**Introduction**

Heterogeneity is becoming a fact of life in HPC, largely driven by demands for increased parallelism and power efficiency over what traditional CPUs can provide. However, extracting the full performance of heterogeneous systems is non-trivial and requires architecture expertise. Retrofitting existing codes for heterogeneity is tedious and error-prone, architecture experts are in short supply, and accelerators are moving targets. Therefore, a single API for transparently executing optimized code on accelerators with minimal intervention is needed for scientific productivity.

**Design**

'Make-your-own' library from modular building blocks
- **User Apps**
- **Backends Layer**
  - CUDA Backend
  - OpenCL Backend
- **APIs Layer**
  - C API
  - Timer Plugin
  - MPI Plugin
  - Fortran 2003 API
  - Future Plugins
- **Layering**
  - D WITH_CUDA
  - D WITH_TIMERS
  - D WITH_MPI

**Accelerated Backends**

The heavy-lifters of the library, selected at runtime by a 'mode' environment variable from those included at compile-time. Include implementations of all C API-supported kernels for a single accelerator model.

**Plugins**

*Automatic Profiling*

Runtime control over backends, plugins, and accelerator device selection

*Performance*

All tests performed end-to-end on a single system containing 2x Intel Xeon E5504 Quad-core CPUs, 2012 FIFA and a Tesla 2050 GPU.

**Related Efforts**

_Solver Frameworks_

**OpenFOAM** [1]

- Pros: Support for useful pre- and post-processing (mesh generation and visualization); many solvers for many domains
- Cons: No internal accelerator support; framework-centric development; cumbersome API and 'case' construction

**PARALUTION** [2]

- Pros: Many matrix storage formats; many solvers; many preconditioners; support for OpenMP, CUDA, and OpenCL on CPUs/GPUs and MIC, plugins for Fortran and OpenFOAM
- Cons: Framework-centric development; interop, with existing code base; no MPI support (yet); asynchronous operations only on CUDA; lack of non-destructive copy to/from Arrays

_Solver Libraries_

**MAGMA** [3]

- Pros: Full BLAS and LAPACK support for CUDA, OpenCL, and MIC; support for several factorizations and eigenvalue problems; smart scheduling of hybrid CPU/GPU algorithms with QUARK directed acyclic graph scheduler; multi-GPU methods
- Cons: CUDA, OpenCL, and MIC variants are separate implementations; no internal MPI support; MKL/AOM dependency poorly documented and cumbersome

**Trilinos** [4]

- Pros: Massive set of capability areas beyond linear algebra, solvers, and meshes; built-in distributed memory support; some preliminary CUDA/MIC work (e.g. Kokkos, Thalos, Tetra packages)
- Cons: Redundancies of capability between packages; breadth of packages difficult to navigate for newcomers

**Future Work**

Continue expanding the API's provided set of kernels and backends with other primitive operations underlying fluid simulations, i.e. Krylov solvers, stencil computations, and various preconditioners

Generalize operations to work on non-3D data, and add primitives for computations on unstructured grids

Generate a third automatically runtime-scheduled backend to transparently execute code across entire node, a la CoreTSAR [7].