This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2021-4101C

kokkos

Kokkos Tools and Performance Tracking

D.Z. Poliakoff, A.J. Powell, J.R. Madsen, E.M. Ridgeway, D. LeBrun-Grandie, S. Rajamanickam, C.R. Trott

ECP

Sandia National Laboratories

Why Kokkos Tools?

- Kokkos app developers want to think in terms of Kokkos semantics, not the C++ implementation of Kokkos
- Tools functionality with *no recompiling*, that only costs the user when in use
  - USAGE:
    KOKKOS_PROFILE_LIBRARY=/path/to/kokkostool.so ./my_application
- Support for profiling, debugging, autotuning

Kokkos Autotuning

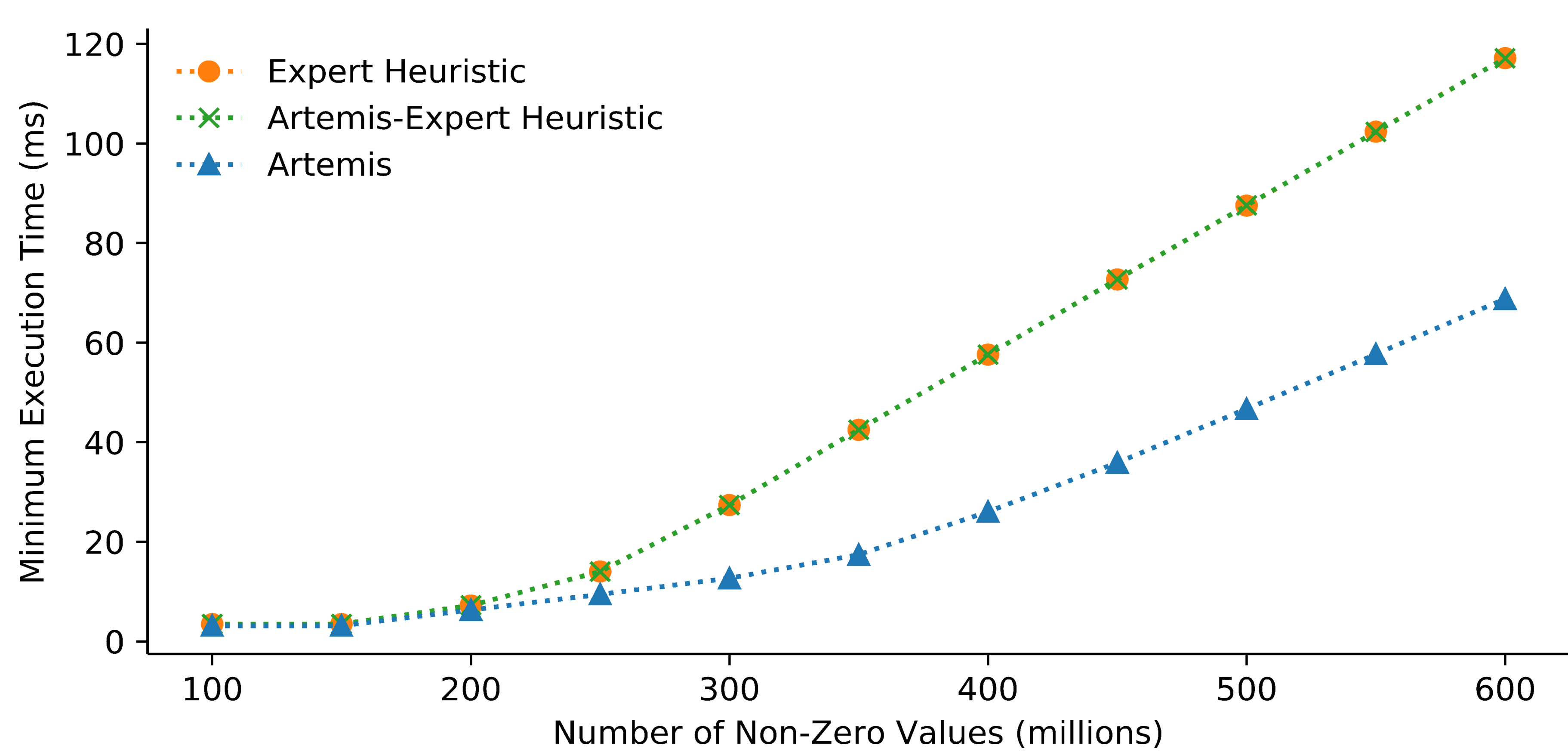

Figure 1. Kokkos Autotuning versus Expert Heuristic Runtime Optimization.

- HPC applications might be expected to use five backends (CUDA/HIP/SYCL/OpenMP/OpenMPTarget) across several architectures and compilers
- Heuristics just won't work
- Kokkos now supports autotuning through its no-recompilation tools interface (KokkosP)
- **Example: Kokkos Kernels SPMV**. Pick a Kokkos TeamPolicy "team size" and "vector length" in a CUDA execution, based on the size of the input vector.
- 2x speedups over well-designed, non-strawman heuristics
- Most users don't need to change their code, this is embedded in Kokkos

```
VariableInfo info;
info.category       = StatisticalCategory::kokkos_value_ratio;
info.type           = ValueType::kokkos_value_int64;
info.valueQuantity  = CandidateValueType::kokkos_value_unbounded;
auto matrix_size_id = declare_input_type( typeName: "matrix_size", info);
while (!done) {
  auto context_id = get_new_context_id();
  begin_context(context_id);
  VariableValue value =
      make_variable_value(matrix_size_id, int64_t(matrix_size));
  set_input_values(context_id,  count: 1, &value);
  parallel_for( str: "matmul", MDRangePolicy<>(start_tile, end_tile),
             KOKKOS_LAMBDA(int x, int y){
             });
  end_context(context_id);
}
```

Figure 2. Implementing Autotuning in Matrix Tiling.

- Implementation sample in which an MDRangePolicy's tile size is tuned based on a matrix size

Kokkos Performance Profiling

```
Kokkos::Tools::declareMetadata("examinimd.lattice_nx",
                               std::to_string(input->lattice_nx));
Kokkos::Tools::declareMetadata("examinimd.input_file",
                               std::string(input->input_file));
Kokkos::Tools::declareMetadata("examinimd.force_type",
                               std::to_string(input->force_type));
```

Figure 3. User-Defined Metadata for Kokkos Application Performance Profiling. Kokkos' built-in instrumentation (in Kokkos::Tools) supports user-defined metadata studies of code performance. Kokkos::Tools has two components: KokkosP Interface and Kokkos Tools. The KokkosP interface is the internal instrumentation layer of Kokkos, and is always available, even in release builds. This built-in instrumentation has near-zero overhead if no tools are loaded. Kokkos Tools use the KokkosP interface, and are loaded at runtime by Kokkos.

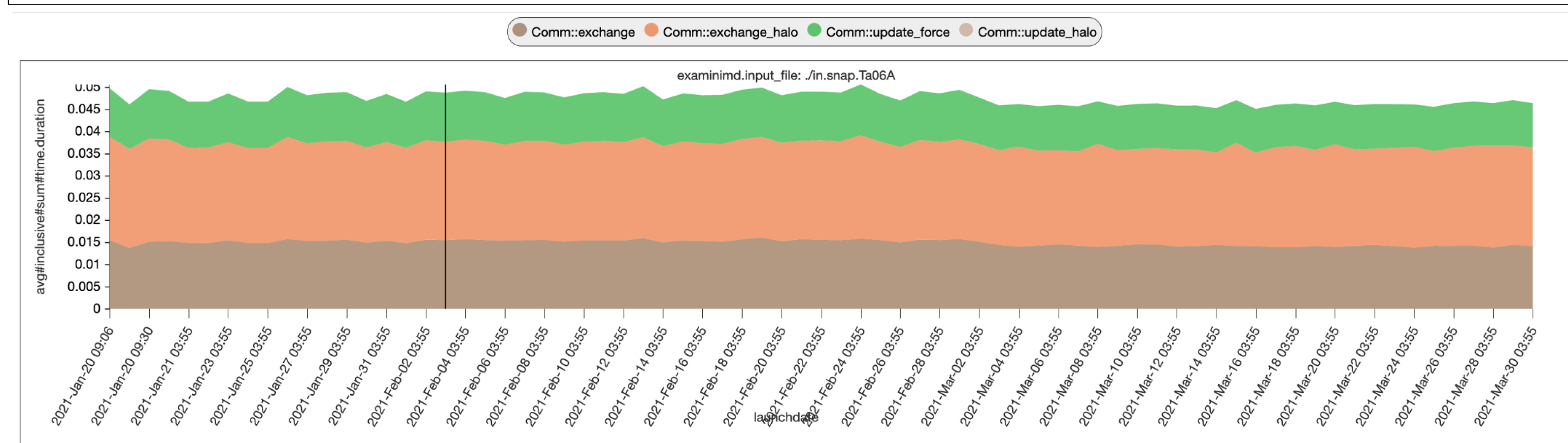

Figure 4. Performance Profile of ExaMiniMD Kokkos Application with SPOT Dashboard Visualization. The SPOT (**S**oftware **P**erformance **O**ptimization **T**racker) dashboard enables multiplatform performance analyses and visualization. Here, ExaMiniMD was instrumented to collect a wide array of metadata potentially useful as performance diagnostics and development pattern assessment. The runtime duration of an input file (in.snap.Ta06A) is shown above; the y-axis represents runtime (seconds), and the x-axis the run date. The different colors represent performance of Comm class functions, the next timed level.

Kernel-Level Performance Testing for Kokkos

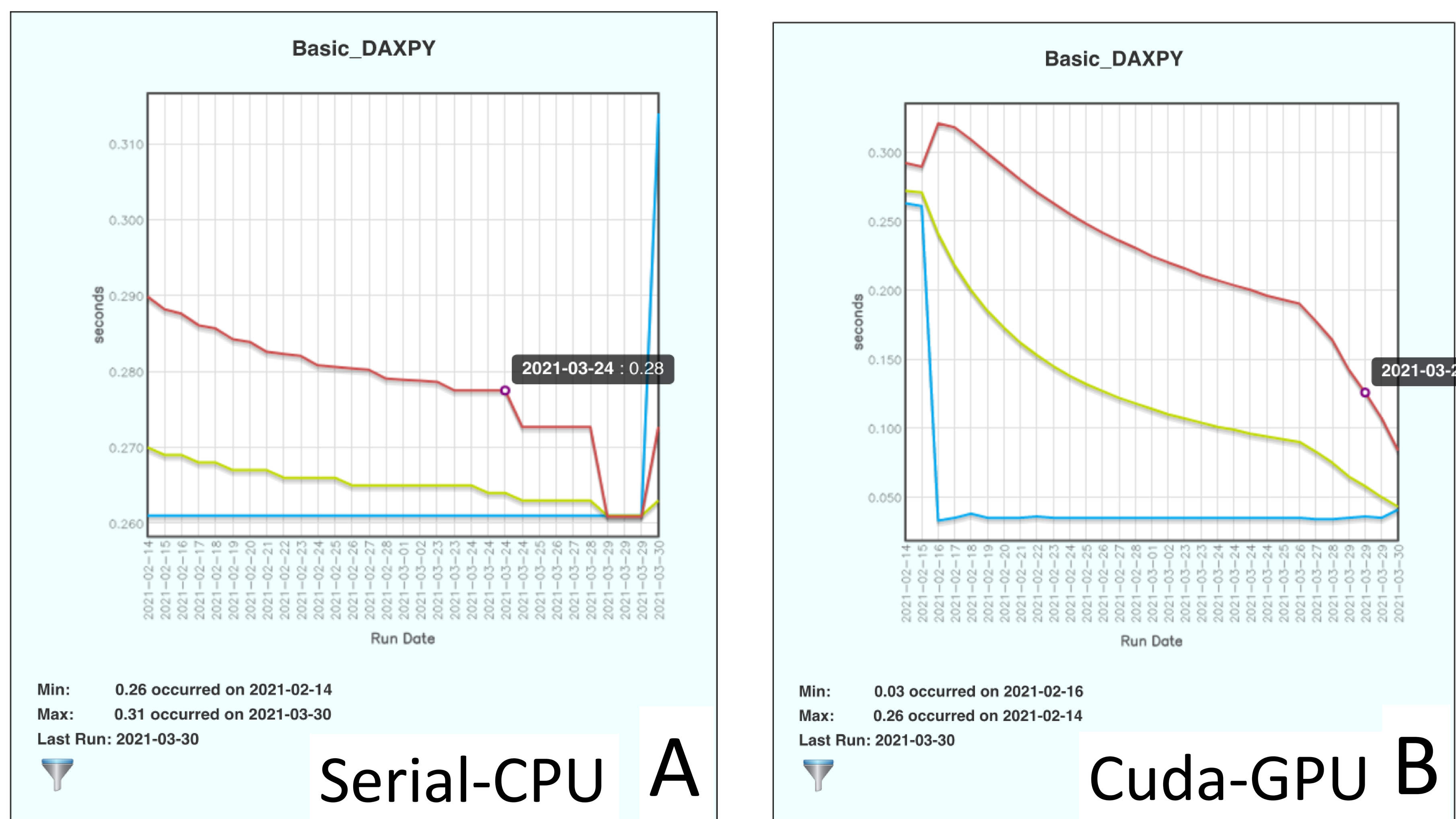

Figure 5: Kernel-Level Performance Testing in Kokkos. Kernel-level testing was implemented for Kokkos using RAJAPerf Suite. To date, we have completed implementation of basic kernels (*e.g.*, IF_QUAD, MULADDSUB, INIT3, *etc.*) in Kokkos for nightly build testing. Results for Kokkos DAXPY (linear combination of vectors, $y = \alpha x + y$) serial (panel A) and Cuda (panel B) implementations on NVIDIA V100. The watchr Jenkins plugin enabled visualization and exploration of nightly build performance data. Kernel-level testing enables efficient identification of performance regression.

Upcoming: Kokkos Debugger Integration

Timeline

- Q3CY21: Hosted SPOT prototypes
- Q3CY21: Initial Kokkos Kernels testing
- Q4CY21: Initial gdb debugger integration

NNSA National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY

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-NA-0003525.

ST 2.3.1.04 SNL ATDM PMR

SAND2019-0170 D