UCRL-JRNL-230844

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Distance-Two Interpolation for Parallel Algebraic Multigrid

H. De Sterck, R. Falgout, J. Nolting, U. M. Yang

May 10, 2007

Numerical Linear Algebra with Applications

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

DISTANCE-TWO INTERPOLATION FOR PARALLEL ALGEBRAIC MULTIGRID

HANS DE STERCK[*§], ROBERT D. FALGOUT[†¶], JOSHUA W. NOLTING[‡‖], AND ULRIKE MEIER YANG[†**]

Abstract. Algebraic multigrid (AMG) is one of the most efficient and scalable parallel algorithms for solving sparse linear systems on unstructured grids. However, for large three-dimensional problems, the coarse grids that are normally used in AMG often lead to growing complexity in terms of memory use and execution time per AMG V-cycle. Sparser coarse grids, such as those obtained by the Parallel Modified Independent Set coarsening algorithm (PMIS) [7], remedy this complexity growth, but lead to non-scalable AMG convergence factors when traditional distance-one interpolation methods are used. In this paper we study the scalability of AMG methods that combine PMIS coarse grids with long distance interpolation methods. AMG performance and scalability is compared for previously introduced interpolation methods as well as new variants of them for a variety of relevant test problems on parallel computers. It is shown that the increased interpolation accuracy largely restores the scalability of AMG convergence factors for PMIS-coarsened grids, and in combination with complexity reducing methods, such as interpolation truncation, one obtains a class of parallel AMG methods that enjoy excellent scalability properties on large parallel computers.

Key words. Algebraic multigrid, long range interpolation, parallel implementation, reduced complexity, truncation

AMS subject classifications. 65F10, 65N55, 65Y05

1. Introduction. Algebraic multigrid (AMG) [2, 5] is an efficient potentially scalable algorithm for sparse linear systems on unstructured grids. However, when applied to large three-dimensional problems, the classical algorithm often generates unreasonably large complexities with regard to memory use as well as computational operations. Recently, we suggested a new parallel coarsening algorithm, called the Parallel Modified Independent Set algorithm (PMIS) [7], which is based on a parallel independet set algorithm suggested in [10]. The use of this coarsening algorithm in combination with a slight modification of Ruge and Stueben's classical interpolation scheme [11], leads to significantly lower complexities as well as significantly lower setup and cycle times. For various test problems, such as isotropic and grid aligned anisotropic diffusion operators, one obtains scalable results, particularly when AMG is used in combination with Krylov methods. However, AMG convergence factors are severely impacted for more complicated problems, such as problems with rotated anisotropies or highly discontinuous material properties. Since we realized that classical interpolation methods, which use only distance-one neighbors for their interpolatory set, were not sufficient for these coarse grids, we decided to investigate interpolation operators which also include distance-two neighbors. In this paper we focus on the following distance-two interpolation operators: we study three methods proposed in [12], namely, standard interpolation, multipass interpolation, and the

[*]Department of Applied Mathematics, University of Waterloo, Waterloo, ON N2L 3G1, Canada
[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, PO Box 808, Livermore, CA 94551, USA
[‡]Department of Applied Mathematics, Campus Box 526, University of Colorado at Boulder, Boulder, CO 80302, USA
[§]hdesterck@uwaterloo.ca
[¶]rfalgout@llnl.gov
[‖]Josh.Nolting@colorado.edu
[**]umyang@llnl.gov

use of Jacobi interpolation to improve other interpolation operators, and we investigate two extensions of classical interpolation, which we denote with 'extended' and 'extended+i' interpolation.

Our investigation shows that all of the long-distance interpolation strategies, except for multipass interpolation, significantly improve AMG convergence factors compared to classical interpolation. Multipass interpolation shows poor numerical scalability, which, however, can be improved with a Krylov accelerator, but it has very small computational complexity. All other long-distance interpolation operators showed increased complexities. While the increase is not very significant for two-dimensional problems, it is of concern in the three-dimensional case. Therefore we also investigated complexity reducing strategies, such as the use of smaller sets of interpolation points and interpolation truncation. The use of these strategies led to AMG methods with significantly improved overall scalability.

The paper is organized as follows. In Section 2 we briefly describe AMG. In Section 3 distance-one interpolation operators are presented, and Section 4 describes long range interpolation operators. In Section 5 the computational cost of the interpolation strategies is investigated, and in Section 6 some sequential numerical results are given, which motivate the following sections. Section 7 presents various complexity reducing strategies. Section 8 investigates the parallel implementation of the methods. Section 9 presents parallel scaling results for a variety of test problems, and Section 10 contains the conclusions.

2. Algebraic Multigrid. In this section we give an outline of the basic principles and techniques that comprise AMG, and we define terminology and notation. Detailed explanations may be found in [4, 11, 12]. Consider a problem of the form

$$Au = f, \tag{2.1}$$

where A is an $n \times n$ matrix with entries a_{ij}. For convenience, the indices are identified with grid points, so that u_i denotes the value of u at point i, and the grid is denoted by $\Omega = \{1, 2, \ldots, n\}$. In any multigrid method, the central idea is that "smooth error," e, that is not eliminated by relaxation must be removed by coarse-grid correction. This is done by solving the residual equation $Ae = r$ on a coarser grid, then interpolating the error back to the fine grid and using it to correct the fine-grid approximation.

Using superscripts to indicate level number, where 1 denotes the finest level so that $A^1 = A$ and $\Omega^1 = \Omega$, AMG needs the following components: "grids" $\Omega^1 \supset \Omega^2 \supset \ldots \supset \Omega^M$, grid operators $A^1, A^2, \ldots, A^M$, interpolation operators P^k, restriction operators R^k (often $R^k = (P^k)^T$), and smoothers S^k, where $k = 1, 2, \ldots M-1$.

Most of these components of AMG are determined in a first step, known as the *setup phase*. During the setup phase, on each level $k, k = 1, \ldots M-1$, Ω^{k+1} is determined using a coarsening algorithm, P^k and R^k are defined and the A^{k+1} is determined using the Galerkin condition $A^{k+1} = R^k A^k P^k$. Once the setup phase is completed, the *solve phase*, a recursively defined cycle, can be performed as follows:

Algorithm: $MGV(A^k, R^k, P^k, S^k, u^k, f^k)$.
If $k = M$, solve $A^M u^M = f^M$ with a direct solver.
Otherwise:
Apply smoother S^k μ_1 times to $A^k u^k = f^k$.
Perform coarse grid correction:
Set $r^k = f^k - A^k u^k$.
Set $r^{k+1} = R^k r^k$.
Apply $MGV(A^{k+1}, R^{k+1}, P^{k+1}, S^{k+1}, e^{k+1}, r^{k+1})$.
Interpolate $e^k = P^k e^{k+1}$.
Correct the solution by $u^k \leftarrow u^k + e^k$.
Apply smoother S^k μ_2 times to $A^k u^k = f^k$.

In the remainder of the paper, index k will be dropped for simplicity. The algorithm above describes a V(μ_1, μ_2)-cycle; other more complex cycles such as W-cycles are described in [4]. In every V-cycle, the error is reduced by a certain factor, which is called the convergence factor. A sequence of V-cycles is executed until the error is reduced below a specified tolerance. For a scalable AMG method, the convergence factor is bounded away from one as a function of the problem size n, and the computational work in both the setup and solve phases is linearly proportional to the problem size n. While AMG was originally developed in the context of symmetric M-matrix problems, AMG has been applied successfully to a much wider class of problems. We assume in this paper that A has positive diagonal elements.

3. Distance-One Interpolation Strategies. In this section, we first give some definitions as well as some general remarks, and then recall the possibly simplest interpolation strategy, the so-called direct interpolation strategy [12]. This is followed by a description of the classical distance-one AMG interpolation method that was introduced by Ruge and Stüben [11].

3.1. Definitions and Remarks. One of the concepts used in the following sections is *strength of connection*. A point j strongly influences a point i or i strongly depends on j if

$$-a_{i,j} > \alpha \max_{k \neq i}(-a_{i,k}), \tag{3.1}$$

where $0 < \alpha < 1$. We set $\alpha = 0.25$ in the remainder of the paper.

We define the *measure* of a point i as the number of points which strongly depend on i. When PMIS coarsening is used, a positive random number that is smaller than 1 is added to the measure to distinguish between neighboring points that strongly influence the same number of points. In the PMIS coarsening algorithm, points that do not strongly influence any other points are initialized as F-points.

Using this concept of strength of connection we define the following sets:

$$\begin{aligned} N_i &= \{j | a_{ij} \neq 0\}, \\ S_i &= \{j \in N_i | j \text{ strongly influences } i\}, \\ F_i^s &= F \cap S_i, \\ C_i^s &= C \cap S_i, \\ N_i^w &= N_i \setminus (F_i^s \cap C_i^s). \end{aligned}$$

In classical AMG [11], the interpolation of the error at the F-point i takes the

form

$$e_i = \sum_{j \in C_i} w_{ij} e_j, \tag{3.2}$$

where w_{ij} is an interpolation weight determining the contribution of the value e_j to e_i, and $C_i \subset C$ is the coarse interpolatory set of F-point i. In most classical approaches to AMG interpolation, C_i is a subset of the nearest neighbors of grid point i, i.e. $C_i \subset N_i$, and longer-range interpolation is not considered.

The points to which i is connected, comprise three sets: C_i^s, F_i^s and N_i^w. Based on assumptions on small residuals for smooth error [2, 4, 11, 12], an interpolation formula can be derived as follows. The assumption that algebraically smooth error has small residuals after relaxation

$$A\,e \approx 0,$$

can be rewritten as

$$a_{ii} e_i \approx - \sum_{j \in N_i} a_{ij} e_j, \tag{3.3}$$

or

$$a_{ii} e_i \approx - \sum_{j \in C_i^s} a_{ij} e_j - \sum_{j \in F_i^s} a_{ij} e_j - \sum_{j \in N_i^w} a_{ij} e_j. \tag{3.4}$$

From this expression, various interpolation formulae can be derived. We use the terminology of [12] for the various interpolation strategies.

3.2. Direct Interpolation. The so-called 'direct interpolation' strategy [12] has one of the most simple interpolation formulae. The coarse interpolatory set is chosen as $C_i = C_i^s$, and

$$w_{ij} = -\frac{a_{ij}}{a_{ii}} \frac{\sum_{k \in N_i} a_{ik}}{\sum_{k \in C_i^s} a_{ik}}, \quad j \in C_i^s. \tag{3.5}$$

This leads to an interpolation, which is often not accurate enough. Nevertheless, we mention this approach here, since various other interpolation operators which we consider are based on it. This method is denoted by 'direct' in the tables presented below. In [12] it is also suggested to separate positive and negative coefficients when determining the weights, a strategy which can help when one encounters large positive off-diagonal matrix coefficients. We do not consider this approach here, since the strategy did not lead to an improvement for the problems we consider here.

3.3. Classical Interpolation. A generally more accurate distance-one interpolation formula is the interpolation suggested by Ruge and Stüben in [11], which we call 'classical interpolation' ('clas'). Again, $C_i = C_i^s$, but the contribution from strongly influencing F-points (the points in F_i^s) in (3.3) is taken into account more carefully. An appropriate approximation for the errors e_j of those strongly influencing F-points may be defined as

$$e_j \approx \frac{\sum_{k \in C_i} a_{jk} e_k}{\sum_{k \in C_i} a_{jk}}. \tag{3.6}$$

This approximation can be justified by the observation that smooth error varies slowly in the direction of strong connection. The denominator simply ensures that constants are interpolated exactly. Replacing the e_j with a sum over the elements k of the coarse interpolatory set C_i corresponds to taking into account strong $F-F$ connections using C-points that are common between the F-points. Note that, when the two F-points i and j do not have a common C-point in C_i^s and C_j^s, the denominator in (3.6) is small or vanishing. Weak connections (from the points in N_i^w) are generally not important and errors $e_j, j \in N_i^w$ are replaced by e_i. This leads to the following formula for the interpolation weights:

$$w_{ij} = -\frac{1}{a_{ii} + \sum\limits_{k \in N_i^w} a_{ik}} \left(a_{ij} + \sum_{k \in F_i^s} \frac{a_{ik} a_{kj}}{\sum\limits_{m \in C_i^s} a_{km}} \right), \quad j \in C_i^s. \tag{3.7}$$

In our experiments this interpolation is further modified as proposed in [9] to avoid extremely large interpolation weights that can lead to divergence.

Now the interpolation above was suggested based on a coarsening algorithm that ensured that two strongly connected F-points always have a common coarse neighbor. Since this condition is no longer guaranteed when using PMIS coarsening [7], it may happen that the term $\sum_{m \in C_i^s} a_{k,m}$ in Eq. (3.7) vanishes. In our previous paper on the PMIS coarsening method [7], we modified interpolation formula (3.7) such that if this case occurs, a_{ik} is added to the diagonal term (the term $a_{ii} + \sum_{k \in N_i^w} a_{ik}$ in Eq. (3.7)), i.e., the strongly influencing neighbor point k of i is treated like a weak connection of i. In what follows, we denote the set of strongly connected neighbors k of i that are F-points but do not have a common C-point, i.e. $C_i^s \cap C_k^s = \emptyset$, by F_i^{s*}. Combining this with the modification suggested in [9] we obtain the following interpolation formula:

$$w_{ij} = -\frac{1}{a_{ii} + \sum\limits_{k \in N_i^w \cup F_i^{s*}} a_{ik}} \left(a_{ij} + \sum_{k \in F_i^s \setminus F_i^{s*}} \frac{a_{ik} \bar{a}_{kj}}{\sum\limits_{m \in C_i^s} \bar{a}_{km}} \right), \quad j \in C_i^s \tag{3.8}$$

where

$$\bar{a}_{ij} = \begin{cases} 0 & \text{if } \operatorname{sign}(a_{ij}) = \operatorname{sign}(a_{ii}) \\ a_{ij} & \text{otherwise.} \end{cases}$$

In this paper we refer to formula (3.8) as 'classical interpolation'. The numerical results that were presented in [7] showed that this interpolation formula, which is based on Ruge and Stüben's original distance-one interpolation formula [11], resulted in AMG methods with acceptable performance when used with PMIS-coarsened grids for various problems, but only when the AMG cycle is accelerated by a Krylov subspace method. Without such acceleration, interpolation formula (3.8) is not accurate enough on PMIS-coarsened grids: AMG convergence factors deteriorate quickly as a function of problem size, and scalability is lost. For various problems, such as problems with rotated anisotropies or problems with large discontinuities, adding Krylov acceleration did not remedy the scalability problems.

One of the issues is that distance-one interpolation schemes do not treat situations like the one illustrated in Figure 3.1 correctly. Here we have an F-point with measure

smaller than 1 that has no coarse neighbors. This situation can occur for example if we have a fairly large strength threshold. Both for classical and direct interpolation, the interpolated error in this point will vanish, and coarse grid correction will not be able to reduce the error in this point.

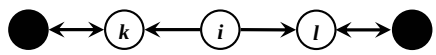

FIG. 3.1. *Example illustrating a situation occurring with PMIS coarsening, which will not correctly be treated by direct or classical interpolation. Black points denote C-points, white points denote F-points, and the arrow from i to l denotes that i strongly depends on l.*

A major topic of this paper is to investigate whether distance-two interpolation methods are able to restore grid-independent convergence to AMG cycles that use PMIS-coarsened grids, without compromising scalability in terms of memory use and execution time per AMG V-cycle.

4. Long-Range Interpolation Strategies. In this section, various long-distance interpolation methods are described. Parallel implementation of some of these interpolation methods and parallel scalability results on PMIS-coarsened grids are discussed later in this paper.

4.1. Multipass Interpolation. Multipass interpolation ('mp') is suggested in [12], and is useful for low complexity coarsening algorithms, particularly so-called aggressive coarsening [12]. We suggested it in [7] as a possible interpolation scheme to fix some of the problems that we saw when using our classical interpolation scheme (3.8). Multipass interpolation proceeds as follows:

1. Use direct interpolation for all F-points i, for which $C_i^s \neq \emptyset$. Place these points in set F^*.
2. For all $i \in F \setminus F^*$ with $F^* \cap F_i^s \neq 0$, replace, in Equation (3.3), for all $j \in F_i^s \cap F^*$, e_j by $\sum_{k \in C_j} w_{jk} e_k$, where C_j is the interpolatory set for e_j. Apply direct interpolation to the new equation. Add i to F^*. Repeat step 2 until $F^* = F$.

Multipass interpolation is fairly cheap. However, it is not very powerful, since it is based on direct interpolation. If applied to PMIS, it still ends up being direct interpolation for most F-points. However, it fixes the situation illustrated in Figure 3.1. If we apply multipass interpolation, the point i will be interpolated by the coarse neighbors (black points) of F-points k and l.

4.2. Jacobi Interpolation. Another approach that remedies convergence issues caused by distance-one interpolation formulae is Jacobi interpolation [12]. This approach uses an existing interpolation formula and applies one or more Jacobi iteration steps to the F-point portion of the interpolation operator leading to a more accurate interpolation operator.

Assuming that A and the interpolation operator $P^{(n)}$ are reordered according to the C/F-splitting and can be written in the following way:

$$A = \begin{pmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{pmatrix}, \qquad P^{(n)} = \begin{pmatrix} P_{FC}^{(n)} \\ I_{CC} \end{pmatrix}, \tag{4.1}$$

then Jacobi iteration on $A_{FF} e_F + A_{FC} e_C = 0$, with initial guess $e_F^{(0)} = P_{FC}^{(0)} e_C$, leads

to

$$P_{FC}^{(n)} = (I_{FF} - D_{FF}^{-1} A_{FF}) P_{FC}^{(n-1)} - D_{FF}^{-1} A_{FC}, \tag{4.2}$$

where D_{FF} is the diagonal matrix containing the diagonal of A_{FF}, and I_{FF} and I_{CC} are identity matrices.

If we apply this approach to a distance-one interpolation operator like classical interpolation, we obtain an improved long distance interpolation operator. This approach is also recommended to be used to improve multipass interpolation. We include results where classical interpolation is used followed by one step of Jacobi interpolation in our numerical experiments and denote them by 'clas+j'.

4.3. Standard Interpolation. Standard interpolation ('std') extends the interpolatory set that is used for direct interpolation [12]. This is done by extending the stencil obtained through (3.3) via substitution of every e_j with $j \in F_i^s$ by $1/a_{jj} \sum_{k \in N_j} a_{jk} e_k$. This leads to the following formula

$$\hat{a}_{ii} e_i + \sum_{j \in \hat{N}_i} \hat{a}_{ij} e_j \approx 0 \tag{4.3}$$

with the new neighborhood $\hat{N}_i = N_i \cup \bigcup_{j \in F_i^s} N_j$ and the new coarse point set $\hat{C}_i = C_i \cup \bigcup_{j \in F_i^s} C_j$. This can greatly increase the size of the interpolatory set.

See the left example in Figure 4.1. Consider point i. Using direct or classical interpolation, i would only be interpolated by the two distance-one coarse points. However, when we include the coarse points of its strong fine neighbors m and n, two additional interpolatory points k and l are added, leading to a potentially more accurate interpolation formula. Standard interpolation is now defined by applying direct interpolation to the new stencil, leading to

$$w_{ij} = -\frac{\hat{a}_{ij}}{\hat{a}_{ii}} \frac{\sum_{k \in \hat{N}_i} \hat{a}_{ik}}{\sum_{k \in \hat{C}_i} \hat{a}_{ik}}. \tag{4.4}$$

4.4. Extended Interpolation. It is possible to extend the classical interpolation formula so that the interpolatory set includes C-points that are distance two away from the F-point to be interpolated, i.e. applying the classical interpolation formula, but using the same interpolatory set that is used in standard interpolation, see Figure 4.1: $\hat{C}_i = C_i \cup \bigcup_{j \in F_i^s} C_j$.

Using the same reasoning that leads to the classical interpolation formula (3.7), the following approximate statement can be made regarding the error at an F-point i:

$$\left(a_{ii} + \sum_{j \in N_i^w} a_{ij} \right) e_i \approx - \sum_{j \in \hat{C}_i} a_{ij} e_j - \sum_{j \in F_i^s} a_{ij} \frac{\sum_{k \in \hat{C}_i} a_{jk} e_k}{\sum_{k \in \hat{C}_i} a_{jk}}. \tag{4.5}$$

It then follows immediately that the interpolation weights using the extended

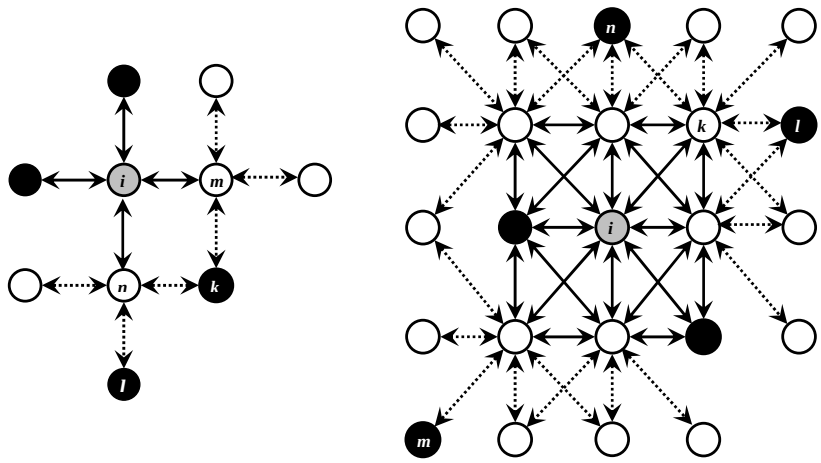

FIG. 4.1. *Example of the interpolatory points for a 5-point stencil (left) and a 9-point stencil (right). The gray point is the point to be interpolated, black points are C-points and white points are F-points.*

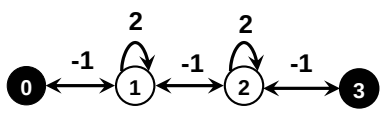

FIG. 4.2. *Finite difference 1D Laplace example.*

coarse interpolatory set $\hat{C}_i$ can be defined as

$$w_{ij} = -\frac{1}{a_{ii} + \sum_{k \in N_i^w} a_{ik}} \left(a_{ij} + \sum_{k \in F_i^s} \frac{a_{ik}\bar{a}_{kj}}{\sum_{m \in \hat{C}_i} \bar{a}_{km}} \right), \quad j \in \hat{C}_i. \tag{4.6}$$

This new interpolation formula deals efficiently with strong $F - F$ connections that do not share a common C-point. We call this interpolation strategy 'extended interpolation' ('ext').

4.5. Extended+i Interpolation. While extended interpolation remedies many problems that occur with classical interpolation, it does not always lead to the desired weights. Consider the case given in Figure 4.2. Here we have a 1-dimensional Laplace-problem generated by finite differences. Points 1 and 2 are strongly connected F-points, and points 0 and 3 are coarse points. Clearly 0, 3 is the interpolatory set for point 1 for the case of extended interpolation. If we apply formula (4.6) to this example to calculate $w_{1,0}$ and $w_{1,3}$, we obtain

$$w_{1,0} = 0.5, \qquad w_{1,3} = 0.5.$$

This is a better result than we would obtain for direct interpolation (3.5) and classical interpolation (3.8):

$$w_{1,0} = 1, \qquad w_{1,3} = 0,$$

but worse than standard interpolation (4.4), for which we get the intuitively best interpolation weights:

$$w_{1,0} = 2/3, \qquad w_{1,3} = 1/3. \tag{4.7}$$

This can be remedied if we include not only connections a_{jk} from strong fine neighbors j of i to points k of the interpolatory set but also connections a_{ji} from j to point i itself. An alternative to expression (3.6) for the error in strongly connected F-points, is then given by

$$e_j \approx \frac{\sum_{k\in C_i\cup\{i\}} a_{jk} e_k}{\sum_{k\in C_i\cup\{i\}} a_{jk}}. \tag{4.8}$$

This can be rewritten as

$$e_j \approx \frac{\sum_{k\in C_i} a_{jk} e_k}{\sum_{k\in C_i\cup\{i\}} a_{jk}} + \frac{a_{ji}\, e_i}{\sum_{k\in C_i\cup\{i\}} a_{jk}}, \tag{4.9}$$

which then, in a similar way as before, leads to interpolation weights

$$w_{ij} = \frac{1}{\tilde{a}_{ii}} \left(a_{ij} + \sum_{k\in F_i^s} a_{ik} \frac{\bar{a}_{kj}}{\sum_{l\in \hat{C}_i\cup\{i\}} \bar{a}_{kl}} \right), \quad j \in \hat{C}_i, \tag{4.10}$$

with now

$$\tilde{a}_{ii} = a_{ii} + \sum_{n\in D_i^w} a_{in} + \sum_{k\in F_i^s} a_{ik} \frac{\bar{a}_{ki}}{\sum_{l\in \hat{C}_i\cup\{i\}} \bar{a}_{kl}}. \tag{4.11}$$

We call this modified extended interpolation 'extended+i', and refer to it by 'ext+i' (or sometimes 'e+i' to save space) in the tables below. If we apply it to the example illustrated in Figure 4.2 we obtain weights (4.7).

5. Computational Cost of Interpolation Strategies. In this section we consider the cost of some of the interpolation operators described in the previous sections. We use the following notations:

N_c	total number of coarse points
N_f	total number of fine points
n_k	average number of distance-k neighbor points
c_k	average number of distance-k interpolatory points
f_k	average number of strong fine distance-k neighbors
w_k	average number of weak distance-k neighbors
s_k	average number of common distance-k interpolatory points
f_w	average number of strong neighbors treated weakly

Here f_w indicates the number of strong F-neighbors that are treated weakly, which occur only for classical interpolation (3.7). Also, s_k denotes the average number of C-points which are distance-one neighbors of $j \in F_i^s$ and also distance-k interpolatory points for i, the point to be interpolated. Note that s_k is usually smaller than c_k and at most equal to c_k. Note also that $n_k = f_k + c_k + w_k$.

In our considerations we assume a compressed sparse row data format, i.e. three arrays are used to store the matrix: a real array that contains the coefficients of the matrix, an integer array that contains the column indices for each coefficient and an integer array that contains pointers to the beginning of each row for the other two arrays. We also assume an additional integer array which indicates whether a point is an F- or a C-point.

For all interpolation operators mentioned before, it is necessary to determine at first the interpolatory set. At the same time the data structure for the interpolation operator can be determined. This can be accomplished by sweeping through each row that belongs to an F-point: coarse neighbors are identified via integer comparisons, and the pointer array for the interpolation operator is generated. For the distance-two interpolation schemes, it is also necessary to check neighbors of strong fine neighbors. This requires n_1 comparisons for direct and classical interpolation, and $(f_1+1)n_1$ comparisons for extended, extended+i and standard interpolation. The final data structure contains $N_c + N_f c_1$ coefficients for classical and direct interpolation, and $N_c + N_f(c_1+c_2)$ coefficients for extended(+i) and standard interpolation.

Next, the interpolation data structure is filled.

For direct interpolation, all that is required is to sweep through a whole row once to compute $\alpha_i = -\sum_{k\in N_i} a_{ik} / \left(a_{ii} \sum_{k\in C_i^s} a_{ik}\right)$ and then multiply the relevant matrix elements a_{ij} with α_i. The sum in the denominator requires an additional n_1 comparisons, and the two summations require n_1+1 additions.

For classical, extended and extended+i interpolation one needs to first compute for each point $k \in F_i^s \setminus F_i^{s*}$, $\alpha_{ik} = a_{ik} / \left(\sum_{m\in D_i^s} a_{km}\right)$ with the appropriate set D_i^s, which is different for each operator. For example, for classical interpolation, this requires $f_1 n_1$ comparisons. After this step, all these coefficients need to be processed again in order to add $\alpha_{ij} a_{kj}$ to the appropriate weights. This requires an additional $f_1 n_1$ comparisons. The number of additions, multiplications and divisions can be determined similarly.

For standard interpolation, at first the new stencil needs to be computed, leading to $f_1 n_1$ additions and multiplications. After this one proceeds just as for direct interpolation with a much larger stencil of size n_1+n_2.

The number of floating point additions, multiplications and divisions to compute all interpolation weights for each F-point are given in Table 5.1. Note that a sum over m elements is treated as m additions, assuming that we are adding to a variable that was originally 0. Note also that occurrences of products of variables, such as $f_i c_i$ or $f_i s_i$, are of order n_i^2, since f_i, c_i, s_i are dependent on n_i. This is also reflected in the results given in Table 5.2 for two specific examples.

Interp.	additions	multiplications	divisions	comparisons
direct	n_1+1	c_1+1	1	$2n_1$
clas	$2f_1s_1+w_1+f_w$	$f_1s_1+c_1$	f_1-f_w+1	$(2f_1+1)n_1$
std	$(f_1+1)n_1+n_2$	$f_1n_1+c_1+c_2$	f_1+1	$(2f_1+2)n_1-f_1+n_2$
ext	$2f_1(s_1+s_2)$ $+w_1$	$f_1(s_1+s_2)$ $+c_1+c_2$	f_1+1	$(3f_1+1)n_1-f_1$
ext+i	$2f_1(s_1+s_2+1)$ $+w_1$	$f_1(s_1+s_2+1)$ $+c_1+c_2$	f_1+1	$(3f_1+1)n_1-f_1$

TABLE 5.1

Computational cost for various interpolation operators.

Let us look at some examples to get an idea about the actual cost involved. First consider a 5-point stencil as in Figure 4.1. Here, we have the following parameters: $c_1 = f_1 = 2$, $w_1 = w_2 = 0$, $n_1 = 4$, $f_w = 2$, $s_1 = 0$, $c_2 = 2$, $f_2 = 3$, $n_2 = 5$, $s_2 = 1.5$. Table 5.2 shows the resulting interpolation cost. Next, we look at an example with a bigger stencil, see Figure 4.1 and Table 5.2. The parameters are now $c_1 = 2$, $f_1 = 6$, $w_1 = w_2 = 0$, $n_1 = 8$, $f_w = 1$, $s_1 = 1$, $c_2 = 3$, $f_2 = 12$, $n_2 = 15$, $s_2 = 1$. We clearly see that a larger stencil significantly increases the ratio of classical over direct interpolation, as well as that of distance-two over distance-one interpolations.

	Left example in Fig. 4.1				Right example in Fig. 4.1			
Interpol.	adds	mults	divs	comps	adds	mults	divs	comps
direct	5	3	1	8	9	3	1	16
clas	2	2	1	20	13	8	6	104
std	17	12	3	19	71	53	7	73
ext	6	7	3	26	24	17	7	146
ext+i	10	9	3	26	36	23	7	146

TABLE 5.2
Cost for examples in Figure 4.1.

Table 5.3 shows the times for calculating these interpolation operators for matrices with stencils of various sizes. Two two-dimensional examples, one with a 5-point and one with a 9-point stencil, were examined on a 1000×1000 grid. The three-dimensional examples, with a 7-point and a 27-point stencil, were examined for an $80 \times 80 \times 80$ grid. We have also included actual measurements of the average number of interpolatory points for these examples. As expected, larger stencils lead to a larger number of operations for each interpolation operator, with a much more significant increase for distance-two interpolation operators, particularly for the 3-dimensional problems. These effects are significant, especially since on coarser levels the stencils become larger and, thus, impact the total setup time.

			Interpolation				
Stencil	c_1	c_2	direct	clas	std	ext	ext+i
5-pt	2.3	1.9	0.27	0.35	0.64	0.51	0.54
9-pt	1.8	2.8	0.36	1.11	2.16	2.09	2.48
7-pt	2.7	4.1	0.19	0.31	0.80	0.73	0.81
27-pt	2.3	7.2	0.40	3.72	8.00	7.43	8.32

TABLE 5.3
Average number of distance-one (c_1) and distance-two (c_2) interpolatory points and times for various interpolation operators.

6. Sequential Numerical Results. While the previous section examined the computational cost for the interpolation operator, we are of course mainly interested in the performance of the complete solver, which also includes coarsening, the generation of the coarse grid operator as well as the solve phase. We apply the new and old interpolation operators here to a variety of test problems from [7] to compare their efficiency. We did not include results using direct interpolation, since it performs worse than classical and multipass interpolation for the problems considered, nor

results using multipass interpolation followed by Jacobi interpolation, since these results were very similar to those obtained for 'clas+j'. All these tests were obtained using AMG as a solver with a strength threshold of $\alpha = 0.25$, and coarse-fine-Gauss-Seidel as a smoother. The iterations were stopped when the relative residual was smaller than 10^{-8}.

	5-point			9-point		
Method	C_{op}	# its	time	C_{op}	# its	time
clas	1.92	244	151.60	1.24	157	100.44
clas+j	2.65	15	26.09	1.65	9	21.03
mp	1.92	244	152.34	1.24	183	115.72
ext	2.54	16	20.24	1.60	10	18.26
ext+i	2.57	11	16.93	1.60	10	18.40
std	2.56	16	20.63	1.60	17	23.06

TABLE 6.1

AMG for the 5-point and the 9-point 2D Laplace problem on a 1000×1000 square with random right hand side using different interpolation operators.

	45 degrees			60 degrees		
Method	C_{op}	# its	time	C_{op}	# its	time
clas	1.90	168	38.60	1.82	>1000	
clas+j	2.39	29	10.50	3.40	424	131.85
mp	1.90	163	37.16	1.82	>1000	
ext	2.07	31	8.75	2.69	217	59.70
ext+i	2.07	11	4.05	2.89	97	29.78
std	2.07	13	4.53	2.89	148	43.68

TABLE 6.2

AMG for a problem with a by 45 degrees and by 60 degrees rotated anisotropy on a 512×512 square using different interpolation operators.

Table 6.1 shows results for the two-dimensional Poisson problem $-\Delta u = f$ using a 5-point finite difference discretization and a 9-point finite element discretization. Table 6.2 shows results for the two-dimensional rotated anisotropic problem

$$-(c^2 + \epsilon s^2)u_{xx} + 2(1-\epsilon)scu_{xy} - (s^2 + \epsilon c^2)u_{yy} = 1 \tag{6.1}$$

with $s = \sin\gamma,\ c = \cos\gamma$, and $\epsilon = 0.001$ with rotation angles $\gamma = 45^o$ and $\gamma = 60^o$.

The use of the distance-two interpolation operators combined with PMIS shows significant improvements over classical and multipass interpolation with regard to number of iterations as well as time. The best interpolation operator here is the ext+i interpolation, which has the lowest number of iterations and times in general. The difference is especially significant in the case of the problems with rotated anisotropies. The operator complexity is larger, however, as was to be expected.

This increase becomes more significant for three-dimensional problems. Here we consider the partial differential equation

$$-(au_x)_x - (au_y)_y - (au_z)_z = f \tag{6.2}$$

on a $n\times n\times n$ cube. For the Laplace problem $a(x,y,z) = 1$, for the problem denoted by 'Jumps' we consider the function $a(x,y,z) = 1000$ for the interior cube $0.1 < x, y, z <$

	7-point			27-point			Jumps		
Method	C_{op}	# its	time	C_{op}	# its	time	C_{op}	# its	time
clas	2.34	45	10.21	1.09	28	10.58	2.50	>1000	
clas+j	5.12	11	20.35	1.34	8	17.10	5.37	15	20.99
mp	2.35	47	10.40	1.10	30	9.39	2.50	80	17.37
ext	4.93	11	16.70	1.35	8	21.32	5.27	15	16.89
ext+i	4.27	9	14.48	1.35	8	21.55	5.10	11	15.96
std	4.20	10	12.78	1.38	10	18.58	5.21	18	17.47

TABLE 6.3

AMG for a 7-point 3D Laplace problem, a problem with a 27-point stencil and a 3D structured PDE problem with jumps on a $60 \times 60 \times 60$ *cube with a random right hand side using different interpolation operators.*

0.9, $a(x, y, z) = 0.01$ for $0 < x, y, z < 0.1$ and the other cubes of size $0.1 \times 0.1 \times 0.1$ that are located at the corners of the unit cube and $a(x, y, z) = 1$ elsewhere. The 27-point problem is a matrix with a 27-point stencil with the value 26 in the interior and -1 elsewhere and is being tested because we also wanted to consider a problem with a larger stencil.

While for these problems AMG convergence factors for distance-two interpolation improve significantly compared to classical and multipass interpolation, as can be seen in Table 6.3, overall times are worse for the 7pt 3D Laplace problem as well as the 27-point problem on a $60 \times 60 \times 60$ grid. The only problem on the $60 \times 60 \times 60$ grid that benefits from distance-two interpolation operators also with regard to time is the problem with jumps, which requires long-distance interpolation to even converge. Using distance-two interpolation operators leads to complexities about twice as large as those obtained when using classical or multi-pass interpolation, which work relatively well for the 7- and 27-point problem on the $60 \times 60 \times 60$ grid. However, when we scale up the problem sizes, they show very good scalability in terms of AMG convergence factors, as can be seen in Figure 6.1, which shows the number of iterations for a 3D 7-pt Laplace problem on a $n \times n \times n$ grid for increasing n. The anticipated large differences in numbers of iterations between distance-one and distance-two interpolations show up in the 2D results of Tables 6.1 and 6.2 on grids with 1000 points per direction, but are not particularly significant yet in the 3D results of Table 6.3 with only 60 points per direction. It is expected, however, that for the large problems that we want to solve on a parallel computer, distance-two interpolation operators will lead to overall better times than classical or multi-pass interpolation due to scalable AMG convergence factors, if the operator complexity can be kept under control. See Section 9 for actual test results.

7. Reducing Operator Complexity. While the methods described in the previous section largely restore grid-independent convergence to AMG cycles that use PMIS-coarsened grids, they also lead to much larger operator complexities for the V-cycles. Therefore, it is necessary to consider ways to reduce these complexities while (hopefully) retaining the improved convergence. In this section we describe a few ways of achieving this.

7.1. Choosing Smaller Interpolatory Sets. It is certainly possible to consider other interpolatory sets, which are larger than C_i^s, but smaller than $\hat{C}_i$. Particularly, it appears that a good interpolatory set would be one that only extends C_i^s for strong $F - F$ connections without a common C-point, since in the other cases

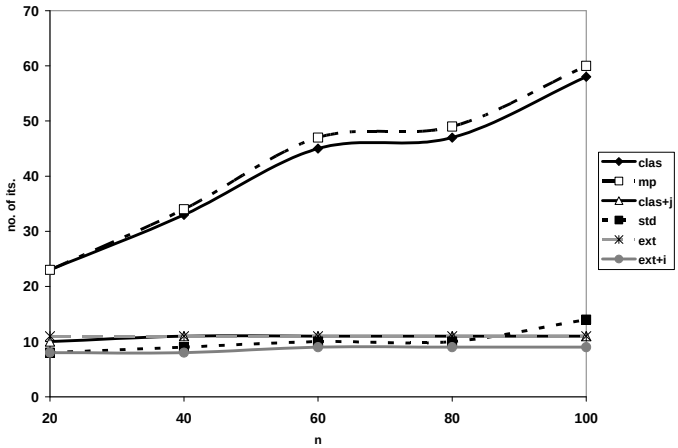

FIG. 6.1. *Number of iterations for PMIS with various interpolation operators for a 3D 7-point Laplace problem on a $n \times n \times n$-grid.*

point i is likely already surrounded by interpolatory points and an extension is not necessary. If we look at the right example in Figure 4.1, we see that neighbor k of i is the only fine neighbor that does not share a C-point with i. Consequently, it may be sufficient to only include points n and l in the extended interpolatory set. Applying this approach to the extended interpolation leads to so-called $F-F$ interpolation [3]. The size of the interpolatory set can be further decreased if we limit the number of C-points added when an F-point is encountered that does not have common coarse neighbors. This has been done in the so-called $F-F1$ interpolation [3], where only the first C-point is added. For the right example in Figure 4.1 this means that only point n or point l would be added to the interpolatory set for i.

Choosing a smaller interpolatory set decreases c_2 and with it s_2, leading to fewer multiplications and additions for the extended interpolation methods. On the other hand, additional operations are needed to determine which coarse neighbors of strong F-points are common C-points. This means that the actual determination of the interpolation operator might not be faster than creating the extended interpolation operators. The real benefit, however, is achieved by the fact that the use of smaller interpolatory sets leads to smaller stencils for the coarse grid operator and hence to smaller overall operator complexities.

Applying these methods to some of our previous test problems, we get the results shown in Tables 7.1 and 7.2. Here, 'x-cc' denotes that interpolation 'x' is used, but the interpolatory set is only extended when there are no common C-points. Similarly, 'x-ccs' is just like 'x-cc', except that for every strong F-point without a common C-point only a single C-point is added.

The results show that 2-dimensional problems do not benefit from this strategy, since operator complexities are only slightly decreased, while the number of iterations increases. Therefore total times increase. However, 3-dimensional problems can be solved much faster due to significantly decreased setup times leading to only half the total times when the 'ccs'-strategy is employed. Again, these beneficial effects are expected to be stronger on larger grids. Indeed, additional numerical tests (not presented here, see [3]) also show that the 'x-cc' and 'x-ccs' distance-two interpolations result in algorithms that are highly scalable as a function of problem size: C_{op} tends

	9-point			45 degrees			60 degrees		
Method	C_{op}	# its	time	C_{op}	# its	time	C_{op}	# its	time
ext	1.60	10	18.26	2.07	31	8.67	2.69	217	59.70
ext-cc	1.45	14	17.35	2.06	34	9.33	2.62	247	66.22
ext-ccs	1.43	15	17.46	2.05	34	9.13	2.42	270	67.96
ext+i	1.60	10	18.40	2.07	11	4.05	2.89	97	29.78
ext+i-cc	1.45	14	17.91	2.05	14	4.72	2.80	117	34.63
ext+i-ccs	1.42	15	17.98	2.04	14	4.73	2.51	143	38.87

TABLE 7.1

AMG for the 9-point 2D Laplace problem on a 1000×1000 square with random right hand side using different interpolation operators and rotated anisotropies of 0.001 on a 512 × 512 grid.

	7-point			27-point			Jumps		
Method	C_{op}	# its	time	C_{op}	# its	time	C_{op}	# its	time
ext	4.93	11	16.70	1.35	8	21.32	5.27	15	16.89
ext-cc	4.62	12	11.11	1.33	7	11.82	4.86	16	11.59
ext-ccs	4.00	12	8.46	1.31	7	10.34	4.23	17	9.61
ext+i	4.27	9	14.48	1.35	8	21.55	5.10	11	15.96
ext+i-cc	4.12	9	9.16	1.33	7	12.48	4.66	13	10.35
ext+i-ccs	3.64	9	7.23	1.31	7	10.95	4.00	14	8.37

TABLE 7.2

AMG for a 7-point and a 27-point 3D Laplace problem and a 3D structured PDE problem with jumps on a $60 \times 60 \times 60$ *cube with a random right hand side using different interpolation operators.*

to a constant that is significantly smaller than the C_{op} value for the 'x' interpolations, and the number of iterations is nearly constant as a function of problem size, and only slightly larger than the number of iterations for the full 'x' interpolation formulas [3]. This shows that using distance-two interpolation formulas with reduced complexities restores the grid-independent convergence and scalability of AMG on PMIS-coarsened grids, without the need for GMRES acceleration. This makes these methods suitable algorithms for large problems on parallel computers, as is discussed below.

7.2. Interpolation Truncation. Another very effective way to reduce complexities is the use of interpolation truncation. There are essentially two ways we can truncate interpolation operators: we can choose a truncation factor θ and eliminate every weight whose absolute value is smaller than this factor, i.e. for which $|w_{ij}| < \theta$ [12], or we can limit the number of coefficients per row, i.e. choose only the k_{max} largest weights in absolute value. In both cases the new weights need to be re-scaled so that the total sums remain unchanged.

Both approaches can lead to significant reductions in setup times and operator complexities, particularly for 3-dimensional problems, but if too much is truncated, the number of iterations rises significantly, as one would expect.

We only report results for one interpolation formula (ext+i) for a 3D example here, see Table 7.3. However, similar results can be obtained using the other interpolation operators. For 2-dimensional problems, truncation leads to an increase in total time similarly as reported for interpolatory set restriction in the previous section.

8. Parallel Implementation. This section describes the parallel implementation and gives a rough idea of the cost involved, with particular focus on the increase

truncation factor				max.# of weights			
θ	C_{op}	# its	time	k_{max}	C_{op}	# its	time
0	4.27	9	14.48				
0.1	4.13	9	10.72	7	3.75	9	8.63
0.2	3.88	9	8.52	6	3.42	9	7.41
0.3	3.39	10	6.82	5	3.01	10	6.42
0.4	3.02	13	6.60	4	2.73	14	6.30
0.5	2.75	20	7.67	3	2.48	24	7.41

TABLE 7.3

Effect of truncation on AMG with ext+i interpolation for a 7-point 3D Laplace problem on a $60 \times 60 \times 60$ *cube with a random right hand side. (Note that 'ass' denotes the average stencil size.)*

in communication required for the distance-two interpolation formulae compared to distance-one interpolation. Since the core computation for the interpolation routines is approximately the same as in the sequential case, we only focus on the additional computations that are required for inter-communication between processors.

In parallel, each matrix is stored using a parallel data format, the ParCSR matrix data structure, which is described and analyzed in detail in [8]. Matrices are distributed across processors by contiguous blocks of rows, which are stored via two compressed sparse row matrices, one storing the local entries and the other one storing the off-processor entries. There is an additional array containing a mapping for the off-processor neighbor points. The data structure also contains the information necessary to retrieve information from distance-one off-processor neighbors. It however does not contain information on off-processor distance-two neighbors, which complicates the parallel implementation of distance-two interpolation operators. When determining these neighbors, there are four scenarios that need to be considered, see Figure 8.1. Consider point i, which is the point to be interpolated to, and is residing on Processor 0. A distance-two neighbor can reside on the same processor as i, like point j; it can be a distance-one neighbor to another point on Proc. 0, like point l, and therefore be already contained in the off-processor mapping; it can be a new point on a neighbor processor, like point k, or it can be located on a processor, which is currently not a neighbor processor to Proc. 0, like point m.

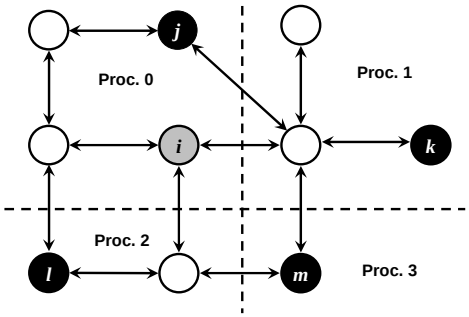

FIG. 8.1. *Example of off-processor distance-two neighbors of point i. Black points are C-points, and white points are F-points.*

There are basically five additional parts that are required for the parallel implementation, and for which we give rough estimates of the cost involved below.

Operations include floating point and integer operations as well as message passing and sends and receives required to communicate data across processors. We use the following notations: n_1 denotes the average number of distance-one neighbors per point, as defined previously, p is the total number of processors, q_i is the average number of distance-i neighbor processors per processor, N_i^o is the average number of distance-i off-processor points and equals the sum of the average number of distance-i off-processor C-points, C_i^o, and distance-i off-processor F-points, F_i^o. Note that the estimates of number of operations and number of processors involved given below are per processor.

1. Communication of C/F splitting for all off-processor neighbor points: This is required for all interpolation operators and takes $O(N_1^o) + O(q_1)$ operations.

2. Obtaining the row information for off-processor distance-one F-points: This step is necessary for classical and distance-two interpolation, but not for direct interpolation, which only uses local matrix coefficients to generate the interpolation formula. It requires $O(n_1 F_1^o) + O(F_1^o) + O(q_1)$ operations.

3. Determining off-processor distance-two points and additional communication information: This step is only required for distance-two interpolation operators. Finding the new off-processor points, which requires checking whether they are already contained in the map and describing the off-processor connections, takes $O(n_1 F_1^o \log(N_1^o))$ operations. Sorting the new information takes $O(N_2^o \log(N_2^o))$ operations. Getting the communication information for the new points using an assumed partition algorithm [1], requires $O(N_2^o) + O(\log p) + O((q_1 + q_2)\log(q_1 + q_2))$ operations. Getting the additional C/F splitting information takes $O(N_2^o) + O(q_1 + q_2)$ operations.

4. Communication of fine_to_coarse mappings: This step requires $O(N_1^o) + O(q_1)$ operations for distance-one interpolation and $O(N_1^o + N_2^o) + O(q_1 + q_2)$ operations for the distance-two interpolation schemes.

5: Generating the interpolation matrix communication package: This step requires $O(C_1^o) + O(\log p) + O(q_1 \log q_1)$ operations for distance-one interpolation and $O(C_1^o + C_2^o) + O(\log p) + O((q_1 + q_2)\log(q_1 + q_2))$ operations for distance-two interpolation. Note that if truncation is used, C_i^o should be replaced by $\tilde{C}_i^o$ with $\tilde{C}_i^o < C_i^o$ for $i = 1, 2$.

Summarizing these results, direct interpolation requires the least amount of communication, followed by classical interpolation. Parallel implementation of distance-two interpolation requires more communication steps and additional data manipulation, and involves more data and neighbor processors. How significantly this overhead impacts the total time depends on many factors, such as the problem size per processor, the stencil size, the computer architecture and more. Parallel scalability results are presented below.

9. Parallel Numerical Results. In this section, we investigate weak scalability of the new interpolation operators by applying the resulting AMG methods to various problems.

The following problems were run on Thunder, an Intel Itanium2 machine with 1024 nodes of 4 processors each, located at Lawrence Livermore National Laboratory, unless we say otherwise. In this section, p denotes the number of processors used.

9.1. Two-Dimensional Problems. We first consider two-dimensional Laplace problems, which perform very poorly for PMIS with classical interpolation. The results we obtained for 5-point and 9-point stencils are very similar. Therefore we list only the results for the 9-point 2D Laplace problem here.

Table 9.1, which contains the number of iterations and total times for this problem, show that classical interpolation performs very poorly, and multipass interpolation even worse. Nevertheless, these methods lead to the lowest operator complexities: 1.24. All long-range interpolation schemes lead to good scalable convergence, with standard-interpolation performing slightly worse than classical followed by Jacobi, extended or extended+i interpolation, which are the overall fastest methods here with the best scalability. Operator complexities are highest for clas+j with 1.65 and about 1.6 for the other three interpolation operators. Also, when choosing the lower complexity versions e+i-cc and e+i-ccs, with complexities of 1.45 and 1.43, convergence deteriorates somewhat compared to e+i. Since for the 2-dimensional problems setup times are fairly small and the improvement in complexities is not very significant, this increase in number of iterations hurts the total times, and therefore there is no advantage in using low-complexity schemes for this problem. Truncated versions lead to even larger total times.

p	clas	clas+j	mp	std	ext	e+i	e+i-cc	e+i-ccs
1	15(88)	3(9)	18(105)	4(15)	3(10)	3(10)	3(12)	3(13)
64	48(245)	4(11)	57(278)	6(20)	4(12)	4(12)	5(16)	5(19)
256	79(400)	5(12)	85(436)	8(27)	5(13)	5(13)	5(19)	6(21)
1024	104(494)	6(13)	n.c.	9(27)	6(14)	6(14)	7(21)	7(21)

TABLE 9.1

Times in seconds (number of iterations) for a 9-point 2D Laplace problem with 300×300 *points per processor; 'n.c.' denotes 'not converging within 500 iterations'.*

Next, we consider the 2-dimensional problem with rotated anisotropy (6.1). The first problem here has an anisotropy of 0.001 rotated by 45 degrees, see Table 9.2. Operator complexities for classical and multipass interpolation are here 1.9; they are 2.4 for classical interpolation followed by Jacobi, and 2.1 for all remaining interpolation operators. Here, extended interpolation performs worse than standard and extended+i interpolation, which gives the best results.

p	clas	clas+j	mp	std	ext	e+i	e+i-cc	e+i-ccs
1	24(116)	7(27)	24(119)	3(11)	7(29)	3(10)	3(12)	3(12)
64	n.c.	12(36)	96(401)	7(22)	11(39)	5(16)	7(21)	7(23)
256	n.c.	13(37)	n.c.	8(25)	12(42)	6(18)	8(25)	8(27)
1024	n.c.	15(40)	n.c.	10(29)	14(45)	8(21)	10(29)	11(31)

TABLE 9.2

Times in seconds (number of iterations) for a 2D problem with a by 45 degrees rotated anisotropy of 0.001 with 300×300 *points per processor; 'n.c.' denotes 'not converging within 500 iterations'.*

In Table 9.3 we consider the harder problem, where the anisotropy is rotated by 60 degrees. Operator complexities are now 1.8 for classical and multipass, 3.4 for clas+j, 2.9 for e+i and std, 2.7 for ext, 2.8 for e+i-cc and 2.5 for e+i-ccs. Here fastest convergence is obtained for the extended+i interpolation, followed by e+i-cc, e+i-ccs, std, and ext. The other interpolations fail to converge within 500 iterations. While long-range interpolation operators improve convergence, it is still not good enough, so this problem should be solved using Krylov subspace acceleration.

9.2. Three-Dimensional Structured Problems. We now consider 3-dimensional problems. Based on the sequential results in Section 6 we expect complexity reduction

p	clas	clas+j	mp	std	ext	e+i	e+i-cc	e+i-ccs
1	n.c.	105(342)	n.c.	30(107)	45(172)	22(79)	24(87)	28(112)
64	n.c.	n.c.	n.c.	79(256)	96(330)	47(152)	59(196)	70(254)
256	n.c.	n.c.	n.c.	95(305)	110(374)	56(176)	70(227)	84(299)
1024	n.c.	n.c.	n.c.	113(357)	123(408)	62(193)	82(263)	100(347)

TABLE 9.3

Times in seconds (number of iterations) for a 2D problem with a by 60 degrees rotated anisotropy of 0.001 with 300×300 *points per processor; 'n.c.' denotes 'not converging within 500 iterations'.*

schemes to make a difference here.

The first problem is a 7-point 3D Laplace problem on a structured cube with $40 \times 40 \times 40$ unknowns per processor. Table 9.4 shows total times in seconds, and number of iterations. While classical interpolation solves the problem, the number of iterations increases rapidly with increasing number of processors and problem size. Multipass interpolation performs better for larger number of processors, but still shows unscalable convergence factors. Applying one step of Jacobi interpolation to classical interpolation leads to perfect scalability in terms of convergence factors, but unfortunately also to rising operator complexities (4.9-5.7), which are twice as large as for classical and multipass interpolation (2.3-2.4). Interestingly, while both standard and extended+i interpolation need less iterations for a small number of processors than extended interpolation, they show worse numerical scalability leading to far less iterations for extended interpolation for large number of processors. However extended interpolation leads to larger complexities (4.7-5.3) compared to extended+i (4.2-4.5) and standard interpolation (4.1-4.4). The complexity reducing strategies lead to the following complexities: ext-cc (4.5-4.9), ext-ccs (3.9-4.2), e+i-cc (4.0-4.3), and e+i-ccs (3.6-3.8). For the sake of saving space, we did not record the results for ext-cc or e+i-cc, but the times and number of iterations for these methods was in between those of ext and ext-ccs, or e+i and e+i-ccs, respectively. Interestingly, the complexity reducing strategies e+i-cc and e+i-ccs show not only better scalability with regard to time, but also better scalability of convergence factors than e+i interpolation in this case.

p	clas	clas+j	mp	std	ext	e+i	ext-ccs	e+i-ccs
1	5(33)	8(11)	6(34)	5(9)	7(11)	6(8)	4(12)	3(9)
64	17(80)	18(12)	16(79)	14(18)	16(12)	12(10)	9(14)	7(11)
512	33(149)	26(12)	28(126)	20(26)	20(14)	17(15)	11(14)	11(11)
1000	39(175)	41(12)	31(138)	26(31)	30(13)	31(39)	15(15)	16(14)
1728	51(229)	63(12)	37(159)	35(41)	46(13)	40(33)	22(15)	24(16)

TABLE 9.4

Total times in seconds (number of iterations) for a 7 point 3D Laplace problem with $40 \times 40 \times 40$ *points per processor.*

For this problem, complexity reducing strategies, thus, are paying off. Table 9.5 shows results for various truncated interpolation schemes. We used the truncation strategy which restricts the number of weights per row using either 4 or 5 for the maximal number of elements. While we present both results for ext and e+i, we present only the faster results for the remaining interpolation schemes for the sake of saving space. We used a truncation factor of 0.1 for clas+j. Operator complexities

p	ext4	ext5	e+i4	e+i5	ext-cc5	e+i-cc5	std5	clas+j0.1
1	3(13)	3(11)	3(12)	3(9)	3(11)	3(9)	4(12)	3(13)
64	6(19)	7(15)	7(19)	7(13)	6(14)	6(13)	9(25)	9(23)
512	9(25)	8(18)	11(28)	10(19)	8(17)	8(17)	15(39)	13(36)
1000	10(25)	11(18)	11(30)	12(20)	10(18)	9(17)	17(39)	15(37)
1728	12(29)	12(21)	13(35)	14(24)	11(21)	11(20)	28(46)	19(45)

TABLE 9.5

Total times in seconds (number of iterations) for a 7 point 3D Laplace problem with $40 \times 40 \times 40$ *points per processor.*

were fairly consistent here across increasing numbers of processors: we obtained 2.9 for ext4, 3.2 for ext5, 2.8 for e+i4, 3.1 for e+i5, 3.2 for ext-cc5, 3.1 for e+i-cc5, 3.2 for std5 and 3.0 for clas+j0.1. Clearly, using four compared to five weights leads to lower complexities, but larger number of iterations. Total times are not significantly different. Comparing the fastest method, e+i-cc5, on 1728 processors to PMIS with classical interpolation, we see a factor of 11 in improvement with regard to number of iterations and a factor of 5 in improvement with regard to total time with a slight increase in complexity.

Table 9.6 shows results for the problem with jumps (6.2), for which PMIS with classical interpolation was shown to completely fail. Multipass interpolation converges here with highly degrading scalability but good complexities of 2.4. Applying Jacobi interpolation to classical interpolation leads to very good convergence, but, due to operator complexities between 5.1 and 5.7, it leads to a much more expensive setup and solve cycle. Applying a truncation factor of 0.1 as in the previous example leads to extremely bad convergence and is not helpful here. Standard interpolation converges very well for small number of processors, but diverges if p is greater or equal to 64. Interestingly enough std4 converges, albeit not very well.

p	mp	clas+j	ext	e+i	ext-ccs	e+i-ccs	std4	ext-cc5
1	11(64)	8(14)	7(14)	6(10)	5(18)	4(15)	6(26)	5(17)
64	35(176)	20(17)	17(17)	15(14)	11(21)	9(19)	18(71)	11(24)
512	58(280)	31(20)	24(24)	21(20)	15(24)	13(21)	27(98)	11(30)
1000	65(306)	35(21)	27(20)	26(21)	19(24)	18(22)	33(113)	14(33)
1728	77(350)	60(21)	73(70)	43(26)	25(29)	29(23)	53(169)	17(36)

TABLE 9.6

Total times in seconds (number of iterations) for a structured 3D problem with jumps with $40 \times 40 \times 40$ *points per processor.*

9.3. Unstructured Problems. In this section, we consider various linear systems on unstructured grids that have been generated by finite element discretizations. All of these problems were run on an Intel Xeon Linux cluster at Lawrence Livermore National Laboratory. The first problem is the three-dimensional diffusion problem $-a_1(x,y,z)u_{xx} - a_2(x,y,z)u_{yy} - a_3(x,y,z)u_{zz} = f$ with Dirichlet boundary conditions on an unstructured cube. The material properties are discontinuous, and there are approximately 90,000 degrees of freedom per processor. See Figure 9.1 for an illustration of the grid used. There are five regions: four layers and the thin stick in the middle of the domain. This grid is further refined when larger number of processors are used. The functions $a_i(x,y,z), i = 1,2,3$ are constant within each of

the five regions of the domains with the following values (4,0.2,1,1,10^4) for $a_1(x, y, z)$, (1,0.2,3,1,10^4) for $a_2(x, y, z)$, and (1,0.01,1,1,10^4) for $a_3(x, y, z)$. We also include some results obtained with CLJP coarsening, which is a parallel coarsening scheme that was designed to ensure that two fine neighbors always have a common coarse neighbor and for which classical interpolation is therefore suitable [5, 9]. We include only results for the extended+i interpolation here, because the extended interpolation performs very similar. As a smoother we used hybrid Gauss-Seidel, which leads to a nonsymmetric preconditioner. Since in practice more complicated problems are usually solved using AMG as a preconditioner for Krylov subspace methods, we use AMG here as a preconditioner for GMRES(10). Note that both classical and multipass interpolation do not converge within 1000 iterations for these problems if they are used without a Krylov subspace method, whereas both extended and extended+i interpolation, as well as classical interpolation on CLJP-coarsened grids, converge well without it, with a somewhat larger number of iterations and slightly slower total times.

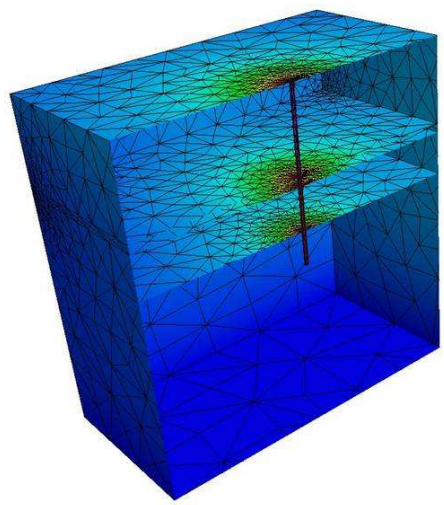

FIG. 9.1. *Grid for the elasticity problem.*

The results in Table 9.7 show that the long-range interpolation operators, with the exception of multipass interpolation, restore the good convergence that was obtained with CLJP. CLJP has very large complexities, however. We also used a truncated version of classical interpolation, restricting the number of weights per fine point to at most 4 to control the complexities. While this hardly affected convergence factors, it signifcantly improved the total times to solution, see Figure 9.2, but still did not achieve perfect scalability. Total times for CLJP with clas4 interpolation are comparable to PMIS with classical interpolation due to the small complexities of PMIS in spite of its significantly worse convergence factors. The use of extended+i and e+i-cc interpolation leads to better scalability than the methods mentioned before due to their lower complexities if compared to CLJP, or their better convergence factors if compared to PMIS with classical interpolation. Multipass interpolation leads to even better timings, but the overall best time and scalability is achieved by applying truncation to 4 weights per fine point to extended+i interpolation.

The second problem is a 3-dimensional linear elasticity problem using the same domain as above. However, a smaller grid size is used, since this problem requires more

	CLJP		PMIS				
p	clas	clas4	clas	mp	e+i	e+i-cc	e+i4
1	9(5.6)	9(4.2)	18(1.5)	20(1.5)	9(2.7)	10(2.2)	9(1.8)
64	11(6.7)	12(4.6)	62(1.5)	34(1.5)	11(3.0)	13(2.3)	13(1.9)
256	11(7.8)	12(5.0)	72(1.5)	34(1.5)	12(2.9)	12(2.3)	13(1.8)
512	11(7.2)	13(4.6)	118(1.5)	35(1.5)	12(3.0)	13(2.4)	12(1.8)
1024	10(8.6)	12(5.2)	162(1.6)	39(1.6)	12(3.4)	12(2.6)	14(2.0)

TABLE 9.7

Number of iterations (operator complexities) for the unstructured 3D problem with jumps. AMG is used here as a preconditioner for GMRES(10).

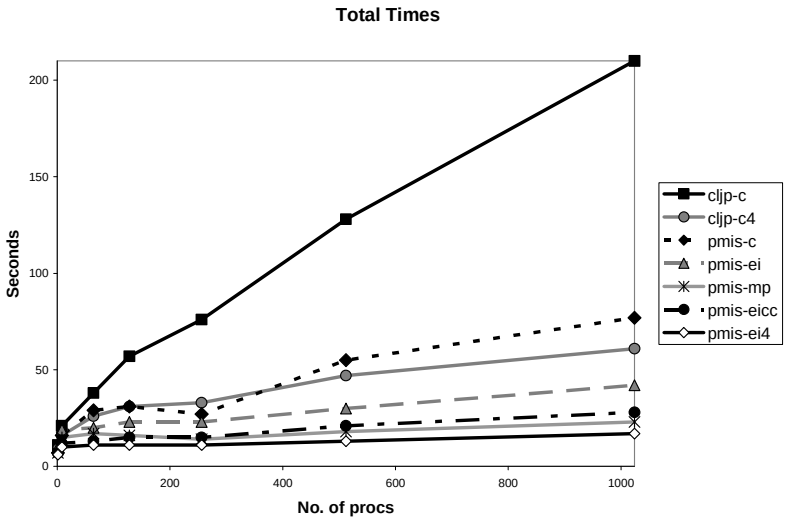

FIG. 9.2. *Total times for diffusion problem with highly discontinuous material properties. AMG is used here as a preconditioner for GMRES(10).*

memory, leading to about 30,000 degrees of freedom per processor. The Poisson ratio chosen for the pile driver in the middle of the domain was chosen to be 0.4, the Poisson ratios in the surrounding regions were 0.1, 0.3, 0.3 and 0.2. Since this a is a systems problem the unknown-based systems AMG method was used. For this problem, the conjugate gradient method was used as an accelerator, and hybrid symmetric Gauss-Seidel as a smoother. The results are given in Table 9.8 and Figure 9.3. Here also extended+i interpolation leads to the lowest run times and best scalability. CLJP ran out of memory for the 512 processor run.

10. Conclusions. We have studied the performance of AMG methods using the PMIS coarsening algorithm in combination with various interpolation operators. PMIS with classical, distance-one interpolation leads to an AMG method with low complexity, but has bad scalability in terms of AMG convergence factors. The use of distance-two interpolation operators restores this scalability. However, it leads to an increase in operator complexity. While this increase was fairly small for two-dimensional problems and was far outweighed by the much improved convergence, for three-dimensional problems complexities were often twice as large, and impacted scalability. To counter this complexity growth, we implemented various complexity

	CLJP		PMIS					
p	clas	clas4	clas	mp	e+i	e+i-cc	e+i-ccs	e+i4
1	64	63	94	93	68	69	72	72
8	83	84	159	131	89	95	96	90
64	92	96	210	179	97	105	112	107
512	-	112	319	247	108	109	123	123
C_{op}	4.5-7.3	3.6-5.4	1.5	1.5	2.5-3.0	2.1-2.4	1.9-2.1	1.9-2.0

TABLE 9.8

Number of iterations for the 3D elasticity problem; range of operator complexities. AMG is used here as a preconditioner for conjugate gradient.

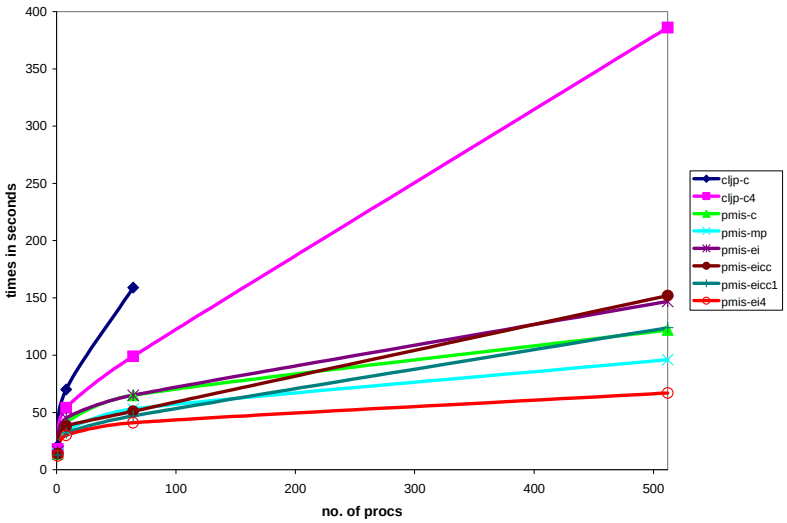

FIG. 9.3. *Total times for the 3D elasticity problem. AMG is used here as a preconditioner for conjugate gradient.*

reducing strategies, such as the use of smaller interpolatory sets and interpolation truncation. The resulting AMG methods, particularly the extended and extended+i interpolation in combination with truncation, lead to very good scalability for a variety of difficult PDE problems on large parallel computers.

Acknowledgments. We thank Tzanio Kolev for providing the unstructured problem generator and Jeff Painter for the Jacobi interpolation routine. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

REFERENCES

[1] A. Baker, R. D. Falgout, U. M. Yang, An assumed partition algorithm for determining processor inter-communication, *Parallel Computing* 32 (2006), 394–414.

[2] A. Brandt, S. F. McCormick, and J. W. Ruge, Algebraic multigrid (AMG) for sparse matrix equations, in *Sparsity and Its Applications*, D. J. Evans, ed., Cambridge University Press, Cambridge, 1984.

[3] J. Butler, *Improving coarsening and interpolation for algebraic multigrid*, Master's thesis in Applied Mathematics, University of Waterloo (2006).

[4] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial* (SIAM, Philadelphia, PA, second ed., 2000).

[5] A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones, Coarse grid selection for parallel algebraic multigrid, in *Proceedings of the fifth international symposium on solving irregularly structured problems in parallel* (Springer-Verlag, New York, 1998).

[6] A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, G. N. Miranda, and J. W. Ruge, Robustness and scalability of algebraic multigrid, *SIAM Journal on Scientific Computing,* 21 (2000), 1886–1908.

[7] H. De Sterck, U. M. Yang, and J. J. Heys, Reducing Complexity in Parallel Algebraic Multigrid Preconditioners, *SIAM Journal on Matrix Analysis and Applications* 27 (2006) 1019–1039.

[8] R. D. Falgout, J.E. Jones, and U.M. Yang, Pursuing scalability for hypre's conceptual interfaces, *ACM Trans. Math. Softw.* 31 (2005) 326–350.

[9] V. E. Henson and U. M. Yang, BoomerAMG: a parallel algebraic multigrid solver and preconditioner, *Applied Numerical Mathematics* 41 (2002) 155–177.

[10] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM Journal on Computing* 15 (1986) 1036–1053.

[11] J. W. Ruge and K. Stüben, Algebraic multigrid (AMG), in : S. F. McCormick, ed., *Multigrid Methods, vol. 3 of Frontiers in Applied Mathematics* (SIAM, Philadelphia, 1987) 73–130.

[12] K. Stüben, Algebraic multigrid (AMG): an introduction with applications, in : U. Trottenberg, C. Oosterlee and A. Schüller, eds., *Multigrid* (Academic Press, 2000).