

Large Scale Finite Element Analysis via Assembly-Free Deflated Conjugate Gradient

Praveen Yadav, Krishnan Suresh
suresh@engr.wisc.edu

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

Abstract

Large-scale finite element analysis with millions of degrees of freedom is becoming commonplace in solid mechanics. The primary computational bottle-neck in such problems is the solution of large linear systems of equations.

In this paper, we propose an assembly-free version of the deflated conjugate gradient (DCG) for solving such equations, where neither the stiffness matrix nor the deflation matrix is assembled. While assembly-free FEA is a well-known concept, the novelty pursued in this paper is the use of assembly-free deflation. The resulting implementation is particularly well suited for large-scale problems, and can be easily ported to multi-core CPU and GPU architectures. For demonstration, we show that one can solve a 50 million degree of freedom system on a single GPU card, equipped with 3 GB of memory.

The second contribution is an extension of the “rigid-body agglomeration” concept used in DCG to a “curvature-sensitive agglomeration”. The latter exploits classic plate and beam theories for efficient deflation of highly ill-conditioned problems arising from thin structures.

1. INTRODUCTION

Finite element analysis (FEA) is a popular numerical method for solving solid mechanics problems. For large scale problems, the main computational bottle-neck in FEA is the solution of linear systems of equations of the form:

$$Kd = f \quad (1.1)$$

Henceforth, the matrix K will be referred to as the stiffness matrix, and it is assumed to be sparse and positive definite. Direct solvers [1] are the default choice today for solving such linear systems. Direct solvers are robust and well-understood, and rely on factoring the stiffness matrix into a Cholesky decomposition:

$$K = LL^T \quad (1.2)$$

This is followed by a triangular solve:

$$d = L^{-T}(L^{-1})f \quad (1.3)$$

However, due to the explicit factorization, direct solvers are memory intensive [2]. To

quote the ANSYS manual [3], “[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 (DOF)* [3]:

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

Since memory-access is the bottle-neck in computer architecture, this translates into an increased wall-clock time.

Large-scale FEA problems with millions of degrees of freedom (DOF) are becoming commonplace in solid mechanics; examples include multi-scale analysis [4], analysis of micro-CT data [5], and analysis of voxelized geometry [6]. Indeed, in some instances, linear systems with billions of DOF must be solved [5]. Direct solvers are ill-suited for such problems.

Instead, one must resort to iterative solvers that do not factorize the stiffness matrix, but compute the solution iteratively [7]. In iterative solvers, the main computational issues are:

1. Efficient implementation of sparse matrix-vector multiplication (SpMV).
2. Accelerating the iterative solver either through an efficient preconditioner and/or through multi-grid/deflation techniques.

Methods to increase the efficiency of SpMV (see [8] for a review) include profile and band-width reduction, efficient storage techniques, graph-theoretic optimization and specialized octree data-structures. In addition, implementation of SpMV on graphics-programmable units (GPUs) has drawn considerable attention [9], [10].

In this paper, we shall exploit element-congruency and assembly-free methods to reduce memory usage, and to accelerate SpMV and CG-iterations, both on the CPU and GPU. Assembly-free SpMV for large-scale finite element analysis on the GPU was recently proposed in [11], but element-congruency was not exploited.

2. LITERATURE REVIEW

Since the stiffness matrices considered in this paper are symmetric positive definite, the conjugate gradient (CG) is the focus of this paper [7]. As is well known [2], [7], [12], CG's convergence can be poor if the stiffness matrix exhibits high condition number, or if the eigen-values of the stiffness matrix are spread out. In solid mechanics, poor convergence of CG is fairly common [2], for example, in the analysis of composite materials, thin structures, multi-scale problems, etc.

Acceleration of CG, i.e., reduction in the number of iterations, is usually achieved through a combination of preconditioners, multi-grid methods and deflation techniques.

2.1 Preconditioners

One of the oldest preconditioners is the Jacobi preconditioner; it does not require the assembly of the stiffness matrix, and is therefore scalable and easily parallelizable. But it is not very effective for many ill-conditioned problems in solid mechanics [2]; this is confirmed later through numerical experiments. Other preconditioners such as Gauss-Seidel and SSOR perform better than Jacobi, but have similar limitations.

The incomplete Cholesky (IC) is perhaps the most robust and efficient preconditioner [13], [14]. It relies on an approximate Cholesky factorization (see Equation (1.2)) where, for example, the lower-triangular matrix L is forced to have the same sparsity-pattern as K . Unfortunately, constructing this preconditioner requires assembly of the stiffness matrix. Further, while efficient implementations of IC exist for single core systems, these are not easily portable to multi-core architectures [5].

2.2 Multi-Grid Methods

Multi-grid methods are gaining popularity for both theoretical and practical reasons. The basic concept behind a *two-level geometric* multi-grid method is illustrated in Figure 1. During a conjugate gradient iteration, the residual over a 'fine-mesh' is *restricted* to a coarse-level through a grid transfer. This is then smoothed at the coarse level, and *prolonged* back to the finer level [15], [16]. This accelerates the convergence of CG, and can be easily generalized to multiple levels for optimal convergence.

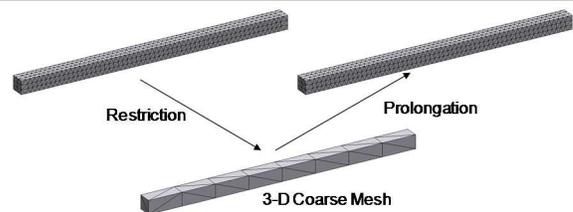


Figure 1: A two-level geometric multi-grid.

In the *algebraic* multi-grid method [17], [18], the restriction and prolongation

operators are constructed in an algebraic fashion, rather than through a geometric mesh transfer. The properties and performance are similar to that of the geometric multi-grid.

Multi-grid methods can be implemented in an assembly-free manner resulting in a low memory foot-print. This was explored by Arbenz and colleagues for large-scale solid mechanics problems [5]. While multi-grid methods perform particularly well for scalar problems and solid mechanics (vector) posed over ‘thick solids’, they are prone to Poisson locking and ill-conditioning for problems posed over ‘thin solids’ [19] and composite materials [20].

Improvements over the multi-grid method for thin structures were proposed in [19], [21], [22], where lower-dimensional models were used instead of coarse-meshes, thus avoiding the locking issue. Here, we explore deflation techniques that can address these issues in a unified manner.

2.3 Deflation

The concept behind deflation [23] is to construct a matrix W , referred to as the *deflation space*, whose columns ‘approximately’ span the low eigen-vectors of the stiffness matrix.

Since computing the eigen-vectors is obviously expensive, Adams and others [12] suggested a simple *agglomeration* technique where finite element nodes are collected into small number of groups. For example, Figure 2 illustrates agglomeration of the finite element nodes into four groups.

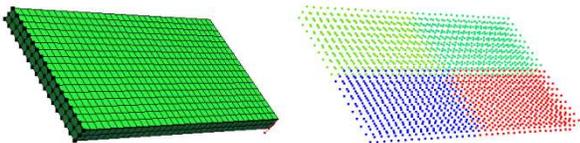


Figure 2: (a) Finite element mesh, (b) agglomeration of mesh nodes into four groups.

Then, to construct 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. Thus, for small rotations, the displacement of each node within a group can be expressed as:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{pmatrix} \lambda_g \quad (2.1)$$

where

$$\lambda_g = \{u_0, v_0, w_0, \theta_x, \theta_y, \theta_z\}^T \quad (2.2)$$

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 (2.1) is essentially a restriction operation similar to that of multi-grid. Indeed, the concept of using rigid-body modes has been explored in the context of multi-grid methods as well [17], [24].

Once the mapping in Equation (2.1) is constructed for all the nodes, these can be ‘assembled’ to result in a deflation matrix W :

$$d = W\lambda \quad (2.3)$$

where d is the $3N$ degrees of freedom, λ is the $6G$ degrees of freedom associated with the groups. One can now exploit the W matrix to create the *deflated conjugate gradient* (DCG) algorithm described below (see [23] for a derivation and theoretical analysis):

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. For $j = 1, 2, \dots, m$, do:
 5.
$$\alpha_{j-1} = \frac{r_{j-1}^T r_{j-1}}{p_{j-1}^T K p_{j-1}}$$

6. $d_j = d_{j-1} + \alpha_{j-1} p_{j-1}$
7. $r_j = r_{j-1} - \alpha_{j-1} K p_{j-1}$
8. $\beta_{j-1} = \frac{r_j^T r_j}{r_{j-1}^T r_{j-1}}$
9. Solve $W^T K W \mu_j = W^T K r_j$ for μ
10. $p_j = \beta_{j-1} p_{j-1} + r_j - W \mu_j$
11. End-do

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

- Within the DCG iteration, 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, and can be significant. Observe that the deflation matrix is also sparse; this is exploited later on for assembly-free implementation of DCG.

The optimal number of groups depends on the number of low order eigen-modes [2]. Later, we shall study the impact of group size on the computational time.

Next, we propose an improvement over the aforementioned rigid-body agglomeration, specifically for thin structures. Thin structures find a wide variety of applications across many disciplines including civil, automotive, aerospace, MEMS, etc., and efficient solution of solid mechanics problems over such structures is of significant importance.

3. THIN STRUCTURES

3.1 Motivation

The rigid-body agglomeration is very effective in capturing the low-order eigen-modes of ‘thick’ solids such as the ones in Figure 3.

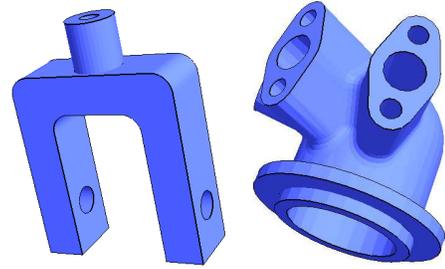


Figure 3: Example of ‘thick’ solids.

On the other hand, consider the ‘thin’ solids in Figure 4; the low order eigen-modes of such solids are significantly different from that of the thick solids in that the curvature effects are not negligible.

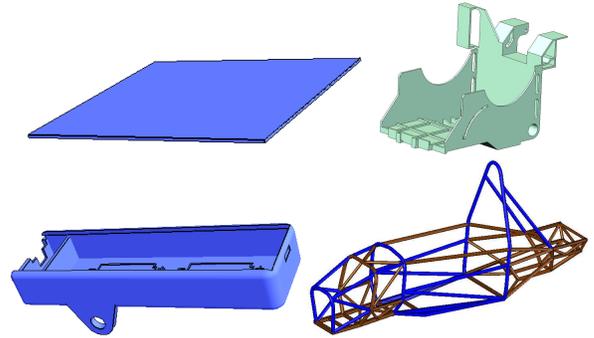


Figure 4: Examples of ‘thin’ solids.

This is illustrated schematically in Figure 5; a large number of groups will be required to effectively capture these modes through rigid-body agglomeration. An alternate concept is proposed next.

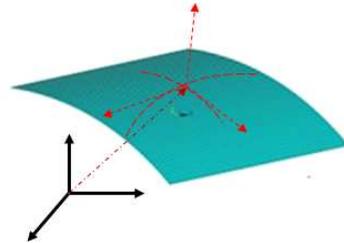


Figure 5: Curvature effects in thin structures.

3.2 Curvature-Sensitive Deflation

The proposed concept is to append the rigid body deflation in Equation (2.1) with

additional curvature variables. Specifically, consider a thin plate whose smallest dimension is in the z-direction. Equation (2.2) is extended as follows:

$$\lambda_g = \{u_0, v_0, w_0, \theta_x, \theta_y, \theta_z, w_{,xx}, w_{,yy}, w_{,xy}\}^T \quad (3.1)$$

Exploiting Kirchoff-Love theory for thin plates [25], the nodal displacement is then expressed as:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = W_n \lambda_g \quad (3.2)$$

where

$$W_n = \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y & -zx & 0 & -zy \\ 0 & 1 & 0 & -z & 0 & x & 0 & -zy & -zx \\ 0 & 0 & 1 & y & -x & 0 & \frac{x^2}{2} & \frac{y^2}{2} & xy \end{pmatrix} \quad (3.3)$$

Equation (3.3) will be referred to as “Kirchoff-Love” agglomeration.

A similar approach can be used to construct the deflation space for beam-problems. Unlike a thin plate, the curvature of the beam varies only along one major axis. Exploiting Euler-Bernoulli theory, the group variables are:

$$\lambda_g = \{u_0, v_0, w_0, \theta_x, \theta_y, \theta_z, w_{,xx}\}^T \quad (3.4)$$

and:

$$W_n = \begin{pmatrix} 1 & 0 & 0 & 0 & z & -y & -zx \\ 0 & 1 & 0 & -z & 0 & x & 0 \\ 0 & 0 & 1 & y & -x & 0 & \frac{x^2}{2} \end{pmatrix} \quad (3.5)$$

Equation (3.5) will be referred to as “Euler-Bernoulli” agglomeration.

4. LIMITED-MEMORY ASSEMBLY-FREE DCG

In this section, we consider a limited-memory assembly-free implementation of the deflated conjugate gradient. The proposed implementation is applicable to

both rigid-body and curvature-sensitive agglomeration. The focus of this Section is on a CPU implementation; GPU implementation is discussed in Section 5.

4.1 Assembly-Free FEA

Assembly-free finite element analysis was proposed by Hughes and others in 1983 [26], but has resurfaced [6] due to the surge in fine-grain parallelization.

The basic concept here is that the stiffness matrix is never assembled; instead, the fundamental matrix operations such as the SpMV are performed in an assembly-free elemental level. In other words, instead of the classic “assemble and then multiply”:

$$Kx \triangleq \left(\prod_{assemble} K_e \right) x \quad (4.1)$$

the strategy is to “multiply and then assemble”:

$$Kx \triangleq \prod_{assemble} (K_e x_e) \quad (4.2)$$

However, assembly-free analysis is not particularly advantageous over classic ‘assembled’ approach unless: (1) the total memory consumption can be reduced, and (2) CG can be accelerated in an assembly-free mode. In the remainder of this paper, we show how both of these can be achieved.

Much of the memory access in deflated conjugate gradient comes from storing/retrieving the stiffness and deflation matrices. These can be dramatically reduced if mesh-congruency can be exploited, as explained in the next Section.

4.2 Exploiting Mesh Congruency

The premise here is that in large-scale meshes, significant number of elements tends to be geometrically congruent. For example, consider the finite element mesh of a composite specimen [27] in Figure 6, consisting of about 83000 elements; the mesh was generated using ANSYS.

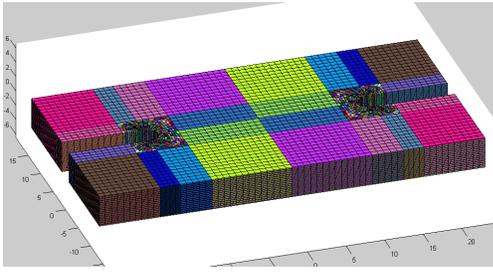


Figure 6: Congruency in a finite element mesh.

Through a simple congruency check [6], one can determine that the mesh contains only 322 distinct elements, i.e., less than 0.4%, are geometrically and materially distinct; these are located near the notch as illustrated in Figure 7.

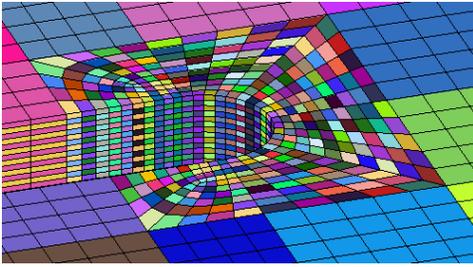
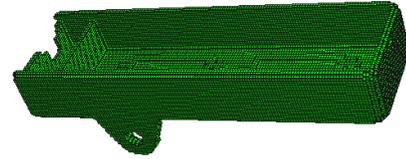


Figure 7: Most of the distinct elements are localized.

Observe that congruent elements will yield identical element stiffness matrix. Thus, in an assembly-free mode, only the distinct element stiffness matrices need to be computed and stored. This dramatically reduces the memory foot-print, and accelerates SpMV.

4.3 Mesh Partitioning

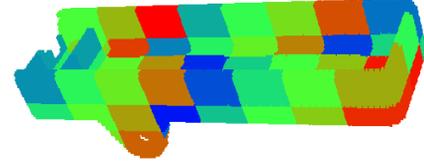
Focusing now on deflation, the first step is agglomeration, i.e., collection of mesh-nodes into a small number of groups. The groups are created by partitioning mesh with contiguous bounding boxes, and nodes are assigned to respective boxes, and empty boxes are eliminated. Figure 8a illustrates a finite element mesh with 50,000 nodes, while Figure 8b illustrates agglomeration of these nodes into 32 groups, and Figure 8c illustrates agglomeration into 64 groups.



(a) Finite element mesh.



(b) Partitioning into 32 groups.



(c) Partitioning into 64 groups.

Figure 8: Partitioning mesh-nodes into groups.

4.4 Assembly-Free Deflation

Just as the stiffness matrix is never assembled to carry out Kx , the deflation matrix W need not be assembled to carry out the two primary operations, namely, $W\lambda$ and $W^T x$. The non-zero values of deflation matrix are either 1 or some combination of relative nodal coordinates. Therefore, both these operations only require the nodal coordinates, and the group mapping.

As before, instead of “assemble and then multiply”, we have “multiply and then assemble”:

$$W\lambda \triangleq \prod_{assemble} (W_n \lambda_n) \quad (4.3)$$

$$W^T x \triangleq \prod_{assemble} (W_n^T x_n) \quad (4.4)$$

Observe that the difference between Equation (4.2) and Equation (4.3) is that the former is an assembly over elements, while the latter is an assembly over nodes.

4.5 Reduced Stiffness Matrix Deflation

Recall in step-3 of the DCG, one must compute the reduced matrix:

$$\tilde{K} \triangleq W^T K W \quad (4.5)$$

In this paper, we assume that the number of groups (G) is much less than the number of nodes (N). Therefore the size (6G x 6G) of the reduced matrix is small compared to the size of the stiffness matrix.

The reduced matrix is computed element-by-element as follows. The element stiffness matrix is divided into an 8x8 block where each block is a 3x3 matrix associated with the degrees of freedom of a node:

$$K_e = \begin{pmatrix} k_{11} & \dots & k_{18} \\ \vdots & \ddots & \vdots \\ k_{81} & \dots & k_{88} \end{pmatrix} \quad (4.6)$$

We then compute Equation (4.5) as follows:

$$W^T K W = \prod_{\text{element assembly}} \sum_{i=1}^8 \sum_{j=1}^8 \{(W_n^T)_j k_{ij} (W_n)_i\} \quad (4.7)$$

Equation (4.7), even though assembly free, is not parallel friendly due to race condition. It is computed once in the CPU, and its Cholesky decomposition is stored for repeated solve.

5. GPU IMPLEMENTATION

In this section we outline the steps for implementing DCG on GPU.

5.1 SpMV

As mentioned earlier, the Sparse Matrix Vector Multiplication (SpMV) Kx is the most expensive computation in DCG. Direct implementation of Equation (4.2) suggests that we assign a thread to each element and update the result element-by-element. However, this obviously creates a race condition when a nodal index connected to multiple elements is simultaneously accessed.

Therefore, a thread is assigned to each node. Then, for all neighboring elements the stiffness coefficients associated with the node and their corresponding nodal DOF are gathered; this is illustrated in Figure 9. This ensures that the product $K_e x_e$ is computed without race conditions.

The memory access for gathering nodal DOF is unfortunately not coalesced since the DOFs are staggered based on element connectivity. However, once the result is computed the update in device memory is coalesced.

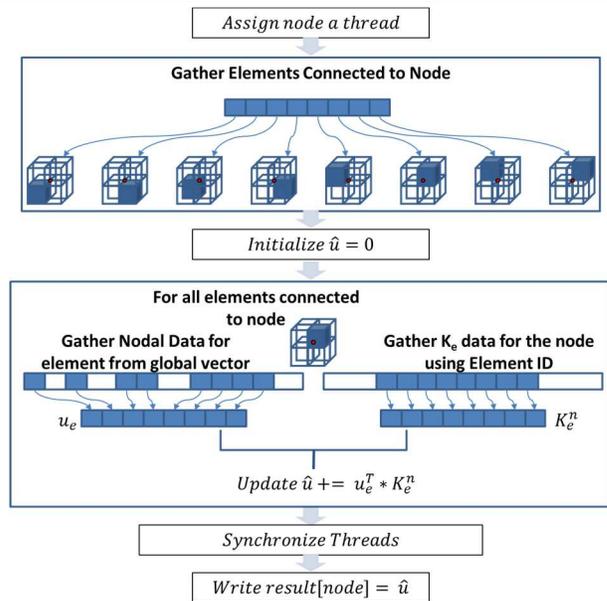


Figure 9: SpMV implementation in GPU.

5.2 Prolongation Operation

The prolongation operation $W\lambda$ is straight forward in that each thread can be assigned to a node. The corresponding group number is determined, and Equation (4.3) is executed. Figure 10 illustrates a schematic diagram of the prolongation operation.

Memory access for prolongation is coalesced for the most part. The nodes can gather the nodal coordinates (x, y, z) in a lock-step method. However gathering the group DOF required for prolongation has the potential for bank conflict. Since the length of the vector associated with a group is small, this is not a serious issue; the result update is fully coalesced.

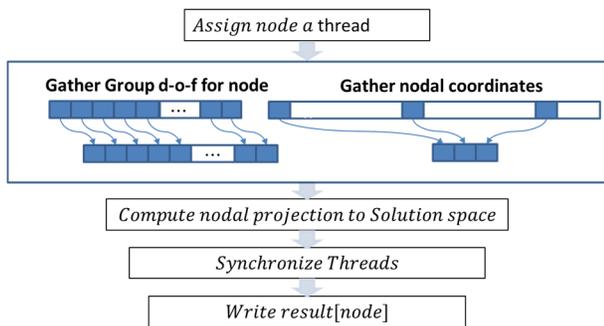


Figure 10: GPU implementation of prolongation.

5.3 Restriction Operation

The restriction operation $W^T x$ is much more challenging to parallelize on the GPU due to potential race conditions. Instead of assigning a thread to each node, a block of threads is assigned to a group. Nodal projections are computed for each thread using Equation (4.4) and saved in shared memory within the block; this is illustrated in Figure 11. Threads are synchronized after the shared memory update. A reduce operation is performed on respective DOFs of the nodal projection to yield resultant vector for the group. The allowable number of threads within the block is thus restricted by the shared memory.

The memory access for this part of the implementation is not coalesced either, as node indexes that belong to the group may skip a large sequence indexes. As shown in Figure 11, the warp may end up with coalesced memory access if a contiguous sequence of indexes is assigned for restriction.

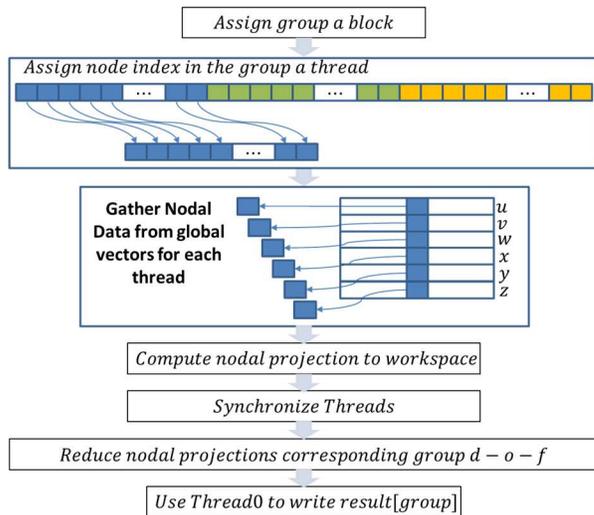


Figure 11: GPU implementation for restriction.

5.4 Other Operations

All of the other steps of CG operations are computed using the standard functions available in the CuBLAS library of CUDA SDK 4.0 [28]. This includes computing the dot products of two given vectors, performing linear vector updates using saxpy/axpy and most importantly using a dense matrix vector solver.

6. NUMERICAL RESULTS

In this section, we present numerical results of the AF-DCG CPU and GPU implementations. Unless otherwise noted, experiments were conducted on a Windows 7 64-bit machine with following specifications:

- AMD Phenom™ II X4-955 processor running at 3.2GHz with 4GB of memory; OpenMP [29] commands were used to parallelize CPU code.
- NVidia GeForce GTX 480 (448 cores) with 0.75GB of device memory.

All computations were run in double-precision, and the relative residual norm for CG-convergence was set to 10^{-8} .

6.1 Congruence of Mesh Elements

First, to illustrate the computational advantages of exploiting mesh congruence, consider the beam in Figure 12. Meshes of increasing density were constructed; observe

that all elements in the mesh are all identical.

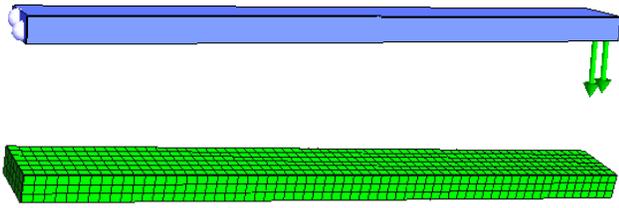


Figure 12: A beam geometry and its mesh.

The time taken to perform a single assembly-free SpMV, i.e., a single Kx , on the CPU, with and without exploiting congruency was computed; the results are summarized in Figure 13.

Observe that the computation associated with the two methods is exactly the same ... the only difference is the memory access time! Further, the overhead of computing the global K matrix has been neglected.

When congruence is not exploited, element stiffness matrices are fetched from memory as needed. With congruence exploitation, all memory requests are mapped to the single element stiffness matrix (that is likely to be in cache memory).

Figure 13 illustrates that a speed-up of 10 can be achieved in SpMV with no additional effort. Since SpMV lies at the core of all iterative methods, this has a far reaching consequence. Of course, in this scenario, the mesh contains one unique element. In practice, meshes typically contain a finite number (≥ 1) of distinct elements; the speed-up will depend on how the element stiffness matrices are grouped and accessed.

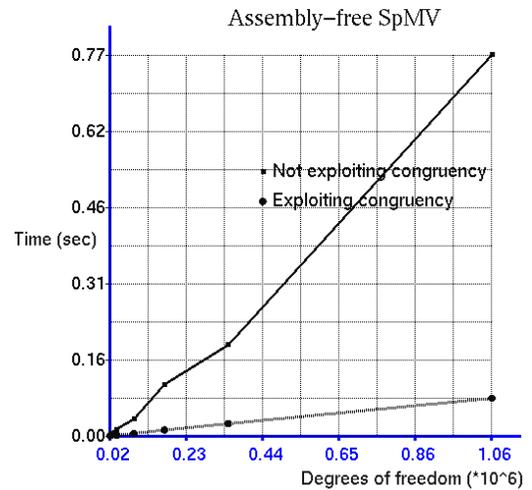


Figure 13: Assembly-free SpMV on the CPU with and without exploiting element-congruency.

6.2 Deflated-CG on a Thick Solid

Having discussed the importance of congruence exploitation, in this experiment we illustrate the impact of rigid body deflation on CG. A knuckle geometry is illustrated in Figure 14a that is fixed at the two horizontal holes, and a vertical force is applied on the third hole; observe that the geometry is relatively 'thick', i.e., there are no plate-like or beam-like features. A voxel mesh comprising of 997,626 elements (3,158,670 DOFs) was generated as illustrated in Figure 14b.

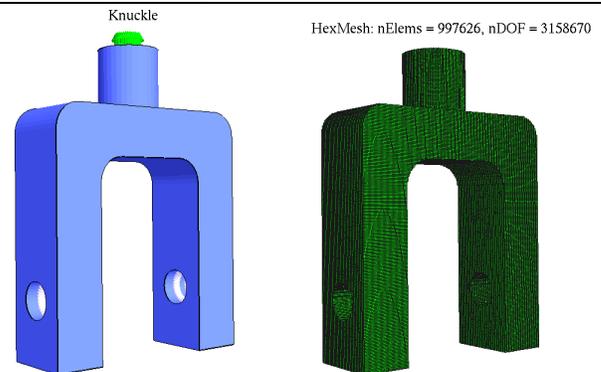


Figure 14 (a) Knuckle geometry and loading.
(b) Voxel mesh with 3.16 million DOF.

To solve the above problem, the Jacobi-PCG took 1741 iterations and 245 seconds on the CPU. The displacement and stress plots are illustrated in Figure 15.

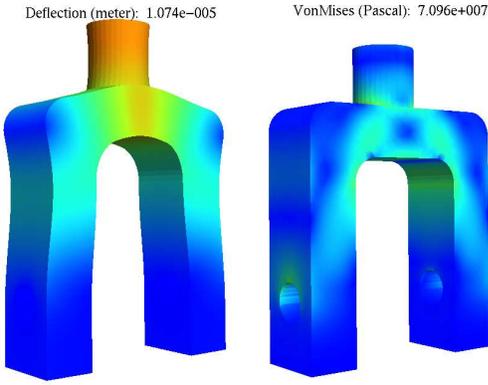


Figure 15 Static displacement and stress for knuckle.

The same system was then solved with different number of *rigid-body* agglomeration groups. For example, Figure 16 illustrates agglomeration into 100 and 1000 groups.

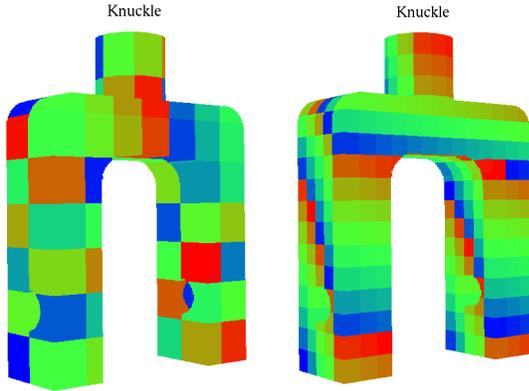


Figure 16: Visual representation of 100 and 1000 agglomeration groups.

The results for varying number of groups are summarized in Table 1. The following observations are worth noting:

- Increasing the groups from zero (pure Jacobi-PCG) to 100 groups reduces the number of CG-iterations by a factor of 10, but the CPU time reduces only by a factor of 4. The underlying reason is that every iteration in DCG entails two SpMV.
- Further, increasing the number of groups beyond a certain limit can lead to an increase in computation time. Finding an optimal number of groups is a topic of future research.

- As the number of iterations reduces, the speed-up gained through GPU also reduces as expected since the bottlenecks are the SpMV requires per iteration, and the $W^T x$ operation that is not amenable to fine-grain parallelism.

Finally, the memory requirements are fairly small even for a 3.15 million DOF.

Table 1: Total iterations and time taken to solve the knuckle with varying number of groups

G	#Iter	CPU Time (s)	GPU Time (s)	GPU Memory (MB)
0	1741	245	36	174
100	182	63	34	210
200	145	54	29	213
400	114	48	28	224
600	95	48	31	252
800	73	46	32	263
1000	69	52	39	293

The convergence plot in Figure 17 illustrates that the Jacobi-PCG converges slowly, but steadily towards the solution, without any stagnation; this is typical of solid mechanics problems posed over ‘thick’ solids. The rigid-body agglomeration leads to a dramatic drop in number of iterations as mentioned earlier.

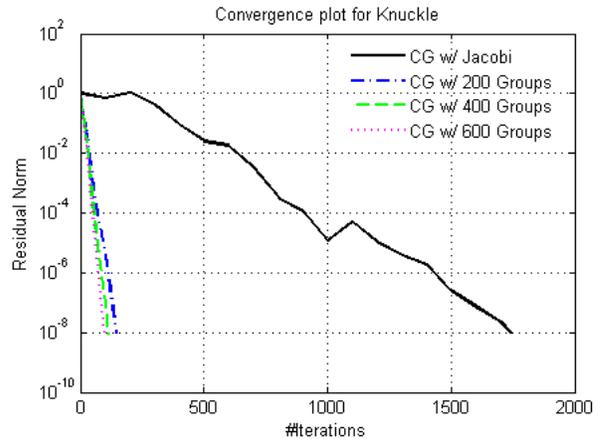


Figure 17: Convergence of DCG vs Jacobi-PCG.

6.4 Thin Solids

In this section, we consider the thin plate illustrated in Figure 18. The dimension of the plate is 100x100x5 (mm); the four side faces are fixed, with a static force applied to the top face. The geometry is discretized using a voxel mesh of 541,696 elements with 2,042,415 DOF.

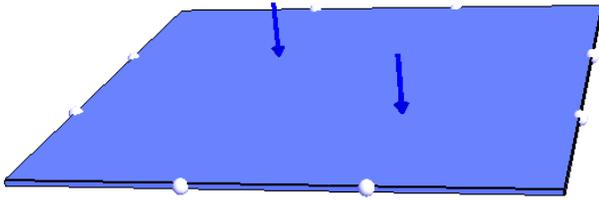


Figure 18: Loading on a thin plate.

The Jacobi-PCG converges to the solution in 6337 iteration which took an average time of 71.308 s.

6.4.1 Rigid Body Agglomeration

Rigid body deflation space was then used for DCG. The results are summarized in Table 2; the conclusions are similar those drawn earlier for the thick solid.

Table 2: Total iterations and time taken.

G	#Iter	CPU Time (s)	GPU Time (s)	GPU Memory (MB)
0	6337	550	71	113
100	736	126	36	138
200	382	72	23	146
300	260	54	23	161
400	199	48	22	178
500	166	49	26	205
600	144	52	33	233

The convergence plot in Figure 19 highlights the effectiveness of DCG in case of thin structures. The presence of numerous low-order eigen-modes leads to stagnation for Jacobi-PCG whereas DCG ensures that the low-order eigen modes are smoothed effectively.

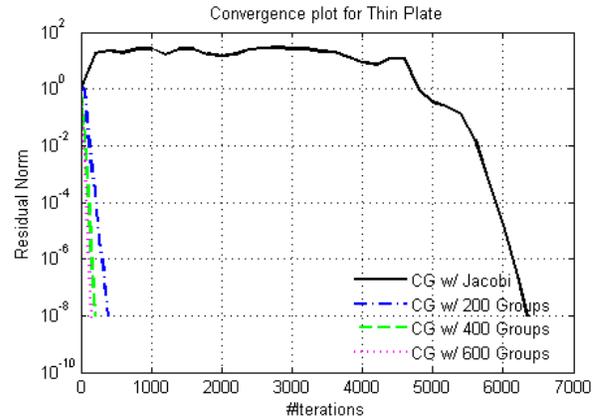


Figure 19: Convergence of DCG vs Jacobi-PCG for thin plate.

6.4.2 Kirchhoff-Love Agglomeration

Next, the above problem was solved using the thin-plate agglomeration; see Equation (3.3). Table 3 summarizes the results. Observe that, although the Kirchhoff-Love agglomeration consumes 30% more degrees of freedom per group, the net-gain is significant. In other words, for the same number of group-DOF, for thin structures, capturing the curvature leads to faster convergence. It is a better alternative for large scale problems with limited memory constraints.

Table 3: Total iterations and time taken.

G	#Iter	CPU Time (s)	GPU Time (s)	GPU Memory (MB)
0	6337	550	71	113
100	256	52	21	140
200	130	35	18	161
300	96	35	22	192
400	76	42	31	233

6.4.3 Computational Bottlenecks

Figure 20 illustrates the CUDA profile for the rigid-body deflation on the GPU for the above problem with 400 groups (the profile is similar for the Kirchhoff-Love agglomeration). Observe that 50% of the time is spent in the K_x SpMV kernel, about 20% is spent on the restriction operation

$W^T x$; the remaining 30% is spent on other tasks.

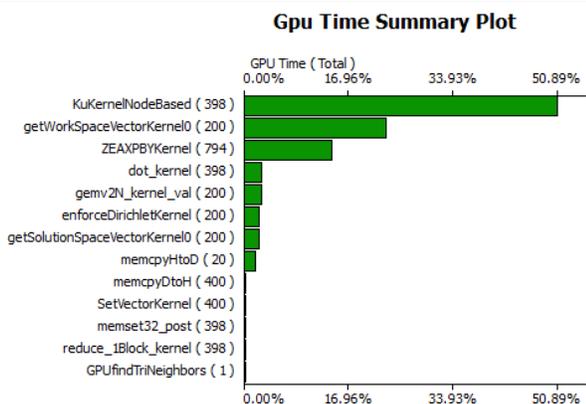


Figure 20: CUDA Profile for RBM deflation

6.5 Large-Scale FEA

Since the algorithm consumes relatively less memory, one can solve reasonably large-scale problem on a typical desktop. To illustrate consider the ‘Thomas’ engine in Figure 21 whose wheel are fixed, and a load is applied as shown.

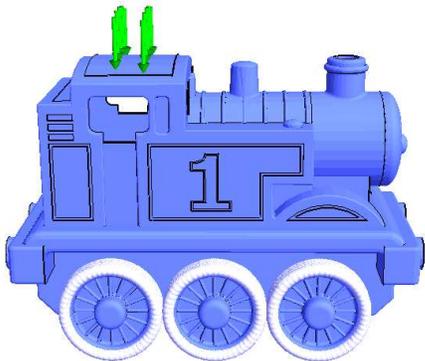


Figure 21: Structural problem over a Thomas engine.

Since the finite element analysis relies on a robust voxelization scheme, the detailed features of the model need not be suppressed. Here, the model was voxelized using 20 million elements, resulting in a 50 million DOF system.

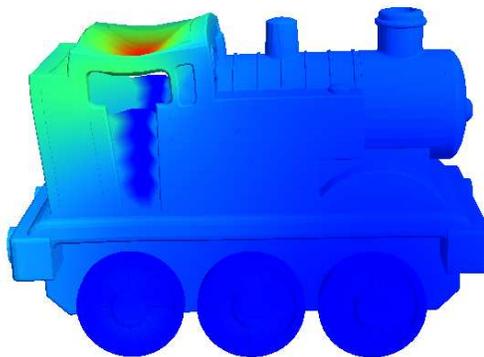


Figure 22: Deflection from a 50 million DOF system.

For this experiment, we used the GTX Titan GPU card with 6GB of memory. The linear system was solved on this GPU using rigid-body agglomeration with 900 groups in 24 minutes, consuming less than 3 GB of memory.

7. CONCLUSION

The main contribution of the paper is an assembly-free deflated conjugate gradient method for solid mechanics. In addition, the concept of ‘‘curvature-sensitive’’ agglomeration was proposed for efficient handling of thin structures. This paper serves as a foundation for future work on: (1) non-linear deformation, (2) composite modeling, and (3) topology optimization [30]. In the current implementation, the reduced matrix computation in step-3 of the DCG algorithm is performed on the CPU. This can take a significant amount of time for large number of groups. Future work will focus on improving the efficiency of this step.

Acknowledgements

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

References

- [1] G. H. Golub, *Matrix Computations*. Baltimore: Johns Hopkins, 1996.
- [2] 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.
- [3] ANSYS 13. ANSYS; www.ansys.com, 2012.
- [4] Y. Efendiev and T. Y. Hou, *Multiscale Finite Element Methods: Theory and Applications*, vol. Vol. 4. New York, NY: Springer, 2009.
- [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] 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.
- [7] Y. Saad, *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.
- [8] S. Williams, L. Oliker, and et. al., “Optimization of sparse matrix-vector multiplication on emerging multicore platforms,” presented at the Proc. 2007 ACM/IEEE Conference on Supercomputing, Reno, Nevada, 2007.
- [9] N. Bell, “Efficient Sparse Matrix-Vector Multiplication on CUDA,” 2008.
- [10] X. Yang, S. Parthasarathy, and P. Sadayappan, “Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining,” presented at the The 37th International Conference on Very Large Data Bases, Seattle, Washington, 2011.
- [11] 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.
- [12] 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.
- [13] M. Benzi and M. Tuma, “A Robust Incomplete Factorization Preconditioner for Positive Definite Matrices,” *Numerical Linear Algebra With Applications*, vol. 10, pp. 385–400, 2003.
- [14] M. Benzi, “Preconditioning Techniques for Large Linear Systems: A Survey,” *Journal of Computational Physics*, vol. 182, pp. 418–477, 2002.
- [15] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*. SIAM, 2000.
- [16] P. Wesseling, “Geometric multigrid with applications to computational fluid dynamics,” *Journal of Computational and Applied Mathematics*, vol. 128, pp. 311–334, 2001.
- [17] M. Griebel, D. Oeltz, and M. A. Schweitzer, “An Algebraic Multigrid Method for Linear Elasticity,” *SIAM J. Sci. Comput*, vol. 25, no. 2, pp. 385–407, 2003.
- [18] E. Karer and J. K. Kraus, “Algebraic multigrid for finite element elasticity equations: Determination of nodal dependence via edge-matrices and two-level convergence,” *International Journal for Numerical Methods in Engineering*, 2010.
- [19] J. Ruge and A. Brandt, “A multigrid approach for elasticity problems on ‘thin’ domains,” in *Multigrid methods: theory, applications, and supercomputing*, vol. 110, S. F. McCormick, Ed. New York: Marcel Dekker Inc, 1988, pp. 541–555.
- [20] T. B. Jonsthovel, M. B. van Gijzen, and et. al., “Comparison of the deflated preconditioned conjugate gradient method and algebraic multigrid for composite materials,” *Computational*

- Mechanics*, vol. 50, no. 3, pp. 321–333, 2012.
- [21] V. Mishra and K. Suresh, “Efficient Analysis of 3-D Plates via Algebraic Reduction,” in *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2009)*, San Diego, CA, 2009, vol. 2, pp. 75–82.
- [22] V. Mishra and K. Suresh, “A Dual-Representation Strategy for the Virtual Assembly of Thin Deformable Objects,” *Virtual Reality*, vol. 16, no. 1, pp. 3–14, 2012.
- [23] 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.
- [24] A. H. Baker, T. V. Kolev, and U. M. Yank, “Improving algebraic multigrid interpolation operators for linear elasticity problems,” *Numerical Linear Algebra With Applications*, vol. 17, pp. 495–517, 2010.
- [25] S. Timoshenko and S. W. Krieger, *Theory of Plates and Shells*. New York: McGraw-Hill Book Company, 1959.
- [26] 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, pp. 241–254, 1983.
- [27] J. Michopoulos, J. C. Hermanson, A. P. Iliopoulos, S. G. Lambrakos, and T. Furukawa, “Data-Driven Design Optimization for Composite Material Characterization,” *J. Comput. Inf. Sci. Eng*, vol. 11, no. 2, 2011.
- [28] NVIDIA Corporation, *NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide*. Santa Clara., 2008.
- [29] “OpenMP.org,” 04-May-2014. [Online]. Available: <http://openmp.org/wp/>.
- [30] K. Suresh, “Efficient Generation of Large-Scale Pareto-Optimal Topologies,” *Structural and Multidisciplinary Optimization*, vol. 47, no. 1, pp. 49–61, 2013.