MueLu - AMG Design and Extensibility

Tobias Wiesner
Andrey Prokopenko
Jonathan Hu

Sandia National Labs

March 3, 2015
Main components of Algebraic Multigrid

**V-cycle (3 levels)**

- $S_0^{pre}$
- $R_1$
- $S_1^{pre}$
- $R_2$
- $S_2$
- $S_0^{post}$
- $P_1$
- $S_1^{post}$
- $P_2$

- $A_i = R_i A_{i-1} P_i$

**Example for aggregates**

**Main components**

- Smoothers $S_i$
- Level transfers $P_i$ and $R_i$
  - Based on aggregates
  - Transfer operator smoothing
- Galerkin product $A_i = R_i A_{i-1} P_i$
Setting up multigrid hierarchies with MueLu

Features:

- MueLu implements all building blocks for state-of-the-art aggregation-based AMG methods
  - Parallel aggregation algorithms
  - Transfer operators
  - Level smoothers
  - Rebalancing

- **Flexibility** through modularity
  - Strict splitting of data and algorithms
  - Advanced software patterns (e.g. Factory classes,...)

- **Usability** through use of xml files
  - Easy-to-use xml format (for beginners and advanced users)
  - The multigrid hierarchy is defined at runtime through xml files

Factory concept

Data processing

- Each building block is represented by a factory
- Each factory knows which input is needed to produce the corresponding output
- Each factory requests one or more input variables
- Each factory produces one or more output variables

Data storage

- All input and output data is (temporarily) stored in data container classes
- There is one data container for each multigrid level
- The hierarchical data dependencies are automatically handled by MueLu
Hierarchical setup process

### Aggregation

Phase I

Phase II

Phase III

### Build level smoother

Use $A_{\ell+1}$ to build fine level smoother

### Prolongator smoothing

Generate $P_{\ell+1}$

### Generate restriction $R_{\ell+1}$

Galerkin product $R_{\ell+1} A_{\ell} P_{\ell+1}$

### Factory collection

```plaintext
--- Factory collection ---

ParameterList name="Factories"

--- Note that ParameterLists must be defined prior to being used ---

--- sub block factories ---

ParameterList name="mySubBlockFactory1"
  Parameter name="block col" type="int" value="0"
  Parameter name="block row" type="int" value="0"

ParameterList name="myAgFact1"
  Parameter name="factory" type="string" value="UncoupledAggregationFactory"
  Parameter name="MaxNodesPerAggragate" type="int" value="12"

ParameterList name="myCoarseMap1"
  Parameter name="factory" type="string" value="CoarseMapFactory"
  Parameter name="striped block id" type="int" value="0"

ParameterList name="myTentativeFact1"
  Parameter name="factory" type="string" value="TentativeFactory"
  Parameter name="Aggregates" type="string" value="myAgFact1"
  ParameterList name="myCoarseMap1"

ParameterList name="mySubBlockFactory2"
  Parameter name="sub block factory name" type="string" value="mySubBlockFactory1"
  Parameter name="null space factory name" type="string" value="null space factory name"
```

FactoryManager

- It is very difficult to define all inter-factory dependencies by hand.
- A FactoryManager makes default choices for missing inter-factory dependencies.
- The default information provided by the FactoryManager may or may not be optimal.

Example

- The user declares a TentativePFObject object which needs Aggregates as input.
- The user does not declare an AggregationFactory which produces Aggregates.
- The FactoryManager provides a default AggregationFactory which is used by the TentativePFObject.
Hierarchical xml files

- Use hierarchical XML files to exactly describe the dependency graph of the factories
- Details on the hierarchical XML files can be found in the MueLu tutorial (chapters 6-11)
- Use the hierarchical framework to plug-in your application-specific factories in the existing framework
Minimal example

Hierarchical XML file for 5 level AMG with symmetric Gauss-Seidel level smoother:

```xml
<ParameterList name="MueLu">
<!-- Factory collection -->
<ParameterList name="Factories">

<!-- Sym.Gauss-Seidel -->
<ParameterList name="Sym.Gauss-Seidel">
  <Parameter name="factory" type="string" value="TrilinosSmoother"/>
  <Parameter name="type" type="string" value="RELAXATION"/>
  <ParameterList name="ParameterList">
    <Parameter name="relaxation: type" type="string" value="Symmetric Gauss-Seidel"/>
    <Parameter name="relaxation: sweeps" type="int" value="1"/>
    <Parameter name="relaxation: damping factor" type="double" value="0.7"/>
  </ParameterList>
</ParameterList>

</ParameterList>

<!-- Definition of the multigrid preconditioner -->
<ParameterList name="Hierarchy">
  <Parameter name="max levels" type="int" value="5"/>
  <Parameter name="coarse: max size" type="int" value="1000"/>
  <Parameter name="verbosity" type="string" value="High"/>
  <ParameterList name="AllLevel">
    <Parameter name="Smoother" type="string" value="Sym.Gauss-Seidel"/>
  </ParameterList>
</ParameterList>

</ParameterList>
</ParameterList>
```

Tobias Wiesner
MueLu - AMG Design and Extensibility
EuroTUG 2015 8 / 25
Natural interface

Natural XML file for 5 level AMG with symmetric Gauss-Seidel level smoother:

```
<ParameterList name="MueLu">
  <Parameter name="max levels" type="int" value="5"/>
  <Parameter name="coarse: max size" type="int" value="1000"/>
  <Parameter name="verbosity" type="string" value="high"/>
  <Parameter name="smoother: type" type="string" value="RELAXATION"/>
  <ParameterList name="smoother: params">
    <Parameter name="relaxation: type" type="string" value="Symmetric Gauss-Seidel"/>
    <Parameter name="relaxation: sweeps" type="int" value="1"/>
    <Parameter name="relaxation: damping factor" type="double" value="0.7"/>
  </ParameterList>
</ParameterList>
```

Natural versus hierarchical XML files:
- Hierarchical XML parameter file longer than natural XML file
- Full flexibility with hierarchical XML file
Demonstration

- Build special aggregates by modifying input for aggregation algorithm
- Create a special filter factory
  - Input: fine level matrix $A_\ell$, splitting information $M, S$
  - Output: modified matrix $A^f_\ell$
- Use filtered $A^f_\ell$ as input for aggregation and transfer operator smoothing (but not for level smoothers)
- Transfer splitting information using a user-provided transfer factory
Extend setup phase by new factories

- Filtering $A_{\ell}$
  - $G(A_{\ell}^T) \rightarrow A_{\ell}'$
  - Aggregation $A_{\ell}$
- Generate $\hat{P}_{\ell+1}$
  - Build $B_{\ell+1}$ by local QR-decomposition of $B_{\ell}$ $\hat{P}_{\ell+1}$
- Prolongator smoothing $P_{\ell+1}$
  - Generate $p_{\ell+1}$
- Galerkin product $R_{\ell+1} A_{\ell} P_{\ell+1}$
- Build level smoother
  - Use $A_{\ell+1}$ to build fine level smoother
- Generate restriction $R_{\ell+1}$
- Transfer user data $A_{\ell+1}$

$B_{\ell}$ $A_{\ell}$ $P_{\ell}$ $R_{\ell}$ $M_{\ell}$ $S_{\ell}$
Computational contact mechanics

- Two-solid bodies example
- Aggregates should not cross contact interface
- Modify input for aggregation algorithm accordingly

Standard aggregation technique: aggregates cross the contact interface

Adapted aggregation technique: aggregates do not cross the contact interface

C++ API in MueLu
C++ interface

```cpp
Hierarchy H(fineA); // generate hierarchy using fine level matrix
H.Setup(); // call multigrid setup (create hierarchy)
H.Iterate(B, nIts, X);// perform nIts iterations with multigrid algorithm (V-Cycle)
```

- Uses reasonable defaults
- Generates smoothed aggregation AMG
C++ interface

Hierarchy H(fineA);  // generate hierarchy using fine level
       // matrix

RCP<TentativePF<FactoryManager> PFact = rcp(new TentativePF<FactoryManager>());
FactoryManager M;     // construct factory manager
M.SetFactory("P", PFact); // define tentative prolongator
       // factory as default factory for
       // generating P
H.Setup(M);          // call multigrid setup (create hierarchy)

H.Iterate(B, nIts, X); // perform nIts iterations with multigrid
       // algorithm (V-Cycle)

Generates unsmoothed aggregation AMG
C++ interface

```cpp
Hierarchy H(fineA); // generate hierarchy using fine level matrix

Teuchos::ParameterList smootherParams;
smootherParams.set("chebyshev: degree", 3);
RCP<SmootherPrototype> smooProto =
    rcp(new TrilinosSmoother("CHEBYSHEV", smootherParams));
RCP<SmootherFactory> smooFact =
    rcp(new SmootherFactory(smooProto));
FactoryManager M;
M.SetFactory("Smoofer", smooFact);

H.Setup(M); // call multigrid setup (create hierarchy)

H.Iterate(B, nIts, X); // perform nIts iterations with multigrid algorithm (V-Cycle)
```

- Generates smoothed aggregation AMG
- Use third degree polynomial smoother
Access MueLu hierarchy data from C++

- Access data containers in `Hierarchy` object
- `Hierarchy::GetNumLevels()` returns number of multigrid levels
- Use keyword `Keep` to access temporary data

```cpp
Hierarchy H(fineA); // generate hierarchy using fine level matrix
H.Setup(); // call multigrid setup (create hierarchy)
```
Access MuELu hierarchy data from C++

- Access data containers in `Hierarchy` object
- `Hierarchy::GetNumLevels()` returns number of multigrid levels
- Use keyword `Keep` to access temporary data

```cpp
Hierarchy H(fineA);  // generate hierarchy using fine level
// matrix
H.Setup();            // call multigrid setup (create hierarchy)

// access data container for level 0 and level 1
RCP<Level> fineLevel = H.GetLevel(0);
RCP<Level> coarseLevel = H.GetLevel(1);
```
Access MuELu hierarchy data from C++

- Access data containers in `Hierarchy` object
- `Hierarchy::GetNumLevels()` returns number of multigrid levels
- Use keyword `Keep` to access temporary data

```cpp
Hierarchy H(fineA); // generate hierarchy using fine level matrix
H.Setup(); // call multigrid setup (create hierarchy)
```

// access data container for level 0 and level 1
```
RCP<Level> fineLevel = H.GetLevel(0);
RCP<Level> coarseLevel = H.GetLevel(1);
```

// extract data (fine level matrix and transfers)
```
RCP<Matrix> A = fineLevel->Get< RCP<Matrix> >("A");
RCP<Matrix> P = coarseLevel->Get< RCP<Matrix> >("P");
RCP<Matrix> R = coarseLevel->Get< RCP<Matrix> >("R");
```

⇒ Use data, e.g., to implement your own V-cycle
Reuse data for several multigrid setups

- Call `Keep` routine for data that you want to keep stored in the level class (and not automatically freed if possible).
- Call `Setup` and use Hierarchy object
- Overwrite input data (fine level $A$)
- Redo `Setup` for new fine level matrix and use (new) Hierarchy
- Do not forget to delete data in level containers as soon as possible to save memory.

```cpp
FactoryManager M;
Hierarchy H(A1);

RCP<Factory> PtentFact = rcp(new TentativePF厂ctory());
M.SetFactory("Ptent", PtentFact);
H.Keep("P", PtentFact.get());

RCP<Factory> AcFact = rcp(new RAPFactory());
M.SetFactory("A", AcFact);
H.Keep("RAP Pattern", AcFact.get());

// first setup call
H.Setup(M);

// -> use H

// Change the problem
RCP<Level> finestLevel = H.GetLevel(0);
finestLevel->Set("A", A2);

// Redo the setup
H.Setup(M);

// -> use H

H.Delete("P", M.GetFactory("Ptent").get());
H.Delete("RAP Pattern", M.GetFactory("A").get());
```
Current research projects (using MueLu)
Current research projects (using MueLu)

- Nonlinear multigrid methods (FAS):
  - Implementation of multigrid FAS scheme
  - Use MueLu aggregation and transfer operators
  - General framework for nonlinear problems based on callback functions (NOX compatible)

- Use MueLu with different types of operators (e.g., high order discretization operators on the finest level and lowest order discretization on coarse levels)

Matthias Mayr, MHPC, TUM
Chris Siefert, SNL
Current research projects (using MueLu)

- MueLu is used within the EQUINOX framework
  - EQUINOX = Environment for quantifying Uncertainty: Integrated aNd Optimized at the eXtreme-scale
  - EQUINOX contains advanced multi-level methods for alleviating the complexity and accelerating the solutions of both deterministic and stochastic extreme-scale solvers
  - MueLu is applied as a preconditioner to an ensemble of samples
  - Use flexibility of Xpetra/Tpetra for a scalar type representing an ensemble (diagonal matrix)
  - Scalability shown up to 512 nodes (BG/Q machine)
  - See also http://equinox.ornl.gov

Prof. M. Gunzburger (FSU)
Prof. J. Wu (Georgia Tech)
C. Webster (ORNL)
R. Archibald (ORNL)
C. Hauck (ORNL)
S. Pannaia (ORNL)
E. Phipps (SNL)
C. Edwards (SNL)
J. Hu (SNL)
S. Rajamanickam (SNL)
and others...
Current research projects (using MueLu)

- General multigrid framework for multiphysics problems
  - Implementation of general multigrid framework for \( n \times n \) block matrices (e.g., FSI problems with constraints, TSI problems, . . . )
  - Use MueLu level smoothers, aggregation methods and single-field transfer operators
  - Own V-cycle implementation and block smoothing strategies

Thank you for your attention
References


