Yale University
Department of Computer Science

DPMG: A Multigrid Solver for the Poisson Equation in Two and Three Dimensions

Craig C. Douglas

YALEU/DCS/TR-951
February 9, 1993

This work was supported in part by the Office of Naval Research (grant N00014-91-J-1576), Yale University, and the Research Division of International Business Machines.

DPMG: A MULTIGRID SOLVER FOR THE POISSON EQUATION IN TWO AND THREE DIMENSIONS*

CRAIG C. DOUGLAS†

Abstract. A fast Poisson solver (DPMG) based on multgrid methods is presented. DPMG can be used with boundary value problems defined on uniform or tensor product grids in two and three dimensions. The calling sequence is described in detail. What the subroutine library does and returns is also described. Examples of DPMG's use are included along with sample output.

Key words. multigrid, fast Poisson solver.

AMS(MOS) subject classifications. 65N20, 65F10, 65F05.

1. Introduction. In this paper, a subroutine library DPMG is discussed in detail. This is a fast, multigrid solver for Poisson's equation in two and three dimensions. The discretization techniques and storage methods are discussed in §1. A brief tutorial on multilevel methods is in §2. The details of how to call DPMG are in §3. Examples are in §4. How to make the package is in §A.

Consider Poisson's equation in a d–dimensional ($d \in \{2,3\}$) rectangular domain Ω:

$$\left\{ \begin{array}{rcl} -\triangle u & = & b \text{ in } \Omega, \\ u & = & g_0 \text{ on } \partial\Omega_0, \\ u_n & = & g_1 \text{ on } \partial\Omega_1, \end{array} \right. \tag{1}$$

where $\partial\Omega_0 \cup \partial\Omega_1 = \partial\Omega$ and $\partial\Omega_0 \cap \partial\Omega_1 = \emptyset$.

This is discretized on grids

$$\bar{\Omega} = \Omega \cup \partial\Omega_0 \cup \partial\Omega_1.$$

In essence, linear systems of the form

$$A_j x_j = b_j$$

are solved approximately for a sequence of grids $\bar{\Omega}_j$. The vectors x_j and b_j can be thought of as "grid functions" on $\bar{\Omega}_j$. The values of b, g_0, and g_1 on $\bar{\Omega}_j$ are stored in b_j (multiplied by the square of the mesh spacing when a uniform mesh is used). The values of g_0 on $\partial\Omega_0$ and an initial guess to the solution u in $\Omega \cup \partial\Omega_1$ are stored in x_j before the call to DPMG.

---

* Report YALEU/DCS/TR–951, Department of Computer Science, Yale University, New Haven, 1993.

† Department of Computer Science, Yale University, P. O. Box 208285, New Haven, CT 06520-8285 and Mathematical Sciences Department, IBM Research Division, Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598. E-mail: *na.cdouglas@na-net.ornl.gov*

DPMG uses a central difference discretization of Poisson's equation, even at Neumann boundary points. Dirichlet boundary points are not eliminated a priori.

The discrete system has a particular form depending on the mesh type (uniform or tensor). A_j is a $(2 \cdot NDIM + 1)$-point operator (i.e., there are at most $2 \cdot NDIM + 1$ nonzeros in any row of A_j). For $NDIM = 2$, A_j has the form

$$\begin{bmatrix}
a_{i,i} & a_{i,i+1} & & & a_{i,i+N_y} & & \\
a_{i,i+1} & \cdot & \cdot & & \cdot & & \\
 & \cdot & \cdot & \cdot & & \cdot & \\
 & & \cdot & \cdot & \cdot & & \cdot \\
a_{i,i+N_y} & & & \cdot & \cdot & & \cdot \\
 & \cdot & & & \cdot & \cdot & \\
 & & \cdot & & & \cdot & \cdot \\
 & & & \cdot & & & \cdot \\
 & & & & \cdot & & \cdot
\end{bmatrix}.$$

Not all of the a_{ik} listed above are nonzero. For example, if row l corresponds to a Dirichlet boundary point, then $a_{ll} = 1$ and $a_{lk} = 0$ if $l \neq k$. For non-Dirichlet points, the corresponding rows l all have

$$a_{ll} = \begin{cases} -4 & uniform\ mesh, \\ -4(h_1 + h_2) & tensor\ mesh, \end{cases}$$

where h_1 and h_2 are the mesh spacing.

For an interior point, with a corresponding row l in A_j,

$$a_{l,l+1} = \begin{cases} 1 & uniform\ mesh, \\ -2h_1 & tensor\ mesh, \end{cases}$$

and

$$a_{l,l+N_y} = \begin{cases} 1 & uniform\ mesh, \\ -2h_2 & tensor\ mesh. \end{cases}$$

For Neumman boundary points, a "reflection" principal is used. This has the effect of doubling the coefficient along the same line. For example, consider a boundary point on the left side of the domain. Then the corresponding row l in A_j is a 4 point operator:

$$[\cdots a_{l-1,l} a_{ll} a_{l,l+1} \cdots 2a_{l,l+N_y} \cdots].$$

At Neumann corner points, the corresponding row l in A_j is a 3 point operator with both of the nonzero entries off the main diagonal doubled, e.g., at the lower left corner:

$$[a_{11} 2a_{12} \cdots 2a_{1,1+N_y} \cdots].$$

When $NDIM = 3$, A_j has the form

$$\begin{bmatrix}
a_{i,i} & a_{i,i+1} & & a_{i,i+N_y} & & & a_{i,i+N_yN_x} & & \\
a_{i,i+1} & \cdot & \cdot & & & \cdot & & \cdot & \\
 & \cdot & \cdot & \cdot & & & \cdot & & \cdot \\
 & & \cdot & \cdot & \cdot & & & \cdot & \\
a_{i,i+N_y} & & & \cdot & \cdot & \cdot & & & \cdot \\
 & \cdot & & & \cdot & \cdot & \cdot & & \\
a_{i,i+N_yN_x} & & \cdot & & & \cdot & \cdot & \cdot & \\
 & \cdot & & \cdot & & & \cdot & \cdot & \cdot \\
 & & \cdot & & \cdot & & & \cdot & \cdot
\end{bmatrix}.$$

At the non-Dirichlet points, the corresponding rows l have a main diagonal entry of

$$a_{ll} = \begin{cases} -6 & uniform\ mesh, \\ -4(h_1 + h_2 + h_3) & tensor\ mesh. \end{cases}$$

For an interior point, with a corresponding row l in A_j,

$$a_{l,l+1} = \begin{cases} 1 & uniform\ mesh \\ -2h_1 & tensor\ mesh, \end{cases}$$

and

$$a_{l,l+N_y} = \begin{cases} 1 & uniform\ mesh, \\ -2h_2 & tensor\ mesh, \end{cases}$$

and

$$a_{l,l+N_yN_x} = \begin{cases} 1 & uniform\ mesh, \\ -2h_3 & tensor\ mesh. \end{cases}$$

Dirichlet points are once again identity rows and the Neumann points are discretized again by reflections.

In the uniform mesh case, the right hand side b_j must have the square of the mesh spacing (i.e., h^2) factored into it. Otherwise the approximate solution returned will be off by a factor of h^{-2}.

2. Introduction to multigrid methods. Once a linear differential equation is discretized, we must solve

$$Ax = b, \quad x \in \mathcal{M},$$

where $\mathcal{M}$ is a vector space. We will solve this using an abstract multilevel (or multigrid) iteration. An auxiliary set of equations are used which each approximate the original one:

$$A_j x_j = b_j, \quad levelf \le j \le levelc, \quad x_j \in \mathcal{M}_j,$$

where *levelc* and *levelf* are natural numbers signifying the coarsest and finest levels, respectively, $A_{levelf} = A$, $x_{levelf} = x$, and $b_{levelf} = b$.

Multigrid solvers frequently use particular features of an elliptic boundary value problem and the domain. There are similar procedures, known as aggregation-disaggregation methods, when A is not derived from partial differential equations; this routine can be used with either of these procedures. The term multigrid is usually applied only to problems based on grids, whereas the term multilevel is applied to problems which may or may not be grid based.

Multilevel methods combine scaled iterative methods (called *smoothers* or *roughers*) with iterative residual correction on coarser levels to reduce the error on a given level. Iteration i on some level $j > levelc$ consists of a smoothing step (introducing an operator $S_j^{(i)}$), a correction step, and another smoothing step (introducing another operator $T_j^{(i)}$). There are μ_j of these iterations. On level $j = levelc$, just smoothing occurs (say, $S_j^{(i)}$); this may be an iterative or a direct procedure like sparse Gaussian elimination.

Common smoothers $S_j^{(i)}$ and $T_j^{(i)}$ are relaxation methods (e.g., Gauss-Seidel, SOR, line or plane methods), preconditioned conjugate direction methods (e.g., conjugate gradients, minimum residuals, Orthomin), and the identity operator.

The correction step involves a two way transfer of information between levels. This is accomplished using mappings between the solution spaces:

$$R_j : \mathcal{M}_j \to \mathcal{M}_{j+1} \quad \text{and} \quad P_{j+1} : \mathcal{M}_{j+1} \to \mathcal{M}_j. \tag{2}$$

These are referred to as *restriction* and *prolongation* operators in the multigrid literature. Typically, P_{j+1} is a standard interpolation operator and R_j is its transpose.

There are two basic linear multilevel algorithms: correction ones and nested iteration ones. Correction multilevel algorithms start on a fine grid and use the coarser levels solely to correct the approximate solution on finer levels (we will define two such algorithms shortly, namely, MGC and MGFAS). Nested iteration multilevel algorithms start on a coarse level and work their way to some finer level, using the approximate solution on coarser levels to produce initial guesses and corrections on the finer levels (we will define two such algorithms shortly, namely, NIC and NIFAS).

Define a k-level correction multigrid algorithm by

ALGORITHM MGC$(k, \{\mu_\ell\}, x_k, f_k)$
(1) If $k = levelc$, then solve $A_k x_k = f_k$ exactly or iteratively
(2) If $k \neq levelc$, then repeat $i = 1, \cdots, \mu_k$:
(2a) Smoothing: $x_k \leftarrow S_k^{(i)}(x_k, f_k)$
(2b) Residual Correction: $x_k \leftarrow x_k +$
$P_{k+1}($ MGC$(k+1, \{\mu_\ell\}, 0, R_k(A_k x_k + f_k)))$
(2c) Smoothing: $x_k \leftarrow T_k^{(i)}(x_k, f_k)$
(3) Return x_k

This definition requires that $\mu_{levelc} = 1$. Examples of the flow of control between levels for the correction algorithm are contained in Figure 1.

Define a k-level nested iteration multigrid algorithm by

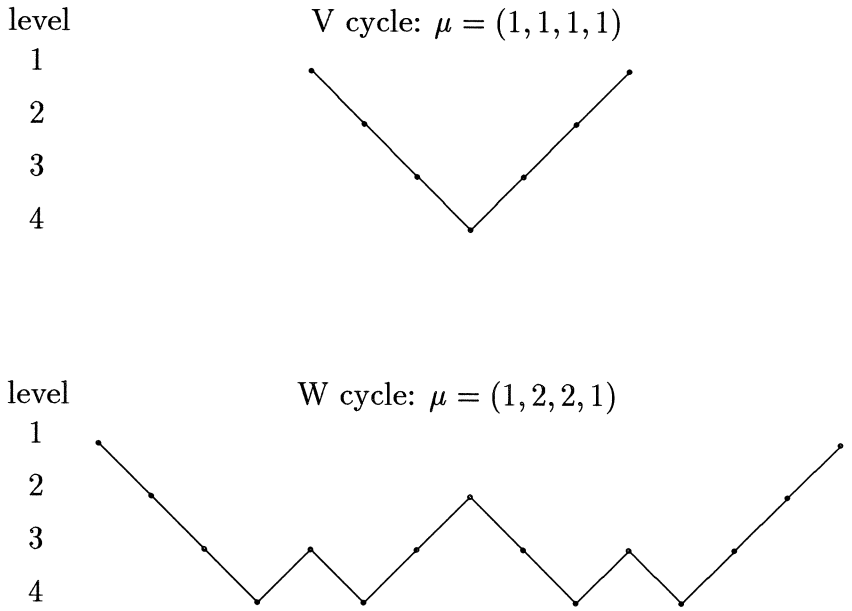

FIG. 1. *Correction Algorithm (MGC) V and W cycles*

ALGORITHM NIC($k, \{\mu_\ell, \psi_\ell\}, x_{levelc}, \{f_\ell\}$)

(1) For $j = levelc, levelc-1, \cdots, k$, do

($1a$) If $j \neq levelc$, then $x_j \leftarrow P_{j+1}x_{j+1}$

($1b$) Set $\mu \leftarrow \mu_j$ and then $\mu_j \leftarrow \psi_j$.

($1c$) $x_j \leftarrow$MGC($j, \{\mu_\ell\}, x_j, f_j$)

($1d$) Restore $\mu_j \leftarrow \mu$.

(2) Return x_k

An example of the flow of control between levels for both of the nested iteration algorithm is contained in Figure 2.

3. Subroutine DPMG. Algorithms MGC and NIC have been encapsulated in the long precision subroutine DPMG, tailored for Poisson's equation (1). It calls the correct multilevel algorithm subroutine(s), which in turn calls the appropriate interpolation, projection, direct solver, and iterative solver routines. DPMG uses the subroutine DAMG [5] to do some of its work.

3.1. Syntax of DPMG. DPMG can be called from either FORTRAN or C using the following convention:

FORTRAN	*CALL DPMG (*	*npts, h, b, x, iparm, resid, bndcnd, aux, naux)*
C	*dpmg (*	*npts, h, b, x, iparm, resid, bndcnd, aux, &naux)*

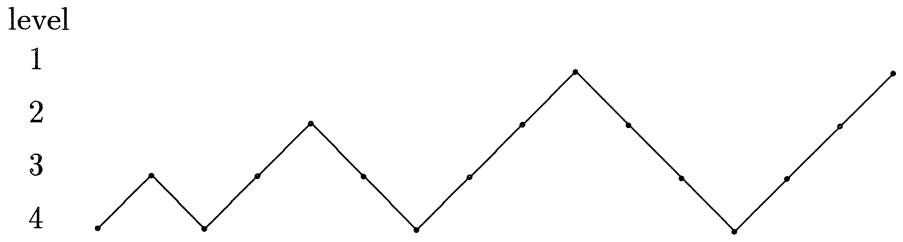

FIG. 2. *Nested Iteration Algorithm (NIC) V cycle*

3.2. On entry to DPMG. The arguments to DPMG have the following meaning:

npts

is a vector with the number of grid points in each dimension (Y, X, and Z coordinates) for the finest grid. Each entry must be odd.
Specified as: a vector of fullword integers of length at least *ndim* (see *iparm*).

h

is a vector with the mesh spacing in each dimension (Y, X, and Z coordinates), dimensioned at least *ndim* (see *iparm* and the Notes).
If $mesh = 1$ in *iparm*, then this is ignored.
Specified as: a vector of long precision real numbers of length at least *ndim* (see *iparm*).

b

is a vector containing the right hand sides b_j, stacked one after the next. See the Notes for a description of how to dimension b and how to stack the b_j inside of b.
Specified as: a vector of long precision real numbers of length at least *bxsize* (see *iparm*).

x

is a vector containing the approximate solutions or corrections x_j, stacked one after the next. See the Notes for a description of how to dimension x and how to stack the x_j inside of x.
Specified as: a vector of long precision real numbers of length at least *bxsize* (see *iparm*).

iparm

is a vector of fullword integer arguments.

- $iparm(1) = mgfn$ determines which of the multilevel algorithms to use:
  1 MGC
  2 NIC
- $iparm(2) = mesh$ determines which type of mesh to use:
  1 uniform spacing in all directions.
  2 tensor product with the spacing specified in h.
- $iparm(3) = bxsize$ is the amount of space of the vectors b and x. See the Notes for a description of how to dimension b and x.
- $iparm(4) = lna$ is the size of the a and ja vectors inside of aux. This is a reserved spot and should not be used.
- $iparm(5) = ndim$ is the number of dimensions of the problem, either 2 or 3.
- $iparm(6) = levels$ is the number of levels. One is the finest level and *levels* is the coarsest one. This must be less or equal to 20, but using more than 5 levels is usually pointless. There is no default.
- $iparm(7) = itrni$ is the number of iterations of NIC. This corresponds to the $\{\psi_\ell\}$ in the definition of NIC in §2. The default is 1, but either 1 or 2 is typical.
- $iparm(8) = itrmg$ is the number of iterations of MGC. This corresponds to the $\{\mu_\ell\}$ in the definition of MGC in §2. The default is 2, but either 1 or 2 is typical.
- $iparm(9) = itrsm$ is the number of iterations of the smoother. The default is 2, but any value in the 1–4 range is typical.
- $iparm(10) = smethd$ is the smoother (see Table 1). See the Notes for suggestions.
- $iparm(11) = dscoar$ is whether or not to use a sparse direct solver (DGSF/DGSS) on the coarsest level. If $iparm(11) = 1$, then factorization is required. If $iparm(11) = 2$, then use the smoother specified by *smethd* instead of the direct solver. If $iparm(11) = 3$, then use the factorization from a previous call to pmg. The default is 1.

TABLE 1
DPMG Solver Information

Solver	Symbolic name	Definition
5	GSNat	Gauss-Seidel, natural ordering
6	GSRedBlack	Gauss-Seidel, red-black ordering

See DAMG for an explanation of the numbering system.

- $iparm(12)$ = *proni* is the prolongation method for Algorithm NIC (see (2)). If $iparm(12) = 1$, then use a second order approximation (i.e.., bilinear or trilinear interpolation). If $iparm(12) = 2$, then use a fourth order approximation (LIM). See the Notes for an explanation of the different methods. The default is 1 if $mesh = 2$ and 2 if $mesh = 1$.
- $iparm(13)$ = *promg* is the prolongation method for Algorithm MGC (see (2)). The only value currently accepted is 1 for a second order approximation (i.e.., bilinear or trilinear interpolation).
- $iparm(14)$ = *rmethd* is the weighting used in the residual correction method in Algorithm MGC (see (2)). If $iparm(14) = 1$, then use a fourth order approximation. If $iparm(14) = 2$, then use a second order approximation based on discrete ℓ_2 operator. If $iparm(14) = 3$, then use a second order approximation based on (bi,tri)linear projection. See the Notes for an explanation of the different methods. The default is 1.
- $iparm(15)$ = *defaults* is used to let DPMG pick reasonable values for most of the elements of *iparm*. These are as follows: $iparm(1) = 1$, $iparm(2) = 1$, $iparm(6)$ =maximum possible, $iparm(8) = 2$, and $iparm(9) = 2$. This corresponds to Algorithm MGC on uniform meshes with the number of levels maximized, a red-black Gauss-Seidel iteration for a smoother, a direct solve on the coarsest level, 2nd order interpolation, and 4th order projection. The user must remember to include the h^2 factor in the right hand side b.

TABLE 2
BNDCND Description

When NDIM=2, *bndcnd*(i) corresponds to

i	*Side*	*x and b elements*
1	*Left*	$(1 : npts(1),\ 1)$
2	*Right*	$(1 : npts(1),\ npts(2))$
3	*Bottom*	$(1,\ 1 : npts(2))$
4	*Top*	$(npts(1),\ 1 : npts(2))$

When NDIM=3, *bndcnd*(i) corresponds to

i	*Side*	*x and b elements*
1	*Left*	$(1 : npts(1),\ 1,\ 1 : npts(3))$
2	*Right*	$(1 : npts(1),\ npts(2),\ 1 : npts(3))$
3	*Bottom*	$(1 : npts(1),\ 1 : npts(2),\ 1)$
4	*Top*	$(1 : npts(1),\ 1 : npts(2),\ npts(3))$
5	*Front*	$(1,\ 1 : npts(2),\ 1 : npts(3))$
6	*Back*	$(npts(1),\ 1 : npts(2),\ 1 : npts(3))$

See DAMG for an explanation of the numbering system.

resid

is a vector containing anything.
Specified as: a vector of long precision real numbers of length at least

$$npts(1) \cdot npts(2) \quad \text{if } ndim = 2,$$
$$npts(1) \cdot npts(2) \cdot npts(3) \quad \text{if } ndim = 3.$$

bndcnd

is a vector containing the boundary conditions. See Table 2 for the ordering of the elements.
Specified as: a vector of fullword integers of length at least $2 \cdot ndim$ (see *iparm*).

aux

is the storage work area used by this subroutine. Its size is specified by *naux*.
Specified as: a vector of long precision real numbers of length *naux*.

naux

is the size of the floating point scratch storage.
Specified as: a fullword integer.

3.3. On return from DPMG. The following arguments to DPMG may change before it returns:

b

contains the right hand side for the finest level and is destroyed on the other levels.

x

contains the approximate solution for the finest level and is destroyed on the other levels.

resid

is a vector where the residuals are stored.
Specified as: a vector of long precision real numbers of length at least

$$\begin{array}{ll} npts(1) \cdot npts(2) & \text{if } ndim = 2, \\ npts(1) \cdot npts(2) \cdot npts(3) & \text{if } ndim = 3. \end{array}$$

aux

is destroyed in unpredictable ways. However, if DPMG is ever going to be called with $dscoar = 3$, *aux* must not be changed between calls.

naux

is the estimate for what *naux* ought to have been if the value supplied in the call to DPMG is too small.

3.4. Errors associated with DPMG. There are three classes of errors: input, input or computational, and computational ones.

3.4.1. Input errors.

1. The number of grid points ($npts(i)$) is not an odd number greater than 2.
2. *bxsize* is too small.
3. *mgfn* is not 1 or 2.
4. The number of dimensions (*ndim* in *iparm*) is not 2 or 3.
5. *levels* (see *iparm*) is less or equal to 0 or is too large. The latter occurs when $levels > 20$ or when the number of grid points in some direction on some level becomes too small or even.
6. *itrni* is negative.
7. *itrmg* is negative.
8. *itrsm* is negative.
9. *smethd* is not 0, 5, or 6.
10. *dscoar* is not 0, 1, 2, or 3.
11. An interpolation method (*pronic* in *iparm*) is requested that is incompatible with the mesh. This occurs when LIM is requested, but a nonuniform mesh ($mesh = 2$ in *iparm*) is used.
12. *pronic* is not 0, 1, or 2.
13. *promgc* is not 0 or 1.
14. *rmethd* is not 0, 1, 2, 3.
15. At least one of the mesh spacings in h is not positive.
16. An element in *bndcnd* is not 0 or 1.

17. *defaults* not 0 or 1.

3.4.2. Input or Computational Errors.

1. *naux* is not large enough.

3.4.3. Computational Errors.

1. The sparse direct solver, used on the coarsest level, could not cope with the vector sizes. Making *aux* larger solves this problem.

3.5. Notes about DPMG.

1. DPMG assumes an odd number of grid points in each direction on all of the levels, i.e., an even number of intervals in each direction. Suppose there are NF grid points in a direction on some level. On the next coarser level there will be $NC = (NF + 1)/2$ grid points in that direction. The distance between any two points in any direction (this is referred to as the mesh spacing, see h) is constant. When the mesh spacing is the same in each direction then the mesh is referred to as uniform; otherwise it is tensor product of the one dimensional meshes in each direction (see *mesh* in *iparm*).
2. The right hand sides $\{b_j\}$ and approximate solutions $\{x_j\}$ are stacked one after each other in the b and x vectors. The finest level's vectors are stored first, followed immediately by the next to finest level's vectors, and so forth. Suppose there are 17^2, 9^2, and 5^2 grid points on 3 levels. This corresponds to $npts(1) = npts(2) = 17$, $npts(3) = 1$, $ndim = 2$, and $levels = 3$. Then

Level (j)	*Number of points*	*Locations in b and x for* b_j *and* x_j
1	289	$1 - 289$
2	81	$290 - 370$
3	25	$371 - 395$

   The minimum for *bxsize* is 395 in this example. Each x_j begins in x at the same location as the corresponding b_j in b.
3. While the order of storage for b and x is (y,x,z), the program will run fastest if the number of unknowns N_y, N_x, and N_z in each dimension is ordered so that $N_y \geq N_x \geq N_z$. This may require transforming the problem.
4. When a direct solve is used on the coarsest level, a matrix using a $2 \cdot ndim + 1$ point stencil is generated and factored by DPMG. The sparse direct factorization routine is called exactly once to factor the matrix and the sparse solver is called to do the actual solves during the execution of a typical call to DPMG. The "storage by rows" sparse matrix format is used (see DAMG). One REAL*8 vector A along with two INTEGER*4 vectors IA and JA are generated inside of *aux*.

   Suppose there are N total grid points on the coarsest level. Vectors IA, JA, and A must be of length greater than

$$(2 \cdot ndim + 1)N$$

each. In addition, 4 words of *aux* are used so that DPMG can be called again. Also, an additional

$$10N + 100$$

locations of *aux* are used by DGSF and DGSS (when the ESSL version of DPMG is used) to store information. Actually, the size of IA, JA, and A is mostly determined by what the sparse factorization routine requires to factor the matrix. For example, on square or cube domains with M grid points in each dimension, with DGSF,

N	*naux minimum*	*naux realistic*
M^2	$20N + 120$	$2N^{3/2} + 20N + 120$
M^3	$24N + 120$	$2N^{5/3} + 24N + 120$

The potential fill in during Gaussian elimination should be at worst

$$\begin{cases} 2M^3 & if\ ndim = 2, \\ 2M^5 & if\ ndim = 3. \end{cases}$$

This is pretty grim since it has to be added to the minimum. Luckily, the N is for the coarsest level, not the finest. See the description of DGSF to determine the actual amount of space that should be allocated to IA, JA, and A.
Some hints are as follows based on DGSF not having to do any compressions:

$ndim = 2$		$ndim = 3$	
N	*naux*	N	*naux*
3^2	400	3^3	850
5^2	1250	5^3	8200
7^2	3100	7^3	53000
9^2	4900	9^3	144000
17^2	58000		

Note that all of *aux* is used no matter how much extra space is given to DPMG by the user. This is a greedy routine.
Finally, most of the computational errors are caused by *naux* (and/or *lna* in *iparm*) being too small. Increasing *naux* will eliminate most error conditions.

5. One of the interpolation methods for Algorithms NIC is a fourth order method referred to as LIM (Local Inversion Method, see [6]. This should only be used with a uniform mesh ($mesh = 1$); it is an error otherwise. This uses the difference operator, similar to a Gauss-Seidel iteration with a three color ordering and a rotated operator, to improve the order of interpolation.
6. The restriction methods are based on stencils. These are described in detail in [2]. The two second order methods are based on $[1, 2, 1]$ and $[1, 4, 1]$ weightings in one dimension. Tensor products are used to generate the stencils in higher dimensions. The fourth order stencil is an average of the $[1, 4, 1]$ tensor product stencil and injection.

7. When the mesh spacing is identical in each direction, the best choices for interpolation and restriction methods are as follows: $rmethd = 1$, $proni = 2$, and $promg = 1$. The theoretical justification for these choices is contained in [2] and [3].
8. When the mesh spacing is not identical in each direction, the best choices for interpolation and restriction methods are as follows: $rmethd = 3$, $proni = 1$, and $promg = 1$.
9. The standard choices for the number of smoothing iterations are 1–4. One of the principals aims of multigrid methods is to reduce the number of iterations of some iterative procedure. This is done by changing levels often while not doing much computation on any level at any one time. This is discussed in detail in [1] and [7].
10. Boundary conditions are determined for each side of the domain as a block, not pointwise (currently). The supported boundary conditions are as follows:

Condition	*Value*
Dirichlet	0
Neumann	1

Whenever one side has a Neumann condition and an adjoining one has a Dirichlet condition, the Dirichlet condition is assumed at the corner point where the boundaries intersect. See Table 2 and the description of *bndcnd*.
WARNING: There must be at least one boundary point with a Dirichlet condition or the problem is ill posed.

4. Examples of DPMG Usage. All runs in this section were on an IBM RISC SYSTEM/6000™. In §4.1, a simple two dimensional problem is explored in depth. In §4.2, a three dimensional problem is examined.

It is implicitly assumed that you have already made a working version of DPMG and DAMG. If you do not know how to do this, please see Appendix A.

4.1. Example 1: a two dimensional problem. The first example solves

$$\begin{cases} U_{xx} + U_{yy} = F & \text{in the unit square } (0,1)^2 \\ U(x,y) = 0 & \text{on } \partial(0,1)^2. \end{cases}$$

F is chosen so that the solution is

$$U(x,y) = x \cdot (x-1) \cdot y \cdot (y-1)$$

or

$$F(x,y) = 2[x \cdot (x-1) + y \cdot (y-1)].$$

Algorithm NIC is used with the red-black Gauss-Seidel smoother on all levels except the coarsest where a direct solver is used. A uniform mesh is used. LIM is used for the

NIC interpolation, the fourth order projection method, and bilinear interpolation for MGC.

Due to the uniform mesh and the discretization method, DPMG requires F in the following form:

$$F(x,y) = 2h^2[x \cdot (x-1) + y \cdot (y-1)],$$

where h is the mesh spacing. So, on level i, the grid points are

$$y_j = (j-1) \cdot h_i, \quad j = 1, \cdots, N_{i,y}$$

and

$$x_k = (k-1) \cdot h_i, \quad k = 1, \cdots, N_{i,x}.$$

Thus, the right hand side on level i is

$$B_i(y_j, x_k) = 2h_i^2[x_k \cdot (x_k - 1) + y_j \cdot (y_j - 1)]. \tag{3}$$

Note that elements of B_i corresponding to boundary points are zero in (3) due to the contrived nature of the example.

A sample FORTRAN main program is presented here. The declarations section is simply,

```
      program main
      integer npts(2) / 5, 5 /
      real*8 h(2) / 0.0, 0.0 /
      real*8 b(34)
      real*8 x(34)
      integer iparm(40) / 40 * 0 /
      real*8 resid(25)
      integer bndcnd(4) / 0, 0, 0, 0 /
      real*8 aux(400)
      integer naux / 400 /
      real*8 hh
      integer i, l2
```

The right hand sides and initial guesses are made by calls to a subroutine bxfn1:

```
      l2 = npts(1) * npts(2) + 1
      hh = 1.0 / (npts(1) - 1)
      call bxfn1( hh, npts(1), npts(2), x, b )
      call bxfn1( 2*hh, (npts(1)+1)/2, (npts(2)+1)/2, x(l2), b(l2) )
```

The various nonzero entries in iparm are filled in:

```
iparm( 1) = 2          % mgfn = nic
iparm( 2) = 1          % mesh = uniform
iparm( 3) = 34         % bxsize
iparm( 5) = 2          % ndim
iparm( 6) = 2          % levels
iparm( 7) = 1          % itrni
iparm( 8) = 1          % itrmg
iparm( 9) = 2          % itrsm
iparm(10) = 6          % smethd = red-black Gauss-Seidel
iparm(11) = 1          % dscoar = factorization needed
iparm(12) = 2          % proni = 2nd order
iparm(13) = 1          % promg = 2nd order
iparm(14) = 1          % rmethd = 4th order
```

DPMG can now be called using the following:

```
call dpmg( npts, h, b, x, iparm, resid, bndcnd, aux, naux )
```

Finally, the results are printed using the following:

```
write (*,*) 'X ='
write (*,'(1p,(5d14.5))') (x(i), i=1,25)
write (*,*) 'RESID ='
write (*,'(1p,(5d14.5))') (resid(i), i=1,25)
end
```

This produces the following output:

```
 X =
0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00
0.00000D+00  3.53275D-02  4.72175D-02  3.53326D-02  0.00000D+00
0.00000D+00  4.72175D-02  6.28526D-02  4.72378D-02  0.00000D+00
0.00000D+00  3.53326D-02  4.72378D-02  3.53377D-02  0.00000D+00
0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00
 RESID =
0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00
0.00000D+00 -6.93889D-18  6.69691D-04  6.93889D-18  0.00000D+00
0.00000D+00  6.69691D-04  0.00000D+00  7.40899D-04  0.00000D+00
0.00000D+00  0.00000D+00  7.40899D-04 -1.38778D-17  0.00000D+00
0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00  0.00000D+00
```

The subroutine bxfn1 is quite simple. One thing to note is that b_i and x_i are treated as matrices in this subroutine while they are just sections of a larger vector in the main program. While this feature of FORTRAN is portable with respect to FORTRAN-77, it is not a very good programming habit.

```
subroutine bxfn1( h, n1, n2, x, b )
real*8 h
integer n1, n2
real*8 x(n1,n2)
real*8 b(n1,n2)
real*8 zero / 0.0 /
real*8 hfac, xfac, xk, yj
integer j, k
hfac = 2.0 * h ** 2
call dcopy( n1*n2, zero, 0, x, 1 )
call dcopy( n1*n2, zero, 0, b, 1 )
do k = 2,n2-1
      xk = (k - 1) * h
      xfac = xk * ( xk - 1.0 )
      do j = 2,n1-1
            yj = (j - 1) * h
            b(j,k) = hfac * ( xfac + yj * ( yj - 1.0 ) )
      enddo
enddo
return
end
```

On a machine where double precision corresponds to 128 bits, the call to dcopy above has to be changed to a call to scopy. Also, some FORTRAN compilers do not recognize real*8 as a legal construct; the obvious change must then be made.

We could just as well have designed subroutine bxfn1 to construct right hand sides for uniform and tensor product grids. For example, if the line

```
hfac = 2.0 * h ** 2
```

in bxfn1 is changed to

```
hfac = 2.0
```

and the line

```
iparm( 2) = 1          % mesh = uniform
```

in the main program is changed to

```
iparm( 2) = 2          % mesh = tensor
```

then we can compare results. What we discover is that the solution X is identical for both cases, but that the elements of the residual vectors are different by exactly a factor of h^2. This is exactly what we would expect to happen.

4.2. Example DPMG-2: 3 dimensional problem. The second example solves

$$\begin{cases} U_{xx} + U_{yy} + U_{zz} = F & \textit{in the unit cube } (0,1)^3 \\ \\ U(x,y,z) = 0 & \textit{on } \partial(0,1)^3. \end{cases}$$

F is chosen so that the solution is

$$U(x,y,z) = x \cdot (x-1) \cdot y \cdot (y-1) \cdot z \cdot (z-1)$$

or

$$F(x,y,z) = 2[x \cdot (x-1) + y \cdot (y-1) + z \cdot (z-1)].$$

where h is the mesh spacing. So, on level i, the grid points are

$$z_m = (m-1) \cdot h_i, \quad m = 1, \cdots, N_{i,z},$$

$$y_j = (j-1) \cdot h_i, \quad j = 1, \cdots, N_{i,y},$$

and

$$x_k = (k-1) \cdot h_i, \quad k = 1, \cdots, N_{i,x}.$$

Thus, the right hand side on level i is

$$B_i(y_j, x_k, z_m) = 2h_i^2[x_k \cdot (x_k - 1) + y_j \cdot (y_j - 1) + z_m \cdot (z_m - 1) +]. \tag{4}$$

Note that elements of B_i corresponding to boundary points are zero in (4) due to the contrived nature of the example.

A sample FORTRAN main program is presented here. The declarations section is simply,

```
program main
integer npts(3) / 5, 5, 5 /
real*8 h(3) / 0.25, 0.25, 0.25 /
real*8 b(152)
real*8 x(152)
integer iparm(40) / 40 * 0 /
real*8 resid(125)
integer bndcnd(6) / 0, 0, 0, 0, 0, 0 /
real*8 aux(1300)
integer naux / 1300 /
real*8 hh
integer i, j, l2
```

The various nonzero entries in iparm are filled in:

```
iparm( 1) = 2        % mgfn = nic
iparm( 2) = 1        % mesh = uniform
iparm( 3) = 152      % bxsize
iparm( 5) = 3        % ndim
iparm( 6) = 2        % levels
iparm( 7) = 1        % itrni
iparm( 8) = 1        % itrmg
iparm( 9) = 2        % itrsm
iparm(10) = 6        % smethd = red-black Gauss-Seidel
iparm(11) = 1        % dscoar = factorization needed
iparm(12) = 2        % proni = 2nd order
iparm(13) = 1        % promg = 2nd order
iparm(14) = 1        % rmethd = 4th order
```

The right hand sides and initial guesses are made by calls to a subroutine bxfn2:

```
      l2 = npts(1) * npts(2) * npts(3) + 1
      hh = 1.0 / (npts(1) - 1)
      call bxfn2( iparm(2), hh, npts(1), npts(2), npts(3), x, b )
      call bxfn2( iparm(2), 2*hh, (npts(1)+1)/2, (npts(2)+1)/2,
     &                  (npts(3)+1)/2, x(l2), b(l2) )
```

Note that the reason why the order of sections differs from that of the main program in §4.1 is due only to using the value of *iparm*(2) in the call to bxfn2. DPMG can now be called using the following:

```
      call dpmg( npts, h, b, x, iparm, resid, bndcnd, aux, naux )
```

Finally, the results are printed using the following:

```
      write (*,*) 'X ='
      do i = 1,125,25
            write (*,'(1p,(5d14.5))') (x(j), j=i,i+24)
            write (*,*) ' '
      enddo
      write (*,*) 'RESID ='
      do i = 1,125,25
            write (*,'(1p,(5d14.5))') (resid(j), j=i,i+24)
            write (*,*) ' '
      enddo
      end
```

This produces the following output:

X =

0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	2.26112D-03
1.99543D-03	0.00000D+00	2.89564D-03	3.16133D-03	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	3.30726D-02	4.28516D-02	3.36787D-02	0.00000D+00
0.00000D+00	4.22524D-02	5.46982D-02	4.23885D-02	0.00000D+00
0.00000D+00	3.30591D-02	4.22861D-02	3.31315D-02	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	4.22320D-02	5.46394D-02	4.23671D-02	0.00000D+00
0.00000D+00	5.45593D-02	7.02792D-02	5.46978D-02	0.00000D+00
0.00000D+00	4.22542D-02	5.47028D-02	4.22992D-02	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	3.29955D-02	4.22416D-02	3.30616D-02	0.00000D+00
0.00000D+00	4.22311D-02	5.46280D-02	4.22751D-02	0.00000D+00
0.00000D+00	3.30791D-02	4.22789D-02	3.31384D-02	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00

RESID =

0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	7.86954D-04	-1.38778D-17	9.91187D-04	0.00000D+00
0.00000D+00	-3.46945D-17	2.19390D-03	2.77556D-17	0.00000D+00
0.00000D+00	1.24961D-03	-2.08167D-17	1.50272D-03	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	-1.38778D-17	1.92763D-03	6.93889D-18	0.00000D+00
0.00000D+00	2.16937D-03	0.00000D+00	2.64020D-03	0.00000D+00
0.00000D+00	-4.85723D-17	2.88137D-03	-4.85723D-17	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	9.55636D-04	-2.77556D-17	1.17336D-03	0.00000D+00
0.00000D+00	-4.85723D-17	2.52430D-03	-3.46945D-17	0.00000D+00
0.00000D+00	1.39784D-03	-4.16334D-17	1.66444D-03	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00
0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00	0.00000D+00

The subroutine bxfn2 is quite similar to bxfn1 (see §4.1):

```
subroutine bxfn2( mesh, h, n1, n2, n3, x, b )
real*8 h
integer mesh, n1, n2, n3
real*8 x(n1,n2,n3)
real*8 b(n1,n2,n3)
real*8 zero / 0.0 /
real*8 hfac, xfac, xk, yj, zfac, zm
integer j, k, m
if ( mesh .eq. 1 ) then
    hfac = 2.0 * h ** 2
else
    hfac = 2.0
endif
call dcopy( n1*n2*n3, zero, 0, x, 1 )
call dcopy( n1*n2*n3, zero, 0, b, 1 )
do m = 2,n3-1
    zm = (m - 1) * h
    zfac = zm * ( zm - 1.0 )
    do k = 2,n2-1
        xk = (k - 1) * h
        xfac = xk * ( xk - 1.0 ) + zfac
        do j = 2,n1-1
            yj = (j - 1) * h
            b(j,k,m) = hfac * ( xfac + yj * ( yj - 1.0 ) )
        enddo
    enddo
enddo
return
end
```

A more realistic version of this subroutine would allow for a vector with initial values for the x, y, and z points and would also pass the h vector intact.

REFERENCES

[1] A. BRANDT, *Multi–level adaptive solutions to boundary–value problems*, Math. Comp., 31 (1977), pp. 333–390.

[2] C. C. DOUGLAS, *Multi–grid algorithms for elliptic boundary–value problems*, PhD thesis, Yale University, May 1982.

[3] ——, *Multi–grid algorithms with applications to elliptic boundary–value problems*, SIAM J. Numer. Anal., 21 (1984), pp. 236–254.

[4] ——, *MGNet: a multigrid and domain decomposition network*, ACM SIGNUM Newsletter, 27 (1992), pp. 2–8.

[5] ——, *DAMG: an abstract multilevel solver*, Tech. Report YALEU/DCS/TR-950, Department of Computer Science, Yale University, New Haven, 1993.

[6] J. H. HYMAN, *Mesh refinement and local inversion of elliptic differential equations*, J. of Comp. Phys., 23 (1977), pp. 124–134.

[7] S. F. MCCORMICK, *Multigrid Methods*, vol. 3 of Frontiers in Applied Mathematics, SIAM Books, Philadelphia, 1987.

A. Making DPMG. Anyone using this code would probably do themselves a favor by sending an e-mail note to the author. Announcements of updates will be made through the MGNet (multigrid network) mailing list [4]. To join MGNet, send a request to mgnet-requests@cs.yale.edu.

The source code for DPMG can be found on the Internet. Two possible anonymous ftp sites are the following:

Machine name	IP address
software.watson.ibm.com	129.34.139.5
casper.cs.yale.edu	128.36.12.1

There are other machines with copies at this point, but these should do. Do not attempt to get the files or unpack them directly on a mainframe unless it is running UNIX; use a workstation initially if your target is a mainframe.

Before all else, make a new directory and change to it:

```
% mkdir madpack4
% cd madpack4
```

where % is the prompt assuming you are using the C-shell. To retrieve information, from your Internet connected machine, run the ftp program with one of the machine names as its argument, e.g.,

```
% ftp software.watson.ibm.com
```

You will be prompted for an account name and password: use the account name *anonymous* and your e–mail address as the password. Then change directory to one with the software and look at the directory (the prompt for the ftp program is "ftp> "):

```
ftp> cd pub/pdes
ftp> dir
```

You should see something like the following:

```
 total 888
 drwxr-xr-x  512     Jul 22 1992    .
 drwxr-xr-x  512     Dec 11 08:50   ..
 -rw-r–r–    3691    Apr 28 1992    AGREE.damg
 -rw-r–r–    3691    Apr 28 1992    AGREE.dpmg
 -rw-r–r–    7229    Jul 16 1992    README.damg
 -rw-r–r–    7997    Jun 03 1992    README.dpmg
 -rw-r–r–    182136  Jul 22 1992    damg.tar.Z
 -rw-r–r–    236555  Jul 16 1992    dpmg.tar.Z
```

WARNING: On MGNet (casper.cs.yale.edu), you will find the codes in the directory *mgnet/madpack4* instead of pub/pdes.

You should get all of the Ascii files first:

```
ftp> prompt
ftp> mget A*
ftp> mget R*
```

Read the two AGREE files since these are your software licenses for DPMG and DAMG (a copy of DPMG's license is Appendix B). Assuming there is nothing in the license that you find objectionable, then get both software packages and quit:

```
ftp> binary
ftp> mget d*
ftp> quit
```

Now you are ready to unpack the files into their own directories:

```
% mkdir damg dpmg
% cd damg
% zcat ../damg.tar | tar xvf -
% cd ../dpmg
% zcat ../dpmg.tar | tar xvf -
```

Both zcat and tar are standard utilities on workstations.

You are now in the dpmg directory. The first thing to notice is that there are a number of subdirectories. These contain specialized versions of subroutines for different types of machine architectures:

Directory	Contains
.	Generic Poisson solver routines, examples, user information, Makefile, etc.
chaining	Routines tailored for machines with hardware multiply–add chaining (e.g., IBM RISC System/6000).
chain-no	Routines tailored for machines without chaining (e.g., SUN Sparc 1 or an Intel 80486 microprocessor).
vec-f77	Routines tailored for machines with vector units and Fortran compilers capable of vectorization (e.g., Cray Y-MP).
vec-370	Routines tailored for IBM 3090 or IBM ES9000 machines with vector units and vector assemblers.

Given a machine, you will want to select the correct set of routines to get as much performance as possible. As is described in §A.1, this has been automated to some extent.

A.1. Making DPMG on a workstation. To make some examples using DPMG, you merely have to run the command

```
% make
```

A number of library files (ones with an extension of ".a") will be produced:

File	Contains
../damg/libamg.a	Abstract multilevel solver
libdpmg.a	Generic Poisson solver routines
chain-no/libpcn.a	Routines tailored for machines without chaining
chaining/libpch.a	Routines tailored for machines with chaining
vec-f77/libpvf.a	Routines tailored for machines with vector units

Most of these are not needed, but are included as an example of how to make them. Also, six executables will be made (only p23d_chain and yale_chain are useful, but the others will still execute correctly).

On an IBM RISC System/6000, you will probably be better off making a new library archive with just the components that are useful. You can do this with the commands,

```
% ar crs libdpmg.a dpmg.o mgp?.o
% ( cd chaining ; ar rs ../libdpmg.a *.o )
```

Then move the file libdpmg.a to someplace easily accessible (e.g., /usr/local/lib). You should also copy the file ../damg/libdamg.a to the same directory. Then you can refer to both files when linking programs as simply

$$-ldpmg - ldamg$$

instead of having to know where the archive files are.

You can type the examples from §4 into your computer yourself or you can get them from MGNet as part of mgnet/madpack4/doc.tar.Z. To unpack the document, use the commands

```
% cd ..
% zcat doc.tar | tar xvf -
```

To compile and link the first example, use the commands

```
% cd doc
% xlf -c dpmg-ex1.f
% xlf -o dpmg-ex1 dpmg-ex1.o -ldpmg -ldamg -lessl
```

To execute the program,

```
% dpmg-ex1
```

A.2. Making DPMG on a mainframe. If your mainframe is running AIX, the directions in §A.1 will work with the exception that the compiler name is different (fvs instead of xlf). The remainder of this section assumes your mainframe is running VM.

First, transfer the file *vec-370/getcmp.exec* to your mainframe. Second, edit it to reflect your workstation login name (replace *bells* with your own login name) and your workstation name (replace *noisy*). Third, execute *getcmp*. The last step will transfer all of the appropriate files to your mainframe, compile them with the correct compiler options, create a library, and compile and link two test programs.

B. DPMG's software license. DPMG and DAMG were written while the author was an IBM employee. As such, IBM owns the software. Be that as it may, IBM has a program for releasing software to the public with very few strings attached. After the author filled out a 17 page form (all answers to the really important questions were "not applicable") and collected signatures (only three), the software was made available to Internet users in July, 1992. The author is indebted to Shmuel Winograd, Ashok Chandra, and Larry Carter for signing this form and to Jim McGroddy for not killing this program.

Note that part of the license specifies that updates and bug notifications will be provided through MGNet. The full text of the license agreement is the remainder of this appendix.

1. COPYRIGHT

   Program Name: DPMG

2. RESEARCH SOFTWARE DISCLAIMER

   As experimental, research software, this program is provided free of charge on an "as is" basis without warranty of any kind, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. IBM does not warrant that the functions contained in this program will meet the user's requirements or that the operation of this program will be uninterrupted or error-free. Acceptance and use of this program constitutes the user's understanding that he will have no recourse to IBM for any actual or consequential damages, including, but not limited to, lost profits or savings, arising out of the use or inability to use this program. Even if the user informs IBM of the possibility of such damages, IBM expects the user of this program to accept the risk of any harm arising out of the use of this program, or the user shall not attempt to use this program for any purpose.

3. USER AGREEMENT

BY ACCEPTANCE AND USE OF THIS EXPERIMENTAL PROGRAM THE USER AGREES TO THE FOLLOWING:

a. This program is provided for the user's personal, non-commercial, experimental use and the user is granted permission to copy this program to the extent reasonably required for such use.

b. All title, ownership and rights to this program and any copies remain with IBM, irrespective of the ownership of the media on which the program resides.

c. The user is permitted to create derivative works to this program. However, all copies of the program and its derivative works must contain the IBM copyright notice, the EXPERIMENTAL SOFTWARE DISCLAIMER and this USER AGREEMENT.

d. By furnishing this program to the user, IBM does NOT grant either directly or by implication, estoppel, or otherwise any license under any patents, patent applications, trademarks, copyrights or other rights belonging to IBM or to any third party, except as expressly provided herein.

e. The user understands and agrees that this program and any derivative works are to be used solely for experimental uses and are not to be sold, distributed to a commercial organization, or be commercially exploited in any manner.

f. IBM requests that the user supply to IBM a copy of any changes, enhancements, or derivative works which the user may create. The user grants IBM and its subsidiaries an irrevocable, nonexclusive, worldwide and royalty-free license to use, execute, reproduce, display, perform, prepare derivative works based upon, and distribute, (INTERNALLY AND EXTERNALLY) copies of any and all such materials and derivative works thereof, and to sublicense others to do any, some, or all of the foregoing, (including supporting documentation).

Copies of these modifications should be sent to:

software@yktvmv.bitnet or
na.cdouglas@na-net.ornl.gov

Announcements of updates will be made through the MGNet (multigrid network) mailing list. To join MGNet, send a request to mgnet-requests@cs.yale.edu.