Vol.2014-OS-130 No.19
2014/7/29

Heterogeneous Architectureを活用したネットワークパケット処理の検討

杉木 章義[1,a)]　佐藤 聡[1]

概要：本論文では，Intel MIC アーキテクチャや CPU/GPU 統合プロセッサに代表される異種混在型アーキテクチャ上でのネットワークパケット処理について検討する．元々は GPGPU 用に記述された Longest Prefix Match (LPM)，Packet Classifier，Aho-Corasick の 3 つのアルゴリズムを OpenCL に移植し，Xeon Phi 上で予備評価を行った結果について報告する．また，汎用マルチコア CPU 上で処理を行った場合との性能比較を行い，今後のアプローチの有効性について検討する．

1. はじめに

近年，SDN（Software-Defined Network）や NFV（Network Functions Virtualization）が注目されており，従来の枠組みにとらわれない，高性能でより柔軟なネットワークパケット処理が必要とされている．従来からの ASIC や FPGA を拡張した高機能化によるアプローチも進められている一方で，Intel Xeon などの汎用的なプロセッサを活用したソフトウェアスイッチにも再び注目が集まっている．

ソフトウェアスイッチは古くから用いられている技術であるが [1], [2]，近年の仮想化技術の進展にともない，Open vSwitch [3] などの仮想スイッチを実現するソフトウェアが登場し，広く活用されている．また，Intel などのベンダーの努力により，PC 向けの汎用プロセッサの性能が大きく向上し，処理能力が 10 Gbps を超えるようなソフトウェアスイッチも実現可能となっている [4]．さらには，OpenOnload [5] や Netmap [6] などの最近の研究成果により，OS（Operating System）のユーザ空間で高速なパケット処理を実現するソフトウェア構成法も明らかになってきている．以上から，通信事業者を中心とした NFV の実現やネットワークアプライアンス機器ベンダーによる汎用プロセッサの活用が進められている．

現状の帯域であれば，Intel Xeon などの汎用プロセッサでもパケット処理を実現可能であるが，40 Gbps や 100 Gbps を超える帯域についても同様に実現可能であるか，慎重な検討が必要である [7]．また，製造プロセスの微細化によるプロセッサの性能向上も限界に近づきつつあり，今後も汎用プロセッサのみに頼った構成を適用すべきかどうか検討が必要である．

本研究では，異種混在型アーキテクチャ（Heterogeneous Architecture）上でのパケット処理に着目し，特に本論文では，Intel MIC（Many Integrated Core）アーキテクチャを活用したアプローチについて検討する．同じ異種混在型の GPGPU（General Purpose GPU）を活用した高速化については，PacketShader [8] や SSLShader [9] などの研究成果で明らかになっている一方で，Xeon Phi などの MIC アーキテクチャを活用した方式については，あまり探索されていない．本研究では，GPGPU を活用する方式と MIC を活用する方式について双方の利点や欠点を議論し，予備的な評価として Xeon Phi を活用したパケット処理に関する実験を実施する．

予備実験では，NVIDIA 製 GPGPU を対象として CUDA で記述された Longest Prefix Match（LPM），Packet Classifier，Aho-Corasick の 3 つのプログラム [10] を OpenCL に移植し，Xeon Phi 上で評価を行った．その結果，LPM に関して Xeon 汎用マルチコア CPU 上での処理に比べて大幅な性能向上が得られた．一方の Packet Classifier と Aho-Corasick に関しては，最適化などの不足により，十分な性能向上が得られなかった．

2. 関連研究

2.1 GPGPU の活用

GPGPU を活用したネットワークパケット処理への応用に関しては，広く研究が進められている．ごく初期の研究として，PacketShader [8] や SSLShader [9] がある．PacketShader では，IP フォワーディングや，OpenFlow スイッ

[1] 筑波大学システム情報系（学術情報メディアセンター）
University of Tsukuba, Tsukuba, Ibaraki 305–8573, Japan
[a)] sugiki@cc.tsukuba.ac.jp

チ，IPSec の暗号化などへの適用の可能性が当時のハードウェアで手広く調べられている．一方の SSLShader では，SSL 通信暗号化や復号化の高速化を目指している．また，GPGPU を活用したこの他の応用に関しても広く研究が進められており，IP アドレスのルックアップ [11] に使用する研究，Content-Centric Network の名前解決 [12] に使用する研究，パケット識別（classification）[13]，パターンマッチ [14] に関する研究，侵入検知システム [15] に応用する研究などがある．また，この他にも Click Modular Router [1] と組み合わせ，柔軟な構成変更を目指した Snap [16] などがある．さらには，異種混在型のマルチコアの活用は耐障害性の面からも検討されている [17]．

本研究では，これらの研究とは異なり，MIC アーキテクチャを活用したネットワークパケット処理の可能性について調査する．GPGPU と MIC のアーキテクチャには共通する部分，相互に異なる部分があり，これらの影響に関して調査する．

2.2 ソフトウェアスタックの改良

一般にソフトウェア実装によるネットワークパケット処理では，ASIC や FPGA などによるハードウェア実装に比べて，柔軟性や拡張性に優れる反面，速度や安定性などが犠牲となっていた．

従来のソフトウェアによるパケット処理の高速化は，カーネル版の Click など，カーネル空間に移植することで高速化を実現していた．カーネル空間に移植する方法では，切り替えによるオーバヘットが低減できる反面，障害やデバッグの際に問題となっていた．また，ユーザ空間の既存のライブラリの活用などに制約があった．

近年，OpenOnload [5] や Netmap [6], [18] などの成果により，ドライバから OS カーネルのパケット処理をバイパスし，ユーザ空間で処理を行うことで，10 Gbps を超える速度で処理できることが分かっている．OS では，従来より BPF（Berkeley Packet Filter），PF_PACKET，PF_RING，libpcap などのパケット処理用のインターフェースが提供されていたが，これらに比べるとはるかに高速になっている．さらに最近では，Intel DPDK（Data Plane Development Kit） [19] などの高品質なライブラリも入手可能となっており，より活用しやすくなっている．

前述の PacketShader や SSLShader などでも，パケット処理用のソフトウェアスタックの改良は行われていたが，本研究ではこれらの成果も活用する．

2.3 アーキテクチャの改良

Intel や AMD などの汎用プロセッサは，アーキテクチャ上，非常に成熟した段階に入っており，さまざまなプログラムに対して安定的に高い性能が実現できるようになっている．ムーアの法則に従い，これまで長期にわたり性能が大幅に向上してきたが，ILP（Instruction-Level Parallelism）や TLP（Thread-Level Parallelism）の改良も難しくなってきていることから，マルチコア化による性能向上へと進んでいる [20], [21]．そのため，プログラマによる明示的な並列性の活用が必要となっている．また，製造プロセスの微細化による余裕を活用して，メモリコントローラや PCI コントローラの汎用 CPU への統合が進んでおり，その点でも I/O 処理の高速化が図られている．

GPGPU や MIC のようなアプローチは，ダイ面積や消費電力，熱容量あたりの性能が高く，近年の性能向上が難しい中で，魅力的な選択肢となっている．特に，ネットワークのパケット処理は元々，並列性が高く，活用しやすいアプリケーションとなっている．

本研究では，特に MIC アーキテクチャを活用した異種混在型構成でのネットワークパケット処理の可能性を探索する．また，将来的には CPU と GPU を統合した統合プロセッサの活用も視野に入れる．

表 1 GPGPU と MIC アーキテクチャの比較

	Xeon Phi 5110P	Tesla K40
Generation	Knights Corner	Kepler
Cores	60 Cores	14 SMX
Logical Cores	240 H/W Threads	2,880 CUDA Cores
SIMD	512-bit SIMD	—
Frequency	1.053 GHz	745 MHz (w/Boost)
Peek (double)	1.011 TFLOPS	1.43 TFLOPS
Peek (single)	2.022 TFLOPS	4.29 TFLOPS
Memory	8GB (GDDR5)	12 GB (GDDR5)
Memory B/W	320 GB/s	288 GB/s
Bus	PCIe 2.0	PCIe 3.0
Power	225W	235W
Price (Approx.)	$2,500	$5,500

3. MIC アーキテクチャの活用

3.1 GPGPU と MIC の概要

表 1 に GPGPU と MIC アーキテクチャを比較した表を示す．両者のアーキテクチャに共通するのは，両者ともに汎用 CPU よりも低速で小さな論理コアを採用する代わりに，多数のコアを並列に並べることで，全体で高い性能を実現することを目指している点である．以上から，ダイ面積あたりの計算能力，および電力効率に優れる一方で，明示的なプログラミングによる活用が必要となっている．

全体の目標は共通する一方で，これらの細部の設計思想は異なっている．GPGPU では，SIMT（Single Instruction Multiple Thread）を採用し，複数のスレッドをウォープ（Warp）という単位でまとめ，スレッドを同時にスケジュール実行することでダイ面積を削減しつつ，効率を高めている．ウォープ内ではプログラムカウンタが共通となっており，プログラム中に分岐が存在した場合には，マスク実行

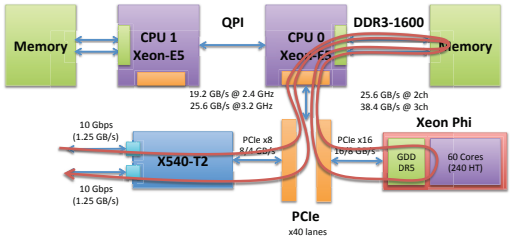

図 1 ネットワーク I/O 処理におけるバス構成

が行われ，性能が低下する．

一方の MIC アーキテクチャでは，Pentium 相当ともいわれるより大きな汎用コアを多数並べることで，高い並列性能を実現している．こうすると GPGPU に比べ，実装できる論理コアは少なくなるが，一方で 512 ビットの SIMD 命令を提供することで，1 TFLOPS 以上の理論性能を実現している．プログラムカウンタの値はそれぞれのコアで独立して保持しているが，一方で SIMD 命令の活用が，効率的な実行のための鍵となっている．

また，より詳細な点として，GPGPU は単体では稼働できず，汎用 CPU に対するアクセラレータの位置づけであるのに対して，MIC では独立した OS が稼働しており，単体で稼働することも可能である．

3.2 GPGPU と MIC に共通する問題

データ転送コスト

GPGPU や MIC の双方をネットワークパケット処理に応用した場合，両者は多くの問題を共有している．最も大きな問題は，一般的な GPGPU における問題と同様に，PCI バスを経由した DMA の転送コストである．そのため，CPU で処理を行うのか，PCI 経由の DMA 転送のコストを超えて GPGPU や MIC で処理を行うのか検討する必要がある．

図 1 にネットワークパケット処理におけるバス構成を示す．(1) まず，Network Interface Card (NIC) に到着したパケットは，DMA 転送により，ホスト CPU のメインメモリに書き込まれる．(2) 次にメインメモリから，GPGPU または MIC のデバイスメモリに再度 DMA 転送が行われる．デバイスメモリからデータを取りながら GPGPU または MIC 上で処理が行われたあと，(3) メインメモリに再度 DMA 転送が行われる．(4) 最後に，NIC に対して DMA 転送を行い，処理が完了する．以上，4 回の転送処理が発生するが，最近の汎用 CPU は NUMA 構成をとっており，異なる CPU 配下の PCI バスやメモリ上に転送先が存在する場合は，さらに QPI 経由での転送コストが発生する．

上記の転送コストを削減する方法として，(1) メインメモリへのコピーを削減する手法，(2) デバイスメモリへのコピーを削減する手法の 2 つが考えられる．

前者のメインメモリのコピーを抑制する手法に関しては，NVIDIA が他の PCI デバイスや他の GPGPU から直接 GPGPU に対して RDMA (Remote DMA) を行う手法として GPUDirect [22] を提案している．しかしながら，利用できるのは Mellanox 製カードでの InfiniBand 転送などごく一部に限られ，広く一般に利用可能にはなっていない．

また，MIC アーキテクチャの Xeon Phi ではホスト CPU と別の OS が稼働しており，PCI Express (PCIe) バスを経由して IP アドレスで通信可能であるが，外部と直接通信できず，何らかのホスト CPU の関与が必要である．

後者のデバイスメモリへのコピーを削減する手法に関しては，AMD の APU (Applicaiton Processing Unit) 統合プロセッサなどのように，汎用 CPU と GPGPU 間でメモリ共有すれば，コピーを削減することができる．ただし，利用可能なダイ面積の関係から，GPGPU の性能やメモリ転送レートに制約がある．

さらに，GPGPU や MIC に処理をオフロードする場合には，Intel Data Direct I/O [23] の効果も大きく低減する．最近の Xeon-E5 や-E7 では，NIC からメインメモリではなく，汎用 CPU の Last Level Cache (LLC) に対して直接転送を行う．オフロードを行う場合にも，LLC からデバイスメモリに転送を行うことができるが，汎用 CPU のみで処理を行う場合に比べて，遅延時間は増加する．

以上から，ネットワークパケット処理を転送コストをこえて GPGPU または MIC へオフロードすべきかどうか，慎重な検討が必要である．

効率的な資源の活用

もう一つの問題は，理論性能と実効性能の乖離である．GPGPU や MIC では，各コアの利用率を高める工夫が必要であるが，ネットワークパケット処理では，外部から入力となる要求がパケットとして流入するため，GPGPU や MIC のコアの利用率を直接制御することができない．

この問題に関しては，すでに PacketShader や RouteBricks などでバッチ処理が有効であることが示されており，MIC アーキテクチャでも同じ手法を活用することができる．

限定された局所性の活用

最後に，ネットワークパケット処理では，大量のパケットが入力データとして流入され，同じく大量にパケットが出力される．従って，入出力データに関して，同じデータを再利用することが少ないことから，I/O コストが重要で，デバイスメモリの活用やキャッシュなどの効果が現れにくい．再利用の効果が現れるのは，処理の際の参照テーブルや集計データなど，ごく一部に限られる．また，ネットワークパケット処理では，処理のスループットに加えて遅延時間も重要で，その点からも適用できる計算処理の内

容に制約がある．

3.3 MIC 活用の利点

GPGPU と MIC では，多くの問題や解決策を共有する反面，MIC アーキテクチャには GPGPU にはない利点がある．

- **タスク並列モデルの活用：** GPGPU では，SIMT を採用している関係から，データ並列モデルを使用した方が有利であるが，MIC では，それぞれのコアが基本的に独立して動作するため，タスク並列とデータ並列の双方のモデルの活用が可能である．特に，タスク並列モデルを使用した場合，さまざまなパケット処理に関係するタスクを混在させた場合の性能低下を GPGPU と比較して抑えられる可能性がある．また，MIC は SIMT ではないため，branch divergence に強く，この点に関しても性能低下が抑えられることが期待される．
- **高い移植性：** MIC アーキテクチャでは，一般的にうたわれているように x86 命令が利用可能であることから，汎用 CPU からのプログラムの移植が容易である．一方で，ベクトル化し，SIMD 命令を活用しなければ期待した大幅な性能向上は得られない．

4. 予備実験

4.1 実験環境

予備実験には，NEC Express5800/HR120a-1 を 1 台使用した，ハードウェア構成の内訳は，Xeon E5-2640 2.50 GHz（6 コア/12 スレッド）を 2 ソケット，64 GB DDR3L-1600 メモリ，1TB SATA HDD 1 台，Xeon Phi 5110P（60 コア，1.053 GHz，8 GB メモリ）である．一方のソフトウェア構成では，64 bit 版の CentOS 6.5（Linux 2.6.32），MPSS 3.2.1，Intel SDK for OpenCL 4.4.0 を使用した．

4.2 実験に使用したプログラム

本実験では，NVIDIA 製 GPGPU を対象に CUDA で記述されたプログラムを OpenCL に移植し，実験を行った．実際には，Snap [16] の論文の実験プログラムのもとになったと推測される G4C [10] を使用し，改良した．

実験では，下記の 3 つのプログラムを使用した．

- **Longest Prefix Match（LPM）：** レイヤ 3 のルーティングで頻繁に使用される LPM を実装したプログラムである．非常に単純なトライ木を実装し，検索を行う．処理対象の各エントリにつき，32 ビットの IPv4 アドレスを入力し，8 ビットのポート番号を出力する．
- **Packet Classifier：** SDN 環境を想定し，送信元・送信先 IPv4 アドレス，送信元・送信先 TCP ポート番号，プロトコル番号の 5 タプルがルールに一致するかどうかを検査する．128 バイトのパケットヘッダの一部を入力し，32 ビットのルールに一致したパターン番号を出力する．
- **Aho-Corasick：** IDS（Intrusion Detection System）などで頻繁に用いられる Aho-Corasick 方式を実装したプログラムである．最大 1,024 バイトの文字列を入力し，32 ビットの一致したパターン番号を出力する．

Intel によって記述された論文 [24] などで指摘されているように，より公平な比較を目指す観点から，汎用 CPU のみを使用するプログラムも単一 CPU のみで動作するものからマルチスレッド化し，実験を行った．

また，本論文では，研究期間の制約からメインメモリ上にあらかじめ読み込まれたデータを Xeon Phi 上で処理を行い，メインメモリに結果を格納する部分のみで評価を行う．これは，ネットワークパケット処理のパイプライン上で，上記の部分が性能上のボトルネックになると予測されるからである．よって，NIC との入出力を含めた統合評価については今後の課題とする．

さらには，本実験ではピーク性能を測定するため，特に CPU キャッシュの効果を妨げないものとする．最後に，今回実験で使用した 3 つのアルゴリズムに関しては，さまざまな改良版が提案されているが，基本的な振る舞いについて把握するために，単純なアルゴリズムを使用する．

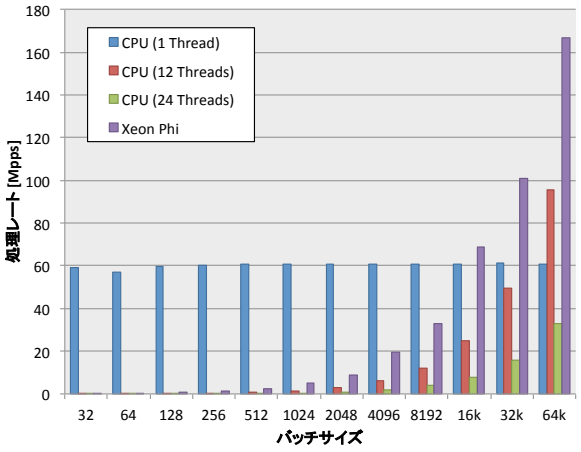

図 2 LPM に関する実験結果

4.3 実験結果

LPM の実験結果

最初に，LPM に関する実験結果を**図 2** に示す．横軸のバッチサイズは Xeon Phi で一度にバッチ処理を行う要求数，縦軸の処理レートは単位時間あたりに要求処理を完了できた数である．本実験では，Xeon Phi を活用する場合，汎用 CPU のみを使用する場合に比べて大幅な性能向上が得られている．汎用 CPU のみを使用する場合については，単一コアを使用する場合，入力に関わらず，約 60 Mpps と一定になっており，バッチサイズが 64K となった場合には，全ての 12 コアを活用する場合とスループットが逆転する．また，汎用 CPU がサポートする Hyper Threading

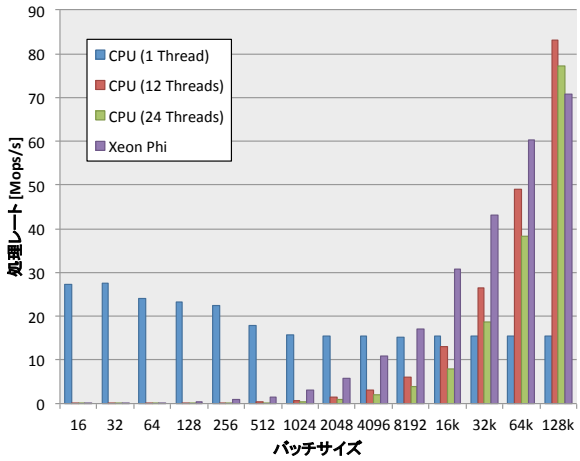

図 3 Packet Classifier に関する実験結果

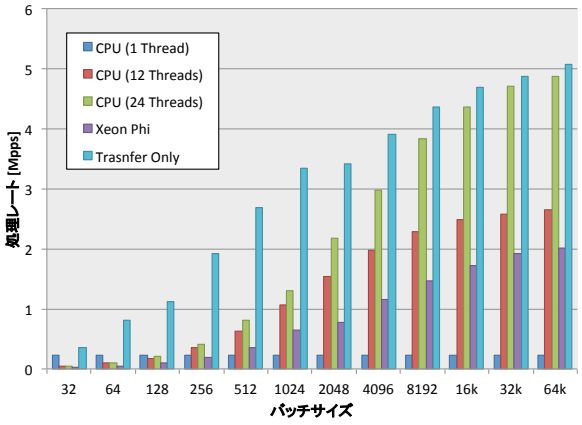

図 4 Aho-Corasick に関する実験結果

（HT）数に対応する 24 スレッドを使用した場合には，12 スレッドの場合に比べて性能が低下している．

Packet Classifier の実験結果

次に，Packet Classifier に関する実験結果を**図 3** に示す．本実験の場合，バッチサイズが 8,192 から 64K の場合で Xeon Phi のスループットが最大となっている．バッチサイズが 128K の場合には，汎用 CPU で 12 スレッドおよび 24 スレッドを使用する場合と性能が逆転する．汎用 CPU を使用する場合については，単一コアの場合，キャッシュの効果により，バッチサイズが大きくなるに従って性能が低下している．また，HT を使用する場合と使用しない場合については，12 スレッドを使用する場合の方が性能が高くなっている．

Aho-Corasick の実験結果

最後に，Aho-Corasick に関する実験結果を**図 4** に示す．本実験の場合については，Xeon Phi の性能が汎用 CPU の性能を下回っている．汎用 CPU のみを使用する場合については，単一コアのみを使用する場合は性能が総じて低くなっており，一方で 12 物理コアを使用する場合に比べて，HT を使用する方が性能が高くなっている．本実験では，メモリのワークセットが大きいためその影響が考えられるが，Xeon Phi への転送のみを行う場合についても参考のために図 4 に示してある．

4.4 予備実験から得られた知見

本実験では，G4C の公開されている版に多数の細やかな問題があり，移植や改良に多くの時間を要した．そのため，性能の最適化には多くの余地がある．特に，MIC アーキテクチャでは，性能の向上のために SIMD 命令の活用が必要であるが，Intel の OpenCL ランタイムがワークグループの次元 0 に対して自動ベクトル化を行うものの，どの程度行われているか確かめる必要がある．手動ベクトル化に関しては，経験の不足もあり，期待した性能向上は得られなかった．一方で，汎用 CPU である Xeon の方も SSE/AVX 命令により，256 bit 幅の SIMD 演算が可能であり，これらを活用した比較も必要である．Intel 製の OpenCL ランタイムでは自動的な SIMD 命令への変換が行われるものの，今回の実験では，これらの命令は活用していない．

これ以上の最適化はブラックボックス環境では難しく，Intel OpenCL SDK の静的コンパイラが出力するアセンブラの解析や，VTune Amplifier などでの詳細な実行時性能プロファイルの取得が必要である．

この他にも，GPGPU と MIC で異なる最適化が必要であることが確認された．元々の G4C の Packet Classifier はローカルメモリ（共有メモリ）を活用するように記述されていたが，Xeon Phi ではグローバルメモリで代替されるため，ローカルメモリを使用しないように変換を行った．また，GPGPU では，branch divergence を避けるため，分岐方向を揃える工夫が必要であったが，MIC では，早くループを抜けた方が性能が向上する場面も一部で確認された．

最後に，今回は移植性を考慮して OpenCL を使用したが，Xeon Phi ではオフロード実行モデルとネイティブ実行モデルの選択が可能であり，また，ライブラリも OpenMP，TBB，MPI，Click Plus，C/C++データ拡張などのさまざまな選択肢がある．よって，今後も Xeon Phi で研究を継続する場合には，OpenCL 以外を選択した方が，性能向上のためのディレクティブなどが充実している可能性もあり，詳細な検討が必要である．

5. まとめと今後の課題

本論文では，異種混在型アーキテクチャの活用を想定し，Xeon Phi を対象に LPM，Packet Classifier，Aho-Corasick の 3 つのプログラムを使用して予備実験を行った．その結果，LPM と Packet Classifier の一部で汎用 CPU の Xeon-E5 に比べて高いスループットが得られた．

MIC アーキテクチャに関しては，次世代の Knights Landing では，独立したソケットタイプの製品が出荷されるこ

とが Intel より発表 [25] されており，現状のコプロセッサではない処理手法も可能である．また，Silvermont ベースのコア，Omni Scale Fabric，高速なメモリ群の採用が発表されており，これらによる性能向上も期待できる．また，現状の世代の Xeon Phi に関しても，性能の面で明らかでない点も多く，詳細な性能解析 [26] も進められている．

また，PCIe バスの転送コストが問題となっているが，AMD 製の APU [27] などでは，統合された CPU と GPGPU 間でメモリ共有が行われており，Kaveri 以降ではキャッシュの一貫性も保たれるようになった．利用可能なダイ面積の関係から，統合型プロセッサでは，多数の GPGPU の計算コアを搭載することができないが，サーバのネットワーク処理のオフロードに活用することが期待される．統合メモリに関しては，NVIDIA に関しても，実装形態は大きく異なるが，CUDA 6 [28] 以降でサポートされている．

最後に汎用 CPU の動向に関しても，これまでメモリや PCIe のコントローラが CPU に統合されたように，製造プロセスの微細化にともないネットワークコントローラの統合も進む可能性がある．実際に Atom C2000 シリーズでは，ネットワークコントローラの統合が行われており，こちらの動向の注視も必要である．

参考文献

[1] Kohler, E., Morris, R., Chen, B., Jannotti, J. and Kaashoek, M. F.: The Click Modular Router, *ACM TOCS*, Vol. 18, No. 3, pp. 263–297 (2000).
[2] Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M. and Ratnasamy, S.: RouteBricks: Exploiting Parallelism to Scale Software Routers, *ACM SOSP'09*, pp. 15–28 (2009).
[3] Pfaff, B., Pettit, J., Koponen, T., Keith Amidon, M. C. and Shenker, S.: Extending Networking into the Virtualization Layer, *HotNets-VIII* (2009).
[4] Intel Corp.: Impressive Packet Processing Performance Enables Greater Workload Consolidation, Technical report, Intel Corp. (2012).
[5] Solarflare: OpenOnload (2008). `http://www.openonload.org/`.
[6] Rizzo, L.: netmap: A Novel Framework for Fast Packet I/O, *USENIX ATC '12*, pp. 101–112 (2012).
[7] Han, S., Jang, K., Park, K. and Moon, S.: Building a Single-Box 100 Gbps Software Router, *lANMAN'10*, pp. 1–4 (2010).
[8] Han, S., Jang, K., Park, K. and Moon, S.: PacketShader: a GPU-Accelerated Software Router, *SIGCOMM'10*, pp. 195–206 (2010).
[9] Jang, K., Han, S., Han, S., Moon, S. and Park, K.: SSLShader: Cheap SSL Acceleration with Commodity Processors, *USENIX NSDI'11* (2011).
[10] Sun, W.: G4C (2012). `https://github.com/wbsun/g4c`.
[11] Zhao, J., Zhang, X., Wang, X. and Xue, X.: Achieving O(1) IP Lookup on GPU-based Software Routers, *SIGCOMM'10*, pp. 429–430 (2010).
[12] Wang, Y., Zu, Y., Zhang, T., Peng, K., Dong, Q., Liu, B., Meng, W., Dai, H., Tian, X., Xu, Z., Wu, H. and Yang, D.: Wire Speed Name Lookup: A GPU-based Approach, *USENIX NSDI'13*, pp. 199–212 (2013).
[13] Kang, K. and Deng, Y. S.: Scalable Packet Classification via GPU Metaprogramming, *DATE' 11*, pp. 1–4 (2011).
[14] Lin, C.-H., Liu, C.-H., Chang, S.-C. and Hon, W.-K.: Memory-efficient Pattern Matching Architectures using Perfect Hashing on Graphic Processing Units, *IEEE INFOCOM'12*, pp. 1978–1986 (2012).
[15] Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E. P. and Ioannidis, S.: Gnort: High Performance Network Intrusion Detection Using Graphics Processors, *RAID '08*, pp. 116–134 (2008).
[16] Sun, W. and Ricci, R.: Fast and Flexible: Parallel Packet Processing with GPUs and Click, *ACM/IEEE ANCS'13*, pp. 25–36 (2013).
[17] Hruby, T., Bos, H. and Tanenbaum, A. S.: When Slower Is Faster: on Heterogeneous Multicores for Reliable Systems, *USENIX ATC'13*, pp. 255–266 (2013).
[18] Rizzo, L., Carbone, M. and Catalli, G.: Transparent Acceleration of Software Packet Forwarding using netmap, *IEEE INFOCOM'12*, pp. 2471–2479 (2012).
[19] Intel Corp.: Packet Processing on Intel Architecture (2012). `http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/packet-processing-is-enhanced-with-software-from-intel-dpdk.html`.
[20] Hennessy, J. L. and Patterson, D. A.: *Computer Architecture*, Morgan Kaufmann, 5th edition (2011).
[21] Patterson, D. A. and Hennessy, J. L.: *Computer Organization and Design*, Morgan Kaufmann, 5th edition (2013).
[22] NVIDIA Corp.: NVIDIA GPUDirect (2010). `https://developer.nvidia.com/gpudirect`.
[23] Intel Corp.: Intel Data Direct I/O Technology (2012). `http://www.intel.com/content/www/us/en/io/direct-data-i-o.html`.
[24] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R. and Dubey, P.: Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU, *ISCA'10*, pp. 451–460 (2010).
[25] Intel Corp.: Intel Re-architects the Fundamental Building Block for High-Performance Computing, Technical report, Intel Corp. (2013).
[26] Fang, J., Sips, H., Zhang, L., Xu, C., Che, Y. and Varbanescu, A. L.: Test-driving Intel Xeon Phi, *ICPE '14*, pp. 137–148 (2014).
[27] Advanced Micro Devices, Inc.: AMD and HSA (2013). `http://www.amd.com/en-us/innovations/software-technologies/hsa`.
[28] NVIDIA Corp.: CUDA Toolkit (2014). `https://developer.nvidia.com/cuda-toolkit`.