Memoro
Scaling an LLVM-Based Heap Profiler

Thierry Treyer
Performance & Capacity Intern

Mark Santaniello
Performance & Capacity Engineer

James Larus
EPFL IC School Dean

EPFL
vector<BigT> getValues(map<Id, BigT>& largeMap, vector<Id>& keys) {

    vector<BigT> values;
    values.reserve(largeMap.size());

    for (const auto& key : keys)
        values.emplace_back(largeMap[key]);

    return values;
}
40 GiB of DRAM wasted per server
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

    vector<BigT> values;
    values.reserve(largeMap.size());

    for (const auto& key : keys) {
        values.emplace_back(largeMap[key]);
    }

    return values;
}
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

    vector<BigT> values;
    values.reserve(largeMap.size());

    for (const auto& key: keys)
        values.emplace_back(largeMap[key]);

    return values;
}
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

    vector<BigT> values;
    values.reserve(keys.size());

    for (const auto& key: keys)
        values.emplace_back(largeMap[key]);

    return values;
}
LLVM-Based Profiler
LLVM-Based Profiler

Manipulate the IR
LLVM-Based Profiler

Infrastructure

Manipulate the IR
LLVM-Based Profiler

Collecting and Displaying data

Infrastructure

Manipulate the IR
Memoro + Facebook

Run-Time Overhead

Visualizer

Open Challenges
Overview

Source Code  Compile  Run  Analyze
Overview

- Source Code
- Compile
- Run
- Analyze

- No modification
Overview

INSTRUMENTATION PASS
(LLVM)

Source Code
- No modification

Compile
- Instrument loads/stores
- Instrument intrinsics
- Collect types

Run

Analyze
Overview

Source Code
- No modification

Compile
- Instrument loads/stores
- Instrument intrinsics
- Collect types

Run
- Intercept alloc/free
- Intercept loads/stores
- Intercept syscalls
- Collect stats

Analyze

INSTRUMENTATION PASS (LLVM)

RUN-TIME (COMPILER-RT)
Overview

Source Code
- No modification

Compile
- Instrument loads/stores
- Instrument intrinsics
- Collect types

Run
- Intercept alloc/free
- Intercept loads/stores
- Intercept syscalls
- Collect stats

Analyzer
- Score AP
- Guide exploration

INSTRUMENTATION PASS (LLVM)
RUN-TIME (COMPILER-RT)
VISUALIZER (ELECTRON)
Overview

**Source Code**
- No modification

**Compile**
- Instrument loads/stores
- Instrument intrinsics
- Collect types

**Run**
- Intercept alloc/free
- Intercept loads/stores
- Intercept syscalls
- Collect stats

**Analyze**
- Score AP
- Guide exploration

---

**INSTRUMENTATION PASS** (LLVM)

**RUN-TIME** (COMPILER-RT)

**VISUALIZER** (ELECTRON)
Memoro + Facebook

Run-Time Overhead

Visualizer

Open Challenges
Memoro + Facebook

Run-Time Overhead

Visualizer

Open Challenges
1,000x slowdown due to Memoro’s run-time
Run-Time Sampling

```c
int sample_count = 0;

void interceptLoadStore(...) {
    // Sample accesses
    if (sample_count++ % access_sampling_rate != 0)
        return;

    /* Process access... */
}
```
Run-Time Sampling

```c
THREADLOCAL int sample_count = 0;

void interceptLoadStore(...) {
    // Sample accesses
    if (sample_count++ % access_sampling_rate != 0)
        return;

    /* Process access... */
}
```
Power to the user!

MEMORO_OPTIONS="..." ./myapp
- access_sampling_rate
- ...

// Public API: memoro_interface.h
#include <memoro_interface.h>
void foo(...) {
    MemoroFlags *mflags = memoro::getFlags();
    mflags->access_sampling_rate = 50;
    /* ... */
}
Time spent by address type

- Primary Heap
- Secondary Heap
- Not Heap

99%
Time spent by address type

- **Primary Heap**
- **Secondary Heap**
- **Stack**

Stack accounts for 99% of the time spent.
The Allocators

Id 0x...

Metadata
- Addr
- Size
- First Access Time
- Access Range Low
...

Primary
- $O(1)$

Secondary - large allocations
- $O(n)$
The Allocators

Metadata
- Addr
- Size
- First Access Time
- Access Range Low
...

Primary
- O(1)

Secondary – large allocations
- O(n)
Issue with non-heap addresses
Issue with non-heap addresses

1. Allocators only know about heap
1. Allocators only know about heap
2. Traverse all allocations to discard them
Issue with non-heap addresses

1. Allocators only know about heap
2. Traverse all allocations to discard them
3. Takes a global lock
Issue with non-heap addresses

1. Allocators only know about heap
2. Traverse all allocations to discard them
3. Takes a global lock
Run-Time Filter
Run-Time Filter

1. *Thread start*: store stack top
Run-Time Filter

1. *Thread start:* store stack top

2. Get current stack bottom
Run-Time Filter

1. **Thread start**: store stack top
2. Get current stack bottom
3. Discard if Addr. in this range
Time spent by address type

- **Primary Heap**: 99%
- **Secondary Heap**: <2%
- **Not heap**: 0%
- **Stack Filtered**: 0%
1,000x slowdown due to Memoro’s run-time
5x

slowdown due to Memoro’s run-time
Memoro + Facebook

Run-Time Overhead

Visualizer

Open Challenges
Memoro + Facebook

Run-Time Overhead

Visualizer

Open Challenges
Truncate
Death by a thousand cuts
Death by a thousand cuts
Death by a thousand cuts

foo() VS. bar()

main()
Death by a thousand cuts
Death by a thousand cuts
Death by a thousand cuts

foo() VS. bar()

main()
Death by a thousand cuts
Death by a thousand cuts
Death by a thousand cuts
Death by a thousand cuts

main()

.....

foo()

main()

.....

bar()
Memoro + Facebook
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

    vector<BigT> values;
    values.reserve(largeMap.size());

    for (const auto& key: keys)
        values.emplace_back(largeMap[key]);

    return values;
}
Demo
Memoro + Facebook

Run-Time Overhead

Visualizer

Open Challenges
Dumping Profile

Your regular service

AtExit()
Dumping Profile

Your regular service → AtExit() → → Document
Dumping Profile

Facebook service

AtExit()
Dumping Profile

Facebook service  \rightarrow  \text{AtExit()}  \rightarrow  \text{AtExit()}  \rightarrow  \text{AtExit()}  \rightarrow  \text{AtExit()}
Dumping Profile

Facebook service

AtExit()
Dumping Profile

Facebook service

lldb

AtExit()

call AtExit()
Dumping Profile

Facebook service

lddb

AtExit()

call AtExit()
Dumping Profile

Facebook service

lldb

AtExit()

call AtExit()

a. Signal to dump (SIGPROF)
Dumping Profile

Facebook service

lldb

AtExit()
call

a. Signal to dump (SIGPROF)
b. Ring buffer + Periodic write
Compile-Time Stack Analysis
Compile-Time Stack Analysis
Compile-Time Stack Analysis

llvm::GetUnderlyingObject()

ld/st
Compile-Time Stack Analysis

ld/st

llvm::GetUnderlyingObject()
Compile-Time Stack Analysis

![Diagram showing the relationship between `GetUnderlyingObject(depth = X)` and the ratio of instrumented load/store operations. The graph shows a downward trend as the depth increases, with labels for `foo()`, `bar()`, and `ld/st` functions.]
Compile-Time Stack Analysis

```
llvm::GetUnderlyingObject()
```

Ratio Instrumented load/store

- 0
- 22500
- 45000
- 67500
- 90000

```
GetUnderlyingObject(depth = X)
```

```
ld/st
```

```
foo()
```

```
bar()
```

```
llvm::GetUnderlyingObject()
```
Thank you!
github.com/epfl-vlsc/memoro