

# Benchmarking compiler optimizations on OpenMP performance

The First Workshop on LLVM Compiler and Tools for HPC – ISC2020

Giorgis Georgakoudis, Ignacio Laguna, Tom Scogland,  
Lawrence Livermore National Laboratory  
Johannes Doerfert, Argonne National Laboratory

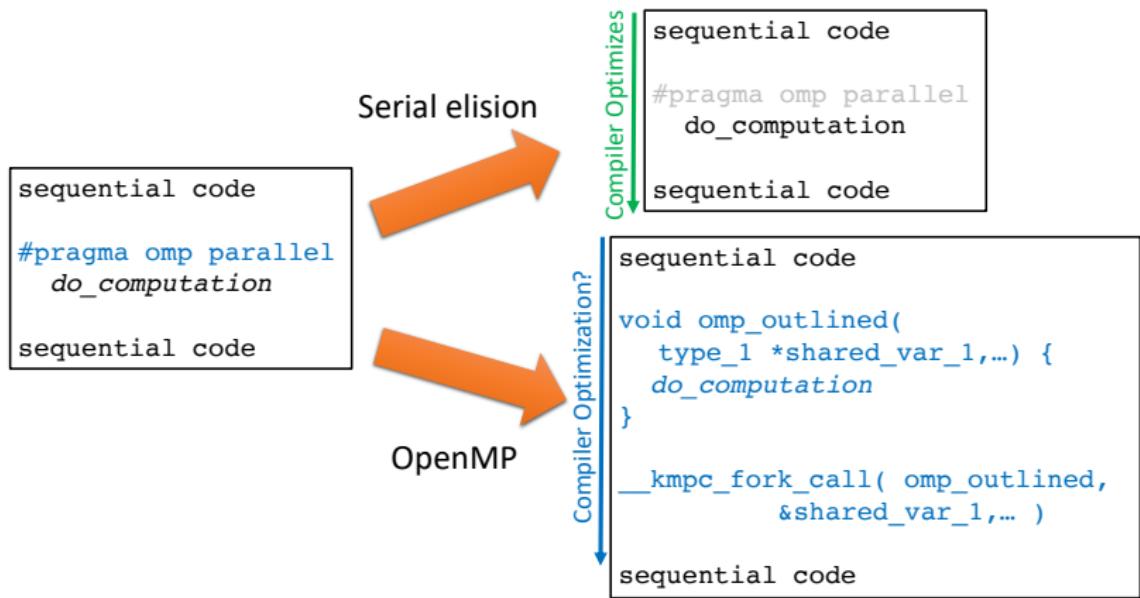


LLNL-PRES-811934

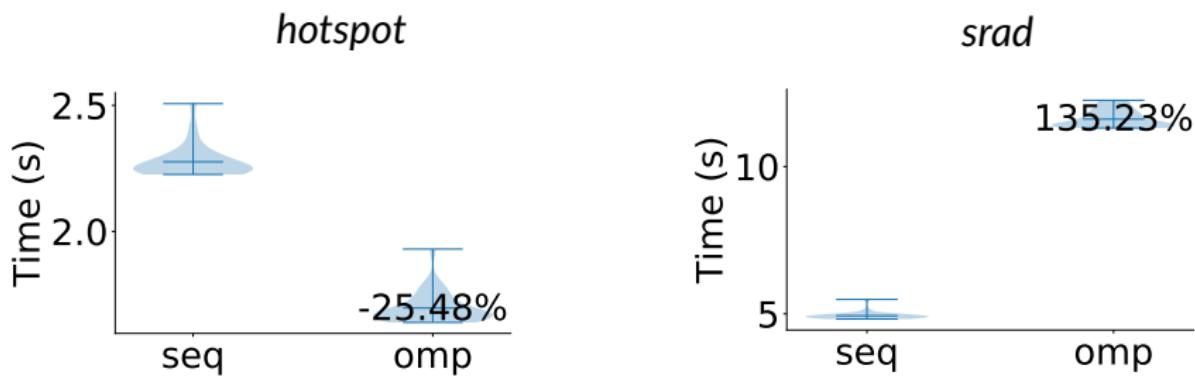
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

 Lawrence Livermore  
National Laboratory

# Problem: Clang/LLVM optimizes differently OpenMP programs versus their serial elision



# Two examples of how compiler optimization using OpenMP impacts performance



- Executing OpenMP single-threaded
- hotspot is 25.48% faster
- srad is 135.23% slower

Why?

# FAROS: A framework for analyzing OpenMP compiler optimization

## Functionality

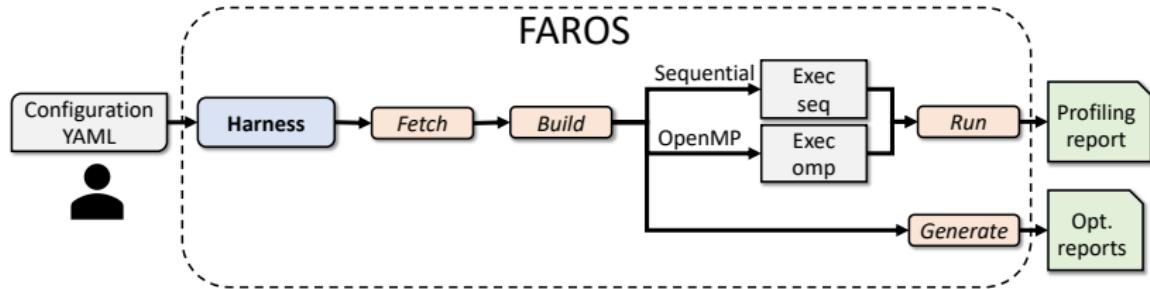
- Pinpoint compiler optimization differences
- Measure performance impact of OpenMP compilation

## Design goals

- Extensible
- Informative
- Reproducible



# The workflow of FAROS



- User provides a YAML configuration of benchmarks
- ### Output
- Profiling report for different build optimizations
  - Compiler optimizations reports based on LLVM remarks

# The interface of the harness script

```
usage: harness.py [-h] -i INPUT [-f] [-b] [-r RUN] [-g] -p PROGRAMS  
                  [PROGRAMS ...] [-s] [-d]
```

Harness for benchmarking a set of programs and compilation options

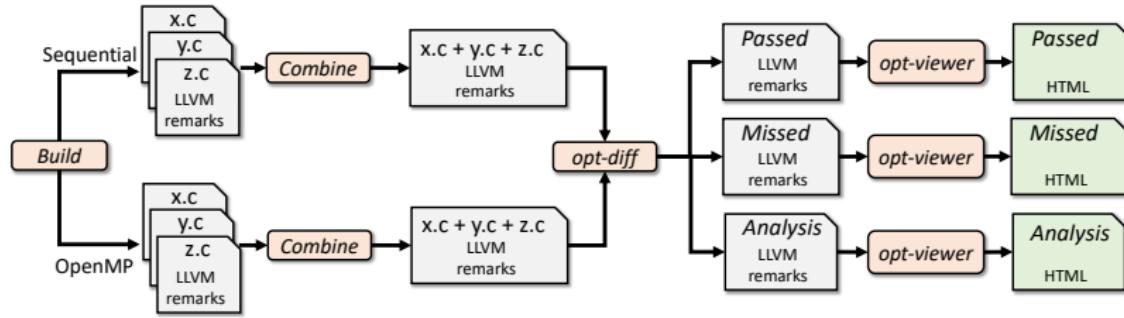
optional arguments:

- h, --help show this help message and exit
- i INPUT, --input INPUT configuration YAML input file for programs
- p PROGRAMS [PROGRAMS ...], --programs PROGRAMS [PROGRAMS ...] selected programs from the config
- f, --fetch fetch program repos (without building)
- b, --build build programs (will fetch too)
- g, --generate generate compilation reports
- r RUN, --run RUN run <repetitions>
- d, --dry-run enable dry run

# The specification of the YAML configuration

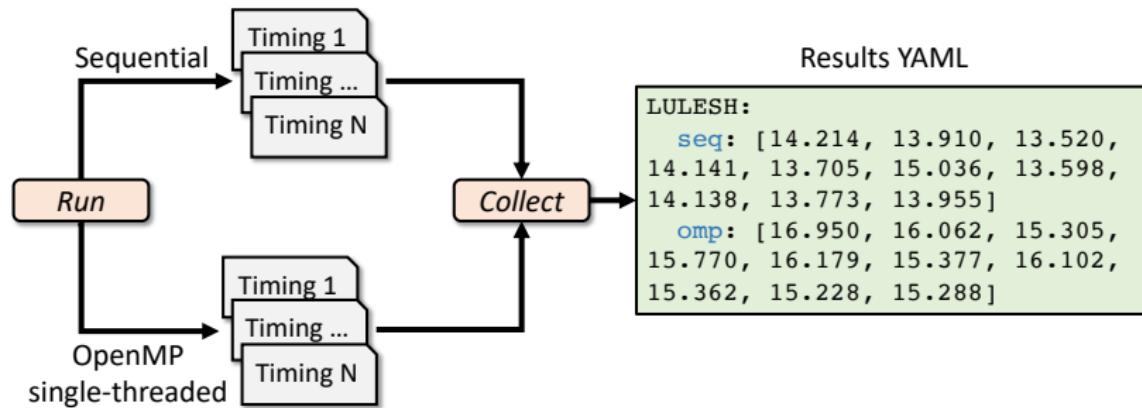
```
LULESH:
  fetch: 'git clone -q https://github.com/LLNL/LULESH.git'
  build_dir: 'LULESH'
  build: {
    seq: 'make CXX=clang++ CXXFLAGS="-g -O3 -march=native
          -I. -Wall -DUSE_MPI=0 -fsave-optimization-record
          -save-stats"',
    omp: 'make CXX=clang++
          CXXFLAGS="-g -fopenmp -O3 -march=native -I. -Wall
          -DUSE_MPI=0 -fsave-optimization-record -save-stats"'
  }
  copy: [ 'lulesh2.0' ]
  run: 'env OMP_NUM_THREADS=1 OMP_PROC_BIND=true ./lulesh2.0'
  input: '-i 500'
  measure : 'Grind time.* (\d+\.\d+) .*overall'
  clean : 'git clean -fx'
```

# FAROS generates compilation reports using LLVM remarks and opt-viewer tools



- LLVM per-source optimization remarks (Passed, Missed, Analysis)
- Combine and generate HTML reports

# FAROS runs programs and collects profiling results in YAML



- Run sequential and single-threaded OpenMP
- Measure end-to-end execution time **or** regex of measure



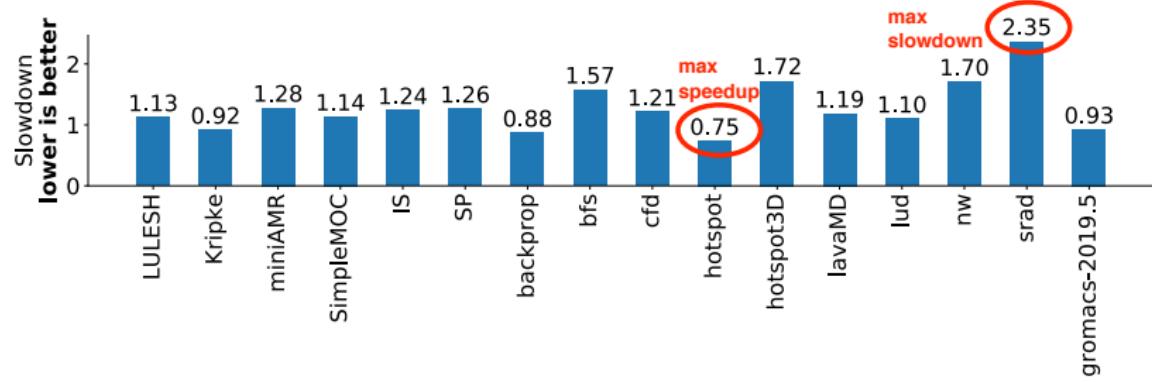
# FAROS integrates 39 OpenMP programs so far

---

<b>HPC proxy/mini/large</b>	AMG, CoMD, CoSP2, Kripke, LULESH, Quicksilver, RSbench, SimpleMOC, XSbench, miniAMR, miniAero, miniFE, hpcg, gromacs-2019.5
<b>NAS</b>	BT, CG, EP, FT, IS, LU, MG, SP
<b>Rodinia</b>	b+tree, backprop, bfs, cfd, heartwall, hotspot, hotspot3D, kmeans, lavaMD, leukocyte, lud, nn, nw, particlefilter, pathfinder, srad, streamcluster

---

# Most programs slow down and few speedup from OpenMP compilation



- Programs with slowdown  $> 1.1 \times$  or slowdown  $< 0.95 \times$
- 12/16 slow down, 4/16 speed up

# OpenMP compilation of *srad* misses vectorization

Diff of passed optimizations for sequential (-) versus OpenMP(+)

Line	Optimization	Source	Inline Context
124			
125		#ifdef OPEN	
126		omp_set_num_threads(nthreads);	
127	+licm +inline +licm	#pragma omp parallel for shared(J, dN, dS, d sinking <u>getelementptr</u> .omp_outlined. debug_ inlined into .omp_ou... hoisting <u>getelementptr</u>	main .omp_outl... main
128		#endif	
129	-licm -licm -licm	for (int i = 0 ; i < rows ; i++) { hoisting <u>icmp</u> hoisting <u>zext</u> sinking <u>zext</u>	main main main
130	-loop-vectorize -licm -licm	for (int j = 0; j < cols; j++) { vectorized loop (vectorization width: 8, interleaved ... sinking <u>zext</u> hoisting <u>zext</u>	main main main



# Analysis pinpoints the problem to fix for srad

Diff of analysis output for sequential (-) versus OpenMP (+)

130

+loop-vectorize

```
for (int j = 0; j < cols; j++) {
```

loop not vectorized: cannot identify array bounds

.omp\_outl...

- Variable cols is shared, thus a pointer
  - Loop bound analysis fails
  - Pointer analysis cannot determine alias-free

SOLVED

- #pragma omp simd or
- IPO value propagation using experimental Attributor<sup>1</sup>

---

<sup>1</sup>Doerfert, Johannes, and Hal Finkel. "Compiler optimizations for OpenMP." In International Workshop on OpenMP, pp. 113-127. Springer, Cham, 2018.



Lawrence Livermore National Laboratory

LLNL-PRES-811934



**OpenMP compilation of *hotspot* enables complete loop unrolling**

Diff of passed optimizations sequential (-) versus OpenMP (+)

	#pragma omp simd	
137	+loop-unroll completely unrolled loop with 2 iterations	.omp_ou...
138	+loop-vectorize vectorized loop (vectorization width: 8, interleaved count: 1)	16 .omp_ou...
	for ( c = c_start; c < c_start + BLOCK_SIZE_C; ++c )	
	hoisting icmp	
	sinking trunc	
	vectorized loop (vectorization width: 8, interleaved co...	
	-licm	single_it...
	-licm	single_it...
	-loop-vectorize	single_it...
	+licm	single_it...
	-licm	single_it...
139	/* Update Temperatures */	
140	result[r*col+c] =temp[r*col+c]+	
	hoisting load	
	hoisting load	.omp_ou...
141	( Cap_1 * (power[r*col+c] +	.omp_ou...
	hoisting load	
	( temp[(r+1)*col+c] + temp[(r-1)*col+c] - 2	.omp_ou...
142	( temp[r*col+c+1] + temp[r*col+c-1] - 2.f*t	
143	(amb_temp - temp[r*col+c]) * Rz_1));	
144		
145	}	

- OpenMP canonical loop constraint helps loop analysis
  - simd dependence-free pointers  $\Rightarrow$  avoids runtime checks

# Refactoring brings sequential *hotspot* on par with OpenMP

**SOLVED**

- Refactor code to simplify loop analysis

```
for(cc = 0; cc < BLOCK_SIZE_C; cc++)  
    c = c_start + cc;  
    ...
```

- Declare pointers as restrict



# Conclusion

## Summary

- OpenMP compilation has complex effects on compiler optimization
- FAROS helps to benchmark and analyze

## Future work

- Improve compilation reporting (LLVM IR, OpenMP remarks)
- Provide recommendations to maximize optimization
- Integrate more profiling information (TAU, HPCToolkit)

# Thank you! Questions?

## Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.