SAND2018-2037C

Exceptional service in the national interest

Sandia National Laboratories

Towards Performance Portable Assembly Tools for Multi-Fluid Plasma Simulations

Roger Pawlowski*, Matt Bettencourt**, Eric Cyr*, Sean Miller*, Edward Phillips**, Eric Phipps*, John Shadid* and Christian Trott*

*Center for Computing Research, Sandia National Laboratories

**Radiation and Electrical Sciences Center, Sandia National Laboratories

SIAM Conference on Parallel Processing for Scientific Computing

March 7-10, 2018 • Tokyo, Japan

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

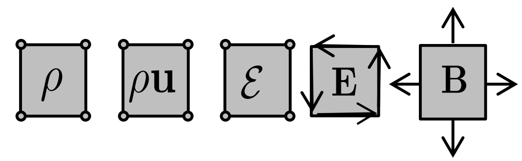
Outline

- Introduction
  - Requirements
  - Example Physics
- Components Description
  - Kokkos, Sacado, Phalanx, Panzer
- Two Design Explorations
  - Hierarchic Parallelism
  - Device DAG: Kernel collapse
- Conclusions

SNL's Mission Requires a Significant Range of Advanced Simulation Capabilities

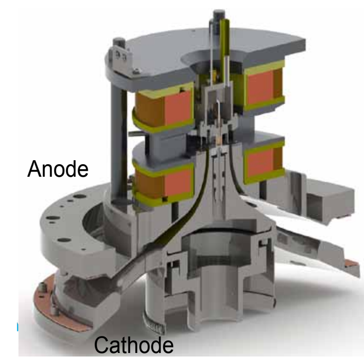
<u>DOE/NNSA and many DOE/SC Mission Drivers are Characterized by:</u>

- Complex strongly coupled physical mechanisms (**multiphysics**)
  - ➢ Strongly coupled nonlinear solvers (**Newton methods**)
  - ➢ Physics-compatible discretizations



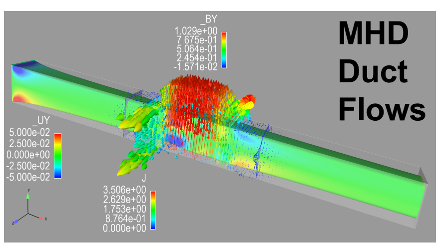
- Large range of interacting time-scales (Multiple-time-scales)
  - ➢ Implicitness (**fully-Implicit** or **implicit/explicit [IMEX]**)

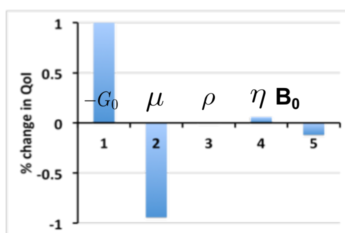
- Complex geometries, multiple length-scales, high-resolution
  - ➢ Unstructured mesh FE (HEX and TET)
  - ➢ Scalable solution (**Krylov methods, physics-based prec., AMG**)

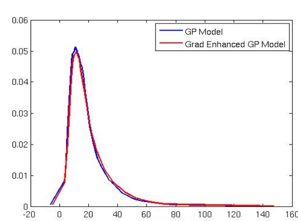


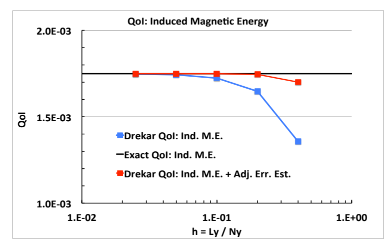
Z Convolute Power-feed

- High consequence decisions informed by modeling / simulation
  - ➢ Beyond forward simulation (**sensitivities, UQ, error est., design opt.**)



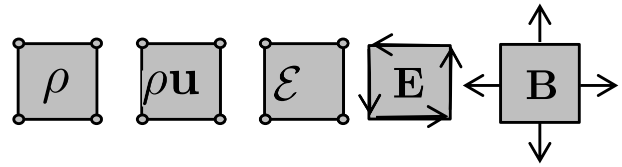






Adjoint-enabled Sensitivities, UQ surrogates, Error-estimates

Multi-fluid 5-Moment Plasma System Model



Density	$\frac{\partial \rho_a}{\partial t} + \nabla \cdot (\rho_a \mathbf{u}_a) = \sum_{b\neq a} (n_a \rho_b \bar{\nu}^+_{ab} - n_b \rho_a \bar{\nu}^-_{ab})$
Momentum	$\frac{\partial (\rho_a \mathbf{u}_a)}{\partial t} + \nabla \cdot (\rho_a \mathbf{u}_a \otimes \mathbf{u}_a + p_a I + \Pi_a) = q_a n_a (\mathbf{E} + \mathbf{u}_a \times \mathbf{B})$ $- \sum_{b\neq a} [\rho_a (\mathbf{u}_a - \mathbf{u}_b) n_b \bar{\nu}^M_{ab} + \rho_b \mathbf{u}_b n_a \bar{\nu}^+_{ab} - \rho_a \mathbf{u}_a n_b \bar{\nu}^-_{ab}]$
Energy	$\frac{\partial \varepsilon_a}{\partial t} + \nabla \cdot ((\varepsilon_a + p_a)\mathbf{u}_a + \Pi_a \cdot \mathbf{u}_a + \mathbf{h}_a) = q_a n_a \mathbf{u}_a \cdot \mathbf{E} + Q_a^{src}$ $- \sum_{b\neq a} \left[(T_a - T_b) k \bar{\nu}^E_{ab} - \rho_a \mathbf{u}_a \cdot (\mathbf{u}_a - \mathbf{u}_b) n_b \bar{\nu}^M_{ab} - n_a \bar{\nu}^+_{ab} \varepsilon_b + n_b \bar{\nu}^-_{ab} \varepsilon_a \right]$
Charge and Current Density	$q = \sum_k q_k n_k$ $\qquad \mathbf{J} = \sum_k q_k n_k \mathbf{u}_k$
Maxwell's Equations	$\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} - \nabla \times \mathbf{B} + \mu_0 \mathbf{J} = \mathbf{0}$ $\qquad \nabla \cdot \mathbf{E} = \frac{q}{\epsilon_0}$ $\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} = \mathbf{0}$ $\qquad \nabla \cdot \mathbf{B} = 0$

IMEX: Time Integration

$$\mathbf{M}\dot{\mathbf{U}} + \mathbf{F} + \mathbf{G} = 0$$

Explicit Hydrodynamics

Implicit EM, EM sources, sources for species $(\rho_a, \rho_a \mathbf{u}_a, \epsilon_a)$ interactions

IMEX splitting for CG

Sandia National Laboratories

$$\partial_t \rho_\alpha + \boldsymbol{u}_\alpha \cdot \nabla \rho_\alpha = -\rho_\alpha \nabla \cdot \boldsymbol{u}_\alpha$$
$$u_\alpha < \frac{\Delta x}{\Delta t}$$

Each operator is associated with one or more plasma scales, which are grouped by color representing their approximate explicit stability limits.

$$\partial_t \boldsymbol{u}_\alpha + \boldsymbol{u}_\alpha \cdot \nabla \boldsymbol{u}_\alpha = -\boldsymbol{u}_\alpha \nabla \cdot \boldsymbol{u}_\alpha - \frac{1}{\rho_\alpha} \nabla P_\alpha + \frac{1}{\rho_\alpha} \nabla \cdot \left(\mu_\alpha \left(\nabla \boldsymbol{u}_\alpha + \nabla \boldsymbol{u}_\alpha^T - \frac{2}{3} \boldsymbol{I} \nabla \cdot \boldsymbol{u}_\alpha \right) \right)$$
$$u_\alpha < \frac{\Delta x}{\Delta t} \qquad v_{s\alpha} < \frac{\Delta x}{\Delta t} \qquad \upsilon_\alpha < \frac{\Delta x^2}{\Delta t}$$

$$+ \frac{q_\alpha}{m_\alpha} \boldsymbol{E} + \frac{q_\alpha}{m_\alpha} \boldsymbol{u}_\alpha \times \boldsymbol{B} - \sum_\beta \nu_{\alpha\beta} \left(\boldsymbol{u}_\alpha - \boldsymbol{u}_\beta \right)$$
$$\omega_{p\alpha} \Delta t < 1 \qquad \omega_{c\alpha} \Delta t < 1 \qquad \nu_{\alpha\beta} \Delta t < 1$$

$$\partial_t P_\alpha + \boldsymbol{u}_\alpha \cdot \nabla P_\alpha = -\gamma P_\alpha \nabla \cdot \boldsymbol{u}_\alpha + \nabla \cdot \left((\gamma - 1) k_\alpha \nabla T_\alpha \right) - \sum_\beta \frac{(\gamma - 1) \nu_{\alpha\beta} \rho_\alpha}{m_\alpha + m_\beta} \left(3 \left(T_\alpha - T_\beta \right) - m_\beta \left(\boldsymbol{u}_\alpha - \boldsymbol{u}_\beta \right)^2 \right)$$
$$u_\alpha < \frac{\Delta x}{\Delta t} \qquad \kappa_\alpha < \frac{\Delta x^2}{\Delta t} \qquad \nu_{\alpha\beta} \Delta t < 1$$

$$\partial_t \boldsymbol{E} - c^2 \nabla \times \boldsymbol{B} = -\frac{1}{\epsilon_0} \sum_\alpha \frac{q_\alpha}{m_\alpha} \rho_\alpha \boldsymbol{u}_\alpha$$
$$c < \frac{\Delta x}{\Delta t} \qquad \omega_{p\alpha} \Delta t < 1$$
$$\partial_t \boldsymbol{B} + \nabla \times \boldsymbol{E} = 0$$

For IMEX-CG each operator can be moved between implicit and explicit evaluation depending on the explicit stability limits.

PDE Tools Design Considerations

Sandia National Laboratories

- Sensitivities are critical!
  - Required for: Implicit and IMEX, steady-state and transient parametric sensitivity analysis, Optimization and Stability/Bifurcation analysis
  - Do not burden analysts/physics experts with analysis algorithm requirements
  - Combinatorial explosion of sensitivity requirements

- Develop PDE discretization tools for next generation architectures
  - Performance portability: CPU, KNL, GPU
  - Based on "Type-2" stack in Trilinos

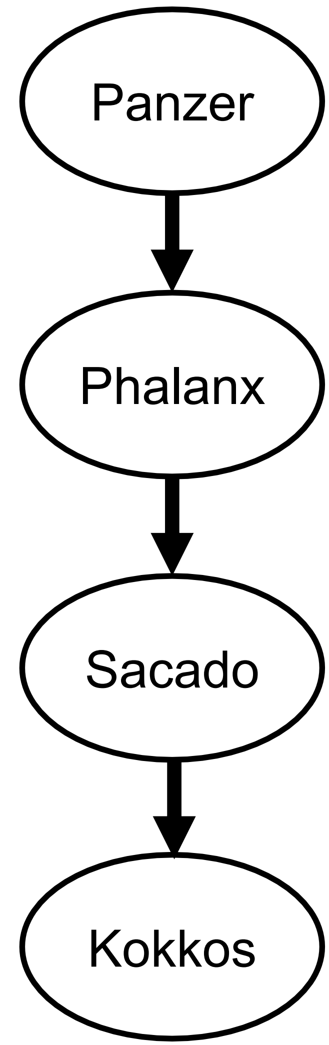
- Handle complexity in multiphysics PDE systems:
  - Complex interdependent coupled physics
  - Multiple proposed mathematical models
  - Different numerical formulations (e.g. space-time discretizations)
  - Supporting multiplicity in models and solution techniques often leads to complex code with ***complicated logic*** and ***fragile software designs***

- No **F**ramework!!! A component-based design:
  - Simple tools with minimal dependencies
  - **Risk mitigation: buy in at different levels**

- No Symbolics/DSL (definition is fuzzy)
  - Legacy code integration path, structured transition
  - Raw C++, access to data structures

Trilinos Discretization Tools Overview

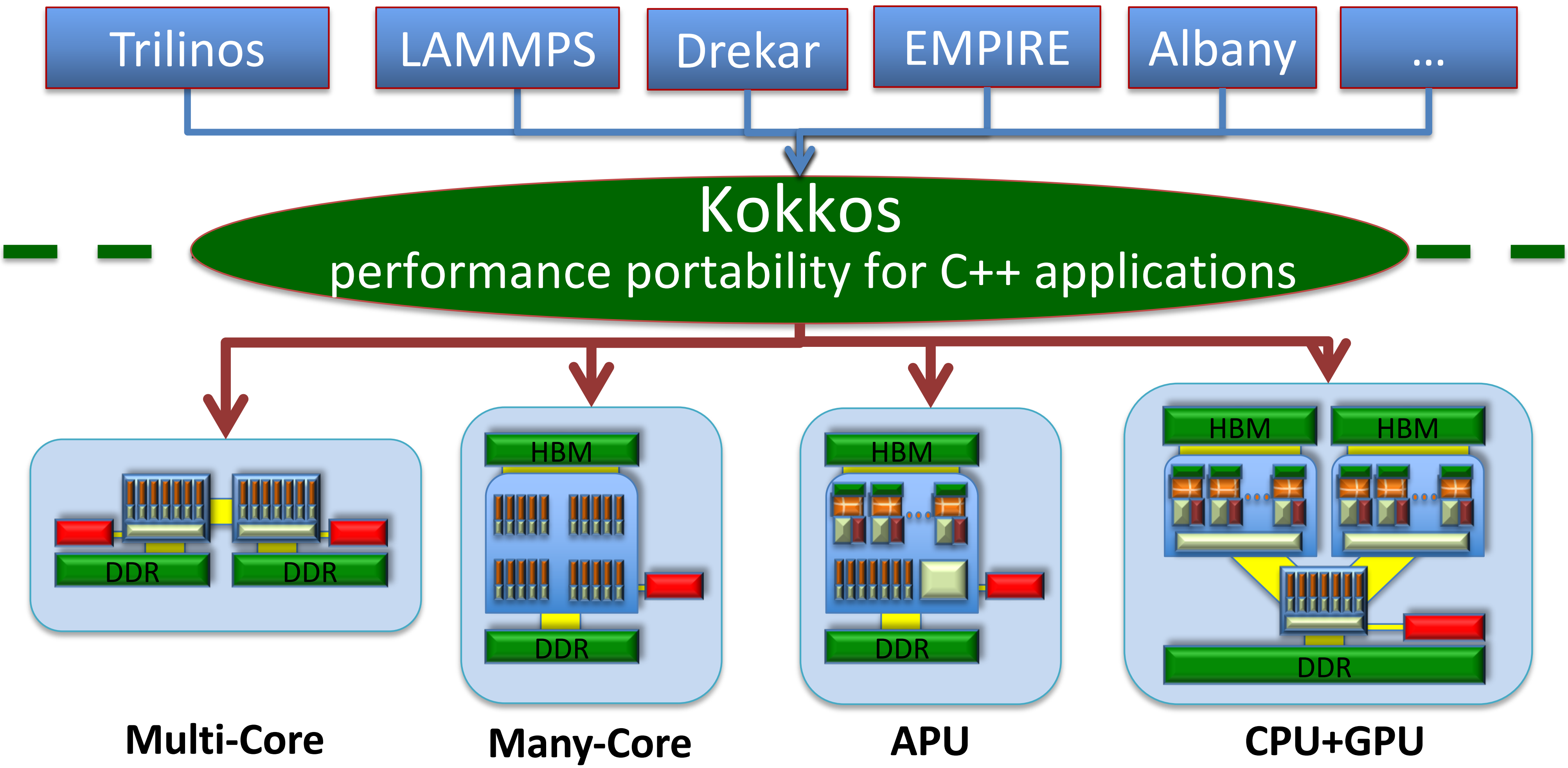
- MPI Related
  - Panzer (Multiphysics Assembly and Utilities)
    - DOF Manager: Global Indexing for mixed bases, mixed equations
      - Connection Manager: Mesh DB abstraction
    - Workset Builder: Mesh over-decomposition for AMT
    - Linear Algebra Builder: Epetra/Tpetra/Thyra
    - Disc-FE: Multiphysics assembly, Mixed Eq Sets, Mixed Bases, BCs, Compatible discretizations, Projections
- Local Node
  - Intrepid2: FE Basis Library
  - Shards: Cell/Element Topology
  - Phalanx: DAG Assembly: flexibility/complexity
  - Sacado: Automatic differentiation scalar types
  - Kokkos: Performance portability



github.com/trilinos/Trilinos

Performance Portability: Kokkos

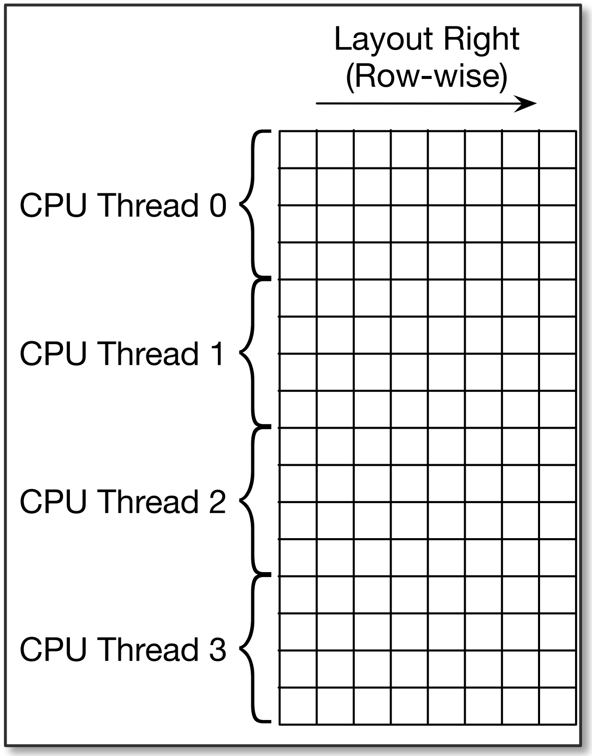
- Performance Portable Thread-Parallel Programming Model in C++
- Multidimensional Array
- Compiletime polymorphic memory layouts: cached vs coalesced memory
- Asynchronous Many Tasking (Experimental)

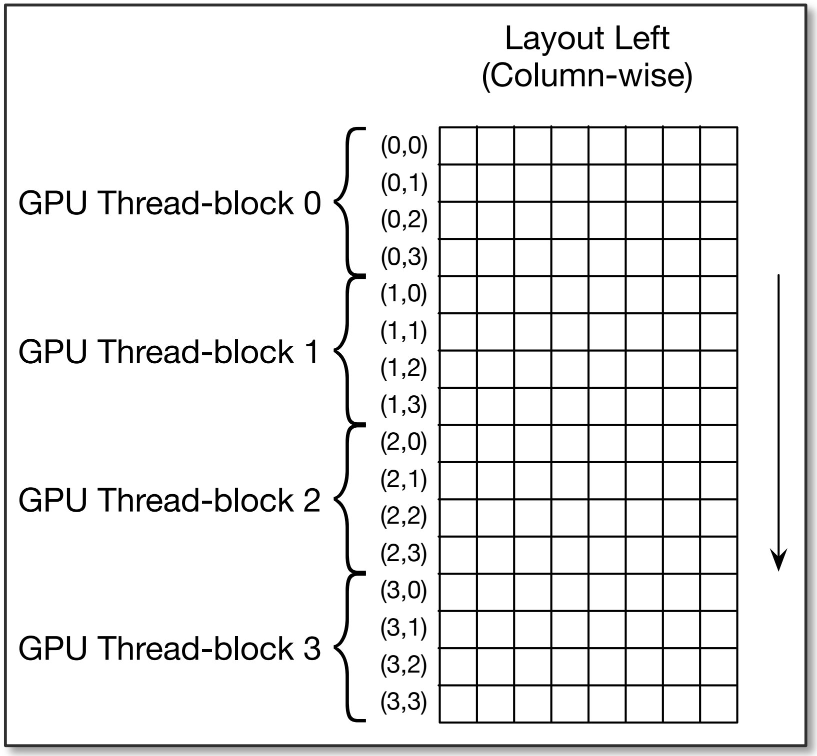


https://github.com/kokkos/kokkos

Kokkos Layout Polymorphism for Performant Memory Accesses

- **CPU/MIC**
  - **Each thread accesses contiguous range of entries**
  - **Ensures neighboring values are in cache**
- **GPU**
  - **Each thread accesses strided range of entries**
  - **Thread group can read all values in one memory transaction**
  - **Ensures coalesced accesses (consecutive threads access consecutive entries)**





Sacado: Template-based Automatic Differentiation

- Implement equations templated on the scalar type
- Libraries provide new scalar types that ***overload the math operators*** to propagate embedded quantities
  - Expression templates for performance
  - Derivatives: FAD, RAD
  - Hessians
  - Stochastic Galerkin: PCE
  - Multipoint: Ensemble (Stokhos)
- Analytic Values (NO FD involved)!

```
template <typename ScalarT>
void computeF(ScalarT* x, ScalarT* f)
{
  f[0] = 2.0 * x[0] + x[1] * x[1];
  f[1] = x[0] * x[0] * x[0] + sin(x[1]);
}
```

double **Fad<double>**

Operation	Forward AD rule
$c = a \pm b$	$\dot{c} = \dot{a} \pm \dot{b}$
$c = ab$	$\dot{c} = a\dot{b} + \dot{a}b$
$c = a/b$	$\dot{c} = (\dot{a} - c\dot{b})/b$
$c = a^r$	$\dot{c} = ra^{r-1}\dot{a}$
$c = \sin(a)$	$\dot{c} = \cos(a)\dot{a}$
$c = \cos(a)$	$\dot{c} = -\sin(a)\dot{a}$
$c = \exp(a)$	$\dot{c} = c\dot{a}$
$c = \log(a)$	$\dot{c} = \dot{a}/a$

```
double* x;
double* f;
…
computeF(x,f);
```

```
DFad<double>* x;
DFad<double>* dfdx;
…
computeF(x,dfdx);
```

Example Scalar Types

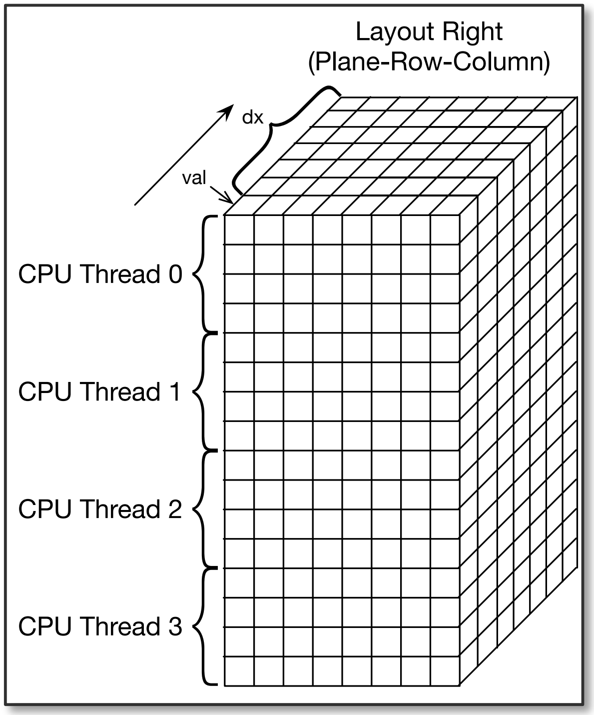
(Trilinos Stokhos and Sacado: E. Phipps)

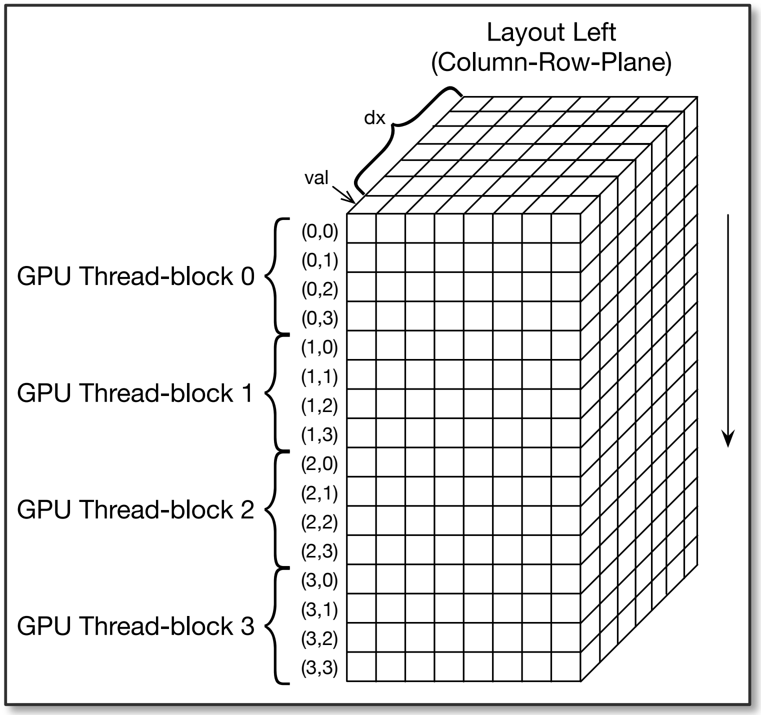
Evaluation Types		Scalar Types
• **Residual**	$F(x,p)$	double
• **Jacobian**	$J = \frac{\partial F}{\partial x}$	DFad<double>
• **Hessian**	$\frac{\partial^2 F}{\partial x_i \partial x_j}$	DFad< SFad<double,N> >
• **Parameter Sensitivities**	$\frac{\partial F}{\partial p}$	DFad<double>
• **Jv**	Jv	DFad<double>

1. **All evaluation types are compiled into single library and managed at runtime from a non-template base class via a template manager.**
2. **Not tied to double (can do arbitrary precision)**
3. **Can mix multiple scalar types in any evaluation type.**
4. **Can specialize any node: Write analytic derivatives for performance!**

Want good AD performance with no modifications to Kokkos kernels

- Achieved by specializing Kokkos::View data structure for Sacado scalar types
  - Rank-r Kokkos::View internally stored as a rank-(r+1) array of doubles
  - Kokkos layout applied to internal rank-(r+1) array





AD Performance Portability

```
Kokkos::View<Sacado::Fad::SFad<double,p>**> A("A",m,n,p+1);  // Create rank-2 array with m rows and n columns
Kokkos::View<Sacado::Fad::SFad<double,p>* > b("b",n,p+1);    // Create rank-1 array with n rows
Kokkos::View<Sacado::Fad::SFad<double,p>* > c("c",m,p+1);    // Create rank-1 array with m rows

// ...

run_mat_vec(A,b,c);
```

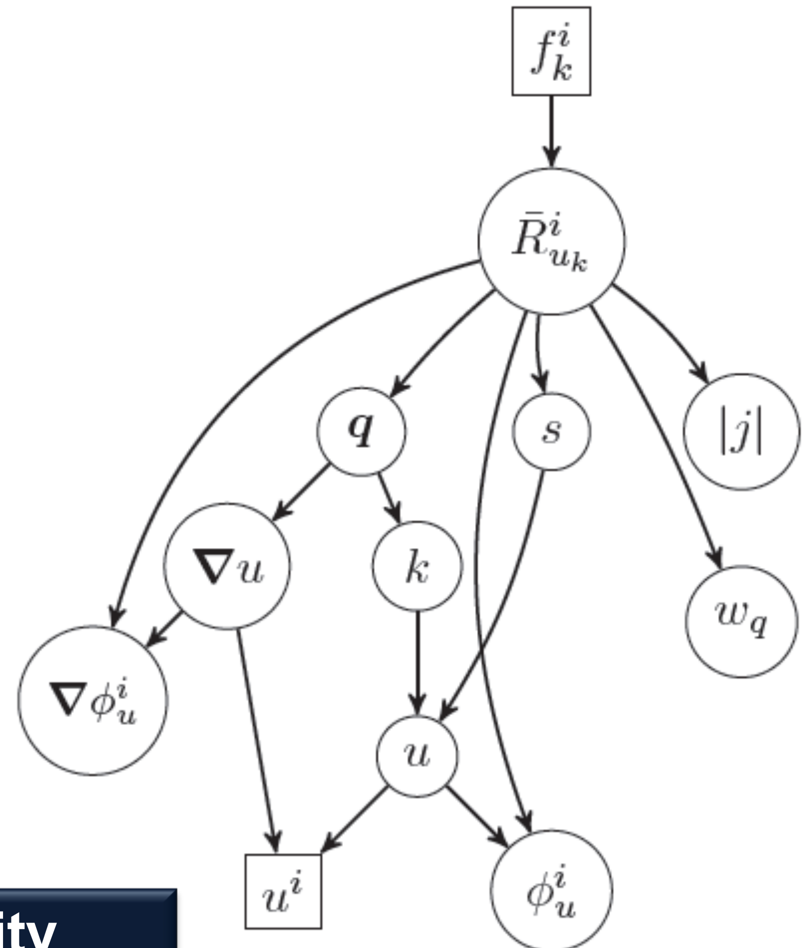
Architecture	Measured Bandwidth (GB/s)	Expected Throughput (GFLOP/s)	Measured Throughput (GFLOP/s)	No View Specialization (GFLOP/s)
Haswell	47.4	22.4	24.3	23.1
MIC	147	69.4	69.4	43.2
GPU	150	70.8	81.2	35.1

- **m = 1e6, n=100, p = 8 (derivative dimension)**
- **Expected Throughput ~ Measured Bandwidth x (4p+2) FLOPS / 8(p+1) Bytes**
- **SFad<double,p> AD data type**

Phalanx: Lightweight DAG-based Expression Evaluation

$$R_u^i = \int_\Omega \left[\phi_u^i \dot{u} - \nabla \phi_u^i \cdot \boldsymbol{q} + \phi_u^i s\right] \, d\Omega$$

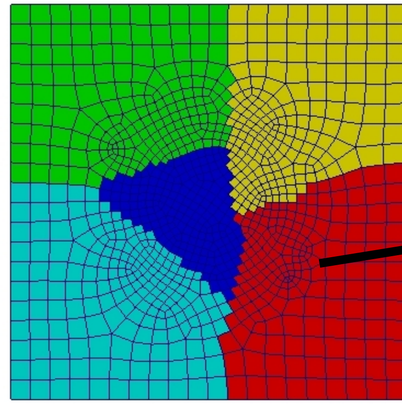
- Decompose a complex model into a graph of simple kernels (functors)
- A node in the graph evaluates one or more temporary **fields**
- Runtime DAG construction of graph
- Supports rapid development, separation of concerns and extensibility.
- Achieves flexible multiphysics assembly
- Leverages Sacado scalar types for non-invasive Jacobian, Hessian, …



DAG-Based Assembly → flexibility

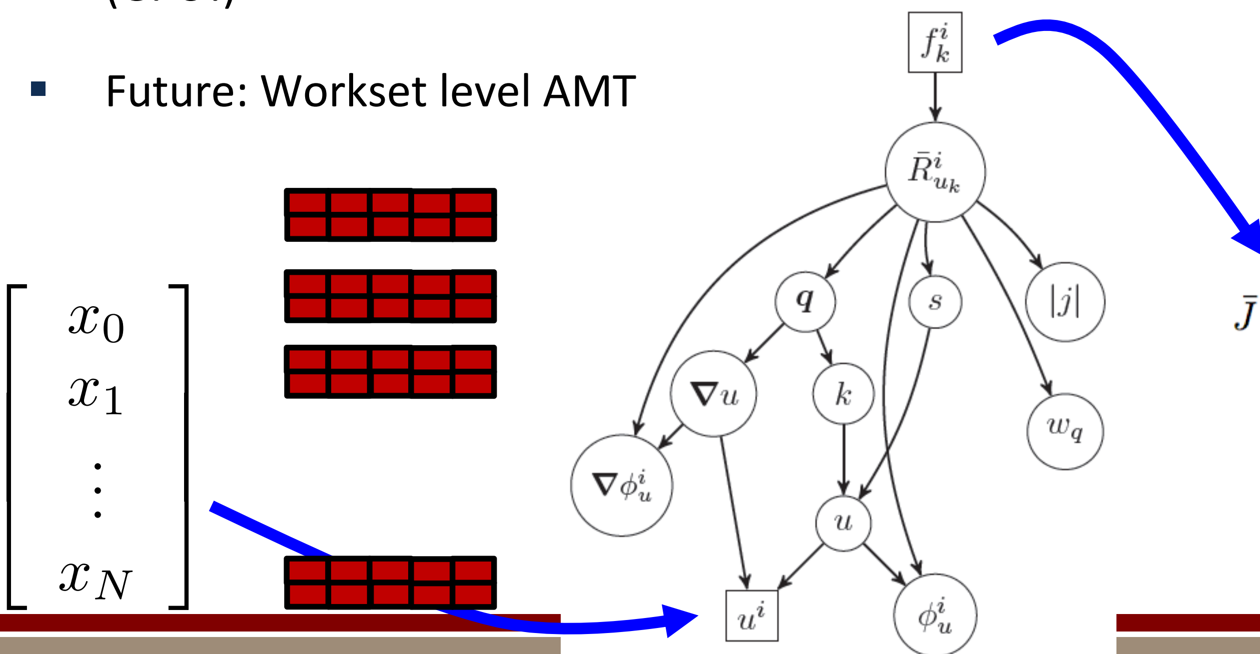
Workset Builder: Data Parallelism

- Batch of elements
  - Same operations, field dimensions, topology
- Fixed memory allocation for DAG
- Multiple worksets per hardware node
- Controls memory for temporaries (GPU!)
- Future: Workset level AMT



MPI Distributed Mesh

Hardware Node (Single MPI Process)



$$\bar{J} = \begin{bmatrix} \bar{J}_{0,0} & \cdots & \cdots & \cdots & \bar{J}_{0,N_f-1} \\ \vdots & \ddots & \vdots & & \vdots \\ \bar{J}_{i,0} & \cdots & \bar{J}_{i,i} & \cdots & \bar{J}_{i,N_f-1} \\ \vdots & & \vdots & \ddots & \vdots \\ \bar{J}_{N_f-1,0} & \cdots & \cdots & \cdots & \bar{J}_{N_f-1,N_f-1} \end{bmatrix}$$

What does a Node look like?

```
template<typename EvalT, typename Traits>
IntegrateDiffusionTerm<EvalT,Traits>::
IntegrateDiffusionTerm(const std::string& flux_name, const Teuchos::RCP<PHX::DataLayout>& flux_layout,
    const std::string& residual_name, const Teuchos::RCP<PHX::DataLayout>& residual_layout) :
    flux(flux_name,flux_layout), residual(residual_name,residual_layout)
{
  this->addContributedField(residual);
  this->addDependentField(flux);
  this->setName("IntegrateDiffusionTerm: "+residual_name);
}
```

Declare DAG Dependencies

```
template<typename EvalT, typename Traits>
void IntegrateDiffusionTerm<EvalT,Traits>::evaluateFields(typename Traits::EvalData workset)
{
  grad_basis = workset.grad_basis_real_;
  weights = workset.weights_;
  cell_measure = workset.det_jac_;
  Kokkos::parallel_for(Kokkos::RangePolicy<PHX::exec_space>(0,workset.num_cells_),*this);
}
```

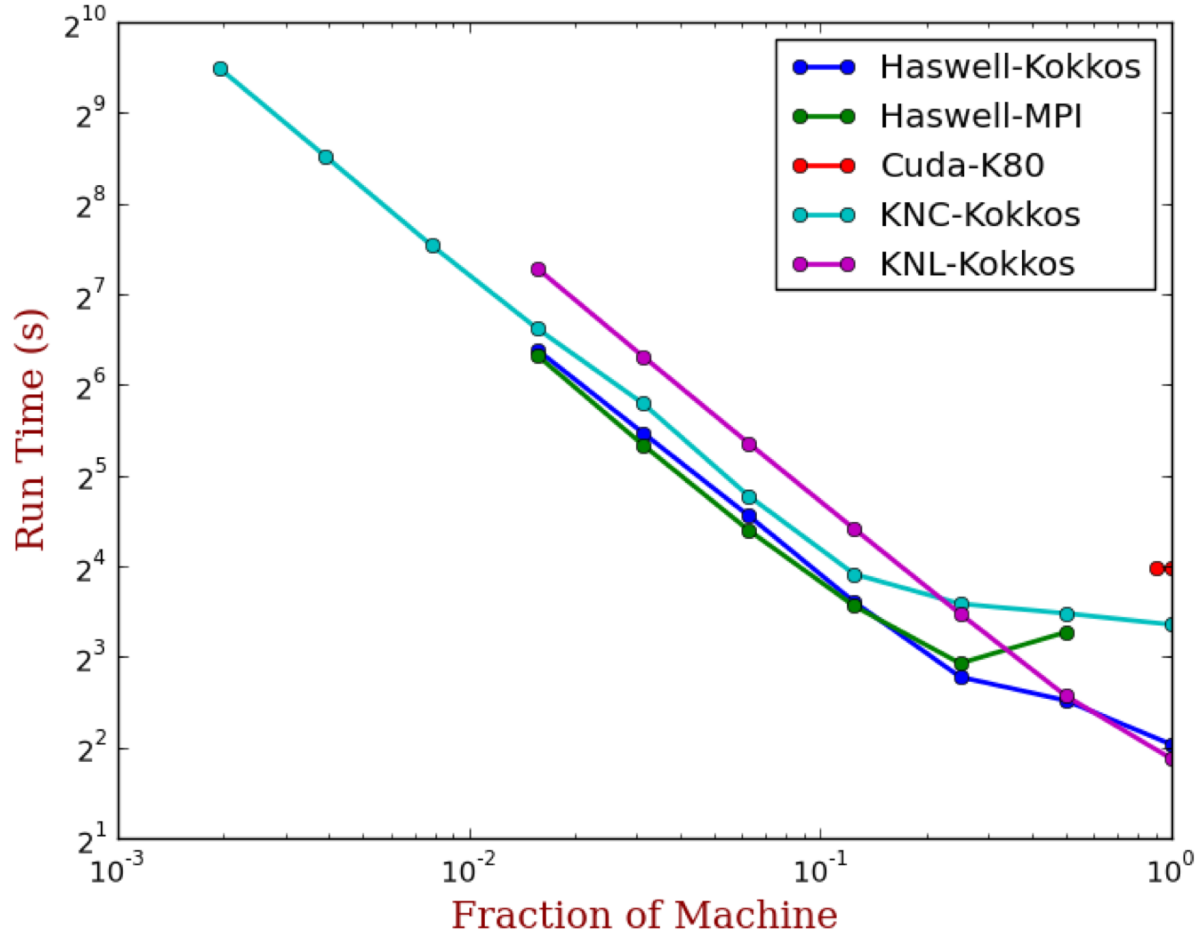
Bind worksets and launch kernel

```
template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void IntegrateDiffusionTerm<EvalT,Traits>::operator()(const Kokkos::TeamPolicy<PHX::exec_space>::member_type& team) const
{
  const int cell = team.league_rank();
  Kokkos::parallel_for(Kokkos::TeamThreadRange(team,0,grad_basis.extent(2)), KOKKOS_LAMBDA (const int& basis) {
    for (int qp = 0; qp < static_cast<int>(grad_basis.extent(1)); ++qp)
      for (int dim = 0; dim < static_cast<int>(grad_basis.extent(3)); ++dim)
        residual(cell,basis) +=  - grad_basis(cell,qp,basis,dim) * flux(cell,qp,dim) * weights(qp) * cell_measure(cell,qp);
  });
}
```

Evaluate values

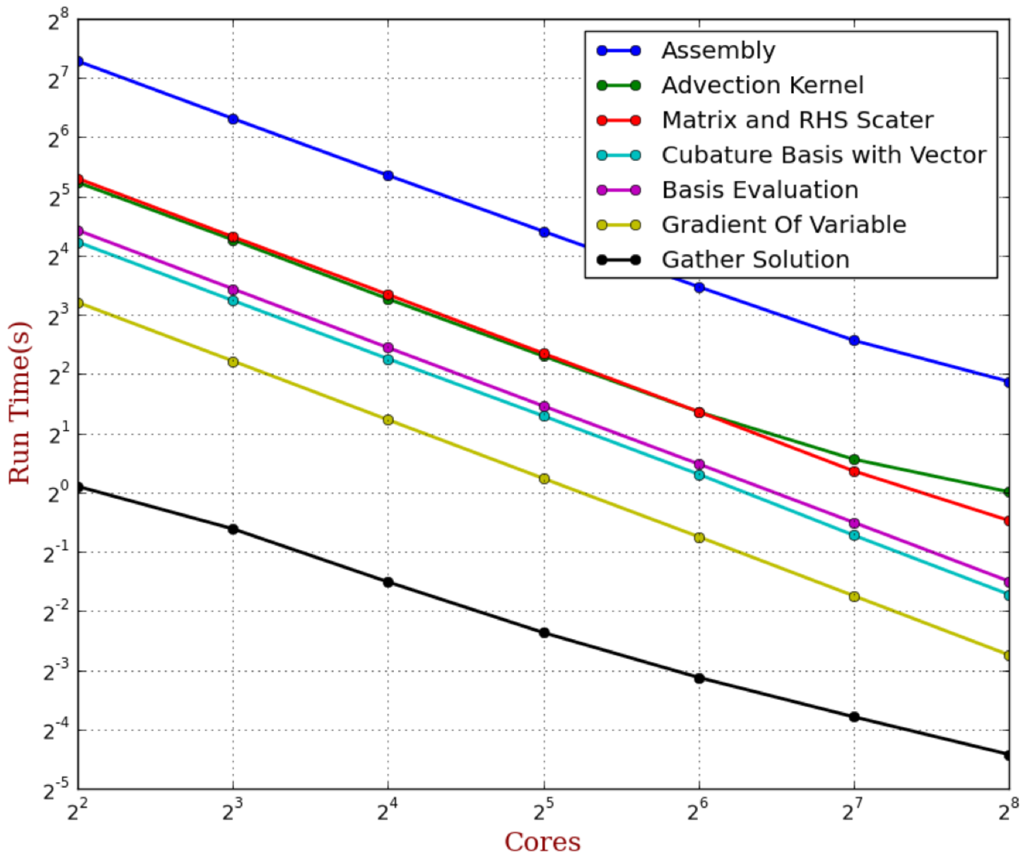
Preliminary Results for Jacobian Assembly

- 2016 Milestone to demonstrate the "ecosystem"
- 16K elements
- Flat/Single level data parallelism (loop over cells)
- Basic MPI (no thread spec.)

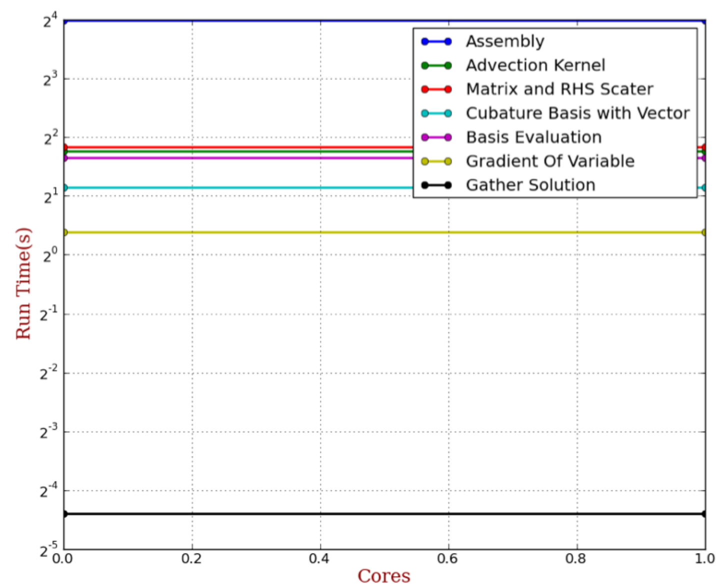


Assembly Runtimes by Kernel

KNL



K20x



CFD Kernel is the high tent pole

CFD Node

Sandia National Laboratories

$$\int_e c\left(\vec{f}(x)\cdot\nabla\varphi_i(x) + s(x)\varphi_i(x)\right)dx$$

```
Kokkos::View<ScalarT****, Layout, ExecSpace> wgb;
Kokkos::View<ScalarT***,  Layout, ExecSpace> flux;
Kokkos::View<ScalarT***,  Layout, ExecSpace> wbs;
Kokkos::View<ScalarT**,   Layout, ExecSpace> src;
Kokkos::View<ScalarT**,   Layout, ExecSpace> residual;
ScalarT coeff;

  for (int cell=0; cell < num_cell; ++cell) {
    for (int basis=0; basis<num_basis; ++basis) {
      ScalarT value(0),value2(0);
      for (int qp=0; qp<num_points; ++qp) {
        for (int dim=0; dim<num_dim; ++dim)
          value += flux(cell,qp,dim)*wgb(cell,basis,qp,dim);
        value2 += src(cell,qp)*wbs(cell,basis,qp);
      }
      residual(cell,basis) = coeff*(value+value2);
    }
  }
```

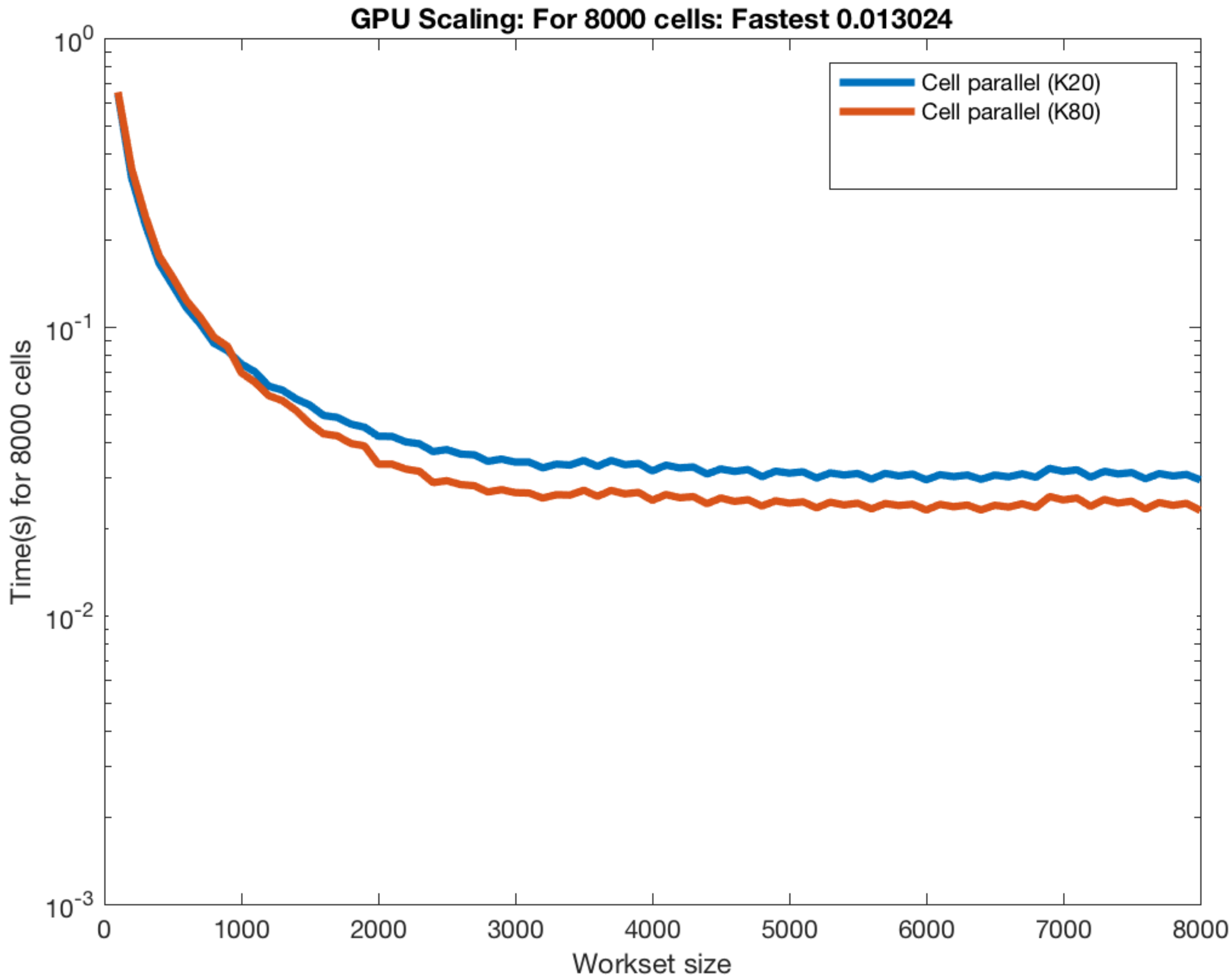
Flat Parallelism (1-level) Kokkos-ified CFD Node

$$\int_e c\left(\vec{f}(x)\cdot\nabla\varphi_i(x)+s(x)\varphi_i(x)\right)dx$$

```
Kokkos::View<ScalarT****, Layout, ExecSpace> wgb;
Kokkos::View<ScalarT***,  Layout, ExecSpace> flux;
Kokkos::View<ScalarT***,  Layout, ExecSpace> wbs;
Kokkos::View<ScalarT**,   Layout, ExecSpace> src;
Kokkos::View<ScalarT**,   Layout, ExecSpace> residual;
ScalarT coeff;

typedef Kokkos::RangePolicy<ExecSpace> Policy;

Kokkos::parallel_for( Policy( 0,num_cell ), KOKKOS_LAMBDA( const int cell )
  {
    for (int basis=0; basis<num_basis; ++basis) {
      ScalarT value(0),value2(0);
      for (int qp=0; qp<num_points; ++qp) {
        for (int dim=0; dim<num_dim; ++dim)
          value += flux(cell,qp,dim)*wgb(cell,basis,qp,dim);
        value2 += src(cell,qp)*wbs(cell,basis,qp);
      }
      residual(cell,basis) = coeff*(value+value2);
    }
  });
```

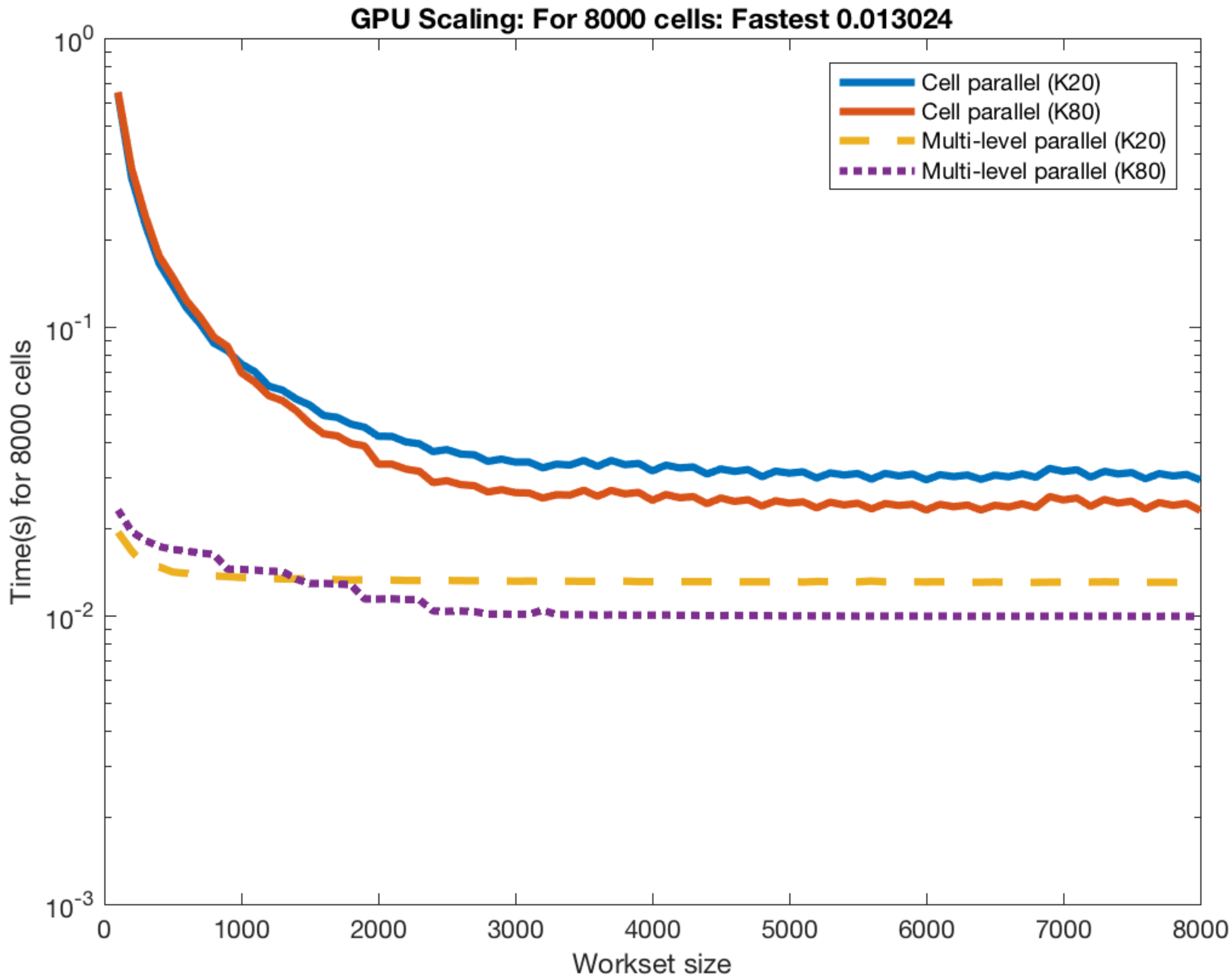
Single CFD Kernel GPU Performance Assessment

- Single level parallelism is insufficient
- Does not expose enough parallelism



Single CFD Kernel
GPU Performance Assessment

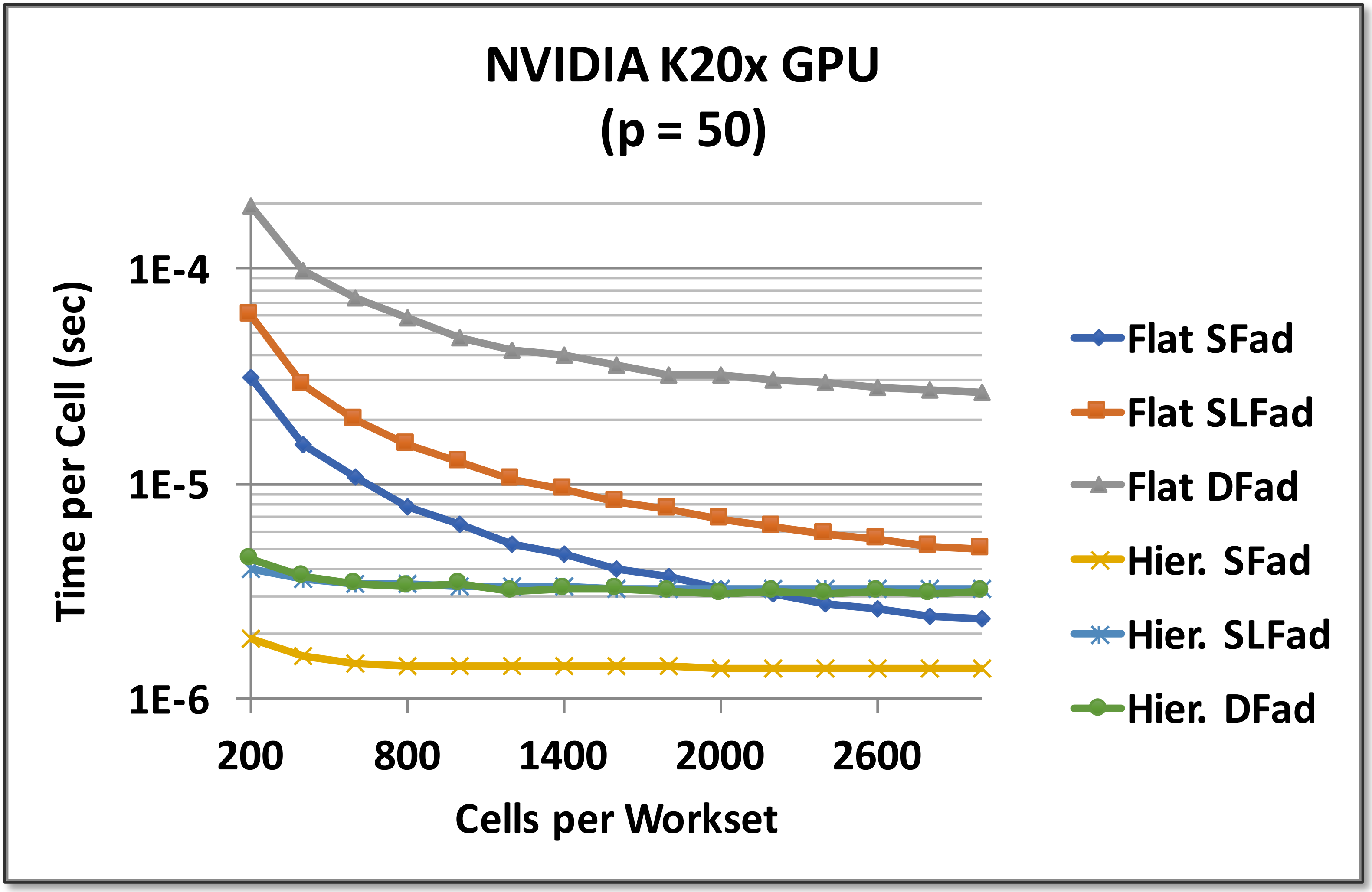
- Single level parallelism is insufficient
- Does not expose enough parallelism
- 3-level hierarchical parallelism shows significant improvement
- Hand coded sensitivity array outside libraries
- ***Key is to parallelize over FAD derivative dimension***



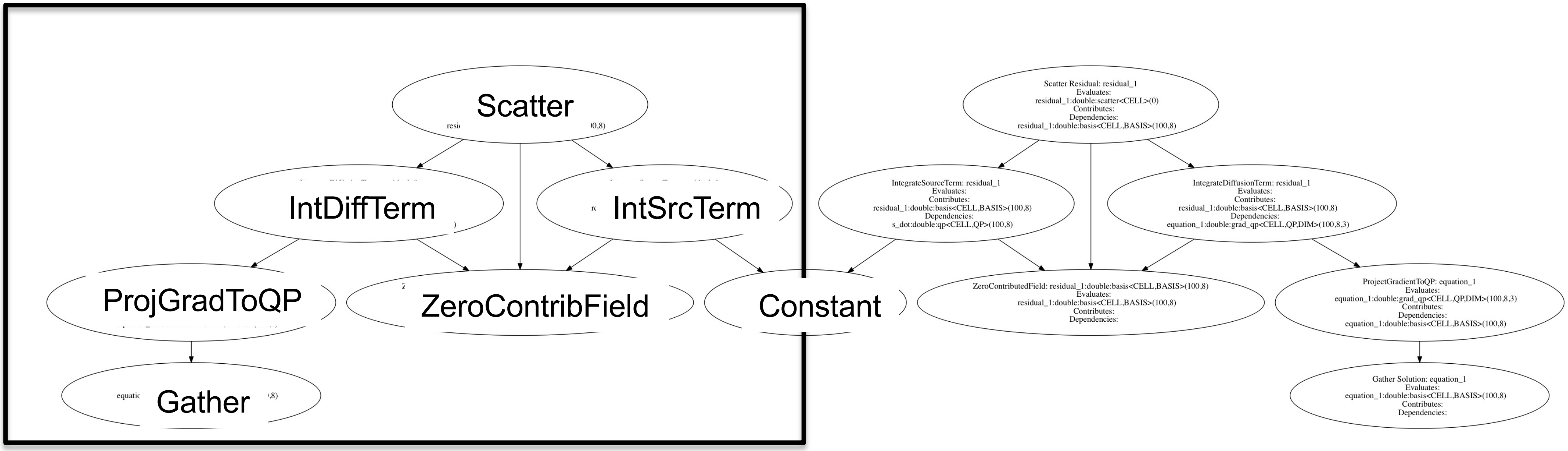
Kernel with Hierarchical DFad

```
Sacado::createGlobalMemoryPool(ExecSpace(), mem_pool_size);

typedef Kokkos::TeamPolicy<ExecSpace> Policy;
const int vector_size = is_cuda ? 32 : 1;
const int team_size = is_cuda ? 256 / vector_size : 1;

Kokkos::parallel_for(
  Policy( num_cell,team_size,vector_size ),
  KOKKOS_LAMBDA( const typename Policy::member_type& team )
  {
    const size_t cell = team.league_rank();
    const int team_index = team.team_rank();

    for (int basis=team_index; basis<num_basis; basis+=team_size) {
      ScalarT value(0),value2(0);
      for (int qp=0; qp<num_points; ++qp) {
        for (int dim=0; dim<num_dim; ++dim)
          value += flux(cell,qp,dim)*wgb(cell,basis,qp,dim);
        value2 += src(cell,qp)*wbs(cell,basis,qp);
      }
      residual(cell,basis) = coeff*(value+value2);
    }
  });

Sacado::destroyGlobalMemoryPool(ExecSpace());
```

Derivative Array Parallelization in Sacado



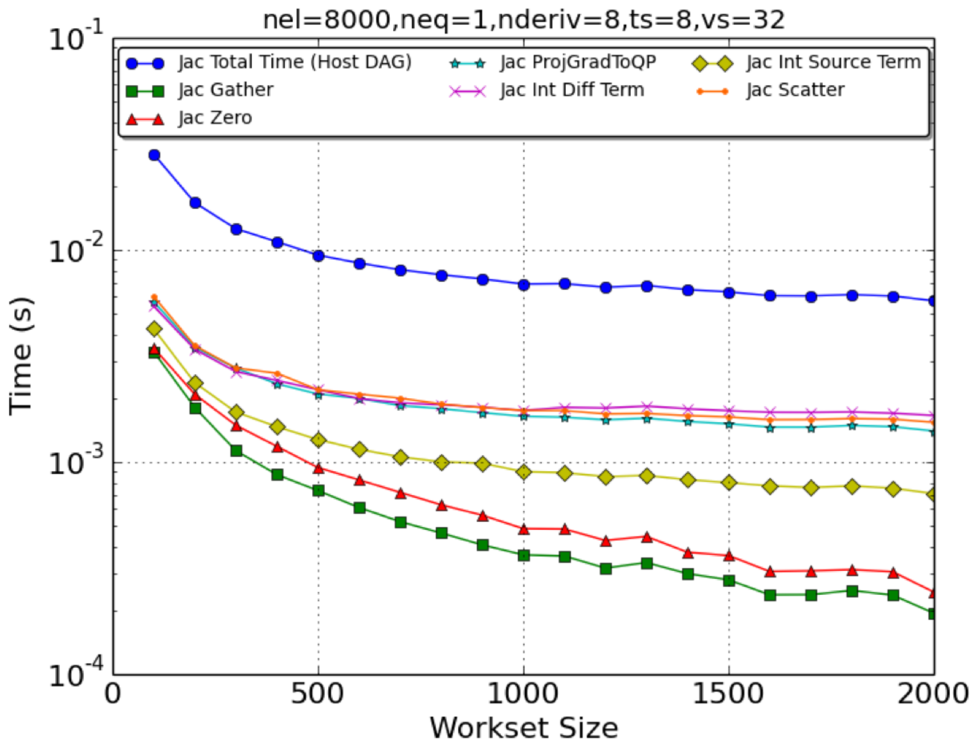
Test Problem DAG

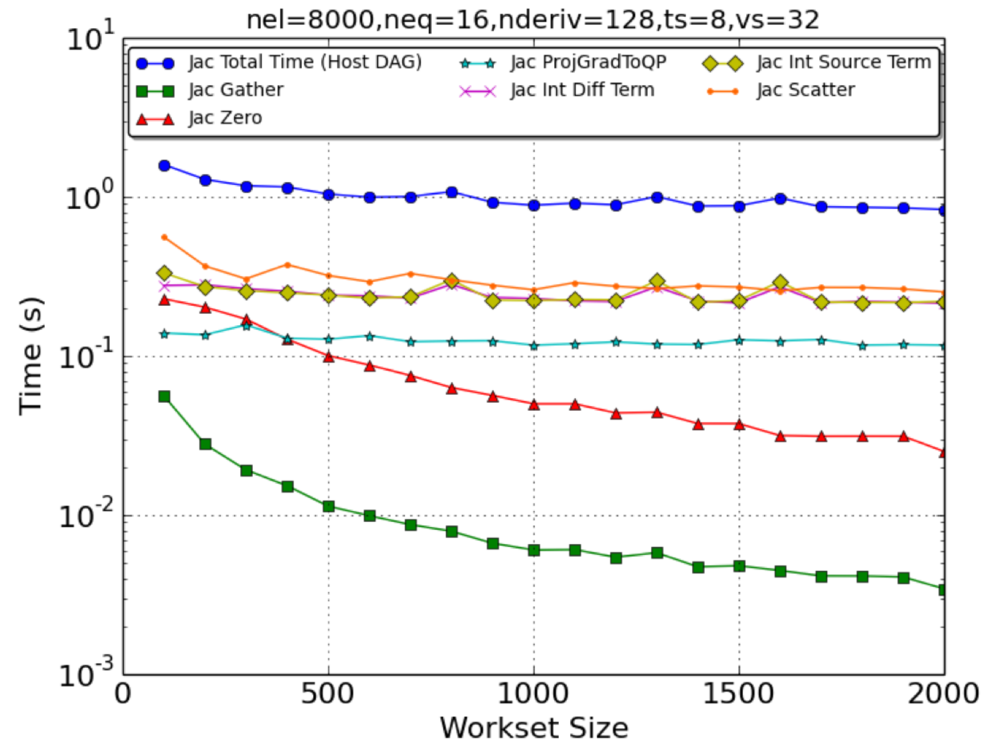


Repeated Unit

- For Multiple equations, a new set of nodes (repeated unit) are added
- Could improve performance by adding grouping all equations into single set of evaluators

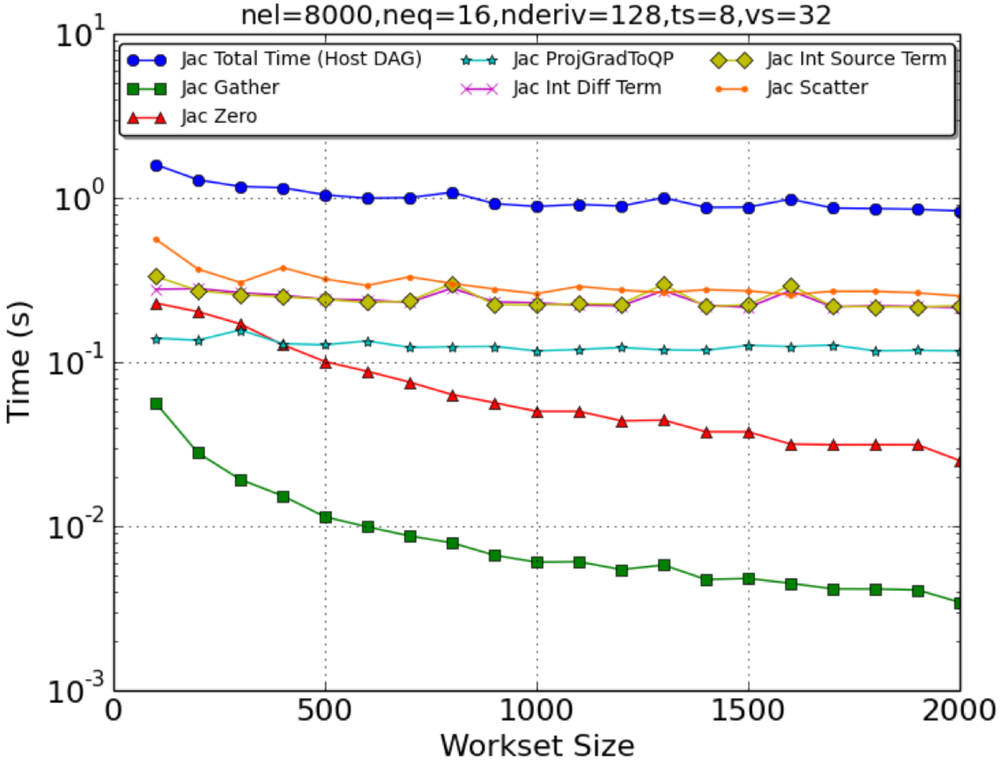
Number of Equations: Jacobian, CUDA P100



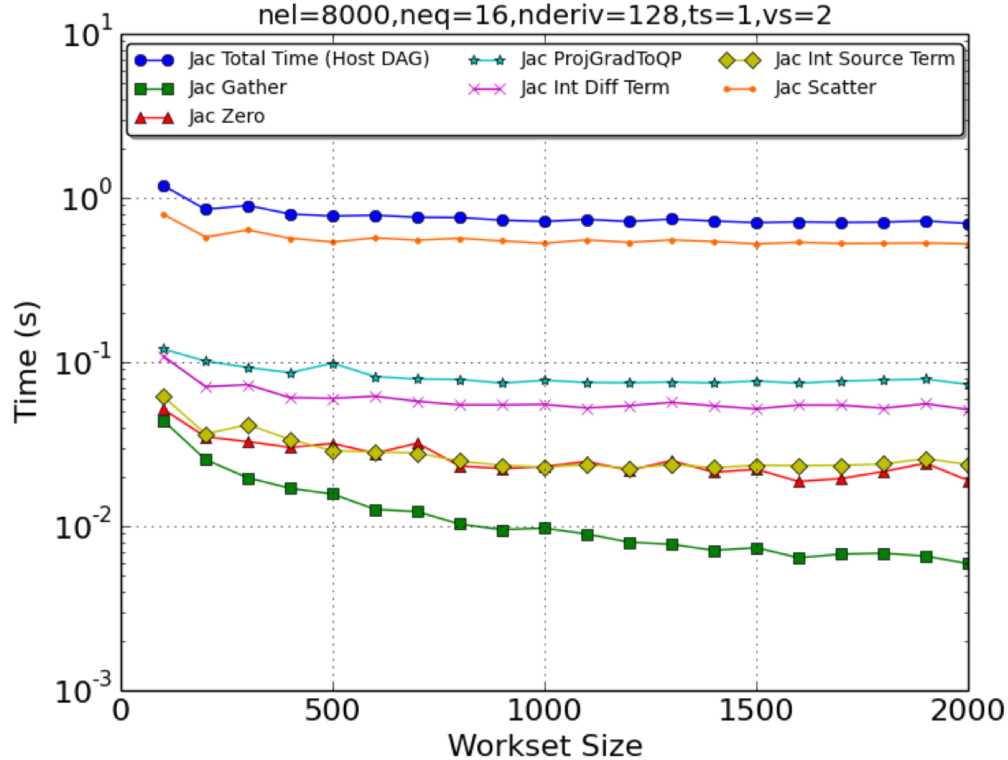


Node Comparison, Jacobian

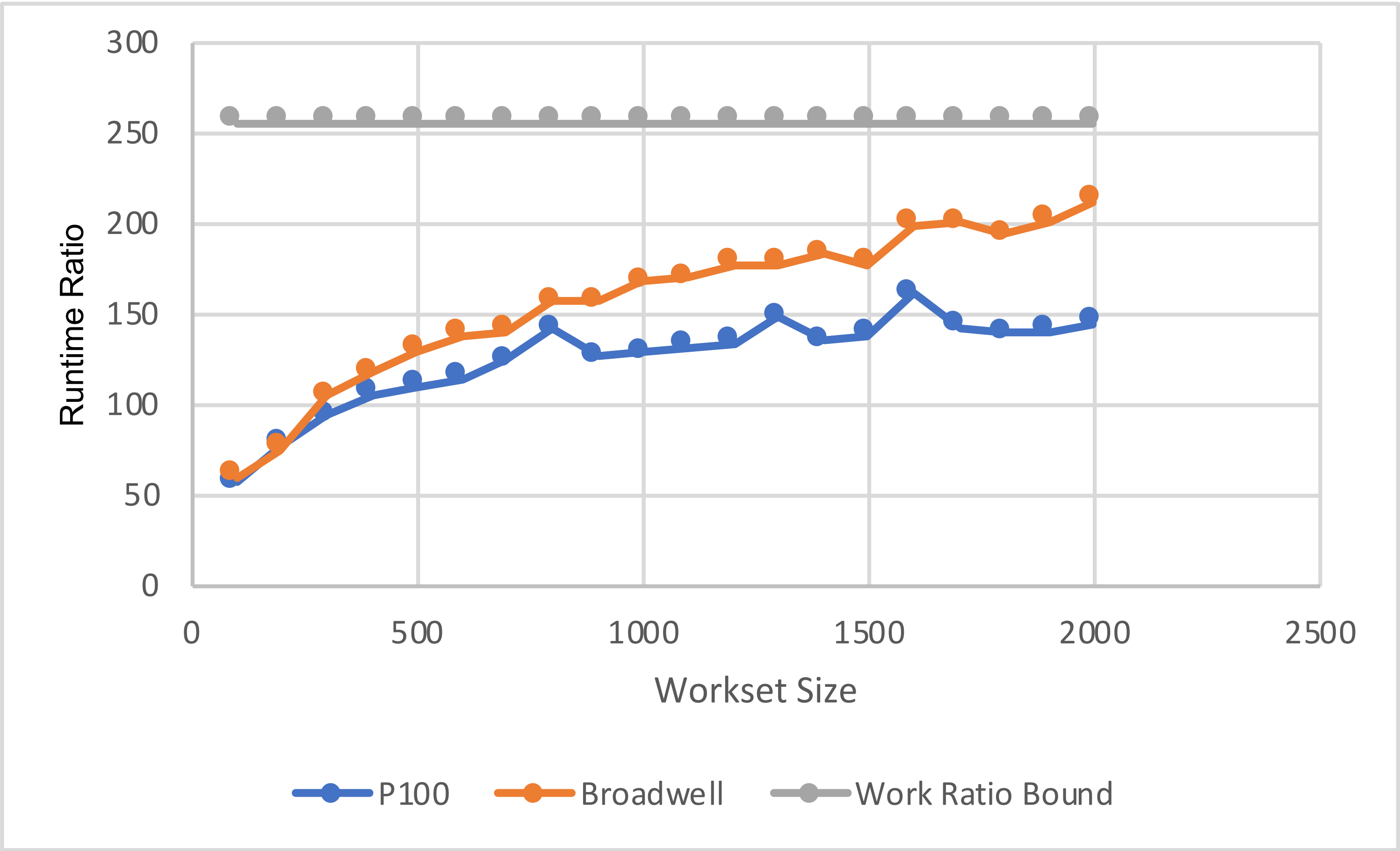
CUDA, P100



Broadwell, 32 cores, 2 hyperthreads/core



Equation Set Scaling



Host vs Device DAG

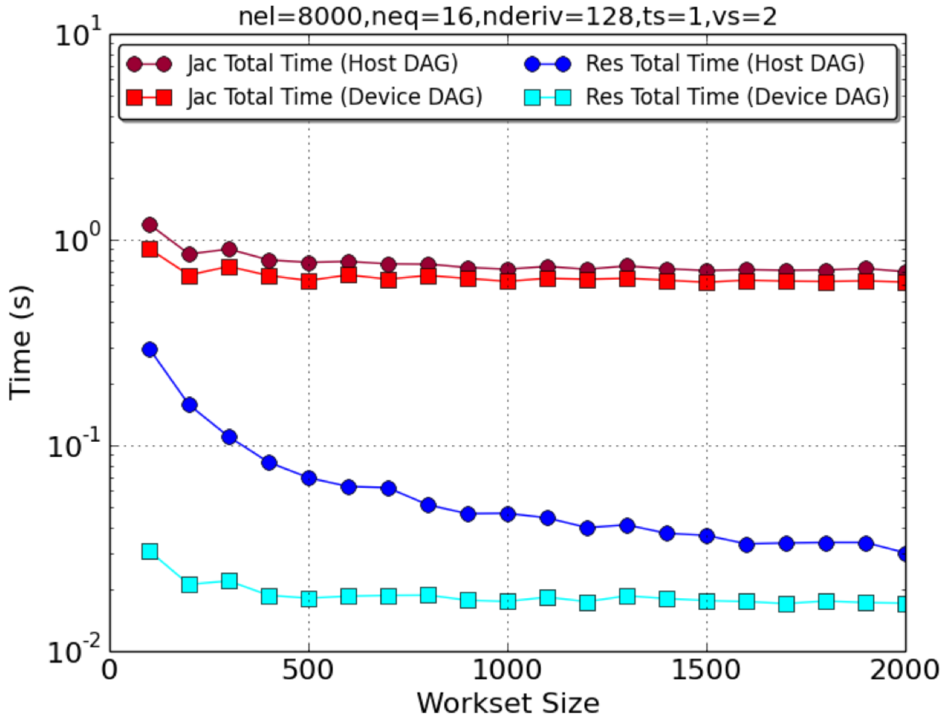
- Traditional Phalanx use is “Host DAG”
  - Each node in DAG launches its own kokkos kernel via parallel_for from host
- New “Device DAG” capability runs all the Kernels on device from a single parallel_for launch
  - Goal: keep values in cache for next functor evaluation

- Device DAG complications:
  - Need a virtual function call to run through a runtime generated list of functors
  - Copy all functors to device and instantiate
  - Requires relocatable device code for CUDA

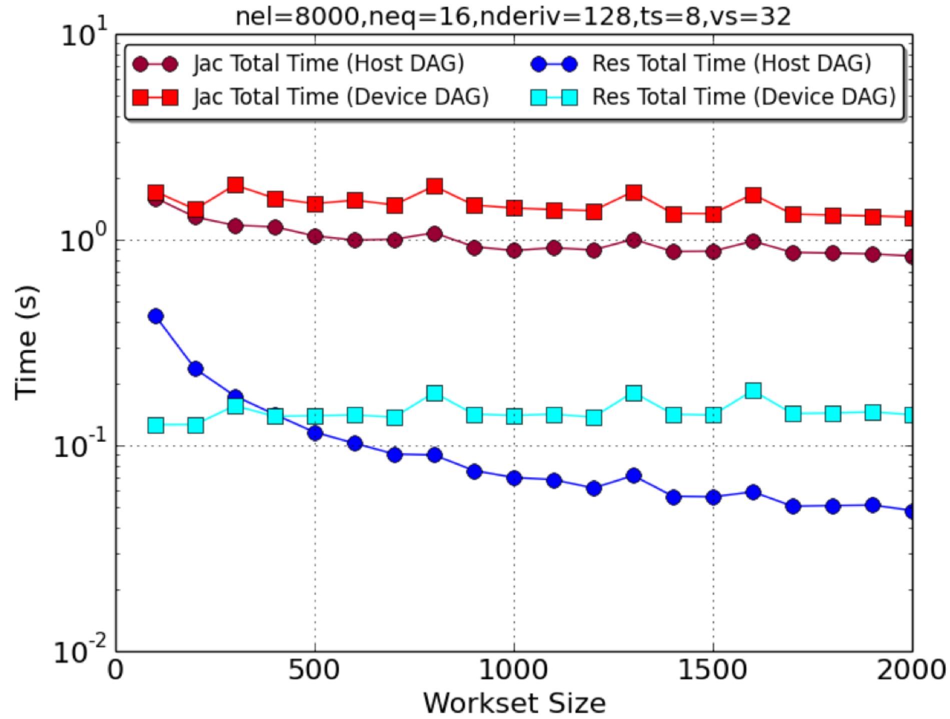
```
template<typename Traits>
struct RunDeviceDag {

  Kokkos::View<PHX::DeviceEvaluatorPtr<Traits>*,PHX::Device> evaluators_;

  ...

  KOKKOS_INLINE_FUNCTION
  void operator()(const TeamPolicy<exec_space>::member_type& team) const
  {
    const int num_evaluators = static_cast<int>(evaluators_.extent(0));
    for (int e=0; e < num_evaluators; ++e) {
      evaluators_(e).ptr->prepareForRecompute(team,data_);
      evaluators_(e).ptr->evaluate(team,data_);
      team.team_barrier();
    }
  }

};
```

Host vs Device DAG Performance, 16 Equations

OpenMP, Broadwell, OMP_NUM_THREADS=36



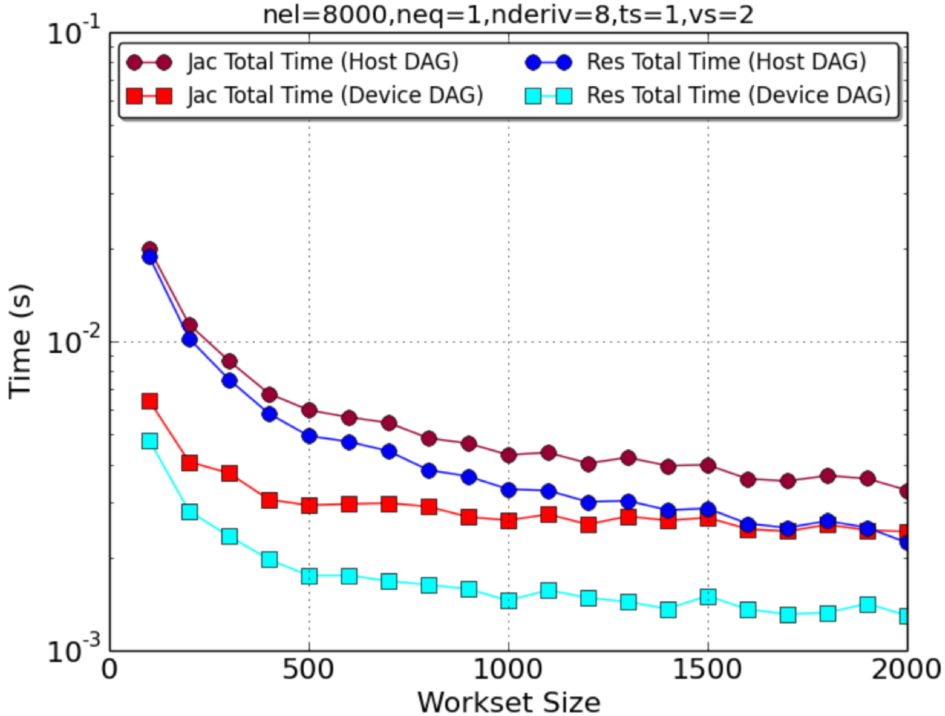
CUDA, P100



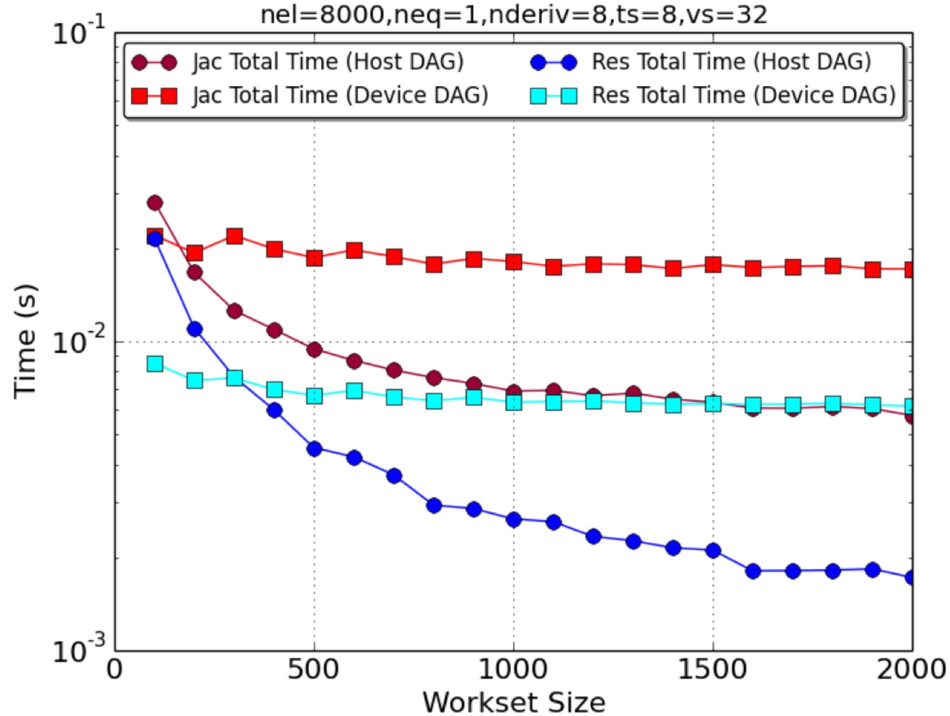
● Host Residual
■ Device Jacobian

Host vs Device DAG Performance, 1 Equation

OpenMP, Broadwell,
OMP_NUM_THREADS=36



CUDA, P100



● Host Residual
■ Device Jacobian

Conclusions

- A number of tools in Trilinos are under development for supporting finite element assembly
  - Risk mitigation: can "buy in" a various levels
  - At a minimum components provide guidance
- Performance portability is not the same as being able to run on an architecture
  - Restrictions from GPUs strongly impacts the code base
- Converting code to be performance portable is application specific. Kokkos team experience:
  - 50% can do simple flat parallelism
  - 30 % need hierarchic
  - 20% need some customization – new algorithms
- Templated scalar types and DAG-assembly allow for complex multiphysics simulations in a manageable code base

Extra Slides

Profiler (500 Element workset, ts=8, vs=32)

- IntegrateDiffusionTerm

16	achieved_occupancy	Achieved Occupancy	0.493292	0.496409	0.495174
16	dram_read_throughput	Device Memory Read Throughput	15.464GB/s	16.712GB/s	16.342GB/s
16	dram_write_throughput	Device Memory Write Throughput	15.354GB/s	16.883GB/s	16.177GB/s
16	gld_efficiency	Global Memory Load Efficiency	41.01%	41.01%	41.01%
16	gst_efficiency	Global Memory Store Efficiency	89.58%	89.58%	89.58%
16	warp_execution_efficiency	Warp Execution Efficiency	100.00%	100.00%	100.00%
16	stall_inst_fetch	Issue Stall Reasons (Instructions Fetch)	0.70%	0.88%	0.78%
16	stall_memory_dependency	Issue Stall Reasons (Data Request)	89.97%	90.54%	90.17%
16	stall_exec_dependency	Issue Stall Reasons (Execution Dependency)	7.09%	7.47%	7.28%
16	stall_memory_throttle	Issue Stall Reasons (Memory Throttle)	0.01%	0.02%	0.02%
16	stall_pipe_busy	Issue Stall Reasons (Pipe Busy)	0.18%	0.20%	0.19%
16	stall_not_selected	Issue Stall Reasons (Not Selected)	0.56%	0.61%	0.59%
16	branch_efficiency	Branch Efficiency	100.00%	100.00%	100.00%
16	gld_throughput	Global Load Throughput	447.61GB/s	482.41GB/s	472.82GB/s
16	gst_throughput	Global Store Throughput	98.255GB/s	105.89GB/s	103.79GB/s
16	local_load_throughput	Local Memory Load Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	local_store_throughput	Local Memory Store Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	sm_activity	Multiprocessor Activity	89.98%	93.70%	91.87%

- ProjectGradientToQP

16	achieved_occupancy	Achieved Occupancy	0.578304	0.614325	0.597506
16	dram_read_throughput	Device Memory Read Throughput	208.10GB/s	222.70GB/s	216.43GB/s
16	dram_write_throughput	Device Memory Write Throughput	208.76GB/s	220.09GB/s	215.84GB/s
16	gld_efficiency	Global Memory Load Efficiency	46.96%	46.96%	46.96%
16	gst_efficiency	Global Memory Store Efficiency	88.48%	88.48%	88.48%
16	warp_execution_efficiency	Warp Execution Efficiency	100.00%	100.00%	100.00%
16	stall_inst_fetch	Issue Stall Reasons (Instructions Fetch)	0.91%	1.49%	1.11%
16	stall_memory_dependency	Issue Stall Reasons (Data Request)	88.08%	89.48%	88.83%
16	stall_exec_dependency	Issue Stall Reasons (Execution Dependency)	6.99%	7.38%	7.13%
16	stall_memory_throttle	Issue Stall Reasons (Memory Throttle)	0.00%	0.01%	0.01%
16	stall_pipe_busy	Issue Stall Reasons (Pipe Busy)	0.28%	0.32%	0.30%
16	stall_not_selected	Issue Stall Reasons (Not Selected)	0.91%	1.09%	1.00%
16	branch_efficiency	Branch Efficiency	100.00%	100.00%	100.00%
16	gld_throughput	Global Load Throughput	898.63GB/s	947.49GB/s	928.98GB/s
16	gst_throughput	Global Store Throughput	235.09GB/s	247.87GB/s	243.03GB/s
16	local_load_throughput	Local Memory Load Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	local_store_throughput	Local Memory Store Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	sm_activity	Multiprocessor Activity	89.74%	93.94%	91.35%

Profiler

- Gather

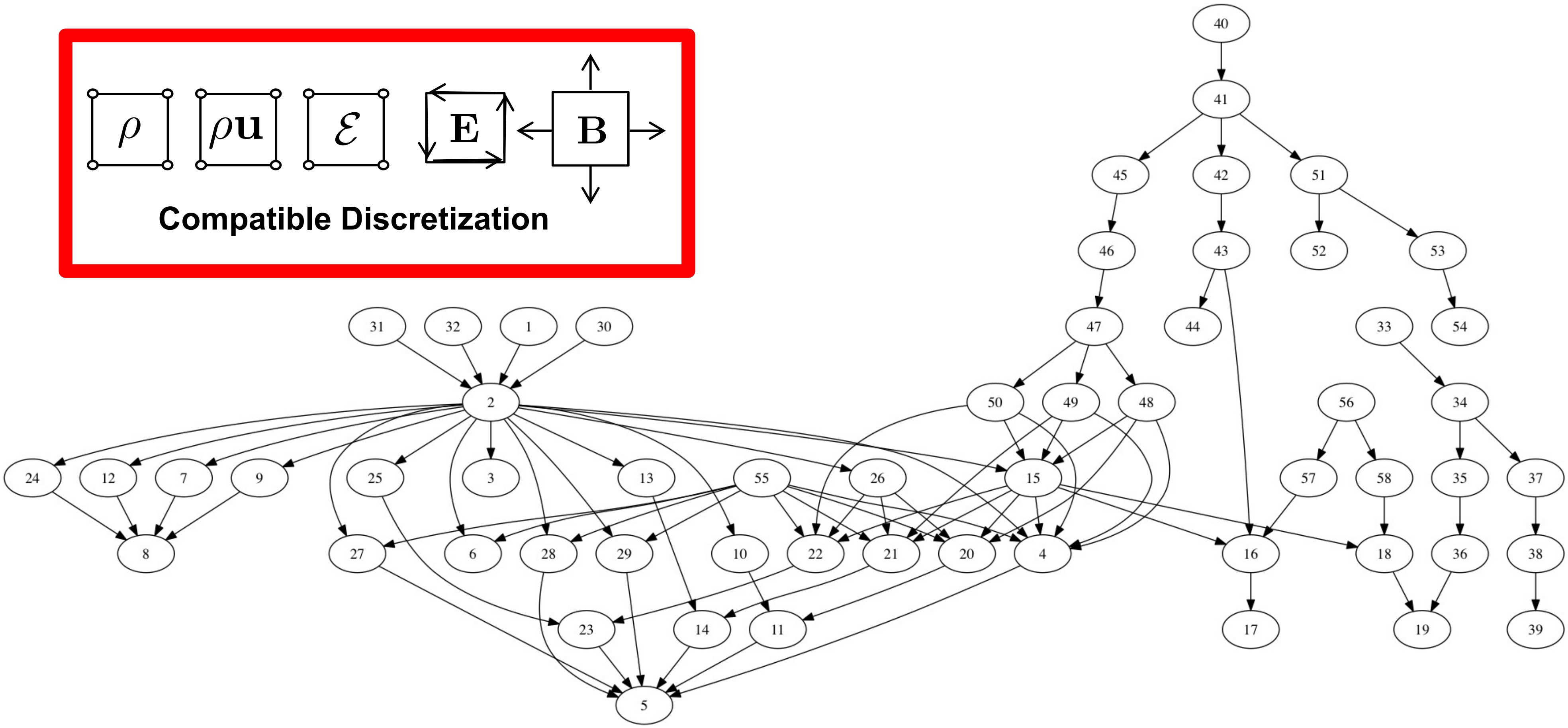
16	achieved_occupancy	Achieved Occupancy	0.289833	0.468549	0.352503
16	dram_read_throughput	Device Memory Read Throughput	22.358GB/s	28.711GB/s	26.816GB/s
16	dram_write_throughput	Device Memory Write Throughput	65.102GB/s	70.407GB/s	67.746GB/s
16	gld_efficiency	Global Memory Load Efficiency	18.75%	18.75%	18.75%
16	gst_efficiency	Global Memory Store Efficiency	25.00%	25.00%	25.00%
16	warp_execution_efficiency	Warp Execution Efficiency	74.90%	74.90%	74.90%
16	stall_inst_fetch	Issue Stall Reasons (Instructions Fetch)	5.41%	59.42%	17.68%
16	stall_memory_dependency	Issue Stall Reasons (Data Request)	2.77%	9.50%	6.37%
16	stall_exec_dependency	Issue Stall Reasons (Execution Dependency)	7.25%	20.80%	16.06%
16	stall_memory_throttle	Issue Stall Reasons (Memory Throttle)	0.02%	0.07%	0.05%
16	stall_pipe_busy	Issue Stall Reasons (Pipe Busy)	0.11%	0.38%	0.28%
16	stall_not_selected	Issue Stall Reasons (Not Selected)	0.40%	1.50%	1.12%
16	branch_efficiency	Branch Efficiency	100.00%	100.00%	100.00%
16	gld_throughput	Global Load Throughput	44.548GB/s	48.068GB/s	46.830GB/s
16	gst_throughput	Global Store Throughput	44.548GB/s	48.068GB/s	46.830GB/s
16	local_load_throughput	Local Memory Load Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	local_store_throughput	Local Memory Store Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	sm_activity	Multiprocessor Activity	40.14%	53.25%	46.02%

- Scatter (includes filling the "free" residual too)

16	achieved_occupancy	Achieved Occupancy	0.518109	0.521589	0.519930
16	dram_read_throughput	Device Memory Read Throughput	6.1323GB/s	6.2414GB/s	6.1784GB/s
16	dram_write_throughput	Device Memory Write Throughput	2.3836GB/s	2.4590GB/s	2.4065GB/s
16	gld_efficiency	Global Memory Load Efficiency	7.20%	7.20%	7.20%
16	gst_efficiency	Global Memory Store Efficiency	0.00%	0.00%	0.00%
16	warp_execution_efficiency	Warp Execution Efficiency	48.52%	48.52%	48.52%
16	stall_inst_fetch	Issue Stall Reasons (Instructions Fetch)	5.53%	5.69%	5.59%
16	stall_memory_dependency	Issue Stall Reasons (Data Request)	53.74%	54.09%	53.95%
16	stall_exec_dependency	Issue Stall Reasons (Execution Dependency)	15.49%	15.55%	15.52%
16	stall_memory_throttle	Issue Stall Reasons (Memory Throttle)	0.00%	0.00%	0.00%
16	stall_pipe_busy	Issue Stall Reasons (Pipe Busy)	0.36%	0.36%	0.36%
16	stall_not_selected	Issue Stall Reasons (Not Selected)	0.77%	0.77%	0.77%
16	branch_efficiency	Branch Efficiency	100.00%	100.00%	100.00%
16	gld_throughput	Global Load Throughput	341.48GB/s	347.80GB/s	344.23GB/s
16	gst_throughput	Global Store Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	local_load_throughput	Local Memory Load Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	local_store_throughput	Local Memory Store Throughput	0.00000B/s	0.00000B/s	0.00000B/s
16	sm_activity	Multiprocessor Activity	87.35%	93.00%	91.11%

More Parallelism: Hybrid Task+Data Parallel Analysis

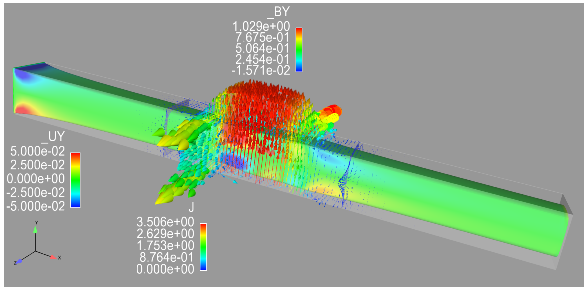
Sandia National Laboratories



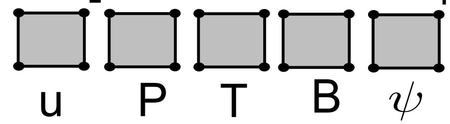
Compatible Discretization

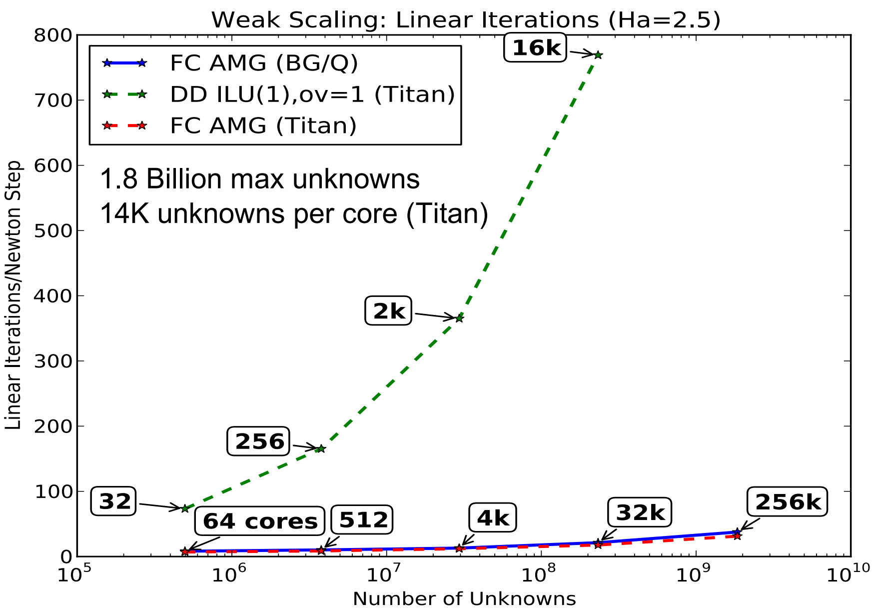
Theoretical speed up for volume assembly DAG (Ignores scheduling overhead)

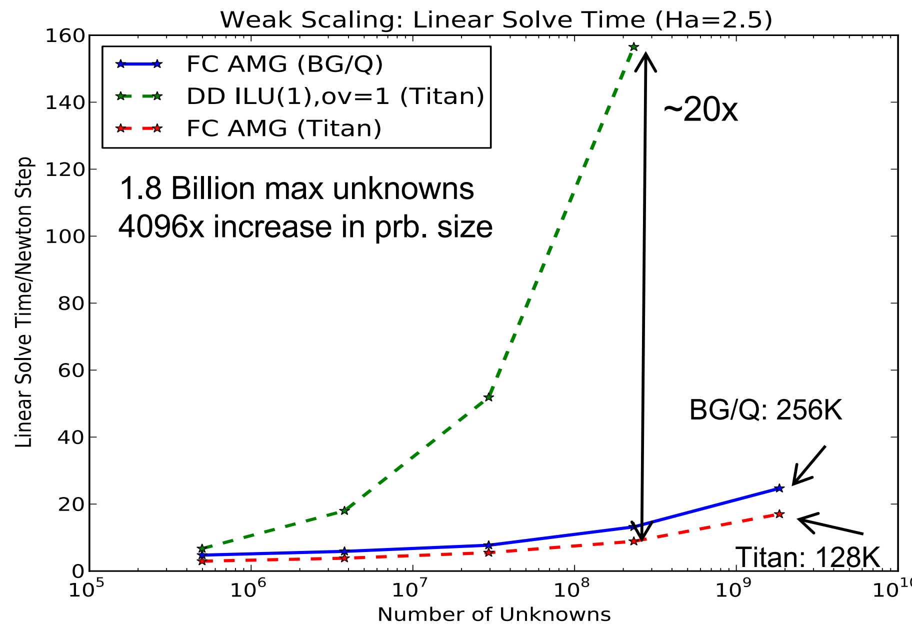
	1 Thread/Task	8 Threads/Task	16 Threads/Task
Jacobian	3.5	4.5	4.9
Residual	3.4	3.4	3.5

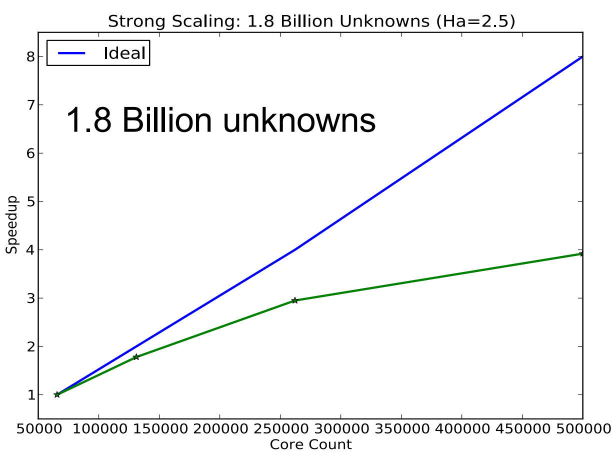


SFE Initial Scaling Studies for Cray XK7 AND BG/Q: 3D MHD Generator [Re = 500, Re_m = 1, Ha = 2.5] (with Paul Lin)









Largest fully-coupled implicit solves demonstrated to date:

- MHD (steady): 10B DoF, 1.25B elem, on 128K cores
- CFD (Transient): 40B DoF, 10.0B elem, on 128K cores
- Poisson sub-block: 3.2B DoF, 3.2B elem, on 1.6M cores

[Preliminary strong scaling of Krylov linear solver + preconditioner (ML: FC – AMG), Tuminaro, Hu, Siefert, Gee et. al.]

(DOE/ORNL Titan Cray XK7: Joule Metric)

Difficulties

- Production codes don't always fit the "count/allocate/fill" paradigm.
  - Compile-time rank (fixed with Kokkos::DynRankView)
  - Runtime decisions and lazy instantiation can be problematic
  - Passing the FAD dimension for temporaries, view factory for AD dimensions

- Portability and Performance are not the same

- No raw references for AD scalar types (use return_type)

- Hiding the derivative array parallelization introduces requirements on developers
  - Templated scalar temporaries
  - Use of a memory pool

- Loss of bracket operator

Multi-fluid plasma model

- Continuity equation:

$$\partial_t \rho_\alpha + \nabla \cdot (\rho_\alpha \boldsymbol{u}_\alpha) = S_\alpha$$

Each species α is represented by a separate density ρ, momentum $\rho\boldsymbol{u}$, and isotropic energy ϵ.

- Momentum equation:

$$\partial_t (\rho_\alpha \boldsymbol{u}_\alpha) + \nabla \cdot (\rho_\alpha \boldsymbol{u}_\alpha \otimes \boldsymbol{u}_\alpha + \boldsymbol{P}_\alpha) = \frac{q_\alpha}{m_\alpha} \rho_\alpha (\boldsymbol{E} + \boldsymbol{u}_\alpha \times \boldsymbol{B}) + \boldsymbol{R}_\alpha + \boldsymbol{u}_\alpha S_\alpha$$

- Energy equation:

$$\partial_t \epsilon_\alpha + \nabla \cdot (\boldsymbol{u}_\alpha \cdot (\epsilon_\alpha \boldsymbol{I} + \boldsymbol{P}_\alpha) + \boldsymbol{q}_\alpha) = \frac{q_\alpha}{m_\alpha} \rho_\alpha \boldsymbol{u}_\alpha \cdot \boldsymbol{E} + Q_\alpha + \boldsymbol{u}_\alpha \cdot \boldsymbol{R}_\alpha + \frac{1}{2} \boldsymbol{u}_\alpha^2 S_\alpha$$

- Ampere's Law:

$$\partial_t \boldsymbol{E} - c^2 \nabla \times \boldsymbol{B} = -\frac{1}{\epsilon_0} \sum_\alpha \frac{q_\alpha}{m_\alpha} \rho_\alpha \boldsymbol{u}_\alpha$$

Spatial operators are discretized using a finite element method.

- Faraday's Law:

$$\partial_t \boldsymbol{B} + \nabla \times \boldsymbol{E} = 0$$

Fluid
Electromagnetic
Inter-fluid

IMEX time integration

- IMEX gives a framework for splitting the model up into implicit and explicit terms:
  - Explicit for slow, non-stiff terms
  - Implicit for fast, stiff terms

Implicit tableau

$$\begin{array}{c|c} c & A \\ \hline & b^t \end{array}$$

Explicit tableau

$$\begin{array}{c|c} \hat{c} & \hat{A} \\ \hline & \hat{b}^t \end{array}$$

$$\partial_t u = f(u,t) + g(u,t)$$

$$u^{(i)} = u^n + \Delta t \sum_{j=0}^{j<i} \hat{A}_{ij} f\left(u^{(j)}, t_n + \hat{c}_j \Delta t\right) + \Delta t \sum_{j=0}^{j\leq i} A_{ij} g\left(u^{(j)}, t_n + c_j \Delta t\right)$$

$$u^{n+1} = u^n + \Delta t \sum_{i=0}^{i<s} \hat{b}_i f\left(u^{(i)}, t_n + \hat{c}_i \Delta t\right) + \Delta t \sum_{i=0}^{i\leq s} b_i g\left(u^{(i)}, t_n + c_i \Delta t\right)$$

- **Objective**: Combine the advantages of implicit and explicit solvers.
  - Take advantage of expensive implicit solver to overstep fast scales, and explicit solver to resolve slow scales.

Compatible discretization for EM

- A physics compatible finite element discretization is used to enforce the divergence constraints for the electric and magnetic fields.
- Fluids are represented by an **HGrad** (node) basis $\rho \in V_{\nabla}$.
- The electric field is represented by an **HCurl** (edge) vector basis $\mathbf{E} \in \boldsymbol{V}_{\nabla\times}$.
- The magnetic field is represented by an **HDiv** (face) vector basis $\mathbf{B} \in \boldsymbol{V}_{\nabla\cdot}$.
- Compatibility is defined by the discrete preservation of the **De Rham Complex**:

$$\nabla \phi_{\nabla} \in \boldsymbol{V}_{\nabla\times} \longrightarrow \nabla\times\boldsymbol{\phi}_{\nabla\times} \in \boldsymbol{V}_{\nabla\cdot} \longrightarrow \nabla\cdot\boldsymbol{\phi}_{\nabla\cdot} \in V_{\mathrm{L}_2}$$

- For Faraday's law, we choose a basis for the electric field such that its curl is fully represented by the basis used by the magnetic field.

$$\partial_t \boldsymbol{B} + \nabla\times\boldsymbol{E} = 0 \qquad\qquad \nabla\cdot\boldsymbol{B} = 0$$

- Since the curl of the electric field is 'globally continuous' w.r.t. a divergence operator, the divergence of that curl is zero over the domain:

$$\nabla\cdot(\partial_t \boldsymbol{B} + \nabla\times\boldsymbol{E}) = \partial_t(\nabla\cdot\boldsymbol{B}) + \nabla\cdot\nabla\times\boldsymbol{E} = \partial_t(\nabla\cdot\boldsymbol{B}) + \sum_i E_i \underbrace{\nabla\cdot\nabla\times\boldsymbol{\phi}^i_{\nabla\times}}_{0} = \partial_t(\nabla\cdot\boldsymbol{B}) = 0$$

- **Result**: The curl operator does not add divergence errors to the magnetic field

Satisfying Gauss' laws in plasmas

Sandia National Laboratories

- **Goal**: Solve plasma-coupled Maxwell's equations and satisfy a divergence constraint:

$$\partial_t \boldsymbol{E} - c^2 \nabla \times \boldsymbol{B} = -\frac{1}{\epsilon_0}\boldsymbol{j} \qquad \partial_t \rho_c + \nabla \cdot \boldsymbol{j} = 0$$

$$\nabla \cdot \boldsymbol{E} = \frac{1}{\epsilon_0}\rho_c$$

- In the **strong, non-discretized form**:

$$\nabla \cdot \left(\partial_t \boldsymbol{E} + \frac{1}{\epsilon_0}\boldsymbol{j} - c^2 \nabla \times \boldsymbol{B} \right) = \partial_t \nabla \cdot \boldsymbol{E} + \frac{1}{\epsilon_0} \nabla \cdot \boldsymbol{j} = \partial_t \left(\nabla \cdot \boldsymbol{E} - \frac{1}{\epsilon_0}\rho_c \right) = 0$$

- In the **weak form**: Choose a basis that supports the divergence constraint as HCurl does not support the divergence operation:

$$\int_\Omega \left(\partial_t \boldsymbol{E} - c^2 \nabla \times \boldsymbol{B} + \frac{1}{\epsilon_0}\boldsymbol{j} \right) \cdot \nabla \phi_\nabla \, dV = \int_\Omega \left(\partial_t \boldsymbol{E} \cdot \nabla \phi_\nabla + \frac{1}{\epsilon_0} \nabla \cdot \boldsymbol{j}\, \phi_\nabla \right) dV + c^2 \int_\Omega \boldsymbol{B} \cdot \underbrace{\nabla \times \nabla \phi_\nabla}_{0} dV$$

$$= \int_\Omega \partial_t \left(\boldsymbol{E} \cdot \nabla \phi_\nabla - \frac{1}{\epsilon_0} \rho_c \, \phi_\nabla \right) dV = 0$$

- Assumes that continuity equation is weakly satisfied:

$$\int_\Omega (\partial_t \rho_c - \nabla \cdot \boldsymbol{j}) \phi_\nabla \, dV = \int_\Omega (\partial_t \rho_c \phi_\nabla + \boldsymbol{j} \cdot \nabla \phi_\nabla) dV = 0 \rightarrow \int_\Omega \partial_t \rho_c \phi_\nabla \, dV = -\int_\Omega \boldsymbol{j} \cdot \nabla \phi_\nabla \, dV$$

Discontinuous Galerkin method

- Discontinuous Galerkin FEM does not assume a globally continuous test function:

Weak form

$$\int_{\Omega} \phi \partial_t u \, dV + \int_{\Omega} \phi \, \nabla \cdot \boldsymbol{f} \, dV - \int_{\Omega} \phi s \, dV = 0$$

Break into elements $\mathrm{K} \in \Omega$ with discontinuous element test function ϕ_i^{K}

$$\sum_K \left[\int_{\mathrm{K}} \phi_i^K \partial_t u \, dV + \int_{\mathrm{K}} \phi_i^K \nabla \cdot \boldsymbol{f} dV - \int_{\mathrm{K}} \phi_i^K s \, dV \right] = 0$$

Apply divergence theorem to flux integral

$$\int_{\mathrm{K}} \phi_i^K \partial_t u \, dV + \oint_{\partial \mathrm{K}} \phi_i^K \hat{\boldsymbol{n}} \cdot \boldsymbol{f} \, dS - \int_{\mathrm{K}} \boldsymbol{f} \cdot \nabla \phi_i^K dV - \int_{\mathrm{K}} \phi_i^K s \, dV = 0$$

- **Consistency**: Fluxes must be single valued on interfaces between elements.
  - **Numerical Flux**: Solution to Riemann problem to generate consistent flux on interfaces.