Refactoring Amanzi-ATS to leverage Tpetra/Kokkos abstractions for heterogeneous architectures

Julien Loiseau, David Moulton, Ethan Coon and Konstantin Lipnikov

Office of Biological and Environmental Research
Earth and Environmental Systems Science Division
Data Management Program

LA-UR-21-31661

Environmental applications

- Climate impacts and feedbacks (carbon and nitrogen cycling)
- Contaminant transport and reactions
- Complex interaction of land surface and subsurface processes

ExaSheds

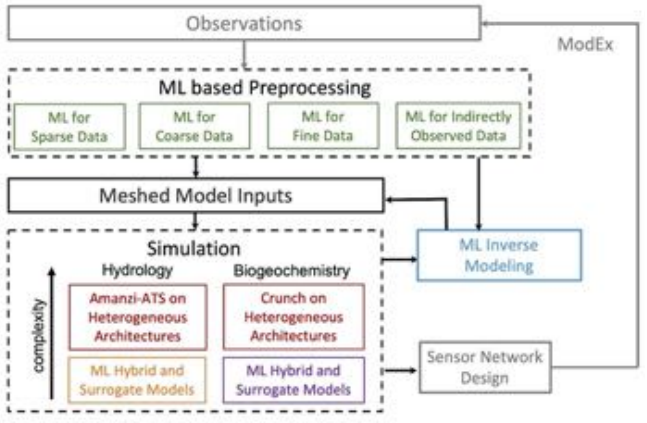
Advancing Watershed System Science using Machine Learning and Data-Intensive Simulation

Scientific Challenge

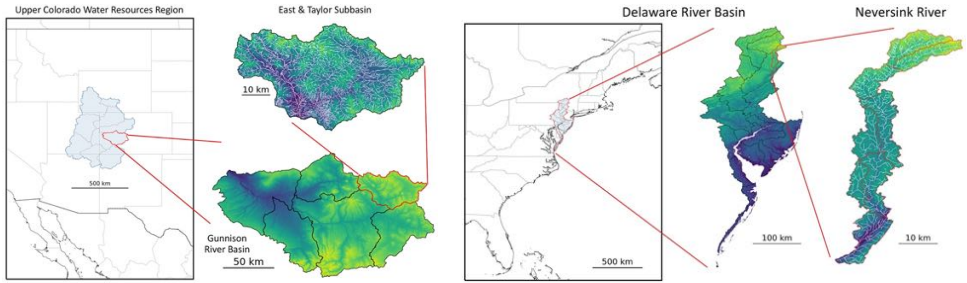
- Develop approaches for machine learning-assisted physics-based simulation of watersheds and river basins.
- Adapt DOE-developed watershed simulation tools to leadership-class computer architectures (GPUs).

Approach

- Work with data from East River Watershed, CO, Upper Colorado Water Resources Region and Delaware River Basin.
- Develop model inputs from sparse, coarse and indirectly related information
- Hybridization of process-resolving and data-driven ML simulations
- Refactor Amanzi-ATS using Tpetra/Kokkos abstractions for heterogeneous architectures.



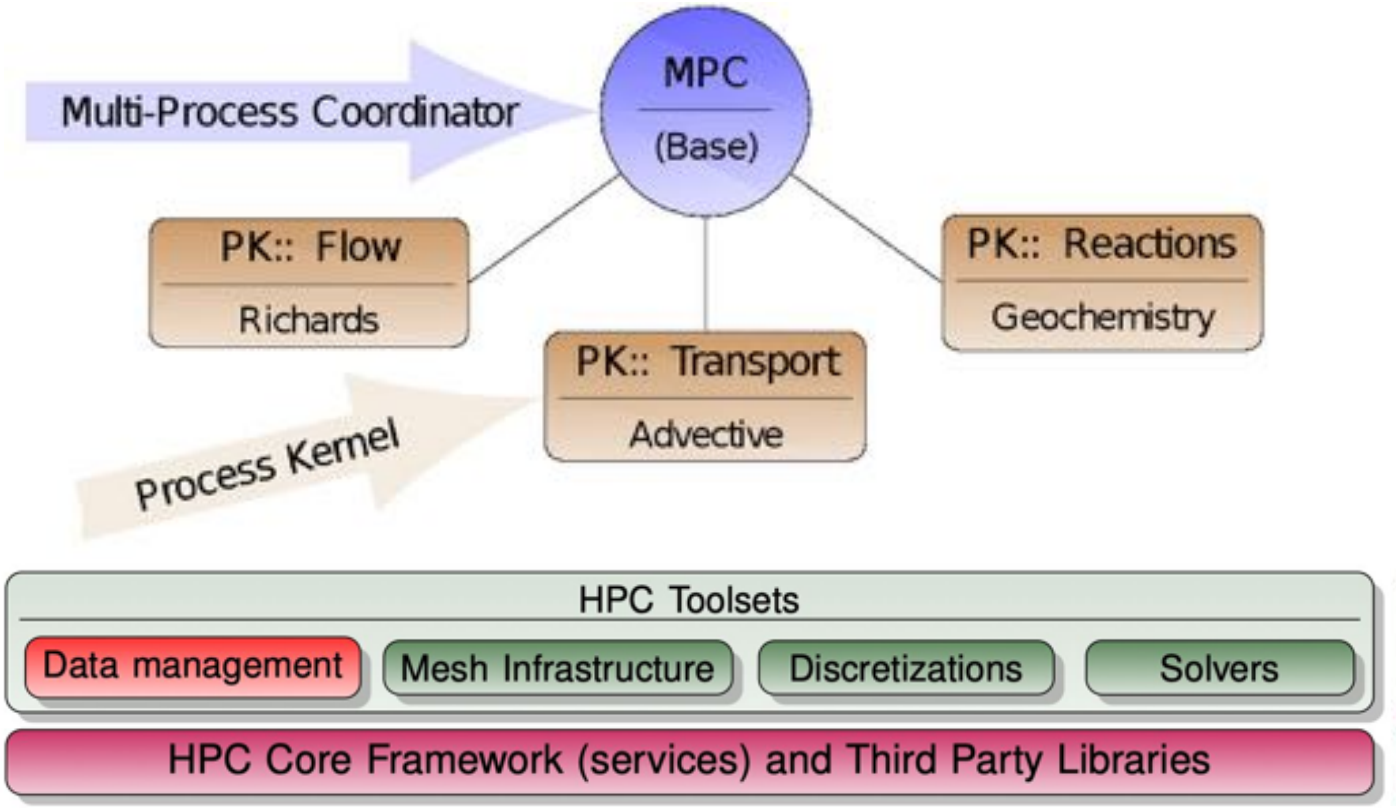
ML-Assisted Simulation Workflow



Upper Colorado River Basin

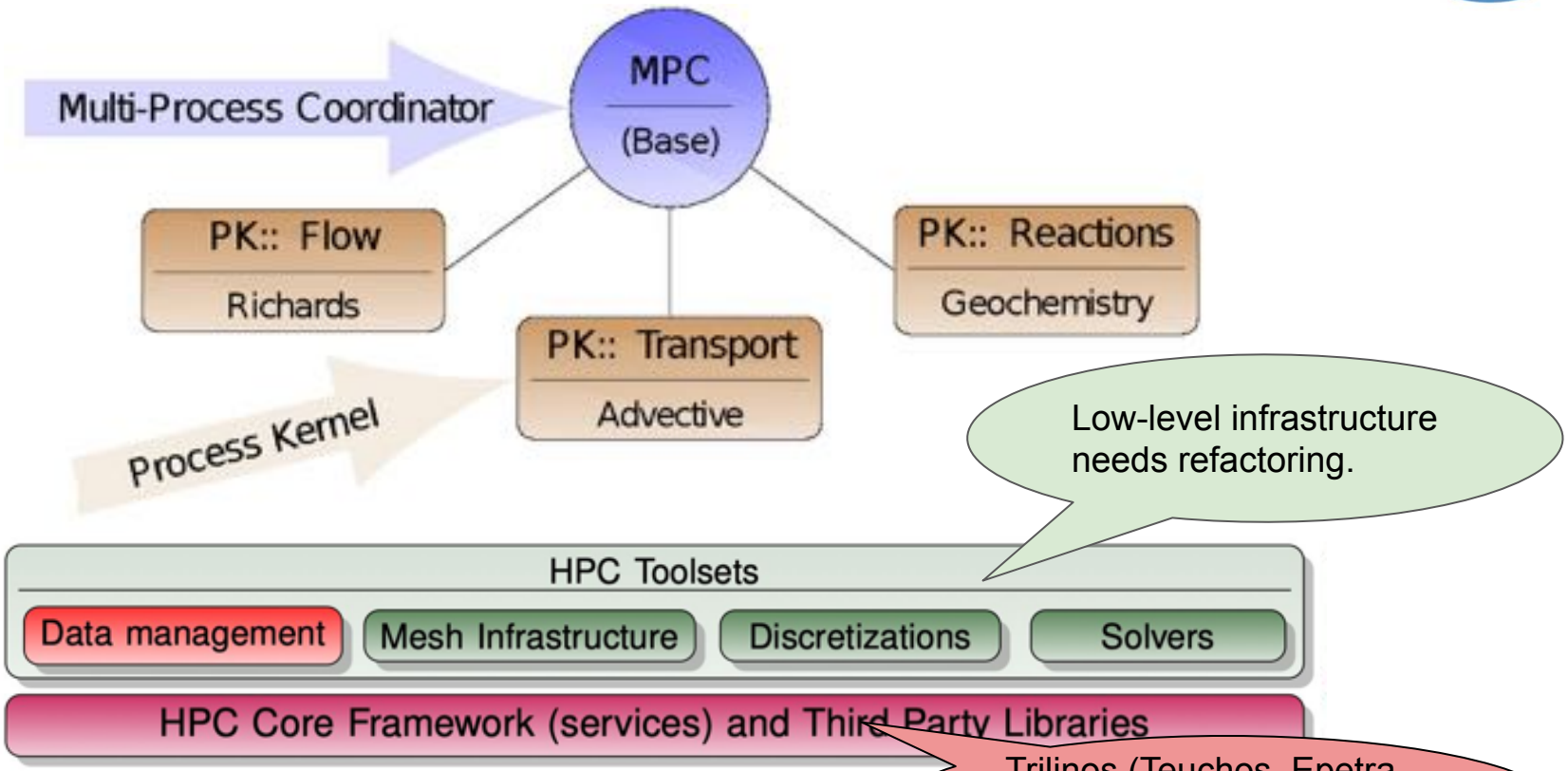
Delaware River Basin

Amanzi-ATS: Application-Centric High-Level Design



The hierarchical and modular design of Amanzi reflects the steps in translating a conceptual model to a numerical model, producing output for analysis.

Amanzi-ATS: Application-Centric High-Level Design



The hierarchical and modular design of Amanzi reflects the step
model to a numerical model, producing output for analysis.

Amanzi-ATS from Epetra to Tpetra

The stable (MPI-only) version of Amanzi-ATS is based on Epetra.
The aim of this task is to migrate Amanzi-ATS toward Tpetra to target heterogeneous architectures.

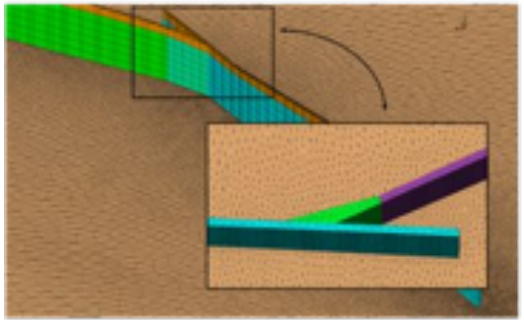
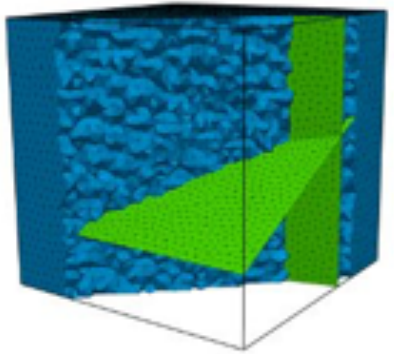
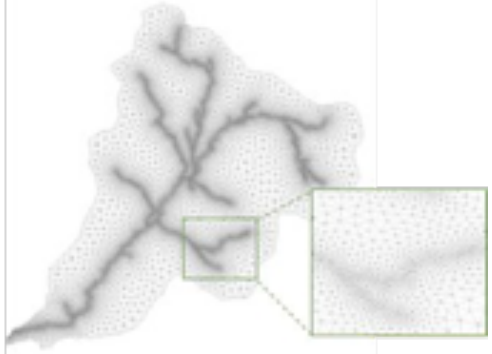
- Generic changes of *Epetra::X* structures to *Tpetra::X*
- In depth change to use *Kokkos*'s data structures and paradigms to target OpenMP and GPU.

Preserving the flexibility of the multiphysics framework and application capabilities relies on our ability to refactor low-level support for unstructured meshes, advanced discretizations and solvers.

https://github.com/amanzi/amanzi/tree/tpetra

Mesh

Amanzi provides a flexible and powerful mesh infrastructure for unstructured polyhedral meshes (currently builds on MSTK or Moab).

- In the stable version a mesh cache was added to improve performance. It did not hold all the mesh data, and its entries are filled only when first accessed.
- The cache approach is vital for a GPU implementation since most of the lazy functions are host only (I/O).

Mesh

- Force the cache evaluation before its utilization.
- Data generated on the Host and copied on Device.

The initial cache was represented using *std* containers and had to be adapted to be usable with Kokkos.

```
std::vector<T>                  ->   Kokkos::View<T*>
std::vector<std::vector<T>>     ->   Kokkos::crs<T*>
```

Accessors were then added to extract the information from the Kokkos data structures.

Mesh

Example, accessor for the faces of a cell (*std::vector<std::vector>>*):

Stable:

```
void cell_get_faces(
    const Entity_ID c,
    std::vector<Entity_ID> *faceids) const
{
  if (!cell2face_info_cached_) cache_cell2face_info_();

  std::vector<Entity_ID> &cfaceids = cell_face_ids_[c];
  *faceids = cfaceids; // copy operation
  }
}
```

Tpetra:

```
KOKKOS_INLINE_FUNCTION void
  cell_get_faces(const Entity_ID cid,
                 Kokkos::View<Entity_ID*>& faceids) const
  {
    assert(cell2face_info_cached_);
    faceids =
      Kokkos::subview(cfi_.entries,
                      Kokkos::make_pair(cfi_.row_map(cid),
                                        cfi_.row_map(cid + 1)));
  }
```

Discretization

The discretization in Amanzi is composed of two libraries:

- Low-level: **Whetstone** for dense matrices, dense vectors and tensors, which are used for elemental matrices in discretizations (FV, MFD, etc.)
  - Flexible approach for better memory handling on CPU
  - **Not suitable for GPU** implementation:
    - Expensive memory allocation, memory transfers and non-contiguous accesses
- High-level: **Operators** for global matrices and assembly
  - Dependent on the changes of Whetstone

Solution:

- Initialize the structure at the beginning of the run
- Recreate the objects as needed from contiguous memory

Discretization: Whetstone

Example: Tensors

```
struct Tensor{
    /*...*/
    int d_, rank_, size_;
    double * data;
};
```

- Stored as Array of Structures: `std::vector<Tensor> tensors;`
- The data pointer can be allocated at any time and ownership can change

Problems:

- Hard to copy on GPU efficiently (one copy per matrix)
- Hard to access on GPU (Matrices potentially not contiguous)

Discretization: Whetstone

Kokkos version:

```
template<class MEMSPACE = DefaultHostMemorySpace>
class Tensor {
  /* ... */
  int d_, rank_, size_;
  Kokkos::View<double*,MEMSPACE> data_;
};
```

Now using CRS to store Tensor arrays.
The fields *d_*, *rank_*, *size_* and *data_* are retrieved from a common CRS structure.

Discretization: Whetstone

CRS stores data/sizes of Tensors:

By using DualView, the data can be initialized on Host or Device and then transferred from one to the other as needed.

The object *i* can be retrieved from *entries_* at indices *row_map_[i]* on the target memory space.

```
template<typename T, int D, class MEMSPACE>
class CSR{
  /* Number of dimension for size */
  static constexpr int dim = D;
  using memory_space = MEMSPACE;
  /* Indices: number of element +1 */
  Kokkos::DualView<int*,MEMSPACE> row_map_;
  /* Data of object */
  Kokkos::DualView<T*,MEMSPACE> data_;
  /* Represents the sizes for matrices/tensors/vectors */
  Kokkos::DualView<int**,MEMSPACE> sizes_;
};
```

Discretization: Whetstone

A TensorVector is based on CRS and provides access the stored elements.

Each object has its own interface:

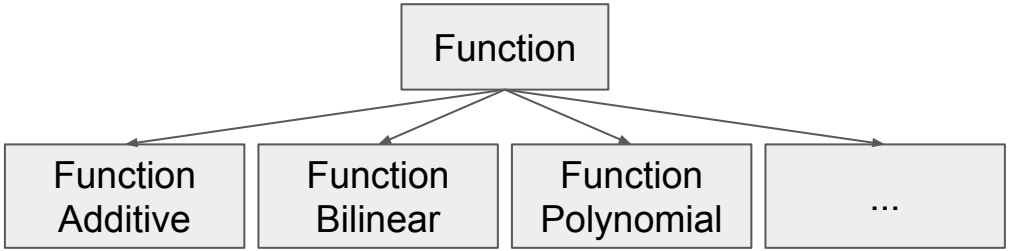
- TensorVector
- DenseMatrix_Vector
- DenseVector_Vector

```
template<class MEMSPACE>
struct TensorVector {
  /* ... */

KOKKOS_INLINE_FUNCTION
  Tensor<DOMS> operator[](const int& i) {
    return std::move(Tensor<DOMS>(d.at(i), d.size(i,0),
                                  d.size(i,1), d.size(i,2)));
  }

  /* ... */

  CSR<double,2,MEMSPACE> d;
  CompositeVectorSpace map;


  bool ghosted;
  bool inited;
};
```

Virtual Functions

User-defined functions add flexibility to model specification in input files.

- Defining boundary and initial conditions, and source terms.
- Support polynomials, math functions, and composition
- Implemented as a vector of virtual object *Function**
  - Resolution is done via the **virtual table**
  - RHS, LHS passed as references of *std::vector<double>*



Problem: Virtual table is not copied onto the Device.
Solution: Use Host for the resolution and start tasks on the Device inside the *Function*'s core.

Solvers and preconditioners

In order to target families of multi-physics problems, Amanzi features a generic solvers and preconditioners interface, and implement:

Non-linear solvers,

- Nonlinear Krylov Acceleration (NKA)
- Jacobian-free Newton-Krylov (JFNK)

implement backtracking and (process-informed) globalization strategies.

Linear solvers,

- Preconditioned Conjugate Gradient (PCG)
- Generalized Minimal Residual (GMRES)

which rely on **preconditioners** for performance.

These preconditioners are a vital piece of this migration to GPU.

Preconditioners in Amanzi/ATS

In the current stable (MPI-based) code, we rely primarily on the three preconditioners:

- **Jacobi**, provided by internal code
- **Block incomplete LU** factorization with thresholding (block-ILUt), provided by the Trilinos IfPack library
- **Algebraic multigrid (AMG)** preconditioner, provided by the HYPRE preconditioning library, accessed through the Trilinos IfPack interface library.

Each of these preconditioners has different scaling and performance characteristics, and is useful for specific tasks (varying from small tests to large full-scale simulations).

Our goal is to provide three preconditioners on GPUs spanning the performance characteristics of those three preconditioners.

Preconditioners

In the migration to Tpetra we have the following preconditioners compiling and running on CPU/GPU with our interface:

- internal: identity, diagonal (Jacobi)
- Ifpack2: ILUT, RILUK
- Ifpack2 interface: ShyLU (FAST_ILUT), Hypre AMG
- MueLu, Ginkgo

We observe acceleration of Amanzi-ATS with Tpetra compared to Epetra on CPU.

On GPU for a simple test, the internal preconditioner Diagonal is still the fastest on our test application despite a very high number of iterations.

Preconditioners: Status

Preconditioner	Run	Status on Amanzi/ATS Tpetra
Ifpack2: ILUT	Y	Little improvement on GPU compared to single core CPU.
Ifpack2: RILUK	-	FV discretization runs but problem with the MFD discretization: too much memory allocated.
ShyLU (Ifpack2: FAST_ILUT)	Y	Good improvement on GPU compared to single CPU
Hypre: AMG (Ifpack2 interface)	Y	Existing interface was copying the matrix line by line. Fix: matrix copied at once from the CRS matrix interface. (issue #9154)
MueLu	N	Ongoing investigation
Ginkgo	Y	Interface to Kokkos added in Amanzi, tested on CPU but not GPU yet.

Preconditioners: Preliminary results

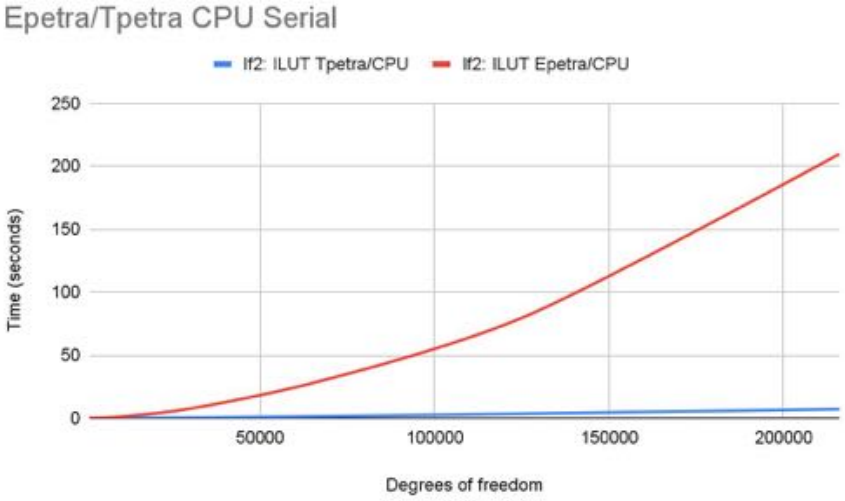
Example: Model Problem

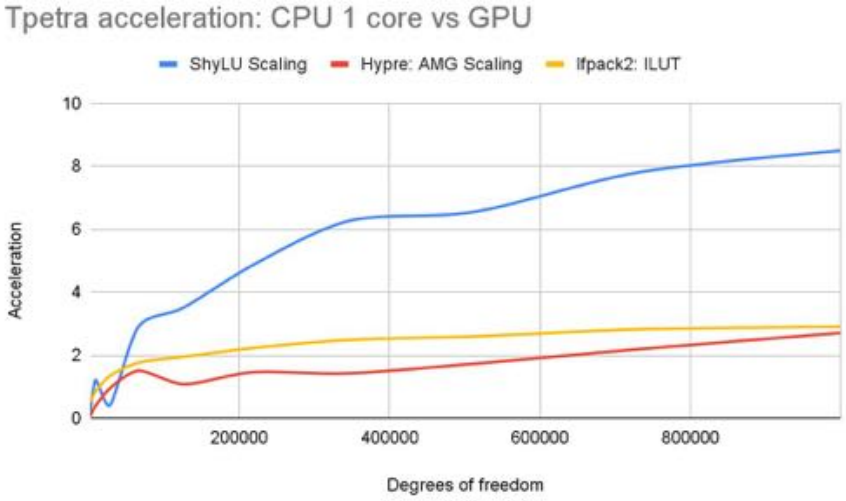
- PDE: Linear steady-state diffusion equation
- Manufactured solution (e.g., various polynomials)
- Boundary conditions: Dirichlet
  - Derived from solution so they are exact
- Discretization: Final Volume
  - Leads to standard 7-point cell-centered discretization
- Mesh type: Structured 3D (orthogonal hexahedral)
- Solver: PCG
- Preconditioner: Uses or is derived from the matrix
- Initial Guess: “zero”.

Preconditioners: Preliminary results

Epetra vs Tpetra implementation:

PCG + X





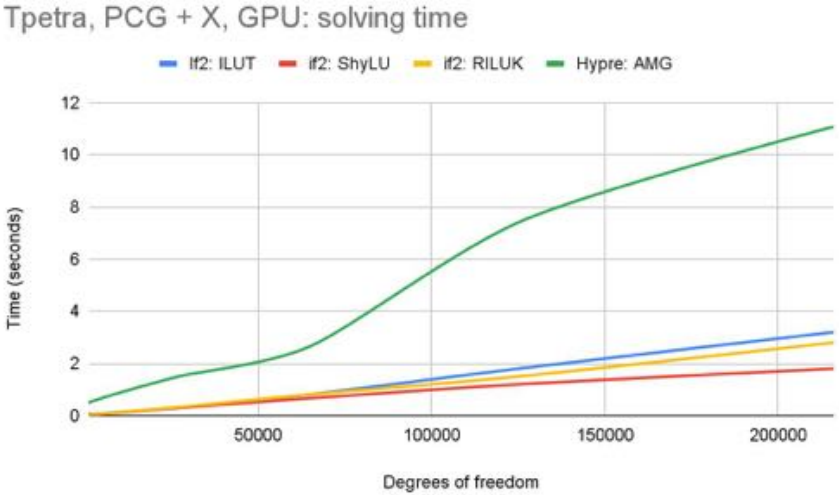
Tpetra implementation is faster than Epetra

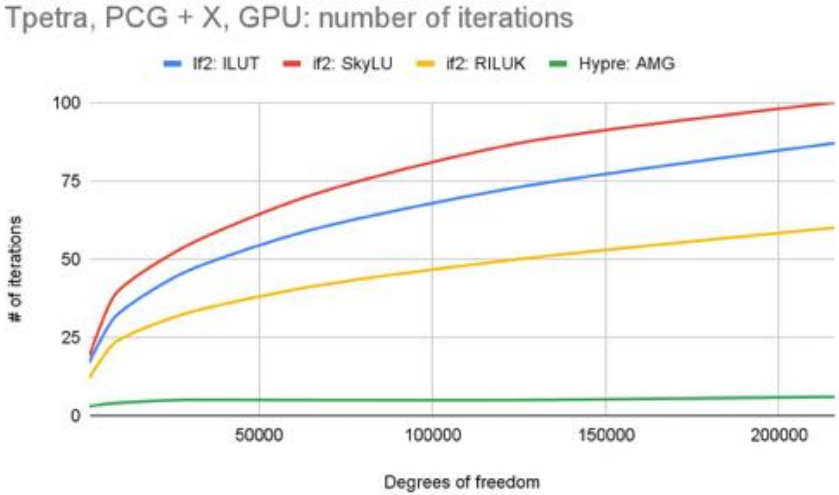
Best is ShyLU with 8x faster on GPU compared to single core CPU

Preconditioners: Preliminary results

Current solvers on GPU:

PCG + X

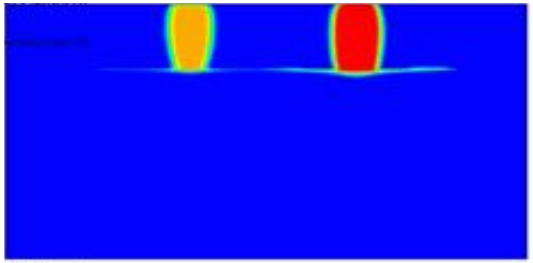




- Hypre is slower but scales optimally, using far fewer iterations than other preconditioners.
- Hypre is more robust (and necessary) on more challenging problems.
- Hypre timing (left) shows GPU memory likely saturated after 40^3 mesh, this might be mitigated by adjusting cycle parameters.

Ongoing work and next steps

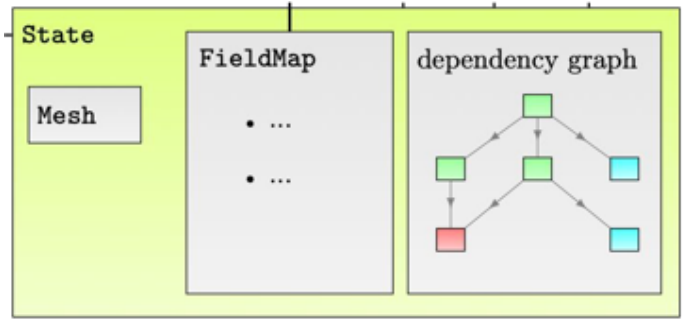
- Finish tests/optimizations on the preconditioners for GPU and compare with OpenMP
- Try to the latest version of Trilinos with no-UVM support on GPU
- Performance analysis on with large MPI multi-GPU
- Add additional, harder tests to better understand preconditioners on more realistic problems*
- Continued evolution of software design for representing physics and complete surface and subsurface PKs.
- Continue to refactor, adding more physics:
  - Coupler integrating surface and subsurface
  - Evapotranspiration, etc



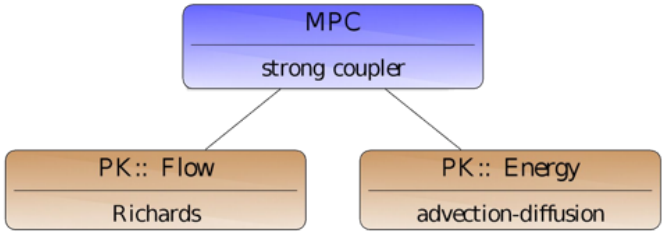
*Next problem for testing: a deep vadose zone infiltration problem that has been benchmarked previously in Amanzi. Above is concentration of a tracer in the corresponding transport problem.

Arcos Multiphysics Framework

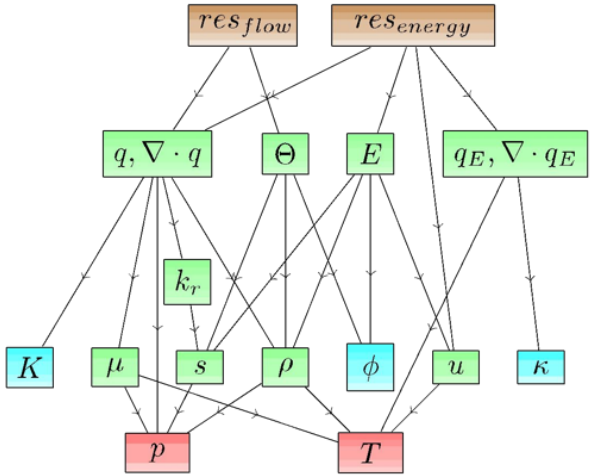
The Arcos multiphysics framework, is a fully functional framework serving Amanzi and ATS, but may also be viewed as a reference implementation to explore interface designs, coupling strategies and outreach to the community.



Dynamic Data Manager



Process Kernel Tree



Dependency Graph

Coon et al. Env. Model. & Soft. 2016