SIAM J. SCI. COMPUT. © 1997 Society for Industrial and Applied Mathematics
Vol. 18, No. 6, pp. 1767–1787, November 1997 012

ELEMENT-BY-ELEMENT PRECONDITIONERS FOR LARGE PARTIALLY SEPARABLE OPTIMIZATION PROBLEMS*

MICHEL J. DAYDÉ[†], JEAN-YVES L'EXCELLENT[‡], AND NICHOLAS I. M. GOULD[§]

Abstract. We study the solution of large-scale nonlinear optimization problems by methods which aim to exploit their inherent structure. In particular, we consider the property of partial separability, first studied by Griewank and Toint [*Nonlinear Optimization*, 1981, pp. 301–312]. A typical minimization method for nonlinear optimization problems approximately solves a sequence of simplified linearized subproblems. In this paper, we explore how partial separability may be exploited by iterative methods for solving these subproblems. We particularly address the issue of computing effective preconditioners for such iterative methods. We concentrate on element-by-element preconditioners which reflect the structure of the problem. We find that the performance of these methods can be considerably improved by amalgamating elements before applying the preconditioners. We report the results of numerical experiments which demonstrate the effectiveness of this approach.

Key words. large-scale optimization, partial separability, preconditioned conjugate gradient, element-by-element preconditioners

AMS subject classifications. 65, 65F10

PII. S1064827594274796

1. Introduction.

In this paper, we study algebraic aspects of the numerical solution of large-scale unconstrained optimization problems. To be specific, we suppose that we wish to minimize a partially separable objective function $f(\boldsymbol{x})$. The function f is said to be *partially separable* (see [15] and [17]) if

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} f_i(\boldsymbol{x}), \tag{1.1}$$

where each *element* function f_i has a large invariant subspace. Typically, this occurs when $f_i(\boldsymbol{x})$ is only a function of a small subset of the variables $\boldsymbol{x}$, but may, of course, happen for other reasons (see, for example, [5]). The decomposition (1.1) is extremely general. Indeed, Griewank and Toint [16] show that any sufficiently differentiable function with a sparse Hessian matrix may be expressed in this form.

In this paper, we shall be concerned with those partially separable functions for which

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} f_i(\boldsymbol{x}^i), \tag{1.2}$$

where

1. each set of *local* variables, $\boldsymbol{x}^i \in \Re^{n_i}$, is a subset of the *global* variables, $\boldsymbol{x} \in \Re^n$, and
2. $n_i \ll n$.

*Received by the editors September 23, 1994; accepted for publication (in revised form) March 22, 1996. Travel was funded in part by the ALLIANCE program from the British Council.
http://www.siam.org/journals/sisc/18-6/27479.html

[†]ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse Cedex, France (dayde@enseeiht.fr).

[‡]ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse Cedex, France. Current address: CERFACS, 42 av. G. Coriolis, 31057 Toulouse Cedex, France (excelle@cerfacs.fr).

[§]Central Computing Department, Rutherford Appleton Laboratory, Oxfordshire, OX11 0QX, England (nimg@letterbox.rl.ac.uk).

Thus, the Hessian matrix of each f_i is a low-rank, sparse matrix—typically it will differ from the zero matrix only in a full block in the rows and columns corresponding to the variables $\boldsymbol{x}^i$. The overall Hessian is thus the sum of extremely sparse matrices, the *element Hessians*, and is thus frequently itself also sparse.

In unconstrained optimization, one is normally concerned with obtaining an (approximate) solution $\boldsymbol{d}$ to the Newton equations

$$\nabla_{xx} f(\boldsymbol{x})\boldsymbol{d} = -\nabla_x f(\boldsymbol{x}). \tag{1.3}$$

If f has the form (1.2), these equations become

$$\left(\sum_{i=1}^{p} \nabla_{xx} f_i(\boldsymbol{x}^i)\right) \boldsymbol{d} = -\sum_{i=1}^{p} \nabla_x f_i(\boldsymbol{x}^i). \tag{1.4}$$

We are thus concerned with constructing efficient methods for solving systems of this form which exploit the algebraic structure as fully as possible.

Putting this in a more general context, we suppose that the real, symmetric, $n \times n$ matrix $\boldsymbol{H}$ may be expressed as

$$\boldsymbol{H} = \sum_{i=1}^{p} \boldsymbol{H}_i, \tag{1.5}$$

where the symmetric *elementary* matrix $\boldsymbol{H}_i$ only has nonzeros in n_i rows and columns. We consider solving the linear system

$$\boldsymbol{H}\boldsymbol{d} = \left(\sum_{i=1}^{p} \boldsymbol{H}_i\right) \boldsymbol{d} = -\boldsymbol{g}, \tag{1.6}$$

where $\boldsymbol{H}$ is large and normally positive definite. Clearly, the system (1.4) is of this form. Similar linear systems arise when solving constrained optimization problems using augmented Lagrangian methods (see, for example, [20], [31], and [5]), and when using finite element methods to solve elliptic partial differential equations (see, for instance, [38]). Both direct and iterative methods may be appropriate for solving (1.6). Frontal or multifrontal direct (factorization) methods (see, for example, [26], [9], and [32]) are appropriate so long as there is room to store the fill-in which occurs during the matrix factorization. If this is not the case, one is forced to consider iterative methods. The symmetry and definiteness of $\boldsymbol{H}$ normally makes the preconditioned conjugate gradient method (see, for example, [14]) the method of choice. The difficulty is, of course, the choice of an effective preconditioner (see [3] for a discussion of general issues).

When designing an iterative solver for the solution of (1.6), certain features appear to be desirable or even crucial.

1. Since the matrix (1.5) is initially unassembled, we do not especially want to assemble it.

2. We would like to find a preconditioner that can be computed elementwise. It is also desirable not to have to assemble this preconditioner.

3. The computations involved in forming and using the preconditioner should ideally be vectorizable and parallelizable, since we are interested in solving large problems. Note that in the conjugate gradient method, the most time consuming parts at each iteration are the matrix–vector product and the operation of the preconditioner on a vector. As the matrix–vector product can be easily parallelized, it is thus crucial to parallelize the preconditioning operation.

4. In an optimization context, it is important that the matrix (1.5) be positive definite, as otherwise the solution to the Newton equations (1.4) may not be a descent direction for f. As there is no absolute guarantee that the matrix (1.5) is positive definite, we would thus like to be able to detect when the matrix is indefinite. Thereafter, we may perturb the original matrix so that in all cases we compute the solution of a positive definite system (see [28] and [2]).

The LANCELOT package for the solution of large-scale nonlinear optimization problems uses a variety of techniques to solve systems of the form (1.4). As well as direct methods, the package allows the use of conjugate gradients with various preconditioners. These include the following:

1. diagonal preconditioners,
2. band preconditioners,
3. incomplete Cholesky preconditioners,
4. expanding band preconditioners, and
5. full-matrix factorization preconditioners.

Further details are given in [6].

We have decided to explore other alternatives, keeping in mind that the preconditioners should take advantage of the structure of the problems given by the partial separability and that we must be able to ensure that the preconditioners are symmetric positive definite. Thus, in this paper, we study the use of the following element-by-element preconditioners:

1. The element matrix factorization (EMF) of Gustafsson and Lindskog [19] based on a Cholesky factorization of each element.
2. The finite element preconditioner (FEP) of Kaasschieter [27].
3. The one-pass (EBE) and two-pass (EBE2) element-by-element preconditioners of Hughes, Levit, and Winget [25] and Ortiz, Pinsky, and Taylor [30] initially described and used in the context of finite element techniques for partial differential equations.
4. The "Gauss–Seidel" EBE preconditioner (GS EBE).

In section 2, we consider how these element-by-element preconditioners perform on examples from a variety of application areas. In section 3, we make some remarks on the influence of the partitioning of the matrix (1.5) into elementary matrices, and show how the methods considered in section 2 may be improved by merging the elements.

We use the following notation: we let $\boldsymbol{I}$ denote the (appropriately dimensioned) identity matrix and we let $\Delta(\boldsymbol{A})$ be the diagonal matrix whose diagonals are the diagonals of $\boldsymbol{A}$.

2. Finite element preconditioners.

In this section, we review the range of element-by-element preconditioners which have been proposed for solving the linear systems that arise from finite element solution of partial differential equations. We note that specific error analyses are possible for particular classes of model differential equations, but a more general analysis is most likely impossible. Thus, all of the preconditioners should be viewed as heuristics which aim to approximate (1.5) at low cost.

2.1. Connectivity matrices.

As we are assuming that the element matrix $\boldsymbol{H}_i \in \Re^{n\times n}$ has nonzeros in just n_i rows and columns, we may write

$$\boldsymbol{H}_i = \boldsymbol{C}_i^T \boldsymbol{H}_i^e \boldsymbol{C}_i, \tag{2.1}$$

where the rows of the *connectivity* matrix $\boldsymbol{C}_i \in \Re^{n_i \times n}$ are simply the rows of the $n \times n$ identity matrix corresponding to the variables used in the element, and $\boldsymbol{H}_i^e \in \Re^{n_i \times n_i}$ is a symmetric matrix which is dense in general. In what follows, we shall say that $\boldsymbol{H}_i$ is positive definite when strictly we mean that $\boldsymbol{H}_i^e$ is positive definite.

2.2. Element matrix factorization. First, we assume that the elementary matrices $\boldsymbol{H}_i^e$ are positive definite. Gustafsson and Lindskog [19] suggest forming a preconditioner by factorizing each elementary matrix into

$$\boldsymbol{H}_i^e = \boldsymbol{L}_i^e \boldsymbol{L}_i^{eT}, \tag{2.2}$$

where $\boldsymbol{L}_i^e \in \Re^{n_i \times n_i}$ is a lower triangular matrix. The preconditioner is then

$$\boldsymbol{P}_{EMF} = \left(\sum_{i=1}^{p} \boldsymbol{L}_i \right) \left(\sum_{i=1}^{p} \boldsymbol{L}_i \right)^T, \tag{2.3}$$

where $\boldsymbol{L}_i = \boldsymbol{C}_i^T \boldsymbol{L}_i^e C_i \in \Re^{n \times n}$. If the numbering of the local variables in each element is in increasing order corresponding to the global variables, it is clear that $\sum_{i=1}^{p} \boldsymbol{L}_i$ is also lower triangular and thus (2.3) is easy to invert. If this is not the case, local variables should be permuted in the elements so that $\sum_{i=1}^{p} \boldsymbol{L}_i$ is lower triangular. More generally, letting $\boldsymbol{L}_i = \bar{\boldsymbol{L}}_i + \boldsymbol{D}_i$, where $\bar{\boldsymbol{L}}_i \in \Re^{n \times n}$ is the strictly lower triangular part of $\boldsymbol{L}_i$ and $\boldsymbol{D}_i = \Delta(\boldsymbol{L}_i) \in \Re^{n \times n}$ is the diagonal of $\boldsymbol{L}_i$, we might choose

$$\boldsymbol{P}_{EMF}(\theta) = \left((1+\theta)^{-1} \sum_{i=1}^{p} \bar{\boldsymbol{L}}_i + (1+\theta) \sum_{i=1}^{p} \boldsymbol{D}_i \right) \left((1+\theta)^{-1} \sum_{i=1}^{p} \bar{\boldsymbol{L}}_i + (1+\theta) \sum_{i=1}^{p} \boldsymbol{D}_i \right)^T, \tag{2.4}$$

where θ is a nonnegative parameter. In our experiments for simplicity we choose $\theta = 0$, but other choices have been suggested by Gustafsson and Lindskog [19] for finite element applications.

2.3. Finite element preconditioner. If $\boldsymbol{H}_i^e$ is positive semidefinite, Kaasschieter [27] has suggested using the alternative element factorization

$$\boldsymbol{H}_i^e = (\boldsymbol{D}_i^e + \bar{\boldsymbol{L}}_i^e) \boldsymbol{D}_i^{e+} (\boldsymbol{D}_i^e + \bar{\boldsymbol{L}}_i^{eT}), \tag{2.5}$$

where $\boldsymbol{D}_i^{e+} \in \Re^{n_i \times n_i}$ is the pseudoinverse of $\boldsymbol{D}_i^e \in \Re^{n_i \times n_i}$. Writing $\boldsymbol{D}_i = \boldsymbol{C}_i^T \boldsymbol{D}_i^e \boldsymbol{C}_i \in \Re^{n \times n}$ and $\boldsymbol{L}_i = \boldsymbol{C}_i^T \bar{\boldsymbol{L}}_i^e \boldsymbol{C}_i \in \Re^{n \times n}$, the FEP is

$$\boldsymbol{P}_{FEP} = \left(\sum_{i=1}^{p} \boldsymbol{D}_i + \sum_{i=1}^{p} \bar{\boldsymbol{L}}_i \right) \left(\sum_{i=1}^{p} \boldsymbol{D}_i \right)^{-1} \left(\sum_{i=1}^{p} \boldsymbol{D}_i + \sum_{i=1}^{p} \bar{\boldsymbol{L}}_i^T \right). \tag{2.6}$$

This preconditioner was also studied by Wathen [37].

2.4. The EBE preconditioner. EBE preconditioners were introduced by Hughes, Levit, and Winget [25] and Ortiz, Pinsky, and Taylor [30] and have been successfully applied in a number of applications in engineering and physics (see, for example, [23], [24], and [11]). A detailed analysis of this technique is given by Wathen [36].

We assume that $\boldsymbol{H}$ is positive definite and express $\boldsymbol{H}$ as

$$\boldsymbol{H} = \sum_{i=1}^{p} \boldsymbol{M}_i + \sum_{i=1}^{p} (\boldsymbol{H}_i - \boldsymbol{M}_i) = \boldsymbol{M} + \sum_{i=1}^{p} (\boldsymbol{H}_i - \boldsymbol{M}_i), \tag{2.7}$$

where $\boldsymbol{M}_i = \Delta(\boldsymbol{H}_i)$ and $\boldsymbol{M} = \sum_{i=1}^p \boldsymbol{M}_i$. Let $\boldsymbol{M} = \boldsymbol{L}_M \boldsymbol{L}_M^T$ be the Cholesky factorization of $\boldsymbol{M}$; of course, $\boldsymbol{L}_M$ is simply a diagonal matrix. Then,

$$\boldsymbol{H} = \boldsymbol{L}_M \left(\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{L}_M^{-1}(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{L}_M^{-T} \right) \boldsymbol{L}_M^T = \boldsymbol{L}_M \left(\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \right) \boldsymbol{L}_M^T, \tag{2.8}$$

where we have defined $\boldsymbol{E}_i = \boldsymbol{L}_M^{-1}(\boldsymbol{H}_i - \boldsymbol{M}_i)\boldsymbol{L}_M^{-T} \in \Re^{n\times n}$. Now consider the sum $\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i$. A simple calculation reveals that

$$\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \approx \prod_{i=1}^{p} (\boldsymbol{I} + \boldsymbol{E}_i), \tag{2.9}$$

where the error in the approximation may be expressed in terms of second- and higher-order products of the components $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ with $i \neq j$. Thus $\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i$ is well approximated by $\prod_{i=1}^p (\boldsymbol{I} + \boldsymbol{E}_i)$ if the the norms of the terms involving products of the $\boldsymbol{E}_i$ are small compared to one. This may be true for various reasons.

1. Individual $\boldsymbol{E}_i$ may be small or zero. This is likely to be true if $\boldsymbol{H}_i$ is strongly diagonal dominant.

2. The product of the overlapping components $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ is small or zero.

Note that the order used for writing the product is not without importance. The preconditioner has to be symmetric as we intend to use conjugate gradients. The $\boldsymbol{E}_i$ are symmetric matrices but $\boldsymbol{E}_i\boldsymbol{E}_j$ is, in general, not. There are, of course, ways of symmetrizing the approximation (2.9), such as writing

$$\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \approx \left(\prod_{i=1}^{p} (\boldsymbol{I} + 1/2\boldsymbol{E}_i) \right) \left(\prod_{i=p}^{1} (\boldsymbol{I} + 1/2\boldsymbol{E}_i) \right), \tag{2.10}$$

but a discussion of probably the best such scheme is deferred until section 2.5.

The first fundamental feature of the EBE preconditioner is that we replace $\boldsymbol{I} + \sum_{i=1}^p \boldsymbol{E}_i$ by $\prod_{i=1}^p (\boldsymbol{I} + \boldsymbol{E}_i)$ in (2.8). We then assume that $\boldsymbol{I} + \boldsymbol{E}_i$ is positive definite and has an LDL^T factorization

$$\boldsymbol{W}_i \overset{\text{def}}{=} \boldsymbol{I} + \boldsymbol{E}_i = \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T; \tag{2.11}$$

the matrix $\boldsymbol{W}_i \in \Re^{n\times n}$ is known as the Winget decomposition of $\boldsymbol{H}_i$ (see [25]). In this case, (2.8), (2.9), and (2.11) imply that

$$\boldsymbol{H} \approx \boldsymbol{L}_M \left(\prod_{i=1}^{p} \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T \right) \boldsymbol{L}_M^T. \tag{2.12}$$

Unfortunately, (2.12) is normally unsymmetric, and thus is not a satisfactory preconditioner. However, if we further assume that a rearrangement of the product

$$\prod_{i=1}^{p} \boldsymbol{L}_i \boldsymbol{D}_i \boldsymbol{L}_i^T \approx \left(\prod_{i=1}^{p} \boldsymbol{L}_i \right) \left(\prod_{i=1}^{p} \boldsymbol{D}_i \right) \left(\prod_{i=p}^{1} \boldsymbol{L}_i^T \right) \tag{2.13}$$

introduces little additional error, we obtain the matrix

$$\boldsymbol{P}_{EBE} = \boldsymbol{L}_M \left(\prod_{i=1}^{p} \boldsymbol{L}_i \right) \left(\prod_{i=1}^{p} \boldsymbol{D}_i \right) \left(\prod_{i=p}^{1} \boldsymbol{L}_i^T \right) \boldsymbol{L}_M^T, \tag{2.14}$$

which may be used as a preconditioner for $\boldsymbol{H}$. Such a matrix is known as the *EBE* preconditioner. Note that the second approximation (2.13) is, as the previous one, exact if there is no overlap between the blocks and will be good under exactly the same circumstances as its predecessor.

Clearly, the efficiency of the EBE preconditioner depends on the partitioning of the initial matrix and on the size of the off-diagonal elements of the elementary matrices. In order to solve efficiently the system of equations $\boldsymbol{P}_{EBE}\boldsymbol{x} = \boldsymbol{y}$, we exploit the decomposition (2.14). We are free to order the elements in any way we choose and may thus encourage parallelism by consecutively ordering nonoverlapping elements so that we can perform groups of forward and backsolve in parallel.

Generalizations of these ideas have been suggested by Daydé, L'Excellent, and Gould [8] but have rarely proved to yield significant improvements over the methods described above.

2.5. Two-pass EBE preconditioners. Here we consider another proposal in the same vein (see [22]). We proceed, as in section 2.4, to approximate the sum $\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i$ by a product of invertible matrices. We dismissed using the approximation (2.12) as a preconditioner because of its nonsymmetry. Instead, we combine (2.8) and the relationship

$$\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i \approx \prod_{i=1}^{p} (\boldsymbol{I} + 1/2\ \boldsymbol{E}_i) \prod_{i=p}^{1} (\boldsymbol{I} + 1/2\ \boldsymbol{E}_i) \tag{2.15}$$

to give the preconditioner

$$\boldsymbol{P}_{EBE2} = \boldsymbol{L}_M \left(\prod_{i=1}^{p} (\boldsymbol{I} + 1/2\ \boldsymbol{E}_i) \right) \left(\prod_{i=p}^{1} (\boldsymbol{I} + 1/2\ \boldsymbol{E}_i) \right) \boldsymbol{L}_M^T. \tag{2.16}$$

Note that (2.16) is positive definite if and only if all the matrices $\boldsymbol{I} + 1/2\ \boldsymbol{E}_i$ are nonsingular. To use this preconditioner, we merely require that each $\boldsymbol{I} + 1/2\ \boldsymbol{E}_i$ is invertible, and to be able to solve systems of equations of the form $\boldsymbol{P}_{EBE2}\boldsymbol{x} = \boldsymbol{y}$ efficiently.

As before, we are free to order the elements in any way we choose and may thus encourage parallelism by consecutively ordering nonoverlapping elements. We may also choose to obtain explicit inverses of the $\boldsymbol{I} + 1/2\ \boldsymbol{E}_i$ to exploit vectorization in the forward and back substitutions.

The main problem with the approximation (2.16) is that the error terms $1/4\ {\boldsymbol{E}_i}^2$, which result from the approximation (2.15), are nonzero even if there is little overlap between distinct element Hessians $\boldsymbol{E}_i$ and $\boldsymbol{E}_j$ $(i \neq j)$. Furthermore, as a solve using EBE2 is roughly twice as expensive as one with EBE, in practice EBE2 is less efficient than EBE. But it can be attractive if we use inverse or approximate inverse elements or if we can find a way to subtract the terms $1/4\ {\boldsymbol{E}_i}^2$ in the solve. One can use higher-order approximations (see [8]), but these do not offer real improvements.

2.6. The GS EBE preconditioner. The GS EBE preconditioner is based on the same decomposition as the EBE preconditioner. But instead of using a Crout factorization, we instead form the decomposition

$$\boldsymbol{E}_i = \boldsymbol{L}_i + \boldsymbol{L}_i^T, \tag{2.17}$$

where $\boldsymbol{L}_i$ is a strictly lower triangular matrix. The preconditioner is then

$$\boldsymbol{P}_{GS} = \boldsymbol{L}_M \prod_{i=1}^{p} (\boldsymbol{I} + \boldsymbol{L}_i) \prod_{i=p}^{1} (\boldsymbol{I} + \boldsymbol{L}_i^T) \boldsymbol{L}_M^T. \tag{2.18}$$

The advantage of this preconditioner is, obviously, that it is very easy to construct. In fact, if the matrix is initially scaled so that its diagonals are ones, the preconditioner need not be explicitly constructed. The principal drawback is that it is not exact, even when there is no overlap between element Hessians, because of the terms $\boldsymbol{L}_i \boldsymbol{L}_i^T$, which arise when approximating $\boldsymbol{I} + \sum_{i=1}^{p} \boldsymbol{E}_i$ by $\prod_{i=1}^{p} (\boldsymbol{I} + \boldsymbol{L}_i) \prod_{i=p}^{1} (\boldsymbol{I} + \boldsymbol{L}_i^T)$.

2.7. Definiteness of the preconditioner. The preconditioning matrix $\boldsymbol{P}$ for the conjugate gradient method should be positive definite. In the context of optimization problems, we have no guarantee that $\boldsymbol{H}$ and a fortiori $\boldsymbol{P}$ are positive definite. We describe here a simple strategy that guarantees that $\boldsymbol{P}$ is positive definite. Note that if the elementary matrices $\boldsymbol{H}_i^e$ are positive definite, $\boldsymbol{H}$ will be positive definite. We also note that when considering the EBE and EBE2 preconditioners, a sufficient condition for $\boldsymbol{P}$ to be positive definite is that all the $\boldsymbol{H}_i^e$, and hence the $\boldsymbol{I} + \boldsymbol{E}_i$, are as follows: if $\boldsymbol{H}_i^e$ positive definite for all i, so are $\boldsymbol{I} + \boldsymbol{E}_i$ and $\boldsymbol{I} + 1/2\ \boldsymbol{E}_i$. However, this is not necessary since the $\boldsymbol{I} + \boldsymbol{E}_i$ and $\boldsymbol{P}$ may be positive definite even if some $\boldsymbol{H}_i^e$ are not. Griewank and Toint [18] study conditions under which partially separable functions have convex decompositions, that is, those functions whose Hessian matrix is the sum of positive semidefinite element Hessians.

The preconditioners we have considered in this section all depend upon the decomposition of a collection of matrices $\boldsymbol{W}_i$, where $\boldsymbol{W}_i = \boldsymbol{H}_i$, $\boldsymbol{I} + \boldsymbol{E}_i$ or $\boldsymbol{I} + 1/2\ \boldsymbol{E}_i$ depending on the preconditioner. Thus it seems natural to consider the use of a modified Cholesky factorization for these decompositions to guarantee that the preconditioner is positive definite. We use the modified Cholesky factorization proposed by Schnabel and Eskow [34] that computes the Cholesky factorization of a matrix $\boldsymbol{W}_i$ if it is positive definite, or the Cholesky factorization of $\boldsymbol{W}_i + \boldsymbol{B}_i$, where $\boldsymbol{B}_i$ is a nonnegative diagonal matrix, otherwise. There is no need to know a priori if $\boldsymbol{W}_i$ is positive definite and the matrix $\boldsymbol{B}_i$ is determined during the factorization process. This modified Cholesky factorization exhibits marginally better properties in terms of computational costs and upper bound on $\|\boldsymbol{W}_i\|_\infty$ than those described by Gill and Murray [12], and Gill, Murray, and Wright [13].

The main drawback of such a strategy for forming the preconditioner is that we may perturb $\boldsymbol{W}_i$ even if the initial matrix is positive definite. Before attempting to form the preconditioner, we should first amalgamate elementary matrices whose sparsity structure lies completely within that of another element. We might also consider amalgamating indefinite elements with positive definite ones if the composite element is positive definite, even if this means introducing structural zeros, that is, zeros within the element. The amalgamation of an indefinite element might, for example, be made with the elementary matrix having the largest intersection. While we have not developed a fail-safe scheme here, the simple strategy just presented appears to work well in practice.

2.8. Experiments with element-by-element preconditioners. A number of these preconditioners are known to work well in practice when applied to classes of problems arising from partial differential equations. In this section, we aim to investigate whether these preconditioners are effective in the more general context of systems which arise from partially separable optimization applications. Although our experiments are far from exhaustive, they do lead to interesting conclusions and are helpful in deciding future directions of research.

Preliminary experiments were performed by Daydé, L'Excellent, and Gould [8] on structured matrices for which the overlap between successive elements was varied. It appeared that element-by-element preconditioners, and particularly EBE, were more effective than diagonal preconditioning as long as there was low overlap between elements. In this case the clustering of eigenvalues of the preconditioned problem was seen to be significantly better than for the unpreconditioned and diagonally preconditioned methods, especially when the problems were ill conditioned.

We now aim to consider real matrices which arise from both PDE and optimization applications.

2.8.1. Set of test matrices. The matrices come either from the Harwell–Boeing collection (CEGB2802, MAN5976, LOCK3491) (see [10]), or are problems in SIF format from the CUTE collection (see [4]). In the case of the Harwell–Boeing, we removed all rows and columns which corresponded to unused variables. Furthermore, we have used random numerical values because they were originally not present. Since the choice of these values influences the conditioning of the matrices, and since this conditioning is relevant to the performance of our solution techniques, we have created two instances of CEGB2802 with significantly differing spectra.

The patterns of CEGB2802 and LOCK3491 arise from structural engineering problems; MAN5976 comes from deformation problems; MAT32 and MAT33 are finite element matrices generated with the SPARSKIT software (see [33]), TORSION1 and NOBNDTOR are quadratic elastic torsion problems arising from an obstacle problem on a square, NET3 is a very ill-conditioned example which arises from the optimization of a high pressure gas network, and CBRATU3D is obtained by discretizing a complex 3D PDE problem in a cubic region. BDEXP, BROYDN7D, and SINQUAD are artificial problems, BDEXP involving a band of exponential terms, BROYDN7D having seven bands, and SINQUAD featuring a nonbanded sparsity pattern. SEMICON2 is a discretized semiconductor problem, while ZIGZAG is a nonlinear optimal control problem with both state and control constraints. SPMSQRT is a tridiagonal matrix square root problem, and CLPLATEB comes from the discretization of the clamped plate problem. Finally, HYDROELL is a hydroelectric reservoir management problem, while GAUSSELM is the problem of maximizing growth in Gaussian elimination with complete pivoting.

A summary of each problem characteristics is given Table 2.1, where n is the order of the matrix, p is the number of elements, and κ is the condition number. The degree of overlap is the average number of elements containing each variable, that is, the sum of the element dimensions divided by the order of the matrix.

2.8.2. Results on test matrices. We ran the preconditioned conjugate gradient method (see, for example, [14]) to solve the system $\boldsymbol{Hx} = \boldsymbol{b}$, starting with the estimate $\boldsymbol{x} = 0$, and stopping as soon as $\|\boldsymbol{Hx} - \boldsymbol{b}\|_2 < 10^{-9}\|\boldsymbol{b}\|_2$. The right-hand side $\boldsymbol{b}$ was either $(1, 1, \ldots, 1)^T$, or problem dependent. All of the experiments reported in this paper were performed in double precision on a single processor of an Alliant FX/80, with vectorization of the inner loops of the solves and the matrix–vector products.

The results of the execution of the preconditioned conjugate gradient algorithm obtained for different preconditioners on our test problems are given in Tables 2.2 and 2.3, where t_{prec} is the time to compute the preconditioner, $\#its$ is the number of iterations, t_{conv} is the time for convergence, and t_{it} is the time per iteration.

Table 2.1
Summary of the characteristics of each test problem.

Problem name	n	p	Min element size	Max element size	Mean element size	Degree of overlap	κ
CEGB2802	2694	108	42	60	58.7	2.4	2.5×10^2
CEGB2802	2694	108	42	60	58.7	2.4	5.7×10^4
MAN5976	5882	785	20	20	20.0	2.7	5.0×10^1
LOCK3491	3416	684	6	24	19.8	4.0	1.3×10^2
MAT32	57	157	1	3	2.2	6.0	1.3×10^1
MAT33	637	273	1	3	2.6	6.0	5.5×10^1
BIGGSB1	998	1001	0	2	2.0	2.0	4.0×10^5
TORSION1	3360	3792	1	5	4.4	5.0	6.7×10^0
NOBNDTOR	480	562	1	5	4.2	4.9	1.8×10^2
CBRATU3D	4934	4934	5	8	7.5	7.5	3.4×10^1
NET3	512	531	1	6	2.6	2.7	2.4×10^9
BDEXP	5000	4998	3	3	3.0	3.0	2.8×10^2
SEMICON2	1000	1000	2	3	3.0	3.0	2.3×10^7
BROYDN7D	10000	15000	2	3	2.7	4.0	1.6×10^1
ZIGZAG	465	600	1	3	2.3	2.9	3.1×10^5
SPMSQRT	1000	1664	2	5	3.0	5.0	4.2×10^2
CLPLATEB	4970	19601	1	71	2.0	7.9	1.6×10^4
SINQUAD	5000	5000	1	3	3.0	3.0	4.6×10^8
HYDROELL	1008	1009	1	215	1.44	1.44	1.9×10^8
GAUSSELM	1128	1136	1	5	3.41	3.44	8.8×10^6

We observe that, unlike the structured matrices of Daydé, L'Excellent, and Gould [8], element-by-element preconditioners do not appear to be significantly more effective than diagonal preconditioning on many of the current test examples. However, we notice that as we might hope, these preconditioners often appear to be more effective than their diagonal counterparts for the ill-conditioned problems, even if the time per iteration for the former is invariably larger than for the latter.

2.8.3. Remarks on the storage and the vectorization. Thus far, we have reported on experiments where packed storage is used for element matrices, that is, where we store only the lower triangular part of each element. In Table 2.4, we compare the time per iteration for runs using the matrix CEGB2802 in which we consider both packed or full storage of the element matrices. We performed the tests on a variety of preconditioners and consider the performance both with and without vectorization. Only the inner loops were vectorized, with vectors of length no larger than 60.

It appears that the full storage is more efficient for vectorized conjugate gradients without, or with diagonal, preconditioning and less efficient for preconditioners involving triangular solves. This is due to the relative efficiency of the computational kernels used both for the matrix–vector products and the triangular solves. An interesting strategy—if extra workspace were available—would be to use full storage for the Hessian, because of the beneficial effect on matrix–vector products, and packed storage for the preconditioners, because of the advantages this gives for triangular solves.

The gain due to vectorization is between 2 and 5, depending on the preconditioner and the storage applied. We would expect to improve these ratios, for example, with longer vectors or more powerful vector computers.

Instead of using factorizations of the $\boldsymbol{W}_i$, it might be better to form explicit inverses, as the sequence of solves may then be replaced by a sequence of matrix–vector products. This has some advantage in terms of using tuned BLAS routines

Table 2.2
Results for our test problems (I).

Problem name	Precondi-tioner	t_{prec}	$\#its$	t_{conv}	t_{it}
BIGGSB1	NONE	0.001	499	33.217	0.066
	DIAG	0.028	499	33.352	0.067
$\kappa = 4.04 \times 10^5$	EBE	0.238	333	74.831	0.224
	EBE2	0.221	328	121.876	0.370
	GSEBE	0.101	334	74.637	0.223
	EMF	0.189	4	0.408	0.082
	FEP	0.454	4	0.418	0.084
CBRATU3D	NONE	0.001	53	33.640	0.623
	DIAG	0.193	53	33.740	0.625
$\kappa = 3.4 \times 10^1$	EBE	5.666	20	39.851	1.898
	EBE2	5.396	20	66.153	3.150
	GSEBE	1.219	19	37.840	1.892
	EMF	12.244	60	71.683	1.175
	FEP	11.146	35	42.481	1.180
CEGB2802	NONE	0.001	121	28.858	0.237
	DIAG	0.033	35	8.550	0.237
$\kappa = 2.5 \times 10^2$	EBE	6.249	12	7.617	0.586
	EBE2	6.100	11	10.838	0.903
	GSEBE	0.549	17	10.498	0.583
	EMF	26.265	47	67.783	1.412
	FEP	25.749	26	38.203	1.415
CEGB2802	NONE	0.001	2032	485.604	0.238
	DIAG	0.033	657	156.278	0.238
$\kappa = 5.7 \times 10^4$	EBE	6.239	120	70.931	0.586
	EBE2	6.098	145	131.965	0.904
	GSEBE	0.550	329	192.571	0.584
	EMF	26.258	708	1001.524	1.413
	FEP	25.749	575	815.452	1.416
LOCK3491	NONE	0.001	68	19.995	0.290
	DIAG	0.068	24	7.256	0.290
$\kappa = 1.3 \times 10^2$	EBE	4.597	9	7.809	0.781
	EBE2	4.442	9	12.505	1.251
	GSEBE	0.659	11	9.318	0.776
	EMF	11.674	62	61.764	0.980
	FEP	11.263	23	23.605	0.984
MAN5976	NONE	0.001	68	23.276	0.337
	DIAG	0.084	28	9.826	0.339
$\kappa = 5.0 \times 10^1$	EBE	5.098	10	9.892	0.899
	EBE2	4.925	11	17.201	1.433
	GSEBE	0.790	12	11.587	0.891
	EMF	12.207	43	57.724	1.312
	FEP	11.698	25	34.276	1.318
MAT32	NONE	0.001	21	0.242	0.011
	DIAG	0.004	21	0.241	0.011
$\kappa = 1.3 \times 10^1$	EBE	0.040	14	0.552	0.037
	EBE2	0.036	12	0.781	0.060
	GSEBE	0.016	15	0.579	0.036
	EMF	0.049	23	0.298	0.012
	FEP	0.061	18	0.244	0.013
MAT33	NONE	0.001	48	2.221	0.045
	DIAG	0.017	48	2.227	0.045
$\kappa = 5.5 \times 10^1$	EBE	0.193	28	4.460	0.154
	EBE2	0.178	24	6.439	0.258
	GSEBE	0.067	30	4.755	0.153
	EMF	0.236	438	23.140	0.053
	FEP	0.342	110	5.866	0.053
NET3	NONE	0.001	3739	156.287	0.0418
	DIAG	0.015	1558	61.930	0.040
$\kappa = 2.4 \times 10^9$	EBE	0.174	694	91.401	0.132
	EBE2	0.162	560	122.796	0.219
	GSEBE	0.063	731	95.894	0.131
	EMF	0.330	731	106.298	0.145
	FEP	0.325	731	106.910	0.146
NOBNDTOR	NONE	0.001	68	3.621	0.052
	DIAG	0.019	68	3.609	0.052
$\kappa = 1.8 \times 10^2$	EBE	0.297	30	5.216	0.168
	EBE2	0.279	33	9.541	0.281
	GSEBE	0.089	31	5.365	0.168
	EMF	0.567	28	1.932	0.067
	FEP	0.525	27	1.874	0.067

Table 2.3
Results for our test problems (II).

Problem name	Precondi-tioner	t_{prec}	$\#its$	t_{conv}	t_{it}
TORSION1	NONE	0.001	24	9.246	0.370
	DIAG	0.130	25	9.627	0.370
$\kappa = 6.7 \times 10^0$	EBE	2.072	9	11.769	1.177
	EBE2	1.942	11	23.497	1.958
	GSEBE	0.647	10	12.893	1.172
	EMF	2.027	13	6.324	0.452
	FEP	1.835	11	5.446	0.454
BDEXP	NONE	0.001	38	15.182	0.389
	DIAG	0.145	34	13.604	0.389
$\kappa = 2.8 \times 10^2$	EBE	0.556	11	15.684	1.307
	EBE2	0.361	13	30.734	2.195
	GSEBE	0.205	13	18.450	1.318
	EMF	2.826	28	14.871	0.513
	FEP	2.883	19	10.267	0.513
BROYDN7D	NONE	0.001	24	27.574	1.103
	DIAG	0.411	21	24.290	1.104
$\kappa = 1.6 \times 10^1$	EBE	1.464	8	33.833	3.759
	EBE2	0.970	9	62.855	6.286
	GSEBE	0.572	9	37.786	3.779
	EMF	7.712	11	16.786	1.399
	FEP	7.739	9	14.025	1.403
CLPLATEB	NONE	0.001	376	488.230	1.295
	DIAG	0.477	382	494.302	1.291
$\kappa = 1.6 \times 10^4$	EBE	1.445	136	608.875	4.444
	EBE2	0.965	161	1204.206	7.433
	GSEBE	0.641	135	606.905	4.463
	EMF	6.699	124	177.545	1.420
	FEP	6.602	123	175.707	1.417
SINQUAD	NONE	0.001	4	1.978	0.396
	DIAG	0.147	4	1.973	0.395
$\kappa = 4.6 \times 10^8$	EBE	0.562	4	6.615	1.323
	EBE2	0.365	8	19.869	2.208
	GSEBE	0.206	5	7.986	1.331
	EMF	41.258	6	3.625	0.518
	FEP	41.303	4	2.593	0.519
HYDROELL	NONE	0.001	167	13.748	0.082
	DIAG	0.026	109	9.006	0.082
$\kappa = 1.9 \times 10^8$	EBE	0.223	51	13.471	0.259
	EBE2	0.188	54	23.139	0.421
	GSEBE	0.058	48	12.693	0.259
	EMF	0.377	54	5.079	0.092
	FEP	0.353	35	3.331	0.093
GAUSSELM	NONE	0.001	2000	195.908	0.098
	DIAG	0.035	879	86.256	0.098
$\kappa = 8.8 \times 10^6$	EBE	0.141	404	129.222	0.319
	EBE2	0.098	359	191.194	0.531
	GSEBE	0.049	405	130.311	0.321
	EMF	1.525	405	420.053	1.035
	FEP	1.542	405	421.159	1.037
ZIGZAG	NONE	0.001	719	32.728	0.045
	DIAG	0.016	439	19.972	0.045
$\kappa = 3.1 \times 10^5$	EBE	0.056	151	22.244	0.146
	EBE2	0.035	178	43.333	0.242
	GSEBE	0.022	156	23.096	0.147
	EMF	0.232	156	172.603	1.099
	FEP	0.239	156	172.714	1.100
SPMSQRT	NONE	0.001	173	23.581	0.136
	DIAG	0.046	128	17.490	0.136
$\kappa = 4.2 \times 10^2$	EBE	0.209	44	20.003	0.445
	EBE2	0.153	45	34.024	0.740
	GSEBE	0.067	46	20.984	0.446
	EMF	0.871	137	23.708	0.172
	FEP	0.911	115	19.918	0.172

to form the products. For EBE2, this could be especially useful since two triangular solves would be replaced by only one matrix–vector product. Unfortunately, the elements in our examples are typically small, and we did not observe any benefit from such a strategy.

TABLE 2.4
Time per iteration in CEGB2802.

Preconditioner used	Scalar mode Packed	Scalar mode Full	Vector mode Packed	Vector mode Full
NONE	0.613	1.02	0.237	0.214
DIAG	0.618	1.03	0.237	0.217
EBE	1.52	1.98	0.586	0.594
EBE2	2.40	2.92	0.903	0.970
GS EBE	1.52	1.97	0.583	0.591

2.9. Conclusions on the use of element-by-element preconditioners. The results of the previous section indicate that element-by-element preconditioners are effective, in terms of the numbers of iterations required and the clustering of eigenvalues of the preconditioned Hessian, particularly if the overlap between blocks is small. EBE seems to be the best of our block preconditioners and it does not require any assembly of the matrix. EMF and FEP do not require an assembly of the matrix either, but the resulting triangular incomplete factors need to be partially assembled, which can make each solve rather costly for large matrices. Their numerical properties depend a lot on the problem, being sometimes better but more often worse than other preconditioners.

A disadvantage of EBE2 and GS EBE is that the terms $1/4\ \boldsymbol{E}_i^2$ and $\boldsymbol{L}_i\boldsymbol{W}^{-1}\boldsymbol{L}_i^T$ may give rise to poor approximations even when there is little overlap between elements. If there is significant overlap, the efficiency of EBE2 and GS EBE is close to that observed for EBE. Usually, the number of iterations required by EBE is smaller than that for EBE2 which is, in turn, smaller than that for GS EBE; the only case where this is not so is for problems with very large overlap. As GS EBE is the easiest preconditioner to construct, it may be beneficial to use GS EBE when we do not need much accuracy in the solution of our system, as, for example, is common in the early stages of optimization calculations. However, in more general cases, we prefer EBE.

We have not found it useful to replace the triangular solves by products of inverses. To obtain some benefit from this, we would need larger elements so as to exploit better vectorization/parallelization.

A difficulty for general matrices is that we have no a priori information on the sizes of the elements. On finite element problems, typically all elements have the same size. We may thus *color* the elements, that is, partition the complete set of elements into subsets, or colors, of independent (nonoverlapping) elements, to encourage both vectorization and parallelization. In our case, vectorization is restricted to the handling of each individual element, so is not usually very effective. A potentially better strategy would be to vectorize at a coarser level, treating blocks of elements of the same size within each color together.

In our experiments, except for ill-conditioned problems, EBE is not significantly more efficient than diagonal preconditioning. We believe that this is true for three reasons. The first is because of the structure of the elements. When there is lower overlap, EBE appears much more efficient than diagonal preconditioning. Amalgamating elements may reduce the number of iterations by decreasing the degree of overlap in the new partition. Second, the vectorization here is not as efficient as it could be. If we knew a priori that all blocks have the same size, it would be possible to vectorize the solve more efficiently, as was reported in previous experiments by, for example, Erhel, Traynard, and Vidrascu [11]. Third, no coloring, parallelization, and specific code optimizations have yet been carried out.

Finally, we believe that further study on real problems with less regular matrix structures will inevitably lead to a better understanding of the classes of problems for which each of the preconditioners we have considered is particularly appropriate.

3. Remarks on the importance of element regrouping.

Let $f(\boldsymbol{x})$ be of the form (1.2). Clearly, the decomposition (1.2) may not be unique, and different decompositions may significantly affect the performance of the preconditioners considered in this paper. In some of the experiments reported in the previous chapter, the limitation in performance of the conjugate gradient method, both for vectorization and parallelization, is due to a nonoptimal choice of the decomposition. Indeed, frequently the local variable set for one element may be completely contained within another and it would pay to merge the two elements into a single superelement or *group*. More generally, the local variable sets for two elements may significantly overlap and again it may be advantageous to merge the elements into a single group.

Conn, Gould, and Toint [7] have considered this problem from the point of view of improving the matrix–vector products at the heart of many iterative methods for minimizing partially separable functions. In this section our scope is larger in that we hope to produce more effective preconditioners by amalgamating similar elements.

The goal of our amalgamation algorithm process is to merge elements which involve many common variables. This will not only decrease the amount of work per iteration but should also improve the quality of the element-by-element preconditioner. A secondary target, which we have not considered here, might be to ensure that the merged elements are of similar sizes so that a coloring of the elements will provide significant gains from vectorization and parallelization (see our comments in section 2.9).

It is clear that determining an optimal partitioning of the elements into groups may be costly, or even impossible. Frequently, the construction of an initial decomposition of f into elements by a user depends more on considerations on the ease of expressing the function and its derivatives, rather than considerations of computational performance. It is our experience with many of the test examples in the CUTE package (see [4]), for example, that the overlap between elements is high (and even that some elements are entirely subsumed by other elements). Therefore, we regard it to be crucial for performance to determine a heuristic to partition the original sets of elements into computationally attractive disjoint groups.

3.1. Goals of the regrouping technique.

Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$, $\boldsymbol{x}^i$ be the set of local variables involved in the elementary function f_i, for $1 \le i \le p$, and $\mathcal{V}_i$ be the set of indices of $\boldsymbol{x}^i$. For example, if

$$f(x_1, x_2, x_3) = f_1(x_1, x_2) + f_2(x_2, x_3),$$

we have

$$\boldsymbol{x}^1 = \{x_1, x_2\}, \boldsymbol{x}^2 = \{x_2, x_3\}, \mathcal{V}_1 = \{1, 2\} \quad \text{and} \quad \mathcal{V}_2 = \{2, 3\}.$$

Our aim is to partition the index set of elements, $\mathcal{P} \overset{\text{def}}{=} \{1, \ldots p\}$, into a collection of *groups* $\{\mathcal{G}_k\}$ such that

1. $\bigcup \mathcal{G}_k = \mathcal{P}$;
2. $\mathcal{G}_k \bigcap \mathcal{G}_l = \emptyset$ for $k \neq l$;
3. if elements i and $j \in \mathcal{G}_k$, either $\mathcal{V}_i \setminus \mathcal{V}_j$ or $\mathcal{V}_j \setminus \mathcal{V}_i$ is small;
4. if element $i \in \mathcal{G}_k$ and element $j \in \mathcal{G}_l$, $(k \neq l)$, both $\mathcal{V}_i \setminus \mathcal{V}_j$ and $\mathcal{V}_j \setminus \mathcal{V}_i$ are large.

Thus, each group aims to collect elements whose overlap is large, while keeping the overlap between groups small.

Once the groups have been determined, all of the elements indexed by a single group will be summed to form a *superelement.* Thereafter, the algorithms described in section 2 of this paper should be applied to the sum of superelements,

$$f(\boldsymbol{x}) = \sum_k s_k(\boldsymbol{x}), \quad \text{where} \quad s_k = \sum_{i \in \mathcal{G}_k} f_i(\boldsymbol{x}^i). \tag{3.1}$$

3.2. Possible amalgamation algorithms. A natural approach, frequently used in sparse linear algebra (see [9] and [1]), is to amalgamate two elements into a group if their overlap is large. The main difficulty is to define a suitable heuristic to control the amalgamation process.

We start by assigning the index of each element to its own group. We call such a group an *elementary* group and denote the group as $\mathcal{G}_i = \{i\}$, $1 \leq i \leq p$. The algorithm proceeds by merging groups until a satisfactory partitioning has been determined. Once a group comprises two or more elements, it will be called an *amalgamated* group. By convention, if we merge groups $\mathcal{G}_i$ and $\mathcal{G}_j$ with $i < j$, we replace $\mathcal{G}_i$ by $\mathcal{G}_i \bigcup \mathcal{G}_j$ and delete $\mathcal{G}_j$. We denote the set of indices of variables used by elements in group $\mathcal{G}_k$ by $\mathcal{V}_k$.

A simple amalgamation algorithm is as follows. The *fill-in* between groups i and j is defined to be twice the product of the cardinalities of the sets $\mathcal{V}_i \setminus \mathcal{V}_j$ and $\mathcal{V}_j \setminus \mathcal{V}_i$; this is in fact the number of extra (zero) entries introduced if the two group Hessians are amalgamated. Of course, this fill-in should be limited to reasonable value, so we let h_{max} be the maximum fill-in value allowed when amalgamating two groups. Then we amalgamate pairs of groups for which the fill-in is smallest, until any remaining amalgamation would produce a fill-in value larger than h_{max}. Ties are broken arbitrarily.

Another approach, suggested by Conn, Gould, and Toint [7] considers the number of flops gained in the matrix–vector product. A list of elements is built for each variable and two elements are amalgamated if it leads to decrease the number of floating point operations in the matrix–vector product.

In this paper, we have chosen to concentrate on an algorithm that aims to decrease the time spent in matrix–vector products and triangular solves on the target architecture. Our overall goal is to reduce the cost of a (preconditioned) conjugate gradient iteration. As the dominant cost of such an iteration is in solving a system involving the preconditioner and in forming a matrix–vector product, it is of interest to reduce these costs.

3.3. An amalgamation algorithm. As a precursor to the main amalgamation algorithm, we apply the following scheme to merge elements which are completely subsumed within others and to construct an *amalgamation* graph. The amalgamation graph has as its nodes the groups and has arcs between the nodes for which there is some benefit from amalgamation. The arcs have associated weights $b(\mathcal{G}_i, \mathcal{G}_j)$ which indicate the possible benefit to be obtained by merging nodes.

```
Initial phase (suppress complete inclusions and construct the amalgamation graph)
For each pair of groups $\mathcal{G}_i$ and $\mathcal{G}_j$, $i < j$, such that $\mathcal{V}_i \cap \mathcal{V}_j$ is not zero,
If $\mathcal{V}_i \setminus \mathcal{V}_j$ or $\mathcal{V}_j \setminus \mathcal{V}_i$ is zero,
    Amalgamate these groups.
Else
    Compute the benefit $b(\mathcal{G}_i, \mathcal{G}_j)$
```

```
        Add the arc ($\mathcal{G}_i,\ \mathcal{G}_j$) with weight $b(\mathcal{G}_i, \mathcal{G}_j)$ to the graph
End if
```

We then apply the main amalgamation algorithm.

Main amalgamation algorithm (simplified)

```
While there exists an arc for which $b$ is positive
    Find the arc $\mathcal{G}_i,\ \mathcal{G}_j,\ b(\mathcal{G}_i, \mathcal{G}_j)$ of the graph for which $b$ is maximum
    Amalgamate the groups $\mathcal{G}_i$ and $\mathcal{G}_j$
    Update all the arcs of the graph incident on $\mathcal{G}_i$ or $\mathcal{G}_j$
End while
End of the algorithm. The new partitioning into groups has been obtained.
```

3.4. Computing the benefit. The choice of amalgamating two elements is based on the estimated gain of CPU time or *benefit* of this amalgamation which is defined as

$$b(\mathcal{G}_i, \mathcal{G}_j) = t(card(\mathcal{V}_i)) + t(card(\mathcal{V}_j)) - t\left(card\left(\mathcal{V}_i \bigcup \mathcal{V}_j\right)\right), \tag{3.2}$$

where $t(i)$ is the estimated time to treat an element of order i.

The most costly operations in a preconditioned conjugate gradient iteration are

(i) p elemental size matrix–vector products, and
(ii) q elemental size triangular solves,

where p is the number of elements, and q is

(i) zero for no or diagonal preconditioning,
(ii) two times p for EBE and GS EBE preconditioning, and
(iii) four times p for EBE2 preconditioning.

In our current experiments, we shall only consider diagonal and EBE preconditioning, as these proved to be the most effective of the preconditioners that we investigated in section 2.8. Thus we have two different possibilities:

1. Minimize the time spent in matrix–vector products. In what follows, this variant will be called **amalg1**. It should be optimal for conjugate gradients without preconditioner or with diagonal preconditioning. In this case, $t(i)$ is an estimate of the time spent to perform a matrix–vector product of order i on the considered architecture. We consider the case where the element matrix is symmetric, stored in packed symmetric storage, and the accesses to the vectors are indirect.

2. Optimize the ordering for use of the EBE (or GS EBE) preconditioner. This variant is called **amalg2**. The time estimate $t(i)$ is now the time spent in a matrix–vector product plus twice the time of a triangular solve with indirect accesses to the right-hand side and the solution.

Note that the time spent in constructing the preconditioner is not taken into account in these costs, as this is an overhead for the whole conjugate gradient process, not just for an individual iteration. The time spent in diagonal products—the matrix–vector products for the diagonal preconditioner or the three products associated with the EBE preconditioner—is not taken into account either, but such products are independent of the ordering.

The times of matrix–vector products and triangular solves for all realistic values of i are computed once and stored in a pair of data files. The values $t(i)$ are read from these files as required and stored in a real array. The benefit $b(\mathcal{G}_i, \mathcal{G}_j)$ can then easily be computed from (3.2). Throughout the algorithm, estimates of the total benefit along with the total time for matrix–vector products and triangular solves before amalgamation are recorded.

3.5. Preliminary tests. We report in Table 3.5 the results of first running the amalgamation algorithm and then using diagonal and EBE preconditioned conjugate gradients to solve the linear system in question. For each test problem, we report the amalgamation time, the number of elements (p), the average size of each element, the number of iterations ($\#its$), the time to construct the preconditioner (t_{prec}), and the time for convergence (t_{conv}). $\uplus$ represents the amalgamation strategy: 0 for no amalgamation, 1 for **amalg1**, and 2 for **amalg2**.

Our implementation is principally in Fortran but the amalgamation graph manipulation is coded in C. All the times reported in this section are in seconds.

It is clear from this table that amalgamation can be very effective. There are always large gains in convergence times, both for diagonal and EBE preconditioners. As we would have expected, it appears that, in most cases, the diagonal preconditioned method is faster with **amalg1** and the EBE is faster with **amalg2**. For some other problems, such as CEGB2802, however, amalgamation is not so effective as the function is already well decomposed. For some problems, **amalg2** is less effective than **amalg1** with an EBE preconditioner, which shows the limitations of our heuristic. Such limitations likely arise because of the indirect addressing of data and the fact that we do not know a priori how the data are accessed in the memory and hence the experiments do not fit exactly with an idealized memory-access model. Another limitation is that the eigenvalue-clustering quality of the preconditioner may happen to be worse with **amalg2** than **amalg1**. This happens, for example, for the test problem MAT33.

While amalgamating does not normally affect the quality of the diagonal preconditioner (i.e., the number of iterations), it improves significantly the numerical behavior of the EBE-preconditioned conjugate gradients method. However, when considering, for example, the test problem NET3, we observe a variation of the number of iterations with diagonal preconditioning. This is entirely because the order of floating point operations is altered by the amalgamation, and different rounding properties come into effect (see [21]) when computing the diagonal preconditioner and matrix–vector products. When considering the EBE preconditioned method for NET3, we observe that the convergence time is significantly faster with **amalg2** than with the original matrix. This may be attributed to the decrease ($694 \rightarrow 145$) in the number of iterations due to a better preconditioner and to the reduction in time (0.13 s $\rightarrow$ 0.034 s) per iteration.

As far as we are concerned, the most important thing to note is that in certain cases, with the best amalgamation strategy, EBE preconditioning is much more efficient than diagonal preconditioning (e.g., 5 versus 21 seconds for NET3, 9.5 versus 70 seconds for SEMICON2, 1.9 versus 7.5 seconds for HYDROELL). Furthermore, even for problems where diagonal behaves better than EBE, the time for convergence is never more than 20% larger for EBE.

3.6. Cost of the algorithm. The times spent in the amalgamation procedure are reported in the second and third columns of Table 3.5. The second column gives the times required to perform the amalgamation and set up data structures for the resulting factors, while the third column shows the additional time required to insert the numerical values into the resulting factors. The algorithm is sequential and has not been optimized.

We consider the reported times for the amalgamation procedure to be significant. Taking these times into account, it would appear that applying the amalgamation algorithm is not always beneficial, particularly if we wish to solve a single linear

TABLE 3.5

Comparison of the preconditioned conjugated gradients applied to the original matrices and the matrices with amalgamation strategies 1 *and* 2 *on* 1 *processor of the Alliant FX/*80*. Sym and num refer to the time taken to perform the amalgamation and set up data structures for the resulting factors, and to the time required to insert the numerical values into the resulting factors, respectively.*

Problem name	$\uplus$	Amalgamation time		p	Mean elt size	Diagonal			EBE		
		sym	num			$\#its$	t_{prec}	t_{conv}	$\#its$	t_{prec}	t_{conv}
NET3	0			538	2.6	1558	0.015	61.1	694	0.17	91.1
	1	1.4	0.12	50	12.2	1664	0.0062	21.9	172	0.13	6.0
$\kappa = 2.4 \times 10^9$	2	1.4	0.13	41	14.4	1601	0.0049	21.2	145	0.14	5.0
BIGGSB1	0			1001	2.0	499	0.027	33.0	333	0.24	74.7
	1	1.52	0.17	124	9.0	499	0.011	11.8	223	0.17	14.6
$\kappa = 4.0 \times 10^5$	2	1.53	0.18	84	12.9	499	0.011	11.4	160	0.18	9.7
TORSION1	0			3792	4.4	25	0.13	9.6	9	2.1	11.8
	1	21.7	1.7	246	23.8	25	0.036	3.7	10	1.4	4.0
$\kappa = 6.7 \times 10^0$	2	23.1	1.8	207	27.1	25	0.036	3.7	10	1.4	4.0
MAT32	0			157	2.2	21	0.0045	0.23	14	0.040	0.54
	1	0.29	0.033	4	16.8	21	0.0011	0.038	10	0.016	0.045
$\kappa = 1.3 \times 10^1$	2	0.29	0.036	3	21.3	21	0.0010	0.038	10	0.016	0.044
MAT33	0			637	2.6	48	0.018	2.20	28	0.19	4.4
	1	2.31	0.15	21	18.7	48	0.0029	0.43	22	0.092	0.51
$\kappa = 5.5 \times 10^1$	2	2.18	0.16	18	21.2	48	0.0028	0.43	23	0.091	0.53
CEGB2802	0			108	58.7	35	0.033	8.56	12	6.3	7.7
	1	0.47	1.9	106	59.4	35	0.033	8.6	12	6.3	7.6
$\kappa = 2.5 \times 10^2$	2	0.48	1.9	102	60.8	35	0.036	8.6	12	6.3	7.6
CEGB2802	0			108	58.7	657	0.033	156.1	120	6.2	71.1
	1	0.47	1.9	106	59.4	659	0.034	157.8	111	6.3	66.0
$\kappa = 5.7 \times 10^4$	2	0.48	1.95	102	60.8	660	0.035	158.1	112	6.3	66.4
CBRATU3D	0			4394	7.54	53	0.19	33.8	20	5.6	40.0
	1	51.3	3.7	886	24.4	53	0.10	26.5	21	7.3	28.6
$\kappa = 3.4 \times 10^1$	2	61.1	3.9	713	27.9	53	0.094	26.0	21	7.6	27.7
NOBNDTOR	0			562	4.2	68	0.021	3.6	30	0.30	5.2
	1	2.47	0.23	40	22.1	68	0.0054	1.3	30	0.20	1.5
$\kappa = 1.8 \times 10^2$	2	2.63	0.24	33	25.0	68	0.0069	1.3	29	0.20	1.5
BDEXP	0			4998	3.0	34	0.15	13.8	11	1.9	15.6
	1	17.4	1.4	313	18.0	34	0.042	4.8	8	1.2	3.1
$\kappa = 2.8 \times 10^1$	2	16.8	1.4	313	18.0	34	0.043	4.7	8	1.2	3.0
BROYDN7D	0			15000	2.7	21	0.42	24.1	8	4.9	33.3
	1	123.3	3.8	625	20.0	21	0.091	6.7	8	3.2	6.9
$\kappa = 1.6 \times 10^1$	2	121.0	3.8	625	20.0	21	0.090	6.7	7	3.3	6.1
CLPLATEB	0			19601	2.0	382	0.48	473.2	136	4.3	589.3
	1	137.6	3.7	414	18.7	382	0.052	69.5	146	1.6	67.9
$\kappa = 1.6 \times 10^4$	2	135.2	3.7	367	19.3	382	0.049	65.8	131	1.6	56.9
GAUSSELM	0			1136	3.4	879	0.036	86.2	404	0.51	127.2
	1	58.0	0.31	132	14.5	879	0.014	37.7	378	0.51	41.7
$\kappa = 8.8 \times 10^6$	2	58.0	0.33	98	18.8	879	0.011	38.0	377	0.58	40.8
HYDROELL	0			1009	1.4	109	0.028	9.0	51	0.33	13.2
	1	0.35	0.44	794	1.3	109	0.023	7.5	8	0.28	1.9
$\kappa = 1.9 \times 10^8$	2	0.34	0.44	794	1.3	109	0.023	7.5	8	0.29	1.9
SEMICON2	0			1000	3.0	2444	0.030	197.7	739	0.38	193.7
	1	2.1	0.28	63	17.8	2443	0.0081	69.5	189	0.23	12.8
$\kappa = 2.3 \times 10^7$	2	2.0	0.27	63	17.8	2466	0.0082	70.0	141	0.23	9.5
SPMSQRT	0			1664	3.0	128	0.048	17.1	44	0.64	19.4
	1	3.9	0.47	56	20.8	128	0.0082	3.9	27	0.27	2.0
$\kappa = 4.2 \times 10^2$	2	4.0	0.47	56	20.8	128	0.0082	3.9	27	0.27	2.0
ZIGZAG	0			600	2.3	439	0.018	19.7	151	0.17	21.7
	1	1.4	0.12	38	16.0	439	0.0045	7.4	138	0.13	5.2
$\kappa = 3.1 \times 10^5$	2	1.5	0.14	26	22.6	439	0.0056	7.6	115	0.15	4.4

system and if the number of groups is large (see CBRATU3D in Table 2.3). If, however, we have to solve a large number of problems with the same matrix, or which have the same elemental structure—such as might occur in a nonlinear optimization or PDE application—amalgamating the elements is essential.

3.7. Conclusions on the use of element amalgamation. In our examples, we have observed that most of the benefit of amalgamations is due to the main amalgamation algorithm. However, it seems natural to start the algorithm with an initial phase in which complete inclusions are suppressed, because it reduces the number of elements without introducing fill-in and obviously saves space and floating point operations.

The implementation of our algorithm is not optimal, but the results obtained are very promising. The cost of the amalgamation procedure is currently very costly, but it is hoped that good heuristics will decrease this preprocessing cost with roughly the same effect. As we may have to solve many systems with the same structure in the course of a nonlinear optimization calculation, a good preprocessing step may pay handsome dividends in the longer run.

The resulting numerical quality of the preconditioner seems to be difficult to appreciate in general. All that we may hope is that reducing the degree of overlap may also decrease the number of iterations. But we note that even if the number of iterations following the amalgamation process is not significantly reduced, the times for the calculation of the preconditioner and for obtaining the solution often decrease. There appears to be a significant advantage in using an EBE rather than a diagonal preconditioner for ill-conditioned problems.

The ratio of EBE to diagonal preconditioner solution times decreases as the amalgamation is applied, both because typically fewer iterations are required following the amalgamation and because an increase in element sizes encourages efficient vectorization for EBE and matrix–vector products—diagonal preconditioning is already completely vectorizable.

4. Final comments.

We have shown that element-by-element preconditioners and amalgamation techniques may be extremely effective for systems of equations which arise in partially separable nonlinear optimization applications, especially for large scale ill-conditioned problems.

Furthermore, element-by-element preconditioners seem to offer great possibilities of vectorization/parallelization on multiprocessor architectures. The next step will be to include such a scheme within a nonlinear optimizer.

It is clear that preprocessing should be applied to any large-scale optimization problem, and that in our case it is important to amalgamate elements and find a suitable coloring for later calculations. This is a difficult task as many criteria need to be taken into account. This preprocessing may well be quite costly, but we expect there to be longer-term payoffs.

There seems to be three main approaches to fully exploiting parallelism:

1. Try to keep the elements to be roughly the same size, with the aim of both vectorizing and parallelizing over the elements within a color.
2. Vectorize the computations within each element, while handling the elements within each color in parallel.
3. Apply graph partitioning techniques to decrease the overlap between elements and exploit the sparsity within large elements.

In any of these cases we will need an outer sequential loop over the colors. In the first case there is both a vector and a parallel loop over the elements—we need many of the elements to be of the same size in each color as the same calculations will be performed on each element. In the second case there is a parallel loop over the elements and the treatment of each element is vectorized—of course, the length of vectors is limited by the element sizes. In the third case there is a parallel loop over the elements and the vectorization is limited by the sparsity of the elements.

If we intend to solve large problems with small elements, the first approach is certainly the best and has been successfully applied to finite element problems. Within a particular color, any element with less than the maximum dimension should be padded with zeros. If the elements are large enough with a range of different sizes, we should opt for the second. The third approach will allow us to handle larger amalgamated elements than the other approaches, which might be profitable from the point of view of numerical quality of the preconditioners. Sparse matrix techniques should be used to compute the factors of the Winget decompositions of the elements, while different graph partitioning techniques need to be studied both for numerical efficiency of the preconditioners and parallel implementation. However, the granularity may become quite large and a compromise should be found between the following:

1. using large elements with low overlap, and
2. maintaining sufficient elements for an efficient parallelization while avoiding an increase in the work at each iteration.

The third approach might be the best for very large ill-conditioned problems. In any event, the three implementations are of interest and maybe a dynamic choice is possible. We shall report on this in a future paper.

Finally, we are currently incorporating these preconditioning techniques within the LANCELOT package, profiting from the preprocessing to optimize other parts of the algorithm such as the matrix–vector products and linear solvers.

Acknowledgments. The authors are extremely grateful to Iain Duff, Stephen Nash, Andy Wathen, and an anonymous referee for their careful reading of, and useful comments on a previous draft of this paper.

REFERENCES

[1] P. Amestoy (1991), *Factorization of Large Sparse Matrices Based on a Multifrontal Approach in a Multiprocessor Environment*, Ph.D. thesis TH/PA/91/2, CERFACS, Toulouse, France.

[2] M. Arioli, T. F. Chan, I. S. Duff, N. I. M. Gould, and J. K. Reid (1993), *Computing a Search Direction for Large-Scale Linearly Constrained Nonlinear Optimization Calculations*, Tech. report RAL-93-066, Rutherford Appleton Laboratory, Chilton, England.

[3] O. Axelsson (1976), *Solution of linear systems of equations: Iterative methods*, in Sparse Matrix Techniques (Copenhagen 1976), V. A. Baker, ed., Springer-Verlag, Berlin.

[4] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint (1993), CUTE: *Constrained and Unconstrained Testing Environment*, Tech. report TR/PA/93/10, CERFACS, Toulouse, France.

[5] A. R. Conn, N. I. M. Gould, and Ph. L. Toint (1990), *An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project*, in Computing Methods in Applied Sciences and Engineering, R. Glowinski and A. Lichnewsky, eds., SIAM, Philadelphia, PA, pp. 42–54.

[6] A. R. Conn, N. I. M. Gould, and Ph. L. Toint (1992), *Large-scale nonlinear constrained optimization*, in Proc. 2nd Internat. Conference on Industrial and Applied Mathematics, R. E. O'Malley, Jr., ed., SIAM, Philadelphia, PA, pp. 51–70; also in Linear Algebra for Large-Scale and Real-Time Applications, M. S. Moonen, G. H. Golub, and B. L. R. DeMoor, eds., NATO ASI Series E: Applied Sciences, Vol. 232, Kluwer Academic Publishers, Norwell, MA, 1993.

[7] A. R. Conn, N. I. M. Gould, and Ph. L. Toint (1994), *Improving the decomposition of partially separable functions in the context of large-scale optimization: A first approach*, in *Large Scale Optimization: State of the Art*, W. W. Hager, D. W. Hearn, and P. M. Pardalos, eds., Kluwer Academic Publishers Group, Norwell, MA, and Dordrecht, the Netherlands, pp. 82–94.

[8] M. J. Daydé, J.-Y. L'Excellent, and N. I. M. Gould (1994), *On the Use of Element-by-Element Preconditioners to Solve Large Scale Partially Separable Optimization Problems*, Tech. report RT/APO/94/4, École Nationale Supérieure d'Électronique, d'Électrotechnique, d'Informatique et d'Hydraulique de Toulouse, Toulouse, France.

[9] I. S. Duff and J. K. Reid (1983), *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9, pp. 302–325.

[10] I. S. Duff, R. G. Grimes, and J. G. Lewis (1989), *Sparse matrix test problems*, ACM Trans. Math. Software, 15, pp. 1–14.

[11] J. Erhel, A. Traynard, and M. Vidrascu (1991), *An element-by-element preconditioned conjugate gradient method implemented on a vector computer*, Parallel Comput., 17, pp. 1051–1065.

[12] P. E. Gill and W. Murray (1974), *Newton-type methods for linearly constrained optimization*, in Numerical Methods for Constrained Optimization, P. E. Gill and W. Murray, eds., Academic Press, London.

[13] P. E. Gill, W. Murray, and M. H. Wright (1981), *Practical Optimization*, Academic Press, London, New York.

[14] G. H. Golub and C. F. Van Loan (1989), *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD.

[15] A. Griewank and Ph. L. Toint (1982), *Local convergence analysis for partitioned quasi-Newton updates*, Numer. Math., 39, pp. 429–448.

[16] A. Griewank and Ph. L. Toint (1982), *On the unconstrained optimization of partially separable functions*, in Nonlinear Optimization, M. J. D. Powell, ed., Academic Press, London, New York, pp. 301–312,

[17] A. Griewank and Ph. L. Toint (1982), *Partitioned variable metric updates for large structured optimization problems*, Numer. Math., 39, pp. 429–448.

[18] A. Griewank and Ph. L. Toint (1984), *On the existence of convex decomposition of partially separable functions*, Math. Programming, 24, pp. 25–49.

[19] I. Gustafsson and G. Lindskog (1986), *A preconditioning technique based on element matrix factorization*, Comput. Methods Appl. Mech., 55, pp. 201–220.

[20] M. R. Hestenes (1969), *Multiplier and gradient methods*, J. Optim. Theory Appl., 4, pp. 303–320.

[21] N. J. Higham (1993), *The accuracy of floating-point summation*, SIAM J. Sci. Statist. Comput., 14, pp. 783–799.

[22] T. J. R. Hughes (1987), *The Finite-Element Method: Linear Static and Dynamic Finite-Element Analysis*, Prentice–Hall, Englewood Cliffs, NJ.

[23] T. J. R. Hughes and R. M. Ferencz (1988), *Fully Vectorized EBE Preconditioners for Nonlinear Solid Mechanics: Applications to Large-Scale Three-Dimensional Continuum, Shell and Contact/Impact Problems*, in Proc. 1st Internat. Symp. Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., SIAM, Philadelphia, PA, pp. 261–280.

[24] T. J. R. Hughes, R. M. Ferencz, and J. O. Hallquits (1987), *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, pp. 215–248.

[25] T. J. R. Hughes, I. Levit, and J. Winget (1983), *An element-by-element solution algorithm for problems of structural and solid mechanics*, Comput. Methods Appl. Mech. Engrg., 36, pp. 241–254.

[26] B. M. Irons (1970), *A frontal solution program for finite-element analysis*, Internat. J. Numer. Methods Engrg., 2, pp. 5–32.

[27] E. F. Kaasschieter (1989), *A general finite element preconditioning for the conjugate gradient method*, BIT, 29, pp. 824–849.

[28] S. G. Nash (1984), *Newton-type minimization via the Lanczos method*, SIAM J. Numer. Anal., 21, pp. 770–788.

[29] B. Nour-Omid and B. N. Parlett (1985), *Element preconditioning using splitting techniques*, SIAM J. Sci. Statist. Comput., 6, pp. 761–770.

[30] M. Ortiz, P. M. Pinsky, and R. L. Taylor (1983), *Unconditionally stable element-by-element algorithms for dynamic problems*, Comput. Methods Appl. Mech. Engrg., 36, pp. 223–239.

[31] M. J. D. Powell (1969), *A method for nonlinear constraints in minimization problems*, in Optimization, R. Fletcher, ed., Academic Press, London, New York, pp. 283–298,

[32] J. K. Reid (1984), *Treesolve, A Fortran Package for Solving Large Sets of Linear Finite Element Equations*, Tech. report HL 84/796, AERE Harwell Laboratory, Harwell, UK.

[33] Y. Saad (1990), SPARSKIT: *A Basic Tool Kit for Sparse Matrix Computations*, Tech. report CSRD TR #1029, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL.

[34] R. B. Schnabel and E. Eskow (1991), *A new modified Cholesky factorization*, SIAM J. Sci. Statist. Comput., 11, pp. 1136–1158.

[35] H. A. Van der Vorst (1985), *The Performance of Fortran Implementations for Preconditioned Conjugate Gradients on Vector Computers*, Tech. report 85-09, Delft University of Technology, Delft, the Netherlands.

[36] A. J. Wathen (1989), *An analysis of some element-by-element techniques*, Comput. Methods Appl. Mech. Engrg., 74, pp. 271–287.

[37] A. J. Wathen (1992), *Singular element preconditioning for the finite element method*, in Iterative Methods in Linear Algebra, R. Beauwens and P. de. Groen, eds., Elsevier/North–Holland, Amsterdam, pp. 531–540.

[38] O. C. Zienkiewicz (1977), *The Finite Element Method*, McGraw–Hill, London.