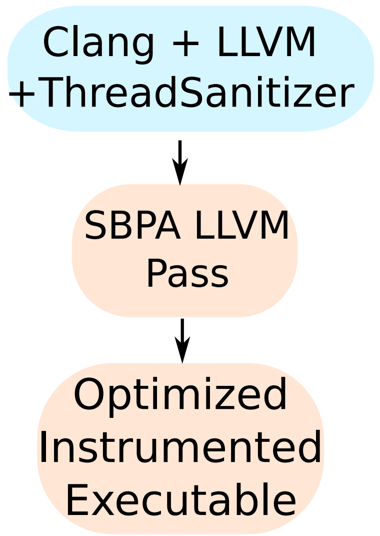
Section Based Program Analysis to Reduce Overhead of Detecting Unsynchronized Thread Communication

Madan Das, Gabriel Southern, Jose Renau
Dept. of Computer Engineering
University of California, Santa Cruz

Features:

- ➔ Static analysis techniques
- ➔ Eliminates instrumentation at compile time
- ➔ Useful for race detection, deterministic executon engines and STMs
- ➔ Works with ThreadSanitizer
- ➔ Works for multi-threaded C and C++ programs
- ➔ Precise section based alias analysis
- ➔ Augmented with verifiable directives
- ➔Validated with parsec, splash and phoenix suites
- ➔Implemented as LLVM pass
- ➔Holistic solution to detect data race issues
- ➔Open source

http://masc.soe.ucsc.edu/sbpa



Clang + LLVM +ThreadSanitizer

SBPA LLVM Pass

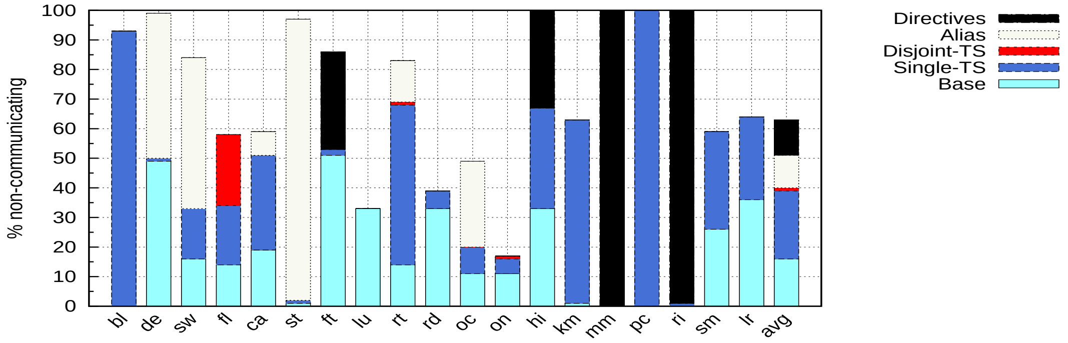
Optimized Instrumented Executable

SBPA is effective!

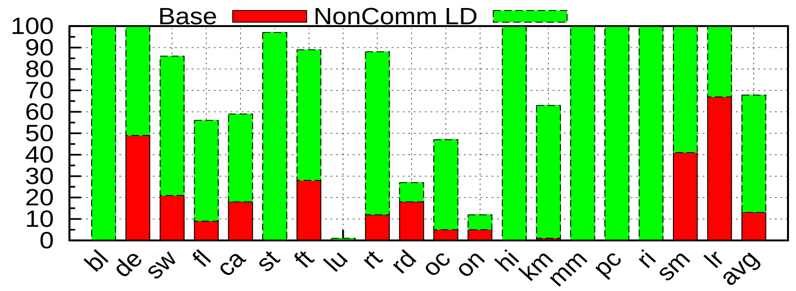
➔ Eliminated 63% of total memory instrumentations

SBPA is accurate

➔ Validated with PARSEC, SPLASH and Phoenix suites



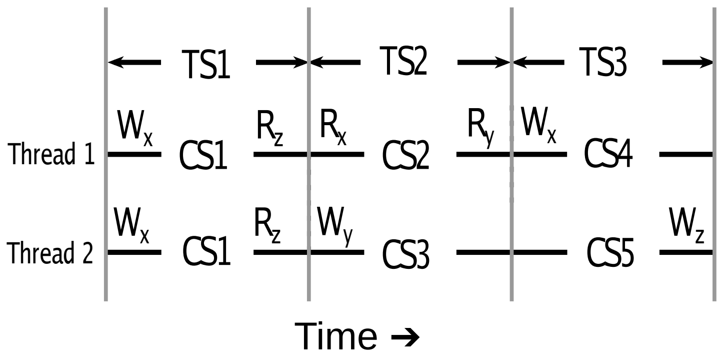
Cumulative effect of applying SBPA, alias improvements, and use of some user directives, including MTROM (multi-threaded read only memory) on all memory accesses



68% of loads are proven non-communicating. Baseline is Coredet.

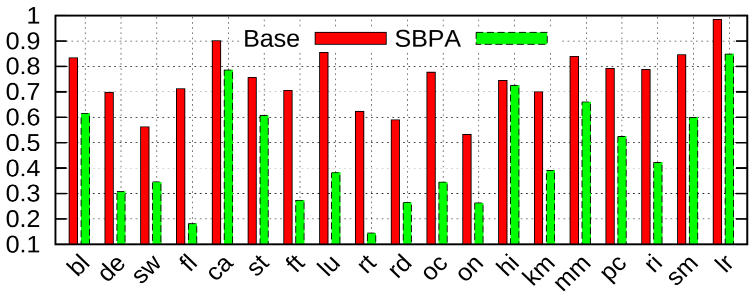
Programs have phases

- ➔ Identifying phases in parallel code can improve precision of alias analysis
- ➔ Most data accesses in parallel code are non-communicating (non-racy and independent in same phase)



Section Identification in SBPA

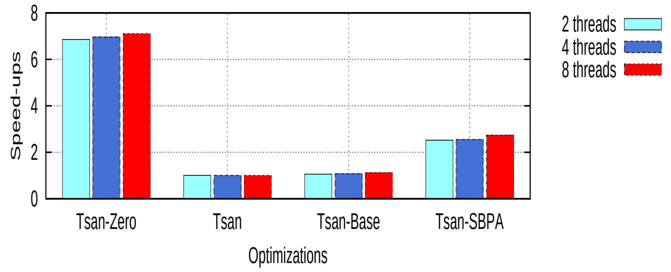
```
1. Build reduced inter-procedural CFG
2. Find multithreaded code sections (MTCS)
      enclosed by create/join
3 foreach MTCS section ts:
.1 let b = beginning of ts
.2 let e = end of ts
.3 while b != e:
 .1 C = reachable barrier nodes starting from b.
.2 if C has a single node
  .1 Code from b to  C is a new thread section
 .2 b = C
  .3 else exit search
```



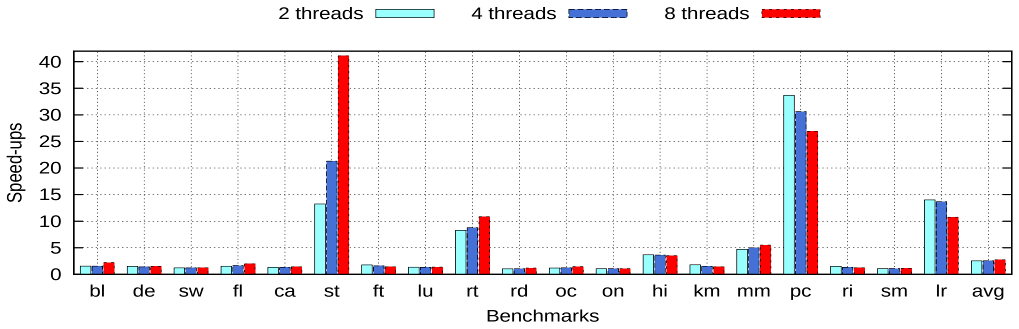
Compilation time normalized to Baseline

Integration with ThreadSanitizer

- ➔ ThreadSanitizer slows down 12.5 times
- ➔ SBPA speeds it up 2.74 times
- ➔ Still detects the same races



Geometric means of speed-ups in different modes



Per benchmark speed-ups with 2, 4 and 8 threads

References:

[1] Reducing Logging Overhead for deterministic Execution,, Madan Das, Gabriel Souhern and Jose Renau, 4th Workshop on Determinism and Correctness in Parallel Programming, 2013.

[2] Dynamic race detection with llvm compiler, Konstantin Serebryany, Alexander Potapenko, Timur Iskhodzhanov, and Dmitriy Vyukov, in Runtime Verification. Springer, 2012, pp. 110–114

[3] CoreDet: a compiler and runtime system for deterministic multithreaded execution, T. Bergan, O. Anderson, J. Devieti, L. Ceze and D. Grossman, 15th edition of ASPLOS on Architectural support for programming languages and operating systems