4B－6

組込み向けJava™実行環境の開発

里山元章，北川健二，十山圭介
(株)日立製作所 システム開発研究所

1. はじめに

Java™[1]は,その処理系がWebブラウザに組込まれ世界中に広まった。最近では、小型の情報機器にもJavaが移植され利用可能になって来ている。Javaはバイトコート[1]と呼ばれる機械非依存な中間命令を解釈実行するインタプリタなため、異なるハードウェア間で同じプログラムを動かすことができる。組込み機器でJavaを利用する利点を以下にまとめた。

(1)高いソフト生産性：PC上のツールが利用可能な上、開発したプログラムが実機で即座に動かせる。

(2)組込み機器上のサービス拡大：ハード提供メーカ以外のベンダによるプログラム開発が可能になる。また、異機種間でのデータ交換や通信の標準化が進んでおり様々な新サービスが期待できる。

2. 技術課題

しかし、組込み機器でJavaを利用する場合、以下のような技術課題がある。

(1)C言語やアセンブラに比べ性能が悪い

(2)Java実行環境自体のサイズが大きい

(3)リアルタイム性が期待できない

(4)利用するメモリ量の予測が困難

リアルタイム性やプログラミング言語的なJavaの評価などの取組みに比べ[2]、組込み機器に向けた高速化について議論されることはあまりない。本稿では、消費電力やコスト面からメモリやCPUなどのハードウェアリソースが限られる組込み機器におけるJavaの高性能化について議論する。

3. Java実行環境の高性能化

Java実行環境は、バイトコード実行部(インタプリタ)、ランタイムライブラリ部、ホストOS、ハードウェアの4階層から成る。実行速度を向上させるには、各階層をバランス良く高速化する必要があるが、本稿ではインタプリタ部の高速化について述べる。

インタプリタ部を高速化するために、バイトコード(仮想命令)を実行直前に機械語命令へ変換するJITコンパイラ技術がある。しかし、JITコンパイラが生成する機械語は、バイトコードに比べ平均して一桁ほどサイズが大きい。メモリを節約するにはインタプリタで実行する方が得策である。そこで、インタプリタとJITコンパイラの両方を共用することにした。

3.1 インタプリタの高速化

実験には日立製SH[3]マイコンを使用した評価ボードを試作し、Sun Microsystems社のJavaOS™を移植して使用した[4]。コンパイラ等の開発環境にはGNUを、性能評価にはCaffeineMark 3.0を使用した。

インタプリタ部分はバイトコードを1命令づつ読みこんでは処理するスタックマシンである。Java実行環境で、CPUを占有する部分は、このインタプリタ部である。

Javaバイトコードは種類200以上あるが、各バイトコードの出現頻度にはかなりのばらつきがある。そこで、高頻度に出現するバイトコードの処理ブロックが、互いにキャッシュ上で競合しド処理ブロックをまとめあげ、全体のサイズも小

[1] Javaおよびすべての Java 関連の商標およびロゴは,米国およびその他の国における米国 Sun Microsystems, Inc. の商標または登録商標である.

Development of Java™ Runtime Environment for Embedded System.
Motoaki Satoyama (e-mail satoyama@sdl.hitachi.co.jp)
Hitachi, Ltd. Systems Development Laboratory
1099 Ohzenji, Asao, kawasaki, 215-0013 Japan

さくした。この他、いくつか細かい改造を施した結果、最適化前に比べて約 2.5 倍強、高速化できた。

一般に組込み機器で使われるキャッシュの容量は比較的小さい CPU (HITACHI SH3 の場合、1 次キャッシュ 8KB、2 次キャッシュ無) において、Java インタプリタ部への上記のような最適化がかなりの功を奏することが分かった。

3.2 省メモリ型 JIT コンパイラ

すでに述べたように JIT コンパイラはより多くのメモリを必要とするため、組込み機器では不利である。そこで、特定の条件に達したメソッドだけをコンパイルすることで省メモリ化することにした。メソッド単位で部分コンパイルする本方式では、広域的な最適化が行えず性能向上の限界は低い。しかし、組込み向けに、メモリ増加を押さえたままで高性能化を図ることを優先した。下記にその方針をまとめる。

(1)コンパイル条件を設定し省メモリ化する
(2)非コンパイル部分の性能を劣化させない
(3)最適化も含めて短時間でコンパイルする
(4)完全な互換性を維持する

当初、実行メソッドの実行頻度をプロファイルし、ある時点で最も実行頻度の高いメソッドからバックグラウンドでコンパイルする方式を取っていた。ところが、この方式では、プロファイルの負荷が大きくコンパイルしない部分の実行性能がかなり劣化することが分かった。

結局、単純に一定の実行回数を超えたメソッドからコンパイルする方式と、プログラマが明示的にメソッド名やクラス名を指定する方法を併用した。実行回数とクラス名/メソッド名(省略または部分指定可能)の全部の条件が一致したときコンパイルを開始する。

また、できるだけメモリ消費を押さえるため、単純にバイトコードを機械語へ変換するのでなく、大きな命令はあらかじめ用意したコンパイル済みブロックを繰り返し共用するようにした。これで、コンパイルによるメモリ消費を 50%程、押さえることができた。

本方式では、マルチスレッドで同じメソッドが走っているときに、どちらかがコンパイルされる、再帰呼出し中にコンパイル条件が成立する場合など実装が難しく互換性をとるのに苦労した。現在、最適化作業を進めているが、現状では、およそ 3 倍程度の高速化しかできてない。

4. おわりに

組込み機器用 Java 実行環境において、インタプリタ部の最適化の効果が非常に大きいことが分かった。一方、JIT コンパイラは、メモリ使用量の割に効果が少ない。例えば、グラフィックス系のベンチマークではほとんど効果が見られなかった。このため、本方式のように部分的に使用する方が得策である。JIT の効果が少ない原因として、組込み機器のようにキャッシュが小さい環境では、コンパイルによって実行ステップ数が増え、キャッシュミスが増加するとが予測できる。今後は、この点も踏まえて JIT コンパイラの組込み向け最適化を進める。

参考文献

1) Tim Lindholm, Frank Yellin: The Java™ 仮想マシン仕様,Addison-Wesley Publishers Japan, Ltd.
2) 石田晴久編集:特集 Java 言語: いま何が課題なのか,情報処理 Vol.39 No.4 (1998).
3) (株)日立製作所: SuperH 製品情報: http://www.hitachi.co.jp/Sicd/Japanese/Products/micom/super_h/technical/seihin.html
4) 組込みシステム用 JavaOS™ :里山、他:情報処理学会シンポジウムシリーズ Vol.98, No.15, pp.47-54