SANDIA REPORT

SAND2004-4820
Unlimited Release
Printed September 2004

Amesos 2.0 Reference Guide

Marzio Sala

Sandia National Laboratories

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

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

SAND2004-4820
Unlimited Release
Printed September 2004

Amesos 2.0 Reference Guide

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

Abstract

This document describes the main functionalities of the AMESOS package, version 2.0. AMESOS, a Trilinos package, 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. Supported libraries include: LAPACK, KLU, UMFPACK, SuperLU, SuperLU_DIST, MUMPS, DSCPACK.

AMESOS provides a common look-and-feel, insulating the user from each package's details, such as matrix and vector formats, and data distribution.

Acknowledgments

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

Amesos 2.0 Reference Guide

Contents

1 Introduction

Aim of the AMESOS package is to provide an object-oriented interface to several sparse direct solvers[1]. AMESOS is developed by (in alphabetical order) T. Davis, M. Heroux, R. Hoekstra, M. Sala, K. Stanley.

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[2], 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 2. The classes covered in this guide are:

- `Amesos_LAPACK`: Interface to LAPACK's dense solvers (in Section 5). This class is the only interface to dense solver in AMESOS;
- `Amesos_KLU`: Interface to AMESOS's internal solver KLU (in Section 6);
- `Amesos_Umfpack`: Interface to Tim Davis's UMFPACK [4], version 4.3 (in Section 7);
- `Amesos_Superlu`: Interface to Xiaoye S. Li's serial SuperLU [9] (in Section 8);
- `Amesos_Superludist`: Interface to Xiaoye S. Li's distributed SuperLU [9] (in Section 9);
- `Amesos_Mumps`: Interface to MUMPS 4.3.1 [1] (in Section 10).
- `Amesos_Dscpack`: Interface to DSCPACK [10] (in Section 11).

A simple fragment of code using AMESOS can read as follows. First, we need to include the header files for AMESOS:

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

Note that these header files will *not* include the header files for the supported libraries (which are of course needed to compile the AMESOS library itself). Now, let define the linear system matrix, the vector that will contain the solution, and the right-hand side as:

```
Epetra_LinearProblem Problem;
Epetra_RowMatrix* A;            // linear system matrix
Epetra_MultiVector* LHS;        // vector that will contain the solution
Epetra_MultiVector* RHS;        // right-hand side
```

[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.

[2]Most interfaces take advantage of linear system matrices that can be casted to `Epetra_CrsMatrix` or `Epetra_VbrMatrix`.

All AMESOS object (derived from pure virtual class `Amesos_BaseSolver`[3]) can be created using the function class `Amesos`[4], as follows:

```
Amesos_BaseSolver * Solver;
Amesos Factory;
char* SolverType = "Amesos_Klu";
Solver = Factory.Create(SolverType, Problem);
```

The complete list of parameters recognized by `Create()` is reported in Table 2.

Remark 1. *It is important to note that all available solvers can be selected simply by changing an input parameter. Some solvers are serial, other parallel; generally, each solver has its own matrix format. However, the user will still have the* same *interface to all of them.*

The `Factory` object will create an `Amesos_Klu` object (if AMESOS has been configure to support this solver). `Factory.Create()` returns 0 if the requested solver is not available. Parameter names are case-sensitive; misspelled parameters will not be recognized. Method `Factory.Query()` can be used to query the factory about the availability of a given solver:

```
char* SolverType = "Amesos_Klu";
bool IsAvailable = Factory.Query(SolverType);
```

Parameters for all AMESOS solvers are specified using a Teuchos[5] 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 1. The user may decide to proceed without calling `SetParameters()`.

`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 1. Some methods of Teuchos::ParameterList class.

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

```
Teuchos::ParameterList List;
```

and parameters can be set as

[3]A pure virtual class is a class that defines interfaces only, and contains no executable code. Pure virtual classes cannot be instantiated; however, it is possible to declare and use pointers and references to a pure virtual class, as normally done with class `Amesos_BaseSolver`.

[4]A function class is a class that contains methods, but no private data. Function classes can be used, for example, to create objects.

[5]AMESOScannot be compiled without the support for TEUCHOS.

```
List.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). The list of parameters that affect all AMESOS solvers are reported in Section 4, while parameters that are specific to a given solver (if any) are reported in the Section of this document dedicated to that solver.

After setting in `Problem` the pointer to the linear system matrix, we can perform the symbolic factorization of the linear system matrix:

```
Solver->SetOperator(A);
AMESOS_CHK_ERR(Solver->SymbolicFactorization());
```

This phase does not require the numerical values of `A`, that can therefore be changed after the call to `SymbolicFactorization()`. However, the nonzero pattern of `A` *cannot* be changed. `AMESOS_CHK_ERR` is a macro (defined in `Amesos_ConfigDefs.h`) that checks the return code. If this return code is not zero, the macro prints out an error message, and returns.

The numeric factorization is performed by

```
AMESOS_CHK_ERR(Solver->NumericFactorization());
```

The values of `RHS` must be set before solving the linear system, which simply reads

```
Problem.SetLHS(LHS);
Problem.SetRHS(RHS);
AMESOS_CHK_ERR(Solver->Solve());
```

Should users need to re-factorize the matrix, they must call `NumericFactorization()`. If the structure of the matrix is changed, they must call `SymbolicFactorization()`. However, it is supposed that the linear system matrix and the solution and right-hand side vectors are still defined with the same `Epetra_Map`.

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
```

Each of the AMESOS classes provides an interface to a third-party direct sparse solver code[6]. 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 also 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`

[6]Exception to this rule is KLU, which is distributed within AMESOS.

Class	Communicator	Matrix type	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_Dscpack`	parallel	symmetric	DSCPACK 1.0

Table 2. Supported interfaces. "serial" means that the supported direct solver is serial. When solving with more than one processor, the linear problem is gathered to process 0, here solved, then the solution is broadcasted to the distributed solution vector. "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.

be a shell variable representing the location of the Trilinos source directory, and `%` the shell prompt sign. Let us suppose that we want to confi gure AMESOS on a LINUX machine with MPI, with support for KLU and UMFPACK. Header fi les for UMFPACK are located in directory `/usr/local/umfpack/include`, while the library, called `libumfpack.a` is located in `/usr/local/umfpack/lib`. The confi gure like will look like:

```
% cd $TRILINOS_HOME
% mkdir LINUX_MPI
% cd LINUX_MPI
% ../configure \
  --with-mpi-compilers \
  --prefix=$TRILINOS_HOME/LINUX_MPI \
  --enable-amesos \
  --enable-amesos-klu \
  --enable-amesos-umfpack \
  --with-incdirs="-I/usr/local/umfpack/include" \
  --with-ldflags="-L/usr/local/umfpack/lib" \
  --with-libs="-lumfpack"
% make
% make install
```

Other flags may be required depending on the location of MPI, BLAS and LAPACK. Supported architectures are reported in Table 3.

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

Table 3. Supported architectures for various interfaces. '●' means that the interface has been successfully compiled, '–' means that it has not been tested.

3 Supported Matrix Formats

Table 4 reports the supported matrix types for all the AMESOS classes. In the table, “Transp” means that AMESOS can solve both the linear system with the linear system matrix and with its transpose. ‘•’ means that the interface can take advantage of the given matrix format, ‘–’ means that it doesn’t.

Class	Transp	Epetra_RowMatrix	Epetra_CrsMatrix	Epetra_VbrMatrix
Amesos_Lapack	yes	•	•	–
Amesos_Klu	yes	•	•	–
Amesos_Umfpack	yes	•	•	–
Amesos_Superlu	no	•	•	–
Amesos_Superludist	no	•	•	–
Amesos_Mumps	yes	•	–	–
Amesos_Dscpack	yes	•	–	–

Table 4. Supported matrix formats. “Transp” means that AMESOS can solve both the linear system with the linear system matrix and with its transpose. ‘•’ means that the interface can take advantage of the given matrix format, ‘–’ means that it doesn’t.

4 Parameters for All AMESOS Solvers

We now list all the parameters that may affect all the AMESOS solvers. To know whether a specifi c 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 defi nite, 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 specifi ed 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 destroyed.
`PrintStatus`	Print some information about the linear system and the solver when the AMESOS object is destroyed.
`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.
`OutputLevel`	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`).
`Refactorize`	"Refactorization" of a matrix refers to the use of a prior symbolic and numeric factorization (including row and column ordering), to factorize a subsequent matrix using the same pivot ordering. This can be significantly faster, but the numerical quality of the factorization may suffer. If `true`, then attempt to re-use the existing symbolic and numeric factorization, to factorize a new matrix using the identical pivot ordering (both row and column ordering) as a prior pivot-capable factorization.

`RcondThreshold`	After a refactorization, an estimate of the reciprocal of the condition number is computed. If this estimate is too small (less than `RcondThreshold`), then the pivot-less factorization is aborted, and the matrix is factorized again with normal numerical pivoting.
`ScaleMethod`	Most methods can scale the input matrix prior to factorization. This typically improves the quality of the factorization and reduces fi ll-in as well. Setting this parameter to zero turns off scaling. A value of 1 selects the method's default scaling method (which may in fact be not to scale at all). A value of 2 means to scale the matrix using the fi rst non-default method the package has, 3 means to use its 2nd alternative method, and so on.

Solver-specifi c 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 specifi c 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!

5 AMESOS Interface to LAPACK

In some cases, the linear system matrix can be of relatively small size, or it can be quite dense, or both. If this happens, it may be convenient to convert the sparse matrix to dense format, then use LAPACK routines.

In order to use LAPACK, AMESOS must be confi gured with the options

```
--enable-amesos-lapack
```

Header fi les and library are automatically located by `configure`.

LAPACK is a (suite of) serial solver(s). AMESOS will gather all matrix rows on processor zero before the symbolic factorization, and all matrix values before the numeric factorization. On process 0, the matrix will be converted to dense storage, using `Epetra_SerialDenseMatrix` objects. A call to `Solve()` requires a gather of the right-hand side on process 0, the local solution of the linear system, and fi nally a scatter operation, to redistribute as necessary the solution vector.

No specifi c parameters are available for this class.

option	type	default value	KLU	UMFPACK	SuperLU_DIST	MUMPS	LAPACK	DSCPACK
`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`	●	●	●	●	●	●
`Refactorize`	`bool`	`false`	●	–	–	–	–	–
`RcondThreshold`	`double`	10^{-12}	●	–	–	–	–	–
`ScaleMethod`	`int`	1	●	–	–	–	–	–

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

6 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 fi ll-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 confi gured with the options

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

The KLU sources are distributed with the AMESOS package. We strongly encourage to confi gure AMESOS with KLU support. KLU is the only interface that is turned on by default.

No specifi c parameters are available for this class.

KLU is a serial solver. AMESOS will gather all matrix rows on processor zero before the symbolic factorization, and all matrix values before the numeric factorization. A call to `Solve()` requires a gather of the right-hand side on process 0, the local solution of the linear system, and fi nally a scatter operation, to redistribute as necessary the solution vector.

7 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 confi gured with the options

```
--enable-amesos-umfpack
```

Location of the header fi les should be specifi ed using `--with-incdirs`, location of the library with `--with-ldflags`, and the library to be linked by `--with-libs`. See Section 2 for an example.

No specifi c parameters are available for this class.

UMFPACK is a serial solver. AMESOS will gather all matrix rows on processor zero before the symbolic factorization, and all matrix values before the numeric factorization. A call to `Solve()` requires a gather of the right-hand side on process 0, the local solution of the linear system, and fi nally a scatter operation, to redistribute as necessary the solution vector.

8 AMESOS Interface to SuperLU 3.0

SuperLU, written by Xiaoye S. Li, is a serial solver. SuperLU is written in ANSI C. 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 manual [5] for more information.

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

```
--enable-amesos-superlu
```

Location of the header files should be specified using `--with-incdirs`, location of the library with `--with-ldflags`, and the library to be linked by `--with-libs`. See Section 2 for an example.

No specific parameters are available for this class.

SuperLU 3.0 is a serial solver. AMESOS will gather all matrix rows on processor zero before the symbolic factorization, and all matrix values before the numeric factorization. A call to `Solve()` requires a gather of the right-hand side on process 0, the local solution of the linear system, and finally a scatter operation, to redistribute as necessary the solution vector.

9 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. 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
```

Location of the header files should be specified using `--with-incdirs`, location of the library with `--with-ldflags`, and the library to be linked by `--with-libs`. See Section 2 for an example.

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 on entry 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`.

10 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/`

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][7]. For details about the

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

algorithms and the implementation, as well as of the input parameters, we refer to [2]

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[8]

```
--enable-amesos-mumps
```

Location of the header files should be specified using `--with-incdirs`, location of the library with `--with-ldflags`, and the library to be linked by `--with-libs`. See Section 2 for an example.

It is also possible to configure with support for the single precision version of MUMPS, using option

```
--enable-amesos-smumps
```

which enables the AMESOS interface support for the single-precision version of MUMPS. This is intended to be used when the precision of the solution is not of primary importance, for example, is AMESOS is used to solve the coarse problem in multilevel preconditioners, like ML [11]. In this case, users may decide to use single-precision solves of the coarse problem to save memory and computational time. As AMESOS is based on the Epetra_LinearProblem class (defined for double precision only), this interface still requires double-precision matrix and vectors. After the solver phase, the single precision vector is copied into the double-precision solution vector of the given Epetra_LinearProblem. If the single precision interface is enabled, this automatically disables the double-precision one.

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.
`Maxis`	(`int`) Set Maxis value.
`Maxs`	(`int`) Set Maxis value.

[8]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`.

`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 available to check 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 invoked). 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 fragment 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();
```

11 AMESOS Interface to DSCPACK

DSCPACK can be used to solve symmetric sparse linear systems of equations. DSCPACK provides a variety of sparsity preserving (fill-reducing) ordering and computes either an LL^T (Cholesky) or LDL^T factorization of the linear system matrix. This solver is written in C, and it uses MPI for inter-processor communication, and the BLAS library for improved chace-performances. The implementation is based on the idea of partitioning the sparse matrix into domains and separators.

We refer to the web site

```
http://www.cse.psu.edu/˜ragavan/dscpack
```

and to the DSCPACK manual [10] for more information.

In order to use DSCPACK, AMESOS must be configured with the options

```
--enable-amesos-dscpack
```

Location of the header files should be specified using `--with-incdirs`, location of the library with `--with-ldflags`, and the library to be linked by `--with-libs`. See Section 2 for an example.

DSCPACK solves the linear system using a number of processors that is a power of 2. If necessary, we linear system matrix will be automatically redistributed on the highest number of processors (either all the processors, or the number specified in `MaxProcs`) that is a power of 2.

No specific parameters are available for this class.

12 Preconditioners Based on AMESOS

AMESOS is used in other TRILINOS packages. In particular, IFPACK can take advantage of AMESOS to define additive overlapping domain decomposition preconditioners (of Schwarz type). We refer to the IFPACK documentation for more details.

Remark 2. AMESOS *is also used by another* TRILINOS *package,* ML. ML *takes advantages of the* AMESOS *interfaces to solve the coarse problem that arises in multilevel preconditioners; see the* ML *guide for more details [11].*

13 Guide to the Examples

The AMESOS distribution contains examples in subdirectory

```
$TRILINOS_HOME/packages/amesos/example
```

Most of the example requires AMESOS to be configured with support for TRIUTILS. TRIUTILS is a Trilinos package, automatically compiled unless the user specifies

```
--disable-triutils
```

TRIUTILS is used to generate the linear system matrix. New users can start from file

```
$TRILINOS_HOME/packages/amesos/example/example_AmesosFactory.cpp
```

which contains detailed comments about all the AMESOS commands. Example

```
$TRILINOS_HOME/packages/amesos/example/example_AmesosFactory_HB.cpp
```

shows how to read a matrix stored in Harwell/Boeing format, redistribute it to all the processes used in the computation, and use AMESOS to solve the corresponding linear system. Finally, example

```
$TRILINOS_HOME/packages/amesos/example/example_AmesosFactory_Tridiag.cpp
```

creates a simple tridiagonal matrix, and solves the corresponding linear system.

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. W alker, 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] P. Raghavan. Domain-separator codes for the parallel solution of sparse linear systems. Technical Report CSE-02-004, Department of Computer Science and Engineering, The Pennsylvania State University, 2002.

[11] M. Sala, J. Hu, and R. Tuminaro. ML 3.1 smoothed aggregation user's guide. Technical Report SAND-4819, Sandia National Laboratories, September 2004.