Creating Concise and Efficient Dynamic Analyses with ALDA

Xiang Cheng
Georgia Institute of Technology

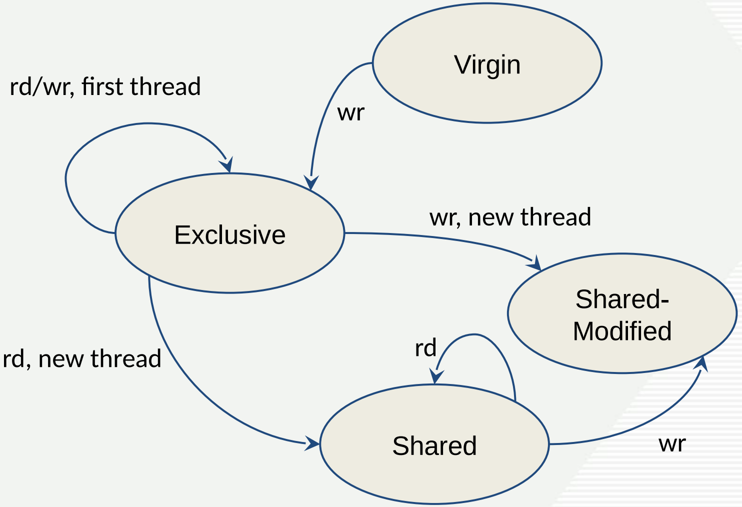
David Devecsery*
Meta Platform. Inc

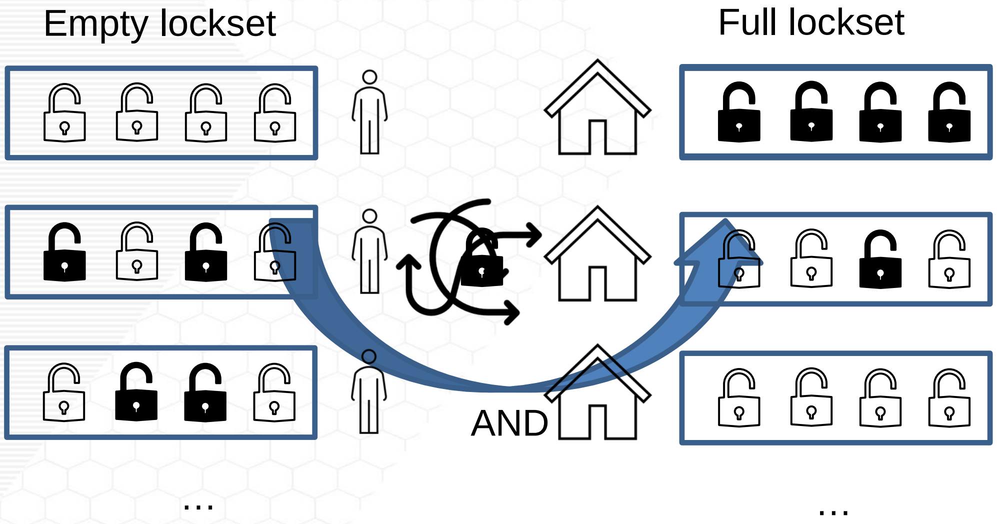
CREATING THE NEXT®

*: Work done while at Georgia Institute of Technology

Motivating Example – Eraser analysis

1. Eraser is a lockset based data race detector[1]
2. The algorithm can be represented by a state machine with 4 states
3. The analysis tracks metadata for each thread/memory address/lock …





---

[1]Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and ThomasAnderson. Eraser: A dynamic data race detector for multithreaded programs.ACM Transactions on Computer Systems (TOCS), 15(4):391–411, 1997

Motivating Example – Eraser analysis

// static transformation table

```
static u_char qtable[]{0, 1, 3, 3};
static u_char wtable[]{1, 3, 3, 3};
static u_char rtable[]{0, 2, 2, 3};
```

// write access

```
if (NEW_THREAD_ACCESS) {
    addr.status = wtable[addr.statu
} else {
    addr.status = qtable[addr.statu
}
addr.lockset &= thread.wlockset;
```

//read access

```
if (NEW_THREAD_ACCESS) {
    addr.status = rtable[addr.status];
}
addr.lockset &= thread.rlockset;
```

Analysis kernel is simple
(~20 line of code)

// Metadata type

```
typedef struct ThreadMeta{
    int tid;
    Set<LOCK> rlockset;
    Set<LOCK> wlockSet;
} ThreadMeta;
         truct AddrMeta{
         tatus;
         nt> threadId;
         OCK> lockset
```

// Metadata declaration

```
Map<Thread, ThreadMeta> thread_meta_map;
Map<Address, AddrMeta>  address_meta_map;
```

The building process ...

Complex data structures

Lock optimization

on ...

Parameter

Micro optimization

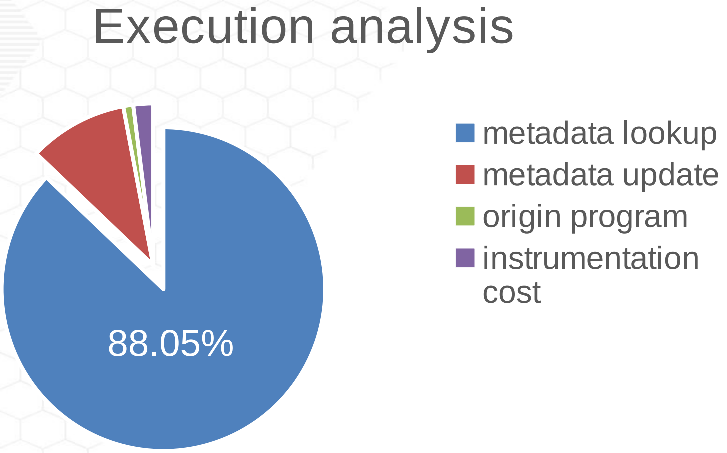
New analysis kernel

~200x

Efficient Implementation (~600 LOC)

Challenge – Metadata optimization

- Compared with simple kernel, the metadata access is the bottleneck
  - Memory access are frequent => dominating performance
  - Metadata is analysis dependent => needs to repeat optimization for every analysis
- Tradition compilers are bad at this
  - Reasoning memory access is hard, e.g. the aliasing is NP hard
- => Can we automate this optimization process?

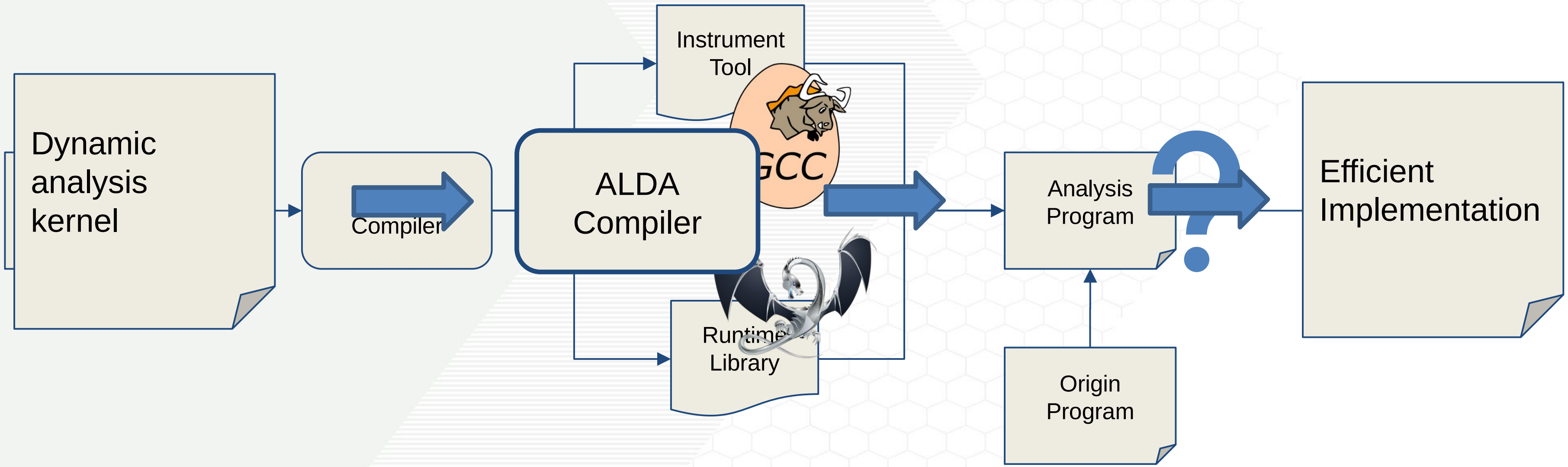


ALDA's Solution & Insights

- Solution
  - Separate the logic and implementation of dynamic analyses, and let a compiler to automatically optimize the input analyses
- Observations
  - Most dynamic analyses kernels are simple algorithms
  - Most optimizations are related to memory access pattern / layout
  - Many dynamic analysis kernels can be represented naturally without loops or indirect memory access, removing the need for memory indirection in language description

ALDA's Key idea and workflow

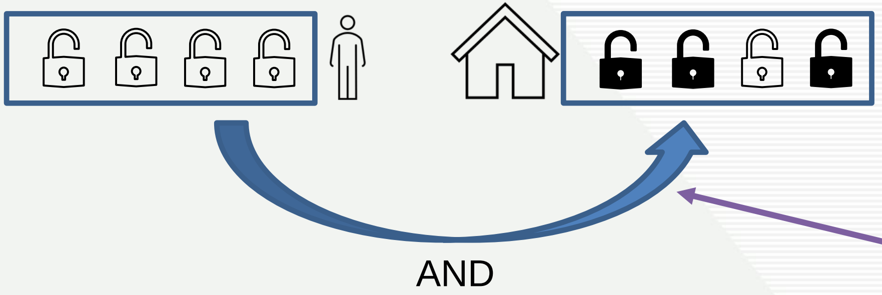
- By restricting language syntax, the compiler can better reason about and optimize metadata access in dynamic analyses.

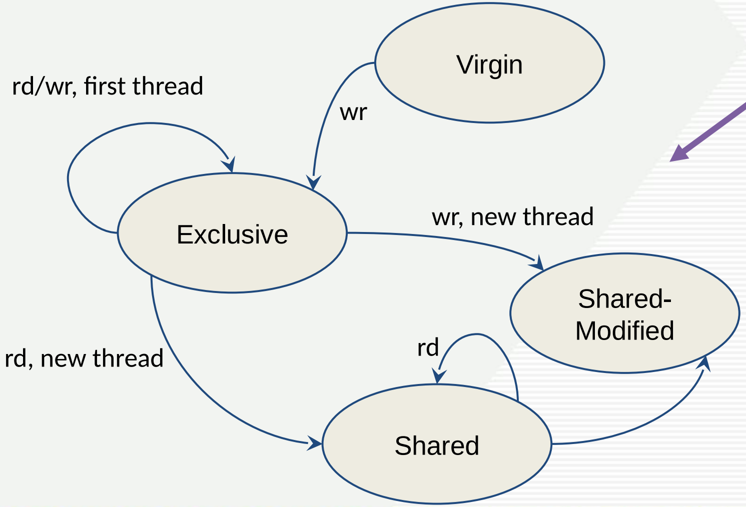


Content of today

- Motivation
- Key insights & idea
- **ALDA Language Design**
- ALDA Optimizations
- Evaluation
- Conclusion

Design ALDA Language





What pieces make up a dynamic analysis?

- ○ What metadata does the analysis need
- ○ What's the logic for analysis
- ○ When to apply such logic

ALDA Syntax - How to specify metadata ?

```
// Metadata of Eraser algorithm in ALDA
address := pointer : sync
tid := threadid : 4
lid := lockid : 256
status := int8
thread2WLock = universe::map(tid, set(lid))
thread2Lock = universe::map(tid, set(lid))
addr2Lock = universe::map(address, universe::set(lid))
addr2Thread = universe::map(address, set(tid))
addr2Status = universe::map(address, status)
```

- Declare the types of data we need to track
- Metadata associations
  - map / set - high level abstractions
  - let compiler to choose the data structure

ALDA Syntax - What’s the logic for analysis

```
// When thread read memory address
onLoad(address addr, tid t) {
    if(!addr2Thread[addr].find(t) && addr2Status[addr] != VIRGIN){
        if(addr2Status[addr] == EXCLUSIVE)
            { addr2Status[addr] = SHARED; }
        addr2Thread[addr].add(t);
    }
    if(addr2Status[addr] > EXCLUSIVE){
        addr2Lock[addr] = addr2Lock[addr] & thread2Lock[t];
    }
}
```

- C like syntax inside function body
  - Except loop statements
  - Disallow pointer/reference types

ALDA Syntax - Where to apply such logic

```
// Instrumentation example of load operation
insert after LoadInst call onLoad($1, $t)
```

- Indicates the location to instrument function
- Can be either a function call or a specific instruction
  - malloc/pthread_create
  - load/cast/xor low level operations

Content of today

- Motivation
- Key insights & idea
- ALDA Language Design
- **ALDA Optimizations**
- Evaluation
- Conclusion

How to optimize the code? - Metadata Coalescing

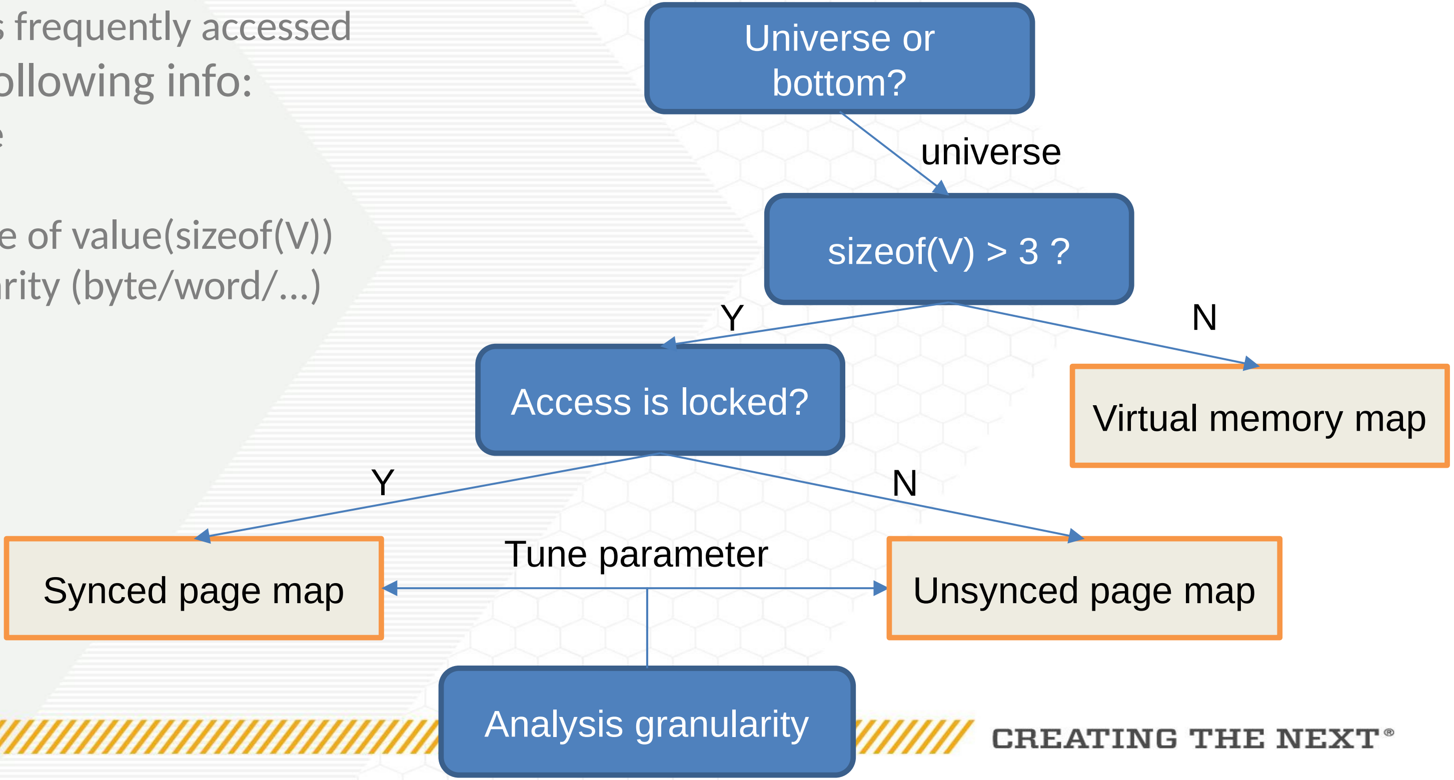
- Due to lack of indirection, memory access patterns are simple
- Our compiler, ALDAcc, performs static analysis to reason the types and memory access patterns
- With the analyses, the compiler relayout the metadata to coalesce them

```
// When thread read memory address
onLoad(address addr, tid t) {
    if(!addr2Thread[addr].find(t) && addr2Status[addr] !=
VIRGIN){
        if(addr2Status[addr] == EXCLUSIVE)
              { addr2Status[addr] = SHARED; }
        addr2Thread[addr].add(t);
    }
    if(addr2Status[addr] > EXCLUSIVE){
        addr2Lock[addr] = addr2Lock[addr] & thread2Lock[t];
    }
}
```

```
// Metadata type
typedef struct AddrMeta{
    int addr2Status;
    Set<int>
addr2Thread;
    Set<LOCK> addr2Lock
} ;
```

How to optimize the code? - Data structure selection

- Pagetable map, virtual-memory based map are widely used for pointer key types
    - ○ the K’s domain size is big: e.g. 2^48 for x86_64
    - ○ data structure is frequently accessed
- Compiler gathers following info:
    - ○ Map initial state
    - ○ Access is locked
    - ○ The memory size of value(sizeof(V))
    - ○ Analysis granularity (byte/word/...)



Content of today

- Motivation
- Key insights & idea
- ALDA Language Design
- ALDA Optimizations
- **Evaluation**
- Conclusion

CSE/set selection/tls/inline/ …

Evaluation*

- Performace
    - Can ALDAcc generate code comparable to hand-tuned implementations?
- Generality
    - Can ALDA represent common dynamic analyses?
- Application
    - Can ALDA be used to build analyses that are otherwise impractical?

Artifacts Evaluated acm Functional V1.1

Artifacts Available acm V1.1

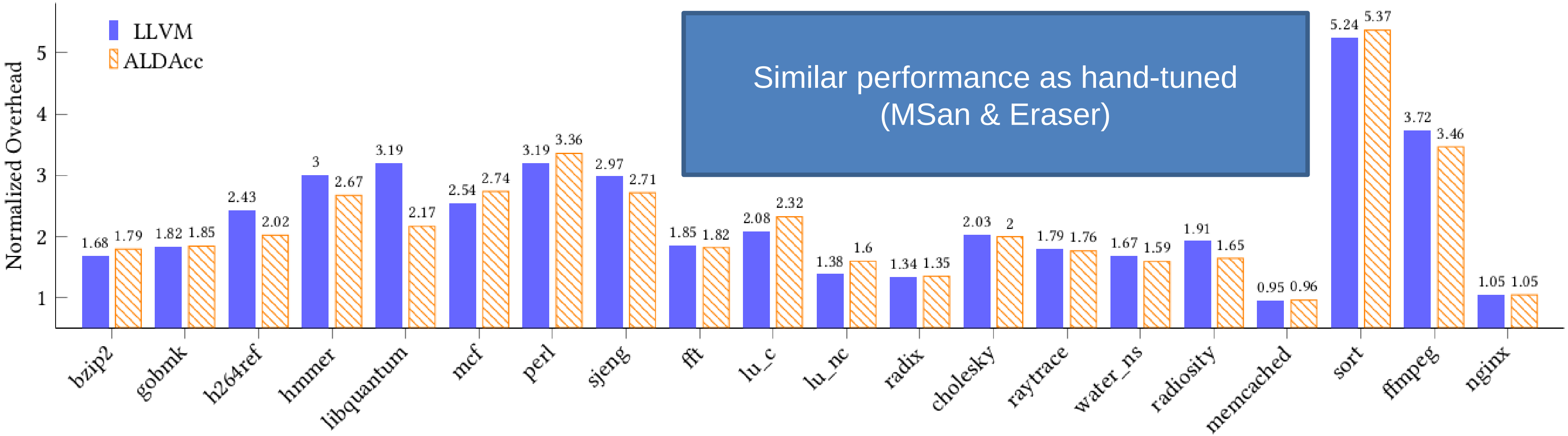
Results Reproduced acm V1.1

---

*: ALDA is publicly available at https://doi.org/10.5281/zenodo.5748338

Performance – compare with hand-tuned implementations

- We use ALDA to reproduce LLVM Memory Sanitizer, Eraser and compare with the hand-tuned implementations.
- We run both programs in SPECInt / SPLASH & 4 real-world applications
- ALDAcc can generate comparable code with hand-tuned implementation



Similar performance as hand-tuned
(MSan & Eraser)

Generality – 6 types of dynamic analyses

We try to use ALDA to implement following dynamic analysis:

Name	LOC	Name	LOC
Eraser	70	MSan	192
UseAfterFree	35	StrictAliasCheck	12
FastTrack	69	TaintTracking	33

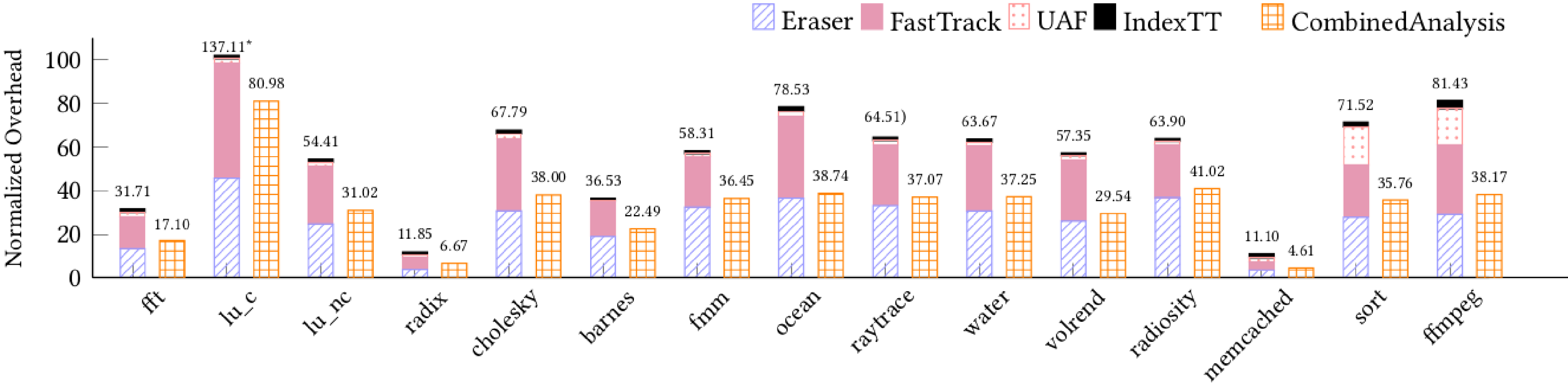
Save >80% line of code

1. Hand-written Eraser takes 600+ LOC
2. LLVM MSan takes at least 8146 LOC
3. ALDAcc's MSan requires a common libc handler take ~1100 LOC

Application – Combined analysis

- Different analyses can’t be easily combined (TSan/MSan can’t run at the same time)
- We use ALDA to easily combine different analysis algorithms together(Eraser/FastTrack/UAF/TaintTracking)

Save 44.8% execution time



*: Bar compressed

Application – SSLSan & ZLibSan

- API misuse widely exists in open-source projects
  - API specific => Common sanitizers can’t catch them
  - Each library has different usage => Requires build for each library
- We use ALDA to build SSLSan and ZLibSan and run them for memcached/nginx/ffmpeg
  - Validate 4 bugs/misuses in three applications

Summary

- We present ALDA, a domain specific language and ALDAcc, that can convert the ALDA program into highly optimized executables
- We describe several static optimizations for ALDA analyses and show their efficiency
- We applied ALDAcc into real world example: library sanitizer / combined analyses
- We look forward to applying ALDA to new analyses and languages

Thanks for listening

Q & A

CREATING THE NEXT®

Related work

- Instrumentation frameworks like LLVM, Intel Pin are basis for ALDA to apply analysis logic into origin program.
- Dynamic analyses framework:
    - Some are focusing on providing utilities to develop dynamic analyses, like Valgrind, they failed to perform metadata access optimization and relayout as ALDA does
    - Some are based on well typed languages like JavaMOP, which avoids the metadata lookup problem