SAND REPORT

SAND2004-XXX
Unlimited Release
Printed May 2004

Amesos 1.0 Reference Guide

Marzio Sala

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2004-XXX
Unlimited Release
Printed May 2004

Amesos 1.0 Reference Guide

Marzio Sala
Computational Math & Algorithms
Sandia National Laboratories
MS 1110 P.O. Box 5800
Albuquerque, NM 87185-1110

Abstract

This document describes the main functinalities of the Amesos package, version 1.0. Amesos, available as part of Trilinos 4.0, provides an object-oriented interface to several serial and parallel sparse direct solvers libraries, for the solution of the linear systems of equations

$$AX = B \tag{1}$$

where A is a real sparse, distributed matrix, defined as an Epetra_RowMatrix object, and X and B are defined as Epetra_MultiVector objects.

Amesos provides a common look-and-feel to several directo solvers, insulating the user from each package's details, such as matrix and vector format, and data distribution.

Amesos 1.0 Reference Guide

Contents

Acknowledgments

The authors would like to acknowledge the support of the ASCI and LDRD programs that funded development of Trilinos.

1 Introduction

Aim of the Amesos package is to provide an object-oriented interface to several sparse direct solvers[1]. For each solver, Amesos provides a C++ interface. All the interfaces have the same look-and-feel, and accept matrices defined as Epetra_RowMatrix objects, and vectors defined as Epetra_MultiVector objects. Amesos makes easy for users to switch from one direct solver library from another.

Amesos contains several classes, as reported in table 1. The classes covered in this guide are:

- `Amesos_KLU`: Interface to Amesos's internal solver KLU (in Section 4).
- `Amesos_Umfpack`: Interface to Tim Davis's UMFPACK [4], version 4.3 (in Section 5);
- `Amesos_Superludist`: Interface to Xiaoye S. Li's distributed SuperLU [9] (in Section 6);
- `Amesos_Mumps`: Interface to MUMPS 4.3.1 [1] (in Section 7).

All the Amesos classes are derived from a base class mode, `Amesos_BaseSolver`. This abstract interface provides the basic functionalities for all Amesos solvers, and allows users to choose different direct solvers very easily — by changing an input parameter. See Section 3 for more details.

Once an Amesos object is defined, the direct solution of the linear system simply reads, for all interfaces,

```
AmesosObject.Solve();
```

or, more generally, by

```
AmesosObject.SymbolicFactorization();
AmesosObject.NumericFactorization();
AmesosObject.Solve();
```

This sequence of commands applies to serial, as well as distributed direct solvers libraries. All necessary data redistribution is automatically managed by Amesos.

2 Configuring and Installation Amesos

Amesos is distributed through the Trilinos project, and can be downloaded from the web site

```
http://software.sandia.gov/trilinos/packages/amesos
```

[1]Amesos is an interface to other packages, mainly developed outside the Trilinos framework. In order to use those packages, the user should carefully check copyright and licensing of those third-party codes. Please refer to the web page or the documentation of each particular package for details.

Class			Interface to
Amesos_Klu	serial	general	KLU
Amesos_Umfpack	serial	general	UMFPACK 4.3
Amesos_Superlu	serial	general	SuperLU 3.0
Amesos_Superludist	parallel	general	SuperLU_DIST 2.0
Amesos_Mumps	parallel	SPD, sym, general	MUMPS 4.3.1
Amesos_Scalapack	parallel	general	ScaLAPACK

Table 1. Supported interfaces. "serial" means that the supported direct solver is a serial one. When solving with more than one processor, the linear problem is shipped to processor 0, solved, then the solution is broadcasted in the solution vector X. "parallel" means that a subset or all the processes in the current communicator will be used by the solver. "general" means general unsymmetric matrix, If "sym" (symmetric matrix) or "SPD" (symmetric positive definite), the direct solver library can take advantage of that particular matrix property.

Each of the Amesos classes provides an interface to a third-party direct sparse solver code[2]. In order to configure and compile a given interface, the user must first install the underlying direct sparse solver code. Generally, the BLAS library is required. Some solvers may need CBLACS, LAPACK, BLACS, ScaLAPACK. Amesos requires Epetra and Teuchos (both part of Trilinos).

Amesos is configured and built using the GNU autoconf [6] and automake [7] tools. To configure Amesos from the Trilinos top directory, a possible procedure is as follows. Let `$TRILINOS_HOME` be a shell variable representing the location of the Trilinos source directory, and `%` the shell prompt sign. In order to configure Trilinos with Amesos, for instance on a LINUX machine with MPI, one may do the following:

```
% cd $TRILINOS_HOME
% mkdir LINUX_MPI
% cd LINUX_MPI
% ../configure --with-mpi-compilers \
               --prefix=$TRILINOS_HOME/LINUX_MPI \
               --enable-amesos \
               FLAGS \
               AMESOS_FLAGS
% make
% make install
```

Here, `FLAGS` represents the set of configure options for other Trilinos packages, and `AMESOS_FLAGS` the configure options specific to Amesos. The configure options required to enable a specific interface are reported in each third-party package's section. A complete list of them can be obtained by typing

[2]Exception to this rule is KLU, which is distributed within Amesos.

```
% $TRILINOS_HOME/packages/amesos/configure --help
```

3 Amesos_BaseSolver: A Generic Interface to Direct Solvers

All Amesos objects are derived from the pure virtual class Amesos_BaseSolver, and can be constructed using the function class `Amesos`. `Amesos` allows a code to delay the decision about which concrete class to use to implement the Amesos_BaseSolver interface. The main goal of this class is to allow the user to select any supported (and enabled at configuration time) direct solver, simply changing an input parameter. Another remarkable advantage of Amesos_BaseSolver is that users does not have to include the header files of the third-party libraries in their code[3].

An example of use of this class is as follows. First, the following header file must be included:

```
#include "Amesos.h"
```

Then, let `A` be an Epetra_RowMatrix object (for instance, and Epetra_CrsMatrix)[4].

We need to define a linear problem,

```
Epetra_LinearProblem * Amesos_LinearProblem =
                       new Epetra_LinearProblem;
Amesos_LinearProblem->SetOperator( A ) ;
```

Now, let `Choice` be a char array variable, with one of the values reported in the first column of table 1. We can construct an `Amesos_BaseSolver` object as follows:

```
Amesos_BaseSolver * A_Base;
Amesos Amesos_Factory;

A_Base = Amesos_Factory.Create(Choice, *Amesos_LinearProblem);
assert(A_Base!=0);
```

If the class requested by `Choice` is not available (because is not installed, or `Choice` is mispelled), `Create()` returns 0.

Symbolic and numeric factorizations are computed using methods

```
A_Base->SymbolicFactorization();
A_Base->NumericFactorization();
```

The numeric factorization phase will check whether a symbolic factorization exists or not. If not, method `SymbolicFactorization()` is invoked. Solution is computed, using

```
A_Base->Solve();
```

[3]Using Amesos_BaseSolver, third-party libraries header files are required in the compilation of Amesos only.

[4]Some solvers can take advantage if the matrix is an Epetra_CrsMatrix or an Epetra_VbrMatrix; this is reported in Table 3.

Architecture	Communicator	KLU	UMFPACK	SuperLU_DIST 2.0	MUMPS 4.3.1	ScaLAPACK
LINUX	SERIAL	•	•	–	–	–
LINUX, GNU	LAM/MPI	•	•	•	–	•
LINUX, Intel	MPICH	•	•	–	•	•
SGI 64	MPI	•	•	•	•	–
DEC/Alpha	MPI	•	•	–	–	–
MAC OS X/G4	MPICH	•	–	–	–	–
Sandia Cplant	MPI	•	•	•	•	–
ASCI Red	MPI	•	•	•	–	–

Table 2. Supported architectures for various interfaces. '•' means that the interface has been successfully compiled, '–' means that it has not been tested.

Class	Epetra_RowMatrix	Epetra_CrsMatrix	Epetra_VrbMatrix
Amesos_Klu	•	•	–
Amesos_Umfpack	•	•	–
Amesos_Superlu	•	•	–
Amesos_Superludist	•	•	–
Amesos_Mumps	•	•	•
Amesos_Scalapack	•	•	–

Table 3. Supported matrix formats. '•' means that the interface can take advantage of the given matrix format, '–' means that it doesn't.

The solution phase will check whether a numeric factorization exists or not. If not, method `NumericFactorization()` is called.

Users must provide the nonzero structure of the matrix for the symbolic phase, and the actual nonzero values for the numeric factorization. Right-hand side and solution vectors must be set before the solution phase, for instance using

```
Amesos_LinearProblem->SetLHS(x);
Amesos_LinearProblem->SetRHS(b);
```

Specific parameters can be set using a Teuchos parameters list, whose definition requires the input file `Teuchos_ParameterList.hpp`. For a detailed description, we refer to the Teuchos documentation. We report the most important methods of this class in Table 4.

`set(Name,Value)`	Add entry `Name` with value and type specified by `Value`. Any C++ type (like int, double, a pointer, etc.) is valid.
`get(Name,DefValue)`	Get value (whose type is automatically specified by `DefValue`). If not present, return `DefValue`.
`subList(Name)`	Get a reference to sublist `List`. If not present, create the sublist.

Table 4. Some methods of Teuchos::ParameterList class.

Here, we simply recall that the parameters list can be created as

```
Teuchos::ParameterList AmesosList;
```

and parameters can be set as

```
AmesosList.set(ParameterName,ParameterValue);
```

Here, `ParameterName` is a string containing the parameter name, and `ParameterValue` is any valid C++ object that specifies the parameter value (for instance, an integer, a pointer to an array or to an object).

Amesos parameters can (possibly) affect all the solvers, or being specific to a given interface. In this latter case, they are defined in a sublist.

We now list all the parameters that may affect all the Amesos solvers. To know whether a specific interface supports a given parameter, we refer to table 5.

`UseTranspose`	If `false`, solve linear system (1). Otherwise, solve the linear system with the transpose matrix A^T.
`MatrixType`	Set it to `SPD` if the matrix is symmetric positive definite, to `symmetric` if symmetric, and to `general` is the matrix is general unsymmetric. At this stage of development, only the MUMPS interface can take advantage of `SPD` and `symmetric`.
`Threshold`	In the conversion from Epetra_RowMatrix to a package's format, do not include elements whose absolute value is below the specified threshold.
`AddZeroToDiag`	If true, in the conversion from Epetra_RowMatrix to a package's format, a zero element will be added to the diagonal if not present.
`PrintTiming`	Print some timing information when the Amesos object is destoyed.
`PrintStatus`	Print some information about the linear system and the solver when the Amesos object is destoyed.
`ComputeVectorNorms`	After solution, compute the 2-norm of each vector in the Epetra_MultiVector B and X.
`ComputeTrueResidual`	After solution, compute the real residual $\|B - AX\|_2$ for all vectors in Epetra_MultiVector.

`MaxProcs`	If positive, the linear system matrix will be distributed on the specified number of processes only (or the all the processes in the MPI communicator if the specified number is greater). If `MaxProcs=-1`, Amesos will estimate using internal heuristics the optimal number of processes that can efficiently solve the linear system. If `MaxProcs=-2`, Amesos will use the square root of the number of processes. If `MaxProcs=-3`, all processes in the communicator will be used. This option may require the conversion of a C++ MPI communicator to a FORTRAN MPI communicator. If this is not supported, the specified value of `MaxProcs` will be ignored, and all the processes in `MPI_COMM_WORLD` will be used.
`MaxProcsMatrix`	The linear system matrix will be distributed over the specified number of processes. This number must be less or equal to `MaxProcs`. See `Maxprocs`. If `MaxProcsMatrix=-4`, then the value of `MaxProcsMatrix` equals that of `MaxProcs`.
`OutpuLevel`	If `0`, no output is printed on the standard output. If `1`, output is reported as specified by other parameters. If `2`, all output is printed (this is equivalent to `PrintTiming == true`, `PrintStatus == true`, `ComputeVectorNorms == true`, `ComputeTrueResidual == true`).
`DebugLevel`	If `1`, some debugging information are printed on the standard output.

Solver-specific parameters are reported in each package's subsection. The general procedure is to create a sublist with a given name (for instance, the sublist for MUMPS is "mumps"), then set all the solver's specific parameters in this sublist. An example is as follows:

```
int ictnl[40];
// defines here the entries of ictnl
Teuchos::ParameterList & AmesosMumpsList =
  AmesosList.sublist("mumps");
AmesosMumpsList.set("ICTNL", ictnl);
```

Parameters and sublists not recognized are simply ignored. Recall that spaces are important, and that parameters list is case sensitive!

option	type	default value	KLU	UMFPACK	SuperLU_DIST	MUMPS	ScaLAPACK
UseTranspose	bool	false	•	•	–	•	•
MatrixType	string	general	–	–	–	•	–
Threshold	double	0.0	–	–	–	•	–
AddZeroToDiag	bool	false	–	–	•	•	–
PrintTiming	bool	false	•	•	–	•	•
PrintStatus	bool	false	•	•	•	•	•
MaxProcs	int	-1	–	–	•	•	•
MaxProcsMatrix	int	-4	–	–	–	•	–
ComputeVectorNorms	bool	false	•	•	•	•	•
ComputeTrueResidual	bool	false	•	•	•	•	•
OutputLevel	int	1	•	•	•	•	•
DebugLevel	int	0	•	•	•	•	•

Table 5. Supported options. '•' means that the interface supports the options, '–' means that it doesn't.

4 Amesos Interface to KLU

KLU is a serial, unblocked code ideal for getting started. Particular classes of matrices, such as circuit matrices, may perform well with KLU.

KLU is Tim Davis' implementation of Gilbert-Peierl's left-looking sparse partial pivoting algorithm, with Eisenstat and Liu's symmetric pruning. It doesn't exploit dense matrix kernels, but it is the only sparse LU factorization algorithm known to be asymptotically optimal, in the sense that it takes time proportional to the number of floating-point operations. It is the precursor to SuperLU, thus the name ("Clark Kent LU"). For very sparse matrices that do not suffer much fill-in (such as most circuit matrices when permuted properly) dense matrix kernels do not help, and the asymptotic run-time is of practical importance.

In order to use KLU, Amesos must be configured with the options

```
--enable-amesos-klu
```

The KLU souces are distributed with the Amesos package.

5 Amesos Interface to UMFPACK 4.3

UMFPACK is a C package copyrighted by Timothy A. Davis. More information can be obtained at the web page

```
http://www.cise.ufl.edu/research/sparse/umfpack
```

In order to use UMFPACK, Amesos must be configured with the options

```
--enable-amesos-umfpack
--with-amesos-umfpacklib=<UMFPACK library>
--with-amesos-umfpackincdir=<UMFPACK include files>
--with-amesos-umfpackamdlib=<AMD library>
--with-amesos-umfpackamdincdir=<AMD include files>
```

UMFPACK is a serial solver. Amesos will take care of moving matrix, solution and right-hand side to processor 0 (using Epetra_Import objects), solve the linear system on processor 0, then broadcast the solution as required.

6 Amesos Interface to SuperLU_DIST 2.0

SuperLU_DIST, written by Xiaoye S. Li, is a parallel extension to the serial SuperLU library. SuperLU_DIST is written in ANSI C, using MPI for communication, and it is targeted for the distributed memory parallel machines. It is copyrighted by The Regents of the University of California, through Lawrence Berkeley National Laboratory. We refer to the web site

```
http://www.nersc.gov/~xiaoye/SuperLU
```

and to the SuperLU_DIST manual [5] for more information.

SuperLU_DIST includes routines to handle both real and complex matrices in double precision. However, as Amesos is currently based on the Epetra package (that does not handle complex matrices), only double precision matrices can be considered.

Amesos_Superludist can solve the linear system on a subset of the processes, as specified in the parameters list. This is done by creating a new process group derived from the MPI group of the Epetra_Comm object, with function `superlu_gridinit()`.

In order to interface with SuperLU_DIST 2.0, Amesos must be configured with the options

```
--enable-amesos-superludist
--with-amesos-superludistlib=<SuperLU_DIST library>
--with-amesos-superludistincdir=<SuperLU_DIST include files>
```

The SuperLU_DIST constructor will look for a sublist, called `Superludist`. The following parameters reflect the behavior of SuperLU_DIST options argument, as specified in the SuperLU_DIST manual [5, pages 55–56]. The user is referred to this manual for a detailed explanation of the reported parameters. Default values are as reported in the SuperLU_DIST manual.

`Fact`	(`string`) Specifies whether or not the factored form of the matrix A is supplied onentry and, if not, how the matrix will be factored. It can be: `DOFACT`, `SamePattern`, `SamePattern_SameRowPerm`, `FACTORED`. Default: `SamePattern_SameRowPerm`.
`Equil`	(`bool`) Specifies whether to equilibrate the system of not. Default: `true`.
`ColPerm`	(`string`) Specifies the column ordering strategy. It can be: `NATURAL`, `MMD_AT_PLUS_A`, `MMD_ATA`, `COLAMD`, `MY_PERMC`. Default: `MMD_AT_PLUS_A`.
`perm_c`	(`int *`) Specifies the ordering to use when `ColPerm = MY_PERMC`.
`RowPerm`	(`string`) Specifies the row ordering strategy. It can be: `NATURAL`, `LargeDiag`, `MY_PERMR`. Default: `LargeDiag`.
`perm_r`	(`int *`) Specifies the ordering to use when `RowPerm = MY_PERMR`.
`ReplaceTinyPivot`	(`bool`) Specifies whether to replace the tiny diagonals with $\varepsilon\|A\|$ during LU factorization. Default: `true`.

`IterRefine`	(`string`) Specifies how to perform iterative refinement. It can be: `NO, DOUBLE, EXTRA`. Default: `DOUBLE`.

7 Amesos Interface to MUMPS 4.3.1

MUMPS ("MUltifrontal Massively Parallel Solver") is a parallel direct solver, written in FORTRAN 90 with C interface, copyrighted by P. R. Amestoy, I. S. Duff, J. Koster, J.-Y. L'Excellent. Up-to-date copies of the MUMPS package can be obtained from the Web page

`http://www.enseeiht.fr/apo/MUMPS/`

Here, for the sake of completeness, we briefly present a broad view of the MUMPS package, so that the reader can better understand the Amesos_Mumps interface. For details about the algorithms and the implementation, as well as of the input parameters, we refer to [2]

MUMPS can solve the original system (1), as well as the transposed system, given an assembled or elemental matrix. Note that only the assembled format is supported by Amesos_Mumps. Mumps offers, among other features, error analysis, iterative refinement, scaling of the original matrix, Schur complement with respect to a prescribed subset of rows. Reordering techniques can take advantage of PORD (distributed within MUMPS), or METIS [8][5].

Amesos_Mumps is based on the distributed double-precision version of MUMPS (which requires MPI, BLAS, BLACS and ScaLAPACK [3]).

In order to interface with MUMPS 4.3.1, Amesos must be configured with the options[6]

```
--enable-amesos-mumps
--with-amesos-mumpslib=<MUMPS library>
--with-amesos-mumpsincdir=<MUMPS include files>
```

The MUMPS constructor will look for a sublist, called `mumps`. The user can set all the MUMPS's parameters, by sticking pointers to the integer array ICNTL and the double array CNTL to the parameters list, or by using the functions reported at the end of this section.

`ICTNL`	(`int[40]`) Pointer to an integer array, containing the integer parameters (see [2, pages 13–17]).
`CTNL`	(`double[5]`) Pointer to an double array, containing the double parameters (see [2, page 17]).
`PermIn`	(`int *`) Use integer vectors of size NumGlobalElements (global dimension of the matrix) as given ordering. `PermIn` must be defined on the host only, and allocated by the user, if the user sets ICNTL(7) = 1.

[5] At this time, METIS ordering is not supported by Amesos_Mumps.

[6] The MUMPS interface can take be used on a subset of the processes. To that aim, it must be possible to convert from a C++ MPI communicator to a FORTRAN MPI communicator. Such a conversion is not always possible. In you experience compilation problems with `Amesos_Mumps`, you can try the option `--disable-amesos-mumps_mpi_c2f`.

`Maxis`	(`int`) Set Maxis value.
`Maxs`	(`int`) Set Maxis value.
`ColPrecScaling`	(`double *`) Use double precision vectors of size NumGlobalElements (global dimension of the matrix) as scaling for columns and rows. The double vector must be defined on the host only, and allocated by the user, if the user sets ICNTL(8) = -1.
`RowPrecScaling`	(`double *`) Use double precision vectors of size NumGlobalElements (global dimension of the matrix) as scaling for columns and rows. The double vector must be defined on the host only, and allocated by the user, if the user sets ICNTL(8) = -1.

Other functions are avaiable to retrive the output values. The following Amesos_Mumps methods are *not* supported by the Amesos_BaseSolver class; hence, the user must create an Amesos_Mumps object in order to take advantage of them.

```
double * GetRINFO()
```

Gets the pointer to the RINFO array (defined on all processes).

```
int * GetINFO()
```

Gets the pointer to the INFO array (defined on all processes).

```
double * GetRINFOG()
```

Gets the pointer to the RINFOG array (defined on host only).

```
int * GetINFOG()
```

Gets the pointer to the INFOG array (defined on host only).

A functionality that is peculiar to MUMPS, is the ability to return the Schur complement matrix, with respect to a specified set of nodes.

```
int ComputeSchurComplement(bool flag,
                           int NumSchurComplementRows,
                           int * SchurComplementRows);
```

This method computes (if `flag` is true) the Schur complement with respect to the set of indices included in the integer array `SchurComplementRows`, of size `NumSchurComplementRows`. This is a *global* Schur complement, and it is formed (as a dense matrix) on processor 0 only.

```
Epetra_CrsMatrix * GetCrsSchurComplement();
```

This method returns the Schur complement in an Epetra_CrsMatrix, on host only. No checks are performed to see whether this action is legal or not (that is, if the call comes after the solver has been invocated). The returned Epetra_CrsMatrix must be freed by the user.

```
Epetra_SerialDenseMatrix * GetDenseSchurComplement();
```

This method returns the Schur complement as a Epetra_SerialDenseMatrix (on host only).

As an example, the following frament of code shows how to use MUMPS to obtain the Schur complement matrix with respect to a given subsets of nodes. First, we need to create an parameter list, and an Amesos_Mumps object.

```
Teuchos:::ParameterList params;
Amesos_Mumps * Solver;
Solver = new Amesos_Mumps(*Problem,params);
```

Then, we define the set of nodes that will constitute the Schur complement matrix. This must be defined on processor 0 only. For instance, one may have:

```
int NumSchurComplementRows = 0;
int * SchurComplementRows = NULL;
if( Comm.MyPID() == 0 ) {
  NumSchurComplementRows = 4;
  SchurComplementRows = new int[NumSchurComplementRows];
  SchurComplementRows[0] = 0;
  SchurComplementRows[1] = 1;
  SchurComplementRows[2] = 2;
  SchurComplementRows[3] = 3;
}
```

Now, we can ask for the Schur complement using

```
Solver->ComputeSchurComplement(true, NumSchurComplementRows,
                                     SchurComplementRows);
```

The Schur complement matrix can be obtain after the solver phase:

```
Solver->Solve();
Epetra_CrsMatrix * SC;
SC = Solver->GetCrsSchurComplement();
Epetra_SerialDenseMatrix * SC_Dense;
SC_Dense = Solver->GetDenseSchurComplement();
```

8 Example Code

In this section we report a complete code that can be used to compare the performances of various direct solvers. The code is based on the Amesos_BaseSolver interface described in Section 3.

First, we need to include the appropriate headers. The variable `HAVE_CONFIG_H` must have been defined – in the file, or at compilation time.

```
#ifndef HAVE_CONFIG_H
#define HAVE_CONFIG_H
#endif
#include "Epetra_ConfigDefs.h"
#ifdef HAVE_MPI
#include "mpi.h"
#include "Epetra_MpiComm.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "Epetra_Vector.h"
#include "Epetra_Time.h"
#include "Amesos_ConfigDefs.h"
#include "Teuchos_ParameterList.hpp"
#include "Amesos.h"
#include "Trilinos_Util_CrsMatrixGallery.h"
```

The code can be run with or without MPI; however, the supported versions of MUMPS and SuperLU_DIST requires MPI.

```
int main(int argc, char *argv[]) {

#ifdef HAVE_MPI
  MPI_Init(&argc, &argv);
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
#else
  Epetra_SerialComm Comm;
#endif
```

Here we use the class Trilinos_Util::CrsMatrixGallery to read and Harwell/Boeing matrix from file, whose name is hardwired in the code for simplicity (see [10, Chap 5]). The name can be read from the input line using class Trilinos_Util_CommandLineParser (see [10]), or the Teuchos package.

```
  Trilinos_Util::CrsMatrixGallery G("hb", Comm);
  G.Set("matrix_name","662_bus.rsa");
```

Class Trilinos_Util::CrsMatrixGallery automatically defines an Epetra_LinearProblem, that can be obtained as follows:

```
  Epetra_LinearProblem * Problem = G.GetLinearProblem();
```

Now, we define a Teuchos parameters list, and set the maximum number of processes by

```
  Teuchos::ParameterList AmesosList;
  AmesosList.set("MaxProcs",8);
```

At this point, we can create an Amesos_BaseSolver object, depending on the run-time choice (here hardcoded for the sake of simplicity as a string):

```
  Amesos_BaseSolver * Solver;
  Amesos              Amesos_Factory;
  // change this as required
  char SolutionLib[] = "umfpack";

  Solver = Amesos_Factory.Create(SolutionLib, *Problem, params );
  if( Solver == 0 ) cerr << "library not available" << endl;
```

We can solve the linear problem:

```
  Epetra_Time Time(Comm);
  Solver->SymbolicFactorization();
  double TimeForSymbolicFactorization = Time.ElapsedTime();

  Time.ResetStartTime();
  Solver->NumericFactorization();
  double TimeForNumericFactorization = Time.ElapsedTime();

  Time.ResetStartTime();
  Solver->Solve();
  double TimeForSolve = Time.ElapsedTime();
```

and finally delete the Amesos_BaseSolver object, and exit the code.

```
  delete Solver;

#ifdef HAVE_MPI
  MPI_Finalize();
#endif

  return( EXIT_SUCCESS );

}
```

References

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. MUMPS home page. http://www.enseeiht.fr/lima/apo/MUMPS, 2003.

[2] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, and J. Koster. *MUltifrontal Massively Parallel Solver (MUMPS Versions 4.3.1) Users' Guide*, 2003.

[3] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Jemmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM Pub., 1997.

[4] T .A. Davis. UMFPACK home page. http://www.cise.ufl.edu/research/sparse/umfpack, 2003.

[5] J. W. Demmel, J. R. Gilbert, and X. S. Li. *SuperLU Users' Guide*, 2003.

[6] Free Software Foundation. Autoconf Home Page. http://www.gnu.org/software/autoconf.

[7] Free Software Foundation. Automake Home Page. http://www.gnu.org/software/automake.

[8] G. Karypis and V. Kumar. METIS: Unstructured graph partitining and sparse matrix ordering sy stem. Technical report, University of Minnesota, Department of Computer Science, 1998.

[9] X. S. Li and J. W. Demmel. SuperLU home page. http://crd.lbl.gov/ xiaoye/SuperLU/, 2003.

[10] M. Sala, M. A. Heroux, and D. Day. *Trilinos Tutorial*, 4.0 edition, 2004.