メニーコアとクラウドを融合するための Network-on-Chip アーキテクチャ
Network-on-Chip Architecture Integrating Many-Cores and Cloud Computing

代表研究者　　松谷 宏紀　　慶應義塾大学 理工学部 専任講師

概要

クラウドコンピューティングとは、ネットワーク上の共有の計算資源をネットワーク経由で利用するモデルであり、主にインターネットで接続された多数の計算機によるサービスを指す。一方で、半導体技術の進歩によって、単一チップ上に多数のプロセッサを搭載したメニーコアプロセッサが実現可能となり、Web サーバのように高いリクエストレベル（スレッドレベル）並列性を持つ用途への応用、つまり、クラウドコンピューティングへの応用が期待されている。

これまで、メニーコア同士の接続には、機能を絞ったチップ内ネットワーク（Network-on-Chip、NoC）が使われてきたが、将来、このようなメニーコアがクラウドコンピューティング環境の一部となるには、チップ内の NoC とチップ外のインターネットをシームレスに接続する技術が必要不可欠である。メニーコアプロセッサは Ethernet や PCI Express などの I/O を持つため、この I/O を介して、チップ内とチップ外でデータのやり取りを行うことになる。本研究では、メニーコアの外部ネットワーク I/O として、インターネットプロトコル（IP）と NoC のプロトコル変換を行う IP-NoC トランスレータを実現した。これより、チップ内・チップ外を意識することなく IP および IP アドレスを用いてチップ内外の計算資源にアクセスすることが可能となり、クラウドコンピューティング環境の一部としてのメニーコアの利用が加速する。

1 はじめに

シングルプロセッサの性能向上が頭打ちになりつつある昨今、複数プロセッサから成るマルチコアプロセッサ、多数のプロセッサから成るメニーコアプロセッサが盛んに研究開発されている。図 1 に最近のマルチコア・メニーコアプロセッサの一覧を示す。例えば、Sun Microsystems（現 Oracle）社の T2 プロセッサ[Shah07]は、UltraSPARC プロセッサを 8 コア搭載したプロセッサであり、主に Web サーバなどリクエストレベル並列性の高い用途に応用されている。TILERA 社の TILE64 プロセッサ［Wentzlaff07］は、VLIW 型のプロセッサ 64 個がチップ内ネットワーク（Network-on-Chip、NoC）［Dally01］で接続された構成をとり、個々のプロセッサでは Linux が動作する。Intel 社の Single-chip Cloud Computer（SCC）［Howard10］においても 48 個の IA-32 プロセッサが NoC で接続された構成をとり、メッセージ交換型（Message Passing Interface 型）の通信ライブラリを用いてプロセッサ間でデータのやり取りを行う。最近では、Intel 社は 60 個のプロセッサを搭載した Xeon Phi [Jeffers13]を製品化した。このように 1 チップに集積されるプロセッサの数は年を追うごとに増えている。独立に動作可能なプロセッサコアを多数有したこのようなメニーコアプロセッサは、Web サーバ、データベースサーバ、トランザクション処理などリクエストレベル並列性［Hennessy11］を有すアプリケーションにおいて高いスループット性能を実現できるため、クラウドコンピューティングを実現するためのハードウェア技術として重要であると考えている。

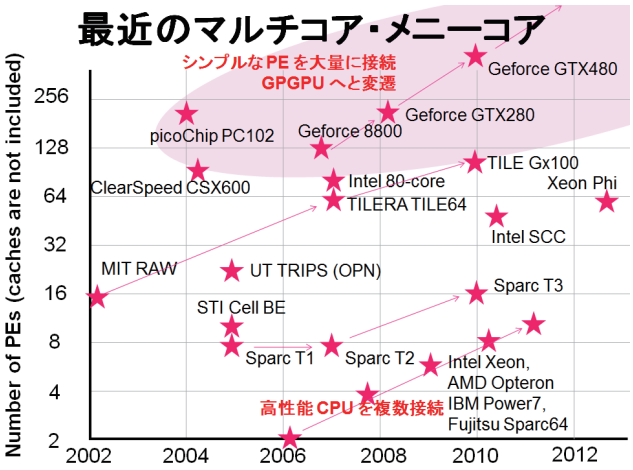

図 1: 最近のマルチコア・メニーコアプロセッサ

クラウドコンピューティングとは、ネットワーク上の共有の計算資源をネットワーク経由で利用するモデルであり、主にインターネットで接続された多数の計算機によるサービスを指す（図 2 上部）。一方で、半導体技術の進歩によって、単一チップ上に多数のプロセッサを搭載したメニーコアプロセッサ（図 2 下部）が実現可能となり、Web サーバのように高いリクエストレベル並列性を持つ用途への応用、つまり、クラウドコンピューティングへの応用が期待されている。

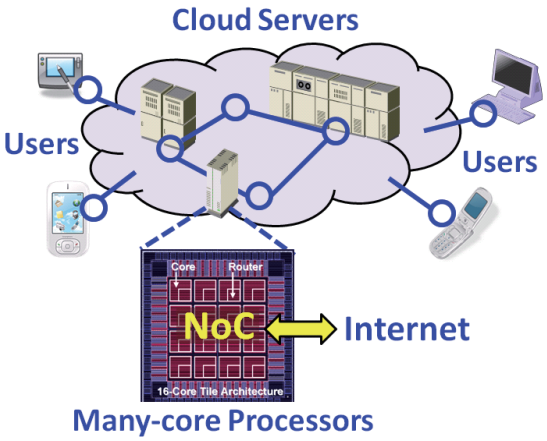

図 2: クラウド、メニーコア、NoC の概要

これまで、メニーコア同士の接続には、機能を絞った NoC が使われてきたが、将来、このようなメニーコアがクラウドコンピューティング環境の一部となるには、チップ内の NoC とチップ外のインターネットをシームレスに接続する技術が必要不可欠である。メニーコアプロセッサは Ethernet や PCI Express などの I/O を持つため、この I/O を介して、チップ内とチップ外でデータのやり取りを行うことになる。本研究では、メニーコアの外部ネットワーク I/O として、インターネットプロトコル（IP）と NoC のプロトコル変換を行う IP-NoC トランスレータ（図 2 黄色矢印の部分）を実現する。これより、チップ内・チップ外を意識することなく IP および IP アドレスを用いてチップ内外の計算資源にアクセスすることが可能となり、クラウドコンピューティング環境の一部としてのメニーコアの利用が加速すると考えている。

本報告書の構成は以下のとおりである。2 章にて提案する IP-NoC トランスレータの全体像について述べる。3 章では、本研究のために構築したメニーコアプロセッサの評価環境について述べ、4 章にて IP-NoC トランスレータの設計と実装について述べる。5 章にて現在までの進捗と今後の展開について述べる。

2 インターネットと NoC を接続する IP-NoC トランスレータの概要

図 3 に本研究が対象とするメニーコアプロセッサのアーキテクチャ図を示す。単一チップ上に多数（図 3 では 16 個）のプロセッサが集積され、それらがオンチップルータ回路で接続されている。また、チップ外との入出力のために Ethernet I/O を装備している。

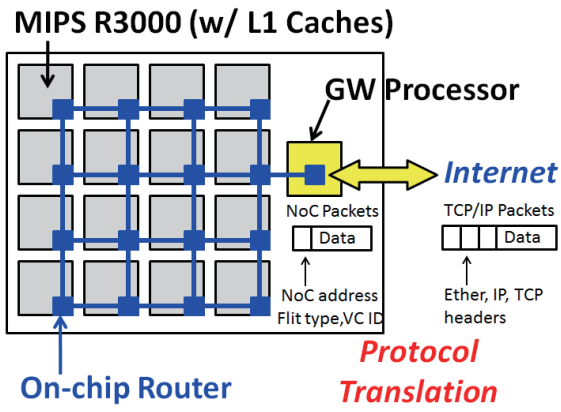

図 3: メニーコアと IP-NoC トランスレータ

プロセッサコアごとに独立したキャッシュおよびメモリ空間を持ち、プロセッサコアごとに独立にオペレーティングシステム（OS）が動作することを想定している。メモリを共有しないためキャッシュコヒーレンスプロトコルは不要であるが、プロセッサコア間の通信には MPI などの通信ライブラリを用いることになる。このような設計思想は Intel 社の SCC [Howard10]に近いと言える。

これらのプロセッサコアはオンチップルータによって接続される。NoC では、オンチップルータごとに数ビットのアドレスが付与されており、当然、インターネットプロトコルとの互換性は無い。また、NoC においてはスイッチング方式として Wormhole 方式もしくは Virtual cut-through 方式が用いられており、インターネットで広く用いられている Store-and-forward 方式とは異なる。このようにチップ内の通信プロトコルとチップ外の通信プロトコルには互換性が一切無いため直接通信することはできないし、そもそも直接通信することを想定して設計されていない。しかし、1 章で述べたとおり、メニーコア上の個々のプロセッサ上で動作する OS に対し、インターネットプロトコルを用いて個別に通信できるようになれば、チップ内外の計算資源はその物理的位置に依らず 1 つの計算資源のプールとして仮想化され、負荷に応じて動的に分配、利用できるようになるためクラウドコンピューティングを実現するうえでメリットが大きいと考えられる。

最近では、メニーコアプロセッサの商用製品がいくつも登場するようになり、Ethernet や PCI Express などの実用的な入出力を持つものが多い。図 3 では黄色矢印の部分が Ethernet I/O である。この部分にインターネットプロトコルと NoC の間のプロトコル変換を行う IP-NoC トランスレータを実装することで、チップ内

とチップ外のノードがインターネットプロトコルを用いて直接通信できるようになる。

3 インターネット接続を有したメニーコアプロセッサの評価環境

本研究で実現する IP-NoC トランスレータを評価するために、本研究では、まず、メニーコアプロセッサの評価プラットホームを構築した。対象とするメニーコアプロセッサでは、2 章で述べたアーキテクチャを想定しており、チップ外ネットワークとの接続のために Ethernet I/O を持つ。本研究では、1) フルシステムシミュレーション環境、2) Verilog HDL モデルによるハードウェアシミュレーション環境の 2 種類の評価環境を構築した。

1 つ目のフルシステムシミュレーション環境は、既存のマルチコアプロセッサのフルシステムシミュレータである GEM5 [Binkert11] を基に構築した。GEM5 ではマルチプロセッサ、キャッシュ、メモリ、NoC などのハードウェアを正確にシミュレーションでき、その上で Linux などの実際のソフトウェアを動作させることができる。GEM5 は C++言語と Python 言語で書かれており、ソースコードは公開されているため、GEM5 の NoC に関する部分を修正することで後述の IP-NoC トランスレータを実装した。

2 つ目のハードウェアシミュレーション環境では、すべての機能をハードウェア記述言語である Verilog HDL 言語で記述することで、cycle-accurate にシミュレーションすることができ、さらに、ハードウェア量や消費電力などコストに関する評価を行うことができるようになる。プロセッサコアとしては、研究計画の時点では OpenRISC の採用を考えていた。OpenRISC はシンプルな構成でありながら、命令・データキャッシュ、メモリ管理ユニットなどを装備しており、開発環境や OS (Linux) も整っている。結局、開発時間の都合から MIPS 命令セットをサポートするシンプルなプロセッサコアである hmc-mips [hmc-mips] を採用したが、将来的には OpenRISC に移行する予定である。オンチップルータ回路としては申請者が開発した実装[Matsutani11]を使っている。図 6 にルータのアーキテクチャ図を示す。スイッチング方式は Virtual cut-through スイッチング、ルーティングアルゴリズムは次元順ルーティング、データ幅（フリット幅）は 128-bit、仮想チャネルを有し、バッファサイズは仮想チャネル当たり 16-flit である。さらに、プロセッサコアとオンチップルータの間のインタフェース回路（Network Interface、NI）として小規模な FIFO バッファを実現した。これらのプロセッサコア、オンチップルータ、NI はすべて論理合成可能な設計となっている。一方、Ethernet コントローラ自体の設計は本研究の範囲を超えるため、今回はシステムレベルでの記述に留めてあり、論理合成可能な実装は行っていない。論理合成可能かつフリーな実装があれば置き換えていきたいと考えている。

図 4 に今回実装した NoC のパケットフォーマットを示す。1 個のパケットは 1 個のヘッダフリットおよび 0 個以上のボディフリットから構成される（最後のボディフリットをテイルフリットと呼ぶ）。フリットの種類はフリットごとに付与した 3-bit の識別子で区別する。オンチップルータでは、ヘッダフリットを受信したらクロスバスイッチを確保し、テイルフリットの通過とともに確保したクロスバスイッチを開放する（図 6）。ヘッダフリットには NoC 中の送信元アドレス、宛先アドレスなどの情報が格納される。アドレスフィールドのビット数は NoC のノード数によって決まる。例えば、NoC 中に 64 個のノードが実装されている場合はアドレスフィールドのビット数は 6-bit となる。4 章では、このような NoC パケットとインターネットパケットとの間のプロトコル変換変換を行う IP-NoC トランスレータの設計と実装について説明する。

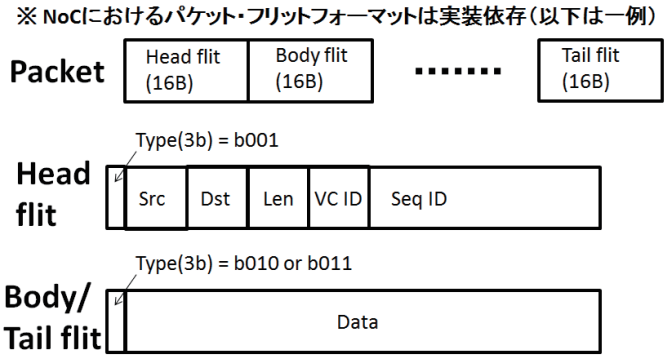

図 4: NoC におけるパケットフォーマット

4 IP-NoC トランスレータの設計と実装

4-1 ソフトウェア版 IP-NoC トランスレータ

上述のメニーコア評価プラットホーム上に、IP-NoC トランスレータをソフトウェアで実現した。IP-NoC トランスレータが動作するプロセッサ GW プロセッサと呼ぶ（図 3 の黄色の四角）。この GW プロセッサを

Ethernet コントローラに接続し、チップ外から受信した TCP/IP パケットを GW プロセッサにてソフトウェアで処理する。ソフトウェア版 IP-NoC トランスレータで行うことは、まず、パケットフォーマットの変換である。図 4 に NoC 内での通信に使用するパケットフォーマットを示し、図 5 に Ethernet ヘッダ、IP ヘッダ、UDP ヘッダを含む一般的な IPv4 のパケットフォーマットを示す。Ethernet フレームの最大サイズは一般的には 1518 Byte である（そのうちヘッダは 18 Byte である）。IP パケットから NoC パケットへの変換は以下の手順に従う。

1. GW プロセッサにおいて Ethernet フレーム全体を受信する。
2. Ethernet フレームから、Ethernet ヘッダ、IP ヘッダ、UDP ヘッダを取り除いたアプリケーションデータを得る。
3. アプリケーションデータをフリットサイズである 16 Byte ごとに分割し、それぞれをボディフリットとする。ただし、最後のボディフリットはテイルフリットとする。
4. IP の宛先アドレスおよび UDP の宛先ポートに応じて、NoC 内の宛先ノードを決定する。このマッピングには専用の変換テーブルを用い、この変換テーブルはユーザレベルアプリケーションから書き換え可能としている。
5. NoC 内の宛先ノードアドレスにしたがい NoC のヘッダフリットを構築する。
6. ヘッダフリット、データフリット、テイルフリットを結合して 1 つの NoC パケットを構築する。
7. 構築した NoC パケットを NoC に注入する。

なお、NoC パケットから IP パケットへの変換の場合はこの逆の手順を踏む。具体的には、NoC 中のプロセッサコアが生成した NoC パケットを GW プロセッサで一度すべて受信してから、NoC パケットの宛先アドレスをもとに宛先 IP アドレスおよび宛先ポート番号を決定する。次に、IP パケットおよび Ethernet フレームを構築し、チップ上の Ethernet コントローラを介してチップ外へ出力する。

IP-NoC トランスレータでは、パケットフォーマットの変換に加えて、TCP のセッション管理、パケットシーケンスのチェック、再送、エラー訂正などの機能もソフトウェアで行う必要がある。

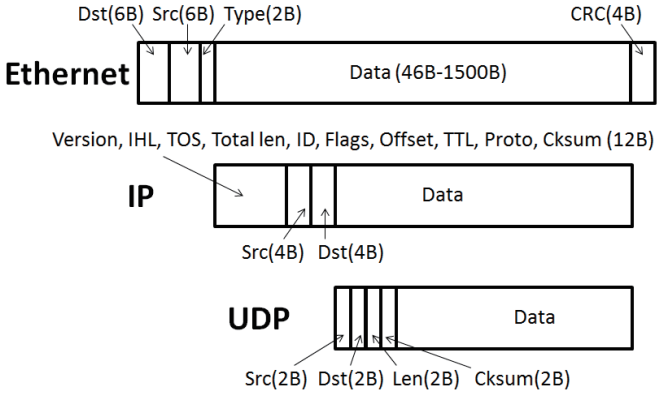

図 5: Ethernet、IP、UDP を含むパケット構造

パケットの変換手順 4 で述べたとおり、IP の宛先アドレスおよび TCP/UDP の宛先ポートに応じて、NoC 内の宛先ノードを決定することにした。つまり、単一の IP アドレスに対し複数の NoC のノードがマッピングされることがあり、この場合、ポート番号によって個々の NoC ノードを区別する。例えば、NoC 中のノード 1 は IP アドレス 192.168.1.10、ポート番号は 20 と 21（FTP）に対応している。一方、別のノード 2 は IP アドレスは 192.168.1.10 であるがポート番号は 80（HTTP）というように、ネットワークサービス（つまり、ポート番号）ごとに異なるプロセッサコアを割り当てることができる。このために、GW プロセッサはチップ内の NoC におけるノードアドレスと IP アドレスおよびポート番号の対応付けをテーブルとして持ち、アプリケーションから書き換え可能にしている。

インターネットで用いられるスイッチング法は Store and forward スイッチングであるのに対し、チップ内では Wormhole もしくは Virtual cut-through スイッチング法が用いられる。そこで、GW プロセッサにおいて、チップ外（チップ内）から来たパケットを一度受信し、スイッチング法を変えてチップ内（チップ外）に再度注入するようにした。このような eject-and-reinjection 操作には、構造デッドロックを回避するという目的もある。

単一チップ上に複数の IP-NoC トランスレータを搭載する場合は、チップ内のノード番号に応じて使用する IP-NoC トランスレータを静的に使い分けるようにした。動的な負荷分散については今後の課題である。

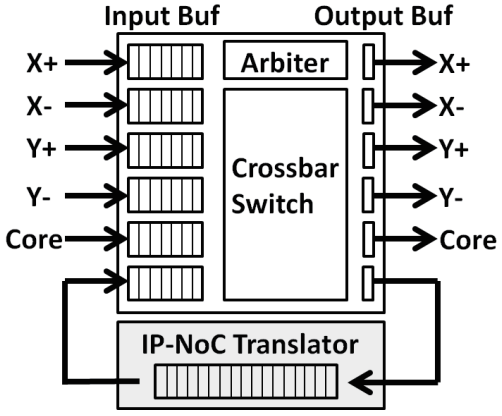

図 6: ハードウェア版 IP-NoC トランスレータ

上述の機能を持つソフトウェア版の IP-NoC トランスレータを C 言語のプログラムとして実装し、メニーコアチッ

プ上の GW プロセッサ上で動作させた。

4-2 ハードウェア版 IP-NoC トランスレータ

IP-NoC トランスレータによるプロトコル変換をソフトウェアとして実行すると、非常に多くのクロックサイクル数がかかる。チップマルチプロセッサにおいては通信遅延がアプリケーション性能に大きな影響を与えるため、IP-NoC トランスレータにおいても低遅延化が必須である。

ハードウェア版の IP-NoC トランスレータは、オンチップルータとプロセッサのインタフェース回路である Network Interface (NI) の機能として実装した。開発には Verilog HDL 言語を用い、Verilog HDL シミュレータを用いて動作確認をした。

図 6 にオンチップルータ回路とハードウェア版 IP-NoC トランスレータの接続を示す。ソフトウェア版と同じく、チップ外 (チップ内) から来たパケットを一度受信し、スイッチング法を変えてチップ内 (チップ外) に再度注入するようにしている。このためにオンチップルータの物理ポートを 1 つ増やしている (図 6 の一番下のポート)。IP-NoC トランスレータ専用ポートを使ってループバックさせることで、IP から NoC プロトコル(もしくは NoC プロトコルから IP)への変換を行う。ハードウェア版 IP-NoC トランスレータには Ethernet フレームを受信するために 1518 Byte のバッファを装備している。チップ内の NoC におけるノードアドレスと IP アドレスおよびポート番号の変換が必要であるため、ハードウェア版 IP-NoC トランスレータでは、このための変換テーブルを小規模メモリとして実装した。

IP-NoC トランスレータは、データリンク層、IP 層、トランスポート層の機能を内蔵する予定であるが、現時点では、トランスポート層は UDP にのみ対応している。TCP はセッション管理などが煩雑であり、ハードウェアによる処理には向かないと判断した。TCP への対応は今後の課題であるが、IP-NoC トランスレータでは対応せずプロセッサコアにおいてソフトウェア処理するのが適していると考えている。

本研究で設計実装した IP-NoC トランスレータは上述 2 種類のシミュレータ上で動作した。今後は Ethernet ポートを持った FPGA 評価ボード (図 7) に移植し、実機での動作検証をできるようにする予定である。

図 7: FPGA 評価ボードと Ethernet の接続

5 まとめと現状の課題

本研究では、インターネットプロトコルと NoC のプロトコル変換を行う IP-NoC トランスレータを実現した。これより、チップ内・チップ外を意識することなく IP および IP アドレスを用いてチップ内外の計算資源にアクセスすることが可能となる。IP-NoC トランスレータの実装として、プロトコル変換をすべてソフトウェアで行うソフトウェア版の IP-NoC トランスレータ、および、ヘッダの付け替えを専用回路で行うハードウェア版の IP-NoC トランスレータを実装した。

平成 25 年 6 月末現在、IP-NoC トランスレータ自体に関する研究成果について、主要な国際会議に論文が採択されて成果発表するには至っていない。実際には、すでにソフトウェア版およびハードウェア版の IP-NoC トランスレータの実装は出来ているので、現状の設計の完成度を高めるとともに、ソフトウェア版およびハードウェア版のさらに詳細な評価を行い、主要な国際会議に論文を投稿する予定である。国際会議での発表後は、国際会議論文の extended version を主要な論文誌へ投稿したい。

以下、本研究の今後の展開について述べる。文献 [Matsutani13a] において、我々は、メニーコアプロセッサのチップを 3 次元方向に複数枚積層することでより多くのプロセッサを単一パッケージ上に実装できるようにした。このような 3 次元メニーコアプロセッサに積層するチップの 1 枚として、本研究で実現した IP-NoC トランスレータを用いることも検討している。既存の 3 次元メニーコアプロセッサに IP-NoC トランスレータ機能を有したチップを 1 枚加えることで、3 次元パッケージ内のすべてのコアがインターネットへの接続性を持つことができるようになり、利便性が向上するものと考えられる。

また、文献 [Matsutani13b] において、我々は、クラウドコンピューティングにおけるデータベース処理を想定して、データベースの問合せ処理の高速化を行った。近年、ビッグデータの利活用のためにデータベ

ース処理の高速化が求められているため、本研究が対象としているクラウドコンピューティング向けのアプリケーションとして最適である。今後、本研究で開発した IP-NoC トランスレータの評価のために[Matsutani13b] で用いたデータベースの問合せ処理を用いる予定である。

半導体集積技術の観点からは、オンチップ・クラウドコンピューティングは十分に実現可能な技術である。今後は、チップ内のプロセッサを、インターネットを介して、チップ外の計算資源とどうつなぐか、また、オンチップクラウドを実現するプログラミングモデルや OS、ソフトウェア開発環境のほうへ研究がシフトしてくと考えられる。本研究で提案する IP-NoC トランスレータはこのための第一歩であり、IP を用いてチップ内の計算資源にアクセスできるようになれば、チップ内外の計算資源はその物理的位置に依らず 1 つの計算資源のプールとして仮想化され、負荷に応じて動的に分配、ユーザが利用できるようになる。

【参考文献】

[Shah07] M. Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Saha, D. Sheahan, L. Spracklen, A. Wynn, "UltraSPARC T2: A highly-threaded, power-efficient, SPARC SOC", Proc. of the IEEE Asian Solid-State Circuits Conference (A-SSCC'07), pp.22-25, Nov 2007.

[Wentzlaff07] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, Anant Agarwal, "On-Chip Interconnection Architecture of the Tile Processor", IEEE Micro, Vol.27, No.5, pp.15-31, Sep 2007.

[Dally01] William J. Dally, Brian Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", Proc. of the Design Automation Conference (DAC'01), pp.684-689, Jun 2001.

[Howard10] Jason Howard, et.al, "A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS", Proc. of the International Solid-State Circuits Conference (ISSCC'10), pp.108-109, Feb 2010.

[Jeffers13] James Jeffers, James Reinders, "Intel Xeon Phi Coprocessor High Performance Programming", Morgan Kaufmann, Mar 2013.

[Hennessy11] John L. Hennessy, David A. Patterson, "Computer Architecture: A Quantitative Approach (5th eds)", Morgan Kaufmann, Sept 2011.

[Matsutani11] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano, Tsutomu Yoshinaga, "Prediction Router: A Low-Latency On-Chip Router Architecture with Multiple Predictors", IEEE Transactions on Computers (TC), Vol.60, No.6, pp.783-799, Jun 2011.

[Binkert11] Nathan Binkert, et.al, "The gem5 simulator", ACM SIGARCH Computer Architecture News archive, Vol.39, No.2,pp.1-7, May 2011.

[hmc-mips] Harvey Mudd College and University of Adelaide's Implementation of the MIPS microprocessor, https://code.google.com/p/hmc-mips/

[Matsutani13a] Hiroki Matsutani, Paul Bogdan, Radu Marculescu, Yasuhiro Take, Daisuke Sasaki, Hao Zhang, Michihiro Koibuchi, Tadahiro Kuroda, Hideharu Amano, "A Case for Wireless 3D NoCs for CMPs", Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC'13), pp.23-28, Jan 2013. (Best Paper Award)

[Matsutani13b] 松谷 宏紀, "サービス指向ルータ向け問合せ処理用ハードウェアの検討", 情報処理学会研究報告 2013-EMB-28 (ETNET'13), No.29, Mar 2013.

〈発 表 資 料〉

題　　名	掲載誌・学会名等	発表年月
A Case for Wireless 3D NoCs for CMPs	Proc. of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13)	2013 年 1 月
サービス指向ルータ向け問合せ処理用ハードウェアの検討	情報処理学会研究報告 2013-EMB-28 No.29	2013 年 3 月