Vol.2013-ARC-206 No.24
2013/8/1

FPGAを用いたメニーコア評価基盤の構築とNoC評価

泊 久信[1,a)]　平木 敬[1,b)]

概要：メニーコアプロセッサの現実的な振る舞いを観測し、よりコア数の多いメニーコアの設計手法を検証するための評価基盤を構築した。プロセッサに実装されるコア数が大幅に増えたときに、合計命令スループットを最大限引き出せるようなプロセッサ機能およびチップ内のネットワークを設計する必要がある。同時に、それらのプロセッサを互いに接続するネットワークも増加したコア数に対応できるような設計である必要がある。シミュレーションベースの評価では、複数コアのエミュレーションを同期させる負荷が大きく、これを軽減させるとシミュレーション結果が不正確になるという問題があった。これにに加え、チップ内のネットワークが輻輳を起こすような状況では、シミュレーションによる性能予測が実機と更に離れてしまう。本研究では、メニーコアプロセッサのチップ内およびチップ外のネットワークの実証実験が可能なFPGA基板を設計し、その上で動作するSH-2命令セットのプロセッサを実装した。さらに、このプロセッサを用いチップ内ネットワークの実験を行い、いくつかのトポロジの資源見積りを行った。

Design of FPGA-based Many-core Evaluation Platform and NoC Evaluation

Hisanobu Tomari[1,a)]　Kei Hiraki[1,b)]

Abstract: We developed a platform for examining realistic behavior of many-core processor and verifying a design method that supports higher core count in a processor. Core functions and on-chip network design that can extract as much instruction throughput as possible is required. An interconnect between many-core processors also needs to scale to the higher core count. Evaluations based on simulated results are not always feasible for large number of cores. Synchronizations of emulated processor cores are one of the most time-consuming parts of the simulation, and accuracy of the simulation needs to be traded off for the simulation speed. In addition when the network congests the simulated performance is further inaccurate. On this paper we developed an FPGA board that we use to verify both on-chip and off-chip interconnects. We have implemented a processor with SH-2 compatible instruction set. Using the board and the processor, an on-chip network is evaluated, and resource usage for topologies of the on-chi network is measured.

1. はじめに

プロセッサ設計の観点では、マルチコア設計は、設計要素であるコアを複数並べることで設計と検証を実現可能な複雑さに収めつつ、製造技術の向上により利用可能になったゲートを有効に性能として利用可能にした技術である。現在、マルチコアプロセッサは10以下の非常に複雑なコアを用いたものから、数10の単純化したコアを実装したものまで存在する。少数の複雑なコアを並べたアプローチは、シングルスレッドで既存製品より高い性能が要求される携帯電話やパソコン向けのCPUの設計として使われている。多数の単純なコアを並べたものは、ネットワーク製品の組み込み向けのものとしてTilera TILE-Gx (36コア [5]), Cavium Octeon II (32コア) が、サーバー向けとしてOracle SPARC T4 (8コア [4]) が、高性能計算向けとし

[1] 東京大学
The University of Tokyo
[a)] tomari@is.s.u-tokyo.ac.jp
[b)] hiraki@is.s.u-tokyo.ac.jp

て Intel Xeon Phi (60 コア) が、研究用のものとして IBM Cyclops (64 コア [7]) が存在し、いずれもシングルスレッドの性能を多かれ少なかれ犠牲にしているものの、処理によっては複雑なコアを少数並べた場合より高スループットな場合がある、といったものである。

メニーコアプロセッサの設計の要は、コアの複雑さとチップ内でのコアの接続方式である。コアの複雑さを決めると、物理的な制約からチップ内に実装可能なコア数が決まり、チップの最大命令スループットはコアあたりの命令スループットをコア数倍したものになる。チップ内でのコアの接続法式は、リングを用いる方式、メッシュを用いる方式などいくつかのトポロジが提案・実装されている。ネットワークは、トポロジだけでなく、チップ内でメモリのキャッシュコヒーレンシを保証するか、ディレクトリの構成をどのようにするかといった選択肢もあり、これらによって最大命令スループットからの性能劣化をいかに小さくするかが課題である。現在のメニーコアプロセッサは、チップ内のコアはすべて同じメモリ空間を見せ、ソフトウェアからキャッシュコヒーレントに見せるためにハードウェア資源を利用している。また、マルチスレッドなソフトウェアも、チップ内のコアを結合する高速なネットワークを利用したキャッシュ間のデータの共有は積極的には行わず、共有データは外部メモリに書き出し、チップ内のネットワークで実際にコヒーレンシが問題になるのは同期の際が主である。

数千コアは現実的に予測されているほか [1]、半導体技術の限界はまだはっきりとしていないが、少なくとも 10nm を切るところまでは、半導体技術の進歩は継続し、チップのゲート数は増加し続けると予想される。その上で設計・検証を現実的な範囲に収めるために、チップ内のコア数が数千から数万を越えるメニーコアを動作させる必要がある可能性が高い。超メニーコアに最適なコアの複雑さ、現在のプロセッサコアには存在しない高速化機能の研究が必要である。チップ外のメモリやネットワークのバンド幅は、Memory Wall [6] としていわれているように、プロセッサの性能総計の向上に見合った向上は見込めないため、チップ内でのデータ共有の高速化が必要である。

従来の我々の超メニーコア研究は、シミュレータを用いて性能予測を行っていた [9]。シミュレータはシングルコアやコア数が小さい場合有効に動作するが、コア数が多い場合特有の困難が生じる。まず、性能の問題である。ホストプロセッサのコアを使ってそれ以上の数のターゲットコアをエミュレートするわけだが、一定の間隔ですべてのターゲットコアを同期させる必要がある。ターゲットプロセッサの命令実行だけをエミュレートする場合、ホストプロセッサのキャッシュが効きある程度の性能がでる。一方、多数のターゲットスレッドの同期処理はホストマシンでのシミュレータのワーキングセットサイズを増大させ、大幅に性能が低下する。同期間隔を粗くすることで性能は大幅に改善するが、この場合一部のターゲットコアだけ処理が先に進み、ターゲットコアで動くプログラムの挙動が変わってしまう場合がある。シミュレーションの性能を大幅に高速化させる他の手法として、トレースを用いることがある。トレースは、実機を用いるなどしてターゲットのプログラム実行させ、ターゲットで実行する命令アドレスやメモリアクセスを記録したもので、プロセッサのエミュレーションをすることなく、評価したい手法に必要なデータのみを用いて挙動を調べることができる。メニーコアの場合、複数のコアがデータを共有し同期処理を行う。データには依存関係があり、あるコアの処理が終わって同期処理が実行されるまで、もう一つのコアはあるデータを使って処理ができないことがある。以上の問題の他、複数スレッドプログラムの正確なトレースの取得が困難であるという問題もある。

チップ内ネットワークのシミュレーションも同様の問題があり、特に有限な帯域を持つパスのシミュレーションを正しく実装するのが困難である。輻輳が起きた場合の振る舞いも、シミュレーションで起きているような振る舞いが実際に起きるのか確証が持てないでいた。

以上の問題を解決し、超メニーコアプロセッサのコア、チップ内ネットワークおよびチップ外ネットワークで実証実験を行うため、我々は FPGA を用いたメニーコア実験基板を製作した。また、この基板で各種メニーコアを実装するための部品として、SH-2 互換命令セットを持つプロセッサを実装した。これらの要素を組み合わせチップ内ネットワークを構成し、このネットワークの基本性能を評価した。また、実際のハードウェアで実装した利点を生かし、各種ネットワークトポロジを実装した場合の資源量を測定した。本論文では、まず基板の設計について紹介し、次にプロセッサの設計とネットワークの概要、最後に評価結果を記す。

2. 基板

今回のメニーコア実装実験の対象がオンチップおよびオフチップのネットワークの実験のため、設計する基板に要求されるのは、十分なコア数を実装することができるよう、できるだけ大きな FPGA と、チップ外のネットワークのための入出力である。また、ネットワークで複数の基板を接続する実験も視野にいれているため、従来の基板で困難だった多数の基板をケーブルが届く範囲に安全に設置できる手法、および十分な FPGA の冷却が可能な設計が要求される。FPGA については、Xilinx の Virtex-6 HXT シリーズで規模が大きい XC6VHX565T を用いた。この規模の FPGA を実装した評価基板は種類が少なく、今回の我々の要求を満たすものがなかったため、基板を設計する必要があった。

構成の柔軟性のため、それぞれの基板に FPGA は 1 つ実装した。これらの基板を任意数接続するためのネットワークポートが必要になる。今回、Gigabit Ethernet では遅く、通常の使い方では外部の MAC が必要になることから、SATA に準じる方式を用いることにした。SATA は半二重で、Virtex-6 では 3 Gbps までのサポートになる。今回、トランシーバーに入力するクロックを SATA 標準の 150 MHz から 156 MHz に変更したため、SATA ケーブルの上のデータレートは 3.125 Gbps である。利用する FPGA と基板面積の制約により、24 ポートを実装した。SATA は 2 対の差動信号ペアで構成されていて、FPGA と SATA コネクタの間には AC カップリング用のコンデンサを実装するのみで済むほか、ケーブルが安価に入手可能である。このケーブルはすべてストレートケーブルのため、24 ポートの SATA コネクタは 12 本ずつ、ホスト側のピン配置のものとデバイス側のピン配置のものに分かれている。

他に、コンソール用の RS-232C ポート、起動イメージ転送用の Gigabit Ethernet、外部メモリとして 2 つの DDR3 SO-DIMM ソケット、およびデバッグ用の LED とスイッチ類を実装した。基板のブロック図を**図 1** に示す。

基板を SATA ケーブルの届く範囲で多数安全に実装する方法として、MicroATX ケースを用いることができるようにした。MicroATX ケースは小型の PC ケースとして安価に入手可能で、この箱に基板を実装するためには基板の大きさとネジ穴の位置を企画書の通りにする必要があった。冷却対策として、Intel の LGA1155 と同じ位置にヒートシンク固定用の穴を開け、PC 用のクーラーが利用する電源供給用のピンを実装した。LGA1155 のクーラーも、PC の CPU 用として様々な種類のものが安価に入手可能である。ただ、Intel の LGA1155 に CPU を実装した場合と、Virtex-6 を基板にハンダ付けした場合では基板からヒートシンクの距離が異なるため、高さの固定が容易に行えそうなデザインのクーラーを選択する必要がある。具

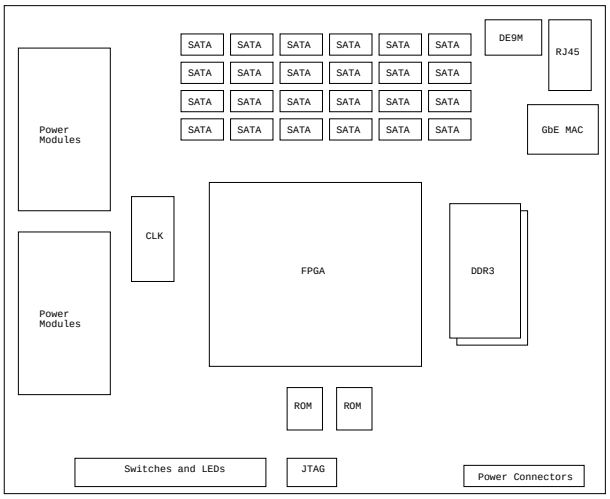

図 1 製作した基板のブロック図

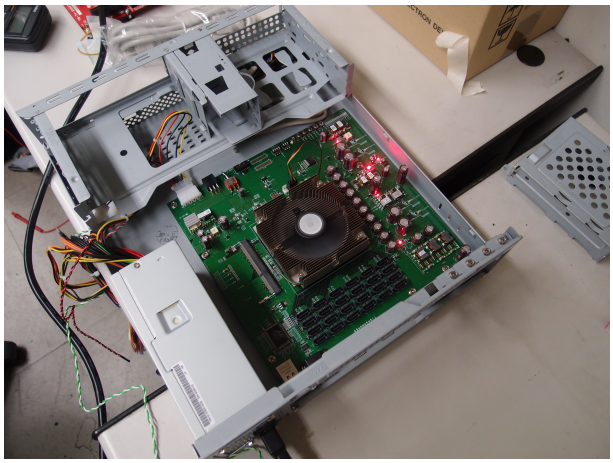

図 2 製作した基板の実装状況

体的には、プッシュピンではなくネジ固定式のもので、固定金具の形状が単純なものを選択した。製造した基板に LGA1155 クーラーを実装し、MicroATX ケースに固定した状態が**図 2** である。

3. コア

メニーコアの構成要素であるコアとして、SH-2 命令セット [2] を持ったプロセッサを VHDL を用い実装した。本プロセッサの特徴は以下の通り：

- SH-2 命令セット (MAC 命令はエミュレータで対応)
- 5 段パイプライン
- 254 レベルの割り込み、トラップに対応
- バスエラーからの復帰に対応 (仮想メモリサポート)
- 命令・データ分離キャッシュ合計 2KB
- キャッシュは Direct-mapped, 16 bytes/line
- ライン単位で Write-back/Write-through, 書込禁止、Cache 禁止指定可能
- バスコントローラは DMA/バスマスタ転送に対応

それぞれの特徴について以下で説明する。

我々は以前メニーコアのシミュレーションを 68000 と 8080, SH-2 命令セットを用い行ってきた [8]。68000 は当初プログラミングの容易さと開発ツールの成熟度、および機能の豊富さからシミュレーションに用いていたが、ハードウェアとして実装する際に FPGA では回路規模が大きくなりすぎる問題点が後から判明した。そのため、大幅に単純化しコア数を増やした場合のメニーコアの振る舞いを調査する意味で 8080 を、パイプライン化しコアあたりの処理性能を向上させたプロセッサとして SH-2 を用いてシミュレーションを行い、8080 については既にハードウェア実装を行った。SH-2 は命令が 16 ビット固定長であり、68000 と命令形式、ニモニックや一部命令のビット表現が似ている点などから移行先として選択した。SH-2 は GCC/Binutils で対応しているため、アセンブラ・コンパイラを新規に開発する必要がない。SH-2 の命令は 2 オペ

ランドで、大半の命令はパイプラインで実装するのに適したレジスタアクセスおよびメモリアクセスを行うが、メモリロード命令はメモリアクセスを 2 回行い積和算を行い、2 つのポインタをインクリメントするという複雑な制御が必要で、後述するバスエラーから復帰可能にする処理が複雑になるため、ハードウェアでの命令サポートをしていない。ただし、無効命令トラップハンドラにエミュレータを記述することで、MAC 命令を含む命令列に変更を加えることなく実行することは可能である。なお、MAC 命令は GCC 4.8.0 を用いて Dhrystone V2.1 をコンパイルした場合、オブジェクト中には出現しない。

パイプラインは標準的な 5 段のパイプラインを実装した。SH-2 は命令デコードが重く、命令デコードの結果バブルを発生させ、命令キャッシュからの命令ストリームを一時停止するパスがプロセッサ内でのクリティカルパスになっている。ポストインクリメントのロード命令はレジスタ書き込みが 2 回発生するため、2 サイクルで実行する。レジスタファイルは 2R1W の構成にすることで、命令間のデータ依存性の解析回路を小さな規模を保っている。

割り込み・トラップについては、SH-2 の割り込み処理では、今後の研究で利用するバスエラー後のバスエラーを起こした命令の再実行が不可能だったため、SH-2 の割り込み処理をベースにしつつ、これらの要求を実現するための機能を実装した。割り込みベクタ表の先頭 2 ロングワードがリセット時のプログラムカウンタ (PC), スタックポインタに割り当てられているため、これら 2 つを除いた 254 レベルを発生させることができる。なお、ベクタ・ベース・レジスタ (VBR) を利用して割り込みベクタを移動することが可能で、この場合先頭 2 レベルも含めた 256 レベルの割り込みが利用可能になる。先頭 14 レベルは外部割り込みで利用可能なほか、無効命令、バスエラーなどもの領域に割り当てられている。トラップはこれ以降の割り込みレベルを利用することになる。

外部割り込み、トラップが発生した場合、実行中の命令が終了次第割り込み処理に移る。これらの要因の場合、実行中の次の命令から実行を再開すればよいため、スタックには次の命令のアドレスとステータスレジスタ (SR) を書き込み、割り込みハンドラ終了時は rte 命令によりスタックに書き込まれた PC から実行を再開する。

命令アクセスまたはデータアクセスでバスエラーが発生した場合、スタックにはバスエラーを発生させたアドレス、バスエラーを発生させた命令のアドレス、SR を書き込む。rte 命令が実行されるまでに OS によりページが補充され、バスエラーを発生させた命令から再実行されることで、仮想メモリやその他のページングを使った高速化手法が実装可能になる。

SH-2 本来の割り込み処理では、スタックには常に実行を中断した命令の PC が記録され、rte 命令ではその次の命令から実行を再開する。積和算命令のように 2 回メモリアクセスを行い、レジスタも 2 つのポインタと演算結果を更新するような命令は、命令の再実行の点で困難を生じる。命令を実行途中から再実行するには、MC68030 のように例外スタックにマイクロコードのステータスもダンプするなどの工夫が必要になる。また、バスエラーを起こしたアドレスが割り込みスタックに残らないため、MMU への状況問い合わせが困難である。今回は MMU は実装していないが、以上のようなコアのページング対応機能および後述するキャッシュの制御信号のサポートにより、MMU の接続が可能になっている。

命令・データキャッシュはそれぞれ 1 KB ずつ、16 bytes/line の direct mapped で write-back 構成になっている。SH-2 の命令セットは即値フィールドが小さく、32-bit long word のロードは PC 相対でポインタを指定しロードすることが多い。この時指定される即値は、PC 相対で指定可能なオフセットの大きさの都合により、その命令が所属する関数の前後に書き込まれているコードをコンパイラは出力する。命令キャッシュのラインサイズを大きくすると、このようなデータキャッシュにより取得されるバイト列が命令キャッシュにより多く存在することになり、命令キャッシュの利用効率が低下する。オリジナルの SH-2 ではこのような問題からか命令・データキャッシュは統合されている。今回のデザインでは、分離キャッシュにすることでレイテンシを削減し、命令実行効率を高めている。キャッシュラインサイズも 16 bytes/line と小さいため、命令およびデータキャッシュの利用効率を著しく低下させることはない。キャッシュから外部のメモリへのアクセスは、外部に 8 バイトバスを用いる SO-DIMM が存在することから、8-byte のバスを用いて行っている。

データキャッシュはページングおよびメモリ写像方式のオンチップネットワークに対応するため、各ラインにライトスルー、キャッシュ禁止、書込禁止、バスエラーのステータスを記録する。これらのステータスは外部メモリからのデータ受信時に同時に受け取る。命令キャッシュは書き込みはないため、バスエラーのステータスのみ存在する。これらのステータスは主にメモリ空間に配置されたデバイスを扱うために存在する。書込禁止及びバスエラーはページングや類似の機能を使った高速化の実装で使うために存在する。バスエラーが起きた場合でも、バスエラーが起きたアドレスにプロセッサがアクセスするまで割り込み処理を開始しないためにバスエラーはキャッシュされる。書込禁止はライトバックキャッシュで書込保護を実現するために必要なステータスである。これらのコアとキャッシュの機能が存在するため、例えばソフトウェア・トランザクショナル・メモリを実装することが可能になる。

DMA/バスマスタ機能は、外部からの要求に応じてバスを明け渡す機能で、今回の実験ではネットワークインター

フェースがバスマスタとしてローカルメモリとリモートメモリ間で転送を実行する。この転送の最中でも、CPU はキャッシュの命令・データを利用して処理を進めることができるようになっている。また、ネットワークで接続された他の CPU から自分のローカルメモリへのアクセスがあった場合でも割り込み等で処理を中断する必要がなくなり、全体の性能向上に寄与している。

4. チップ内ネットワーク

チップ内ネットワークは、コアとそれぞれのコアに対応したローカルメモリ、ネットワークロジックをあわせた単位 PE を結合する。ローカルメモリは命令・データキャッシュより下層に存在し、PE 内のコアおよびネットワークを利用して PE 外からもアクセス可能になっている。今回の研究では、ローカルメモリはコアあたり 4 KB を実装した。PE 同士を結合するネットワークは、以前の研究で提案した方式を改良し、あるコアは直接接続されたコアのメモリの一部を読み書きできるリフレクティブ・メモリ方式 [3] を利用した。メモリを共有するのは各ネットワークパスの両端につながる 2 つの PE のみなので、直接接続されていない PE にデータをやりとりするためには経路上の PE すべてでソフトウェアを用いてメモリブロックをリレーする必要がある。

各 PE からは 4 チャネルのネットワークバスを出し、次数 4 までのネットワークで実験可能にした。それぞれのチャネルは CPU クロックに同期した動作をし、双方向でアドレス 9 ビット、データ幅 64 ビットの通信が可能になっている。このアドレスとデータ幅により、リモートのメモリのウインドウは 4 KB となる。今回の実装ではこれは各 PE のローカルメモリのサイズと一致し、ローカルメモリ全域がチップ内ネットワークに公開されることになる。ローカルメモリはハードウェア資源の節約の目的でシングルポートを利用しているため、4 方向のリモートからのアクセスと CPU からのアクセスは排他的に行う必要がある。今回の実装は CPU が最優先でバスを所有し、それ以外はポート番号順に優先度が設定されているが、高負荷時に処理がデッドロックしないようにするためには 4 つのネットワークが公平にローカルメモリを占有する仕組みが必要である。

シミュレータのみで実験していた場合に問題にならなかったが、実機で実験する場合は初期化の方法を適切に作る必要がある。外部からデータを読み込める PE は限定されるため、ロードしたプログラムを全 PE に分配し、そのプログラムをそれぞれの PE で実行開始する必要がある。そのため、このネットワークのチャネルには割り込みを発生させる信号を追加してあり、隣接する PE からローカルメモリにプログラムを書き込んでもらい、終了したら割り込みをかけて実行する方式を採用した。外部との通信

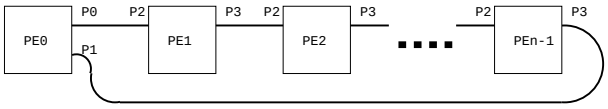

図 3 レイテンシ実験で用いた接続

のため、PE 番号 0 の CPU に MC6850 (ACIA) と同機能の RS-232C コントローラと、Gigabit Ethernet インターフェースが接続されている。また、初期化用に PE 0 にはモニタが書き込まれている ROM を接続した。これらの機能を利用することで、実機で任意のプログラムを動的に読み込むことが可能である。

5. 評価

5.1 ネットワークのレイテンシ

構成したネットワークのレイテンシを測定するため、複数個の PE をリングで接続し、リングを 1 週してデータが戻ってくるまでの時間を測定した。使用したネットワークの接続は**図 3** に示す。通信プログラムはアセンブリで記述し、CPU とネットワークの動作クロックは 50 MHz (20 ns) として測定を行った。リングに接続される PE 数を変化させ、4 バイトのデータの送信処理を開始してからデータを受信するまでの時間をプロットしたのが**図 4** である。

平均すると 580 ns/hop であり、これは現在のメニーコアに比べ大変遅い結果となっているが、これは CPU およびネットワークの動作クロックが 50 MHz と通常のプロセッサに比較し遅いからである。現状より動作速度を改善できたとしても、FPGA では動作周波数をある一定以上上げるのは困難である。サイクル時間とネットワークの遅延時間で比較すると、現在のメニーコアで遅延が短い Tilera TILE-Gx36 が 1,200 MHz のクロックでチップ上の 36 こあどこでも 70ns 前後の遅延となっているため、キャッシュコヒーレントな共有メモリをすべてのコアで実現する場合に比べ、隣接するコアとの通信以外は却って遅くなっている。また、パス毎に数十 ns の遅延のばらつきがあるのは、

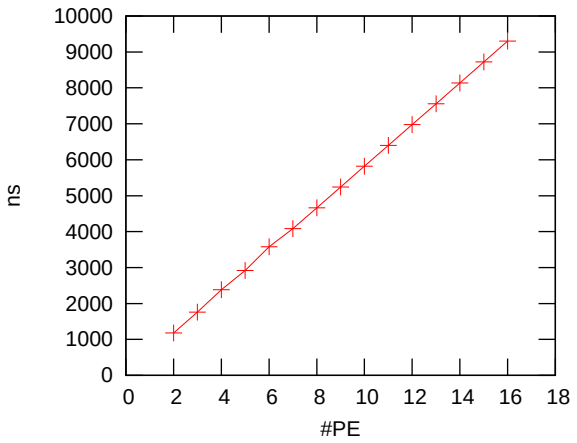

図 4 リングを 1 週するのにかかる時間

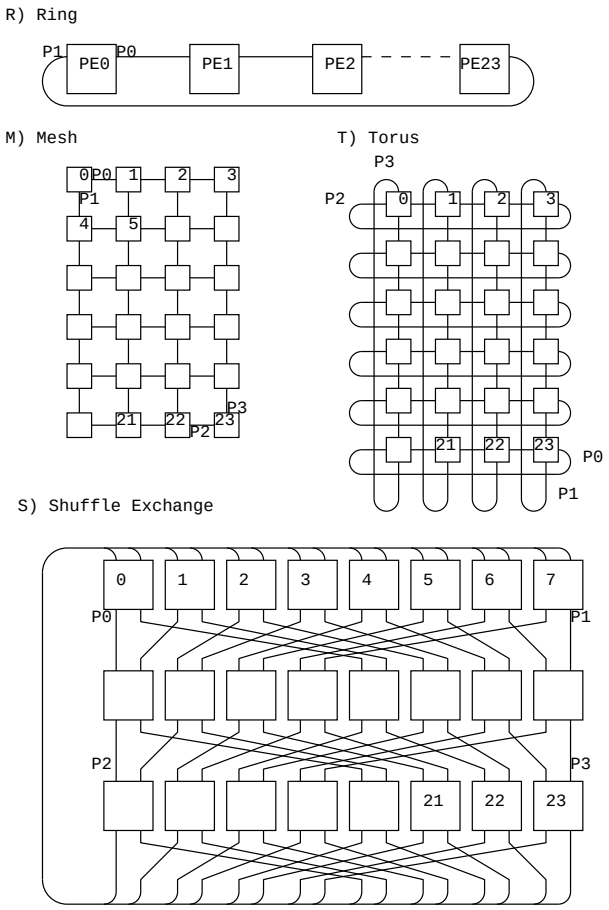

図 5 ハードウェア資源量測定で用いた接続方式

表 1 各トポロジの FPGA 占有スライス数

Ring	Mesh	Torus	Shuffle Exchange
53,724	55,358	58,576	55,721

リモートのメモリを取得するタイミングとそのメモリがリモートの CPU で取得されるタイミングの重なり方、および同期処理が行われ始める際の CPU の状態による。

5.2 各種トポロジでのハードウェア資源量

チップ内ネットワークのトポロジを変化させた際のハードウェア資源の使用量を測定するため、作成した基板で各種トポロジの 24 コアのシステムを実装し、FPGA の使用スライス数を比較した。**図 5** が利用したトポロジの一覧である。図中に PE から出る 4 本のネットワークバスのうち何番目をどの方向に使用したか、ポート番号も併記した。Xilinx ISE 14.4 を利用した結果の Number of occupied slices の値を**表 1** に示す。利用した XC6VHX565T の最大スライス数は 88,560 なので、スライス利用率はいずれのネットワークの場合も 6 割前後である。

Shuffle Exchange で接続した場合でも、Torus で接続した場合に比べ資源使用率は低く、Mesh と同じレベルの資源で実現可能なことが分かる。Ring は資源使用率は最低になるものの、遅延時間を埋め合わせるほどの差はない。一方、歩留まり向上のため製造不良のコアを簡単に無効化できるのはこれらのネットワークの中では Ring のみである。

6. おわりに

シミュレータでは再現が困難な、メニーコアプロセッサの正確な振る舞いを調べるため、評価基盤として大規模な FPGA を用いた基板およびその上で動作するメニーコアプロセッサを実装した。設計したチップ内ネットワークの性能を現在のメニーコア製品と比較した結果、ソフトウェアでデータをリレーする本研究の現在の実装が性能ペナルティとなることが判明した。また、4 種類のチップ内ネットワークのトポロジを FPGA で実装し、それぞれの資源使用率を比較することで、ネットワークの複雑さが具体的にどれだけのハードウェア資源の増加に寄与するか定量的に評価した。Shuffle Exchange をチップ内ネットワークで利用しても、Mesh と同じレベルの資源利用率のなることがわかった。今後、チップ内ネットワークのルーティングをハードウェアで実装し、チップ間を結合するネットワークと協調動作させる必要がある。

参考文献

[1] Borkar, S.: Thousand core chips: a technology perspective, *DAC '07: Proceedings of the 44th annual Design Automation Conference*, New York, NY, USA, ACM, pp. 746–749 (online), DOI: http://doi.acm.org/10.1145/1278480.1278667 (2007).

[2] Hitachi America Ltd: SuperH RISC Engine SH-1/SH-2 Programming Manual (1996).

[3] Lucci, S., Gertner, I., Gupta, A. and Hegde, U.: Reflective-memory multiprocessor, *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, Vol. 1, pp. 85 –94 vol.1 (online), DOI: 10.1109/HICSS.1995.375406 (1995).

[4] Shah, M., Golla, R., Grohoski, G., Jordan, P., Barreh, J., Brooks, J., Greenberg, M., Levinsky, G., Luttrell, M., Olson, C., Samoail, Z., Smittle, M. and Ziaja, T.: Sparc T4: A Dynamically Threaded Server-on-a-Chip, *Micro, IEEE*, Vol. 32, No. 2, pp. 8–19 (online), DOI: 10.1109/MM.2012.1 (2012).

[5] Tilera Corporation: Tile Processor Architecture Overview for the TILE-Gx Series, No. UG130 (2012).

[6] Wulf, W. A. and McKee, S. A.: Hitting the memory wall: implications of the obvious, *SIGARCH Comput. Archit. News*, Vol. 23, No. 1, pp. 20–24 (online), DOI: http://doi.acm.org/10.1145/216585.216588 (1995).

[7] Zhang, Y. P., Jeong, T., Chen, F., Wu, H., Nitzsche, R. and Gao, G.: A study of the on-chip interconnection network for the IBM Cyclops64 multi-core architecture, *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p. 10 pp. (online), DOI: 10.1109/IPDPS.2006.1639301 (2006).

[8] 泊 久信, 平木 敬：コヒーレントでないメモリシステムへのアーキテクチャ支援, Vol. 研究報告 計算機アーキテクチャ（ARC） 2010-ARC-190, No. 3 (2010).

[9] 泊 久信, 平木 敬：1600 万計算コア超メニーコアアーキテクチャのシミュレーション, Vol. 研究報告 計算機アーキテクチャ（ARC） 2012-ARC-201, No. 6 (2012).