4J-2

ユーザプログラムの応答性の低下を防ぐ実行時コンパイラ構成法

西岡 隆司 † 鈴来 和久 †† 毛利 公一 ††† 大久保 英嗣 †††
† 立命館大学理工学部 †† 立命館大学大学院理工学研究科 ††† 立命館大学情報理工学部

1 はじめに

近年, Java を始めとする中間言語を持つ言語において, プログラムの実行速度を向上させるために実行時コンパイラ方式が広く利用されている. 多くの実行時コンパイラは, ユーザプログラムの実行を中断し, コンパイル処理を優先して行う. これにより, コンパイル処理の要求から完了までの時間が最小となり, ユーザプログラムの多くの部分をコンパイル済みのコードで高速に実行することが可能となるためである.

しかし, 実行時コンパイラのコンパイル処理時間が長い場合や, ユーザプログラムが許容できる中断時間が短い場合, プログラムの応答性の低下や時間制約のある処理の実行が妨げられるといった問題が生じる. 一般に, 時間制約がある処理を行う場合, 処理に優先度を付与し, 時間制約の厳しい処理から実行する. そこで, コンパイル処理を行うスレッドの優先度に関しても適切に設定を行うことで, 高優先度スレッドの動作を妨げずにコンパイル処理が可能になると考えられる. 本稿では, コンパイルを行うスレッドを動的に変更する実行時コンパイラの構成法について述べる.

2 解決すべき課題

2.1 想定環境

本研究で想定する実行時コンパイラは, 大部分のコードをインタプリタで実行し, 一部の実行頻度の高いコードを最適化コンパイルする適応的コンパイラである. コンパイル処理を要求したユーザスレッドは, コンパイル処理の完了を待たずにインタプリタで実行を継続することができるとする. 仮想マシンが管理するユーザスレッドの実装は, ネイティブスレッド方式を想定している. すなわち, オペレーティングシステムが管理する軽量プロセスとユーザスレッドを１対１に対応させ, スケジューリングをオペレーティングシステムが行うことを想定している. ユーザスレッドが持つ優先度は, 異なる軽量プロセスの優先度に対応する. ユーザプログラムは, 応答性が要求される処理を高優先度スレッドで行うよう作成されている必要がある.

A Construction Method of JIT Compiler for Preventing the Degradation of User Program's Responsiveness
Takashi Nishioka†, Kazuhisa Suzuki††, Koichi Mouri†††, and Eiji Okubo†††
†College of Science and Engineering, Ritsumeikan Univ.
††Graduate School of Science and Engineering, Ritsumeikan Univ.
†††College of Information Science and Engineering, Ritsumeikan Univ.

2.2 ユーザプログラムの応答性の低下

ユーザプログラムにおける応答性の低下の要因として, ごみ集めによる予測不可能な停止時間と実行時コンパイルによる予測不可能な停止時間がある. 前者は, ごみ集めによる最悪中断時間を保証する実時間ごみ集め [1] を用いることで解決できる. 後者は, 特にハードウェアが低速である場合や実時間処理が求められる場合に問題となる.

- ハードウェアの性能が低い場合

  プログラム実行中のコンパイル処理は, 長期間の実行停止を発生させる. 具体例として, プロセッサの処理能力が低い携帯電話上でアプリケーションを実行させることを考える. キー入力への応答処理中にコンパイル処理が発生すると, 応答処理の完了が大きく遅れるため, ユーザにとって使い難いアプリケーションとなる.

- 実時間処理が求められる場合

  プログラム実行中のコンパイル処理は, 予期しない実行停止を発生させる. 具体例として, 機器を制御するプログラムを考える. タイマイベント発生により機器の制御を短い時間で終えなければならないとき, コンパイル処理が発生すると, デッドラインまでに処理が完了せず, 障害を引き起こす.

2.3 コンパイルスレッドの優先度の決定手法

前節で述べた実行時コンパイルの問題を解決するため, ユーザスレッドの優先度に基づき, コンパイルスレッドの優先度を決定する. ただし, コンパイルスレッドの優先度は, コンパイラスレッドに横取りされることによるユーザスレッドの中断時間に影響を及ぼす.

コンパイル処理の実行時間は, コンパイルスレッドの優先度を他のスレッドより高くすることにより短縮できる. したがって, 低優先度のスレッドがコンパイル処理を要求する場合, そのスレッドより高い優先度をコンパイルスレッドに与えることにより, コンパイル処理の実行時間を短縮できる. しかし, キー入力応答処理などの高い応答性を必要とするスレッドの実行を妨げないよう, これらの高優先度スレッドより低い優先度をコンパイルスレッドに与える必要がある.

本稿では, 以上の条件を満たすコンパイルスレッドの優先度の決定手法として, ユーザが作成したスレッドの優先度の値から, 最高値と最低値の中間値を求め, これをコンパイルスレッドの優先度とする手法を提案する.

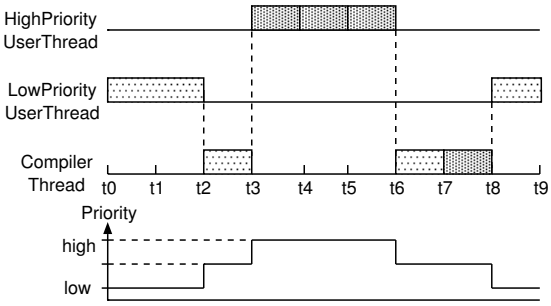

図 1　コンパイルスレッドの動作例

本手法により, 高優先度スレッドの応答性の低下を防ぎ, コンパイル済みのコードを実行することによるプログラム全体の処理時間の短縮を実現する.

3　動的に優先度を変更する実行時コンパイラ

3.1　実装

本稿で提案する実行時コンパイラは, jdk-6-b64 に変更を加える形で実装している. 実装を行うために重要となるクラスは, 中間言語のコンパイル要求を表現するコンパイルタスク, コンパイルタスクを格納するコンパイルキュー, 実行時にコンパイル処理を行うコンパイルスレッドである. 以下, 追加した項目を挙げる.

1. 現時点の仮想マシンに存在するユーザが作成したすべてスレッドを列挙し, それらのスレッドの優先度の値における最高値と最低値の中間値を求める関数を追加した.

2. 本稿で提案するコンパイルスレッドの優先度を 1 で求められた優先度に変更する処理をコンパイルキューからコンパイルタスクを取り出す処理の後へ追加した.

3.2　コンパイルスレッドの動作例

提案手法を実装した実行時コンパイラの動作例を図 1 に示す. 時刻 t_2 において, 低優先度スレッドからコンパイル要求が発生し, コンパイルスレッドの優先度は, 高優先度スレッドと低優先度スレッドが持つ優先度の中間値に変更される. 時刻 t_3 において, イベントが発生して高優先度スレッドが, 実行可能状態となり, 低優先度のコンパイルスレッドを横取りする. 時刻 t_4 において, 高優先度スレッドからコンパイル要求が発生する. しかし, 高優先度スレッドは, コンパイルスレッドの優先度より高いため, インタプリタで実行を継続する. 時刻 t_5 まで, 実行頻度の高いコードをインタプリタで実行する. 時刻 t_6 において, 高優先度スレッドは, 応答処理が完了し, 待ち状態に遷移する. 時刻 t_7 において, 低優先度スレッドから要求されたコンパイル処理が完了し, 高優先度スレッドから要求されたコンパイル処理を開始する. 時刻 t_8 において, 低優先度スレッドに横取りされることなくコンパイル処理を完了する.

表 1　実験結果

テストケース (優先度)	遅延時間 最大値	遅延時間 平均値	演算時間 (合計)
最高値	38341 μs	219 μs	939 ms
中間値	756 μs	9 μs	992 ms
最低値	650 μs	8 μs	1172 ms

4　応答性能と実行速度性能の評価

本稿で提案する実行時コンパイラの機能を評価するために, Intel Celeron プロセッサ 1.5 GHz, メモリ 512 MByte の計算機で実験を行った. Java 仮想マシンは, 3 章で述べた機能を追加実装している. 実行時コンパイル処理の負荷が高くなる環境をエミュレートするため, コンパイル処理を 10 回繰り返して行うよう変更している. また, Linux カーネル 2.6.14 に realtime-preempt パッチをあて, カーネルコンフィグレーションにおいて Preemption Mode オプションを PREEMPT RT(Complete Preemption) としてコンパイルしたカーネル上で動作させている.

性能評価プログラムは, プログラムの応答性能を評価するための高優先度スレッドと, プログラムの実行速度を評価するための低優先度スレッドから構成される. 高優先度スレッドは, RealTimeClock を読み出す処理を 1/1024 秒周期で実行する. 低優先度スレッドは, 行列演算を実行する. ごみ集めによるユーザプログラムの実行中断を避けるため, プログラム開始時に必要なオブジェクトを作成し, 実験中のオブジェクト生成を避けている. 実験結果を表 1 に示す.

表 1 において, 遅延時間は, 高優先度スレッドの応答性を示す指標であり, 演算時間は, 低優先度スレッドの実行速度性能を示す指標である. コンパイラスレッドの優先度をコンパイル要求時におけるすべてのスレッドの中間の優先度とすることで, 最低優先度に設定した場合に近い応答性能と最高優先度に設定した場合に近い実行速度性能を示していることがわかる.

5　おわりに

本稿では, ユーザが作成したスレッドの優先度に基づき, コンパイルスレッドの優先度を動的に変化させる実行時コンパイラ構成法を提案し, 評価を行った. 本稿で述べた実行時コンパイラ構成を実現することにより, 高優先度ユーザスレッドにおけるプログラム実行を妨げることなく, 全体として高速なプログラムの実行が可能となる.

参考文献

[1] Fridtjof Siebert : Hard Real-Time Garbage Collection in the Jamaica Virtual Machine, RTCSA 99, pp.96-102(1999).