larger mesh size.

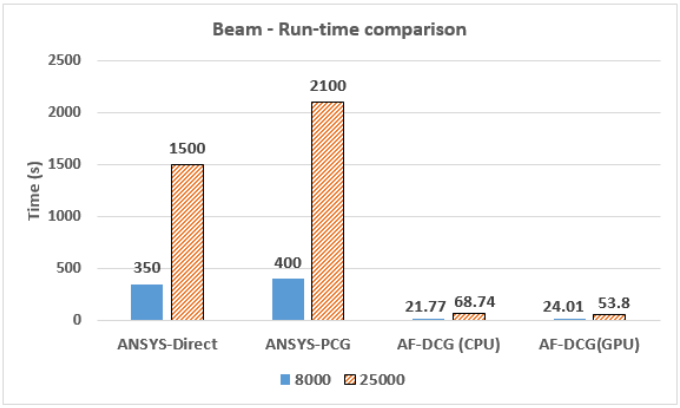


Figure 10: Run-time comparison of ANSYS vs. AF-DCG.

4.2 Loss of Accuracy due to Voxelization

Next, we compare the accuracy of the proposed method in capturing stresses. The geometry is illustrated in Figure 11; material properties are those of steel. A force of 5000 (N) was applied at time $t = 0$, and maintained thereafter. The total analysis time is 0.008 seconds with $\Delta t = 0.00002(s)$; the damping coefficients are $\alpha^d = 0(s^{-1}); \beta^d = 2E-5(s)$.

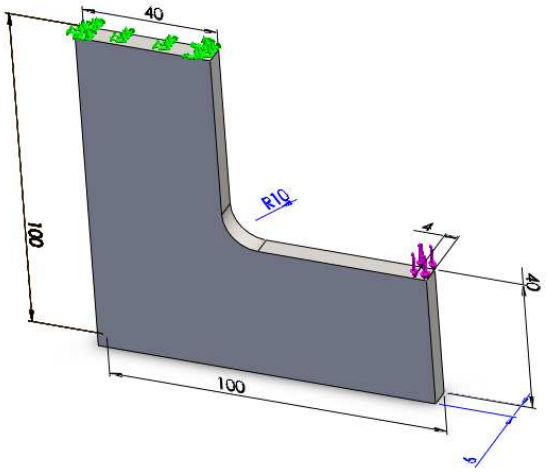


Figure 11: L-Bracket geometry (dimensions in mm) and loading.

The geometry was discretized using 4,000 elements. Despite the fact that we rely on a non-conforming voxel-mesh (as opposed to a high-quality conforming mesh in ANSYS), the stress predictions at steady state are as follows: ANSYS predicts a stress of 445 MPa, SolidWorks predicts a stress of 448 MPa, while the proposed method estimates a stress of 425 MPa, i.e., an under prediction.

Figure 12 illustrates the normalized maximum von Mises stress as a function of time for ANSYS, while Figure 13 illustrates the same for AF-DCG.

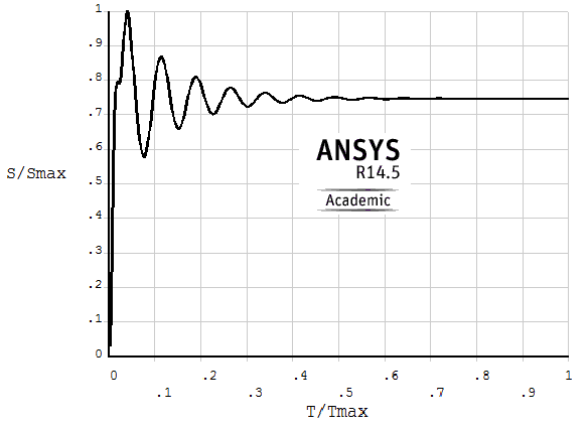


Figure 12: Normalized stress for the L-bracket (ANSYS).

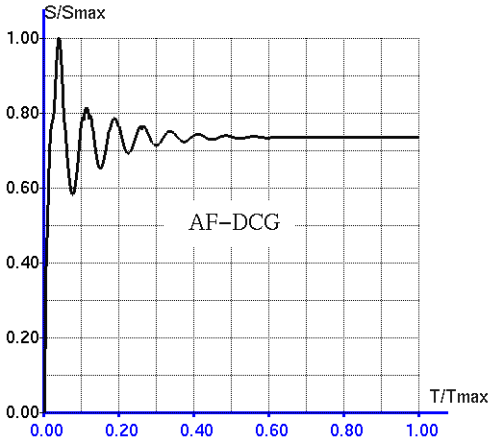


Figure 13: Normalized stress for the L-bracket (AF-DCG).

4.3 Advantages of Iterative Solvers over Direct

While it is difficult to quantify the advantages of one class of solvers over another (since efficiency of solvers depend strongly on the problem, implementation, hardware, etc.), we present an example illustrating the computational cost (time taken) and memory consumed by direct and iterative solvers.

Specifically, we solved a *static finite element problem* over the rocker illustrated in Figure 14a using SolidWorks 2014 [26], for various (quadratic tetrahedral) mesh densities. This problem was chosen since it contains thin sections that typically pose challenges to iterative solvers; the deformation of the rocker is illustrated in Figure 14b.

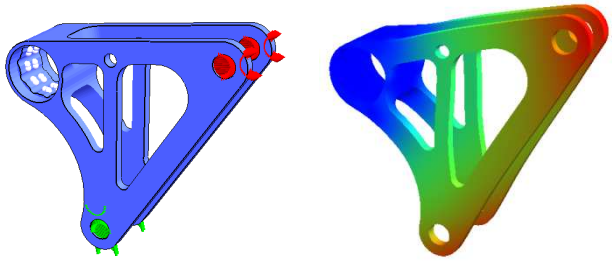


Figure 14: Rocker problem: (a) Loading, and (b) displacement

Figure 15 illustrates the time taken in seconds by the two solvers in SolidWorks. As one can observe, the direct solver takes about 2 minutes to solve a million DOF problem, while the iterative solver requires less than 20 seconds.

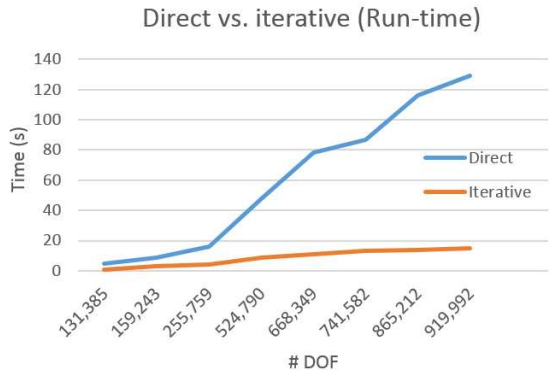


Figure 15: Time taken (secs) by SolidWorks 2014 [26] direct and iterative solvers, as a function of the degrees of freedom (DOF).

Figure 16 illustrates the maximum memory required by the two solvers; the direct solver requires about 5 GBytes to solve a million DOF problem, while the iterative solver requires about 0.6 GBytes. Increased memory consumption directly leads to increased 'computational' time.

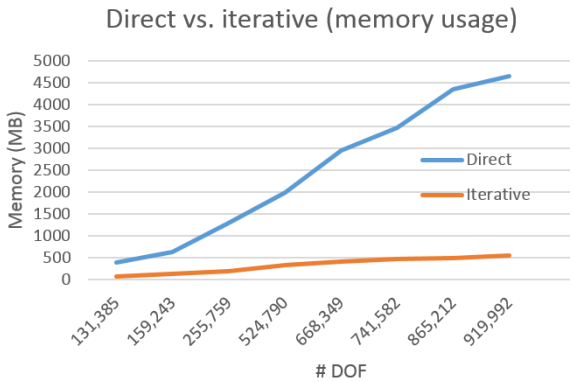


Figure 16: Memory consumed by SolidWorks 2014 [26] direct and iterative solvers, as a function of degrees of freedom.

4.4 Importance of Deflation

Having established the superiority of iterative solvers, in this paper, we use a specific iterative solver, namely deflated conjugated gradient. The purpose of this experiment is to highlight the importance of deflation, especially for thin structures whose stiffness matrices are typically ill-conditioned.

The geometry and loading were illustrated earlier in Figure 14. The geometry was discretized into 40000 hexahedral elements, and the transient analysis time was set to 0.0125 seconds while $\Delta t = 0.0001(s)$.

The damping coefficients being $\alpha^d = 0(s^{-1}); \beta^d = 2E-5(s)$

Figure 17 illustrates the number of conjugate gradient iterations with and without deflation. DCG-16 implies that deflation with 16 groups was used, while DCG-64 implies that deflation with 64 groups was used.

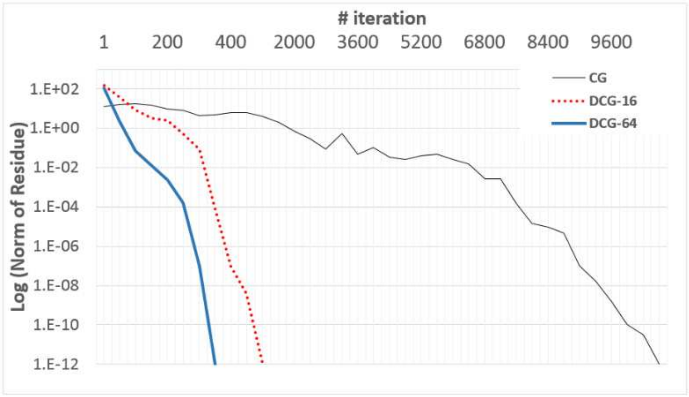


Figure 17: Faster convergence due to deflation.

4.5 Robustness of Voxelization

This experiment illustrates the robustness of voxelization. Consider the battery-holder in Figure 18; the small features present in the geometry can result in meshing-failure for a conforming mesh algorithm. A voxel-mesh is insensitive to such details since it only approximates the geometry up to the resolution of the mesh. The geometry was discretized using 80,000 elements as illustrated earlier in Figure 3. A total step-force of 1 N was applied on all the battery locations.

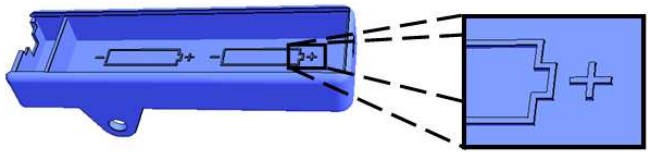


Figure 18: Battery holder geometry with small features.

The analysis time 0.04 seconds with a time step of 0.0001 ; the damping coefficients are $\alpha^d = 0(s^{-1}); \beta^d = 0.0001(s)$

Figure 19 illustrates the battery holder's relative stress through the analysis. Figure 20 illustrates the run-times in CPU and GPU, i.e., a speed-up of approximately 2.5 was achieved through the implementation of SpMV on the GPU.

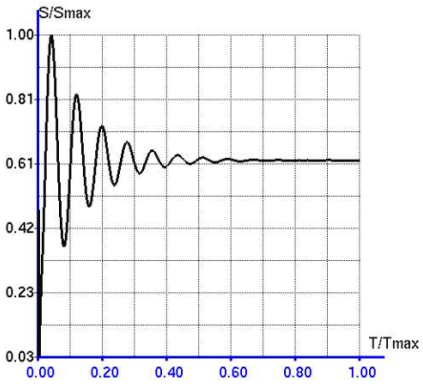


Figure 19: Battery holder's relative maximum stress.

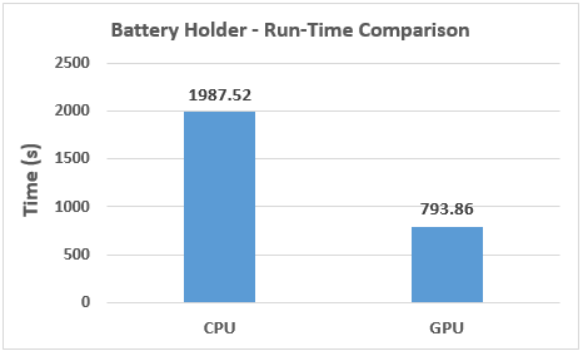


Figure 20: Battery holder: run-time comparison.

4.6 Industrial Application of Proposed Method

Electronic circuit boards undergo severe fatigue during shipping, and are usually geometrically complex. This makes them an ideal candidate for the present work. Here, we study the Arduino MEGA 2560 (Figure 21), a microcontroller widely used for R/C applications. The board was clamped at the four mounting holes, and a sinusoidal force was applied on one of the ICs.

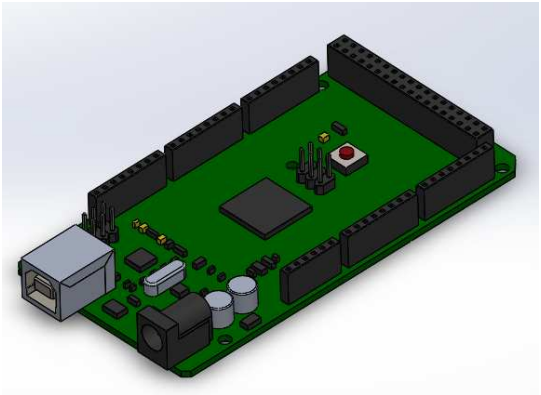


Figure 21: Arduino MEGO 2560.

Since the model is highly detailed, the mesh required over 300,000 voxels. The voxel mesh is illustrated in Figure 22.

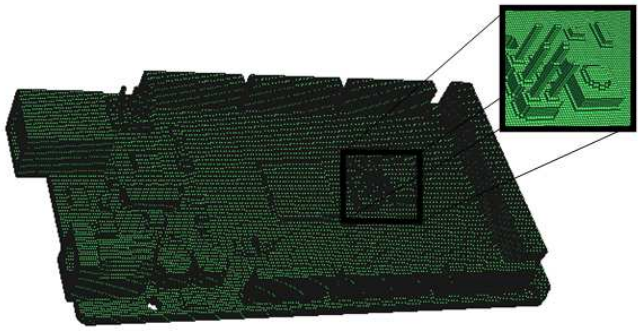


Figure 22: Arduino MEGA 2560: Voxel mesh.

A typical displacement field is illustrated in Figure 23.

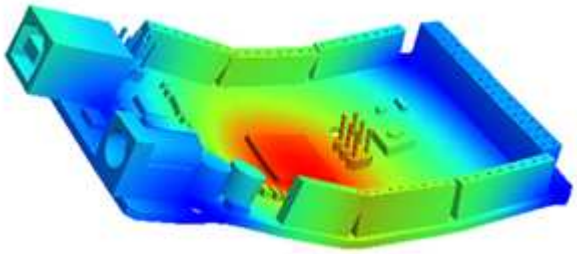


Figure 23: Arduino MEGA 2560: Displacement field.

The experiment was run both on CPU and GPU. The run-time on GPU is considerably smaller, which demonstrates the importance of fine-grain parallelization in large-scale problems.

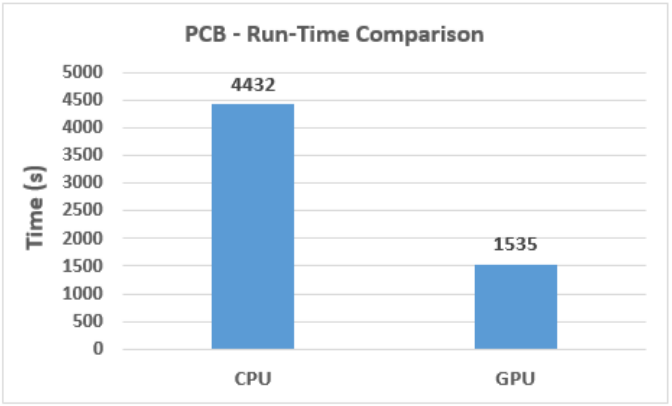


Figure 24: PCB test. Run-time comparison of CPU and GPU.

Figure 25 illustrates the normalized maximum Stress (at the fixed holes) through time.

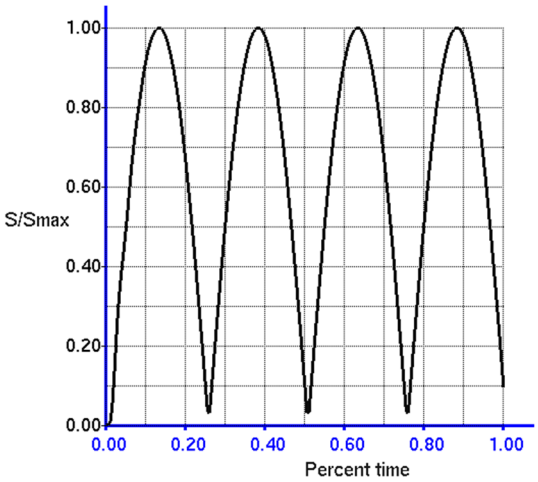


Figure 25: PCB test: normalized maximum stress

5. CONCLUSIONS

The main contribution of the paper is an efficient method for transient analysis of elastic systems based on the Newmark-beta algorithm. As illustrated, the proposed method is sufficiently accurate for initial stages of design, and significantly faster than the commercial software ANSYS, at least for simple geometries of Figure 7 and Figure 11. Further comparisons for complicated geometries and other existing commercial methods is required to affirm the robustness and speed of this method. The software will be made available through the author's research website www.ersl.wisc.edu. This paper serves as a foundation for future work on: (1) 'drop tests', (2) fatigue modeling, and (3) crack initiation studies.

Acknowledgements

The authors would like to thank the support of National Science Foundation through grants CMMI-1232508 and CMMI-1161474.

References

[1] R. D. Cook, D. S. Malkus, M. E. Plesha, and R. Witt, *Concepts and Applications of Finite Element Analysis*, 4th ed. John Wiley & Sons, 2002.

[2] R. D. Cook, *Concepts and applications of finite element*. New York, NY: John Wiley & Sons, 2007.

[3] N. M. Newmark, "A Method of Computation for Structural Dynamics," *Journal of the Engineering Mechanics Division*, vol. 85, no. EM3, pp. 67–94, 1959.

[4] *ANSYS 13*. ANSYS; www.ansys.com, 2012.

[5] P. Arbenz, G. H. van Lenthe, and et. al., "A Scalable Multi-level Preconditioner for Matrix-Free μ-Finite Element Analysis of Human Bone Structures," *International Journal for Numerical Methods in Engineering*, vol. 73, no. 7, pp. 927–947, 2008.

[6] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[7] M. Adams, "Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics," *International Journal for Numerical Methods in Engineering*, vol. 55, no. 2, pp. 519–534, 2002.

[8] N. Bell, "Efficient Sparse Matrix-Vector Multiplication on CUDA," NVidia Corporation, NVidia Technical Report NVR-2008-004, 2008.

[9] T. J. R. Hughes, I. Levit, and J. Winget, "An element-by-element solution algorithm for problems of structural and solid mechanics," *Comput Meth Appl Mech Eng*, vol. 36, no. 2, pp. 241–254, 1983.

[10] C. E. Augarde, A. Ramage, and J. Staudacher, "An element-based displacement preconditioner for linear elasticity problems," *Computers and Structures*, vol. 84, no. 31–32, pp. 2306–2315, 2006.

[11] S. Bellavia, J. Gondzio, and B. Morini, "A Matrix-Free Preconditioner for Sparse Symmetric Positive Definite Systems and Least-Squares Problems," *SIAM J. Sci. Comput.*, vol. 35, no. 1, pp. A192–A211, Jan. 2013.

[12] K. Suresh and P. Yadav, "Large-Scale Modal Analysis on Multi-Core Architectures," in *Proceedings of the ASME 2012 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Chicago, IL, 2012.

[13] E. Mueller, X. Guo, R. Scheichl, and S. Shi, "Matrix-free GPU implementation of a preconditioned conjugate gradient solver for anisotropic elliptic PDEs," *arXiv:1302.7193 [cs, math]*, Feb. 2013.

[14] A. Akbariyeh and et. al., "Application of GPU-Based Computing to Large Scale Finite Element Analysis of Three-Dimensional Structures," in *Proceedings of the Eighth International Conference on Engineering Computational Technology*, Stirlingshire, United Kingdom, 2012, p. Paper 6.

[15] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h, "A Deflated Version of the Conjugate Gradient Algorithm," *SIAM Journal on Scientific Computing*, vol. 21, no. 5, pp. 1909–1926, 2000.

[16] A. Duster, J. Parvizian, Z. Yang, and E. Rank, "The Finite Cell Method for 3D problems of solid mechanics.," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, pp. 3768–3782, 2008.

[17] E. A. Karabassi, G. Papaioannou, and T. Theoharis, "A Fast Depth-Buffer-Based Voxelization Algorithm," *Journal of Graphics Tools*, vol. 4, no. 4, pp. 5–10, 1999.

[18] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method for Solid and Structural Mechanics*. Elsevier, 2005.

[19] E. L. Wilson, R. L. Taylor, W. P. Doherty, and J. Ghaboussi, "Incompatible displacement models," *Numerical and computer methods in structural mechanics.*, vol. A 74–17756 06–32, pp. 43–57, 1973.

[20] H. H. Taiebat and J. P. Carter, "Three-Dimensional Non-Conforming Elements," Centre for Geotechnical Research, The University of Sydney, Sydney, R808, 2001.

[21] D. Goddeke, R. Strzodka, and S. Turek, "Performance and accuracy of hardware-oriented native-emulated- and mixed-precision solvers in FEM simulations," *International Journal of Parallel , Emergent and Distributed Systems*, vol. 22, no. 4, pp. 221–256, 2007.

[22] D. Melanz, N. Khude, P. Jayakumar, and D. Negrut, "A Matrix-Free Newton–Krylov Parallel Implicit Implementation of the Absolute Nodal Coordinate Formulation," *Journal of Computational and Nonlinear Dynamics*, vol. 9, no. 1, p. 011006, 2014.

[23] R. Aubry, F. Mut, S. Dey, and R. Lohner, "Deflated preconditioned conjugate gradient solvers for linear elasticity," *International Journal for Numerical Methods in Engineering*, vol. 88, pp. 1112–1127, 2011.

[24] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*. SIAM, 2000.

[25] NVIDIA Corporation, *NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide*. Santa Clara., 2008.

[26] SolidWorks, *SolidWorks; www.solidworks.com*. 2005.