論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Supporting multi-scope and multi-level compilation in a meta-tracing just-in-time compiler
著者(和文)	伊澤侑祐
Author(English)	Yusuke Izawa
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第12328号, 授与年月日:2023年3月26日, 学位の種別:課程博士, 審査員:増原 英彦,遠藤 敏夫,南出 靖彦,脇田 建,渡部 卓雄,千葉 滋
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第12328号, Conferred date:2023/3/26, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Category(English)	Doctoral Thesis
種別(和文)	論文要旨
Type(English)	Summary

（博士課程）
Doctoral Program

論 文 要 旨

THESIS SUMMARY

系・コース： Department of, Graduate major in	数理・計算科学 数理・計算科学	系 コース	
学生氏名： Student's Name	伊澤侑祐		

申請学位（専攻分野）： Academic Degree Requested	博士 Doctor of （ 理学 ）
指導教員（主）： Academic Supervisor(main)	増原英彦
指導教員（副）： Academic Supervisor(sub)	

要旨（英文 800 語程度）
Thesis Summary（approx.800 English Words ）

A just-in-time (JIT) compiler is an essential technique in today's virtual machine (VM) s such as JVM, CLR, and JavaScript VMs to achieve high performance. In contrast to an ahead-of-time compiler like GCC, a JIT compiler compiles frequently executed code parts into native code at runtime. Since implementing a VM, including a JIT compiler, requires much engineering effort, a meta-JIT compiler framework has gained publicity in the field of VM implementation. Many of today's programming languages are realized using virtual machines. Among them, the meta-JIT compiler framework, which generates a virtual machine for a language from an interpreter that defines the language specification, is attracting attention. Meta-JIT compiler frameworks, such as RPython and Truffle/Graal, are used in the field of programming language implementation, and the usefulness of the system was demonstrated by actually realizing VMs of Python, Ruby, R, JavaScript, and so forth. However, the current JIT compilers provided by meta-JIT compiler frameworks have only a fixed compilation policy. They cannot compete with various compilation policies developed by JIT compilers in VMs for specific languages.

To further improve the performance of the JIT compiler generated by a meta-JIT compiler framework, a new compilation technology is needed to support multiple compilation scopes and levels. The dissertation shows that an interpreter, a language specification given to the meta-JIT compiler framework, is not only the semantic definition of the language but also the compilation policy and that multiple compilation policies can be realized on the meta-tracing JIT compiler framework. The dissertation proposes a multi-role meta-tracing JIT compilation that mixes different compilation scopes and levels in a meta-tracing JIT compiler framework. This technique is achieved by using a hint instruction, a pseudo-function inserted into an interpreter.

Method-based and trace-based heavyweight compilation policies have their advantages and disadvantages. While method-based compilation is standard and works well on average, trace-based compilation works better on programs with straight-line control flow. However, method-based compilation needs careful management not to fail in code explosion since including a not-frequently executed code path is inevitable. Further, the trace-based compilation performs poorly at compiling programs with complex control flow due to the path-divergence problem. A region-based compilation policy, which HHVM investigates, can use the two compilation scopes. Still, the current JIT compiler provided by a meta-tracing JIT compiler cannot use the promising compilation scope.
To take advantage of the two in a meta-JIT compiler framework, the first part of this dissertation proposes a technique that mixes optimizing method- and trace-based compilation policies in a meta-tracing JIT compiler. This allows meta-tracing JIT compilation to use two different compilation policies, method-based and trace-based, without having to implement separate compilers from scratch. This part also proposes the prototype implementation called BacCaml based on the MinCaml compiler. The help evaluation using BacCaml and MinCaml confirmed the existence of programs in which hybrid JIT compilation improves peak performance by about 50% over existing approaches using a single compilation strategy.

In contrast to heavyweight JIT compilation, lightweight compilation quickly emits code by applying less costly optimizations. Because the JIT compiler's compile time tends to dominate in the initial phase, using lightweight compilation in the initial phase helps improve the overall performance by reducing warm-up time. Several JavaScript VMs, such as V8 and JavaScriptCore, support lightweight compilation because of improving the initial performance of a browser loading web content. The second part of this dissertation proposes a threaded code generation technique, which allows a real-world meta-tracing JIT compiler, RPython, to method-based lightweight compilation. This part presents a two-level compilation mixing

threaded code generation and the (meta-)tracing JIT compilation. This compilation technique aims to balance the code quality and the compilation speed. While the BacCaml approach builds its meta-JIT compiler, this approach only partially extends the compiler basis. Still, it controls the behavior of a meta-tracing JIT compiler by a hint instruction, a pseudo function that is inserted into an interpreter. With the help of this architecture, language implementers can implement and customize the compilation/optimization level in an interpreter and determine their own compile/optimization level transition rules. This part implements threaded code generation and two-level compilation on PySOM, a Smalltalk dialect comprising about 14,000 lines of RPython code. The evaluation and experiment using the customized PySOM and RPython show that the proposed technique improves the overall performance in large-scale programs by up to about 14%.

To conclude, this dissertation proposes a technique to support multi-scope and multi-level compilation in a meta-tracing JIT compiler framework. With the help of customizing an interpreter definition, a meta-tracing JIT compiler can acquire different compilation behavior along with its original compilation. The evaluation using the prototype for real-world implementations can confirm that there are programs that improve performance compared to existing methods.

備考 ： 論文要旨は、和文 2000 字と英文 300 語を 1 部ずつ提出するか、もしくは英文 800 語を 1 部提出してください。