Literature Review of Leading Research in Compiler Optimizations

Timothy Van Slyke

February 26, 2018

Abstract

TODO: Many sections are currently just lorem ipsum'd and have yet to be written. All other sections mainly consist of talking points.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Introduction

TODO: Collapse some of the subsections directly into the introduction and just let the discussion flow on its own.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Rationale/Question/Problem

Mobile platform power saving…Modern advice is to "get it done quickly so the device can go back to sleep".
Desktop computing bottlenecked by memory latency

- There exist well-known cache-friendly access patterns but they don't always correspond with maintainable/readable code.

Desktop computing bottlenecked by memory latency
Most if not all desktop-related rationale applies to servers.
Additionally, server farms face power consumption problems similar to those that faced by mobile platforms (but for different reasons).

Primary question: What new research is being done to improve software speed and efficiency at the compiler level?

Background

Modern compilers have

Many of the optimizations made by modern comilers for static languages like C, C++, and Java are speculative rely an conservative deductions and assumptions that said compilers can infer about the code that they analyze.

C programmers' expectations about code generation ...
JIT compilation ...

- Popular for VM runtimes.
- Difficult for native code.

Cache latency is one of the largest factors in program speed ...

Methods

We've analyze publications from LIST OF CONFERENCES ...

Modern Compiler Optimizations

Many of the optimizations made by modern comilers for static languages like C, C++, and Java are speculative and rely an conservative deductions and assumptions that said compilers can infer about the code that they analyze.

Cache-Oriented Optimizations and Memory-Bound Code

Often the most important factor that determines the speed of software on modern computer systems is the extent to which the software is able to exploit the benefits CPU cache.[1] Much work is being done to speculatively optimize code for optimal memory access patterns and to reduce CPU stall time by rerdering and eliminating memory-bound operations.

[1] This factor is relevant on both desktop and mobile platforms. Exceptions to this rule are microcontrollers and other very small systems.

Languages with reference semantics (Java, C#, Python) suffer poor cache performance because of their implicit object model. ...

- Every object is dynamically allocated.
- Scattered data – no control over allocation patterns or memory layout.

Optimizing comilers for these languages suffer from few opportunities for alias analysis and must fight an uphill battle when it comes to optimizing for cache-friendly access pattens.

TODO: Move this discussion and associated visuals to introduction/background?

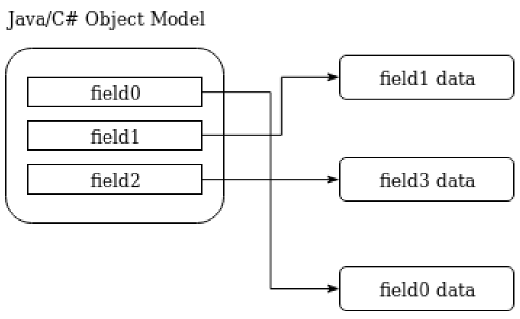

Figure 1: Visualization of the object model for languages with reference semantics. Object fields are stored sparsely and object references are aliased liberally. This leads to cache-unfriendly memory access patterns and poor conditions for alias analysis.

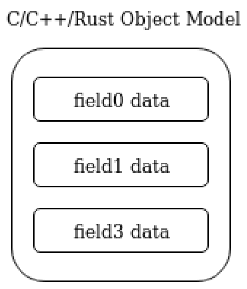

Figure 2: Visualization of the object model for languages with value semantics. Object fields are stored in a dense fashion and can only be aliased with explicit action in the code. This leads to more cache-friendly memory access patterns and allows for reasonable alias analysis.

Software Prefetching (via code generation) ...
Loop fusion and reordering ...

Runtime Solutions

JIT compilation ...
Non-JIT runtime code modification ...
Non-trivial code generation (compile-time) ...

- Many source have a theme of generating runtime checks for information that cannot be known at compiler time.
  - i.e. optimization decisions made at runtime after the necessary information is made available.

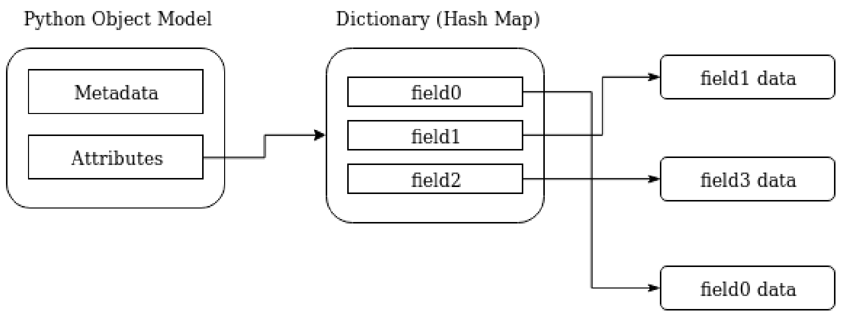

Figure 3: As a special case, the Python object model is, with only a few exceptions, implemented with a hash map; accessing object fields implies at least two layers of indirection. This allows for highly dynamic code, but leads to extremely poor memory access patterns and makes compile-time alias analysis infeasable.

Optimizing Parallel Code

Automatic parallelization of code ...
Optimization of synchronization schemes ...

Misc. or Yet-To-Be-Named Section

Alias analysis ...
...

Results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

References

[1] S. Ainsworth and T. M. Jones, "Software prefetching for indirect memory accesses," in *Proceedings of the 2017 International Symposium on Code Generation and Optimization*, ser. CGO '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 305–317. [Online]. Available: http://dl.acm.org/citation.cfm?id=3049832.3049865

[2] S. Dissegna, F. Logozzo, and F. Ranzato, "Tracing compilation by abstract interpretation," *SIGPLAN Not.*, vol. 49, no. 1, pp. 47–59, Jan. 2014. [Online]. Available: http://doi.acm.org/10.1145/2578855.2535866

[3] Y. Ko, B. Burgstaller, and B. Scholz, "Laminarir: Compile-time queues for structured streams," *SIGPLAN Not.*, vol. 50, no. 6, pp. 121–130, June 2015. [Online]. Available: http://doi.acm.org/10.1145/2813885.2737994

[4] J. Lifflander and S. Krishnamoorthy, "Cache locality optimization for recursive programs," *SIGPLAN Not.*, vol. 52, no. 6, pp. 1–16, June 2017. [Online]. Available: http://doi.acm.org/10.1145/3140587.3062385

[5] S. Mehta and P.-C. Yew, "Improving compiler scalability: Optimizing large programs at small price," *SIGPLAN Not.*, vol. 50, no. 6, pp. 143–152, June 2015. [Online]. Available: http://doi.acm.org/10.1145/2813885.2737954

[6] V. Paisante, M. Maalej, L. Barbosa, L. Gonnord, and F. M. Quintão Pereira, "Symbolic range analysis of pointers," in *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, ser. CGO '16. New York, NY, USA: ACM, 2016, pp. 171–181. [Online]. Available: http://doi.acm.org/10.1145/2854038.2854050

[7] T. Rompf, A. K. Sujeeth, K. J. Brown, H. Lee, H. Chafi, and K. Olukotun, "Surgical precision jit compilers," *SIGPLAN Not.*, vol. 49, no. 6, pp. 41–52, June 2014. [Online]. Available: http://doi.acm.org/10.1145/2666356.2594316

[8] K. Stock, M. Kong, T. Grosser, L.-N. Pouchet, F. Rastello, J. Ramanujam, and P. Sadayappan, "A framework for enhancing data reuse via associative reordering," *SIGPLAN Not.*, vol. 49, no. 6, pp. 65–76, June 2014. [Online]. Available: http://doi.acm.org/10.1145/2666356.2594342

[9] K. A. Tran, T. E. Carlson, K. Koukos, M. SjÃd'lander, V. Spiliopoulos, S. Kaxiras, and A. Jimborean, "Clairvoyance: Look-ahead

compile-time scheduling," in *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Feb 2017, pp. 171–184.