SAND2012-9327P

MueLu – A Flexible, Parallel Multigrid Framework

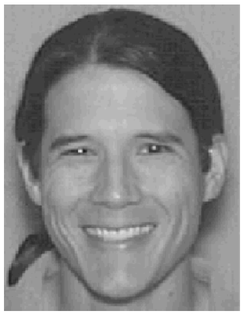
Trilinos User Group Meeting

Oct. 30 – Nov. 1, 2012

Jonathan Hu

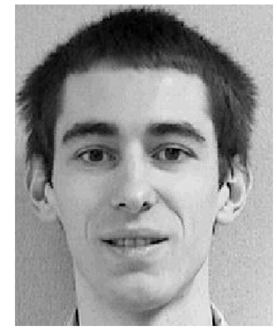
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Development Team



Jonathan Hu

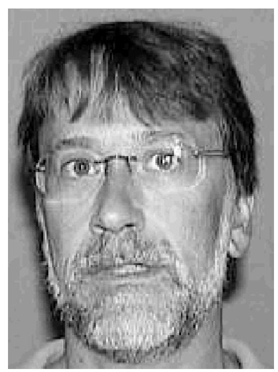
Jeremie Gaidamour



new

Andrey Prokopenko

Tobias Wiesner
(TU Munich)



Ray Tuminaro

Chris Siefert

Outline

- Design and Motivation
- User interfaces
- Case study: smoothed aggregation
  - Reuse possibilities

Design and Motivation

Motivation for a New Multigrid Library

- Trilinos already has mature multigrid library, ML
  - Algorithms for Poisson, Elasticity, Petrov-Galerkin, H(curl), H(div)
  - Algorithms have been exercised extensively.
  - Broad user base
- However …
  - ML weakly linked to other Trilinos capabilities (e.g., smoothers)
  - C-based, only scalar type “double” supported explicitly
  - Over 50K lines of source code
    - Maintainability, extensibility

Sandia National Laboratories

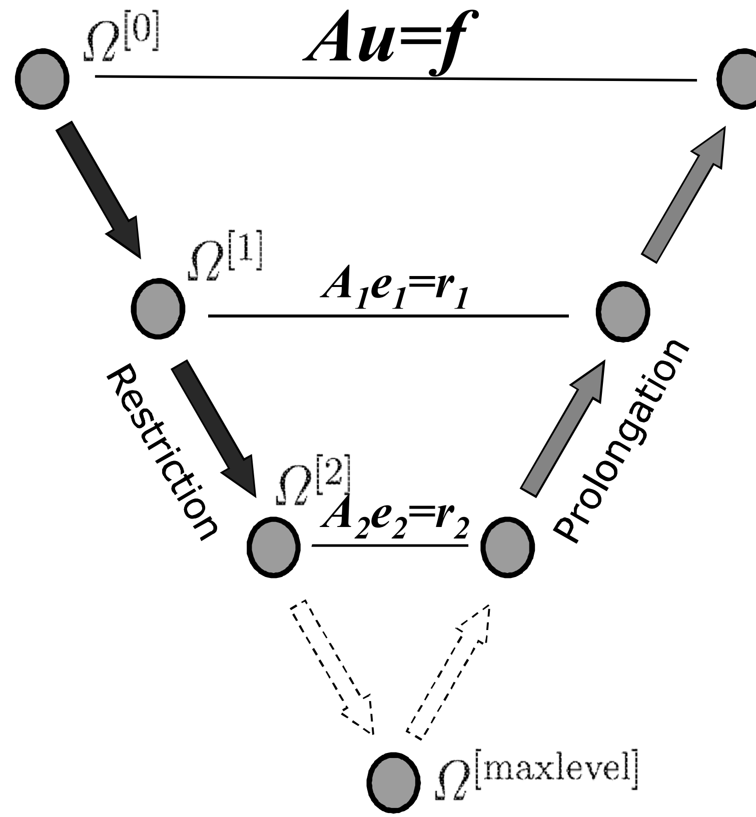
Objectives for New Multigrid Framework

- Templating on scalar, ordinal types
- Advanced architectures
  - Kokkos support for various compute node types
    - Hybrid parallelism: MPI, MPI+threads, MPI+MPI
    - GPUs eventually
- Extensibility
  - Facilitate development of other algorithms
    - Energy minimization methods
    - Geometric, classic algebraic multigrid, …
  - Ability to combine several types of multigrid
- Preconditioner reuse
  - Reduce setup expense

Sandia National Laboratories

Multigrid Basics

- **Two main components**
  - **Smoothers**
    - **Approximate solves on each level**
    - **"Cheaply" reduces particular error components**
    - **On coarsest level, smoother = A_i^{-1} (usually)**
  - **Grid Transfers**
    - **Moves data between levels**
    - **Must represent components that smoothers can't reduce**
- **Algebraic Multigrid (AMG)**
  - **AMG generates grid transfers**
  - **AMG generates coarse grid A_i's**



Sandia National Laboratories

Current MueLu Capabilities

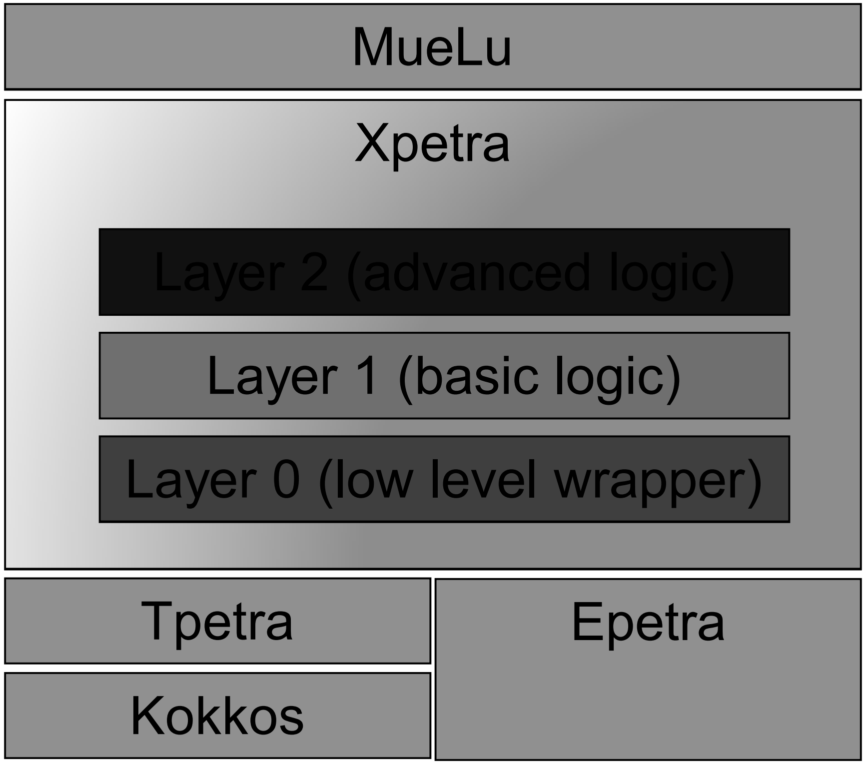
- Grid Transfer Algorithms
  - Smoothed aggregation, Petrov Galerkin
- Smoothers
  - SOR, ILU, Polynomial (Ifpack, Ifpack2)
- Direct solvers
  - KLU, SuperLU, SuperLUDist (Amesos, Amesos2)
- Sparse linear algebra (Epetra, Tpetra)
- Krylov acceleration (Belos, AztecOO)

Xpetra

- **Wrapper for Epetra and Tpetra**
  - **Based on Tpetra interfaces**
  - **Allows unified access to either linear algebra library**

- **Layer concept:**
  - **Layer 2: blocked operators**
  - **Layer 1: operator views**
  - **Layer 0: low level E/Tpetra wrappers (automatically generated code)**

- **MueLu algorithms are written using Xpetra**



Sandia National Laboratories

Design Overview

- MueLu makes heavy use of “factory” pattern
  - Factories: classes that generate objects
- Preconditioner is created by chaining together factories that create grid transfers, smoothers, coarse grid Galerkin triple-matrix product
- FactoryManager manages these dependencies
- User is not required to specify these dependencies (or even know they exist).

User Interfaces

MueLu – User Interfaces

- MueLu can be customized as follows:
  - XML input files
  - Parameter lists (key-value pairs)
  - Directly through C++ interfaces

- New/casual users
  - Minimal interface
  - Sensible defaults provided automatically

- Advanced users
  - Can customize or replace any component of multigrid algorithm.

MueLu – A Simple C++ Example

```
// Creation of fine matrix A, solution X, right-hand side B not shown

// Allocate hierarchy object and insert A
Hierarchy H(fineA);

H.Setup();

H.Iterate(B,nits,X);
```

- **Generates smoothed aggregation multigrid preconditioner.**
- **Uses reasonable defaults.**
- **As we'll see, these can changed easily.**

Customizing the Preconditioner

```
// Creation of fine matrix A, solution X, right-hand side B not shown

// Allocate hierarchy object and insert A
Hierarchy H(fineA);

RCP<TentativePFactory> ProlongatorFact = rcp( new TentativePFactory() );
Teuchos::ParameterList smootherParamList;
smootherParamList.set("Chebyshev: degree", 3);
RCP<SmootherPrototype> smootherPrototype     = rcp( new TrilinosSmoother("Chebyshev
", smootherParamList) );

FactoryManager M;
M.SetFactory("P",ProlongatorFact);
M.Set("Smoother",SmootherPrototype);

H.Setup(M);

int its=10;
H.Iterate(B,nits,X);
```

Customizing the Preconditioner

```
// Creation of fine matrix A, solution X, right-hand side B not shown

// Allocate hierarchy object and insert A
Hierarchy H(fineA);

RCP<TentativePFactory> ProlongatorFact = rcp( new TentativePFactory() );
Teuchos::ParameterList smootherParamList;
smootherParamList.set("Chebyshev: degree", 3);
RCP<SmootherPrototype> smootherPrototype     = rcp( new TrilinosSmoother("Chebyshev
', smootherParamList) );

FactoryManager M;
M.SetFactory("P",ProlongatorFact);
M.Set("Smoother",SmootherPrototype);

H.Setup(M);

int its=10;
H.Iterate(B,nits,X);
```

- Use unsmoothed prolongator
  - Rcp == smart pointer

Customizing the Preconditioner

```
// Creation of fine matrix A, solution X, right-hand side B not shown

// Allocate hierarchy object and insert A
Hierarchy H(fineA);

RCP<TentativePFactory> ProlongatorFact = rcp( new TentativePFactory() );
Teuchos::ParameterList smootherParamList;
smootherParamList.set("Chebyshev: degree", 3);
RCP<SmootherPrototype> smootherPrototype     = rcp( new TrilinosSmoother("Chebyshev
", smootherParamList) );

FactoryManager M;
M.SetFactory("P",ProlongatorFact);
M.Set("Smoother",SmootherPrototype);

H.Setup(M);

int its=10;
H.Iterate(B,nits,X);
```

- Use degree 3 polynomial smoother
  - Parameter list == key/value pairs
  - Smoother prototype

Sandia National Laboratories

Customizing the Preconditioner

```
// Creation of fine matrix A, solution X, right-hand side B not shown

// Allocate hierarchy object and insert A
Hierarchy H(fineA);

RCP<TentativePFactory> ProlongatorFact = rcp( new TentativePFactory() );
Teuchos::ParameterList smootherParamList;
smootherParamList.set("Chebyshev: degree", 3);
RCP<SmootherPrototype> smootherPrototype     = rcp( new TrilinosSmoother("Chebyshev
', smootherParamList) );

FactoryManager M;
M.SetFactory("P",ProlongatorFact);
M.Set("Smoother",SmootherPrototype);

H.Setup(M);

int its=10;
H.Iterate(B,nits,X);
```

- **Register changes with Factory Manager and pass to Setup.**

Sandia National Laboratories

The Factory Manager

- **Holds default factories to be used during multigrid setup.**
- **Can have one** FactoryManager **per level.**
- **User can selectively specify alternatives.**

  FactoryManager M;

  M.SetFactory("Aggregation",UCAggFact);
- **The hierarchy set up process queries the** FactoryManager **for proper factory for each algorithmic component.**

Accessing MueLu Through XML

```
//read in XML file...

ParameterListInterpreter mueLuFactory(xmlFileName);
RCP<Hierarchy> H = mueLuFactory.CreateHierarchy();
H->GetLevel(0)->Set("A", A);

mueLuFactory.SetupHierarchy(*H);

int nIts = 10;
H->Iterate(*B, nIts, *X);
```

```
<ParameterList name="MueLu">
    <Parameter name="numDesiredLevel" type="int" value="10"/>
    <Parameter name="maxCoarseSize" type="int" value="500"/>

    <ParameterList name="FineLevel">
       <Parameter name="startLevel" type="int" value="0"/>
       <Parameter name="Smoother"   type="string" value="Chebyshev"/>
       <Parameter name="Aggregates" type="string" value="UCAggregationFactor
    </ParameterList>

    <ParameterList name="CoarsestLevel">
       <Parameter name="startLevel"   type="int" value="-1"/>
       <Parameter name="CoarseSolver" type="string" value="DirectSolver"/>
    </ParameterList>
</ParameterList>
```

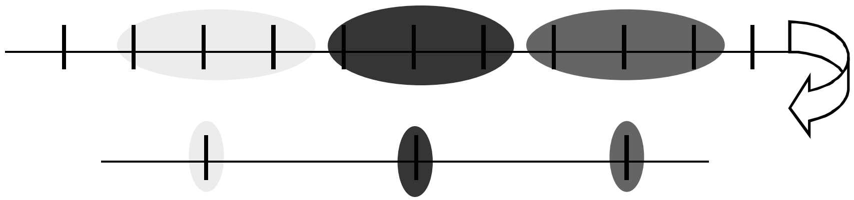
Sandia National Laboratories

Case Study: Smoothed Aggregation Multigrid

Sandia National Laboratories

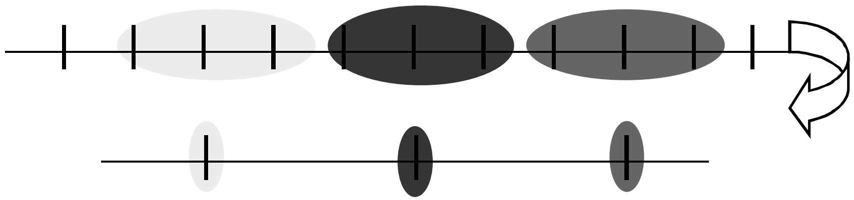
Smoothed Aggregation Setup

- **Group fine unknowns into *aggregates* to form coarse unknowns**



Smoothed Aggregation Setup

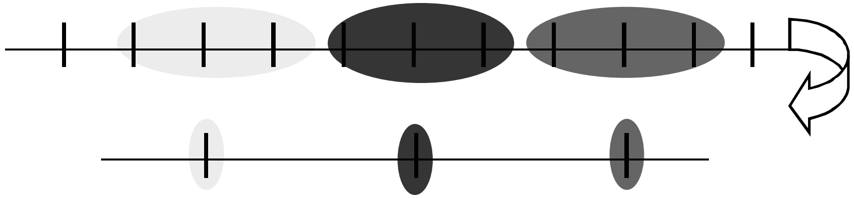
- Group fine unknowns into *aggregates* to form coarse unknowns



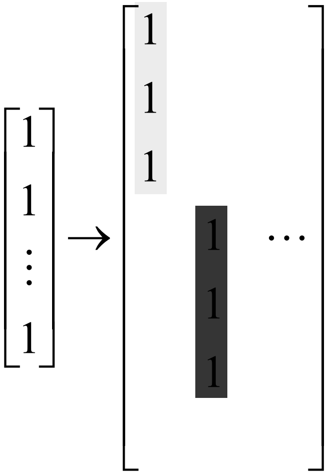
- Partition given nullspace $B_{(h)}$ across aggregates to have local support

$$\begin{bmatrix}1\\1\\\vdots\\1\end{bmatrix} \rightarrow \begin{bmatrix}1 & & \\ 1 & & \\ 1 & & \\ & 1 & \cdots \\ & 1 & \\ & 1 & \\ & & \end{bmatrix}$$

Smoothed Aggregation Setup



- **Group fine unknowns into *aggregates* to form coarse unknowns**

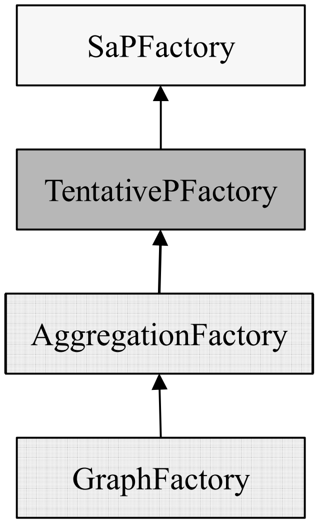


- **Partition given nullspace $B_{(h)}$ across aggregates to have local support**
- **Calculate $QR=B_{(h)}$ to get initial prolongator P^{tent} (=Q) and coarse nullspace (R).**
- **Form final prolongator** $P^{sm} = (I - \omega D^{-1}A)P^{tent}$

Sandia National Laboratories

Case Study: Smoothed Aggregation

Dependency Graph



- **Possible call sequences to generate P^{sm}**

```
1) PFact = SaPFactory();

2) PtentFact = TentativePFactory();
   PFact = SaPFactory(PtentFact);

3) AggFact = AggregationFactory();
   Ptent = TentativePFactory(AggFact);
   PFact = SaPFactory(Ptent);
```

- **Data dependencies must be maintained between factories.**

Management of Data Dependencies

- Level **class manages data storage**
- **Factories exchange data by taking** Level **classes as arguments to** Build **method:**
  - Build(currentLevel) **or**
  - Build(fineLevel,coarseLevel)
- **Factories declare on** Level **the data that they require, along with generating factories, or** FactoryManager **provides generating strategy.**

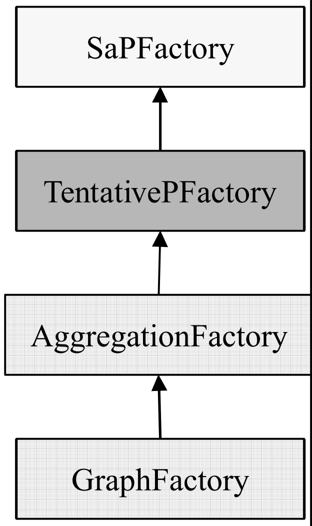
Advantages of Data Management on Level

- Level **manages data deallocation once all requests satisfied**
- **Generating factory does not need to know what other factories require data**
- **Data reuse**
  - **Any data (aggregates, *P*, …) can be retained by user request for reuse in later runs.**
  - **Data can be retained for later analysis.**
  - **Almost any reuse granularity is possible.**

Sandia National Laboratories

Example: Smoothed Aggregation

```
AggFact = AggregationFactory();
Ptent = TentativePFactory(AggFact);
Pfact=SaPFactory(Ptent);
```



- Pfact **registers with** Level **its need for** P^{tent}**, along with generating factory** Ptent.
- Ptent **registers with** Level **its need for aggregate data, along with generating factory** (AggFact)
- AggFact **generates aggregates, stores on** Level.
- **After** Ptent **accesses aggregates,** Level **frees data.**
- **After** Pfact **access** P^{tent}, Level **frees data.**

User does not need to manage data dependencies.

Sandia National Laboratories

Summary

- Current status
  - Copyrighted with open-source BSD style license
  - Part of publicly available Trilinos anonymous clone
  - We still support ML.
- Ongoing/Future work
  - Grid transfers based on constrained minimization (aka energy minimization)
  - Improving documentation, application interfaces
  - Big driver for FY13 is templated stack milestone requirements
  - Performance optimizations

Sandia National Laboratories