Improving LLVM – Applied Compiler R&D

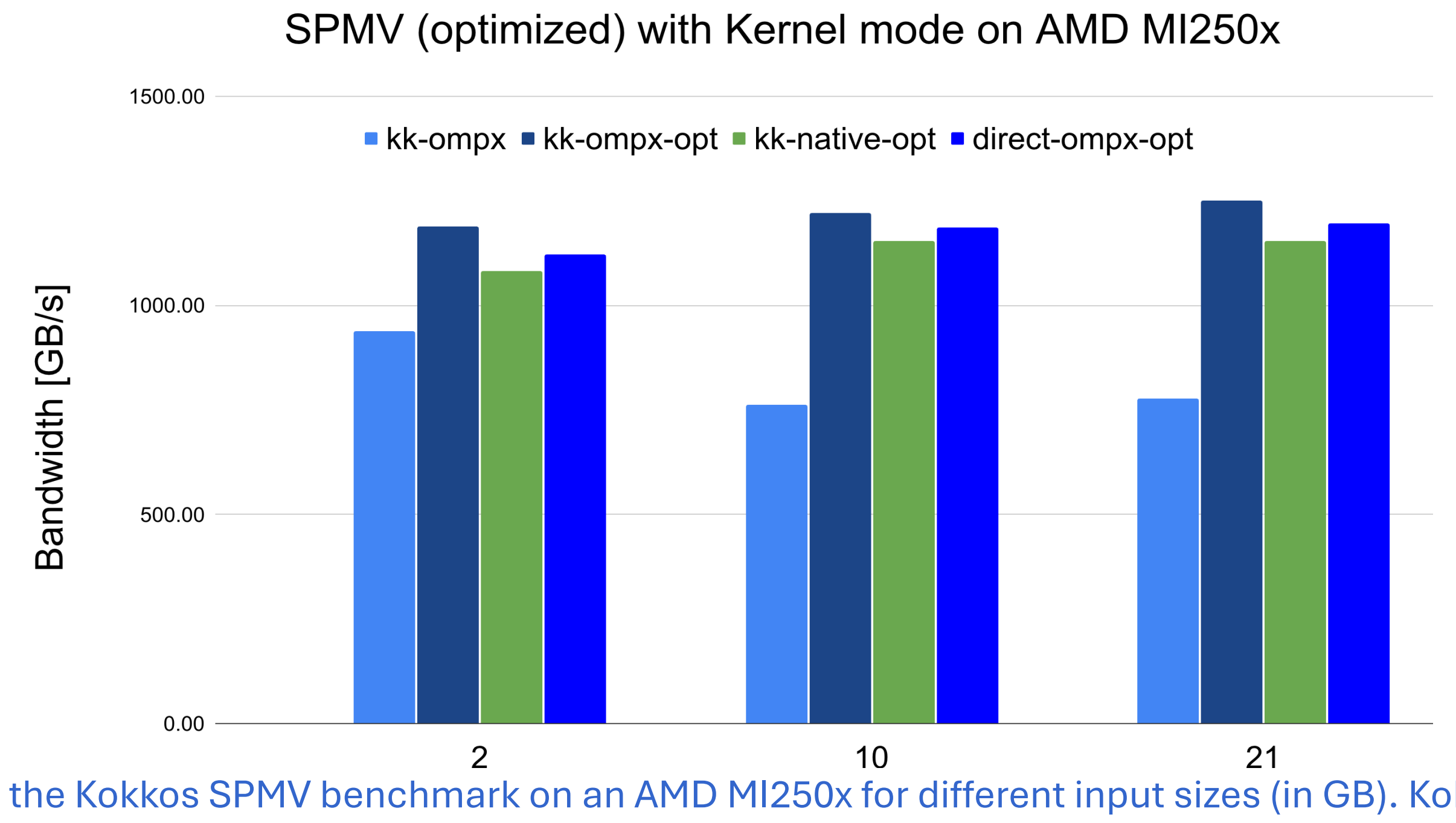
Opening the “Compiler Black Box” for users and developers

Johannes Doerfert (LLNL), K. Sala (LLNL), E. Wright (LLNL), B. Shan (Stony Brook University), E. McDonough (LLNL/Penn State), S. Tian (AMD), S. Olivier (SNL), R. Gayatri (LBNL)

Compiler technology is a foundation of computing and offers vast areas for research and development, including:
- improved execution performance (optimization, new features, etc.)
- lower development time (compile time, debugging, etc.)
- tooling for insight generation and ML-applications

LLVM/OpenMP Offload Extended: Native GPU Performance via a Portable Model

OpenMP’s GPU capabilities are portable, but performance is lacking. We extended OpenMP offload, and Kokkos “OMPX” can match HIP/CUDA.

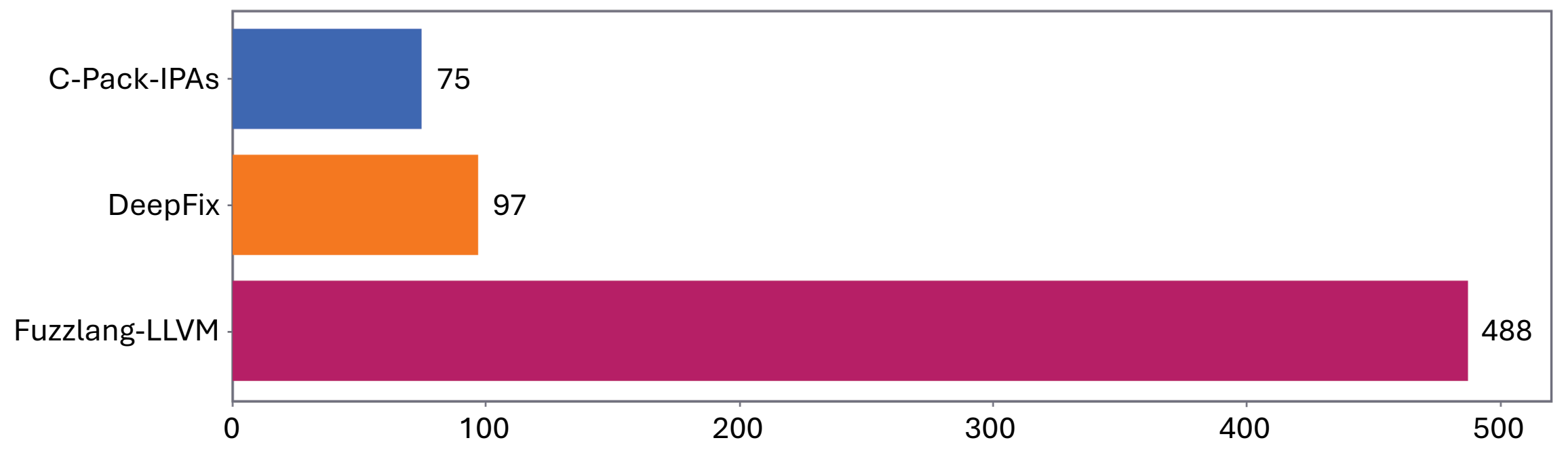


Performance of the Kokkos SPMV benchmark on an AMD MI250x for different input sizes (in GB). Kokkos using extended OpenMP offload (available via LLVM) outperforms the native HIP backend and matches CUDA on NVIDIA GPUs.

Fuzzlang: Properly Teaching LLMs about Compiler Errors

(Compiler) Machine learning (ML) work needs principled approaches with quantified success metrics and scalable datasets.

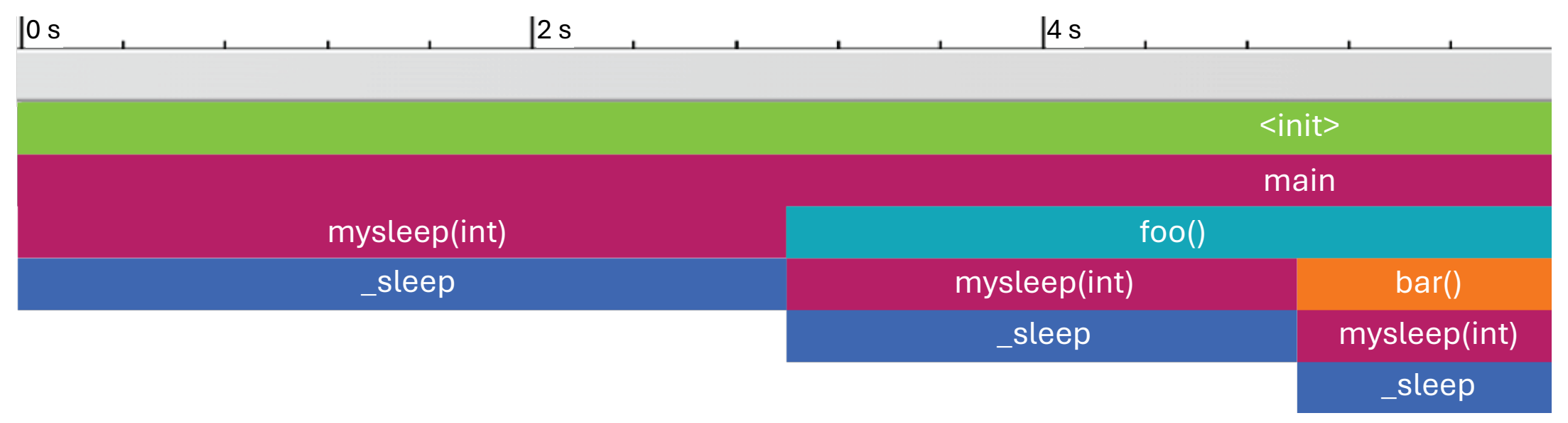
Step 1: Generate and curate compiler error examples with known solutions
Step 2: Fine-tune LLMs (Llama 3.8B: 37% -> 93%, GPT-4o-mini: 72% -> 97%)



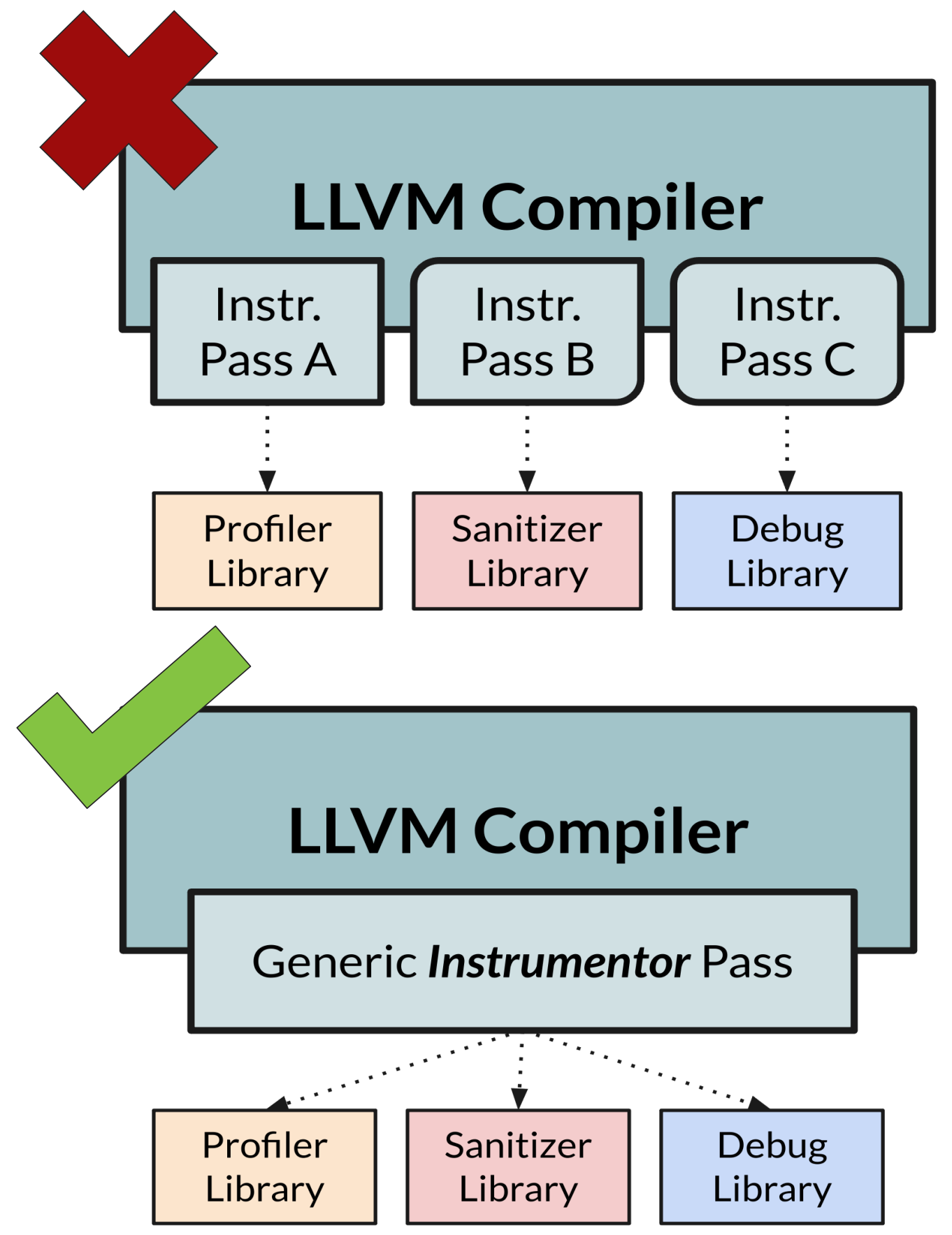
Comparison of the number of distinct error kinds contained in state-of-the-art “error datasets” and Fuzzlang-LLVM. By quantifying the error types, we can measure completeness and identify errors that are not covered sufficiently.

LLVM/Instrumentor: Code Instrumentation for You

Generic, customizable, and easily extendable instrumentation for any code, any HPC compiler, and without prior compiler knowledge. Users only provide configuration as JSON and runtime, the LLVM/Instrumentor does the heavy lifting.



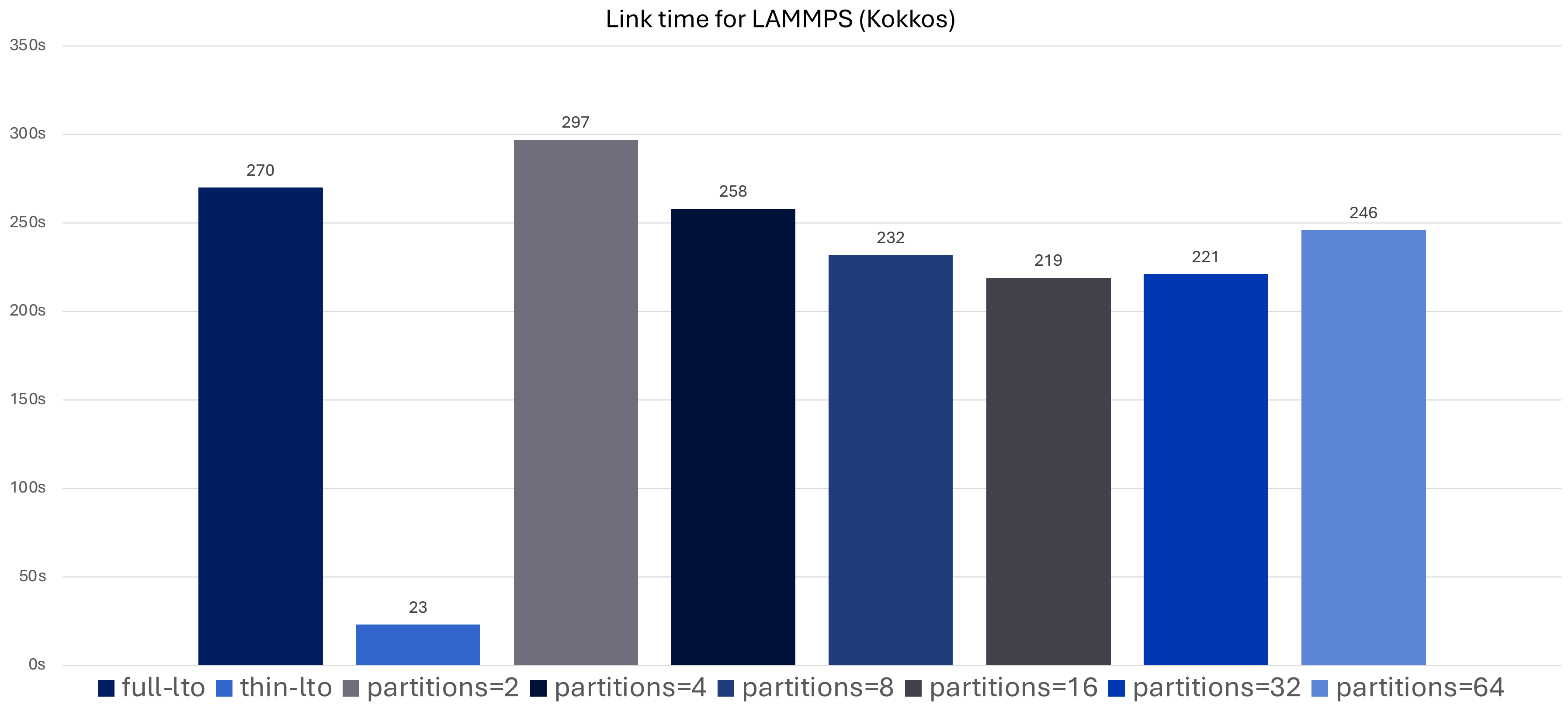
Execution trace captured by a profiler written in 56 lines of C++ and 26 lines of JSON configuration. Visualized with Chrome tracing.



Eliminate duplicate instrumentation efforts and provide sophisticated tooling to end-users w/o compiler knowledge.

Beyond Performance: Reducing (GPU) Compile Times

Compile and link times are less flashy than performance but crucial to ensure productivity and decrease costs, especially since there is little user influence.



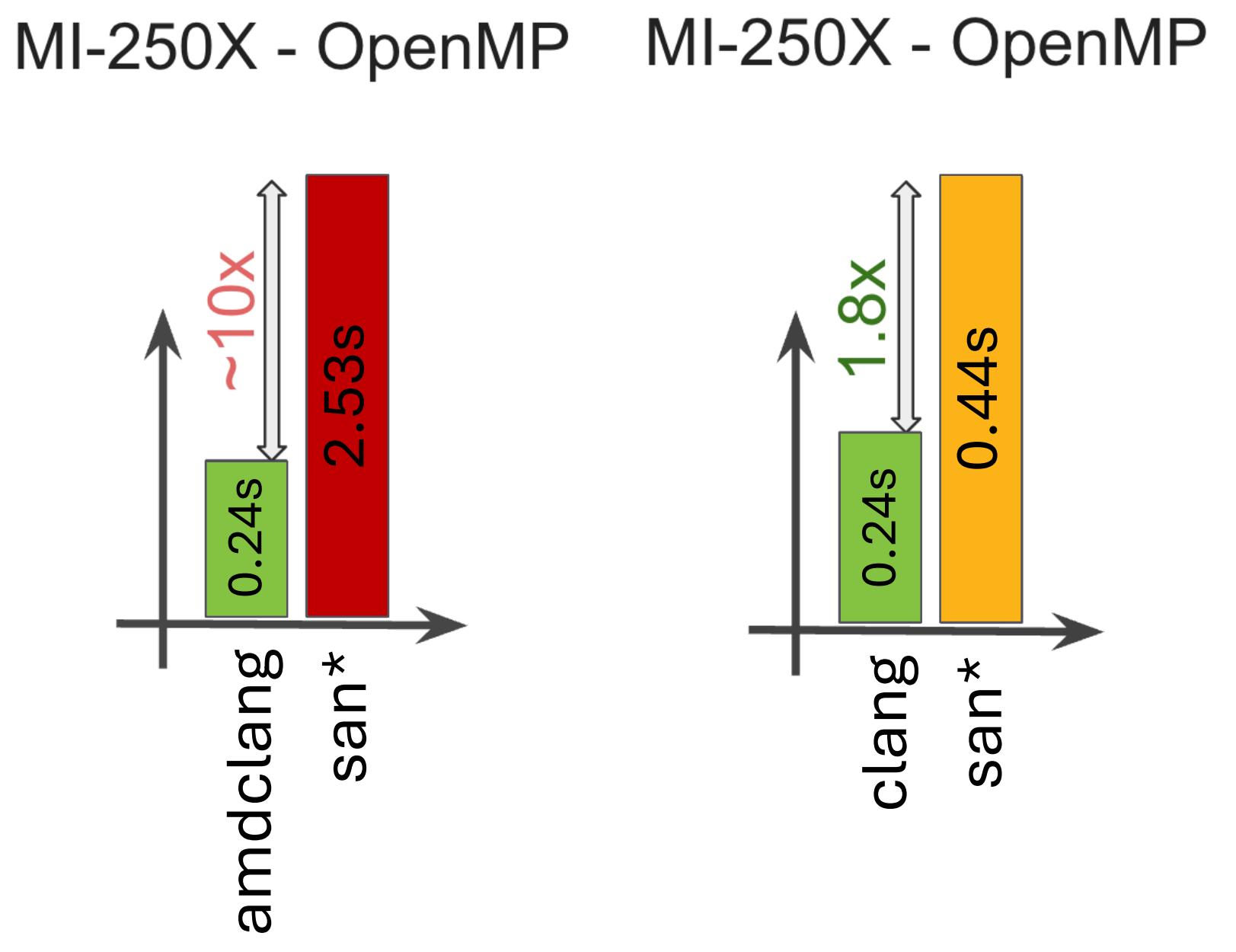
The AMD GPU link time of the LAMMPS Kokkos version with different Link-Time-Optimization (LTO) approaches. ”full-lto” is the AMD GPU default, “thin-lto” is our solution developed with AMD, “partitioned-lto” is the AMD-developed solution based on our initial proposal. **11.7x speedup** means developers can test changes without a coffee break.

LLVM/Objsan: A Heterogenous Address Sanitizer for HPC

Sanitizer for CPU and GPU code that encodes user object's information into the pointer for faster out-of-bounds checking.

Reduction of extra memory accesses by reusing (cached) bounds information across code. Preliminary results show large performance improvements on GPUs and for sparse/dense HPC codes on CPUs compared to “classic” ASAN.

Future work can utilize the ideas for performance tooling and other sanitizers, including race checkers.



Preliminary performance results comparing the AMD address sanitizer (left) and our new LLVM/Objsan (right) for the XSBench proxy application on an MI-250X AMD GPU. Note: AMD’s ASAN only sanitized heap and global memory and Objsan also checked and shared stack memory.

LLVM/InputGen: Executing Code for Testing, Tuning, and Training

Code datasets are available and used to train ML models, however, we lack ways to provide execution data at scale. InputGen makes static code executable by generating stateful inputs.

Inputs can also be recorded for parts of a program to *create standalone proxy applications* with ease.

Language	Ran (all)
C	598,600 (90%)
C++	6,302,387 (90%)
Julia	3,423,596 (92%)
Rust	5,427,870 (85%)
Swift	5,720,465 (94%)
Total	21,472,918 (90%)

Number of functions from the ComPile dataset made executable via InputGen.

Conclusions

Working directly in the (LLVM) compiler layer and collaborating with developers of abstraction layers (Kokkos/RAJA) allows us to bring tangible benefits to scientific HPC applications. Improved performance is only one pathway, and we consider the entire application lifecycle, including code development, compilation (time), testing, debugging, performance tuning, and eventually modernization.

Compiler technology offers a direct path to make scientists more efficient and effective

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-2003863

National Nuclear Security Administration