一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
NS2013-101 (2013-10)

TCP/IPスタックにおけるチェックサム計算のGPUオフロードによる性能向上手法

坪内　佑樹†　長谷川　剛††　谷口　義明††　中野　博隆††　松岡　茂登††

† 大阪大学 大学院情報科学研究科　〒565-0871 大阪府吹田市山田丘 1-5
†† 大阪大学 サイバーメディアセンター　〒560-0043 大阪府豊中市待兼山町 1-32
E-mail: †y-tubout@ist.osaka-u.ac.jp, ††{hasegawa,y-tanigu,nakano,matsuoka}@cmc.osaka-u.ac.jp

あらまし　Ethernet Jumbo Frameの登場により、特にデータセンタなどの閉じたネットワーク環境において送受信されるフレームサイズが増大している。フレームサイズの増大に伴い、TCP/IPスタックにおけるチェックサム計算に要するCPU負荷が増大する。本報告では、大きな帯域幅のメモリをもつGraphics Processing Unit (GPU)にチェックサム計算をオフロードすることにより、CPU負荷を削減し、データ転送スループットを向上させる手法を提案する。具体的には、CPU-GPU間のパケット転送効率を向上させるためのパケットキューイング手法、及び、GPU上で複数のパケットを同時処理するためのGPUマルチプロセッサを用いたパケット分散処理手法の2つを提案する。ユーザランドで動作するチェックサム計算の簡易実装により性能を評価し、チェックサム計算のGPUオフロードによって、データ転送性能が最大で13%向上することを示す。

キーワード　Transmission Control Protocol (TCP)、Grphics Processing Unit (GPU)、チェックサム、メモリアクセス

A GPU Offloading method for improving performance of checksum computation in the TCP/IP stack

Yuuki TSUBOUCHI†, Go HASEGAWA††, Yoshiaki TANIGUCHI††, Hirotaka NAKANO††, and Morito MATSUOKA††

† Graduate School of Information Science and Technology, Osaka University
Yamadaoka 1-5, Suita, Osaka, 565-0871 Japan
†† Cybermedia Center, Osaka University
Machikaneyama-cho 1-32, Toyonaka, Osaka, 560-0043 Japan
E-mail: †y-tubout@ist.osaka-u.ac.jp, ††{hasegawa,y-tanigu,nakano,matsuoka}@cmc.osaka-u.ac.jp

Abstract　The size of ethernet frames is becoming larger and larger due to the utilization of Ethernet Jumbo Frame option, especially in closed network environment such as data center networks. Increasing frame size would cause the large overhead for checksum calculation in TCP/IP protocol processing, that increase the CPU load. In this report we propose the scheme for decreasing CPU load and improving data transmission throughput by offloading checksum calculation to Graphics Processing Unit (GPU). Our scheme consists of the following two methods: packet queueing method to improve the packet transmission throughput between CPU and GPU, and the packet processing method exploiting the advantage of GPU's multiprocessor architecture. We evaluate the performance of the proposed scheme by simple experiments using the user-land implementation and confirm that the proposed scheme can improve the TCP data transmission throughput by 13 %, that is almost the same as the case when the checksum calculation is canceled.

Key words　Transmission Control Protocol (TCP)、Grphics Processing Unit (GPU)、Checksum, Memory access

1. まえがき

40 ギガビットイーサネットおよび 100 ギガビットイーサネットの登場により、ネットワークの高速化が顕著である。今後も、ネットワークのさらなる高速化が予想される一方で、CPU のクロック周波数は近年向上していない。したがって、ネットワークの高速化に伴い、OS の TCP/IP スタック処理に要する CPU 負荷が増大すると予想される。この問題に対し、CPU コアの増加による CPU の性能向上に着目した、プロトコル処理のマルチコア分散手法 [1], [2] がある。本手法は、複数 TCP コネクションの処理を CPU の複数コアへ分散させる。しかし、この手法では単一の TCP コネクションの処理を分散させることはできない。そのため、バックアップストレージへのデータ転送などのように、用いる TCP コネクション数が少ないアプリケーションについては、CPU コア数の増加による高速化は困難である。

TCP/IP スタックにおける処理において高い負荷が発生するものとして、以下の 2 点が挙げられる。1 つ目は、チェックサム計算 [3] やカーネルランド-ユーザランド間のパケットデータのコピーのような、パケットサイズに依存した負荷が発生する処理 (以降でバイト単位処理と呼ぶ) であり、2 つ目は、ネットワークインタフェースカード (Network Interface Card: NIC) から OS カーネルへの割り込み処理やパケットの管理構造体へのメモリアクセスなどの、パケットサイズに依存しない処理 (パケット単位処理) である。

CPU の負荷を軽減し、データ転送スループットを向上するための既存手法として、NIC へバイト単位処理またはパケット単位処理をオフロードする手法がある。しかし、一般的に、NIC のメモリ帯域幅は主記憶と比較して小さいため、パケットデータ全体にアクセスする必要があるために、メモリアクセス時間の長い処理となるバイト単位処理を、多数のパケットに対して NIC が実行することは非効率的である。

これらの問題は、Ethernet Jumbo Frame オプションの登場により、より顕著となる。Ethernet Jumbo Frame は、特にデータセンター内通信などの閉じたネットワーク環境における通信において効率的にパケットを処理するための方法として用いられている。Ethernet Jumbo Frame を用いることにより、Maximum Transmission Unit (MTU) が大きくなるため、TCP/IP スタックにおけるバイト単位処理に要する負荷が増大する。そのため、上述したような CPU 負荷増大の問題や、NIC へオフロードする場合の問題が大きくなる。

近年、グラフィクス処理のための演算ユニットである Graphics Processing Unit (GPU) の性能向上が顕著である。また、グラフィクス処理だけでなく、汎用計算の高速化に対する GPU の利用も盛んに行われている。GPU はグラフィックデータへの高速アクセスを実現するために、高い帯域幅を持つメモリを搭載している。そのため、バイト単位処理の実行に向いていると考えられる。しかしながら、TCP/IP スタック処理におけるバイト単位処理を GPU を用いて高速化した事例は著者らの知る限り存在しない。

そこで本報告では、GPU に TCP のチェックサム計算をオフロードすることにより、CPU 負荷を削減し、スループットを向上させる手法を提案する。具体的には、まず、CPU-GPU 間のパケット転送効率を向上させるためのパケットキューイング手法を提案する。これは、キューに格納された受信パケットをまとめて GPU に転送することにより、CPU-GPU 間のパケット転送回数を軽減するのである。次に、GPU 上で複数のパケットを並列処理するための、GPU マルチプロセッサを用いたパケット分散処理手法を提案する。これは、GPU が持つ複数のプロセッサコアに対して、1 つずつのパケットを割り当て、チェックサム計算を並列実行するものである。

提案手法の性能評価には、ユーザランドで動作するチェックサム計算の簡易実装を用いる。簡易実装は、Raw ソケットを用いて送信端末から受信端末へパケットを送信し、受信端末は受信パケットに対してチェックサム計算を実行する。チェックサム計算を従来通り CPU を用いて行う場合と、GPU にオフロードする場合、及びチェックサム計算を行わない場合のデータ転送スループットを比較することによって、提案手法である、チェックサム計算の GPU オフロードが有効であることを示す。

以下、2. 章で GPU のアーキテクチャおよびチェックサム計算の既存の高速化手法について説明する。3. 章では、本報告において提案する GPU チェックサムオフロードについて説明する。4. 章では、評価用のチェックサム計算の簡易実装を用いて提案手法の性能評価結果を示す。最後に 5. 章で本報告のまとめと今後の課題について述べる。

2. 研究の背景

本節では、研究の背景として、2.1 節にて GPU のアーキテクチャついて説明し、2.2 節にて TCP チェックサム計算について述べる。

2.1 Graphics Processing Unit (GPU)

GPU はグラフィクス処理のための演算ユニットである。近年グラフィクス処理だけでなく、汎用計算の高速化のために GPU を利用する試み盛んに行われている。GPU は数多くの Single Instruction, Multiple Data (SIMD) 型プロセッサと高い帯域幅を持つメモリを搭載しており、大量のデータを並列処理することにより、高い演算性能を実現している。GPU の高い演算性能を汎用的なアプリケーションに適用するための開発環境として、NVIDIA 社は Compute Unified Device Architecture (CUDA) [4] を提供している。CUDA は C 言語の拡張構文およびライブラリとして実装されており、グラフィクスの専門知識がなくとも GPU を演算に用いるアプリケーションを開発することができる。

2.1.1 GPU のアーキテクチャ

本節では GPU のアーキテクチャについて説明する。なお、本報告では NVIDIA 社の Geforce GTX680 を用いているため、本節の内容は、NVIDIA 社の GPU のアーキテクチャ、および Geforce GTX680 のスペックに基づいている。

図 1 に NVIDIA 製の GPU のアーキテクチャを示す。GPU のプロセッサは階層的な構造を持つ。具体的には、1 つの Stream-

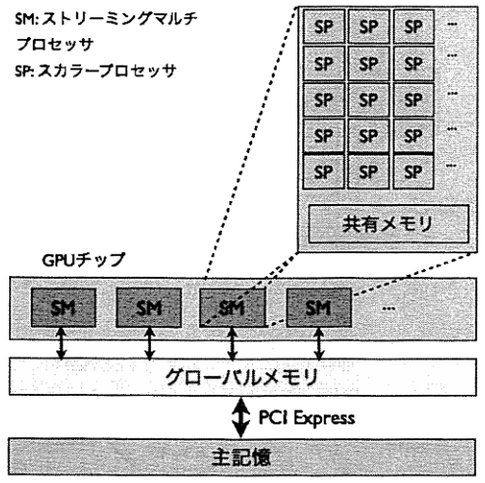

図 1　GPU のアーキテクチャ

ing Multiprocessor (SM) が複数の Scalar Processor (SP) から構成されている。SM は配下の SP 群を用いて SIMD 演算を実行する。例えば、4. 節の評価に用いる NVIDIA Geforce GTX 680 は 8 個の SM がそれぞれ 192 個の SP を持つため、SP の総数は 1,536 個となる。GPU のメモリも、CPU と同様に階層的な構造を持つ。すなわち、全ての SM から共通してアクセス可能なグローバルメモリ、及び、それぞれの SM の配下の SP がアクセスを行う共有メモリがある。グローバルメモリと主記憶は PCI Express バスで接続されており、データ転送が可能となっている。GPU のメモリ容量は 2GB 程度であり、CPU 側のメモリと比較して少ないが、GPU のメモリ帯域幅は CPU のメモリ帯域幅の約 10 倍である。この特性を活かしたプログラミングにより、GPU による汎用計算の高速化を図ることができる。

グローバルメモリは約 2GB の容量を持つが、共有メモリの容量は高々16KB である。しかし、共有メモリはグローバルメモリと比較してさらに高い帯域幅をもつため、複数の SM で共有する必要のないデータ共有メモリに置くことにより、さらなる処理の高速化が可能となる。また、SM はスレッド切り替えによるレジスタの退避のためのオーバヘッドを避けるために、ビット幅が 32 の 65,536 個のレジスタを持つ。

2.1.2　CUDA のプログラミングモデル

本研究においては、NVIDIA 社が提供している統合開発環境である Compute Unified Device Architecture (CUDA) を用いてプログラミングを行う。CUDA を用いることにより、GPU を Single Instruction Multiple Thread (SIMT) 型プロセッサとして利用することができる。SIMT 型プロセッサでは、GPU における最小の実行単位であるスレッド (以降では GPU スレッドと呼ぶ) が単一の演算命令を実行する。複数のスレッドを同時起動することにより、複数の演算命令を同時実行できる。一般的に、CPU はパイプライン処理によってメモリアクセスなどの待ち時間を隠蔽する。一方 CUDA では、SM が実行するスレッドを次々と切り替えることにより、待ち時間を隠蔽する。CPU のマルチスレッド処理ではそれぞれのスレッドが異なる種類の命令を実行できる。それに対して、GPU では 1 つの SM がもつ全ての GPU スレッドは同じ種類の演算命令を実行する。

以下では、CUDA プログラムの実行手順を説明する。CUDA では GPU が実行するジョブの単位をカーネルと呼ぶ。OS のカーネルと混同しないために、以降ではそれぞれ CUDA カーネルおよび OS カーネルと呼ぶ。

（1）　計算対象データのメモリ領域および計算結果格納のためのメモリ領域を CPU からグローバルメモリに対して確保する。

（2）　主記憶からグローバルメモリへデータ転送する。

（3）　CPU が CUDA カーネルを起動する。このとき、グローバルメモリに確保したメモリ領域の先頭アドレスおよび使用する CUDA スレッド数を指定する。

（4）　GPU 上で CUDA カーネルが実行され、CPU に戻すべき計算結果がグローバルメモリに書き込まれる。

（5）　CPU がグローバルメモリ上の計算結果を参照し、主記憶に書き込む。

2.2　TCP チェックサム計算

TCP チェックサムの目的は、送受信される TCP セグメントに対して誤りを検出することである。TCP チェックサムによる誤り検出は以下の手順で行われる。送信側は送信パケットのペイロードに対してチェックサムを計算し、計算結果を TCP ヘッダのチェックサムフィールドに格納する。受信側は送信側と同様に受信パケットのペイロードに対してチェックサムを計算し、計算結果を TCP ヘッダのチェックサムフィールドの値と比較する。比較の結果、値が一致していけなければ、通信途中で TCP セグメントのビット誤りや改ざんが起きたと判断し、パケットを廃棄する。TCP チェックサムはバイト単位処理であるため、TCP/IP スタックの代表的な CPU 負荷の 1 つであり、特に高スループットなデータ転送の際に性能のボトルネックとなることが知られている。

TCP におけるチェックサム計算アルゴリズムとしては、RFC 1071 に定義されているインターネットチェックサムが用いられる。インターネットチェックサムは 16 ビット整数列の総和計算を基にしているため、パケットデータの長さに比例して CPU 負荷が発生する。パケットサイズが小さい場合、チェックサム計算のようなバイト単位処理の負荷は小さいと考えられるが、Ethernet Jumbo Frame 環境の利用などによりパケットサイズが大きくなると、バイト単位処理であるチェックサム計算の処理負荷の割合が大きくなる。そのため、TCP データ転送のスループットを向上させるためには、大きなパケットサイズを用いた場合のバイト単位処理に要する負荷を削減する必要がある。

2.3　TCP/IP プロトコル処理の高速化

初期の TCP の性能評価 [5] によると、TCP においてボトルネックとなるのはバイト単位処理であると報告されている。そこで、ZeroCopy I/O [6] や NIC へのチェックサムオフロードなどの技術が NIC に実装された。[7] の性能評価によると、パケットサイズが小さければ、パケット単位処理のオーバヘッドが TCP における主要なボトルネックとなると報告されている。パケット単位処理のオーバヘッドを削減するために、Large Segmentation Offload (LSO)、Large Receive Offload

(LRO) [8]、及び Interrupt Coalescing などの技術が NIC に実装されている。

一方、TCP/IP スタック処理の一部ではなく、全ての処理をオフロードする TCP Offload Engine (TOE) がある。TOE の性能解析結果 [9] によると、TOE によりデータ転送性能が向上することが明らかとなっている。しかし、TOE を用いても性能向上しない場合があり、例えば、Ethernet Jumbo Frame オプションを使用した際には性能差がほとんど無いことが示されている。

さらに、マルチコア CPU 環境において、TCP/IP スタック処理の性能がコア数に応じて向上しない報告 [10] がある。これは、パケットの受信処理が特定のコアに偏ることが原因である。パケットの受信処理を各コアに分散するために、Receive Side Scaling (RSS)、Receive Packet Steering (RPS)、Receive Flow Steering (RFS), Direct Cache Access (DCA) などの技術 [2], [11] がある。

本報告では、GPU を用いることにより TCP データ転送の高速化を目指す。これにより、TOE などの特定用途のハードウェアに依存せずにデータ転送効率を向上することが可能となる。また、CPU にかかる負荷を GPU にオフロードすることによって、CPU がデータ転送処理以外の処理を実行することが可能となるため、ネットワークアプリケーション全体の性能向上が期待される。

3. チェックサム計算の GPU オフロード手法

本章では、提案手法であるチェックサム計算の GPU オフロード手法について説明する。3.1 節にて提案手法のシステム構成を示し、3.2 節にて CPU-GPU 間データ転送のためのパケットキューイング手法を、3.3 節にてパケット分散処理手法についてそれぞれ述べる。

3.1 システム構成

図 2 に提案手法のシステム構成を示す。NIC と GPU は PCI Express バスを介してメモリコントローラに接続されており、Direct Memory Access (DMA) によりデータ転送が行われる。図の赤線はパケット受信時のパケットの流れを示している。まず、パケットは NIC からメモリコントローラを経由して主記憶のカーネル空間に渡される (赤線 A)。次に、カーネル空間において TCP/IP スタック処理が実行される。その際、チェックサム計算のためにパケットデータが GPU へ渡される (赤線 B)。GPU によってチェックサム計算が行われると、その結果が主記憶のカーネル空間に戻される (赤線 C)。TCP/IP スタック処理を完了すると、パケットデータは主記憶のユーザ空間で動作するアプリケーションに渡される (赤線 D)。

3.2 CPU-GPU 間データ転送のためのパケットキューイング手法

一般的に、CPU と GPU との間のデータ転送は PCI バスを経由しなければならないため、コストの高い処理である。提案手法の性能向上のためには、図 2 の赤線 B,C に示しているデータ転送のコストを軽減することが求められる。そのため、提案手法では、以下に説明するパケットキューイング手法を用いる

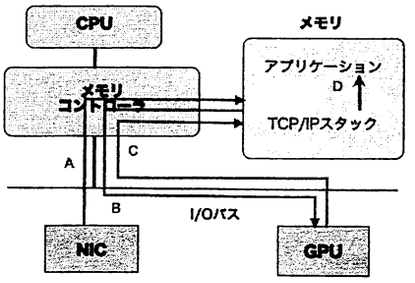

図 2 システム構成

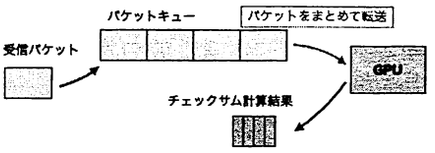

図 3 パケットキューイング手法

ことにより、CPU と GPU との間のデータ転送回数を削減する。そのために用意するパケットキューは固定長のキューであり、受信パケットがキューの要素として格納される。キューが満たされれば格納されている全てのパケットを GPU に転送する。これにより、パケット毎に CPU と GPU との間のデータ転送が発生することを避けることができる。

CUDA を用いる場合、CPU と GPU との間の転送対象のデータは、連続した主記憶アドレスに格納されていなければならない。したがって、キューに格納されている全てのパケットを同時に転送するために、パケットをキューに格納するときには、パケットが連続した主記憶アドレスに格納されるようにする。なお、パケットキューのメモリ確保にかかるオーバヘッドを削減するために、OS の起動時にパケットキューのためのメモリを確保し、以降の処理では 1 つのパケットキューを利用し続ける。

3.3 パケット分散処理手法

提案手法では、GPU 上でのチェックサム計算、すなわち、総和計算を高速化するために、並列リダクション [12] を用いる。CUDA を用いた並列リダクションは、データサイズがおよそ 2MB の 1 個のデータに対して、全ての SM を用いて総和計算を実行することができる。しかし、Ethernet Jumbo Frame を用いたとしても、パケットサイズは高々10KB 程度である。そのため、GPU のプロセッサを効率良く利用するためには、全ての SM が同時に処理するデータサイズを増加させることが必要である。そこで、図 4 に示すように、1 つの SM に 1 つのパケットを割り当てることにより、複数のパケットのチェックサム計算を、複数の SM を用いて分散的に行う。各 SM は割り当てられたパケットに対して並列リダクションを用いてチェックサムを計算し、計算結果をグローバルメモリに格納する。また、チェックサム計算時のメモリアクセスを高速化するために、分散処理を行う前に、グローバルメモリ上のパケットを各 SM がもつ共有メモリに転送する。

さらに、1 つの SM に割り当てられたパケットのチェックサム計算に含まれる総和計算を、その SM の配下の SP 群に割り

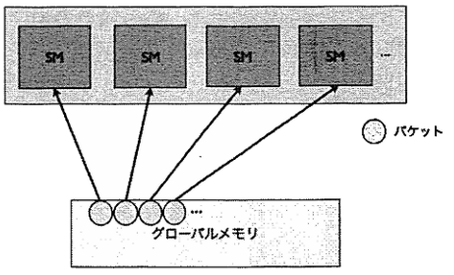

図 4　パケット分散処理手法

表 1　PC 環境

CPU	Core i5 3470 3.2GHz 2 コア
主記憶	8GB
HDD	1TB
NIC	Mellanox ConnectX3 10GBASE-T 2 ポート
OS	CentOS 5.9
GPU	NVIDIA Geforce GTX680

当てる。

4. 性能評価

本章では、3. 章において提案した GPU へのチェックサムオフロード手法を、評価用のチェックサム計算の簡易実装を用いて性能評価する。

4.1 評価環境

実験には、10 Gbps Ethernet リンクにより接続された 2 台の端末を用いる。2 台の端末をそれぞれ送信側端末および受信側端末とし、送信側端末から複数のパケットを送信し、受信側端末において受信パケットに対してチェックサム計算を実行する。各端末に用いた PC 環境を表 1 に示す。なお、受信側受信側端末にのみ GPU が搭載されている。また、評価に用いる GPU 環境を表 2 に示す。

実験では、チェックサム計算を CPU を用いて行った場合 (以降では CPU 実装と呼ぶ)、チェックサム計算を GPU にオフロードする場合 (GPU 実装)、及び、チェックサム計算を行わない場合 (チェックサム省略) の性能を比較する。性能評価指標としては、受信側端末において処理が終了したデータサイズを、データ転送時間で割った値をスループットと定義して用いる。図 3 にスループット測定に使用したパラメータを示す。以降の性能評価結果においては、10 回の試行の平均値を用いている。

4.1.1 性能評価結果

図 5 に、パケットサイズ (MTU) を変化させた場合における、チェックサム省略、CPU 実装、及び GPU 実装のスループットを示す。図から、特に MTU が大きい場合に、GPU 実装は CPU 実装に比べて高いスループットを示しており、チェックサム省略に近い性能を示していることがわかる。具体的には、GPU 実装によって、CPU 実装に対して最大 13%の性能向上を確認した。これは以下の理由によると考えられる。チェックサム計算はバイト単位処理であるため、パケットサイズが増加するにつれて、全体のプロトコル処理負荷に占めるチェックサム計算負荷の割合が大きくなる。そのため、パケットサイズが

表 2　GPU 環境

名称	NVIDIA Geforce GTX680
コア数	1536
SM 数	8
SP 数	192
コアクロック周波数	1.202 GHz
メモリ容量	2048MB
メモリインタフェース	256bit-GDDR5
メモリ帯域幅	192.2 Gbps
I/O バスインタフェース	PCI Express 3.0 x8
I/O バス帯域幅	片方向 64 Gbps
消費電力	195W

表 3　実験パラメータ

送信パケット数	10,000
パケットキュー長	1024
NIC バッファサイズ	8192
カーネル受信バッファサイズ (bytes)	268,435,456

表 4　データ転送中の CPU アイドル時間の割合

パケットサイズ (Bytes)	1480	9480
CPU 実装 (%)	0	0.99
GPU 実装 (%)	21.2	28.0

表 5　CPU 処理時間の割合

パケットサイズ (Bytes)	1480	9480
チェックサム計算 (CPU 実装)	35.5%	42.8%
カーネル・ユーザランド間パケットコピー (CPU 実装)	17.2%	19.7%
CUDA 関連の処理 (CPU 実装)	0%	0%
チェックサム計算 (GPU 実装)	0%	0%
カーネル・ユーザランド間パケットコピー (GPU 実装)	16.2%	21.0%
CUDA 関連の処理 (GPU 実装)	36.7%	28.2%
チェックサム計算 (チェックサム省略)	0%	0%
カーネル・ユーザランド間パケットコピー (チェックサム省略)	22.1%	34.2%
CUDA 関連の処理 (チェックサム省略)	0%	0%

増加すると、チェックサム計算を GPU へオフロードすることの効果が高くなる。

表 4 に、MTU が 1480 バイト及び 9480 バイトの場合における、データ転送中の CPU アイドル時間の割合を示す。この表より、CPU 実装の場合においては、データ転送中に CPU がアイドルになることがほとんど無いことがわかる。これは、データ転送処理に CPU の処理性能を使い切っていることを示している。一方、GPU 実装の場合にはアイドル時間が 20%以上存在する。この結果から、提案手法による、チェックサム計算の GPU へのオフロードにより、CPU の負荷を低下させることができているといえる。

表 5 に、各手法を用いたデータ受信処理において、受信側端末の CPU 時間が消費された処理とその割合を示している。この結果は、oprofile によって取得した。この表より、パケットサイズが 1480 バイトから 9480 バイトに増加することによって、CPU 実装におけるチェックサム計算の負荷が 35.5%から

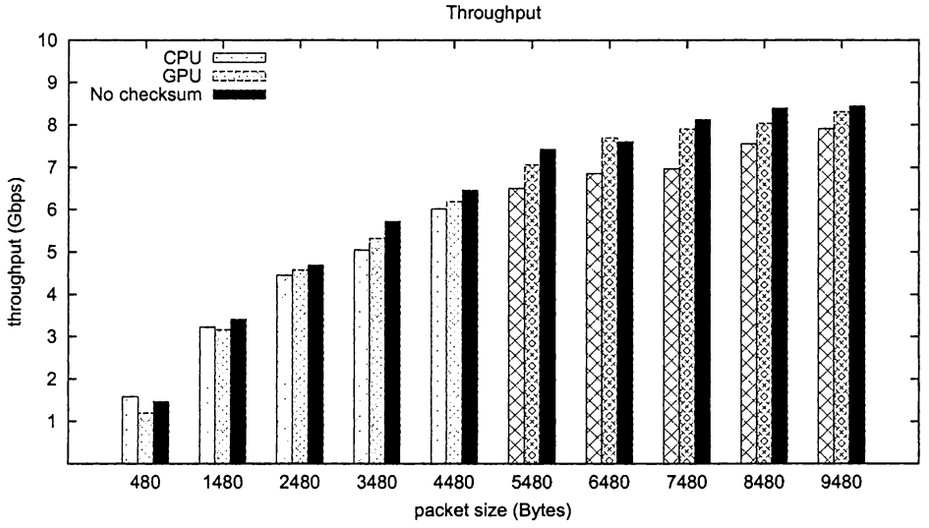

図 5 チェックサム省略、CPU 実装および GPU 実装のスループット

42.8%に増大していることがわかる。一方、GPU 実装における CUDA 関連の処理の負荷は、36.7%から 28.2%に低下している。これは、提案手法において、キューに格納するパケットサイズが大きくなった結果、CPU と GPU との間のパケット転送効率が向上するためであると考えられる。このことが、GPU 実装のスループット向上に繋がっていると考えられる。

これらの結果から、アプリケーション処理などに CPU 処理能力が使われているような環境下では、さらに GPU 実装と CPU 実装の性能差が現れると考えられる。このような環境下での性能評価は今後の課題としたい。

5. まとめと今後の課題

本報告では、GPU に TCP のチェックサム計算をオフロードすることにより、TCP/IP のプロトコル処理にかかる CPU 負荷を削減し、データ転送スループットを向上させる手法を提案した。具体的には、CPU と GPU との間のパケット転送効率を向上させるためのパケットキューイング手法、及び、GPU 上で複数のパケットを同時処理するためのパケット分散処理処理手法の 2 つを提案した。ユーザランドで動作するチェックサム計算の簡易実装により性能を評価し、チェックサム計算の GPU オフロードにより、CPU によるチェックサムを行う場合に比べてデータ転送スループットが最大で 13%向上することを示した。

今後の課題として、実装を改善することによるさらなるスループット向上が挙げられる。また、CPU 利用率などの評価指標の導入や、CPU 負荷が高い状況などの様々な環境における性能評価も重要である。さらに、IPSec の暗号化処理などの、高い CPU 負荷が発生する処理を GPU にオフロードさせる手法の提案なども行う予定である。

文 献

[1] A. L. C. Paul Willmann, Scott Rixner, "An evaluation of network stack parallelization strategies in modern operating systems," in *Proceedings of USENIX Annual Technical Conference*, pp. 91–96, May 2006.

[2] M. C. Wenji Wu, Phil DeMar, "A transport-friendly NIC for multicore/multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, pp. 607–615, Apr. 2012.

[3] A. Rijsinghani, "RFC 1624: Computation of the internet checksum via incremental update," *Internet Request for Comments*, vol. 1624, May 1994.

[4] N. Corporation, "CUDA programming guide version 5.0," July 2013.

[5] J. R. David D. Clark, Van Jacobson and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, vol. 27, June 1989.

[6] J. Chu and S. Inc, "Zero-copy TCP in solaris," in *Proceedings of the USENIX 1996 Annual Technical Conference*, Dec. 1996.

[7] J. Kay and J. Pasquale, "Profiling and reducing processing overheads in TCP/IP," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 817–828, Dec. 1996.

[8] L. Grossman, "Large receive offload implementation in neterion 10gbe ethernet driver," in *Proceedings of Ottawa Linux Symposium*, 2005.

[9] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda, "Performance characterization of a 10-gigabit ethernet TOE," in *Proceedings of the 13th Symposium on High Performance Interconnects*, 2005.

[10] V. A. Shourya P. Bhattacharya, "A measurement study of the linux TCP/IP stack performance and scalability on SMP systems," in *Proceedings of Communication System Software and Middleware*, Aug. 2006.

[11] W. d. B. Tom Herbert, "Scaling in the linux networking stack." available at `http://www.mjmwired.net/kernel/Documentation/networking/scaling.txt`.

[12] M. Harris, "Optimizing parallel reduction in CUDA," available at `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduct pdf`.