CFD Simulations with Panzer

Trilinos User's Group Meeting 2022

Ryan Glasby

Steve Hamilton

Stuart Slattery

Outline

- Theory
- Code
- Linear Solvers and Preconditioning

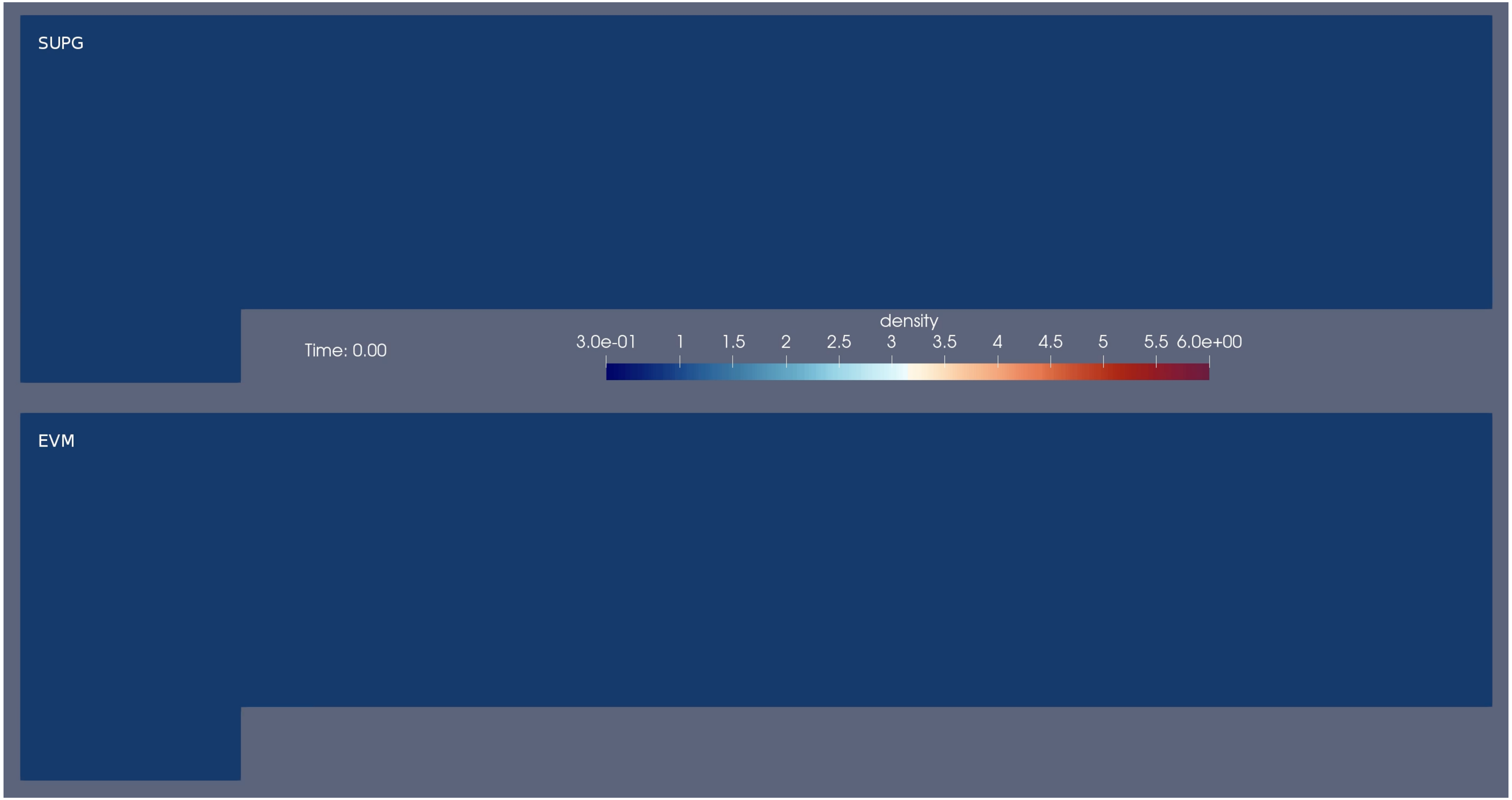
Theory

Ryan Glasby

PDE

- Navier-Stokes/Euler/(Open ended)
- PDE definition: written in weak conservative form evaluated at quadrature points
  - Time derivative, convective operator, viscous operator, source terms
  - Numerical artificial diffusion for convective operators
  - Boundary flux dot with normal vector operators
    - Boundary state supplied -> convective (flux evaluated), viscous (penalty term)
    - Dirichlet/Neumann

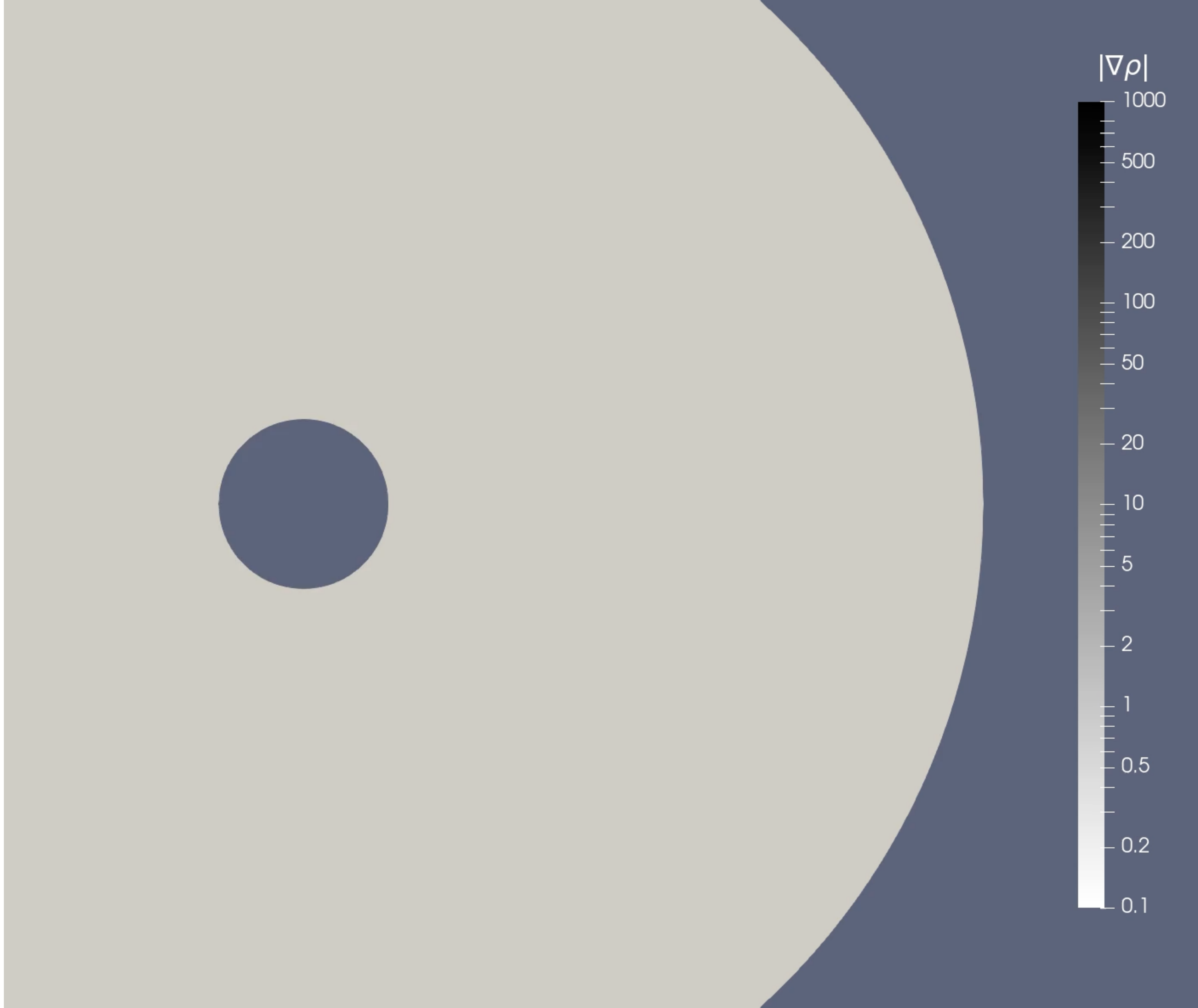
SUPG vs. EVM



Oak Ridge National Laboratory

Time Integration

Mach 17
Gnoffo Cylinder



Code

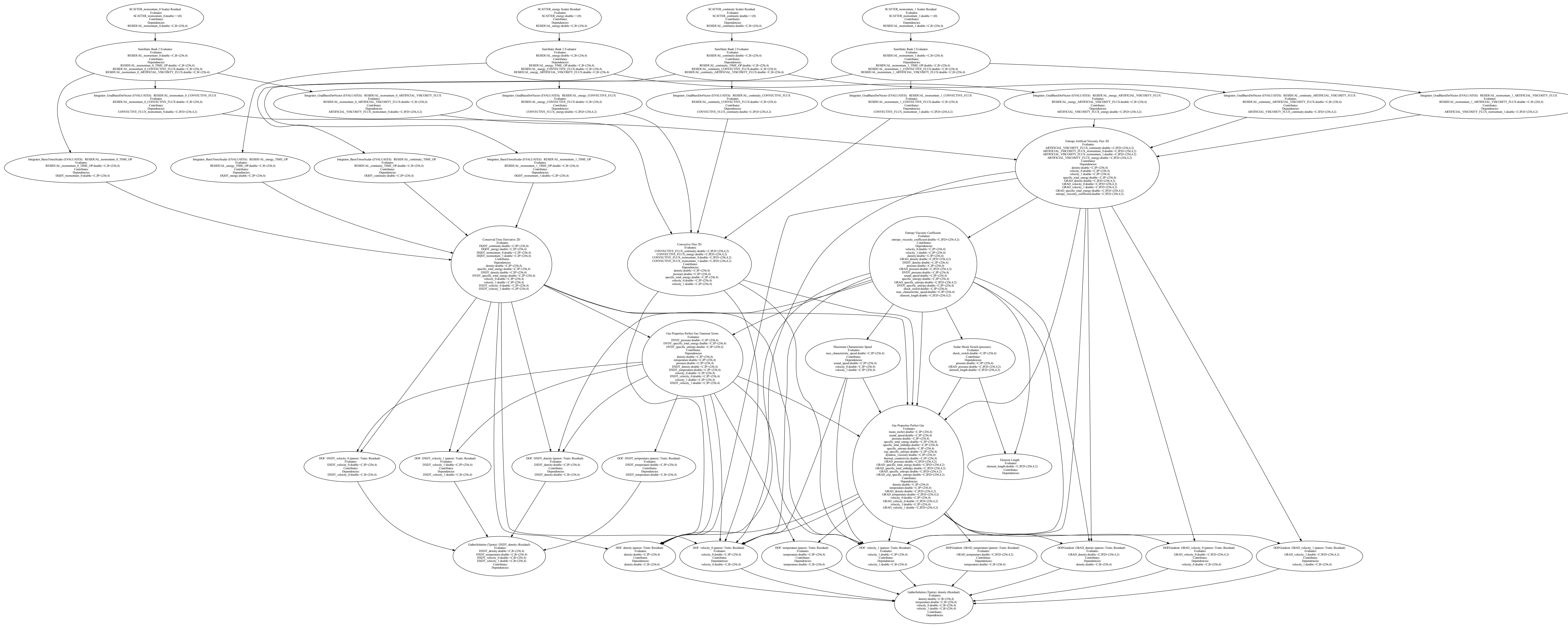
Stuart Slattery

Use of Trilinos

- Amesos
- Amesos2
- Belos
- Epetra
- Exodus
- Ifpack
- Ifpack2
- Intrepid2
- Kokkos
- Muelu
- NOX
- Panzer
- Phalanx
- Seacas
- Sacado
- Shards
- Stratimikos
- STK
- Tempus
- Teuchos
- Thyra
- Tpetra
- Zoltan

~40% of Trilinos packages

Phalanx Execution Graph


SCATTER_momentum_0 Scatter Residual
SCATTER_energy Scatter Residual
SCATTER_continuity Scatter Residual
SCATTER_momentum_1 Scatter Residual
SumStatic Rank 2 Evaluator
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_momentum_0_CONVECTIVE_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_momentum_0_ARTIFICIAL_VISCOSITY_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_energy_CONVECTIVE_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_continuity_CONVECTIVE_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_momentum_1_CONVECTIVE_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_energy_ARTIFICIAL_VISCOSITY_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_continuity_ARTIFICIAL_VISCOSITY_FLUX
Integrator_GradBasisDotVector (EVALUATES): RESIDUAL_momentum_1_ARTIFICIAL_VISCOSITY_FLUX
Integrator_BasisTimesScalar (EVALUATES): RESIDUAL_momentum_0_TIME_OP
Integrator_BasisTimesScalar (EVALUATES): RESIDUAL_energy_TIME_OP
Integrator_BasisTimesScalar (EVALUATES): RESIDUAL_continuity_TIME_OP
Integrator_BasisTimesScalar (EVALUATES): RESIDUAL_momentum_1_TIME_OP
Entropy Artificial Viscosity Flux 2D
Conserved Time Derivative 2D
Convective Flux 2D
Entropy Viscosity Coefficient
Gas Properties Perfect Gas Transient Terms
Maximum Characteristic Speed
Scalar Shock Switch (pressure)
Gas Properties Perfect Gas
Element Length
DOF: DXDT_velocity_0 (panzer::Traits::Residual)
DOF: DXDT_velocity_1 (panzer::Traits::Residual)
DOF: DXDT_density (panzer::Traits::Residual)
DOF: DXDT_temperature (panzer::Traits::Residual)
GatherSolution (Tpetra): DXDT_density (Residual)
DOF: density (panzer::Traits::Residual)
DOF: velocity_0 (panzer::Traits::Residual)
DOF: temperature (panzer::Traits::Residual)
DOF: velocity_1 (panzer::Traits::Residual)
DOFGradient: GRAD_temperature (panzer::Traits::Residual)
DOFGradient: GRAD_density (panzer::Traits::Residual)
DOFGradient: GRAD_velocity_0 (panzer::Traits::Residual)
DOFGradient: GRAD_velocity_1 (panzer::Traits::Residual)
GatherSolution (Tpetra): density (Residual)

Parameterization: Objectives

- Allow developers to declare scalars in objects that may be parameterized
- Allow users to declare global scalar parameters in the input file and map those global parameters to local object parameters
- Orchestrate the updating of parameters automatically from the Thyra `ModelEvaluator` parameter interface
- Examples:
  - Scalar values in material properties
  - Scalar values in boundary conditions
  - Scalar values in initial conditions

Parameterization: User Input

```
<ParameterList name=”Scalar Parameters">
   <ParameterList>
     <Parameter name="Name" type="string" value="Thermal Conductivity” />
     <Parameter name="Nominal Value" type="double" value="1.23e-4" />
   </ParameterList>
   <ParameterList>
     <Parameter name="Name" type="string" value="Inflow Velocity" />
     <Parameter name="Nominal Value" type="double" value="0.53" />
   </ParameterList>
</ParameterList>
```

Parameterization: Using parameters in objects

```
<ParameterList>
  <Parameter name="Sideset ID"        type="string" value="left"         />
  <Parameter name="Element Block ID"  type="string" value="eblock-0_0"  />
  <Parameter name="Strategy"          type="string" value="BoundaryFlux"/>
  <ParameterList name="Data">
    <Parameter name="Type" type="string" value="Farfield"/>
    <Parameter name="Farfield Density"  type="double" value="1.0"/>
    <Parameter name="Farfield Velocity 0"  type="ScalarParameter" value="Inflow Velocity"/>
    <Parameter name="Farfield Velocity 1"  type="double" value="0.0"/>
    <Parameter name="Farfield Temperature"  type="double" value="1.0"/>
  </ParameterList>
</ParameterList>
```

- `GlobalScalarParameter` will be a special type for which we write `ParameterList` serialization
- A user can toggle between a standard `double` value and a `ScalarParameter` value to link an object parameter to a global parameter.

Parameterization: Design Considerations

- Updated parameters get assigned before every `evalModel()` call in `Panzer::ModelEvaluator` and reset back to nominal values after
  - This means an automated process during graph execution must be created to automatically update parameters – we can't interject our own code otherwise
- An observer pattern is used to automatically update the parameters in an object based on the content of the Thyra parameter vectors provided to `evalModel()`
  - Parameter observers must be registered upon construction. This will occur in all factories as well as any hand-built observers. An object must manage the state of the observers and trigger their update
  - Parameters hold reference to object data and update automatically in `update()`
- New evaluator that is always at bottom of graph and triggers parameter update. This will allow for exodus graph evaluation with parameters.
  - Will still need to get parameters into parameter library before Exodus output
  - Make a dummy field upon which all evaluators depend. Write our own base class upon which all of our evaluators depend. This base class depends on the dummy field. The dummy field is evaluated in the `ParameterEvaluator` and hence triggers parameter evaluation prior to all graph evaluations

Parameterization: Parameter Distribution

```cpp
//---------------------------------------------------------------------------//
template<class EvalType, class Traits>
ScalarParameterEvaluator<EvalType, Traits>::ScalarParameterEvaluator(
    const Teuchos::RCP<ScalarParameterManager<EvalType>>& param_manager,
    const Teuchos::RCP<panzer::GlobalData>& global_data)
    : _param_manager(param_manager)
    , _global_data(global_data)
{
    auto dummy_layout = Teuchos::rcp(new PHX::MDALayout<panzer::Dummy>(0));
    _param_update_trigger = Teuchos::rcp(
        new PHX::Tag<scalar_type>("scalar_parameter_eval", dummy_layout));
    this->addEvaluatedField(*_param_update_trigger);
    this->setName("Scalar Parameter Evaluation");
}

//---------------------------------------------------------------------------//
template<class EvalType, class Traits>
void ScalarParameterEvaluator<EvalType, Traits>::preEvaluate(
    typename Traits::PreEvalData)
{
    _param_manager->update(*_global_data);
}

//---------------------------------------------------------------------------//
template<class EvalType, class Traits>
void ScalarParameterEvaluator<EvalType, Traits>::evaluateFields(
    typename Traits::EvalData)
{
}
```

Parameterization: New Evaluators

```cpp
template<class EvalType, class Traits>
class EvaluatorWithParameter
    : public EvaluatorBase<EvalType, Traits>,
      public Response::ScalarParameterObserver<EvalType>
{
  public:
    using scalar_type = typename EvalType::ScalarT;

    EvaluatorWithParameter(const panzer::IntegrationRule& ir,
                           const Teuchos::ParameterList& params)
        : _pre_f1("pre_f1", ir.dl_scalar)
        , _pre_f2("pre_f2", ir.dl_scalar)
        , _pre_f3("pre_f3", ir.dl_scalar)
        , _f1("f1", ir.dl_scalar)
        , _f2("f2", ir.dl_scalar)
        , _f3("f3", ir.dl_scalar)
    {
        this->addEvaluatedField(_pre_f1);
        this->addEvaluatedField(_pre_f2);
        this->addEvaluatedField(_pre_f3);
        this->addEvaluatedField(_f1);
        this->addEvaluatedField(_f2);
        this->addEvaluatedField(_f3);
        this->registerParameter("p1", 2.0, params, _p1);
        this->registerParameter("p2", 3.0, params, _p2);
        this->registerParameter("p3", 4.0, params, _p3);
        this->setName("EvaluatorWithParameter");
    }

  protected:
    void updateStateWithNewParameters() override
    {
        // Get the parameters here to make sure these are available in
        // evaluators to do pre-evaluate calculations.
        _pre_p1 = _p1;
        _pre_p2 = _p2;
        _pre_p3 = _p3;
    }

    void preEvaluateImpl(typename Traits::PreEvalData) override
    {
        _pre_f1.deep_copy(_pre_p1);
        _pre_f2.deep_copy(_pre_p2);
        _pre_f3.deep_copy(_pre_p3);
    }

    void evaluateFieldsImpl(typename Traits::EvalData) override
    {
        _f1.deep_copy(_p1);
        _f2.deep_copy(_p2);
        _f3.deep_copy(_p3);
    }
};
```

```
// <ParameterList>
//     <Parameter name="p2"  type="ScalarParameter" value="Parameter2"/>
//     <Parameter name="p3"  type="double" value="9.2"/>
// </ParameterList>
//
// Note that p1 is not included to trigger the default evaluation.
```

OAK RIDGE National Laboratory

Development Ideas

Short Term:

- High-order element topology support (quadratic and cubic) in both Exodus and Panzer assembly
- General Panzer/Trilinos support for CUDA (non-UVM) and HIP

Long Term:

- Tangent support in Tpetra
- Hessian support in Panzer
- Delayed Jacobian evaluation in Panzer

Linear Solvers and Preconditioning

Steven Hamilton

Solver considerations

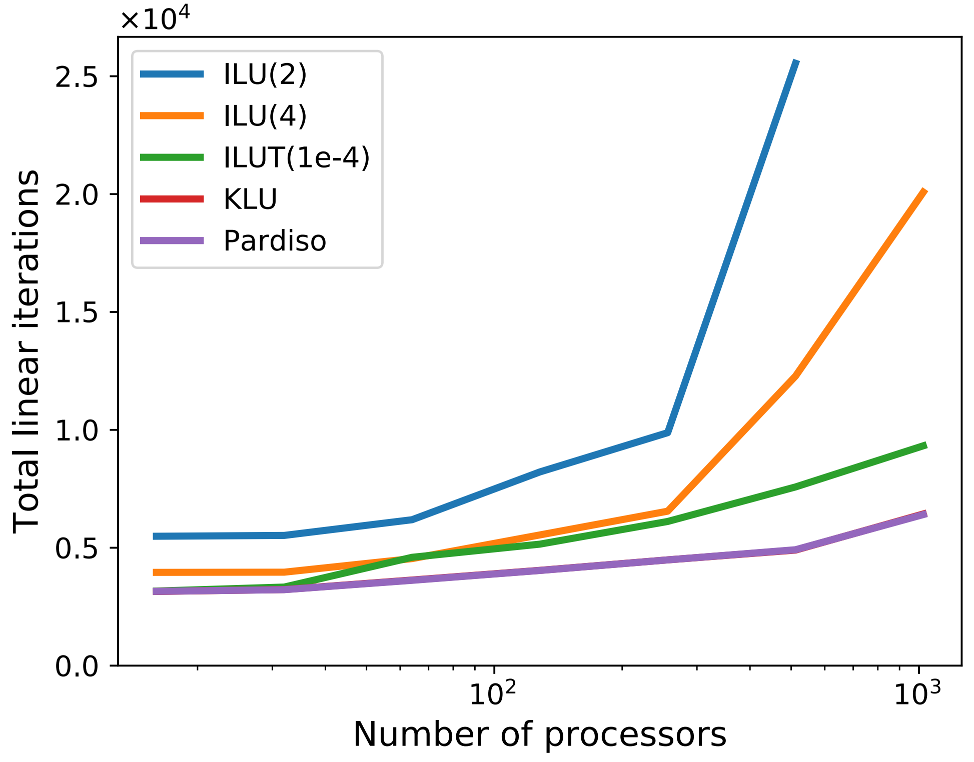
- Primarily targeting multithreaded CPUs for near-term
  - Support for both Nvidia and AMD GPUs is important for future
- Currently focused on algebraic preconditioning (with AD-generated Jacobian matrices)
- Main solver strategy is GMRES (w/o restart if possible) and overlapping additive Schwarz preconditioner
  - Incomplete factorizations or sparse direct solvers for local solves
- Algebraic multigrid performs poorly on all but trivial problems
  - Similar behavior in ML, MueLu, and BoomerAMG
  - Ongoing work to understand challenges
- Overlap and threading (larger local domains) improve robustness and strong scaling

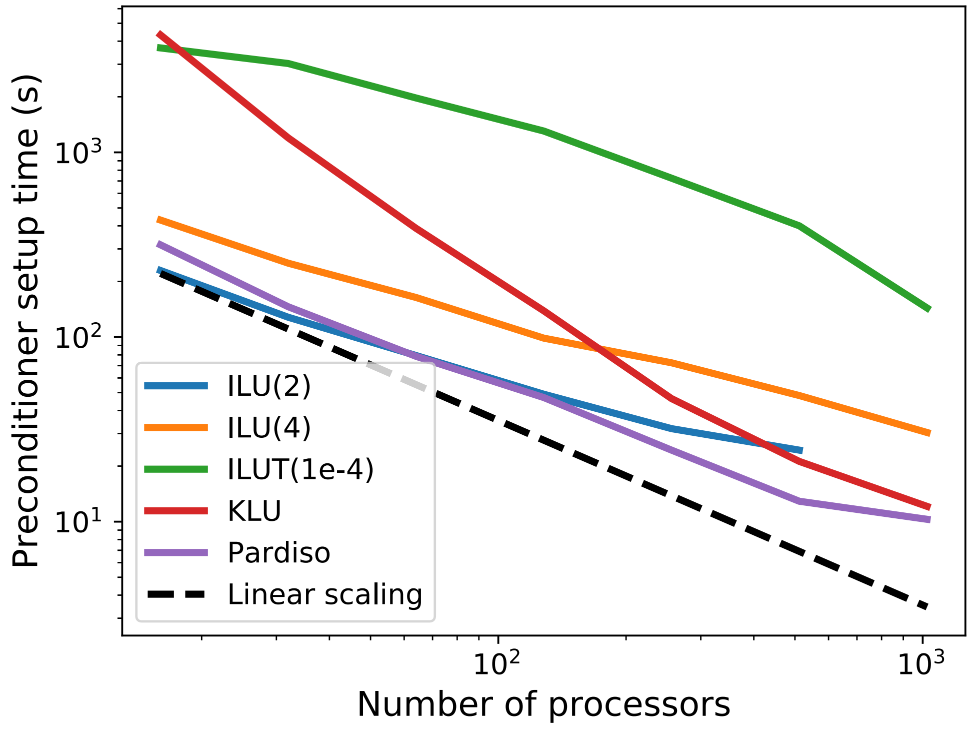
Solver selection

- Stratimikos allows extensive solver and preconditioner selection from input file
  - Belos, Ifpack(2), Amesos(2), ML, MueLu...
- Both Epetra and Tpetra stacks are supported for complete flexibility
  - Ifpack2 doesn't recognize overlap for additive Schwarz
  - Epetra implementations of some TPLs (e.g., PardisoMKL?) seem to be more robust
- For Nvidia GPUs, cuSOLVER GLU sparse direct solver is used
  - One-time host factorization, refactor on device
  - Not in public documentation

Solver performance

- MPI only (1 thread per rank), 2x 64-core AMD CPUs
- Global matrix size: 120k spatial elements, 600k DOFs





Preconditioner comparison

- 2x 64-core AMD CPUs, 2x Nvidia V100 GPUs per node
- Global Matrix size: 120k spatial elements, 600k DOFs

Device	CPU	CPU	CPU	GPU
MPI ranks	128	16	2	2
OpenMP threads per rank	1	8	64	1
Local solver	KLU2	Pardiso	Pardiso	GLU
Total linear solve (s)	500.8	471.5	929.9	964.5
Total preconditioner setup (s)	43.9	106.3	825.4	330.8
One-time CPU prec. setup (s)	—	—	—	64.3
Total preconditioner apply (s)	224.6	232.3	81.6	550.6
Total linear iterations	32,513	22,172	3,737	3,757
Avg. linear iterations per Newton	86.7	59.1	10.0	10.0

Native preconditioner implementation

- Panzer provides method to construct Thyra linear solver factory from parameters
  - Internally uses Stratimikos
  - Registers mesh coordinates with AMG solvers
- Stratimikos factory registration is instance based
  - No way to inject user preconditioner

```
// Replicate behavior of panzer_stk::buildLOWSFactory
RCP<Thyra::LinearOpWithSolveFactoryBase<>>
LOWSFactoryBuilder::buildLOWS(RCP<ParameterList> params)
{
    Stratimikos::DefaultLinearSolverBuilder builder;

    using Base = Thyra::PreconditionerFactoryBase<>;

    // Register Trilinos factories that aren't automatic
    using Ifpack2Factory = Thyra::Ifpack2PreconditionerFactory<...>>;
    builder.setPreconditioningStrategyFactory(
        abstractFactoryStd<Base, Ifpack2Factory>(), "Ifpack2");
    Stratimikos::enableMueLu<...>(builder, "MueLu");

    // Register our own preconditioner factory with Stratimikos
    builder.setPreconditioningStrategyFactory(
        abstractFactoryStd<Base, NativeFactory>(), "Native");

    builder.setParameterList(params);
    RCP<Thyra::LinearOpWithSolveFactoryBase<>> lowsFactory
        = createLinearSolveStrategy(builder);

    return lowsFactory;
}
```

Native preconditioner implementation (2)

```
// Replicate behavior of panzer_stk::buildLOWSFactory
RCP<Thyra::LinearOpWithSolveFactoryBase<>>
LOWSFactoryBuilder::buildLOWS(RCP<ParameterList> params)
{
    Stratimikos::DefaultLinearSolverBuilder builder;

    using Base = Thyra::PreconditionerFactoryBase<>;

    // Register Trilinos factories that aren't automatic
    using Ifpack2Factory = Thyra::Ifpack2PreconditionerFactory<...>>;
    builder.setPreconditioningStrategyFactory(
        abstractFactoryStd<Base, Ifpack2Factory>(), "Ifpack2");
    Stratimikos::enableMueLu<...>(builder, "MueLu");

    // Register our own preconditioner factory with Stratimikos
    builder.setPreconditioningStrategyFactory(
        abstractFactoryStd<Base, NativeFactory>(), "Native");

    builder.setParameterList(params);
    RCP<Thyra::LinearOpWithSolveFactoryBase<>> lowsFactory
        = createLinearSolveStrategy(builder);

    return lowsFactory;
}
```

```
// Replicate behavior of panzer_stk::buildLOWSFactory
RCP<Thyra::LinearOpWithSolveFactoryBase<>>
LOWSFactoryBuilder::buildLOWS(RCP<ParameterList> params)
{
    Stratimikos::DefaultLinearSolverBuilder builder;

    using Base = Thyra::PreconditionerFactoryBase<>;

    // Register Trilinos factories that aren't automatic
    using Ifpack2Factory = Thyra::Ifpack2PreconditionerFactory<...>>;
    builder.setPreconditioningStrategyFactory(
        abstractFactoryStd<Base, Ifpack2Factory>(), "Ifpack2");
    Stratimikos::enableMueLu<...>(builder, "MueLu");

    // Register our own preconditioner factory with Stratimikos
    builder.setPreconditioningStrategyFactory(
        abstractFactoryStd<Base, NativeFactory>(), "Native");

    builder.setParameterList(params);
    RCP<Thyra::LinearOpWithSolveFactoryBase<>> lowsFactory
        = createLinearSolveStrategy(builder);

    return lowsFactory;
}
```

```
class NativeFactory : public Thyra::PreconditionerFactoryBase<>
{
    void initializePrec(
        const Teuchos::RCP<const Thyra::LinearOpSourceBase<>>& fwdOpSrc,
        Thyra::PreconditionerBase<>* precOp,
        const Thyra::ESupportSolveUse supportSolveUse
        = Thyra::SUPPORT_SOLVE_UNSPECIFIED) const
    {
        // Lots of Thyra stuff here...
        ...

        // Build AdditiveSchwarz and register our local solver
        auto schwarz = rcp(
            new Ifpack2::AdditiveSchwarz<>(tpetra_matrix));
        auto inner_prec = Teuchos::rcp(new LocalGLUSolver());
        schwarz->setInnerPreconditioner(inner_prec);

        // More Thyra stuff...
        ...
    }
};
```

Native preconditioner implementation (3)

```cpp
class NativeFactory : public Thyra::PreconditionerFactoryBase<>
{
    void initializePrec(
        const Teuchos::RCP<const Thyra::LinearOpSourceBase<>>& fwdOpSrc,
        Thyra::PreconditionerBase<>* precOp,
        const Thyra::ESupportSolveUse supportSolveUse
        = Thyra::SUPPORT_SOLVE_UNSPECIFIED) const
    {
        // Lots of Thyra stuff here...
        ...

        // Build AdditiveSchwarz and register our local solver
        auto schwarz = rcp(
            new Ifpack2::AdditiveSchwarz<>(tpetra_matrix));
        auto inner_prec = Teuchos::rcp(new LocalGLUSolver());
        schwarz->setInnerPreconditioner(inner_prec);

        // More Thyra stuff...
        ...
    }
};
```

```cpp
class LocalGLUSolver : public Ifpack2::Preconditioner<>,
                       public Ifpack2::Details::CanChangeMatrix<>
{
    // This is NOT a Tpetra::CrsMatrix
    void setMatrix(const RCP<const Tpetra::RowMatrix<>& A);

    // Perform one-time host-side setup
    void initialize();

    // Refactor local matrix on device
    void compute();

    // Solve local matrix
    void apply(const Tpetra::MultiVector<>& x,
               Tpetra::MultiVector<>& y,
               Teuchos::ETransp mode,
               double alpha,
               double beta) const;
};
```

Wishlist

- More preconditioners "automatically" supported by Stratimikos
  - Including documentation on valid parameters
- Simpler approach to registering user-implemented preconditioners
- More GPU-capable preconditioners
  - When AMG doesn't work, there aren't many alternatives...
  - cuSOLVER for Nvidia GPUs, AMD path is unclear (SuperLU?)
  - Refactor-based sparse direct solvers?