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.

SAND2022-2215C

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.

A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos

SIAM-PP'22, MS-38

Akihiro Hayashi, Matthew Whitlock, Sri Raj Paul,
Nicolas Morales, Keita Teranishi, Vivek Sarkar

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.

A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos

MOTIVATION

Georgia Tech

Sandia National Laboratories

Performance Portability

- Write one implementation that
  - can run on multiple platforms
  - can enable suitable memory access patterns for different platforms
  - can exploit platform-specific features
- Examples:
  - Kokkos
  - RAJA
  - DPC++

Kokkos

- What's Kokkos?
  - Kokkos enables single performance portable codes
    - ✓ CPUs, NVIDIA GPUs, …
  - Kokkos provides data abstractions critical for performance portability not available in OpenMP or OpenACC (Kokkos::View)
  - Essential parallel constructs
    - ✓ Kokkos::parallel_for();
    - ✓ Kokkos::parallel_reduce();
    - ✓ Kokkos::parallel_scan();

Georgia Tech

Sandia National Laboratories

Kokkos Example

```
1  View<int*> x("x", N);                              // Constuct a View (length: N)
2  // Parallel For
3  parallel_for(RangePolicy<>(0, N),                  // 1D Range (0 to N)
4               [=] (int i) {                         // C++11 lambda
5                 x(i) = i;                           // Computational Body
6               });
7
8  int result = 0;
9  // Parallel Reduce
10 parallel_reduce(RangePolicy<>(0, N),               // 1D Range (0 to N)
11                 [=] (int i, int &update) {         // C++11 lambda
12                   update += x(i);                  // Computational Body
13                 }, result);                        // Value to update
14 std::cout << result << std::endl;
```

Georgia Tech

Sandia National Laboratories

Enabling Resiliency in Performance Portable Programs

- Why important?
  - Increasing the number of nodes from 10,000 to 22,000 increases the probability of failures by a factor of 20x on the Blue Waters System [1]
- Typical approach: Manual checkpointing
  - e.g., for Kokkos [2]
- Research question: Any way to automate it?

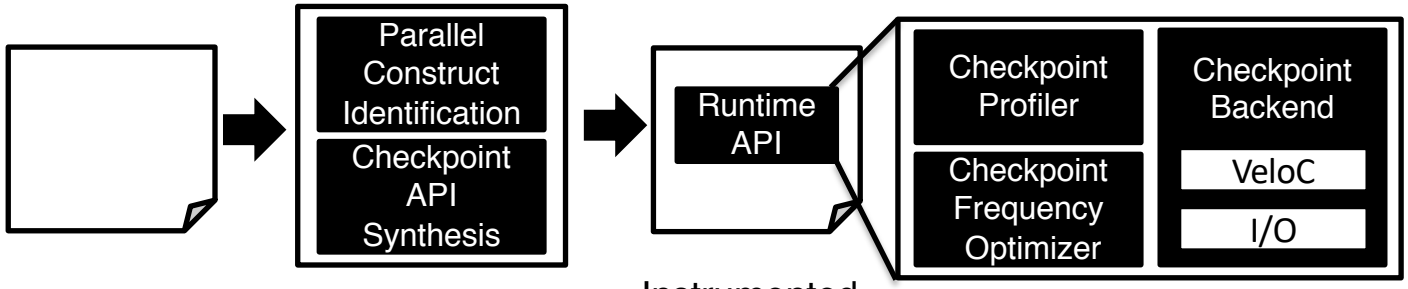
[1] Martino et al. “Measuring and Understanding Extreme-Scale Application Resilience: A Field Study of 5,000,000 HPC Application Runs” (DSN’15)

[2] Morales et al. “Towards High Performance Resilience Using Performance Portable Abstractions” (Euro-Par’21)

Georgia Tech

Sandia National Laboratories

Big picture



- We propose:
  - a source-to-source compiler that analyzes Kokkos programs and automatically generates Checkpoint/restart API
  - a runtime provides
    - ✓ 1) platform-independent Checkpoint/restart API routines
    - ✓ 2) optimizes the frequency of checkpointing when possible

Cooperate

Georgia Tech

Sandia National Laboratories

Related Work

	Language	IR	Note
Elkhouly et al.	Sequential C + directive on data access	LLVM IR	Sequential
Bronevetsky et al.	OpenMP C	ROSE	OpenMP
Lehr et al.	MPI + C	LLVM IR	Type Safe Checking Only
Shahneous et al.	OpenSHMEM	LLVM IR	Suggestion Only
Our Approach	Kokkos (C++)	Clang AST	Full translation

Note: compilers are typically used to perform selective instruction duplication (SID) e.g., Swift

Georgia Tech

Sandia National Laboratories

Related work (Cont'd)

- clang-tidy for Kokkos (not for resilience)

```
$ clang-tidy kokkos-implicit-this-capture.cpp -checks=-\*,kokkos\* -- -
I$PWD/Inputs/kokkos
…
warning: Lambda passed to parallel_for implicitly captures this. [kokkos-
implicit-this-capture]
        10, KOKKOS_LAMBDA(int x) { my_int = x; });
```

https://github.com/kokkos/llvm-project/tree/master/
https://github.com/kokkos/llvm-project/tree/master/clang-tools-extra/clang-tidy/kokkos

Georgia Tech Sandia National Laboratories

A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos

DESIGN

Georgia Tech

Sandia National Laboratories

High-level Overview (lambda)

Original Code

```
1  Kokkos::View<int*> A(“A”, N);
2  Kokkos::View<int*> B(“B”, N);
3  for (int t = 0; t < T; t++) {
4    Kokkos::parallel_for(”modA”,
5    N, KOKKOS_LAMBDA (int i) {
6      A(i) = B(i);
7    });
8    …
9  }
```

Translated

```
1  Kokkos::View<int*> A(“A”, N);
2  Kokkos::View<int*> B(“B”, N);
3  loop_enter();
4  for (int t = 0; t < T; t++) {
5    loop_start(t);
6    Kokkos::parallel_for(”modA”,
7    N, KOKKOS_LAMBDA (int i) {
8      A(i) = B(i);
9    });
10   checkpoint(A);
11   …
12   loop_end(t);
13 }
14 loop_exit();
```

Georgia Tech

Sandia National Laboratories

High-level Overview (functor)

Original Code

```
1  Kokkos::View<int*> A("A", N);
2  Kokkos::View<int*> B("B", N);
3  for (int t = 0; t < T; t++) {
4    Functor functor(A, B);
5    Kokkos::parallel_for("modA",
6    N, functor);
7  …
8  }
```

```
struct Functor {
 void operator ()(const size_t i)
 const {
      A(idx) = B(idx);
 }
 // synthesized by translator
 void checkpoint() { checkpoint(A); }
};
```

Translated

```
1  Kokkos::View<int*> A("A", N);
2  Kokkos::View<int*> B("B", N);
3  loop_enter();
4  for (int t = 0; t < T; t++) {
5    loop_start(t);
6    Functor functor(A, B);
7    Kokkos::parallel_for("modA",
8    N, functor);
9    functor.checkpoint(A);
10   …
11   loop_end(t);
12 }
13 loop_exit();
```

Georgia Tech

Sandia National Laboratories

Compiler

- Open question: How to analyze C++ program?
  - LLVM IR-based
    - ✓ Pros: Lots of analyzer and optimizer passes
    - ✓ Cons: It's very hard to locate the start/end of a specific function
  - Clang AST-based (Our approach)
    - ✓ Pros: it's relatively easy to locate the start/end of a specific function
    - ✓ Cons: its analyzers/optimizers would not be as powerful as LLVM's ones

```
// ClangAST
LambdaExpr … '(lambda at test.cpp:55:40)'
| …
| - CompoundStmt …
      BinaryOperator … int' lvalue '='
      | -CXXoperatorCallExpr … 'int' lvalue '()'
      | | …
      | | -DeclRefExpr … 'const Kokkos::View<int *>' … 'A'
…
```

Georgia Tech National Laboratories

Compiler (Cont'd)

- Checkpoint API insertion
  1. Identify Kokkos constructs in user's program
     - ✓ Kokkos::parallel_for, reduce, scan
  2. Analyze the body of a lambda expression/the parenthesis operator and identify which View is used
  3. Insert checkpoint API immediately after Kokkos::parallel_for
     - ✓ Optimization: do not checkpoint a View that is read-only
- Loop API Insertion
  - Insert loop_enter(), loop_start(), loop_end(), loop_exit() to all for-loops

Note: the current implementation performs the above actions for each lambda and it does not do any data flow analyses to analyze the lifetime of Views.

How robust is the compiler?

- Our compiler is able to translate 9 Kokkos applications including 3 real-world applications

Benchmark Suite	Benchmark
N/A	Heat-Dist, NimbleSM
Polybench MPI (stencil)	Jacobi-1D/2D, Heat-3D, Fdtd-2D, Seidel-2D
Mantevo	miniAero/miniFE

Georgia Tech

Sandia National Laboratories

CAC Runtime

- Provides a generic checkpoint/restart API independent of datatype and checkpoint runtime
  - Uses Kokkos::ViewHolderBase for de-templating Views but keeping memory information
    - ✓ Kokkos::View<int*>, Kokkos::View<double*>, …
  - Keeps format of Resilient Kokkos [1] configuration file to enable changing checkpoint backends without application code changes
    - ✓ VeloC, StdFile, …

Georgia Tech

Sandia National Laboratories

[1] Morales et al. “Towards High Performance Resilience Using Performance Portable Abstractions” (Euro-Par’21)

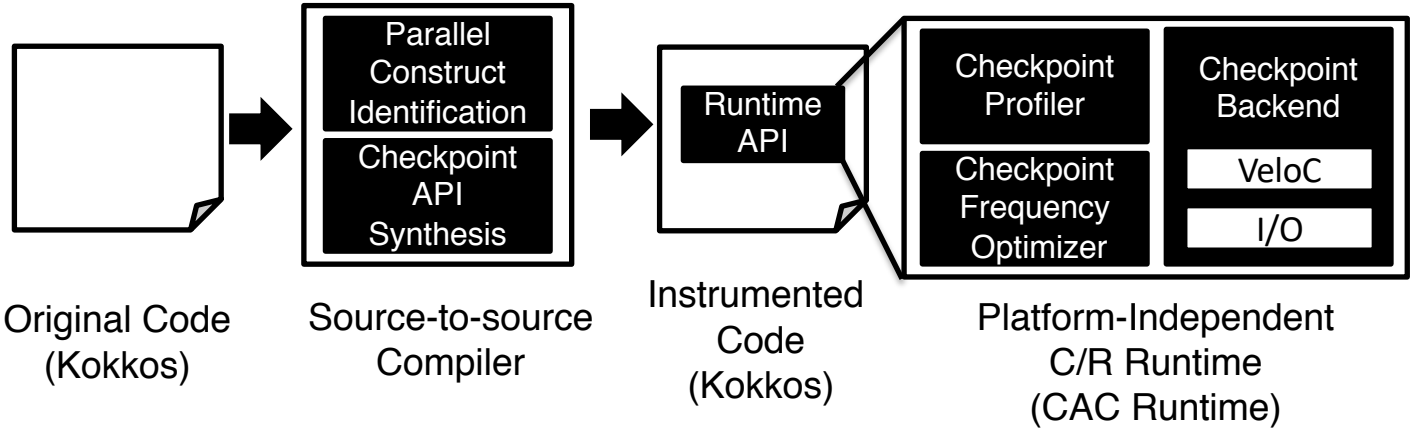
CAC Runtime (Cont'd)

- Gathers checkpoint regions within loops for write aggregations
- Automatically updates loop iteration variables after recovery
- Dynamically optimizes checkpoint frequency
  - User provides estimated MTBF
  - Calculates time-weighted average checkpoint cost

A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos

IMPLEMENTATION

Implementation



- Compiler: implemented as a Clang's tool (executable)
- Runtime: a standalone library which depends on
  - Kokkos-Resilience (Proprietary)
  - VeloC: https://veloc.readthedocs.io/en/latest/

Georgia Tech

Sandia National Laboratories

A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos

PRELIMINARY EVALUATIONS

Applications

- Platform:
  - Sandia Machine: Kahuna
    - ✓ a dual socket intel E5-2683v3 2.00GHz CPU (28 cores)
    - ✓ High memory data-analytics machine (256GB)
    - ✓ 56GB/s FDR Infiniband network
    - ✓ CephFS distributed filesystem
- Application: Heat-distribution (MPI+Kokkos)
  - 4-points iterative stencil kernel
  - Variants:
    - ✓ Raw VeloC
    - ✓ Auto-translated by the compiler + CACRuntime

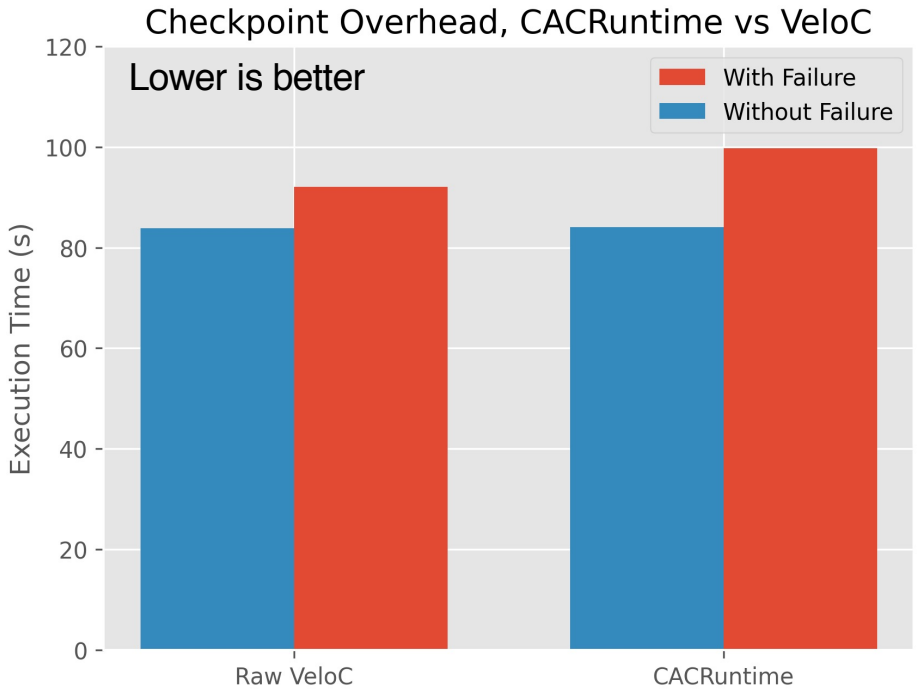
Georgia Tech

Sandia National Laboratories

Heat-distribution

- Parameters
  - 8 nodes
    - ✓256MB checkpoint per node
  - 6 Checkpoints per (non-failing) run
- Contrived MTTF
  - On the order of seconds

Heat-distribution Results



- CACRuntime
  - Very similar to the baseline without failures
  - Some more overhead in recovery
    - ✓ Doesn't know to skip restoring initialization views
    - ✓ Checkpoints an additional time after recovery
  - Our version checkpoints as frequently as the raw version after frequency optimization
  - Our version does not require any manual checkpointing

Georgia Tech

Sandia National Laboratories

A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos

CONCLUSIONS

Georgia Tech

Sandia National Laboratories

Conclusions

- Introduced A Cooperative Compiler and Runtime Checkpoint/Restart Approach for Kokkos
  - Compiler automatically generates checkpoint/restart API
    - ✓ robust enough to translate 9 Kokkos+MPI applications
  - Runtime automatically
    - ✓ performs write aggregation
    - ✓ optimizes the frequency of checkpointing
    - ✓ updates loop iteration variables after recovery

Georgia Tech

Sandia National Laboratories

Future Work

- Evaluate other applications
  - An MPI version of Polybench
  - Mantevo
  - NimbleSM
- Compare our approach with Kokkos-Resilience [1]
- Enhance the CACRuntime
  - Asynchronous Checkpointing
  - Runtime Alias Checking
- Use GPUs

Georgia Tech

Sandia National Laboratories

[1] Morales et al. “Towards High Performance Resilience Using Performance Portable Abstractions” (Euro-Par’21)