MueLu User’s Guide 1.0
(Trilinos version 11.12)

Andrey Prokopenko, Jonathan J. Hu, Tobias A. Wiesner, Christopher M. Siefert, Raymond S. Tuminaro

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

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.
MueLu User’s Guide 1.0
(Trilinos version 11.12)

Andrey Prokopenko
Scalable Algorithms
Sandia National Laboratories
Mailstop 1318
P.O. Box 5800
Albuquerque, NM 87185-1318
aprokop@sandia.gov

Tobias Wiesner
Institute for Computational Mechanics
Technische Universität München
Boltzmanstraße 15
85747 Garching, Germany
wiesner@lnm.mw.tum.de

Jonathan J. Hu
Scalable Algorithms
Sandia National Laboratories
Mailstop 9159
P.O. Box 0969
Livermore, CA 94551-0969
jhu@sandia.gov

Christopher M. Siefert
Computational Multiphysics
Sandia National Laboratories
Mailstop 1322
P.O. Box 5800
Albuquerque, NM 87185-1322
csiefer@sandia.gov

Raymond S. Tuminaro
Computational Mathematics
Sandia National Laboratories
Mailstop 9159
P.O. Box 0969
Livermore, CA 94551-0969
rstumin@sandia.gov
Abstract

This is the official user guide for MUELU multigrid library in Trilinos version 11.12. This guide provides an overview of MUELU, its capabilities, and instructions for new users who want to start using MUELU with a minimum of effort. Detailed information is given on how to drive MUELU through its XML interface. Links to more advanced use cases are given. This guide gives information on how to achieve good parallel performance, as well as how to introduce new algorithms. Finally, readers will find a comprehensive listing of available MUELU options. Any options not documented in this manual should be considered strictly experimental.
Acknowledgment

Many people have helped develop MUELU and/or provided valuable feedback, and we would like to acknowledge their contributions here: Tom Benson, Julian Cortial, Eric Cyr, Stefan Domino, Travis Fisher, Jeremie Gaidamour, Axel Gerstenberger, Chetan Jhurani, Mark Hoemmen, Paul Lin, Eric Phipps, Siva Rajamanickam, Nico Schlömer, and Paul Tsuji.
# Contents

1 Introduction ................................................................. 11

2 Multigrid background ..................................................... 13

3 Getting Started ............................................................... 15
   3.1 Overview of MUELU .................................................. 15
   3.2 Configuration and Build ............................................. 16
      3.2.1 Dependencies .................................................. 16
      3.2.2 Configuration .................................................. 18
   3.3 Examples in code ..................................................... 18
      3.3.1 MUELU as a preconditioner within BELOS .................. 19
      3.3.2 MUELU as a preconditioner for AZTECOO .................. 21
      3.3.3 Further remarks .............................................. 22

4 Performance tips ............................................................ 23

5 MUELU options .............................................................. 25
   5.1 Using parameters on individual levels ............................ 25
   5.2 Parameter validation ................................................. 25
   5.3 General options ....................................................... 26
   5.4 Smoothing and coarse solver options ............................... 27
   5.5 Aggregation options .................................................. 30
   5.6 Rebalancing options .................................................. 31
   5.7 Multigrid algorithm options ......................................... 32
5.8 Reuse options ................................................................. 33
5.9 Miscellaneous options .................................................. 34

References 35

Appendix

A Copyright and License 37

B ML compatibility 39

B.1 Usage of ML parameter lists with MUELU ......................... 39
B.1.1 MLParameterListInterpreter ........................................ 39
B.1.2 ML2MueLuParameterTranslator ................................. 40
B.2 Compatible ML parameters ........................................... 41
B.2.1 General ML options .................................................. 41
B.2.2 Smoothing and coarse solver options ............................ 42
B.2.3 Transfer operator options ........................................... 43
B.2.4 Rebalancing options .................................................. 43
List of Figures
List of Tables

3.1 MUELU’s required and optional dependencies, subdivided by whether a dependency is that of the MUELU library itself (Library), or of some MUELU test (Testing). ................................................................. 17

5.1 Verbosity levels. ................................................................. 26
5.2 Supported problem types (“–” means not set). ....................... 26
5.3 Commonly used smoothers provided by IFPACK/IFPACK2. Note that these smoothers can also be used as coarse grid solvers. ......................................................... 28
5.4 Commonly used direct solvers provided by AMESOS/AMESOS2 ................................. 28
5.5 Available coarsening schemes. ............................................ 30
5.6 Available multigrid algorithms for generating grid transfer matrices. ...................... 32
5.7 Available coarsening schemes. ............................................ 34
Chapter 1

Introduction

This guide gives an overview of MUELU’s capabilities. If you are looking for a tutorial, please refer to the MUELU tutorial in muelu/doc/Tutorial (see also [17]). New users should start with §3. It strives to give the new user all the information he/she might need to begin using MUELU quickly. Users interested in performance, especially in parallel context, should refer to §4. Users looking for a particular option should consult §5, containing a complete set of supported options in MUELU.

If you find any errors or omissions in this guide, have comments or suggestions, or would like to contribute to MUELU development, please contact the MUELU users list, or developers list.
Chapter 2

Multigrid background

Here we provide a brief multigrid introduction (see [5] or [14] for more information). A multigrid solver tries to approximate the original problem of interest with a sequence of smaller (coarser) problems. The solutions from the coarser problems are combined in order to accelerate convergence of the original (fine) problem on the finest grid. A simple multilevel iteration is illustrated in Algorithm 1.

**Algorithm 1** V-cycle multigrid with $N$ levels to solve $Ax = b$.

```
A_0 = A

function MULTILEVEL(A_k, b, u, k)
    // Solve $A_k u = b$ (k is current grid level)
    u = S_m^1(A_k, b, u)
    if ($k \neq N - 1$) then
        P_k = determine_interpolant(A_k)
        R_k = determine_restrictor(A_k)
        \[ \hat{r}_{k+1} = R_k (b - A_k u) \]
        A_{k+1} = R_k A_k P_k
        v = 0
        MULTILEVEL(\hat{A}_{k+1}, \hat{r}_{k+1}, v, k + 1)
        u = u + P_k v
    end if
end function
```

In the multigrid iteration in Algorithm 1, the $S_m^1()$’s and $S_m^2()$’s are called *pre-smoothers* and *post-smoothers*. They are approximate solvers (e.g., symmetric Gauss-Seidel), with the subscript $m$ denoting the number of applications of the approximate solution method. The purpose of a smoother is to quickly reduce certain error modes in the approximate solution on a level $k$. For symmetric problems, the pre- and post-smoothers should be chosen to maintain symmetry (e.g., forward Gauss-Seidel for the pre-smoother and backward Gauss-Seidel for the post-smoother). The $P_k$’s are *interpolation* matrices that transfer solutions from coarse levels to finer levels. The $R_k$’s are *restriction* matrices that restrict a fine level solution to a coarser level. In a geometric multigrid, $P_k$’s and $R_k$’s are determined by the application, whereas in an algebraic multigrid they are automatically generated. For symmetric problems, typically $R_k = P_k^T$. For nonsymmetric problems, this is not necessarily true. The $A_k$’s are the coarse level problems, and are generated through
a Galerkin (triple matrix) product.

Please note that the algebraic multigrid algorithms implemented in MUELU generate the grid transfers $P_k$ automatically and the coarse problems $A_k$ via a sparse triple matrix product. TRILINOS provides a wide selection of smoothers and direct solvers for use in MUELU through the IFPACK, IFPACK2, AMESOS, and AMESOS2 packages (see §5).
Chapter 3

Getting Started

This section is meant to get you using MUEL as quickly as possible. §3 gives a summary of MUEL’s design. §3.2 lists MUEL’s dependencies on other TRILINOS libraries and provides a sample cmake configuration line. Finally, code examples using the XML interface are given in §3.3.

3.1 Overview of MUEL

MUEL is an extensible algebraic multigrid (AMG) library that is part of the TRILINOS project. MUEL works with Epetra (32-bit version ¹) and TPETRA matrix types. The library is written in C++ and allows for different ordinal (index) and scalar types. MUEL is designed to be efficient on many different computer architectures, from workstations to supercomputers, relying on “MPI+X” principle, where “X” can be threading or CUDA.

MUEL provides a number of different multigrid algorithms:

1. smoothed aggregation AMG (for Poisson-like and elasticity problems);
2. Petrov-Galerkin aggregation AMG (for convection-diffusion problems);
3. energy-minimizing AMG;
4. aggregation-based AMG for problems arising from the eddy current formulation of Maxwell’s equations.

MUEL’s software design allows for the rapid introduction of new multigrid algorithms. The most important features of MUEL can be summarized as:

Easy-to-use interface

MUEL has a user-friendly parameter input deck which covers most important use cases. Reasonable defaults are provided for common problem types (see Table 5.2).

¹Support for the Epetra 64-bit version is planned.
Modern object-oriented software architecture

*MUELU* is written completely in C++ as a modular object-oriented multigrid framework, which provides flexibility to combine and reuse existing components to develop novel multigrid methods.

Extensibility

Due to its flexible design, *MUELU* is an excellent toolkit for research on novel multigrid concepts. Experienced multigrid users have full access to the underlying framework through an advanced XML based interface. Expert users may use and extend the C++ API directly.

Integration with TRILINOS library

As a package of TRILINOS, *MUELU* is well integrated into the TRILINOS environment. *MUELU* can be used with either the *Tpetra* or *Epetra* (32-bit) linear algebra stack. It is templated on the local index, global index, scalar, and compute node types. This makes *MUELU* ready for future developments.

Broad range of supported platforms

*MUELU* runs on wide variety of architectures, from desktop workstations to parallel Linux clusters and supercomputers ([10]).

Open source

*MUELU* is freely available through a simplified BSD license (see Appendix A).

3.2 Configuration and Build

*MUELU* has been compiled successfully under Linux with the following C++ compilers: GNU versions 4.1 and later, Intel versions 12.1/13.1, and clang versions 3.2 and later. In the future, we recommend using compilers supporting C++11 standard.

3.2.1 Dependencies

Required Dependencies

*MUELU* requires that *Teuchos* and either *Epetra/IPACK* or *Tpetra/IPack2* are enabled.

Recommended Dependencies

We strongly recommend that you enable the following TRILINOS libraries along with *MUELU*:

- **Epetra stack**: AztecOO, Epetra, Amesos, Ipack, Isorropia, Galeri, Zoltan;
Tutorial Dependencies

In order to run the MUELU Tutorial [17] located in muelu/doc/Tutorial, MUELU must be configured with the following dependencies enabled:

AZTECOO, AMESOS, AMESOS2, BELOS, EPETRA, IFPACK, IFPACK2, ISORROPIA, GALERI, TPETRA, ZOLTAN, ZOLTAN2.

Note that the MUELU tutorial [17] comes with a VirtualBox image with a pre-installed Linux and TRILINOS. In this way, a user can immediately begin experimenting with MUELU without having to install the TRILINOS libraries. Therefore, it is an ideal starting point to get in touch with MUELU.

Complete List of Direct Dependencies

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Required Library</th>
<th>Required Testing</th>
<th>Optional Library</th>
<th>Optional Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMESOS</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMESOS2</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AZTECOO</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BELOS</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPETRA</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFPACK</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IFPACK2</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISORROPIA</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GALERI</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KOKKOSCLASSIC</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEUCHOS</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPETRA</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XPETRA</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZOLTAN</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZOLTAN2</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boost</td>
<td></td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>BLAS</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAPACK</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI</td>
<td>×</td>
<td>×</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1. MUELU’s required and optional dependencies, subdivided by whether a dependency is that of the MUELU library itself (Library), or of some MUELU test (Testing).
Table 3.1 lists the dependencies of MUELU, both required and optional. If an optional dependency is not present, the tests requiring that dependency are not built.

- **AMESOS/AMESOS2** are necessary if one wants to use a sparse direct solve on the coarsest level. **ZOLTAN/ZOLTAN2** are necessary if one wants to use matrix rebalancing in parallel runs (see §4). **AZTECOO/BELOS** are necessary if one wants to test MUELU as a preconditioner instead of a solver.

- **MUELU** has also been successfully tested with SuperLU 4.1 and SuperLU 4.2.

- Some packages that MUELU depends on may come with additional requirements for third party libraries, which are not listed here as explicit dependencies of MUELU. It is highly recommended to install ParMetis 3.1.1 or newer for **ZOLTAN**, ParMetis 4.0.x for **ZOLTAN2**, and SuperLU 4.1 or newer for **AMESOS/AMESOS2**.

### 3.2.2 Configuration

The preferred way to configure and build MUELU is to do that outside of the source directory. Here we provide a sample configure script that will enable MUELU and all of its optional dependencies:

```bash
export TRILINOS_HOME=/path/to/your/Trilinos/source/directory
cmake -D BUILD_SHARED_LIBS:BOOL=ON \ 
  -D CMAKE_BUILD_TYPE:STRING="RELEASE" \ 
  -D CMAKE_CXX_FLAGS:STRING="-g" \ 
  -D Trilinos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \ 
  -D Trilinos_ENABLE_TESTS:BOOL=OFF \ 
  -D Trilinos_ENABLE_EXAMPLES:BOOL=OFF \ 
  -D Trilinos_ENABLE_MueLu:BOOL=ON \ 
  -D MueLu_ENABLE_TESTS:STRING=ON \ 
  -D MueLu_ENABLE_EXAMPLES:STRING=ON \ 
  -D TPL_ENABLE_BLAS:BOOL=ON \ 
  -D TPL_ENABLE_MPI:BOOL=ON \ 
  ${TRILINOS_HOME}
```

More configure examples can be found in Trilinos/sampleScripts. For more information on configuring, see the TRILINOS Cmake Quickstart guide [1].

### 3.3 Examples in code

The most commonly used scenario involving MUELU is using a multigrid preconditioner inside an iterative linear solver. In TRILINOS, a user has a choice between **EPETRA** and **TPETRA**
for the underlying linear algebra library. Important Krylov subspace methods (such as preconditioned CG and GMRES) are provided in TRILINOS packages AZTECOO (EPETRA) and BELOS (EPETRA/TPETRA).

At this point, we assume that the reader is comfortable with TEUCHOS referenced-counted pointers (RCPs) for memory management (an introduction to RCPs can be found in [3]) and the Teuchos::ParameterList class [13].

### 3.3.1 MUELU as a preconditioner within BELOS

The following code shows the basic steps of how to use a MUELU multigrid preconditioner with TPETRA linear algebra library and with a linear solver from BELOS. To keep the example short and clear, we skip the template parameters and focus on the algorithmic outline of setting up a linear solver. For further details, a user may refer to the examples and test directories.

First, we create the MUELU multigrid preconditioner. It can be done in a few ways. For instance, multigrid parameters can be read from an XML file (e.g., `mueluOptions.xml` in the example below).

```cpp
Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...
std::string optionsFile = "mueluOptions.xml";
Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner =
    MueLu::CreateTpetraPreconditioner(A, optionsFile);
```

The XML file contains multigrid options. A typical file with MUELU parameters looks like the following.

```xml
<ParameterList name="MueLu">
  <Parameter name="verbosity" type="string" value="low"/>
  <Parameter name="max levels" type="int" value="3"/>
  <Parameter name="coarse: max size" type="int" value="10"/>
  <Parameter name="multigrid algorithm" type="string" value="sa"/>
  <!-- Damped Jacobi smoothing -->
  <Parameter name="smoother: type" type="string" value="RELAXATION"/>
  <ParameterList name="smoother: params">
    <Parameter name="relaxation: type" type="string" value="Jacobi"/>
    <Parameter name="relaxation: sweeps" type="int" value="1"/>
    <Parameter name="relaxation: damping factor" type="double" value="0.9"/>
  </ParameterList>
  <!-- Aggregation -->
</ParameterList>
```

19
It defines a three level smoothed aggregation multigrid algorithm. The aggregation size is between three and nine (2D)/27 (3D) nodes. One sweep with a damped Jacobi method is used as a level smoother. By default, a direct solver is applied on the coarsest level. A complete list of available parameters and valid parameter choices is given in §5 of this User’s Guide.

Users can also construct a multigrid preconditioner using a provided ParameterList without accessing any files in the following manner.

```cpp
Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...
Teuchos::ParameterList paramList;
paramList.set("verbosity", "low");
paramList.set("max levels", 3);
paramList.set("coarse: max size", 10);
paramList.set("multigrid algorithm", "sa");
// ...
Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner =
    MueLu::CreateTpetraPreconditioner(A, paramList);
```

Besides the linear operator $A$, we also need an initial guess vector for the solution $X$ and a right hand side vector $B$ for solving a linear system.

```cpp
Teuchos::RCP<const Tpetra::Map<> > map = A->getDomainMap();

// Create initial vectors
Teuchos::RCP<Tpetra::MultiVector<> > B, X;
X = Teuchos::rcp( new Tpetra::MultiVector<> (map,numrhs) );
Belos::MultiVecTraits<>::MvRandom( *X );
B = Teuchos::rcp( new Tpetra::MultiVector<> (map,numrhs) );
Belos::OperatorTraits<>::Apply( *A, *X, *B );
Belos::MultiVecTraits<>::MvInit( *X, 0.0 );
```

To generate a dummy example, the above code first declares two vectors. Then, a right hand side vector is calculated as the matrix-vector product of a random vector with the operator $A$. Finally, an initial guess is initialized with zeros.

Then, one can define a Belos::LinearProblem object where the mueLuPreconditioner is used for left preconditioning

```cpp
Belos::LinearProblem<> problem( A, X, B );
problem->setLeftPrec(mueLuPreconditioner);
```
Next, we set up a BELOS solver using some basic parameters

```cpp
Teuchos::ParameterList belosList;
belosList.set( "Block Size", 1 );
belosList.set( "Maximum Iterations", 100 );
belosList.set( "Convergence Tolerance", 1e-10 );
belosList.set( "Output Frequency", 1 );
belosList.set( "Verbosity", Belos::TimingDetails + Belos::FinalSummary );
Belos::BlockCGSolMgr<> solver( rcp(&problem,false), rcp(&belosList,false) );
```

Finally, we solve the system.

```cpp
Belos::ReturnType ret = solver.solve();
```

### 3.3.2 MUELU as a preconditioner for AZTECOO

For EPETRA, users have two library options: BELOS (recommended) and AZTECOO. AZTECOO and BELOS both provide fast and mature implementations of common iterative Krylov linear solvers. BELOS has additional capabilities, such as Krylov subspace recycling and “tall skinny QR”.

Constructing a MUELU preconditioner for Epetra operators is done in a similar manner to Tpetra.

```cpp
Teuchos::RCP<Epetra_CrsMatrix> A;
// create A here ...
Teuchos::RCP<MueLu::EpetraOperator> mueLuPreconditioner;
std::string optionsFile = "mueluOptions.xml";
mueLuPreconditioner = MueLu::CreateEpetraPreconditioner(A, optionsFile);
```

MUELU parameters are generally Epetra/Tpetra agnostic, thus the XML parameter file could be the same as §3.3.1.

Furthermore, we assume that a right hand side vector and a solution vector with the initial guess are defined.

```cpp
Teuchos::RCP<const Epetra_Map> map = A->DomainMap();
Teuchos::RCP<Epetra_Vector> B = Teuchos::rcp(new Epetra_Vector(map));
Teuchos::RCP<Epetra_Vector> X = Teuchos::rcp(new Epetra_Vector(map));
X->PutScalar(0.0);
```

Then, an Epetra_LinearProblem can be defined.
Epetra_LinearProblem epetraProblem(A.get(), X.get(), B.get());

The following code constructs an AZTECOO CG solver.

AztecOO aztecSolver(epetraProblem);
aztecSolver.SetAztecOption(AZ_solver, AZ_cg);
aztecSolver.SetPrecOperator(mueLuPreconditioner.get());

Finally, the linear system is solved.

```c++
int maxIts = 100;
double tol = 1e-10;
aztecSolver.Iterate(maxIts, tol);
```

### 3.3.3 Further remarks

This section is only meant to give a brief introduction on how to use MUELU as a preconditioner within the TRILINOS packages for iterative solvers. There are other, more complicated, ways to use MUELU as a preconditioner for BELOS and AZTECOO through the XPETRA interface. Of course, MUELU can also work as standalone multigrid solver. For more information on these topics, the reader may refer to the examples and tests in the MUELU source directory (Trilinos/packages/muelu), as well as to the MUELU tutorial [17]. For in-depth details of MUELU applied to multiphysics problems, please see [16].
In practice, it can be very challenging to find an appropriate set of multigrid parameters for a specific problem, especially if few details are known about the underlying linear system. In this Chapter, we provide some advice for improving multigrid performance.

For optimizing multigrid parameters, it is highly recommended to set the verbosity to high or extreme for MUELU to output more information (e.g., for the effect of the chosen parameters to the aggregation and coarsening process).

Some general advice:

- Choose appropriate iterative linear solver (e.g., GMRES for non-symmetric problems).
- Start with the recommended settings for particular problem types. See Table 5.2.
- Choose reasonable basic multigrid parameters (see §5.3), including maximum number of multigrid levels (max_levels) and maximum allowed coarse size of the problem (coarse: max_size). Take fine level problem size and sparsity pattern into account for a reasonable choice of these parameters.
- Select an appropriate transfer operator strategy (see §5.7). For symmetric problems, you should start with smoothed aggregation multigrid. For non-symmetric problems, a Petrov-Galerkin smoothed aggregation method is a good starting point, though smoothed aggregation may also perform well.
- Enable implicit restrictor construction (transpose: use implicit) for symmetric problems.
- Find good level smoothers (see §5.4). If a problem is symmetric positive definite, choose a smoother with a matrix-vector computational kernel, such as the Chebyshev polynomial smoother. If you are using relaxation smoothers, we recommend starting with optimizing the damping parameter. Once you have found a good damping parameter for your problem, you can increase the number of smoothing iterations.
- Adjust aggregation parameters if you experience bad coarsening ratios (see §5.5). Particularly, try adjusting the minimum (aggregation: min_agg_size) and maximum (aggregation: max_agg_size) aggregation parameters. For a 2D (3D) isotropic problem on a regular mesh, the aggregate size should be about 9 (27) nodes per aggregate.
• Replace a direct solver with an iterative method (coarse: type) if your coarse level solution becomes too expensive (see §5.4).

Some advice for parallel runs include:

1. Enable matrix rebalancing when running in parallel (repartition: enable).

2. Use smoothers invariant to the number of processors, such as polynomial smoothing, if possible.

3. Use uncoupled aggregation instead of coupled, as later requires significantly more communication.

4. Adjust rebalancing parameters (see §5.6). Try choosing rebalancing parameters so that you end up with one processor on the coarsest level for the direct solver (this avoids additional communication).

5. Enable implicit rebalancing of prolongators and restrictors (repartition: rebalance P and R).
Chapter 5

MUELU options

In this section, we report the complete list of MUELU input parameters. It is important to notice, however, that MUELU relies on other TRILINOS packages to provide support for some of its algorithms. For instance, IFPACK/IFPACK2 provide standard smoothers like Jacobi, Gauss-Seidel or Chebyshev, while AMESOS/AMESOS2 provide access to direct solvers. The parameters affecting the behavior of the algorithms in those packages are simply passed by MUELU to a routine from the corresponding library. Please consult corresponding packages’ documentation for a full list of supported algorithms and corresponding parameters.

5.1 Using parameters on individual levels

Some of the parameters that affect the preconditioner can in principle be different from level to level. By default, parameters affect all levels in a multigrid hierarchy.

The settings on a particular levels can be changed by using level sublists. A level sublist is a ParameterList sublist with the name “level XX”, where XX is the level number. The parameter names in the sublist do not require any modifications. For example, the following fragment of code

```xml
<ParameterList name="level 2">
  <Parameter name="smoother: type" type="string" value="CHEBYSHEV"/>
</ParameterList>
```

changes the smoother for level 2 to be a polynomial smoother.

5.2 Parameter validation

By default, MUELU validates the input parameter list. A parameter that is misspelled, is unknown, or has an incorrect value type will cause an exception to be thrown and execution to halt.

Spaces are important within a parameter’s name. Please separate words by just one space, and make sure there are no leading or trailing spaces.
The option print initial parameters prints the initial list given to the interpreter. The option print unused parameters prints the list of unused parameters.

### 5.3 General options

<table>
<thead>
<tr>
<th>Verbosity level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>No output</td>
</tr>
<tr>
<td>low</td>
<td>Errors, important warnings, and some statistics</td>
</tr>
<tr>
<td>medium</td>
<td>Same as low, but with more statistics</td>
</tr>
<tr>
<td>high</td>
<td>Errors, all warnings, and all statistics</td>
</tr>
<tr>
<td>extreme</td>
<td>Same as high, but also includes output from other packages (i.e., ZOLTAN)</td>
</tr>
</tbody>
</table>

*Table 5.1.* Verbosity levels.

<table>
<thead>
<tr>
<th>Problem type</th>
<th>Multigrid algorithm</th>
<th>Block size</th>
<th>Smoother</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Poisson-2D</td>
<td>Smoothed aggregation</td>
<td>1</td>
<td>Chebyshev</td>
</tr>
<tr>
<td>Poisson-3D</td>
<td>Smoothed aggregation</td>
<td>1</td>
<td>Chebyshev</td>
</tr>
<tr>
<td>Elasticity-2D</td>
<td>Smoothed aggregation</td>
<td>2</td>
<td>Chebyshev</td>
</tr>
<tr>
<td>Elasticity-3D</td>
<td>Smoothed aggregation</td>
<td>3</td>
<td>Chebyshev</td>
</tr>
<tr>
<td>ConvectionDiffusion</td>
<td>Petrov-Galerkin AMG</td>
<td>1</td>
<td>Gauss-Seidel</td>
</tr>
<tr>
<td>MHD</td>
<td>Unsmoothed aggregation</td>
<td>–</td>
<td>Additive Schwarz method with one level of overlap and ILU(0) as a subdomain solver</td>
</tr>
</tbody>
</table>

*Table 5.2.* Supported problem types (“–” means not set).

- **problem:** type  
  [string] Type of problem to be solved. Possible values: see Table 5.2. **Default:** "unknown".

- **verbosity**  
  [string] Control of the amount of printed information. Possible values: see Table 5.1. **Default:** "high".
number of equations | [int] Number of PDE equations at each grid node. Only constant block size is considered. **Default:** 1.

max levels | [int] Maximum number of levels in a hierarchy. **Default:** 10.

cycle type | [string] Multigrid cycle type. Possible values: "V", "W". **Default:** "V".

problem: symmetric | [bool] Symmetry of a problem. This setting affects the construction of a restrictor. If set to true, the restrictor is set to be the transpose of a prolongator. If set to false, underlying multigrid algorithm makes the decision. **Default:** true.

### 5.4 Smoothing and coarse solver options

*MUELU* relies on other *TRILINOS* packages to provide level smoothers and coarse solvers. *IFPACK* and *IFPACK2* provide standard smoothers (see Table 5.3), and *AMESOS* and *AMESOS2* provide direct solvers (see Table 5.4). Note that it is completely possible to use any level smoother as a direct solver.

**MUELU** checks parameters `smoother: * type` and `coarse: type` to determine:

- what package to use (i.e., is it a smoother or a direct solver);
- (possibly) transform a smoother type

- *IFPACK* and *IFPACK2* use different smoother type names, e.g., “point relaxation standalone” vs “RELAXATION”. **MUELU** tries to follow *IFPACK2* notation for smoother types. Please consult *IFPACK2* documentation [8] for more information.

The parameter lists `smoother: * params` and `coarse: params` are passed directly to the corresponding package without any examination of their content. Please consult the documentation of the corresponding packages for a list of possible values.

By default, **MUELU** uses one sweep of symmetric Gauss-Seidel for both pre- and post-smoothing, and SuperLU for coarse system solver.
smoother: type

<table>
<thead>
<tr>
<th>RELAXATION</th>
<th>Point relaxation smoothers, including Jacobi, Gauss-Seidel, symmetric Gauss-Seidel, etc. The exact smoother is chosen by specifying relaxation: type parameter in the smoother: params sublist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHEBYSHEV</td>
<td>Chebyshev polynomial smoother.</td>
</tr>
<tr>
<td>ILUT, RILUK</td>
<td>Local (processor-based) incomplete factorization methods.</td>
</tr>
</tbody>
</table>

Table 5.3. Commonly used smoothers provided by IFPACK/IFPACK2. Note that these smoothers can also be used as coarse grid solvers.

<table>
<thead>
<tr>
<th>coarse: type</th>
<th>Amesos</th>
<th>Amesos2</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLU</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>KLU2</td>
<td>x</td>
<td>Default AMEOS solver [7].</td>
</tr>
<tr>
<td>SuperLU</td>
<td>x</td>
<td>Default AMEOS2 solver [4].</td>
</tr>
<tr>
<td>SuperLU_dist</td>
<td>x</td>
<td>Third-party serial sparse direct solver [9].</td>
</tr>
<tr>
<td>Umfpack</td>
<td>x</td>
<td>Third-party solver [6].</td>
</tr>
<tr>
<td>Mumps</td>
<td>x</td>
<td>Third-party solver [2].</td>
</tr>
</tbody>
</table>

Table 5.4. Commonly used direct solvers provided by AME-SOS/AMESOS2

smoother: pre or post [string] Pre- and post-smoother combination. Possible values: "pre" (only pre-smoother), "post" (only post-smoother), "both" (both pre-and post-smoothers), "none" (no smoothing). Default: "both".

smoother: type [string] Smoother type. Possible values: see Table 5.3. Default: "RELAXATION".

smoother: pre type [string] Pre-smoother type. Possible values: see Table 5.3. Default: "RELAXATION".

smoother: post type [string] Post-smoother type. Possible values: see Table 5.3. Default: "RELAXATION".
smoother: params [ParameterList] Smoother parameters. For standard smoothers, MUELU passes them directly to the appropriate package library.

smoother: pre params [ParameterList] Pre-smoother parameters. For standard smoothers, MUELU passes them directly to the appropriate package library.

smoother: post params [ParameterList] Post-smoother parameters. For standard smoothers, MUELU passes them directly to the appropriate package library.

smoother: overlap [int] Smoother subdomain overlap. Default: 0.

smoother: pre overlap [int] Pre-smoother subdomain overlap. Default: 0.


coarse: type [string] Coarse solver. Possible values: see Table ??.. Default: "SuperLU".

coarse: params [ParameterList] Coarse solver parameters. MUELU passes them directly to the appropriate package library.

Attempts to construct aggregates of optimal size ($3^d$ nodes in $d$ dimensions). Each process works independently, and aggregates cannot span multiple processes.

Attempts to construct aggregates of optimal size ($3^d$ nodes in $d$ dimensions). Aggregates are allowed to cross processor boundaries. **Use carefully.** If unsure, use uncoupled instead.

**Table 5.5.** Available coarsening schemes.

## 5.5 Aggregation options

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregation: type</td>
<td>[string]</td>
<td>Aggregation scheme. Possible values: see Table 5.5. <strong>Default:</strong> &quot;uncoupled&quot;.</td>
</tr>
<tr>
<td>aggregation: ordering</td>
<td>[string]</td>
<td>Node ordering strategy. Possible values: &quot;natural&quot; (local index order), &quot;graph&quot; (filtered graph breadth-first order), &quot;random&quot; (random local index order). <strong>Default:</strong> &quot;natural&quot;.</td>
</tr>
<tr>
<td>aggregation: drop scheme</td>
<td>[string]</td>
<td>Connectivity dropping scheme for a graph used in aggregation. Possible values: &quot;classical&quot;, &quot;distance laplacian&quot;. <strong>Default:</strong> &quot;classical&quot;.</td>
</tr>
<tr>
<td>aggregation: drop tol</td>
<td>[double]</td>
<td>Connectivity dropping threshold for a graph used in aggregation. <strong>Default:</strong> 0.0.</td>
</tr>
<tr>
<td>aggregation: min agg size</td>
<td>[int]</td>
<td>Minimum size of an aggregate. <strong>Default:</strong> 2.</td>
</tr>
<tr>
<td>aggregation: max agg size</td>
<td>[int]</td>
<td>Maximum size of an aggregate (-1 means unlimited). <strong>Default:</strong> -1.</td>
</tr>
<tr>
<td>aggregation: Dirichlet threshold</td>
<td>[double]</td>
<td>Threshold for determining whether entries are zero during Dirichlet row detection. <strong>Default:</strong> 0.0.</td>
</tr>
</tbody>
</table>
aggregation: export visualization data
[bool] Export data for visualization post-processing.
Default: false.

5.6 Rebalancing options

repartition: enable

repartition: partitioner
[string] Partitioning package to use. Possible values: ”zoltan” (ZOLTAN library), ”zoltan2” (ZOLTAN2 library). Default: ”zoltan2”.

repartition: params
[ParameterList] Partitioner parameters. MUELU passes them directly to the appropriate package library.

repartition: start level
[int] Minimum level to run partitioner. MUELU does not rebalance levels finer than this one. Default: 2.

repartition: min rows per proc
[int] Minimum number of rows per processor. If the actual number is smaller, then rebalancing occurs. Default: 800.

repartition: max imbalance
[double] Maximum nonzero imbalance ratio. If the actual number is larger, the rebalancing occurs. Default: 1.2.

repartition: remap parts
[bool] Postprocessing for partitioning to reduce data migration. Default: true.

repartition: rebalance P and R
[bool] Explicit rebalancing of R and P during the setup. This speeds up the solve, but slows down the setup phases. Default: false.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sa</td>
<td>Classic smoothed aggregation [15]</td>
</tr>
<tr>
<td>unsmoothed</td>
<td>Aggregation-based, same as sa but without damped Jacobi prolongator improvement step</td>
</tr>
<tr>
<td>pg</td>
<td>Prolongator smoothing using $A$, restriction smoothing using $A^T$, local damping factors [12]</td>
</tr>
<tr>
<td>emin</td>
<td>Constrained minimization of energy in basis functions of grid transfer operator [18, 11]</td>
</tr>
</tbody>
</table>

**Table 5.6.** Available multigrid algorithms for generating grid transfer matrices.

### 5.7 Multigrid algorithm options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multigrid algorithm</td>
<td>[string] Multigrid method. Possible values: see Table 5.6. Default: &quot;sa&quot;.</td>
</tr>
<tr>
<td>semicoarsen: coarsen rate</td>
<td>[int] Rate at which to coarsen unknowns in the z direction. Default: 3.</td>
</tr>
<tr>
<td>sa: use filtered matrix</td>
<td>[bool] Matrix to use for smoothing the tentative prolongator. The two options are: to use the original matrix, and to use the filtered matrix with filtering based on filtered graph used for aggregation. Default: true.</td>
</tr>
<tr>
<td>filtered matrix: use lumping</td>
<td>[bool] Lump (add to diagonal) dropped entries during the construction of a filtered matrix. This allows user to preserve constant nullspace. Default: true.</td>
</tr>
<tr>
<td>filtered matrix: reuse eigenvalue</td>
<td>[bool] Skip eigenvalue calculation during the construction of a filtered matrix by reusing eigenvalue estimate from the original matrix. This allows us to skip heavy computation, but may lead to poorer convergence. Default: true.</td>
</tr>
</tbody>
</table>

32
**emin: iterative method**  
[string] Iterative method to use for energy minimization of initial prolongator in energy-minimization. Possible values: "cg" (conjugate gradient), "sd" (steepest descent). **Default:** "cg".

**emin: num iterations**  
[int] Number of iterations to minimize initial prolongator energy in energy-minimization. **Default:** 2.

**emin: num reuse iterations**  
[int] Number of iterations to minimize the reused prolongator energy in energy-minimization. **Default:** 1.

**emin: pattern**  
[string] Sparsity pattern to use for energy minimization. Possible values: "AkPtent". **Default:** "AkPtent".

**emin: pattern order**  
[int] Matrix order for the "AkPtent" pattern. **Default:** 1.

### 5.8 Reuse options

Reuse options are currently only used with sa multigrid algorithm. We also assume that the matrix preserves graph structure, and only matrix values change.

In addition, please note that not all combinations of multigrid algorithms and reuse options are valid, or even make sense. For instance, the "emin" reuse option should only be use with "emin" multigrid algorithm.

**reuse: type**  
[string] Reuse options for consecutive hierarchy construction. This speeds up the setup phase, but may lead to poorer convergence. Possible values: see Table 5.7. **Default:** "none".
<table>
<thead>
<tr>
<th>none</th>
<th>No reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>emin</td>
<td>Reuse old prolongator as an initial guess to energy minimization, and reuse the prolongator pattern</td>
</tr>
<tr>
<td>RP</td>
<td>Reuse smoothed prolongator and restrictor. Smoothers are recomputed. RP should reuse matrix graphs for matrix-matrix product, but currently that is disabled as only PETRA supports it.</td>
</tr>
<tr>
<td>tP</td>
<td>Reuse tentative prolongator. The graphs of smoothed prolongator and matrices in Galerkin product are reused only if filtering is not being used (i.e., either sa: use filtered matrix or aggregation: drop tol is false)</td>
</tr>
<tr>
<td>RAP</td>
<td>Recompute only the finest level smoothers, reuse all other operators</td>
</tr>
<tr>
<td>full</td>
<td>Reuse everything</td>
</tr>
</tbody>
</table>

**Table 5.7.** Available coarsening schemes.

### 5.9 Miscellaneous options

- **export data**
  - [ParameterList] Exporting a subset of the hierarchy data in a file. Currently, the list can contain any of three parameter names ("A", "P", "R") of type string and value "{levels separated by commas}". A matrix with a name "X" is saved in three MatrixMarket files: a) data is saved in X_level.mm; b) its row map is saved in rowmap_X_level.mm; c) its column map is saved in colmap_X_level.mm.

- **print initial parameters**
  - [bool] Print parameters provided for a hierarchy construction. **Default:** true.

- **print unused parameters**
  - [bool] Print parameters unused during a hierarchy construction. **Default:** true.

- **transpose: use implicit**
  - [bool] Use implicit transpose for the restriction operator. **Default:** false.
References


Appendix A

Copyright and License

MueLu: A package for multigrid based preconditioning

Copyright 2012 Sandia Corporation

Under the terms of Contract DE–AC04–94AL85000 with Sandia Corporation, the U.S. Government retains certain rights in this software.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the Corporation nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Appendix B

ML compatibility

MUELU provides a basic compatibility layer for ML parameter lists. This allows ML users to quickly perform some experiments with MUELU.

First and most important: Long term, we would like to have users use the new MUELU interface, as that is where most of new features will be made accessible. One should make note of the fact that it may not be possible to make ML deck do exactly same things in ML and MUELU, as internally ML implicitly makes some decision that we have no control over and which could be different from MUELU.

There are basically two distinct ways to use ML input parameters with MUELU:

MLParameterListInterpreter: This class is the pendant of the ParameterListInterpreter class for the MUELU parameters. It accepts parameter lists or XML files with ML parameters and generates a MUELU multigrid hierarchy. It supports only a well-defined subset of ML parameters which have a equivalent parameter in MUELU.

ML2MueLuParameterTranslator: This class is a simple wrapper class which translates ML parameters to the corresponding MUELU parameters. It has to be used in combination with the MUELU ParameterListInterpreter class to generate a MUELU multigrid hierarchy. It is also meant to be used in combination with the CreateEpetraPreconditioner and CreateTpetraPreconditioner routines (see §3.3). It supports only a small subset of the ML parameters.

B.1 Usage of ML parameter lists with MUELU

B.1.1 MLParameterListInterpreter

The MLParameterListInterpreter directly accepts a ParameterList containing ML parameters. It also interprets the null space: vectors and the null space: dimension ML parameters. However, it is recommended to provide the near null space vectors directly in the MUELU way as shown in the following code snippet.
Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...

// XML file containing ML parameters
std::string xmlFile = "mlParameters.xml"
Teuchos::ParameterList paramList;
Teuchos::updateParametersFromXmlFileAndBroadcast(xmlFile, Teuchos::Ptr<Teuchos ::ParameterList>(&paramList), *comm);

// use ParameterListInterpreter with MueLu parameters as input
Teuchos::RCP<ParameterListInterpreter> mueluFactory = Teuchos::rcp(new MLParameterListInterpreter(*paramList));

RCP<Hierarchy> H = mueluFactory->CreateHierarchy();
H->GetLevel(0)->Set<RCP<Matrix> >("A", A);
H->GetLevel(0)->Set("Nullspace", nullspace);
H->GetLevel(0)->Set("Coordinates", coordinates);
mueluFactory->SetupHierarchy(*H);

Note that the MLParameterListInterpreter only supports a basic set of ML parameters allowing to build smoothed aggregation transfer operators (see §B.2 for a list of compatible ML parameters).

### B.1.2 ML2MueLuParameterTranslator

The ML2MueLuParameterTranslator class is a simple wrapper translating ML parameters to the corresponding MUELU parameters. This allows the usage of the simple CreateEpetraPreconditioner and CreateTpetraPreconditioner interface with ML parameters:

Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...

// XML file containing ML parameters
std::string xmlFile = "mlParameters.xml"
Teuchos::ParameterList paramList;
Teuchos::updateParametersFromXmlFileAndBroadcast(xmlFile, Teuchos::Ptr<Teuchos ::ParameterList>(&paramList), *comm);

// translate ML parameters to MueLu parameters
RCP<ParameterList> mueluParamList = Teuchos::getParametersFromXmlString(MueLu::ML2MueLuParameterTranslator::translate(paramList,"SA"));

Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner = MueLu::CreateTpetraPreconditioner(A, mueluParamList);
In a similar way, ML input parameters can be used with the standard MueLu parameter list interpreter class. Note that the near null space vectors have to be provided in the MueLu way.

```cpp
Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...

// XML file containing ML parameters
std::string xmlFile = "mlParameters.xml"
Teuchos::ParameterList paramList;
Teuchos::updateParametersFromXmlFileAndBroadcast(xmlFile, Teuchos::Ptr<Teuchos::ParameterList>(&paramList), *comm);

// translate ML parameters to MueLu parameters
Teuchos::RCP<ParameterList> mueluParamList = Teuchos::getParametersFromXmlString(MueLu::ML2MueLuParameterTranslator::translate(paramList,"SA"));

// use ParameterListInterpreter with MueLu parameters as input
Teuchos::RCP<HierarchyManager> mueluFactory = Teuchos::rcp(new ParameterListInterpreter(*mueluParamList));

RCP<Hierarchy> H = mueluFactory->CreateHierarchy();
H->GetLevel(0)->Set<RCP<Matrix> >("A", A);
H->GetLevel(0)->Set("Nullspace", nullspace);
H->GetLevel(0)->Set("Coordinates", coordinates);
mueluFactory->SetupHierarch(*H);
```

Note that the set of supported ML parameters is very limited. Please refer to §B.2 for a list of all compatible ML parameters.

## B.2 Compatible ML parameters

### B.2.1 General ML options

<table>
<thead>
<tr>
<th>ML output</th>
<th>[int] Control of the amount of printed information. Possible values: 0-10 with 0=no output and 10=maximum verbosity. <strong>Default</strong>: 0 <strong>Compatibility</strong>: MLParameterListInterpreter, ML2MueLuParameterTranslator.</th>
</tr>
</thead>
</table>

| PDE equations | [int] Number of PDE equations at each grid node. Only constant block size is considered. **Default**: 1 **Compatibility**: MLParameterListInterpreter, ML2MueLuParameterTranslator. |
max levels [int] Maximum number of levels in a hierarchy. Default: 10 Compatibility: MLParameterListInterpreter, ML2MueLuParameterTranslator.

prec type [string] Multigrid cycle type. Possible values: "MGV", "MGW". Other values are NOT supported by MueLu. Default: "MGV" Compatibility: MLParameterListInterpreter, ML2MueLuParameterTranslator.

### B.2.2 Smoothing and coarse solver options


smoother: pre or post [string] Pre- and post-smoother combination. Possible values: "pre" (only pre-smoother), "post" (only post-smoother), "both" (both pre-and post-smoothers). Default: "both" Compatibility: MLParameterListInterpreter, ML2MueLuParameterTranslator.
max size


course: type


B.2.3 Transfer operator options

energy minimization: enable


aggregation: damping factor


B.2.4 Rebalancing options

repartition: enable


repartition: start level

[int] Minimum level to run partitioner. MUELU does not rebalance levels finer than this one. Default: 1 Compatibility: MLParameterListInterpreter.

repartition: min per proc

[int] Minimum number of rows per processor. If the actual number if smaller, then rebalancing occurs. Default: 512 Compatibility: MLParameterListInterpreter.
repartition: max min ratio

[double] Maximum nonzero imbalance ratio. If the actual number is larger, the rebalancing occurs. **Default:** 1.3 **Compatibility:** MLParameterListInterpreter.
DISTRIBUTION:

1 Tobias Wiesner  
   Institute for Computational Mechanics  
   Technische Universität München  
   Boltzmanstraße 15  
   85747 Garching, Germany

1 MS 1320     Michael Heroux, 1446  
1 MS 1318     Robert Hoekstra, 1446  
1 MS 1320     Mark Hoemmen, 1446  
1 MS 1320     Paul Lin, 1446  
1 MS 1318     Andrey Prokopenko, 1426  
1 MS 1322     Christopher Siefert, 1443  
1 MS 0899     Technical Library, 9536 (electronic copy)
**DISTRIBUTION:**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Code</th>
<th>Name and Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MS 9159</td>
<td>Jonathan Hu, 1426</td>
</tr>
<tr>
<td>1</td>
<td>MS 9159</td>
<td>Paul Tsuji, 1442</td>
</tr>
<tr>
<td>1</td>
<td>MS 9159</td>
<td>Raymond Tuminaro, 1442</td>
</tr>
<tr>
<td>1</td>
<td>MS 0899</td>
<td>Technical Library, 8944 (electronic copy)</td>
</tr>
</tbody>
</table>