Exceptional service in the national interest

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
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
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

1 ExaWind

2 Node parallelism opportunities

3 Concrete Kokkos example: Multigrid preconditioner

4 Conclusion
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 challenges

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 → 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

1. ExaWind
2. Node parallelism opportunities
3. Concrete Kokkos example: Multigrid preconditioner
4. Conclusion
Structured algorithm in MueLu

- background mesh is structured
- "low hanging fruit" well know optimization opportunities
- new code path in MueLu → implementation/design freedom
- computationally expensive component → good impact if successful
Initial progress monitoring (Kokkos::OpenMP)

Number of OpenMP threads

Execution time [s]

Setup on Blake

- setup
- RAP
- reference
- reference 2
- interpolation
- constantP
- aggregation
- nullspace
- coordinates

April 24, 2019
## Does performance translate to GPU/CUDA?

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

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

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