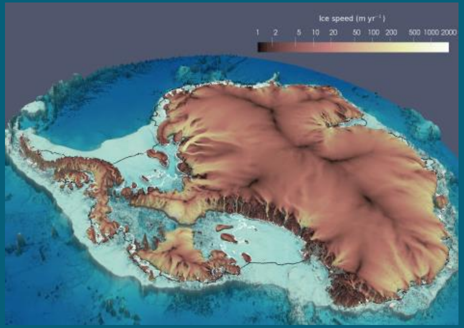
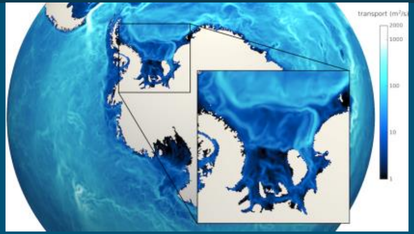
SAND2021-14919PE

Sandia National Laboratories

Automated Performance Testing and Tuning





December 2nd, 2021

PRESENTED BY

Jerry Watkins

Contributors: Max Carlson, Carolyn Kao, Kyle Shan, Irina Tezaur

Trilinos User-Developer Group Meeting

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

SAND

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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) Automated Performance Testing
2) Automated Performance Tuning
3) Conclusions/Discussion

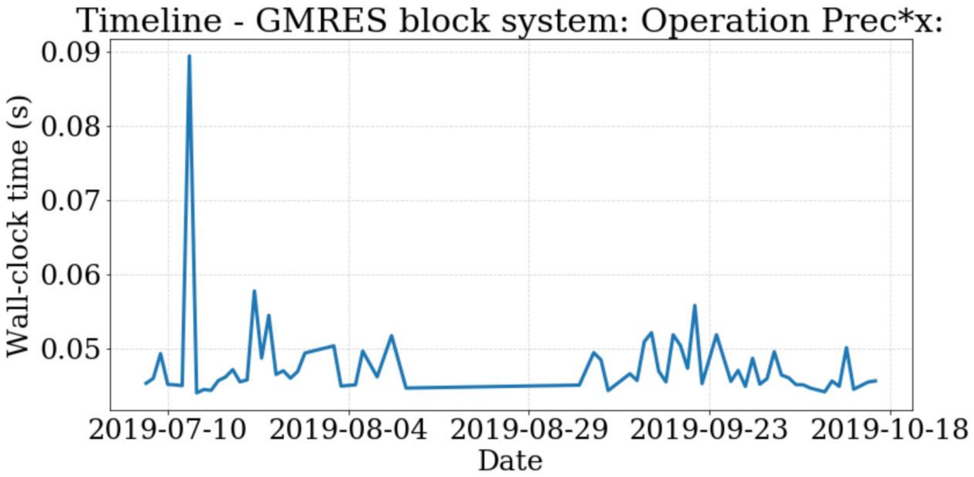
Automated Performance Testing

Motivation – Automated performance testing

1) **Maintaining performance and portability** in the presence of active development
   - Code changes can cause performance regressions (compiler/algorithmic optimizations)
   - Compiler/TPL changes can cause performance regressions (updates)
   - Architecture changes can cause performance regressions (CPU->GPU)

2) **Improving performance and portability** in the presence of active development
   - Performance can vary greatly with code changes (robustness)
   - Performance can vary greatly between compiler, architecture (CPU/GPU)
   - Performance can vary greatly between executions (noise)

3) **Manual testing/analysis is increasingly infeasible**
   - Directly tied to developer productivity
   - Progress has been made towards automating this task
   - Creating a performance test can be difficult
   - Are we doing better?

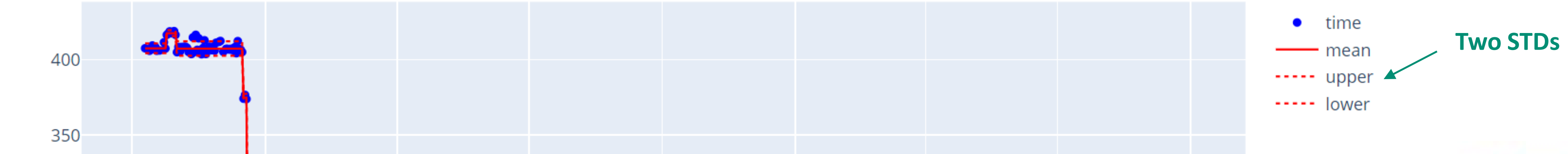


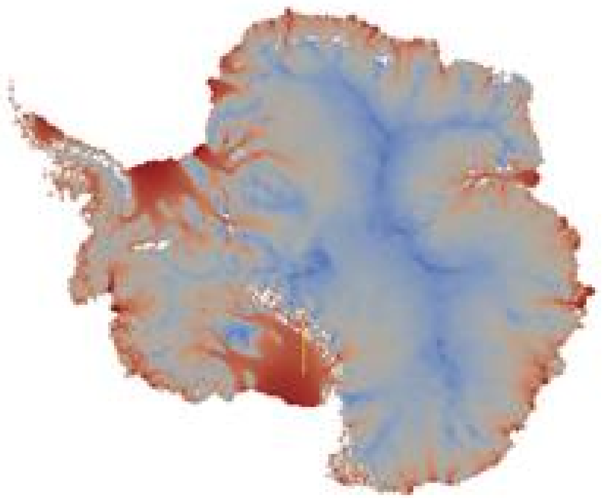
Time series data for code executed daily

Changepoint detection for performance testing

Maintaining performance and portability through **only** time series data plots still requires an expert to determine significant changes

- **Changepoint detection:** process of finding abrupt variations in time series data





Total simulation time for a 2-20km resolution Antarctica mesh, executed nightly in Albany Land Ice

Changepoint detection for performance testing

Single Changepoint:

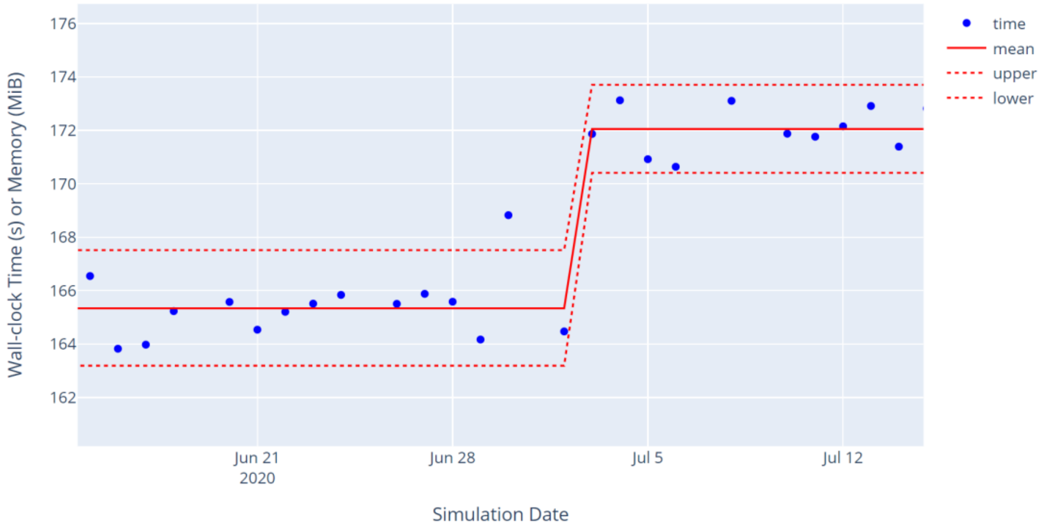
Given time series data: $X = \{x_1, x_2, \ldots, x_n\}$, and subset: $X_i^j = \{x_i, x_{i+1}, \ldots, x_j\}$,

Hypothesis tests:

Changepoint

All possible changepoints

$$H_0 : f_1^{\nu-1} = f_\nu^n, \qquad \forall \nu \in \mathcal{K},$$
$$H_A : f_1^{\nu-1} \neq f_\nu^n, \qquad \nu \in \mathcal{K},$$

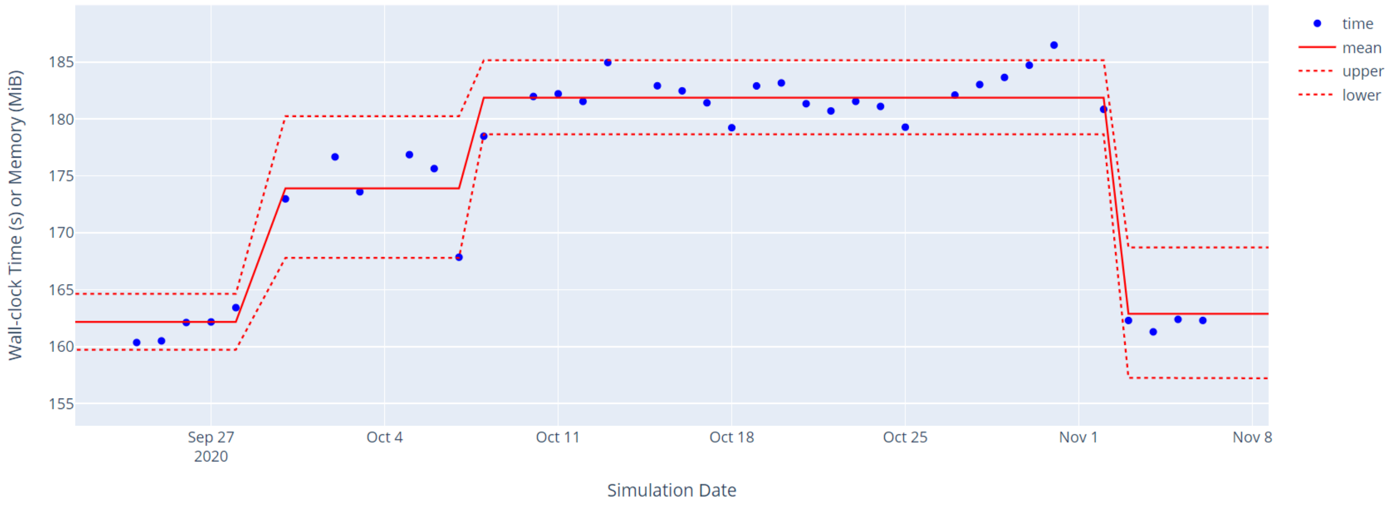


- **Null hypothesis** – X belongs to a single distribution
- **Alternative hypothesis** – there exists a changepoint ν s.t. $X_1^{\nu-1}$ and X_ν^n belong to two separate distributions
- Two-sample Student's t-test (equal variance), other options not tested
- Bonferroni correction used for multiple hypothesis testing: α/k
- Only $k = 10$ number of tests determined from largest changes in time series
- Outliers removed above ~3 STD, up to 10%

Changepoint detection for performance testing

Multiple Changepoint:

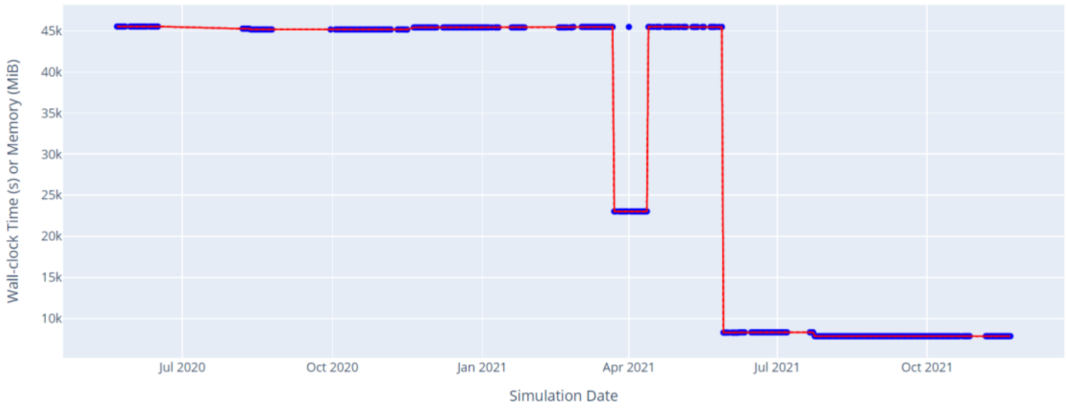
- Sequential algorithm
  - Store changepoints as they appear, new subset is created after change
  - $m = 3$ consecutive detections required before confirming changepoint



  - Max sample size or "lookback window" set to, $w = 30$, to avoid hypersensitivity

Implementation:

- Performance metrics are store in json files
- Automated post-processing in python
- Results uploaded in html and email reports
  - https://sandialabs.github.io/ikalash.github.io/



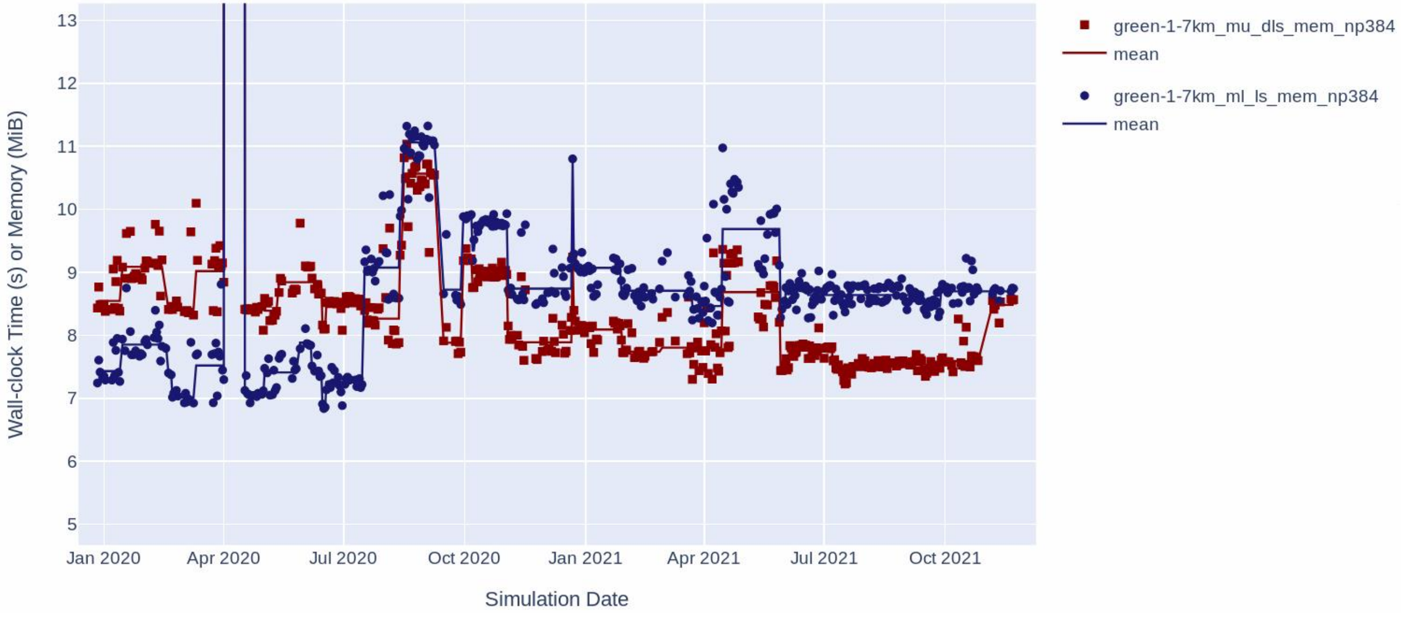
Example of improvements: Kokkos memory 45k->8k MiB [Greenland Ice Sheet, 1-7km, First-order Stokes]

Performance comparisons

Performance Analysis:

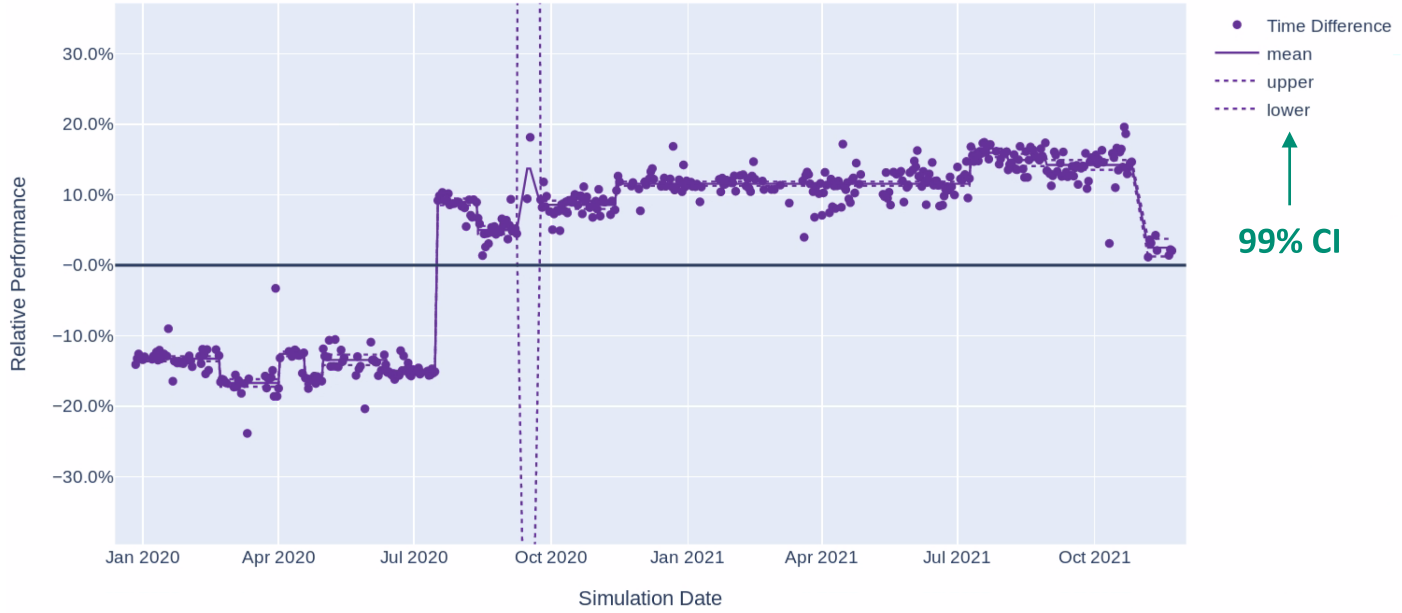
Given two time series: X and Y

- Compute difference
- Find changepoints
- Compute mean between changepoints and 99% confidence interval



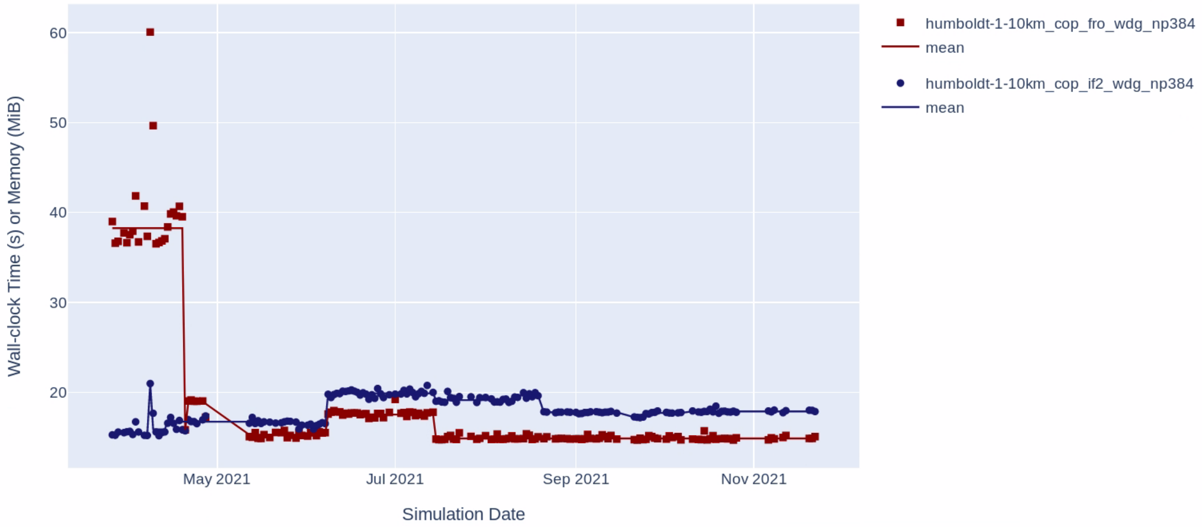
Example:

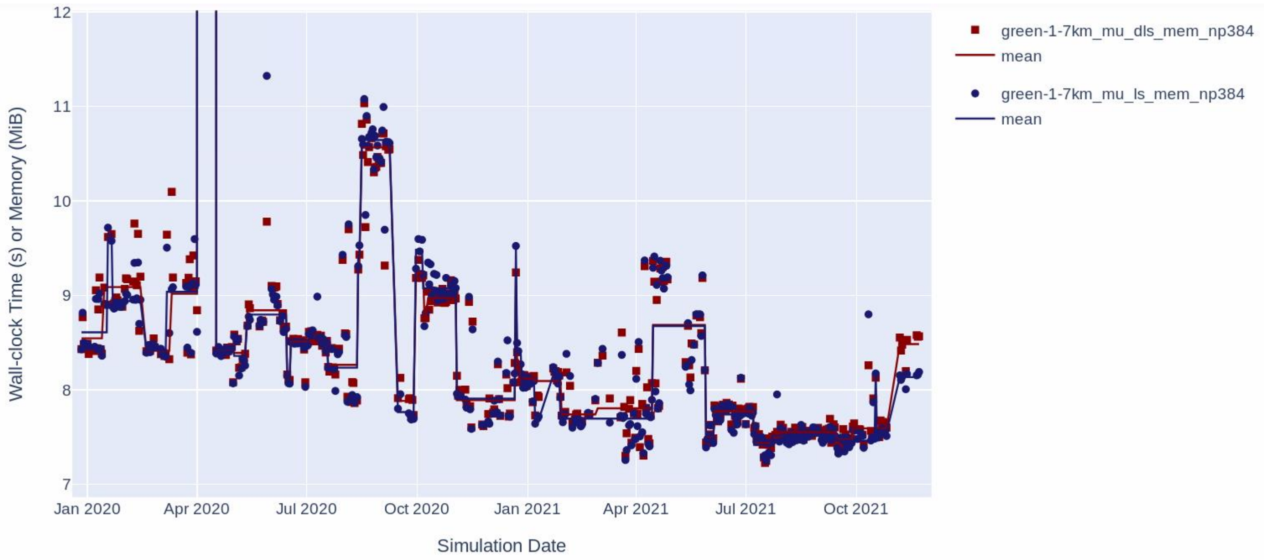
- Red/Squares: MueLu
- Blue/Circles: ML
- Latest results:
  - Starting Nov. 6th (8 samples)
  - Relative difference mean: **2.49%**
  - **99% CI: (1.16%, 3.81%)**



Performance comparisons

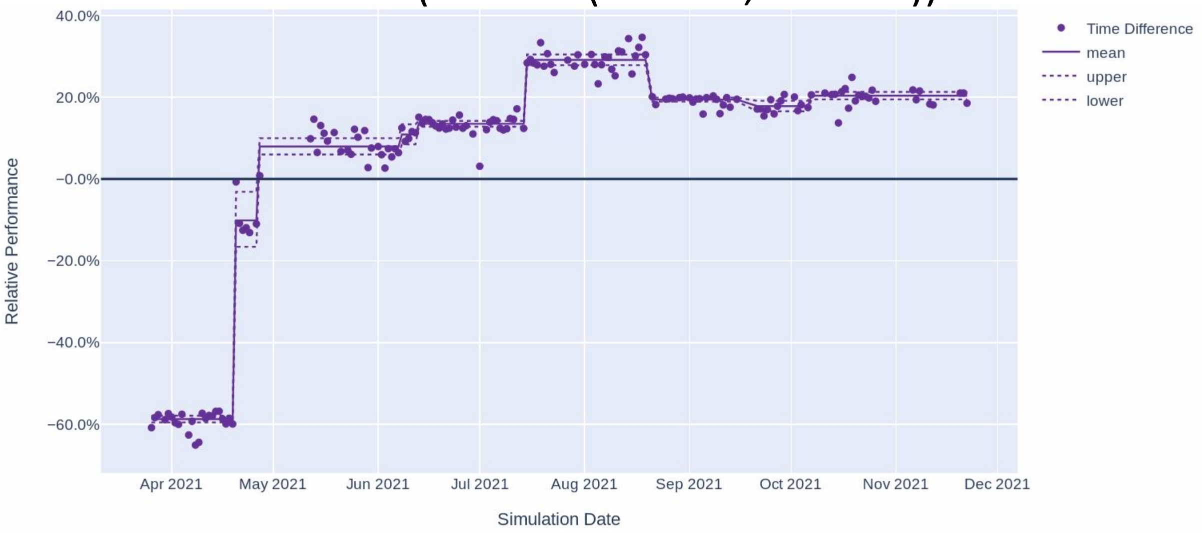
More Examples:

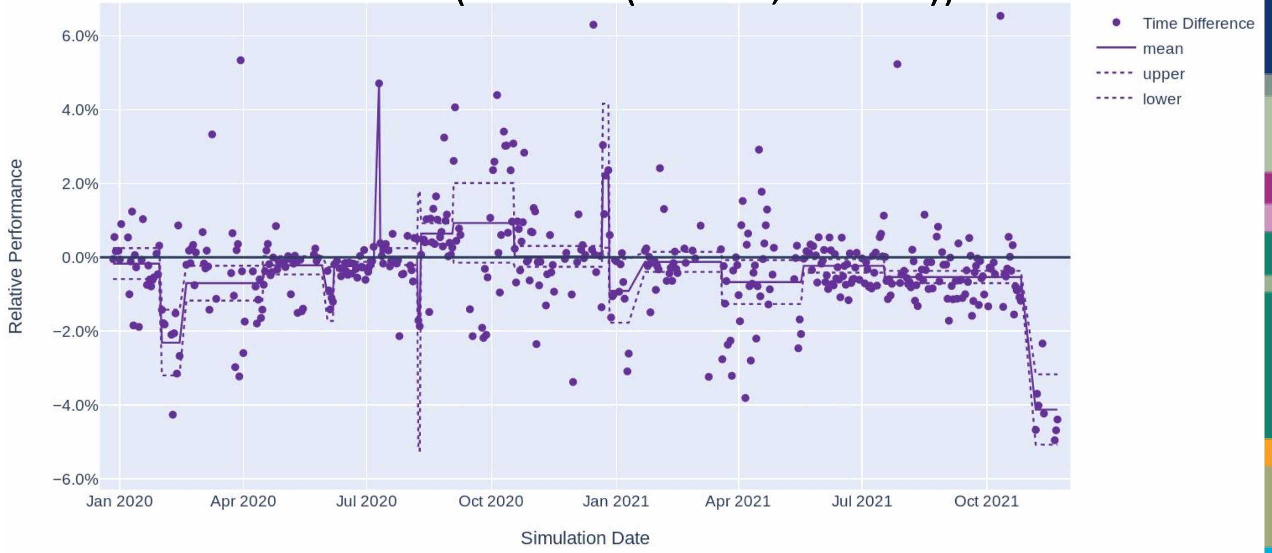




Ifpack2/FROSch:
20.40% (99% CI: (19.48%, 21.30%))

MueLu/Ifpack2 without/with block decoupling:
-4.12% (99% CI: (-5.13%, -3.11%))





Automated Performance Tuning

Motivation – Automated performance tuning

<u>Problem Description:</u>

- Find a **robust** set of parameters for optimal **performance** and **accuracy**.
- Often many runtime parameters to choose from (e.g. discretization, solver)
- Abundance of research/development on this topic

<u>Motivations are similar to performance testing:</u>

1) **Maintaining performance and portability** in the presence of active development
   - Code changes can cause optimal parameters to shift (algorithmic optimizations)
   - Compiler/TPL changes can cause optimal parameters to shift (new parameters)
   - Architecture changes can cause optimal parameters to shift (CPU->GPU)
2) **Improving performance and portability** in the presence of active development
   - Optimal parameters can vary greatly between compiler, architecture (CPU/GPU)
   - Performance can vary greatly with code changes (robustness)
3) **Manual tuning is increasingly infeasible**
   - Directly tied to developer/user productivity
   - Parameters become outdated

Blackbox optimization for performance tuning

Grid/Random Search:

- Simple, can be used for parameter exploration

Implementation:

- **Utilize performance test** to check robustness, performance and accuracy
- Parameters are in input files with **yaml** format
- Files are modified with python and **scikit-learn** is used for parameter selection

Partially Explored Extensions:

- Sequential optimization algorithms (model-based, Bayesian optimization)
- Sandia or open-source libraries (GPTune, Dakota)
- Integrate into performance testing framework (automated tuning)

Blackbox optimization for performance tuning

Example: Albany Land Ice multigrid preconditioner smoothers

Smoother parameters:

- Limited to three levels, two smoothers
- Good parameter ranges provided by Christian/Ichi

```
type: RELAXATION
ParameterList:
    'relaxation: type': MT Gauss-Seidel
    'relaxation: sweeps': positive integer
    'relaxation: damping factor': positive real number

type: RELAXATION
ParameterList:
    'relaxation: type': Two-stage Gauss-Seidel
    'relaxation: sweeps': positive integer
    'relaxation: inner damping factor': positive real number

type: CHEBYSHEV
ParameterList:
    'chebyshev: degree': positive integer
    'chebyshev: ratio eigenvalue': positive real number
    'chebyshev: eigenvalue max iterations': positive integer
```

Results:

- Applied to four cases (Greenland, 3-20km)
  - Different equations
  - Different architectures (CPU/GPU)
- 100 iterations, random search
- Timer: NOX Preconditioner + Linear Solve

Cases	Manual Tuning (sec.)	Autotuning (sec.)	Speedup
blake_vel	3.533972	2.658731	1.33x
blake_ent	3.07725	2.036044	1.51x
weaver_vel	19.13084	16.30672	1.17x
weaver_ent	19.76345	15.00014	1.32x

Cases	#Passed Runs	#Failed Runs	%Failure
blake_vel	70	30	30%
blake_ent	37	63	63%
weaver_vel	71	29	29%
weaver_ent	26	74	74%

Conclusions/Discussion

Conclusions/Discussion

Automated Performance Testing

- Changepoint detection adds some level of confidence to changes in performance (regressions/improvements)
  - Doesn’t always work – sometimes too sensitive – trade-offs when tuning
  - Still requires good performance tests/metrics
  - Still requires human-in-the-loop to address regressions
  - Large number of tests/metrics could be overwhelming

Automated Performance Tuning

- Blackbox optimization coupled with nightly testing adds some level of confidence in optimal parameters
  - Preliminary results are promising but more research needed
  - Large number of failures, raises questions about robustness/applicability
  - Optimization is expensive!