Preprocessing of Sparse Unassembled Linear Systems for Efficient Solution Using Element-by-element Preconditioners[1]

Michel J. Daydé[2], Jean-Yves L'Excellent[2], and Nicholas I. M. Gould[3]

1 Introduction

We consider the solution of systems of linear equations of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where

$$\boldsymbol{A} = \sum_{i=1}^{p} \boldsymbol{A}_i. \tag{2}$$

Each $\boldsymbol{A}_i \in \Re^{n \times n}$ is symmetric and has non-zeros entries only at the intersection of n_i rows and columns. It can thus be represented by a dense elementary matrix $\boldsymbol{A}_i^e \in \Re^{n_i \times n_i}$ such that $\boldsymbol{A}_i = \boldsymbol{C}_i^T \ \boldsymbol{A}_i^e \ \boldsymbol{C}_i$, where the rows of the *connectivity matrix* $\boldsymbol{C}_i$ are simply the rows of the $n \times n$ identity matrix corresponding to the variables used in the element.

The conjugate gradient method combined with element-by-element preconditioners has proved to be effective on linear systems that may arise from finite element and optimization problems. Daydé, L'Excellent and Gould (1994) show that amalgamating elements before constructing such a preconditioner can dramatically improve the speed and numerical behaviour of the method. The effectiveness of the preconditioner depends crucially on the overlap between elements. The amalgamation process typically reduces the overlap between elements, and it is this which leads to improvements in performance. However amalgamation also typically increases the amount of storage required, as zeros may be explicitly stored within amalgamated elements. Thus there is a natural tradeoff between the improvement in the quality of the preconditioner and the storage and effort required to use it.

Our preprocessing step consists into grouping the elements into sets, assembling the elements within each set into a *super-element*, and then applying an element-by-element technique to the super-elements instead of the original $\boldsymbol{A}_i^e$. In this paper, we consider this preprocessing step in detail.

Firstly, we compare the partitions obtained by the "bottom-up" amalgamation method proposed by Daydé et al. (1994) using different thresholds, and consider the possibility of using a sparse storage scheme for the super-elements instead of a dense scheme. We then consider an alternative "top-down" amalgamation technique which uses global structural informations to assign the super-elements. These techniques are compared on a variety of test problems. Further details can be found in L'Excellent (1995).

2 Environment and test problems

We assume that $\boldsymbol{A}$ is positive definite, and use the preconditioned conjugate gradient method, stopping when $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\| < 10^{-9}\|\boldsymbol{b}\|$. In this paper, we concentrate on the

[1]Travel was funded by the ALLIANCE program from the British Council

[2]ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse Cedex, France

[3]Central Computing Department, Rutherford Appleton Laboratory, Oxfordshire, OX11 0QX, England.

EBE preconditioner, first introduced by Hughes, Levit and Winget (1983) and Ortiz, Pinsky and Taylor (1983) and successfully applied in a number of applications in engineering and physics. It is defined by

$$\boldsymbol{P}_{EBE} = \sqrt{\boldsymbol{W}} \prod_{i=1}^{p} \boldsymbol{L}_i \prod_{i=1}^{p} \boldsymbol{D}_i \prod_{i=p}^{1} \boldsymbol{L}_i^T \sqrt{\boldsymbol{W}}, \tag{3}$$

where $\boldsymbol{W}$ is the diagonal of $\boldsymbol{A}$, and $\boldsymbol{L}_i\boldsymbol{D}_i\boldsymbol{L}_i^T$ is an $\boldsymbol{LDL}^T$ factorization of the *Winget decompositions* of each elementary matrix $\boldsymbol{A}_i$ defined to be equal to $\boldsymbol{I}+\sqrt{\boldsymbol{W}}^{-1}(\boldsymbol{A}_i - \boldsymbol{W}_i)\sqrt{\boldsymbol{W}}^{-1}$, where $\boldsymbol{W}_i$ is the diagonal of $\boldsymbol{A}_i$. This preconditioner has been widely used on finite element problems. It offers considerable scope for parallelism, both in the construction of the preconditioner and during the matrix-vector product and solves that occur during the conjugate gradient calculations.

Problem name	n	p	Min element size	Max element size	Mean element size	Degree of overlap	κ
BIGGSB1	998	1001	0	2	2.0	2.0	4.0×10^5
NOBNDTOR	480	562	1	5	4.2	4.9	1.8×10^2
MAT33	637	273	1	3	2.6	6.0	5.5×10^1
CBRATU3D	4394	4394	5	8	7.5	7.5	3.4×10^1
NET3	512	531	1	6	2.6	2.7	2.4×10^9

Table 1: Summary of the characteristics of each test problem

We describe in Table 1 some of the test matrices that will be used in our experiments. Other problems are described in Daydé et al. (1994). Some important characteristics are the condition number κ, which gives an idea of the difficulty of the problem; the sizes of the elements (larger elements often encourage faster execution rates); the degree of overlap which characterizes the structure of a problem in terms of overlap between elements. It is defined as the average number of elements sharing each variable and is an indicator as to how well element-by-element preconditioners will behave on the problem. We now describe the two amalgamation strategies that will be compared.

3 Bottom-up strategy: elemental amalgamation

3.1 Amalgamation algorithm

A variety of amalgamation techniques are considered in detail by Daydé et al. (1994). Here, we only consider the most successful of these. Let $\mathcal{G}_i$ denote an element, $\mathcal{V}_i$ denote the set of indices of variables used by the element $\mathcal{G}_i$, and $|\mathcal{V}_i|$ denote the cardinal of $\mathcal{V}_i$. Finally, let $tim(i)$ be the estimated time spent in a matrix-vector product of order i for the diagonal (DIAG) preconditioner (strategy **amalg1**); or in a matrix-vector product and in two triangular solves of order i for the EBE preconditioner (strategy **amalg2**). The amalgamation process we have used computes the *benefit*:

$$b(\mathcal{G}_i, \mathcal{G}_j) = tim(|\mathcal{V}_i|) + tim(|\mathcal{V}_j|) - tim(|\mathcal{V}_i \cup \mathcal{V}_j|) \tag{4}$$

for all pairs of elements and amalgamates the pair with the largest benefit so long as it is lower than a *threshold* value. Note that $tim(i)$ only depends on i and can be computed once and for all. These machine-dependent costs are stored into files

and determined during the installation of the software. If we are only interested in reducing the time per iteration, the best value for the threshold would be zero, but smaller values will result in further amalgamations and, hence, possibly better preconditioners.
When using such an algorithm, the super-elements are stored as dense matrices and thus some zeros may be explicitly stored. The density of the super-elements depends on the structure of the initial elements and, because of the dependence of the amalgamation on $tim(.)$, also on the target computer. For example, on an ALLIANT FX/80 and using a threshold of zero, we typically obtain elements with a density around 0.3, which is quite large for the elements to be treated as sparse. On RISC architectures, the density of the elements obtained can be even larger, since the amalgamation process is stopped earlier (long vectors are not required for efficiency as on a vector processor). However, we have suggested that continued amalgamation is often beneficial to the quality of the preconditioner, and this results in a reduction in the density of the super-elements. Thus it may well be advantageous to use a sparse representation of the super-elements.

3.2 Experiments on amalgamation

Detailed results of amalgamation for various preconditioners can be found in Daydé et al. (1994) and L'Excellent (1995). We show in Figure 1 the effect of amalgamating elements for diagonal and EBE preconditioning with a threshold equal to 0 on a range of test problems using strategies **amalg1** and **amalg2** respectively. The problems are sorted by increasing condition number and large gains in execution time are obtained using amalgamation when the elements are initially small and overlap significantly. The results are scaled so that the time for construction and convergence are compared to those obtained using diagonal preconditioning without amalgamation. We observe that with amalgamation, EBE is more efficient than diagonal preconditioning as soon as the problem is sufficiently hard to solve (problems 6 to 15). For diagonal preconditioning, the gains from amalgamation are due to the better execution rates whereas for EBE, the gains are due both to a better execution rate and a smaller number of iterations.

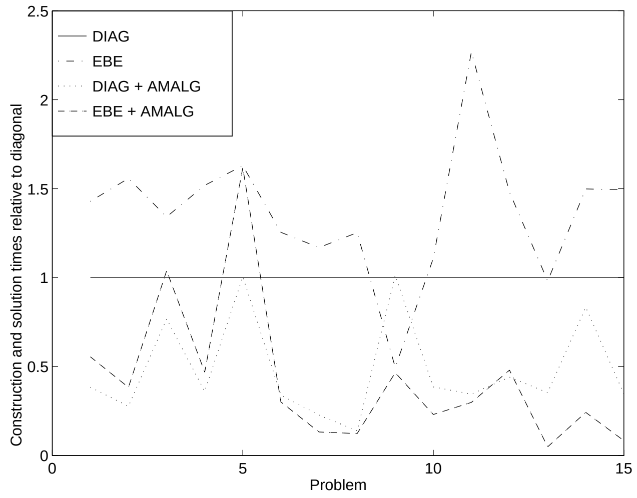

Figure 1: Comparison of DIAG and EBE on ALLIANT FX/80 (results are scaled with respect to diagonal preconditioning without amalgamation).

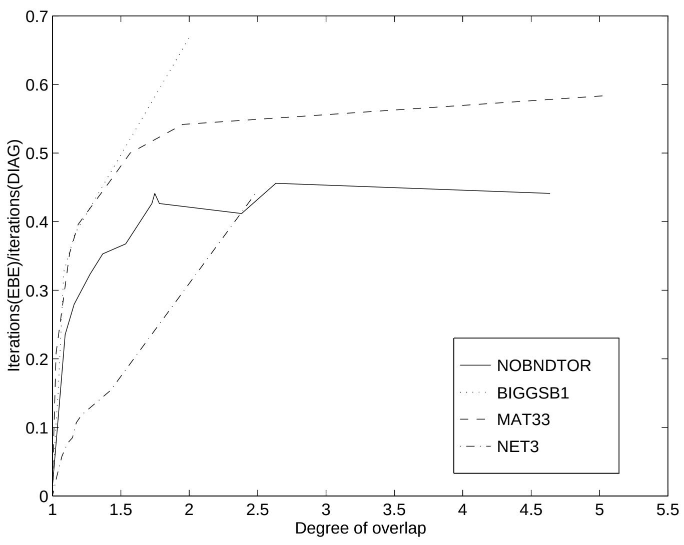

Figure 2: Number of iterations of EBE compared to DIAG, depending on the degree of overlap obtained with different amalgamation thresholds.

3.3 Modifying the amalgamation threshold

As the threshold decreases, so do the degree of overlap of the structure obtained by amalgamation and, thus, the number of iterations, as can be seen in Figure 2, where we report the ratio between the number of EBE iterations and the number of iterations of the diagonal preconditioner for the problems BIGGSB1, MAT33, NET3 and NOBNDTOR.

We now consider the test problem NET3 to study the behaviour of EBE preconditioning in terms of number of iterations and its performance using different amalgamation thresholds. Some results, obtained on a SPARC-10 RISC workstation, are presented in Table 2 (dense storage columns). In this table, the average density is defined to be the ratio

$$d = \frac{\sum_{i=1}^{p} nz_i}{\sum_{i=1}^{p} s_i^2}, \tag{5}$$

where p is the number of elements, nz_i is the number of non-zero entries in the element i, and s_i is the size of the i^{th} element. Const. time refers to the construction time of the preconditioner. Here, the time spent in the solves (Time solves) only refers to the solution steps on the preconditioner and excludes the time of matrix-vector products and other cg-related operations (such as dot products). This is simply so that we can compare the results of Table 2 for the dense storage scheme with those in the next section where we consider sparse storage. As matrix-vector products and dot-products are performed in the same way in both cases — using the dense initial elements, the construction time and the time spent in the solves are the more meaningful parameters to be compared. Note that the best choice of an amalgamation threshold depends on the total time to obtain the solution which is not reported here.

The fact that the convergence occurs in one iteration while there are still 8 elements in the last line of the table is because there are 7 small independent elements in this problem. In this case, the method is a direct solver. For such a small ill-conditioned problem, we observe that the decrease in the number of iterations is large enough to compensate for both the increase of the cost per iteration and for the construction time.

				Dense storage			Sparse storage				
								Minimum degree		Natural ordering	
Threshold	# elts	Average elt size	Average density	# its	Const. time	Time solves	# its	Const. time	Time solves	Const. time	Time solves
0.01	465	2.7	1.00	684	0.01	3.58	675	0.14	6.49	0.12	6.49
0.0	104	7.0	0.51	243	0.02	0.76	231	0.07	1.29	0.06	1.29
-0.00004	52	11.9	0.34	187	0.02	0.75	183	0.07	0.87	0.05	1.02
-0.00012	39	15.1	0.26	142	0.02	0.63	138	0.06	0.63	0.06	0.79
-0.0002	29	19.6	0.19	123	0.02	0.69	122	0.06	0.54	0.07	0.76
-0.001	19	28.7	0.11	87	0.04	0.71	84	0.06	0.36	0.08	0.64
-0.004	13	40.9	0.06	57	0.07	0.87	58	0.06	0.23	0.16	0.63
-0.02	9	57.6	0.02	17	0.22	0.65	16	0.06	0.08	0.77	0.41
-1000.0	8	64.0	0.01	1	0.66	0.14	1	0.07	0.02	2.35	0.06

Table 2: Results of amalgamation obtained for the problem NET3 using different thresholds on a SPARC-10 workstation using dense and sparse kernels in the the factorizations and triangular solves.

We show in Table 3 the impact of varying the amalgamation threshold on the MFlops rate and the speed-up on a vector multiprocessor : the ALLIANT FX/80. Sol. time is the time for converging to the solution. Amalgamation improves vectorization while parallelization is less efficient because of the decrease in the number of elements. We have not included the MFlops rate when the threshold is -0.02 since our estimate is inaccurate, as some anticipated operations within the ALLIANT BLAS on the zero elements stored within the super-elements are not actually performed.

Threshold	# elts	Average elt size	Overlap degree.	# its	Amalg. time(s)	Const. time(s)	Sol. time(s)	MFlops	Speed-up 8 procs
No amalg.	538	2.6	2.71	694	-	0.18	173.53	0.1	5.5
0.1	465	2.7	2.48	687	0.38	0.17	148.48	0.1	5.4
0.0	32	17.9	1.12	135	1.23	0.16	7.83	1.1	3.5
-0.00004	31	18.4	1.11	131	1.23	0.16	7.61	1.1	3.4
-0.00008	29	19.5	1.10	123	1.24	0.17	7.22	1.2	2.9
-0.00012	28	20.2	1.10	120	1.24	0.17	7.01	1.2	2.9
-0.00016	27	20.9	1.10	117	1.25	0.17	6.87	1.2	3.5
-0.0004	23	23.9	1.08	91	1.27	0.19	5.47	1.4	3.1
-0.0006	21	26.0	1.07	87	1.28	0.21	5.37	1.5	2.7
-0.002	17	31.6	1.05	66	1.32	0.26	4.47	1.9	2.2
-0.004	14	38.0	1.04	54	1.38	0.33	4.27	2.3	1.9
-0.01	10	52.0	1.02	25	1.84	1.05	4.60	5.1	1.1
-0.02	8	64.0	1.00	1	2.43	2.21	0.61	-	1.1

Table 3: Effect of varying the amalgamation threshold on ALLIANT FX/80 for the problem NET3.

3.4 Use of sparse storage

In the results of the previous section, we observe that the density of the super-elements decreases as the amalgamation process proceeds. It thus seems natural to use a sparse storage for these elements when the threshold value and thus the element density becomes sufficiently small. In Table 2, we report the construction time and the time spent in the solves of the EBE preconditioner with a sparse storage of the super-elements for the test problem NET3. The factorization of the Winget decomposition of each super-element is obtained using the sparse, symmetric

solver MA27 from the Harwell Subroutine Library (1995). We compare two different orderings for the elimination of the variables: the minimum degree ordering and the natural order of the elements. Except for very small and dense elementary matrices, it appears that minimum degree is always better.
The density of the elements decreases with the threshold. We observe that sparse storage becomes more efficient than dense storage for threshold values smaller than -0.00012, which corresponds to a density smaller than 0.26. This is especially true for the triangular systems for which there is less overhead using sparse storage than for the factorization. Of course, for this particular problem (small and ill-conditioned, with little fill-in during the factorization), a direct method making use of the sparsity of the matrix would be the best choice.
We now consider in Table 4 a larger, but better conditioned problem, CBRATU3D ($\kappa = 3.4 \times 10^1$). This problem is of order 4394, initially with 4394 elements, that arises from a 3D problem and for which there is significant fill-in during sparse factorization. As before, we compare the construction time of the preconditioner and the time spent in the solves using different threshold values for the amalgamation on ALLIANT FX/80 and SUN SPARC-10. The fact that the density is not equal to 1.0 for the initial problem is due to the presence of zeros in the original elements.

					Sparse storage			Dense storage		
Computer	Threshold	# elts	Average elt size	Average density	# its	Const. time	Time solves	# its	Const. time	Time solves
ALLIANT FX/80	No amalg.	4394	7.5	0.69	-	-	-	20	5.8	69.2
	0.1	4394	7.5	0.69	20	35.0	38.4	20	5.8	69.2
	0.0	713	27.9	0.28	21	26.7	25.2	21	7.7	29.4
	-0.00004	547	33.8	0.25	21	27.6	24.3	21	8.3	28.1
	-0.00016	489	36.6	0.24	21	27.8	23.8	20	8.7	26.4
	-0.0006	319	49.2	0.19	20	29.1	21.0	20	10.4	25.6
	-0.002	179	75.5	0.14	19	31.4	18.6	20	13.9	26.2
	-0.005	106	111.7	0.11	19	34.2	17.8	19	19.7	27.1
	-0.1	25	338.6	0.03	17	52.9	16.1	18	71.3	68.8
SPARC-10	No amalg.	4394	7.5	0.69	-	-	-	20	0.6	2.9
	0.1	4394	7.5	0.69	20	2.8	5.0	20	0.6	2.9
	0.00	4253	7.7	0.68	20	2.7	4.9	21	0.7	3.1
	-0.00004	1751	15.3	0.41	22	2.4	4.7	22	0.8	3.7
	-0.00008	788	27.0	0.27	20	2.6	4.0	20	1.2	4.2
	-0.00016	723	28.7	0.26	20	2.6	3.8	20	1.2	4.3
	-0.0002	705	29.1	0.26	20	2.6	3.8	20	1.3	4.5
	-0.0004	363	46.8	0.19	20	2.8	3.7	20	1.9	5.6
	-0.0006	307	53.0	0.18	20	2.8	3.7	20	2.1	6.0
	-0.001	248	61.9	0.16	20	3.1	3.8	19	2.2	6.0
	-0.002	162	83.6	0.13	19	3.2	3.6	19	3.2	7.0
	-0.005	77	146.6	0.08	19	3.8	3.9	19	7.3	9.9

Table 4: Comparison of sparse and dense storage using different amalgamation thresholds on SUN SPARC-10 and ALLIANT FX/80 for the test problem CBRATU3D.

On SPARC-10, the sparse storage is seen to be better when the density of the elements falls below 0.2. However, since the problem is well conditioned, the amalgamation is not very effective here, and the best solution time is obtained with dense computations. Notice that the different roundoff properties of the sparse and dense variants sometimes results in slight variations in the numbers of iterations performed.
If we consider a vector computer, such as the ALLIANT FX/80, amalgamation is useful with dense super-element storage. But again, there is a significant overhead in using sparse elements since the number of iterations is initially small and not

significantly reduced by amalgamation, and because of the large amount of fill-in during the factorizations. Therefore, in this case, it is seen to be preferable to use long vectors rather than sparse elements. This would not be true if the the ratio of convergence to construction time was larger, or if the number of iterations had been reduced more significantly as the number of elements decreased.

3.5 Conclusion

When solving a well conditioned problem, sparse elements do not appear to be very useful. For an ill-conditioned problem, amalgamation often reduces the number of iterations. Sparse elements can then be effective provided sufficient amalgamation occurs. We believe that the main use of sparse super-elements is those large-scale ill-conditioned problems for which direct factorization gives rise to significant fill-in. We note that the criteria used in amalgamation strategies presupposes a dense storage scheme will be used. If the super-elements are treated as sparse matrices, other techniques may be suitable. We now consider this possibility.

4 Top-down strategy: use of graph partitioning techniques

If the super-elements are considered to be sparse, we have little reason to use an amalgamation strategy which is based on the times or numbers of operations of dense linear algebra kernels, such as that proposed in the previous section. That strategy is based on local criteria. In this section, we investigate a global strategy in which we partition the whole connectivity graph of the elements rather than amalgamate elements locally. If the problem arises from the discretization of a physical problem, it is often better to use the physical properties of the domain to determine a first partition of the graph. If such information is not available, there are automatic graph partitioning algorithms that aim at partitioning the graph into a desired number of approximately equal-sized portions.
Our objectives remain to minimize the size of the overlap between elements — this improves the convergence of the EBE-based method; and limit the costs of factorization and solution on the elements. A compromise has to be reached between these two factors in order to obtain an optimal strategy. As this compromise depends on many parameters, such as the structure of the problem, its conditioning, and the time to construct the preconditioner relative to the time for convergence, we perform experiments to investigate these issues.

4.1 Choice of a graph partitioning algorithm

The vertices of the connectivity graph we are considering here are the elements. An edge exists between two vertices (elements) if they share at least one variable. Given our stated objectives in the previous section, the criterion used to partition the graph should depend on the number of variables shared between the partitions, and should also aim to balance the cost of the subsequent sparse factorizations. For example it is often less expensive to factorize two matrices of dimension roughly $m/2$ than one matrix of dimension roughly m and a second of low dimension. Furthermore, this also gives a good balance for a parallel implementation.
For this preliminary study, we have used the graph partitioning package METIS (see Karypis and Kumar, 1995). Many strategies are available within the package that aim at partitioning a graph into k partitions. We almost always use the provided default options which are: coarsening phase using sorted heavy edge matching, uncoarsening phase using combination of boundary greedy and boundary

Kernighan-Lin, and bisection using graph growing partition followed by boundary Kernighan-Lin.
As we have already seen, it is preferable from the point of building an effective preconditioner that the number of shared variables between the partitions be small. However, METIS minimizes the number of edges cut, trying to keep approximately the same number of vertices in each partition. For certain problems, we have tried using differing weights for the edges and the vertices of the graph (see L'Excellent, 1995), but this seems only useful for problems with a very irregular structure. For the test problems described in this paper, the use of equal node and edge weights make the preprocessing faster and gives partitions of sufficient quality.
The construction of the preconditioner requires the $\boldsymbol{LDL}^T$ factorization of the Winget decomposition of sparse super-elements. The ideal solver should accept in entry dense symmetric unassembled elements (see MUSSELS, Amestoy, Daydé, Duff, L'Excellent, Gould, Reid and Scott, 1994). However, as this software is not yet available, we use the multifrontal code MA27 from the Harwell Subroutine Library (1995). A first phase consists into finding an ordering of the variables so that the fill-in is kept to a minimum along the factorization. We always use minimum degree, which is the default option of MA27, since it gives the best results. The solution routines of MA27 were modified so that lower triangular, diagonal, and upper triangular systems can be solved separately. MA27 accepts the matrix in the Harwell-Boeing format $(i, j, value)$ and values corresponding to a same position (i, j) are summed, which avoids the assembly of the super-elements before the call. The other subroutines are provided in the PAREBE package (see Daydé, L'Excellent and Gould, 1995). All the factorizations can be performed in parallel, but we do not exploit this in the current implementation. If the number of elements is sufficiently large, a colouring can be applied to parallelize the solve. Note that one of the advantage of using a graph partitioning algorithm is that the number of colours is often small. Matrix-vector products are performed element-wise using the initial elements. The time for a matrix-vector product does not depend on the number of partitions, but is not optimal here since the initial elements are used. Some gain could be expected by using amalgamation techniques within partitions.

4.2 Experiments

In the experiments #elts is the number of partitions. The elements are always treated as sparse. A value of 1 means a direct method (1 iteration). For a value equal to the initial number of elements p, the initial structure is used and the elements are treated as sparse. For these two special values, no use of METIS is made. min size, max size, and avg size are respectively the minimum, the maximum and the average sizes of the super-elements obtained. deg ovl is the degree of overlap of the new structure, that is the average number of elements containing each variable. var 1el is the number of variables contained by only 1 super-element. var 2el is the number of variables contained by exactly 2 super-elements. #its is the number of iterations required by the method to obtain a residual norm lower than 10^{-9}. time prec is the time to compute the preconditioner, that is the sum of the times for determining the Winget decompositions, and for performing the symbolic and numerical factorizations for all elements. time sol is the time spent in the solution of the preconditioned system at each iteration.
It is difficult to define a good criterion to measure the quality of an elementary decomposition. In order to compare the partitions obtained by different partitioning techniques, we study the degree of overlap (deg ovl) as a function of the number of elements (#elts) for the test problem CBRATU3D. This is illustrated in the Figure 3. The amalgamation algorithm is as defined in section 2 and uses an absolute criterion based on the time per iteration (estimates are performed on an

HP/715). The graph is partitioned using METIS with default options and unit vertex and edge weights. Observe that the curves are very close, which indicates that the graph partitioning technique performs as well as amalgamation when considering the degree of overlap.

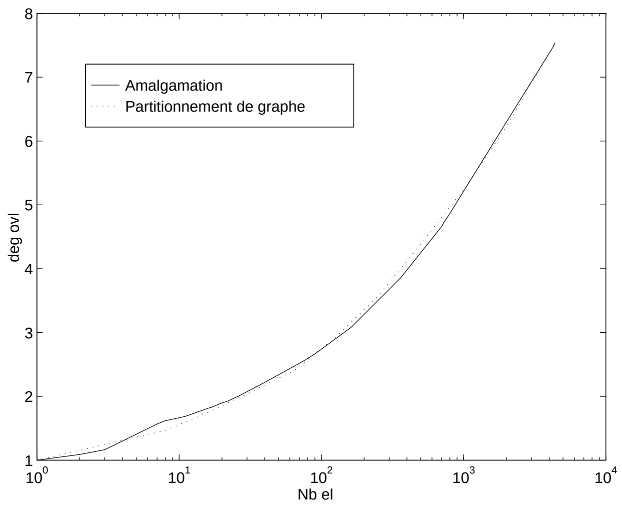

Figure 3: Degree of overlap as a function of the number of elements obtained with amalgamation and graph partitioning for the test problem CBRATU3D.

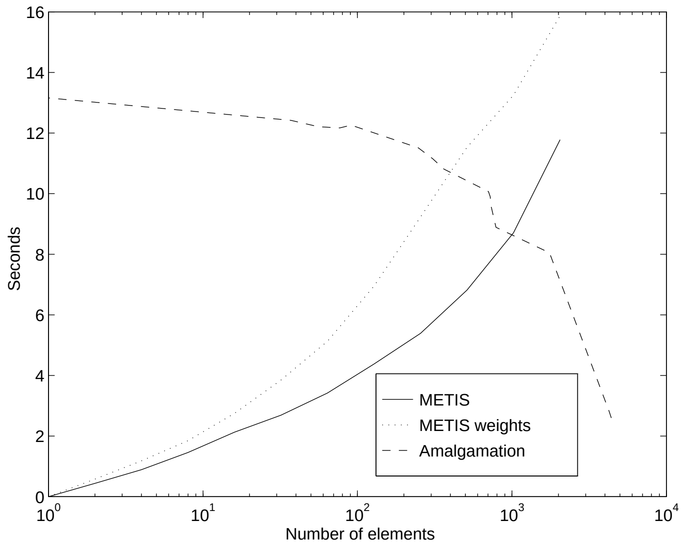

Figure 4: Time for symbolic amalgamation and graph partitioning using METIS as a function of the number of elements obtained on the HP/715 for CBRATU3D.

While the resulting partitions are not necessarily better than those obtained with amalgamation, the computation of the partitions with METIS is often much faster than the use of an amalgamation algorithm, as is shown in Figure 4. Clearly, the amalgamation scheme is cheaper when a large number of elements are required, while the opposite is true for the partitioning algorithm. For reasonable numbers of partitions, the time of METIS is always smaller. While these times are only given for the test problem CBRATU3D, for which the preprocessing step is quite costly, similar effects were observed on the other large problems.

We now study the behaviour of the method described above for different numbers

of partitions. We use a recursive algorithm that aims to partition the graph into 2^k portions, but note that METIS also allows more general partitions. From Section 2, we already know that the use of sparse elements is not very effective if the problem is well conditioned. We have also observed that for ill-conditioned problems of small size or with low fill-in, direct methods can be the most efficient alternative. Thus, we now consider harder problems, such as 3D finite element problems, where a direct solver is too costly.
Table 5 gives convergence results for the CBRATU3D test problem with different numbers of partitions on the ALLIANT FX/80 and on the HP/715. In Tables 5–6, the last column ("Total") represents the total time for the solution of the linear system. The run 4394^* in the last row of the tables indicates the results for the initial, unamalgamated problem using dense elements. As we have already observed for this very well conditioned problem, the number of iterations does not decrease significantly with the number of partitions. Thus, the initial decomposition with dense kernels remains the best compromise. For such a problem, we can use a large number of partitions, but when considering the two last lines of the table which compare using the p initial elements with a sparse or dense representation, we see that our code would benefit from switching to a dense-storage mode for elements with many nonzeros.
We now consider a problem with the same structure, but the numerical values are modified so that the convergence is harder. The eigenvalues in each element are randomly chosen in the range $[10^{-9}, 10^{9}]$. The results for this example are given in Table 6. A direct method is still too costly and the EBE preconditioner applied to the initial matrix requires too many iterations. However, now the combination of sparse super-elements with graph partitioning gives much better results. The best compromise is obtained for 32 partitions, but there are benefits for a larger range of number of partitions, specifically from 8 to 256 on the ALLIANT and from 16 to 512 on the HP.

							ALLIANT FX/80				HP/715			
# elts	min size	max size	avg size	deg ovl	var 1el	var 2el	# its	time prec	time sol	Total	# its	time prec	time sol	Total
1	4394	4394	4394.0	1.00	4394	0	1	1737.9	5.8	1745.0	1	280.2	1.6	282.0
2	2534	2536	2535.0	1.15	3718	676	12	516.5	23.3	547.9	12	80.7	5.7	87.3
4	1436	1446	1440.0	1.31	3134	1160	14	191.7	18.8	219.9	14	27.4	4.1	32.5
8	790	820	805.0	1.47	2648	1476	15	123.1	17.2	150.3	15	15.5	3.5	20.0
16	430	531	480.4	1.75	1772	2020	15	60.7	13.8	84.6	15	6.1	2.3	9.5
64	**127**	**203**	**164.9**	**2.40**	**666**	**1781**	**16**	**35.6**	**13.5**	**59.8**	**16**	**2.8**	**1.7**	**5.6**
512	24	48	37.9	4.41	0	72	21	26.3	24.4	64.5	21	2.0	2.4	5.8
1024	11	34	22.6	5.27	0	14	22	26.1	29.1	69.6	22	2.0	2.8	6.4
4394	5	8	7.5	7.54	0	0	20	34.7	38.8	86.6	20	2.8	3.9	8.0
4394*	**5**	**8**	**7.5**	**7.54**	**0**	**0**	**20**	**5.66**	**26.6**	**45.6**	**20**	**0.4**	**3.1**	**4.9**

Table 5: Results for the test problem CBRATU3D on ALLIANT FX/80 and on HP/715.

5 Conclusion

We believe that these preliminary results indicate that the amalgamation methods considered here hold promise, especially for large, ill-conditioned problems for which direct methods are inappropriate and simple element-by-element preconditioners ineffective. However, we must be cautious as our proposals are merely heuristics, and believe that further experimentation is necessary to assess the full potential of

							ALLIANT FX/80				HP/715			
# elts	min size	max size	avg size	deg ovl	var 1el	var 2el	# its	time prec	time sol	Total	# its	time prec	time sol	Total
1	4394	4394	4394.0	1.00	4394	0	1	1738.8	5.83	1745.9	1	281.9	1.5	283.6
2	2534	2536	2535.0	1.15	3718	676	141	517.3	255.5	861.5	141	80.96	62.8	152.9
4	1436	1446	1440.0	1.31	3134	1160	167	192.3	210.8	508.4	166	27.49	46.4	84.6
8	790	820	805.0	1.47	2648	1476	155	124.0	168.4	390.2	155	15.6	33.8	59.4
16	430	531	480.4	1.749	1772	2020	172	60.8	150.2	319.3	173	6.16	25.3	42.6
32	**232**	**337**	**283.9**	**2.07**	**1096**	**2093**	**185**	**41.7**	**148.2**	**306.2**	**185**	**3.70**	**21.0**	**36.5**
64	127	203	164.9	2.40	666	1781	209	34.9	167.3	333.4	209	2.88	20.7	37.1
512	24	48	37.9	4.41	0	72	249	26.3	276.9	459.6	249	2.02	27.0	45.0
1024	11	34	22.6	5.27	0	14	253	26.3	321.8	507.7	253	2.06	31.2	49.6
4394	5	8	7.54	7.54	0	0	230	34.9	426.4	605.6	231	2.66	42.8	60.6
4394*	5	8	7.54	7.54	0	0	230	5.72	298.6	450.4	231	0.44	34.3	49.3

Table 6: Modified ill-conditioned problem CBRATU3D on ALLIANT FX/80 and HP/715.

the methods.
The technique can also be applied to other element-by-element preconditioners or to block methods. For example, the assembly of some super-elements would decrease the cost of the solution step for the GS EBE preconditioner (see Hughes et al., 1983, and Ortiz et al., 1983).

References

Amestoy, P. R., Daydé, M. J., Duff, I. S., L'Excellent, J.-Y., Gould, N. I. M., Reid, J. K. and Scott, J. A. (1994), Mussels – a multifrontal sparse symmetric element-input linear solver, Technical report, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England. Working note.

Daydé, M. J., L'Excellent, J.-Y. and Gould, N. I. M. (1994), On the use of element-by-element preconditioners to solve large scale partially separable optimization problems, Technical Report RT/APO/94/4, ENSEEIHT-IRIT, Toulouse, France. submitted to the SIAM J. Sci. Comput.

Daydé, M. J., L'Excellent, J.-Y. and Gould, N. I. M. (1995), User's guide to PAREBE : parallel element-by-element preconditioners for the conjugate gradient algorithm, Technical report, ENSEEIHT-IRIT, Toulouse, France. To appear.

Harwell Subroutine Library (1995), *A catalogue of subroutines (release 12)*, AEA Technology, Harwell, Oxfordshire, England.

Hughes, T. J. R., Levit, I. and Winget, J. (1983), 'An element-by-element solution algorithm for problems of structural and solid mechanics', *Comput. Methods Appl. Mech. Eng.* **36**, 241–254.

Karypis, G. and Kumar, V. (1995), METIS : Unstructured graph partitionning and sparse matrix ordering system, version 2.0, Technical report, University of Minnesota, Department of Computer Science, Minneapolis.

L'Excellent, J.-Y. (1995), Utilisation de préconditionneurs élément-par-élément pour la résolution de problèmes d'optimisation de grande taille, PhD thesis, ENSEEIHT-INPT.

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