Vol.2019-OS-147 No.1
2019/7/24

物理マシンと協調動作可能なソフトウェアによるPCIeデバイスエミュレーション手法

空閑 洋平[1]　松谷 健史[2]　中村 遼[1]　関谷 勇司[1]

概要：PCI Express (PCIe) は，ネットワークやストレージ，専用アクセラレータ間のインターコネクトとして広く普及した．一方で，PCIe リンク内の通信はハードウェア間で閉じているため，ソフトウェアによる通信内容の観察や再現が難しい．QEMU では，ソフトウェアで PCIe デバイスを作成できるが，作成した PCIe デバイスは QEMU 上でしか動作できず，物理ホストと接続できない．本稿では，PCIe 通信がパケット通信である点に着目し，PCIe 通信に用いられる TLP を IP でカプセル化することで，ソフトウェアとハードウェア間で PCIe 通信する手法を提案する．本提案手法のソフトウェア環境では，Socket を用いて PCIe 通信をパケット処理する．本提案手法によって，ハードウェア間に閉じていた PCIe 通信をソフトウェアを用いてより柔軟に解析・観察・再現可能にし，PCIe デバイスのプロトタイピングの簡易化に貢献する．

1. はじめに

PCI Express (PCIe) は，Point-to-Point で接続された PCIe リンク間で，パケット転送によってデータ通信する．PCIe 通信は，TLP (Transaction Layer Packet) と呼ばれるパケットを用いてデータ転送し，PCIe スイッチを用いることで PCIe トポロジの拡張できる．最近のデータセンタでは，専用アクセラレータの接続インタフェースとして PCIe が採用される [1], [2]．また，CPU やメインメモリをバイパスして，直接デバイス同士がデータ転送する Peer-to-Peer (P2P) 接続の事例が登場している [3], [4]．

本稿では，データセンタにおける PCIe を利用したシステム開発に関する 2 つの問題を解決するためのシステムを検討した．1 つ目は，PCIe デバイスは，ホストの CPU やメモリ，チップセットと連動したエミュレーション環境の構築が難しい点である．PCIe デバイスの開発には，FPGA のようなプログラマブル ASIC や，QEMU のようなソフトウェアによるシミュレーションを利用される [5]．ソフトウェアによる開発は，ハードウェア開発に比べて容易だが，物理デバイスと連携が難しいため実際のデータセンタ環境とは異なる単純な構成での検証のみで使用する．特に専用アクセラレータのプロトタイプデバイスの開発は，しばしば性能がでるように開発する必要があるのでプロトタイプのハードルが高い．

2 つ目は，PCIe プロトコルのデバッグが非常に困難な点である．PCIe 通信は，PCIe デバイスとチップセットに実装された Root Complex 間で通信する．PCIe 通信は，PCIe リンク間で閉じているため，ソフトウェア環境から PCIe リンク上での通信内容を観測できない．また，PCIe には，データの読み書き以外に割り込み，デバイス仮想化，キャッシュ制御といったチップセット側と協調した拡張機能がある．しかし，PCIe のこのような挙動調査は多くの場合チップセットのデータシートに頼る必要があり，実機による PCIe データ通信の読み書きや観察には，専用機器を用いるか，またはチップセットのパフォーマンスカウンタを利用する限定的な観測方法となる [6]．特に，P2P DMA をするような CPU やシステムメモリをバイパスするような PCIe 通信下では，通信性能の計測やボトルネックの特定が困難となる．

表 1　本研究のゴール: ソフトウェアによる PCIe デバイスと物理ハードウェア間での PCIe 通信手法

		PCIe デバイス	
		ソフトウェア	ハードウェア
チップセット	ソフトウェア	QEMU	−
	ハードウェア	SoftPCIe	FPGA/ASIC

本稿では，図 1 に示した通り，QEMU のようなソフトウェア実装の PCIe デバイス実装と物理ハードウェアが協調動作できる PCIe デバイスのエミュレーション手法 SoftPCIe を提案する．本提案手法は，PCIe の通信アーキテクチャがネットワーク通信と同じパケット通信であることに注目した．パケット通信では，通信ヘッダに通信先などの通信情報を記述し，送信したいデータを通信ヘッダでカプセル化する．SoftPCIe では，TLP をネットワー

[1] 東京大学　情報基盤センター
Information Technology Center, The University of Tokyo
[2] 慶應義塾大学　政策・メディア研究科
Graduate School of Media and Governance, Keio University

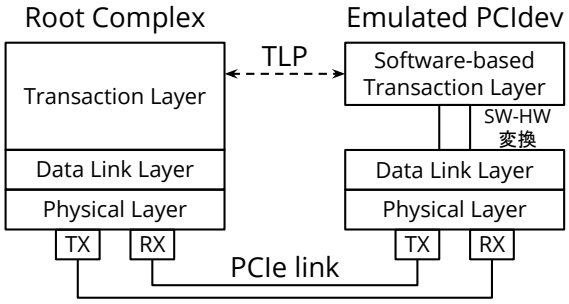


図 1　本提案システムのアプローチ

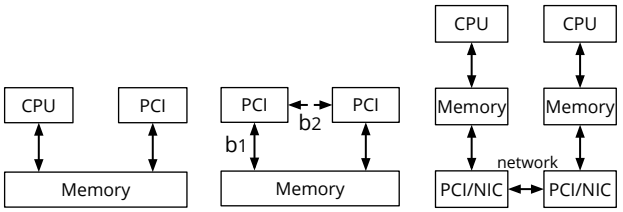
ク通信のヘッダでカプセル化することで，ソフトウェアとハードウェア間で PCIe 通信する．本提案手法により，ネットワークのデバッグ手法をそのまま PCIe データ通信に適応可能になり，PCIe 通信の観察・生成が容易になる．例えば，ネットワークプログラミングでは，tcpdump や wireshark を用いることで通信中のパケットを観測できる．また，SoftPCIe を用いることで，ネットワーププログラミングによって，PCIe パケットを容易に送受信でき，PCIe デバイスのエミュレーションが可能になる．

本稿では，データセンタにおける PCIe デバイスを用いたシステム構成を整理して，本提案手法が適応可能なことを議論した．また，実際にソフトウェアとハードウェア間での PCIe 通信をするための SoftPCIe NIC を設計，実装した．SoftPCIe NIC を用いることで，NVMe などの PCIe デバイスをソフトウェアで実装し，チップセット側機能を実機無しに利用可能になる．今まで検証が困難であったチップセット側機能，性能検証が容易になり，PCIe を中心とした分散システム開発の敷居を下げることに貢献する．

現在の SoftPCIe NIC は，PCIe 仕様のの基本的なデータ通信部分のみをソフトウェアでプログラミング可能になっている．今後は，PCIe のデータリンク層やコンフィグレーション仕様をソフトウェアライブラリとして実装することで，RDMA システムの輻輳制御や，SR-IOV などのハードウェア仮想化をソフトウェアで制御可能にし，今後のアクセラレーションデバイスを中心としたデータセンタシステムの開発への貢献を目指す．

2. 本提案手法のアプローチ

QEMU では，Intel Q35 を基にしたチップセット機能をソフトウェアで実装している [7]．QEMU は，未実装システムのアーキテクチャ検討，デバイスドライバの開発，OS などのソフトウェアとの連携試験と，研究，製品開発に広く利用されている [8], [9]．Intel Q35 は PCIe の基本的な機能に対応しており，未実装の PCIe デバイスを QEMU 上の PCIe デバイスとして実装することで，開発者はデバイスの実機無しにデバイスドライバの開発や，OS 上でのデバイスの取扱方法の検討に利用できる．実際に QEMU リポジトリには，NIC，NVMe など PCIe デバイスが実装されている．最近の例では，NVMe デバイスや，その機能の拡張 NVMe CMB 機能，また NVDIMM などの新しいカテゴリのハードウェアが QEMU 上にソフトウェアとして実装されてデバイス・ドライバ検証に利用されている．



(a) CPU-to-Device　(b) P2P DMA　(c) RDMA

図 2　PCIe デバイスの接続構成

一方で，QEMU のチップセットは PCIe 機能を OS に対してのハードウェア機能の動作のみをエミュレーションしているため，物理ホストと協調動作には利用できない．図 1 は，本提案手法である SoftPCIe のアプローチを表す．PCIe の仕様は，必要な機能を物理層，データリンク層，トランザクション層に分割し，それぞれの層でデータにヘッダをカプセル化する．本提案手法は，ソフトウェアによる PCIe デバイス実装と物理ハードウェア間の PCIe データ通信を変換する専用 NIC (SoftPCIe NIC と呼称) を用いる．SoftPCIe NIC は，PCIe リンクを流れるデータ通信パケットである TLP (Transaction Layer Packet) を Ethernet/IP/UDP パケットにカプセル化，Ethernet ポートを経由して外部ホストにネットワーク通信する．外部ホストは，ホスト自身がエミュレーションされた PCIe デバイスのように振る舞う．外部ホストは，TLP をカプセル化した IP パケットを受け取り，通常の OS のネットワークスタックを経由して，ユーザランドプロセスに TLP を転送する．結果，ハードウェアに閉じてブラックボックス化していた PCIe 通信は，ネットワークと PCIe 間でのパケットレベルで透過的になり，ソフトウェアで PCIe パケットをプログラミング可能になる．TLP ヘッダをソフトウェアで操作することで，CPU や任意の PCIe デバイスに DMA アクセスしてデータを転送が可能になる．

本提案手法の関連研究として ExpEther が挙げられる [10]．ExpEther は，SoftPCIe と同じように TLP を Ethernet ヘッダでカプセル化し，PCIe リンクを延伸できる．ExpEther では，Ethernet フレームは ExpEther 専用ハードウェアで処理される．本提案手法は，通信リンクの片方をソフトウェアにすることで，ソフトウェアで TLP を処理できる．

3. 本提案手法の想定環境

本章では，本提案手法を議論するために，PCIe デバイスを用いたシステム構成を整理した．図 2 は，データセンタにおける PCIe デバイスを用いた一般的な続構成を表す．

(a) は，CPU と PCIe デバイスが連携する一般的な PCIe 接続構成を表す．例えば，サーバ構成では，CPU と，NIC や GPU，ストレージなどをこの構成で接続する．高速データ伝送が必要な PCIe デバイスは，CPU，PCI デバイスがそれぞれ独立してメインメモリにデータを読み書きする．そのために，PCIe デバイスはバスマスタとして振る舞い，メインメモリの指定したアドレス空間にデータを DMA write, DMA read する．CPU はメモリ空間をポーリングや PCI デバイスからの割り込みをトリガとしてデータを読み書きを行う．

(b) は，PCIe デバイスを 1 台のサーバ機に 2 台以上接続して，PCIe デバイス間でデータ転送する構成を表す．PCI デバイス間の連携には，同一のメモリ空間を共有することでデータ転送する方法 (b1) と，それぞれの PCIe デバイスが外部から読み書き可能なメモリを持ち MMIO 経由で直接そのデバイスにデータ転送する方法 (b2) が考えられる．b2 は，PCIe デバイス自身で外部通信用のメモリ空間を持つことで，PCIe デバイスが直接別の PCIe デバイスに DMA する構成です．Nvidia の GPU や NVMe 1.3 の CMB 対応デバイスはメモリ空間を持ち，そのメモリ空間を MMIO でシステムに認識させる．通信 PCIe デバイスは MMIO されているデバイスは，メインメモリであっても PCIe であっても同じく PCIe による DMA することができる．NVMEoF や NVIDIA の GPU Direct がこの接続方法を用いている．

(c) は，2 台のサーバ機を用いた RDMA による分散構成を表す．PCIe デバイスは，Ethernet などのネットワークと PCIe 2 つのインタフェースを持ち，その間を Ethernet などによってデータ転送する．RDMA の構成では，ネットワークを経由して，遠隔マシンのメモリ空間を非同期に読み書きを行う．PCIe にはメモリ読み書きの認証機構がないため，RDMA 先のメモリ空間は NIC とそのドライバで読み書き可能なメモリ空間を管理する．Infiiband や RoCE などの RDMA システムが本構成になる．

本稿では，これらのデータセンタの構成で利用することを想定した，PCIe デバイスのエミュレーションのアーキテクチャを検討した．

4. Ethernet/IP 通信と PCIe 通信の比較

一般的に，Linux のような現在の OS 環境では，ソフトウェアで実装されたネットワークスタックでパケット処理する．一方，PCIe 通信では，PCIe デバイスやチップセットといったハードウェア上でパケット処理する．本章では，ソフトウェアによって PCIe パケットが処理可能か議論するために，それぞれのパケット処理方法を比較検討した．特に，ソフトウェアよるパケット処理で問題になる可能性がある TLP のパケットフォーマット，タイムアウト時間について比較した．

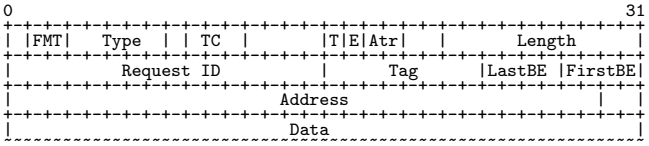


図 3　PCI Express TLP 3DW Header format

4.1 TLP と IP パケットヘッダの比較

はじめに，TLP ヘッダが IP ヘッダのようにソフトウェアで処理可能か検討する．図 3 に 32b アドレッシング時の TLP ヘッダ構造を表す．

TLP ヘッダには，リクエスタ ID というヘッダフィールドがあり，デバイスのバス番号，デバイス番号，ファンクション番号で構成される．デバイスは，リクエスタ ID を参照することで，Root Complex を頂点としたときの PCIe トポロジ上で，対象のデバイスを一意に特定し，対象デバイスに TLP を送信する．TLP を受信したデバイスは，受信パケットの TLP ヘッダのリクエスタ ID によって送信元デバイスを特定する．つまり，リクエスタ ID は，ネットワークにおけるパケット処理における，IP ヘッダの送信先アドレスにあたる．通常，チップセットでは，リクエスタ ID 内のバス番号 0 が Root Complex となる．PCIe デバイスが CPU とデータ通信する場合は，バス番号 0 を指定して TLP を送信することで Root Complex を経由して CPU とデータ通信する．

次に，データ読み書きの要求や確認応答など，送信元デバイスにパケット応答が必要な場合は，コンプリーションを用いて返答する．コンプリーションヘッダ構造は，送信先としてリクエスタ ID，自分の送信元情報として，コンプリータ ID をセットして TLP で応答することでデータ通信する．つまり，コンプリータ ID は，IP ヘッダの送信元 IP アドレスにあたる．

TLP パケットの種類は，Fmt(2bit)，Type(5bit) フィールドによって指定する．Fmt を 00b または 01b と，Type を 00000b を指定することでメモリ読み込み要求，Fmt を 10b または 11b と Type を 00001b を指定することでメモリ書き込み要求，Fmt を 00b または 01b と Type を 00100b によってデバイスの設定を Root Complex に通知するためのコンフィグ読み書き要求パケットを表す．Fmt フィールドの 00b と 01b の違いは，メモリアドレスの長さ 32bit，64bit によって使い分ける．また，MSI-X などのハードウェア割り込みは，特定のメモリアドレスに対して，メモリライトリクエストを実行することで実現されている．

その他にもデータセンタで広く使われている PCIe の機能の多くがトランザクション層の機能によって実現されている．多くのクラウド環境では，各 VM に対して物理 NIC を PCIe の SR-IOV 機能を使って仮想化し，それぞれ PCI Passthrough して接続している．SR-IOV は，コンフィグ

レーションフィールドを複数記載することで，複数の PCIe セットのコンフィグレーションを指定することで実現している．コンフィグレーション通信は，PCIe デバイスがリセットするタイミングで Root Complex と各デバイス間で実行される．また，PCIe はホットスワップに対応しているため，起動中の任意のタイミングで PCIe デバイスの設定を更新できる．

本節では，PCIe の TLP ヘッダは，IP のように TLP ヘッダによって送信元デバイスと送信先デバイスを指定して通信できることを確認した．また，PCIe の割り込みやデバイスの設定についても TLP を用いて実現されているため，ソフトウェアによって TLP ヘッダの構築が可能になった場合，これらのヘッダフィールドを書き換えるのみで制御できることを確認した．

4.2 TLP のタイムアウト時間

PCIe のデータリンク層では，TLP の再送制御を管理する．各 TLP にはデータリンク層でシーケンス番号が設定され，シーケンス番号の抜けによってパケットの消失を発見する．また，PCIe では，許容可能なタイムアウト時間を各デバイスごとでクラス指定できる．一般的にソフトウェアによるパケット処理はハードウェアに比べて処理遅延が大きくなる．そのため，ソフトウェアによるパケット処理の処理遅延によっては，本来ハードウェア処理を想定しているチップセットで TLP 受信がタイムアウトしてしまう可能性がある．

しかし，例えば，データセンタ用 NIC である Intel X520 のタイムアウトは 50us から 50ms 程度になっており，ソフトウェアによるプロセッシングでも十分パケット処理可能であると考えられる．よって，本提案手法であるソフトウェアによる TLP をパケット処理することが十分に可能であると考えられる．

5. システム設計

本稿では，3 章で SoftPCIe の想定構成を整理し，4 章ではソフトウェアによる TLP パケット処理が可能であることを仕様から確認した．本章では，これらの結果を考慮して SoftPCIe NIC を設計した．

5.1 システム概要

PCIe デバイスによる DMA 通信は，ホスト CPU と独立してデバイスが自律動作する．例えば，DMA 通信を使うことで，CPU や OS がクラッシュしていても，PCIe デバイスは自律してシステムメモリを操作できる．そのため，PCIe デバイスを物理デバイスとしてエミュレーションするためには，ホスト CPU と独立した環境で実行する必要がある．ホスト CPU と独立した環境を構築するためには，PCIe 操作を動作させるための別 CPU を用いる方法と，外部ホストから通信を用いて RDMA する方法が考えられる．本システムでは，3 章の想定構成の 1 つである RDMA 構成として動作することを考慮し，外部ホストからの RDMA 方式を選択した．

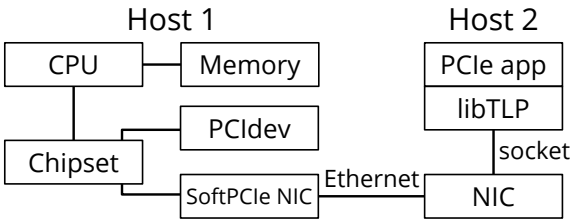


図 4　システム構成

図 4 は，SoftPCIe のシステム構成を表す．本システムでは，2 台のホスト間を Ethernet で接続し，TLP を Ethernet/IP/UDP ヘッダでカプセル化することによって，外部ホストで PCIe デバイスの動作をエミュレーションする．図中の Host 1 はホスト PC，Host 2 は PCIe デバイスのエミュレーションを担当する．Host 1 は，Ethernet パケットと TLP を変換する専用 PCIe アダプタである SoftPCIe NIC を使用して Host 2 と接続する．Host 2 は，市販 Ethernet NIC を利用し，通常の Socket プログラミングで TLP パケット処理を実装，Host 1 と UDP 通信する．

SoftPCIe NIC は，PCIe インタフェースと Ethernet ポートを持ち，ハードウェアとソフトウェア間で TLP パケットを転送する．Host 2 から Host 1 に通信する場合は，受信した UDP パケットを PCIe パケットかチェックし，Ethernet/IP/UDP ヘッダを外し，取り出した TLP を自身の PCIe リンクに送信する．Host 1 から Host 2 に通信する場合は，SoftPCIe NIC 宛に送信された TLP を，ハードウェア上で Ethernet/IP/UDP ヘッダでカプセル化して，Ethernet リンクに Ethernet フレームを送信する．その結果，Host 1 視点では，Host 2 からソフトウェアで送信された TLP は，すべて SoftPCIe NIC が生成・送信した TLP として処理される．

本アーキテクチャによって，3 章で整理した本提案手法の想定環境下での PCIe デバイスをエミュレーションできる．PCIe では，リクエスタ ID に Root Complex を指定し，任意のアドレスを指定することで，メモリマップされたすべてのメモリ空間にアクセス可能になる．メモリマップされたメモリ空間では，MMIO とシステムメモリをアドレスの指定のみで同じアクセス方法で制御できる．そのため，図 2 (a) と (b) の構成は，本システム構成で動作できる．また，SoftPCIe NIC は，Ethernet 部を RDMA の通信部分と見立てることで，SoftPCIe NIC を RDMA NIC として利用できる．SoftPCIe NIC を RDMA NIC として OS に認識さるためのデバイスドライバを利用することで，(c) のプロトタイピングにも利用できる．

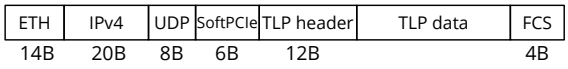


図 5　パケットフォーマット

5.2　SoftPCIe パケットフォーマット

本節では，SoftPCIe NIC のパケットフォーマットを定義する．SoftPCIe NIC では，以下の理由によって，IP ヘッダ，UDP ヘッダ，SoftPCIe 専用ヘッダで TLP をカプセル化する．IP ヘッダは，SoftPCIe NIC を RDMA システムでの利用を想定しているため，データセンタにおける L3 スイッチ対応のために採用した．UDP ヘッダは，エミュレーション側ホストによる TLP の処理がソフトウェアであるため，OS のユーザ空間でのパケット処理しやすくする目的で採用した．ソフトウェアによるパケット処理速度の点においても，エミュレーション側ホストが汎用 NIC を用いるため，UDP パケット処理の NIC Offloading 機能による高速化が期待できる．

また，SoftPCIe NIC では，本システム専用ヘッダ（SoftPCIe ヘッダ）6B を UDP ヘッダと TLP ヘッダの間に挿入する．SoftPCIe ヘッダは，SoftPCIe NIC が生成する Ethernet フレームのサイズを，Ethernet の最小フレームサイズである 64B 以上にするために挿入する．TLP の最小サイズは，32b メモリアドレスのデータペイロード無しのパケットで 12B になる．つまり，TLP 12B をカプセル化した際の Ethernet，IPv4，UDP ヘッダ，SoftPCIe ヘッダ，TLP ヘッダの合計値が 64B になる．将来的には，この専用ヘッダを用いて，TLP の処理時間計測など PCIe パケット処理のデバッグ機能搭載を検討している．

5.3　データ転送性能

本節では，前節で定義した SoftPCIe パケットを用いた際にエミュレーション可能な PCIe のリンク速度，データ転送速度を計算した．パケット通信では，ヘッダサイズと転送データの最大帯域幅はトレードオフの関係にあり，ヘッダサイズを減らすことでデータ転送の利用可能帯域が増加する．

SoftPCIe でエミュレーション可能な PCIe 帯域 B_{pcie} は，使用する Ethernet リンク速度 B_{eth} から，ヘッダオーバヘッドとなる Ethernet ヘッダ 14B，FCS 4B，IP ヘッダ 20B，UDP ヘッダ 8B，SoftPCIe ヘッダ 6B の合計である $IPpkt\_Hdr$ と，Ethernet の仕様上のオーバヘッド ETH_Gap (Preamble 12B，Inter Frame Gap 8B) を除いたものになる．よって，エミュレーション可能な PCIe 通信帯域は，使用する TLP サイズを TLP_Size としたとき，$B_{pcie} = B_{eth}\frac{TLP\_Size}{TLP\_Size+IPpkt\_Hdr+ETH\_Gap}$ となる．例えば，Ethernet に 10Gb/s リンクかつ TLP サイズが 140B(TLP ヘッダ TLP_Hdr 12B+ペイロード TLP_Data 128B) の場合，B_{pcie} は約 6.60Gb/s となる．

表 2　SoftPCIe NIC でエミュレーション可能な PCIe 転送速度

Ethernet 速度	PCIe 転送速度	RDMA データ転送速度
1GE	0.66 Gb/s	0.60 Gb/s
10GE	6.60 Gb/s	6.03 Gb/s
40GE	26.41 Gb/s	24.15 Gb/s
100GE	66.03 Gb/s	60.37 Gb/s

$256B\ MPS, 12B\ TLP\_Hdr, 128B\ TLP\_Data$ の場合

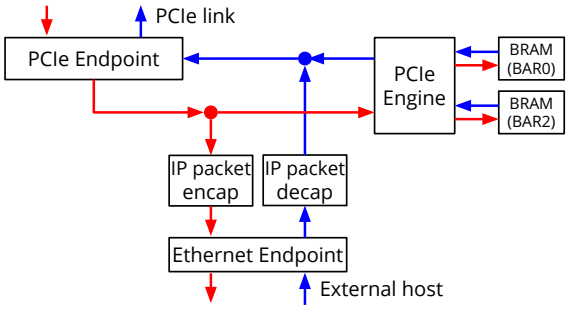


図 6　SoftPCIe NIC の回路構成

注意事項として，本計算結果は，Maximum Payload Size (MPS) を考慮していない．実際の PCIe 通信では，デバイスごとに設定されている Max Payload Size (MPS) 情報を交換して，全ノードから最小の MPS を最大パケット長としてデータ転送する．TLP によるメモリリクエストでは，MPS を超えるサイズを要求，その返答パケットである Completion packet を MPS サイズで分割して連続して返答する．例えば，MPS 128B の PCIe リンク上での 512B のメモリリード要求では，128B コンプリーション 4 パケットで返答する．

次に，実際のユースケースとして，SoftPCIe NIC を用いて RDMA read の転送速度を計算する．RDMA read の転送速度は，B_{pcie} から TLP ヘッダサイズ TLP_Hdr を除いた値になる．使用する Ethernet リンクを 10GE，RDMA read の転送データサイズ TLP_Data を 128B，TLP_Hdr を 12B とした場合，RDMA read の転送速度は，$B_{rdma} = B_{pcie}\frac{TLP\_Data}{TLP\_Data+TLP\_Hdr}$ で約 6.03Gb/s となる．その他の Ethernet 速度の場合を表 2 に示す．

本結果より，例えば，SoftPCIe NIC に Ethernet 10GE を用いて RDMA read する場合，外部ホストの任意のメモリ空間 4GB を約 5.3 秒でフルスキャンできる．

6.　SoftPCIe NIC 実装

本稿では，実際に SoftPCIe NIC のプロトタイプを実装した．図 6 は，SoftPCIe NIC 回路構成を表す．

SoftPCIe NIC のプロトタイプ実装は，Xilinx KC705 FPGA 開発ボードを用いた．FPGA 上に UDP/IP/Ethernet カプセル化回路，TLP パケット処理を行う PCIe Engine を実装した．ホストからの TLP は，PCIe Endpoint IP を経由して直接，IP パケットのカプセル化回路にデータ転送する．そして，Ethernet Endpoint IP 経由で IP パケット

```
1  // Request DMA read
2  rc = sendto(txfd, txd, txdlen, 0,
3          (struct sockaddr *)&to, sizeof(to));
4
5  // Recieved Completion packets
6  struct TLP3DWh *rxp = (struct TLP3DWh *)&rxd;
7  for (;;) {
8          len = recvfrom(rxfd, rxd, sizeof(rxd) - 1, 0,
9                  (struct sockaddr *)&from, &fromlen);
10         printf("Payload␣%d\n", rxp->data);
11 }
```

図 7　DMA read の TLP 送受信部ソースコード抜粋

を外部ホストに送信する．Ethernet Endpoint IP から受信した IP パケットは，IP パケットヘッダを取り除き，取り出した TLP を PCIe Endpoint IP に転送する．

PCIe Engine は，ホストからのメモリ要求に返信するコンプリーション生成の機能を持つ．本 SoftPCIe NIC 実装には，PCIe Engine の有効化・無効化 2 つのモードを搭載した．SoftPCIe の用途では，コンプリーションはソフトウェアで生成するため，本来 NIC ハードウェア上には PCIe Engine は必要ない．しかし，その場合，ソフトウェア上で DMA 通信，ホストからの PIO 通信両方の機能を必ず実装しなければいけない．そこで，PCIe Engine を有効化すると，ソフトウェアでコンプリーションを返信する必要がなくなり，DMA 通信のみを使用するソフトウェアを作る際に有効である．

6.1　ソフトウェア実装

SoftPCIe 環境では，TLP を IP パケットでカプセル化し，Socket(2) を用いて UDP パケットを送受信する．本稿では，実際に実装した SoftPCIe NIC を用いて，PCIe デバイスのエミュレーションをソフトウェアで動作させた．図 7 は，特定の物理アドレスを指定して DMA read する PCIe 実装の TLP 送受信部のソースコードを表す．TLP パケットは，IP パケット同様にヘッダ構造が定義されているため，IP ヘッダと同様の方法で操作できる．構築した TLP は，Socket(2) を用いて UDP パケットの送受信することで SoftPCIe NIC とデータ通信できる．本ソースコードによって，SoftPCIe NIC を用いることで，一般的な C 言語のネットワークプログラミングによって PCIe デバイスをエミュレーションできることを確認した．

7.　まとめ

本稿では，物理ホストに接続可能な，ソフトウェアによる PCIe デバイスエミュレーション環境である SoftPCIe を提案した．はじめに，PCIe デバイスを，ホストと PCIe デバイス間，PCIe デバイス間，RDMA 環境で動作させることを想定し，それぞれの構成で利用できるシステムを設計した．実際に，本稿では，PCIe がパケット通信であることに注目し，ソフトウェアとハードウェア間で PCIe パケット送受信する SoftPCIe NIC を設計，実装した．SoftPCIe を用いることで，ソフトウェアで実装した PCIe デバイスが実機の CPU や GPU，ストレージと直接 PCIe 通信可能になる．PCIe デバイスは UDP を用いたネットワークプログラミングで実装できることから，PCIe デバイスのプロトタイピングが容易であることを確認した．

今後は，PCIe のトランザクション層機能をユーザランドライブラリとして実装する．本ライブラリの API は，QEMU の PCIe 関数を参考にして設計し，QEMU 環境で開発した PCIe デバイスを，SoftPCIe 環境でもそのまま動作できるか検討する．SoftPCIe NIC と本ライブラリを用いることで，未実装の PCIe デバイスをソフトウェアによってエミュレーションし，実際のデータセンタ環境で動作検証が可能になる．

謝辞

本研究は JSPS 科研費 JP18K18043 の助成を受けたものです。

参考文献

[1] Jouppi, N. P. et al.: In-Datacenter Performance Analysis of a Tensor Processing Unit, *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, New York, NY, USA, ACM, pp. 1–12 (2017).

[2] Lee, K., Rao, V. and Arnold, W. C.: Accelerating Facebook's infrastructure with application-specific hardware, https://code.fb.com/data-center-engineering/accelerating-infrastructure/.

[3] NVIDIA: Developing a Linux Kernel Module using GPUDirect RDMA, https://docs.nvidia.com/cuda/gpudirect-rdma/.

[4] Mellanox: HowTo Configure NVMe over Fabrics, https://community.mellanox.com/s/article/howto-configure-nvme-over-fabrics.

[5] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, Berkeley, CA, USA, USENIX Association, pp. 41–41 (2005).

[6] Intel: Performance Counter Monitor, https://software.intel.com/en-us/articles/intel-performance-counter-monitor.

[7] Liguori, A.: QEMU, Features/Q35, https://wiki.qemu.org/Features/Q35.

[8] Enterprise, H. P.: Emulation of Fabric-Attached Memory for The Machine, https://github.com/FabricAttachedMemory/Emulation.

[9] Feldman, S.: Rocker: switchdev prototyping vehicle, http://wiki.netfilter.org/pablo/netdev0.1/papers/Rocker-switchdev-prototyping-vehicle.pdf.

[10] Suzuki, J., Hidaka, Y., Higuchi, J., Yoshikawa, T. and Iwata, A.: ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform, *14th IEEE Symposium on High-Performance Interconnects (HOTI'06)*, pp. 45–51 (2006).