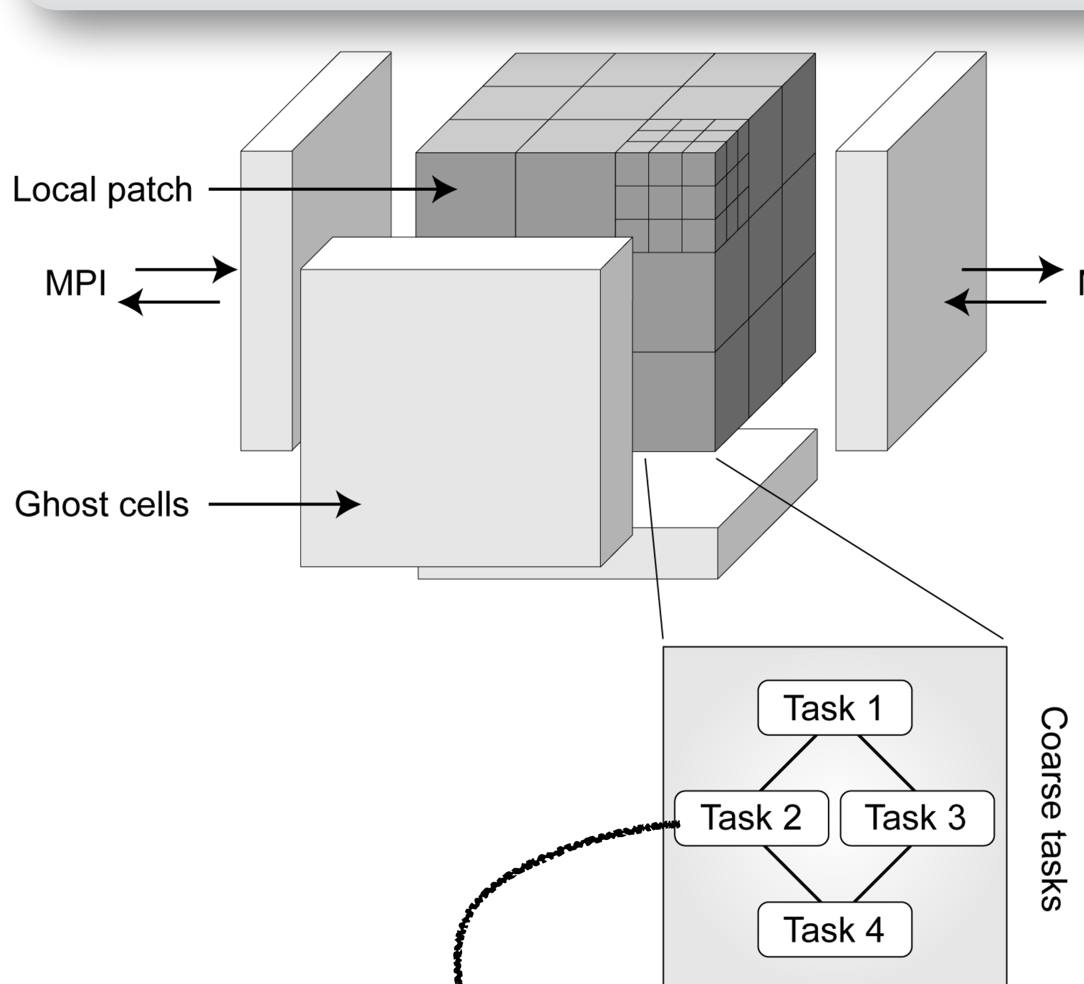
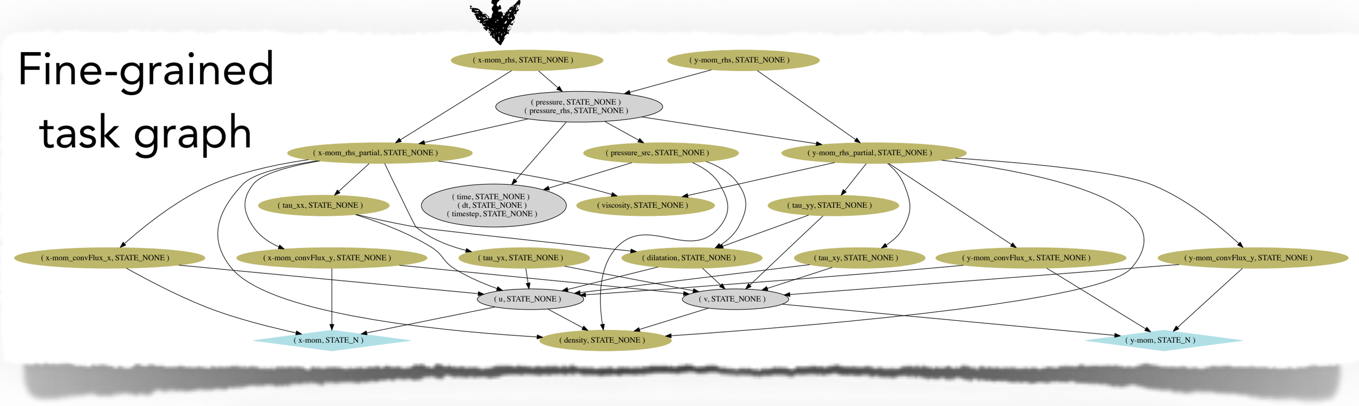
Flexible, Efficient Abstractions for High Performance Computation on Current and Emerging Architectures

James C. Sutherland, Matthew Might, Christopher Earl, Tony Saad, Abhishek Bagusetty

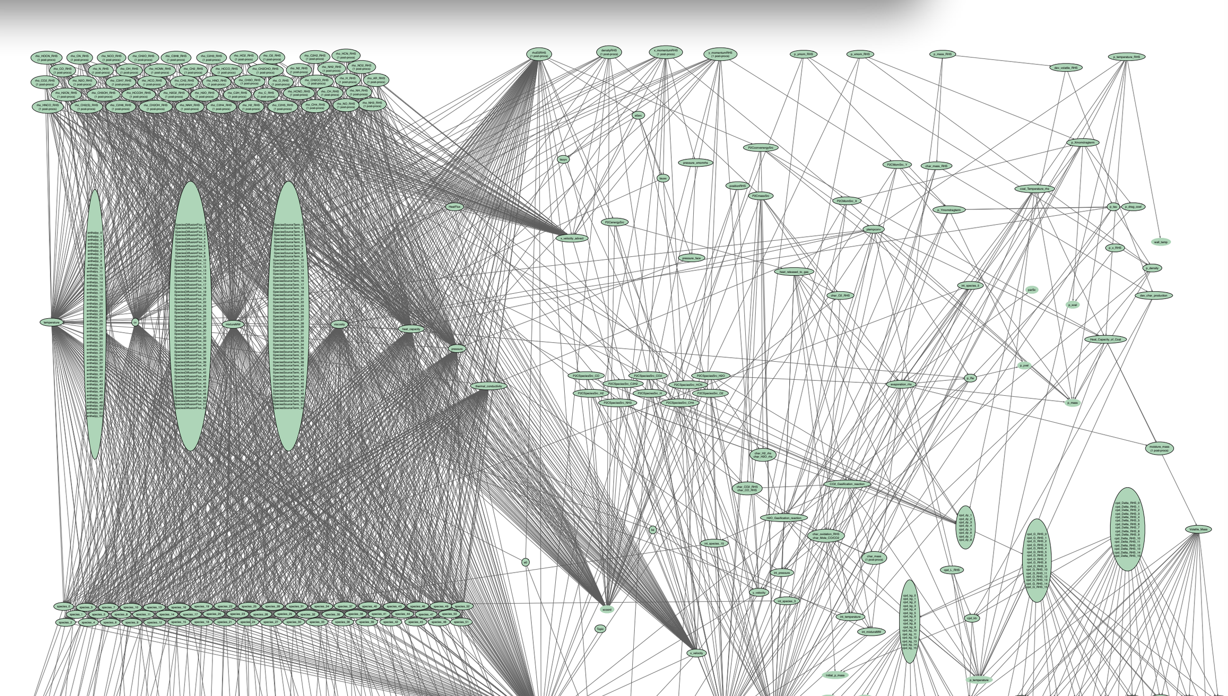
Flexibility via DAG representation of problem

- **Tame complexity** arising from multi physics software design: multiplicity of models with different nonlinear couplings, etc.
- Expose & exploit **hierarchical parallelism** (both data and task parallelism).
- **Overlap** communication & computation.
- **Automate** memory management, data movement and task scheduling.





- Automatically generate dependencies.
- Deduce algorithm from dependencies.
- Use task-parallelism from DAG/task graph.
- Use data-parallelism within each node.
- Automate memory management (host-device transfers, etc.).



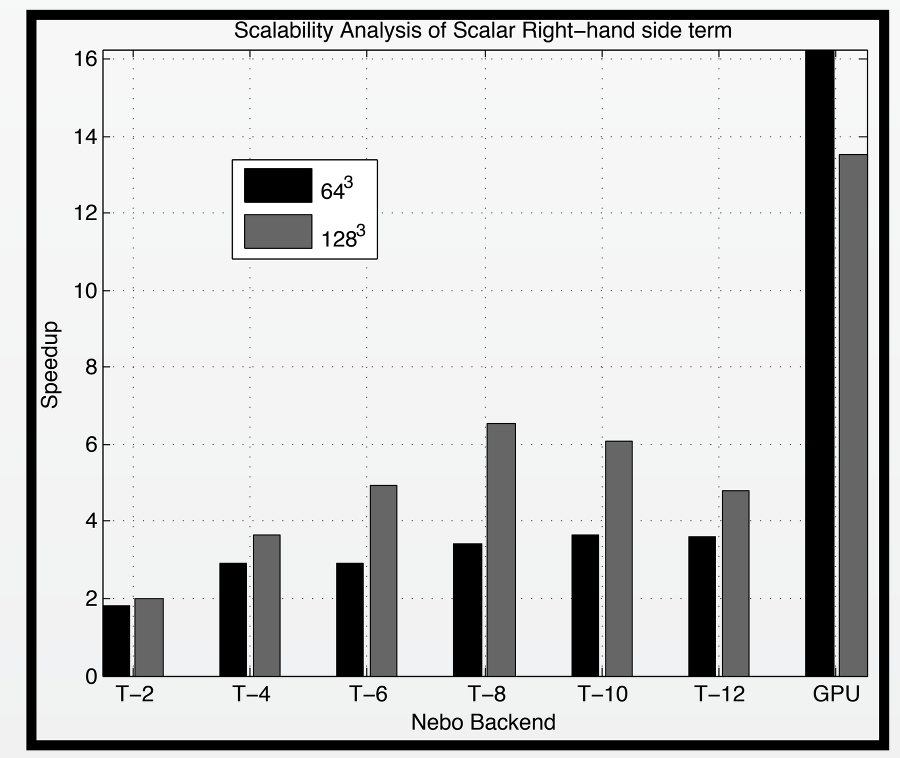
Example DAG for a coal combustion problem.

Example: Scalar PDE right-hand-side evaluation

$$\frac{\partial T}{\partial t} = \nabla \cdot (\lambda \nabla T)$$

```
rhs <<= divX( interpX(lambda) * gradX(temp) )
      + divY( interpY(lambda) * gradY(temp) )
      + divZ( interpZ(lambda) * gradZ(temp) );
```

- Code compiles down to a single loop / GPU kernel.
- Code deploys on single- & multi-core as well as GPU.
- Improvements to code are immediately felt through the whole code base.



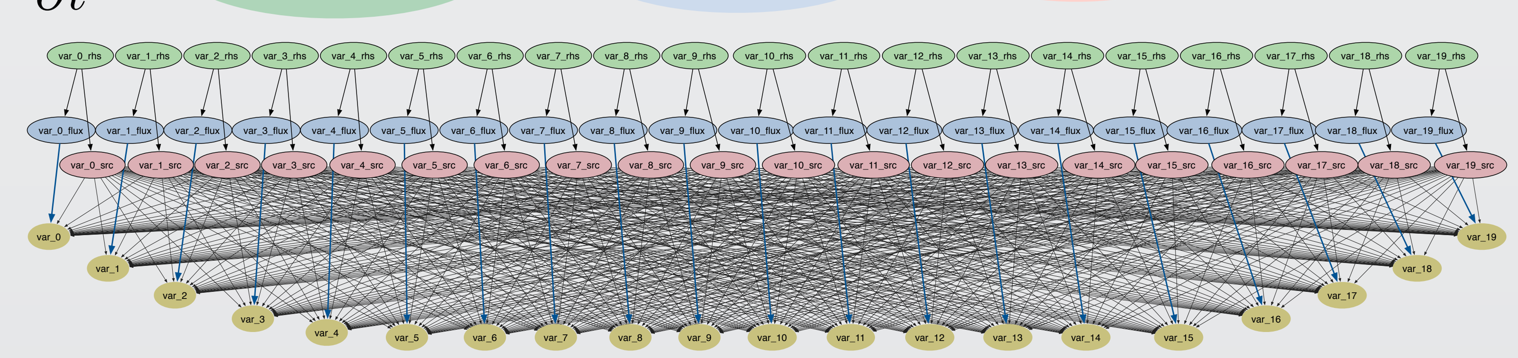
Example: vectorized conditionals

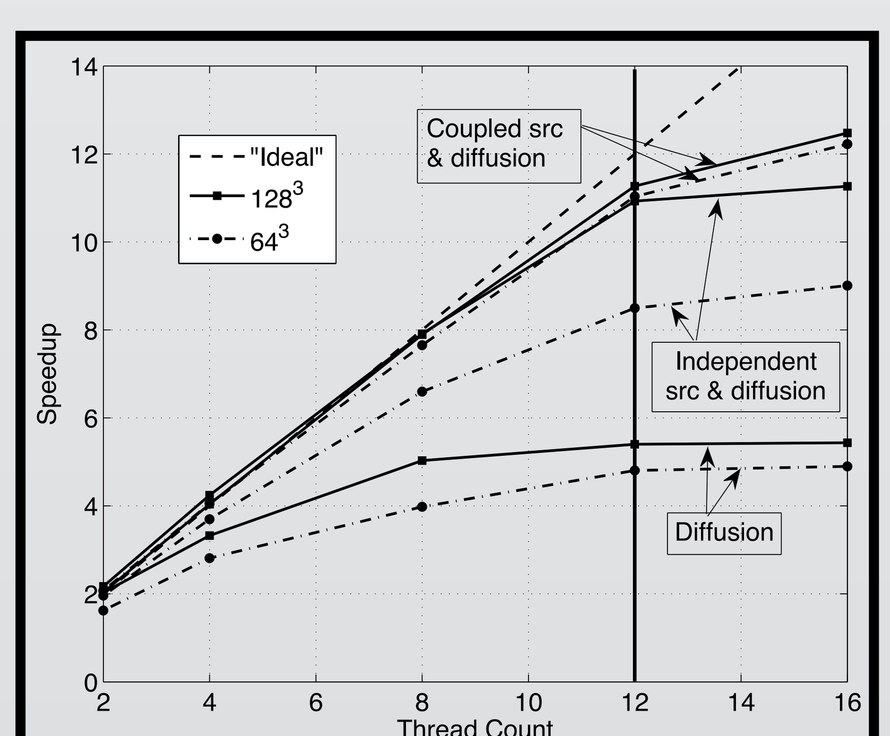
$$d = \begin{cases} s_1 & u > 0 \\ s_2 & u < 0 \\ \frac{s_1+s_2}{2} & \text{otherwise} \end{cases}$$

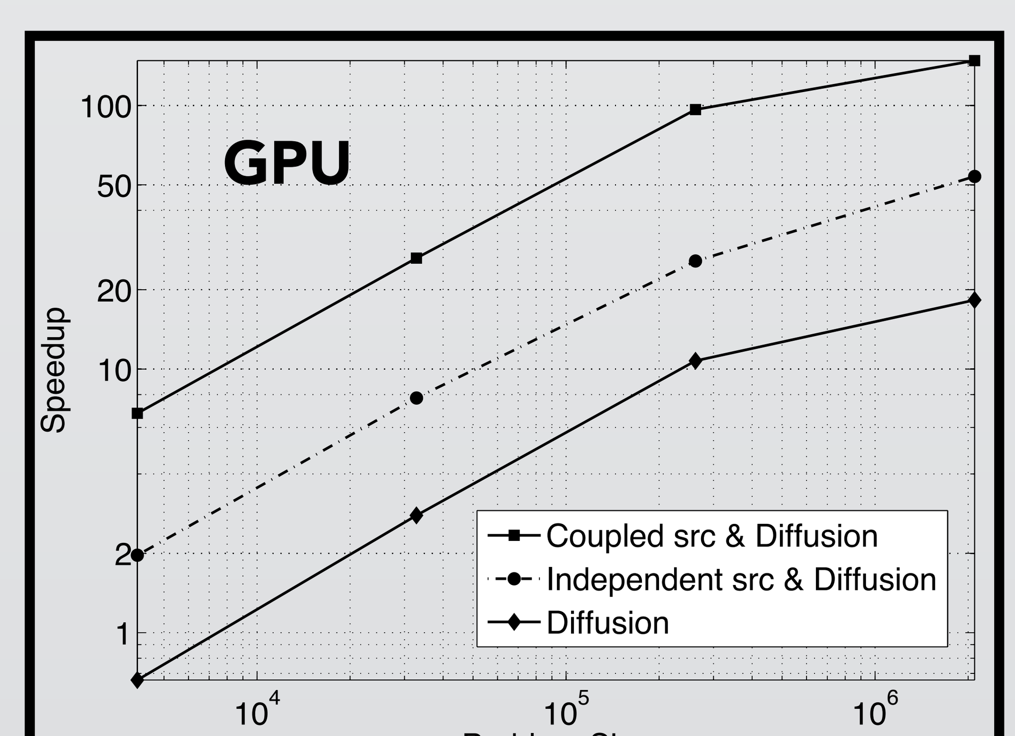
```
d <<= cond( aVel > 0.0, minusField )
          ( aVel < 0.0, plusField  )
          ( 0.5 * (minusField + plusField) );
```

Example: system of diffusion/reaction PDEs

$$\frac{\partial \phi_i}{\partial t} = -\nabla \cdot \mathbf{J}_i + s_i \qquad \mathbf{J}_i = -\Gamma \nabla \phi_i \qquad s_i = f(\phi_j)$$

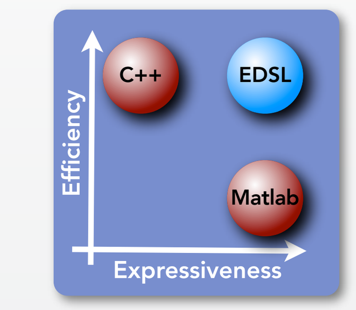


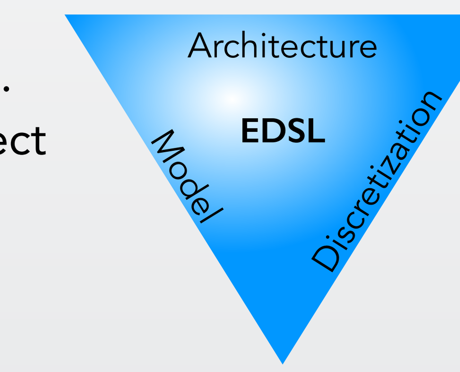




Domain Specific Language Goals:

- **Expressive**: Express intent, not implementation.
- **High-performance**: Match or exceed hand-tuned code.
- **Portable**: Migrate to multicore, GPU by auto-generating optimized "back-end" code.
- **Adoptable**: Maintain compatibility & interoperability by embedding in C++.
- **Error-checked**: Write robust and correct code with strong typing and type inference.

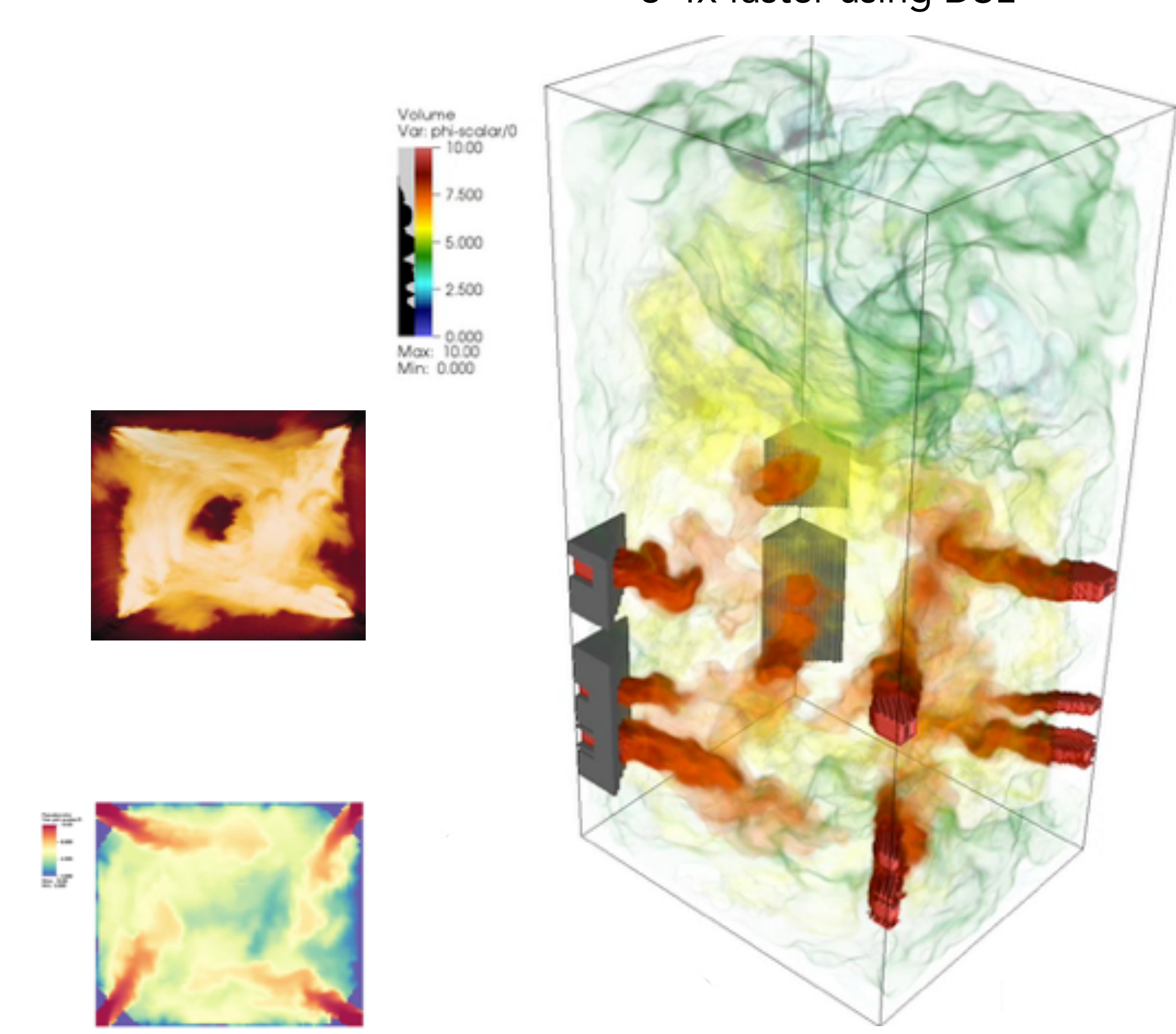





```
// field type inference:
typedef FaceTypes<FieldT>::XFace  XFluxT;
typedef FaceTypes<FieldT>::YFace  YFluxT;
typedef FaceTypes<FieldT>::ZFace  ZFluxT;

// operator type inference:
typedef OpTypes<FieldT>::DivX  DivX;
typedef OpTypes<FieldT>::DivY  DivY;
typedef OpTypes<FieldT>::DivZ  DivZ;
```

3-4x faster using DSL



US DOE award DE-NA0000740
NSF PetaApps award 0904631