


A scalable multi-level preconditioner for matrix-free μ -finite element analysis of human bone structures

Report**Author(s):**

Arbenz, Peter; van Lenthe, G. Harry; Mennel, Uche; [Müller, Ralph](#) ; Sala, Marzio

Publication date:

2011

Permanent link:

<https://doi.org/10.3929/ethz-a-006784136>

Rights / license:

[In Copyright - Non-Commercial Use Permitted](#)

Originally published in:

Technical Report / ETH Zurich, Department of Computer Science 543

A SCALABLE MULTI-LEVEL PRECONDITIONER FOR MATRIX-FREE μ -FINITE ELEMENT ANALYSIS OF HUMAN BONE STRUCTURES

PETER ARBENZ*, G. HARRY VAN LENTHE[†], UCHE MENNEL*, RALPH MÜLLER[†],
AND MARZIO SALA*

ABSTRACT. The recent advances in microarchitectural bone imaging are disclosing the possibility to assess both the apparent density and the trabecular microstructure of intact bones in a single measurement. Coupling these imaging possibilities with microstructural finite element (μ FE) analysis offers a powerful tool to improve bone stiffness and strength assessment for individual fracture risk prediction.

Many elements are needed to accurately represent the intricate microarchitectural structure of bone; hence, the resulting μ FE models possess a very large number of degrees of freedom. In order to be solved quickly and reliably on state-of-the-art parallel computers, the μ FE analyses require advanced solution techniques. In this paper, we investigate the solution of the resulting systems of linear equations by the conjugate gradient algorithm, preconditioned by aggregation-based multigrid methods. We introduce a variant of the preconditioner that does not need assembling the system matrix but uses element-by-element techniques to build the multilevel hierarchy. The preconditioner exploits the voxel approach that is common in bone structure analysis, it has modest memory requirements, while being at the same time robust and scalable.

Using the proposed methods, we have solved in less than 10 minutes a model of trabecular bone composed of 247'734'272 elements, leading to a matrix with 1'178'736'360 rows, using only 1024 CRAY XT3 processors. The ability to solve, for the first time, large biomedical problems with over 1 billion degrees of freedom on a routine basis will help us improve our understanding of the influence of densitometric, morphological and loading factors in the etiology of osteoporotic fractures such as commonly experienced at the hip, the spine and the wrist.

Keywords: *Algebraic Multigrid, Aggregation Methods, Matrix-Free Preconditioning, Micro Finite Element Analysis.*

1. INTRODUCTION

Osteoporosis is a disease characterized by low bone mass and deterioration of bone microarchitecture. It leads to increased bone fragility and risk of fracture, particularly of the hip, spine, and wrist. Worldwide, lifetime risk for osteoporotic fractures in women is estimated close to 40%; in men risk is 13% [32]. As reported by the World Health Organization, osteoporosis is second only to cardiovascular disease as a leading health care problem. Osteoporotic fractures are a major cause of severe long-term pain and physical disability, and have an enormous impact on the individual, society and health care systems. For the clinician, predicting fracture risk for individual patients is mainly restricted to the quantitative analysis of bone density. Several studies have shown that bone strength, an indicator for bone fracture risk, is only predicted moderately by bone density. This is not surprising: bones are not solid structures, but are made up of an outer shell of compact bone, enclosing a core of trabecular bone. This trabecular bone, which is located at the end of long bones (e.g., radius; Fig. 1.A) and in cuboidal bones (e.g., spine; Fig. 1.B), is a porous structure that contributes significantly to the load-bearing capacity of the

* ETH Zurich, Institute of Computational Science, 8092 Zurich, Switzerland.

[†] ETH Zurich, Institute for Biomechanics, 8092 Zurich, Switzerland.

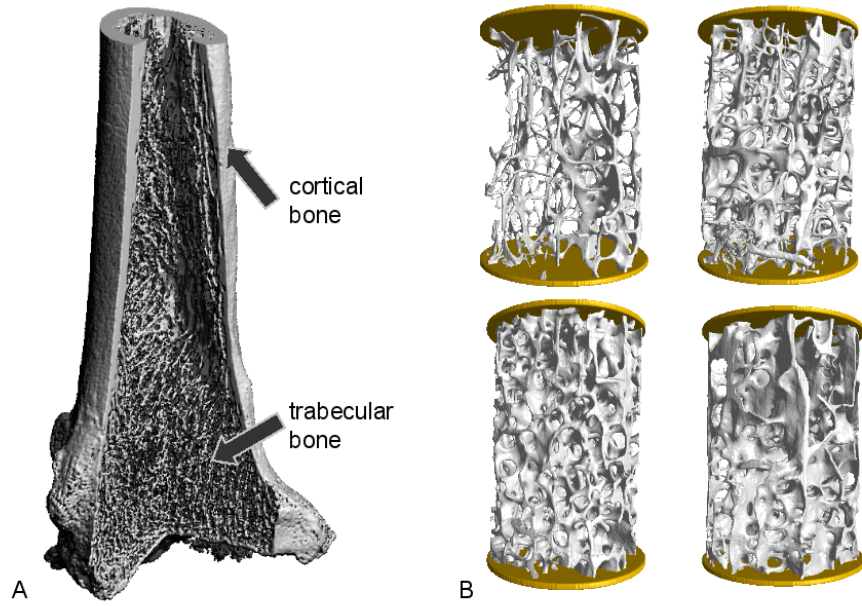


FIGURE 1. A. Distal part of a human radius, showing cortical and trabecular bone as imaged using micro-CT scanning; part of the bone was artificially removed to be able to look inside the bone;
 B: four trabecular bone specimens (10 mm height, 8 mm diameter) taken from human vertebrae. Note the large variance between samples, in the amount of bone, but also in bone microarchitecture. (Fig. 1.B courtesy of Dr. Martin Stauber, ETH Zurich, Switzerland)

human skeleton. It is not a random structure, but its trabeculae typically run in the main loading direction. Mechanical testing has shown huge heterogeneity in bone mechanical properties, not only across sites and specimens, but even within the same bone these properties can differ 50-fold. Testing also showed that bone is not equally strong in all directions. This mechanical anisotropy is expressed as the ratio of the stiffness in the strongest direction to the stiffness in the weakest direction; it can range from basically 1 (no preferential orientation) to over 10 for both stiffness and strength [26].

Investigation of the mechanical properties of trabecular bone presents a challenge due to its high porosity and complex architecture, both of which, as mentioned before, vary substantially between anatomic sites and across individuals. A promising technique that takes bone microarchitecture into account is microstructural finite element (μ FE) analysis [30, 37]. Detailed μ FE models have typically been obtained through high-resolution micro-computed tomography (μ CT) imaging of trabecular bone specimens. Typically, these μ FE models represent trabecular regions in the order of 5 to 10mm cubes, a scale at which the bone behaves as a continuum. By taking the fine trabecular network into account, these models have the advantage that the anisotropic properties of trabecular bone are automatically accounted for, even when at the material level the properties are isotropic. For linear deformation conditions, comparison between biomechanical compression tests and μ FE show very good agreement when a single homogeneous, isotropic tissue modulus is applied [25, 28]. This holds true for normal as well as osteoporotic bone [23].

Ideally, the development of *in vivo* imaging systems with microstructural resolution better than $50\ \mu\text{m}$ would allow measurement of patients at different time points and at different anatomical sites. Unfortunately, such systems are not yet available, but the resolution at peripheral sites has reached a level ($82\ \mu\text{m}$) that allows elucidation of individual microstructural bone elements. Using this technique, two recent cross-sectional studies have shown sex- and age-related changes in trabecular structure and in cortical thickness of the distal radius [10, 27]. The acquired structural data was able to differentiate between osteopenic women with and without history of fracture, whereas measurements of bone mineral density (BMD) did not [10].

Many finite elements are needed to accurately represent an intact human bone with its intricate microarchitecture; hence, the resulting μ FE models possess a very large number of degrees of freedom. In order to be solved quickly and reliably on state-of-the-art parallel computers, the FE analyses require advanced solution techniques. In this paper, we adopt a formulation based on the strain-displacement formulation of linear elasticity [46, 54]. We assume the materials involved to be isotropic and homogeneous. The discrete formulation is based on a standard finite element (voxel) discretization, the displacements in the three axis directions each being represented by piecewise trilinear polynomials. The finite element discretization leads to a linear system of equations

$$(1) \quad K\mathbf{u} = \mathbf{f},$$

where K is a sparse symmetric positive definite matrix. The number of nonzeros of K can be written as $\text{nnz}(K) = \nu n$, where n is the order of K and ν is the average number of nonzeros per row; with piecewise trilinear polynomials in a 3D rectangular grid, $\nu \leq 81$.

Since the size of K rules out direct solvers, the method of choice for solving (1) is the conjugate gradient (CG) algorithm [7, 40]. CG solvers only require that the action of K on a given vector can be computed. So, two solution strategies arise:

- In *matrix-ready methods*, the matrix K is assembled and fully stored in memory, requiring memory space for about νn floating point numbers. Additionally, integer pointers are needed to handle the sparse matrix data structure. Exploiting the 3×3 block structure of K about $\nu n/9$ pointers suffice.
- In *matrix-free methods* the global stiffness matrix K is never assembled. In a finite element context, the representation

$$(2) \quad K = \sum_e T_e K_e T_e^T$$

can be employed. In this *element-by-element* (EBE) approach the element stiffness matrix K_e and the topology matrix T_e occur, see e.g. [5, 16, 24, 38, 46]. The T_e consist of a few columns of the identity matrix of order n representing the mapping of the local to the global node numbers. In fact, the T_e are stored in a large so-called element-to-node table.

An alternative row-by-row (RBR) technique has been proposed in [38] and has similar memory saving. However, an important advantage of the EBE method over the RBR method is that different (isotropic) Young's moduli can be specified for each element, while competitive RBR methods allow only one material for the entire computational domain. Since we are interested in cases where several materials coexist, we only focus on EBE methods.

Matrix-free methods are particularly favorable for problems arising from voxel conversions, since all elements generated in the voxel conversion have exactly the same shape, size and orientation. This entails that all element stiffness matrices related to a given material are *identical*. So, a single 24×24 matrix K_e has to be formed. (The order of K_e is 24, reflecting the three displacements at the eight vertices of the voxel element.)

The major drawback of the EBE approach is the difficulty to define efficient preconditioners. In fact, although a considerable wealth of research on preconditioning for SPD systems has been published, surprisingly little of this applies to matrix-free environments. Not having access to the fully assembled matrix prevents the usage of many algebraic preconditioners. Therefore, the emphasis has been on the use of preconditioners derived from simplifications of the problem for which explicit matrices can be readily constructed. Widely used preconditioners for matrix-free environments are Jacobi preconditioning (diagonal scaling) or EBE preconditioning [24]. The analysis of Wathen [29, 53] shows that these methods are spectrally equivalent to the unpreconditioned system and hence are not suited for large scale problems.

The situation is radically different if K is assembled. Then, powerful scalable preconditioners based on multigrid ideas can be used. In [3], the algebraic multigrid method based on aggregation has been successfully used for the solution of linear systems arising in conventional analysis of bone strength. Aggregation-based methods are also supported by theoretical results [51], and have been proven scalable up to thousands of processors [3, 4, 31] for a variety of applications. However, to the best of our knowledge, no article has yet shown the applicability of AMG preconditioners to matrix-free approaches in bone analysis. The challenge is to construct an AMG preconditioner with good convergence rate that is at the same time cheap to set up and to apply.

In this paper, we introduce a novel preconditioner that can effectively be used in matrix-free environments. This preconditioner has modest requirements in terms of CPU time, is only slightly slower than smoothed aggregation preconditioners for matrix-ready problems, is scalable up to thousands of processors and billions of unknowns, is orders of magnitude faster than the widely adopted Jacobi or element-by-element preconditioner, and reduces the memory consumption with respect to AMG preconditioners for matrix-ready problems by a factor of about 3 to 3.5, while converging in a comparable number of iterations. This memory savings is the main improvement over the approach presented by Adams [3] and later implemented by us in the Trilinos framework [4, 33]. The proposed method can also be applied in other areas in which K exists only in the form of a matrix-vector multiplication routine provided that the graph of the K is available or can be cheaply generated.

This paper is organized as follows. In §2 we describe the mathematical model and the system of linear equations it entails. In §3 we introduce multigrid preconditioners, and in §4 we focus on smoothed aggregation preconditioners. A novel scalable, multigrid preconditioner for matrix-free problems is presented in §5. Software issues are addressed in §6. Numerical results are reported in §7. Finally, conclusions are drawn in §8.

2. ITERATIVE SOLUTION OF LINEAR SYSTEMS

The conjugate gradient (CG) algorithm is the method of choice for solving the sparse symmetric positive definite problem (1). The number of CG iterations required to reach a prescribed tolerance depends on the spectral properties of the linear system matrix K , and in particular its *condition number* $\kappa(K)$, defined as the ratio of largest and smallest eigenvalue of K [7, 40]. It is well-known that for second-order elliptic boundary value problems like in linear elasticity discretized by Lagrangian finite elements, $\kappa(K) = \mathcal{O}(h^{-2})$. Here, h is the edge length of the voxel elements. Note that in 3D the number of unknowns n in (1) behaves like $n = \mathcal{O}(h^{-3})$. So, $\kappa(K) \gg 1$ if h is small as in μ FE analyses. To avoid slow convergence, (1) is preconditioned, i.e., it is replaced by an equation of the form

$$(3) \quad KB\mathbf{v} = \mathbf{f}, \quad \mathbf{v} = B^{-1}\mathbf{u},$$

where B is the *preconditioner*, chosen such that $\kappa(KB) \ll \kappa(K)$. The definition of B is the key component of any CG solver and the main focus of this paper. In general terms, an efficient preconditioner B is a transformation that

- (1) *is a good approximation of K^{-1} in some sense,*
- (2) *is cheap to construct, and*
- (3) *is cheap to apply.*

B should act so that KB is near to being the identity matrix and its eigenvalues are clustered within a sufficiently small region of the complex plane. If the three criteria are satisfied then solving the preconditioned iterative method including the building of the preconditioner outperforms that of the non-preconditioned iterative method. The literature on preconditioning is vast. We refer the reader to [40, 45] and the references therein for more details. In the sequel we will focus uniquely on multigrid preconditioners, summarized in §3, and, in particular, on aggregation based preconditioners in §4. In addition to the general terms mentioned before we will deal with issues like

- *parallel construction* of the preconditioner, to take advantage of the architecture of modern supercomputers, and
- *matrix-free representation* of the preconditioner, to save the most memory consuming matrix in the multigrid hierarchy.

For the latter we assume that the matrix graph is either given or cheaply computable.

3. MULTIGRID PRECONDITIONERS

In a multigrid method [11, 20, 48] one tries to approximate the original problem of interest on a hierarchy of levels and use ‘solutions’ from coarser levels to accelerate the convergence on the finer levels. This approach is motivated by the fact that relaxation methods, such as Jacobi or Gauss-Seidel iteration, are effective at reducing the high-frequency components of the error. In the multigrid context, these iterative solvers are called *smoothers*. Smoothers are, however, ineffective at reducing the low-frequency components of the error. Intuitively, these low-frequency components can be redefined as high-frequency components by representing them on coarser grids. That is, the approximate solution provided by a smoother can be projected to a smaller space to provide a coarser-grid correction, and this procedure is applied recursively until the coarsest level, where a direct solver can be employed.

Algorithm 3.1 Multilevel procedure with L levels

```

1: procedure MultiLevelSolve( $K_\ell, \mathbf{b}_\ell, \mathbf{x}_\ell, \ell$ )
2: if  $\ell == L-1$  then
3:   Solve  $K_\ell \mathbf{x}_\ell = \mathbf{b}_\ell$                                      {Direct solve on the coarsest level}
4: else
5:    $\mathbf{x}_\ell = S_\ell(K_\ell, \mathbf{b}_\ell, \mathbf{0})$                                      {Presmoothing}
6:    $\mathbf{r}_\ell = \mathbf{b}_\ell - K_\ell \mathbf{x}_\ell$                                      {Calculation of the residual}
7:    $\mathbf{b}_{\ell+1} = R_\ell \mathbf{r}_\ell$                                      {Restriction}
8:    $\mathbf{v}_{\ell+1} = \mathbf{0}$ 
9:   MultiLevelSolve( $K_{\ell+1}, \mathbf{b}_{\ell+1}, \mathbf{v}_{\ell+1}, \ell+1$ )
10:   $\mathbf{x}_\ell = \mathbf{x}_\ell + P_\ell \mathbf{v}_{\ell+1}$                                      {Coarse grid correction}
11:   $\mathbf{x}_\ell = S_\ell(K_\ell, \mathbf{b}_\ell, \mathbf{x}_\ell)$                                      {Postsmoothing}
12: end if
13: end procedure

```

A typical multigrid procedure (known as V-cycle) is reported in Algorithm 3.1. In this procedure, $\ell=0$ denotes the finest level (that is, $K_0 = K$), K_ℓ is the discretization of the operator on level ℓ of the problem, P_ℓ and R_ℓ are the prolongator and restriction operators from level $\ell+1$ to ℓ , respectively, and S_ℓ is the smoother for level ℓ . The corresponding preconditioner simply reads $\text{MultiLevelSolve}(K, \mathbf{b}, \mathbf{x}, 0)$.

Algorithm 3.2 Setup procedure for an abstract multigrid solver

```

1: Define the number of levels,  $L$ 
2: for level  $\ell = 0, \dots, L - 1$  do
3:   if  $\ell < L - 1$  then
4:     Define prolongator  $P_\ell$ 
5:      $R_\ell = P_\ell^T$ 
6:      $K_{\ell+1} = R_\ell K_\ell P_\ell$ 
7:     Define smoother  $S_\ell$ 
8:   else
9:     Prepare for solving with  $K_\ell$ 
10:  end if
11: end for

```

The key aspect in all multigrid methods is the definition of the auxiliary operators P_ℓ , R_ℓ , and K_ℓ . A general setup procedure for a multigrid procedure is reported in Algorithm 3.2. Note that we impose $R_\ell = P_\ell^T$ and presmoothing equal postsmoothing such that the final preconditioner is symmetric. Note also that, since $K_{\ell+1} = R_\ell K_\ell P_\ell$, the multigrid setup is completed as soon as the prolongator operator P_ℓ and the smoother S_ℓ are defined. These operators can be defined by using a user-provided hierarchy of grids as in *geometric multigrid*, or by resorting to algebraic manipulations of the linear system matrix and possibly a few cheaply available information on the problem at hand (like a near null space or near kernel), as in AMG. Theoretical and numerical studies have shown that the converge rate of AMG in several applications is equivalent or comparable to that of geometric multigrid methods.

Two AMG approaches have emerged in the recent years. In the ‘classical’ Ruge–Stüben AMG [39] a subset of the nodes of a certain level are identified as coarse-level nodes (so-called C-nodes) and finer-level nodes (F-nodes). In the AMGe variant information of the finite element mesh is taken into account in the coarsening procedure [12]. In smoothed aggregation (SA) multigrid [51, 52] nodes of a certain level are grouped into contiguous subsets, called aggregates, that form the nodes of the next coarser level. Both, classical and aggregation based approaches, are implemented in open-source high-performance libraries [19, 35, 43]. A comparison of methods can be found in [1]. Since the Ruge–Stüben AMG extracts all information from the system matrix, our matrix-free approach works only with SA multigrid, whose key aspects are reported in §4.

4. AGGREGATION-BASED ALGEBRAIC MULTIGRID PRECONDITIONERS

Algorithm 3.2 indicates that the decisive ingredients of an algebraic multigrid solver are the prolongator operator P_ℓ and the smoother S_ℓ that have to be defined on each level ℓ , starting with the finest level 0. The definition of the prolongator operator P_ℓ with aggregation procedures requires the following phases:

- (1) A graph representation of K_ℓ , denoted by \mathcal{G}_ℓ , is defined. For scalar problems, \mathcal{G}_ℓ is simply the graph of K_ℓ . For vector PDEs as the one considered in this paper, the graph is defined in a block fashion, meaning that one graph vertex is associated with all the unknowns at

a grid vertex. (i, j) is an edge of the graph if there are any nonzeros in the block matrix defined by the i^{th} block row and j^{th} block column. For highly anisotropic problems, it is often advantageous to drop weak connections [15].

- (2) The vertices of \mathcal{G}_ℓ are grouped into contiguous and disjoint subsets, called *aggregates*, that effectively represent the vertices of the coarser grid $\mathcal{G}_{\ell+1}$.

Aggregates can be defined in many ways, and their construction can significantly affect the convergence rate and the complexity of the coarser grids. In a standard algebraic multigrid method the most common way to create the aggregates is to resort to a greedy algorithm, where an initial node is chosen along with all of its nearest neighbors. The net effect of this type of procedure is to generate aggregates which are ‘ball-like’ with an approximate diameter of three graph vertices, see [50]. This means that on a 3D regular Cartesian grid, each aggregate contains $3 \times 3 \times 3 = 27$ vertices. For large problems, this approach leads to many aggregates. Therefore, the aggregation procedure must in general be repeated several times until a coarse matrix is obtained that is small enough to be solved efficiently by a direct solver. A way to reduce the number of levels is *aggressive coarsening*, which consist in grouping a larger number of graph vertices (values from 50 to 200 are suggested in [31]) in each aggregate. These larger aggregates are constructed by means of a graph partitioner like METIS or PARMETIS [34].

- (3) Once the coarser grid is determined, a grid transfer operator must be defined. The idea of aggregation is to construct first a *tentative* prolongation operator, that takes care of the near kernel of $K_{\ell+1}$. This is done by “chopping” the near kernel $\mathcal{R}(B_\ell)$ of K_ℓ in as many pieces as there are aggregates,

$$B_\ell = \begin{bmatrix} B_1^{(\ell)} \\ \vdots \\ B_{n_{\ell+1}}^{(\ell)} \end{bmatrix}.$$

The rows of $B_j^{(\ell)}$ are precisely the rows of B_ℓ corresponding to the grid points that have been assigned to the j^{th} aggregate. If $B_j^{(\ell)} = Q_j^{(\ell)} R_j^{(\ell)}$ is the QR factorization of $B_j^{(\ell)}$ then we have [13]

$$B_\ell = \tilde{P}_\ell B_{\ell+1}, \quad \tilde{P}_\ell^T \tilde{P}_\ell = I_{n_{\ell+1}},$$

with

$$\tilde{P}_\ell = \text{diag}(Q_1^{(\ell)}, \dots, Q_{n_{\ell+1}}^{(\ell)}) \quad \text{and} \quad B_{\ell+1} = \begin{bmatrix} R_1^{(\ell)} \\ \vdots \\ R_{n_{\ell+1}}^{(\ell)} \end{bmatrix}.$$

The columns of $B_{\ell+1}$ span the near kernel of $K_{\ell+1}$. Notice that the matrices K_ℓ are not used in the construction of the tentative prolongators and the near kernels B_ℓ . The chopping is done entirely based on information provided by the graphs \mathcal{G}_ℓ .

At this point, we have defined a *tentative prolongator* operator, \tilde{P}_ℓ . When we set $P_\ell = \tilde{P}_\ell$, then the method is referred to as *nonsmoothed aggregation* (NSA). Otherwise, we move to the next step.

- (4) For elliptic problems, it is advisable to perform an additional step, to obtain *smoothed aggregation* (SA). The idea is to improve the tentative prolongator \tilde{P}_ℓ to pull energy out of the basis functions. The smoothing is typically performed with one step of a damped Jacobi iteration to obtain the final prolongator,

$$(4) \quad P_\ell = (I_{n_\ell} - \omega_\ell D_\ell^{-1} K_\ell) \tilde{P}_\ell,$$

where $K_\ell \in \mathbb{R}^{n_\ell \times n_\ell}$, I_{n_ℓ} is the identity matrix of size n_ℓ , $D_\ell = \text{diag}(K_\ell)$, and ω_ℓ is a damping parameter, typically defined as

$$(5) \quad \omega_\ell = \frac{4/3}{\lambda_{\max}(D_\ell^{-1}K_\ell)},$$

where $\lambda_{\max}(D_\ell^{-1}K_\ell)$ indicates the largest eigenvalue of $D_\ell^{-1}K_\ell$. Formula (4) ensures that the support of the basis functions do not extend beyond the nearest neighbors of aggregates, thus maintaining a “reasonable” complexity of the resulting multigrid operator.

The reader is referred to [9, 51] for more details.

5. A NEW MULTIGRID PRECONDITIONER FOR MATRIX-FREE SOLVERS

In this section we present an algebraic multigrid preconditioner that can be applied to problems where the underlying matrix is available only through matrix-vector computations, and the graph of $K = K_0$ is explicitly available. Note that, for linear, bilinear or trilinear Lagrangian finite elements, the graph of K_0 coincides with the grid connectivity, and can therefore be easily and cheaply constructed from the grid data structures.

The key observation that lead us to propose this method is that K_1 is considerably smaller than K_0 ; as noted in §4, the reduction in size between two consecutive levels can vary from about 27 to about two orders of magnitude, depending on how aggressive the coursening is executed. As such, K_1 does not have more nonzeros per row than K_0 , and storing K_1 will require a small fraction of the memory used for K_0 . From the multigrid literature it is known that the memory consumption of the smoothed aggregation AMG preconditioner *and* the linear system matrix K_0 is roughly 1.4 times the storage required for K_0 alone. (This number is known as *operator complexity*.) Ideally, our matrix-free AMG approach should require only 0.4 times the storage that would be required for K_0 , as no storage is required for the matrix itself. The gain in memory space is therefore

$$\frac{1.4}{0.4} = 3.5,$$

meaning that, our matrix-free AMG can solve problems 3.5 larger than the plain multilevel preconditioner on a given number of processors. Of course, this ratio is just an estimate since it does not take into account the memory needed for additional data structures and auxiliary vectors. The numerical results reported in §7 however indicate that the memory savings are in fact about 3 to 4.

Exploiting this observation, we define an AMG preconditioner that operates on matrix-free problems by explicitly constructing K_1 in an efficient manner. Once K_1 is stored in memory, we can adopt a standard smoothed aggregation method for matrix-ready problems to generate the remaining matrices K_ℓ . This is equivalent to replacing step 6 of Algorithm 3.2 with a different triple matrix-matrix product for $\ell = 0$ only, as described in §5.1. As regards the S_ℓ operators, for $\ell = 0$ we need to solely use matrix-vector products, while for $\ell > 0$ we can adopt more conventional smoothers. This is outlined in §5.2.

5.1. Construction of K_1 . The construction of K_1 is divided into three phases:

- (1) Definition of the aggregates. This operation only requires the graph \mathcal{G}_0 of the grid.
- (2) Definition of the tentative prolongator \tilde{P}_0 . This requires the aggregates defined in the previous step, and the near kernel $\mathcal{R}(B_0)$ of K_0 . Once \tilde{P}_0 has been defined, we set $P_0 = (\tilde{P}_0)^T$, $R_0 = P_0^T$.

(3) Computation of the (i, j) block-elements of K_1 ,

$$(6) \quad K_1(i, j) = \Phi_i^T K_0 \Phi_j.$$

Φ_i is the i -th block column of \tilde{P}_0 , for non-smoothed aggregation methods, and as

$$(7) \quad K_1(i, j) = \Phi_i^T (I_0 - \omega_0 K_0 D_0^{-1}) K_0 (I_0 - \omega_0 D_0^{-1} K_0) \Phi_j,$$

for smoothed aggregation methods.

The only computationally expensive phase is the evaluation of either (6) or (7). Indeed, neither equation should be used directly, as K_1 is sparse and therefore most of its elements are zero. If the intersection of the supports of two basis functions Φ_i and Φ_j is zero, then their inner product in the K_0 -energy will be zero as well. The idea is therefore to collapse several Φ_i , so that a single application of K_0 can be used to evaluate a possibly large group of (block) elements of K_1 . The problem we are facing is similar to Jacobian evaluation [18]. This requires that the columns belonging to the same group have all their nonzero elements on different rows. Coleman and Moré [17] discussed the ordering in which the columns should be considered, in order to minimize the number of groups. They showed that, for a general sparse pattern, the problem is equivalent to a certain coloring problem on a suitable graph, and proposed the use of graph coloring to obtain a small number of groups. All entries sharing a color can then be evaluated at the same time, reducing the number of matrix-vector products.

The coloring adopted here is of the Curtis–Powell–Reed variety; two columns of the same color contain no row in which both have a nonzero. Thus, K_1 can be approximated by applying K_0 to all columns of the same color simultaneously. We note that obtaining an optimal or nearly optimal coloring for matrices arising from structured grid is simple, while efficient parallel algorithms for coloring matrices from unstructured grids remains an active research topic.

It is important to note that the required distance (in a graph sense) between two graph vertices of the same color varies depending on the application. A simple and cheap-to-compute distance-1 graph coloring, for example, can be used to estimate the matrix diagonal (which will be required by the smoother, see later §5.2). Equation (6) requires a distance-2 coloring. This means that every colored vertex of the graph does not share a neighbor with any other node of the same color. Equation (7) requires a distance-3 coloring. This increases the number of colors, and therefore the applications of K_0 and the memory requirements. For these reasons, in the following we will just focus on (6).

The cost of the construction of K_1 can be easily estimated as a function of the number χ of colors. Recalling that the kernel dimension is 6 for elasticity, the number of matrix-vector products is 6χ . In the numerical experiments reported in Section 7, we counted from 15 to 25 colors, meaning that about one hundred matrix-vector products will be required to fully determine K_1 .

We conclude this subsection by noting that the proposed preconditioner can be interpreted as a two-level hybrid domain decomposition preconditioner [45], that uses aggregation ideas for the construction of the level-1 matrix, and one or a few applications of a multigrid solver based on aggregation for the coarse problem.

5.2. Definition of Matrix-free Smoothers. It is convenient (and efficient) to adapt *Chebyshev polynomial preconditioners* to become multigrid smoothers [2]. Sparse approximate inverses have been proposed as parallel smoothers [14]. However, their setup time and memory cost are too high. Chebyshev smoothers, instead, have a modest setup cost, and only require the application of the diagonally-scaled operator. Further, their performances are independent of the number of processors used in the computation.

The setup phase of Chebyshev smoothers consists of determining an interval that contains the upper part of the spectrum of $D_\ell^{-1}K_\ell$. The upper end of this interval is obtained by a few (here 10) steps of the Lanczos algorithm together with a security margin [2]. The lower end is simply set to a fixed fraction of the upper bound.

6. SOFTWARE ISSUES

This section describes the software implementation that is used to obtain the numerical results reported in the next section. Our fully parallel implementation, called PARFE [44], originates from a FORTRAN77 program based on a serial FE code proposed in [46]. PARFE has been developed for massively parallel, distributed memory architectures, is written in C++ with support for MPI and optimized BLAS and LAPACK for computationally intensive tasks on dense matrices and vectors.

All PARFE’s data structures are distributed across the available processors. Mesh data structures are partitioned using the so-called *vertex-oriented distribution* [36], which is followed by most high-performance scientific packages like TRILINOS [22], HYPRE [19] and PETSC [8]. As a result, PARFE can be easily interfaced with the aforementioned libraries.

PARFE’s I/O operations are performed by the Hierarchical Data Format (HDF5) library [21]. HDF5 is a public domain library aiming at parallel, binary and portable I/O. It has been selected because it allows easy handling of files containing several GBytes of data. For the problems considered in this paper, file size is indeed an important issue: for example, if stored in ASCII text format, a file containing a problem of about 150 million unknowns requires over 3GB of storage. Its binary equivalent stored using HDF5 requires “only” 2GB, but more importantly, can be read or written in parallel on any architecture that supports HDF5, an important advantage for high performance computing.

To operate efficiently, high-performance HDF5 I/O requires linearly distributed data, while all the other software phases demands to equilibrate the workload assigned to each processor and to minimize the information exchange. Roughly speaking, the workload of a process depends on the number of vertices and elements assigned to that process while the communication volume depends on the number of edges connecting neighboring subdomains. An efficient way to achieve both goals is to adopt a mesh partitioning tool, either based on variants of coordinate bisection or multilevel graph partitioning. Since coordinate bisection methods only require the vertex coordinates, they can be used to provide a starting partition for more efficient and more costly graph partitioning algorithms, as performed by the PARMETIS [34] library. Fig. 2 displays the initial linear distribution (induced by the HDF5 reader), and the final distribution produced by PARMETIS.

As regards distributed linear algebra objects like vectors and matrices, we have adopted the EPETRA library [22]. EPETRA, based on object-oriented design, furnishes efficient and flexible implementations of fundamental distributed linear algebra objects like vectors, graphs and sparse matrices. It also offers concise but powerful data redistribution capabilities. For matrix-ready problems, K is stored in the distributed Variable Block Row (VBR) format [49]. This format is particularly suited for vector problems, as it defines K as a matrix composed of dense 3×3 matrices. The underlying graph describing the non-zero pattern only has to store the block indices and hence needs less memory and can be constructed faster. When using a matrix-free approach, instead, K is defined by a PARFE-specific class, which is used to apply K to input vectors [33].

EPETRA objects are accepted by the CG implementation given by the AZTECOO library [49] and by the multilevel preconditioning package ML [43]. ML offers a large variety of aggregation

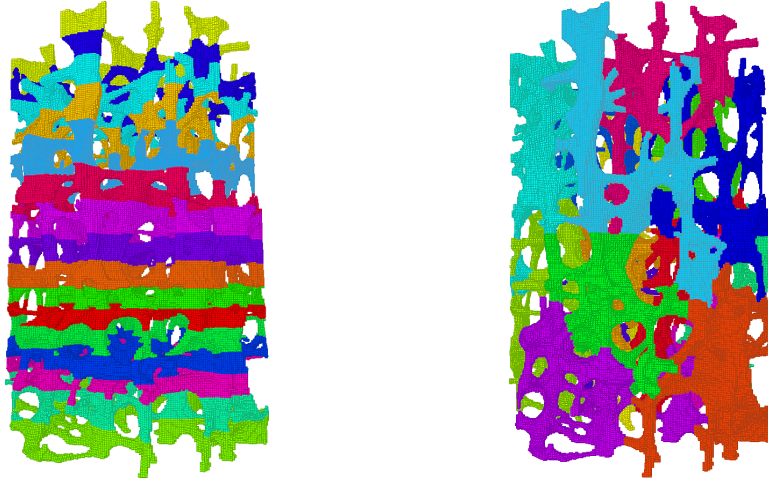


FIGURE 2. Result of repartitioning a 1.5 million nodes highly porous model with 16 processors. The left picture shows the initial linear partition, while the picture to the right shows a well balanced repartition.

schemes, smoothers and coarse solvers, either developed within ML itself, or as part of the IFPACK [42] and AMESOS [41] packages. In particular, the matrix-ready multigrid is implemented by ML's `MultiLevelPreconditioner` class, while the parallel matrix-free multigrid is implemented by ML's `MatrixFreePreconditioner`. Both classes define black-box solvers, that require as input either the matrix or the operator and its graph, a near kernel, and a list of options. EPETRA, ML, IFPACK and AMESOS are all available for download at the TRILINOS web site [47].

7. NUMERICAL EXPERIMENTS

In this section, we will present numerical results, obtained on a CRAY XT3 computer, located at the Swiss National Supercomputing Centre (CSCS, Manno, Switzerland). The machine consists of 1664 2.6 GHz AMD Opteron single-core processors equipped with 2GB of memory connected by a Cray SeaStar interconnect, see <http://www.cscs.ch> for details. In all simulations, the linear system (1) has been solved up to a tolerance of 10^{-5} in the relative residual, with a zero initial vector. We have grouped the numerical experiments in *problem sets* that are used to discuss different aspects of our algorithm.

7.1. Problem set JACOBI. First we compare the performances of matrix-free AMG and Jacobi preconditioners for a mesh composed of 320'751 elements and 521'118 vertices, resulting in a linear system with 1'563'354 equations. Only one processor is used in this experiment. The finest-level coarsening of the AMG preconditioner is based on aggressive coarsening, and each aggregate contains on average 128 nodes, while all other levels are built with standard diameter-3 coarsening. PCG with AMG preconditioning converged in 138 iteration steps, with Jacobi preconditioning in as many as 4719.

7.2. Problem set WEAK_SCAL. We now focus on the weak scalability of AMG preconditioners. The term weak scalability refers to a scaling study where the problem size *per processor* is kept constant. These problems were obtained by mirroring a small cube of human trabecular bone, scanned with a high resolution μ CT system. This procedure was used to obtain arbitrary sized cubic models as illustrated in Fig. 3. An overview of the problem sizes is reported in Table 1.

name	elements	nodes	matrix rows	file size
c01	60'482	98'381	295'143	9
c02	483'856	774'717	2'324'151	74
c03	1'633'014	2'609'611	7'828'833	250
c04	3'870'848	6'164'270	18'492'810	593
c05	7'560'250	12'038'629	36'115'887	1'157
c06	13'064'112	20'766'855	62'300'565	1'997
c07	20'745'326	32'983'631	98'950'893	3'172
c08	30'966'784	49'180'668	147'542'004	4'732
c09	44'091'378	70'042'813	210'128'439	6.737
c10	60'482'000	96'003'905	288'011'715	9'235
c12	104'512'896	165'834'762	497'504'286	15'953
c14	165'962'608	263'271'435	789'814'305	25'327
c15	204'126'750	323'887'399	971'662'197	31'155
c16	247'734'272	392'912'120	1'178'736'360	37'798

TABLE 1. Overview of problem sizes for problem set `WEAK_SCAL`. File size is expressed in MBytes.

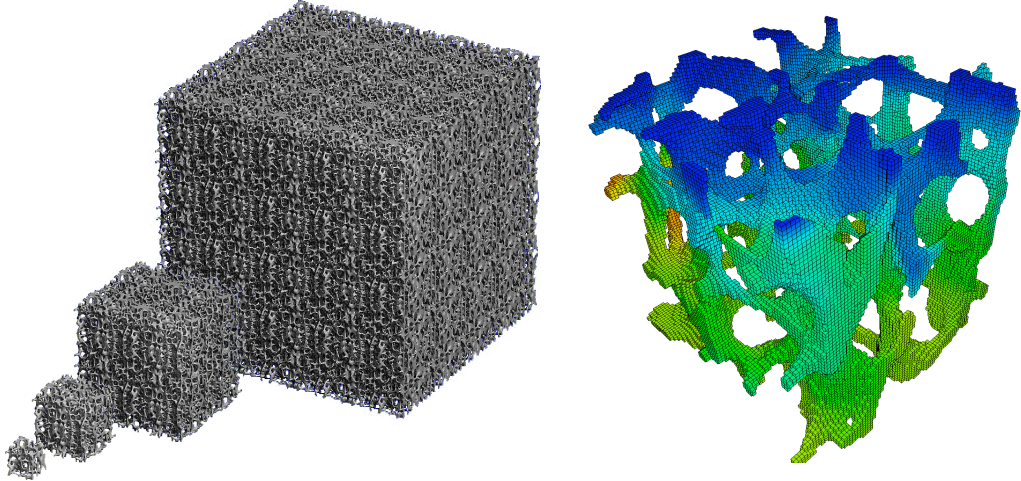


FIGURE 3. On the left, bone tissue models generated by 3D mirroring to test weak scalability, displayed models for 1, 8, 27, and 64 processors. On the right, a zoom on the 1-processor model stresses.

The matrix size n is roughly $300'000 \times x^3$ for problem cx . The loading conditions simulate the compression tests that can be performed experimentally.

A perfectly weakly scalable code executes in a time that is independent of the number of processors employed. We have analyzed matrix-ready methods using problems from `c01` to `c10`, employing $p = x^3$ processors for problem cx . We could solve problems `c03` to `c16` with the matrix-free methods using the same amount of memory per processor. We estimate that the current matrix-ready implementation requires more than 4000 CRAY XT3 processors to solve problem `c16`. Note that the matrix-ready implementation admits about 300'000 degrees of freedom per CRAY processor.

name	p	t_{input}	t_{repart}	t_{matrix}	t_{prec}	t_{solve}	t_{output}	t_{total}	n_{it}
c01	1	1.25	2.28	6.25	8.58	28.9	0.10	47.32	51
c02	8	1.27	3.84	6.64	9.03	31.0	0.52	52.28	53
c03	27	2.00	4.18	7.03	9.67	34.2	0.78	57.88	56
c04	64	3.65	4.20	7.12	10.1	32.6	1.33	58.94	53
c05	125	5.03	4.78	7.26	15.9	32.7	2.33	67.97	52
c06	216	8.23	4.92	7.26	15.9	32.3	3.81	72.47	51
c07	343	9.58	5.27	7.38	16.1	31.6	5.25	75.21	49
c08	512	17.3	5.39	7.29	17.0	30.2	8.03	85.33	47
c09	729	21.0	6.18	7.36	24.0	30.2	11.0	99.78	45
c10	1024	17.9	7.68	7.76	19.8	31.8	21.0	106.02	45

TABLE 2. Problem set **WEAK_SCAL** for matrix-ready environments. Execution times in seconds are given for reading input data, repartition the mesh for load balancing, matrix assembly, setup of the preconditioner, solving with the conjugate gradient method, writing the results, and the total execution time. Column n_{it} displays iterations steps until convergence.

The CPU times for the different phases of matrix-ready methods, as well as the CG iteration counts n_{it} are reported in Table 2. The CPU times are displayed in graphical form in Fig. 4, too. The AMG preconditioner is defined by a smoothed aggregation procedure, with a Chebyshev smoother for all levels except the coarsest one, where a serial direct solver is adopted. All phases scale almost perfectly, except I/O, which still needs improvements. The lack of I/O scalability

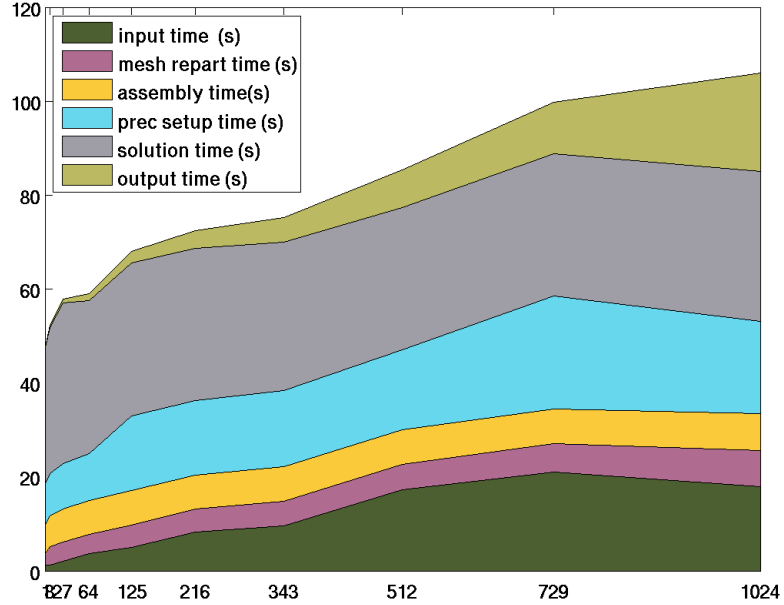


FIGURE 4. Problem set **WEAK_SCAL** with the matrix-ready AMG preconditioner: Execution times in seconds of the various subtasks versus processor numbers.

is due to the limited number of available I/O nodes with respect to computational nodes and, probably, due to congestion of the SeaStar network. Nevertheless, note that the 210M degrees of freedom test `c09` is solved in less than 100 seconds on $9^3 = 729$ CRAY processors.

name	p	t_{prec}	t_{solve}	n_{it}	χ	m_{prec}
c03	2	56.3	85.4	21	15	456
c04	16	79.4	88.3	20	16	434
c05	32	88.7	87.8	20	16	501
c07	86	87.4	95.0	21	17	484
c09	144	88.6	86.2	21	18	476
c09	183	96.7	96.4	21	16	516
c10	260	86.7	98.4	22	16	483
c12	460	103	97.8	22	18	506
c15	860	156	105	22	17	512
c16	1024	225	106	22	17	444

TABLE 3. Problem set `WEAK_SCAL` for matrix-free environments. Execution times in seconds are given for setting up the preconditioner (t_{prec}) and solving with the conjugate gradient method (t_{solve}). n_{it} displays iterations steps until convergence. χ gives the colors used during the preconditioner setup (cf. §5.1). m_{prec} shows the memory requirement of the preconditioner in MB.

Results for matrix-free methods are reported in Table 3. The table focuses on the linear system solution; the other phases, being executed by the exact same code, behave as in Table 2 and are not reported. (The actual numbers differ from those of Table 2 as the problem sizes differ.) The matrix-free preconditioner uses non-smoothed aggregation coarsening (with aggressive coarsening) to define the level-1 matrix, then classical smoothed aggregation. The level-0 smoother is given by 5 steps of the Chebyshev method, two sweeps of symmetric Gauss-Seidel with damping factor of 0.67 for all the other levels except the coarsest one, where a serial direct solver is used. The largest considered problem, `c16`, with about 1/4 billion elements and about 1.2 billion matrix rows, only requires 21 iteration steps and less than 6 minutes for solving the linear system. The overall time for this simulation was about 10 minutes. Clearly, forming as well as solving with the matrix-free preconditioner takes longer the matrix-ready preconditioner. Nevertheless, by solving a certain problem on a smaller number of processors reduces the number of messages sent but increases the average message length, which both have a positive impact on execution time.

7.3. Problem set `STRONG_SCAL`. In this problem set we analyse the strong scalability of AMG preconditioners. Strong scalability means that the problem size is kept fixed, independent on the number of processors. We have considered a simulated compression test of the distal part (of 20% of the length) of a human radius; see Fig. 5. This bone was scanned *in vitro* with a resolution of 93 μm . The model consisted of 3'687'438 elements and 4'841'054 vertices. The material properties used are Young's modulus $E = 17$ GPa and Poisson ratio $\nu = 0.3$. The preconditioner is defined as in `WEAK_SCAL`. When assembled, the linear system matrix consisted of 14'523'162 rows and 986'399'316 nonzeros. We needed at least 44 processors on the computer used for our testing. In this case, on each processor 1312 MBytes were required to store K , and additional 266 MBytes to store the preconditioner. On the same number of processors, adopting a matrix-free approach, required only 131 MBytes of memory for the matrix data structures, and 194 MBytes for the preconditioner. The minimal number of processors required to run this with the matrix-free

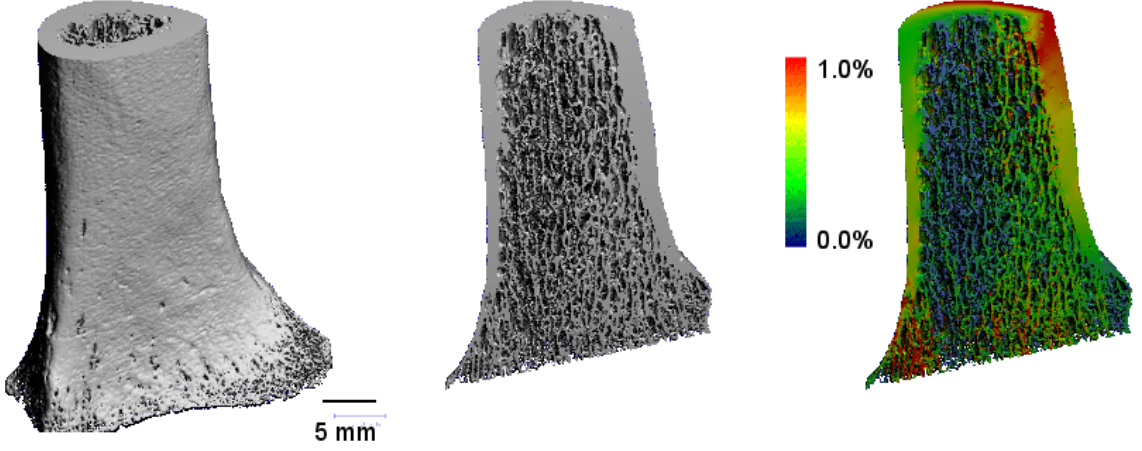


FIGURE 5. Finite element modeling of the distal part (20% of the length) of the radius in a human forearm. Left: full μ FE model; middle: part of the model removed to show the trabecular architecture; right: calculated effective strain distribution, as resulting from an axial compression test. (Images courtesy of Andreas Wirth, ETH Zurich, Switzerland)

	$p = 12$	$p = 16$	$p = 20$	$p = 36$	$p = 40$	$p = 44$	$p = 60$	$p = 72$
t'_{total}	†	†	†	†	†	167.91	120.89	92.17
t''_{total}	1060.54	765.23	514.23	417.51	360.72	344.50	186.44	193.33

TABLE 4. Problem set **STRONG_SCAL**. CPU time in seconds required to solve the problem using matrix-free (top) and matrix-ready preconditioners (bottom) on p processors. The symbol † indicates failure to run because of lack of memory.

approach was 12, or about 1'210'263 matrix rows per processor. Table 4 reports the CPU time for several values of p .

7.4. Problem set COMP. We finally report the memory usage and the setup time required in matrix-ready and matrix-free environments to build an AMG preconditioner. We have considered nonsmoothed aggregation (with diameter-3 aggregates) and Chebyshev smoothers. This problem set gives an estimation of the memory savings of the preconditioner presented in §5. Results are reported in Table 5. By adopting matrix-free AMG preconditioners, for the considered problems we estimate to solve problems about 3 times bigger. This is counterbalanced by an increased setup time, which is about 4 times larger than that of an equivalent matrix-ready AMG preconditioner.

8. CONCLUSIONS AND FUTURE DEVELOPMENTS

We have described the model and the implementation details for multi-level μ FE analysis of human bone structures. We have in particular shown how to construct an aggregation based multilevel preconditioner without assembling and storing the system matrix, i.e., the matrix of the finest grid. This reduces the memory consumption of the overall program by a factor 3 to 4. By using the presented techniques, we have solved a model of trabecular bone composed

problem			matrix-ready			matrix-free				
name	p	m_{mesh}	m'_{mat}	m'_{prec}	t'_{prec}	m''_{mat}	m''_{prec}	t''_{prec}	$\frac{m'_{\text{mat}}+m'_{\text{prec}}}{m''_{\text{mat}}+m''_{\text{prec}}}$	$\frac{t'_{\text{prec}}}{t''_{\text{prec}}}$
c01	1	101	867	290	5.43	54	296	47.84	3.30	8.81
c02	8	123	876	308	5.51	89	328	27.37	2.83	4.96
c03	27	124	877	320	6.90	90	340	30.94	2.78	4.48
c04	64	123	882	303	7.80	88	324	34.44	2.31	4.99
c05	125	124	878	310	10.92	89	335	35.44	2.80	3.24
c06	216	125	912	323	9.92	92	331	36.75	2.91	3.70
c07	343	124	802	299	10.64	84	316	40.65	2.75	3.82
c08	512	126	928	325	9.39	93	342	42.13	2.88	4.48

TABLE 5. Problem set **COMP**: comparison of memory requirements (matrix and preconditioner) and preconditioner setup time for matrix-ready and matrix-free preconditioners.

by 247'734'272 elements, leading to a linear system with 1'178'736'360 equations, in less than 9 minutes using 1024 CRAY XT3 processors. Numerical results show excellent weak and strong scalability of the method and of the overall code, in terms of both iteration count and CPU time.

The short solution times which we have obtained is an important step towards the assessment of the mechanical quality of bone *in vivo* on a routine basis. Furthermore, solving the even larger models of whole bones measured *in vitro* becomes possible. We expect these findings to improve our understanding of the influence of densitometric, morphological and loading factors in the etiology of spontaneous fractures of the hip and the spine.

Obviously, the complexity of trabecular bones is pushing researchers to extend the linear elasticity model to incorporate nonlinear effects (where the nonlinearity is given by the nonlinear properties, as well as large deformations and geometric instabilities), time derivatives, mesh smoothing and mesh adaption, just to mention a few. Efficient implicit nonlinear methods are generally built around one of the numerous variants of Newton's method [3, 6]. Newton's method is also well-known to exhibit a convergence rate that is independent of spatial resolution in systems arising from elliptic-like PDEs. The fundamental algorithmic challenge for Newton's method is to solve linear systems with the Jacobian matrix, for which the AMG solver described in this paper can be applied.

ACKNOWLEDGMENTS

The computations on the Cray XT3 have been performed in the framework of a Large User Project grant of the Swiss National Supercomputing Centre (CSCS), Manno, Switzerland. We acknowledge the help of the XT3 support team of the CSCS. G. H. van Lenthe and R. Müller were funded in parts through the Swiss National Science Foundation (PP-104317/1).

REFERENCES

- [1] M. Adams. Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics. *Internat. J. Numer. Methods Engrg.*, 55(5):519–534, 2002.
- [2] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss–Seidel. *J. Comput. Phys.*, 188(2):593–610, 2003.
- [3] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos. Ultrascable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In *ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing*, 2004. Available from <http://www.sc-conference.org/sc2004/schedule/pdfs/pap111.pdf>.

- [4] P. Arbenz, G. H. van Lenthe, U. Mennel, R. Müller, and M. Sala. Multi-level μ -finite element analysis for human bone structures. In *Proceedings of the PARA'06 State-of-the-Art in Scientific Computing*. Umeå, Sweden, June 18–21, 2006 (accepted).
- [5] C. E. Augarde, A. Ramage, and J. Staudacher. An element-based displacement preconditioner for linear elasticity problems. *Computers and Structures*, 84(31–32):2306–2315, 2006.
- [6] C. E. Augarde, A. Ramage, and J. Staudacher. Element-based preconditioners for elasto-plastic problems in geotechnical engineering. *Internat. J. Numer. Methods Engrg.*, (in press), 2007. doi:10.1002/nme.1947.
- [7] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, Cambridge, 1994.
- [8] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
- [9] R. Blaheta. Algebraic multilevel methods with aggregations: An overview. In I. Lirkov, S. Margenov, and J. Waśniewski, editors, *Large-Scale Scientific Computing*, pages 3–14. Springer, 2005. (Lecture Notes in Computer Science, 3743).
- [10] S. Boutroy, M.L. Boussein, F. Munoz, and P.D. Delmas. In vivo assessment of trabecular bone microarchitecture by high-resolution peripheral quantitative computed tomography. *J. Clin. Endocrinol. Metab.*, 90(12):6508–6515, 2005.
- [11] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31(138):333–390, 1977.
- [12] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Algebraic multigrid based on element interpolation (AMGe). *SIAM J. Sci. Comput.*, 22(5):1570–1592, 2000.
- [13] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge. Adaptive smoothed aggregation (α SA) multigrid. *SIAM J. Sci. Comput.*, 25(6):1896–1920, 2004. Reprinted in *SIAM Rev.* 47(2): 317–346, 2005.
- [14] O. Bröker, M. J. Grote, C. Mayer, and A. Reusken. Robust parallel smoothing for multigrid via sparse approximate inverses. *SIAM J. Sci. Comput.*, 23(4):1396–1417, 2001.
- [15] T. F. Chan and P. Vaněk. Detection of strong coupling in algebraic multigrid solvers. In E. Dick, K. Riemslaagh, and J. Vierendeels, editors, *Multigrid methods VI*, pages 11–23. Springer, 2000. (Lecture Notes in Computational Science and Engineering, 14).
- [16] R. T. Chapman and D. L. Cox. A unique element storage implementation of the vectorized element by element preconditioned conjugate gradient. In *Iterative Equation Solvers for Structural Mechanics Problems*, pages 57–65, New York, 1991. ASME.
- [17] T. F. Coleman and J. J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20:187 – 209, 1984.
- [18] A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *IMA J. Appl. Math.*, 13(1):117–119, 1974.
- [19] R. D. Falgout and U. M. Yang. HYPRE: a library of high performance preconditioners. In P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, and A. G. Hoekstra, editors, *Computational Science - ICCS 2002 Part III*, pages 632–641. Springer, 2002. (Lecture Notes in Computer Science, 2331).
- [20] W. Hackbusch. *Multi-grid Methods and Applications*. Springer, Berlin, 1985.
- [21] *HDF5: Hierarchical Data Format*, 2005. Reference Manual and User’s Guide are available from <http://hdf.ncsa.uiuc.edu/HDF5/doc/>.
- [22] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.
- [23] J. Homminga, B.R. McCreadie, R. Huiskes, and H. Weinans. The dependence of the elastic properties of osteoporotic cancellous bone on volume fraction and fabric. *J. Biomech.*, 36(10):1461–1467, 2003.
- [24] J. R. Hughes, R. M. Ferencz, and J. O. Hallquist. Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients. *Comput. Methods Appl. Mech. Engrg.*, 61(2):215–248, 1987.
- [25] J. Kabel, B. van Rietbergen, M. Dalstra, A. Odgaard, and R. Huiskes. The role of an effective isotropic tissue modulus in the elastic properties of cancellous bone. *J. Biomech.*, 32(7):673–680, 1999.
- [26] T. M. Keaveny, E. F. Morgan, G. L. Niebur, and O. C. Yeh. Biomechanics of trabecular bone. *Annu. Rev. Biomed. Engrg.*, 3:307–333, 2001.

- [27] S. Khosla, B. L. Riggs, E. J. Atkinson, A. L. Oberg, L. J. McDaniel, M. Holets, J. M. Peterson, and L. J. Melton III. Effects of sex and age on bone microstructure at the ultradistal radius: A population-based noninvasive in vivo assessment. *J. Bone Miner. Res.*, 21(1):124–131, 2006.
- [28] A. J. Ladd, J. H. Kinney, D. L. Haupt, and S. A. Goldstein. Finite-element modeling of trabecular bone: comparison with mechanical testing and determination of tissue modulus. *J. Orthop. Res.*, 16(5):622–628, 1998.
- [29] H.-C. Lee and A. J. Wathen. On element-by-element preconditioning for general elliptic problems. *Comput. Methods Appl. Mech. Engrg.*, 92(2):215–229, 1991.
- [30] G. H. van Lenthe and R. Müller. Prediction of failure load using micro-finite element analysis models: Towards in vivo strength assessment. *Drug Discovery Today: Technologies*, 3(2):221–229, 2006.
- [31] P. T. Lin, M. Sala, J. N. Shadid, and R. S. Tuminaro. Performance of fully coupled algebraic multilevel domain decomposition preconditioners for incompressible flow and transport. *Internat. J. Numer. Methods Engrg.*, 67(2):208–225, 2006.
- [32] L. J. Melton III, E. A. Chrischilles, C. Cooper, A. W. Lane, and B. L. Riggs. How many women have osteoporosis? *J. Bone Miner. Res.*, 7(9):1005–1010, 1992. Reprinted in *J. Bone Miner. Res.* 20(5): 886–892, 2005.
- [33] U. Mennel. High performance computing for the structural analysis of human bones. Master thesis, ETH Zurich, Institute of Computational Science, May 2006.
- [34] METIS - Family of Multilevel Partitioning Algorithms. <http://glaros.dtc.umn.edu/gkhome/views/metis>.
- [35] The Prometheus Project Home Page. http://www.columbia.edu/~ma2325/prom_intro.html.
- [36] A. Quarteroni, M. Sala, and A. Valli. An interface-strip domain decomposition preconditioner. *SIAM J. Sci. Comput.*, 28(2):498–516, 2006.
- [37] B. van Rietbergen, H. Weinans, R. Huiskes, and A. Odgaard. A new method to determine trabecular bone elastic properties and loading using micromechanical finite-elements models. *J. Biomech.*, 28(1):69–81, 1995.
- [38] B. van Rietbergen, H. Weinans, R. Huiskes, and B. J. W. Polman. Computational strategies for iterative solutions of large FEM applications employing voxel data. *Internat. J. Numer. Methods Engrg.*, 39(16):2743–2767, 1996.
- [39] J. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. McCormick, editor, *Multigrid Methods*, Frontiers in Applied Mathematics, pages 73–130, Philadelphia, PA, 1987. SIAM.
- [40] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, 2nd edition, 2003.
- [41] M. Sala. Amesos 2.0 reference guide. Technical Report SAND-4820, Sandia National Laboratories, September 2004. Available from [47].
- [42] M. Sala and M. Heroux. Robust algebraic preconditioners with IFPACK 3.0. Technical Report SAND-0662, Sandia National Laboratories, 2005. Available from [47].
- [43] M. Sala, J. Hu, and R. Tuminaro. ML 3.1 smoothed aggregation user’s guide. Technical Report SAND-4819, Sandia National Laboratories, September 2004. Available from [47].
- [44] M. Sala, U. Mennel, P. Arbenz, C. Flaig, H.G. van Lenthe, and A. Wirth. The PARFE Project Home Page. <http://parfe.sourceforge.net>.
- [45] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, Cambridge, 1996.
- [46] I. M. Smith and D. V. Griffiths. *Programming the Finite Element Method*. Wiley, Chichester, 4th edition, 2004.
- [47] The Trilinos Project Home Page. <http://software.sandia.gov/trilinos>.
- [48] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2000.
- [49] R. Tuminaro, M. Heroux, S. Hutchinson, and J. Shadid. Official Aztec user’s guide: Version 2.1. Technical Report Sand99-8801J, Sandia National Laboratories, Albuquerque NM, November 1999. Available from [47].
- [50] R. S. Tuminaro. Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines. In *Supercomputing ’00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM)*, Washington DC, 2000. IEEE Computer Society. (5 pages).
- [51] P. Vaněk, M. Brezina, and J. Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numer. Math.*, 88(3):559–579, 2001.
- [52] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56(3):179–196, 1996.
- [53] A. J. Wathen. An analysis of some element-by-element techniques. *Comput. Methods Appl. Mech. Engrg.*, 74(3):271–287, 1989.
- [54] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method for Solid and Structural Mechanics*. Elsevier, Oxford, 6th edition, 2005.