

Hypre Project

Raphael Bar-Or, Janine Kennedy, Kourosh

Matlab Section:	2
Matlab Results:.....	3
Instructions for Installing Hypre	5
Basic Build of Hypre.....	5
Building with gcc	6
Matlab Source Code.....	8
Hypre Source Code	9

Matlab Section:

In this project we were asked to evaluate the convergence performance of the pe3k.mtx matrix added to the identity given various preconditioners determined by availability in a particular package (in our case this was Hypre). The thought was that we would first solve the problem in Matlab with incomplete Cholesky factorization as a preconditioner to see what kind of speedup we should expect. We evaluated this behavior in both the number of iterations to convergence (rate of convergence), and the total time to convergence including the calculation of the preconditioner.

Matlab Results:

It is important to note that the problem is quite ill-conditioned if we omit adding the identity to `pe3k.mtx`. We initially attempted to solve the ill conditioned problem and found it to be unsolvable with incomplete Cholesky, while it converged very slowly without preconditioning.

The convergence for $(pe3k.mtx + I)x = b$ where b is a vector of ones is shown in the following figures. Figure 1 shows the rate of convergence with respect to the residual at each iteration for various choices of drop tolerance. Note that as the drop tolerance decreases (richer preconditioner) the slope of converges increases. The problem converges in 8 iterations when using a drop tolerance of $1e-4$ while it converges in 80 iterations with no preconditioning. Note that for very coarse drop tolerances the convergence is actually worse than with no preconditioning at all (droptol= 1 & 0.1).

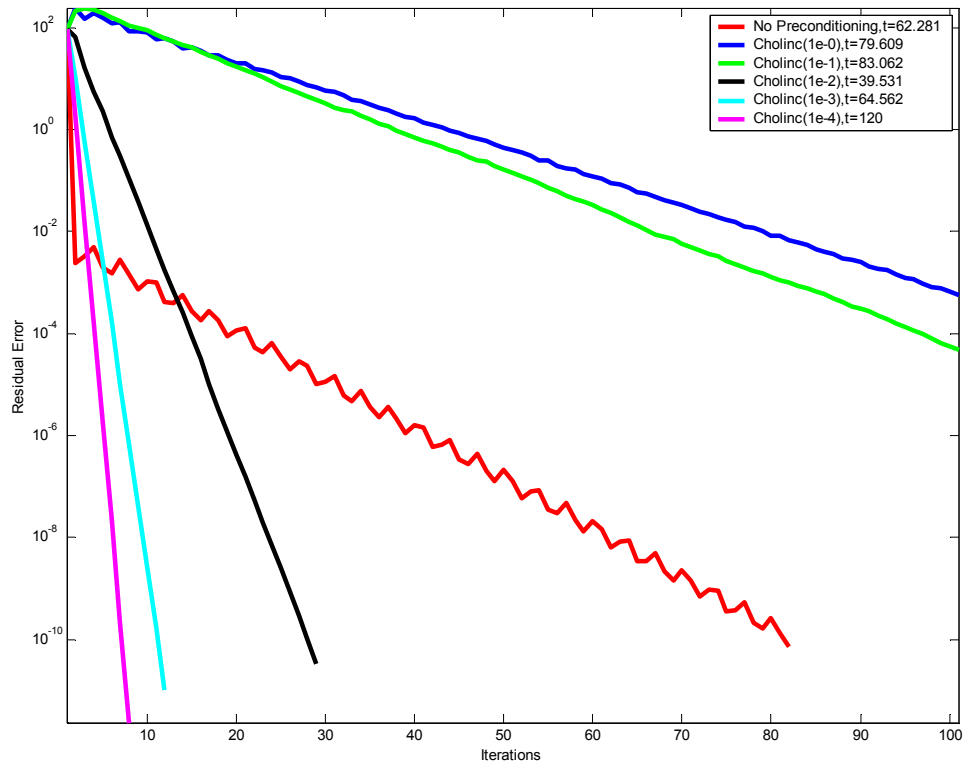


Figure 1: Convergence rates for various preconditioners

From figure 1 it may seem that the richer preconditioners are always preferred. To confirm this we must consider not only the convergence rate, but also the time it takes to create the preconditioners. We investigate this behavior in figure 2 where we time the problem including any preprocessing such as calculation of preconditioners.

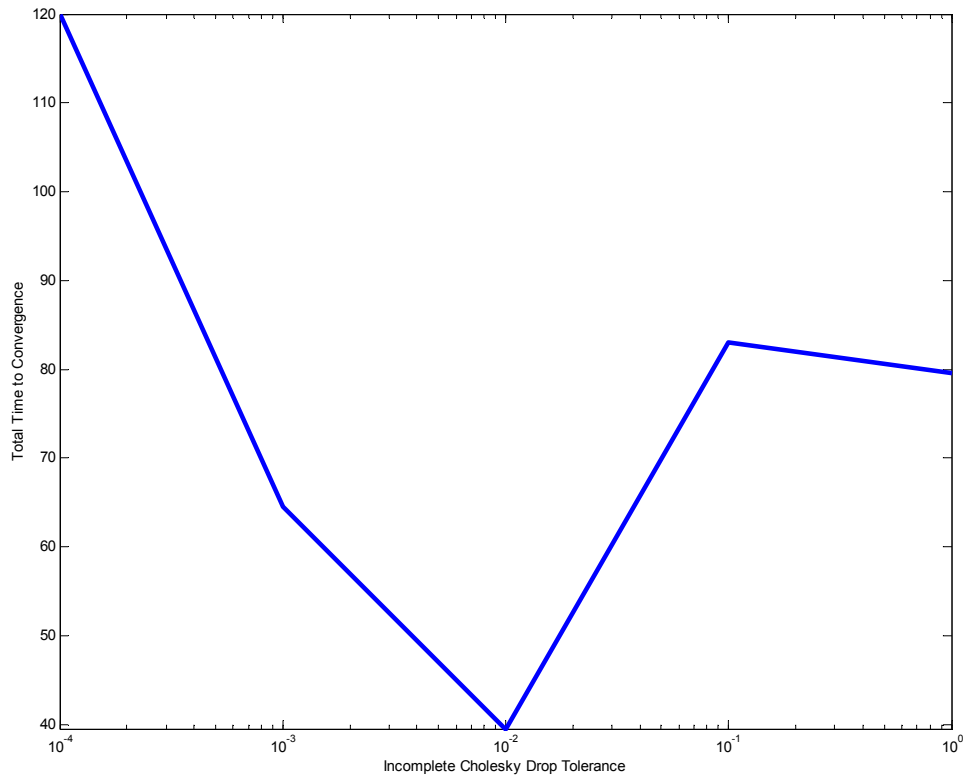


Figure 2: Convergence rates for various drop tolerances

We see here that when considering total time to convergence, an intermediate preconditioner is actually preferable. We note that a drop tolerance of $1e-2$ seems to be near optimal for this type of preconditioning. This result underscores the need to balance the complexity of calculating the preconditioner with the incremental improvement in convergence.

Instructions for Installing Hypre

The software and documentation for Hypre may be downloaded for free from the Hypre Project web site: <http://www.llnl.gov/CASC/hypre>

There are many ways to configure and build the hypre package. The most important difference is the choice of MPI library that you select for the message processing interface. Hypre needs to be correctly linked to one of these MPI libraries in order to compile and run correctly.

For instance on beowulf you can choose to link in `/opt/scali/bin` or `/opt/mpich/bin` (this is actually a unix link to the `/opt/mpich-1.2.3/bin` directory). If you choose `/opt/scali/bin` the nodes communicate over the fast SCI connection, otherwise choose `/opt/mpich/bin` to have nodes communicate over the 100 Mb ethernet connection. Due to some problems with the MPI interface on beowulf it is suggested that you link in `/opt/mpich/bin`.

It is necessary to verify that your path includes these directories before proceeding with the following installation instructions. (For oldmath check `/opt/bin` for correct directory names since they are slightly different.)

1. `%echo $PATH`
2. check to see that your path has `/opt/mpich/bin:/opt/scali/bin`. Listing them in your PATH variable in this order forces unix to look in the directory `/opt/mpich/bin` for executables like `mpirun` and `mpicc`, before looking in `/opt/scali/bin`. We add `/opt/scali/bin` to the path as well in case we want to run `mpimon` which is not part of `/opt/mpich/bin`.
3. If you don't see these directories in your PATH. You can remedy this by changing PATH in the `.login` file of your home directory or you can execute the following each time you start a session on beowulf:
4. `%setenv PATH ${PATH}:/opt/mpich/bin:/opt/scali/bin`

Basic Build of Hypre

This build is recommended. It uses hypre's defaults and seems to work consistently on both beowulf and oldmath, provided your path is set as described above. The resulting build uses `/opt/mpich/bin` for message processing and uses the `mpicc` and `mpiCC` compilers, (see notes below for compilation with `gcc`)

1. `%zcat hypre-1.6.0.tar | tar xvf -`
2. `%cd hypre-1.6.0/src`
3. `%configure`
4. `%make`

5. NOTE: If trying a new configuration for hypre it is helpful to execute the following commands:
6. `%make veryclean`
7. `%make`
8. `%cd hypre-1.6.0/test`
9. It has been our experience that `fei_linear_solvers.C` is not compiled during the make executed in step 4. Therefore one can execute:
10. `%mpiCC fei_linear_solvers`
11. `%test_solver.sh`
12. Check the various `*.err` files. They should be empty. `*.log` files have the output from the executables called by the `test_solvers.sh` driver.

Building with gcc

1. Set path as described above.
2. Replace `%configure` from the Basic Build instructions above with:
3. `%configure --with-CC=gcc --with-CXX=g++`
4. `--with-CFLAGS="-D_REENTRANT -I/opt/mpich/include -L/opt/mpich/lib -lmpich" --with-CXXFLAGS="-D_REENTRANT -I/opt/mpich/include -L/opt/mpich/lib -Wl -Bstatic -lmpich -Bdynamic -ldl -lpthread" --with-mpi-include=/opt/mpich/include --with-mpi-lib-dirs=/opt/mpich/lib \with-mpi-libs=mpich`
5. Please note that the configure script is very sensitive about new line characters and spacing.
6. Continue with steps from above.

Hypre is much more difficult to work with than Petsc, this is due to the very poor documentation and the lack of example problems. The `*.c` files in the test directory are an obvious place to begin, but these c-programs are close to 3000 lines long and have almost no in line documentation. This makes it difficult to determine the minimal code necessary for solving the problem set forth in the project description. The following describes the course of action pursued by us in an effort to solve the project problem with the hypre package. At the current time we have a code that compiles without error and exits successfully (does not crash) for small problems, for instance problems with less than 5000 non-zero entries. Unfortunately this code does not converge even when the solution is trivial. In addition we still get segmentation faults when loading the data for the project problem, therefore we were not able to solve the problem.

1. Read documentation and review hypre source code and directory structures.
2. Build hypre. We tried various configurations before realizing that `/opt/scali/bin` was not working properly on beowulf. Based on some information from both Jan and Rob we were able to build on oldmath using mpich.

3. Communicated with Rico and Rob regarding compilation and linking of river.c source.
4. Wrote program and debugged to get command line compilation to working.
5. Learned about Makefiles. Created Makefile after linking issues with MPI were resolved.
6. Debugging took 2 weeks. Successfully compiled code would crash with segmentation violations.
7. The seg faults were due to problems with MPI and hypre. Determining which was which took the bulk of our time.
8. The MPI problem was resolved by switching to /opt/mpich/bin.
9. Spent more time trying different configurations on beowulf with the hope that we code build hypre with gcc and use a debugger to work out the segmentation problems occurring in the functions called by our routine.
10. We determined the error in the hypre code was due to an initialization problem in our hypre code. The fix was quite easy. But the lack of hypre documentation made finding the bug very time consuming.
11. We dropped our pursuit of compiling hypre with the correct symbol tables for functions in the hypre library.
12. Code stopped crashing during load for small problems, but did not converge even for the identity.
13. Proceeded to add a variety of preconditioning matrices to our source code. Trial and error approach was again the only possibility since documentation is not specific about the different solver requirements. All methods had problems during compilation or run except for the HYPRE_ParCSRDiagScale precondition which seems to work. Once again this seemed to be a linking and build issue with Hypres optimized libraries.
14. Implemented timing function which seemed to change the results produced by the solver. Removed timing functions.
15. Implementation with large project problem resulted in new segmentation violations when loading data. Due to time constraints we could not rectify this new issue or problems with convergence of the systems.

Matlab Source Code

```
clear;
load data;
% data must contain a matrix A of and a right hand
% side b
% in this case A=pe3k.mtx+I

% pe3k.mtx was read in using the mmread.m available on
% the web from the matrix market website

tic;
[x,flag,relres,iter,resvec] = pcg(A,b,1e-12,100);
time0=toc
semilogy(1:iter+1,resvec,'-r');
hold on

tic
M = cholinc(A,1e-0);
[x,flag,relres,iter,resvec] = pcg(A,b,1e-12,100,M',M);
time1=toc
semilogy(1:iter+1,resvec,'-b');

tic
M = cholinc(A,1e-1);
[x,flag,relres,iter,resvec] = pcg(A,b,1e-12,100,M',M);
time2=toc
semilogy(1:iter+1,resvec,'-g');

tic
M = cholinc(A,1e-2);
[x,flag,relres,iter,resvec] = pcg(A,b,1e-12,100,M',M);
time3=toc
semilogy(1:iter+1,resvec,'-k');

tic
M = cholinc(A,1e-3);
[x,flag,relres,iter,resvec] = pcg(A,b,1e-12,100,M',M);
time4=toc
semilogy(1:iter+1,resvec,'-c');

tic
M = cholinc(A,1e-4);
[x,flag,relres,iter,resvec] = pcg(A,b,1e-12,100,M',M);
time5=toc
semilogy(1:iter+1,resvec,'-m');

hold off

legend(...
['No Preconditioning,t=' num2str(time0)], ...
['Cholinc(1e-0),t=' num2str(time1)], ...
['Cholinc(1e-1),t=' num2str(time2)], ...
['Cholinc(1e-2),t=' num2str(time3)], ...
['Cholinc(1e-3),t=' num2str(time4)], ...
['Cholinc(1e-4),t=' num2str(time5)] ...
);
axis tight;
xlabel('Iterations');
ylabel('Residual Error');

droptol=[1,1e-1,1e-2,1e-3,1e-4];
times=[time1 time2 time3 time4 time5];
figure;
semilogx(droptol,times);
axis tight;
xlabel('Incomplete Cholesky Drop Tolerance');
ylabel('Total Time to Convergence');
```

Hypre Source Code

```

/*****
/* program for reading a file into IJMatrix and Solving */
*****/

#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <assert.h>

/*-----*
 * HYPRE includes
 *-----*/

#include "utilities/utilities.h"
#include "HYPRE.h"
#include "parcsr_mv/HYPRE_parcsr_mv.h"
#include "IJ_mv.h"
#include "IJ_mv/HYPRE_IJ_mv.h"
#include "HYPRE_parcsr_ls.h"
#include "krylov.h"

/*-----*
 * local defines and local and external functions
 *-----*/

extern void HYPRE_LSI_Get_IJAMatrixFromFile(double **val, int **ia,
      int **ja, int *N, double **rhs, char *matfile, char *rhsfile);

/*****
/* main program */
*****/

main(int argc, char *argv[])
{
    int    mypid, nprocs, *ia, *ja, nrows, nnz, chunksize, mybegin, myend;
    int    i, local_nrows, blksize=1, ierr, *row_sizes, ncnt;
    int    num_iterations;
    double final_res_norm;
    double tol = 1.e-6;
    double *val, *rhs, *values;
    void   * object;
    HYPRE_IJMatrix    IJA;
    HYPRE_IJVector    IJB;
    HYPRE_IJVector    IJX;
    HYPRE_ParCSRMatrix A_csr;
    HYPRE_ParVector    b_csr;
    HYPRE_ParVector    x_csr;
    HYPRE_Solver        pcg_solver;
    HYPRE_Solver        pcg_precond, pcg_precond_gotten;

    /*-----*
     * initialize parallel platform
     *-----*/

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &mypid);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

    /*-----*
     * read the matrix and rhs and broadcast
     *-----*/

```

```

if ( mypid == 0 )
{
    HYPRE_LSI_Get_IJAMatrixFromFile(&val, &ia, &ja, &nrows, &rhs,
                                   "matrix.data", "rhs.data");

    nnz = ia[nrows];
    MPI_Bcast(&nrows, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&nnz, 1, MPI_INT, 0, MPI_COMM_WORLD);

    MPI_Bcast(ia, nrows+1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(ja, nnz, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(val, nnz, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(rhs, nrows, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
else
{
    MPI_Bcast(&nrows, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&nnz, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if ((ia=(int *) malloc((nrows+1)*sizeof(int)))==NULL){
        printf("Could not allocate memory.\n");
        assert(0);
    }
    if ((ja=(int *) malloc(nnz*sizeof(int)))==NULL){
        printf("Could not allocate memory.\n");
        assert(0);
    }
    if ((val=(double *) malloc(nnz*sizeof(double)))==NULL){
        printf("Could not allocate memory.\n");
        assert(0);
    }
    if ((rhs=(double *) malloc(nrows*sizeof(double)))==NULL){
        printf("Could not allocate memory.\n");
        assert(0);
    }

    MPI_Bcast(ia, nrows+1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(ja, nnz, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(val, nnz, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(rhs, nrows, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}

chunksize = nrows / blksize;
if ( chunksize * blksize != nrows )
{
    printf("Cannot put into matrix blocks with block size 3\n");
    exit(1);
}
chunksize = chunksize / nprocs;
mybegin = chunksize * mypid * blksize;
myend = chunksize * (mypid + 1) * blksize - 1;
if ( mypid == nprocs-1 ) myend = nrows - 1;
printf("Processor %d : begin/end = %d %d\n", mypid, mybegin, myend);
fflush(stdout);

/*-----
 * create matrix in the HYPRE context
 *-----*/

local_nrows = myend - mybegin + 1;
ierr = HYPRE_IJMatrixCreate(MPI_COMM_WORLD, mybegin, myend,
                            mybegin, myend, &IJA);

ierr += HYPRE_IJMatrixSetObjectType(IJA, HYPRE_PARCSR);
assert(!ierr);

row_sizes = (int *) malloc( local_nrows * sizeof(int) );
for ( i = mybegin; i <= myend; i++ )
    row_sizes[i-mybegin] = ia[i+1] - ia[i];

```

```

ierr = HYPRE_IJMatrixSetRowSizes(IJA, row_sizes);
ierr += HYPRE_IJMatrixInitialize(IJA);
assert(!ierr);
free( row_sizes );

for ( i = mybegin; i <= myend; i++ )
{
    ncnt = ia[i+1] - ia[i];
    ierr = HYPRE_IJMatrixSetValues(IJA, 1, &ncnt, (const int *) &i,
                                   (const int *) &(ja[ia[i]]), (const double *) &(val[ia[i]]));
    assert(!ierr);
}
ierr = HYPRE_IJMatrixAssemble(IJA);
assert( !ierr );
HYPRE_IJMatrixGetObject(IJA, &object);
A_csr = (HYPRE_ParCSRMatrix) object;

/*Initialize RHS vector*/
HYPRE_IJVectorCreate(MPI_COMM_WORLD, mybegin, myend, &IJB);
HYPRE_IJVectorSetObjectType(IJB, HYPRE_PARCSR);
ierr = HYPRE_IJVectorInitialize(IJB);
for ( i = mybegin; i <= myend; i++ )
    HYPRE_IJVectorSetValues(IJB, 1, (const int *) &i,
                            (const double *) &(rhs[i]));
ierr += HYPRE_IJVectorAssemble(IJB);
assert(!ierr);

ierr = HYPRE_IJVectorGetObject(IJB, &object );
b_csr = (HYPRE_ParVector) object;

/*Initialize solution vector*/
HYPRE_IJVectorCreate(MPI_COMM_WORLD, mybegin, myend, &IJX);
HYPRE_IJVectorSetObjectType(IJX, HYPRE_PARCSR);
ierr = HYPRE_IJVectorInitialize(IJX);

values = hypre_CTAlloc(double, local_nrows);
for (i = 0; i < local_nrows; i++)
    values[i] = 0.;
HYPRE_IJVectorSetValues(IJX, local_nrows, NULL, values);
hypre_TFree(values);

ierr += HYPRE_IJVectorAssemble(IJX);
assert(!ierr);

ierr = HYPRE_IJVectorGetObject(IJX, &object );
x_csr = (HYPRE_ParVector) object;

/*-----
 * solve with pcg
 *-----*/

HYPRE_ParCSRPCGCreate(MPI_COMM_WORLD, &pcg_solver);

HYPRE_ParCSRPCGSetMaxIter(pcg_solver, 500);
HYPRE_ParCSRPCGSetTol(pcg_solver, tol);
HYPRE_ParCSRPCGSetTwoNorm(pcg_solver, 1);
HYPRE_ParCSRPCGSetRelChange(pcg_solver, 0);
HYPRE_ParCSRPCGSetLogging(pcg_solver, 1);

/*-----*/
/* use diagonal scaling as preconditioner */
/*-----*/
pcg_precond = NULL;

HYPRE_ParCSRPCGSetPrecond(pcg_solver,
                          (HYPRE_PtrToSolverFcn) HYPRE_ParCSRDiagScale,
                          (HYPRE_PtrToSolverFcn) HYPRE_ParCSRDiagScaleSetup,
                          pcg_precond);

HYPRE_ParCSRPCGGetPrecond(pcg_solver, &pcg_precond_gotten);
if (pcg_precond_gotten != pcg_precond)
{

```

```

        printf("HYPRE_ParCSRPCGGetPrecond got bad precondition\n");
        return(-1);
    }
    else if (mypid == 0)
        printf("HYPRE_ParCSRPCGGetPrecond got good precondition\n");

/*-----*/
/*Set up Solver and solve*/
/*-----*/
    HYPRE_ParCSRPCGSetup(pcg_solver, A_csr, b_csr, x_csr);

    HYPRE_ParCSRPCGSolve(pcg_solver, A_csr, b_csr, x_csr);
    HYPRE_ParCSRPCGGetNumIterations(pcg_solver, &num_iterations);
    HYPRE_ParCSRPCGGetFinalRelativeResidualNorm(pcg_solver, &final_res_norm);

    HYPRE_ParCSRPCGDestroy(pcg_solver);

    if (mypid == 0)
    {
        printf("\n");
        printf("Iterations = %d\n", num_iterations);
        printf("Final Relative Residual Norm = %e\n", final_res_norm);
        printf("\n");
    }

/*-----*/
/* clean up
*-----*/
    free( ia );
    free( ja );
    free( val );
    free( rhs );

    HYPRE_IJMatrixDestroy(IJA);
    HYPRE_IJVectorDestroy(IJB);
    HYPRE_IJVectorDestroy(IJX);
    MPI_Finalize();
}

```

Makefile

```

# Generated automatically from Makefile.in by configure.
#BHEADER*****
# (c) 1998 The Regents of the University of California
#
# See the file COPYRIGHT_and_DISCLAIMER for a complete copyright
# notice, contact person, and disclaimer.
#
# $Revision: 2.7 $
#EHEADER*****

.SUFFIXES:
.SUFFIXES: .c .C .f .o .CXX.o

srcdir = .

CCpg=pgcc
CC= mpicc
CXX= mpiCC
F77=mpif77

C_COMPILE_FLAGS= -O
CXX_COMPILE_FLAGS= -O
F77_COMPILE_FLAGS= -O
LD_LINK_FLAGS=
INCLUDES=
CDEFS = -DHYPRE_TIMING
BABEL_HYPRE_INCLUDES =

CFLAGS = \

```

```
{C_COMPILE_FLAGS}\
-I$(srcdir)\
-I$(srcdir)/..\
${BABEL_HYPRE_INCLUDES}\
-I../hypre/include\
${CINCLUDES}\
${CDEFS}

FFLAGS = \
  ${F77_COMPILE_FLAGS}\
  ${BABEL_HYPRE_INCLUDES}

MPILIBFLAGS =
LIBFLAGS = -lm
LDLIBFLAGS =
BLASLIBFLAGS = -lhypre_blas
PETSCLIBFLAGS =
BABELLIBFLAGS =

LFLAGS =\
  ${LD_LINK_FLAGS}\
  -L../hypre/lib\
  -lhypre_sstruct_ls\
  -lhypre_sstruct_mv\
  -lhypre_struct_ls\
  -lhypre_struct_mv\
  -lhypre_parcsr_ls\
  -lhypre_DistributedMatrixPilotSolver \
  -lhypre_ParaSails \
  -lhypre_Euclid \
  -lhypre_MatrixMatrix \
  -lhypre_DistributedMatrix \
  -lhypre_IJ_mv\
  -lhypre_parcsr_mv\
  -lhypre_seq_mv\
  -lkrylov\
  -lhypre_utilities\
  ${PETSCLIBFLAGS} \
  ${BLASLIBFLAGS} \
  ${MPILIBFLAGS} ${LIBFLAGS} ${LDLIBFLAGS}

#####
# Beta definitions
#####

BETA_CFLAGS = ${CFLAGS}

BETA_LFLAGS =\
  ${LD_LINK_FLAGS}\
  -L../hypre/lib\
  -lhypre_sstruct_ls\
  -lhypre_sstruct_mv\
  -lhypre_struct_ls\
  -lhypre_struct_mv\
  -lhypre_parcsr_ls\
  -lhypre_DistributedMatrixPilotSolver \
  -lhypre_ParaSails \
  -lhypre_Euclid \
  -lhypre_MatrixMatrix \
  -lhypre_DistributedMatrix \
  -lhypre_IJ_mv\
  -lhypre_parcsr_mv\
  -lhypre_seq_mv\
  -lkrylov\
  -lhypre_utilities\
  ${PETSCLIBFLAGS} \
  ${BLASLIBFLAGS} \
  ${MPILIBFLAGS} ${LIBFLAGS} ${LDLIBFLAGS}

BETA77_LFLAGS =\
  ${LD_LINK_FLAGS}\
  -L.\
```

```
-L../hypre/lib\  
-lHYPRE_parcsr_ls\  
-lHYPRE_DistributedMatrixPilotSolver \  
-lHYPRE_ParaSails \  
-lHYPRE_Euclid \  
-lHYPRE_MatrixMatrix \  
-lHYPRE_DistributedMatrix \  
-lHYPRE_IJ_mv\  
-lHYPRE_parcsr_mv\  
-lHYPRE_seq_mv\  
-lHYPRE_struct_ls\  
-lHYPRE_struct_mv\  
-lHYPRE_utilities\  
-lkrylov\  
{BLASLIBFLAGS} \  
{MPILIBFLAGS} {LIBFLAGS} {LDLIBFLAGS}  
  
STRUCT_LFLAGS =\  
{LD_LINK_FLAGS}\  
-L../hypre/lib\  
-lHYPRE_struct_ls\  
-lHYPRE_struct_mv\  
-lkrylov\  
-lHYPRE_utilities\  
{PETSCLIBFLAGS} \  
{MPILIBFLAGS} {LIBFLAGS} {LDLIBFLAGS}  
  
SSTRUCT_LFLAGS =\  
{LD_LINK_FLAGS}\  
-L../hypre/lib\  
-lHYPRE_sstruct_ls\  
-lHYPRE_sstruct_mv\  
-lHYPRE_struct_ls\  
-lHYPRE_struct_mv\  
-lHYPRE_parcsr_ls\  
-lHYPRE_DistributedMatrixPilotSolver \  
-lHYPRE_ParaSails \  
-lHYPRE_MatrixMatrix \  
-lHYPRE_DistributedMatrix \  
-lHYPRE_IJ_mv\  
-lHYPRE_parcsr_mv\  
-lHYPRE_seq_mv\  
-lkrylov\  
-lHYPRE_utilities\  
{PETSCLIBFLAGS} \  
{BLASLIBFLAGS} \  
{MPILIBFLAGS} {LIBFLAGS} {LDLIBFLAGS}  
  
IJ_LS_LFLAGS =\  
{LD_LINK_FLAGS}\  
-L../hypre/lib\  
-lHYPRE_parcsr_ls\  
-lHYPRE_DistributedMatrixPilotSolver \  
-lHYPRE_ParaSails \  
-lHYPRE_Euclid \  
-lHYPRE_MatrixMatrix \  
-lHYPRE_DistributedMatrix \  
-lHYPRE_IJ_mv\  
-lHYPRE_parcsr_mv\  
-lHYPRE_seq_mv\  
-lkrylov\  
-lHYPRE_utilities\  
{PETSCLIBFLAGS} \  
{BLASLIBFLAGS} \  
{MPILIBFLAGS} {LIBFLAGS} {LDLIBFLAGS}  
  
IJ_MV_LFLAGS =\  
{LD_LINK_FLAGS}\  
-L../hypre/lib\  
-lHYPRE_parcsr_ls\  
-lHYPRE_DistributedMatrixPilotSolver \  
-lHYPRE_ParaSails \  
-lHYPRE_Euclid \  
-lHYPRE_MatrixMatrix \  
-lHYPRE_DistributedMatrix \  
-lHYPRE_IJ_mv\  
-lHYPRE_parcsr_mv\  
-lHYPRE_seq_mv\  
-lkrylov\  
-lHYPRE_utilities\  
{PETSCLIBFLAGS} \  
{BLASLIBFLAGS} \  
{MPILIBFLAGS} {LIBFLAGS} {LDLIBFLAGS}
```

```
-lHYPRE_Euclid \
-lHYPRE_MatrixMatrix \
-lHYPRE_DistributedMatrix \
-lHYPRE_IJ_mv\
-lHYPRE_parcsr_mv\
-lHYPRE_seq_mv\
-lkrylov\
-lHYPRE_utilities\
${PETSCLIBFLAGS} \
${BLASLIBFLAGS} \
${MPILIBFLAGS} ${LIBFLAGS} ${LDLIBFLAGS}

FEICXX_LFLAGS =\
${LD_LINK_FLAGS}\
-L../hypre/lib\
-lHYPRE_LSI\
${BLASLIBFLAGS} \
${MPILIBFLAGS} ${LIBFLAGS} ${LDLIBFLAGS}

HYPRE_DIR = /home/grads/rargenta/hypre-1.6.0/src

#####
# Targets
#####

HYPRE_STANDARD_DRIVERS =\
  readijmatrix.c

HYPRE_BABEL_DRIVERS =\
  struct_linear_solvers_b.c\
  IJ_linear_solvers_b.c

HYPRE_DRIVERS_CXX =\
  ${HYPRE_DRIVERS:.c=.CXX.C}

HYPRE_FEI_DRIVERS_CXX =\
  fei_linear_solvers.C

HYPRE_STANDARD_DRIVERS_F77 =\
  f77_IJ_linear_solvers.f\
  f77_IJ_matrix_vector.f\
  f77_struct_linear_solvers.f

HYPRE_BABEL_DRIVERS_CXX =

HYPRE_BABEL_DRIVERS_F77 =\
  f77_struct_linear_solvers_b.f

HYPRE_BETA_DRIVERS =

HYPRE_BETA_DRIVERS_F77 =

#ifdef is gmake only, can't be used...
#ifdef BABELLIBFLAGS
  HYPRE_DRIVERS = ${HYPRE_STANDARD_DRIVERS}
  HYPRE_DRIVERS_F77 = ${HYPRE_STANDARD_DRIVERS_F77}
#else
# HYPRE_DRIVERS = ${HYPRE_STANDARD_DRIVERS} ${HYPRE_BABEL_DRIVERS}
# HYPRE_DRIVERS_F77 = ${HYPRE_STANDARD_DRIVERS_F77} ${HYPRE_BABEL_DRIVERS_F77}
#endif

HYPRE_DRIVER_OBJS=${HYPRE_DRIVERS:.c=.o}
HYPRE_DRIVER_CXX_OBJS=${HYPRE_DRIVERS_CXX:.C=.o}
HYPRE_DRIVER_F77_OBJS=${HYPRE_DRIVERS_F77:.f=.o}
HYPRE_BETA_DRIVER_OBJS=${HYPRE_BETA_DRIVERS:.c=.o}
HYPRE_BETA_DRIVER_F77_OBJS=${HYPRE_BETA_DRIVERS_F77:.f=.o}
HYPRE_FEI_DRIVER_CXX_OBJS=${HYPRE_FEI_DRIVERS_CXX:.C=.o}

HYPRE_DRIVER_EXECS=${HYPRE_DRIVERS:.c=}
HYPRE_DRIVER_CXX_EXECS=${HYPRE_DRIVERS_CXX:.C=}
HYPRE_DRIVER_F77_EXECS=${HYPRE_DRIVERS_F77:.f=}
HYPRE_BETA_DRIVER_EXECS=${HYPRE_BETA_DRIVERS:.c=}
```

```
HYPRE_BETA_DRIVER_F77_EXECS=${HYPRE_BETA_DRIVERS_F77:.f=}
HYPRE_FEI_DRIVER_CXX_EXECS=${HYPRE_FEI_DRIVERS_CXX:.C=}

HYPRE_BABEL_DRIVER_OBJS=${HYPRE_BABEL_DRIVERS:.c.o}
HYPRE_BABEL_DRIVER_CXX_OBJS=${HYPRE_BABEL_DRIVERS_CXX:.C.o}
HYPRE_BABEL_DRIVER_F77_OBJS=${HYPRE_BABEL_DRIVERS_F77:.f.o}

HYPRE_BABEL_DRIVER_EXECS=${HYPRE_BABEL_DRIVERS:.c=}
HYPRE_BABEL_DRIVER_CXX_EXECS=${HYPRE_BABEL_DRIVERS_CXX:.C=}
HYPRE_BABEL_DRIVER_F77_EXECS=${HYPRE_BABEL_DRIVERS_F77:.f=}

all: ${HYPRE_DRIVER_EXECS}

all+: ${HYPRE_DRIVER_CXX_EXECS}

all77: ${HYPRE_DRIVER_F77_EXECS}

beta: all ${HYPRE_BETA_DRIVER_EXECS}

beta+: all+

fei+: ${HYPRE_FEI_DRIVER_CXX_EXECS}

beta77: all77 ${HYPRE_BETA_DRIVER_F77_EXECS}

babel: ${HYPRE_BABEL_DRIVER_EXECS} ${HYPRE_BABEL_DRIVER_F77_EXECS}

install:

clean:
    @rm -f *.o

veryclean: clean
    @rm -f ${HYPRE_DRIVER_EXECS}
    @rm -f ${HYPRE_DRIVER_CXX_EXECS}
    @rm -f ${HYPRE_DRIVER_F77_EXECS}
    @rm -f ${HYPRE_BABEL_DRIVER_EXECS}
    @rm -f ${HYPRE_BABEL_DRIVER_CXX_EXECS}
    @rm -f ${HYPRE_BABEL_DRIVER_F77_EXECS}
    @rm -f ${HYPRE_BETA_DRIVER_EXECS}
    @rm -f ${HYPRE_BETA_DRIVER_F77_EXECS}
    @rm -f ${HYPRE_FEI_DRIVER_CXX_EXECS}

#####
# Rules
#####

PURIFY = purify
PURIFY_TO_FILE = purify \
    -log-file=struct_linear_solvers.purify \
    -append-logfile=yes

readijmatrix: readijmatrix.o
    @echo "Building" $@ "... "
    ${CC} -o $@ $@.o ${STRUCT_LFLAGS} -DHYPRE_FEI -DHYPRE_LSI

#####
# Generic rules
#####

.c.o:
    ${CC} -o $@ -c ${CFLAGS} $<

.C.o:
    ${CXX} -o $@ -c -DNOFEI ${CFLAGS} $<

.f.o:
    ${F77} -o $@ -c ${FFLAGS} $<

# rule to compile C files with C++
.c.CXX.o:
```

```
cp -f $< ${@:.o=.C}  
{CXX} -o $@ -c {CFLAGS} ${@:.o=.C}  
rm -f ${@:.o=.C}
```