Exceptional service in the national interest

Sandia National Laboratories

April 24, 2019

ExaWind

Luc Berger-Vergiat, Jonathan Hu,
Siva Rajamanickam, Chris Lucchini
Center for Computing Research

Sandia National Laboratories
Albuquerque, New Mexico USA
SAND 2019-4589 PE

EXASCALE COMPUTING PROJECT

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

1 ExaWind

2 Node parallelism opportunities

3 Concrete Kokkos example: Multigrid preconditioner

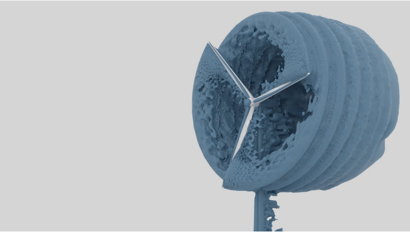
4 Conclusion

ExaWind overview

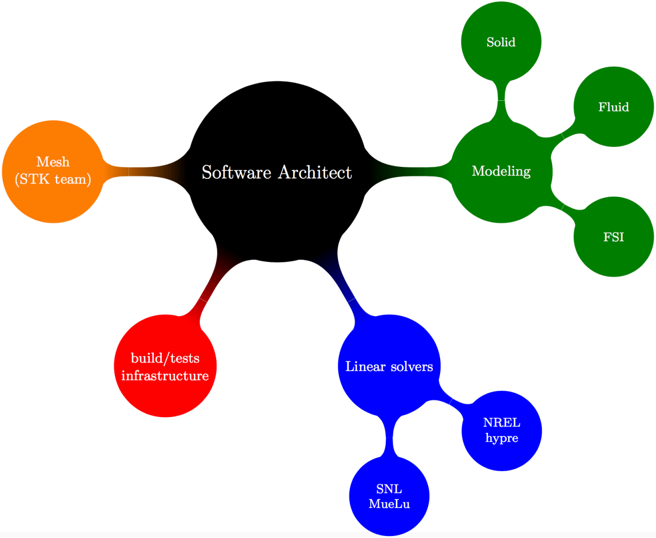
- ECP funded project for the simulation of wind farms:

  "The focus of the exa-scale wind project is to develop new simulation capabilities to improve our understanding of the performance of whole wind plants."

- Joint effort: NREL, Sandia, Oak Ridge and UT Austin



Development team organization



Code base

- Nalu-Wind: fork of Nalu, CFD code (uses STK, Tpetra, Belos, MueLu, ...)
- openFAST: FSI code interfacing Nalu-Wind to solid simulation
- AMReX: provides generic structured background mesh
- Tioga: overset mesh library used to mesh wind turbines
- Hypre: linear solver/preconditioner library

Outline

Key motivating factors

- Strategic: alignment with ECP goals + synergy with other exascale libraries
- Technical: to simulate wind farm steady-state within computational allocation
- Scientific: hard to publish new HPC research while ignoring CUDA/openMP

Technical chalenges

1. detect critical components to optimize?
   - mesh handling (STK package)
   - linear system assembly (mainly Tpetra)
   - linear solver (Trilinos solvers/Hypre solvers)
2. inter-library performance?
   - Trilinos packages have internal performance testing
   - leverage xSDK member libraries for interface performance
   - share raw data to avoid extra manipulation (raw pointers)
   - write custom adapters

Implementation approach

Multiple stakeholders $\rightarrow$ multiple approaches:

- AMReX: openMP, openACC and raw CUDA
- Hypre: CUDA at LLNL + custom CUDA features developed at NREL
- Trilinos: extensive use of Kokkos + Kokkos-Kernels

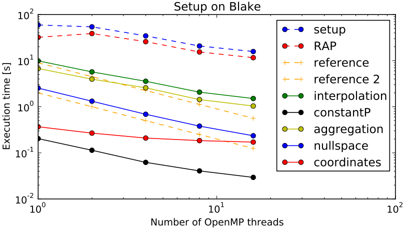
Leads to opportunities for comparison (Hypre vs Trilinos) but challenging integration and debugging (need expertise on multiple frameworks).

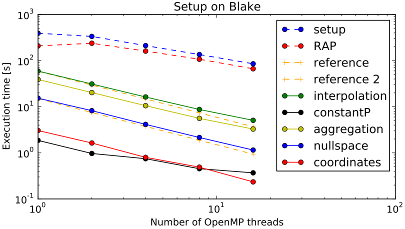
Outline

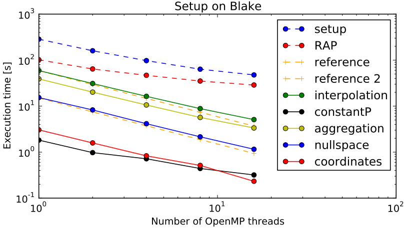
Structured algorithm in MueLu

- background mesh is structured
- "low hanging fruit" well know optimization opportunities
- new code path in MueLu $\rightarrow$ implementation/design freedom
- computationally expensive component $\rightarrow$ good impact if successful

Initial progress monitoring (Kokkos::OpenMP)







Does performance translate to GPU/CUDA?

Skylake

level	0	1	2	3	4
hierarchy setup	0.21	1.07	0.343	0.245	0.173
RAP		0.794	0.167	0.0774	0.0604
Smoother setup	0.185	0.054	0.0352	0.0352	
Geo Interp		0.135	0.0697	0.0665	0.0627
Struct Agg		0.0589	0.031	0.0282	0.0266
fill P		0.00653	0.00469	0.00437	0.00413
coarse coords		0.0472	0.0153	0.0152	0.0126
coarse NSP		0.00885	0.00614	0.00564	0.0061

P100

level	0	1	2	3	4
hierarchy setup	3.12	5.05	4.14	3.8	2.45
RAP		2.06	1.62	1.59	1.46
Smoother setup	3.08	1.9	1.49	1.13	
Geo Interp		0.741	0.639	0.64	0.656
Struct Agg		0.254	0.247	0.263	0.236
fill P		0.123	0.112	0.108	0.107
coarse coords		0.221	0.137	0.129	0.173
coarse NSP		0.0667	0.0664	0.064	0.0637

All timings are in seconds and averaged over 100 runs

First thoughts on Cuda performance

- run 8M, 1M and 125k problems on GPU as smaller grid do not make sense
- no scaling under 1M dofs?
- set "compute local triangular constants" to `false` for graph of P
- probably skip a bunch of things in `CoalesceDrop`, mainly need `Amalgamation` and again maybe not?
- `Kokkos::View::initialize()` seems expensive?
- hierarchical parallelism to increase scalability?
- using `kp_space_time_stack.so` and/or `kp_kernel_timer.so`, maybe nvprof at some point

Outline

1. ExaWind
2. Node parallelism opportunities
3. Concrete Kokkos example: Multigrid preconditioner
4. **Conclusion**

Some lessons learned

- Performance should be part of the design (from library/TPL selection to data structures/kernels implementation)
- Look for low hanging fruit/high impact kernels
- Monitor continuously! Kokkos tools provides easy diagnostics
- exploit compiler report and profiling (-qopt-report, nvprof)

Future effort:

1. pursue higher GPU scalability
2. examine launch overhead
3. use deterministic nature of structured RAP product