Abstract

Abstract—When software does not meet performance requirements, difficult decisions are made to change central data structures which may be costly financially and increase development time. In addition, monitoring how these data structures are used, and trying to understand performance implications of any change may prevent any evolution of the original infrastructure. Thus, radical revisions to software may be avoided due to the barriers of time and engineering complexity costs.

Our solution to helping developers make infrastructure changes to improve performance is to provide a refactoring tool where developers may swap data structures. Our tool preserves correctness by utilizing the software's test suite and also measures performance automatically of the swapped data structure. We believe there is need for such a tool to help encourage more radical revisions and experimentation in large software projects to improve performance.

Our frameworks success will be evaluated based on preserving the correctness of the software within a developer created test suite while providing performance information based on modified data structures

ICSME 2019

Lib Metamorphosis:

A Performance Analysis Framework for Exchanging Data Structures in Performance Sensitive Applications

Mike Shah, Ph.D. (@MichaelShah, www.mshah.io)
Assistant Teaching Professor at Northeastern University
Duration ~5-7 minutes + time for questions

A Classic Problem (1/2)

- You are writing a piece of software that will manipulate some data
- You need to store, access, and modify the data
- What data structure do you choose?

A Classic Problem (2/2)

- You are writing a piece of software that will manipulate some data
- You need to store, access, and modify the data
- What data structure do you choose?
- An educated guess is the data structure with a good average-case complexity?
  - Or the worse-case?

Data Structure	Time Complexity			
	Average			
	Access	Search	Insertion	Deletion
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))

~~A Classic Problem~~ Another problem (1/2)

- You have profiled a program you are working on in a large team and determine it is running too slowly
  - (You of course did not write the original software)
- You want to try changing the data structures to improve performance
  - It will take a lot of effort to replace the data structure
  - And you do not know if your effort will yield a reliable performance improvement

~~A Classic Problem~~ Another problem (2/2)

- You have profiled a program you are working on i
  it is running
  - (You of co
    software)
- You want to
  structures to
  - It will take
    structure
  - And you do not know if your effort will yield a reliable performance improvement

Do you try a *radical revision* of your data structures to improve performance?

Our Preliminary Solution:

A tool for data structure profiling and data structure swapping

Part 1 - Data Profiler

- Our preliminary work involves gathering information about data structures
  - What data structure are you using?
  - What operations are you calling from it.

Data Structure	Time	Function Calls ...
std::vector	-90.0%	100
vector.push_back()	-99.0%	99
vector.at()	-1.0%	1
std::queue	-10.0%	100
queue.enqueue()	-50.0%	50
queue.dequeue()	-50.0%	50

Part 1 - Data Profiler (Implementation)

- We are using the LLVM compiler infrastructure to instrument the C++ STL to collect this information of metrics like:
  - time, number of functions, cache misses, etc.
  - (BS/MS student Robert Carney currently working on this)

Data Structure	Time	Function Calls ...
std::vector	-90.0%	100
vector.push_back()	-99.0%	99
vector.at()	-1.0%	1
std::queue	-10.0%	100
queue.enqueue()	-50.0%	50
queue.dequeue()	-50.0%	50

Part 2 - Data Structure Swap (1/2)

- We then are working on using LLVM to automatically swap a data structure with our own implementation
  - Thus avoiding any actual change to the source (we operate on the intermediate representation)

Data Structure	Time	Function Calls ...
std::vector	-90.0%	100
vector.push_back()	-99.0%	99
vector.at()	-1.0%	1
std::queue	-10.0%	100
queue.enqueue()	-50.0%	50
queue.dequeue()	-50.0%	50

Part 2 - Data Structure Swap (2/2)

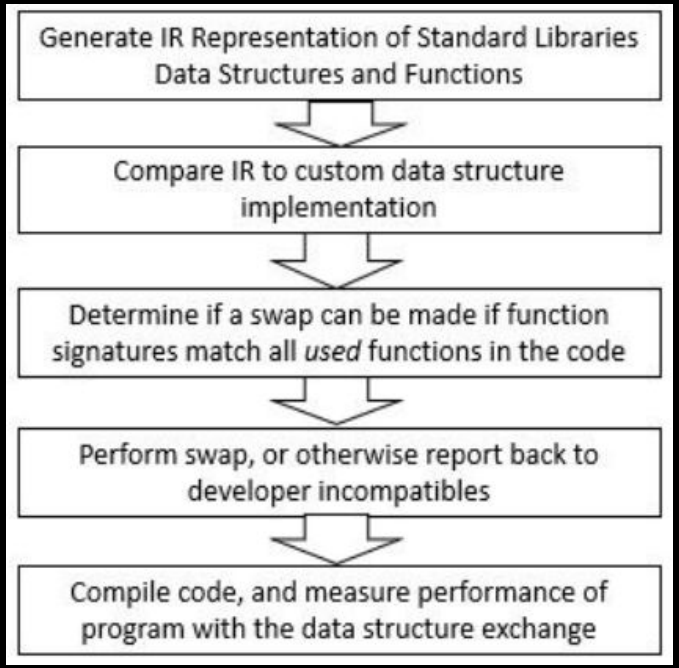
- We th LVM to automatically swap a data structure with o
  - T the source (we operate on the intermediate

Hmm, lets replace this data structure and measure performance

Data St	ime	Function Calls ...
std::vector	-90.0%	100
vector.push_back()	-99.0%	99
vector.at()	-1.0%	1
std::queue	-10.0%	100
queue.enqueue()	-50.0%	50
queue.dequeue()	-50.0%	50

Current Progress

- We have LLVM Infrastructure for instrumentation of data structures
- We can instrument parts of the C++ Standard Template Library (STL)
  - (The STL is optimized for the general case, so we think we can beat performance in specific domains)
- We are working on the data structure swap and how to measure if performance was increased
  - Likely using tools like Stabilizer by Curtsinger and Berger
    - STABILIZER: Statistically Sound Performance Evaluation



We think

- LLVM is the right approach
  - We may need capabilities to perform further static and data flow analysis
  - We might want to have the ability to only change *some instances* of data structures
    - (perhaps based on collection size, data types used, or frequency specific operations like adding and removing data)
- At the least
  - We think having a profile of how much time spent in data structures will be useful
- Our challenge
  - Ensuring a swap does not break program correctness
  - **We think** we can rely on program test suites.

Related Work

- Brainy: Effective Selection of Data Structures by Jung et al.
- Chameleon: Adaptive Selection of Collections by Shachum et al.

ICSME 2019

Thank you!

Mike Shah, Ph.D. (@MichaelShah, www.mshah.io)
Assistant Teaching Professor at Northeastern University
Duration ~5-7 minutes + time for questions