仮想計算機におけるデバイスエミュレーションの部分評価を用いた高速化

山本 悠輔[†] 新城 靖[†,‡] 榮樂 英樹[†] 板野 肯三[†,‡] 佐藤 聡[†] 中井 央[†] 加藤 和彦[†,‡]

筑波大学[†] 科学技術振興機構[‡]

1 はじめに

仮想計算機 (Virtual Machine, VM) において、入出力性能を改善することは重要な課題になっている。伝統的な仮想計算機では、実機と同じ抽象を提供するために、仮想計算機モニタ (Virtual Machine Monitor, VMM) において入出力デバイスのエミュレーションが行われている。入出力デバイスのエミュレーションは、重たい処理を含み、仮想計算機の入出力性能を大きく低下させる原因になっている。この方式に基づく仮想計算機の入出力方式を、**エミュレーション方式**と呼ぶことにする。

仮想計算機の入出力性能を改善する方法として、準仮想化に基づき特殊なデバイスドライバをゲスト OS(operating system) に組込む方法がある。準仮想化 (para-virtualization) とは、実機を仮想化する際に仮想化が困難な部分についてはあえて仮想化を行わず、実機とは異なる抽象を提供するような仮想化である。準仮想ドライバとは、ゲスト OS の中で他の一般のデバイスドライバと同じインタフェースを実装しているが、入出力においてハードウェアのデバイスを操作するのではなく VMM の機能を利用するものである。この方式に基づく仮想計算機の入出力方式を、**準仮想方式**と呼ぶことにする。

準仮想方式を使うと、エミュレーション方式と比較して入出力性能を大きく改善することができる。しかしながら、準仮想方式には、VMM ごとにデバイスドライバを開発しなければならないという問題がある。たとえば、VMware Workstation[4] では、Linux, Solaris, FreeBSD, および Windows のために準仮想ドライバを提供しているが、他の OS には提供していない。

この論文では、エミュレーションによる性能低下の問題と準仮想化における移植の問題を同時に解決する方法を提案する。具体的には、まず既存のゲスト OS デバイスドライバと VMM のデバイスエミュレータがともに多くの場合 C 言語で記述している点に注目し、両者を関数呼出しにより結合する。次に、特化 (specialization) または部分評価 (partial evaluation) を用いることにより、両者を合わせて高速化する。

現在、提案手法を、我々が開発している LilyVM において実装している。LilyVM は、x86 アーキテクチャを対象とした、準仮想化に基づく VMM である [2, 6]。LilyVM の特徴は、言語処理系を利用することで、ゲスト OS のうちアーキテクチャ依存部分の移植の労力を削減していることにある。LilyVM では、ゲスト OS と同じアドレス空間に実行時ライブラリを置く。この言語処理系を利用することと実行時ライブラリの存在が、本提案手法に適している。

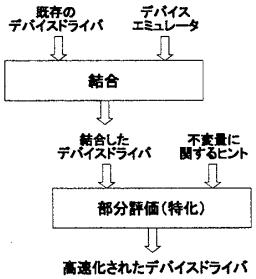

図 1 部分評価を用いた高速な準仮想ドライバの自動生成

2 部分評価を用いた高速な準仮想ドライバの自動生成

本研究では、仮想化にともなうオーバーヘッドの削減のために部分評価を用いる。図 1 に、提案方式の概要を示す。本方式では、まず、既存の実機用のデバイスドライバを用意する。このデバイスドライバは、ゲスト OS に固有のコードと入出力命令の発行を含む。次に、VMM からデバイスのエミュレータを抜き出す。このエミュレータは、ゲスト OS からは独立しており、入出力命令を受け取ると動作する手続きの集合になっている。このエミュレータは、ゲスト OS と同じアドレス空間で動作し、最終的に入出力が必要な時には、ハイパーバイザコールを用いてホスト OS の機能を利用する。

提案方式の 2 段階の処理では、高速化なデバイスドライバを自動生成する。第 1 段階では、デバイスドライバとデバイスエミュレータを結合する。第 2 段階では、結合したデバイスドライバを部分評価により高速化する。

2.1 デバイスドライバとデバイスエミュレータの結合

本研究では、ゲスト OS のデバイスドライバと VMM のデバイスエミュレータを結合する。両者は、ともに C 言語で記述させているが、デバイスドライバは、アセンブリ言語で入出力命令を実行しているので、単純には結合することはできない。よく設計されたデバイスドライバは入出力命令を行う部分を少数の関数に押し込めているという特徴がある。具体的には、inb() や outb() といった名前の関数やマクロを呼び出して、入出力命令を実行している。

本研究では、入出力命令が少数の関数で行われているという特徴を活用し、スタブを追加することで、デバイスドライバと

Speeding up device emulation by partial evaluation in virtual machines

Yusuke Yamamoto, Yasushi Shinjo, Hideki Eiraku, Kozo Itano, Akira Satoshi, Hisashi Nakai, Kazuhiko Kato

† University of Tsukuba

‡ Japan Science and Technology Agency

デバイスエミュレータを結合する。スタブは、デバイスドライバが用いている入出力を行う関数やマクロと同じ引数をとり、同じ結果を返すが、実際には入出力を行わずにエミュレータ内の手続きを呼び出す。

スタブへの置き換えは、デバイスドライバのソースプログラムを修正するのではなく、C 言語のヘッダファイルを置き換えることで実現する。このヘッダファイルは、ゲスト OS ごとに作成する必要があるが、内容はそれほど複雑なものにはならない。

2.2 Tempo によるコンパイル時特化と実行時特化

本研究では、部分評価を行う言語処理系として Tempo を用いる。Tempo は、C 言語、および、Java 言語を対象とした部分評価、または、特化を行う言語処理系である [1]。Tempo は、**コンパイル時特化** (Compile-time specialization) と**実行時特化** (run-time specialization) の両者を行うことができる。Tempo は、C 言語で記述されたプログラムと C 言語、および、ML 言語で記述されたヒントを読みこむ。ヒントは、主に不変量 (invariants) を記述したものである。Tempo は、束縛時解析 (Binding Time Analysis) と呼ばれる手法で、プログラムの中の変数が不変量にのみ依存していることを発見する。不変量がすべてコンパイル時にわかる場合、コンパイル時特化に基づき、より高度な C 言語のプログラムを出力する。不変量が実行時 (関数を呼ぶ前) にわかる場合、実行時特化を行い、その結果として、C 言語で記述された 2 つのプログラムを出力する。1 つは、テンプレートと呼ばれ、内部に後で埋めるべき「穴」を持つ。もう 1 つは、実行時特化プログラムであり、テンプレートを元に特化されたプログラムを動的に生成する。

本研究では、デバイスドライバとデバイスエミュレータを結合し、全体を Tempo を用いて特化を行うが、基本的にはコンパイル時特化だけを用いたいと考えている。その理由は、実行時のコード生成のオーバヘッドがないことや生成されたコードの管理が不要であることがあげられる。

2.3 結合したデバイスドライバの部分評価による高速化

本研究では、結合したデバイスドライバを部分評価により高速化する。コンパイル時に特化を行うためには、コンパイル時に知られる不変量が多い方がよい。しかしながら、デバイスのエミュレーションにおいては、実行時 (OS 起動時) になって初めて決定される不変量もある。

PCI バスには、個々の PC ごとに様々なデバイスが装着される。したがって、通常の OS は、起動時にプローブを行い、デバイスに対して動的にポート番号と割込み番号を割り当てている。このポート番号と割込み番号は、不変量であり、これを利用して特化を行うことができる。しかしながら、動的に決定されるため、コンパイル時には不明であり、本来ならば実行時特化が必要になる。

本研究では、本来ならば動的に決定されるポート番号を形式的に決定することで、コンパイル時特化を可能にする。形式的なポート番号とは、Tempo による解析を行う時にのみ有効なポート番号である。このような形式的なポート番号は、Tempo による解析が完了し、部分評価が完全に行われれば、出力されるコードには現れることはない。コンパイル時の特化により、手続きがインラインで展開され、いくつかの層がつぶされる。また、if 文が削除されたり、ループが展開されることもある。結果として、高速なデバイスドライバが自動生成されることになる。

3 関連研究

Paravirt_ops[3] は、Linux 2.6.20 (x86) で導入された関数の表である。この表は、仮想化することが困難な命令を含む。たとえば、割込みの禁止・許可、ページテーブルの設定を行う命令を含んでいる。この表は、OS の起動時に各 VMM ごとに切り替えられる。現在、実機、Xen、および、VMware についてこの表が実装されている。この表は、入出力命令を含んでいない。Paravirt_ops と比較して本研究の特徴は、部分評価を行っていること、入出力命令を扱っていること、および、Linux 以外の OS にも対応できることである。

VMware VMI(Virtual Machine Interface)[5] は、標準的な準仮想化を目指して提案されたインタフェースである。これは、BIOS ROM のように使われることを想定して設計されている。ゲスト OS をこのインタフェースを使うように書き換えれば、VMware Workstation 等のこのインタフェースを実装した VMM では高速に動作する。このインタフェースは、Paravirt_ops に含まれているような特権命令の他に、入出力命令も含んでいる。VMware VMI と比較して、本研究の特徴は、部分評価を行っていること、および、言語処理系により書き換えの手間を削減していることにある。

4 おわりに

この論文では、部分評価を用いた高速な準仮想ドライバの自動生成を行う手法について述べた。今後の課題は、デバイスのエミュレータを実現し、本提案手法で生成したデバイスドライバと手書きの準仮想ドライバの性能を比較する。

参考文献

[1] Consel, C., Hornof, L., Marlet, R., Muller, G., Thibault, S., Volanschi, E.-N., Lawall, J. and Noyé, J.: Tempo: specializing systems applications and beyond, *ACM Comput. Surv.*, pp. 1299–03 (2003).

[2] 榮樂英樹, 新城靖, 加藤和彦.: カーネル・レベル・コードによるユーザ・レベル VMM の移植性の向上, 情報処理学会第 104 回 システムソフトウェアとオペレーティング・システム研究会, pp. 17–24 (2007 年 1 月).

[3] Russell, R.: Paravirt_ops (2006).

[4] Sugerman, J., Venkitachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *Proceedings of 2002 USENIX Annual Technical Conference*, pp. 1–14 (2001).

[5] VMware: Paravirtualization API Version 2.5 (2006).

[6] 山本悠輔, 新城靖, 榮樂英樹, 板野肯三, 佐藤聡, 中井央, 加藤和彦.: 仮想計算機におけるデバイスエミュレーションの特化による高速化, 情報処理学会第 107 回 システムソフトウェアとオペレーティング・システム研究会 (2008 年 1 月).