発表概要

実行時統計と Just-in-time コンパイルによるロックの選択的最適化

大平 怜[†] 平木 敬[†]

近年，Java バーチャルマシン（JVM）の上で動作するサーバ用コンポーネントが広く普及している．これらのコンポーネントは高速化のために実行時にネイティブコードにコンパイルされ（JIT コンパイル），最適化される．しかし従来の Java の JIT コンパイラに関する研究ではバイトコード最適化のみが対象とされてきた．本発表では，JIT コンパイラがシステムレベルの実行時最適化を行う手法を提案する．提案手法はネイティブコード中のシステム管理用コードを実行時統計と JIT コンパイラを用いて最適化するものであり，本発表においては特定のロックの動作を変化させスループットと公平性を両立させる最適化に適用する．サーバ側 Java では同一オブジェクトのロックに対して複数スレッドからの同時アクセスが頻繁に起こる．このため，ロック獲得中のプリエンプトを許す従来のロックを用いると，コンテンションによりタイムスライスに対してコンテキスト切替えの頻度が高すぎたり低すぎたりする．この問題の解決手法として，ロック獲得中のプリエンプトを遅延させる temporally non-preemption（TNP）が提案されている．しかし，TNP ではコンテンションが起きないロックについて無駄なオーバヘッドが加算される．我々の手法では，コンテンションが起きるロックを実行時統計を用いて検出し，JIT コンパイルによりそのロックに TNP を適用する．我々はこの手法を JVM の実装の 1 つである Kaffe に実装し評価を行った．その結果，TNP に対してオーバヘッドが最大で 3.4%削減され，かつコンテキスト切替の頻度とタイムスライスの一致によりスループットと公平性が両立していることが確認された．

Selective Optimization of Locks by Runtime Statistics and Just-in-time Compilation

Rei Odaira[†] and Kei Hiraki[†]

Recently, server-side components running on a Java virtual machine (JVM) have become popular. These components are compiled into native code (JIT compilation) and optimized at runtime for high performance. However, previous researches on JIT compilers for Java have been focused on only bytecode optimizations. In this presentation, we propose a method in which a JIT compiler performs system-level runtime optimizations. The proposed method is the one which optimizes system management code inside native code by using runtime statistics and a JIT compiler. In this presentation, we apply the method to the optimization that manages both throughput and fairness by changing behavior of particular locks. In server-side Java, it occurs frequently that multiple threads access the same object lock simultaneously. Thus if we use usual locks that allow preemption during critical sections, contention makes the frequency of context switches too high or too low, compared with the time slice. To solve the problem, temporally non-preemption (TNP) was proposed: the method which delays preemption during critical sections. TNP, however, increases unnecessary overhead on locks that induce no contention. Using runtime statistics, our method detects locks at which contention occurs and applies TNP to the locks by JIT compilation. We implemented the method to Kaffe, one of the implementations of JVM, and evaluated it. The result shows that our method reduces overhead at a maximum of 3.4% against TNP, and manages both throughput and fairness thanks to the consistency between the frequency of context switches and the time slice.

（平成 14 年 8 月 21 日発表）

† 東京大学情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo