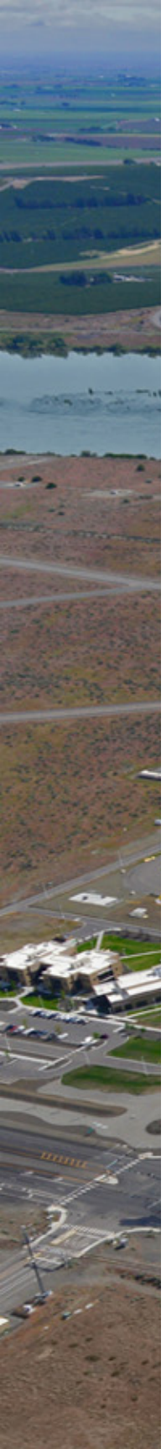


Pacific Northwest
NATIONAL LABORATORY

HPC Software Verification in Action: A Case Study with Tensor Transposition

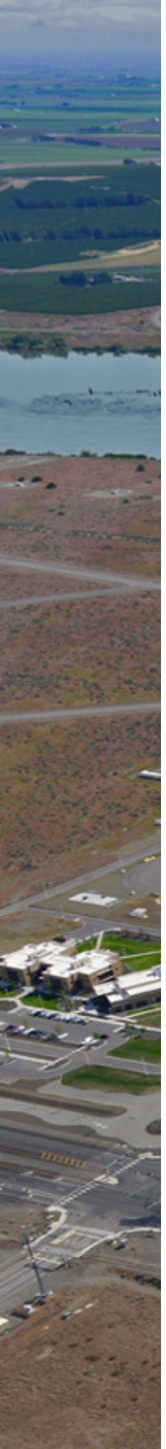
Erdal Mutlu, Ajay Panyala, Sriram Krishnamoorthy

Post Doctoral Research Associate

erdal.mutlu@pnnl.gov

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

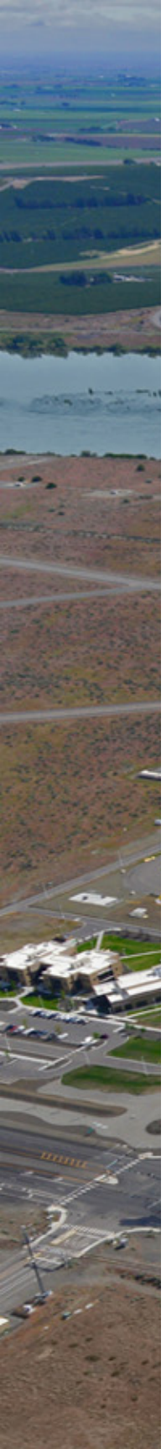


Pacific Northwest NATIONAL LABORATORY

Correctness in HPC

- Verification challenges in HPC
  - hard to specify scientific applications
  - different sources of non-determinism
    - ✓ concurrency
    - ✓ approximation
    - ✓ randomness
  - floating point arithmetic

- Where to start?
  - functional equivalence checking
    - ✓ optimized implementation vs simple trusted implementation
    - ✓ model checking techniques
    - ✓ symbolic execution



Pacific Northwest NATIONAL LABORATORY

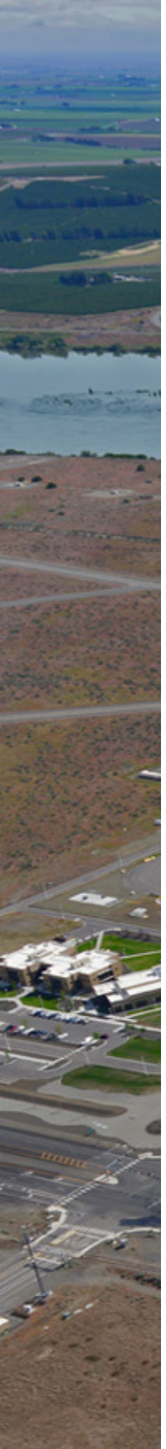
Tensor Algebra

- Tensor Operations
  - majority of machine learning (ML) application
  - scientific applications: quantum chemistry
- Tensor Transposition

$$A_{\Pi(i_0,i_1,\ldots,i_{d-1})} \leftarrow B_{i_0,i_1,\ldots,i_{d-1}}$$

- E.g. 2-D Matrix transposition : A(j, i) = B(i, j)

```
for(int i = 0; i < row_size; i++){
  for(int j = 0; j < col_size; j++){
    src_id = i * col_size + j;
    dest_id = j * row_size + j;
    A[dest_id] = B[src_id];
  }
}
```

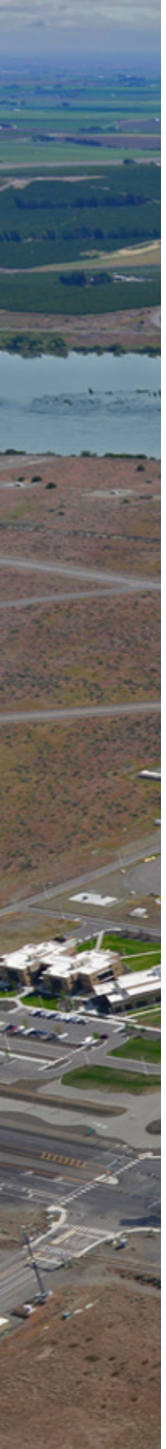


Pacific Northwest NATIONAL LABORATORY

Tensor Transpose Implementations

- NWChem implementation
  - specialized implementations for 2-D and 4-D tensors
  - optimized for production use
  - uses heavy pointer arithmetic
- TAMM implementation
  - generalized C++ implementation
  - supports reduction
- Reference implementation
  - specialized for each permutation

```
int  void ref_impl_3D_0_2_1(double *dest,
  in                        const double *src,
void tce_sort(double *A, double *B, /*..*/) {
  //...
  for (i = 0; i < bSize[3]; i++) {
    aPtr[2] = aPtr[3];
    for (j = 0; j < bSize[2]; j++) {
      aPtr[1] = aPtr[2];
      for (k = 0; k < bSize[1]; k++) {
        aPtr[0] = aPtr[1];
        for (l = 0; l < bSize[0]; l++) {
          *bPtr++ = *aPtr[0];                    buf,
          aPtr[0] += bJump[0];                   ) {
        }
        aPtr[1] += bJump[1];
      }
      aPtr[2] += bJump[2];
    }
    aPtr[3] += bJump[3];
  }                                              ) {
                                                 lest)];
  //...
}
      } else {
        ref_impl_3D_0_1_2(dest, src, dims);
  }   }
}   }
```

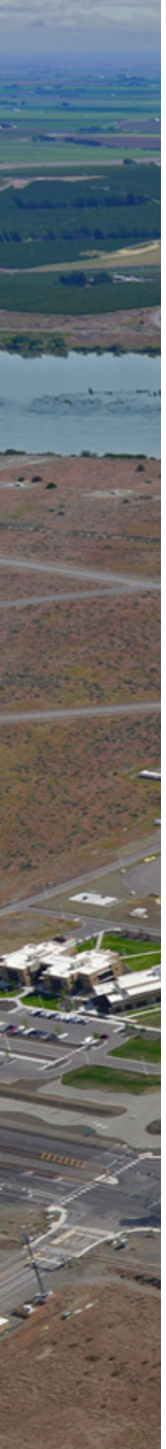


Pacific Northwest NATIONAL LABORATORY

Equivalence Checking Tools

- CIVL: the concurrency intermediate verification language
  - symbolic execution-based model checking
  - supports many HPC-relevant parallel programming models
    - ✓ OpenMP, pthreads, CUDA
  - checks different properties
    - ✓ race freedom, deadlock freedom, assertion violation

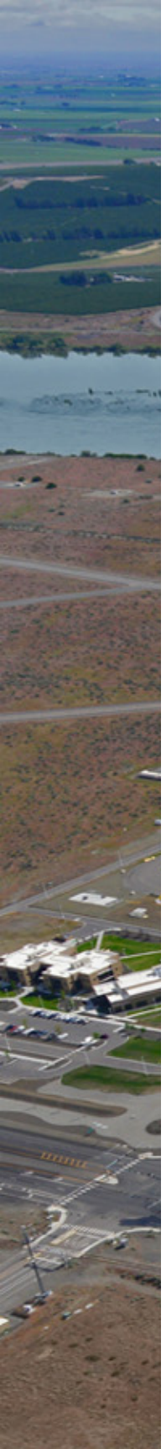
- CodeThorn
  - trace-based equality checking technique
  - re-write updates into a normalized form
  - compare resulting traces



Pacific Northwest NATIONAL LABORATORY

Research Questions

- Can current equivalence checking tools be used in **production code**?
- How **scalable** are these tools' performance?
- Where these tools **fall short**?



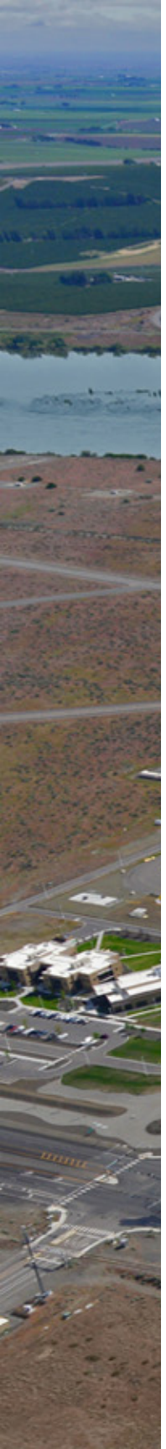
Pacific Northwest NATIONAL LABORATORY

Experimental Setup

- Compared 2, 3 and 4 dimensional tensor transpose implementations
  - NWChem
  - TAMM
  - Reference

- CIVL setup
  - Bounded input size
    - ✓ $assume(1 <= N <= BOUND) ;
  - Fixed input size
    - ✓ $input N = X;

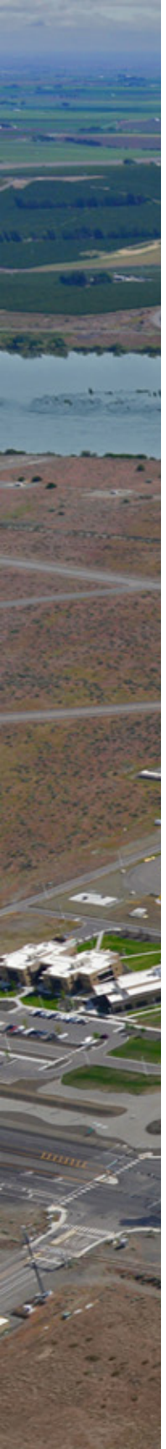
- CodeThorn setup
  - Fixed input size



Pacific Northwest NATIONAL LABORATORY

Research Questions

- Can current equivalence checking tools be used in **production code**?
  - Worked for us (with some code modification) ☺
  - Valuable input during development
- How **scalable** are these tools?
- Where these tools **fall short**?



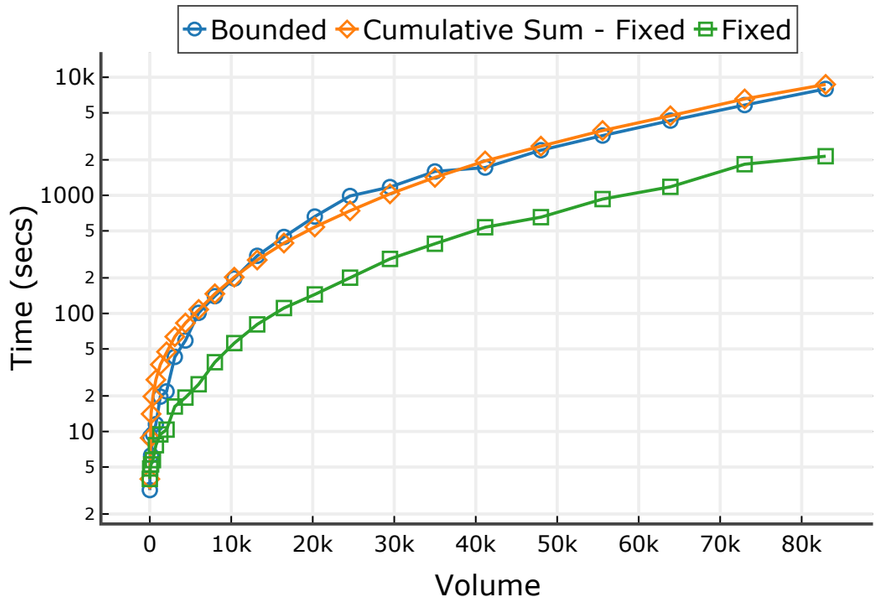
Pacific Northwest NATIONAL LABORATORY

Research Questions

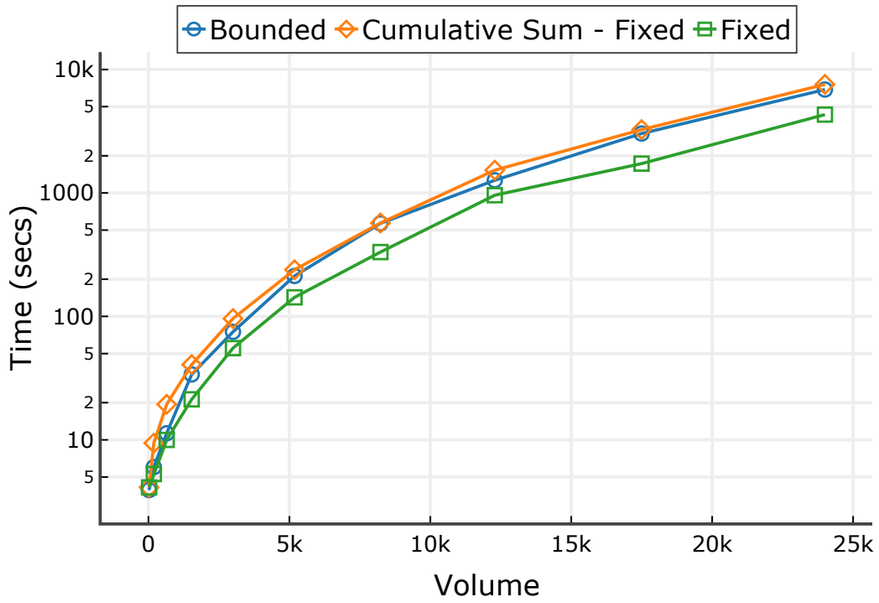
- Can current equivalence checking tools be used in **production code**?
  - Worked for us (with some code modification) ☺
  - Valuable input during development
- How **scalable** are these tools?
  - Works relatively well in small size inputs
  - Using different modes helps
- Where these tools fall **short**?

Results – CIVL

- Functional equality checking for NWChem against reference implementation

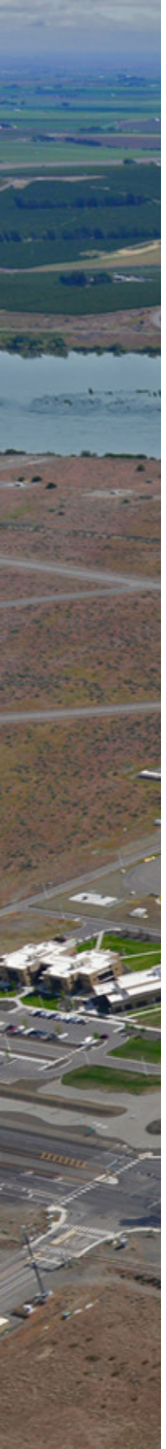


3D - NWChem vs Ref



4D - NWChem vs Ref

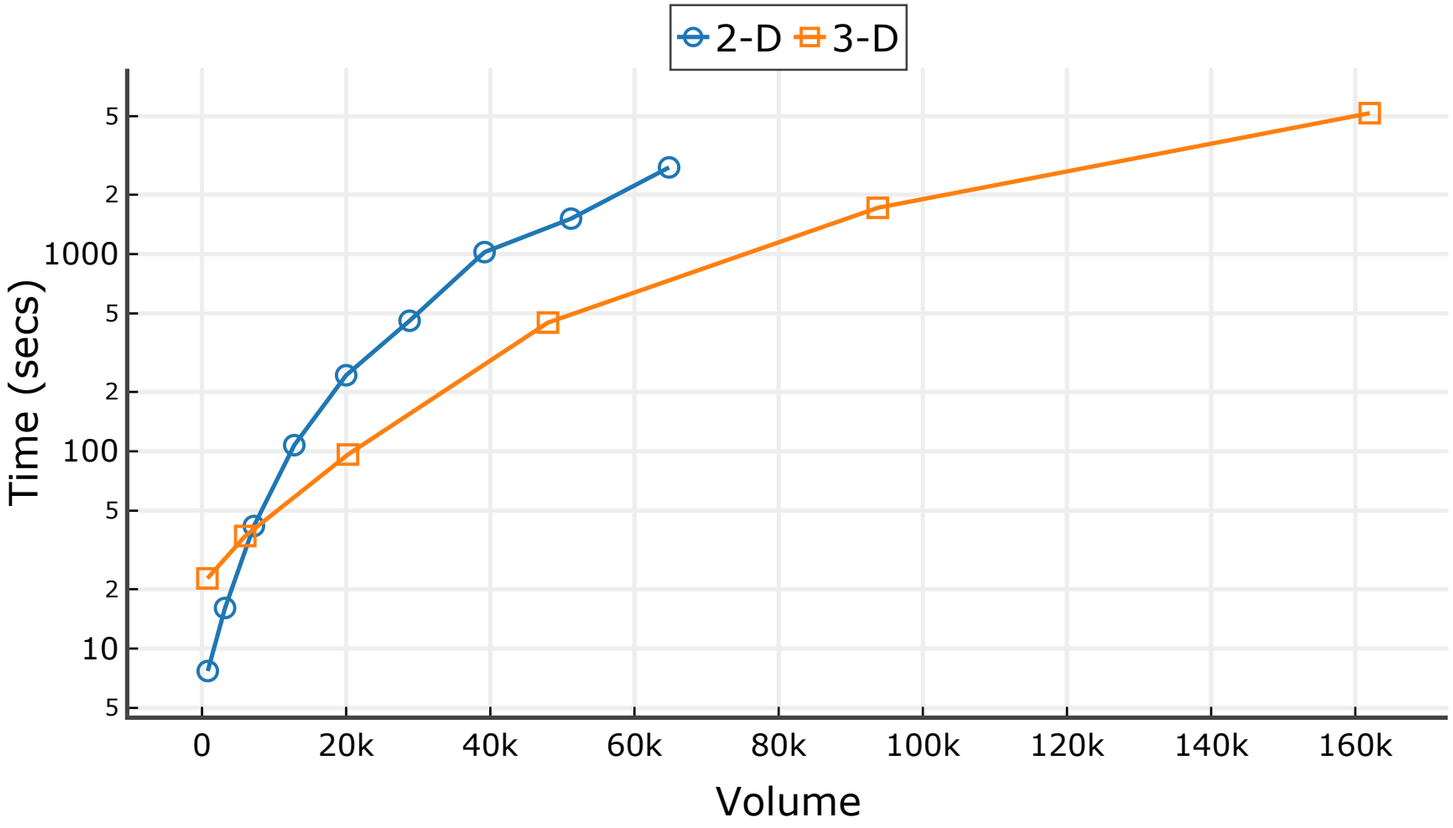
Volume = # of Permutations * Loop Volume

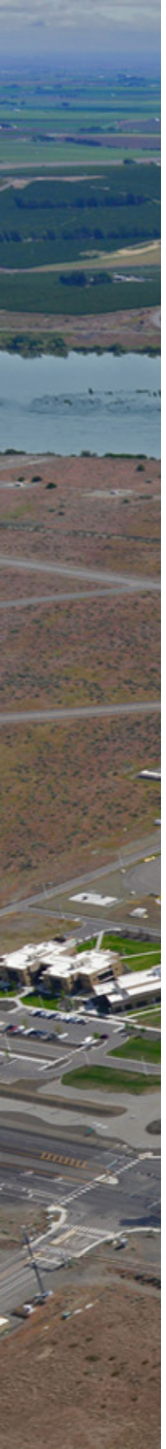


Pacific Northwest NATIONAL LABORATORY

Results – CodeThorn

- Functional equality checking with CodeThorn

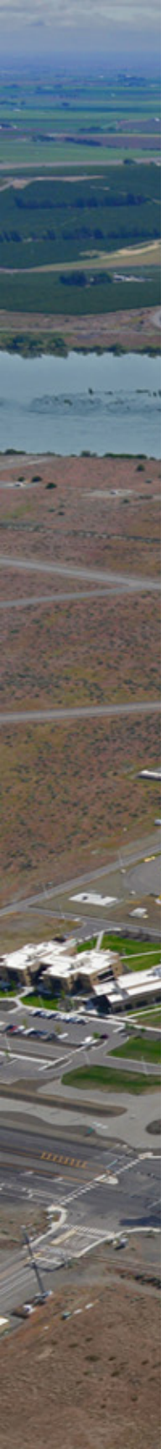




Pacific Northwest NATIONAL LABORATORY

Research Questions

- Can current equivalence checking tools be used in **production code**?
  - Worked for us (with some code modification) ☺
  - Valuable input during development
- How **scalable** are these tools?
  - Works relatively well in small size inputs
  - Using different modes helps
- Where these tools **fall short**?
  - Supporting programming languages (C++)
  - Problems with pointer arithmetic



Pacific Northwest NATIONAL LABORATORY

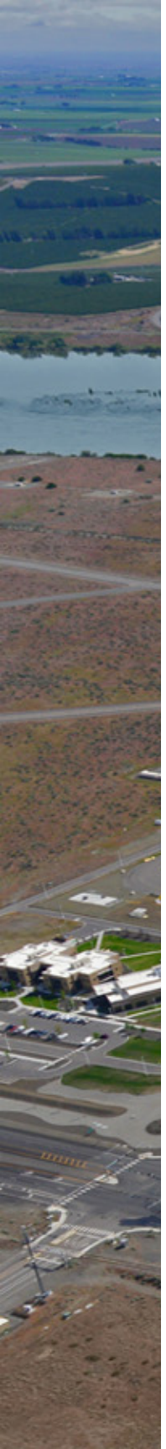
Conclusion

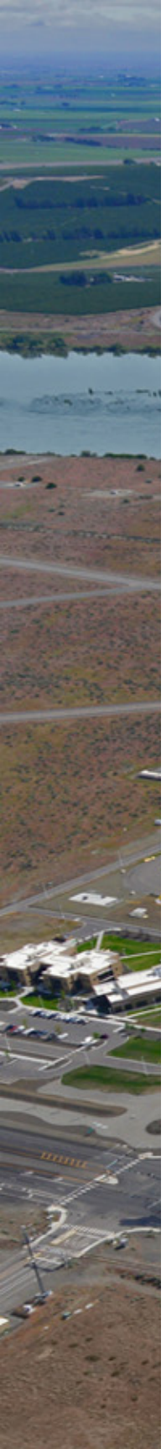
- Emerging heterogeneous architectures in HPC:
  - complex optimized versions of the same algorithm
  - hard to write specify/verify
  - floating point arithmetic

- Functional equivalence checking
  - both, CIVL and CodeThorn, can be used in production codes
  - symbolic evaluation based
  - works relatively well for small input size
  - some caveats:
    - ✓ subset of C programming language is supported
    - ✓ false positives (case with pointer arithmetic)

Pacific Northwest
NATIONAL LABORATORY

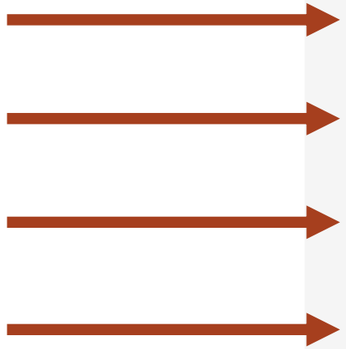
Thank you





Pacific Northwest NATIONAL LABORATORY

False Positive – CIVL



```
void tce_sort(double *A, double *B, /*..*/) {
  //...
  for (i = 0; i < bSize[3]; i++) {
    aPtr[2] = aPtr[3];
    for (j = 0; j < bSize[2]; j++) {
      aPtr[1] = aPtr[2];
      for (k = 0; k < bSize[1]; k++) {
        aPtr[0] = aPtr[1];
        for (l = 0; l < bSize[0]; l++) {
          *bPtr++ = *aPtr[0];
          aPtr[0] += bJump[0];
        }
        aPtr[1] += bJump[1];
      }
      aPtr[2] += bJump[2];
    }
    aPtr[3] += bJump[3];
  }
  //...
}
```