# global array matrix multiply routines.
#
CC = gcc

# where the GA include files live (e.g., ga.h)
GA_INCLUDES = /usr/local/ga/include
# where the GA library files live (e.g., libglobal.a)
GA_LIBS = /usr/local/ga/lib

# where the MPI include files live (e.g., mpi.h)
MPI_INCLUDES = /usr/local/mpich/include
# where the MPI library files live (e.g., libmpi.a or libmpich.a)
MPI_LIBS = /usr/local/mpich/lib
# name of the MPI library to use (e.g., mpi or mpich)
MPI_LIBNAME = −lmpich

DOCSRC = Makefile main.c gen_ga_A.c gen_ga_B.c gen_ga_C.c
DOCSRC += gen_A.c gen_B.c gen_C.c ga_blocked.c
DOCSRC += print_mat.c CPU_Time.c Wall_Time.c
A2PSFLAGS = −−output=gamatrix.ps −−medium=Letter −1
A2PSFLAGS += −−file−align=fill −−toc

OBJ = main.o gen_ga_A.o gen_ga_B.o gen_ga_C.o ga_blocked.o
OBJ += gen_A.o gen_B.o gen_C.o print_mat.o CPU_Time.o Wall_Time.o

INCLUDE = −I. −I${GA_INCLUDES} −I${MPI_INCLUDES}
LIBS = −lm −L${GA_LIBS} −lglobal −larmci −lma −L${MPI_LIBS} ${MPI_LIBNAME}
CFLAGS = −O −Wall

.c.o: $*.c
    $(CC) −c $(CFLAGS) $(INCLUDE) $*.c

gamatrix: $(OBJ)
    $(CC) $(CFLAGS) −o $@ $(OBJ) $(LIBS)

main.o: genABC_proto.h CWtime.h mymin.h mymax.h myabs.h
gen_A.o: genABC.h
gen_B.o: genABC.h
gen_C.o: genABC.h
gen_ga_A.o: mymin.h genABC_proto.h
gen_ga_B.o: mymin.h genABC_proto.h
gen_ga_C.o: mymin.h genABC_proto.h
ga_blocked.o: mymin.h
docs: gamatrix.ps gamatrix.pdf
    @echo " all docs made 

# docs: gamatrix.ps gamatrix.pdf
    ps2pdf gamatrix.ps

# gamatrix.ps: $(DOCSRC)
    a2ps $(A2PSFLAGS) $(DOCSRC)

clean: ; rm −f $(OBJ)
realclean: ; rm −f $(OBJ) gamatrix gamatrix.ps gamatrix.pdf
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "CWtime.h"
#include "macommon.h"
#include "macdecls.h"
#include "mpi.h"
#include "ga.h"
#include "genABC_proto.h"
#include "mymax.h"
#include "mymin.h"
#include "myabs.h"

#define DEBUGPRINT 0
#define ZERO ((double)0.0)

void Usage(void)
{
    printf("Usage: gamatrix rowsa <integer> colsa <integer> colsb <integer>\n");
    fflush(stdout);
}

int parsemyargs(int argc, char** argv, int *rA, int *cA, int *cB)
{
    int i;
    int found_ra=0;
    int found_ca=0;
    int found_cb=0;
    if (argc < 7) {
        Usage();
        GA_Error("wrong number of arguments",911);
    }
    for (i=1;i<argc;i++) {
        if (!strcmp("rowsa",argv[i])) {
            if (!found_ra){
                *rA = atoi(argv[i+1]);
                found_ra++;
            } else {
                printf("rowsa specified twice\n");
                exit(((int)911));
            }
        }
        if (!strcmp("colsa",argv[i])) {
            if (!found_ca){
                *cA = atoi(argv[i+1]);
                found_ca++;
            } else {
                printf("colsa specified twice\n");
                exit(((int)911));
            }
        }
    }
}
if (!strcmp("colsb", argv[i])) {
  if (!found_cb) {
    *cB = atoi(argv[i+1]);
    found_cb++;
  } else {
    printf("colsb specified twice\n");
    exit((int)911);
  }
}

if ((found_ra) && (found_ca) && (found_cb)) return 0;
printf("Errant command line:\n");
for (i=0; i<argc; i++)
  printf("%s ", argv[i]);
printf("\n");
return 1;

/*****************************************************
 * function prototypes
*****************************************************
void gen_ga_A(int g_A, int rows, int cols, int blocksize);
void gen_ga_B(int g_B, int rows, int cols, int blocksize);
void gen_ga_C(int g_C, int rows, int cols, int crossdim, int blocksize);
void print_mat(double buffer[], int num_row, int num_col);
void ga_blocked(int g_A, int g_B, int g_C, int blocksize);

/*****************************************************
 * Main driver routine
*****************************************************
int main(int argc, char *argv[])
{
  int blocksize=32;
  int heap, stack;
  int g_A, g_B, g_C, g_CA, g_Diff;
  int rowsA, colsA, rowsB, colsB, rowsC, colsC;
  int me, nproc;
  int dims[2];
  double alpha, beta, norm;
  double call, wall, c0, w0;

  if (MPI_SUCCESS != MPI_Init(&argc, &argv)) {
    fprintf(stderr,"MPI_Init failed\n");
    MPI_Abort(MPI_COMM_WORLD,((int)911));
  }
  GA_Initialize();
  wall = ZERO - Wall_Time();
  call = ZERO - CPU_Time();
  me = GA_Nodeid();
  nproc = GA_Nnodes();

  if (me==0) {
    if (parsemyargs(argc, argv, &rowsA, &colsA, &colsB)) {
      Usage();
      GA_Error("parse errors", 911);
    }
  }
  GA_Brdcst(&rowsA,sizeof(int),0);
  GA_Brdcst(&colsA,sizeof(int),0);
  GA_Brdcst(&colsB,sizeof(int),0);
rowsC = rowsA;
rowsB = colsA;
colsC = colsB;
GA_Sync();

stack = 3*blocksize+blocksize;
stack = MYMAX(10000,stack);
heap = rowsA*colsA + rowsB*colsB + 2*rowsC*colsC;
heap = heap/nproc;
heap = (int)((double)heap*(double)1.2);
heap = MYMAX(10000,heap);
if (me==0)
    printf(" MA memory: stack=%d, heap=%d\n",stack,heap);
if (!MA_init(MT_C_DBL,stack,heap))
    GA_Error("MA_init failed badly",911);

if (me==0) {
    printf("Global Array Matrix Multiply Code (C)\n");
    printf(" A(%d,%d)*B(%d,%d) = C(%d,%d)\n", 
            rowsA,colsA,
            rowsB,colsB,
            rowsC,colsC);
    if (nproc==1)
        printf(" program running on %d processor\n",nproc);
    else
        printf(" program running on %d processors\n",nproc);
    printf(" blocksize for blocking algorithm is: %d doubles\n", 
            blocksize);
}
dims[0] = rowsA; dims[1] = colsA;
g_A = NGA_Create(MT_C_DBL,2,dims,"A", (int*)NULL);
if (!g_A) GA_Error("failed to create A matrix",911);
GA_Zero(g_A);
if (DEBUGPRINT)
    GA_Print_distribution(g_A);

dims[0] = rowsB; dims[1] = colsB;
g_B = NGA_Create(MT_C_DBL,2,dims,"B", (int*)NULL);
if (!g_B) GA_Error("failed to create B matrix",911);
GA_Zero(g_B);
if (DEBUGPRINT)
    GA_Print_distribution(g_B);

dims[0] = rowsC; dims[1] = colsC;
g_C = NGA_Create(MT_C_DBL,2,dims,"Computed C", (int*)NULL);
if (!g_C) GA_Error("failed to create C matrix",911);
GA_Zero(g_C);
if (DEBUGPRINT)
    GA_Print_distribution(g_C);

dims[0] = rowsC; dims[1] = colsC;
g_CA = GA_Duplicate(g_C,"C Analytical");
if (!g_CA) GA_Error("failed to create Analytical C matrix",911);
GA_Zero(g_CA);
if (DEBUGPRINT)
    GA_Print_distribution(g_CA);

w0 = ZERO-Wall_Time(); c0=ZERO-CPU_Time();
gen_ga_A(g_A, rowsA, colsA, 64);
c0 += CPU_Time(); w0 += Wall_Time();
if (me==0)
    printf("Gen A : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
        MYABS(c0), MYMAX(c0, w0));
if (DEBUGPRINT)
    GA_Print(g_A);
GA_Sync();
w0 = ZERO-Wall_Time(); c0=ZERO-CPU_Time();
if (me==0)
    printf("Gen A : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
        MYABS(c0), MYMAX(c0, w0));
if (DEBUGPRINT)
    GA_Print(g_A);
GA_Sync();
if (me==0)
    printf("Gen B : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
        MYABS(c0), MYMAX(c0, w0));
if (DEBUGPRINT)
    GA_Print(g_B);
GA_Sync();
c0 += CPU_Time(); w0 += Wall_Time();
if (me==0)
    printf("Gen C : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
        MYABS(c0), MYMAX(c0, w0));
    fflush(stdout);
if (me==0) {
    printf("Compute C : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
        MYABS(c0), MYMAX(c0, w0));
    fflush(stdout);
    }
if (DEBUGPRINT)
    GA_Print(g_CA);
GA_Sync();
dims[0] = rowsC; dims[1] = colsC;
g_Diff = GA_Duplicate(g_C, "Difference Matrix");
if (!g_Diff) GA_Error("failed to create Difference matrix", 911);
alpha = (double)1.0;
beta  = −alpha;
w0 = ZERO-Wall_Time(); c0=ZERO-CPU_Time();
if (me==0) {
    printf("Difference Norm is %.12e\n", norm);
    printf("Norm time : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
        MYABS(c0), MYMAX(c0, w0));
    fflush(stdout);
    }
GA_Destroy(g_Diff);
GA_Destroy(g_C);
GA_Destroy(g_CA);
call += CPU_Time();
wall += Wall_Time();
printf("Total : CPU time: %10.2f sec  Wall time: %10.2f sec\n",
      MYABS(call),MYMAX(call,wall));
fflush(stdout);
if (me==0) {
    MA_summarize_allocated_blocks();
    GA_Print_stats();
    fflush(stdout);
}
GA_Terminate();
if (MPI_SUCCESS != MPI_Finalize()) {
    fprintf(stderr,"MPI Finalize failed\n");
    MPI_Abort(MPI_COMM_WORLD,((int)911));
}
return 0;
}
#include <stdio.h>
#include <stdlib.h>
#include "ga.h"
#include "genABC_proto.h"
#include "mymin.h"

void gen_ga_A(int g_A, int rows, int cols, int blocksize)
{
    int nbr, nbc;
    int ilo, ihi, jlo, jhi;
    int lo[2], hi[2];
    int nsize;
    int ib, jb;
    int ipass;
    int nproc, me;
    double *block2use;

    me = GA_Nodeid();
    nproc = GA_Nnodes();
    block2use = (double *)malloc(blocksize*blocksize*sizeof(double));
    nbr = 0; nbc = 0;
    nbr = rows/blocksize;
    if (rows%blocksize) nbr++;
    nbc = cols/blocksize;
    if (cols%blocksize) nbc++;
    ipass = -1;
    for(ib=0; ib<nbr; ib++) {
        ilo = ib*blocksize;
        ihi = MYMIN((ilo+blocksize),(rows-1));
        for(jb=0; jb<nbc; jb++) {
            jlo = jb*blocksize;
            jhi = MYMIN((jlo+blocksize),(cols-1));
            ipass++;
            if ((ipass%nproc)==me) {
                gen_A(block2use,ilo,ihi,jlo,jhi,rows,cols);
                lo[0]=ilo;    lo[1]=jlo;
                hi[0]=ihi;    hi[1]=jhi;
                nsize = (jhi-jlo+1);
                NGA_Put(g_A,lo,hi,block2use,&nsize);
            }
        }
    }
    free(block2use);
}
#include <stdio.h>
#include <stdlib.h>
#include "ga.h"
#include "genABC_proto.h"
#include "mymin.h"

void gen_ga_B(int g_B, int rows, int cols, int blocksize) {
    int nbr, nbc;
    int ilo, ihi, jlo, jhi;
    int lo[2], hi[2];
    int nsize;
    int ib, jb;
    int ipass;
    int nproc, me;
    double *block2use;

    me = GA_Nodeid();
    nproc = GA_Nnodes();
    block2use = (double *)malloc(blocksize*blocksize*sizeof(double));
    nbr = 0; nbc = 0;
    if (rows%blocksize) nbr++;
    if (cols%blocksize) nbc++;
    ipass = -1;
    for (ib=0; ib<nbr; ib++) {
        ilo = ib*blocksize;
        ihi = MYMIN((ilo+blocksize), (rows-1));
        for (jb=0; jb<nbc; jb++) {
            jlo = jb*blocksize;
            jhi = MYMIN((jlo+blocksize), (cols-1));
            ipass++;
            if ((ipass%nproc)==me) {
                gen_B(block2use, ilo, ihi, jlo, jhi, rows, cols);
                lo[0]=ilo;   lo[1]=jlo;
                hi[0]=ihi;   hi[1]=jhi;
                nsize = (jhi-jlo+1);
                NGA_Put(g_B, lo, hi, block2use, &nsize);
            }
        }
    }
    free(block2use);
}

#include <stdio.h>
#include <stdlib.h>
#include "ga.h"
#include "genABC_proto.h"
#include "mymin.h"

void gen_ga_C(int g_C, int rows, int cols, int crossdim, int blocksize)
{
    int nbr, nbc;
    int ilo, ihi, jlo, jhi;
    int lo[2], hi[2];
    int nsize;
    int ib, jb;
    int ipass;
    int nproc, me;
    double *block2use;

    me = GA_Nodeid();
    nproc = GA_Nnodes();
    block2use = (double *)malloc(blocksize*blocksize*sizeof(double));
    nbr =0; nbc=0;
    nbr = rows/blocksize;
    if (rows%blocksize) nbr++;
    nbc = cols/blocksize;
    if (cols%blocksize) nbc++;
    ipass = −1;
    for(ib=0;ib<nbr;ib++) {
        ilo = ib*blocksize;
        ihi = MYMIN((ilo+blocksize),(rows-1));
        for(jb=0;jb<nbc;jb++) {
            jlo = jb*blocksize;
            jhi = MYMIN((jlo+blocksize),(cols-1));
            ipass++;
            if ((ipass%nproc)==me) {
                gen_C(block2use,ilo,ihi,jlo,jhi,rows,cols,crossdim);
                lo[0]=ilo;   lo[1]=jlo;
                hi[0]=ihi;   hi[1]=jhi;
                nsize = (jhi−jlo+1);
                NGA_Put(g_C,lo,hi,block2use,&nsize);
            }
        }
    }
    free(block2use);
}
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Copyright (c) 2000, 2001, 2002, 2003
Iowa State University of Science and Technology

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
$Id: gen_A.c,v 1.3 2005/04/04 04:53:31 rickyk Exp $
Written by Ricky A. Kendall, Ames Laboratory.

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "genABC.h"

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Routine to generate matrix A
ilo, ihi, jlo, jhi are C based indices values of the matrix,
NOT loop limits.
rowdim_A, coldim_A is the dimension of the matrix A.

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
void gen_A(double *buffer,
    int ilo, int ihi,
    int jlo, int jhi,
    int rowdim_A, int coldim_A)
{
    /*
    functional form of A(i,j) = ai + bj + c
    */
    int i, j, count;    /* buffer index and loop indices */
    int iend, jend;    /* loop limits */
    double a, b, c;    /* equation constants */
    int til, tih, tjl, tjh;    /* argument validity test variables */
    /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
    Test validity of argument ranges
    /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
    til = (ilo < 0 || ilo > (rowdim_A-1));     /* is ilo valid */
    tih = (ihi < 0 || ihi > (rowdim_A-1));     /* ihi */
    tjl = (jlo < 0 || jlo > (coldim_A-1));     /* jlo */
    tjh = (jhi < 0 || jhi > (coldim_A-1));     /* jhi */
    if (til || tih || tjl || tjh ) {
        (void)printf("−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−\n");
        (void)printf(" gen_A: fatal argument error %d%dl%dl%dl\n",til,tih,tjl,tjh);
(void) printf("I range %d to %d \n", ilo, ihi);
(void) printf("J range %d to %d \n", jlo, jhi);
(void) printf(" rows=%d columns=%d \n", rowdim_A, coldim_A);
(void) exit((int) 911); /* in case of emergency call */
}

/* initialize constants from definitions in the generation include file. Initialize count which indexes the input/output buffer.*/
a = A_VAL;
b = B_VAL;
c = C_VAL;

/* Determine end loops so that (0,0,0,0) will give the first element*/
if (iend > rowdim_A)  
  iend=rowdim_A; /* should not be executed but for safety */
if (jend > coldim_A)  
  jend=coldim_A; /* should not be executed but for safety */

/* Compute desired element, patch, or full matrix of Matrix A*/
count = 0;
for (i=ilo;i<iend;i++) {
  for (j=jlo;j<jend;j++) {
    buffer[count] = a*(double)i +b*(double)j + c;
    if (PRINTIT)
      (void) printf("A count=%d, i=%d j=%d iend=%d jend=%d value=%f\n", count, i, j, iend, jend, buffer[count]);
    count++;
  }
}

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "genABC.h"

Routine to generate matrix B

ilo, ihi, jlo, jhi are C based indices values of the matrix, NOT loop limits.
rowdim_A, coldim_A is the dimension of the matrix B.

void gen_B(double *buffer,
           int ilo, int ihi,
           int jlo, int jhi,
           int rowdim_B, int coldim_B)
{
    /*
     * functional form of B(i,j) = di + ej + f
     */
    int i, j, count;    /* buffer index and loop indices */
    int iend, jend;    /* loop limits */
    double d, e, f;    /* equation constants */
    int til, tih, tjl, tjh;    /* argument validity test variables */
    
    Test validity of argument ranges
    
    til = (ilo < 0 || ilo > (rowdim_B-1));    /* is ilo valid */
    tih = (ihi < 0 || ihi > (rowdim_B-1));    /* ihi */
    tjl = (jlo < 0 || jlo > (coldim_B-1));    /* jlo */
    tjh = (jhi < 0 || jhi > (coldim_B-1));    /* jhi */
    if (til || tih || tjl || tjh ) {
        (void)printf("----------********----------\n");
        (void)printf(" gen_B: fatal argument error %d%d%d%d\n",til,tih,tjl,tjh);
(void)printf("I range %d to %d \n",ilo,ihi);
(void)printf("J range %d to %d \n",jlo,jhi);
(void)printf("rows=%d columns=%d \n",rowdim_B,coldim_B);
(void)exit((int) 911); /* in case of emergency call */
}

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
initialize constants from definitions in the
generation include file. Initialize count which
indexes the input/output buffer.
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
d = D_VAL;
e = E_VAL;
f = F_VAL;
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Determine end loops so that (0,0,0,0) will give the first element
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
iend = ihi + 1;
if (iend > rowdim_B) iend=rowdim_B;
jend = jhi + 1;
if (jend > coldim_B) jend=coldim_B;
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Compute desired element, patch, or full matrix of Matrix B
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
count = 0;
for (i=ilo;i<iend;i++) {
  for (j=jlo;j<jend;j++) {
    buffer[count] = d*(double)i + e*(double)j + f;
    if (PRINTIT)
      (void)printf("B count=%d, i=%d j=%d iend=%d jend=%d value=%f\n",
                  count,i,j,iend,jend,buffer[count]);
    count++;
  }
}
/*-−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Copyright (c) 2000, 2001, 2002, 2003
Iowa State University of Science and Technology

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
/*-−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

Written by Ricky A. Kendall, Ames Laboratory.

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "genABC.h"

/*-−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*
Routine to generate matrix C
ilo, ihi, jlo, jhi are C based indices values of the matrix,
N OT loop limits.
rowdim_A, coldim_A is the dimension of the matrix C.
crossDim is the inner dimension (Columns_of_A or Rows_of_B)
of the matrix multiply
/*-−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
void gen_C(double *buffer, int ilo, int ihi, int jlo, int jhi,
int rowdim_C, int coldim_C, int crossDim)
{
    /*
functional form of A(i,j) = ai + bj +c
functional form of B(i,j) = di + ej +f
C(i,j) = sum(k,k=0,crossDim-1)
    {k*(d*(a*i+b*k+c)+b*(e*j+f))+(a*i+c)*(e*j+f)}
*/
    int i, j, count;          /* buffer index and loop indices */
    int iend, jend;          /* loop limits */
    double a, b, c;          /* equation constants for A*/
    double d, e, f;          /* more equation constants for B*/
    int til, tih, tjl, tjh;  /* argument validity test variables */
    double const0, const1, const2;
    double dN, aipc, ejpf;
    /*-----------------------------*/
    Test validity of argument ranges
    /*-----------------------------*/
    til = (ilo < 0 || ilo > (rowdim_C-1)) ; /* is ilo valid */
```c
int tih = (ihi < 0 || ihi > (rowdim_C-1)); /* ihi */
int tjl = (jlo < 0 || jlo > (coldim_C-1)); /* jlo */
int tjh = (jhi < 0 || jhi > (coldim_C-1)); /* jhi */
if (tih || tih || tjl || tjh ) {
    (void)printf("−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−");
    (void)printf("gen_C: fatal argument error %d%d%d%d\n",til,tih,tjl,tjh);
    (void)printf("I range %d to %d \n",ilo,ihi);
    (void)printf("J range %d to %d \n",jlo,jhi);
    (void)printf("rows=%d columns=%d \n",rowdim_C,coldim_C);
    (void)exit((int) 911); /* in case of emergency call */
}

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
initialize constants from definitions in the
generation include file. Initialize count which
indexes the input/output buffer.
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
a = A_VAL;  b = B_VAL;  c = C_VAL;
d = D_VAL;  e = E_VAL;  f = F_VAL;
count = 0;
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Determine end loops so that (0,0,0,0) will give the first element
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
iend = ihi + 1;
if (iend > rowdim_C) iend=rowdim_C;
jend = jhi + 1;
if (jend > coldim_C) jend=coldim_C;
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
Compute desired element, patch, or full matrix of Matrix C
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
dN = (double)crossDim;
for (i=ilo;i<iend;i++) {
    aipc = (a*(double)i+c);
    for (j=jlo;j<jend;j++) {
        ejpf = (e*(double)j+f);
        const0 = aipc*ejpf*dN;
        const1 = (d*aipc+b*ejpf)*dN*(dN-1)/(double)2;
        const2 = b*d*dN*(dN-1)*((double)2*dN-1)/(double)6;
        buffer[count] = const0 + const1 + const2;
        count++;
    }
}
```
#include <stdio.h>
#include <stdlib.h>
#include "macommon.h"
#include "macdecls.h"
#include "ga.h"
#include "mymin.h"

void zeroit(double *buf, int size)
{
    int i;
    for(i=0;i<size;i++)
        buf[i] = (double)0.0;
}

void ga_blocked(int g_A, int g_B, int g_C, int blocksize)
{
    int ndim, dims[2], lo[2], hi[2];
    int rowsA, colsA, rowsB, colsB, rowsC, colsC;
    int type;
    int ib, nbi, jb, nbj, kb, nbk;
    int ilo, ihi, jlo, jhi, klo, khi;
    int nsizes, nsizex, nsizexj;
    int ipa, ipb, ipc;
    int i, j, k;
    int me, nproc, ipass;
    double *bufa, *bufb, *bufc;
    double one = (double)1.0;
    double vala;

    NGA_Inquire(g_A, &type, &ndim, dims);
    if (ndim != 2)
        GA_Error("Number of dimensions of A is wrong!", ndim);
    rowsA = dims[0];
    colsA = dims[1];
    NGA_Inquire(g_B, &type, &ndim, dims);
    if (ndim != 2)
        GA_Error("Number of dimensions of B is wrong!", ndim);
    rowsB = dims[0];
    colsB = dims[1];
    rowsC = rowsA;
    colsC = colsB;

    /*-----------------------------------------------*
    * Owner computes over blocks of A and B for C component
    */
    /*-----------------------------------------------*/
    me = GA_Nodeid();
    nproc = GA_Nnodes();
    NGA_Distribution(g_C, me, lo, hi);

    nbi = rowsC/blocksize;
    nbj = colsC/blocksize;
    nbk = colsA/blocksize;
    if (rowsC%blocksize) nbi++;
    if (colsC%blocksize) nbj++;
    if (colsA%blocksize) nbk++;
    bufa = (double *)malloc(blocksize*blocksize*sizeof(double));
    bufb = (double *)malloc(blocksize*blocksize*sizeof(double));
bufc = (double *)malloc(blocksize*blocksize*sizeof(double));
zeroit(bufa,(blocksize*blocksize));
zeroit(bufb,(blocksize*blocksize));
zeroit(bufc,(blocksize*blocksize));

ipass = -1;
for(ib=0;ib<nbi;ib++) {
ilo = ib*blocksize;
ithi = MYMIN((ilo+blocksize-1),(rowsC-1));
nsizei = (ithi-ilo+1);
ipass++;
if ((ipass%nproc)==me) {
    for(kb=0;kb<nbk;kb++) {
        klo = kb*blocksize;
khi = MYMIN((klo+blocksize-1),(colsA-1));
        nsizek = (khi-klo+1);
        lo[0]=ilo; hi[0]=ithi;
        lo[1]=klo; hi[1]=khi;
        NGA_Get(g_A,lo,hi,bufa,&nsizek);
        for(jb=0;jb<nbj;jb++) {
            jlo = jb*blocksize;
jhi = MYMIN((jlo+blocksize-1),(colsC-1));
            nsizej = (jhi-jlo+1);
            lo[0]=klo; hi[0]=khi;
            lo[1]=jlo; hi[1]=jhi;
            NGA_Get(g_B,lo,hi,bufb,&nsizej);
            for(i=ilo;i<=ithi;i++) {
                for(k=klo;k<=khi;k++) {
                    ipa = (i-ilo)*nsizek+(k-klo);
                    vala = bufa[ipa];
                    for(j=jlo;j<=jhi;j++) {
                        ipb = (k-klo)*nsizej+(j-jlo);
                        ipc = (i-ilo)*nsizej+(j-jlo);
                        bufc[ipc] += vala*bufb[ipb];
                    }
                }
            }
        }
    }
}
zeroit(bufc,(nsizei*nsizej));

free(bufa);
free(bufb);
free(bufc);
/* File scoped print boolean for matrices and information */
#define PRINTIT 1

routine to print matrices. Used only for testing phase.

```
#include <stdio.h>

void print_mat(double buffer[], int num_row, int num_col)
{
    int i, j;
    int count = 0;
    if (PRINTIT) {
        (void)printf("\n\n−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−\n");
        for(i=0;i<num_row;i++) {
            for(j=0;j<num_col;j++) {
                (void)printf("%10f ",buffer[count]);
                count++;
            }
        }
        (void)printf("\n\n−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−\n");
        (void)fflush(stdout);
    }
}
```
Copyright (c) 2000, 2001, 2002, 2003
Iowa State University of Science and Technology

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>
#include <stdio.h>

double CPU_Time(void)
{
    struct rusage rusage_out;

    (void) getrusage (RUSAGE_SELF, &rusage_out);

    return ((double)rusage_out.ru_utime.tv_usec* 1E-6 +
        (double)(rusage_out.ru_utime.tv_sec) +
        (double)rusage_out.ru_stime.tv_usec* 1E-6 +
        (double)(rusage_out.ru_stime.tv_sec));
}
Return the current Wall Time Stamp in seconds
uses standard gettimeofday function

#include <sys/time.h>
#include <unistd.h>
double Wall_Time(void)
{
  struct timeval mytp;
  double seconds;
  if(!(gettimeofday(&mytp, (struct timezone *)NULL))) {
    seconds = (double) mytp.tv_sec;
    seconds += (double) mytp.tv_usec*(double)1.0e-06;
    return seconds;
  }
  else {
    return (double) 911;
  }
}
<table>
<thead>
<tr>
<th></th>
<th>File Name</th>
<th>Sheets</th>
<th>Pages</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Makefile</td>
<td>1 to 1</td>
<td>1− 1</td>
<td>1 (1)</td>
</tr>
<tr>
<td>2</td>
<td>main.c</td>
<td>2 to 6</td>
<td>2− 6</td>
<td>5 (5)</td>
</tr>
<tr>
<td>3</td>
<td>gen_ga_A.c</td>
<td>7 to 7</td>
<td>7− 7</td>
<td>1 (1)</td>
</tr>
<tr>
<td>4</td>
<td>gen_ga_B.c</td>
<td>8 to 8</td>
<td>8− 8</td>
<td>1 (1)</td>
</tr>
<tr>
<td>5</td>
<td>gen_ga_C.c</td>
<td>9 to 9</td>
<td>9− 9</td>
<td>1 (1)</td>
</tr>
<tr>
<td>6</td>
<td>gen_A.c</td>
<td>10 to 11</td>
<td>10− 11</td>
<td>2 (2)</td>
</tr>
<tr>
<td>7</td>
<td>gen_B.c</td>
<td>12 to 13</td>
<td>12− 13</td>
<td>2 (2)</td>
</tr>
<tr>
<td>8</td>
<td>gen_C.c</td>
<td>14 to 15</td>
<td>14− 15</td>
<td>2 (2)</td>
</tr>
<tr>
<td>9</td>
<td>ga_blocked.c</td>
<td>16 to 17</td>
<td>16− 17</td>
<td>2 (2)</td>
</tr>
<tr>
<td>10</td>
<td>print_mat.c</td>
<td>18 to 18</td>
<td>18− 18</td>
<td>1 (1)</td>
</tr>
<tr>
<td>11</td>
<td>CPU_Time.c</td>
<td>19 to 19</td>
<td>19− 19</td>
<td>1 (1)</td>
</tr>
<tr>
<td>12</td>
<td>Wall_Time.c</td>
<td>20 to 20</td>
<td>20− 20</td>
<td>1 (1)</td>
</tr>
</tbody>
</table>