UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL ATOMIC ENERGY AGENCY
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/994-6

SPRING COLLEGES IN COMPUTATIONAL PHYSICS

19 May - 27 June 1997

MULTIGRID TECHNIQUES

C. REBBI
Boston University
Department of Physics
590 Commonwealth Ave.
Boston, Massachusetts 01125
U.S.A.

Preliminary notes from a course on Computational Physics - copyright by Claudio Rebbi, 1995 - for distribution to participants only.

Multigrid techniques

We have seen how the rate of convergence of relaxation algorithms depends on the spectrum of eigenvalues of the discretized differential operator. The closer the factors r_{max} and $|r_{min}|$ are to one, the slower becomes the rate of convergence. On the other hand, the example we considered in some detail has shown us that the distance of r_{max} from 1 is proportional to the square of the lattice spacing (cfr. Eqs. (25) and (26a) of the notes of Oct. 11) and this is a rather typical situation. This suggests that we might improve the convergence by working on a coarser lattice, with larger lattice spacing dx. However, a coarser lattice will be, per se, obviously unstisfactory because of the poorer resolution it provides. The way out of the dilemma lies in the observation that the modes responsible for slow convergence, i.e. the eigenvectors associated with the largest values of $r^{(i)}$, are typically the long-range modes, those exibiting the slowest variation of the field. These can be well reproduced also on a coarser lattice. We can therefore put together the advantages that derive from the faster convergence on a coarser lattice and the higher resolution of a finer lattice if we adopt a "multigrid" technique, which uses both lattices. As a matter of fact, the scheme can be iterated, and one can use several lattices of different degrees of coarseness. The term "multigrid methods" refers indeed, in its generality, to those techniques that use multiple scales of discretization.

Let us now consider now the multigrid technique in some more detail.
We wish to solve

$$A\phi = \rho \tag{1}$$

We take ϕ to represent the actual value of the field, as stored in the memory of the computer, at some definite point in the iterative procedure. We will use the symbol ϕ_{exact} to characterize the actual solution of Eq. (1).
We define a residue (note: r is an array, just as ϕ and ρ)

$$r = \rho - A\phi \tag{2}$$

and a correction or error

$$e = \phi_{exact} - \phi \tag{3}$$

From Eqs. (1-3) and the fact that ϕ_{exact} satisfies $A\phi_{exact} = \rho$ we see that the error obeys the equation

$$Ae = r \tag{4}$$

Of coarse, we do not know e. Knowing e would be equivalent to knowing the exact solution, because, having defined r from the current value of ϕ, if we were able to solve Eq. (3), we could then obtain ϕ_{exact} simply by adding to ϕ the correction e.

The basic idea of the multigrid method is that both r and e will be approximated on a coarser lattice, where an approximate calculation of ϵ will be performed. The algorithm proceeds through the following steps:

1) start from a given ϕ, calculate r according to

$$r = \rho - A\phi \tag{2}$$

2) project r onto a coarser lattice (the details will be given later) to obtain r^c. Notice that r^c is an array defined over a lattice with much fewer points than the original one and has correspondingly fewer components;

3) define a projection A^c of the original operator over the coarser lattice;

4) starting from an initial value $e^c = 0$ perform some number of steps of an iterative procedure for solving Eq. (4): $A^c\, e^c = r^c$;

5) interpolate the error e^c onto the original fine lattice (again, detail will be given later) to obtain the fine-grid correction e and add this to ϕ to obtain a new value for the field.

Notice that all of the five steps in the algorithm correspond to well defined procedures, that could be implemented by suitable subroutines. It is indeed convenient to introduce a subroutine like notation also for purposes of explanation. Let us thus make reference to the five steps above in terms of "subroutines":

1) residue(r, phi, rho) - This subroutine receives the current value of the field ϕ and the array ρ and returns r. We do not add A to the list of arguments, because the very sparse matrix A is never stored explicitely, but is rather "hard-wired" in the subroutine. It would be appropriate, however, to pass to the subroutine the parameters which enter in the definition of A, such as dx and m. We will leave these implicit, nevertheless.

2) project(rc, r) - The meaning is obvious.

3) define_Ac - As we will see, all this module amounts to is a calculation of the parameters of the coarse grid operator A^c. This will often be totally straightforward, e.g it may reduce to the redefinition $dx = 2 * dx$ and will not require a separate subroutine.

4) relax(ec, r, ec0, nit) - Starting from e_0^c (which may be 0) perform *nit* steps of the iterative procedure used to solve $A^c\, e^c = r^c$.

5) interpolate(phi, ec) - Given the correction e^c over the coarser lattice project it first onto the fine lattice to obtain an array e and add e to the current value of ϕ: $\phi = \phi + e$. This overwrites the initial value of ϕ, which is passed as argument to the subroutine.

The definition of the projection from the fine lattice to the coarse lattice is not unique. Several different projection methods can be used. The definition itself of the coarse lattice leaves some degree of arbitrariness. But, in any case, the projection must conform to the underlying principle that we wish to have a mechanism for approximating well the long-range modes responsible for slow convergence on the coarser lattice. We will define now two possible projection precedures for a two-dimensional square lattice. These are the most obvious procedures, those that would be used in most applications. Moreover,

the student may easily derive from these two examples guidelines for implementing more complex projections, for the cases where such may be needed.

A) We take the sites of the coarse lattice to be the centers of 2×2 cells of the finer lattice and define the coarse lattice residue as the average of the fine lattice residue over the cell. In program notation, assuming that r and rc have been dimensioned as r(0:N-1,0:N-1) and rc(0:N/2-1,0:N/2-1) we would have (for all i,j ranging over 0:N/2-1, and with proper attention to the boundary conditions):

```
r(i,j)=0.25*(r(2*i,2*j)+ r(2*i+1,2*j)+ r(2*i+1,2*j+1)+ r(2*i,2*j+1))
```

B) We identify the sites of the coarse lattice with the sites of even parity of the fine lattice and take the new residue to be a weigthed average of the residues at the site itself (weigth 1), at the nearest neighbor sites (weight 1/2, the fine site is shared between two coarse sites) and at the next nearest neighbor sites (weight 1/4, the fine site is shared among four coarse sites):

```
r(i,j)=0.25*(r(2*i,2*j)+ 0.5*(r(2*i+1,2*j)+ r(2*i,2*j+1)+ r(2*i-1,2*j)+r(2*i,2*j-1))
  +0.25*(r(2*i+1,2*j+1)+ r(2*i-1,2*j+1)+ r(2*i-1,2*j-1)+r(2*i+1,2*j-1)))
```

Correspondingly, the interpolation operator will be as follows:

A) the coarse correction e^c will be added (with weight 1) to the value of the field at all the 4 (fine lattice) sites of the cell;

B) the coarse correction e^c will be added with weight 1 to the field at the same site in the fine lattice, with weight 1/2 to the fields at the nearest neighbor sites and with weight 1/4 to the fields at the next nearest neighbor sites. (Notice that all the field elements over the fine lattice will receive contributions with weights that add up to 1).

Finally, about the coarse lattice operator A^c, the most important point is that it must have eigenvectors and eigenvalues that approximate well, in the part of the spectrum corresponding to the long range modes, those of the original operator A. In many cases, especially in connection with the discretization of simple differential operators such as the Laplace operator, the prescription to follow will be obvious. In the examples of differential operators we have considered in the previous lectures as coarse grid operator we would choose the same one we used on the fine grid, but with dx replaced by $2dx$. However, in some more elaborate problems, it is not obvious how to choose an optimal operator, and this may become a topic of research.

```
#include<stdio.h>
#include<math.h>
#define L 32
#define SIZE 1364
#define IS 2
#define JS 1
#define MAXLEVL 50
#define M2 0.04
#define DM2 0.06
#define A 1

extern relax_();
extern residue_();
extern max_res_();
extern project_();
extern interp_();

main()
{
/*
  This program calls the functions relax_, residue_, max_res_, project_
  and interp_ to solve a discretized differential equation by a multigrid
  Gauss-Seidel relaxation algorithm.  The equation to be solved is defined
  in the comments at the beginning of the function relax_.  The details
  of the multigrid cycle are defined (input) by the user at the beginning
  of the run.
*/
    int ncy, nlevl, currlevl, level[MAXLEVL], it, nit[MAXLEVL];
    float f[SIZE], s[SIZE], r[SIZE], k[SIZE];
    float ac, *fp, *sp, *rp, *kp, rn, nf, avres, maxres;
    int lc, i, cy, step;

                        /* Print parameters for the simulation: */

    printf("Lattice size: %d x %d, M2: %f, DM2: %f, source=1 in %d,%d \n\n",
            L, L, M2, DM2, IS, JS);

                        /* Input the parameters of the calculation: */

    printf("Please enter the total number of multigrid cycles:  ");
    scanf(" %d",&ncy);
    puts("");
    puts("Please enter the composition of the multigrid cycles in the");
    puts("following manner.  Enter the level number first, 0 being the");
    puts("finest level, 1 the next coarser level etc., then the");
    puts("corresponding number of iterations, then the next level number,");
    puts("then the corresponding number of iterations, etc. until you");
    puts("terminate the cycle by entering -1 for the level number.\n");
    puts("Please notice: the level numbers must be contiguous, for example");
    puts("0 1 0 1 2 1 0, and must begin and terminate with 0, although");
    printf("it is possible to go through a definite level witout any ");
    puts("iterations.\n");

    nlevl=0;
    EnterLevel:
```

```
        printf("Level:  ");
        scanf(" %d",&currlevl);
        if (currlevl>=0)
        {
            level[nlevl]=currlevl;
            printf("Number of iterations:  ");
            scanf(" %d",&it);
            nit[nlevl]=it;
            nlevl++;
            goto EnterLevel;
        }
    puts("");

                        /* Initialize variables and pointers: */

    ac=A;
    lc=L;
    fp=f;
    sp=s;
    rp=r;
    kp=k;
    nf=pow(2.,31.);

    for (i=0; i<lc*lc; i++)
    {
        f[i]=0;
        s[i]=0;
        rn=rand()/nf-0.5;
        k[i]=M2+DM2*rn;
    }
    s[lc*IS+JS]=1;
                        /* Define the k variable for the coarser levels: */

    while (lc>2)
    {
        project_(kp, &lc);
        kp=kp+lc*lc;
        lc=lc/2;
    }
                        /* Reset the current lattice size and the pointer
                           to k: */
    lc=L;
    kp=k;
                        /* Begin multigrid cycles: */

    for (cy=1; cy<=ncy; cy++)
    {
                        /* Calculate and print information on the residue: */

        residue_(r, f, s, k, &ac, &lc);
        max_res_(&avres, &maxres, r, &lc);
        printf (" At cycle %d, av. res. = %f, max res. = %f\n",cy-1,
                  avres, maxres);

        currlevl=0;
        for (step=0; step<nlevl; step++)
```

```
        {
                        /* If the level goes up, calculate the residue,
                           project it to the coarser lattice, reset
                           the pointers to the coarser lattice, redefine
                           the current lattice spacing and lattice size,
                           copy the residue into the source term for the
                           coarser lattice and set the field to zero: */

            if(level[step]>currlevl)
            {
                residue_(rp, fp, sp, kp, &ac, &lc);
                project_(rp, &lc);
                rp=rp+lc*lc;
                fp=fp+lc*lc;
                sp=sp+lc*lc;
                kp=kp+lc*lc;
                ac=2*ac;
                lc=lc/2;
                for (i=0; i<lc*lc; i++)
                {
                    *(sp+i)=(*(rp+i));
                    *(fp+i)=0;
                }
            }
                        /* If the level goes down, reset the pointers
                           to the finer lattice, redefine the current
                           lattice spacing and lattice size, interpolate
                           the correction, contained in the field variable
                           for the coarser lattice, and add it to the
                           field of the finer lattice (interp interpolates
                           and adds): */

            if(level[step]<currlevl)
            {
                lc=2*lc;
                rp=rp-lc*lc;
                fp=fp-lc*lc;
                sp=sp-lc*lc;
                kp=kp-lc*lc;
                ac=ac/2;
                interp_(fp, &lc);
            }
                        /* Perform the required number of Gauss-Seidel
                           relaxations: */

            currlevl=level[step];
            it=nit[step];
            relax_(fp, sp, kp, &ac, &lc, &it);
        }
    }
                        /* Calculate and print information on the
                           final residue: */

    residue_(r, f, s, k, &ac, &lc);
    max_res_(&avres, &maxres, r, &lc);
    printf (" At cycle %d, av. res. = %f, max res. = %f\n",ncy,


                avres, maxres);
}
```

```
      PROGRAM mgf
c
c  This program calls the functions relax, residue, max_res, project
c  and interp to solve a discretized differential equation by a multigrid
c  Gauss-Seidel relaxation algorithm.  The equation to be solved is defined
c  in the comments at the beginning of the function relax.  The details
c  of the multigrid cycle are defined (input) by the user at the beginning
c  of the run.
c
      IMPLICIT none
      INTEGER L, SIZE, IS, JS, MAXLEVL
      REAL M2, DM2, A
      PARAMETER (L=32, SIZE=1364, IS=2, JS=1, MAXLEVL=50)
      PARAMETER (M2=0.04, DM2=0.06, A=1)
      INTEGER ncy, nlevl, currlevl, level(0:MAXLEVL-1), it,
     +        nit(0:MAXLEVL-1)
      REAL f(0:SIZE-1), s(0:SIZE-1), r(0:SIZE-1), k(0:SIZE-1)
      INTEGER fp, sp, rp, kp
      REAL ac, rn, avres, maxres
      REAL rand
      INTEGER lc, i, cy, step
c
c                          Print parameters for the simulation:
c
      PRINT '(''Lattice size: '',I3,''x'',I3,'' M2: '',F7.4,'' DM2: '',
     + F7.4,'' source=1 in '',I3,'','',I3)', L, L, M2, DM2, IS, JS
      PRINT *
c
c                          Input the parameters of the calculation:
c
      PRINT '(''Please enter the total number of multigrid cycles:  '',
     +$)'
      READ *, ncy
      PRINT *
      PRINT '(''Please enter the composition of the multigrid cycles in
     +the'')'
      PRINT '(''following manner.  Enter the level number first, 0 being
     + the'')'
      PRINT '(''finest level, 1 the next coarser level etc., then the''
     +)'
      PRINT '(''corresponding number of iterations, then the next level
     +number,'')'
      PRINT '(''then the corresponding number of iterations, etc. until
     +you'')'
      PRINT '(''terminate the cycle by entering -1 for the level number.
     +'')'
      PRINT '(''Please notice: the level numbers must be contiguous, for
     + example'')'
      PRINT '(''0 1 0 1 2 1 0, and must begin and terminate with 0, alth
     +ough'')'
      PRINT '(''it is possible to go through a definite level witout any
     + iterations.'')'
      PRINT *
      nlevl=0
 10   CONTINUE
        PRINT '(''Level:  '',$)'
```

```
        READ *, currlevl
        IF (currlevl.GE.0) THEN
          level(nlevl)=currlevl
          PRINT '(''Number of iterations:  '',$)'
          READ *, it
          nit(nlevl)=it
          nlevl=nlevl+1
          GOTO 10
        ENDIF
      PRINT *
c
c                         Initialize variables and pointers:
c
      ac=A
      lc=L
      fp=0
      sp=0
      rp=0
      kp=0
      DO 20 i=0, lc**2-1
        f(i)=0
        s(i)=0
        rn=rand(0)-0.5
        k(i)=M2+DM2*rn
 20   CONTINUE
      s(lc*IS+JS)=1

 30   CONTINUE
        CALL project(k(kp), lc)
        kp=kp+lc*lc
        lc=lc/2
      IF(lc.GT.2) GOTO 30

      lc=L
      kp=0

      DO 40 cy=1, ncy
c
c                         Calculate and print information on the residue:
c
        CALL residue(r, f, s, k, ac, lc)
        CALL max_res(avres, maxres, r, lc)
        PRINT '(''At cycle'',I4,'', av. res. ='',F9.6,'', max res. ='',
     +          F9.6)',cy-1, avres, maxres

        currlevl=0
        DO 50 step=0, nlevl-1
c
c                           If the level goes up, calculate the residue,
c                           project it to the coarser lattice, reset
c                           the pointers to the coarser lattice, redefine
c                           the current lattice spacing and lattice size,
c                           copy the residue into the source term for the
c                           coarser lattice and set the field to zero:
c
          IF(level(step).GT.currlevl) THEN
```

```
            CALL residue(r(rp), f(fp), s(sp), k(kp), ac, lc)
            CALL project(r(rp), lc)
            rp=rp+lc*lc
            fp=fp+lc*lc
            sp=sp+lc*lc
            kp=kp+lc*lc
            ac=2*ac
            lc=lc/2
            DO 60 i=0, lc**2-1
              s(sp+i)=r(rp+i)
              f(fp+i)=0
 60         CONTINUE
          ENDIF
c
c                           If the level goes down, reset the pointers
c                           to the finer lattice, redefine the current
c                           lattice spacing and lattice size, interpolate
c                           the correction, contained in the field variable
c                           for the coarser lattice, and add it to the
c                           field of the finer lattice (interp interpolates
c                           and adds):
c
          IF(level(step).LT.currlevl) THEN
            lc=2*lc
            rp=rp-lc*lc
            fp=fp-lc*lc
            sp=sp-lc*lc
            kp=kp-lc*lc
            ac=ac/2
            CALL interp(f(fp), lc)
          ENDIF
          currlevl=level(step)
          it=nit(step)
          CALL relax(f(fp), s(sp), k(kp), ac, lc, it)
 50     CONTINUE
 40   CONTINUE
c
c                           Calculate and print information on the
c                           final residue:
c
        CALL residue(r, f, s, k, ac, lc)
        CALL max_res(avres, maxres, r, lc)
        PRINT '(''At cycle'',I4,'', av. res. ='',F9.6,'', max res. ='',
     +          F9.6)',ncy, avres, maxres

      END
```

```
#include<math.h>

relax_(f, s, k, a, l, nit)
int *l,*nit;
float f[], s[], k[], *a;
/*
    This function receives the array f with the current values of the
    field, the source array s, the array k with variable potential (or
    mass squared) parameter and the pointers to the lattice spacing a,
    the integer size of the lattice l and the integer nit, and must
    return in f (overwriting the initial values) the result of nit
    iterations of the Gauss-Seidel relaxation algorithm for the equation

                          M f = s.

    The arrays f, s and k are all dimensioned in the calling program.
    They are one-dimensional arrays of size l**2 and the variables are
    stored according to the following conventions.

    The lattice is a square lattice of size l*l with periodic boundary
    conditions.  l is a power of 2 (this fact may be taken advantage of
    in order to impose periodicity through bitwise logic operations).
    Let the lattice coordinates be (i,j) where i and j range from 0 to
    l-1.  Then the elements of the arrays are stored in the order
    (0,0), (0,1), ..., (0,l-1), (1,0), (1,1), ..., (1,l-1) etc., i.e.
    the index of the elemnt with coordinate i,j is l*i+j.

    The matrix M is defined as follows.  The (i,j) elemnt of M e is
    given by

    (M f)(i,j)=
       (4*f(i,j)-f(i+1,j)-f(i-1,j)-f(i,j+1)-f(i,j-1))/a**2+k(i,j)*f(i,j)

    where the values i+1, i-1, j+1 and j-1 are to be interpreted by taking
    into account the periodic boundary conditions (thus, if i=l-1, i+1
    should be taken to be 0).

    The Gauss-Seidel relaxation must upgrade the values of f in the same
    order in which they are stored in memory.
*/
{
    int it, i, ifwd, ibwd, j, jfwd, jbwd, ij;
    float aux;
    aux=1/((*a)*(*a));
    for (it=1; it<=*nit; it++)
    {
        for (i=0; i<*l; i++)
        {
/*
    Here and below define the indices of the nearest neighbors in the
    forward (ifwd, jfwd) and backward (ibwd, jbwd) directions, respectively,
    by bitwise operations to impose periodicity with a lattice size
    l which is a power of 2.
*/
            ifwd=i+1&*l-1;
```

```
            ibwd=i+*l-1&*l-1;
            for (j=0; j<*l; j++)
            {
                jfwd=j+1&*l-1;
                jbwd=j+*l-1&*l-1;
                ij=(*l)*i+j;
                f[ij]=(s[ij]+aux*(f[*l*ifwd+j]+f[*l*ibwd+j]+f[*l*i+jfwd]
                      +f[*l*i+jbwd]))/(4*aux+k[ij]);
            }
        }
    }
}

residue_(r, f, s, k, a, l)
int *l;
float r[], f[], s[], k[], *a;
/*
    This function receives the field array f, the source array s, the
    variable parameter array k and the pointers to the lattice spacing a
    and the integer size of the lattice l, and must return in the array
    r the residue of the equation  M f = s, i.e.

                          r  =  s - M x.

    All the arrays are dimensioned in the calling program.  For the
    conventions relative to the indices of the arrays and the definition
    of the matrix M, see the comments in the function relax.
*/
{
    int i, ifwd, ibwd, j, jfwd, jbwd, ij;
    float aux;
    aux=1/((*a)*(*a));
    for (i=0; i<*l; i++)
    {
        ifwd=i+1&*l-1;
        ibwd=i+*l-1&*l-1;
        for (j=0; j<*l; j++)
        {
            jfwd=j+1&*l-1;
            jbwd=j+*l-1&*l-1;
            ij=(*l)*i+j;
            r[ij]=s[ij]-aux*(4*f[ij]-f[*l*ifwd+j]-f[*l*ibwd+j]
                  -f[*l*i+jfwd]-f[*l*i+jbwd])-k[ij]*f[ij];
        }
    }
}

max_res_(avres, maxres, r, l)
int *l;
float *avres, *maxres, r[];
/*
    This function receives the array r and the pointers to the integer
    size of the lattice l and to the variables avres, maxres.  r is
    dimensioned by the calling program and is a one-dimensional array of
    sixe l**2.  max_res must return in avres the average residue, i.e.
    the squared root of the sum of the squares of the elements of r
```

```
    divided by the number of elements l*l, and in maxres the maximum of
    the absolute values of the elements of r.
*/
{
    int i;
    float sum, r2, maxres2;
    sum=0;
    maxres2=0;
    for (i=0; i<(*l)*(*l); i++)
    {
        r2=r[i]*r[i];
        if (r2>maxres2) maxres2=r2;
        sum+=r2;
    }
    *avres=sqrt(sum/((*l)*(*l)));
    *maxres=sqrt(maxres2);
}

project_(r, l)
int *l;
float r[];
/*
    This function receives a one-dimensional array r, dimensioned in the
    calling program, and the integer pointer to the size of the fine
    lattice l.  r contains in its first l**2 elements the values of a
    residue vector on a l*l lattice, stored according to the conventions
    explained in the comments at the beginning of the function relax.
    project must return in the subsequent (l/2)**2 elements of r (i.e.,
    in the elements with index l**2, l**2+1 ..., l**2+(l/2)**2-1)
    the projection from the fine lattice to the coarse lattice.
    More specifically, if we denote with ic,jc the coordinates of a generic
    point of the coarse lattice, the element of r corresponding to these
    coordinates must contain the average of the four elements of the fine
    lattice corresponding to the fine lattice coordinates

    i,j  =  2*ic,2*jc   2*ic+1,2*jc   2*ic,2*jc+1   and   2*ic+1,2*jc+1

    The indexing on the coarse lattice follows, of course, the same
    ordering conventions as used for the fine lattice, apart from the
    obvious change in the range of the indices.
*/
{
    int l2, lc, ic, jc;
    l2=(*l)*(*l);
    lc=(*l)/2;
    for (ic=0; ic<lc; ic++)
    {
        for (jc=0; jc<lc; jc++)
        {
            r[l2+lc*ic+jc]=(r[*l*2*ic+2*jc]+r[*l*(2*ic+1)+2*jc]
                            +r[*l*2*ic+2*jc+1]+r[*l*(2*ic+1)+2*jc+1])/4;
        }
    }
}

interp_(f, l)
```

```
int *l;
float f[];
/*
    This function receives a one-dimensional array f, dimensioned in the
    calling program, and the integer pointer to the size of the fine
    lattice l.  f contains in its elements with index l**2, l**2+1 ...,
    l**2+(l/2)**2-1) the values of the error (i.e. correction) vector
    on a coarse (l/2)*(l/2) lattice, stored according to the conventions
    explained in the comments at the beginning of the function relax.
    interp must return in the first l**2 elements of the same array the
    sum of the original values of the field on the fine lattice with the
    interpolation of the error from the coarse lattice to the fine
    lattice.  More specifically, if we denote with ic,jc the coordinates
    of a generic point of the coarse lattice, the element of f
    corresponding to these coordinates must be added to the four
    elements of the field f on the fine lattice with coordinates

    i,j  =  2*ic,2*jc   2*ic+1,2*jc   2*ic,2*jc+1   and   2*ic+1,2*jc+1
*/
{
    int l2, lc, ic, jc;
    float aux;
    l2=(*l)*(*l);
    lc=(*l)/2;
    for (ic=0; ic<lc; ic++)
    {
        for (jc=0; jc<lc; jc++)
        {
            aux=f[l2+lc*ic+jc];
            f[*l*2*ic+2*jc]+=aux;
            f[*l*(2*ic+1)+2*jc]+=aux;
            f[*l*2*ic+2*jc+1]+=aux;
            f[*l*(2*ic+1)+2*jc+1]+=aux;
        }
    }
}
```

```
Lattice size:  32x 32 M2:  0.0400 DM2:  0.0600 source=1 in   2,  1

Please enter the total number of multigrid cycles:  10

Please enter the composition of the multigrid cycles in the
following manner.  Enter the level number first, 0 being the
finest level, 1 the next coarser level etc., then the
corresponding number of iterations, then the next level number,
then the corresponding number of iterations, etc. until you
terminate the cycle by entering -1 for the level number.
Please notice: the level numbers must be contiguous, for example
0 1 0 1 2 1 0, and must begin and terminate with 0, although
it is possible to go through a definite level witout any iterations.

Level: 0
Number of iterations: 10
Level: -1

At cycle   0, av. res. = 0.031250, max res. = 1.000000
At cycle   1, av. res. = 0.002862, max res. = 0.026376
At cycle   2, av. res. = 0.001883, max res. = 0.013568
At cycle   3, av. res. = 0.001450, max res. = 0.008809
At cycle   4, av. res. = 0.001191, max res. = 0.006358
At cycle   5, av. res. = 0.001018, max res. = 0.004875
At cycle   6, av. res. = 0.000896, max res. = 0.003888
At cycle   7, av. res. = 0.000805, max res. = 0.003191
At cycle   8, av. res. = 0.000734, max res. = 0.002677
At cycle   9, av. res. = 0.000678, max res. = 0.002286
At cycle  10, av. res. = 0.000632, max res. = 0.001982


Lattice size:  32x 32 M2:  0.0400 DM2:  0.0600 source=1 in   2,  1

Please enter the total number of multigrid cycles:  10

Please enter the composition of the multigrid cycles in the
following manner.  Enter the level number first, 0 being the
finest level, 1 the next coarser level etc., then the
corresponding number of iterations, then the next level number,
then the corresponding number of iterations, etc. until you
terminate the cycle by entering -1 for the level number.
Please notice: the level numbers must be contiguous, for example
0 1 0 1 2 1 0, and must begin and terminate with 0, although
it is possible to go through a definite level witout any iterations.

Level:  0
Number of iterations:  3
Level:  1
Number of iterations:  4
Level:  0
Number of iterations:  3
Level:  -1

At cycle   0, av. res. = 0.031250, max res. = 1.000000
At cycle   1, av. res. = 0.001882, max res. = 0.019156
At cycle   2, av. res. = 0.001136, max res. = 0.006709
```

```
At cycle   3, av. res. = 0.000850, max res. = 0.003895
At cycle   4, av. res. = 0.000696, max res. = 0.002630
At cycle   5, av. res. = 0.000597, max res. = 0.001924
At cycle   6, av. res. = 0.000524, max res. = 0.001489
At cycle   7, av. res. = 0.000465, max res. = 0.001203
At cycle   8, av. res. = 0.000414, max res. = 0.001004
At cycle   9, av. res. = 0.000371, max res. = 0.000857
At cycle  10, av. res. = 0.000332, max res. = 0.000744


Lattice size:  32x 32 M2:  0.0400 DM2:  0.0600 source=1 in   2,  1

Please enter the total number of multigrid cycles:  10

Please enter the composition of the multigrid cycles in the
following manner.  Enter the level number first, 0 being the
finest level, 1 the next coarser level etc., then the
corresponding number of iterations, then the next level number,
then the corresponding number of iterations, etc. until you
terminate the cycle by entering -1 for the level number.
Please notice: the level numbers must be contiguous, for example
0 1 0 1 2 1 0, and must begin and terminate with 0, although
it is possible to go through a definite level witout any iterations.

Level:  0
Number of iterations:  2
Level:  1
Number of iterations:  2
Level:  2
Number of iterations:  2
Level:  1
Number of iterations:  2
Level:  0
Number of iterations:  2
Level:  -1

At cycle   0, av. res. = 0.031250, max res. = 1.000000
At cycle   1, av. res. = 0.001497, max res. = 0.014112
At cycle   2, av. res. = 0.000648, max res. = 0.002333
At cycle   3, av. res. = 0.000468, max res. = 0.001142
At cycle   4, av. res. = 0.000359, max res. = 0.000836
At cycle   5, av. res. = 0.000279, max res. = 0.000646
At cycle   6, av. res. = 0.000217, max res. = 0.000504
At cycle   7, av. res. = 0.000168, max res. = 0.000393
At cycle   8, av. res. = 0.000131, max res. = 0.000307
At cycle   9, av. res. = 0.000102, max res. = 0.000239
At cycle  10, av. res. = 0.000079, max res. = 0.000186


Lattice size:  32x 32 M2:  0.0400 DM2:  0.0600 source=1 in   2,  1

Please enter the total number of multigrid cycles:  10

Please enter the composition of the multigrid cycles in the
following manner.  Enter the level number first, 0 being the
finest level, 1 the next coarser level etc., then the
```

corresponding number of iterations, then the next level number,
then the corresponding number of iterations, etc. until you
terminate the cycle by entering -1 for the level number.
Please notice: the level numbers must be contiguous, for example
0 1 0 1 2 1 0, and must begin and terminate with 0, although
it is possible to go through a definite level witout any iterations.

```
Level:  0
Number of iterations:  2
Level:  1
Number of iterations:  1
Level:  2
Number of iterations:  1
Level:  3
Number of iterations:  1
Level:  4
Number of iterations:  1
Level:  3
Number of iterations:  1
Level:  2
Number of iterations:  1
Level:  1
Number of iterations:  1
Level:  0
Number of iterations:  1
Level:  -1

At cycle   0, av. res. = 0.031250, max res. = 1.000000
At cycle   1, av. res. = 0.004645, max res. = 0.063010
At cycle   2, av. res. = 0.000449, max res. = 0.003472
At cycle   3, av. res. = 0.000063, max res. = 0.000923
At cycle   4, av. res. = 0.000008, max res. = 0.000063
At cycle   5, av. res. = 0.000001, max res. = 0.000015
At cycle   6, av. res. = 0.000000, max res. = 0.000002
At cycle   7, av. res. = 0.000000, max res. = 0.000000
At cycle   8, av. res. = 0.000000, max res. = 0.000000
At cycle   9, av. res. = 0.000000, max res. = 0.000000
At cycle  10, av. res. = 0.000000, max res. = 0.000000
```