A Parallel Multigrid Preconditioner for the Simulation of Large Fracture Networks

Rahul S. Sampath, Pallab Barai and Phani K.V.V. Nukala

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6164, USA

Abstract. Computational modeling of fracture in disordered materials using discrete lattice models requires the solution of a linear system of equations every time a new lattice bond is broken. Solving these linear systems of equations successively is the most expensive part of fracture simulations using large three-dimensional networks. In this paper, we present a parallel multigrid preconditioned conjugate gradient algorithm to solve these linear systems. Numerical experiments demonstate that this algorithm performs significantly better than the algorithms previously used to solve this problem.

1. Introduction

Progressive damage evolution leading to failure of disordered quasi-brittle materials has been studied extensively using various types of discrete lattice models [1, 2]. In these discrete lattice models, damage is accumulated progressively by breaking one bond at a time until the lattice system falls apart. Algebraically, the process of simulating fracture using these models is equivalent to solving a new set of linear equations

$$\mathbf{A}_n \mathbf{v}_n = \mathbf{f}_n, \qquad n = 0, 1, 2, \ldots \tag{1}$$

every time a new lattice bond is broken. In this paper, we restrict our attention to the cubic lattice system with random fuse model [3]. For this system, $\mathbf{A}_n$ represents the lattice conductance matrix, $\mathbf{f}_n$ is the applied nodal current and $\mathbf{v}_n$ is the nodal potential. Equation 1 is solved successively until the matrix $\mathbf{A}_n$ becomes singular (this is when the lattice separates into two pieces). Solving these equations is the most expensive part of a fracture simulation.

Large scale numerical simulation of these discrete lattice networks has been limited due to several reasons: (1) The number of broken bonds at failure, n_f, increases with system size‡ L as $n_f \sim O(L^{1.8})$ in 2D and $n_f \sim O(L^{2.7})$ in 3D. So, the number of times Eq. 1 has to be solved increases with increasing lattice system size, L. (2) Although direct solvers combined with low-rank updating schemes [4] are efficient for simulating fracture networks in 2D, they are not suitable for simulating large 3D networks [5]. Moreover, parallelization is essential to simulate large 3D networks and iterative solvers

‡ A cubic lattice system of size L has L bonds in each direction.

are known to have better parallel scalability compared to direct solvers. Consequently, parallel iterative solvers are in common use for simulating large 3D networks. (3) The convergence rate of the standard iterative solvers deteriorates with increasing condition number, and for the conductivity matrices encountered in fuse lattice problems, the condition number increases with system size L. In addition, the condition number increases with the number of broken bonds for a fixed system size L [5, 6]. This last property causes the *critical slow down* associated with the iterative solvers close to the macroscopic fracture. That is, as the lattice system gets closer to macroscopic fracture, the number of iterations required to attain a fixed accuracy increases significantly. This becomes particularly significant for large lattices. (4) Although the performance of iterative solvers can be significantly improved with the use of good preconditioners, to our knowledge, the best preconditioner used until now to simulate large 3D fracture networks in parallel has been the block-Jacobi preconditioner in which the incomplete LU factorization (ILU(0)) [7] is applied on each block [5]. Using this preconditioner with the conjugate gradient (CG) algorithm, simulations of cubic lattice systems of sizes up to $L = 128$ were done in Ref. [5]. However, since the blocks are not shared by multiple CPUs, the number of blocks increases as the number of CPUs increases and this leads to a deterioration in the performance of the preconditioner [7]. Also, to use this preconditioner we must recompute the ILU factors each time a bond is broken, which can be expensive. It is possible to update the ILU factors less frequently but this will result in a deterioration in the convergence rate.

In this paper, we first develop a preconditioner for the linear system in Eq. 1 based on the multigrid algorithm [8, 9] and then investigate its effectiveness in simulating very large fracture networks. Multigrid algorithms were originally developed for solving certain types of elliptic partial differential equations such as the Poisson equation, but they have been extended and applied to many other problems as well. A distinguishing feature of multigrid algorithms is that their convergence rate does not deteriorate with increasing problem size. Some multigrid implementations even obtain optimal complexity by combining this feature with an operation count that is linear in the number of unknowns. In addition, they possess very good scalability and thus can be used to solve very large systems using several thousands of processors [10, 11, 12, 13, 14, 15, 16, 17].

The motivation to use a multigrid preconditioner for the fuse problem comes directly from the structure of the matrix $\mathbf{A}_n$ in Eq. 1. The matrix $\mathbf{A}_n$ is formed by assembling the contributions from each of non-broken bonds using a standard finite element assembly technique [18]. Specifically, the element contribution $\mathbf{A}_n^e$ to the assembled matrix $\mathbf{A}_n$ is given by

$$\mathbf{A}_n^e = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2}$$

and the assembled $\mathbf{A}_n$ is given by

$$\mathbf{A}_n = \sum_e \mathbf{N}_e \epsilon_e \mathbf{A}_n^e \mathbf{N}_e^T \tag{3}$$

where, the summation is carried over all the bonds (broken as well as non-broken) in the cubic lattice, $\mathbf{N}_e$ is the matrix of ones and zeros that maps the vertices of the bond e to the corresponding degrees of freedom in the vector $\mathbf{v}_n$ and ϵ_e is one if bond e is non-broken and zero if it is broken. Note the similarity between the matrix $\mathbf{A}_0^h = \frac{1}{h^2}\mathbf{A}_0$, where $h = \frac{1}{L}$, and the matrix corresponding to a finite difference discretization (with $(L+1)$ nodes in each direction) of the 3D Laplacian operator on the unit cube using the standard 7-point stencil; these two matrices only differ in the entries corresponding to the degrees of freedom on the boundaries of the lattice. Similarly, we can view the matrix $\mathbf{A}_n^h = \frac{1}{h^2}\mathbf{A}_n$ as a perturbation of the matrix corresponding to the Laplacian operator. Since the multigrid algorithm has been effective in solving systems arising from the 3D Laplacian operator, we anticipate that it will be efficient for analysing the fracture of disordered materials using 3D lattice networks too.

The paper is organized as follows. Section 2 describes a multigrid preconditioner, which is used with the CG algorithm to solve the linear systems

$$\frac{1}{h^2}\mathbf{A}_n\mathbf{v}_n = \frac{1}{h^2}\mathbf{f}_n, \qquad n = 0, 1, 2, \ldots \tag{4}$$

Note that Eq. 1 and Eq. 4 have the same solution. In Section 3, we compare the performance of the multigrid and block-Jacobi preconditioners for solving these linear systems. Finally, we present the conclusions in Section 4.

2. Multigrid preconditioner

The multigrid algorithm is an iterative method to solve for $\mathbf{v}^h$ in $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$ where, $\mathbf{A}^h = \frac{1}{h^2}\mathbf{A}_n$ and $\mathbf{f}^h = \frac{1}{h^2}\mathbf{f}_n$ for our application. In the above setting, we denoted $\mathbf{A}_n^h$ and $\mathbf{f}_n^h$ by $\mathbf{A}^h$ and $\mathbf{f}^h$ respectively without any loss of generality. The multigrid algorithm uses a sequence of grids, $\{\Omega^h, \Omega^{2h}, \Omega^{4h}, \ldots\}$, where $h = \frac{1}{L}$ and Ω^{kh} denotes a cubic lattice with $\frac{L}{k}$ bonds in each direction for $k = 1, 2, 4, \ldots$. One iteration of the multigrid algorithm involves (1) the computation of an approximate solution to the problem using a few iterations of an iterative relaxation scheme such as the damped Jacobi method and (2) the improvement of this solution using approximate solutions to coarser problems defined on the coarser grids, $\{\Omega^{2h}, \Omega^{4h}, \ldots\}$. The first step is called *smoothing* and the second step is called *coarse grid correction.*

The multigrid algorithm is based on the following observations: First, the iterative relaxation schemes like the damped Jacobi method tend to eliminate the oscillatory (high frequency) modes of the error quickly but are not very effective at removing the smooth components of the error; Second, the smooth components of the error can be made to appear oscillatory when projected onto a coarser grid. The multigrid algorithm tries to remove the high frequency components of the error using smoothing and the remaining components of the error using coarse grid correction.

Consequently, to implement a multigrid method we need (1) a *smoother*, which is an iterative relaxation scheme, (2) a *restriction* operator ($\mathbf{R}$), which projects a fine grid vector onto the immediately coarser grid, (3) a *prolongation* operator ($\mathbf{P}$), which interpolates a coarse grid vector onto the immediately finer grid, and (4) an appropriate coarse grid problem.

In our implementation, we use the damped Jacobi method as the smoother, standard linear interpolation as the prolongation operator and *full weighting* as the restriction operator. The full weighting operator is just the transpose of the standard linear interpolation operator multiplied by a constant.

We construct the coarse grid operator ($\mathbf{A}^{2h}$) in a manner similar to the construction of the fine grid operator ($\mathbf{A}^{h}$) with the difference being that we use Ω^{2h} instead of Ω^{h} and we use a coarsened version of ϵ_e. We coarsen ϵ_e by setting the value for each coarse grid bond to be equal to the average of the values for the underlying fine grid bonds. Note that while ϵ_e is either one or zero on the finest grid, it can have fractional values on the coarser grids.

While the coarse grid operator is equivalent to a re-discretization of the fine grid operator, the same is not true for the right hand side of the coarse grid problem. Instead, the right hand side of the coarse grid problem is the result of restricting the fine grid residual vector. Thus, the coarse grid problem in the multigrid algorithm is equivalent to a re-discretization of the residual equation§ rather than a re-discretization of the original equation. The coarse grid problem using the residual equation will compute the coarse representation of the error in the current fine grid approximation to the solution and the coarse grid problem using the original equation will compute the coarse representation of the solution. While the latter can only be used to generate a good initial guess for the fine grid iteration, the former can be used to improve the fine grid approximation in each iteration.

```
Algorithm 2.1: MG-V($\mathbf{v}^h, \mathbf{f}^h$)

  comment: A multigrid V-cycle.
  if $\Omega^h$ is the coarsest grid
    then Solve for $\mathbf{v}^h$ in $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$ using a direct solver
    else
      Perform $\mu$ smoothing iterations on $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$
      Restrict the residual: $\mathbf{f}^{2h} \leftarrow \mathbf{R}^h(\mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h)$
      Set the initial guess for $\Omega^{2h}$: $\mathbf{v}^{2h} \leftarrow \mathbf{0}$
      Recurse using $\Omega^{2h}$: $\mathbf{v}^{2h} \leftarrow$ MG-V($\mathbf{v}^{2h}$, $\mathbf{f}^{2h}$)
      Prolong $\mathbf{v}^{2h}$ and correct $\mathbf{v}^h$: $\mathbf{v}^h \leftarrow \mathbf{v}^h + \mathbf{P}^h\mathbf{v}^{2h}$
      Perform $\mu$ smoothing iterations on $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$
  return ($\mathbf{v}^h$)
```

Due to the similarity between the coarse grid and fine grid operators, we use a multigrid algorithm to solve the coarse grid problem and this leads to a recursive

§ For any linear equation $\mathbf{Ax} = \mathbf{b}$, the corresponding residual equation is defined as $\mathbf{Ae} = (\mathbf{b} - \mathbf{Au})$, where u is any arbitrary vector. Solving the residual equation is equivalent to solving the original equation because $\mathbf{x} = \mathbf{u} + \mathbf{e}$.

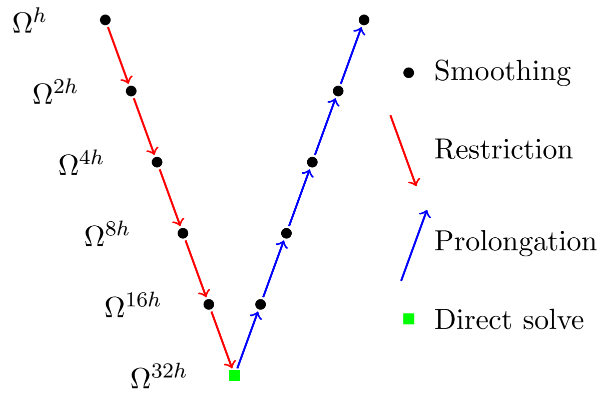

Figure 1. Illustration of a multigrid V-cycle using 6 grids.

algorithm. Each multigrid iteration visits all the grids in order from the finest to the coarsest grid and then works its way back to the finest grid. This iteration is known as a *V-cycle* and is illustrated in Fig. 1. On each grid (except the coarsest), a few smoothing iterations are performed and the residual at the end of these iterations is restricted to the immediate coarser grid; this restricted residual forms the right hand side for the problem on that grid. This step is repeated until the coarsest grid is reached. At the coarsest grid, a direct solver is used to solve the problem and the solution is prolonged and added to the solution vector on the immediate finer grid. A few more smoothing iterations are performed on this finer grid and the solution at the end of these iterations is prolonged and added to the solution vector on the next finer grid. This last step is repeated until the finest grid is reached. Algorithm 2.1 gives the pseudocode for this algorithm. We use one V-cycle (with $\mu = 2$) as a preconditioner to the CG algorithm. We implemented this preconditioner (Algorithm 2.1) using the PETSc [19, 20] and MPI libraries.

3. Results

In this Section, we compare the performance of the multigrid and block-Jacobi preconditioners for solving the linear systems given in Eq. 1 for different lattice system sizes, L, and different number of CPUs. In our experiments, we used the implementation of the block-Jacobi preconditioner found in the PETSc library. In the block-Jacobi preconditioner, ILU(0) was applied on each block. All the experiments were performed on the *Jaguar* supercomputer at Oak Ridge National Laboratory (ORNL). The architectural details for this supercomputer can be found in [21].

We report the sequential run-time to simulate the failure of a cubic lattice system using the multigrid and block-Jacobi preconditioners for different lattice system sizes in Table 1. The block-Jacobi algorithm is faster for small systems and the multigrid algorithm is faster for larger systems. We also report the parallel run-time to simulate the failure of a cubic lattice system of size $L = 64$ using the multigrid and block-Jacobi

Table 1. Time to simulate the failure of a cubic lattice system of size L using the multigrid and block-Jacobi preconditioners. The number of broken bonds at failure, n_f, is also reported. This experiment was performed on a single CPU. All timings are reported in seconds.

L	n_f	Multigrid	Block-Jacobi
10	672	8.302	3.8278
16	2565	121.53	76.51
24	7151	1879.2	1220.3
32	17094	5202.2	8769.6
64	119343	256921.9	806457.6

Table 2. Time to simulate the failure of a cubic lattice system of size $L = 64$ using the multigrid and block-Jacobi preconditioners. There were 119342 broken bonds at failure. All timings are reported in seconds.

CPUs	Multigrid	Block-Jacobi
16	37863.4	101258.9
64	25855	30025.3

preconditioners on 16 and 64 CPUs in Table 2. It is clear that multigrid is significantly faster than the block-Jacobi for the 16 CPU case. For the 64 CPU case, multigrid is still faster than block-Jacobi but the difference is not as impressive; however, even for this case multigrid is about 50% faster than block-Jacobi for about 80% of the simulation.

In Fig. 2, we report the number of CG iterations required to break each bond of a cubic lattice system of size $L = 64$ until failure. It is clear that multigrid takes significantly fewer CG iterations than the block-Jacobi algorithm. Moreover, the number of CG iterations required for solving each linear system is insensitive to the number of CPUs used for a multigrid preconditioner. On the other hand, the number of CG iterations typically increases with the number of CPUs used for a block-Jacobi preconditioner. This behavior can be attributed to the fact that the number of blocks for the block-Jacobi preconditioner increases with an increase in the number of CPUs.

Due to the restrictions of the batch system for running jobs on Jaguar, we could not run the entire simulation in one go. Instead, we had to use a restarting mechanism to complete the entire simulation. In particular, for the simulation in Fig. 2 multigrid was restarted after 80,000 bonds were broken and block-Jacobi was restarted once after 50,000 bonds were broken and once after 90,000 bonds were broken. The first CG solve after a restart used an initial guess of zero. In all other cases, we used the solution to the previous linear system as the initial guess. This is the cause for the spikes seen in Fig. 2 corresponding to 0 and 80,000 broken bonds for multigrid and 0, 50,000 and 90,000 broken bonds for block-Jacobi.

The number of broken bonds at the peak load was around 105,000 for the lattice system used in Fig. 2. From peak load until failure, the conductance matrix for the lattice becomes increasingly singular. This is the likely cause for the increase in the

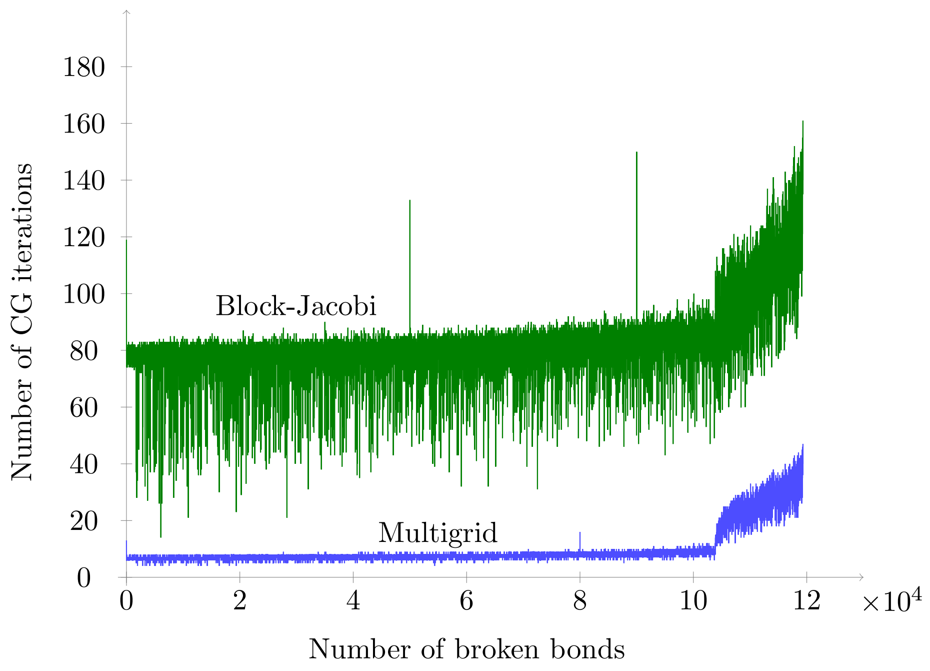

Figure 2. The number of CG iterations required to solve the linear systems (Eq. 1) using the multigrid and block-Jacobi preconditioners. This experiment was performed on a cubic lattice system of size $L = 64$ using 64 CPUs.

number of CG iterations near the end of the simulation as shown in Fig. 2.

In Fig. 3(a) - 3(b), we report the run-time to break a fixed number of bonds in the initial, middle and final stages of simulation for cubic lattice systems of different sizes using the multigrid and block-Jacobi preconditioners on different number of CPUs. This demonstrates the fixed-size or strong scalability of the two algorithms. For both the algorithms, there is a cost that grows with the number of CPUs. For the multigrid algorithm this is due to the communication required for restriction, prolongation and smoothing and for the block-Jacobi algorithm this is due to the fact that the number of CG iterations typically increases with an increase in the number of CPUs.

4. Conclusions

We have developed a parallel multigrid preconditioned conjugate gradient algorithm to solve the linear system of equations that arise in fracture simulations using discrete lattice models. The best known parallel algorithm to solve these linear systems for large 3D fracture networks is the conjugate gradient algorithm with a block-Jacobi preconditioner in which ILU(0) is applied on each block. We compared the performance of the multigrid and block-Jacobi preconditioners for simulating the fracture of cubic lattice systems of different sizes using different number of CPUs. Overall, we found the multigrid preconditioner to be significantly faster than the block-Jacobi preconditioner for cubic lattices of sizes greater than or equal to 32. The extension of this multigrid

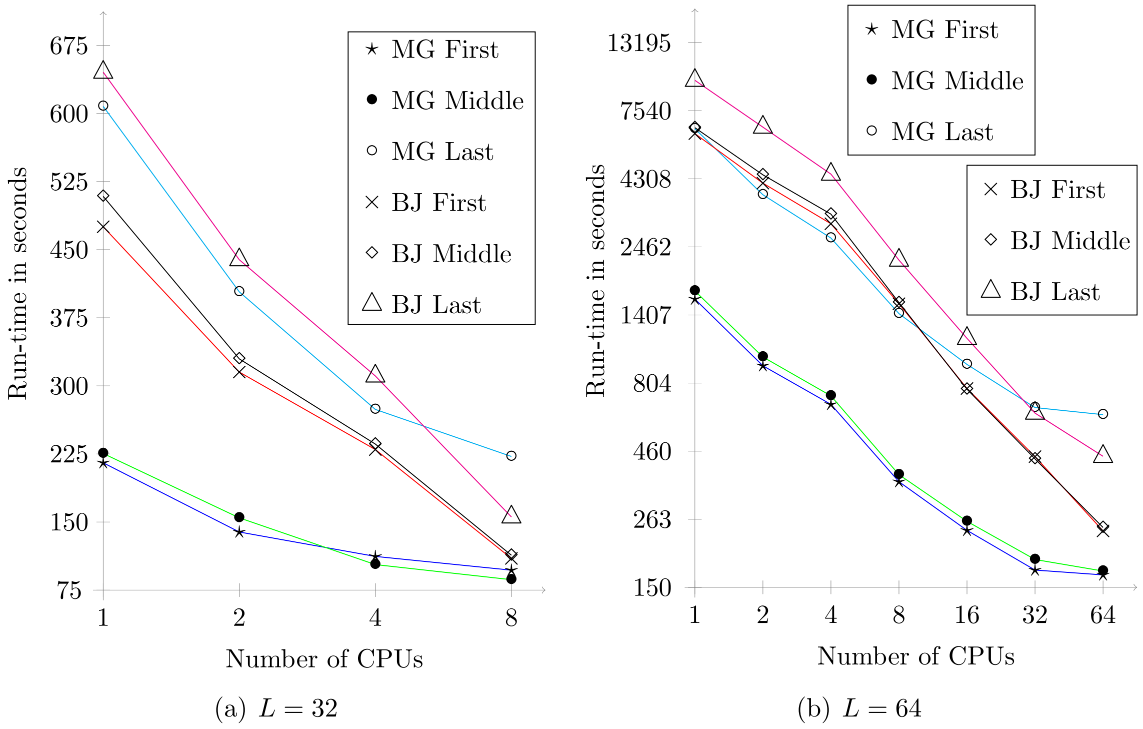

Figure 3. Fixed-size scalability for the multigrid and block-Jacobi algorithms for lattice systems of sizes (a) $L = 32$ and (b) $L = 64$. We report the run-time (using different number of CPUs) to break 1000 bonds in the initial, middle and final stages of simulation using the multigrid and block-Jacobi preconditioners. MG refers to multigrid, BJ refers to block-Jacobi and first, middle and last refer to the initial, middle and final stages of simulation respectively.

framework to fracture of elastic and perfectly plastic 3D spring and beam lattice systems remains to be investigated.

5. Acknowledgment

R.S. Sampath and P. Barai are supported by Oak Ridge National Laboratory's (ORNL) Postdoctoral Research Associates and Postmaster's Research Participation Programs under contract numbers DE-AC05-00OR22725 and DE-AC05-00OR22750, respectively. P.K.V.V. Nukala is sponsored by the Office of Advanced Scientific Computing Research, U.S. Department of Energy under contract number DE-AC05-00OR22725 with UT-Battelle, LLC.

[1] Hansen A and Roux S 2000 *Statistical toolbox for damage and fracture* (Springer, New York) pp 17–101

[2] Alava M J, Nukala P K V V and Zapperi S 2006 *Advances in Physics* **55(3)** 349–476

[3] de Arcangelis L, Redner S and Herrmann H J 1985 *Journal of Physics (Paris) Letters* **46(13)** 585–590

[4] Nukala P K V V and Simunovic S 2003 *J. Phys. A: Math. Gen.* **36** 11403–11412

[5] Nukala P K V V, Simunovic S, Zapperi S and Alava M 2007 *Journal of computer-aided material design* **14** 25–35

[6] Nukala P K V V and Simunovic S 2004 *J. Phys. A: Math. Gen.* **37** 2093–2103

[7] Saad Y 2003 *Iterative Methods for Sparse Linear Systems* 2nd ed (Philadelphia, PA: Society for Industrial and Applied Mathematics)

[8] Briggs W L, Henson V E and McCormick S F 2000 *A multigrid tutorial (2nd ed.)* (Philadelphia, PA, USA: Society for Industrial and Applied Mathematics)

[9] Trottenberg, U and Oosterlee, C W and Schuller, A 2001 *Multigrid* (San Diego, CA: Academic Press Inc.)

[10] Adams M F, Bayraktar H H, Keaveny T M and Papadopoulos P 2004 Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing* (ACM/IEEE)

[11] Adams M and Demmel J W 1999 Parallel multigrid solver for 3d unstructured finite element problems *SC '99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing* (New York, NY, USA: ACM Press)

[12] Henson V E and Yang U M 2002 *Applied Numerical Mathematics* **41** 155 – 177

[13] Sampath R S, Adavani S S, Sundar H, Lashuk I and Biros G 2008 Dendro: parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing* (Piscataway, NJ, USA: IEEE Press) pp 1 – 12

[14] Tatebe O and Oyanagi Y 1994 Efficient implementation of the multigrid preconditioned Conjugate Gradient method on distributed memory machines *Supercomputing '94: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing* (New York, NY, USA: ACM) pp 194 – 203

[15] Gradl T and Rude U 2008 High performance multigrid in current large scale parallel computers *Proceedings of the 9th Workshop on Parallel Systems and Algorithms (PASA)* GI Edition: Lecture Notes in Informatics (Dresden, Germany)

[16] Bergen B, Gradl T, Hulsemann F and Rude U 2006 *Computing in Science and Engineering* **8** 56 – 62

[17] Bergen B, Hulsemann F and Rude U 2005 Is 1.7×10^{10} Unknowns the Largest Finite Element System that Can Be Solved Today? *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing* (Washington, DC, USA: IEEE Computer Society)

[18] Hughes T J R 1987 *The Finite Element Method* (Prentice-Hall)

[19] Balay S, Buschelman K, Gropp W D, Kaushik D, Knepley M G, McInnes L C, Smith B F and Zhang H 2001 PETSc home page http://www.mcs.anl.gov/petsc

[20] Balay S, Buschelman K, Eijkhout V, Gropp W D, Kaushik D, Knepley M G, McInnes L C, Smith B F and Zhang H 2004 PETSc users manual Tech. Rep. ANL-95/11 - Revision 2.1.5 Argonne National Laboratory

[21] NCCS Jaguar's system architecture http://www.nccs.gov/computing-resources/jaguar