Vol.2022-OS-157 No.7
Vol.2022-IOT-59 No.7
2022/9/13

Root権限使用可能なArmスパコン実現に向けた軽量ハイパバイザの設計と実装

森 真誠[1,3,a)] 深井 貴明[1,b)] 山本 啓二[2,c)] 広渕 崇宏[1,d)] 朝香 卓也[3,e)]

概要：近年，爆発的に増加するデータを処理するために，多くのソフトウェアが利用されている．これらを共用 HPC システム上で利用する需要はあるものの，現在の共用 HPC システムはユーザが管理者権限を使用 (以後，root 化) できず，これらソフトウェアの導入に多くの労力を要する．また，システムレベルの最適化も root 化ができないため共用 HPC 環境では困難である．一方で共用 HPC 環境で単に root 化を許可すると，ハードウェアへの恒久的な変更などセキュリティや運用上で問題ある操作を防げない．また，ユーザのジョブがシステムの設定を変更できるため，ジョブの実行毎に設定変更の影響を取り除くためのマシン再起動が必要となるものの，再起動処理には多くの時間を要する．仮想計算機を用いるとこれらを解決できるものの，仮想化処理による性能劣化があり HPC システムには適さない．本研究では，上記課題を解決しユーザの root 化を可能とするシステムを提案する．提案システムは軽量なハイパバイザによって (1) ハードウェアの恒久的な変更を防ぐ機能と (2) ジョブ実行後マシンの再起動なしにシステムを既定の状態へ戻す機能を提供する．本手法を富岳と同系統システムで評価するため，Arm プロセッサで動作する軽量ハイパバイザ MilvusVisor と上記 2 つの機能を設計および実装した．本実装による実験の結果，メモリ性能における性能劣化について 2 %以下で上記機能を実現できることを確認した．

1. はじめに

近年，爆発的に増加するデータを処理するために，多くのソフトウェアが利用されている．これらを共用 HPC システム上で利用する需要はあるものの，現在の共用 HPC システムはユーザが管理者権限を使用 (以後，root 化) できないためユーザランドドライバの使用，割り込みポリシーの変更，バックグラウンドタスクの調節といった性能向上に寄与し得る操作が困難である．またパッケージマネージャが使用できず，プレインストールされていないソフトウェアを使用する際は独自にソースコードからビルドする必要となる場合があるなど多くの労力を要する．一方で共用 HPC 環境で単に root 化を許可すると，セキュリティや運用上で問題ある操作を防げない．また，ユーザがシステムの設定を変更できるため，ユーザの切り替えを行う際に設定変更の影響を取り除くためのマシン再起動が必要となるものの，共用 HPC システムの再起動処理には通常多くの時間を要する．

上記の問題を解決するためには，ユーザに管理者権限を付与し OS の改変を許可しつつも，(1) デバイスに恒久的な影響を与える変更を阻止し，(2) ユーザの利用が終了したあとは時間をかけずにシステムを既定の状態に戻し，なおかつ (3) 性能劣化を抑える必要がある．

ユーザを root 化するとユーザは OS の設定を自由に改変できる． OS はデバイスの管理及び制御を行っているため， root 化されたユーザは OS を介してデバイスの設定領域のアクセスを自由に行えるようになる．デバイスの設定領域の中には Firmware や NIC デバイスの MAC アドレスなど重要なものが含まれ，これらの改変や破壊が行われるとシステム全体の運用が困難になったり，デバイスの故障や不正な動作を引き起こしたりする場合がある．このため root 化されたユーザによるデバイスへの恒久的な影響を与える変更から保護する必要がある．

共用 HPC 環境では，ユーザが目的の処理を終えたあとは，すぐに次のユーザへシステムが割り当てられるようになっている．この際にそのまま次のユーザにシステムを割り当てると，前のユーザの変更が残ってしまい，これが次のユーザが予期していない環境だと正しく処理を続行でき

1 国立研究開発法人 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
2 国立研究開発法人 理化学研究所 計算科学研究センター
RIKEN Center for Computational Science
3 東京都立大学 Tokyo Metropolitan University
a) manami.mori@aist.go.jp
b) takaaki.fukai@aist.go.jp
c) keiji.yamamoto@riken.jp
d) t.hirofuchi@aist.go.jp
e) asaka@tmu.ac.jp

ⓒ 2022 Information Processing Society of Japan

ない．このためユーザの切り替えを行う際に前のユーザが変更したシステムを既定の状態に戻し次のユーザが正常に利用できるようにしなければならない．システムの再起動を行えば実現できるが， HPC システムの再起動には通常多くの時間が必要となり，システム管理者による操作が必要な場合もある．よってユーザの切り替えの度にシステムの再起動を行うのは非効率であり，システムの再起動を行わずに高速でシステムを既定の状態に戻す必要がある．

仮想マシンモニタを使用すれば前述の課題を解決できるものの，デバイスの仮想化などによるオーバーヘッドが大きい．共用 HPC 環境では高い計算性能やレイテンシの少ないノード間通信が実現されており，仮想化によってこの性能や特性を損なうことは致命的である．このため，共用 HPC 環境で仮想マシンモニタの使用は適さない．

本研究では，上記の課題を解決するために準パススルー型仮想マシンモニタアーキテクチャを採用した軽量ハイパバイザを提案する．これを用いて OS からのデバイスに恒久的な影響を与える変更を遮断する．また，ユーザが HPC システムでの作業を終了しようとした場合やシステムの終了及び再起動を試みた際にこれを捕捉し，システムのメモリを書き換え，デバイスの初期化を行うことで予め指定した状態まで戻し，次のユーザに迅速に既定の環境を提供できるようにする．今回はスーパーコンピュータ富岳と同じ CPU である A64FX を搭載した FX700 を対象とした．

以降，2 章で設計，3 章で実装，4 章で評価，5 章で関連研究について述べ，6 章でまとめる．

2. 設計

本章では，提案する共有 HPC における軽量ハイパバイザの設計について述べる．まず，設計方針と前提とする共用 HPC 環境を述べた後，ハイパバイザの設計およびデバイス保護と高速リストア機能の設計について述べる．

2.1 設計方針

ユーザを root 化するとユーザは OS を自由に改変できる．このため OS にデバイス保護などの機能を追加してもユーザが無効化する可能性がある．そこで，提案手法は OS に依存せず動作するハイパバイザとして設計する．

ハイパバイザは性能劣化を避けるために準パススルー型仮想マシンモニタアーキテクチャ [1] を採用する．これは VM を一つしか動作させないことを前提にデバイスの仮想化を行わず，制御が必要な I/O のみ捕捉してハイパバイザで処理するものである．

ハイパバイザではデバイスの保護と，ユーザが利用を終了した際にシステムの状態を既定の状態に戻す（以後，高速リストア）機能を実装する．ハイパバイザはこれらの機能を実現する際にのみ動作し，それ以外では必要最低限の処理しか行わないようにする．

2.2 前提条件

本研究では，共用 HPC としてスーパーコンピュータ富岳に搭載されている A64FX のアーキテクチャである ARMv8.2-A 及び Arm 社の提唱する Server Base System Architecture Level 3[2] を前提としている．

Armv8-A 以降の 64bit アーキテクチャである AArch64 では CPU の動作権限モードが EL3 から EL0 の 4 種類存在する．この内 EL3 はファームウェア向けの動作権限モードであり， EL2 がハイパバイザ向け，EL1 が OS 向けとなっている．提案手法では EL2 で軽量ハイパバイザが動作し，ゲスト OS が EL1 で動作するものとする．

2.3 2 段階ページング

Armv8-A 以降の仮想化支援機能の一つに Stage 2 translation[3] が存在する．これによって，EL1 でのゲスト仮想アドレスからゲスト物理アドレスのアドレス変換に加え，EL2 でのゲスト物理アドレスからホスト物理アドレスへの変換もハードウェアによって行われる．Stage 2 translation では，ゲスト物理アドレスに対応するホスト物理アドレスが存在しない場合や権限上許可されていないアクセスをゲスト OS が行った場合にハイパバイザに処理が遷移する．

準パススルー型仮想マシンモニタアーキテクチャでは動作するゲスト OS は原則として一つであるため， EL1 で動作するゲスト OS からのメモリアクセスでは，通常仮想物理アドレスからホスト物理アドレスは変更せずそのままマップしアクセスできるようにする．一方で，ゲスト OS からのアクセスを捕捉したいメモリ領域のページは EL1 からのアクセス権限を変更しアクセスを制限することで，これらのアクセスを EL2 で動作するハイパバイザで捕捉及び処理している．後述のハイパバイザやデバイスの保護はこの仕組みを用いている．

2.4 ハイパバイザの保護

提案手法では，ハイパバイザは起動時に固定長のメモリを確保してその領域を動作中に使用する．このメモリ領域に対して，悪意のあるゲスト OS が意図的に書き込みを行い，ハイパバイザの破壊を試みる可能性がある．これを防ぐために Stage 2 translation で該当メモリ領域へのマッピングやアクセス権限を駆使しゲスト OS からハイパバイザの確保したメモリ領域へのアクセスを防いでいる．

また，悪意のあるゲスト OS がデバイスによるメモリアクセス (Direct Memory Access, DMA) を用いてハイパバイザの確保したメモリ領域へのアクセスや改変を行わせる可能性がある．これを DMA Attack と呼ぶ．このような試みに対しても，ページテーブルを用いて当該領域にアクセスできないようにするなどして，ハイパバイザを適切に保護する．

2.5 準パススルードライバによるデバイスの保護

準パススルー型仮想マシンモニタアーキテクチャでは通常デバイスは仮想化せず，できるだけゲスト OS からのデバイスへのアクセスを素通りさせる．しかし，ゲスト OS からのデバイスを恒久的に変更しうるアクセスは捕捉し遮断するなど，場合によってはデバイスへのアクセスをフックする必要がある．このようにデバイスの仮想化を行わず，アクセスの一部分のみをフックするようなドライバを準パススルードライバと呼ぶ．

AArch64 では，外部デバイス制御は基本的に Memory Mapped I/O（MMIO）として実現される．このため本手法では準パススルードライバ内で Stage 2 translation を使い，ハイパバイザで事前に保護対象のデバイスの MMIO メモリ領域 のうち恒久的な変更が可能な部分へのゲスト OS からのアクセスを捕捉するようにしておく．これにより，ゲスト OS が該当領域へアクセスしようとした際に EL2 で動作しているハイパバイザに制御が移るようにできる．ハイパバイザで命令の内容を解析しデバイスの恒久的な変更を行おうとしていた場合はこの命令の実行を阻止し，それ以外の場合は命令をそのまま実行した場合と同様のエミュレーションを行う．

2.6 高速リストア

本研究ではユーザがゲスト OS に変更を与えた場合にも，次に利用するユーザには影響を与えないようにする必要がある．また，ユーザがシステムのシャットダウンを行おうとした場合もこれを阻止しなければならない．共用 HPC の再起動には長時間を要する場合も多いため，マシンの再起動を避けつつ，高速に次のユーザに既定の環境を提供する必要がある．このためハイパバイザは，ユーザがゲスト OS の利用を終了したことを検知した場合に，システムの制御を取得しメモリやデバイスの状態を既定の環境まで戻した後に再びゲスト OS に制御を移し，次のユーザが利用可能な状態にする．

3. 実装

本章では，提案する手法を用いた MilvusVisor の実装について述べる．まず， MilvusVisor 全般の実装方法や実装に用いた環境について述べた後，起動方法やハイパバイザの保護，MMIO のアクセス制御及び各デバイスの保護手法，及び高速リストアの実装について説明する．

なお，MilvusVisor のソースコードは公開している [4].

3.1 実装環境

MilvusVisor の開発はフルスクラッチで行った．準パススルー型アーキテクチャのハイパバイザ自体は関連研究でも述べるようにすでに存在しているものの，これらの多くは x86 CPU の仮想化支援機能を前提としている．これらは Armv8-A の仮想化支援機能と設計が大きく異なるため，これらのハイパバイザを改良するよりもフルスクラッチで開発した方が効率が良いと判断した．

MilvusVisor の開発言語には Rust を使用した．これはフルスクラッチで開発する際に，できるだけバグを生み出さないようにする必要があり， Rust の所有権モデルによる高い信頼性を用いてこれの実現を試みた．

3.2 ハイパバイザの起動

提案手法では，ゲスト OS が起動する前にハイパバイザを起動する必要がある．UEFI の仕様では UEFI アプリケーション及び OS ローダは EL2 または EL1 で動作するとされている．この仕様を利用し， MilvusVisor はハイパバイザのローダが EL2 で動作することを前提とし， EL2 権限でハイパバイザの設定を行い，ハイパバイザ本体を読み込み終えた後， UEFI アプリケーション実行環境に関する設定を EL1 に適切にコピーし，権限レベルを EL1 に移行後 UEFI ファームウェアへ制御を返却するように実装した．このようにすることで，MilvusVisor 起動後は無改変の OS およびそのブートローダーが通常通り起動できる．

3.3 ハイパバイザの保護

MilvusVisor ではハイパバイザのロードを行う際に UEFI から固定長の連続したメモリを確保する．確保した領域は UEFI のメモリマップ上の属性を使用不可領域 (EfiUnusableMemory) に変更する．このようにすることでゲスト OS がこのメモリ領域を使用するのを防いだ．

しかしゲスト OS がハイパバイザを攻撃するために意図的にこの領域にアクセスする可能性がある．これを防ぐためにゲスト OS を起動する前にハイパバイザで Stage 2 translation を用いて，ゲスト OS からのアクセスは確保したメモリ領域のうちの 1 ページ (以後，ダミーページ) にすべて変換されるようにした．また MilvusVisor ではダミーページを使用しないようにする．このようにすることで，ゲスト OS からのハイパバイザの確保したメモリ領域へのアクセスはすべてダミーページへのアクセスに変換され，ハイパバイザの動作には一切の影響を与えない．

3.4 DMA Attack からのハイパバイザの保護

DMA Attack からハイパバイザを保護するために，System Memory Management Unit (SMMU) を使用した．これはデバイスからのメモリアクセスに対しても Stage 2 translation などを適用させるデバイスである．MilvusVisor では CPU の Stage 2 translation で使用するページテーブルを SMMU でも用いることで，DMA によるハイパバイザ領域へのアクセスもダミーページへのアクセスに変換している．SMMU はハイパバイザだけでなくゲスト OS も使用する場合がある．このため， MilvusVisor で

はゲスト OS から SMMU への制御は Stage 2 translation に関する設定以外すべてパススルーとし，ゲスト OS が SMMU の設定を有効化する際にそのアクセスを捕捉して上記アドレス変換の設定をハイパバイザで追加した後に SMMU の設定を有効化している．

3.5 MMIO のフック

ゲスト OS からデバイスのレジスタへのアクセスは MMIO を用いて行われる．このため， MilvusVisor では MMIO アクセスのフックは Stage 2 translation を用いて実現する．フックでアクセスを補足した後は，ハイパバイザの準パススルードライバで決められた処理を行う．

Stage 2 translation によるデバイスの MMIO 空間へのアクセスの捕捉は，該当領域のページテーブルのエントリを無効にするのではなく Read/Write の属性を変更することで実現した．これはデバイスの保護を行う際に MMIO への書き込み要求のみを捕捉し処理したい場合があり，この場合には該当領域のページテーブルエントリの Write 属性を無効にすることで，ゲスト OS からの書き込みアクセスのみを捕捉することが可能となり，読み込み要求は捕捉せずに済む．このような実装を行うことで，後述するデバイスドライバの呼び出し回数などの削減が可能となり，処理速度の向上が望める．

ゲスト OS が保護対象の MMIO へのアクセスを行おうとした場合，該当命令は実行されずに MilvusVisor に制御が移行する．MilvusVisor ではゲスト OS がアクセスしようとしたメモリアドレスを取得し，捕捉した命令を解析して命令の種類や書き込む値などを取得する．これらを元に対応した準パススルードライバを呼び出す．ドライバでは渡された情報をもとにゲスト OS の要求の許可，内容の変更，または遮断を決定する．決定を元に実際の MMIO へのアクセスを MilvusVisor が行い，読み込みの場合は対応するレジスタに値を格納する．その後，インデックスレジスタや命令ポインタレジスタの値などを調節した後にゲスト OS に制御を戻す．

3.6 デバイスの保護

今回の実装では，FX700 に搭載されているデバイスのうち，Intel I210 と Mellanox MT27800 の保護を実装した．以下ではそれぞれの保護の実装について述べる．

3.6.1 Intel I210 の保護

MilvusVisor では FX700 に搭載されている Ethernet デバイスである Intel I210 の EEPROM を含む Flash Memory の書き込みからの保護を実装した． EEPROM は不揮発性フラッシュメモリの一種であり，通信に用いる MAC アドレスや重要な設定などを保持している．これに意図しないデータや悪意のあるデータを書き込むことでハードウェアが誤動作したり不正な挙動したりする可能性がある．

Intel I210 では EEPROM や Flash Memory へのアクセスは PCI Configuration Space に記述されている MMIO アドレス空間を通じて以下のレジスタを操作することで実現される．

- **EEWR** EEPROM への書き込みを行うレジスタ
- **FLSWDATA** Flash Memory の書き込みや削除を行うレジスタ
- **iNVM** Flash Memory の先頭領域を EEWR や FLSWDATA を用いずにメモリへの書き込みで更新できる MMIO 領域

今回は一切の書き込みを遮断するため，上記のレジスタへの書き込み要求は全て遮断した．また， EEWR では書き込みが成功したか否かは同レジスタを読み出してステートが DONE であるかで確認するため， MilvusVisor では同レジスタの読み込み要求も捕捉し常にステートを DONE にした値をゲスト OS に渡すようにした．

3.6.2 Mellanox MT27800 の保護

MilvusVisor では FX700 に搭載されている Infiniband デバイスである Mellanox MT27800 のファームウェアアップデートからの保護を実装した．

Mellanox MT27800 はファームウェアアップデートツールとして flint というツールが提供されている [5]．このツールによりデバイスの現在のファームウェアの内容の確認や書き換えが可能となっている．flint は上記の操作を行う際にデバイスの PCI Configuration Space 内に存在するセマフォを取得しようと試みる．

MlivusVisor ではゲスト OS からこのセマフォへのアクセスを遮断する．これにより flint はセマフォの獲得に失敗し，後続のファームウェアの確認，アップデート，及び書き込みが不能となる．

3.7 高速リストアの実装

MilvusVisor ではユーザがゲスト OS の使用を終了しシャットダウンしようとした際にゲスト OS を既定の状態（以後，リストアポイント）に復元をするように実装した．

ゲスト OS はシステムを終了する際に Arm 社の定める Power State Coordination Interface (PSCI) [6] に基づき EL3 で動作するファームウェアに対しシャットダウン要求を行う．MilvusVisor ではこの要求を捕捉及び遮断した後に予め取得していたスナップショットを利用してメモリの内容をリストアポイントの状態に書き戻し，デバイスのリセットを行うようにした．

リストアポイントとして，ゲスト OS により UEFI の ExitBootServices 関数が呼び出され，ゲスト OS に制御が返される瞬間を選択した．ExitBootServices 関数は，UEFI ファームウェアが提供している BootService を終了し， UEFI ファームウェアが管理しているデバイスを開放するための関数であり，通常 OS が起動する際に一度だけ

呼び出される．

リストアポイントをこの瞬間にした理由は，通常 ExitBootServices 関数呼び出し後にゲスト OS がデバイスの初期化などの処理を行うため，この状態に戻せば MilvusVisor によるデバイスの初期化が不要となるためである．また，ExitBootServices 関数を呼び出す際はゲスト OS のメモリ使用量が少なく，スナップショットも小さくできる．

以下に， MilvusVisor でのメモリのスナップショットの作成方法と高速リストアの実行方法について述べる．

3.7.1 スナップショットの作成方法

MilvusVisor のロード時に UEFI が提供するメモリマップを元に，必ずスナップショットを取るメモリ領域と ExitBootServices 関数が呼び出されるまでにゲスト OS からアクセスがあった場合にのみスナップショットを作成するメモリ領域 (以後，オンデマンド領域) を決定する．次にオンデマンド領域を Stage 2 translation を使用しゲスト OS がアクセスしようとした際にハイパバイザで捕捉するように設定する．同時に ExitBootServices 関数のメモリアドレスを取得し，先頭 1 命令を保存した上でハイパバイザ呼び出し命令 (以後，HVC 命令) に変更する．これによりゲスト OS が ExitBootServices 関数を呼び出すために該当アドレスにジャンプした際，ハイパバイザへ制御が移行するようになる．

MilvusVisor のロードが完了し，ゲスト OS が起動した際にオンデマンド領域にアクセスを行った際に MilvusVisor へ制御が移行する．このときアクセスしたアドレスを記録し，スナップショットの対象に加え， Stage 2 translation を変更し，該当アドレスを捕捉しないようにする．これを繰り返し，オンデマンド領域のうちスナップショットを作成すべきメモリ領域を決定していく．

ゲスト OS が ExitBootServices 関数を呼び出した際に HVC 命令により MilvusVisor に制御が移行する．ここで戻り先のアドレスが格納されるリンクレジスタの値を取得し，リストアポイントのアドレスを取得する．これを記録し，先頭 1 命令を保存した上で HVC 命令に変更する．これらの処理を行った後，ExitBootServices 関数の最初の 1 命令を元の命令に復元した上で制御を戻す．その後 ExitBootServices 関数が処理を終了し，リターン命令で呼び出し元に戻った際に HVC 命令で MilvusVisor へ制御が移行する．ここで， ExitBootServices 関数の結果を参照し，成功していた場合は命令を復元した上で，スナップショットを作成すべきメモリ領域の情報を元にメモリの内容を読み取りスナップショットを作成する．その後 Stage 2 translation のページテーブルを元に戻し，デバイスの保護などが行えるようにする．その後にゲスト OS に制御を戻す．

3.7.2 高速リストアの実行

ゲスト OS が PSCI に基づきシャットダウンを行おうと EL3 で動作するファームウェアを呼び出そうとした際に MilvusVisor でこれを捕捉し遮断する．

その後，動作中のすべての CPU コアの制御を取得するために Stage 2 translation のページテーブルを変更し，ゲスト OS からのメモリアクセス全てを捕捉するようにする．また，全 CPU コアに対しメモリキャッシュを破棄するように通知した上で待機状態の CPU コアを稼働状態にさせる．これにより稼働状態になった CPU コアは命令をメモリから読みだそうとし，Stage 2 translation により MilvusVisor に制御が移行する．この時, ExitBootServices 関数を実行していた CPU コア以外は全て停止させる．

動作している CPU コアが 1 つになったことを確認した後，スナップショットを元にメモリの内容を書き戻す．その後，割り込みコントローラやシステムレジスタの設定など最小限の初期化を行った後に EL1 への戻り先をリストアポイントに設定し， EL1 に処理を戻す．これにより，MilvusVisor をロード後に最初に起動したゲスト OS が再び起動処理を行い，次のユーザが利用可能となる．

表 1 FX700 の構成

CPU	A64FX (2.0 GHz, 48 Cores)
メモリ	32 GiB (HBM2, 4 NUMA Nodes)
NIC	Intel I210
Infiniband	Mellanox MT27800
SSD	NVMe 512GB
OS	Red Hat Enterprise Linux release 8.4 (Ootpa)

4. 評価

本章では，MilvusVisor を用いた提案手法の機能面および性能面の評価について述べる．機能面は，ハイパバイザの保護，デバイスの恒久的な変更からの保護，及び高速リストアの機能評価を行った．また，性能面は CPU, メモリ，および Infiniband 通信における性能評価を行った．なお MilvusVisor のバージョン 1.0.0 相当で評価を行った．

評価に使用した FX700 の構成を**表 1** に示す．

4.1 機能評価

この章では， 本研究で提案した手法が MilvusVisor で想定通り動作するか否かを実験で確認した結果を述べる．

4.1.1 ハイパバイザへの書き込みからの保護

この実験では，MilvusVisor が起動時に確保したメモリ領域へ書き込みを行う UEFI Application を作成し，高速リストアを無効にした MilvusVisor を読み込んだ後に実行した．この結果，MilvusVisor の動作に影響はなかったため，正しく保護できることを確認した．

4.1.2 DMA Attack からの保護

この実験では，NVMe の Identify Command を発行し，結果の格納先を MilvusVisor の確保したメモリ領域にして DMA Attack を行うような UEFI Application を作成

```
$ ethtool -e eno1 length 6
Offset          Values
------          ------
0x0000:         2c d4 44 ce 8b 1a
$ ethtool -E eno1 magic 0x15338086 offset 0x5
    value 0x00 length 1
$ ethtool -e eno1 length 6
Offset          Values
------          ------
0x0000:         2c d4 44 ce 8b 00
```

図 1 ethtool による EEPROM の書き換えとその確認

```
$ ethtool -e eno1 length 6
Offset          Values
------          ------
0x0000:         2c d4 44 ce 8b 1b
$ ethtool -E eno1 magic 0x15338086 offset 0x5
    value 0x00 length 1
$ ethtool -e eno1 length 6
Offset          Values
------          ------
0x0000:         2c d4 44 ce 8b 1b
```

図 2 MilvusVisor による EEPROM 書き込みからの保護

し， MilvusVisor を読み込んだ後に実行した．この結果，MilvusVisor の動作に影響はなかったため，正しく保護できる事を確認した．

4.1.3 ethtool を用いた Intel I210 の EEPROM 書き込みからの保護

ここでは ethtool[7] を用いて， EEPROM への書き込みを試みた．ethtool では EEPROM の内容を表示及び書き込みが可能である．MilvusVisor を起動していない環境にて，EEPROM の先頭 6 Bytes の表示， 5Byte 目に 0 を書き込み，再度先頭 6 Bytes の表示を行った結果を**図 1** に示す．書き込み後，再び値を確認すると書き込まれた値が反映されており，保護されていないことが分かる．

MilvusVisor を起動した後に，同様の手順を行った結果を**図 2** に示す．ethtool では EEPROM への書き込みを EEWR レジスタを介して行っており， EEWR レジスタに値を書き込んだ後に同レジスタの DONE ビットが 1 になるまで待機する．MilvusVisor ではゲスト OS が EEWR レジスタを読み込んだ際に DONE ビットのみを 1 にした値を返すため， ethtool は正常に書き込めたと判断し，正常に終了する．しかし，実際には書き込まれていないため，直後再び EEPROM を読み込むと値が変わってない． これにより， MilvusVisor で Intel I210 の EEPROM の書き込み保護が行われていることを確認した．

```
$ flint -d /dev/mst/mt4119_pciconf0 -i fw-
    ConnectX5.bin burn
-E- Cannot open Device: /dev/mst/
    mt4119_pciconf0. Can not obtain Flash
    semaphore. You can run "flint -
    clear_semaphore -d <device>" to force
    semaphore unlock. See help for details.
$ flint -clear_semaphore -d /dev/mst/
    mt4119_pciconf0
Warning: Taking flash lock even though
    semaphore is set.
-E- Failed to open Device: No such file or
    directory. MFE_CR_ERROR
```

図 3 Flint によるファームウェアアップデート

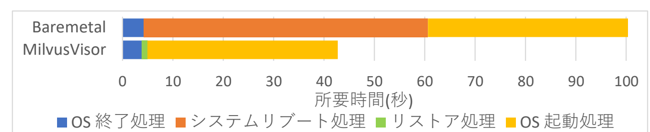

図 4 高速リストアと通常再起動の所要時間差

4.1.4 Flint を用いた Mellanox MT27800 のファームウェアアップデートからの保護

ここでは FX700 の ダウンロードページ [8] のファームウェアアップデート用のバイナリを用いて Mellanox MT27800 のアップデートを試みた．MilvusVisor を読み込んだ状態で flint を実行した結果を**図 3** に示す．

flint の burn コマンドを使用し，ファームウェアアップデートを試みるが， MilvusVisor によりデバイスの Flash 書き込み用のセマフォの取得を阻害され，“Can not obtain Flash semaphore.” と表示されている．また，その際に指示されたコマンドを実行してもエラーとなっている．この結果から， MilvusVisor により Mellanox MT27800 のファームウェアアップデートから保護が行えていることを確認した．

4.1.5 高速リストアと再起動の速度差

MilvusVisor による高速リストアに要する時間と通常の再起動（Baremetal）に要する時間を**図 4** に示す．なお計測時間は reboot コマンドを実行してから，シリアルポートでログインシェルが出力されるまでの時間とした．

通常の再起動ではシステムリブート処理として，ファームウェアの初期化や POST 処理などに 1 分近く時間を要していたが，高速リストアではこのような処理は行わないため，時間を短縮できている．また，ゲスト OS の起動処理も高速リストアでは ExitBootServices を起点に行われるため，時間の短縮が行えている．

全体として，次のユーザが利用可能になるまでの所要時間を高速リストアによって通常の再起動の半分以下に削減できていることを確認した．

ⓒ 2022 Information Processing Society of Japan

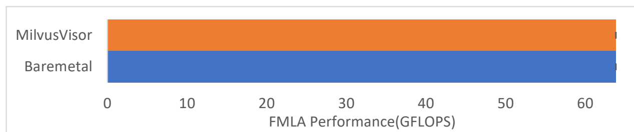

図 5 fmla_a64fx.exe による CPU 処理性能

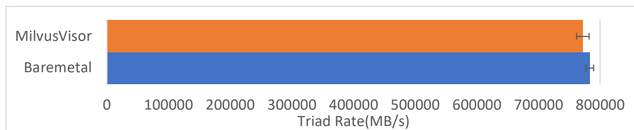

図 6 STREAM による Triad 処理性能

4.2 性能評価

この章では， MilvusVisor を動作させたことによる性能の低下を計測及び評価した．

なお評価には表 1 に示したデバイスを使用し，直接 Linux を動作させた状態（Baremetal）と，MilvusVisor を読み込んだ上で Linux を動作させた状態（MilvusVisor）のそれぞれで計測した．

4.2.1 CPU 性能

行列加算命令の実行時間を用いて CPU の性能を測定する fmla_a64fx.exe[9] を用いて CPU 性能の比較を行った．

Baremetal と，MilvusVisor のそれぞれの状態において，fmla_a64fx.exe を 100 回実行し，得られた FMLA 性能の平均と標準誤差を**図 5** に示す．なお横軸は，行列加算命令における GFLOPS 値である．また，エラーバーは標準誤差を示している．

図 5 に示すように， MilvusVisor による CPU の処理性能への影響はほとんど見られなかった．

4.2.2 メモリアクセス性能

A64FX 向けに最適化された STREAM[10] を用いて，Triad 処理の BestRate の比較を行った．

Baremetal と，MilvusVisor のそれぞれの状態において，STREAM を 20 回実行し，得られた性能の平均と標準誤差を**図 6** に示す．なお横軸は，Triad における スループットの平均値であり，単位は MB/s である．また，エラーバーは標準誤差を示している．

図 6 では， MilvusVisor を起動させた場合に平均で 11097.35(MB/s) ほどの性能の劣化が見られた．平均の性能比として Baremetal に対して MilvusVisor は 98.6% であるため，性能に与える影響は少ないと言える．

4.2.3 Infiniband 性能

Baremetal と，MilvusVisor のそれぞれの状態において，ib_send_bw と ib_send_lat を使用して Mellanox MT27800 のスループットとレイテンシの性能測定をおこなった．ib_send_bw と ib_send_lat は Infiniband 用のパフォーマンステストツールである [11]．なお，評価には RC 通信を用いて行った．

Baremetal と，MilvusVisor のそれぞれの状態において，

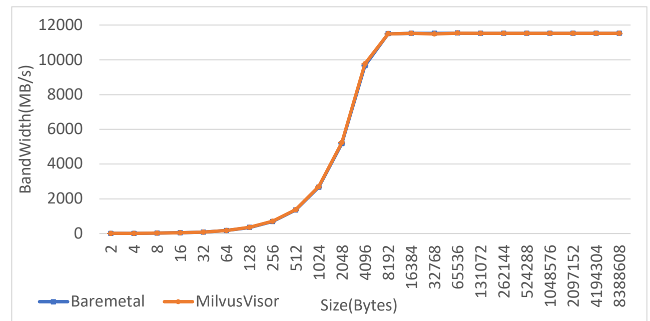

図 7 ib_send_bw による RC 通信におけるスループット

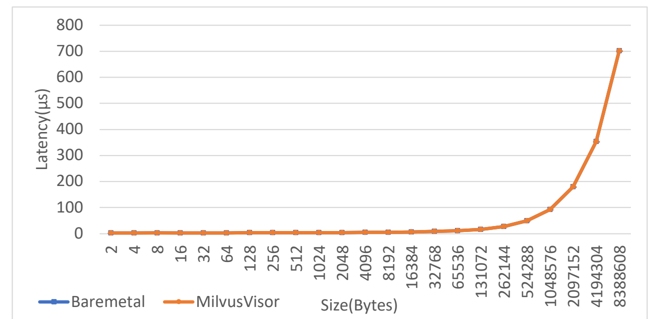

図 8 ib_send_lat による RC 通信におけるレイテンシ

ib_send_bw を 100 回実行し， BW average の平均の比較を行った結果を**図 7** に示す．なお，横軸は一回の送信のメッセージサイズであり単位は Bytes である．また縦軸は通信帯域であり単位は MB/s である．

同様に ib_send_lat を 100 回実行し， BW average の平均の比較を行った結果を**図 8** に示す．横軸は一回の送信のメッセージサイズであり単位は Bytes である．縦軸はレイテンシであり単位は μs である．

スループットは図 7 に示すように MilvusVisor による性能劣化はほとんど見られなかった．レイテンシに関しては，図 8 に示すように MilvusVisor において 0.5μs 程度の増加となっている．これより Infiniband における仮想化のオーバーヘッドは非常に低く抑えられていると言える．

5. 関連研究

準パススルー型仮想マシンモニタアーキテクチャのハイパバイザとして BitVisor[1] が存在する．BitVisor の実装は x86_64 アーキテクチャの仮想化支援機能である Intel VT や AMD SVM などを前提としているが，本研究では Arm スパコンに対応するため Armv8-A 以降における仮想化支援機能を前提としている点が異なる．

BMCArmor[12] はベアメタルクラウドにおいて悪意のあるユーザによるクラウド上のデバイスの破壊などを準パススルー型仮想マシンモニタによって防ぐ研究である．デバイスの破壊や改ざんを防ぐという点で同じであるが BMCArmor は BitVisor をベースとしているため本研究の対象である Armv8-A アーキテクチャの CPU では動作し

ⓒ 2022 Information Processing Society of Japan

ない．またベアメタルクラウドでは中長期的にユーザが専有し OS のセットアップなどは通常ユーザ自身が行うのに対し，共用 HPC 環境ではジョブの実行時のみユーザが計算ノードを専有し，ジョブ終了後は速やかに次のユーザがジョブの実行が可能でないといけないという点が異なる．

山木田ら [13] の研究では，OS の起動時の実行フェーズを予め保存することで，カーネルパニックなどによる OS 再起動時にこれらの情報をもとに再起動に要する時間を削減する方法を提案している．この手法は OS を用いた手法である一方で，我々の手法は OS に依存しないことを目的とした手法であり OS が改ざんされていても既定の状態にシステムを戻せる点が異なる．

Armv8-A CPU 対応のハイパバイザとして KVM[14] や Xen[15] が挙げられるが，これらにはホスト OS が存在したり，デバイスの仮想化したりしているためオーバーヘッドが大きく，高パーフォマンスが求められる共用 HPC 環境には向かない．

オーバーヘッドが少ない仮想化として Singularity[16] などのコンテナ技術が存在し，デバイスの破壊を防ぎつつユーザに管理者権限を渡すことが可能である．しかしコンテナを用いた手法では OS のカーネルをホストと共用しているため，カーネルモジュールの変更などシステム全体の設定変更が行えない．一方で本研究では OS 全体の管理者権限をユーザに提供するためこのような制限は生じない．

6. まとめ

本論文では，共用 HPC システムにおいてユーザの root 化を行い自由度の高い環境を提供しつつもハードウェアの恒久的な変更などを防ぎ，ジョブの切り替えを迅速に行うために，準パススルー型仮想マシンモニタアーキテクチャによる手法を提案した．実装として MilvusVisor を開発し，ハードウェアの保護及び高速リストアの機能検証及び，CPU 性能，メモリ性能，及び Infiniband 性能の劣化を測定した．実験の結果，搭載されている Intel 社の I210 NIC と Mellanox 社の MT27800 Infiniband のファームウェア書き込みやアップデートをゲスト OS から行えないように防止し，ゲスト OS がシャットダウンを試みた際はこれを捕捉し既定の状態までシステムを戻し，再起動に比べ高速に次のユーザが利用可能な状態にできたことを確認した．また MilvusVisor による性能劣化はほとんど見られなかった．

謝辞　本研究は JSPS 科研費 JP21K17727 の助成を受けたものである．

参考文献

[1] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: A Thin Hypervisor for Enforcing i/o Device Security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, New York, NY, USA, Association for Computing Machinery, p. 121–130 (online), DOI: 10.1145/1508293.1508311 (2009).

[2] Arm Limited: Arm Server Base System Architecture 7.0, , available from ⟨https://developer.arm.com/documentation/den0029/f⟩ (accessed 2022-08-03).

[3] Arm Limited: Learn the architecture - AArch64 virtualization Stage 2 translation, , available from ⟨https://developer.arm.com/documentation/102142/0100/Stage-2-translation⟩ (accessed 2022-08-03).

[4] Mori, M. and Fukai, T.: MilvusVisor - A thin-hypervisor that runs on aarch64 CPUs, , available from ⟨https://github.com/RIKEN-RCCS/MilvusVisor⟩ (accessed 2022-08-03).

[5] NVIDIA Corporation: Mellanox NVIDIA Firmware Tools (MFT), , available from ⟨https://network.nvidia.com/products/adapter-software/firmware-tools/⟩ (accessed 2022-08-03).

[6] Arm Limited: Arm Power State Coordination Interface, , available from ⟨https://developer.arm.com/documentation/den0022/db⟩ (accessed 2022-08-03).

[7] Kubecek, M.: ethtool - utility for controlling network drivers and hardware, , available from ⟨https://mirrors.edge.kernel.org/pub/software/network/ethtool/⟩ (accessed 2022-08-03).

[8] FUJITSU: FUJITSU Supercomputer PRIMEHPC ダウンロード：富士通，，入手先 ⟨https://www.fujitsu.com/jp/products/computing/servers/supercomputer/downloads/⟩（参照 2022-08-05）.

[9] Linford, J. C.: FMLA, , available from ⟨https://gitlab.com/arm-hpc/training/arm-sve-tools/-/tree/master/06_A64FX/01_fmla⟩ (accessed 2022-08-03).

[10] Linford, J. C.: STREAM, , available from ⟨https://gitlab.com/arm-hpc/training/arm-sve-tools/-/tree/master/06_A64FX/02_stream/06_stream_zfill_acle⟩ (accessed 2022-08-03).

[11] NVIDIA Corporation: Perftest Package, , available from ⟨https://support.mellanox.com/s/article/perftest-package⟩ (accessed 2022-08-03).

[12] Fukai, T., Takekoshi, S., Azuma, K., Shinagawa, T. and Kato, K.: BMCArmor: A Hardware Protection Scheme for Bare-Metal Clouds, *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 322–330 (online), DOI: 10.1109/CloudCom.2017.43 (2017).

[13] 山木田和哉，山田浩史，河野健二：起動フェーズの再現性に着目した OS 再起動高速化手法，研究報告システムソフトウェアと オペレーティング・システム（OS）， Vol. 2011, No. 4, pp. 1–11（オンライン），入手先 ⟨https://cir.nii.ac.jp/crid/1571417126986384640⟩ (2011).

[14] KVM: Main Page — KVM,, , available from ⟨https://www.linux-kvm.org/page/Main_Page⟩ (accessed 2022-08-03).

[15] XEN PROJECT: Home - Xen Project, , available from ⟨https://xenproject.org/⟩ (accessed 2022-08-03).

[16] Sylabs™, Inc.: Home — Sylabs, , available from ⟨https://sylabs.io/⟩ (accessed 2022-08-03).

ⓒ 2022 Information Processing Society of Japan