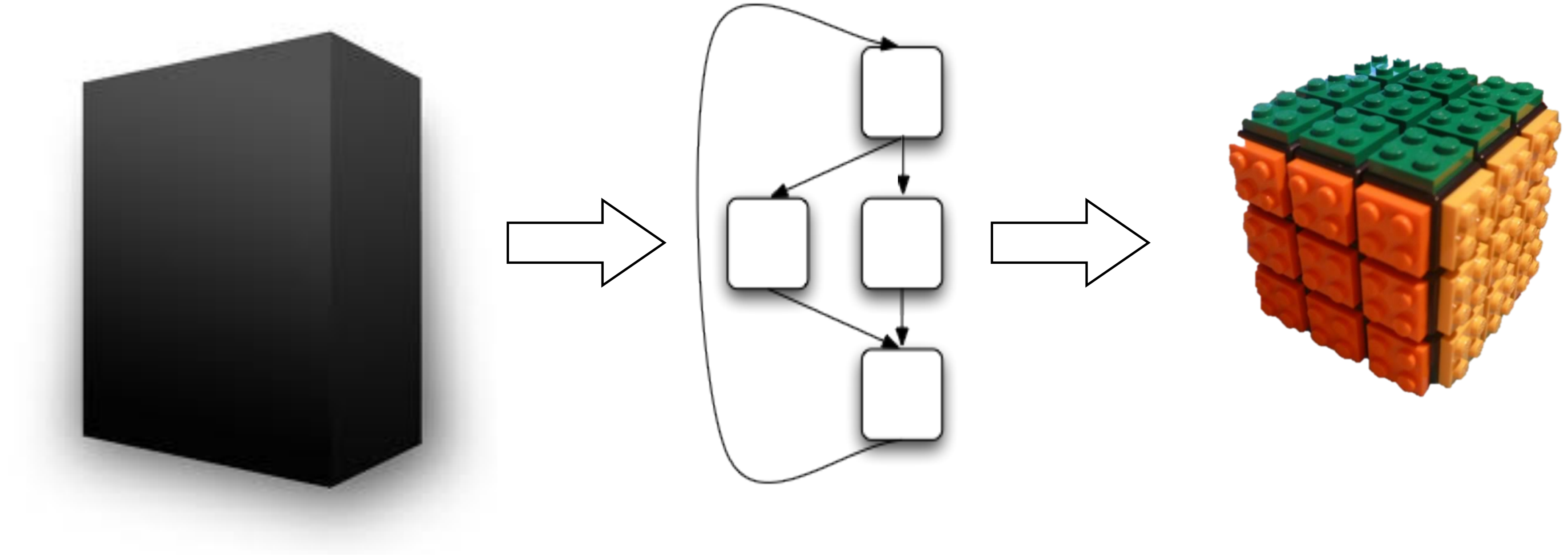
Recent and Upcoming Advances in the Dyninst Toolkits

Sasha Nicolas and Xiaozhu Meng

Paradyn Project

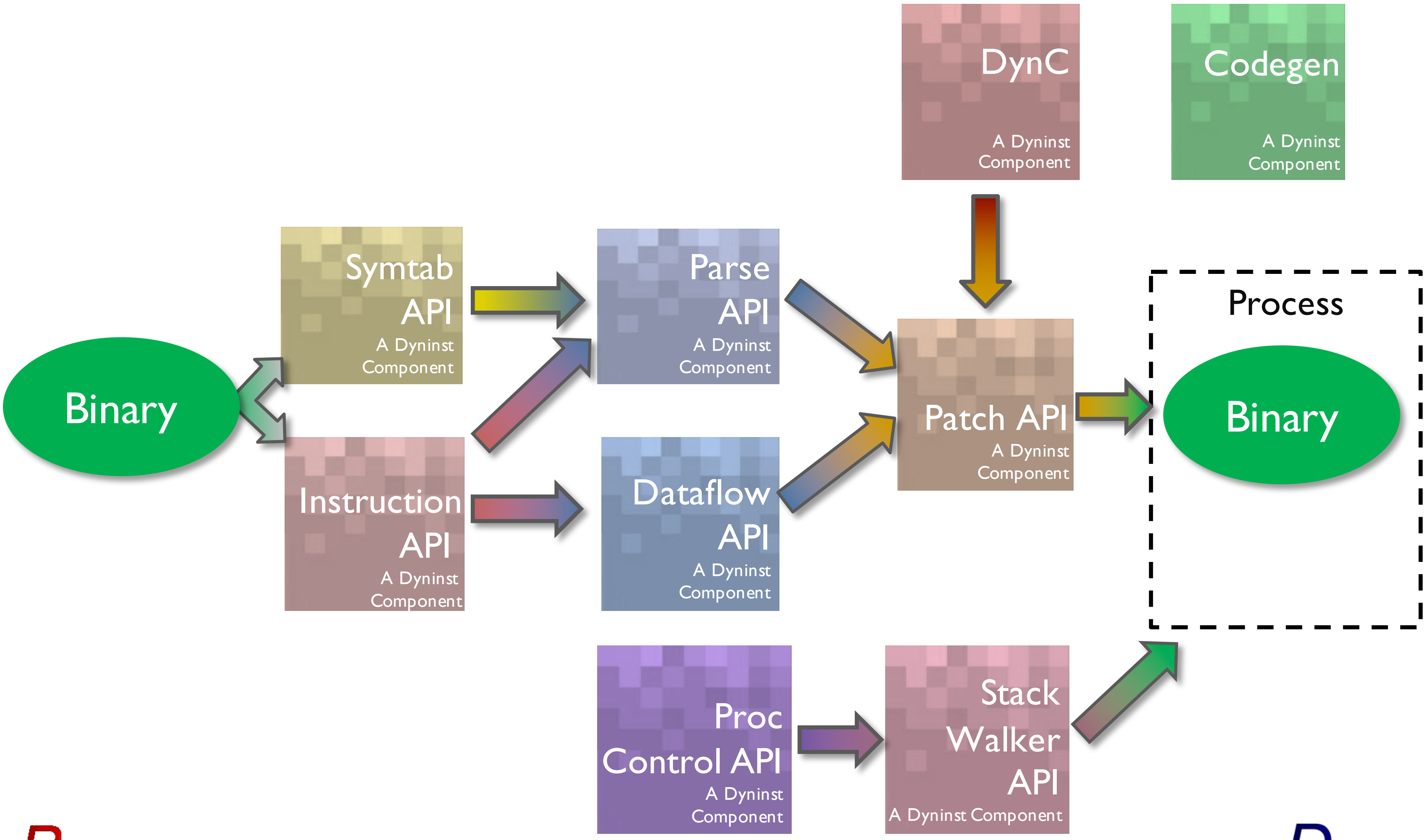
Petascale Tools Workshop
Solitude, UT
July 8-12, 2018

A Brief Introduction to Dyninst



Dyninst: a tool for static and dynamic binary instrumentation and modification

Dyninst and the Components



New Dyninst developments since 09/2017

Working towards full Dyninst support to new architectures

- ARMv8 (64 bit)
- ARM 32 bit and Thumb 16 and 32 bit
- PowerPC 8 & 9

Improve analysis speed by parallelization

- Code parsing
- DWARF parsing

New Dyninst developments since 09/2017

Jump table analysis is integrated and tested cross-platform, including ARM, PowerPC, and x86.

Limited CUDA support

Read-only queries of SymtabAPI

Dyninst github master builds with the latest version of Spack out of the box

Notable fixes since 09/2017

Libdw port

ARM StackwalkAPI

- Signal handlers
- Alternate stacks

Cross-architecture binary analysis:

endianness for parsing try/catch blocks

Lots of bug fixes, including in the test suite.

ARMv8 (64 bit) Porting

Goal:

port Dyninst capabilities to the architecture ARMv8 (aarch64).

Status: in progress.

ARMv8 (64 bit) Porting

SymtabAPI – fully working.

InstructionAPI – working with small bugs to be fixed.

ParseAPI – fully woking.

DataflowAPI – fully working, liveness analysis fixed.

PatchAPI – fully working.

ProcControlAPI – fully working.

StackwalkerAPI – fully working.

DyninstAPI – development in progress. Current focus.

Function relocation is complete.

IRPC for malloc fixed.

ARMv8 (64 bit) - Function relocation

Goal:

Relocate (copy and move, modifying address space) all functions in a binary without changing the behavior of the program.

Why is it important?

Basis for instrumentation in Dyninst. This consists of copying existing code to a new address and adjusting branching instructions.

Status: complete.

ARMv8 (64 bit) - Code generation

Goal: Instrument programs on ARMv8 (64 bit).

Status: in progress.

Implemented: inserting calls, if-else statements, relational and arithmetic operations, read/write variable, non-recursive/recursive base tramp.

Next: non-void return functions, arbitrary points, replace function, local variables, array variables, unary operators, struct elements, type compatibility, user defined fields, call site parameter referencing, instrument loops, monitor call sites.

(Test 1 to 12 in test suite are passing)

ARM 32-bit port

Led by University of Maryland

ARM 32 can switch between ARM and Thumb instructions

- Distinguish mixed ARM and Thumb instructions
- Not offered by other tools

Dyninst component status:

- SymtabAPI, InstructionAPI, and ParseAPI are working well for internal tests
- Binary rewriting targeted for mid-September

Power 8 and 9

SymtabAPI, ProccontrolAPI, StackwalkAPI, ParseAPI, and DataflowAPI are working well

InstructionAPI misses some newly added vector instructions

DyninstAPI mostly works, but the following functionality needs fixing:

- Creating PLT for binary rewriting
- Function wrapping

Improve the speed of binary analysis

Dyninst and other parsers analyze binaries in serial

The pynamic benchmark from LLNL generates a1.5GB binary, containing 560MB code and 1.1 million functions

- Serial code parsing takes 210 seconds
- Serial DWARF parsing takes180 seconds

Take advantage of extra cores

- Parallel code parsing
- Parallel DWARF parsing

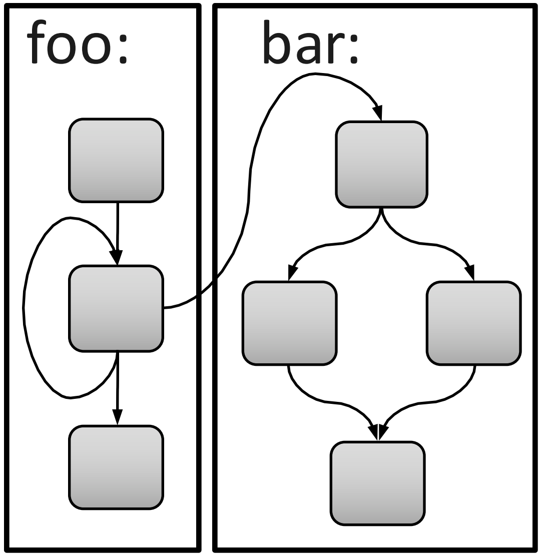
Determine the granularity of parallelism

- Parallel parsing between functions and serial parsing within a function
- Parallel parsing within functions (too complex)

Main challenge of parallel code parsing

Our code parser (or any other parser) is not designed with parallelism in mind

- Excessive global states



Parallel instruction decoding

global states in instruction decoder

Parallel creation of edges, blocks, functions

global index for blocks and functions

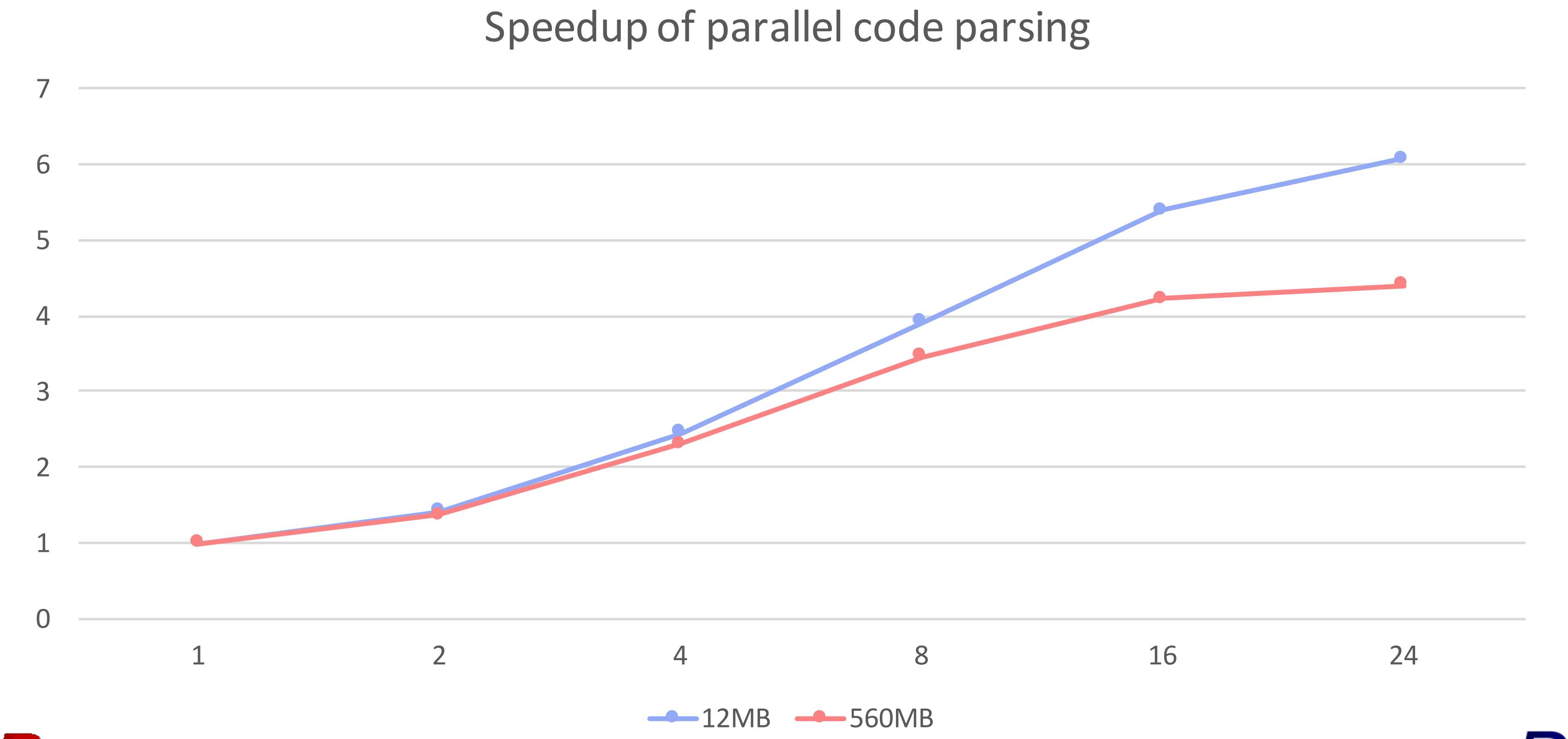
- Dispersed reads and writes to global states

Road to high performance parallel code parsing

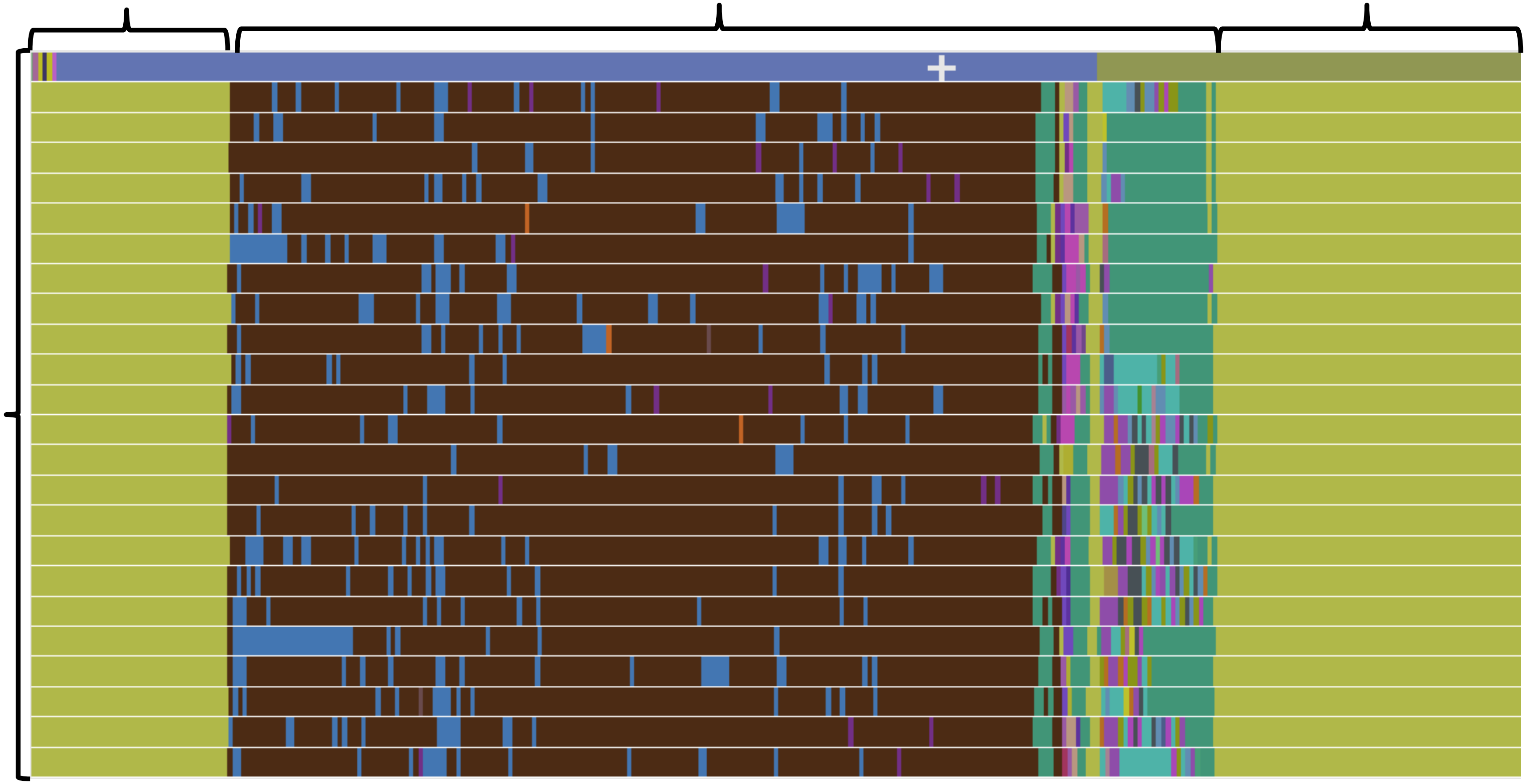
- Identify shared data structures and critical sections
  - Create functions and basic blocks if not already exist
  - Instruction objects are passed by value
- Reduce serial accesses
  - Use concurrent hash maps
- Identify redundant operations
  - Remove unnecessary updates of block ranges
- Separate concurrent operations from serial operations
  - Split into phases

Parallel code parsing results

Measure speedup for binaries in different sizes (the size of the .text secion)



Parallel CFG parsing under HPCTraceViewer



Serial parsing initialization

Parallel function parsing

Serial function finalization

24 threads

Improve the speed of binary analysis

Dyninst and other parsers analyze binaries in serial

The pynamic benchmark from LLNL generates a1.5GB binary, containing 560MB code and 1.1 million functions

- Serial code parsing takes 210 seconds
- Serial DWARF parsing takes180 seconds

Take advantage of extra cores

- Parallel code parsing takes 48 seconds
- Parallel DWARF parsing

Parallel DWARF parsing

Serial DWARF parsing is the performance bottleneck

- About 80% of the total analysis time

The tree structure of DWARF is parsed recursively, leading to natural parallelism

However, DWARF parsing libraries such as libdwarf and libdw do not fully support parallel queries

Dyninst switched from libdwarf to libdw as libdwarf is not thread-safe at all

Issues of using libdw for parallel DWARF parsing

Libdw is not thread-safe

- Internal memory allocation is not thread-safe
- Internal hash map and glibc binary search trees are not thread-safe

Ideally, we would like to fix these inside libdw

- Switch to standard malloc
- Switch to concurrent hash map and concurrent search tree

Will we create performance issues for other users who only use libdw in serial?

Dyninst 10.0

Parallel code parsing

Power 8 & 9 support

Partial ARM instrumentation support

Completed jump table analysis

Switch to depend on libdw

API breaking changes:

- InstructionDecoder::decode() previously returns Instruction::Ptr, will return Instruction in 10.0
- Operation.h is renamed to Operation_impl.h

Dyninst collaborators

Thanks the contribution from

John Mellor-Crummey and Mark Krentel (Rice)

Josh Stone (Ret Hat)

Jim Galarowicz (Open|SpeedShop)

Bob Moench (Cray)

Dyninst users

Dyninst is used by

- HPCToolkit (Rice)
- SystemTap (Red Hat)
- Open|SpeedShop
- ATP (Cray)
- TAU (U of Oregon, LANL, and Research Center Julich)
- VUSec (Vrjie Universiteit Amsterdam)

Software info

Main project page:
https://github.com/dyninst/dyninst

- Issue tracker
- Releases and manuals

LGPL

Contributions welcome