2008－OS－108 (26)
2008／4／24

インタラクティブな組込みソフトウェアの動作検証に適したCPU速度制御手法

吉田哲也　山田浩史　河野健二

慶應義塾大学 理工学部 情報工学科

E-mail: {tetsuyay, yamada}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

携帯型のビデオプレイヤなどの組込みシステムでは，高い応答性を必要とするGUIや実時間処理を要するソフトウェアを動作させることが一般的になっている．通常，組込みソフトウェアの検証はデスクトップPC上でのシミュレーションやエミュレーションによって行われるため，こうした時間依存性の高いソフトウェアの応答性やビデオ再生の品質などを実時間で検証することは難しい．時間依存性の高い組込みソフトウェアの動作検証を容易にするため，本論文では，仮想機械モニタ (VMM) を用いて，組込みハードウェアと同等の速度でソフトウェアを動作させる手法であるFoxyLargoを提案する．FoxyLargoは，遅いCPU上のソフトウェアの動作を仮想マシン (VM) 内で再現するために，1) 細粒度のVMスケジューリング，2) クロック割込み契機のVMスケジューリング，3) 割込みの即時通知を行う．実験を行った結果，FoxyLargo上で動作するMPEGプレイヤによるフレームデコーディングのふるまいが，遅いCPU速度の実機上と同等になることが確認できた．

Slowing Down CPU Speed for Embedded Interactive Software Testing

Tetsuya Yoshida　Hiroshi Yamada　Kenji Kono

Department of Information and Computer Science, Keio University
E-mail: {tetsuyay, yamada}@sslab.ics.keio.ac.jp, kono@ics.keio.ac.jp

Embedded systems like portable video systems generally run interactive software such as GUIs and video players. Embedded software is usually verified in simulation- or emulation-based test environments so that it is difficult to verify the sensory responsiveness or QoS of interactive software. To ease the verifications, we present FoxyLargo, which slows down CPU speed with a virtual machine monitor (VMM) so that interactive software on it runs as slowly as on embedded hardware. To reproduce within a virtual machine (VM) the behavior of the software running on the real embedded hardware, FoxyLargo performs 1) fine-grained VM scheduling, 2) clock-tick interrupt based VM scheduling, 3) immediate notification of hardware interrupts. The experimental results demonstrated that an MPEG player on FoxyLargo behaved as that on slow CPUs.

1 はじめに

携帯型のビデオプレイヤなどの組込みシステムでは，高い応答性を必要とするGUIや実時間処理を要するソフトウェアを動作させることが一般的になっている．組込みシステムは，開発時間を短縮するため，ソフトウェアとハードウェアが並行して開発される．そのため，ソフトウェアの開発段階では，実際の組込みハードウェア上でテストができない．そこで，デスクトップPC上でシミュレーションやエミュレーションを用いたテストが行われている．仮想プラットフォーム [1] と呼ばれるテスト環境では，サイクルレベルや機能レベルでハードウェアをエミュレーションし，その上で組込みソフトウェアのシミュレーションを行う．この手法は，ソフトウェア／ハードウェア間のインタフェースのテストや，ソフトウェアの機能の正当性のテストに有効である．しかし，実時間ではなくシミュレーション時間でのテストであるため，時間依存性の高いGUIやビデオプレイヤの再生品質の動作検証が難しい．そのため，これらインタラクティブなソフトウェアを実時間で動作検証できるテスト環境が有用である．

組込みソフトウェアの動作検証を行う際，組込みCPUとテストに使われるデスクトップPCのCPUの速度差が問題となる．デスクトップPCのCPU速度は，組込みCPUの速度より速いため，デスクトップPC上で組込みソフトウェアの動作検証をするためには，CPU速度を制限する手法が必要である．

インタラクティブなソフトウェアの動作検証を目的としたCPU速度制限手法には，いくつかの課題がある．まず，任意のCPU速度を実現できる必要がある．組込みCPUの速度は多岐にわたり，例えばARMプロセッサの速度は1 MHzから1 GHzという範囲で20種類以上ある．これらの速度を全て実現するためには，細かい粒度でCPU速度を変更できる必要がある．動的電圧スケーリング (DVS) [2] という，ハードウェアレベルでCPU速度を制限する手法があるものの，提供するCPU速度の種類が

少ないため，本論文の目的には適さない．

もうひとつの手法として，仮想機械モニタ (VMM) を利用して CPU 速度を制限する方法がある．VMM は仮想機械 (VM) に与えるハードウェア資源を制御できるので，VM に与える CPU サイクルを制限できる．しかし，既存の VMM は長い周期 (Xen [3] は 30 msec 以上) で VM をスケジュールするため，VM は長時間の動作と休止を繰り返す．その結果，VM 内で動作するソフトウェアの周期タスクのデッドラインミスを引き起こす．

また，インタラクティブなソフトウェアはユーザの入力や他のハードウェアからの割込みを受け付けて動作する．例えば，ネットワークインタフェースからパケット到着割込みを受信した場合，ソフトウェアはパケットの処理を行う．これらの入力は通常，CPU の速度に関わらず即時に通知されるため，CPU 速度を制限した場合でも即時に通知する必要がある．

本論文では，上記の課題を考慮し，VMM を利用して CPU 速度を制限する手法である *FoxyLargo* を提案する．FoxyLargo は，1) 細粒度の VM スケジューリング，2) 割込みの即時通知，3) クロック割込み契機の VM スケジューリング を行う．これらの手法により，インタラクティブなソフトウェアを，あたかも遅い CPU 上で動作しているかのように動作させることが可能になっている．FoxyLargo を既存の VMM である Xen [3] に実装し，FoxyLargo が遅い CPU を持つ実機上で動作するインタラクティブなソフトウェアのふるまいを再現できることを実証するための実験を行った．FoxyLargo 上で動作する MPEG プレイヤのデコーディングのふるまいを測定し，遅い CPU 速度の実機上で動作させた場合と同様のふるまいをすることを確かめた．

2 CPU 速度制限手法に関する考察

CPU 速度を制限する手法として，既存の手法がいくつか挙げられる．しかし，これらの手法はインタラクティブなソフトウェアの動作検証には適さない．本章では，既存手法の不十分な点を明確化する．

2.1 動的電圧スケーリング

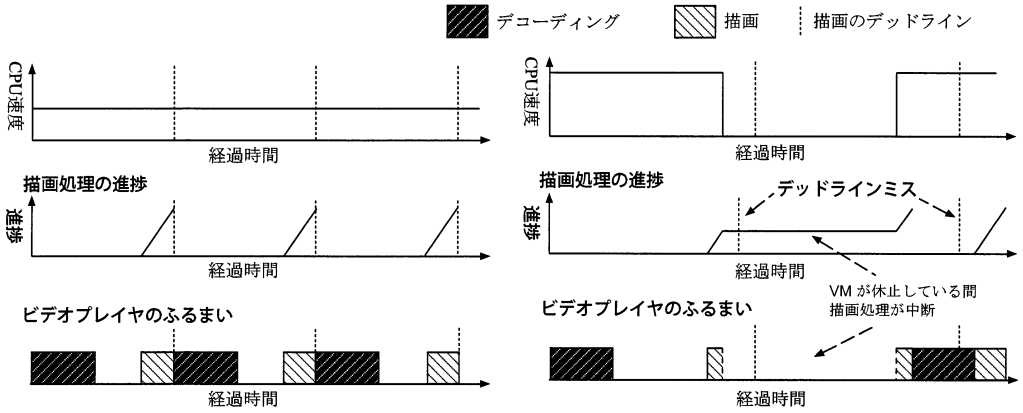
動的電圧スケーリング (DVS) は，近年の CPU に組み込まれている機能であり，CPU 速度をハードウェアレベルで動的に変更できる．例えば Intel 社の CPU には，*SpeedStep* [4] と呼ばれる DVS 機能が組み込まれている．DVS は，CPU チップを動作させるクロックシグナルを周期的に止めることで，CPU 速度を遅くする．クロックシグナルレベルの制御なので，目的の CPU 速度を正確に実現できる．

しかし，DVS は実現できる速度の種類が限られている．SpeedStep では 8 種類の速度しか選択できない．これに対し，組込み CPU の速度は種類が多く，速度の幅も広い．例えば，ARM という組込み CPU の速度は 20 種類以上あり，100 MHz から 1 GHz の幅がある．そのため，これら全ての速度を DVS で実現することはできない．

2.2 仮想機械技術

仮想機械技術を利用して CPU 速度を制限することができる．仮想機械モニタ (VMM) は仮想機械 (VM) に与えるハードウェア資源を制御できるため，VM に与える CPU サイクルを制限することで，その VM の CPU 速度を制限できる．

しかし，既存の VMM による CPU 速度の制限には，以下のような問題がある．まず，VM に与えるタイムスライスが長いため，VM スケジューリングの粒度が粗い．インタラクティブなソフトウェアは短い周期でタスクを処理することが多い．粒度の粗いスケジューリングでは，周期タスクを処理するタイミングで CPU が割り当てられず，周期タスクのデッドラインミスが起きる．例えば，33 msec に 1 度フレームのデコーディングと描画を行うビデオプレイヤを，Xen 上の VM 内で動作させることを考える (図 1)．Xen は 30 msec を最短周期として VM スケジューリングを行う．図 1(a) は，遅い CPU 速度の実機上で動作するビデオプレイヤである．デッドラインに間に合うタイミングでデコーディングと描画が行われている．これに対し，図 1(b) は，50% の CPU シェアを与えた VM 内で動作するビデオプレイヤである．1 フレーム目の描画途中で VM が停止してしまい，デッドラインミスが起きている．また，2 フレーム目でも描画処理がデッドラインに間に合っていない．この様に，VM スケジューリングの周期が長いと，周期タスクのデッドラインミスが起き，動画再生が安定しない．そのため，この VM 内で動画再生が不安定であっても，その原因がビデオプレイヤとテスト環境のどちらにあるのかを特定



(a) 目的の CPU 速度の実機　　(b) 既存の VMM を利用して CPU 速度を制限

図 1: 周期タスクの進捗の比較

できない.

さらに，インタラクティブな操作に対する応答性の検証も難しい．休止している VM に対して起こった割込みは，その VM が起動するまで遅延される．したがって，VM 内でインタラクティブなソフトウェアをテストした場合に，ユーザの入力に対する反応が遅かったとしても，テストしたソフトウェアに不具合があるとは断定できない.

3 FoxyLargo

前章で考察した既存技術の問題点を考慮し，VMM を利用して CPU 速度を制限する手法である *FoxyLargo* を提案する．ここで，FoxyLargo は既存の VMM と違い，VM を 1 つしか動作させないことを前提とする．FoxyLargo はソフトウェアのテスト環境を作り出すことが目的であるため，対象のソフトウェアを動作させる VM が 1 つ動作すれば十分である.

3.1 細粒度の VM スケジューリング

一般に，ビデオプレイヤなどの時間依存性の高いソフトウェアは短い周期 T でタスクを実行する．FoxyLargo は，インタラクティブなソフトウェアが周期タスクの周期 T あたりに消費する CPU サイクル数が，目的の CPU 速度の実機上で動作した場合と同じになるように VM をスケジュールする．これにより，FoxyLargo 上で動作するインタラクティブなソフトウェアが周期 T あたりに処理する仕事量が，目的の CPU 速度の実機上で動作した場合と同じになる．その結果，インタラクティブなソフトウェアの各周期タスクのふるまいが，目的の CPU 上の実機上で動作した場合と同じになる.

以上から FoxyLargo は，周期タスクの周期 T より短い周期で VM をスケジューリングし，VM 内で動作するインタラクティブなソフトウェアが小刻みに CPU サイクルを消費するようにする．その結果，周期 T あたりに消費する CPU サイクル数が，目的の CPU 速度の実機上で動作した場合と近くなる.

VM に与えるタイムスライスは以下の式で決定する．t_{period} は VM をスケジュールする周期，t_{vm} は VM に与えるタイムスライスの長さ，S_{target} は目的の CPU 速度，S_{real} は実際の CPU 速度を表す.

$$t_{vm} = t_{period} \cdot \frac{S_{target}}{S_{real}}$$

例として，FoxyLargo を用いて CPU 速度を 50 % に減速し，その上でビデオプレイヤを動かす場合を図 2 に示す．各周期で消費する CPU サイクル数が，目的の CPU 速度を持つ実機上で動作した場合と同じになっているため，各周期の仕事量が目的の CPU 速度を持つ実機上で動作した場合と同様になり，図 1(a) で示した実機上の場合とほぼ同じタイミング，同じ処理時間でデコーディングと描画が行われる.

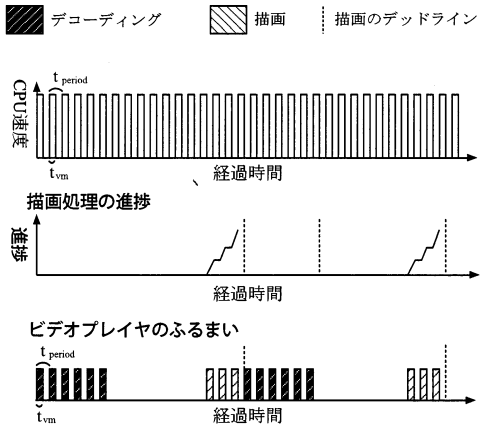


図 2: FoxyLargo 上で動作する周期タスク

3.2 割込みの即時通知

インタラクティブなソフトウェアのふるまいを再現するためには，割込みを考慮する必要がある．これらのソフトウェアは，ユーザからの入力を受けて動作することが多い．これらの入力の通知が遅れると，ソフトウェアがイベント処理に長い時間を要しているように見えてしまう．したがって，VM 内でソフトウェアをテストする場合，割込みを即時に通知する必要がある．

しかし，既存の VMM では VM が休止している間，割込みが遅延される．FoxyLargo では，VM が休止中に割込みが起こった場合，VM を再開して割込みを即時に通知する．

再開する VM の動作時間は，VM に与えるタイムスライスより短い時間にする．割込みハンドラの実行が終わるように，再開する VM の動作時間を 100 μsec に設定した．

しかし，このように休止中の VM を再開してしまうと，VM に対して過剰に CPU サイクルを割り当てることになる．このため，VM に与える CPU サイクルが割込みの頻度に依存してしまい，CPU 速度が一定にならない．これに対処するため，再開する VM に与える CPU サイクルを，次のスケジューリングで VM に与える CPU サイクルから差し引く．この様子を図 3 に示す．図 3 では，VM が休止している間に 4 回の割込みが起こっているため，次に VM に与えるタイムスライスから，4 回の割込み処理で消費した CPU サイクルを減らしている．この

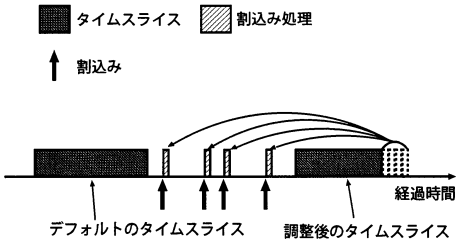


図 3: 割込みの即時通知を考慮した VM スケジューリング

方法により，VM が消費する CPU サイクルが一定に保たれ，CPU 速度が一定になる．

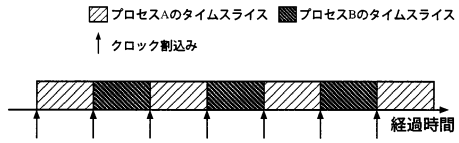
3.3 クロック割込み VM スケジューリング

インタラクティブなソフトウェアのふるまいを VM 内で再現するためには，ゲスト OS による時間管理を考慮する必要がある．OS が時間管理を正しく行わないと，プロセススケジューラの動作が実機上で動作する OS と異なってしまう．その結果，プロセススケジューラによるインタラクティブなソフトウェアのスケジューリングが，実機上で動作させた場合と異なってしまうため，インタラクティブなソフトウェアのふるまいが実機上で動作させた場合と異なってしまう．

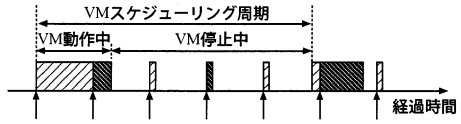
一般に，汎用 OS はクロック割込みを利用して時間経過を管理しており，クロック割込みが遅延されると時間管理を正しく行えない．例えば，プロセススケジューラはクロック割込み契機で動作するため，クロック割込みが遅延されると実機上と異なるタイミングでスケジューラが動作してしまう．そこで FoxyLargo は，ゲスト OS に対してクロック割込みを即時に通知する．

しかし，OS が時間管理を正しく行うためには，クロック割込みを即時に通知するだけでは不十分である．図 4 を使い，VM スケジューリングの周期の違いによる，ゲスト OS のプロセススケジューラのふるまいの違いを説明する．図 4(a) は，実機上の OS の場合である．クロック割込み間のプロセスの動作時間は全て同じである．

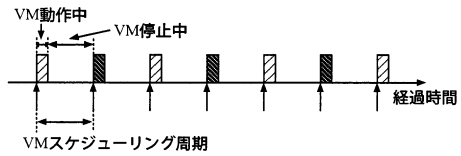
図 4(b) は，VM スケジューリングの周期がクロック割込み周期より長い FoxyLargo 上のゲスト OS の場合である．3 回目のクロック割込みは VM が休止



(a) 実機上の OS



(b) FoxyLargo 上の OS
(クロック割込み周期 > VM スケジューリング周期)



(c) FoxyLargo 上の OS
(クロック割込み周期 = VM スケジューリング周期)

図 4: OS 上のプロセススケジューリングの比較

中に起こるため，VM を再開してクロック割込みが通知され，ゲスト OS がプロセス A をスケジュールする．再開した VM は短時間しか動作しないため，プロセス A の動作は中断される．そして，次のクロック割込みの際には，中断したプロセス A ではなく，プロセス B がスケジュールされる．これは，プロセスのタイムスライスの長さがクロック割込みの回数で表されているため，4 回目のクロック割込みでプロセス A はタイムスライスを使い切ったとみなされるためである．一方，VM 動作中に起こる 1 回目と 2 回目のクロック割込みの間は，プロセス A は動作し続ける．このように，VM スケジューリングのタイミングによって，プロセスの動作時間が変化してしまう．

FoxyLargo はこの問題を考慮し，クロック割込みを契機として VM をスケジューリングする．図 4(c) に示したように，VM スケジューリングの周期とプロセススケジューリングの周期が一致していると，クロック割込み間のプロセスの動作時間が全て同じになる．

また，3.1 節で示したように，FoxyLargo はインタラクティブなソフトウェアの周期タスクより短い周期で VM をスケジュールする必要がある．クロック割込み周期の VM スケジューリングはこの条件を満たしている．例えば，ビデオプレイヤの描画周期は 33 msec であるのに対し，Linux 2.6 のクロック割込み周期は 1 msec である．

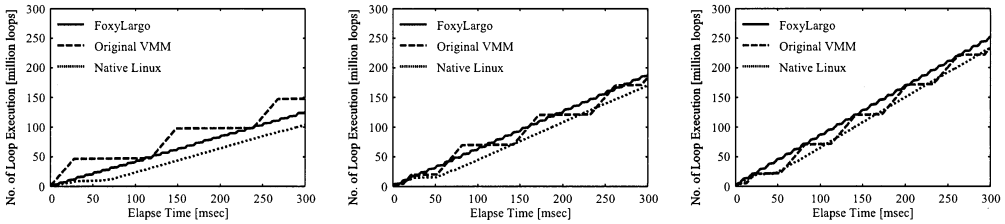
4 実装

FoxyLargo の有効性を示すため，FoxyLargo を Xen [3] 3.0.4-1 上に実装した．Xen は x86 アーキテクチャ上で動作する VMM である．Xen では，仮想化のオーバーヘッドを削減するために準仮想化 (paravirtualization) [5] を採用している．FoxyLargo の実装には準仮想化に関わる API を使用していないため，FoxyLargo は準仮想化を利用することなく実現可能である．したがって，VMware [6] など他の VMM にも原理上適用可能である．

なお現段階では，割込みの即時通知機能の対象をネットワークインタフェースカード (NIC) からの割込みとクロック割込みのみとしている．その他の割込みに対する即時通知機能は現在実装中である．

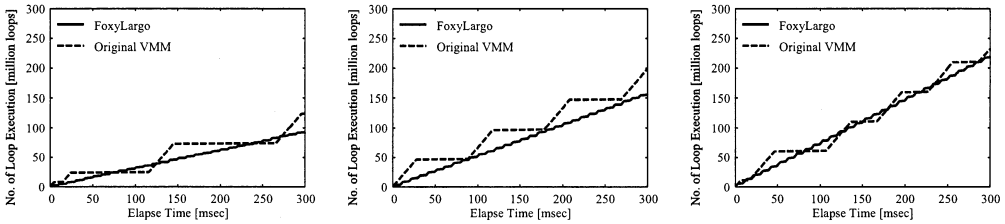
5 実験

FoxyLargo の有効性を示すために，3 つの実験を行った．実験に用いた PC は全て Pentium 4 2.4 GHz の CPU，512 MB のメモリ，1 Gbps の Ethernet NIC を備えている．既存の手法との比較を行うため，3 種類の構成でそれぞれ実験を行った．1 つ目の構成では，DVS を利用して CPU 速度を制限した．この構成を *Native Linux* と呼ぶ．実機上で Linux 2.6.22 を動作させ，その他の構成とメモリサイズを同じにするためにメモリを 256 MB に制限した．DVS はハードウェアレベルで正確に CPU 速度を制限できるため，本実験ではこの構成の実験結果を理想値として扱う．2 つ目の構成では，既存の VMM を利用して CPU 速度を制限した．この構成を *Original VMM* と呼ぶ．VMM は Xen 3.0.4-1 を用い，ドライバドメインとゲストドメインでは共に Linux 2.6.16 を動作させた．メモリは，ゲストドメインに 256 MB，ドライバドメインに 200 MB を割り当てた．3 つ目の構成では，FoxyLargo を利用して CPU 速度を制限した．この構成を *FoxyLargo* と呼ぶ．VMM は，FoxyLargo を Xen 3.0.4-1 に適用したものを用いた．各ドメインの構成は Original VMM と同じである．



(a) 25% speed (b) 37.5% speed (c) 50% speed

図 5: ループ処理の比較 (SpeedStep で実現可能な速度)



(a) 18.75% speed (b) 31.25% speed (c) 43.75% speed

図 6: ループ処理の比較 (SpeedStep で実現不可能な速度)

5.1 マイクロベンチマーク

5.1.1 細粒度の VM スケジューリングの効果

FoxyLargo 上で動作するソフトウェアが，1) 細粒度で CPU サイクルを消費する，2) 様々な CPU 速度を実現するという 2 つの目的を達成していることを確かめるため，CPU インテンシブなベンチマークによる実験を行った．このベンチマークは，ループ処理の中で 1000 回のループ毎にシステム時間を記録する．前述した 3 構成のそれぞれでベンチマークを動作させた．

実験結果を図 5，6 に示す．横軸に経過時間，縦軸にループの回数をとっている．図 5 は，SpeedStep で実現できる速度の場合である．この図から，FoxyLargo 上でのループ処理の進捗が線形であり，Native Linux 上とほぼ同じであることがわかる．したがって，FoxyLargo 上のソフトウェアが細粒度で CPU サイクルを消費しているといえる．一方，Original VMM 上のループ処理の進捗は線形でなく，Native Linux の場合と大きく異なる．これは，Original VMM の VM スケジューリングの周期が長いため，ベンチマークが動作しているゲストドメインが長時間休止してしまうためである．

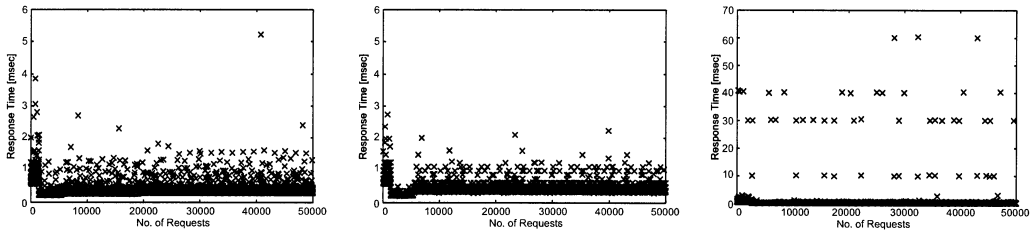
図 6 は，SpeedStep で実現できない速度で測定した結果である．この結果から，SpeedStep では実現できない速度でも FoxyLargo は実現可能であり，かつ細粒度で CPU サイクルを消費していることがわかる．

5.1.2 割込み即時通知の効果

FoxyLargo による割込みの即時通知の効果を示すため，ネットワークの応答性を測定した．前述した 3 つの構成を持つクライアントをそれぞれ用意し，Web サーバに対して HEAD リクエストを 50000 回送り，応答時間を測定した．ここで，サーバマシンでは，Apache 2.2.6 を Linux 2.6.22 上で動作させた．また，クライアントとサーバはギガビットイーサネットスイッチで接続した．

結果を図 7 に示す．横軸は HEAD リクエストの番号，縦軸は応答時間を表している．この結果から，FoxyLargo 上で測定した応答時間は，Native Linux の場合に近いことがわかる．このことから，FoxyLargo 上ではサーバからの応答を即時に受け取っており，Native Linux と同等の応答性があると言える．

一方，Original VMM における結果は，Native Linux



(a) Native Linux (25% speed)　(b) FoxyLargo (25% speed)　(c) Original VMM (25% speed)

図 7: HEAD リクエストの応答時間の比較

と FoxyLargo の場合とは大きく異なっていることがわかる．ここで図 7 において，縦軸のスケールが異なっており，Native Linux と FoxyLargo は 1 msec，Original VMM は 20 msec となっている．Native Linux と FoxyLargo における応答時間は最大で 5 msec 程度である．それに対し，Original VMM 上では最大 120 msec となっている．これは，VM が休止中に割込みが起こった場合，VM がスケジューリングされるまで割込みが遅延されていることを示している．

5.2 マクロベンチマーク

FoxyLargo が実際のソフトウェアに対しても有効であることを示すため，*mpeg play* [7] という MPEG ビデオプレイヤで実験を行った．前述した 3 構成のそれぞれで mpeg play を動作させて MPEG 動画を再生し，各フレームのデコーディング時間を記録した．そして，デコーディング時間の分布を比較した．使用した動画は，1800 フレームで構成されている．

結果を図 8 に示す．横軸はフレーム番号，縦軸はデコーディング時間を表している．まず，FoxyLargo における平均デコーディング時間は 6.34 msec，Native Linux における平均デコーディング時間は 6.33msc であり，その差は 0.142% となった．また，FoxyLargo におけるデコーディング時間の分散は 0.00274，Native Linux におけるデコーディング時間の分散は 0.00280 であり，その差は −2.00% となった．

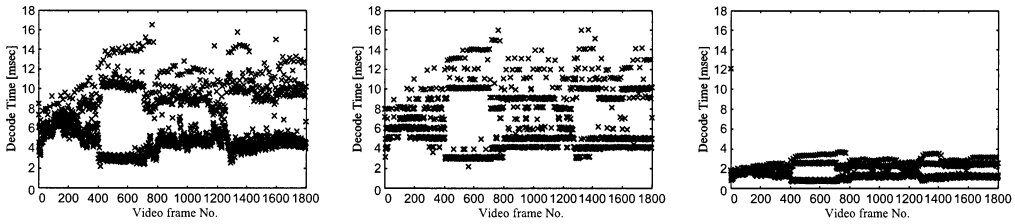
一方，Original VMM における平均デコーディング時間は 1.63 msec であり，Native Linux における平均デコーディング時間との差が −74.3.% となった．また，Original VMM におけるデコーディング時間の分散は 0.000688 であり，Native Linux におけるデコーディング時間の分散との差は −75.4% となった．

FoxyLargo は VM が一度に動作する時間が短く，デコーディング処理が小刻みに行われるため，デコーディング処理の進捗が Native Linux に近い．その結果，デコーディング時間が Native Linux に近くなる．一方，Original VMM は VM が一度に長時間動作するため，VM が一度動作するだけデコーディング処理が終わる．その結果，デコーディング処理の進捗が Native Linux より早くなるため，デコーディング時間が Native Linux より短い．

6 関連研究

組込みソフトウェアのテストを容易にするため，さまざまな研究が行われている．Justila [8] はハードウェア／ソフトウェア間のインターフェースのテストを自動化する．Hong ら [1] は仮想プラットホームを利用した開発のケーススタディを示しており，仮想プラットホームはソースコードの最適化に有効である反面，シミュレーション時間が実時間と比較して長くなることを報告している．我々の調査した限り，実時間でインタラクティブなソフトウェアの動作検証を行えるテスト環境はなかった．

本研究の他にも仮想機械技術をソフトウェアのテストに応用する技術が提案されている．Time travel virtual machine (TTVM) [9] は，VMM を利用し，OS に対する割込みなどのイベントのログを取る．このログを利用して非決定的なイベントが起こすバグを再現し，OS のデバッグを容易にする．TTVM は，



(a) Native Linux (25% speed) (b) FoxyLargo (25% speed) (c) Original VMM (25% speed)

図 8: MPEG ビデオフレームのデコーディング時間の比較

テスト環境と実際の環境のハードウェア資源の違いは考慮していない．これに対しFoxyLargoは，テスト環境のCPU速度を変更することで，実際の環境と同じCPU速度を実現する．

Time dilating [10]は，VMMを利用し，クロック割込みの周期を長くすることでゲストOS上の時間経過を遅くし，実機より速いネットワーク速度を持つテスト環境を構築する．DieCast [11]は，ゲストOS上の時間経過を遅くすることで，実機と同じ性能を持つVMを一つの実機上に複数動作させる．これにより，少数の実機で大規模なネットワークサービスの挙動を再現するテスト環境を構築する．これら2つの手法は，シミュレーション時間におけるソフトウェアの速度を変える手法であり，実時間における速度は変化しない．これに対しFoxyLargoは，実時間におけるインタラクティブなソフトウェアの速度を遅くする．

7 まとめ

本論文では，インタラクティブなソフトウェアの動作検証のためにCPU速度を制御する手法であるFoxyLargoを提案した．FoxyLargoはVMMを利用し，1) 細粒度のVMスケジューリング，2) クロック割込み契機のVMスケジューリング，3) 割込みの即時通知を行うことで，VM内のソフトウェアを，あたかも遅いCPU上で動作しているように動作させる．MPEGビデオプレイヤによるフレームデコーディングのふるまいを比較する実験を通じ，FoxyLargo上で動作するインタラクティブなソフトウェアが，遅いCPU速度の実機上と同等の速度で動作することを確認した．今後は，ネットワークインタフェースやディスクなどのデバイスの速度を遅くすることで，組込みハードウェア上で動作するインタラクティブなソフトウェアのふるまいをより正確に再現する手法を研究する．

参考文献

[1] Hong, S., Yoo, S., Lee, S., Lee, S., Nam, H. J., Yoo, B.-S., Hwang, J., Song, D., Kim, J., Kim, J., Jin, H., Choi, K.-M., Kong, J.-T. and Eo, S.: Creation and Utilization of a Virtual Platform for Embedded Software Optimization: An Industrial Case Study, *Proceedings of the International Conference on Hardware/Software codesign and System Synthesis*, pp. 235–240 (2006).

[2] Tam, D., Tsang, W. and Drula, C.: Dynamic Voltage Scaling in Mobile Devices, Csc2228 project final report, University of Toront (2003).

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proceedings of the Symposium on Operating Systems Principles*, pp. 299–310 (2003).

[4] INTEL CORPORATION: *Intel®64 and IA-32 Architectures Software Developer's Manual*, `http://www.intel.com/design/processor/manuals/253668.pdf` (2007).

[5] Whitaker, A., Shaw, M. and Gribble, S. D.: Scale and Performance in the Denali Isolation Kernel, *Proceedings of the Symposium on Operating Systems Design and Implementation*, pp. 195–210 (2002).

[6] Waldspurger, C. A.: Memory Resource Management in VMware ESX Server, *Proceedings of the Symposium on Operating System Design and Implementation*, pp. 181–194 (2002).

[7] Rowe, L.: The Berkeley MPEG Player. `http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_play.html`.

[8] Seo, J., Sung, A., Choi, B. and Kang, S.: Automated Embedded Software Testing on an Emulated Target Board, *Proceeding of the International Workshop on Automation of Software Test*, pp. 9–9 (2007).

[9] King, S. T., Dunlap, G. W. and Chen, P. M.: Debugging Operating Systems with Time-Traveling Virtual Machines, *Proceedings of the USENIX Annual Technical Conference*, pp. 1–15 (2005).

[10] Gupta, D., Yocum, K., McNett, M., Snoeren, A. C., Vahdat, A. and Voelker, G. M.: To infinity and beyond: time-warped network emulation, *Proceedings of the Symposium on Networked Systems Design & Implementation*, pp. 87–100 (2006).

[11] Gupta, D., Vishwanath, K. V. and Vahdat, A.: DieCast: Testing Distributed Systems with an Accurate Scale Model, *Proceedings of the Symposium on Networked Systems Design & Implementation* (2008).