User's guide to a distributed parallel solver for sparse linear systems obtained from THCM using METIS, MRILU and MPI

A. Meijster (a.meijster@rc.rug.nl)
Center for High Performance Computing and Visualisation
University of Groningen
P.O. Box 11044
9700 CA Groningen

F.W. Wubs (wubs@math.rug.nl)
Institute for Mathematics and Computing Science
University of Groningen
P.O. Box 800
9700 AV Groningen

The user's guide only treats the application of the MPI based version to the Ocean Flow Model THCM. Since in essence we built a parallel shell around MRILU we do not treat the user input for MRILU extensively here. At some places in this document we refer to the technical description in the report "A distributed parallel solver for sparse linear systems obtained from THCM using METIS, MRILU and MPI"

1 Main routines

1.1 For the master processor

There are a few important routines to the user on the master processor:

FindOrdering constructs the permutation vector corresponding to the partitioning. Since in THCM the structure of the matrix does not change, the partitioning needs only to be done for one, i.e. the first, matrix. The routine also sends the array `domain` to the slave processors. The input to this routine, which is part of the class `CSRmatrix` (file `csr.cxx`), is the matrix generated by THCM, the number of planes in the vertical direction, the position of the full row. The definition is
`FindOrdering(int myrank, int nparts, int layers, int fullrow,`

`int *domain, int *blocksz, int *perm);`
A short description of the input parameters:

`myrank` is the number of the processor running the master task. The value of `myrank` is the number of processors minus one.

`nparts` is the number of processors claimed by the user.

`layers` is the number of vertical layers in the simulation.

`fullrow` is the position of the full row in the matrix.

Output parameters:

`domain` gives the start positions of the various domains including that of the separators in the reordered matrix.

`blocksz` gives the sizes of the domains and the number of separators.

`perm` gives the permutation vector.

ReordFilterSend permutes the matrix, the right-hand side and the initial guess using the permutation vector `perm` and distributes it over the processors; it returns a full matrix corresponding to the separators on the master. It is also part of the class `CSRmatrix`. The definition is
`double **ReordFilterSend(int nparts, int *domain, int *blocksz, int *perm, double *x, double *b);`
There are only two parameters not explained before: the initial guess `x` and the right-hand side `b`. Both are in- and output parameters. On output they contain the whole reordered vectors.

MasterFactor Factors the Schur complement (file `mastersolve.cxx`). It also makes a CSR matrix from the output of `ReordFilterSend`. The definition is given by
`void MasterFactor(int myrank, int nparts, int *domain, int nrows, int nsep, double **Rfull, int *Ridx, int *ixLU, int *ixdiLU, int *ixDiag, int *ixPerm);`
The new input parameters are below.

`nrows` is the order of the matrix.

`nsep` is the number of separators.

`Rfull` is the output of `ReordFilterSend`.

The output parameters are given below.

`Ridx` is the location of the descriptor of the CSR matrix corresponding to `Rfull` in the MRILU workspace buffer.

`ixLU` is the location of the descriptor of the CSR matrix containing the LDU factorization of the Schur complement.

`ixdiLU` gives the location of the array containing the positions of the last non-zero off-diagonal element of matrix L in the CSR matrix corresponding to ixLU.

`ixDiag` gives the location of the array containing the inverse of the diagonal of the LDU factorization.

`ixPerm` gives the location of the array containing the permutation array of the factorization.

MasterSolve solves the linear system using FGMRES. The definition is
```
void MasterSolve(int myrank, int nparts, int *domain, int nrows,
int nsep, int Ridx, int ixLU, int ixdiLU, int ixDiag, int ixPerm,
double *x, double *b);
```
All parameters are already described before.

MasterFreePrec frees the memory for the preconditioner in the MRILU workspace buffer. Its definition is:
```
void MasterFreePrec(int ixLU, int ixdiLU, int ixDiag, int ixPerm);
```
All parameters are already described before.

1.2 For the slaves

On the slaves there are a number of counterparts of the routines described for the master.

BroadcastDomain receives the array `domain` from the master (send in `FindOrdering`). Its definition is
```
void BroadcastDomain(int sender, int *domain)
```
The input parameter `sender` is the processor number of the master.

ReceiveMatVec receives those parts of the matrix, right-hand side and initial guess from the master which have to be treated by the calling slave processor. These matrices are send by the master routine `ReordFilterSend`
The definition is:
```
void ReceiveMatVec(int myrank, int masterrank, int nparts, int
*domain, int *Aidx, int *Eidx, int *Fidx, int *xIdx, int *rhsIdx);
```
The new input parameters:

`myrank` is the processor number of the slave.

`masterrank` is the processor number of the master (hence 0).

The output parameters:

`Aidx` is the location in the MRILU workspace buffer of the descriptor of the CSR matrix corresponding to the A_{ii} matrix to be treated on the calling slave processor.

`Eidx` is the location of the descriptor of the CSR matrix corresponding to the A_{ij} matrix to be treated on the calling slave processor.

`Fidx` is the location of the descriptor of the CSR matrix corresponding to the A_{ji} matrix to be treated on the calling slave processor.

`xIdx` is the location of the initial guess (also the solution vector) in the workspace buffer (only the relevant part for this processor).

`rhsIdx` is the location of the right-hand side in the workspace buffer (only the relevant part for this processor).

SlaveFactor computes the factorization on the current slave for the matrix indicated by `Aidx`. Furthermore it computes in a full matrix its contribution to the Schur complement and sends it to the master (file: `slavesolve.cxx`). Its definition is:

```
void SlaveFactor(int myrank, int masterrank, int nparts, int *domain,
int *Aidx, int Eidx, int Fidx, int *ixPrc);
```

The only new (output) parameter is `ixPrc` containing the location of the descriptor for the MRILU incomplete factorization.

SlaveSolve runs FGMRES on the current slave processor. Its definitions is

```
void SlaveSolve(int myrank, int masterrank, int nparts, int *domain,
int Aidx, int Eidx, int Fidx, int ixPrc, int xIdx, int rhsIdx);
```

All parameters have been described above.

SlaveFreePrec frees the space in the workspace buffer occupied by the preconditioner. Its definition is

```
void SlaveFreePrec(int ixPrc);
```

2 Parameter settings

All parameters are set in the file `PARS.f`. The parameters for MRILU, used in the inner iteration, are as before. In this preliminary version, no exact elimination should be used within MRILU, since it is not anticipated enough for that in the partitioning process. Furthermore, one should choose GlobFrac=1.0 in order to force that MRILU starts directly with ILUT part.

For the ILUT factorization on the master and the solution with FGMRES in the outer iteration, the parameters are set at the end of routine `PARS` and they are communicated via common blocks in `slvextr.inc`. For FGMRES have to be set only the maximum number of iterations `MaxNItsF`, the restart value `Mfgrmres`, the reduction factor `RedTolF` and the absolute value `AbsTolF` for the residual. The iteration is stopped if one of these last two values is satisfied.

Another important parameter is the size of the workspace buffer. This is set in file `wsb.common`.

3 Overview of the code

The program is a mixture of C++ and FORTRAN77. All MPI related routines are in C++ using the C MPI library. Users should be aware that problems may occur if also a FORTRAN MPI library is invoked.

Here an overview of the sources:

Makefile The compilation and linking can be done via make using the descriptions in this file.

main.cxx is the routine from where the routine master (in master.cxx) and slave (in slave.cxx) are called depending on the processor number.

master.cxx is the main example routine running on the master in which a matrix is read in, a sample right-hand side is constructed, the partitioning is formed, the factorization of the Schur complement is made, and the system is solved.

mastersolve.cxx contains the routines for the call of the factorization and FGMRES on the master.

fmastersolve.F contains the FORTRAN code used on the master processor, i.e. the factorization of the Schur complement and the master part of FGMRES.

slave.cxx is the main example program for the slaves in which the slave's part of the matrix is received and factored. Furthermore the slave's solution part is invoked here.

slavesolve.cxx contains the main routines for receiving, factoring and solving.

fslavesolve.F contains the FORTRAN part of the factorization and solution process on the slaves.

csr.cxx contains all kinds of matrix operations including IO for matrices in CSR format.

merge.cxx contains binary tree addition routines.

schurmerge.cxx contains send and receive routines for the contribution of the slaves to the Schur complement, and computes the Schur complement.

communicate.cxx contains routines for the communication of vectors, CSR matrices, etc.

wsbmalloc.cxx consists of routines that call MRILU FORTRAN routines that allocate space in the MRILU workspace buffer for vectors, CSR matrices, etc.

PARS.f contains the parameter settings for MRILU, ILUT and FGMRES.

wsb.common contains the common block for the MRILU workspace buffer. Currently the magnitude must be set by hand.

wsreqst.inc defines a few constants needed to use the MRILU workspace buffer.

glbpars.inc contains a common block with a few global parameters of MRILU.

solpars.inc contains a common block with the parameters for the solution part of MRILU.

slvextr.inc contains two common blocks with parameters for the MRILU routine `incldu` and the added FGMRES, respectively.

Of course many of the .cxx files are accompanied by a header file.