SAND2017-11645PE

ECP ExaWind Experience in Transitioning Nalu to MPI+X

Stefan P. Domino
Computational Thermal and Fluid Mechanics Department
Sandia National Laboratories, Albuquerque, NM

University of Utah PSAAP-2 TST Meeting
Salt Lake City, Utah
November 1st and 2nd, 2017

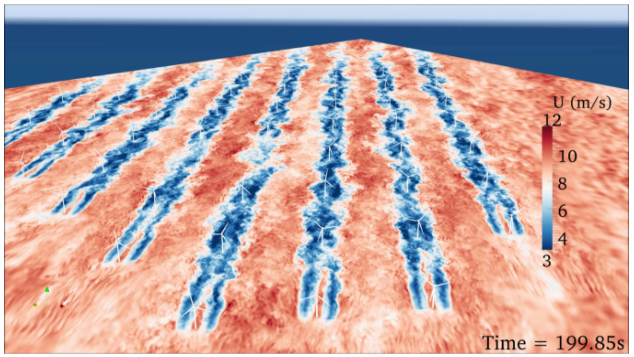
SAND-TBD

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-685 NA0003525. This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nations exascale computing imperative.

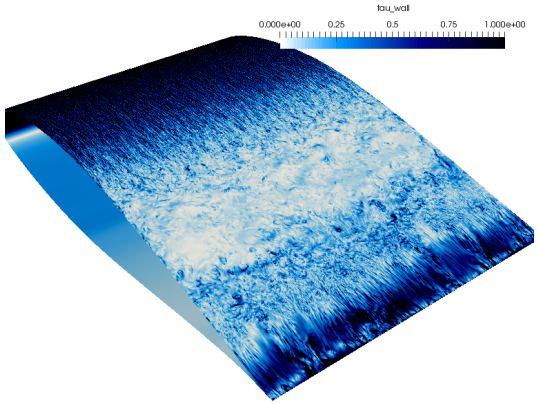
Sandia National Laboratories

Value of Proposition: Deployment of HPC towards HF Wind Plant Modeling

Current Wind Plant Flow Physical Models + **Scalable flow solver technology** = ***Paradigm shift* in wind plant design & operation**



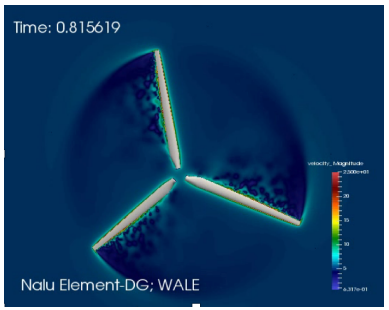
Wind plant wake simulation using NREL SOWFA tool (current state of the art)
Millions of elements, low-order



Near-blade simulation using SNL Nalu code on Trinity
Billions of elements, P=1, 2, …

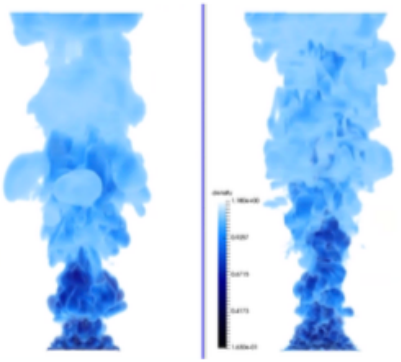
Enabling SNL technology

- **Wind plant simulation that *simultaneously resolves wakes and near-blade flow*.**
- **Allows the engineer to fully characterize linkages between *turbine design, site characteristics,* and *plant performance*.**

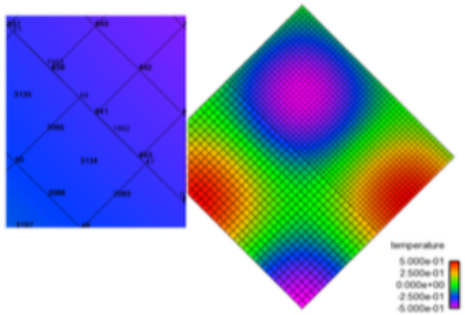


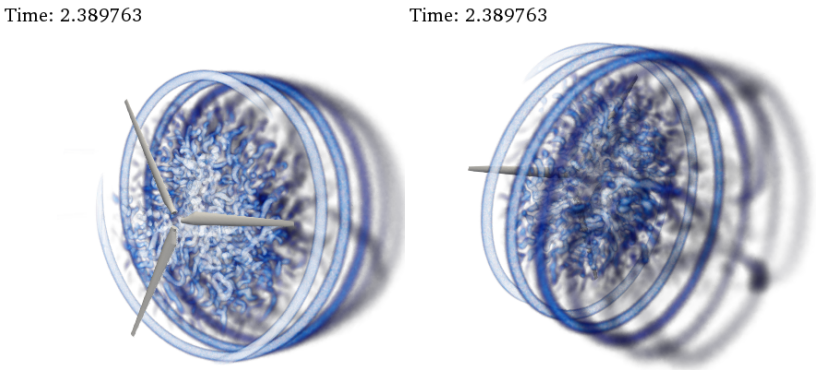
ExaWind Sample of Research Topics

- Low-/higher-order tradespace for LES
- Sliding mesh and/or overset
- Advanced stabilization techniques for nonlinear PDEs
- Increased solver performance at scale; **O(100)** billion elements
- Matrix storage reduction techniques for higher-order (static condensation)
- AMG coarsening strategies
- In situ matrix modification
- Efficient parallel searches
- Kokkos integration
- NGP focused for Exascale on open-source



Low/high-order (with NC interfaces)





Multi-lab

Diverse Core Competency Team

NREL
NATIONAL RENEWABLE ENERGY LABORATORY

- Project lead
- Flow Physics and FSI modeling
- Application performance/scaling

Sandia National Laboratories

- Unstructured Algorithms and software design
- Software infrastructure/NGP performance
- Physics model/discretization interface

OAK RIDGE
National Laboratory

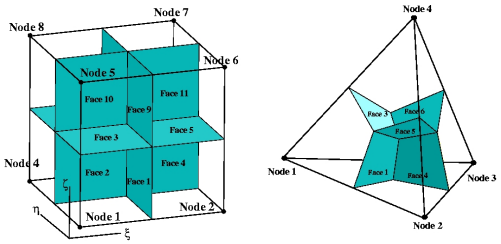
- Software integration
- NGP scaling and performance

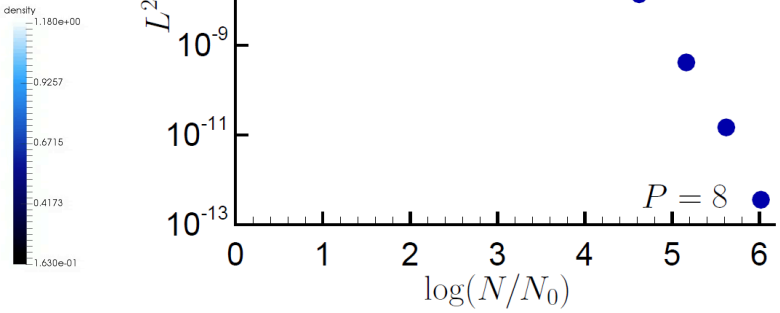
THE UNIVERSITY OF TEXAS AT AUSTIN

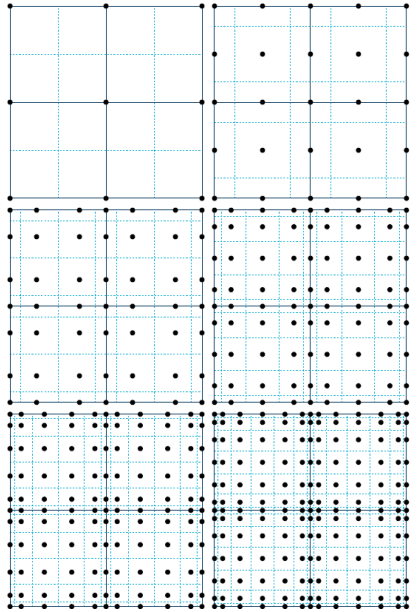
- Turbulence modeling
- Uncertainty quantification

Funding streams both from: Office of Science ECP and EERE A2e

Extension of CVFEM to Higher-order





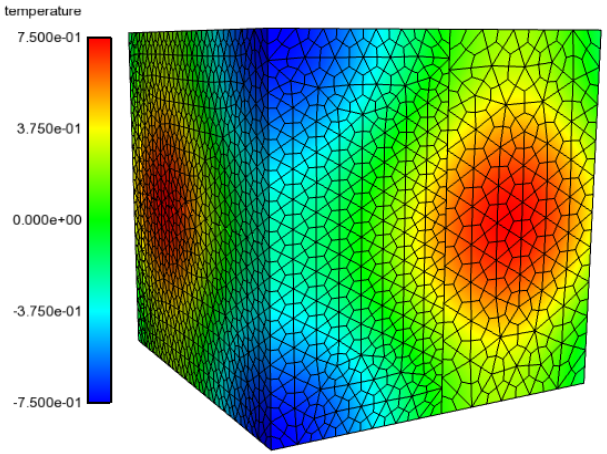


1-meter helium plume simulation comparing P=1 (left) and P=4 (right) on the same number of node mesh (shown: volume-rendered density)

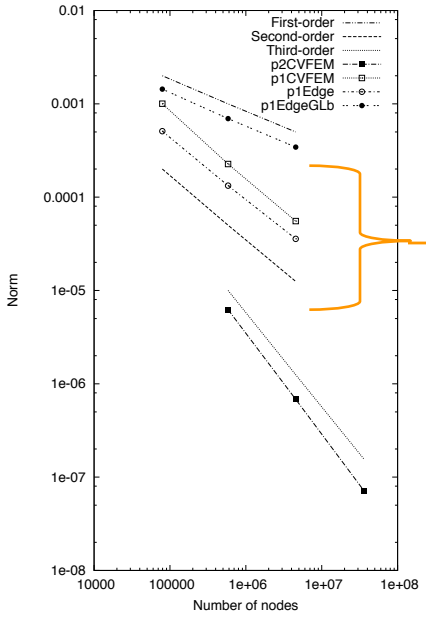
Spectral CVFEM convergence (left) and polynomial promotion (right) from P=1 to P=6 outlining the dual mesh configurations (four-element patch)

Why Higher-order for low-Mach Unstructured LES?

- **Experience shows >> one order of magnitude error reduction at the coarsest mesh for P=1 and P=2**
- **LES resolved advection can improve**



Non-conformal DG/CVFEM Laplace MMS



- P=2 Thex27 simulations (at the same node-count) results in > 1 order of magnitude error reduction compared to the P=1 Thex8 Hex8

>10x norm reduction for Hex27 vs Hex8 on the same nodes

- Much more local work to possibly exploit for higher-order algorithms
- Challenges:
  - memory footprint, solvers, and preconditioners

Towards Performance and Portability

- **Although Nalu is a relatively new code base (by way of Tri-lab code lifecycles), the V1.1 code base, i.e., prior to ExaWind is:**
  - non-threaded
  - non-SIMD
  - non-GPU (FY20 plan)
  - heavy on std::vector usages for data gather (with re-size)
  - Computational kernels include a low-order edge- and element-based scheme in addition to a P>1 hex element
- **How does an application code "protect" itself from future changes?**
- **The SNL ASC/ATDM efforts are heavily leaveraged on Kokkos for the performance/portability path forward**

A FY17/Q1 View of Nalu

- **MPI-based, lots of resize, physics within specialized heterogeneous algorithms, e.g., AssembleMomentumElemSolverAlg.C**
- **Very much , a non-DSL design…**
  - The developer manages rhs(), lhs(), with rho()*u_j()*n_j()*dS

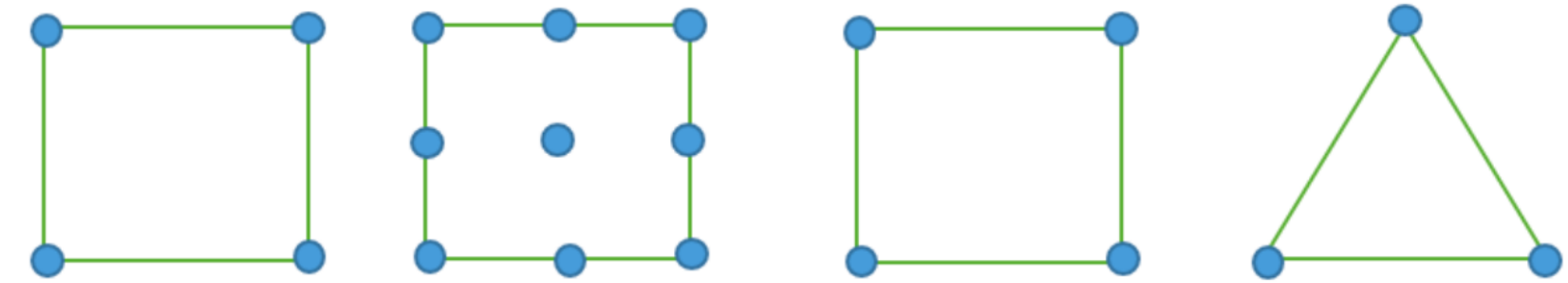


Figure 4: Heterogeneous topologies example.

Consider a heterogeneous mesh use case

Sierra Toolkit (STK)/Nalu Interface: stk::selector and stk::bucket

```
// define the selector; locally owned, the parts I have served up and active
stk::mesh::Selector s_locally_owned_union = metaData_.locally_owned_part()
  & stk::mesh::selectUnion(partVec_)
  & !(realm_.get_inactive_selector());
```

Listing 1: Basic selector usage in STK.

```
// given the defined selector, extract the buckets of type ''element''
stk::mesh::BucketVector const& elem_buckets
  = bulkData_.get_buckets( stk::topology::ELEMENT_RANK, s_locally_owned_union );

// loop over the vector of buckets
for ( const stk::mesh::Bucket* bptr : elem_buckets) {
  const stk::mesh::Bucket & bucket = bptr;

  // extract master element (homogeneous over buckets)
  MasterElement *meSCS = realm_.get_surface_master_element(bucket.topology());

  for ( stk::mesh::Entity elem : bucket ) {
    // operate on this elem
    // etc...
  }
}
```

Common code for all specialized algorithms

Listing 2: Basic bucket looping STK.

Nalu Algorithm Abstraction: nalu::Kernel

- nalu::Kernel is a part of a PDE that is activated for a given simulation, e.g., advection_diffusion, time_derivative, buoyancy, etc.
- The superset of all fields, master element calls, e.g., grad_op(), area_vec(), required for all kernels are provided in prereq_data

```
Algorithm::execute()
{
  for(elem : elements) {
    //gather coords, compute gradients, etc
    for(kernel : computeKernels) {
      kernel->execute(elem, prereq_data, lhs, rhs);
    }
    apply_coeff(lhs, rhs, ...);
  }
}
```

Polymorphic challenge on GPU addressed by Aria FY17/Q4 L2

Listing 3: Basic Algorithm/Kernel loop structure.

Nalu Algorithm Abstraction: nalu::Kernel::Kernel()

- nalu::Kernel is templated on AlgTraits, e.g., integration rule such as nodesPerElement_, numIntgPoints_, etc.

```
template<typename AlgTraits>
MomentumNSOElemKernel<AlgTraits>::MomentumNSOElemKernel(
  ElemDataRequests& dataPreReqs)
{
  // define master element rule for this kernel
  MasterElement *meSCS
    = sierra::nalu::MasterElementRepo::get_surface_master_element(AlgTraits::topo_);

  // add ME rule
  dataPreReqs.add_cvfem_surface_me(meSCS);

  // add fields to gather
  dataPreReqs.add_coordinates_field(*coordinates_, AlgTraits::nDim_, CURRENT_COORDINATES);
  dataPreReqs.add_gathered_nodal_field(*velocityNp1_, AlgTraits::nDim_);

  // add ME calls
  dataPreReqs.add_master_element_call(SCS_GIJ, CURRENT_COORDINATES);
}
```

Listing 4: Attributes of a kernel; part A the constructor.

Nalu Algorithm Abstraction: nalu::Kernel::execute()

```
template<typename AlgTraits>
void
MomentumNSOElemKernel<AlgTraits>::execute(
  SharedMemView<DoubleType**>& lhs,
  SharedMemView<DoubleType *>& rhs,
  ScratchViews<DoubleType>& scratchViews)
{
  SharedMemView<DoubleType**>& v_uNp1
    = scratchViews.get_scratch_view_2D(*velocityNp1_);
  SharedMemView<DoubleType***>& v_gijUpper
    = scratchViews.get_me_views(CURRENT_COORDINATES).gijUpper;

  for ( int ip = 0; ip < AlgTraits::numScsIp_; ++ip ) {

    // determine scs values of interest
    for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {

    // assemble each component
    for ( int k = 0; k < AlgTraits::nDim_; ++k ) {

      // determine scs values of interest
      for ( int ic = 0; ic < AlgTraits::nodesPerElement_; ++ic ) {

        // save off velocityUnp1 for component k
        const DoubleType& ukNp1 = v_uNp1(ic,k);

        // denominator for nu as well as terms for "upwind" nu
        for ( int i = 0; i < AlgTraits::nDim_; ++i ) {
          for ( int j = 0; j < AlgTraits::nDim_; ++j ) {
            gUpperMagGradQ += constant*v_gijUpper(ip,i,j);
          }
        }
      }
    }
  }
}
```

Thread-local scratch arrays using A Kokkos SharedMemView

Templated

MD-array rather than error-prone pointer arithmatic

Listing 5: Attributes of a kernel; part B the body.

Nalu Algorithm Abstraction: Kokkos integration, parallel_for()

- **Team-based nested Kokkos::parallel_for() model**

```
Algorithm::execute()
{
  int bytes_per_thread = //scratch bytes per element
  auto team_exec = get_team_policy(... bytes_per_thread, ...);
  Kokkos::parallel_for(team_exec, buckets, [&](team)
  {
    ScratchViews scratchViews(topo, meSCS, dataNeeded ...);
    Kokkos::parallel_for(team, bucket.size()) {
      fill_pre_req_data(dataNeeded, elem, ..., scratchViews);
      for(kernel : computeKernels) {
         kernel->execute(elem, lhs, rhs, scratchViews);
      }
    }
    apply_coeff(lhs, rhs, ...);
  }
}
```

Listing 7: Thread-parallel Algorithm structure.

Nested parallel_for() thread-team parallelism.

Each bucket is processed by a team, inner loop over elements is split among threads in a team.

scratchViews is created in outer loop, contains views into scratch-memory allocation (no heap-alloc is done here).

fill_pre-req_data() fills thread- local storage for a given element (shared over all kernels)

Nalu Algorithm Abstraction: STK::SIMD

- **Explicit SIMD instructions allow Nalu to take advantage of vector processing units even when auto-vectorization (compiler-based) fails**

```
stk::simd::Double x = 1.0;
stk::simd::Double y = 2.0;
stk::simd::Double z = stk::math::sqrt(x*y);
```

Listing 8: Basic SIMD Usage.

- **STK::simd::Double is an array of doubles at the enabled vector instruction set**

```
void execute(const DoubleType* densityAtElemNodes, DoubleType* rhs)
{
    for(i : nodesPerElement) {
        rhs[i] = f(densityAtElemNodes[i]);
    }
}
```

Listing 9: Function using DoubleType.

Nalu Algorithm Abstraction: STK::SIMD

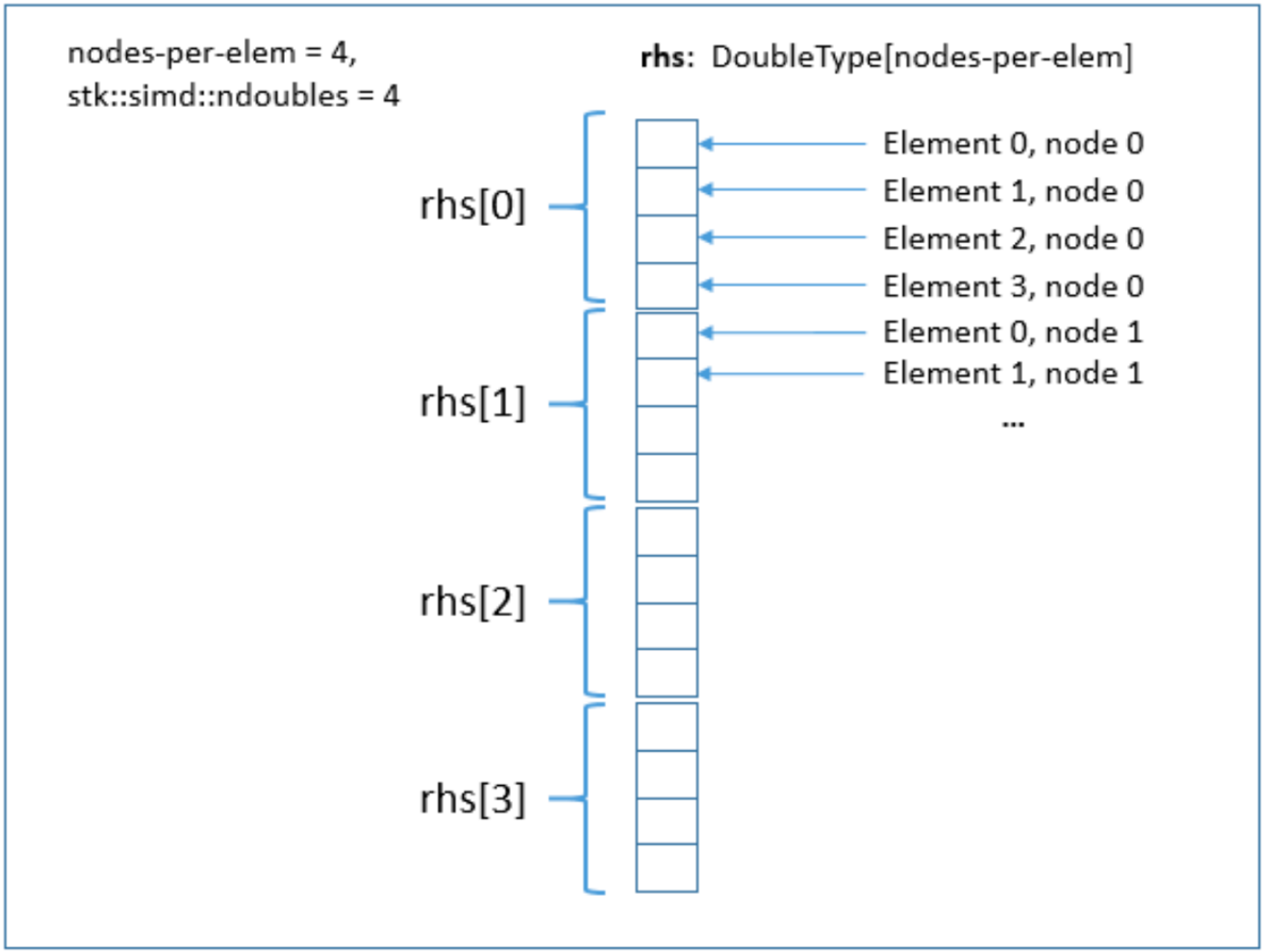


Figure 5: Interleaved array illustration.

When DoubleType is stk::simd::double and stk::simd::ndoubles is 4

The previous function computes 4 element rhs contributions from the 4 elements-worth of nodal density

Application SIMD code must, therefore, gather, copy and interleave data manually

A note on Performance

- Data from FY17/Q4 ExaWind study, “Deploy Nalu/Kokkos algorithmic infrastructure with performance benchmarking”
- Project is focused on KNL
- Unit-test and code application-driven (solver performance performed (GMRES/SGS) however, will not be reported here (learning KNL/threading expertise on Cori, Trinity, etc)
- Interior master element methods transitioned from F77-based to Kokkos/SIMD for increased performance

Unit Test Findings; SIMD

- **Unit test was promising: >5x improvement KNL**

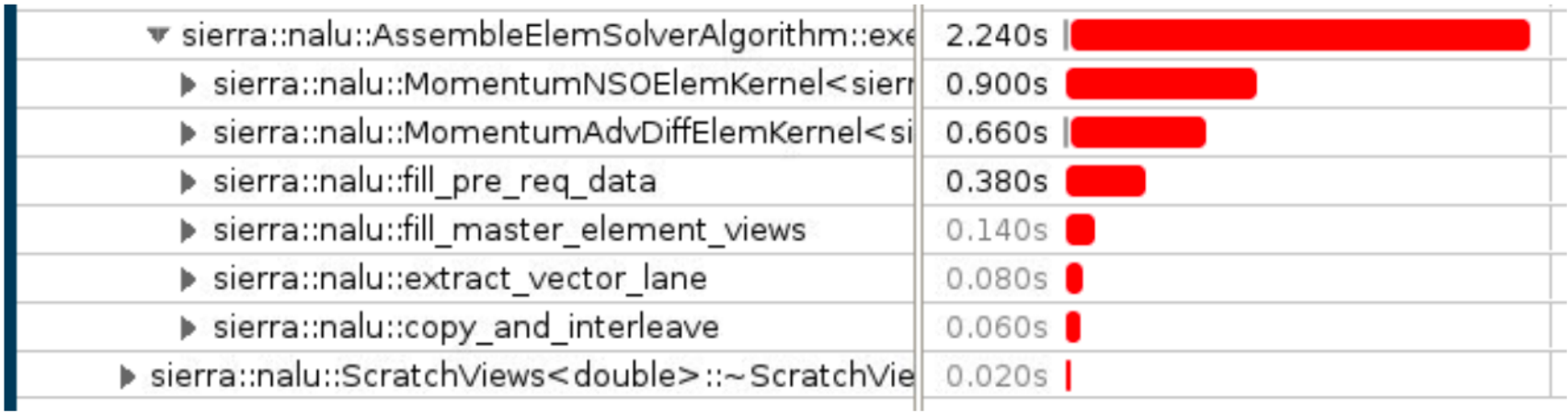


Figure 7: Two Momentum Kernels unit-test, 125000 Hex-8 elements, KNL architecture.

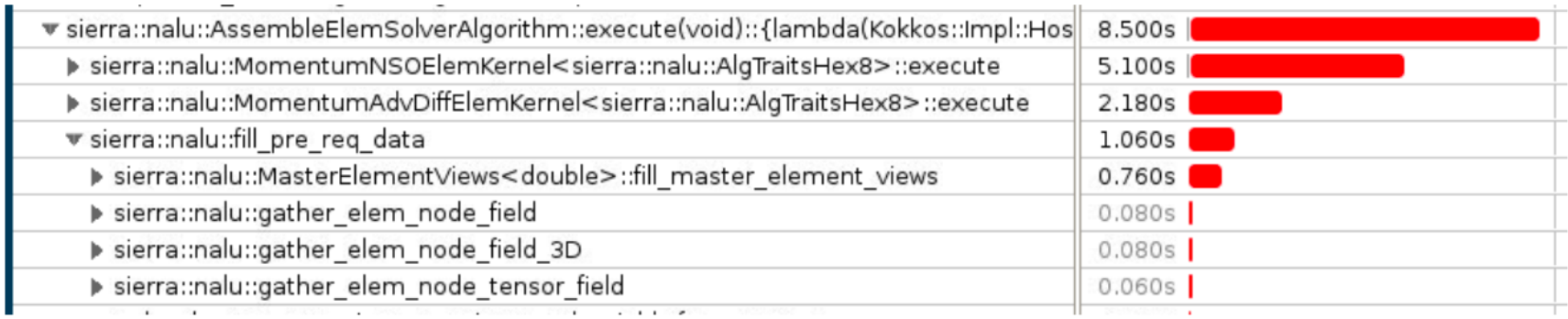


Figure 6: Two Momentum Kernels unit-test, no SIMD, 125000 Hex-8 elements, KNL architecture.

Haswell/KNL Application Code Matrix Assembly Performance

- Test case: Re 50k turbulent open-jet (LES)
- Modest sized problems, O(150)M; RHS and LHS timings captured

Table 1: Table showing the details of parallel execution (MPI ranks and OpenMP threads) for the various code configuration results shown in this section.

	Haswell				KNL			
Code Configuration	Num. Nodes	MPI ranks	Ranks per node	OMP Threads	Nodes Nodes	MPI ranks	Ranks per node	OMP Threads
Coarse $P=1$ mesh (17.5M HEX-8 elements)								
Hybrid	5	160	32	1	5	320	64	1
Baseline	5	160	32	1	5	320	64	1
C1	5	160	32	1	5	320	64	1
C2	5	160	32	1	5	320	64	1
C3	10	160	16	2	5	320	64	2
C4	10	160	16	2	5	320	64	2
Fine $P=1$ mesh (140M HEX-8 elements)								
Hybrid	40	1280	32	1	40	2560	64	1
Baseline	40	1280	32	1	40	2560	64	1
C1	40	1280	32	1	40	2560	64	1
C2	40	1280	32	1	40	2560	64	1
C3	80	1280	16	2	40	2560	64	2
C4	80	1280	16	2	40	2560	64	2
Coarse $P=2$ mesh (17.5M HEX-27 elements)								
Baseline	40	1280	32	1	40	2560	64	1
C1	40	1280	32	1	40	2560	64	1
C2	40	1280	32	1	40	2560	64	1
C3	80	1280	16	2	80	2560	32	2
C4	80	1280	16	2	80	2560	32	2

Haswell General Application Findings

With and without SIMD

Table 2: Matrix assembly timing comparisons on Haswell partition of NERSC Cori system. Code configurations: H – Hybrid element/node algorithm; B – Baseline non-consolidated algorithm; C1 – consolidated kernel algorithms; C2 – C1 with SIMD datatypes; C3 – C2 with OpenMP threading; C4 – C3 with `sumInto(...)` and SIMD interleave optimizations.

	Matrix assembly time (s)						Speedup ratio						
Equation	Hybrid	Baseline	Conf. C1	Conf. C2	Conf. C3	Conf. C4	B:C1	C1:C2	B:C2	B:C3	B:C4	H:B	H:C4
Coarse $P=1$ mesh (17.5M HEX-8 elements)													
Momentum	233.9	264.5	278.0	219.2	154.7	115.9	0.95	1.27	1.21	1.71	2.28	0.88	2.02
Continuity	60.9	60.9	61.2	50.8	36.8	27.2	0.99	1.21	1.20	1.66	2.24	1.00	2.24
TKE	70.1	93.2	94.3	70.1	52.2	41.9	0.99	1.34	1.33	1.78	2.23	0.75	1.67
Mix. Frac.	68.8	91.7	86.6	62.5	45.9	36.1	1.06	1.38	1.47	2.00	2.54	0.75	1.91
Fine $P=1$ mesh (140M HEX-8 elements)													
Momentum	246.9	277.8	290.1	228.8	161.1	118.2	0.96	1.27	1.21	1.72	2.35	0.89	2.09
Continuity	64.4	64.4	65.1	54.4	38.5	28.9	0.99	1.20	1.18	1.67	2.23	1.00	2.23
TKE	73.9	97.7	99.2	73.9	55.6	44.9	0.98	1.34	1.32	1.76	2.17	0.76	1.64
Mix. Frac.	72.5	96.1	90.9	66.1	48.2	38.3	1.06	1.38	1.45	1.99	2.51	0.76	1.89
Coarse $P=2$ mesh (17.5M HEX-27 elements)													
Momentum	–	967.3	937.6	697.7	470.3	392.2	1.03	1.34	1.39	2.06	2.47	–	–
Continuity	–	249.4	201.1	172.0	118.9	95.5	1.24	1.17	1.45	2.10	2.61	–	–
TKE	–	254.5	277.1	257.9	157.6	132.7	0.92	1.07	0.99	1.61	1.92	–	–
Mix. Frac.	–	251.7	261.4	243.0	147.4	123.6	0.96	1.08	1.04	1.71	2.04	–	–

With and without threading

KNL General Application Findings

With and without SIMD

Table 3: Matrix assembly timing comparisons on KNL partition of NERSC Cori system. Code configurations: H – Hybrid element/node algorithm; B – Baseline non-consolidated algorithm; C1 – consolidated kernel algorithms; C2 – C1 with SIMD datatypes; C3 – C2 with OpenMP threading; C4 – C3 with `sumInto(...)` and SIMD interleave optimizations.

	Matrix assembly time (s)						Speedup ratio						
Equation	Hybrid	Baseline	Conf. C1	Conf. C2	Conf. C3	Conf. C4	B:C1	C1:C2	B:C2	B:C3	B:C4	H:B	H:C4
Coarse $P=1$ mesh (17.5M HEX-8 elements)													
Momentum	329.6	398.0	473.0	352.0	250.7	203.1	0.84	1.34	1.13	1.59	1.96	0.83	1.62
Continuity	91.7	91.7	98.1	73.3	54.2	41.1	0.94	1.34	1.25	1.69	2.23	1.00	2.23
TKE	117.6	156.5	170.7	101.5	81.4	64.6	0.92	1.68	1.54	1.92	2.42	0.75	1.82
Mix. Frac.	115.2	154.0	152.2	87.3	67.0	53.0	1.01	1.74	1.76	2.30	2.90	0.75	2.17
Fine $P=1$ mesh (140M HEX-8 elements)													
Momentum	343.0	419.4	493.0	374.9	266.2	220.9	0.85	1.32	1.12	1.58	1.90	0.82	1.55
Continuity	97.1	97.1	105.4	80.2	61.0	46.9	0.92	1.31	1.21	1.59	2.07	1.00	2.07
TKE	123.8	165.7	184.6	114.9	94.3	76.8	0.90	1.61	1.44	1.76	2.16	0.75	1.61
Mix. Frac.	121.6	163.3	162.6	96.7	76.4	61.5	1.00	1.68	1.69	2.14	2.66	0.74	1.98
Coarse $P=2$ mesh (17.5M HEX-27 elements)													
Momentum	–	1746.8	1642.3	894.0	456.2	379.4	1.06	1.84	1.95	3.83	4.60	–	–
Continuity	–	370.8	308.5	228.9	114.6	94.8	1.20	1.35	1.62	3.23	3.91	–	–
TKE	–	449.6	496.8	290.1	146.9	126.1	0.90	1.71	1.55	3.06	3.57	–	–
Mix. Frac.	–	446.0	461.0	276.9	140.2	119.7	0.97	1.67	1.61	3.18	3.73	–	–

With and without threading

Conclusions

- **The ECP-funded project is focused on low-Mach unstructured turbulent flow using implicit matrix solves**
- **The Nalu open-source code base is the base code from which NGP improvements are being made**
- **Kokkos has been used for the performance/portability design**
- **stk::SIMD is used for vectorization when and if the compiler can not perform this task**
- **KNL and Haswell performance shows modest increase in SIMD**
- **Higher-order kernels are showing larger speed-ups**
- **P=2 kernel now < 2x that of the same P=1 kernel**
- **Path forward:**
  - Convert all remaining master elements for hybrid meshes
  - Continue testing and improving of Kernels/solvers

Thanks to the Nalu NGP Applications team

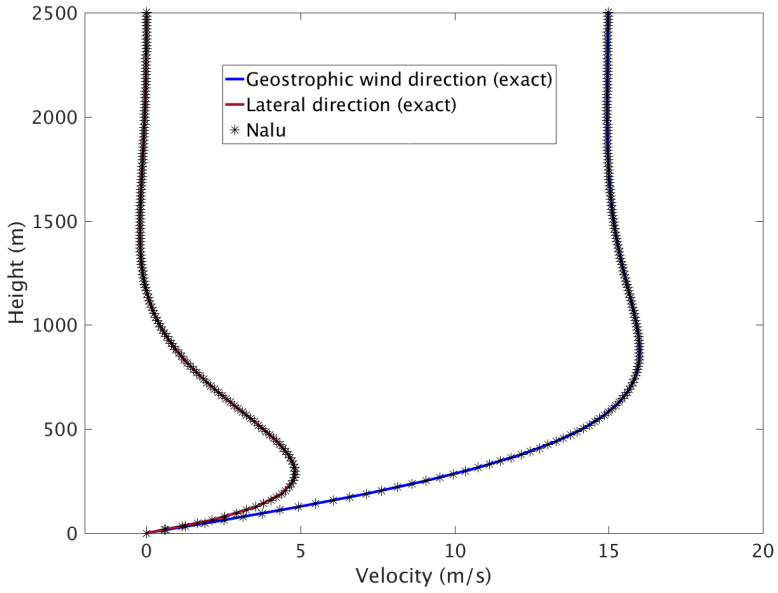
- **Robert Knauss and Alan Williams (SNL ExaWind)**
- **Shreyas Ananthan (NREL ExaWind)**

- **Thanks also to synergistic work conducted at SNL:**
  - Cristian Trott, FY16 Nalu/STK NGP prototyping
  - The Sierra Thermal/Fluids Aria team:
    - Victor Brunini
    - Jonathan Clousen
  - STK::SIMD:
    - Mike Tupek

Performance Benchmarking of Atmospheric Boundary Layer LES

- **Physics models required for wind-plant scale simulation of the atmospheric boundary layer (ABL)**
  - Surface shear stress boundary condition
  - Coriolis source term

Nalu Verification result for Ekman spiral exact solution



Nalu simulation of ABL driven by atmospheric pressure gradient balanced with Coriolis acceleration.