InfiniBand を PCI パススルーで用いる仮想化 HPC クラスタの性能評価

高野了成[†] 池上努[†]
広渕崇宏[†] 田中良夫[†]

クラウド資源を活用した高性能計算（HPC）の可能性を探るために、HPC アプリケーションの性能測定を通じて、InfiniBand を PCI パススルー経由で利用した仮想化 HPC クラスタの性能を評価した。ハイブリッド並列アプリケーションを 16 ノード上で実行した結果、MPI 通信のスループットに対する PCI パススルーの大きな効果が確認でき、粗粒度を前提としたアルゴリズムであれば物理計算機に匹敵する並列化効率が得られることを確認した。これより HPC 向け Infrastructure as a Service (IaaS) を提供するのに十分に実用的な性能を得られるという見込みを得た。

Performance evaluation of a virtualized HPC cluster equipped with PCI passthrough InfiniBand devices

Ryousei Takano ,[†] Tsutomu Ikegami ,[†] Takahiro Hirofuchi [†]
and Yoshio Tanaka [†]

The feasibility of the cloud computing in the field of high performance computing (HPC) is assessed by measuring the performance of HPC applications on a virtualized cluster system equipped with PCI passthrough InfiniBand devices. We evaluate some hybrid parallel applications on 16 compute nodes. The result shows PCI passthrough produces great improvement for MPI communication throughput, and the parallel efficiency of coarse-grained parallel applications is comparable to the real machines. This paper leads to a positive prospect that Infrastructure as a Service (IaaS) for HPC users is feasible.

1. はじめに

クラウドコンピューティングは、計算資源を抽象化して運用する手段として近年その利用が拡大している。中でも、計算機のコモディティ化の進展を背景に、計算機ハードウェアそのものを仮想化する Infrastructure as a Service (IaaS) が実用的なサービスとして提供されるようになった。特に、クラウドコンピューティングに対する高性能計算 (High Performance Computing, HPC) の高い需要を受け、Amazon EC2[1] の Cluster Compute Instance や Open Cirrus[2] の HPC オンデマンドサービスなど、HPC を意識した IaaS が現れ始めたことは注目に値する。Amazon EC2 の Cluster Compute Instance では、仮想計算機イメージを物理サーバへ配備するために仮想化技術が用いられているが、サーバの集約化は行わず、1 台の物理サーバをそのまま仮想化した構成を採用している。

仮想化のオーバヘッドを考えると、HPC アプリケーションを実行するのに仮想計算機を用いるのは得策とは言えず、物理計算機をそのまま使用する方が効率が良い。しかし、HPC 向け計算資源の仮想化はユーザ側にも実行環境整備の省力化の点でメリットが大きい。物理的にはネットワークで接続された複数のスーパーコンピュータにより構成されるインフラを、クラウドコンピューティングのように容易に利用できる仕組みの実現が期待されているが、従来型のアプローチでは利用するサイトごとに HPC アプリケーションを整備していく必要がある。HPC アプリケーションは往々にして実験的なコードを含んでおり、親切なインストーラがいつも利用できるとは限らない。この場合、計算機環境の調査と設定・バイナリの構築・テストデータを用いたバイナリの検証といった一連の作業を、利用するサイト毎に繰り返していくことになる。これは特に巨大アプリケーションでは大変な作業で、利用サイトの追加を阻む大きな要因となっている[3),4)]。ここでもし各サイトで共通の仮想計算機を設定し、計算機環

† 独立行政法人 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)

境の差異を仮想計算機のレベルで吸収できれば、アプリケーションの整備は 1 回で済む。

そこで、我々はユーザの利便性と性能の追求の両立を目指した、HPC 向け IaaS の構築を構想している。まず、ユーザが手元の計算機でアプリケーションを含む仮想計算機イメージを作成、テストする。これを任意のサイトに配備し、ユーザの要求に応じた規模の仮想化 HPC クラスタをオンデマンドに構築する。サイト内のインターコネクトは InfiniBand や 10 Gigabit Ethernet (10 GbE) など、HPC 用途に耐え得る高速なデバイスを前提とする。仮想計算機イメージはインターコネクトへの依存性を排除し、サービスが提供されるサイトで利用可能な最善のインターコネクトを選択し実行できる、性能可搬性を保証する。

我々は先行研究[5] において上記のようなクラウドコンピューティング環境を想定した予備評価を行った。計算ノード単体の計算性能に対する仮想化のオーバヘッドはアプリケーションに依存するものの、5〜15 % 程度に収まった。一方、16 ノードを用いた MPI 通信性能は、最善でも実ハードウェアの半分程度に止まり、インターコネクトに用いた 10 GbE の仮想化オーバヘッドが実用上無視できないことがわかった。高速インターコネクトの仮想化オーバヘッドを極力回避するためには、ゲスト OS から物理デバイスに対して直接入出力処理を実行できる、PCI パススルーを用いることが現実的な解であると考える。しかし、PCI パススルーを用いた仮想化環境での HPC アプリケーション実行の検討はまだ十分に行われていない。具体的には、アプリケーションの並列化効率に与える影響、Xen や KVM などの実装方法の違いに起因するレイテンシや挙動の違いを検討する必要がある。本論文では、仮想化 HPC クラスタを構築し、HPC アプリケーションの性能測定を通じて、PCI パススルーを経由した InfiniBand 利用の効果を検証し、今後解決すべき技術的課題を明らかにする。

2. 仮想計算機の HPC 利用

2.1 仮想計算機モニタの概要

仮想計算機を HPC 用途で用いる場合、CPU の仮想化オーバヘッドは無視できるが、メモリおよび入出力（IO）の仮想化オーバヘッドを考慮する必要がある。仮想計算機の実現方式は、ゲスト OS に改変を必要とする準仮想化と、ゲスト OS への改変が不要な完全仮想化に大別できる。本節では、各方式における仮想 CPU のスケジューリング、メモリおよび IO 仮想化について簡単に示す。なお、本論文では準仮想化に対応する仮想計算機モニタ（Virtual Machine Monitor, VMM）として Xen、完全仮想化として KVM を評価対象とする。また、最近の CPU は Intel VT-x など、完全仮想化支援機能を有しており、KVM これを前提に実装されている。

Xen では、VMM 上で動作する仮想計算機のことをドメインと呼び、実計算機へのアクセスや他のドメインを管理する特権的な仮想計算機をドメイン 0 と呼ぶ。仮想 CPU は VMM によってスケジューリングされる。一方、KVM では、VMM はホスト OS である Linux のカーネルモジュールとして実装されており、実計算機へのアクセスはユーザランドの QEMU プロセスを経由して実行される。QEMU プロセス内では、仮想 CPU と 1 対 1 に対応するスレッドが生成され、Linux カーネルのスケジューラによって、通常プロセスと同様にスケジューリングされる。

メモリ管理に関しては、仮想計算機の仮想アドレス（Guest Virtual Addres, GVA）から物理アドレス（Guest Physical Address, GPA）へ、さらに物理計算機の物理アドレス（Host Physical Address, HPA）へと、2 段階のアドレス変換が必要となる。

準仮想化では、ゲスト OS に改変を加えることで、CPU がゲスト OS によって管理されるページテーブルを参照して、GVA から HPA へ直接変換することを可能にする。

一方、完全仮想化では、ゲスト OS に改変を加えることはできないので、ゲスト OS のページテーブルとは独立に、GPA から HPA へ変換するページテーブルを VMM が管理する。準仮想化と比べると、これら 2 つのページテーブルの内容を同期する必要があるので、アドレス変換のオーバヘッドは大きくなる。VMM におけるページテーブルの実現手段として、ソフトウェアによるシャドウページテーブル方式と、ハードウェアによる拡張ページテーブル（Extended Page Table, EPT）方式が存在する。シャドウページテーブル方式では、ページテーブルエントリの生成・更新毎に例外を発生させることで VMM へ遷移する必要がある。また、仮想計算機の切替え毎に TLB の全フラッシュが必要である。これらのオーバヘッドを削減するための CPU 拡張機能として EPT 方式が提案された。CPU は、ゲスト OS が管理するページテーブルと VMM が管理する拡張ページテーブルの双方を参照できるので、例外トラップは不要となる。さらに TLB タグに仮想プロセッサ ID が拡張され、仮想計算機の切替えに伴う TLB 全フラッシュも不要となる。しかし、アドレス変換時のページ検索により多くのメ

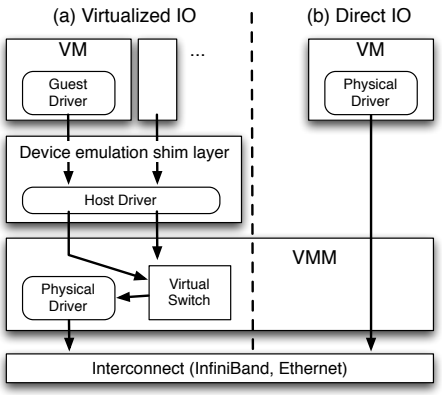


図 1 仮想計算機の IO アーキテクチャ
Fig. 1 IO architecture on a virtual machine.

モリ参照が必要となるため、レイテンシが大きくなる欠点がある。特に TLB ミスが多い場合には顕著な性能低下が現れる[6]。

本論文の実験では、Xen は準仮想化方式、KVM は EPT 方式を利用する。

2.2 仮想計算機の IO アーキテクチャ

仮想計算機における IO アーキテクチャの概要を **図 1** に示す。 ゲスト OS から物理デバイスにアクセスする方式は、仮想化 IO（図 1 左）と直接 IO（図 1 右）に大別できる。前者はスケーラビリティ、後者は性能の点で有利であり、トレードオフの関係にある。なお、以降の説明ではネットワークデバイスを仮定する。

仮想化 IO 方式では、ゲスト OS のゲストドライバが、デバイス模擬層のホストドライバと一組で動作し、仮想スイッチを介してパケットを授受する。実装の詳細には違いがあるが、Xen のスプリットデバイスドライバや、KVM の virtio はこの方式である。複数の仮想計算機から物理デバイスを共有することに理論上の制限はなく、特殊なハードウェア支援も不要である。しかし、オーバヘッドが大きいという問題が存在する。

直接 IO 方式では、VMM をバイパスし、デバイスに直接アクセスできるので、その入出力性能は物理計算機に匹敵する。CPU やチップセットは、Intel VT-d などの仮想化支援機能に対応している必要があり、具体的には PCI パススルーや Single Root IO Virtualization (SR-IOV) などの方式が存在する。データ転送に関しては、GVA から HPA へのアドレス変換をハードウェアにオフロードすることで VMM の介在を不要にできるが、割込みに関しては、IRQ が共有されたり、仮想 CPU が実行可能状態にない可能性もあり得るので、まず VMM が割込みを受け取り、それを仮想計算機に通知する実装となっている。この処理は割込みインジェクションと呼ばれ、レイテンシの増加要因になり得る[7]。割込みインジェクションの実装は、次に示す通り異なる。Xen では、割込みはイベントとして抽象化され、イベントチャネルを経由して VMM からゲスト OS に通知される。KVM では VT-d の仕組みを利用しており、VMM は仮想計算機のコンテキストなどを格納する VMCS (Virtual Machine Control Structure) 領域に発生した割込み原因を示すビットを設定するだけである。後は割込みが可能になった時点で、CPU からゲスト OS に対して割込みが発生する。

2.3 仮想化 IO 方式の性能問題

ここでは仮想化 IO 方式のオーバヘッドについて、10 GbE を用いた 2 つの事例について述べる。

Amazon EC2 Cluster Compute Instance（以下、EC2 と記す）は、8 個の 64 ビット CPU コアと 23 GB のメモリを搭載した仮想計算機であり、インターコネクトとして 10 GbE を提供している。880 インスタンス（7040 CPU コア) 全体での実効性能は 41.82 TFLOPS であり、2010 年 11 月時点の TOP500 ランキングは 231 番目である。その LINPACK 実行効率は理論ピーク性能の約 50%に過ぎず、10 GbE を採用した TOP500 計算機の平均実行効率が 80%程度であることを考えると、これは非常に効率が悪い。その原因は仮想化のオーバヘッド、中でも特にネットワークの準仮想化ドライバにあると推測する。

AIST Green Cloud (AGC) クラスタにおいて MPI 通信性能を計測した。なお、実験環境の詳細は 3 節で述べる。1 GB のメッセージを送信したときのスループットは、物理計算機で 967.3 MB/s であったのに対して、Xen の準仮想化では 304.9 MB/s、完全仮想化では 175.4 MB/s、さらに KVM では 98.2 MB/s と著しい性能低下を観測した。なお、EC2 は Xen の完全仮想化で、かつインターコネクトに対して準仮想化ドライバを採用しているが、AGC の同条件を比較して、391.2 MB/s と倍以上の性能を示している。この原因として、論文 5) で考察したように、ネットワーク周りのチューニングの違いなどが考えられるが、詳細は不明である。

以上の結果を受けて、本論文では性能の追求の観点から、インターコネクトを仮想化 IO 方式ではなく、直接 IO 方式である PCI パススルーを用いて構築した仮想化クラスタの性能を評価する。

3. 実　　験

3.1 実 験 環 境

実験には AIST Green Cloud (AGC) クラスタの一

表 1 AGC クラスタの諸元
Table 1 AGC Cluster specifications.

Node PC	
CPU	Intel Xeon E5540/2.53GHz x2
Chipset	Intel 5520
Memory	48 GB DDR3
InfiniBand	Mellanox ConnectX (MT26428)
Ethernet	Broadcom NetXtreme II (BCM57710)
Switch	
InfiniBand	Mellanox M3601Q
Ethernet	Dell PowerConnect M8024

部の 16 ノードを使用した。AGC の計算ノードは、ブレードサーバ Dell PowerEdge M610 で構成されており、16 ノードが 1 エンクロージャに格納されている。各エンクロージャは InfiniBand QDR (Quad Data Rate) と 10 Gigabit Ethernet (10GbE) のブレードスイッチを持ち、InfiniBand に関しては 16 ノードで 1 つのサブネットを構成している。**表 1** に AGC クラスタの諸元をまとめる。

各ノードは、Quad-core Nehalem (E5540 2.53 GHz) を 2 基搭載し、各 CPU ソケットに 24 GB、計 48 GB のメモリが接続されている。ノード間は InfiniBand および 10 GbE で接続されているが、本実験では InfiniBand だけを利用する。Hyper Threading は無効化した。PCI パススルーに対応するため、ConnectX のファームウェアを 2.6.00 から 2.7.80 へ更新した。

OS は物理計算機 (Bare Metal Machine, BMM)、仮想計算機共に 64 ビット版の Linux ディストリビューション CentOS 5.5 を使用した。

物理計算機には 1 台の仮想計算機を起動し、それぞれに 8 個の CPU コアをすべてと、45 GB のメモリを割り当てた。仮想化環境は Xen 3.4.3 および KVM を用いて構築した。KVM に関しては、CentOS 5.5 のカーネルが 2.6.18 ベースと古く、PCI パススルーに対応していないため、カーネル 2.6.32.28 を用いた。また、QEMU-KVM は git リポジトリのバージョン 0.13.50 を用いた。なお、QEMU 0.13.0-rc1 以降には、後述するベンチマークプログラムから利用する Intel Math Kernel Library (MKL) が、マルチスレッド環境で動作しない問題が存在する。これを回避するパッチ[8] を作成し、適用した。

比較のため、Amazon EC2 Cluster Compute Instance 上でも同じ実験を実施した。各インスタンスには Quad-core Nehalem (X5570 2.93 GHz) 2 基とメモリ 23 GB が割り当てられる。HyperThreading は有効になっており、見掛け上 16 コア存在する。しかし予備的な計算の結果、ノードあたり 16 スレッド起動しても性能が上がらず、むしろ若干劣化することから、実験はノードあたりの最大スレッド数を 8 に制限して実施した。OS は Amazon 側で用意されている CentOS 5.4 をそのまま用いた。仮想化環境は完全仮想化で、ネットワークインタフェースは 10 GbE であるが、PCI パススルーではなく、準仮想化ドライバが用いられている。

3.2 ベンチマークプログラムと予備実験

MPI の基本通信性能は Intel MPI Benchmarks 3.2 を用いて測定した。ノードあたりのメモリ搭載量が増加の傾向にあることから、特にメッセージサイズの大きな所まで (最大 1 GB) 調べた。なお、仮想化環境下において、時間測定の精度が劣化する可能性を考慮し、各イテレーションの実行回数を 100 万回に増やして求めた平均値を結果とした。

HPC アプリケーションでのベンチマーク測定には、NAS Parallel Benchmarks 3.3.1 (NPB) および自作のアプリケーション Bloss を用いた。NPB は MPI と OpenMP のハイブリッド並列性能を測定する Multi-Zone 版（NPB MZ）[9] を選択した。NPB MZ には LU、SP、BT の 3 種類のベンチマークが含まれるが、LU はプロセス数の上限が 16 に制限されることから省いた。SP と BT は共に 3 次元空間における非定常圧縮性 Navier-Stokes 方程式を ADI 法を用いて解くベンチマークであるが、BT の方がメッシュの分割が一様ではなく、負荷分散が難しい。問題サイズは クラス C を選択した。クラス C は全体で 800 MB 程度のメモリを使用する。予備実験の結果より、MPI 1 プロセスあたり 2 本の OpenMP スレッドを割り当てた。したがって 16 ノードで最大 64 プロセスとなる。

Bloss はブロック櫻井・杉浦法を用いた疎行列非線形固有値問題の内部固有値解法アプリケーションである[10),11)]。プログラムは最大 10 GB のメモリを必要とする OpenMP 並列ジョブを MPI で束ねる構成をとっており、MPI 部分は比較的粗粒度の並列となっている。MPI 通信パタンが単純であること (最大 1 GB の集団通信が主体)、また OpenMP 部分で大規模なメモリアクセスが発生することが特徴である。Bloss の実行では、主にメモリ要求量の観点から MPI 1 プロセスあたり 4 本の OpenMP スレッドを割当てた。したがって 16 ノードで最大 32 プロセスとなる。

コンパイラは gcc/gfortran 4.1.2 を用い、最適化オプションは`-O3 -fopenmp` を指定した。Bloss は並列数値計算ライブラリとして、Intel Math Kernel Library (MKL) 11.1 を用いた。MPI 実装は OpenMPI 1.4 を用いた。InfiniBand 使用時に、MPI 集団通信

の性能を向上させるために、実行時オプションとして`--mca mpi_leave_pinned 0` を付加した。実行バイナリは BMM 上で 1 回だけ生成し、これを全環境で流用した。

上記で示したような HPC アプリケーションの性能は、CPU アフィニティの設定に大きく影響を受ける。そこで、仮想 CPU のスケジューリングに関して、CPU アフィニティの効果を調べるために、物理 CPU と仮想 CPU の対応を 1 対 1 に固定することで、Bloss の実行時間に違いが現れるかを計測した。その結果、Xen では、CPU アフィニティ設定の効果が明らかである一方、KVM では、明に CPU アフィニティを設定する必要はなく、むしろ設定が逆効果になる場合も限定的であるが観測された。これを受けて、以下の評価では、Xen に関しては物理 CPU と仮想 CPU の対応を固定化し、KVM に関してはプロセススケジューラの負荷分散に対応付けを任せることで実験を行った。

4. 結果と考察

4.1 MPI: 1 対 1 通信性能

MPI 通信の基本性能として、ノード内およびノード間の 1 対 1 通信性能を調べた。Nehalem は NUMA アーキテクチャなので、リモートメモリへのアクセスはローカルメモリのそれよりもレイテンシが大きい。CPU ソケット内、CPU ソケット間、ノード間、それぞれの片道レイテンシを測定した。ゲスト OS から明に CPU とソケットの関係を指定できるように、この実験に関しては、Xen と KVM 共に仮想 CPU と物理 CPU の対応を固定化した。計測には、IMB PingPong ベンチマークを用いて、メッセージサイズは 0 バイトに設定した。なお、ノード内通信には共有メモリ、ノード間通信には InfiniBand を用いた。

結果を**表 2** に示す。括弧内の数字は、ソケット内通信を基準にした相対値である。BMM ではソケット間通信はソケット内通信の 2.1 倍、ノード間通信は 4.4 倍の時間を要することがわかる。したがって、スレッドを適切なコアに割り当てる負荷分散が重要になる。

ノード内通信に関しては、Xen よりも KVM の方がオーバヘッドが大きい。これはメモリ仮想化方式の違いに由来すると考える。Xen と KVM では、厳密な比較は難しいが、TLB ミスによる EPT 方式のオーバヘッドが露わになった可能性が高い。更なる調査のために、Xen の準仮想化と完全仮想化を比較や、プロファイラによる TLB ミスの解析などが必要である。一方、ノード間通信のオーバヘッドは Xen の方が大きく、レイテンシは BMM や KVM のほぼ 2 倍であった。これは PCI パススルー処理時の割込みインジェクションのオーバヘッドが大きいことが原因と考える。

表 2 片道レイテンシ [usec]
Table 2 One-way latency [usec]

VM type	intra-socket	inter-socket	inter-node
BMM	0.41 (1.00)	0.86 (2.10)	1.79 (4.37)
Xen	0.41 (1.00)	0.83 (2.02)	3.30 (8.05)
KVM	0.54 (1.00)	1.21 (2.24)	1.71 (3.17)

さらにメッセージサイズを最大 1 GB まで変えながら、スループットの変化を計測した。結果を**図 2** に示す。BMM でのピークスループットは、ソケット内で 5.7 GB/s、ソケット間で 3.9 GB/s、ノード間で 2.5 GB/s 弱であった。なお、InfiniBand QDR の理論データレートは 4 GB/s だが、PCI Express 接続で律速され、InfiniBand verbs を直接用いた通信の実測値は 3.4 GB/s であった。これより OpenMPI 由来の性能低下が約 1 GB/s であることがわかる。

ノード内通信に関しては、BMM と Xen の性能は比較的近いが、KVM の性能が悪い。ノード間通信に関しては、PCI パススルーを用いれば BMM と遜色がないスループット性能を得られることがわかる。ただし、Xen の場合、メッセージサイズが 128 バイト以下で性能の低下が見られる。

4.2 HPC アプリケーション: ノード単体の性能

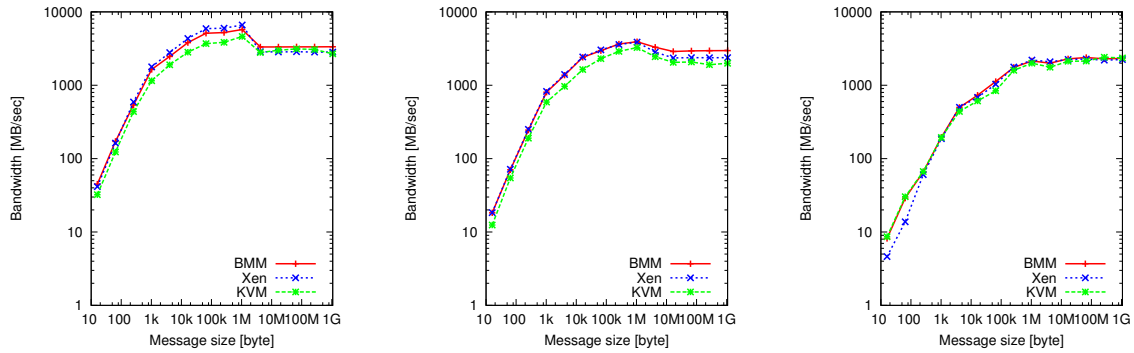
仮想化がノード単体の性能に及ぼす影響について調べる。ここでは主にメモリの仮想化がオーバヘッド要因となり得る。ノード上の 8 CPU コアを全て使用するため、NPB MZ では 4 プロセス、Bloss では 2 プロセスを立ち上げ、実行時間を測定した。結果を**表 3** にまとめる。Xen と KVM はほぼ同じ性能を示し、BMM との比較から仮想化のオーバヘッドはアプリケーションに依存して 5〜15% 程度あることがわかる。

SP-MZ のオーバヘッドは、Xen と KVM それぞれで 16%と 20%であった。BT-MZ のオーバヘッドは、Xen と KVM それぞれで 4%と 10%であった。Bloss のオーバヘッドは、Xen と KVM それぞれで 7%と 9%であった。ノード間通信は行わないので、オーバヘッドは主にメモリ仮想化に由来すると考えられる。したがって、この結果からも Xen の方がメモリ仮想化のオーバヘッドが小さいことがわかる。

EC2 は AGC と比べて 基本 CPU 性能が良いのでもっともよい性能を示しているが、EC2 でも同様のオーバヘッドを生じていると思われる。

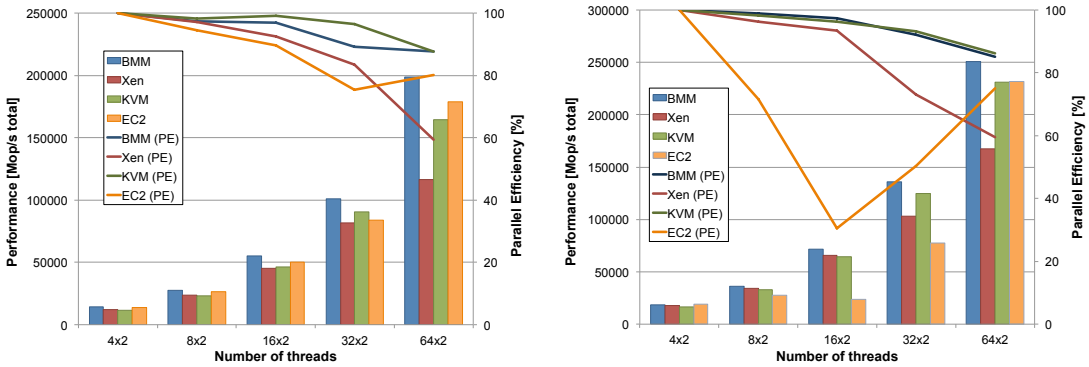
4.3 HPC アプリケーション: 並列性能

ノード数が増えたときの並列性能を調べるために、プロセス数を 4 から 64 まで増やしながら NPB MZ



(a) Intra-socket (b) Inter-socket (c) Inter-node

図 2 MPI 1 対 1 通信のスループット幅
Fig. 2 MPI point-to-point communication throughput.



(a) SP-MZ (b) BT-MZ

図 3 NPB MZ の性能と並列化効率
Fig. 3 Performance and parallel efficiencies of NPB MZ.

表 3 計算ノード単体の性能 (計算時間)
Table 3 A single node performance (wall clock time).

VM type	SP-MZ [sec]	BT-MZ [sec]	Bloss [min]
BMM	86.45	132.06	21.00
Xen	100.12	137.66	22.38
KVM	104.36	144.92	22.98
EC2	88.00	126.01	20.00

を実行した。結果を**図 3** に示す。並列化効率は 1 ノード（4 プロセス）実行時の性能に対して算出した。

AGC 上の実験では、SP-MZ、BT-MZ 共に並列化効率は通信性能を素直に反映した結果となった。アルゴリズム由来の負荷分散の容易さから、BT-MZ よりも SP-MZ の並列化効率が高い。仮想化技術の違いの視点で比較すると、概して、Xen よりも KVM の方が並列化効率が高い。これは NPB MZ は比較的細粒度の通信を行うので、レイテンシ増加の影響が反映されているからと考える。SP-MZ の 16 ノード実行時では、BMM と KVM がそれぞれ 88%、88%の並列化効率を示しているのに対して、Xen では 59%に止まっている。また、BT-MZ でも、BMM と KVM の並列化効率が 85%と 86%であるのに対して、Xen では 59%に止まっている。16 ノード実行時に性能が急落する原因は今後より詳細に解析する必要がある。一方、EC2 の BT-MZ では不可解な挙動を示している。詳細な理由は不明だが、負荷分散に失敗しているものと思われる。

Bloss ではプロセス数を 2 から 32 まで増やしながら並列化効率を測定した。結果を **図 4** に示す。並列化効率は 1 ノード (2 プロセス) 時を基準に算出した。

Bloss には各プロセスで処理が重複する箇所があり、本質的に並列化効率の低下が避けられない。このアルゴリズム由来の並列化効率 (ideal) をグラフ中にあわせて示した。この曲線との差分が、通信由来の並列化効率の低下分となる。ノード数が増えるにしたがい並

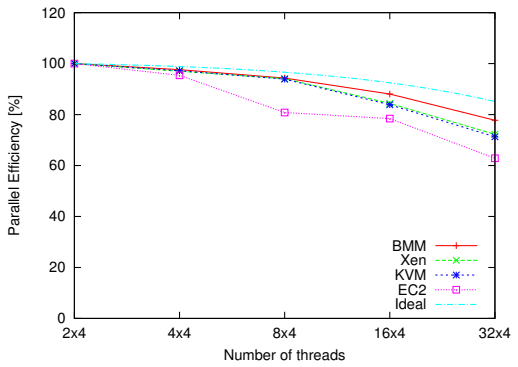


図 4 Bloss の並列化効率
Fig. 4 Parallel efficiencies of Bloss.

列化効率は低下し、16 ノード時で BMM が 78%、Xen は 72%、KVM が 71%となるが、これらの差は数%と小さく、仮想化環境でも十分な並列化効率を達成できていると言える。一方、EC2 は 63%と大きく効率が低下している。Bloss における MPI 通信は NPB MZ と比較すると比較的粗粒度で、メッセージサイズの大きな集団通信が主体である。したがって、レイテンシよりもスループットが性能に大きく貢献する。この結果から、スループットに対する PCI パススルーの効果が確認できる。

5. 関 連 研 究

仮想計算機上の通信性能を改善する試みとして、(1) 仮想化デバイス処理の最適化、(2) VMM バイパス技術の利用、(3) 通信を考慮した仮想計算機スケジューリングの改善の 3 つが考えられる。

TCP/IP 通信性能の評価や改善に関しては、論文 12) など数多くの先行研究が存在する。例えば、KVM の準仮想化ドライバ virtio-net の性能を改善するために、ホストドライバを QEMU プロセス内ではなく、ホスト OS のカーネル内で動かすことで、レイテンシの削減とスループットの向上を実現する vhost-net が提案されている。しかし、10 GbE 以上の環境では、物理計算機と同等の性能は達成できない。

Wang ら[13] は Amazon EC2 上の通信性能を解析し、1 つの物理 CPU コア上で複数の仮想計算機が動作するスモールインスタンス環境における TCP 通信問題について報告している。Kangarlou ら[14] は、その原因として、仮想化デバイスのオーバヘッド以上に、仮想計算機のスケジューリングに起因するラウンドトリップ時間の増加による影響が支配的であると指摘している。そこで、TCP コネクションを仮想スイッチ内で終端し、Ack パケット処理をオフロードすることで、TCP 通信性能を改善する仕組みを提案している。

本庄ら[15] は、Xen 上で MPI プログラムを実行した場合、通信完了待ち時に実行される MPI_Waitall がビジーループする影響で、仮想計算機間のスケジューリングが滞り、結果として性能が大きく低下する問題を報告している。このように仮想 CPU がスピンロックなどでビジーループしている場合に、次の仮想 CPU がスケジューリングされず、実質的に物理 CPU が何も実行できない問題は Lock holder preemption と呼ばれる。この問題に対して Pause-loop exiting などの手法が提案されており、Linux カーネル 2.6.33 や Xen 4.0 以降で対応されている。

本論文では、物理 CPU を複数の仮想計算機から共有しないことを前提としているので、上記の問題は発生しない。また、仮想スイッチ自体のオーバヘッドも大きいので、VMM をバイパスする直接 IO 技術を積極的に活用することを考えている。

準仮想化ドライバの考えを応用することで、VT-d などのハードウェア支援を必要としない VMM バイパス機構も提案されている[16]。ただし、VMM ごとに準仮想化ドライバを実装する必要があり、現時点では Xen 以外で利用できない。

本論文では、PCI パススルーを利用して VMM のバイパスを実現しているが、InfiniBand を用いた場合に、HPC アプリケーションの実行に与える仮想化の影響を詳細に解析した研究はまだ少ない。Nathan ら[17] は、Xen および KVM 環境において、InfiniBand を PCI パススルーで用いた評価も試みているが、4 ノードにおける NPB MPI の評価にとどまっている。また、使用されている KVM 環境に無視できない性能問題が存在するので、仮想化の影響を正確に知ることはできない。本論文では、MPI と OpenMP のハイブリッド並列を対象とした、より現実的な HPC アプリケーションでの評価を行い、その並列化効率を示した。

直接 IO 方式では、ゲスト OS が VMM をバイパスするので、仮想計算機のマイグレーションが困難になる。Zhai ら[18] らはパススルーデバイスと準仮想化デバイスの bonding、および PCI hotplug の併用による問題解決方式を提案している。

6. ま と め

本論文では、InfiniBand を PCI パススルーで用いる仮想化 HPC クラスタを構築し、仮想化が計算機性能に及ぼす影響を、計算性能と MPI 通信性能の観点から評価した。その結果、HPC 向け IaaS を提供するのに十分に実用的な性能を得られるという見込みを得た。

計算性能はアプリケーションの性質に依存するものの、大体 5〜15 % のオーバヘッドと見込まれる。特に Xen の場合は、物理 CPU と仮想 CPU が 1 対 1 に固定するように CPU アフィニティを設定することが重要である。KVM の場合は、Linux カーネルのスケジューラが仮想 CPU をアフィニティを考慮しつつ、適切に負荷分散して動かすため、明に CPU アフィニティを設定する必要はない。MPI 通信性能に対する PCI パススルーの効果は大きく、レイテンシに関しては多少のオーバヘッドはあるものの、スループットに関しては物理計算機と遜色ない性能を得られた。したがって、通信量が多くても、粗粒度を前提としたアルゴリズムであれば十分、仮想化環境での実行に耐えると考える。割り込みインジェクションの高速化やメモリ仮想化の最適化の開発が進めば、さらにオーバヘッドを抑えることが可能になる。

まとめると、仮想 CPU に対する適切なアフィニティの制御、Xen における割込みインジェクションのオーバヘッド、KVM におけるメモリ仮想化のオーバヘッドなどの技術的課題が明らかになった。今後は、これらの問題をさらに詳細に解析し、その根本的な原因を追求する予定である。また、これらは InfiniBand 固有の問題ではない。デバイスドライバの問題から、10 GbE における PCI パススルーの効果は検証できなかったが、今後の課題とする。

また、PCI パススルーはゲスト OS が直接デバイスを制御するため、複数の仮想計算機からの共有やマイグレーション対応に関して問題がある。今後の予定として、前者の問題に対しては SR-IOV デバイスの利用を、後者の問題に対しては Zhai ら[18] が提案しているライブマイグレーション機構の検討を行い、仮想化 HPC クラスタの高度化を進めていきたい。

謝辞 実験環境の整備では (株) 創夢の大田氏にご尽力いただいた。謹んで感謝の意を表する。

参 考 文 献

1) Amazon Elastic Compute Cloud (Amazon EC2): `http://aws.amazon.com/ec2/`.
2) Campbell, R., et al.: Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research, *USENIX HotCloud* (2009).
3) Takemiya, H., et al.: Sustainable Adaptive Grid Supercomputing: Multiscale Simulation of Semiconductor Processing across the Pacific, *ACM/IEEE Conference on Supercomputing* (2006).
4) Ikegami, T., et al.: GridFMO – Quantum Chemistry of Proteins on the Grid, *IEEE/ACM International Conference on Grid Computing (Grid 2007)*, pp. 153–160 (2007).
5) 池上努他: クラウドコンピューティングの性能評価, 情報処理学会研究報告, Vol. 2010-HPC-128, No. 14, pp. 1–6 (2010).
6) Wang, X., et al.: Selective Hardware/Software Memory Virtualization, *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, pp. 217–226 (2011).
7) 渡邉和樹他: Xen における PCI Passthrough の性能評価, 情報処理学会研究報告, Vol. 2010-OS-113, No. 3, pp. 1–8 (2010).
8) Takano, R.: [PATCH] Revert "qemu-kvm: Bring qemu_init_vcpu back home", `http://marc.info/?l=kvm&m=129847188906481`.
9) Jin, H. et al.: Performance characteristics of the multi-zone NAS parallel benchmarks, *Journal of Parallel and Distributed Computing*, Vol. 66, pp. 674–685 (2006).
10) Ikegami, T., et al.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai-Sugiura projection method, *J. Comp. Appl. Math.*, Vol. 233, pp. 1927–1936 (2010).
11) Sakurai, T. and Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration, *J. Comp. Appl. Math.*, Vol. 159, pp. 119–128 (2003).
12) Menon, A. and Zwaenepoel, W.: Optimizing TCP Receive Performance, *USENIX Annual Technical Conference*, pp. 85–98 (2008).
13) Wang, G. and Ng, T. S. E.: The Impact of Virtualization on Network Performance of Amazon EC2 Data Center, *IEEE INFOCOM* (2010).
14) Kangarlou, A., et al.: vSnoop: Improving TCP Throughput in Virtualized Environment via Acknowledgement Offload, *ACM/IEEE Conference on Supercomputing* (2010).
15) 本庄賢光他: VM 上の MPI プログラムの通信オーバヘッドの性能評価, コンピュータシステムシンポジウム 2010, 情報処理学会, pp. 91–100 (2010).
16) Liu, J., et al.: High Performance VMM-Bypass I/O in Virtual Machines, *USENIX Annual Technical Conference*, pp. 29–42 (2006).
17) Regola, N. and Ducom, J.-C.: Recommendations for Virtualization Technologies in High Performance Computing, *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 409–416 (2010).
18) Zhai, E., et al.: Live Migration with Pass-through Device for Linux VM, *Ottawa Linux Symposium*, pp. 261–267 (2008).