Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

マルチコア向け複数カーネル実行機構におけるデバイス共有

下沢 拓[†1] 石川 裕[†1,†2]

本稿では，マルチコアマシン上で動作する複数カーネル間で通信を行うためのカーネル間通信機構を提案し，SHIMOS 機構に統合を行い，この通信機構を用いた仮想ネットワーク・ブロックデバイスの実装を述べる．SHIMOS 機構はカーネルに変更を加えることで資源を分割し，仮想マシンと異なりオーバーヘッドなく複数のカーネルを実行できるものである．SHIMOS 機構により並列実行させた複数の Linux カーネル上でベンチマークを実行し，Linux カーネルのコンパイルでは，ネイティブより 0.2%，仮想マシンより 25% 高速に実行できることを示す．

Shared Devices in a Multiple Kernel Execution Mechanism for Multicore Processors

Taku Shimosawa[†1] and Yutaka Ishikawa[†1,†2]

In this paper, we propose an inter-kernel communication mechanism to provide communication among multiple kernels in a multicore machine. This inter-kernel communication mechanism is integrated into the SHIMOS mechanism. The SHIMOS mechanism partitions resources by modifying kernels and realizes execution of multiple kernels within a single machine without overhead unlike virtual machine monitors. We also describe an implementation of virtual network and block devices using the inter-kernel communication. We conduct several benchmarks on the multiple Linux kernels executed by the SHIMOS mechanism, and show that SHIMOS achieves 0.2% better performance than the native kernel and 25% better than a virtual machine in the compilation benchmark.

†1 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

1. はじめに

マルチコアプロセッサは，コモディティマシンから組み込み用途のプロセッサまで幅広く用いられている[1)–3)]．このようなマルチコアプロセッサを用いた環境は一般化しているが，すべてのアプリケーションが増加する並列計算能力を利用できているとはいえない．

計算機センターにおいては，複数のアプリケーションを同時に用いたり，あるいは，複数のユーザーによって共有することによって，マルチコアマシンを効率的に利用することができる．この場合，アプリケーションのみでなく，複数のオペレーティングシステムを同時に実行することには，三つの利点がある．一つは，異なるオペレーティングシステムを要求するアプリケーションを同時に動作させることができること，もう一つは，ユーザーごとに隔離された環境を提供することができること，最後に，複数のアプリケーションのシステムコール等によってカーネル内で発生する競合を減らすことができることである．

複数のオペレーティングシステムを動作させる既存手法として，論理分割[4)] と仮想マシンモニタ (VMM)[5)] が挙げられる．前者は，ハードウェアもしくはファームウェア機構によって，マシンの資源を分割し，複数のオペレーティングシステムを起動するものである．この手法は，オーバーヘッドが小さい利点があり，IBM のメインフレーム[6)] や HP のサーバー[7)] などに搭載されているが，コモディティマシンにおいて採用された機種は我々の知る限りない．

後者の仮想マシンモニタは，ソフトウェアによって仮想的なマシン環境を作り，その上でオペレーティングシステムを動作させるものである．コモディティマシンにおいては，VMWare[8)]，Xen[9)]，KVM[10)] などが挙げられる．仮想マシンを実現し，またオーバーヘッドを少なくするための技術として，準仮想化[11)]，ハードウェアによる支援[12),13)] など多くが提案，採用されてきたが，仮想マシンの本質として特権命令や I/O 処理に対するオーバーヘッドは回避することはできていない．

我々は，性能低下を防ぎながら複数のオペレーティングシステムを実行することのできる新たな機構として，SHIMOS (Single Hardware with Independent Multiple Operating Systems) 機構を提案し，設計実装を行った[14)]．これは，オペレーティングシステムカーネルに対して改変を加えることで実現され，カーネル自身が利用する資源を制限することで複数カーネルの同時実行を可能にするものである．SHIMOS 機構は，仮想マシンのようなオーバーヘッドがなく，かつ特殊なハードウェアを必要としないという利点がある．SHIMOS 機構は，仮想マシンと異なり，各カーネル間の保護は提供しない．このため，カーネルが異常

Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

動作した場合，システムの動作は保証されないが，これは，高性能なカーネル実行環境を提供することを優先するという設計思想に基づいたものである．この結果，論文[14]において，x86 アーキテクチャを対象とした実装で，Xen よりも 134%の高速化を実現したことを示した．また，論文[15]において，SH アーキテクチャを対象とした実装を延べ，可搬性を示した．

SHIMOS 機構は，効率的な複数カーネル実行機構を提供するが，I/O デバイスを共有できないという大きな制限が存在する．この制限は，カーネルの数だけデバイスが必要ということであり，多くの環境で受け入れられるものではない．本稿では，デバイスの共有を行う一般的な枠組みを提供するためのカーネル間通信機構を示し，この通信機構を統合した SHIMOS 機構における仮想ネットワークデバイス，ブロックデバイスの実装も示す．また，SHIMOS 及び仮想マシンによって複数カーネルを動かし，その上でいくつかのベンチマークを実行して評価を示す．このうち，Linux コンパイルベンチマークでは，Xen より約 25%，KVM より約 34%高い性能となることを示す．

本稿の構成は，以下のとおりである．次節において，複数カーネル実行機構の概略を述べる．第 3 節において，カーネル間通信機構の設計と実装を述べる．第 4 節において，カーネル間通信機構を用いた仮想ネットワークおよびブロックデバイスの Linux に対する実装を示す．第 5 節で関連研究を示し，第 6 節では，マイクロベンチマークとアプリケーションによるベンチマークを行い，仮想デバイスの性能についての評価を示す．第 7 節で結論を述べる．

2. SHIMOS 機構

本節では，筆者らが提案した複数カーネル実行機構である SHIMOS 機構について簡単に述べる．SHIMOS 機構は，カーネルに変更を加え，各カーネル自身が利用する資源を制限することにより資源分割を実現するものである．そのため，各カーネルは特権命令や I/O 処理も含めて，CPU によって直接実行されるため，カーネルの実行においてオーバーヘッドが存在しない．以下において，CPU，メモリ，デバイスの分割がどのように行われているかを述べる．

2.1 CPU

CPU の分割は，カーネルが他のコアを起こす時にそのコアが使えるかどうかの条件判定を行うことによって実現する．この条件は，後で述べる共有メモリ領域にある資源の割り当て表に基づいて行われる．

他方，いかに複数のカーネルを起動するか，という問題がある．SHIMOS 機構では，最初に，BIOS などの通常のブート手順によって起動されたカーネルが，KLoader と呼ばれるカーネルモジュールを介して，他カーネルをメモリに配置し，CPU コアを起動しカーネルを実行させることによって実現する．

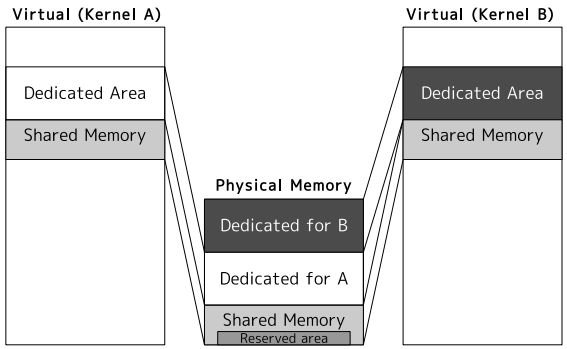

図 1 SHIMOS Linux における物理-仮想メモリマップ

2.2 メ モ リ

メモリは，各カーネルに専有させる部分と，カーネル間で共有する共有領域の二つの部分からなる．例として，SHIMOS 機構上で動く Linux における仮想メモリマップを**図 1** に示す．専有部分は，各カーネルが排他的に利用する領域で，カーネルが通常使用する領域として用いられる．他方，共有部分は，三つの目的で用いられる．一つは，第 3.1 節で述べるように，カーネル間通信機構で用いるため，もう一つは，資源割当て表などの各カーネルのメタデータを保持するため，最後に，アーキテクチャ上の制限や要求により，各カーネルが起動時などにアクセスするためである．

2.3 デ バ イ ス

デバイスには，レガシーなデバイスと，PCI やその後継バスを用いたデバイスの二種類が主に存在する．前者については，統一的なアクセスの方法がないために，各デバイスのドライバコードを修正することで，分割を行った．後者については，PCI 系のデバイスは，バス番号，デバイス番号，ファンクション番号の三つ組みによってデバイスが識別され，これに基づき資源の割り当て表を参照することで分割を実現する．

3. カーネル間通信機構

SHIMOS 機構に統合されるカーネル間通信機構は，共有，転送，通知という三つの部分

Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

からなる．以下において，それらの設計と実装について述べる．

3.1 共有メモリ

異なるアドレス空間を用いるカーネル間で通信を行うためには，カーネル間で共有されるメモリ領域が必要になる．また，この領域に対するアクセスを容易にするために，すべてのカーネルから同一仮想アドレスでアクセスできる領域を導入した．この領域からは，カーネル間通信パケット (以下では，IKC パケットと表記する)，パケットキュー，後述する共有メモリ方式におけるデータが確保される．

3.2 転 送 方 式

異なるカーネル間において転送を行う際に，二つの大きな課題がある．一つは，通信するカーネル間でデータ表現が異なること，もう一つは，転送したデータに対するアクセスのオーバーヘッドを小さくすることである．

第一の課題は，仮想デバイスは，デバイスのリクエストを通信により伝えなければならないが，カーネルの種類，あるいはバージョンによってそのデータ構造が違うことによるものである．たとえば，Linux カーネルにおいてネットワークパケットを表す `sk_buff` 構造体は，パケットのプロトコル階層ごとのヘッダ位置などの情報を含んでいるが，これらはバージョンによって変化している．他方，パケットのデータ自体は，カーネルのバージョンに依存するものではない．

このため，IKC パケットは，カーネル内のデータ構造をカーネルに依存しない形に変換したものと，実際のデータへのポインタを持つ構造としている．仮想デバイスは，カーネル内でのデバイスリクエストを IKC パケットに変換し，送信先カーネルの仮想デバイスは，IKC パケットをカーネルの構造体に変換したうえで，物理デバイスに対してリクエストを行う．

データの受け渡しに関しては，データの位置に関して二つの場合が考えられる．一つは，前述した共有メモリ領域内にある場合で，もう一つは，それ以外の一般の部分にある場合である．それぞれの方法に対応するために，共有メモリ方式とページ転送方式という二つの方式を提供する．以下でそれぞれの方式について述べ，また，アクセスのオーバーヘッドを減らすためにコピーをいかに避けるかについて述べる．

3.2.1 共有メモリ方式

共有メモリ方式は，共有メモリ領域からアロケータを通じて確保したデータを転送するものである．転送するメモリ領域は，この共有メモリ領域に限られるが，すべてのカーネルから同じ仮想アドレスによってアクセス可能な領域であるため，アクセスに対して特別な処理を必要としない利点がある．またこの結果として，コピーを避けることができる．

この方式を実現するためには，すべてのカーネル間で共通に用いられるメモリアロケータが必要となる．メモリアロケータは複数のカーネルで共有されるために，アロケータ内でスリープすることができない．我々は，単純なアルゴリズム (計算量は，確保に $O(n)$，解放に $O(1)$) で実装を行った．また，断片化を防ぐために，1024 バイト単位のブロックでの確保とした．メモリアロケータを共通化することで，転送されたデータの解放も，転送の受信側のカーネルで行うことができ，データの流れを単純化することができる．

3.2.2 ページ転送方式

ページ転送方式は，共有メモリ領域外のメモリを転送するためのものである．この方式も，前述の共有メモリ方式と同様に，コピーを行わない転送を可能にするが，メモリのマップを行うコストが存在する．

ページ転送方式では，転送するデータの物理アドレスが転送される．受信したカーネルが，その物理アドレスをマップしアクセスを行う．この方式で発生するオーバーヘッドは，ページテーブルの変更や TLB ミスなどのマップにかかわるコストである．そのため，この方式は，ブロックデバイスリクエストなどの大きなデータを少ない頻度で送信する場合に用いられる．また，解放も送信元のカーネルで行う必要があるので，解放可能であることを通知するパケットを送信元に送り返す必要がある．ただし，これらはデバイス要求が完了通知を行うパケットで表すことができるので，ブロックデバイスなど完了通知が必要なデバイスでは，オーバーヘッドにはならない．

3.3 通　　知

カーネル間通信においては，パケットの存在を他カーネルに通知することが必要になる．この通信機構では，カーネル間通信 (IPI) を用いて行う．IPI を受けた時，各カーネルは，自分の IKC パケットキューを確認し，パケットがあればその処理を行う．

4. 仮想デバイス

本節では，x86 アーキテクチャの Linux 2.6.26 に対しての仮想デバイスの実装について，ネットワークデバイスとブロックデバイスについて述べる．

4.1 ネットワークデバイス

仮想ネットワークデバイスの実装では，共有メモリ転送を用いた．ネットワークパケットを共有メモリアロケータを用いて共有メモリ領域から確保するために，`alloc_skb` 関数および `free_skb` 関数を改変し，Linux カーネル内でパケットを表す `sk_buff` 構造体の確保時にバッファを共有領域から確保，解放するようにした．構造体そのものは，カーネル間で必

Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

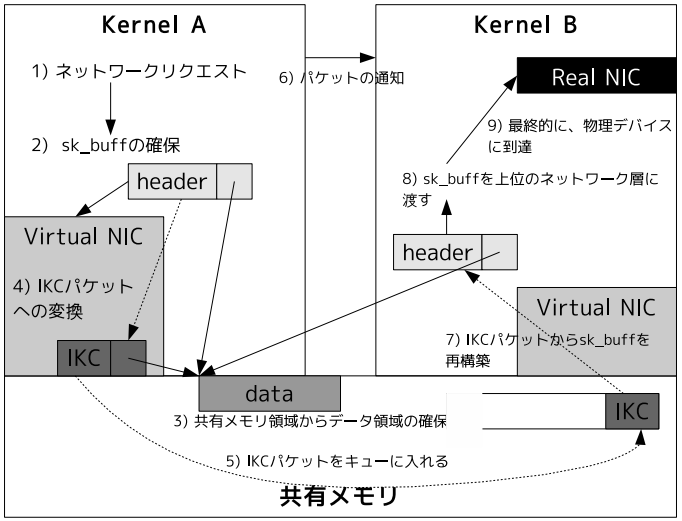

図 2 仮想ネットワークデバイスの実装

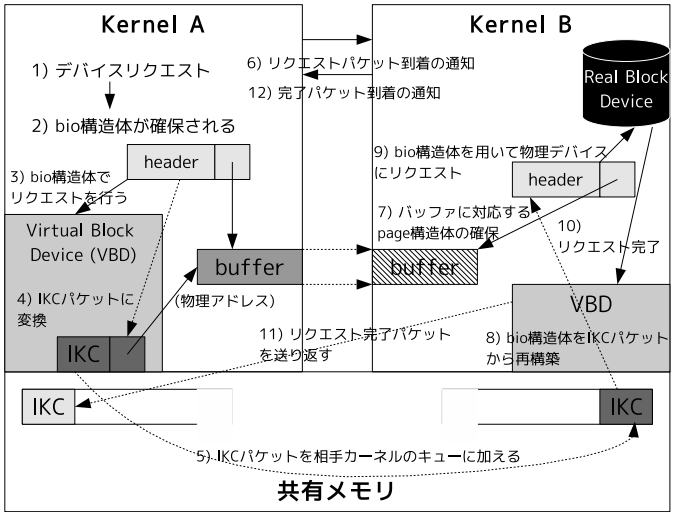

図 3 仮想ブロックデバイスの実装

ずしも同じではないので，共有せず通常のカーネルメモリから確保される．

カーネル間通信機構を用いた仮想ネットワークデバイスの処理を**図 2** に示す．この図では，カーネル A から仮想ネットワークデバイスを介して，カーネル B に対してパケットを伝達するときの処理を示している．(1) 最初に，カーネル A でネットワークパケットのリクエストが発行される．(2) カーネル A は，`alloc_skb` 関数により `sk_buff` 構造体を確保する．(3) 前述したように，構造体のヘッダは，通常のメモリ領域から確保され，データを格納するバッファは共有メモリ領域から確保される．(4) `sk_buff` 構造体が，仮想ネットワークデバイスに渡され，仮想デバイスは IKC パケットに変換する．(5) 変換された IKC パケットは，カーネル B のキューに入れられる．(6) 必要であれば，カーネル B に対して通知を行う．(7) カーネル B は，IKC パケットをキューから取り出し，`sk_buff` 構造体を再構築する．(8) 再構築した `sk_buff` を上部のネットワーク層に投げる．(9) 最後に，物理ネットワークデバイスに到達する．

ネットワークパケットを表す IKC パケットは，データのバッファ内のオフセット，データサイズ，バッファサイズを含み，これらが `sk_buff` を再構成するのに十分である．

4.2 ブロックデバイス

仮想ブロックデバイスの実装として，ページ転送方式を用いた．これは，仮想ブロックデバイスに対するリクエストは，カーネル内の任意のアドレスに対するものとして行われるからであり，ネットワークデバイスの場合のようにバッファを一定のメモリ領域に制限するのが困難であるからである．また，ブロックデバイスで行われるリクエストの単位が大きく，かつその要求される転送速度も，ネットワークデバイスの場合よりも小さいので，ページ転送方式のコストは無視できると考えられる．

カーネル間通信機構を用いた仮想ブロックデバイス処理の流れを**図 3** に示す．この図は，カーネル A から仮想ブロックデバイスを介して，カーネル B がもつ物理ディスクにアクセスするときの処理を示している．(1) カーネル A 上のアプリケーションによって，物理デバイスに対して読み込みか書き込みが行われようとする．(2) カーネル内でリクエストを表す `bio` 構造体が確保され，データを読み書きするバッファの存在するページが指定される．(3) 仮想ブロックデバイスに対して，この `bio` 構造体によってリクエストが行われる．(4) 仮想ブロックデバイスは，`bio` 構造体を IKC パケットに変換する．(5) カーネル B のパケットキューに IKC パケットを入れる．(6) カーネル B に対してパケットの到着の通知を行う．(7)

Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

カーネル B の仮想ブロックデバイスが，IKC パケットをキューから取り出し，対象となるバッファのページに対応する page 構造体を作成する．(8) このページをバッファして，bio 構造体を作成する．(9) 物理デバイスに対して，この bio 構造体でリクエストを行う．(10) 物理デバイスによるリクエスト完了時には，page 構造体を解放し，(11) 要求完了を表す IKC パケットをカーネル A のキューに入れる．(12) カーネル A に対し，必要であればパケットの通知を行う．(13) カーネル A の仮想デバイスは，元のカーネル内リクエストの完了通知関数を呼び出す．

また，Linux カーネルにおいては，ページを扱うために，page 構造体が必要になるが，元のカーネルは線形のアドレスを仮定し，page 構造体も一次元の配列であることを仮定している．そのために，任意のページのマップを行う処理を行うために，page 構造体に物理アドレスを格納するメンバを加える，という最も単純な方法をとった．

5. 関連研究

Xen においては，ドメイン間の通信は，event channel と呼ばれる通知機構と，grant table によるページ転送を用いて行われる[16]．Grant table は hypervisor を介して作られるページのリストで，一つのドメインが他のドメインに対してアクセスを許可するメモリページのリストである．これにより，ドメイン間でゼロコピー通信を実現している．さらに，I/O 処理の安全性を保ちつつ Xen のドメイン間通信やネットワーク処理を高速化する手法が提案されている[17),18)]．

仮想マシンから I/O を直接扱えるようにするハードウェア支援技術も多く規格が策定され，対応した製品が提供され始めている．IOV[19)] は，単一の PCI-Express デバイスを複数の仮想マシンからの共有をデバイスがサポートする規格である．この規格をサポートするデバイスは，仮想マシンごとのコンテキスト保存を行うことができる．また，VT-d[20)] は，デバイスの扱うアドレスに対してアドレス変換を行うことにより，仮想マシンから直接デバイスを扱うことを可能にする．これらの技術を用いることにより，ソフトウェア側での特殊な処理を必要とせずに単一デバイスの共有を仮想マシン間で行うことができる．

Microvisor[21)] は，メンテナンス時のみ仮想化によって複数の OS を起動しオンラインでのメンテナンスを実現し，かつ，平常時は直接実行することにより仮想化のオーバーヘッドを避ける手法である．しかし，デバイスの共有は行わないため，メンテナンス用に余分なデバイスを用意していなければならないという欠点がある．

表 1 評価環境

モデル	DELL Precision 490
CPU	Intel Xeon 5130 (dual-core, 2.0GHz) x 2
メモリ	DDR2 667MHz FB-DIMM 1024 MB x 2
HDD	SATA 250GB x 3, PATA 120GB
OS	Linux 2.6.26
NIC	Intel (e1000, PCI-X)

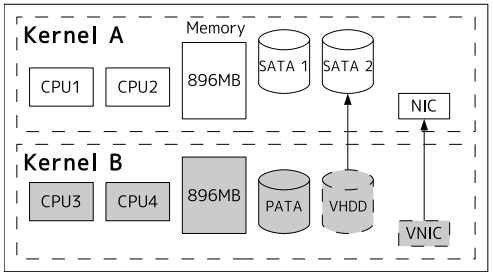

図 4 カーネル間の資源割り当て

6. 評　　価

この節では，仮想デバイス機能を加えた SHIMOS 機構の Linux 2.6.26 への実装に対して行った評価について述べる．評価に使用したマシンの構成は，**表 1** に示す通りである．

以下の実験では，SHIMOS および仮想マシンモニタを用いて，二つのカーネルをこのマシン上で同時に動作させ，それらのカーネルの上でベンチマークプログラムを実行するということを行った．比較に使用した仮想マシンモニタの種類及びバージョンは，Xen 3.2 と KVM 62 である．なお，Xen のドライバドメイン (Dom0) としては，OpenSUSE プロジェクトによる Linux 2.6.26 カーネルを用い，それ以外のカーネルは Linux 2.6.26 のバニラカーネルを用い，カーネルの設定をなるべく共通になるようにした．

カーネルごとの資源割り当てを**図 4** に示す．各カーネルには，896 MB のメモリと 2 つの CPU コアを割り当て，片方のカーネルに SATA コントローラ，もう一方のカーネルに PATA コントローラを割り当てた．さらに，SATA HDD のうち一つを，PATA カーネルから使えるように仮想ブロックデバイスを経由して使用可能にした．また，一方にネットワークコントローラ (e1000) を割り当て，他方は仮想ネットワークを介してネットワークを使え

Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

表 2 The ping による RTT 計測結果

	Native	SHIMOS	Xen	KVM
ping RTT	5 μs	10 μs	66 μs	4043 μs

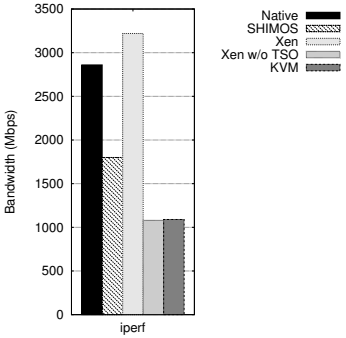

図 5 iperf によるカーネル間のバンド幅

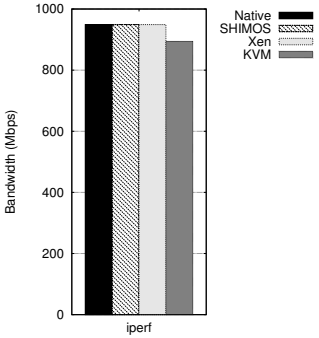

図 6 仮想デバイスによるネットワーク性能に対する影響

るようにした．

さらに，比較のためにネイティブ Linux カーネルの値として，2 カーネル分の資源 (4 CPU コア, 1792MB) を与えたカーネルで，ベンチマークを実行した場合の結果も示す．

6.1 マイクロベンチマーク

6.1.1 ネットワークデバイス

最初に，仮想ネットワークデバイスを用いた，カーネル間通信の性能の評価を行った．評価には，RTT の測定として *ping* コマンド，バンド幅の測定として *iperf*[22] を用いた．結果をそれぞれ**表 2** および**図 5** に示す．なお，Native の場合は，ループバックデバイスの性能を示した．

表 2 から，RTT に関しては，ループバックに次いで，SHIMOS が最も良い値となっていることがわかる．しかし，図 5 にある通り，バンド幅に関しては，Xen よりも悪い結果となっている．この図では，Xen のネットワークデバイスの TSO (TCP Segmentation Offloading) 機能を無効にした場合の結果も示しており，SHIMOS はこれより良い結果となっている．このことから，SHIMOS が Xen よりも悪い性能となった原因は，ネットワークデバイスの最適化の問題であると考えられる．

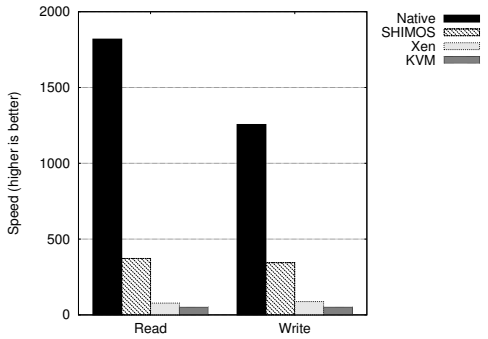

図 7 仮想ブロックデバイスによる RAM ディスクへの読み書き性能

次に，仮想デバイスおよびカーネル間通信機構の実装による，実際のネットワーク性能に対する影響の評価を行った．評価マシン上のカーネルから，外部のサーバー (CoreDuo 2.16GHz, メモリ 2048MB, e1000 NIC) に対して，*iperf* を用いてバンド幅の測定を行った．このとき，SHIMOS では，仮想ネットワークデバイスを経由して，外部のサーバーと通信を行った．結果を**図 6** に示す．この結果，SHIMOS と Xen はネットワーク性能の低下が見られなかったが，KVM では約 5.8% のバンド幅の減少となった．

6.1.2 ブロックデバイス

ブロックデバイスの性能を評価するために，一つのカーネル上の RAM ディスクに対し，他カーネルから仮想ブロックデバイスを介して読み書きを行う実験を行った．なお，仮想マシンの場合は，Dom0 もしくはホストカーネルの RAM ディスクに対する読み書きの性能としている．メモリに対するアクセスは，ディスクに対するアクセスより十分高速であるので，この計測はブロックデバイスの潜在的な速度を測定できる．実験には，ディスクキャッシュの効果を除くために `O_DIRECT` オプションにより `open` したディスクに，4KB 単位で `read` もしくは `write` システムコールを発行し，2GB のデータを読み書きする時間を測定した．結果を，データアクセスの速度にしたものを**図 7** に示す．この図では，SHIMOS の仮想ブロックデバイスは，read では Xen よりも 4.78 倍，KVM より 7.29 倍高速となり，write では Xen より 3.93 倍，KVM より 6.76 倍速くなった．

6.2 アプリケーションベンチマーク

6.2.1 SPECint ベンチマーク

最初に，SPEC CPU2006[23] の int セットを用いて，計算性能のベンチマークを行った．二つ

Vol.2009-ARC-183 No.18
Vol.2009-OS-111 No.18
2009/4/23

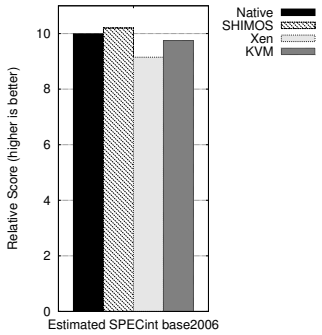

図 8 SPECint 2006 ベンチマークの結果 (estimated)

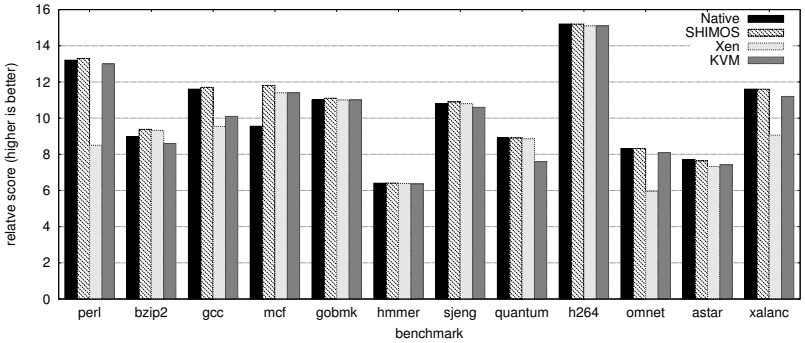

図 9 SPECint 2006 各サブベンチマークの結果 (estimated)

のカーネル上で，一つずつ SPECint ベンチマークのインスタンスを動かし，その “estimated SPECint_base” の値の平均値を**図 8** に示す．また，SPECint の各サブベンチマークの結果を**図 9** に示す．なお，ベンチマークのインスタンスは，2 コアを持つ各カーネルに対して一つずつ動かしたため，CPU はすべて使用していない．これは，CPU 数だけインスタンスを起動するには，評価マシンのメモリが不足したためである．

この結果より，SHIMOS では Xen よりも 11%，KVM よりも 4.5%改善したことがわかる．また，Native の場合よりも，約 2%高い性能が得られている．これは，カーネルを分割してベンチマークを実行することにより，カーネル内での競合が減ったためと考えられる．

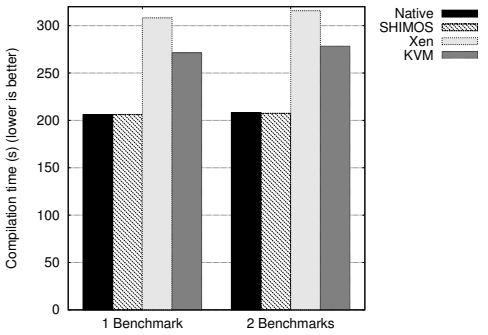

図 10 カーネルコンパイルベンチマークの結果

6.2.2 カーネルコンパイルベンチマーク

計算性能とディスク処理性能を評価するために，Linux カーネルのコンパイルを行った．コンパイル対象のカーネルは Linux 2.6.26 で，デフォルトの設定でコンパイルを行った．まず，コンパイルのジョブを 1 カーネル上で 1 つ実行した時の結果 (“1 Benchmark”) と，2 カーネル上で 1 つずつ同時に実行した時の結果の平均 (“2 Benchmarks”) を**図 10** に示す．なお，コンパイルは，図 4 に示したディスクのうち，SATA2 （“1 Benchmark”），もしくは，SATA1 と 2(“2 Benchmarks”) を用いて行った．SHIMOS の場合は，SATA2 は仮想ブロックデバイスを経由してベンチマークを行った．

この図から，1 つのジョブを実行した場合には，SHIMOS では，仮想デバイスを経由することで性能が Native の場合と比較して 0.1%低下していることがわかる．他方，2 つジョブを実行した場合では，逆に 0.2%性能が向上した．仮想マシンと比較した場合，両方の場合ともに，Xen とは 25%，KVM とは 34%程度高い性能となった．

7. ま と め

本稿では，複数カーネル実行機構である SHIMOS においてカーネル間の通信を実現する，カーネル間通信機構を示した．そして，SHIMOS 機構においてデバイスの共有を実現するために，このカーネル間通信機構を用いた仮想ネットワークデバイスおよび仮想ブロックデバイスの設計と実装を示した．

さらに，Xen や KVM といった仮想マシンと比較して，仮想デバイスを加えた SHIMOS 機構が高い性能を出すことを示した．ネットワークデバイスでは，レイテンシは仮想マシ

ンよりも少なく，バンド幅は，TSO を有効にした Xen を除いて最も高い値を実現した．ブロックデバイスにおいても，Xen より 3.93 倍高い性能を実現した．次に，アプリケーションレベルのベンチマークとして，SPECint2006 ベンチマークの同時実行を行い，ネイティブ環境よりも 2%，Xen よりも 11%，KVM よりも 4.5%改善したことを示した．Linux コンパイルベンチマークでは，複数ジョブの同時実行時にネイティブカーネルよりも 0.2%，KVM よりも 25%，Xen よりも 34%高い性能が得られた．

謝辞 本研究の一部は，科学技術振興機構 戦略的創造研究推進事業 (CREST) (領域名：実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム) 技術課題：「高信頼組込みシングルシステムイメージ OS」による．

参 考 文 献

1) Intel: Intel Core i7 Processor Extreme Edition and Intel Core i7 Processor, `http://download.intel.com/design/processor/datashts/320834.pdf`.
2) Advanced Micro Devices: AMD Opteron Processor Product Data Sheet, `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23932.pdf`.
3) ARM: ARM11 MPCore, `http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html`.
4) Borden, T.L., Henessy, J.P. and W.Rymarczyk, J.: Multiple operating systems on one processor complex, *IBM Systems Journal*, Vol.28 No.1, pp.104–123 (1979).
5) Meyer, R.A. and Seawright, L.H.: A virtual machine time-sharing system, *IBM Systems Journal*, Vol.9 No.3, pp.199–218 (1970).
6) International Business Machines Corporation: *Logical Partitions on System i5: A Guide to Planning and Configuring LPAR with HMC on System i* (2006).
7) Hewlett-Packard Company: *Installing and Managing HP-UX Virtual Partitions (vPars) Ninth Edition* (2006).
8) Adams, K. and Agesen, O.: A comparison of software and hardware techniques for x86 virtualization, *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, ACM, pp.2–13 (2006).
9) 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 ACM Symposium on Operating Systems Principles*, pp.164–177 (2003).
10) Qumranet Inc.: KVM - Kernel-based Virtualization Machine, `http://www.qumranet.com/files/white_papers/KVM_Whitepaper.pdf`.
11) Whitaker, A., Shaw, M. and Gribble, S.D.: Scale and performance in the Denali isolation kernel, *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, New York, NY, USA, ACM, pp.195–209 (2002).
12) Advanced Micro Devices: *AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine Architecture Reference Manual* (2005).
13) Neiger, G., Santoni, A., Leung, F., Rodgers, D. and Uhlig, R.: Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, *Intel Technology Journal*, Vol.10 Issue 3, pp.167–177 (2006).
14) Shimosawa, T., Matsuba, H. and Ishikawa, Y.: Logical Partitioning without Architectural Supports, *IEEE International Computer Software and Applications Conference*, pp.355–364 (2008).
15) 下沢　拓，藤田　肇，石川　裕：マルチコア SH における複数カーネル実行機構の設計と実装，情報処理学会研究報告 (2008-OS-109, SWoPP2008)，pp.25–32 (2008).
16) Fraser, K., Hand, S., Neugebauer, R., Pratt, I., Warfield, A. and Williamson, M.: Safe hardware access with the Xen virtual machine monitor, *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)* (2004).
17) Kim, K., Kim, C., Jung, S.-I., Shin, H.-S. and Kim, J.-S.: Inter-domain socket communications supporting high performance and full binary compatibility on Xen, *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ACM, pp.11–20 (2008).
18) Ram, K.K., Santos, J.R., Turner, Y., Cox, A.L. and Rixner, S.: Achieving 10 Gb/s using Safe and Transparent Network Interface Virtualization, *VEE '09: Proceedings of the 5th ACM/USENIX international conference on Virtual execution environments*, pp.61–70 (2009).
19) PCI-SIG: I/O Virtualization, `http://www.pcisig.com/specifications/iov/`.
20) Abramson, D., Jackson, J., Muthrasanallur, S., Neiger, G., Regnier, G., Sankaran, R., Schoinas, I., Uhlig, R., Vembu, B. and Wiegert, J.: Intel Virtualization Technology for Directed I/O, *Intel Technology Journal*, Vol.10 Issue 3, pp.179–192 (2006).
21) Lowell, D.E., Saito, Y. and Samberg, E.J.: Devirtualizable virtual machines enabling general, single-node, online maintenance, *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pp.211–223 (2004).
22) Gates, M., Tirumala, A., Ferguson, J., Dugan, J., Qin, F., Gibbs, K. and Estabrook, J.: iperf Project Page, `http://sourceforge.net/projects/iperf`.
23) Standard Performance Evaluation Corporation: SPEC CINT2006 Benchmarks, `http://www.spec.org/cpu2006/CINT2006/`.