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

信学技報
TECHNICAL REPORT OF IEICE
電子情報通信学会 NV 研究会 2013-7-19

ネットワーク・プロセッサのためのオープンで高級で移植可能なプログラミング環境

金田 泰
日立製作所 中央研究所
〒244-0817 神奈川県横浜市戸塚区吉田町 292

Yasusi.Kanada.yq@hitachi.com

あらまし ネットワーク・プロセッサは高性能でプログラマブルなネットワークのためにひろく使用されている．しかし，ネットワーク・プロセッサのプログラムは移植性と開発者数がかぎられ，開発コストが高い．この問題を解決するためにオープンで高級かつ移植性のあるプログラミング言語 CSP とそのための開発環境 +Net を開発した．この環境においては，パケットの高速処理に必要な SRAM と DRAM のつかいわけをプログラマにできるだけ意識させずに高いスループットがえられるようにした．ネットワーク・プロセッサ Cavium Octeon のためのプロトタイプを実装し，ネットワーク仮想化基盤の一部を使用して評価した結果，かんたんなプログラムでワイヤレートにちかい 7.5 Gbps 以上の性能をえた．

キーワード ネットワーク・プロセッサ，プログラマビリティ，移植性，SRAM，DRAM，Octeon，ネットワーク仮想化．

Open, High-level, and Portable Programming Environment for Network Processors

Yasusi Kanada
Central Research Laboratory, Hitachi, Ltd.
Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan

Yasusi.Kanada.yq@hitachi.com

Abstract Network processors are used for high-performance programmable networks. However, programs for network processors are limited in portability and number of developers, so the development cost is high. To solve this problem, open, high-level, and portable programming language called “CSP” and a development environment called “+Net” have been developed. In this environment, high throughput can be obtained without programmers’ significant awareness of SRAM/DRAM distinction. A prototype using Cavium Octeon, a network processor, has been developed, and it performs 7.5 Gbps or more in simple programs in an evaluation using part of the network virtualization platform.

Keywords Network processor, Programmability, Portability, SRAM, DRAM, Octeon, Network virtualization.

1. はじめに

広域ネットワーク技術への要求が多様化するなかで，インターネット関連技術は複雑化し，新規の要求にこたえることが困難になってきている．そのため，ネットワークを仮想化し，個々の仮想ネットワークを構築し，それぞれの要求に独立に応えることを可能にする仮想化ネットワーク・アーキテクチャが有力な解となりつつある．この仮想化ネットワークの利点をいかすには，ワイヤ・レートまたはそれにちかい高速なネットワーク機能をネットワーク・ベンダやサービス・プロバイダが容易に実現することができるプログラマビリティを実現する必要がある．

このプログラマビリティを実現するためのひとつのハードウェアの候補が Intel IXP [Gog 03], Cavium Octeon [Cav 10] 等のネットワーク・プロセッサである．ネットワーク・プロセッサの導入によって，すでにさまざまな高度なネットワーク機能をもつ機器が実現されてきている．しかし，ネットワーク・プロセッサを使用するための開発にはつぎのような問題点がある．

- **移植性がないこと**: アセンブリ言語にちかい低水準言語を使用せざるをえないため，プログラムに移植性がない．
- **開発者が限定されること**: プログラム開発に特殊技術を要するうえ，開発に必要な知識が流通していないため，開

発能力をもつベンダや開発者コミュニティが限定されている.

- **コスト高**: 開発者が限定され, 特定ベンダの技術や開発環境に依存し, 技術の習得に時間がかかるため, 開発コストがおおきい.

今後, 多様なネットワーク機能を実現していくためには, 移植性のあるプログラミング言語を用意し, 多数のプログラマにとってプログラミングが容易にできる開発環境をととのえて, これらの問題点を解決する必要があるとかんがえられる. そのため, この研究においては, 仮想化ノード (VNode) プロジェクト [Nak 10] において開発した仮想化ノード (VNode) [Nak 12][Kan 12a] およびネットワーク収容装置 NACE (または NC) [Kan 12b] にノード処理やプロトコル変換処理などを高速で実行することができるネットワーク・プロセッサを使用したボードを付加し, このプロセッサのために高速処理が容易に記述できるプログラミング言語とプログラミング環境を提供することをおもな目的としている.

一般に, 高速なマルチコアのネットワーク・プロセッサにおいては, ワイヤレートの処理のためには一部のデータを SRAM に格納する必要があるが, 大量のパケット・データを SRAM に格納することはできないため, 大半のデータは DRAM に格納する必要がある. IXP などのネットワーク・プロセッサにおいては, このようなハードウェア要素をプログラマが明確に意識し, つかいわける必要がある. Octeon のようにキャッシュをもつ汎用の CPU コアを使用したネットワーク・プロセッサにおいてはプログラマの負担は軽減されているが, それでもハードウェアを意識する必要がある.

ネットワーク・ノードのプログラミング環境として Click [Koh 00] があり, そのネットワーク・プロセッサのための実装として NP-Click [Nir 04] がある. また, ネットワーク・プロセッサ IXP のためには Shangri-la というプログラミング環境と Baker というプログラミング言語 [Che 05] が開発されている. しかし, これらのいずれにおいても, プログラマはハードウェアとくに SRAM と DRAM との区別を意識する必要がある.

しかし, このような SRAM, DRAM のつかいわけの指示やハードウェア依存のプログラム記述はプログラマのおおきな負担となり, プログラミングのコスト・アップにつながる. また, このようなプログラムに実行時にパラメタをわたすとき, ハードウェアまたは その専用ソフトウェアごとにことなる方法をとる必要があり, これもプログラマの負担になる. この研究の目的はこのようなつかいわけやパラメタわたしを処理系が汎用的な方法で自動的に実現することにより, それをプログラマが意識せずに, ハードウェアやその専用ソフトウェアなどに依存しない方法でできるようにすることである.

2. パケット処理言語 CSP

上記の目的を達成するために, ネットワーク・プロセッサのためのプログラミング環境 +Net を開発し, そのための汎用性のある高級言語 CSP を設計・実装した. +Net 上の CSP 処理系を +Net CSP とよぶ. CSP は Java や C++ のプログラマにあつかいやすいように Java にちかい構文を採用し, 文やメソッドの種類や意味も Java にちかづけるようにした. そのため, Java と比較しながら, CSP の設計方針と仕様, プログラム例などについてのべる.

2.1 CSP の基本設計

まず CSP の 4 つの設計方針についてのべる.

- **Immutable なバイト列としてのパケット**

  パケットは可変長なのでバイト列 (文字列) あるいはビット列としてあつかうことができる. ビット列はあつかいにくいので, 文字列と同様なバイト列としてあつかう. すなわち, パケットを OSI におけるような TCP/IP/Ethernet というような階層化されたオブジェクト (階層ごとに隠蔽されたオブジェクト) ないし抽象データ型としてはあつかわない. これによって, パケットの操作に関する最適化が比較的容易におこなえるようになる. Java は TCP/IP を従来の方法であつかっているのに対して, CSP におけるプロトコルのあつかいは Java とはまったくことなる. 動作環境として Linux や Microsoft Windows などの OS を使用するときは, 文字列型としてのパケットを IP パケットなどにキャストすることによって, それらの環境で用意されているプロトコル処理の関数やメソッドを使用することができる.

  また, Java やその他のおおくの言語における文字列と同様に, CSP ではパケットを immutable な (かきかえられない) オブジェクトとしてあつかう. これは, パケットに操作をおこなったとき, 操作前のオブジェクトと操作後のオブジェクトとで同一のデータ領域を共用できるようにするためである.

- **バイト列とパケットをくべつするか?**

  バイト列とパケットは論理的には同一とみなすことができるが, プログラム上でのあつかいはおおきくことなる. 効率やコンパイラでのあつかいやすさをかんがえても, ことなる型としてくべつするのが適切である. そのため, バイト列型 String とはことなる Packet 型を新設する.

  これらの型の実装においてはつぎのような仮定をおく.

  - バイト列は基本的に全体がキャッシュ可能なメモリ (基本は DRAM 上だが必要に応じて SRAM にキャッシュされる) にあるものとみなす.
  - パケットは基本的には先頭だけがキャッシュされ, 後続部分は DRAM にだけある (キャッシュできない) ものとみな

す．ただし，みじかいパケットは全体がキャッシュされるし，SRAM 上だけにある場合もある．

- **部分列生成におけるパケット処理メソッドとバイト列処理メソッドとの関係**

  パケットの一部を除去してことなるパケットを生成したり (subpacket)，パケットからバイト列を生成したり (substring) することができる．これらのパケット操作はいずれもバイト列の操作としては substring に対応するが，結果型がことなるためにくべつされる．結果型によるオーバーローディングを許容する言語においてはこれらにともに substring という名称をあたえることができる．しかし，CSP においては結果型によるオーバーローディングをみとめないため，substring と subpacket という，ことなる名称をあたえる．すなわち，substring は String 型のオブジェクトをかえし，subpacket は Packet 型のオブジェクトをかえす．

- **列の連接におけるパケット処理メソッドと列処理メソッドとの関係**

  バイト列どうしを連接してバイト列を生成したり (concat)，バイト列とパケットを連接してことなるパケットを生成したり (new Packet) することができる．これらに同名のメソッドを使用するようにすることも可能だが，CSP においては後者のための方法を限定している．当面，容易に実装できるようにするためである．論理的にはパケットどうしを連接してパケットを生成することが可能である．しかし，実用上の意味があまりないとかんがえられるため，このような連接のためのメソッドは用意しない．

```
001 import NetStream1;
002 import NetStream2;

003 class AddRemMAC {
004   NetStream out1;
005   NetStream out2;

006   public AddRemMAC(NetStream port1 > process1,
007                    NetStream port2 > process2) {
008     out1 = port1;
009     out2 = port2;
010   }

011   void process1(Packet i) {
                        // Port 1 to 2 (no VLAN -> no VLAN)
012     Packet o = new Packet(i.substring(0,14),i);
          //  MAC header of original packet (i: Original packet)
013     out2.put(o);
014   }

015   void process2(Packet i) {
                        // Port 2 to 1 (no VLAN -> no VLAN)
016     Packet o = i.subpacket(14);
          // remove MAC header (no VLAN)
017     out1.put(o);
018   }

019   void main() {
020     new AddRemMAC(new NetStream1(),
021                   new NetStream2());
022   }
023 }
```

図 1 単純な MAC ヘッダ追加 / 削除の CSP プログラム

2.2 プログラム例

言語仕様の概要をしめすため，図 1 にパケットの MAC ヘッダの削除 / 追加をおこなうプログラムを例としてしめす．CSP においては，パケットはその種類あるいは処理方法のちがいによってクラスにわけられる．このプログラムにおいては **NetStream1** および **NetStream2** という 2 つのパケット・ストリームをあつかう．

このプログラムの機能は **NetStream1** から入力されたパケットにはその MAC ヘッダをコピーして先頭に追加して **NetStream2** に出力し，同時に **NetStream2** から入力されたパケットからは MAC ヘッダを先頭から削除して **NetStream1** に出力することである．いずれにおいても，CSP プログラムを単純化するため，もとのパケットにどのようなヘッダがあるかはチェックしていない (したがって，たとえば **NetStream2** から入力されたパケットに MAC ヘッダがついていなくても，MAC ヘッダぶんのデータが削除されたうえで出力される)．しかし，チェック・コードを挿入するのは容易である．

クラス **AddRemMAC** が初期化されると，関数 **main()** が実行される．関数 **main()** においては **AddRemMAC** のインスタンス (singleton) を 1 個生成している．その引数として，2 個のパケット・ストリームが生成されたうえでわたされている．これらのパケット・ストリームは，インスタンス生成されることによってはじめて，パケット入出力が可能になる．なお，これらのパケット・ストリームがハードウェアのどのインターフェースに対応するかは，このプログラムの外部であたえられる (プログラム内では指定されない)．

クラス **AddRemMAC** の構築子においては，その引数宣言において，最初の引数 **port1** への入力がメソッド **process1** にわたされ (**NetStream port1 > process1**)，2 番めの引数 **port2** への入力がメソッド **process2** にわたされることが指定されている．このような引数宣言の記法は CSP 固有のものである．これらの引数の値 (パケット・ストリーム) はクラス内で参照できるようにするためにクラス変数 **out1**, **out2** に代入されている．メソッド **process1** および **process2** には一度に 1 個ずつ，パケットがわたされる．CSP においてはパケット入力がつねにこのようにあつかわれるため，入力のためのメソッドや文は存在しない．

NetStream1 から入力されたパケットをあつかうメソッド **process1** においては，入力パケット **i** の先頭 14 バイト (MAC ヘッダが 14 バイトであることを仮定) を部分バイト列として生成して (**i.substring(0, 14)**) **i** の先頭に連接したパケット **o** を生成し (**new Packet(..., i)**)，それを **NetStream2** (**out2**) に出力している．

NetStream2 から入力されたパケットをあつかうメソッド

process2 においては，入力パケット **i** の先頭 14 バイトを除外した部分 (MAC ヘッダが 14 バイトであることを仮定) を部分パケットとして生成して (**i.subpacket(14)**)，それを **NetStream1** (**out1**) に出力している．

2.3 CSP におけるパケット操作

CSP におけるパケット操作の方法について，よりくわしくのべる．

2.3.1 パケット入出力

前記の例にもとづいて，CSP におけるパケット入出力の方法について説明する．第 1 にパケット入力の方法について説明する．図 1 の例におけるように，CSP においてはパケットはその種類あるいは処理方法のちがいによってクラスにわけられ，パケット処理のためのメソッドが定義される．パケットが入力されるごとにそのクラスの指定されたメソッドがよびだされるため，パケット入力のためにメソッドをよびだしたり，文を記述したりする必要はない (パケット入力を明示的に記述することはできない)．

第 2 にパケット出力の方法について説明する．図 1 の例におけるように，CSP においてはパケット出力には出力するべきパケット・ストリーム (図 1 においては **NetStream1** および **NetStream2**) がもつ **put** メソッドを使用する．

2.3.2 部分列生成

2.1 節においてのべたように，CSP においては substring と subpacket とをくべつする．そのため，部分列をもとめる操作にはつぎの 3 種類がある．

1. **Packet Packet.subpacket(from);**
2. **String Packet.substring(from [, to]);**
3. **String String.substring(from [, to]);**

プログラマ (開発者) は生成されるのが (入力パケットに依存する) パケットかどうかを意識する必要がある．1. においては，指定された範囲がすべて SRAM 上だけにあると実行時に判定されたとき，エラーと判定する．2. においては，指定された範囲にキャッシュされていない (DRAM 上だけにある) データがふくまれるとき，エラーと判定する．3. においては，指定された範囲がすべて SRAM 上にあるとき以外はエラーと判定する．

2.3.3 列の連接

2.1 節においてのべたように，CSP においてはパケットの連接にはバイト列の連接とはことなる方法を使用する．すなわち，列の連接にはつぎの 2 種類がある．

1. **new Packet(String, …, String, Packet);**
2. **String String.concat(String);**

1. でつくられるデータは頭部がキャッシュされ，尾部がキャッシュされていない．2. でつくられるデータは全体がキャッシュされている．

2.3.4 列の生成

まえの 2 つの節においては，すでに存在する列をもとにして，あたらしい列を生成するメソッドについてのべた．この節においては，まだ列が存在しないところから列を生成する方法についてのべる．列の連接の一方法として記述した new Packet() は，本来は列をまったくあらたに生成するための構築子である．基本は String をもとに Packet を生成する．

- **new Packet(String);**
  前節の構築子と同様に複数個の引数を許容することも可能である (前節とのちがいは，すべての引数が String だということである)．

2.3.5 列-整数 間変換

バイト列の操作ではバイト単位の操作しかできないが，パケット操作においてはバイト境界に制約されない操作が必要になる．また，パケットからとりだされた値によるインデクシング (配列アクセス等) も必要になる．これらの操作を可能にするため，列と整数とのあいだの変換関数を用意した．

- **String.itos(ivalue, length)**
  String クラスの静的メソッド String.itos(ivalue, length) は整数値 ivalue を 1 バイトごとにくぎって，ながさ length のバイト列に変換する．たとえば，String.itos(257, 2) は整数 257 (x0101) をバイト列 "\x01\x01" に変換する．Java には整数を文字列に変換する String.valueOf(ivalue) が存在するが，これは整数値をあらわす文字列をもとめるのに対して，CSP のメソッド String.itos() は内部表現をそのままにして型だけ変換する．

- **Integer.stoi(svalue)**
  Integer クラスの静的メソッド Integer.stoi(svalue) は列 svalue を 256 進整数とみなして 64 bit の整数値に変換する．たとえば，Integer.stoi("\x01\x01") の結果は 257 (x0101) である．変換後の整数値が 64 bit に限定されるため，Integer.stoi() への引数は 8 バイト以下の列でなければならない．Java には文字列を整数に変換する Integer.valueOf(svalue) が存在するが，これは文字列があらわす整数値をもとめるのに対して，CSP のメソッド Integer.stoi() は内部表現をそのままにして型だけ変換する．

3. 実装法

CSP の実装法についてのべる．その最大の特徴は，プログラマが意識しなくても SRAM と DRAM とのつかいわけができるようにするため，パケットというひとつのデータ型に 4 つの表現形をもうけていることである．

おおくのネットワーク・プロセッサにおいて，入力されたパケットは入力データ表現用の形式で表現され，出力されるパ

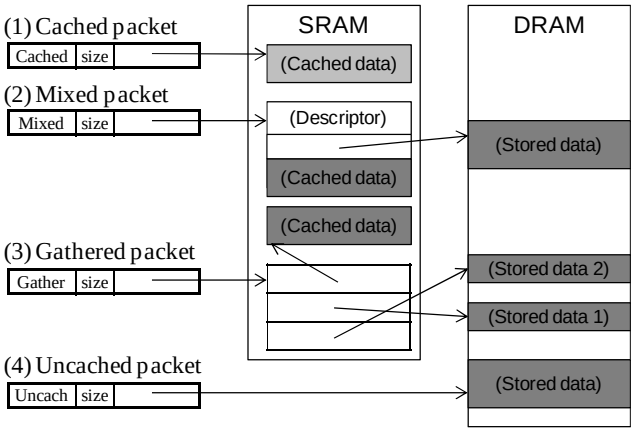

図 2 4 つのパケット表現形

ケットは出力データ表現用の形式で表現される．C 言語によるプログラミングにおいては通常これらをバラバラにあつかうが，CSP を使用することにより，これらを仕様上，統一的にあつかえるようにし，コンパイラがそれらを実装上，統一的にあつかえるようにする必要がある．

パケットのデータ表現は本質的にはつぎの 4 種類だとかんがえられる (**図 2** 参照)．

- **Cached**: パケット全体が SRAM 上にある．そのコピーが DRAM 上にあることは仮定しない．
- **Mixed**: パケット先頭部分 (バイト数は可変) が SRAM 上にあり，パケット全体が DRAM 上にある．
- **Gathered**: パケットの構成要素が複数の部分から構成されている．各部分は不連続な領域にある．Gathered パケットは要素の配列または要素の linked list によって表現される．
- **Uncached**: パケット全体が DRAM 上にある．そのコピーが SRAM 上にあることは仮定しない．

これらのうち Cached 形式と Uncached 形式は単純な構造をしているが，それらとくらべると Mixed 形式および Gathered 形式は複雑であり，特殊だということができる．以下，これらのデータ形式がなぜ必要とされるかについてのべる．

- Mixed 形式が必要なのは，パケット処理においては入力されたパケットのヘッダだけを加工して出力することが多いためである．このような場合はパケット入力時にハードウェアによってヘッダをふくむパケット先頭部分だけを SRAM に格納し，のこりを DRAM に格納することが有効である．DRAM に格納されたデータをロードして処理するとスループットが低下し，ワイヤ・レートでの処理ができなくなる．しかし，パケットの先頭を SRAM に格納することによって，ヘッダの処理でスループットが低下するのをふせぐことができる．
- Gathered 形式が必要なのは DRAM あるいは SRAM に格納された複数のデータ (列) を連接してパケットを生成するときである．このようなときに全データをあらかじめ DRAM の連続領域におこうとするとデータを DRAM から DRAM にコピーする必要が生じる．

上記の 4 種類以外のケースもありうるが，すべてを考慮すると複雑になりすぎるため，すくなくとも当面はこれら 4 種類だけを実現することをかんがえる．これらを実行時にくべつする必要が生じることがあるので，データ構造のなかにこれらをくべつするためのタグをもうける必要がある．タグはデータ・ポインタ上にもうけることがのぞましい．しかし，そのばあいタグを既存のデータ構造にうめこむことになるため，互換性に関して注意が必要である．

なお，上記のデータ構造は概念的なものであり，具体的なデータ構造はネットワーク・プロセッサの種類に依存し，図 2 とはことなる構造をとる必要が生じる．たとえば，構造化された (すなわち複数のフィールドやフラグなどをふくむ) ポインタがハードウェアによってサポートされているときは，タグもポインタのフィールドのひとつとしてポインタにうめこむことがのぞましいとかんがえられる．しかし，未使用の領域がないときは，アドレスの上位ビットを使用するなど，既存の領域をうまく使用する必要がある．

部分列生成，列の連接，列の生成などに対応する具体的な操作はデータ表現に依存する．すなわち，これらの操作はデータ表現ごとに場合わけしておこなう必要がある．ここではその詳細はのべないが，操作によるパケット・データ表現 (状態) の変化を**図 3** に状態遷移図としてしめす (エラーが発生する場合もあるが，それはこの図には記述していない)．

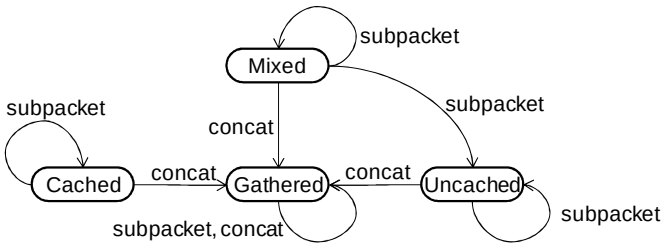

図 3 パケット操作による 4 つのデータ表現間の状態遷移

4. プロトタイピングと評価

プログラミング環境 +Net を試作し，そのうえに Octeon のための CSP 処理系のプロトタイプ +Net CSP を開発した．このプロトタイプを図 1 にしめした MAC ヘッダ削除 / 追加プログラム AddRemMAC と，パケットを加工せずに通過させる Pass-through のプログラムを使用して評価した．評価には VNode プロジェクトによって開発されたネットワーク仮想化基盤を構成するノードであるネットワーク収容装置 (Network ACcommodation Equipment, NACE) [Kan 12b] に Octeon を搭載したネットワーク・プロセッサ・ボードを接続しておこなった．使

用したボードは GE 社の WANic-56512 であり, 12 個の 750-MHz の Octeon Plus コアを搭載している. この構成を使用することによって, ネットワーク仮想化基盤にさまざまなパケット・フォーマットをもつ外部ネットワークを接続することができるようにすることをめざしている.

性能測定の結果を**図 4** にしめす. 前記の 2 つのプログラムにおいて, パケット・サイズが 256 バイト以上のとき, ワイヤレートにちかい 7.5 Gbps をこえるスループットが実現されている.

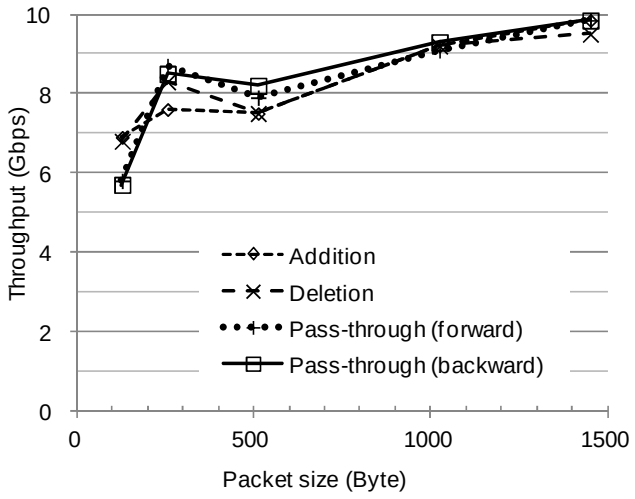

図 4 MAC ヘッダ追加 / 削除のプログラムの測定性能

5. 結言

ネットワーク・プロセッサのためのオープンで高級で移植性のあるプログラミング言語 CSP とそのための開発環境 +Net を開発した. CSP を使用することにより, プログラマは SRAM と DRAM のくべつを意識せずにそれらをつかいわけることができる. この環境でパケットをうまくあつかうため, 言語実装にあたってパケットというひとつのデータ型の 4 つのデータ表現をつかいわけることを提案した.

+Net 上に Octeon のための CSP 処理系のプロトタイプ +Net CSP を開発して評価した. その結果, 単純なパケット・ヘッダ追加 / 削除の CSP プログラムなどにおいて, パケット・サイズが 256 バイト以上のとき, ワイヤレートにちかい 7.5 Gbps をこえるスループットを実現した. この結果はこの方法がネットワーク・プロセッサのプログラミングを容易に低コストにするために有望なものであることをしめすとともに, 仮想化基盤 (VNode) にそのプログラミング機能を提供できる可能性をしめしている.

謝辞

この報告の内容は情報通信研究機構 (NICT) の委託研究「新世代ネットワークを支えるネットワーク仮想化基盤技術の研究開発」(課題ア) の研究開発成果をふくむ.

参考文献

[Cav 10] "OCTEON Programmer's Guide, The Fundamentals", Cavium Networks, 2010, http://university.-caviumnetworks.com/downloads/-Mini_version_of_Prog_Guide_EDU_July_2010.pdf

[Che 05] Chen, M. K., Xiao Feng Li , Lian, R., Lin, J. H., Lixia Liu, Tao Liu, and Ju, R., "Shangri-La: Achieving High Performance from Compiled Network Applications while Enabling Ease of Programming", *2005 ACM SIGPLAN Conference on Programming Language Design and Implementation* (*PLDI'05*), pp. 224–236, 2005.

[Gog 03] Goglin, S. D., Hooper, D., Kumar, A, and Yavatkar, R., "Advanced Software Framework, Tools, and Languages for the IXP Family", *Intel Technology Journal*, Vol. 7, No. 4, pp. 64–76, 2003.

[Kan 12a] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-Virtualization Nodes that Support Mutually Independent Development and Evolution of Components", *IEEE International Conference on Communication Systems* (*ICCS 2012*), November 2012.

[Kan 12b] Kanada, Y., Shiraishi, K., and Nakao, A., "High-performance Network Accommodation into Slices and In-slice Switching Using A Type of Virtualization Node", *2nd International Conference on Advanced Communications and Computation* (*Infocomp 2012*), IARIA, October 2012.

[Koh 00] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Frans Kaashoek, M., "The Click Modular Router", *ACM Transactions on Computer Systems* (*TOCS*), Vol. 18, No. 3, pp. 263–297, 2000.

[Nak 10] Nakao, A., "Virtual Node Project — Virtualization Technology for Building New-Generation Networks", *NICT News*, No. 393, pp. 1–6, Jun 2010.

[Nak 12] Nakao, A., "VNode: A Deeply Programmable Network Testbed Through Network Virtualization", *3rd IEICE Technical Committee on Network Virtualization*, March 2012, http://www.ieice.org/~nv/05-nv20120302-nakao.pdf

[Nir 04] Niraj Shah, Plishker, W., Kaushik Ravindran, and Keutzer, K., "NP-Click: a Productive Software Development Approach for Network Processors", *IEEE Micro*, Vol. 24, No. 5, pp. 45–54, 2004.