第 VII 部

ネットワークプロセッサを用いたアプリケーション開発

第 7 部
ネットワークプロセッサを用いたアプリケーション開発

第 1 章　はじめに

1.1 ネットワークプロセッサとは

ネットワークプロセッサ（以下 NP と略記）とは、ネットワークトラフィックを処理するための専用プロセッサである。

1.1.1 NP の特徴

現在多くの企業がそれぞれ異なるアーキテクチャで NP を開発しているが、NP の大まかな構成は以下のようなものである。

NP の構成は、1 つもしくは複数の処理要素（Processing Element：以下 PE と略記）と制御用の汎用コア・プロセッサ、Hash 計算など複雑な処理を専門に行うハードウェア、内部のキャッシュメモリ、そして外部メモリやネットワークインタフェースなどの周辺機器とのインタフェースからなる。

PE は高速なパケット処理を専門に行う要素であり、ハードウェアによるマルチスレッド化により、メモリの読み書き、ネットワークインタフェースからの割り込みなどの I/O 処理によって生じる処理時間を隠蔽することによって処理速度の向上を図る工夫がなされている。

PE は高速なパケット処理を実現できるが、ルーティングテーブルの書き換え、エラーパケットの処理など複雑な処理は行えない。複雑な処理は制御用に用いられている汎用コア・プロセッサによって処理される。

NP の特徴として、ソフトウェアによるプログラムが可能であるという特徴がある。PE 上で動くパケット処理用のプログラムと Core Processor 上で動く制御用コードを NP 上にロードし起動することで NP 上のアプリケーションは動作する。

1.1.2 近年の NP への注目の高まり

近年 NP が注目されている理由は NP のもつ処理能力の高さとソフトウェアによりプログラム可能な特徴のためである。

NP が注目されるようになった背景として、まず、ASIC（Application-Specific Integrated Circuits）を用いた開発に莫大な費用と時間がかかることである。近年のルータ実装においては、高速化を実現するために ASIC を用いたパケット処理プロセッサをネットワークインタフェース近傍に配置しているが、今後の超高速ネットワークへの対応や新しく標準化されたプロトコル、新しいサービスなどの機能を即座に ASIC を用いたルータに取り入れていくことは開発コストの面で非常に難しい。NP を用いると、ソフトウェアによるプログラムが可能で並列化による性能向上が容易に実現できる。そのため、NP を用いることで短期間で安価な製品開発が可能になると期待されている。

また、アクティブネットワーク技術に代表されるように、より高次のネットワークサービスの構築が望まれるようになったことである。アクティブネットワーク技術は、ネットワークサービスをエンドノードによる処理だけでなく、ルータなどの中継ノードにおいても処理を行うことでより動的なサービスを実現するものである。より柔軟なサービスを実現し、かつ高速処理を行うことが求められているため、NP の処理性能とプログラム可能な特徴に注目が集まっている。

最後に、ゲートウェイやサーバホスト等のエンドノードの性能向上が求められていることが挙げられる。近年のネットワーク帯域向上はプロセッサの処理能力の向上に比べ遥かに大きい。そのため、エンドノードにおける TCP/IP 処理やアプリケーション層の処理の一部を NP に移すことで処理能力の向上が図れるのではないかと期待されている。

1.2 ネットワークプロセッサ・ワーキンググループの設立

2003 年 3 月に WIDE Project 内において NP の技術調査・研究を目的としてネットワークプロセッサワーキンググループ（以下 NP-WG と略記）を設立した。

1.2.1 活動目的

WIDE Project 内でも NP に関連した研究は陣崎ら [348] や星野ら [355] などにより以前から行われてきた。また、それ以外の参加企業、団体、個人においても NP に関する関心は高まっており、個々に調査・研究が進められてきている。しかし、NP の処理性能や、可能性はまだ未知数な部分が多く、NP によって実現可能なこと、NP を用いることで解決できる問題、NP の特性を生かしたアプリケーションが何であるかなどは明らかになっていない。そこで、大学、企業、個人が集まる WIDE Project において、さまざまな観点から評価・検討し、NP の持つ可能性または技術的限界を議論していくことを NP-WG の目的のひとつとしている。

また、NP を用いたアプリケーション開発において、PE 上で動かすプログラムには、その性質上、非常に低レベルなプログラミングが要求される。開発環境や開発に用いる言語、用いる NP の特性を熟知する必要があるが、若い分野であるため参考となるドキュメントも乏しく、熟知するまでには多くの労力と時間を費やす。そのため、C 言語などを用いた汎用プロセッサ上でのソフトウェアプログラミングよりも開発のノウハウを覚えるまでの時間がかかり、NP を研究の実装手段として用いる際の大きな障壁となっている。

そこで、NP を用いた開発に携わったことのあるワーキンググループの参加者が開発で得た知見をメーリングリストなどを通じて交換し、それを取りまとめて web などを用い公開することで、新たに NP を用いた開発を行う者へのプログラミングにおける障壁を下げることも目的としている。

1.2.2 活動内容

2003 年 3 月に行われた WIDE camp 内の NP-WG BoF で決定した NP-WG の短期的な活動内容は以下の項目である。

1. NP によって実現可能な範囲、NP を活かすことのできる応用分野の絞込み
2. NP を用いたアプリケーション開発
3. NP を用いたアプリケーション開発を行う上での環境整備、情報共有
4. NP を用いた研究開発を行える人材育成

NP の技術研究開発はこれまで産業界が主導で行っており、そのため非公開な情報も多く、また研究成果の学術的な体系化も不十分である。そのため、NP を用いることによって解決できる問題は何か、NP によって新たに実現できる機能は何か、NP を導入できる研究分野は何か、といったことは明らかになっていない。NP の持つポテンシャルを調査し、NP を利用することで問題解決が図れる応用分野、NP を用いることでは問題解決とならない研究分野の整理を行うことが重要であり、NP-WG での活動内容のひとつとしている。

一方で、比較的容易に入手できる NP 搭載ボードおよび開発環境を用いてのアプリケーション開発を意欲的に行っていくことも活動方針のひとつである。NP 自体は実装手段の 1 つであるため、多分野にわたって多くの研究者が NP を研究の応用に用いることが望まれる。

しかし、NP におけるソフトウェア開発には低レベルなプログラミングが要求され、用いる NP のマイクロアーキテクチャ、メモリ構造、開発環境に熟知する必要がある。多くの場合、参考となる文献は開発環境のマニュアルしかなく、用いる NP や開発環境の特性を理解するためにかなりの労力を費やしてしまい、新たに NP を応用に取り入れる場面において障壁となっている。

そこで、実際に開発を行っていくうえで得た知見をまとめ、それを広く公開することにより NP を用いたアプリケーション開発を行う上で役立つ情報の共有を行っていく。それにより、NP を用いたアプリケーション開発を行いやすい環境を整備し利用することで、NP を用いた研究開発を行える人材の育成を目指す。この人材育成は、NP の開発環境がある奈良先端科学技術大学院大学と北陸先端科学技術大学院大学が率先して行っていく。

第 2 章　今年度の活動

2.1 投稿論文

今年度の NP-WG の成果として、2003 年 5 月の IA 研究会（電子情報通信学会インターネットアーキテクチャ研究会）にワーキンググループのメンバーから 3 本の論文が発表された。

奈良先端科学技術大学院大学の河合栄治らは「ネッ

トワークプロセッサ技術に関するサーベイ」[344] によりネットワークプロセッサに関して主に学術機関から発表された数々の論文をアーキテクチャ、性能評価、プログラミング、アプリケーションの 4 分野に分類しまとめている。

また、NP を用いたアプリケーション開発の例として、横河電機の田中貴志らにより「NP のトレースバックシステムへの応用」[350] が投稿され、パケットの通過記録を残す Footmaker を NP 上で実装したことにより、汎用 PC と比べパケット長の短い場合においても理論限界値に処理性能を発揮したことを示している。

北陸先端科学技術大学院大学の渡辺林音らによる「ネットワークプロセッサを用いた MPLS ラベルスイッチングルータの実装」[360] では、汎用 PC 上で研究・実装が行われてきた MPLS の研究プラットホームである AYAME の開発に NP を応用し、AYAME における高速パケットスイッチングの実現に関して考察している。この論文では外部接続された NP 搭載ボード NAPPI[122] でネットワークスタックのパケット処理部分を処理することにより、ソフトウェア実装の利点を生かした高速ルータ実現の可能性を示している。また、汎用 PC 上で動作する OS と NP 上で動作する OS との間でのデータ交換、ルーティングテーブルなどの重要なデータの同期等、汎用 PC 上での実装に NP を応用する場合の一例として、応用時に出現する問題へ 1 つの回答を示しており、他の汎用 PC 上のソフトウェア実装を NP へ応用する際に役立つ多くの知見をこの論文から得ることができる。

以下、第 3 章で河合らによる「ネットワーク技術に関するサーベイ」を、第 4 章で田中らによる「NP を用いたトレースバックシステムへの応用」を、第 5 章において渡辺らによる「ネットワークプロセッサを用いた MPLS ラベルスイッチングルータの実装」を掲載する。

第 3 章　ネットワークプロセッサ技術に関するサーベイ

本章では、2003 年 5 月に行われた電子情報通信学会で河合らによって発表された論文を掲載する。

概要

今後の超高速ネットワーク時代を実現する技術の 1 つとして、ネットワークプロセッサ（NP）が注目を集めている。NP とは、ネットワーク処理のための専用のプロセッサであり、ルータなどの中継装置だけでなく、サーバなどのエンドホストにも搭載することができる。本稿では、この NP 技術およびその研究動向に関してサーベイを行う。具体的には、アーキテクチャ、性能評価、プログラミング、アプリケーション技術の各分野について論じる。

Abstract

Network processor (NP) is a promising technology that will support tomorrow's ultra high speed networking. The NP is a sub-system dedicated to network processing with special processors, memory buffers, and network interfaces, and it can be used not only in network routers but in server hosts. In this paper, we survey the NP technologies and its research trends. We discuss NP architectures, performance evaluation scheme, programming environments, and its application.

3.1 はじめに

近年のネットワークプロセッサ技術の発展は目覚ましいものがある。ネットワークプロセッサ（以下 NP と略す）とは、ネットワークトラフィックを処理をするための専用のプロセッサであり、ルータなどの中継ノードだけでなく、サーバなどのエンドノードでも利用が可能になってきている。もともと、高レベルな処理を外部の補助プロセッサで処理するというアイデア自体は新しいものではない [40, 233]。現在、NP が注目されるようになった背景には、次のような点が挙げられる。

まず、ルータのパケット転送能力の向上および QoS（Quality of Service）機能などの新しい機能の搭載を短時間で実現するためである。近年のルータの実装においては、高速化を実現するために ASIC（Application-Specific Integrated Circuits）を用いたパケット処理プロセッサをネットワークインタフェースの近傍に配置している [50]。しかし、今後の超高速ネットワークをサポートするためには開発に莫大な費用と時間がかかるため、ソフトウェアに

よるプログラムが可能で並列化による性能向上が容易な NP を用いることで、短期間で安価な製品開発が可能になると期待されている。

次に挙げられるのが、アクティブネットワーク技術 [283, 357] に代表されるように、より高次のネットワークサービスの構築が望まれるようになったことである。アクティブネットワーク技術は、ネットワークサービスをエンドノードによる処理だけでなく、ルータなどの中継ノードにおいても処理を行うことでより動的なサービスを実現するものである。中継ノードにおけるより柔軟な処理能力を実現するために NP を応用することが期待されている。

最後に、ゲートウェイやサーバホスト等のエンドノードの性能向上が求められていることが挙げられる。一般的に、プロセッサの処理能力は 18 カ月に 2 倍向上する（Moore の法則）ことが観測されている。一方で、近年のネットワーク帯域向上の速度は、この計算機の処理能力向上の速度よりもはるかに大きい [165]。そのため、エンドノードにおける TCP/IP 処理やアプリケーション層の処理を一部 NP に移すことで、処理能力の向上が期待できる。

このように大きな期待を集めている NP であるが、技術研究開発はこれまで産業界が主導して行ってきている [51]。そのため非公開な情報も多く、また研究成果の学術的な体系化も不十分であるのが現状である。本稿では、NP に関する学術論文を中心にサーベイする。特に、現在の NP に関する最先端研究が、どのように相互に関連しどこに向かおうとしているのかを少しでも明らかにできればと考えている。

以下本稿の構成を示す。まず、3.2 で NP アーキテクチャに関する研究を概観する。次に 3.3 では、NP における性能評価手法について述べる。ここでは、ベンチマークおよびシミュレーションに分類している。3.4 では、NP におけるプログラミングに関する研究について概観する。また、3.5 では、現在主流となっている高速レイヤ 3 パケット転送以外の機能の NP における実現について研究成果を概観する。最後に 3.6 で将来の NP 研究に関する展望を簡単にまとめる。

3.2 アーキテクチャ

NP で最も重要なのがアーキテクチャであり、これまでに多くのベンダーから様々なアーキテクチャを持つ NP がリリースされている。それらは、並列処理の実現方法、特定の目的のためのハードウエアの構成、メモリ構成、チップ上の通信機構、ネットワークインタフェース等の周辺装置との接続方法、などの特徴によって分類することができる [259]。しかし、それらの要素は互いに密に関連するため、本稿では NP 上に 1 つもしくは複数配置される処理要素（Processor Element：以下 PE と略す）のマイクロアーキテクチャと、NP における PE やメモリ、その他の周辺機器の配置および配線を含めたシステムアーキテクチャに分類し、特に各研究のアプローチの違いについて概観する。

3.2.1 マイクロアーキテクチャ

PE の要件

NP におけるマイクロアーキテクチャの要件を検討している初期の文献には [49, 213] がある。[213] では NP における処理の特徴を、(1)データ転送の負荷が高い、(2)割り込みが非常に高い頻度で発生する、(3)複雑な状態管理を必要とする処理が多い、(4)ルーティングなどでは表検索が重要、(5)ワードやバイトの境界をまたぐ処理もある、とまとめ、ハードウエアによるマルチスレッドのサポートや、コアの命令セットに関する検討を行っている。実装したチップを用いた評価では、1 Mbps のデータに対しておよそ 2 MIPS の処理能力で十分であり、汎用プロセッサより有利にパケット処理が可能であることを示している。

また、文献 [49] では、マイクロアーキテクチャとして Super-scalar Processor（SS）、Fine-grained Multithreaded Processor（FGMT）、Single Chip Multiprocessor（CMT）、Simultaneous Multithreaded Processor（SMT）の 4 種類について、典型的なワークロードに対して SMT が最も良い性能を発揮すると結論づけている。具体的には、SMT もしくは CMT アーキテクチャでクロック速度が 500 MHz のプロセッサを用いれば、単純なレイヤ 3 パケット転送であれば 10 Gbps を越える処理能力を、さらには MD5 や 3DES においても 1 Gbps を越える処理能力を持つと推測している。

文献 [215] では、エンドノードにおける TCP offloading のための NP を設計している。TCP ではパケット間の状態情報を管理する必要があることから、一般的なレイヤ 3 をターゲットにした NP とは異なる構造が必要となる。ここでは、パケット内で完結

するデータ処理を行うPEと、状態情報の検索やPEの制御、ホストプロセッサおよびメモリとの通信と行うプロセッサ（Micro Controller：以下μCと略す）に機能分離し、μCの要件を検討している。μCの処理はラインスピード（最小のパケットで帯域幅を埋める速度）では処理できないため、パケットサイズ等を考慮しなければならないとしている。

キャッシュ

NPにおけるキャッシュは非常に議論するのが難しい分野の1つである。パケットデータの参照回数が低いためデータキャッシュが有効ではないこと、命令キャッシュはマイクロおよびシステムアーキテクチャに大きく依存することなどが原因である。その中で、文献 [38, 91] では、レイヤ3転送におけるルーティングテーブルの高速検索手法に焦点を当て、キャッシュの側面から考察している。メモリキャッシュの構成においては、キャッシュのサイズ、ブロックサイズ、連想記憶の幅（associativity）が重要であり、効率的な検索を実現するためにルーティングテーブルのエントリ数を効率よく圧縮する技術を提案している。

3.2.2 システムアーキテクチャ

NPの基本的な目標の1つに処理の並列度の向上がある。そのため、並列アーキテクチャのNPへの応用を考察した研究が存在する。

まず、NP実装に関する初期の研究としてComet[348]がある。Cometでは、チェックサムやテーブル検索などの機能単位をFPGA（Field Programmable Gate Array）を用いて実装し、それらを汎用プロセッサを用いて制御する方式をとっている。これは、現在主流となっているNPのアーキテクチャと同じであり、先駆的な研究として注目に値する。さらには、Commetボードのクラスタ構成法も検討し、高性能かつ柔軟なルータの構築を実現している。

文献 [262] では、柔軟なQoSを実現するための高度に並列化されたNPアーキテクチャを提案している。提案したアーキテクチャでは、1つのLSI上に16個のPEをグループに分割した上で階層構造的に配置し、各PE内の2つのコアとレジスタファイルをグループ内通信バスで直結し同期させることで、PE間の高い相互通信機能を実現している。また、資源アクセス制御については、コンパイル時にあらかじめ全てのPEのコードを参照しておくことで静的に解決し、プロセッサコアはレジスタファイル上でのみ演算処理を行うことで全体で静的に同期した処理を実現している。

また、超並列計算機の様に非常に多数のRISCベースのプロセッサをチップ上に配置し、プロセッサ間を縦横に接続した静的なチャネルと宛先を記述できる動的なチャネルによってプロセッサ間通信を実現するアーキテクチャ（Raw Processor[310]と呼ばれる）も提案されている [37]。これは、コンパイル時に可能な最適化はもちろんのこと、プロセッサのスケジューリングをソフトウェアで制御することも可能であるという特徴を持つ。

また、マルチプロセッサベースのNPにおけるプロセッサ間の通信機構（Octagonと呼んでいる）に焦点を当てた研究には [147] がある。Octagonは、8個のプロセッサおよびそれらを結ぶ12本のポイントツーポイント結合より構成され、ネットワーククロスバスイッチよりも配線が少なく、最悪でも2ホップで任意のプロセッサ間も通信可能で総合的なスループットが高いという特徴を持つ。

文献 [180] では、非常に多数のスレッドをサポートするアーキテクチャへのアプローチとして、スレッド間のメモリ同期問題をハードウエアでサポートしたり、それぞれのプロセッサにおけるローカルなメモリアクセス手法に加えてグローバルなメモリ空間も用意するハイブリッドな構造にする手法について述べている。これにより、スレッドのコア集合における配置問題をプログラミングから切りはなして実現している。

また、並列計算機へのアプローチとしてデータ駆動の概念を用いたNPの研究もある [353]。ここでは、データ駆動型プロセッサを用いた終端ホスト向けTCP/IP off-loadingエンジンを開発している。アプローチとしては、各パイプラインのステージにおける処理時間を考察し、総合的なスループットの向上を図るように工夫したり、ボトルネックとなる個所を1チップ集積したりすることでスループットを向上させる手法を紹介している。

文献 [36] では、IBMが提唱しているBlueGene/Cyclops cellularアーキテクチャ [107] を用いたNPについて述べている。各PEの命令キャッシュの有効性を向上させるために、パイプライン処理を採用し、

[262] と同様プロトコル処理のプロセッサへの割り当てやメモリ管理は静的に行っている。実際のアプリケーションとしてファイバチャネルと Infiniband のプロトコル変換機能を実装し、シミュレーションによる評価を行っている。

3.3 性能評価

システムの性能評価では、ベンチマークやシミュレーションといった直接的な評価に加え、システムの抽象化を通じた設計空間の探索がアーキテクチャ開発においては有効である。本節ではそれぞれについて述べる。

3.3.1 ベンチマーク

ベンチマークを行うためには、NP に特化したワークロードを決定する必要がある。文献 [325] では、ヘッダ処理としてルーティングテーブル検索、パケットのフラグメンテーション処理、QoS のためのキュー管理、TCP トラフィックモニタリングを選出している。一方でペイロード処理には、暗号アルゴリズム、データ圧縮、Forward Error Correction（FEC）、画像圧縮を選出している。また、文献 [183] では計測対象の範囲を広げ、ベンチマークプログラムをマイクロレベル、IP レベル、アプリケーションレベルに分類している。

ベンチマークのもう 1 つの視点として、異なるアーキテクチャ間の性能比較がある。異なるアーキテクチャの比較を行うためには、システムレベルでの機能の抽象化が必要となってくる。文献 [31] では、ベンチマークプログラムをシステムの階層構造から、システムレベル、機能レベル、マイクロレベル、ハードウエアレベルに分類して定義することにより NP システムのモデル化を行い、様々な NP アーキテクチャに適用可能になるようにしている。また、ベンチマークの方法論を含めたフレームワーク化の議論を行っているのが文献 [298] である。そこでは、ベンチマークにおける 3 つの仕様、(1)機能仕様、(2)環境仕様、(3)測定仕様、を定めている。中心的役割を果たす環境仕様ではネットワークインタフェースやその制御インタフェース、トラフィックや負荷状況などを定義し、ベンチマークの汎用性を高めている。

3.3.2 シミュレータ

NP におけるシミュレーションの場合、ある特定のシステムの挙動を詳細に再現するためのものと、特定の実装に依存しない抽象的なフレームワークにより様々な構成のシステム性能を評価するためのものに分類することができる。前者のものは、多くの NP システムの開発環境に付随して配布されている。ここでは、後者に分類されるものについて述べる。

シミュレーションフレームワーク

NP システムを抽象化する場合、パケットのデータフローに従うのが開発者にとって容易である。文献 [274] では、システムを (1)トラフィック生成部、(2)入力解析部、(3)入力インタフェース、(4)プロセッサ部、(5)出力インタフェース、(6)出力解析部、にコンポーネント化し、クロック同期を用いた実時間シミュレーションによる正確な評価を実現している。ただし、シミュレータの実装が Cisco の NP である Toaster2 の構造に依拠しており、他の NP にも適用可能なように汎用性を向上することが求められる。

また、NP のシミュレーションに既存の汎用 OS 環境を利用したものもある。文献 [234] では、エンドノードにおける TCP off-loading エンジン開発をターゲットとしたシミュレーションフレームワーク（Countach と呼んでいる）について述べている。これは、Unix 上でユーザレベル TCP スタックを用い、細粒度のフックを埋め込むことでプログラムのメモリ参照の振舞いおよび各種装置におけるイベント処理の正確な同期をシミュレートすることができる。

設計空間の探索

NP 開発では、様々なシステム構成およびワークロードに対する性能評価を通じてより高い性能を達成することが重要となる。ここでは、特にそのような設計空間の探索を実現する技術に関する研究について述べる。

設計空間をシミュレーションする場合、アーキテクチャの変更を容易するためにモジュール化が重要となる。文献 [47] では、高度にモジュール化されたソフトウェアルータである Click[159] をベースにした、NP の処理能力モデルを構築している。Click における各機能のモジュール化およびイベント管理機能を活用し、ネットワークプロセッサシステムのモデル（プロセッサやメモリ、接続チャネル）を追加することで、システム全体の性能の柔軟なシミュレー

ションを実現している。

また、設計空間の探索は抽象度の高い作業であるため、シミュレーションよりも抽象度の高い解析的なシステムモデルも有用である。文献 [83] では、NP システムを 1 つの I/O インタフェースと複数のクラスタに分離し、各クラスタには複数の PE および 1 つのメモリインタフェースを搭載し、各 PE は命令セットとデータキャッシュを有するモデルを提案している。これらの構成要素においてパラメータ化を行い、ベンチマークワークロードに対する性能を解析的に算出している。また、文献 [84] では、このフレームワークを用いて処理能力と消費電力に関する分析も行っている。さらには、文献 [92] では解析的なフレームワーク [292] を用いて PE 間の接続トポロジというより高レベルなシステム構成に関する検討を行っている。

一方で、これらの解析的なフレームワークは、抽象度が高すぎるため実際のシステムの特性を十分に反映できない問題点があることには注意が必要である。例えば、解析的モデルでは PE によるパイプライン構成をとるより各 PE がパケット処理のすべてを行う方式の方が優れているとしている [92] が、実際には命令キャッシュの制約やチップ面積、通信コストの問題などからパイプライン構成を取る方が現実的である場合が多い。

遅延の解析

NP においては確定的な性能の確保が重要視されるため、コードブロックにおける最悪実行時間の分析が重要となる。文献 [48] では、マルチスレッド環境を想定した最悪実行時間の算出法について述べている。具体的には、実行コードの制御フローグラフ（Control Flow Graph: CFG）をもとにそれぞれのブロックの実行回数に関する制約を全て列挙し、線形計画法により最もコストの高いパスの実行時間を算出するというものである。マルチスレッドへの対応においては、CFG に中断ノードおよび中断エッジという概念を導入し、スレッド間制御切替えおよび切替えによるメモリアクセス遅延時間隠蔽もサポートしている。このような解析で得られる最悪実行時間は、理論的最悪値であることから悲観的（pessimistic）過ぎる傾向があるが、NP の場合は CFG が比較的単純であり、大きなプログラムでもシミュレーションによる実測時間の 4 倍程度であると報告されている。

3.4 プログラミング

一般的に NP におけるソフトウェア開発には低レベルなプログラミングが要求され、コストが高いという問題がある。そこで、NP のソフトウェア開発を容易にするプログラミング環境に関する研究がいくつかなされている。こうした研究の多くが、3.3 で述べたようなシステムの抽象化を基礎にしている。

3.4.1 プログラミングフレームワーク

文献 [260] では、先に紹介した高いモジュール性をもつソフトウェアルータ Click[159] のフレームワークを NP に応用したプログラミングモデル NP-Click を提案している。Click のフレームワークを用いることで、NP の様々な機能を抽象化し、それらの上に各種モジュールを提供することを可能にしている。性能評価では、Intel の IXP1200 を用いた実装において、パケットサイズが十分大きい場合には IXP1200 用の専用開発環境である Microengine C[124] による実装と比較してほぼ遜色のない性能を実現している。

文献 [182] では、別のアプローチによるモジュール化支援プログラミング環境 NEPAL（NEtwork Processor Application Language）を提案している。先の NP-Click では NP 特有機能の抽象化を言語レベルで実現したのに対して、NEPAL ではランタイムによる動的な制御機構によって実現している。しかし性能評価においては、単一プロセッサ上で実行した場合に対するチップマルチプロセッサ（CMP）の性能向上しか議論されておらず、確定的な性能が達成されるかどうか不明である。一般的にはランタイムを用いた手法はオーバヘッドが大きく [260]、3.2.2 で述べたような超並列アーキテクチャのようなスケーラビリティを重視したアーキテクチャなどで用いられると考えられる。

また、文献 [22] では、言語レベルの静的なモジュール化とランタイムによる動的なモジュール化の中間的なアプローチとして、コンポーネントと呼ぶコードの最小単位を動的にバインドしてモジュールを構成する手法について提案している。具体的には、プログラミングを容易にする階層的な抽象化、コンポーネント間のデータの受け渡しのためのレジスタの割り当て方式、コンポーネントの動的なバインド手法を設計し実装している。性能評価では、動的なモジュール構成のないモノリシックな実装に対して 2%程度と非常に小さい性能低下を実現している。

3.4.2 スケジューリング

NP では、高いスループットの達成のために複数のプロセッサを搭載しマルチスレッドをハードウエアでサポートしている場合が多い。例えば、Intel の IXP1200 ではメモリアクセス遅延隠蔽を目的とした non-preemptive マルチスレッド環境を提供している。また、確定的な遅延および帯域幅を実現するために、静的なスレッドスケジューリングを用いるものもある [36, 262]。一方で、より柔軟なマルチスレッド環境ではソフトウェアによる支援が必要となる。文献 [269] では、NP をマルチプロセッサでマルチスレッドをサポートするリアルタイムシステムと定義し、公平性を満たす Pfair（Proportionate Fairness）スケジューリングと呼ばれる方式の NP への適用を提案している。

3.4.3 コンパイラ

NP では特殊なマイクロアーキテクチャを採用することが多く、ワードやバイトの境界をまたぐようなデータ処理など汎用プロセッサにはない命令も効率よく処理する必要がある。ここでは、ネットワーキングの領域から逸脱するため紹介するだけにとどめるが、そのような NP 用コンパイラの最適化手法の研究として文献 [154, 308, 309] などがある。また、NP の適用先として有望視されているアクティブネットワークでは、各ルータのアーキテクチャに依存しない実行コードの注入が必要となる。文献 [155] では、NP における Just-in-Time（JIT）コンパイラの可能性について議論している。

3.5 アプリケーション

NP を高速なレイヤ 3 ルーティングだけでなく、より高いレイヤの処理に用いる研究がいくつかなされている。本節ではそれらについて紹介する。

3.5.1 パケットの分類

パケットの分類はネットワークセキュリティ機能の中心的役割を担っており、NP による高速処理が期待されている。文献 [294] では、DDoS（Distributed Denial of Service）攻撃を軽減するためのパケットフィルタリング機能（NetBouncer[295]）の NP への実装について、特に Intel の IXP1200 上の実装に関する知見がまとめられている。また、文献 [161] では、アクセス制御リスト（ACL）を用いたパケットの分類機能を、NP 上で実現するための要件について述べている。この研究では、従来では機密情報にあたるため公にはあまり議論されてこなかった ISP や企業で用いられているファイアウォールの ACL を用いて、より現実的な環境に適した方式を提案している。結論としては、(1) IP アドレス対（送信者と受信者）に関する検索とトランスポート層の情報に関する検索を分離する、(2) IP アドレス対の重複する部分は別途管理し、IP アドレス対に対して 1 つのフィルタを返すようにする、(3)トランスポート層のフィールド検査ルールはそれほど多くないので、特別なハードウエア装置を用いた検索をサポートする、という要件を導出している。

3.5.2 レイヤ 4 機能の off-loading

NP の適用先の 1 つとして、エンドノードにおける TCP/IP 処理の NP への移行が検討されている。文献 [352] では、ワークステーション用周辺機器としての TCP/IP 通信ボードを提案している。実装にあたっては、ネットワーク処理特有の機能をプロトコルヘッダ処理部と資源管理部に分離し、プロトコルヘッダ処理部の各種機能を LSI で実装している。これらの機能を汎用 RISC プロセッサ（クロック 33 MHz の SPARClite）を用いてプログラム可能にしている。コネクション多重化に対応していないなどの不備はあるが、先駆的な研究の 1 つに挙げられる。

他にも、これまでに述べた文献 [215, 234, 353] も TCP/IP 処理の off-loading を目的としている。

3.5.3 ストレージサービス

文献 [336] では、ネットワークストレージサービスを組み込み用プロセッサを搭載したハードウエアで実現するシステムを開発している。システム設計的にはボード内の通信バスとして PCI を用いていたり、ネットワーク処理も Linux カーネルを用いているため、正確には NP の応用とは言えないが、ネットワークストレージは NP の格好のアプリケーションとして期待されている [36]。

3.5.4 レイヤ 7 補助プロセッサ

文献 [181] では、レイヤ 7 の処理を補助する NP 用の補助プロセッサを提案している。提案しているプロセッサはバレルシフタに似た構造を持ち、HTTP における URL ベースのスイッチなどで必要となる

パターンマッチング機能や、IP ルーティングにおける radix-tree 検索機能、セキュリティ機能等で利用される MD5 アルゴリズムなどへの応用例を示し、シミュレーションによる性能評価を行っている。また、文献 [184] では、このアクセラレータの FPGA（Field Programmable Gate Arrays）チップにおける実装を検討している。

3.6 おわりに

本稿では、NP におけるアーキテクチャ、性能評価、プログラミング、アプリケーションの各研究分野について成果を概観した。一方で、本稿では学術論文以外の文献（各種製品情報など）についてはほとんど触れることができなかった点と、学術論文についても網羅することを目的とはしていないため多くの漏れがある点については注意が必要である。特に 3.2 で述べたアーキテクチャについては、多くのベンダーから多種多様な NP がリリースされているにも関わらず、ほとんど触れることができなかった。それらについては、各社から製品とともにリリースされているホワイトペーパーや、ベンダー向けフォーラムの Network Processing Forum（NPF）[1]や、コンファレンスの Network Processor Conference[2]を参照すると良い。

これから NP は、多種多様なアーキテクチャが考案される初期の研究開発段階から、実用化を含めた次の段階へ移行するだろう。特に、高いスループットを要する基幹ルータへ応用が可能かどうかが NP の当初の目的から 1 つの焦点になると考えられる。例えば、ソフトウェアルータとしてデファクトスタンダードの地位を確立している Zebra の商用版である ZebOS Advanced Routing Suite では NP を用いることができ [131]、実用化が進められている。また、レイヤ 3 パケット転送以外のキラーアプリケーションの開発も求められている。このように、今後はより実用的な技術の研究開発が重要になるだろう。

1 http://www.npforum.org/
2 http://www.networkprocessors.com/

第 4 章 NP を用いたトレースバックシステムへの応用

本章では、2003 年 5 月に行われた電子情報通信学会で田中らによって発表された論文を掲載する。

概要

トレースバック手法の 1 つであるハッシュベースのトレースバックでは、経路中でパケット毎にハッシュ値を演算・記録し、問合せパケットに対応したハッシュ値が記録されているかどうかを検索することによりトレースを行う。本稿では、Intel のネットワークプロセッサ（NP）IXP1200 シリーズを利用することにより GbE クラスのネットワークへの対応を可能にした、ハッシュベーストレースバックシステムの実装について述べるとともに、その評価結果と課題をまとめる。

Abstract

Hash-based traceback, one of packet traceback methods, traces IP packets by calculating the hash for each traversing packet, recording the hash in its database and searching the database for the queried packet's hash. Network processor is a good choice to realize these functions in high traffic networks. We implemented a hash-based traceback system on Intel's network processor IXP1240. This paper describes the implementation of the system which can be applied to GbE class networks, its evaluation results and its problems.

4.1 はじめに

不正なパケットを大量に送りつけてサービスを妨害する DoS（Denial of Service）攻撃や、複数地点から DoS 攻撃を行う DDoS（Distributed DoS）攻撃が、社会的な問題となっている。DoS 攻撃への対処の方法として、DoS 攻撃パケットの送信元になるべく近いところで、フィルタなどの手段により DoS

攻撃パケットを制限することがあげられる。しかし、攻撃パケットの送信元 IP address が偽装されている場合もあり、真の送信元の調査は手間のかかる作業である。この問題を解決するための手法は、総称して「IP Traceback」手法と呼ばれており、ダイジェスト方式 [252, 264] やマーキング方式 [253] など様々な方式が提案されている [343]。一方、対象となるネットワークは高速化が進んでおり、流れるパケット全てをソフトウェアのみで処理することが難しくなっている。しかし高速化のために ASIC などを用いたハードウェア主体の実装は、その開発費用・期間・人材面で制約が大きく、まだ仕様が確立されていないトレースバックのようなアプリケーションに適用することは難しい。そのような状況の中、ハードウェア実装に近い高速性と、ソフトウェア処理による柔軟性を併せ持ったネットワークプロセッサ（NP）が注目されている。我々は、高速ネットワークに対応可能なダイジェスト方式のトレースバックシステムを、NP を用いて実装した。本稿では、そのトレースバックシステムの実装および評価結果と課題について報告する。

4.2 Traceback System

4.2.1 システムの構成

我々が作成したトレースバックシステムは、図 4.1 に示すように Footmarker、Manager、Gate の 3 つの構成要素からなる。Footmarker は各観測点に設置され、そこを通過するパケットのダイジェスト（ハッシュ）を計算し記録する。Manager は Footmarker の管理を行うと共に、Gate からの検索要求を受け付けて、Footmarker に問合せを行い回答する。Gate は本システムと他システムとのゲートウェイの役割を担う。Intrusion Detection System (IDS) やパケットダンプツールなどから得た攻撃パケットを入力とし、Manager に問合せを行う。

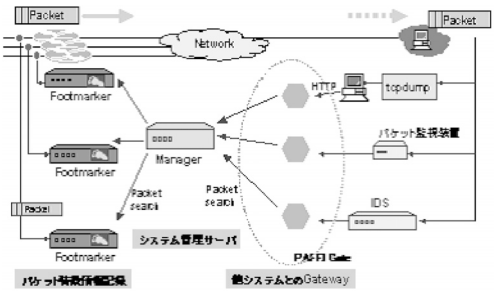

図 4.1.　Traceback System の構成

4.3 Footmarker の設計

4.3.1 Footmarker の概要

Footmarker では、接続するネットワークに流れる全てのパケットのハッシュ値を算出するため、その流量に対応した高速なパケット処理が必要とされる。我々は、この処理のため、ネットワークプロセッサを利用したネットワークボードを作成した。本研究で採用したネットワークプロセッサは、Intel 社の IXP1240 であり、その内部はパケット処理用 RISC プロセッサである 6 つのマイクロエンジンと、これらを管理制御するプロセッサである StrongARM コアから構成されている [125, 126, 143]。Footmarker のブロック図（ハードウェア）を図 4.2 に示す。Gigabit Ethernet (GbE) のネットワークに対応するため 1000Base-SX の観測用インタフェースを 2 ポート実装した（表 4.1）。パケットの処理は IXP1240（マイクロエンジン、StrongARM コア）上のソフトウェア（ファームウェア）で実現している。

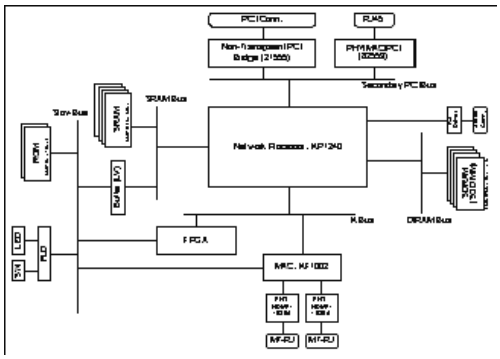

図 4.2.　Footmarker のハードウェアブロック図

表 4.1.　GbE 対応 Footmarker のハードウェア仕様

Network Processor	Intel IXP1240 (232 MHz)
Onboard Memory	SDRAM: 256 Mbytes SRAM: 8 Mbytes Flash ROM: 8 Mbytes
Network I/F	1000Base-SX × 2 10/100Base-TX × 1
Console	Serial I/F
Card Form	Full-size PCI
Power Disspation	20 W (estimated)
Optional Feature	200,000 gate FPGA on IO bus

4.3.2 Footmarker の構成

Footmarker のソフトウェアは、以下の 3 つのモジュールから構成される（図 4.3 参照）。

(1) Packet Search Engine

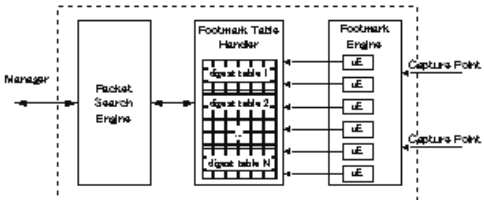

図 4.3. Footmarker のモジュール構成

PAFFI Manager からのパケット検索問合わせを受け付け、それに返答する。検索パケットのハッシュ値を算出し、パケット通過記録への問合わせを行う。

(2) Footmark Table Handler
パケット通過記録（Footmark）とそのアクセス手段を提供する。通過記録は複数の Digest table により構成される。

(3) Footmark Engine
ネットワーク上を流れるパケットのハッシュ値を順次算出し、パケット通過記録として保存する。

4.3.3 Footmarker Engine とネットワークプロセッサ

Footmarker の構成の中で、処理速度が要求されるのは、Footmark Engine の部分である。この処理を IXP1240 上のマイクロエンジン部で行っている。具体的な内容は、パケットの受信、ハッシュ値の算出、テーブルへの記録である。

ハッシュ値の算出には、IXP に内蔵されている 64 bit ハッシュ演算ユニットを利用し、テーブルへのマークは、SRAM 制御ユニット中のビットマーク機能を利用することで、ネットワークプロセッサの持つハードウェアアシストを活用している。

4.4 評価

測定対象と負荷発生装置を図 4.4 のように接続する。負荷発生装置（Flame Thrower）で特定の長さ（順次変更）のパケットを繰り返し生成し、測定対象に送信してその処理能力を測定する。比較対照として GbE の NIC を取り付けた汎用 PC（表 4.2）を用意し、併せて測定を行った。

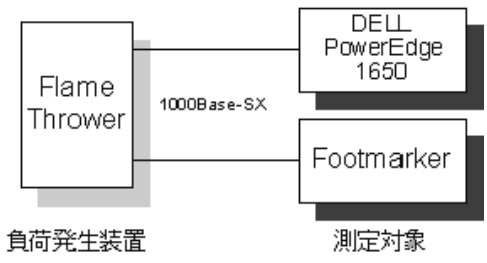

図 4.4. 測定環境

表 4.2. 比較対照 PC の仕様概略

Model	DELL Power Edge 1650
Processor	Intel Pentium III (1.13 GHz)
Memory	SDRAM: 1.0 Gbytes Primary Cache: 512 Kbytes
Network I/F	Intel Pro-1000F
OS	FreeBSD 4.7

4.4.1 GbE 対応 Footmarker のキャプチャ性能測定結果

Footmarker に通常の動作を行わせ、3 種類のパケットを流して測定した結果を図 4.5 に示す。

最小パケット（64 bytes）から最大パケット（1518 bytes）まで全域に渡り理論値に近い性能を出していることが分る。

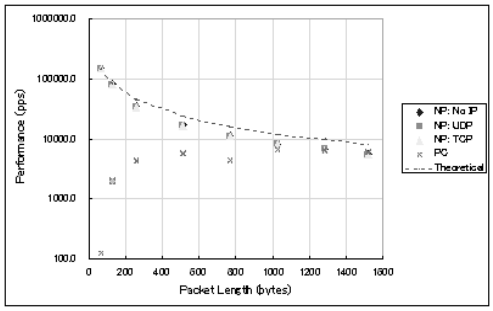

図 4.5. 測定結果

4.4.2 PC ＋汎用 OS のキャプチャ性能測定結果

汎用 PC 上で動く Footmarker プログラムは現時点では存在しないので、PC での測定に当っては一般的に用いられているパケットキャプチャソフトウェア “tcpdump” を使用した。

Footmarker の動作に近づけるため、TCP だけを選択するフィルタオプションを指定して測定を行った。結果を図 4.5 に重ねて示す。

4.5 考察

ネットワーク機器の性能を考える上では、単純な流量（bps, bits per second）ではなく、毎秒の処理パケット数（pps, packets per second）が重要な指標となる。パケット単位で特徴情報を記録する Footmarker にお

いてもそれは同様である。測定結果のグラフ（図 4.5）からは、以下のようなことが分る。

- パケット長が大きい場合には、NP ベースの実装も汎用 PC も理論限界値に近いパケット処理能力を持っている。
- しかし、パケット長が小さい場合には、汎用 PC の処理能力は急激に悪化する。パケット毎に上がる割込みの負荷が重いためと考えられる。
- それに対して、NP ベースの実装ではパケット長が小さい場合にも理論限界値に近い処理能力を発揮している。

なお、Footmarker ではキャプチャだけではなくパケットの特徴情報を得るためにハッシュ値の生成などの処理まで行っているのに対し、今回測定した PC では tcpdump のフィルタリングまでしか行っていない。同じ処理を行わせた場合には、処理性能差は今回の結果よりもさらに開くことが予想される。

また、現時点での Footmarker のプログラムは改善の余地を多く残している。NP のマイクロエンジン部はその並列性を活かすプログラムの最適化によって性能が大きく向上するので、今後のチューニングによってさらに高い性能を示すことも期待できる。

実ネットワークでは、100%のビットレートのトラフィックが流れ続けることはまずないので、今回の NP ベース Footmarker 程度の処理性能があれば GbE ネットワーク環境においても実用上問題のない IP Traceback システムが構築できる見込みが得られた。

4.6 まとめ

ハッシュベースのトレースバックシステムをネットワークプロセッサ上で実装し、評価を行った結果、PC に比べよいパフォーマンスを得られた。

今後、プログラムの最適化をさらに進め、このようなアプリケーションへのネットワークプロセッサの適用についてより詳細な評価を行う予定である。

第 5 章　ネットワークプロセッサを用いた MPLS ラベルスイッチングルータの実装

本章では、2003 年 5 月に行われた電子情報通信学会で渡辺らによって発表された論文を掲載する。

概要

ソフトウェアルータは柔軟性が高く機能の拡張が容易である一方で、ルータとしての性能は専用ルータと比較して非常に低く、適用可能な範囲が限定される場合が多い。専用ルータとの性能格差を低減する技術の 1 つとしてネットワーク処理に特化したプロセッサ（NP）の開発および利用が注目されている。既存のネットワークスタックの一部機能を外部 NP で処理させることで、ソフトウェアルータの柔軟性を損なわずにシステム全体の高速化が可能である。本稿では、汎用 PC 上で動作するソフトウェア MPLS ルータとして設計実装された AYAME に対して、Intel 社のネットワークプロセッサ IXP1200 を外部に接続したハイブリッドシステムを用いた、ソフトウェアルータの高速化の設計と実装を述べる。

Abstract

The type of software router has very high flexibility and functional extensibility, whereas it has much lower performance and less wide application than the exclusive hardware router in many case.

Development and utilization of the processor optimized for the network processing (NP) has attracted much attention as one of techniques to decrease a performance gap between software and hardware routers. When outside NP module handle a part of existing network stack functions, whole system can be made faster without little damage to the flexibility of software implementations. This paper discusses about the design and implementation of the hybrid router with the IXP1200 intel network processor on generic PC to improve the forwarding performance of the AYAME software MPLS router.

5.1 はじめに

インターネット上のトラフィックは、利用者数の増加とアプリケーションの要求帯域量の増加に比例して広帯域化している。この要求に対処するために最近のルータの多くは専用 ASIC（Application Specific Integrated Circuit）を用いたハードウェアパケットスイッチング機構による高速／広帯域パケット処理を実現している。一方で専用 ASIC の設計・実装に

はソフトウェアによる実装と比較して開発および製造のコストが非常に大きいため、インターネットにおける新規技術標準化への対応が困難である。

このような背景から、パケットスイッチング等のネットワーク処理に最適化された特定用途向けプロセッサであるネットワークプロセッサ（NP）が注目されている。NP を用いることで、高速パケット転送を提供しつつ、新機能への対応に関してもソフトウェア実装並みの柔軟性が可能となる。

著者らは、次世代インターネット技術として注目されている MPLS の研究開発プラットフォームである AYAME[302] の開発を進めている。AYAME は NetBSD オペレーティングシステム上で動作するソフトウェア MPLS（Multiprotocol Label Switching）[247] 研究開発プラットフォームである。AYAME は基本的な MPLS LSR（Label Switching Router）の機能を実装しつつ、MPLS を用いた新規技術に関する研究に用いるための拡張性を重視しており、既存シグナリングプロトコルの実験的拡張などが常に進められている。本稿では、NP を用いた AYAME における高速パケットスイッチングの実現に関して報告する。

5.2 ネットワークプロセッサ

ネットワークプロセッサ（NP: Network Processor）は、パケットスイッチング等のネットワーク処理に最適化された特定用途向けプロセッサである。NP は様々なアーキテクチャが提案されているが、一般的にはネットワーク関連の処理を高速に行うためのハードウェアおよび命令セットで構成される。専用のハードウェアでしか実現できなかったネットワーク処理速度をプログラム可能なプロセッサで実現できるため高性能かつ拡張性が高い機器の開発が可能である。

本稿ではソフトウェアルータ実装を拡張するうえで Intel 製の IXP1200 プロセッサ[3]を搭載した横河電機(株)製の “ネットワークプロセッサカード NAPPI1200[4]” を採用した。NAPPI は比較的安価かつ入手性に優れる NP 開発環境である。

5.2.1 IXP1200 アーキテクチャ概略

IXP1200 は、6 個のパケット処理特化プロセッサ MicroEngine と StrongARM アーキテクチャの汎用プロセッサで構成される。

MicroEngine: 高速なパケット処理に特化した命令セットをもつパケット処理専用プロセッサである。ハードウエアレベルでマルチスレッドをサポートするため低レイテンシでコンテキストスイッチが可能であり、専用のパケット書き換え機構などによる高い転送能力を発揮するが、汎用性は低く複雑な処理は実現できない。

StrongARM: 主に MicroEngine をサポートする目的で利用される汎用プロセッサ。一般には Linux や VxWorks といった OS が稼動しており、MicroEngine の動作を制御する補助システムを提供するために用いられる。また、MicroEngine で処理できなかった例外パケットの処理にも用いられる。

5.2.2 NAPPI1200 概略

ネットワークプロセッサカード NAPPI1200 は、IXP1200 と付随する周辺チップやネットワークインタフェースおよびメモリ等が搭載された NP 評価用ボードである（図 5.1）。ボード上には、IXP1200 に接続された FastEthernet（FE）インタフェースが 3 系統と補助用の FE が 1 系統実装されている。前者は、IXP1200 の MicroEngine と直結されており、本系統からの入出力は MicroEngine による高速処理が可能であるが、後者は StrongARM に接続されており、StrongARM 上の OS から操作される。以下、本稿では StrongARM に直接接続されているインタフェースを、他のインタフェースと区別し制御用 FE

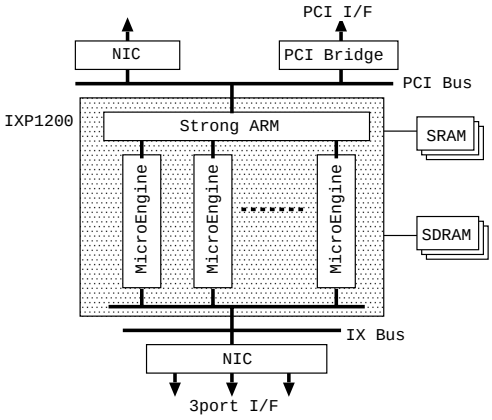

図 5.1. NAPPI1200 の構成

3 http://www.intel.co.jp/design/network/products/npfamily/ixp1200.htm
4 http://www.netstar.co.jp/products/NAPPI/

ポートと呼ぶ。また、NAPPI1200 上には 8 Mbyte の SRAM と 256 MByte の SDRAM が搭載されている。これらのメモリは、全ての MicroEngine および StrongARM から直接参照が可能であり、

- プログラムの格納
- パケット転送時のパケットバッファ
- パケット転送時に用いる FIB の格納

などに用いられる。実際に MicroEngine 上で動作するプログラムを記述する際は、動作速度・容量など特性の異なるこれらのメモリを、その用途に応じて使い分けることになる。

5.3 設計

MPLS 実装 AYAME は、NetBSD のカーネルを拡張して実装されているラベル付きパケット転送機構（LSE: Label Switching Engine）[337] と、ユーザ空間で動作するシグナリングデーモン等の制御機構で構成される。制御機構については、常に開発が進められている AYAME の新たな機能との整合性を維持するために特に手を加える事なく NetBSD 上で動作させながら、AYAME のパケット転送機構を NP を用いて高速化させる拡張を行った。既存 AYAME 実装と NP を用いた拡張の概略を図 5.2 に示す。

転送機構を NP 上で動作させることにより転送機構と制御機構が別のプロセッサ上で動作することとなる。このため、既存のソフトウェア実装においては、単一のテーブルとして保持していた FIB（Forwarding Information Base）情報が、

- 制御機構の動作する PC 上の FIB
- 転送機構の動作する NP 上の FIB

と複数になり、その同期の手法等を考慮する必要が生じる。以下に、NP および PC における本実装の設計について述べる。

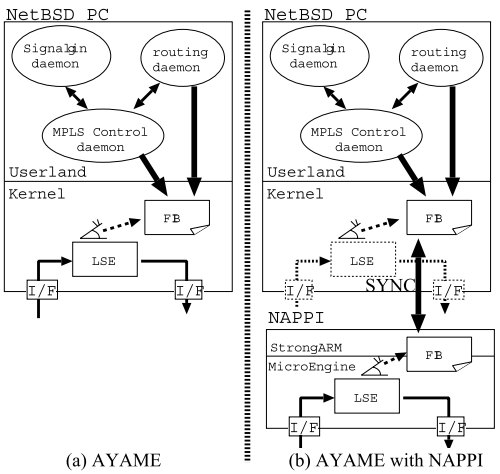

図 5.2.　既存 AYAME 実装と NP を用いた拡張

5.3.1 NP 上のパケット転送機構

NP 上には AYAME LSE を実装した。MPLS LSR における LSE の処理は以下の手順で行われる。

1. ネットワークインタフェースからパケットを受信し、
2. ヘッダ内のラベル値を取り出し、
3. NHLFE テーブルを参照し、
4. PUSH/POP/SWAP など目的の処理をし、
5. 適切な I/F から適切なノードに対して出力する。

今回の実装においては、LSR 上で最も一般的な動作であるラベルスワップの機能のみを NP 上のパケット転送機構で実現することとした。この場合、パケット転送には、以下のエントリが格納されたテーブル（FIB）が必要である。

- 入力パケットのラベル値（検索鍵）
- 出力先インタフェース
- 次ホップノードの MAC アドレス
- 出力パケットに付けるべきラベル値

なお、今回対象としている NAPPI1200 ボードは、イーサネットインタフェースのみを提供しているため、フレーム中のラベル格納手法として SHIM ヘッダを用いたもののみを扱うこととし、ラベル空間としては一般に多く用いられている Global Label Space のみをサポートすることとした。

5.3.2 PC 上の制御機構

先述の通り、制御機構は NetBSD が動作する PC 上で実現する。シグナリングや経路制御など、既存の AYAME 実装に対して手を加えることなく NP 上のインタフェースを取り扱える設計とすることが重要である。

これは、NetBSD 上で動作する各制御システムに対して、NP 上のインタフェースをあたかも “PC に直接搭載されているインタフェース” のように見せることで実現できる（図 5.3）。このような構成とすることで、各制御システムは NP 上のインタフェースを一般のインタフェースと同等に制御できることとなり、それぞれの制御システムに対して全く拡張を施すことなく NP 上のインタフェースを扱うことができる。

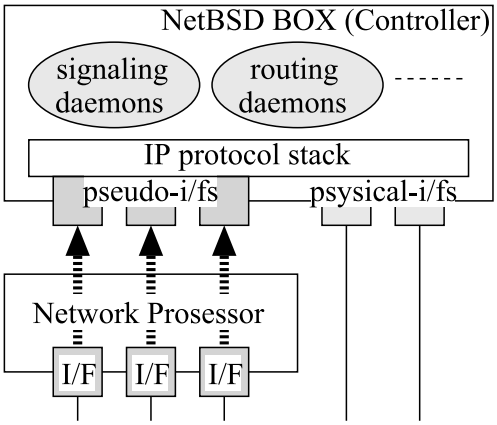

図 5.3. 実インタフェースと NP 上のインタフェースの同一視

5.3.3 PC–NP 間における FIB の同期

NP 上でパケット転送を実現する際に必要となる FIB 情報は、5.3.1 節で述べた通りである。PC 上では、5.3.2 節で述べた通り、経路制御・シグナリングなどの制御機構が NP の存在にかかわらず動作しており、NetBSD のカーネル空間内に FIB 情報として格納されている。この PC 上の FIB 情報のうち、NP 上で必要とされるものを NP との間で同期する必要がある。

制御機構の置かれた NetBSD PC 上で保持している FIB 情報は、先に挙げた NP における FIB 情報とほぼ同等なものである。ただし、PC 側では、MPLS ラベル付きパケットに対して SWAP のみならず、PUSH/POP などの処理も行うため、若干情報量が多い。また、PC 側では、次ホップの情報は IP アドレスとして保持しており、インタフェースからのパケット送出時に ARP テーブルを参照することで MAC アドレスを求めている。今回の実装においては、転送処理に最低限必要な情報のみを NP 上に置くこととしたため、ARP 解決後の MAC アドレスを次ホップノードの情報として NP に伝達することとした。

5.4 実装

前節で述べた設計に基づき、実際に実装を行った。

5.4.1 NP 上での LSE の実装

LSE は、ネットワークインタフェースから入力されたラベル付きのパケットに対し、NHLFE（Next Hop Label Forwarding Entry）の情報にもとづくラベルの SWAP、PUSH、POP の操作を行い次ホップノードに対しパケットを送出する機構である。

まず、NP 上での NHLFE テーブルの構成に関して述べる。5.3.1 でも述べた通り、転送処理に必要となる情報は、入力パケットのラベル値を検索鍵とし、ラベルに対する処理、出力ラベル値、出力インタフェースと次ホップノードの MAC アドレスである。さらに、出力時には各出力インタフェースに対応する MAC アドレスを出力イーサフレームの始点アドレスとして格納する必要がある。この情報は、出力インタフェース番号を鍵に別テーブルから検索し取得することも可能であるが、出力処理の簡素化・高速化のため、今回は NHLFE テーブルにも出力時の始点 MAC アドレスとして格納することにした。今回用いた NHLFE テーブルの構造を図 5.4 に示す。なお、運用形態によっては NHLFE エントリ数は比較的大きな数となる場合も考えられるため、今回の実装では、この NHLFE テーブルは SDRAM 上に格納することとした。

次に実際の入力パケットに対する MicroEngine の処理について述べる。その概要を図 5.5 に、詳細を以下に示す。

まず、MicroEngine は、ネットワークからのパケット到着を確認すると、パケットを SDRAM に格納する。そして、転送処理において参照・書き換えが行われるイーサネットヘッダと SHIM ヘッダ部分（計 18 byte）を、SDRAM 転送レジスタ (32 bit）に格納する。ここで、IXP1200 の MicroEngine における SDRAM の転送単位は 8 byte なので、連続した 6 つ

入力ラベル (20 bit)	始点 MAC (48 bit)	終点 MAC (48 bit)	出力ラベル (20 bit)	処理 (2 bit)	出力 I/F (4 bit)
:	:	:	:	:	:
:	:	:	:	:	:

図 5.4. NP 上の NHLFE テーブルの構成

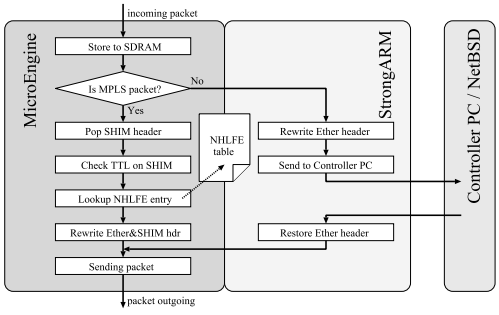

図 5.5. NP 上の LSE におけるパケット転送動作

の SDRAM 転送レジスタを用いて計 24 byte を読み込む。

次に、該当パケットに対応する NHLFE を検索するために、SDRAM 転送レジスタに格納されたパケットデータから SHIM ヘッダ部分を汎用レジスタに取得し、その汎用レジスタ中のラベル値を検索鍵とし前述した NHLFE テーブルを検索する。ここで獲得した NHLFE に応じて、パケットに対する処理を行うわけだが、今回はラベルスワップのみを想定しているので、NHLFE の情報にもとづいて、

- SHIM ヘッダ内の TTL 値の減算
- SHIM ヘッダ内のラベル値の書き換え
- イーサネットヘッダの始点・終点アドレスの書き換え

を行う。なお、この書き換え処理は、先ほど SDRAM 転送レジスタ上に格納したフレームヘッダ部分に対して行う。

この書き換え処理の後、先に NHLFE で指定された出力インタフェース番号に従い、パケットを送出する。なお、今回の実装にあたり、パケット送出部分を始めインタフェイスの直接操作に関わる部分に関しては、IXP1200 開発環境とともに配布されているサンプルコードを用いた。

5.4.2 NP 上のインタフェースの PC 側での扱い

5.3.2 節で述べたように、NP によるパケット転送機構を用いた環境で AYAME のシグナリング機構や経路制御機構を手を加えることなく利用するためには、NP 上のネットワークインタフェースを制御 PC 上に直接搭載されたものと同等に見せる必要がある。そこで、制御 PC 上の NetBSD において、NP 上のインタフェースを扱うための仮想インタフェースを実装した。NP 側では、先に述べたようにラベルスワップを行うパケット転送のみを扱っており、自ノードを終点とするパケットは制御 PC 側に回送する。制御 PC 側では、この NP から回送されて来たパケットを仮想インタフェースから入力されたパケットと扱い、また、仮想インタフェースから出力したパケットを NP 側へ回送し NP のインタフェースから出力する。

ここで、NP と制御 PC 間でのパケットの回送手法について述べる。NAPPI1200 ボードは制御 PC の PCI バスに挿入されており 5.2 節で示した PCI ブリッジを介して接続されている。しかし、現時点でこの PCI ブリッジを制御 PC 側で扱うためのドライバ等が用意されていないため、今回は NP と制御 PC の間の通信に PCI バスを用いることを断念した。それに代わり、NAPPI 上の制御用 FE と制御 PC 上の FE を接続し、このネットワークを介して NP と制御 PC 間の通信を行うこととした。

制御 PC–NP 間フレーム回送

まず、NP で受信した自ノード宛のパケットを制御 PC 側に回送する手法について述べる（図 5.6(a)）。この回送には、NP と制御 PC 間のネットワークを用いるわけだが、制御 PC に対し受信パケットを正しく届けるためには、NP において受信したイーサネットフレームをそのまま PC 側に届ければよいわけではなく、宛先 MAC アドレスを制御 PC の MAC アドレスに書き換える必要がある。さらに、NP 上には転送に用いる 3 つのポートがあり、回送したパケットがどのポートから入力されたパケットなのか制御 PC 側で認識する必要がある。また、制御 PC 側では、受信したイーサフレームが実際には NP 上のポートから入力されたフレームであることを認識し、仮想インタフェースからの入力と扱わなければならない。そこで、NP において受信した自ノード宛パケットは、その *ethertype* フィールドを書き換えて NP 上のインタフェース番号を識別できる情報を埋め込んだ上で、制御 PC に送出することとした。ここで送出するフレームの *ethertype* は以下とした。

- *15–12bit*: NP で受信したフレームを指す（`0xB`）
- *11–4bit*: payload の種類を示す
- *3–0bit*: NP 上のインタフェース番号を示す

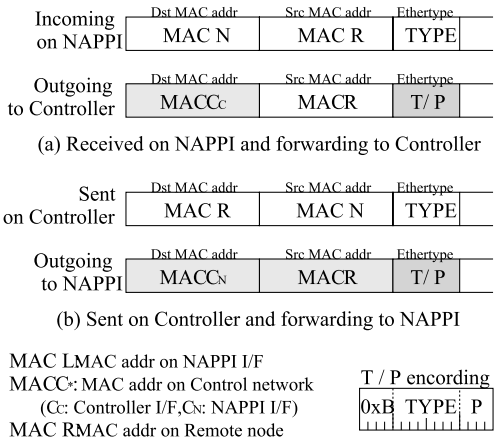

図 5.6.　フレーム回送時のイーサネットヘッダ書き換え

また、*11–4bit* のペイロードタイプは、我々が扱うもののみを IP フレームを `0x01`、ARP フレームを `0x02` のように独自に規定した。該当フレームを受信した制御 PC 側では、*ethertype* の *15–12bit* が `0xB` であることで NP から回送されたフレームであることを検出し、上述の逆変換によりフレームを復元した後、仮想インタフェースからの入力として扱う。

次に、自ノード発のパケットを NP 上のインタフェースからの送出とするためのパケット回送手法について述べる（図 5.6(b)）。制御 PC から送出されるパケットは、仮想インタフェースを介して送出され、制御 PC–NP 間のネットワークを介して回送され、NP 上のインタフェースから送出される。このようなパケットも、制御 PC–NP 間のネットワークで正しく NP に届け、また、NP が複数のポートのいずれから送出すべきかを判別できるようにするため、フレームの操作を行う必要がある。まず、該当フレームを正しく NP に届けるために、宛先 MAC アドレスを NP 上の制御 FE ポートの MAC アドレスに書き換える。元々の宛先 MAC アドレスは始点 MAC アドレスとして格納する。さらに、先に述べた NP から制御 PC へパケットを回送する場合と同一の手法で *ethertype* の書き換えを行い、フレームを NP へ回送する。このようなフレームを制御 FE ポートから受信した NP は、逆変換を行い元フレームを復元したのち、*ethertype* に指定された送出インタフェースからフレームを送出する。一見、この手法では元フレームの始点 MAC アドレスの情報が失われているようにも見えるが、NP 側で、送出ポートの情報から始点 MAC アドレス情報が復元できる。

ここで挙げた手法とは別の手法として、NP–制御 PC 間のフレーム回送の際には Ethernet over Ethernet のトンネリング手法を用いるという手段もある。この場合、構造が容易かつ回送できる *ethertype* に制限が加わらないという利点もあるが、回送できるフレーム長に対する制限やパケットサイズ増大によるオーバヘッドの影響を考慮し、今回は上述した手法を採用した。特に、現状のインターネットでの運用においては、MTU 値が 1500 未満となることは致命的である。

制御 PC 上での仮想インタフェースの実装

制御 PC 上で、NAPPI1200 上のインタフェースを直接接続インタフェースと同等にユーザ空間プログラムに見せるための仮想インタフェース *veth* を実装した。この実装は、制御 PC 上で動作する NetBSD カーネルに対する拡張である。

仮想インタフェース *veth* ドライバは、インタフェースタイプなどは一般の直接接続されているイーサネットインタフェースと同一のものとした。通常のインタフェースドライバでは、ハードウェアからの割り込み要求によりパケット入力処理が駆動され、また、パケット出力ではハードウェアに対してパケット送出処理を行う。これに対し、我々の実装した仮想インタフェース *veth* は、特定の *ethertype* を持つパケットが NP と接続している実インタフェースに入力されると *veth* の入力関数に渡され *veth* インタフェースからの入力として扱い、また、*veth* の出力関数は出力フレームの書き換え処理を行った上で NP と接続している実インタフェースから出力する。以下にその実装の詳細を述べる。

まず、NP から回送されてきたパケットの処理を行う部分について述べる。一般に、NetBSD においては、イーサネットインタフェースからの入力フレームはイーサネット共通入力処理関数である *ether_input* 関数によって、その *ethertype* からネットワーク層プロトコルを決定しネットワーク層入力関数に渡される。我々は、この *ether_input* 関数を拡張し、先に述べた、*ethertype* の *15-12bit* が `0xB` であるフレームを検出し、このフレームを *veth* ドライバの入力関数 *veth_input* に渡すこととした。*veth_input* では、前節で述べた手法により NP で受信した状態のフレームに復元した上で、入力インタフェース情報を *vethN*（*N* は NP 上のインタフェース番号）に書き換えた上で、一般のイーサネットインタフェースの処理と同様にこのフレームを *ether_input* 関数に渡す。これにより、NP で受信したフレームはあたかも *vethN* インタフェースで受信したように見える。

次に、制御 PC 側から送出する際の処理について述べる。制御 PC から *veth* ドライバを介した送出パケットは、一般のイーサネットインタフェースにおける処理と同様に *ether_output* 関数によりイーサネットフレームが作られた上で、*veth_output* 関数へと渡される。*veth_output* 関数では、前節で述べた手法によりイーサネットヘッダの書き換えを行う。その上で、NP が接続されている実インタフェースの出力関数 *xxx_output* を、*ether_output* 関数からの呼び出しと同等の方法で呼び出し、実インタフェース

のパケット出力処理を駆動する。これにより、*veth* インタフェースから出力されたパケットは実インタフェースを介し NP へと回送されることとなる。

5.4.3 FIB 情報の同期機構

AYAME ではラベルスイッチング処理内容を示す FIB 情報は、制御機構によって管理されており NetBSD カーネル中で保持されている。カーネル内の LSE では FIB 情報を直接参照できるが、NP を用いて外部 LSE を動作させる場合には NP 側で参照可能でなければならない。本実装では 5.4.1 節でも述べたように、NP 上の LSE はパケット転送処理の度に制御 PC 側の FIB 情報を参照せず、必要な最低限の FIB 情報を NP 側で保持する。このような、制御 PC 側の FIB 情報をマスタ、NP 側の FIB 情報をキャッシュとした動作を実現するためには制御 PC 上の FIB 情報を NP 上の FIB 情報へ反映して同期を取る必要がある。

NP 上に配置する FIB 情報は、5.4.1 節で述べた通り図 5.4 に示すものである。これらの情報のマスタは、制御 PC 上の NHLFE テーブルおよび ARP テーブルである。今回の実装では NetBSD のユーザ空間に kvm（kernel virtual memory interface）を用いた FIB 同期のデーモンを作成し、カーネル内の NHLFE テーブルおよび ARP テーブルを定期的に読み出し、NP における格納形態に加工した上で NP に対して通知することとした。NP 側では、制御 PC からの FIB 同期情報を StrongARM プロセッサで動作する FIB 同期のデーモンで受信し、MicroEngine が参照可能な SDRAM に格納する。このような仕組みで、制御 PC 上の FIB 情報がパケット転送処理に実際に用いられる NP 上の FIB 情報テーブルに反映される。

5.5 動作実験と課題

実装の動作を検証するため、図 5.7 に示す実験ネットワークを構築した。実験により得られた知見と本実装の問題点およびその解決策について述べる。

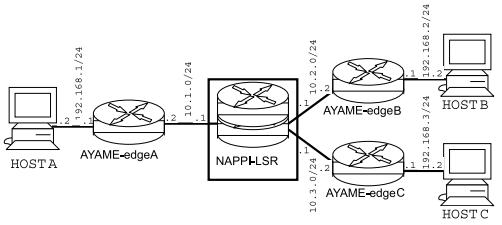

図 5.7.　実験ネットワーク

5.5.1 動作検証

本実装では MPLS のコアルータ機能のみを提供しているため、NP 搭載 AYAME LSR をコアルータとし、エッジルータにソフトウェア実装の AYAME LSR を配置した（図 5.7）。各 LSR ではシグナリングプロトコル LDP[3] によるラベルシグナリングを行う *ldpd* を動作させた。

今回実装した仮想インタフェース機能の検証として LDP ピアリングの正常動作の確認を行った。また telnet や ftp 等のアプリケーションを用いて NP 搭載 AYAME LSR の制御 PC と隣接ノード間で正常にデータ通信が行えていることを確認した。さらにラベルスワッピング機能の動作を検証するために Host A、Host B、Host C の間で TTL 値を調節した ping パケットを送受信した各 LSR 間のセグメントでは、パケットモニタを接続して LSP を介して ping パケットが転送されていることを確認した。また、telnet および ftp などの TCP アプリケーションにおいても同様に Host A、Host B、Host C 間の通信が LSP を介して転送されることを確認した。

5.5.2 実装における課題

一部リンクを切断した際の挙動や *ldpd* によらず手動でラベルスイッチング規則の設定を行った際の挙動調査などから、いくつかの課題が得られた。以下にそれらの課題を挙げる。

FIB 情報の同期　実験では、制御 PC 側の FIB 情報が更新された場合、NP 上の FIB 情報がその更新に追従するまでに若干の時間を要した。原因は制御 PC 側の FIB 情報を非同期ポーリングで取得する設計となっている点である。FIB 情報の更新の遅れは、障害時等のパケット到達性回復までの時間に影響を与えるため、FIB 情報更新をトリガとして同期更新する方式に変更する必要がある。

ARP エントリ期限　上述の実験ネットワークにおいて、*ldpd* を停止し手動でラベルスイッチング規則を設定した場合に、隣接ノードの ARP エントリが期限切れとなり転送処理が停止する現象が確認された。*ldpd* 動作中は制御 PC から隣接ノードに対する定期的通信が行われているためそれらに対する ARP

エントリは維持される。しかしそのような通信が存在しない場合、NP 上で転送処理が完結すると該当パケットの転送処理を行った事実が制御 PC に伝わらず、制御 PC 上の ARP テーブルが更新されない。

その他実装上の課題　今回の実装では実現を見送ったものとして以下が挙げられる。

- NP での SWAP 以外のラベル処理
- NP 上の FIB 情報の最適な配置の検討
- NP–制御 PC 間の PCI を用いた通信

まず、NP 上での SWAP 以外のラベル処理が実装されていないため本実装はコアルータ以外の動作はできない。エッジルータ分野における NP 搭載 AYAME LSR への要望も大きく、SWAP 以外のラベル処理の実現が強く求められている。また IP パケットの転送処理については制御 PC 側で行ったが、エッジルータとしての動作を考慮した場合、IP 転送機能も NP 上で動作させることが強く求められると考えられる。

次に、NP 上での FIB 情報の配置については、今回の実装ではメモリ容量の問題からその全体を SDRAM に配置した。SDRAM は読み書きに SRAM に比べ 2 倍程度の時間を要することを考慮すると、テーブルサイズが比較的小さな場合の配置手法や、大きな場合でも頻繁に用いられるものをキャッシュとして SRAM に持つなどのより最適な配置を検討する価値がある。

また、NP–制御 PC 間の通信方法について見ると、NAPPI1200 ボードには非対称 PCI ブリッジが用意されていたが、ホスト PC 側の OS および NP 側 OS のサポートの双方が必要なため利用を断念した。これらを用いることで効果的な接続が可能であるため今後の対応を検討すべきである。

5.6 まとめ

今回の論じた実装はコンセプトを実証するためのプロトタイプであり、設計自体の評価を行うことはできないが、ソフトウェア実装で実現されたネットワークスタックのパケット処理部分を外部接続された NP を用いて並列処理することで、ソフトウェア実装の利点を維持した高速ルータの実現が可能であることを示すことができた。

今後は、AYAME におけるパケット配送機構として利用可能な NP 実装を実現し、AYAME の適用範囲を拡大していく予定である。

謝辞

本研究は、北陸先端科学技術大学院大学と横河電機(株)との共同研究として進められた。関係諸氏に深謝する。

第 6 章　おわりに

6.1 おわりに

この報告では、NP-WG としての目標を示し、ワーキンググループのメンバーによって 2003 年度に発表された論文を紹介した。

活動目標として掲げたアプリケーション開発、環境整備、情報共有、そして人材育成は奈良先端科学技術大学院大学と北陸先端科学技術大学院大学でそれぞれ行われており、2003 年 9 月以降、両大学院で盛んな情報共有や技術交換を行っている。しかし、人材育成はまだ途中段階であり、報告可能な成果としてまとまっていないため、本報告においては掲載しない。

NP はまだまだ技術的に発展段階にあり、NP の持つ可能性や利用用途はまだ未知数である。多分野の研究者が NP に興味を持ち、NP そのものの研究、NP を用いたアプリケーション開発をさまざまな視点から行っていくことが、NP の可能性や利用用途を広げていく。多岐にわたる様々な研究分野で NP が応用されて行くことを NP-WG として望む。