A Fast Matrix-Free Elasto-Plastic Solver for Predicting Residual Stresses in Additive Manufacturing

Abstract

Process planning for additive manufacturing (AM) today relies heavily on multi-physics, multi-scale simulation. The focus of this paper is on one aspect of AM simulation, namely, part-level elasto-plastic simulation for residual stress and distortion predictions. This is one of the crucial steps in AM process optimization, but is computationally expensive, often requiring the use of large computer clusters.

The primary bottleneck in elasto-plastic simulation is the repeated solution of large linear systems of equations. While there is a wide range of linear solvers, most cannot exploit the unique structured nature of the mesh underlying AM. Here, we revisit a specific matrix-free solver, namely rigid-body deflated solver that has been very successful for solving large linear-elastic problems in such scenarios. The salient feature of this solver is that the stiffness matrix is never assembled, thereby reducing the memory requirements significantly, leading to large computational gains.

The objective of this paper is to extend the above solver to elasto-plasticity by efficiently updating the element tangent stiffness matrices, and the corresponding deflation matrix. The performance of the proposed method is evaluated on a benchmark problem using multi-core CPU and GPU architectures, and compared against ANSYS. Then, part-level residual stresses and distortion are predicted using the proposed solver. The present work is restricted to associative plasticity with von-Mises yield criteria, but can be extended to other plasticity models.

Keywords: Elasto-plasticity, Matrix-free deflation, Selective laser melting, Residual stresses.

1. Introduction

Additive manufacturing (AM) has gained significant importance in recent years due to its ability to produce intricate designs, with minimal tooling [1]. In particular, the selective laser melting (SLM) process relies on a laser source to melt metal powder, layer-by-layer. SLM can produce parts with high complexity and precision, and excellent mechanical properties. However, due to the inherent rapid heating and cooling, significant residual stresses can develop [2], resulting in build failure, recoater induced damage, distortion, warping, dimensional inaccuracy, etc [3] [4], [5].

The ability to predict these residual stresses through simulation is essential for process understanding and improvement. As is now well recognized, SLM simulation is inherently complex, entailing multiple physics (thermal, fluid and structural) [6] and multiple-scales (micro [7], meso [8] and part-level [9]) (see [10] [11] [12] for details). Typical SLM simulations include melt pool modeling [13], microstructure predictions [14],

part-level thermal analysis [15], part-level distortion analysis [16], etc. While each of these is critical for a comprehensive understanding of the SLM process, the focus of this paper is on part-level elasto-plastic analysis for predicting residual stresses and distortion. In particular, the paper addresses the underlying computational challenges.

The remainder of this paper is organized as follows. In Section 2, an overview of SLM simulation is presented, followed by a discussion on the key elasto-plastic computational challenge, namely, repeated solution of large systems of equations. Since prior work on solution of linear systems is vast, only critical observations relevant to this paper are made. In Section 3, a specific matrix-free method for solving large systems of equations is considered. Given its inherent advantages in AM simulation, this method is extended for solving elasto-plastic problems. This is followed by a description of the proposed algorithm. In Section 4, the performance of the proposed method is evaluated on a benchmark problem using multi-core CPU and GPU architectures, and compared against ANSYS. Then, in Section 5, the proposed method is employed for residual stress prediction using inherent-strain analysis. Conclusions and future work are summarized in Section 6.

2. Literature review

Part-level SLM simulation is of significant interest today. This is however non-trivial since the temperature changes are large and rapid, and therefore the simulation entails small time steps, and a fine resolution of the geometry. In addition, the behavior is strongly non-linear (temperature dependent conductivity, elasto-plasticity, etc). All of these challenges lead to a common computational challenge: repeated solution of large linear systems of equations with millions of degrees of freedom.

Despite significant advances in the solution of linear systems [17] [18], AM part-level simulation can take hours or even days [9]. To quote [6]: *For example, the run-time for these two models on a 16 core dual Xeon system with 14 core utilization, were six hours (without phase transition), and approximately 70 hours when phase transition was included.* In particular, elasto-plastic analysis is one of the major computational bottlenecks. As indicated in [9] where thermo-mechanical simulation of AM process was carried out with adaptive mesh coarsening, elasto-plastic analysis was twice as expensive as non-linear thermal analysis. Furthermore, due to the size of the underlying matrices, standard desktops are often insufficient, entailing the use of expensive computer clusters.

Various strategies have been proposed to alleviate some of these challenges. For example, adaptive mesh techniques are often used [9], [15], [19], with fine discretization near regions of interest. However, repeated re-meshing will lead to additional computational and robustness issues. Therefore, a uniform voxel-based mesh is typically used in SLM simulation (see Section 3) via a quiet or inactive element approach. In the quiet approach, all finite elements within the part are assembled into the global stiffness matrix, but the elements yet to be deposited are assigned void material properties. The quiet method is simple to implement, but is inefficient and can lead to ill-conditioning. The inactive method uses an evolving mesh, and includes

only the active elements for simulation. This can be beneficial especially during the initial stages. However, both methods involve explicit assembly of the underlying (tangent) stiffness matrix.

A fundamental premise of this paper is that *the computational bottleneck in the solution of linear system is memory access* [20]. In other words, fetching data from memory is excruciatingly slow compared to computing on it. This is a well known observation; for example, on a typical Intel I7 desktop, it takes 1 nano-second to multiply two doubles on the CPU, but it takes 70 nano-seconds to fetch these from memory [21].

Thus, data-compaction and limiting memory usage is key to computational speed-up. With this in mind, we propose here a voxel-based, matrix-free method for solving the elasto-plastic problem. The proposed method does not require the assembly of the underlying tangent matrix. Instead, the data is stored in a limited number of element matrices, and the linear system is solved efficiently using a novel incremental deflation method.

3. Proposed Methodology

Before discussing the proposed methodology, the finite element model and underlying assumptions are described next.

3.1. Finite Element Discretization

Most part-level SLM simulations today rely on a structured voxel mesh (as opposed to an unstructured tetrahedral mesh). All elements in a voxel mesh are geometrically identical; see Fig. 1; such meshes are simple to generate, and have low memory foot-print. Furthermore, in SLM simulation, voxelization is often the only practical approach for achieving fine resolution. Further, the voxel mesh ties-in conveniently with the layer-by-layer fabrication. A drawback of a voxel mesh is that it does not necessarily conform to the boundary; this is rarely of concern in such applications due to the fine resolution of the voxel mesh.

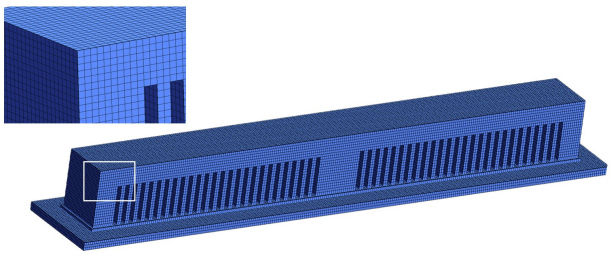

Figure 1: Voxelization of the twin-cantilever model.

3.2. Finite Element Model

Once the part is discretized, SLM part-level simulation is typically carried out in two distinct steps: (1) a transient thermal analysis (either layer-by-layer, or scan-by-scan), with or without phase-change modeling [9], followed by a (2) elasto-plastic analysis that uses the thermal load to compute residual stresses (again

layer-by-layer, or scan-by-scan). In this paper, we will assume that a thermal load has been computed, and will focus our attention on the second phase of elasto-plastic analysis.

For elasto-plastic analysis, one must employ a suitable mathematical model to capture plasticity. While there are various such models [22], we will use the small-deformation associative plasticity model with von-Mises yield criteria [23]; see **??**. The proposed methodology is not restricted to this model, i.e., other plasticity models can be used as well. Given a plasticity model, an elasto-plastic analysis (at each step during the deposition process) leads to the following nonlinear equilibrium equation [23]:

$$0 = f_{th} - F_{\text{int}}(u) \tag{1}$$

where f_{th} is the thermal load, and F_{int} is the internal force given by:

$$F_{\text{int}}(u) = \sum_{n_e} \int_{v_e} B^T \boldsymbol{\sigma}(u) dv \tag{2}$$

Here, B^T is strain-displacement matrix, σ is the Cauchy stress tensor and n_e is the number of elements.

Due to the nonlinear nature of this problem, the equilibrium state must be solved in an iterative manner. Specifically, one must repeatedly solve the (differential) linear system of equations:

$$K_t^k \left(u^{k-1}\right) \delta u^k = f_{th} - F_{\text{int}} \left(u^{k-1}\right) \tag{3}$$

where the tangent stiffness matrix K_t^k is given by:

$$K_t^k = \sum_{n_e} \int_{v_e} B^T D_t^k B dv \tag{4}$$

and D_t^k is the tangent constitutive matrix described in A.21. Then the displacement is updated via:

$$u^k = \delta u^k + u^{k-1} \tag{5}$$

The reader is referred to [23] for additional details.

To provide a computational context for this problem, note that:

- Assuming a simple layer-by-layer simulation, 10,000 or more simulations steps will be needed.
- At each simulation step, the linear system must be solved 4 to 8 times to find the equilibrium state.
- Finally, the typical size of the matrix K_t^k is in the order of 10^6.

These facts highlight the importance of solving Eqn. 3 efficiently. For simplicity, in the remainder of this paper, we will rewrite Eqn. 3 as:

$$K_t \delta u = \delta F \tag{6}$$

There are numerous methods for solving such equations, and these can be classified into two broad classes: direct and iterative. Direct solvers are robust, but are memory intensive, and therefore not suitable

for solving large systems of equations. Iterative methods, on the other hand, are designed for large-scale problems. In particular, conjugate gradient is one of the most popular iterative methods [24].

CG solvers compute increasingly accurate solutions δu_j to Eqn. 6 through residual minimization (see [25] for details). Residual minimization, in turn, entails sparse matrix-vector multiplication (SpMV) of the form $K_t \delta u_j$. Thus, to reduce the computational cost of solving Eqn. 6 via iterative methods one must:

- Reduce the cost of sparse matrix-vector multiplication (SpMV) $K_t \delta u$, for any given vector δu.
- Reduce the number of CG iterations needed for convergence.

The remainder of this paper will focus on these two tasks, within the context of elasto-plastic simulation.

3.3. Matrix-Free SpMV

Let us consider the first task of computing $K_t \delta u$. The classic approach is to *assemble the matrix, and then multiply with the vector*, i.e.,

$$K_t \delta u \equiv \left[\prod_{assemble} K_t^{(e)} \right] \delta u \tag{7}$$

Here $K_t^{(e)}$ are the element tangent stiffness matrices. Almost all commercial solvers rely on this well-established *matrix-ready* approach. Further, when the underlying mesh is unstructured, this is often the only viable approach.

However, when the mesh is structured, an alternate approach is the *matrix-free* approach [26] that advocates *multiply at an element level, then assemble*, i.e.,

$$K_t \delta u \equiv \prod_{assemble} \left(K_t^{(e)} \delta u^{(e)} \right) \tag{8}$$

While the two are mathematically equivalent, the matrix-free approach is particularly effective here due to the voxel mesh (see Fig. 1). In particular, if the material is purely *elastic*, only one instance of the element stiffness matrix $K_t^{(e)}$ needs to be computed and stored. This significantly reduces the memory requirements, and consequently the computational time.

However in *elasto-plastic simulation*, although all elements are geometrically identical (due to the voxel mesh), the element tangent matrices will differ due to the differing material state. Fortunately, we can still achieve a significant reduction in memory by observing that $K_t^{(e)}$, for each element, can be expressed as the sum of two components [23]:

$$K_t^{(e)} = K_{t(elastic)}^{(e)} + K_{t(\Delta)}^{(e)} \tag{9}$$

where

$$K_{t(elastic)}^{(e)} = \int_{v_e} \left[B^T \left(D_{elastic} \right) B \right] dv \tag{10}$$

and

$$K_{t(\Delta)}^{k(e)} = \int_{v_e} \left[B^T \left(D_{\mathrm{t}}^k - D_{elastic} \right) B \right] dv \tag{11}$$

In the above equations, $D_{elastic}$ is the standard elastic constitutive matrix, and D_t^k is the tangent constitutive matrix described in Appendix A.

Observe the following regarding the elastic component $K^{(e)}_{t(elastic)}$: (1) it does not change during the entire SLM simulation, and (2) it is identical for all elements. In other words, only one instance of $K^{(e)}_{t(elastic)}$ needs to be computed and stored at the start of the simulation. On the other hand, $K^{k(e)}_{t(\Delta)}$ must be computed during each Newton iteration k, and for each element that is currently exhibiting plastic behavior. Since only a small percentage of elements are typically on the plastic envelope (see numerical experiments), a significant reduction in memory usage (and therefore increase in computational speed) can be achieved, as we demonstrate later on.

Thus, we express the matrix-free version of SpMV for elasto-plasticity as:

$$K_t \delta u \equiv \prod_{assemble} \left(K^{(e)}_{t(elastic)} \delta u^{(e)} \right) + \prod_{assemble} \left(K^{k(e)}_{t(\Delta)} \delta u^{(e)} \right) \tag{12}$$

The first component must be computed for all elements, while the second component must be computed only for elements exhibiting plastic behavior. This simple splitting leads to fast SpMV as we shall later demonstrate.

3.4. Incremental Rigid Body Deflation

Next, we focus on reducing the number of CG iterations, needed for convergence. In traditional matrix-ready implementations of CG (where the matrix is assembled), one can rely on pre-conditioners such as incomplete-LU to accelerate CG. However, in the matrix-free approach, such pre-conditioners are not easy to compute since the global matrix is not readily available. In other words, not having access to the fully assembled matrix prevents the use of classic matrix-ready pre-conditioners.

Several simple matrix-free pre-conditioners have been proposed; these include Jacobi preconditioning (diagonal scaling), element-by-element preconditioning [27], element based symmetric Gauss-Seidel, element matrix factorization (EMF) [28], [29]. The analysis in [28], [29] shows that these methods are not efficient for large-scale problems.

Multigrid methods [30] and rigid-body-deflation, on the other hand, have shown to be been very effective for large scale problems [31], [32], [33]. The two methods are similar and the underlying concept is to aggregate topologically close-by finite-element nodes (or degrees of freedom) to eliminate the ill-effects of small eigenvalues. The aggregation process largely differentiates the two methods; see [34] for a comparative analysis. In this paper, we use the matrix-free rigid-body-deflation method due its superior performance in the context of elasticity problems [20], [34], with the goal of extending it to elasto-plastic problems.

The concept behind the rigid-body deflation method is to construct a deflation matrix W, whose columns approximately span the low eigenvectors of the global matrix. Obviously, since the computation of low eigenvectors is expensive, the idea is to utilize the rigid body modes of agglomerated groups of nodes as an approximate representation of low order eigenmodes [35] [36]. The definition of the W matrix depends

entirely on the finite element mesh. To illustrate, consider the finite element mesh in Fig. 2a. The first step is to aggregate the mesh nodes into a small number of groups; for example, the nodes in Fig. 2a are aggregated into 4 groups as illustrated in Fig. 2b.

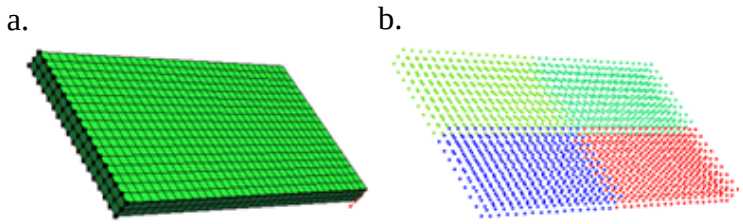

Figure 2: The aggregation concept [20].

Then, all nodes within a group are treated as a rigid-body, i.e., the degrees of freedom associated with every node within a group are expressed as follows:

$$\begin{Bmatrix} \delta u_x \\ \delta u_y \\ \delta u_z \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix} \mu_g \tag{13}$$

where $\mu_g = \begin{Bmatrix} u_0 & v_0 & w_0 & \theta_x & \theta_y & \theta_z \end{Bmatrix}^T$ are the six unknown rigid body parameters associated with that group. The global vector δu can then be expressed as:

$$\delta u = W\mu \tag{14}$$

In the above example, since there are four groups, the vector μ is of length 24, i.e., the the number of columns of W is 24. In practice, the number of groups can vary from 100 to 2000, i.e., the number of columns of W ranges from from 600 to 12000. We will later study the impact of this choice on convergence. Note that W is not constructed explicitly; see discussion below.

Given the projection $\delta u = W\mu$, the matrix problem in Eqn. 6 reduces to

$$\widetilde{K}_t \mu = \widetilde{f} \tag{15}$$

where

$$\widetilde{K}_t = W^T K_t W \tag{16}$$

is the deflated tangent matrix and

$$\widetilde{f} = W^T \delta F \tag{17}$$

Since $\widetilde{K}_t$ is small, the linear system can be solved efficiently using a direct solver. Further, since the matrix W is sparse and purely mesh dependent, the matrix $\widetilde{K}_t$ can be computed without explicitly constructing W or K_t. Later on, we describe how the matrix $\widetilde{K}_t$ is employed in the deflated-CG algorithm.

In contrast to elastic simulation, in elasto-plastic simulation, the deflated matrix $\widetilde{K}_t$ changes during every Newton iteration, and must be recomputed. This can get computationally expensive. To reduce this computation, we note that only the $K_{t(\Delta)}$ component of the tangent stiffness matrix changes during the iteration.

This leads to the concept of incremental deflation matrix $\widetilde{\delta K}$ which is defined as the difference in the deflated tangent matrix between two successive iteration:

$$\widetilde{\delta K} = W^T \left(K^k_{t(\Delta)} - K^{k-1}_{t(\Delta)} \right) W \tag{18}$$

As the numerical experiments later confirm, this reduces the computational cost significantly.

3.5. Algorithm

Given these concepts, the main algorithm for the iterative solution of Eqn. 3 is described below.

Algorithm 1 Newton's Semi-smooth iterative method

Input: The thermal load f_{th}

Output: The displacement u

1: **procedure** NEWTON ITERATION
2: Initialize u^0
3: Compute the elastic element tangent matrix $K^{(e)}_{t(elastic)}$
4: $k \leftarrow 1$
5: **while** $\left\|\delta u^k\right\| / \left(\left\|u^k\right\| + \left\|u^{k-1}\right\|\right) > \varepsilon_{Newton}$ **do**
6: Compute $K^{k(e)}_{t(\Delta)}$, δF^k ▷ Elastic predictor, plastic corrector algorithm [See Appendix A]
7: Solve $K^k_t \delta u^k = \delta F^k$ ▷ Algorithm [2]
8: $u^k = \delta u^k + u^{k-1}$
9: $k = k + 1$
10: **end while**
11: **end procedure**

The algorithm for solving step 6 in the above algorithm is described below.

Algorithm 2 Incremental Deflated CG

Input: $\delta F^k, W, K_t^k$

Output: δu^k

1: **procedure** INCREMENTAL RIGID BODY DEFLATION ▷ Solve Eq. (3)

2: **if** $k \leftarrow 1$ **then**

3: Construct $\widetilde{K_t}^k = W^T K_t^1 W$ and set $\widetilde{\delta K}^k = 0$

4: **else**

5: $\widetilde{\delta K}^k = W^T \left(K_{t(\Delta)}^k - K_{t(\Delta)}^{k-1} \right) W$; $\widetilde{K_t}^k = \widetilde{K}^{k-1} + \widetilde{\delta K}^k$

6: **end if**

7: Let $\delta u_0 = 0$ and $r_0 = \delta F^k - K_t^k \delta u_0$

8: Solve $\widetilde{K_t}^k \hat{\mu}_0 = W^T K_t^k r_0$ for $\hat{\mu}_0$ and set $p_0 = r_0 - W\hat{\mu}_0$. ▷ Standard Deflated CG algorithm

9: $j \leftarrow 1$

10: **while** $\|r_{j-1}\| > \varepsilon_{CG}$ **do**

11: $\alpha_{j-1} = r_{j-1}^T r_{j-1} / p_{j-1}^T K_t^k p_{j-1}$

12: $\delta u_j = \delta u_{j-1} + \alpha_{j-1} p_{j-1}$

13: $r_j = r_{j-1} - \alpha_{j-1} K_t^k p_{j-1}$

14: $\beta_{j-1} = r_j^T r_j / r_{j-1}^T r_{j-1}$

15: Solve $\widetilde{K_t}^k \hat{\mu}_j = W^T K_t^k r_j$ for $\hat{\mu}_j$

16: $p_j = \beta_{j-1} p_{j-1} + r_j - W\hat{\mu}_j$

17: $j = j + 1$

18: **end while**

19: **end procedure**

Note that the computationally expensive steps are 11 and 15; SpMV is required in both steps, and a direct solver [37] is used in step 15. Alternately, one can use recursive deflation in step 15.

4. Benchmark Experiments

As a demonstration of the proposed method, we first consider its performance using a simple benchmark problem proposed in [23]. The 2D benchmark problem considered in [23] is illustrated in Fig.3a, where all units are in meters. Here, we consider the equivalent 3D problem with 1 meter thickness and sliding boundary conditions on the out-of-plane surfaces. The applied traction on the top surface in Fig. 3a can be varied to achieve different levels of plasticity. A typical deformation and von-Mises stress plot is illustrated in Fig. 3b.

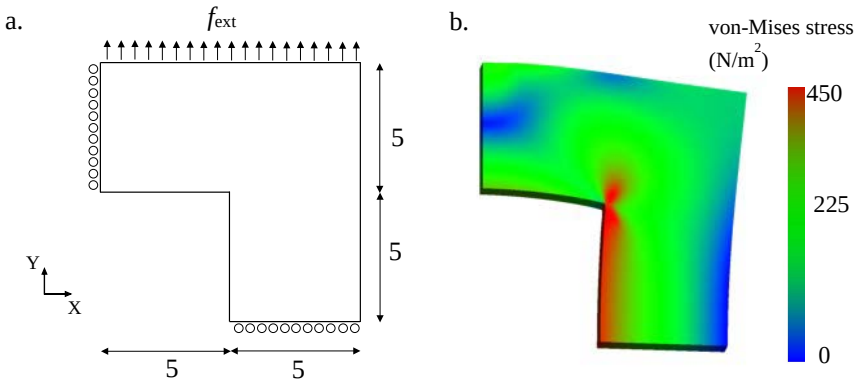

Figure 3: L Bracket geometry subject to plastic deformation.

The benchmark experiments are conducted under the following conditions:

- The stopping criterion ε_{Newton} in Algorithm 1 is 10^{-8}
- The stopping criterion ε_{CG} in Algorithm 2 is 10^{-9}.
- The material parameters are (to be consistent with [23]): $E = 206900\ N/m^2$ (Young's modulus), $\nu = 0.29$ (Poisson's ratio), and yield stress Y=450 N/m^2.
- The 3D geometry is discretized into 1,000,000 hexahedral elements (unless otherwise stated), and 8 Gauss quadrature points are used per element for numerical integration.
- The implementation is in C++, on a standard Windows 10 desktop with Intel(R) Core(TM) i9-9820X CPU running at 3.3 GHz with 16 GB memory. For parallel implementation, OpenMP is used for matrix-vector $K_t\delta u$ (SpMV) operations (see Section 4.2) performed in parallel.
- For the GPU implementation, NVIDIA GeForce RTX 2070 with 2304 cores and 8GB GDDR6 memory configuration) with CUDA is used.

4.1. Questions being Investigated

The questions being investigated through the experiments below are:

- **SpMV**: Does the proposed matrix-free SpMV implementation for $K_t\delta u$ retain its efficiency in the presence of plasticity? How does the performance depend on the number of elements?
- **Deflated Matrix**: Is the proposed incremental method for computing the deflated matrix $W^T K_t W$ significantly better than the conventional approach? How does it depend on the percentage of plasticity and the number of elements?
- **Deflation**: Does deflation improve CG convergence in the presence of plasticity, and how does the convergence depend on the number of elements?

- **ANSYS Comparison**: Is the proposed method accurate, and how does its accuracy and performance compare against a typical commercial solver such as ANSYS?
- **Impact of Voxelization**: What is the impact of voxelization, as compared to a conforming mesh, on the percentage of plasticity predicted?

Each of these questions is answered in the following sections.

4.2. Experiments on SpMV

The first experiment focuses on the SpMV operation, i.e., computing $K_t \delta u$, since it is by far the most time-consuming part of the algorithm. For linear elasticity, it was reported in [20] that significant gains in SpMV can be achieved via a matrix-free approach. The objective here is to investigate the same via the modified SpMV approach for different levels of plasticity.

The results for computation using CPU are summarized in Fig. 4 for 1 million elements; they confirm that matrix-free SpMV is indeed much more efficient than the classical assembled approach, even for large percentages of plasticity (for simplicity, the reported time for the assembled approach is for zero plasticity; the reported time increases with increasing plasticity).

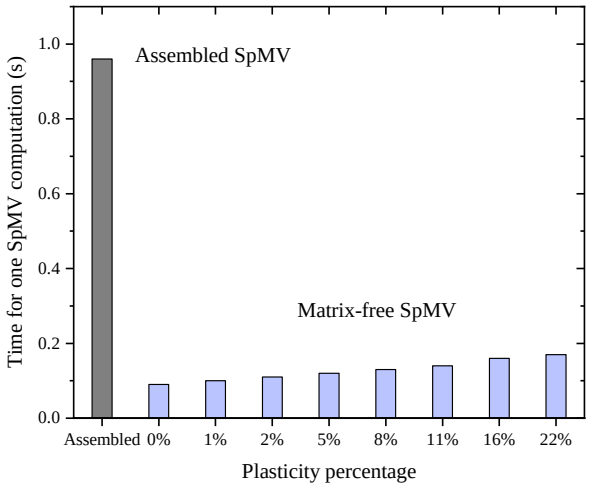

Figure 4: SpMV computation time for increasing plasticity percentage.

Next, the percentage of plasticity was kept constant at 1% and the number of elements was varied, and the time taken to carry out a matrix-free SpMV was noted. As one can observe in Fig. 5, matrix-free SpMV behaves linearly with respect to the number of elements. This result is to be expected; similar results were observed for higher percentage of plasticity as well. Note that the relationship between the degrees of freedom N_{DOF} and the number of elements n_e depends on the structure of the mesh; for this example, the two were approximately related as follows: $N_{DOF} = 3.002 \times n_e$.

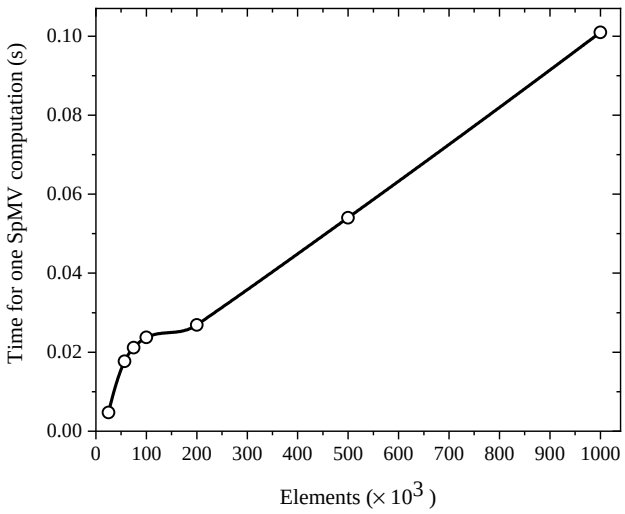

Figure 5: SpMV computation time for increasing number of elements.

4.3. Experiments on Deflated Matrix Computation

Next, we consider the proposed incremental approach to computing $W^T K_t W$ and compare it against the conventional approach of computing it afresh during each Newton iteration.

First, the number of elements is kept constant at 1 million, and the percentage of plasticity was varied. The time to compute $W^T K_t W$ via the two approaches are summarized in Fig. 6. The computational time for the conventional (full) method is almost independent of the plasticity percentage. The incremental approach is less expensive, but rises with increasing levels of plasticity. This is to be expected since the incremental portion increases with plasticity.

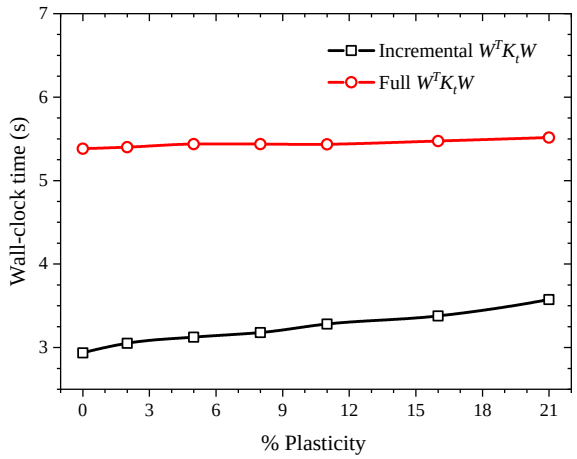

Figure 6: $W^T K_t W$ computation time for increasing plasticity percentage.

While the difference in computational gain appears to be minimal for 1 million elements, we observed a

significantly larger gain for higher density mesh. Specifically, the plasticity was kept constant at 1% and the number of elements was varied. The time comparison for the two approaches is summarized in Fig. 7. Note that, in incremental deflation, the deflation matrix is only computed when an element is/was in a plastic state. For 1% plasticity, the number of such elements is relatively small, and therefore the computation time is only weakly dependent on the total number of elements. On the other hand, for full deflation, the deflation matrix is computed for all elements. and is therefore strongly dependent on the number of elements.

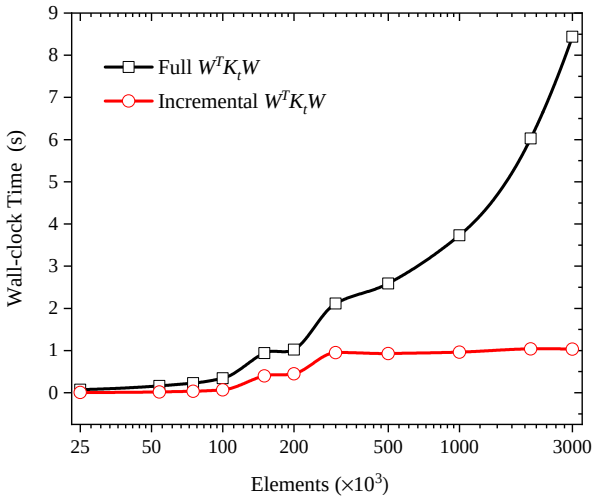

Figure 7: Time required for constructing deflated matrix $\widetilde{K_t}$ $(= W^T K_t W)$ for increasing number of elements

4.4. Experiments on Deflation

The next set of experiments is on the importance of deflation while solving Eqn. (3). It was reported in [20] that, for pure elasticity, deflation can be very effective in accelerating CG. Here, a numerical experiment with 1% plasticity was conducted with varying number of deflation groups, and the CG-convergence rate was recorded. We observe in Fig. 8 that, with no deflation, pure CG exhibits very poor convergence, while, with deflation, rapid convergence can be observed. As illustrated in Fig. 8, a few hundred deflation groups is typically sufficient.

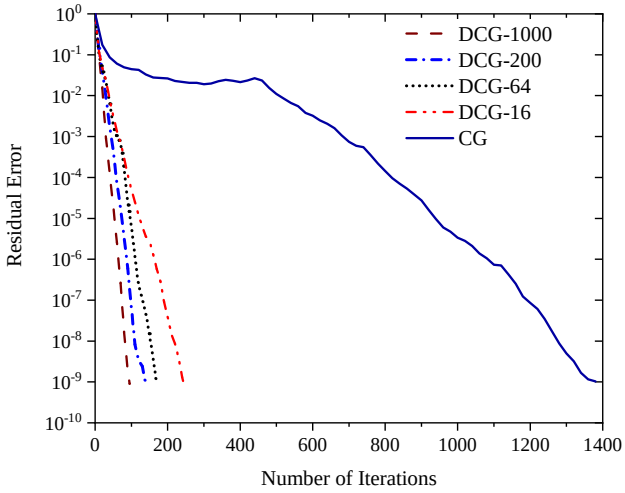

Figure 8: Impact of deflation on CG convergence.

Fig. 9 illustrates the computational time for increasing number of elements and varying DCG groups, and fixed percentage of plasticity. The importance of DCG is more pronounced as the number of elements is increased.

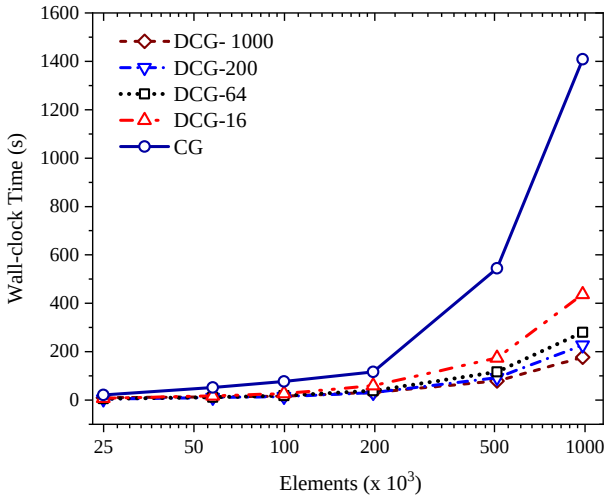

Figure 9: Impact of deflation groups on computation time.

Next, the number of elements was kept constant at 1 million, and the percentage of plasticity was varied. Fig. 10 illustrates a dependence of number of DCG iterations on plasticity. Note that the x-axis corresponds to increasing load, i.e., increasing percentage plasticity at equilibrium, while the y-axis corresponds to the observed DCG iterations, during each Newton iteration. The numerical experiments illustrate the following:

- Zero Plasticity: For a loading condition corresponding to zero plasticity at equilibrium, only two

Newton iterations ($k = 2$) are needed; further, the number of DCG iterations is small in both Newton iterations since the material is in a pure elastic state.

- Increasing Plasticity: As the loading is increased (along the x-axis), the number of Newton iterations required to reach equilibrium also increases. For the first Newton iteration ($k = 1$), the number of DCG iterations matches that of "zero plasticity" since it corresponds to an initial elastic state. For higher values of k, DCG iterations increases, but not necessarily monotonically.

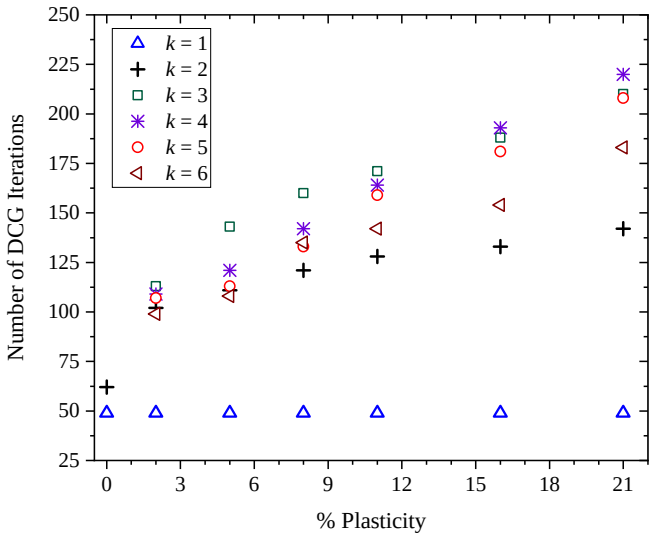

Figure 10: DCG iterations for increasing plasticity percentage

4.5. Comparison against ANSYS

The proposed method was compared with ANSYS 19.1 Mechanical APDL module for the above benchmark problem. In ANSYS, the solid 185 type brick element with 8 nodes was used with identical plasticity model, and the solver was the defaulty assembled pre-conditioned CG, supported by ANSYS.

The first comparison was on accuracy: the number of elements was kept constant at 500,000 and the percentage of plasticity was varied. Fig. 11 captures the difference in the maximum displacement between ANSYS and proposed method. The difference is minimal and can be attributed to the different convergence criteria; this difference did not vary when the number of elements was increased. Further, note that the maximum stress is the yield stress and is therefore not reported here.

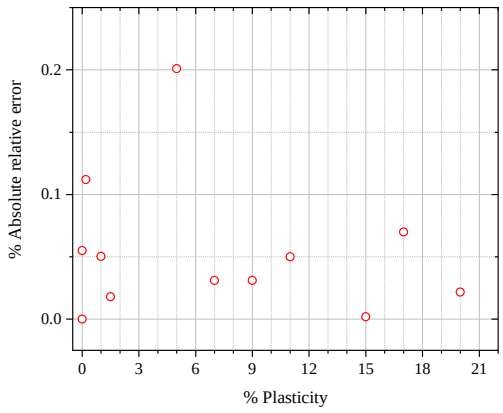

Figure 11: Difference in the maximum displacement predictions between ANSYS and proposed method.

The next numerical experiment was on speed comparison between the proposed method and ANSYS, a commercial software. Recall that steps 11 and 15 of Algorithm 2 are computationally expensive SpMV operations. Parallelization of SpMV is critical, and was implemented on the CPU through OpenMP, and on the GPU, using CUDA using the strategy proposed in [20]; see Algorithm 3.

Algorithm 3 Matrix-free SpMV on GPU

Input: $K^k_{t(\Delta)}, K^{(e)}_{(elastic)}, x$

Output: y

1: **procedure** SpMV on GPU: Host code

2: Call Cuda Kernal for SpMV: $y_{(elastic)} = \text{KElasticSpMV}(K^{(e)}_{t(elastic)}, x)$ ▷ Algorithm 4

3: Call Cuda Kernal for SpMV: $y_{(\Delta)} = \text{KDeltaSpMV}(K^k_{t(\Delta)}, x)$ ▷ Algorithm 5

4: Synchronize threads

5: $y \leftarrow y_{(elastic)} + y_{(\Delta)}$

6: **end procedure**

The pseudo code for step 2 in Algorithm 3 is described below.

Algorithm 4 Cuda Kernal for SpMV: $y_{(elastic)}$ = KElasticSpMV($K^{(e)}_{t(elastic)}, x$)

Input: $K^{(e)}_{t(elastic)}, x$

Output: $y_{(elastic)}$

procedure CUDA KERNAL FOR MATRIX-FREE SPMV: $y_{(elastic)} = K_{t(elastic)}x$

  $threadId \leftarrow blockIdx.x * blockDim.x + threadIdx.x$

  $node \leftarrow threadId$ ▷ Each node is assigned a thread

  if $node < MaxNodes$ **then**

    $NDof$ = gather_Dofs_corresponding_to_$node$_from_global_input_vector_x ()

    $Neighboring\_elements$ = find_all_Elements_associated_with_the_$node$ ()

    $temp \leftarrow 0$

    for each element $e \in Neighboring\_elements$ **do**

      dof = find_Dofs_of_element_e_corresponding_to_$node$ ()

      $temp += K^{(e)}_{t(elastic)}[dof, dof] * x[NDof]$ ▷ Note that $K^{(e)}_{t(elastic)}$ is same for all elements

    end for

    $y_{(elastic)}[NDof] \leftarrow temp$

  end if

  return $y_{(elastic)}$

end procedure

The pseudo code for step 3 in Algorithm 3 is described below.

Algorithm 5 Cuda Kernal for SpMV: $y_{(\Delta)}$ = KDeltaSpMV($K^k_{t(\Delta)}, x$)

Input: $K^k_{t(\Delta)}, x$

Output: $y_{(\Delta)}$

1: **procedure** CUDA KERNAL FOR MATRIX-FREE SPMV: $y_{(\Delta)} = K^k_{t(\Delta)}x$
2: $threadId \leftarrow blockIdx.x * blockDim.x + threadIdx.x$
3: $node \leftarrow threadId$ ▷ Each node is assigned a thread
4: $plastic\_Elements$ = set_of_all_plastic_elements()
5: **if** $node < MaxNodes$ **then**
6: $NDof$ = gather_Dofs_corresponding_to_$node$_from_global_input_vector_ x ()
7: $Neighboring\_elements$ = find_all_Elements_associated_with_the_$node$ ()
8: $temp \leftarrow 0$
9: **for each** element $e \in Neighboring\_elements$ **do**
10: **if** $e \in plastic\_Elements$ **then**
11: dof = find_Dofs_of_element_e_corresponding_to_$node$ ()
12: $K^{k(e)}_{t(\Delta)}$ = get_Elemental_$K^k_{t(\Delta)}$_correonding_to_e () ▷ $K^{k(e)}_{t(\Delta)}$ is different for all plastic elements
13: $temp += K^{k(e)}_{t(\Delta)}[dof, dof] * x[NDof]$
14: **end if**
15: **end for**
16: $y_{(\Delta)}[NDof] \leftarrow temp$
17: **end if**
18: return $y_{(\Delta)}$
19: **end procedure**

Other major operations involved in the the GPU implementation of Algorithm 2 include: (1) the restriction operation $y = W^T x$ in steps 8 and 15, and (2) the prologation operation $W\hat{\mu}$ in steps 8 and 16. In the restriction operation, each deflation group is assigned a thread as illustrated in Algorithm 6. On the other hand, each node is assigned a thread in prolongation operation (see Algorithm 7), which computes a part of the output vector corresponding to the assigned node. Low-level operations such as dot-product and vector addition were implemented using CUBLAS library.

Algorithm 6 Cuda Kernal for Restriction $y = W^T x$ operation

Input: Mesh and deflation group information (W), x

Output: Vector y

procedure CUDA KERNAL FOR RESTRICTION OPERATION: $y = W^T x$
  $threadId \leftarrow blockIdx.x * blockDim.x + threadIdx.x$
  $group \leftarrow threadId$ ▷ Each deflation group is assigned a thread
  if $group < MaxGroups$ **then**
    $GDOF$ = gather_Dofs_corresponding_to_$group$ ()
    $temp \leftarrow 0$
    for each $node \in group$ **do**
      $NDOF$ = gather_Dofs_corresponding_to_$node$_from_input_global_vector_ x ()
      $temp += W^T * x[NDOF]$
    end for
    $y[GDOF] \leftarrow temp$
  end if
  return y
end procedure

Algorithm 7 Cuda Kernal for Prolongation $t = W\hat{\mu}$ operation

Input: Mesh and deflation group information (W), $\hat{\mu}$

Output: Vector t

procedure CUDA KERNAL FOR PROLONGATION OPERATION: $t = W\hat{\mu}$
  $threadId \leftarrow blockIdx.x * blockDim.x + threadIdx.x$
  $node \leftarrow threadId$ ▷ Each node is assigned a thread
  if $node < MaxNodes$ **then**
    $group$ = find_the_Deflation_Group_of_$node$ ()
    $GDof$ = gather_Dofs_corresponding_to_$group$_from_input_global_vector_ $\hat{\mu}$ ()
    $NDof$ = gather_Dofs_corresponding_to_$node$_from_output_global_vector_ t ()
    $t[NDof] = W * \hat{\mu}[GDof]$
  end if
  return t
end procedure

The percentage of plasticity was kept constant at 1% and the number of elements was varied. Fig. 12 compares the time required for ANSYS (PCG solver) against the proposed method. As one can observe the proposed method (CPU) exhibits a 10X increase in speed for 1 million elements. Even larger gains were

observed for higher percentage of plasticity. The ANSYS direct solver was 12X slower than its PCG solver for 1 million elements.

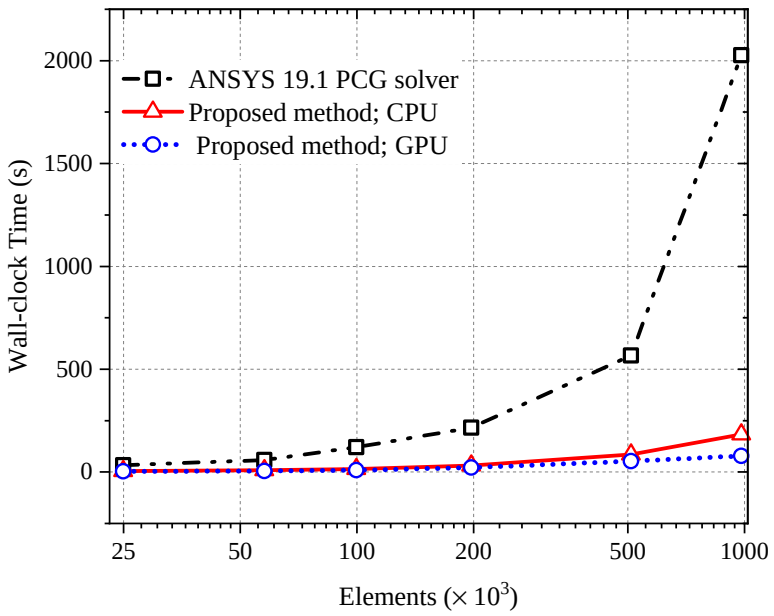

Figure 12: Computational time vs. number of elements using ANSYS and proposed method.

The data corresponding to Fig. 12 is listed in Table 1, for clarity.

Table 1: Comparing proposed method against ANSYS for increasing number of elements.

#Elements ($\times 10^3$)	Proposed method; CPU (s)	Proposed method; GPU (s)	ANSYS; PCG/CPU (s)
25	4.31	2.57	33.60
57	8.71	4.80	58.80
100	14.31	8.21	120
200	32.15	21.41	216
500	85.39	52.07	566.77
1000	183.08	78.23	2027.08

4.6. Effect of voxelization

In the previous example, there was no meshing error induced by the voxelization. Here, we consider a filleted L-Bracket illustrated in Fig. 13. The boundary conditions are identical to the boundary conditions on the L-bracket geometry without fillet; see Fig. 3a.

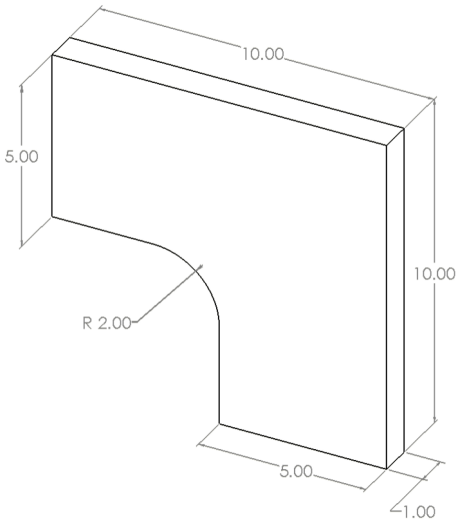

Figure 13: L Bracket with fillet geometry.

Fig. 14a illustrates the conforming mesh used in ANSYS simulation with approximately 100K elements, while Fig. 14b illustrates the voxel-based mesh used in this paper, with approximately 100K elements.

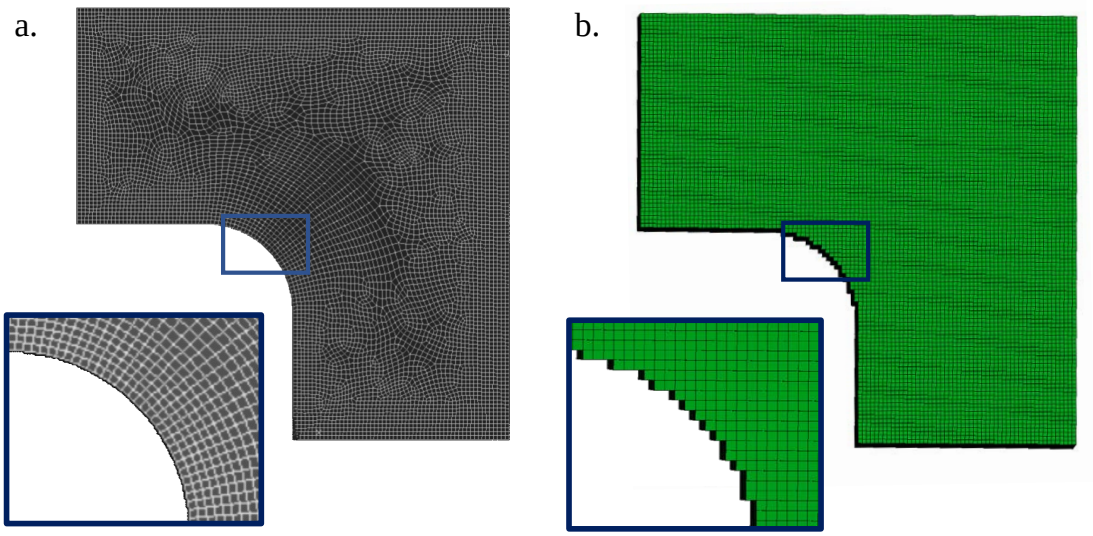

Figure 14: Mesh discretization: (a) conforming mesh (ANSYS) and (b) voxel mesh (proposed).

The objective here is to compare the percentage plasticity predicted by ANSYS and proposed method, i.e., to study the impact of voxelization on predicted plasticity.

In the first experiment, the load was kept constant at 1440 N, and the number of elements was varied. Fig. 15 compares the percentage plasticity predicted by ANSYS, and the proposed method. While the two differ by around 1%, observe that even with a conforming mesh, the ANSYS predicted percentage varies by almost 1%.

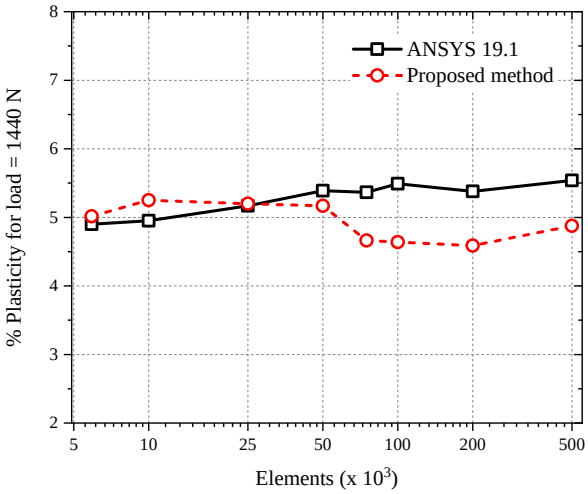

Figure 15: Comparison of plasticity percentage obtained by ANSYS and proposed method for varying number of elements at a fixed load.

Next, the number of elements was kept constant at 100K, and the load was varied (see Fig. 16). Once again, the plasticity percentage predicted differ, by at most 1%.

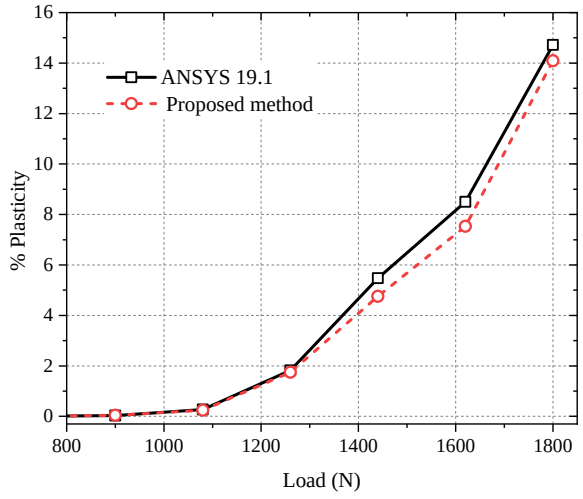

Figure 16: Percentage plasticity predicted by ANSYS and proposed method for increasing loads, with fixed number of elements.

5. Residual Stress Prediction in SLM

As mentioned earlier, residual stress predictions in SLM is typically carried out in two distinct phases: (1) a transient thermal analysis (either layer-by-layer or scan-by-scan), followed by a (2) elasto-plastic analysis that uses the thermal load to compute residual stresses (again layer-by-layer, or scan-by-scan).

In this paper, since the focus is on the second phase, we will replace the thermal analysis with an inherent strain approach [38], [39]. The inherent strain approach creates an equivalent thermal load via an off-line thermal-analysis on a sample line/patch. It is beyond the scope of this paper to discuss the merits and weaknesses of the inherent strain strategy [40]. We merely use inherent strain analysis to provide the thermal load for elasto-plastic analysis, and this can be replaced by a full scale thermal analysis, if necessary.

The sample part [38], [41] is illustrated in Fig. 17. The part is discretized using 662,892 voxel elements (740,499 nodes). The finite element mesh was illustrated earlier in Fig. 1. The material parameters are: E= 104000 MPa (Young's modulus), ν=0.33 (Poisson's ratio), Y=768 MPa (Yield strength). The previously mentioned elasto-plastic model was employed. The inherent strain vector was set to (-0.009, 0.007, -0.009) [42] where the three components are in the following order: (1) parallel to scan direction, (2) along the build direction and (3) transverse to scan direction in-plane.

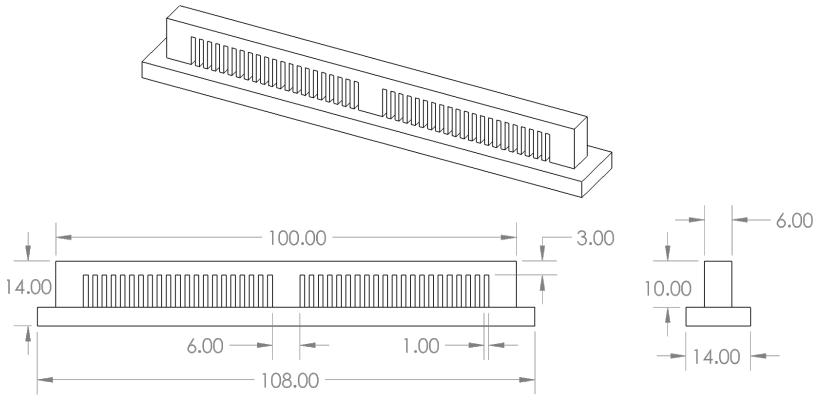

Figure 17: Twin-cantilever geometry with dimensions in mm.

We have chosen a layer-by-layer simulation [42] for demonstration purposes:

- The part (excluding the base plate whose thickness is 4 mm as illustrated) is divided in to 20 numerical layers; the height of each layer is 0.5 mm.
- Initially, the first layer is activated while other layers are kept inactive. Inherent strain is applied to the first layer (equivalent to a thermal load); Newton iteration is performed to compute the plastic strain and residual stresses.
- Inherent strain is then removed from the first layer (but the plastic strain is retained for accumulation). Then the second layer is activated and inherent strain is applied, followed by Newton iteration.
- The process is repeated until layers are completed.

Typically, at the end of the simulation, a cutting process is simulated [39], [41].

Fig. 18 illustrates the plasticity percentage in the entire model as the layers are progressively deposited. Observe that the percentage is fairly low, attesting to our prior assumptions. Observe that at layer 13 there

is an abrupt change in geometry and a corresponding increase in total number of elements. It was observed that only elements above the comb segments turned plastic, and we therefore observe a drop in percentage plasticity.

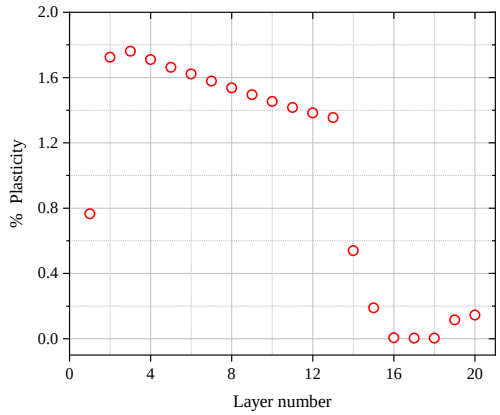

Figure 18: Percent plasticity in the model as layers are deposited.

A typical deformation plot is illustrated in Fig. 19 and the stress plot is illustrated in Fig. 20. The wall-clock time for the full simulation was 6 hours. (Due to CPU-time licensing restrictions, the equivalent model could not be executed on ANSYS.)

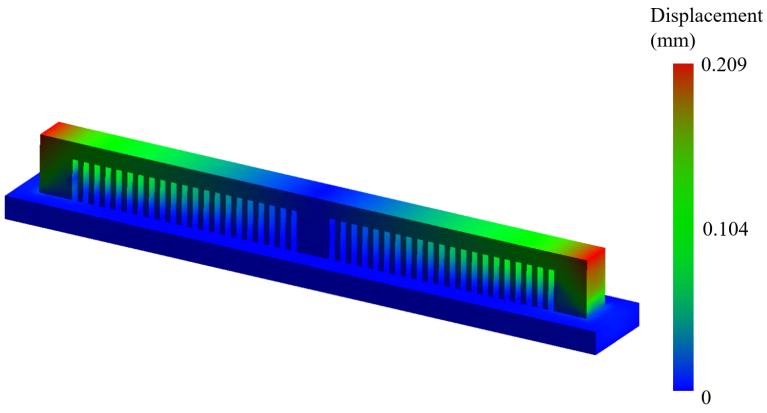

Figure 19: Twin-cantilever residual deformation (unit:mm).

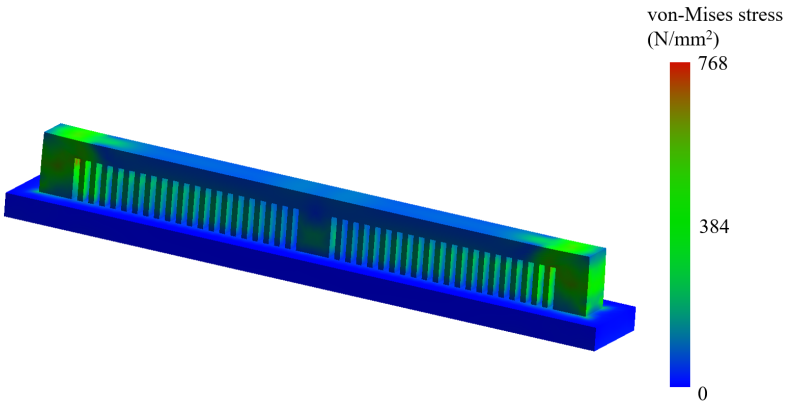

Figure 20: Twin-cantilever von-Mises stress.

6. Conclusions

In this paper, we have proposed and demonstrated a fast elasto-plastic solver that can be employed to solve a variety of structural problems. In particular, the proposed solver is well suited for residual stress and deformation predictions in selective laser melting (SLM) where one must repeatedly solve a series of elasto-plastic problems over a large finite element mesh. The ability to rapidly predict residual stresses is essential for process understanding and improvement. The proposed extension to the fast deflated matrix-free solver [20] to elasto-plastic problems is a step in this direction.

In the introduction, we argued that 10,000 layers is typical needed for SLM. However, in the numerical experiments, we limited ourselves to 20 layers due to practical (computational time) constraints. Given the scalable nature of the proposed algorithm, we believe that 10,000 layers is achievable with further improvements, and will be addressed in the future. Additional future work includes: (1) replacing the inherent strain analysis with transient thermal analysis [43], (2) extending the solver to other plasticity models, and (3) comparing the predictions against experimental results.

Acknowledgements

The authors would like to acknowledge the support from National Science Foundation (NSF) through the grant CMMI-1561899. Prof. Suresh is a consulting Chief Scientific Officer of SciArt, Corp, which has licensed the Pareto technology, developed in Prof. Suresh's lab, through Wisconsin Alumni Research Foundation.

Appendix A. Elasto-plastic material modeling

Appendix A.1. Elasto-plastic material model

The materials used in SLM are usually ductile metals which can be modeled as elastic-perfectly plastic. To describe the underlying model, small deformation assumption with associative flow rule is widely employed [44].

Under small deformation plasticity theory, the (total) infinitesimal strain $\boldsymbol{\varepsilon}$ can be decomposed into a plastic component $\boldsymbol{\varepsilon}_p$ and an elastic component $\boldsymbol{\varepsilon}_e$:

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}_p + \boldsymbol{\varepsilon}_e. \tag{A.1}$$

Assuming isotropic behavior, the Cauchy stress tensor $\boldsymbol{\sigma}$ is obtained using the elasticity tensor $\mathbb{C}$:

$$\boldsymbol{\sigma} = \mathbb{C} : (\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_p), \tag{A.2}$$

where

$$\mathbb{C} = K\mathbf{I} \otimes \mathbf{I} + 2G\mathbb{I}_D. \tag{A.3}$$

Here, G and K are rigidity and bulk moduli of the material, respectively. $\mathbb{I}_D = \mathbb{I} - \frac{1}{3}\mathbf{I} \otimes \mathbf{I}$, where $\mathbf{I} \otimes \mathbf{I}$ is tensor product of unit second order tensors. To determine the plastic strain $\boldsymbol{\varepsilon}_p$ in Eqn. A.2, knowledge of flow rule and hardening behavior is required, and this is described next.

The state of the material – elastic or plastic – is determined by the yield function Ψ, where the yield surface (plastic envelope) is represented as $\Psi = 0$. For ductile metals, yielding is commonly described by pressure independent von-Mises yield criterion. According to this criterion, plastic yielding begins when the deviatoric stress reaches a critical value, and the yield function is given by:

$$\Psi(\boldsymbol{\sigma}, \boldsymbol{\beta}) = |\mathbf{s}_{tr}(\boldsymbol{\sigma}, \boldsymbol{\beta})| - Y. \tag{A.4}$$

Here, $Y = \sqrt{\frac{2}{3}}\sigma_y$ where σ_y is the initial yield stress, and the deviatoric stress is given by:

$$\mathbf{s}_{tr} = \mathbb{I}_{\mathrm{D}}\boldsymbol{\sigma} - \boldsymbol{\beta}, \tag{A.5}$$

where $\boldsymbol{\beta}$ is the hardening variable relating thermodynamical forces.

The variable $\boldsymbol{\beta}$ depends on the hardening model used; for linear kinematic hardening,

$$\boldsymbol{\beta} = a\boldsymbol{\chi} \tag{A.6}$$

where a is the (positive) hardening constant and $\boldsymbol{\chi}$ is internal hardening variable. In this study, we assume $a = 0$, i.e., a perfectly plastic material. For perfect plasticity, $\boldsymbol{\beta}$ and $\boldsymbol{\chi}$ variables drop out of the model. However, we have included them in the discussion since the algorithms proposed in the paper can be easily applied to linear kinematic or isotropic hardening behavior.

In metals, associative flow rule is typically employed to find the incremental plastic strain and hardening variable. Incremental plastic strain is given by:

$$\dot{\boldsymbol{\varepsilon}}_p = \dot{\lambda}\frac{\partial \Psi}{\partial \boldsymbol{\sigma}} \tag{A.7}$$

and the variable $\boldsymbol{\chi}$ evolves according to:

$$\dot{\boldsymbol{\chi}} = -\dot{\lambda}\frac{\partial \Psi}{\partial \boldsymbol{\beta}} \tag{A.8}$$

where, $\dot{\lambda}$ is the plastic multiplier.

Finally, Karush-Kuhn-Tucker (KKT) and consistency conditions are required to complete the set of constitutive model to describe plastic loading and unloading:

$$\dot{\lambda} \geq 0, \;\; \Psi\left(\boldsymbol{\sigma}, \boldsymbol{\beta}\right) \leq 0, \;\; \dot{\lambda}\Psi\left(\boldsymbol{\sigma}, \boldsymbol{\beta}\right) = 0 \quad (KKT\ conditions) \tag{A.9}$$

$$\dot{\lambda}\dot{\Psi}\left(\boldsymbol{\sigma}, \boldsymbol{\beta}\right) = 0 \;\; (consistency\ condition) \tag{A.10}$$

In the finite element context, to discretize Eqns. A.7 and A.8, we employ an unconditionally stable (implicit) backward Euler scheme, using the elastic predictor-plastic corrector method [22] described below.

Appendix A.2. Elastic predictor-plastic corrector method

Given the (total) infinitesimal strain tensor $\boldsymbol{\varepsilon}^k$ at time step k and the variables $\boldsymbol{\varepsilon}_p^{k-1}$ and $\boldsymbol{\chi}^{k-1}$ at time step $k-1$, the aim is to find plastic strain $\boldsymbol{\varepsilon}_p^k$, tangent stiffness operator D_t^k and $\boldsymbol{\chi}^k$ at time step k.

Step 1: Elastic predictor

With known $\boldsymbol{\varepsilon}^k$, $\boldsymbol{\varepsilon}_p^{k-1}$ and $\boldsymbol{\chi}^{k-1}$, trial Cauchy stress tensor $\boldsymbol{\sigma}_{tr}^k$ is computed according to A.2 as:

$$\boldsymbol{\sigma}_{tr}^k = \mathbb{C} : \left(\boldsymbol{\varepsilon}^k - \boldsymbol{\varepsilon}_p^{k-1}\right) \tag{A.11}$$

According to Eqn. A.4, von-Mises yield criterion becomes:

$$\Psi\left(\boldsymbol{\sigma}_{tr}^k, \boldsymbol{\beta}^{k-1}\right) = \left|\mathbf{s}_{tr}\left(\boldsymbol{\sigma}_{tr}^k, \boldsymbol{\beta}^{k-1}\right)\right| - Y \tag{A.12}$$

and from Eqn. A.5,

$$\mathbf{s}_{tr} = \mathbb{I}_D \boldsymbol{\sigma}_{tr}^k - \boldsymbol{\beta}^{k-1}. \tag{A.13}$$

If $\Psi\left(\boldsymbol{\sigma}_{tr}^k, \boldsymbol{\beta}^{k-1}\right) \leq 0$, then the current step is an elastic step with

$$\Delta\dot{\lambda} = 0, \;\; \boldsymbol{\sigma}^k = \boldsymbol{\sigma}_{tr}^k, \;\; \boldsymbol{\varepsilon}_p^k = \boldsymbol{\varepsilon}_p^{k-1}, \;\; \boldsymbol{\chi}^k = \boldsymbol{\chi}^{k-1}, \;\; \boldsymbol{\beta}^k = \boldsymbol{\beta}^{k-1}, \;\; D_t^k = \mathbb{C}. \tag{A.14}$$

Otherwise, if $\Psi\left(\boldsymbol{\sigma}_{tr}^k, \boldsymbol{\beta}^{k-1}\right) > 0$, then the algorithm proceeds to the next step.

Step 2: Plastic corrector

In this step, $\Delta\dot{\lambda} > 0$ and the flow rules are discretized using implicit Euler scheme to give the following expressions [23] for linear kinematic hardening with von-Mises yield criteria:

$$\boldsymbol{\varepsilon}_p^k = \boldsymbol{\varepsilon}_p^{k-1} + \frac{\left(\left|\mathbf{s}_{tr}^k\right| - Y\right)\mathbf{n}_{tr}^k}{2G + a} \tag{A.15}$$

and

$$\boldsymbol{\beta}^k = \boldsymbol{\beta}^{k-1} + \frac{a\left(\left|\mathbf{s}_{tr}^k\right| - Y\right)\mathbf{n}_{tr}^k}{2G + a} \tag{A.16}$$

where, $\mathbf{n}_{tr}^k = \mathbf{s}_{tr}^k / \left|\mathbf{s}_{tr}^k\right|$. Cauchy stress tensor is updated as:

$$\boldsymbol{\sigma}^k = \boldsymbol{\sigma}_{tr}^k - \frac{2G\left(\left|\mathbf{s}_{tr}^k\right| - Y\right)\mathbf{n}_{tr}^k}{2G + a} \tag{A.17}$$

and consistent tangent operator is given as:

$$D_t^k = \mathbb{C} - 2GI_D + \frac{4G^2}{2G + a}\frac{Y}{\left|\mathbf{s}_{tr}^k\right|}\left(\mathbb{I}_D - \mathbf{n}_{tr}^k \otimes \mathbf{n}_{tr}^k\right). \tag{A.18}$$

Appendix A.3. Non-linear FEA for elastic-perfectly plastic material

In the framework of non-linear FEA with Newton-like iterative method, the above predictor corrector algorithm is used to obtain the incremental displacement vector δu^k at each step (Newton iteration) k. The procedure to obtain incremental displacements for elastic-perfectly plastic material model ($a = 0$), for small strain plasticity with associative flow rule and von-Mises yield criteria (J-2 plasticity) is summarized below.

Given nodal displacement vector u^{k-1} and plastic strain tensor $\boldsymbol{\varepsilon}_p^{k-1}$ from previous step, the goal is to obtain incremental displacement vector δu^k and thus, nodal displacement vector u^k at the current (k^{th}) Newton iteration. From nodal displacement u^{k-1}, the total strain is written as

$$\boldsymbol{\varepsilon}^k = Bu^{k-1}, \tag{A.19}$$

where B is strain-displacement matrix. With plastic strain $\boldsymbol{\varepsilon}_p^{k-1}$ and total strain $\boldsymbol{\varepsilon}^k$ known, we compute trial stress $\boldsymbol{\sigma}_{tr}^k$ given by Eqn. A.11 and trial deviatoric stress $\mathbf{s}_{tr}^k = \mathbb{I}_{\mathrm{D}}\boldsymbol{\sigma}_{\mathrm{tr}}^{\mathrm{k}}$. Depending on the yield criterion, Cauchy stress tensor is updated as follows:

$$\boldsymbol{\sigma}^k = \begin{cases} \boldsymbol{\sigma}_{tr}^k, \ \text{ if } \left|\mathbf{s}_{tr}^k\right| \leq Y \\ \\ \boldsymbol{\sigma}_{tr}^k - \left(\left|\mathbf{s}_{tr}^k\right| - Y\right)\mathbf{n}_{tr}^k, \ \text{ if } \left|\mathbf{s}_{tr}^k\right| > Y \end{cases} \tag{A.20}$$

where, $\mathbf{n}_{tr}^k = \mathbf{s}_{tr}^k / \left|\mathbf{s}_{tr}^k\right|$. This expression of Cauchy stress tensor is used for computing internal force vector F_{int} given by Eqn. 2.

The consistent tangent operator is obtained by the expression,

$$D_t^k = \begin{cases} \mathbb{C}, \ \text{ if } \left|\mathbf{s}_{tr}^k\right| \leq Y \\ \\ \mathbb{C} - 2GI_D + 2G\frac{Y}{\left|\mathbf{s}_{tr}^k\right|}\left(\mathbb{I}_D - \mathbf{n}_{tr}^k \otimes \mathbf{n}_{tr}^k\right), \ \text{ if } \left|\mathbf{s}_{tr}^k\right| > Y \end{cases} \tag{A.21}$$

This is employed to compute the tangent stiffness matrix in Eqn. 4.

Using F_{int} and D_t^k, incremental nodal displacement vector δu^k is then obtained by solving Eqn. 3. Using δu^k, we obtain the nodal displacement vector u^k using Eqn. 5.

Plastic strain is updated at every step k as:

$$\boldsymbol{\varepsilon}_p^k = \begin{cases} \boldsymbol{\varepsilon}_p^{k-1}, \ \text{ if } \ \left|\mathbf{s}_{tr}^k\right| \leq Y \\ \\ \boldsymbol{\varepsilon}_p^{k-1} + \frac{\left(\left|\mathbf{s}_{tr}^k\right| - Y\right)\mathbf{n}_{tr}^k}{2G}, \ \text{ if } \ \left|\mathbf{s}_{tr}^k\right| > Y \end{cases} \tag{A.22}$$

Generally, when $k = 1$, u^0 and $\boldsymbol{\varepsilon}_p^0$ values are initialized to zero.

References

[1] D. Herzog, V. Seyda, E. Wycisk, C. Emmelmann, Additive manufacturing of metals, Acta Materialia 117 (2016) 371–392.

[2] N. Levkulich, S. Semiatin, J. Gockel, J. Middendorf, A. DeWald, N. Klingbeil, The effect of process parameters on residual stress evolution and distortion in the laser powder bed fusion of ti-6al-4v, Additive Manufacturing 28 (2019) 475–484.

[3] E. Liverani, S. Toschi, L. Ceschini, A. Fortunato, Effect of selective laser melting (slm) process parameters on microstructure and mechanical properties of 316l austenitic stainless steel, Journal of Materials Processing Technology 249 (2017) 255–263.

[4] M. Behandish, A. M. Mirzendehdel, S. Nelaturi, A classification of topological discrepancies in additive manufacturing, Computer-Aided Design 115 (2019) 206–217.

[5] B. J. Walker, B. L. Cox, U. Cikla, G. M. De Bellefon, B. Rankouhi, L. J. Steiner, P. Mahadumrongkul, G. Petry, M. Thevamaran, R. Swader, et al., An investigation into the challenges of using metal additive manufacturing for the production of patient-specific aneurysm clips, Journal of Medical Devices 13 (3).

[6] J. G. Michopoulos, A. P. Iliopoulos, J. C. Steuben, A. J. Birnbaum, S. G. Lambrakos, On the multiphysics modeling challenges for metal additive manufacturing processes, Additive Manufacturing 22 (2018) 784–799.

[7] S. Paul, I. Gupta, R. K. Singh, Characterization and modeling of microscale preplaced powder cladding via fiber laser, Journal of Manufacturing Science and Engineering 137 (3) (2015) 031019.

[8] S. A. Khairallah, A. Anderson, Mesoscopic simulation model of selective laser melting of stainless steel powder, Journal of Materials Processing Technology 214 (11) (2014) 2627–2636.

[9] E. R. Denlinger, J. Irwin, P. Michaleris, Thermomechanical modeling of additive manufacturing large parts, Journal of Manufacturing Science and Engineering 136 (6) (2014) 061007.

[10] A. Bandyopadhyay, K. D. Traxel, Invited review article: metal-additive manufacturing—modeling strategies for application-optimized designs, Additive manufacturing 22 (2018) 758–774.

[11] T.-N. Le, Y.-L. Lo, H.-C. Tran, Multi-scale modeling of selective electron beam melting of ti6al4v titanium alloy, The International Journal of Advanced Manufacturing Technology (2019) 1–19.

[12] M. M. Francois, A. Sun, W. E. King, N. J. Henson, D. Tourret, C. A. Bronkhorst, N. N. Carlson, C. K. Newman, T. S. Haut, J. Bakosi, et al., Modeling of additive manufacturing processes for metals: Challenges and opportunities, Current Opinion in Solid State and Materials Science 21 (LA-UR-16-24513; SAND-2017-6832J).

[13] E. Mirkoohi, D. E. Seivers, H. Garmestani, S. Y. Liang, Heat source modeling in selective laser melting, Materials 12 (13) (2019) 2052.

[14] T. DebRoy, H. Wei, J. Zuback, T. Mukherjee, J. Elmer, J. Milewski, A. M. Beese, A. Wilson-Heid, A. De, W. Zhang, Additive manufacturing of metallic components–process, structure and properties, Progress in Materials Science 92 (2018) 112–224.

[15] P. Foteinopoulos, A. Papacharalampopoulos, P. Stavropoulos, On thermal modeling of additive manufacturing processes, CIRP Journal of Manufacturing Science and Technology 20 (2018) 66–83.

[16] Y. Li, K. Zhou, P. Tan, S. B. Tor, C. K. Chua, K. F. Leong, Modeling temperature and residual stress fields in selective laser melting, International Journal of Mechanical Sciences 136 (2018) 24–35.

[17] Y. Saad, Iterative methods for sparse linear systems, Vol. 82, siam, 2003.

[18] O. Axelsson, Iterative solution methods. cambridge univ, Press, Cambridge.

[19] N. Patil, D. Pal, H. K. Rafi, K. Zeng, A. Moreland, A. Hicks, D. Beeler, B. Stucker, A generalized feed forward dynamic adaptive mesh refinement and derefinement finite element framework for metal laser sintering—part i: Formulation and algorithm development, Journal of Manufacturing Science and Engineering 137 (4) (2015) 041001.

[20] P. Yadav, K. Suresh, Large scale finite element analysis via assembly-free deflated conjugate gradient, Journal of Computing and Information Science in Engineering 14 (4) (2014) 041008.

[21] J. L. Gustafson, The End of Error: Unum Computing, Chapman and Hall/CRC, 2017.

[22] J. C. Simo, T. J. Hughes, Computational inelasticity, Vol. 7, Springer Science & Business Media, 2006.

[23] M. Čermák, S. Sysala, J. Valdman, Efficient and flexible matlab implementation of 2d and 3d elastoplastic problems, Applied Mathematics and Computation 355 (2019) 595–614.

[24] J. R. Shewchuk, et al., An introduction to the conjugate gradient method without the agonizing pain (1994).

[25] L. N. Trefethen, D. Bau III, Numerical linear algebra, Vol. 50, Siam, 1997.

[26] T. J. Hughes, I. Levit, J. Winget, An element-by-element solution algorithm for problems of structural and solid mechanics, Computer Methods in Applied Mechanics and Engineering 36 (2) (1983) 241–254.

[27] T. J. Hughes, R. M. Ferencz, J. O. Hallquist, Large-scale vectorized implicit calculations in solid mechanics on a cray x-mp/48 utilizing ebe preconditioned conjugate gradients, Computer Methods in Applied Mechanics and Engineering 61 (2) (1987) 215–248.

[28] C. Augarde, A. Ramage, J. Staudacher, An element-based displacement preconditioner for linear elasticity problems, Computers & structures 84 (31-32) (2006) 2306–2315.

[29] A. Wathen, An analysis of some element-by-element techniques, Computer Methods in Applied Mechanics and Engineering 74 (3) (1989) 271–287.

[30] W. Briggs, V. Henson, S. McCormick, A multigrid tutorial, 1043 siam, Philadelphia, PA 1044.

[31] R. A. Nicolaides, Deflation of conjugate gradients with applications to boundary value problems, SIAM Journal on Numerical Analysis 24 (2) (1987) 355–365.

[32] P. Arbenz, G. H. van Lenthe, U. Mennel, R. Müller, M. Sala, A scalable multi-level preconditioner for matrix-free μ-finite element analysis of human bone structures, International Journal for Numerical Methods in Engineering 73 (7) (2008) 927–947.

[33] M. Adams, Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics, International Journal for Numerical Methods in Engineering 55 (5) (2002) 519–534.

[34] T. Jönsthövel, M. Van Gijzen, S. MacLachlan, C. Vuik, A. Scarpas, Comparison of the deflated preconditioned conjugate gradient method and algebraic multigrid for composite materials, Computational Mechanics 50 (3) (2012) 321–333.

[35] C. Augarde, A. Ramage, J. Staudacher, Element-based preconditioners for elasto-plastic problems in geotechnical engineering, International journal for numerical methods in engineering 71 (7) (2007) 757–779.

[36] T. Jonsthovel, M. B. van Gijzen, C. Vuik, A. Scarpas, On the use of rigid body modes in the deflated preconditioned conjugate gradient method, SIAM Journal on Scientific Computing 35 (1) (2013) B207–B225.

[37] Eigen library, `http://eigen.tuxfamily.org/index.php?title=Main_Page`.

[38] M. Bugatti, Q. Semeraro, Limitations of the inherent strain method in simulating powder bed fusion processes, Additive Manufacturing 23 (2018) 329–346.

[39] Q. Chen, X. Liang, D. Hayduke, J. Liu, L. Cheng, J. Oskin, R. Whitmore, A. C. To, An inherent strain based multiscale modeling framework for simulating part-scale residual deformation for direct metal laser sintering, Additive Manufacturing 28 (2019) 406–418.

[40] N. Keller, V. Ploshikhin, New method for fast predictions of residual stress and distortion of am parts, in: Solid Freeform Fabrication Symposium (SFF), Austin, TX, Aug, 2014, pp. 4–6.

[41] I. Setien, M. Chiumenti, S. van der Veen, M. San Sebastian, F. Garciandía, A. Echeverría, Empirical methodology to determine inherent strains in additive manufacturing, Computers & Mathematics with Applications.

[42] X. Liang, Q. Chen, L. Cheng, D. Hayduke, A. C. To, Modified inherent strain method for efficient prediction of residual deformation in direct metal laser sintered components, Computational Mechanics (2019) 1–15.

[43] N. Hodge, R. Ferencz, J. Solberg, Implementation of a thermomechanical model for the simulation of selective laser melting, Computational Mechanics 54 (1) (2014) 33–51.

[44] J. Lubliner, Plasticity theory, Courier Corporation, 2008.