森基金研究成果報告書 2015 年度 (平成 27 年度)

GPU を用いた 低遅延なパケット処理システムの実現

慶應義塾大学 政策・メディア研究科
上野 幸杜

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

近年、データセンターやバックボーンなどの性能とコストの両方が重視される環境において、ルーティングやスイッチングなどの処理を汎用計算機によって行う試みが注目されている [1]。従来、これらの処理は専用に設計された専用機材により行われていた。ハードウェアによるパケット処理を行う専用装置は、想定された用途では高性能であるという利点がある。しかし、ハードウェアは開発コストが高く、設計の変更が難しいため、多様化するネットワーク機器への要求に迅速に対応することが難しい。

一方で、汎用計算機に搭載される CPU はパケット処理のような特定の用途に最適化されていないため、特定の用途に最適化されたハードウェアを性能で上回ることは難しい。しかし、CPU の高速化、PCI-Express の普及によるバスの広帯域化、専用機器と比較して圧倒的な低価格化などにより、汎用計算機を用いた安価で高速なパケット処理システムを構築する手法が模索されている [2][3][4]。

このような手法の 1 つとして、CPU 以外の外部演算装置に対するパケット処理のオフロード (ハードウェアアクセラレーション) がある。特に、GPU は本来画像処理用途のために開発された演算装置だが、現在では汎用的な計算用途への応用 (GPGPU) が可能となり、この仕組みをハードウェアアクセラレーションに応用する手法が模索されている [5][6][7][8]。GPU は CPU-メインメモリ間と比較して内蔵メモリに対する広いバス幅を持ち、かつ多数の演算ユニットによる並列処理が可能であるため、パケット処理に応用した場合、一般に広帯域化が実現できる。

1.2　ハードウェアアクセラレーションにおける問題点

本研究が着目する問題点は、ハードウェアアクセラレーションを用いてパケット処理を行う際に、外部演算装置の持つ演算能力ではなく Network Interface Card(NIC) と外部演算装置間のバスがボトルネックとなる点である。

GPU のような演算装置は多数のコアと広帯域な内蔵メモリを持つため、潜在的なパケット処理能力が高い。しかし、いずれの先行研究も NIC との間でのパケット転送過程において送受信ともにメインメモリを経由しており、遅延・帯域の両面で改善余地があると考えられる。

1.3 本研究の目的と貢献

本研究では、汎用計算機を用いた安価で高速なパケット処理システムを実現することを目的として、パケット処理をハードウェアアクセラレーションした際に内部バスがボトルネックとなることを軽減する手法を示す。

1.2 節で述べた問題点に着目し、汎用計算機上で動作するソフトウェアルータにおいてハードウェアアクセラレーションを行う際に、NIC-GPU 間でのパケットの直接転送を実現する。

本研究で示す手法により、ハードウェアアクセラレーションを用いた際により低遅延・広帯域なパケット転送が可能になるため、これまでと比較してより安価で高速なパケット処理システムを実現することができる。

第2章　アプローチ

本章では、第 1 章の議論を受けて想定環境を整理した上で、本研究が取る手法について述べる。

2.1　想定環境

第 1 章で述べたように、汎用計算機における高性能なパケット処理技術によって恩恵を受けるのは、データセンターやバックボーンなど商用サービスのために構築されるネットワーク設備である。商用サービスにおいては遅延・帯域などが重視される一方で、そのネットワークにかけるコストも重要視される。専用機器と比べ安価な汎用計算機における高性能なパケット処理技術が商用サービスを提供するのに十分な品質となれば、これらの事業者にとっては有益である。

現在の商用サービスにおけるエンドポイントには専らデータセンタに設置される汎用計算機が用いられており、搭載される通信規格として主流となっているのは 1Gbps Ethernet または 10Gbps Ethernet である。そのような商用サービスが提供するサービスは、汎用計算機上で動作する Linux、BSD 等の汎用 OS とその上で動作するアプリケーションによってサービスを行う場合が多い。また、データセンタでは、それらのエンドポイントを収容するスイッチやルータとして、10Gbps Ethernet を多数収容し 40Gbps Ethernet または 100Gbps Ethernet での通信を行うことができる装置が主流となりつつある。上記を踏まえ、本研究では 1Gbps Ethernet または 10Gbps Ethernet によって収容されたエンドポイントからの通信を集約する用途を想定したパケット処理機構を設計・実装する。

一方で、データセンターやバックボーンの中でも、金融サービス High Performance Computing(HPC) 等を対象として特に遅延を重視したシステムが存在する [9][10]。そのような環境においては専用ハードウェア・専用プロトコル等を開発することがコストに見合っているため、本研究の対象とはしない。

2.2　機能要件

本研究では、現在の商用データセンタ及びバックボーンでの使用に耐える性能を、汎用計算機上で動作するソフトウェアルータによって達成し、かつその性能を最大化することを目標とする。

2.1 節で述べた通り、現在のサーバ収容リンクにおける主流は 1Gbps Ethernet または 10Gbps Ethernet である。従って、シングルフローにおいて 10Gbps(14.88Mpps) を超えるトラフィックが流れることは考えにくく、シングルフローにおいてこのトラフィックレートを処理できる性能があれば通常の用途に耐えられるといえる。また、現在の通信プロトコルの主流は TCP であり、シングルフローの処理過程においてパケットの順序が入れ替わると通信性能が劣化する [11]。そのため、シングルフローではパケットの順序が保たれる必要がある。

一方で、データセンタやバックボーンでは、ルータは各エンドポイントからのフローを集約する装置としての用途が考えられる。ソフトウェアルータをこのような用途に使用する場合、汎用計算機においてパケット処理性能に主な影響を与えるのは演算装置であるため、演算装置の数に応じて性能が向上する実装でなければならない。

さらに、**??**節で述べた割り込みの集約技術など、汎用計算機におけるパケット処理には遅延と帯域のトレードオフが存在する。データセンタやバックボーンなど実トラフィックが流れる環境においては、どちらか一方のみを考えるとサービス品質に影響が及ぶため、本研究では転送時の遅延については、最低限の基準を設けた上で帯域を最大化する。遅延の基準については、先行研究における平均的な性能 [5] から、500 マイクロ秒以下とする。

以上を踏まえ、現在の商用データセンタ及びバックボーンにおけるソフトウェアルータの機能要件を以下にまとめる。

- シングルフローにおいて 10Gbps(14.88Mpps) のパケット処理性能を有する
- シングルフローではパケットの転送順序が保たれる
- 演算装置の規模に応じて性能を向上可能なアーキテクチャである
- ポート間転送における遅延が 500 マイクロ秒以下である

2.3 本研究の手法

本節では、第**??**章で述べたパケット処理技術及び、2.2 節で述べた機能要件を踏まえ、本研究の手法について検討する。本研究では、第**??**章で述べた技術のうち、特にハードウェアアクセラレーションに着目する。ハードウェアアクセラレーションの持つ利点として、データリンク層またはネットワーク層の処理だけでなくさらに高いレイヤでの処理についても応用が可能であることが挙げられる。現在主流である CPU によるパケット処理では、先行研究によってボトルネックが PCI Express による内部バスであることが示されてはいるものの、演算性能についてもネットワーク層までの処理でそのほとんどを使い切ってしまっているため、今後の性能改善及び上位レイヤへの応用が難しい。一方で、ハードウェアアクセラレーションを用いた場合、オフロード先のデバイス次第では上位レイヤへの応用が可能であり、性能向上の余地が大きい。本研究では、この点を重視し、パケット処理のオフロードを行うこととする。また、GPU

は、FPGA や ASIC、NPU などと異なり、ソフトウェアによるパケット処理の実装が可能であるため、上位レイヤへの応用時に実装コストが低いという利点がある。そのため、本研究ではオフロード先演算装置として GPU を選択する。ただし、本研究では GPU を用いるものの、オフロード先の演算装置によらず汎用的に適用できるよう設計を工夫し、ハードウェアアクセラレーション自体の有用性を示すことを目標とする。

一方で、パケット処理技術の高速化技術としてハードウェアアクセラレーションを用いた場合、最終的なボトルネックは汎用計算機内部のバスとなることがわかっている [5]。従って、汎用計算機内部のバスを有効活用することが、さらなるパケット処理性能の向上に必要であると考えられる。本研究では、この点を GPU メモリ上の領域をパケットバッファとして用い、NIC-GPU 間のデータ転送を直接行うことで改善する。図 2.1 に既存の GPU を使用したパケット処理フレームワークのデータ転送フローと本研究のデータ転送フローの比較を示す。既存のフレームワークでは、NIC からの DMA 用領域としてメインメモリを用い、GPU でパケット処理を行う際に再度 PCI Express を経由して GPU メモリへのデータ転送を行う。一方で、本研究では NIC から GPU メモリに対して直接 DMA 転送を行うことでメインメモリへのデータ転送を回避する。この手法が、本研究の新規性となる。

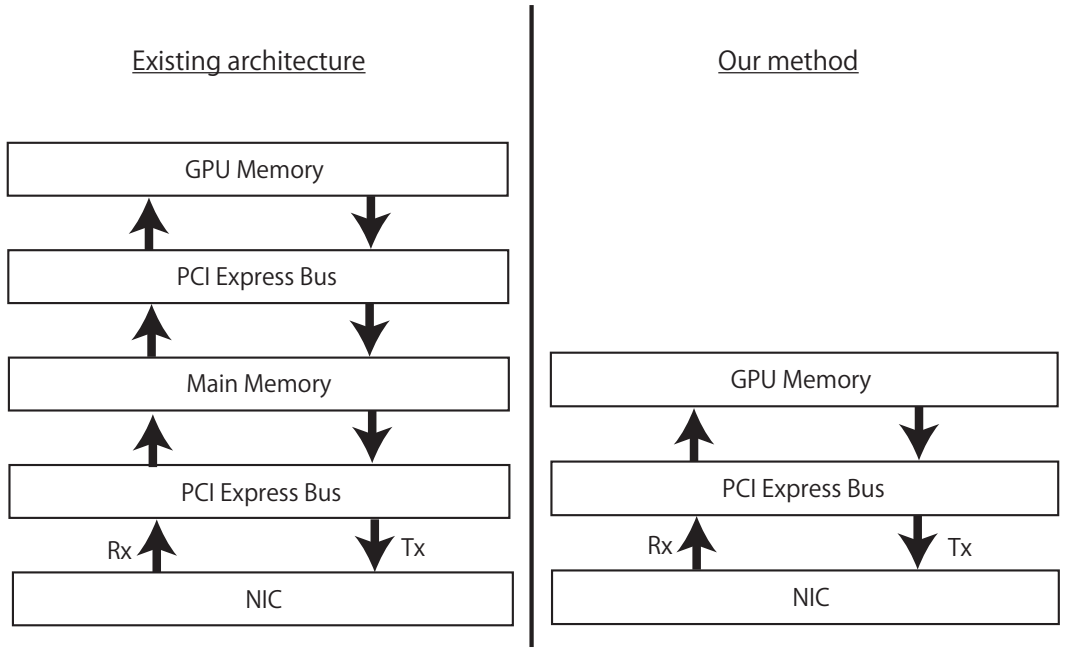

図 2.1: パケット処理フレームワークのデータ転送フローの比較

本研究では、この手法の有用性を検証するため、NIC と GPU 間の直接データ転送を行うパケット処理機構に加え、CPU のみを用いる実装及び既存のフレームワークと同様 GPU を用いるが直接データ転送を行わない実装を行い、性能を比較評価する。それぞれの手法では、適用可能なパケット処理技術を全て適用するが、第??章で述べたパケット処理技術の中には、同時に適用できないものが存在する。特に、ハードウェアアクセラレーションは適用する際の実装上の制約が厳しいため、同時に適用すること

が難しい技術が多い。以下に、それぞれの実装についてまとめる。

- NIC-CPU

  NIC-CPU では、ハードウェアアクセラレーションを用いず、演算装置として CPU のみを用いる。現在の汎用計算機におけるパケット処理実装としては CPU のみを用いたパケット処理フレームワークが主流であり、本手法は最も実装コストが低く、かつ適用可能なパケット処理技術が多い。ただし、この手法では演算装置として CPU のみを用いるため、性能を維持しつつ上位レイヤの機能を実装することが難しい。

- NIC-CPU-GPU

  NIC-CPU-GPU では、ハードウェアアクセラレーションを用いてパケット処理を行う。本手法では、割り込みの処理やデスクリプタリングの操作など逐次処理が中心となる処理は CPU で行い、パケットのヘッダ処理や IP ルックアップなど並列化が可能な部分を GPU にオフロードする。本手法は既存研究と同じくパケットバッファとしてメインメモリを用いるため、パケット受信時は NIC からメインメモリへデータを転送後、GPU に再度データを転送することとなる。送信時は逆に、GPU からメインメモリへデータを転送後、NIC へ送出する。

- NIC-GPU

  NIC-GPU では、基本的な処理は NIC-CPU-GPU と同様となるが、パケットバッファとして GPU メモリを用いる。そのため、内部バスを用いたデータ転送が CPU のみの実装と同じく一往復分となり、内部バスの帯域を節約できる。本手法を確立した既存研究は存在しないものの、ハードウェアアクセラレーションにおいて内部バスがボトルネックとなることが明らかとなっているため、パケット処理性能の改善に有利であると考えられる。

2.4 本章のまとめ

本節では本章の議論を整理し、要点を示す。まず、本研究では商用データセンタ及びバックボーンを想定し、1Gbps Ethernet または 10Gbps Ethernet によって接続されたエンドポイントからの通信を集約する用途を想定したパケット処理フレームワークを設計・実装する。このとき、特に遅延よりも帯域を重視し、金融システムや HPC 等の用途は想定しない。

このような想定環境において、本研究では機能要件を 1) シングルフローにおいて 10Gbps(14.88Mpps) のパケット処理性能を有する 2) シングルフローではパケットの転送順序が保たれる 3) 演算装置の数に応じて内部バスの上限までパケット処理性能がスケールする 4) ポート間転送における遅延が 500 マイクロ秒以下であることとする。

以上の議論を踏まえ、本研究では GPU によるハードウェアアクセレーションを用いた上で NIC-GPU 間の直接データ転送を実現することにより、汎用計算機上で動作す

るパケット処理機構においてより高い性能を達成することを目標とする。そこで本研究では、演算装置として CPU のみを用いる NIC-CPU、既存フレームワークと同様にハードウェアアクセラレーションを用いる NIC-CPU-GPU、NIC-GPU 間の直接データ転送を実現した NIC-GPU の 3 通りの実装を行い、NIC-GPU において性能が改善されることを確認する。上記 3 通りの手法と??節においてまとめた手法の対応を表 2.1 に示す。

表 2.1: 本研究のパケット処理機構と既存手法の対応

手法	NIC-CPU	NIC-CPU-GPU	NIC-GPU
カーネルバイパス	○	○	○
ゼロコピー	○	×	○
割り込みの軽減	○	○	○
プリフェッチ	○	×	×
バッチング	○	○	○
CPU 拡張命令	×	×	×
マルチキュー	○	○	○
マルチスレッド	○	○	○
ヒュージページ	○	○	×
ハードウェア アクセラレーション	×	○	○

第3章　実装

本章では、本研究のパケット処理機構の実装について述べる。

3.1　実装環境

表 3.1 に本研究の実装環境を示す。本研究では、NIC として Intel X520 を用いる。また、GPU として nvidia 社製 quadro K4000 を用い、GPU での処理は nvidia 社製 GPU 向け統合開発環境である CUDA を用いて実装した。

表 3.1: 本研究の実装環境

OS	Debian GNU/Linux 8.2 jessie
カーネル	3.16.0-4-amd64
使用言語	C 言語
コンパイラ	gcc version 4.9.2 及び nvcc version 7.5.17
GPU 開発環境	CUDA version 7.5.18

3.2　各実装の基礎となるパケット処理機構

本研究では、CPU 上のみで動作するパケット処理機構を基礎として実装し、これに改変を加えることで NIC-CPU-GPU 及び NIC-GPU の実装を行った。本節では、基礎となるパケット処理機構の概要を述べる。図 3.1 に基礎となるパケット処理機構の実装の概要を示す。本研究で実装したパケット処理スタックは、階層化された 3 つのモジュールからなり、大部分がユーザスペースで動作する。その理由として、CUDA はユーザスペースでの使用を前提として設計されているため、カーネル内からは GPU 上で動作するプログラムを実行することができない点、Linux におけるヒュージページがユーザスペースから使用することを前提とした実装となっている点、Intel DPDK の例から、ユーザスペースであっても性能面の機能要件を満たす実装を作成可能であることが示されている点が挙げられる。また、各手法ごとに異なる点は、各モジュールの関数を差し替えることで実装した。

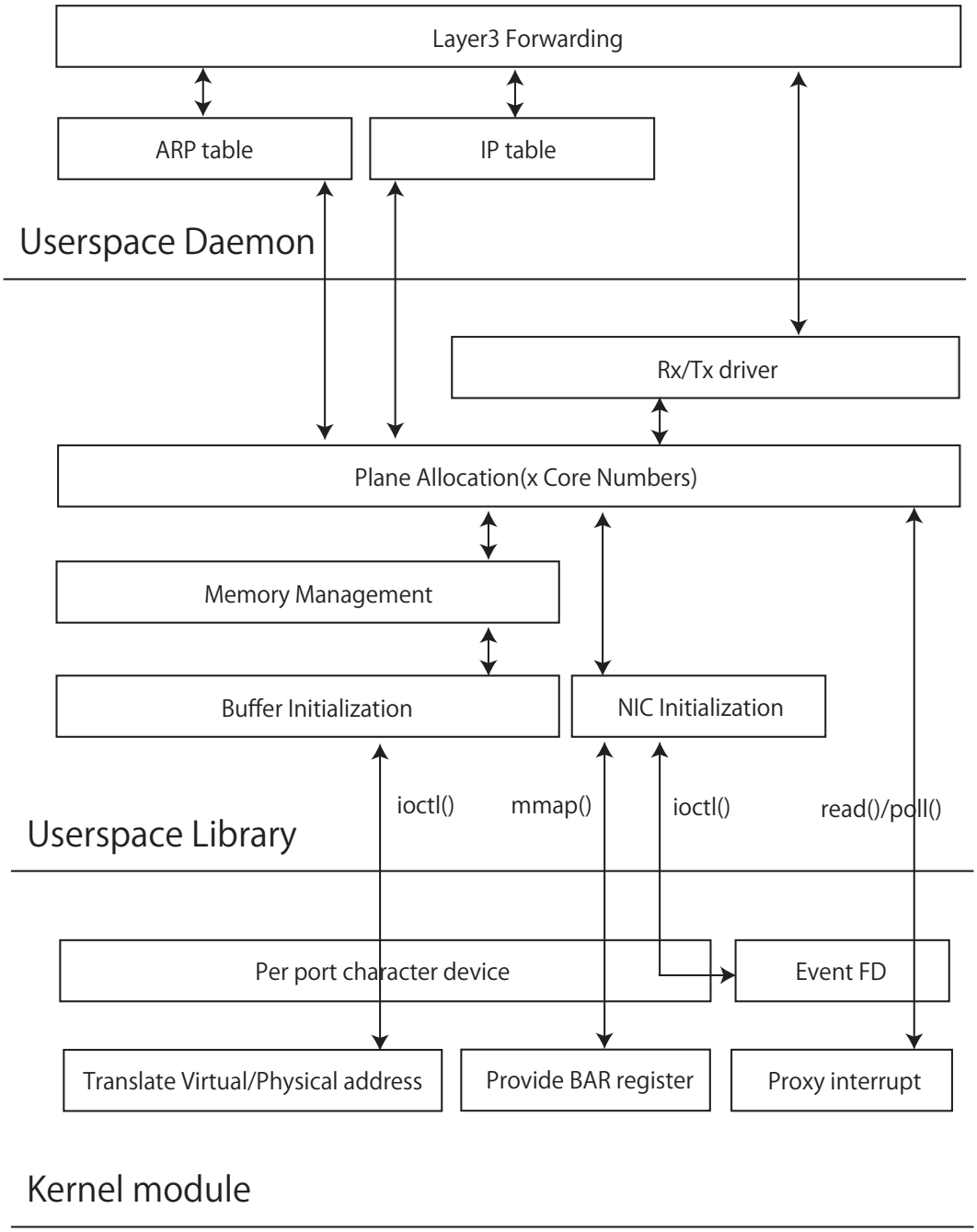

図 3.1: 基礎となるパケット処理機構の実装の概要

3.2.1 カーネルモジュール

本モジュールは、Linux においてカーネル内からのみアクセスできる資源をユーザスペースへ仲介する。具体的には、以下の機能を持つ。

各ポートをキャラクタデバイスとして抽象化

本モジュールはPCIデバイスとしてIntel X520を検知すると、それぞれのポートについて、ハードウェアリセット、MACアドレス等各種パラメータの取得等最低限の初期化操作を行ったのち、それぞれのポートについてキャラクタデバイスを作成する。ユーザスペースのプログラムはこのキャラクタデバイスを通してmmap()システムコールを発行することにより、設定用BARをユーザスペースのメモリ空間にマップすることが可能である。ユーザスペースのプログラムは、マップしたBARを通してNICの送受信設定、割り込みの有効化・無効化、マルチキューの作成等の操作を行う。また、本モジュールはioctl()システムコールを通して、NICの割り込みベクタ要求、リンクアップ及びリンクダウン、情報取得を行う機能を持つ。これにより、ユーザスペースのプログラムはカーネルAPIにアクセスすることなくNICに関するすべての操作を行うことができる。

DMA領域の物理アドレス取得

本モジュールは、ユーザスペースにおいて確保されたメモリ領域の物理アドレスをカーネルAPIを用いて取得し、ユーザスペースへ返す機能を持つ。これにより、ユーザスペースのプログラムは取得したヒュージページの物理アドレスを取得可能となり、取得した物理アドレスを用いてNICにDMA命令を発行することでカーネル内のバッファを経由することなくパケットを送受信することができる。

eventfdによる割り込みの通知

本モジュールは、Linuxにおけるイベント通知APIであるeventfdを用いて、ユーザランドに割り込みを中継する機能を持つ。eventfdは、64bitの値のみ送受信可能なファイルディスクリプタとして機能し、poll()もしくはepoll()システムコール等で待ち受けることが可能である。あらかじめユーザスペースからioctl()システムコールによってeventfdのファイルディスクリプタを受け取り、そのファイルディスクリプタに関するコンテキスト構造体を割り込みハンドラのクッキーとして登録することで、割り込みハンドラからeventfdのファイルディスクリプタを待ち受けるプロセスないしスレッドに対して通知を行うことができる。

3.2.2 ユーザスペースライブラリ

本モジュールは、カーネルモジュールが抽象化したキャラクタデバイスの操作等、NIC制御において必要な機能をAPIとして提供する。また、NICの持つ各資源とCPU側の各資源を対応させ、抽象化する機能を提供する。具体的には以下の機能を持つ。

NIC のキュー初期化

本モジュールは、NIC の BAR を操作することで、NIC の持つマルチキュー機能を初期化・設定する機能を持つ。本実装で使用する Intel X520 はパケットヘッダのハッシュ値によって最大 16 キューにパケットを振り分ける機能を有しており、本モジュールはこれを利用して最大 16 キューを初期化・送受信可能な状態にする。

割り込みの初期化

本モジュールは、割り込み通知に必要な eventfd の作成を行い、ioctl() システムコールを用いてその登録及び割り込みベクタの要求を行う機能を有する。ユーザスペースデーモンはマルチキューの初期化と本機能を合わせて利用することで、それぞれのキューについて別個のファイルディスクリプタを介して割り込み通知を受け取ることができる。

メモリ管理

本モジュールは、パケットバッファ領域及びデスクリプタリング領域をヒュージページ上に確保し、カーネルモジュールより取得した物理アドレス情報と合わせて仮想アドレスと物理アドレスの変換を行う機能を提供する。また、ヒュージページ上の未使用領域をプールし、動的な割り当て及び解放機能を提供する。ユーザスペースデーモンは ARP テーブル及びルーティングテーブルの構築、種々の構造体の保存先としてこの未使用領域を利用することで、TLB キャッシュミスを軽減することができる。

デスクリプタリングの操作

本モジュールは、メインメモリ上に確保されたデスクリプタリングを操作し、パケットの送受信を行う機能を提供する。本実装は受信・送信ともに、指定された数を一度に取り出すことが可能であり、ユーザスペースデーモンがバッチングを適用することを可能にする。

プレーンの作成

本モジュールは、NIC が持つ各資源と CPU 側の各資源を対応させ、抽象化する機能を提供する。本実装では CPU の各コアが受け持つ資源の集合をプレーンと呼ぶ。具体的にプレーンに割り当てられる資源は、各ポートから各コアへ 1 つずつ割り当てられたキュー及びそれに対応する割り込み通知用ファイルディスクリプタ、デスクリプタリング、パケットバッファ用のメモリ領域等である。各コアはプレーン内の資源のみで独立してパケット処理を行うことができるため、他のコアと同期を行う必要がない。そのため、ユーザスペースデーモンは処理コストの高いロック、コア間のデータコピー等を回避することができる。

3.2.3 ユーザスペースデーモン

本モジュールは、ユーザスペースライブラリの各 API を使用してパケットの送受信を実行するとともに、ネットワーク層までのパケット処理を行う。これらの処理は、NIC のキュー数の上限である 16 を上限としたコア数分のスレッドによって並列で行い、コア間の共有資源は存在しない。具体的には、以下の機能を持つ。

ARP/ND テーブル・ルーティングテーブルの構築

本モジュールは、ネットワーク層のパケット処理に必要な ARP/ND テーブル及びルーティングテーブルをヒュージページ上に構築する。ARP/ND テーブルは、16bit 長のハッシュテーブルにより構築し、コリジョンが発生した場合は連鎖法により対応する。また、ルーティングテーブルは DIR24-8 アルゴリズム [12] に手を加え、16bit のテーブルに宛先アドレス長を上限として 8bit のテーブルを連鎖させることで構築する。これらのテーブルの構築は、netlink ソケットを介して通知されるカーネルのエントリ情報に基づく。したがって、テーブルの情報は自動的にカーネルと同期され、既存の ip コマンド等を利用して設定することが可能である。

ネットワーク層の転送処理

本モジュールは、構築した ARP/ND テーブル及びルーティングテーブルに基づき、ネットワーク層の転送処理を行う。ネットワーク層の転送処理は割り込み通知用ファイルディスクリプタの着信をトリガとして実行され、一度の割り込みにつき 1024 パケットを上限として、バッチングによる ARP/ND テーブル及びルーティングテーブルのルックアップを行う。ルックアップの結果、パケットの宛先アドレスがローカルホスト宛だった場合は、TAP インターフェースを用いてカーネルへパケットを注入する。そうでない場合は、パケットの TTL 減算、IP チェックサム計算、Ethernet ヘッダの宛先・送信元アドレスの書き換え等を行った後、転送先ポートよりパケットを送信する。

3.3 パケット処理機構 (NIC-CPU) の実装

前述したパケット処理機構は本研究の 3 つの実装の基礎であると同時に、NIC 制御及びパケット処理共に CPU のみを用いる実装であるため、本手法では前述の実装をそのまま用いた。

3.4 パケット処理機構 (NIC-CPU-GPU) の実装

NIC-CPU-GPU は、前述した基礎となるパケット処理機構にいくつかの変更を加えることで実装した。本実装の概要について図 3.2 に示す。

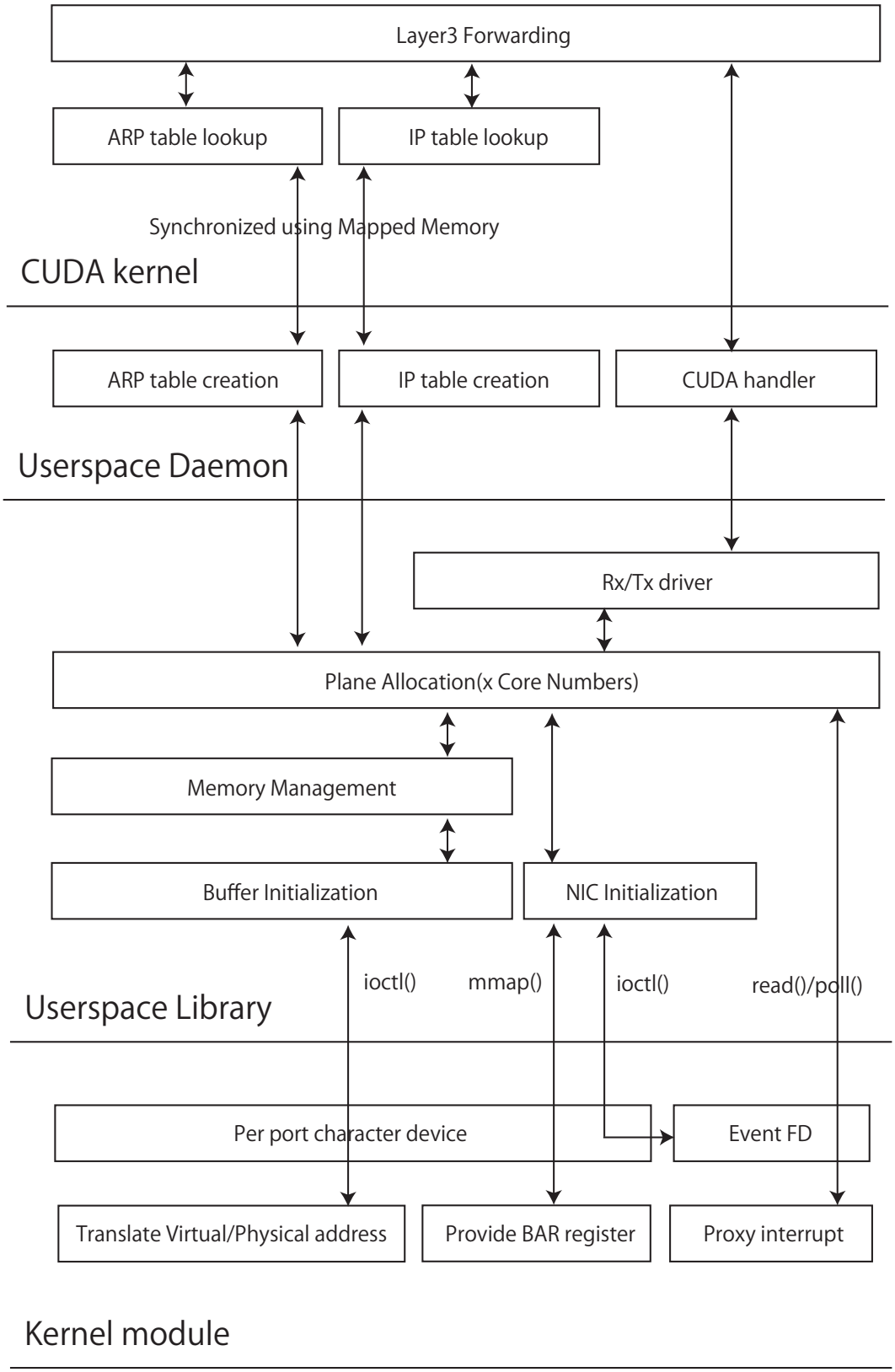

図 3.2: NIC-CPU-GPU の実装の概要

本実装では、パケット処理を GPU 内で実行するため、基礎となるパケット処理機構のユーザスペースデーモンからプリフェッチ関数の呼び出しを削除した。また、ユーザスペースライブラリにおけるメモリ管理機能に変更を加え、パケットバッファ用のヒュージページを確保する際に、確保したヒュージページを Mapped Memory として扱う変更を加えた。これにより、パケットバッファに対する読み込み及び書き込みは、明示的な同期処理を行うことなく GPU カーネルの実行前後に CUDA によって自動的に同期される。

さらに、ユーザスペースライブラリにおいて、仮想アドレスから DMA に用いる物理アドレスに変換する際、常に 2 バイトのオフセットを加えるよう変更した。これは、現行の nvidia 社製 GPU が 4 バイト境界をまたぐメモリアクセスをサポートしていないためである。図 3.3 にオフセットを加えない場合のヘッダとメモリ境界の関係を、図 3.4 にオフセットを加えた場合のヘッダとメモリ境界の関係を示す。本実装において GPU がアクセスする必要のあるヘッダ情報は宛先 IP アドレス、IP ヘッダの TTL フィールド、同チェックサムフィールド、宛先 MAC アドレス、送信元 MAC アドレスである。このとき、Ethernet ヘッダは通常 14 バイトであるため、先頭に 2 バイトのオフセットを加えることにより全てのフィールドが 4 バイト境界に収まることとなり、nvidia 社製 GPU によるアクセスが可能となる。

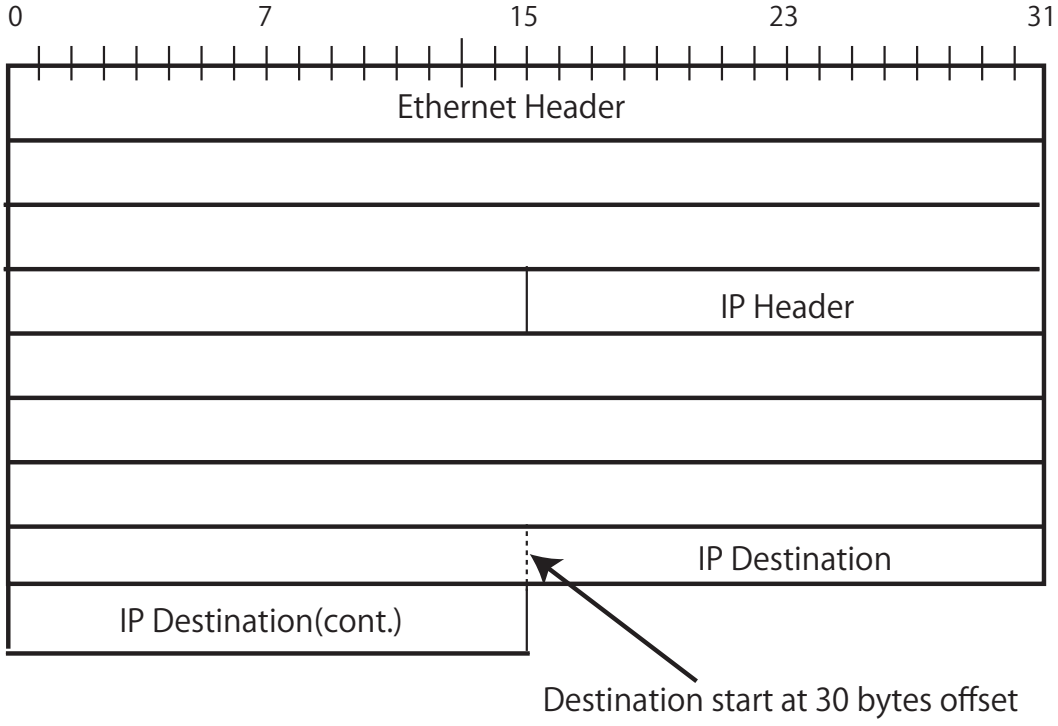

図 3.3: オフセットを加えない場合のヘッダとメモリ境界の関係

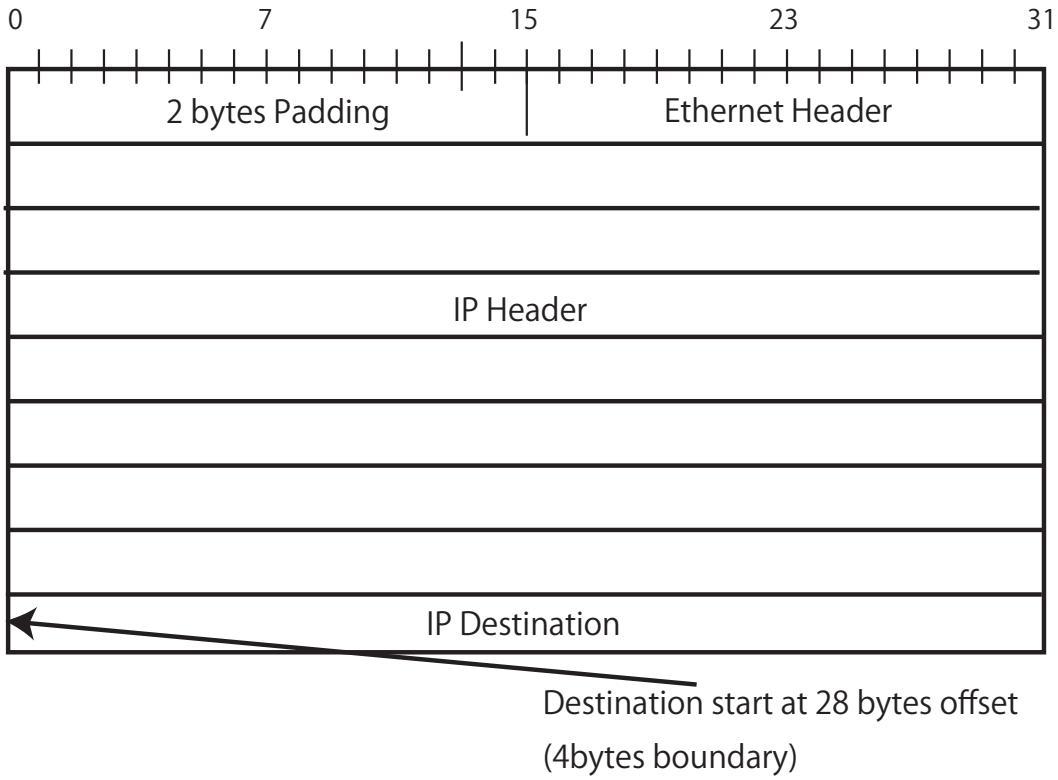

図 3.4: オフセットを加えない場合のヘッダとメモリ境界の関係

また、ユーザスペースデーモンにおいて、ARP/ND テーブル及びルーティングテーブルをヒュージページの未使用領域ではなく cudaManagedMalloc() を使用して確保した Unified Memory による GPU メモリ領域を使用するよう変更を加えた。Mapped Memory となっているヒュージページを使用しても実装は可能だが、ARP/ND テーブル及びルーティングテーブルはメインメモリ上の物理連続な領域に配置する必要がなく、パケットバッファと比較して更新頻度が低いため、ARP/ND ルックアップ及び IP ルックアップの度にメインメモリへのアクセスを行うのは性能面で不利であるためである。

さらに、ユーザスペースデーモンにおけるネットワーク層の転送処理を nvcc によってコンパイル可能な形に移植し、4 ブロック 256 スレッドを最大として並列に処理できるよう変更を行った。具体的には、IP ルックアップにおけるルックアップ処理を再帰的なアルゴリズムから繰り返し構造によるアルゴリズムに変更し、最終的な転送先ポートを CPU に配列の形で返すものとした。

3.5 パケット処理機構 (NIC-GPU) の実装

NIC-GPU は、NIC-CPU-GPU にさらに変更を加え、パケットバッファ用のヒュージページを GPU メモリ上の物理連続な領域と置き換えることで実装した。本実装では、GPU メモリ領域の物理アドレスを取得するため、GPU Direct を使用する。GPU Direct

はカーネル空間内の API として実装されており、カーネル空間において nvidia_p2p_get_pages() 関数を呼び出すことで CUDA を使用して確保した GPU 仮想メモリ領域を全て実際の GPU メモリ上に割り当て、物理アドレスを取得することができる。本実装では GPU Direct を使用するための専用カーネルモジュールを新たに実装した。本実装のカーネルモジュールを含めた実装の概要について、図 3.5 に示す。

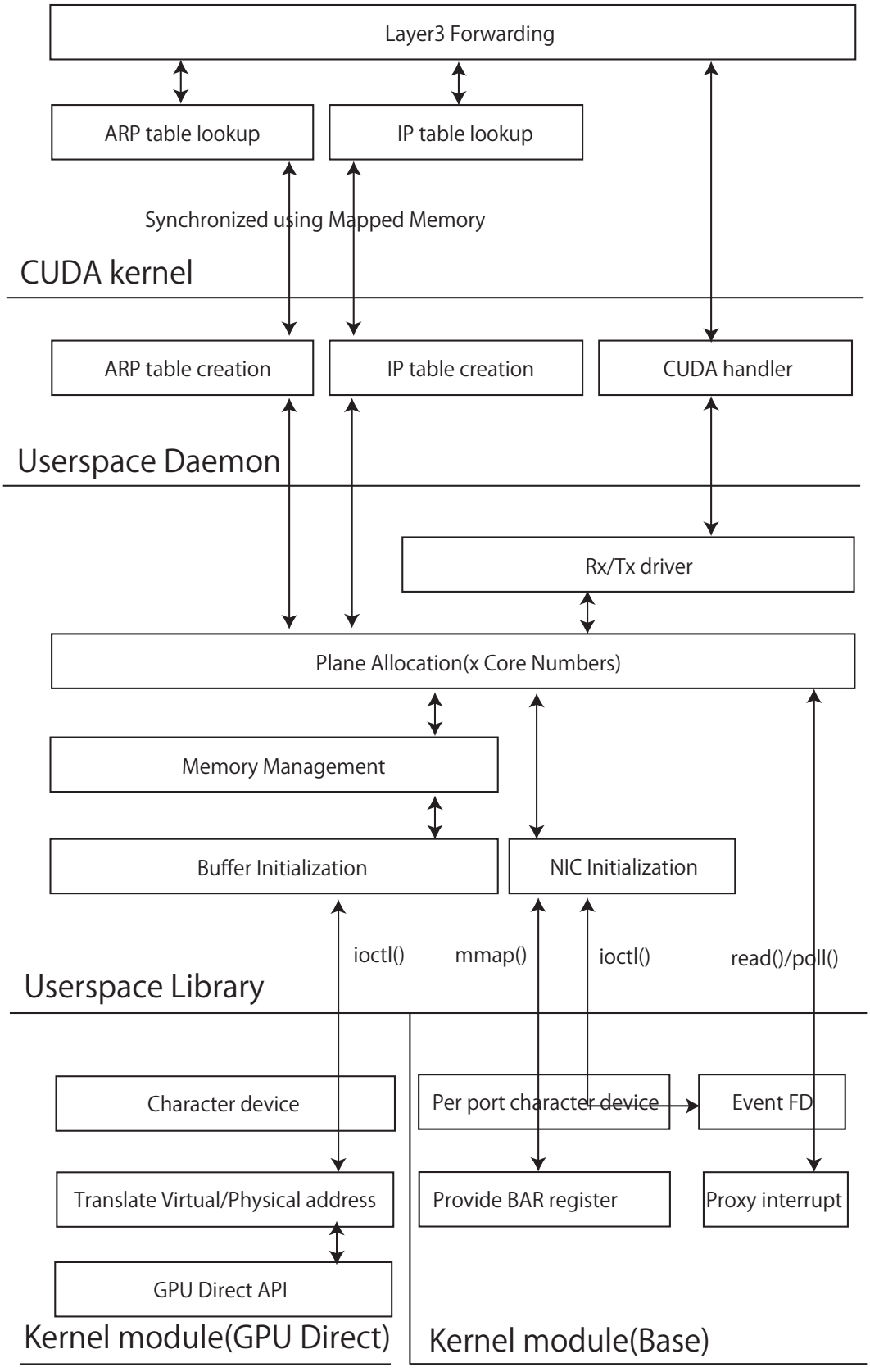

図 3.5: NIC-GPU の実装の概要

本カーネルモジュールはキャラクタデバイスの形でユーザスペースにインターフェースを提供する。ユーザスペースライブラリは、パケットバッファを確保する際にこのキャラクタデバイスを介して ioctl() システムコールにより割り当てられた GPU メモリの仮想アドレスをカーネルモジュールに通知する。カーネルモジュールは GPU Direct を使用して物理アドレスを解決後、結果をユーザスペースに返す。本カーネルモジュールを利用することで、GPU メモリを DMA 用のバッファ領域として使用することが可能になるため、NIC-GPU 間の直接的なデータ転送が実現する。

3.6 本章のまとめ

本節では本章の議論を整理し、要点を示す。本章では、実装の前提となる技術として GPGPU、CUDA、GPU Direct について述べた。その上で、**??**章で述べた 3 つの手法の設計を基に行った実装について述べた。本研究のパケット処理機構は、カーネルモジュール、ユーザスペースライブラリ、ユーザスペースデーモンからなり、3 つの手法において異なる部分を、関数を置き換えることで実装した。

カーネルモジュールは、Linux においてカーネル内からのみアクセスできる資源をユーザスペースへ仲介する。具体的には NIC の各ポートをキャラクタデバイスとして抽象化し、それを介して割り込みの中継、ヒュージページの物理アドレス取得等の機能を提供する。

ユーザスペースライブラリは、カーネルモジュールが提供するキャラクタデバイスを用いて、NIC のキュー及び割り込みの初期化を行い、メモリ管理やデスクリプタリングの操作などパケット処理に必要な機能を提供する。また、NIC の各資源と CPU 側の各資源をプレーンという単位で抽象化し、並列でのパケット処理においてコアが独立して動作することを可能にする。

ユーザスペースデーモンは受信したパケットのヘッダを基にネットワーク層の転送処理を行う。ネットワーク層の転送処理はカーネルと同期して構築された ARP/ND テーブル及びルーティングテーブルに基づいて行う。ARP/ND テーブルは 16bit のハッシュテーブルを使用して構築し、ルーティングテーブルは DIR-24-8 を基に 16bit のテーブルに 8bit のテーブルを連鎖させることで構築する。

第4章　評価

本章では、本研究で設計・実装したパケット処理機構の評価を行う。

4.1　評価項目

2.2 節で示した通り、機能要件は 1) シングルフローにおいて 10Gbps Ethernet の理論転送レートに近いパケット処理性能を有する 2) シングルフローではパケットの転送順序が保たれる 3) 演算装置の規模に応じて性能を向上可能なアーキテクチャである 4) ポート間転送における遅延が 500 マイクロ秒以下であること、の 4 つである。このうち、2)、3) は本研究で実装したパケット処理機構が持つ機能として実現される。本研究では、1) 及び 4) の機能要件を踏まえ、Round Trip Time 及びパケット処理性能を評価項目とする。また、本研究では、設計・実装したパケット処理機構がソフトウェアであることを踏まえ、参考として転送時の CPU 負荷を計測する。以下に、本研究で計測を行う項目をまとめる。

- ネットワーク層の転送処理時の Round Trip Time
- ネットワーク層の転送処理時のパケット処理性能
- ネットワーク層の転送処理時の CPU 負荷

4.2　実験計画

本節では、4.1 で挙げた計測項目について実験を行うための計画を述べる。本研究では、各手法の実装について 1) ネットワーク層の転送処理時の Round Trip Time 2) ネットワーク層の転送処理時の Round Trip Time 3) ネットワーク層の転送処理時の CPU 負荷、について実験を行い、性能を評価する。

本研究で設計・実装したパケット処理機構は、2.1 節で述べたように 1Gbps Ethernet または 10Gbps Ethernet によって接続されたエンドポイントを集約する用途を想定している。本研究の目的は、汎用計算機を用いた安価で高速なパケット処理システムを実現することであるため、これらの想定下で提案した手法を適用したパケット処理機構の通信性能を評価する必要がある。そのため、本研究では 10Gbps Ethernet によって接続された汎用計算機間のトラフィックをネットワーク層で転送する環境を構築し、実験を行う。

4.2.1 実験環境

本実験で構築する実験環境のトポロジを図 4.1 に示す。本実験では、2 台のエンドポイント及び 1 台の L3 フォワーディング用計算機を用意し、エンドポイント間でトラフィックを流す。表 4.1 に L3 フォワーディング用計算機のスペックを示す。L3 フォワーディング用計算機では、Intel 社の CPU を用いるが、その際 HyperThreading[13] 及び VT-d[14] を無効化し、電力消費モードをパフォーマンス優先とする。測定時は、NUMA アーキテクチャの影響を避けるため、L3 フォワーディング用計算機の CPU ソケットを 1 つのみ使用し、単一のフローが単一のコアのみで処理され、かつ割り込みを処理するコアがパケットを処理するコアと同一となるよう設定する。

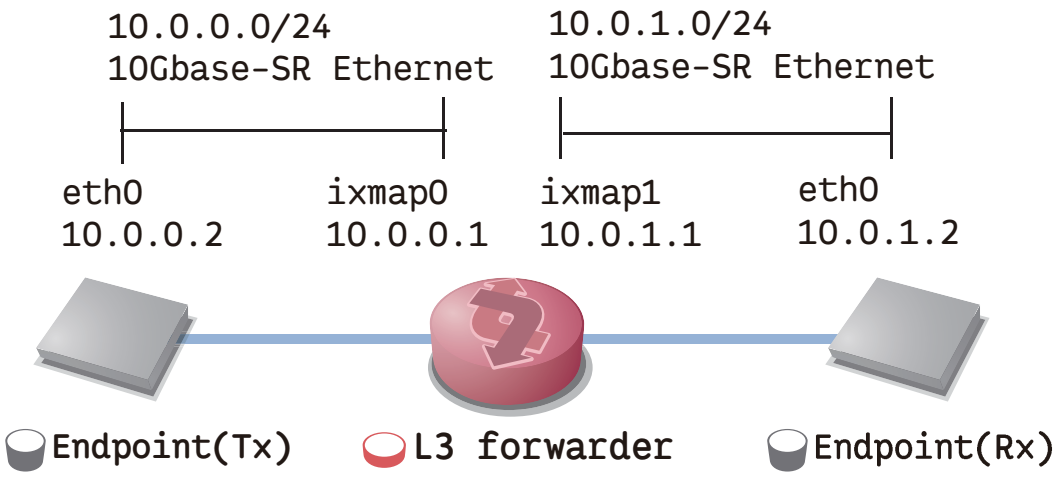

図 4.1: 実験環境のトポロジ

表 4.1: L3 フォワーディング用計算機のスペック

CPU	Intel(R) Xeon(R) CPU E5-2697 v3 (2sockets)
コア数	14
メモリ	256GB DDR4
NIC	Intel X520 82599ES 2port

表 4.2 に本実験で使用するエンドポイントのスペックを示す。エンドポイントのスペックは送信側・受信側ともに同一である。エンドポイントでは、10Gbps のトラフィックを送受信するため、Netmap を用いたトラフィックジェネレータプログラム (pktgen) を動作させる。遅延については、ping コマンドを用いて Round Trip Time(RTT) の計測を行う。また、CPU 負荷については top コマンドを用いてコア毎の負荷を計測する。

また、本実験では計算機間の接続に 10Gbps Ethernet を用いる。10Gbps Ethernet の NIC は各計算機に対し PCI Express Gen2.0 x8 にて接続する。計算機間の接続には光ファイバを用い、10Gbase-SR SFP+によって接続する。

表 4.2: エンドポイントのスペック

CPU	Intel(R) Xeon(R) CPU E3-1240 v3 (1socket)
コア数	4
メモリ	16GB DDR3
NIC	Intel X520 82599ES 2port

4.2.2 実験に使用するシステムの構築

計測のため、以下で述べる手順でシステムの構築を行った。構築したシステムについて、NIC-CPU の概要を図 4.2 に、NIC-CPU-GPU の概要を図 4.3 に、NIC-GPU の概要を 4.4 に示す。本実験は、NIC 上の 2 つのポートを使用し、片方向通信にて行った。実験に使用したシステムでは、各ポートに受信側・送信側それぞれ 1 つのキューを作成し、対応するデスクリプタリングのサイズは 4094 とした。また、NIC のフローコントロールは無効化し、受信側バッチングの最大数は 1024、送信側バッチングの最大数は 4094 とした。各ポートで使用するために事前に確保するパケットバッファは、フレーム MTU を 1518 とした上で 2048 バイトを 8192 個分とした。各ポートからの割り込みは CPU コア 0 に固定的に割り当て、NIC-CPU ではパケット処理も CPU コア 0 のみで行うよう設定した。一方、NIC-CPU-GPU 及び NIC-GPU では、到着したパケット数に応じて CUDA カーネルを最大 4 つのマルチプロセッサで 256 スレッド分並列に起動するよう設定した。

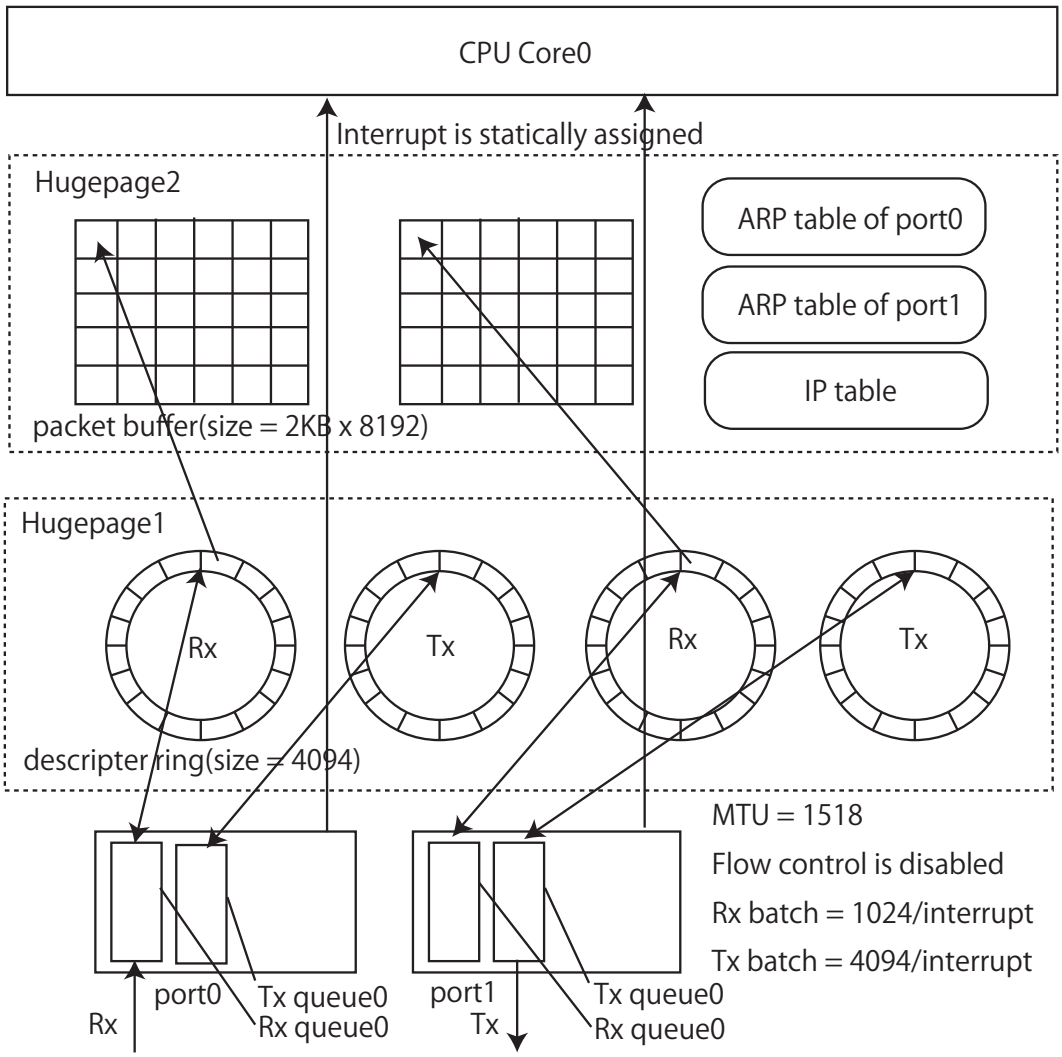

図 4.2: 構築したシステムの概要 (NIC-CPU)

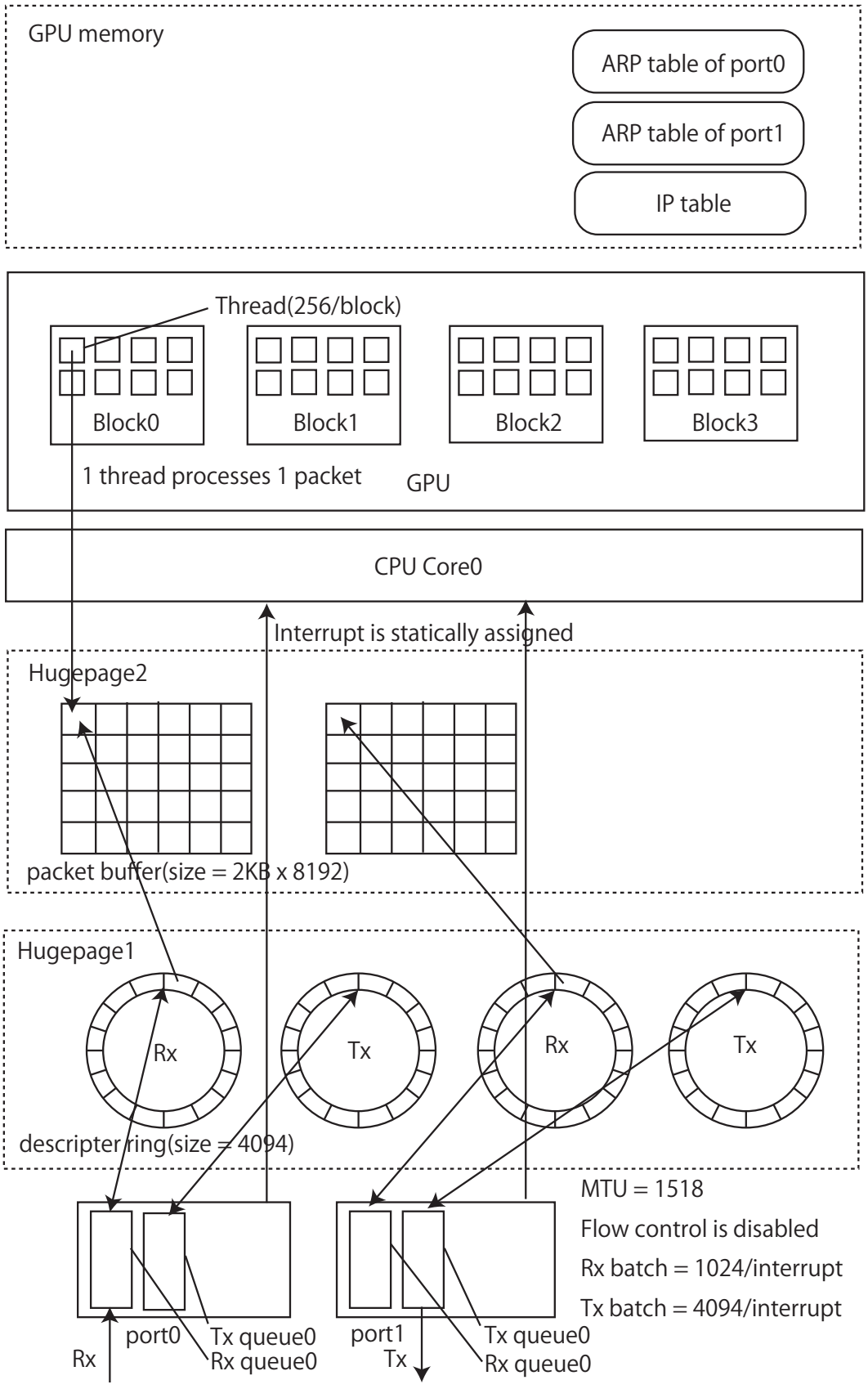

図 4.3: 構築したシステムの概要 (NIC-CPU-GPU)

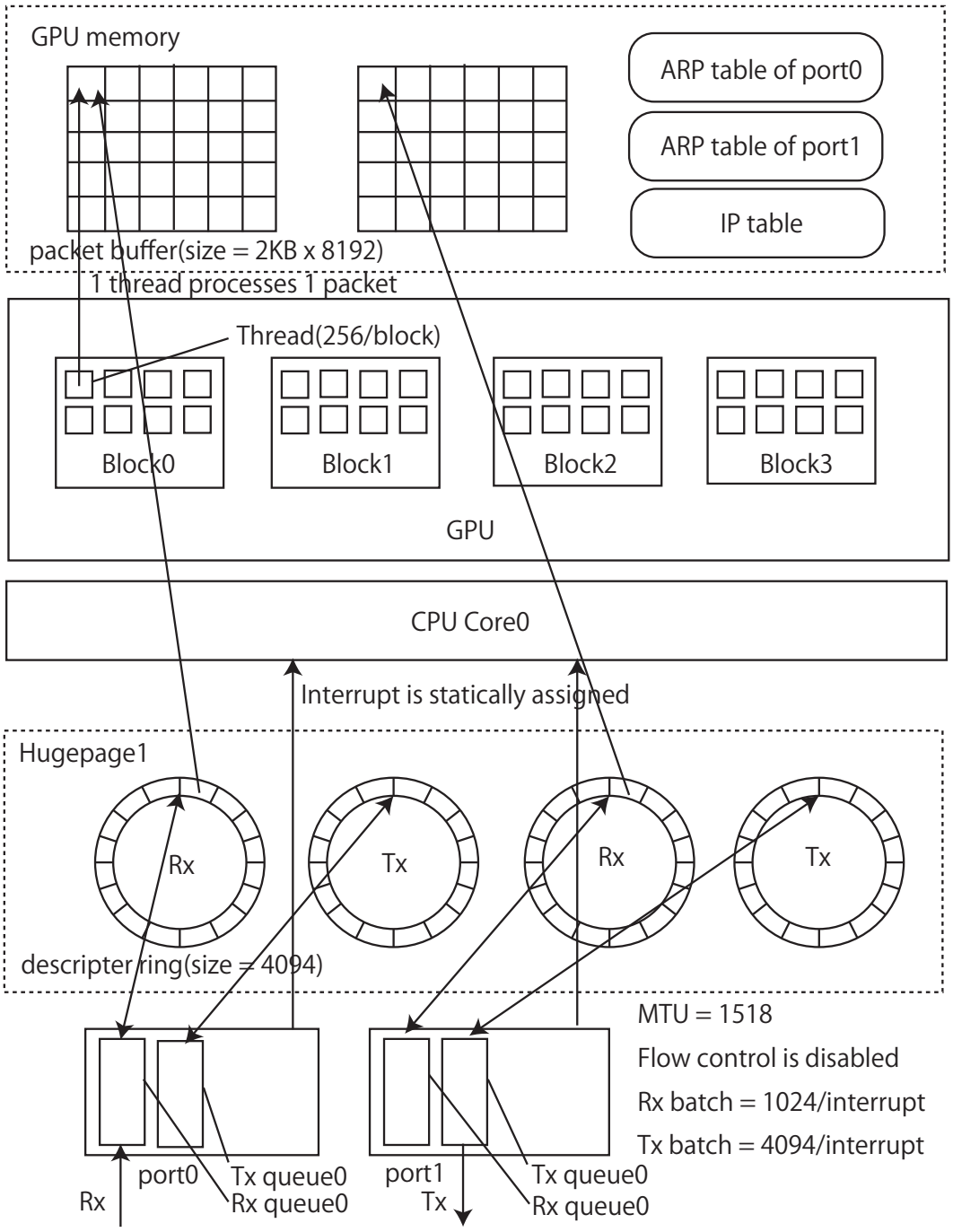

図 4.4: 構築したシステムの概要 (NIC-GPU)

L3 フォワーディング用計算機では、まずカーネルモジュールの読み込みを行った後、フォワーディング用プロセスを起動する。プロセス起動後は、各ポートに対し IP アドレスの設定を行う。各ポートは Linux カーネルが提供する TAP インターフェースとして抽象化されているため、以下のコマンドを用いて IP アドレスの設定を行った。

```
ip addr add 10.0.0.1/24 dev ixmap0
ip addr add 10.0.1.1/24 dev ixmap1
```

また、送信側エンドポイント用計算機では、Netmap 上で動作する pktgen を動作させるため、netmap 用カーネルモジュールの読み込みを行った後、以下のコマンドを用いて IP アドレスの設定及びフローコントロール機能の無効化を行った。

```
ethtool -A eth0 tx off
ethtool -A eth0 rx off
ip addr add 10.0.0.2/24 dev eth0
ip route add 10.0.1.0/24 via 10.0.0.1 dev eth0
```

受信側エンドポイントにおいても送信側と同様の設定を行った。

```
ethtool -A eth0 tx off
ethtool -A eth0 rx off
ip addr add 10.0.1.2/24 dev eth0
ip route add 10.0.0.0/24 via 10.0.1.1 dev eth0
```

本実験では、上記の環境下で、Netmap 上で動作する pktgen、ping コマンド、top コマンドを使用して計測を行う。

4.3 計測

4.3.1 Round Trip Time

Round Trip Time(RTT) は、エンドポイント間で ping を用いて計測した。ping を用いて 1 秒に一回のパケット送受信を計 20 回行い、その平均値を結果とした。また、計測はフレームサイズを変え、フレームサイズが 128/256/512/1024/1280/1518 バイトのそれぞれの場合について行った。このとき、64 バイトのフレームは ICMP ヘッダを含めることができず RTT の計測ができないため実施しない。計測の結果を図 4.5 に示す。

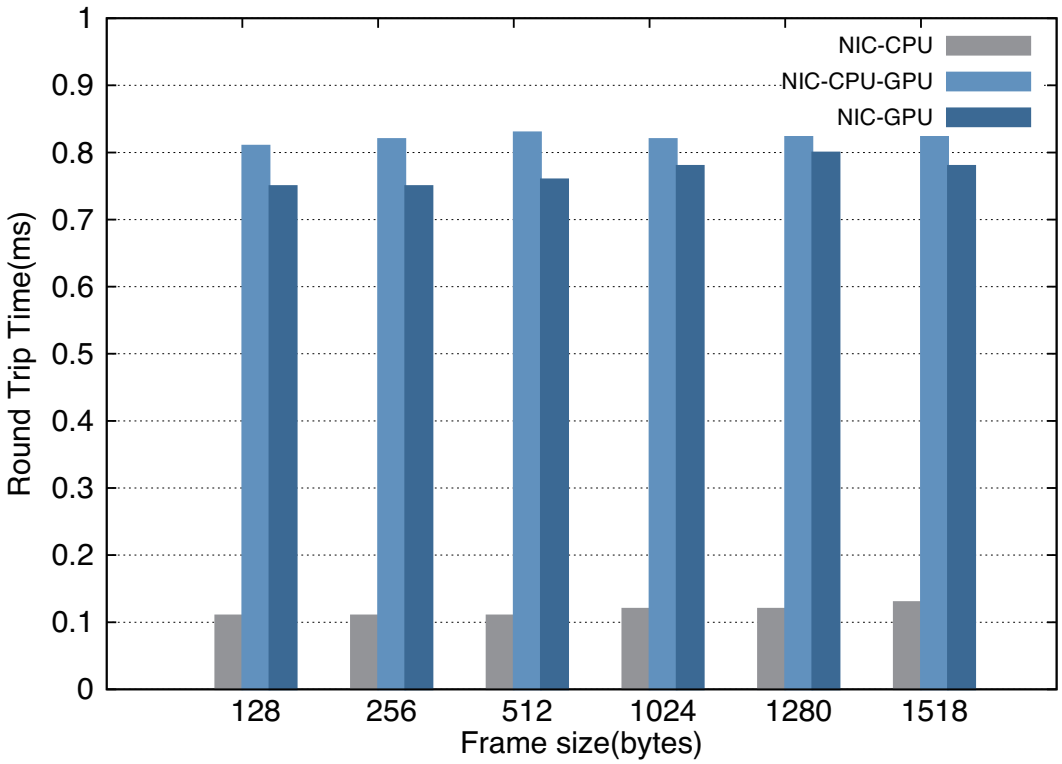

図 4.5: 各実装の Round Trip Time

結果から、全ての実装でフレームサイズによらず RTT が一定しており、かつ RTT を半分にした値は機能要件である 500 マイクロ秒以下の遅延であることがわかった。ただし、NIC-CPU と比較して NIC-CPU-GPU 及び NIC-GPU の RTT が全てのフレームサイズで大きくなっており、750 ミリ秒から 850 ミリ秒の範囲となっている。NIC-CPU-GPU 及び NIC-GPU では割り込みの処理や送受信のためのデスクリプタリングの操作に CPU を使用しており、パケットバッファのアドレスや送信ポートなど、パケットバッファ以外のパラメータを受け渡す必要がある。このような CPU と GPU 間のカーネル実行に伴う入出力が遅延を増大させていると考えられる。

4.3.2 パケット処理性能

パケット処理性能は、送信側エンドポイントからトラフィックを生成することで計測した。トラフィックの生成は 60 秒間行い、対向のゲスト OS が受信したトラフィック量から 1 秒あたりのパケット数及びトラフィック量を算出し結果とした。計測は生成するフレームサイズが 64/128/256/512/1024/1280/1518 バイトのそれぞれの場合について行った。また、宛先 IP アドレス及び送信元 IP アドレスは一定の値を用い、L3 フォワーディング用の計算機に投入する経路数は動作に必要な最低限の数にて行った。計測の結果のうち Packet Per Second(pps) 単位のものを図 4.6 に、Bit Per Second(bps) 単位のものを図 4.7 に示す。なお、4.2.2 節で述べたように、本計測は CPU コアを 1 つのみ使い片方向通信にて実施している。

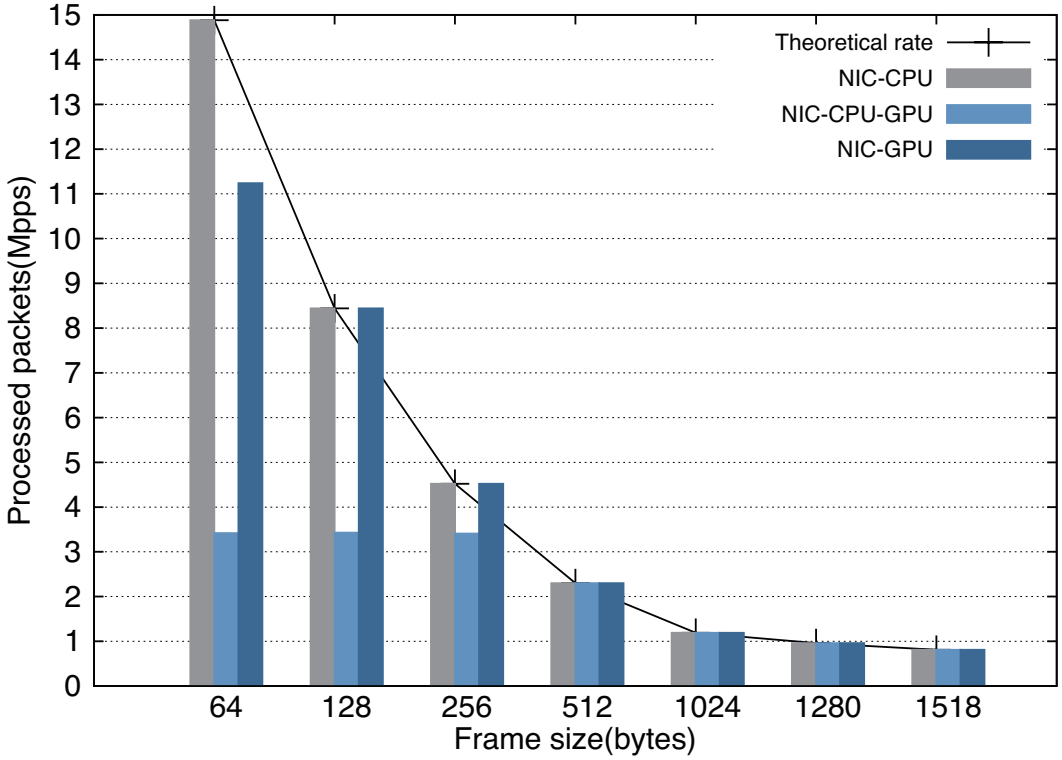

図 4.6: 各実装のパケット処理性能 (pps)

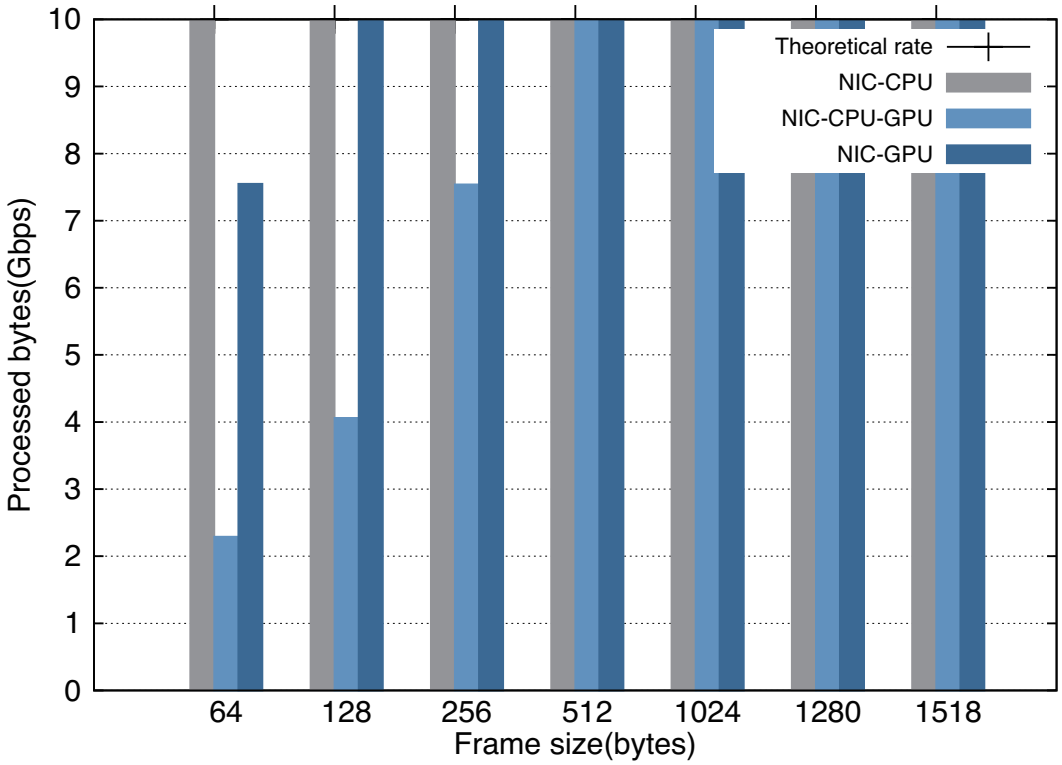

図 4.7: 各実装のパケット処理性能 (bps)

結果から、まず NIC-CPU 実装においては全てのフレームサイズで機能要件である理論最大レートを達成していることがわかった。一方で NIC-CPU-GPU では、512 バイト以上のフレームサイズで、NIC-GPU では 128 バイト以上のフレームサイズで理論最大レートを達成していることがわかった。NIC-GPU の性能が全てのフレームサイズにおいて NIC-CPU-GPU を上回ることから、NIC-GPU 間での直接データ転送を行うことにより、メインメモリを経由する場合と比較してより高い性能を達成できることがわかった。

ただし、64 バイトフレーム時の NIC-GPU のパケット処理性能は 11.2Mpps 程度であり、本研究の実装では NIC-CPU の性能が NIC-CPU-GPU 及び NIC-GPU を上回っている。GPU を用いたハードウェアアクセラレーションを用いて CPU のみの実装を上回る性能を達成している先行研究が多数存在していることから、この結果はアーキテクチャ上の問題ではなく、本研究の実装を最適化することで解決できると考えられる。特に、本研究では 3 章で述べたように IP ルックアップ及び ARP/ND ルックアップのアルゴリズムとして全ての実装で共通のものを使用している。これらのアルゴリズムはもともと CPU によって処理することを前提として考案されたものであり、GPU での処理に最適化されているわけではないため、この部分については性能改善の余地があると考えられる。

4.3.3 CPU 負荷

通信時の L3 フォワーディング用計算機の CPU 負荷は、4.3.2 節と同様の方法でエンドポイント間において理論最大レートのトラフィックを生成し、その際の CPU 負荷を計測することで行った。本実験の結果を図 4.8 に示す。トラフィックの生成は 60 秒間行い、そのうち前後 20 秒間を除いた 20 秒間 L3 フォワーディング用計算機において CPU 負荷を取得し、その平均値を結果とした。本実験は CPU コアを 1 つのみ使用しているため、コア単位での最大負荷を 100%とした。

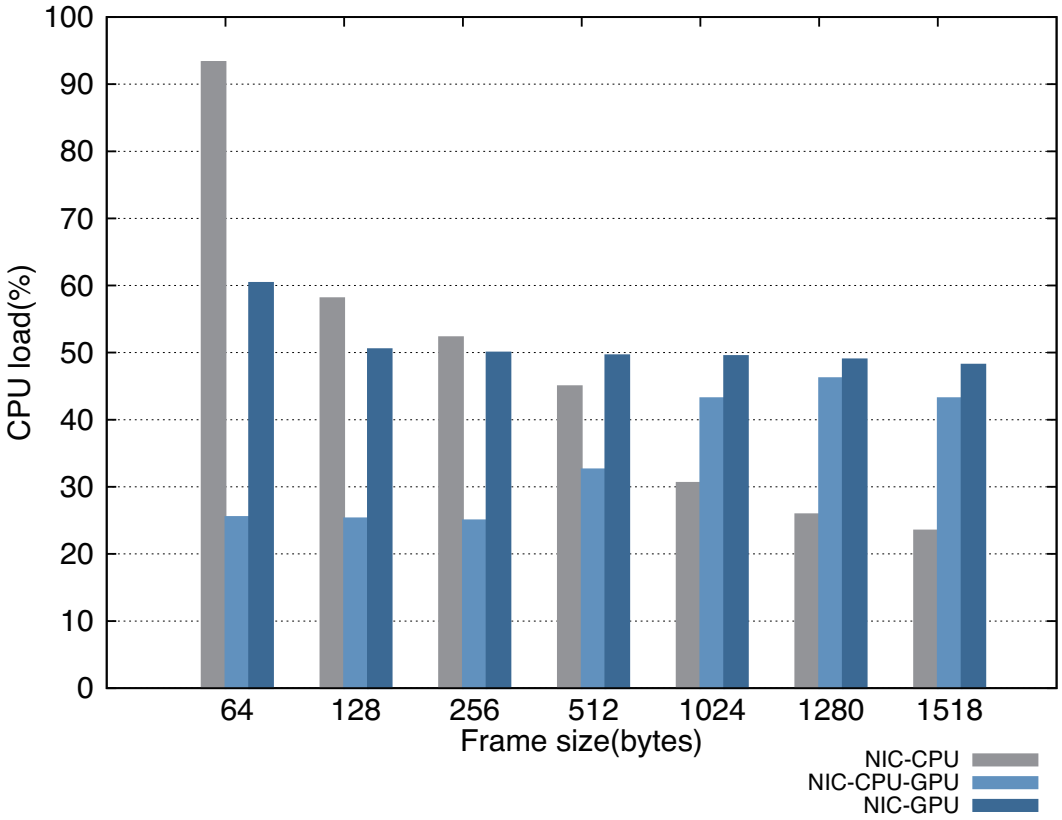

図 4.8: 各実装の CPU 負荷

結果から、全てのフレームサイズが 64 から 256 バイトのとき、NIC-GPU の CPU 負荷は NIC-CPU と比較して低いことがわかった。同様に、フレームサイズが 64 から 512 バイトのとき、NIC-CPU-GPU の CPU 負荷は NIC-CPU と比較して低いことがわかった。さらに、NIC-CPU における CPU 負荷の傾向とは異なり、GPU を用いた場合はフレームサイズが大きくなるにつれて CPU 負荷が増大している。この結果から、GPU を用いたハードウェアアクセラレーションが有効な場合、パケットの転送レート以外の要因によって CPU 負荷が変化していると考えられる。その一因として、NIC-CPU-GPU 及び NIC-GPU においても割り込み処理及び NIC のデスクリプタリングの操作は CPU によって行っているため、CPU-GPU 間の同期が必要になる点が考えられる。本研究の実装では、GPU 内でパケット処理を行っている間、CPU 側はブロッキングによる GPU 処理の完了待ちを行う必要があり、CPU 使用率にはこのブロッキングに消費される CPU サイクルが含まれる。

本研究では、原因をより詳細に分析するため、NIC-GPU における 1518 バイトでの転送時に Linux 上で動作するプロファイラである perf を用いてプロファイリングを行った。perf でのプロファイリング結果のサンプル数上位 20 件を以下に示す。

```
11.01%  libcuda.so.352.39     [.] 0x00000000008786d8
 7.45%  libcuda.so.352.39     [.] 0x00000000008786c6
 7.16%  [vdso]                [.] __vdso_clock_gettime
```

```
5.80%  [kernel]              [k] system_call
4.48%  [kernel]              [k] system_call_after_swapgs
3.92%  [kernel]              [k] copy_user_enhanced_fast_string
1.86%  [kernel]              [k] read_tsc
1.71%  [kernel]              [k] sys_clock_gettime
1.70%  [kernel]              [k] getrawmonotonic
1.56%  libcuda.so.352.39     [.] 0x00000000008785c4
1.50%  libcuda.so.352.39     [.] 0x000000000022e210
1.43%  libcuda.so.352.39     [.] 0x000000000022e4df
1.40%  [kernel]              [k] native_read_tsc
1.31%  [kernel]              [k] sysret_check
1.10%  libixmap.so.1.0.0     [.] ixmap_rx_clean
1.01%  [kernel]              [k] irq_entries_start
0.86%  libpthread-2.19.so    [.] pthread_mutex_lock
0.84%  libc-2.19.so          [.] __clock_gettime
0.74%  libcuda.so.352.39     [.] 0x000000000022e4da
0.68%  [kernel]              [k] _set_stream_running
```

perf によるプロファイリング結果から、CPU 実行時間の大部分が CUDA のライブラリ関数に消費されていることがわかる。L3 フォワーディング時に頻繁に呼び出される CUDA のライブラリ関数は CPU と GPU 間の同期に要するもののみであるため、CPU が GPU 内処理の完了待ちをする時間が CPU 負荷に計上されていることがわかる。この間、CPU において他の処理を実行することは可能であるため、本結果は実用途において問題となるものではない。ただし、この同期処理についても CPU-GPU 間の同期処理をする間デスクリプタリングの操作をあらかじめ行っておくなど、より高い性能を達成するための実装上の工夫の余地がある。

4.4 実験のまとめと考察

本研究では、設計・実装したパケット処理機構が 2.2 節で示した機能要件を満たし、かつ NIC と GPU 間の直接データ転送がパケット処理性能を改善することを確認するため、1) ネットワーク層の転送処理時の Round Trip Time 2) ネットワーク層の転送処理時のパケット処理性能の計測を行った。また、設計・実装したパケット処理機構がソフトウェアであることを踏まえ、参考として 3) ネットワーク層の転送処理時の CPU 負荷を計測した。

1) に関する実験の結果から、設計・実装したいずれのパケット処理機構も遅延に関する機能要件を満たしていることがわかった。ただし、全てのフレームサイズにおいて NIC-CPU は 0.1 ミリ秒程度の RTT であり、NIC-CPU-GPU 及び NIC-GPU は 0.8 ミリ秒程度の RTT であった。NIC-CPU-GPU 及び NIC-GPU では、CPU と GPU 間のカーネル実行に伴う入出力が遅延を増大させていると考えられる。

２) に関する実験の結果から、NIC-CPU は全てのフレームサイズにおいてネットワーク層の転送処理が理論最大レートにて可能であることがわかった。また、NIC-CPU-GPU は 512 バイト以上、NIC-GPU は 128 バイト以上でネットワーク層の転送処理が

理論最大レートにて可能であることがわかった。結果から、本研究で提案した手法である NIC と GPU 間の直接転送を適用することにより、通常の GPU を用いたハードウェアアクセラレーションを行う場合と比較して高い性能を達成できることがわかった。ただし、NIC-GPU ではフレームサイズが 128 バイト以下のとき理論最大レートを達成していない。この点については、先行研究において理論最大レートの性能を達成した例があるため、ARP/ND ルックアップ及び IP ルックアップなど、GPU 内部の処理を最適化することで性能を改善できると考えられる。

3) に関する実験の結果から、フレームサイズが大きい場合は NIC-CPU と比較して NIC-CPU-GPU 及び NIC-GPU の CPU 負荷が高いことがわかった。ただし、perf によるプロファイリング結果から、この CPU 負荷は CPU と GPU 間の同期に要する GPU 処理の完了待ちが計上された結果であると考えられ、実用途に問題はないと言える。

4.5 本章のまとめ

本節では本章の議論を整理し、要点を示す。本章では、本研究で設計・実装したパケット処理機構の遅延、パケット処理性能、CPU 負荷を計測した。計測の結果から、NIC と GPU 間の直接データ転送にはパケット処理性能を改善する効果があることがわかった。

一方で、NIC-CPU と比較して NIC-CPU-GPU 及び NIC-GPU は性能の点で劣るため、ARP/ND ルックアップ及び IP ルックアップの GPU に向けた最適化など、実装の点で改善の余地があることがわかった。

第5章　結論

本研究では、汎用計算機を用いた安価で高速なパケット処理システムを実現することを目的として、パケット処理機構を設計・実装した。その手法として、本研究では汎用計算機におけるパケット処理技術の中で、ハードウェアアクセラレーションの持つ利点、データリンク層またはネットワーク層の処理だけでなくさらに高いレイヤでの処理についても応用が可能であることを重視し、ハードウェアアクセラレーションに特に着目した。ハードウェアアクセラレーションを用いたパケット処理では、内部バスがボトルネックとなることが先行研究により示されているため、本研究では NIC-GPU 間の DMA を実現することによりメインメモリを経由する PCI Express によるデータ転送を回避するよう設計・実装を行った。

本研究で実装したパケット処理機構により、ハードウェアアクセラレーションを用いつつも内部バスの使用を抑えることができ、汎用計算機におけるパケット処理性能を改善できたことを示すため、10Gbps Ethernet によって接続された汎用計算機間のトラフィックをネットワーク層で転送する環境を構築し、評価を行った。評価は遅延、パケット処理性能、通信時の CPU 負荷を計測することによって行い、NIC と GPU 間の直接データ転送を行うパケット処理機構は、メインメモリを経由する従来の実装形態より高いパケット処理性能を得られることを示した。

参考文献

[1] R. Bolla, R. Bruschi, and A. Ranieri. Performance and power consumption modeling for green cots software router. In *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*, pages 1–8, Jan 2009.

[2] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast userspace packet processing. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*, pages 5–16. IEEE Computer Society, 2015.

[3] Luigi Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX Annual Technical Conference*, pages 101–112, 2012.

[4] Raffaele Bolla and Roberto Bruschi. Linux software router: Data plane optimization and performance evaluation. *Journal of Networks*, 2(3):6–17, 2007.

[5] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: a gpu-accelerated software router. *ACM SIGCOMM Computer Communication Review*, 41(4):195–206, 2011.

[6] Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis. Gaspp: a gpu-accelerated stateful packet processing framework. In *USENIX ATC*, 2014.

[7] Anuj Kalia, Dong Zhou, Michael Kaminsky, and David G Andersen. Raising the bar for using gpus in software packet processing.

[8] Weibin Sun and Robert Ricci. Fast and flexible: Parallel packet processing with gpus and click. In *Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 25–36. IEEE Press, 2013.

[9] Joel Hasbrouck and Gideon Saar. Low-latency trading. *Journal of Financial Markets*, 16(4):646–679, 2013.

[10] Stephen M Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K Ousterhout. It 's time for low latency. In *Proceedings of the 13th USENIX*

conference on Hot topics in operating systems, page 11. USENIX Association, 2011.

[11] Michael Laor and Lior Gendel. The effect of packet reordering in a backbone link on application throughput. *Network, IEEE*, 16(5):28–36, 2002.

[12] Pankaj Gupta, Steven Lin, and Nick McKeown. Routing lookups in hardware at memory access speeds. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1240–1247. IEEE.

[13] Debbie Marr. Hyper-threading technology in the netburst® microarchitecture. *14th Hot Chips*, 2002.

[14] R Hiremane. Intel virtualization technology for directed i/o (intel vt-d). *Technology@ Intel Magazine*, 4(10), 2007.