Vol.2012-ARC-199 No.1
2012/3/27

メニーコアプロセッサを対象とした柔軟性を有するハードウェアバリア機構の提案

曽我 武史[†1] 近藤 正章[†2] 佐々木 広[†1]
平尾 智也[†1] 井上 弘士[†1]

チップ上に多数のプロセッサコアを搭載するメニーコアプロセッサの使い方として、複数のコアで数個をグループを構成し、それぞれに異なるアプリケーションやタスクを実行させるマルチスレッド・マルチプログラム実行法がある。複数コアで一つの処理を行う場合にはコア間で同期を取る必要がある。一般に、同期処理はソフトウェアで実装されるが、コア数の増加に伴い、ソフトウェア同期に要する時間や、同期完了時刻のばらつきが大きくなるという問題が生じる。本研究では、ハードウェアバリアを用いることによって、これらの問題を解決する。また、バリアに参加するコアを選択可能にする回路を追加することで、コアグループ間での独立した同期処理を実現すると共に、複数グループに対して同時に異なるバリア同期を実施することを可能にしている。実験により、ソフトウェアバリアに対して 66 倍のレイテンシ改善を実現した。また、ハードウェア記述言語を用いた設計を行った結果、実験に使用したメニーコアプロセッサモデルに対して最大 1.8%の回路規模の増加で実現できることが分かった。

1. はじめに

近年の、プロセス技術やハードウェア技術、ソフトウェア技術等の進歩により、チップ上に複数のプロセッサコアを搭載したマルチコアプロセッサが広く普及するに至った。また、より多くのプロセッサコアを搭載した、メニーコアプロセッサについても研究が及んでおり、例えば 100 個のプロセッサコアを搭載した商用化チップも存在する[1)]。メニーコアプロセッサにおいて高い実効性能を実現するためには、搭載したコアの数に見合っただけの並列性を活用できなければならない。例えば、画像処理や科学技術計算のように、極めて高い並列性が内在する場合においてはコアの稼働率を高めることができる。しかしながら、十分な並列性が抽出できない場合が多く存在するのも事実である。したがって、メニーコアプロセッサの応用範囲を拡大しその普及を図るためには、プログラム内並列性のみならず、プログラム間並列性をも積極的に活用しなければならない。すなわち、メニーコアプロセッサにおいては、効率的なマルチスレッド・マルチプログラム実行環境の実現が極めて重要となる。具体的には、OS、メール、ゲーム、コンパイラなどの複数ソフトウェアが同時に動作する組込みシステムや、複数ユーザが異なるジョブを投入するデータセンター・サーバ応用が考えられる。

並列化された複数のプログラムを同時に実行する場合の課題として、プロセッサコア間での同期処理の高効率化が挙げられる。同期はプロセッサ間の歩調を合わせるために必要であり、例えば、複数プロセッサコアがメモリやバスなどの共通資源を使用する際にアクセス順序を決めるときや、複数プロセッサコアが同時に動作を開始したい場合などに実行される。ここで、ある同期に参加するプロセッサコアの集合を**コアグループ**と呼ぶ。一般に、並列化プログラムの実行に使用するプロセッサコアの数は、当該プログラムに内在する並列性の度合いに大きく依存する。これに加え、複数のプログラムが同時に実行される状況においては、複数のコアグループそれぞれが同期処理を発行する場合もある。したがって、メニーコアプロセッサにおいては、複数の様々なコアグループに対して独立かつ並列に実行可能な同期機構の実現が必要となる。

同期手法の代表的なものの一つとしてバリア同期がある。これは、同期

†1 九州大学大学院システム情報科学研究院
†2 電気通信大学大学院情報システム学研究科

の対象となる各プロセッサコアが実行するプログラム中に、対象コア全てが許可を出すまで停止する指示を埋め込むことにより同期を実現する手法であり、実際に MPI、OpenMP、pthread 等の多くの並列コンピューティング環境に実装されている。一般に、バリア同期は専用ハードウェアの追加なしに実現可能ではあるが、数多くのプロセッサコアが参加する場合には、(1) バリア同期に要する時間が大きくなり頻繁に使用するとソフトウェア実行性能の低下要因となる。(2) 各プロセッサコアのバリア処理完了時刻に大きくばらつきが発生して、事前のチューニングが期待通りに反映されなくなる、などの問題が発生する。そのため、HPC 分野で使用されるような大規模システムにおいてはハードウェアバリア機構が実装されていることが多い。大規模システムでのハードウェアバリア機構は、筐体間やチップ間を繋ぐ長距離ネットワークを制御する多機能なネットワークコントローラの存在を前提として設計されているため回路規模が大きく、そのままメニーコアプロセッサのハードウェアバリアとして使用することは難しい。また、大規模システムでは配線コスト低減を目的としてハードウェアバリアで使用するネットワークを他の通信機能と共有しているため、常に最短レイテンシでのバリア同期実行は期待できない。

そこで、本研究では、メニーコアプロセッサでの利用を前提とし、柔軟性を有する新しいハードウェアバリア方式を提案する。具体的には、(1) 低レイテンシでばらつきが小さく、(2) バリアネットワークの分割を可能にし、かつ、(3) 小規模な回路量で実現できるハードウェアバリア機構を考案する。また、この小規模なバリアネットワークを複数用意することで、多様なプロセスが動作するであろうメニーコアプロセッサにおいても対応可能な柔軟なハードウェアバリア機構を実現する。

本稿の構成は以下の通りである。第 2 節では提案するハードウェアバリア機構について説明し、第 3 節ではハードウェアバリアの効果を実験結果によって検証する。第 4 節では関連研究を示し、第 5 節において本稿をまとめる。

2. 柔軟性を有するハードウェアバリア機構

2.1 ハードウェアバリア機構

本節では、ある固定的なコアグループを対象とし、提案するハードウェアバリア機構（以下、バリア機構と略す）の基本構造と動作を説明する。なお、様々なプロセッサコア（以下、コアと略す）で構成されるコアグループへの対応、ならびに、複数コアグループでの並列同期処理の実現に関しては次節以降で詳細を述べる。本バリア機構の実現には以下のハードウェア要素が必要となる。

- **バリアネットワーク**：バリア専用の双方向二分木ネットワーク。4 コアでコアグループを構成した場合の例を**図 3** に示す。
- **葉ノード**：各コアに追加されるハードウェア・モジュール。以下に示す 2 種類の 1 ビット・レジスタを有する。
  - **同期設定レジスタ**：当該コアがバリア同期の完了待ち状態にあるか否かを指示するレジスタ。当該コアのプログラム実行が同期ポイントに到達したら'1' にセットされる。また、次に説明する同期状態表示レジスタがリセットされると'0' にリセットされる。
  - **同期状態表示レジスタ**：根ノードが管理するバリア機構状態レジスタの複製を保持するレジスタ。
- **根ノード**：コアグループにおける同期処理実行状態を管理するハードウェア・モジュール。子ノードとなる全ての葉ノードが管理対象コアグループとなる（ただし、次節で説明するマスク機能を用いた場合を除く）。図 3 の例では、根ノードが管理するコアグループは葉ノード 0~ 3 である。管理下にある葉ノードから同期設定レジスタ値を集計し、全てのコアが同期ポイントに到達したか否かを判定する。内部には 1 ビットの**同期状態レジスタ**を有する。提案するバリア機構には以下に示す 2 つの状態があり、同期状態レジスタにより現状態を表す。状態遷移を**図 2** に示す。
  - **バリア完了待ち／受付可能状態**：コアグループに属するコアがあ

る同期処理を実行中、もしくは、次のバリア同期処理を実行可能な状態（図 2W_SET）。このとき、同期状態レジスタは'0' にリセットされる。

– **バリア完了通知待ち状態**：バリア同期が完了し、全葉ノードに対して完了通知をブロードキャストしている状態（図 2W_RST）。このとき、同期状態レジスタは'1' にセットされる。

- **内部ノード**：根ノードと葉ノードの中間に位置するハードウェア・モジュール。内部構造は根ノードと同じであり、バリアネットワークを分割して複数コアグループの同期処理を実現する場合には根ノードとして動作する。

以下、**図 1** を用いてバリア機構の動作を説明する。なお、バリア機構の状態は「バリア完了待ち／受付可能状態」であり、全ての葉ノードの同期設定レジスタの初期値は'0' とする（つまり、次のバリアを実行可能な状態）（図 1（a））。

（1） 同期処理において、コア 2 が同期ポイントに到達する。この時、当該コアの同期設定レジスタが'1' に設定される。

（2） コア 2 の同期設定レジスタの値が内部ノード 1 に送信される。この時、他方の子ノードの同期設定レジスタの値がセットされていなければ、同期待ち状態を継続する（図 1（b））。

（3） コア 3 が同期ポイントに到達し、同期設定レジスタが'1' に設定される。これにより、内部ノード 1 は親ノード（ここでは根ノード）に対して同期設定レジスタの値を伝搬させる（図 1（c））。

（4） コア 0 ならびにコア 1 が同期ポイントに到達すると、根ノードは全てのコアが同期ポイントに到達したと判定する。これにより、各葉ノードに対して同期処理完了通知の信号を送り、バリア完了通知待ち状態に遷移する（図 1（d））。

（5） 各葉ノードは、根ノードからバリア完了通知を受け取った後、同期設定レジスタを'0' にリセットし、これが根ノードへと伝搬する（図 1（e））。

（6） 根ノードは、全ての葉ノードの同期設定レジスタの値がリセットされた事を判定し、バリア機構の状態をバリア完了待ち／受付可能状態にする。これにより、次の新たなバリア同期を受け付けることが可能となる（図 1（a））。

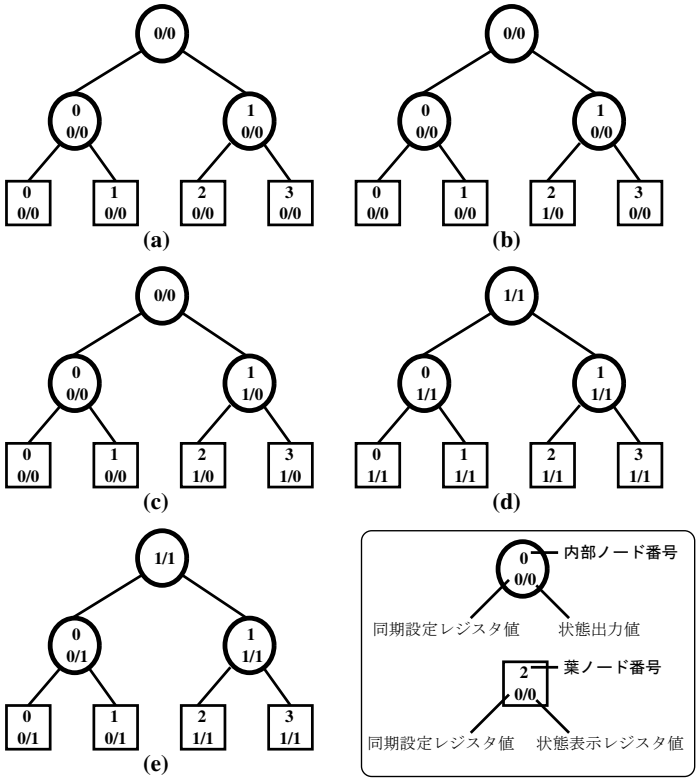

図 1 バリア機構の動作

内部ノードは、バリア完了待ち／受付可能状態の場合には、2 つの子

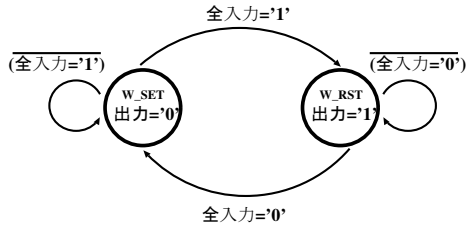

図 2 バリア同期の状態遷移

ノードから入力した同期設定レジスタの値の論理積をとる。一方、バリア完了通知待ち状態の場合には、これらの値の論理和を親ノードに回送する。これらの値が根に向かって伝搬することにより、根ノードは全ての葉ノードの状況を把握することができる（**図 4**）。

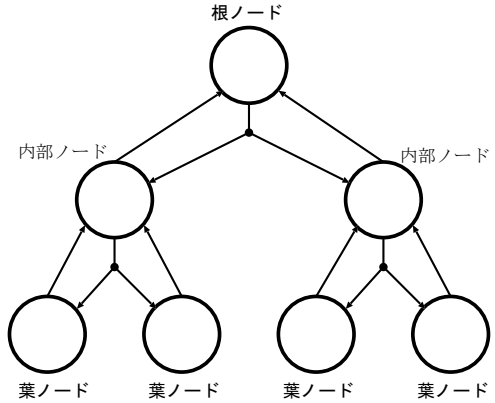

図 3 4 ノード（=4 コア）時のバリアネットワーク

2.2 柔軟なハードウェアバリア処理の実現

前節では、根ノードに対応する全ての葉ノードがコアグループに含まれる場合を想定していた。しかしながら、第 1 節で述べたように、マルチ

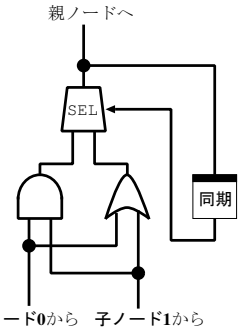

図 4 内部ノードにおける同期処理

プログラム実行環境下においては、複数の様々なコアグループに対して独立かつ並列に実行可能な同期処理をサポートしなければならない。そこで提案方式においては、接続設定情報の変更によりバリアネットワークを論理的に分割し、一つのバリアネットワーク内で複数のバリア同期を実行できるように拡張する。具体的には、以下に示す 2 つの機能により、柔軟なハードウェアバリアが可能となる。

- **内部ノードにおける折返し機能のサポート**：内部ノードは根ノードと同じハードウェア構成を採る。したがって、内部ノードは根ノードと同一の機能を有することが可能である。そこで、ある内部ノードを根ノードとして動作させることでバリアネットワークを複数の部分木に分割する。**図 7** は、8 個の葉ノードを有するバリアネットワークを 4 分割した例である。例えば、葉ノード 2, 3 はコアグループ B を構成している。この場合、親ノードとなる内部ノードに対して折返し機能を適用することで根ノードとして動作させる。内部ノードに対して折返し機能が適用されると、**図 5** に示すように、子ノードから回送された信号（同期設定レジスタの値）の論理演算結果を親ノードには回送せず、同期処理完了通知信号として子ノードに転送する。例えば、両方の子ノードが同期ポイントに到達した場合には対応する同期設定レ

ジスタが‘1’にセットされ、これらの論理積がとられる。この結果が、当該子ノードに対して同期処理完了通知信号としてフィードバックされる。また、親ノードに対しては親ノードから回送されてきた同期処理完了通知信号の反転値を転送することで、上位（根ノードに近い方）のノードに対する影響を排除する。このように部分木を用いて小規模なバリアネットワークを複数構成することにより、各コアグループは独立かつ並列にバリア同期処理を行うことができるようになる。

- **葉ノードの切り離し（マスク）機能のサポート**：葉ノードに対して適用される設定であり、葉ノードに対応するコアがバリアに参加するか否かを指定する。バリア機構内部では、マスクが有効であれば葉ノードの同期設定レジスタの値が親ノードに送信される。一方、無効な場合には、状態表示レジスタの反転値を親ノードに対して出力することで、上位のノードに対する影響を排除する（**図 6**）。

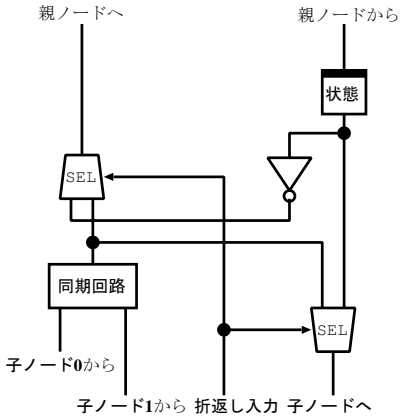

図 5 折返し機能

2.3 多重化による柔軟性の拡大

マスク・折返しによる設定では部分木単位のグループ構成以外は行えな

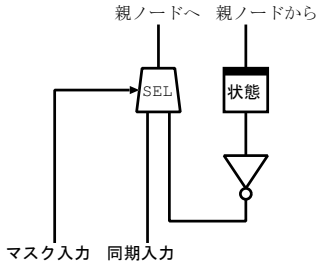

図 6 マスク機能

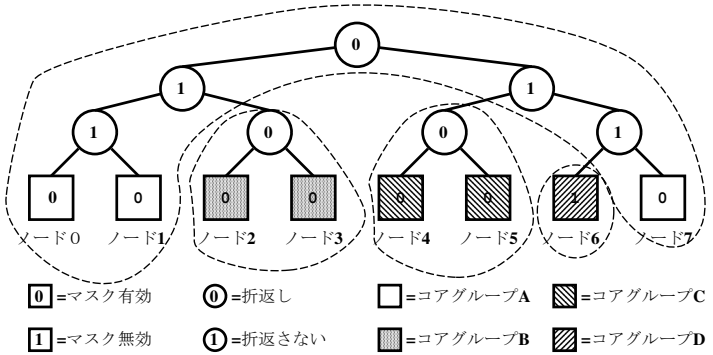

図 7 折返し・マスクを利用したグループ分割

いため、それ以外のグループ構成には、バリアネットワークを複数用意すること（多重化）により対応する。**図 8** は、8 ノードからなるバリアネットワークを 2 つを使用して、ノード番号 {0,2,4,6}、{1,3,5,7} で組み分けされた、2 つのグループに分割した例となっている。

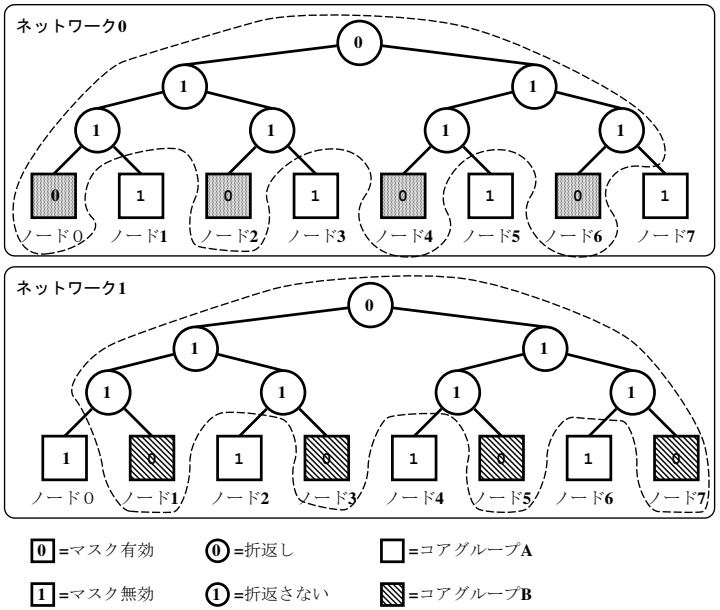

図 8 多重化したネットワークによるグループ分割

実際には、前節で説明した折返し機能、マスク機能、ならびに、多重化を組み合わせることにより、本バリア機構は少レイテンシでばらつきのないハードウェアバリアを、様々なノードの組み合わせで実行することを可能にしている。

3. 評　　価

本節では、バリア機構の評価をバリア同期時間の測定および回路規模の計数によって行う。本評価では、メニーコアアーキテクチャ評価環境である SMYLEref[2] の RTL 設計を使用した。SMYLEref は、MIPS R3000 アーキテクチャをベースとした複数のプロセッサコアをバス結合でクラスタ構成し、多数のクラスタを 2 次元メッシュのオンチップネットワークで結合したアーキテクチャとなっている。本実験では 1 クラスタ、16 コアのモデルを作成し、16 個のコアに対して 8 つのバリアネットワークを実装した。また、バリア制御のために各コア内にレジスタを実装し、各コアがレジスタを読み書きすることによって、バリア同期を実施可能にした。レジスタは、自コアが使用する葉ノード及び、15 ある内部ノードと根ノードの内一つの構成および同期を制御する（**図 9**）。

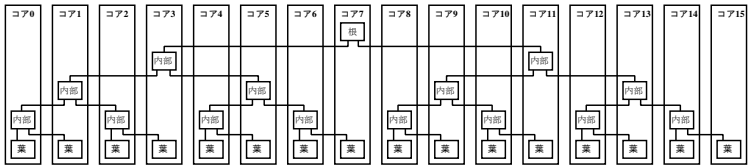

図 9 ノードの構成変更を行うコアの割り当て

3.1 バリア同期時間測定

バリア同期時間を測定するため、**図 3.1** に示す評価用プログラムを用いた。本評価用プログラムは、最初に各コアが担当する、葉ノード、内部ノードのマスク情報および、折返し情報をレジスタに書込む（barrier_config 関数）。次にバリア同期の実行（barrier 関数）を 2 回繰り返す。バリア関数を 2 回実行するのは、関数のプログラムをキャッシュに載せることと、全てのコアが同時に計測対象のバリア関数実行を開始するためである。

barrier 関数内では以下の処理を行う。

```
void main(void)
{
 barrier_config();
 barrier();
 barrier();
 exit;
}

void barrier(void)
{
 check_barrier_reset();
 barrier_sync_set();
 check_barrier_set();
 barrier_sync_reset();
 return;
}
```

図 10 ハードウェアバリアテストプログラムの構成

(1) **check_barrier_reset:** 状態表示が'0' になるまでレジスタを読込を繰り返す。
(2) **barrier_sync_set:** 同期設定='1' をレジスタに書込む。
(3) **check_barrier_set:** 状態表示が'1' になるまでレジスタを読込を繰り返す。
(4) **barrier_sync_reset:** 同期設定='0' をレジスタに書込む。

バリア同期時間としては、2 回目の barrier 関数実行に要したクロックサイクル数を計測した。計測は、xilinx 社製 RTL シミュレータ ISIM（0.40d 版）を使用して行った。また、比較ためにハードウェアバリアを使用しないテストプログラム 2 種に対しても計測を行った。これらのテストでは、2 回目のバリア関数実行前にハードウェアバリア関数を 2 回実行して、全コアが同時にバリア関数を実行できるようにしている。2 種のプログラムはそれぞれ、バリア同期の手法[3] が異なっており、一方は小ノード数向けの master-slave 方式、もう一方は中以上のノード数向けの butterfly 方式を用いている。

16 コア全てがバリア同期に参加した時の計測結果を表 1 に示す。 表

表 1 バリア同期時間測定

種別	開始最速	開始最遅	完了最速	完了最遅	完了-完了
HW	0	8	55	63	8
MS	0	8	4,837	5,471	634
BUT	0	8	3,891	4,131	240

単位:クロックサイクル

中の' 種別' は測定対象の種別で、'HW' はハードウェアバリア、'MS' は Master-Slave 方式、'BUT' は butterfly 方式を示す。' 開始最速'、' 開始最遅'、' 完了最速'、' 完了最遅' はそれぞれ開始最速を起点とした相対経過クロックサイクル数を示す（**図 11**）。' 開始最速' は barrier 関数の実行開始が最も早かったコアの相対経過クロックサイクル数で、表中ではここを起点とするため、全ての種別において値は 0 となる。' 開始最遅' は、barrier 関数の実行開始が最も遅かったコアの相対経過クロック数を示し、' 完了最速' は、barrier 関数の完了が最も速かったコア、完了最遅は最も遅かったコアの相対経過クロック数を示す。' 完了-完了' は完了最速と完了最遅のクロック差を示す。' 完了最遅' の値を関数実行レイテンシ、' 完了-完了' をコア間のばらつきと考えると、HW は、MS に対してレイテンシを 1/87 に、ばらつきを 1/79 に、BUT に対してはレイテンシを 1/66 に、ばらつきを 1/30 に改善している。

3.2 回路規模評価

回路規模は Xilinx 社製の論理合成ツール xst（ISE 13.10.40 版）を使用して求めた。合成対象 FPGA には Vertex-6（ML605 評価ボード）を指定している。表 2 にバリア同期時間測定に使用した SMYLEref モデルの合成結果、及び、そこからハードウェアバリア部分のみを取り出した合成結果を示す。

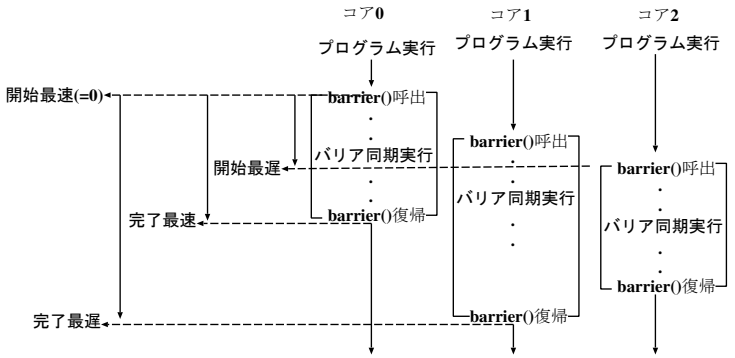

図 11 測定対象

表 2 FPGA 合成による回路規模

Type	BAR	SMYLE	BAR/SMYLE(%)
LUT	2537	192877	1.32
Reg	2361	130289	1.81
RAM	0	117	0

表中の'Type' は計数された対象で、'LUT' は LUT、'Reg' はレジスタ、'RAM' は 36Kbit RAM(ブロック RAM) を示す。また、'BAR' はバリア同期部分、'SMYLE' は SMYLEref の合成結果を表す。'BAR/SMYLE' は LUT、Reg 及び RAM それぞれについて SMYLE に対する BAR の百分率である。

BAR/SMYLE はコアの増加によっては変わらないが、クラスタが増加すれば減少する。コア当たり 32KByte 程度のメモリに必要な面積のことも鑑みると、SMYREref メニーコアプロセッサに対してバリアのハードウェア回路は十分に小規模であると考えられる。

4. 関 連 研 究

大規模システムにおいては、IBM、NEC、富士通等各社がシステムに比して小規模かつ高速なハードウェアバリア機能を実装している[4)5)6)]。しかしながら、いずれもマルチコアプロセッサ内で扱う規模と比すれば大きなものである。

オンチップマルチコア・メニーコアプロセッサ用の研究も行われている[7)8)9)]。villa らは、コア間データ通信用の NoC に回路追加を行ってバリアの高速化を行っている。ネットワークをデータと共用しているため、バリア同期のレイテンシは固定にならない。Ito ら及び hofler らは、専用のネットワークを用いてバリア同期を行っているが、いずれもネットワーク内のコアのグループ分けをハードウェア回路によって行う機能は持っていない。

5. 終 わ り に

本論文では、小規模なハードウェアバリア回路にバリアネットワーク分割機能を持たせ、さらにネットワークを多重化することにより、高速、安定かつ、柔軟なバリア同期を実現する手法を提案した。

実験では、バリア同期時間評価において、ハードウェアバリア機構を使用した場合レイテンシが 1/66、ばらつきは 1/30 となり、提案手法の有効性が示された。また、回路規模評価では、ハードウェアバリア回路はメニーコアプロセッサ全体に対して、LUT 数で 1,3%、レジスタ数で 1.8%であり、小規模なものとなっている。

本論文では、ハードウェアバリアに特化した回路の提案を行ったが、大規模システムにおいて、ハードウェアバリアがリダクション演算機能の一部として実装されるケースが見られる。そこで、メニーコアプロセッサにおけるリダクション演算用ネットワークの検討、ならびに、ハードウェアバリア用ネットワークとの統合の効果を検討するといった課題にも取り組んでいきたい。

謝辞 本研究は，一部，独立行政法人 新エネルギー・産業技術総合開発機構 (NEDO) の支援による．

参 考 文 献

1) Tilera: TILE-Gx-3000 Series, Tilera (online), available from ⟨http://www.tilera.com/products/processors/TILE-Gx-3000⟩ (accessed 2012-02-17).
2) グェンチュオンソン，レイジャオ，近藤正章，平尾智也，井上弘士：FPGA を用いたメニーコア・アーキテクチャ SMYLEref の評価環境の構築，情報処理学会研究報告，Vol.198, No.15, pp.1–7 (2012).
3) Wilkinson, B. and Allen, M.: *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, Upper Saddle River, NJ, USA (1999).
4) Gara, A., Blumrich, M.A., Chen, D., Chiu, G. L.-T., Coteus, P., Giampapa, M. E., Haring, R. A., Heidelberger, P., Hoenicke, D., Kopcsay, G.V., Liebsch, T.A., Ohmacht, M., Steinmacher-Burow, B.D. and andP. Vranas, T.T.: Overview of the Blue Gene/L system architecture, *IBM Journal of Research and Development*, Vol.49, No.2, pp.195–212 (2005).
5) Habata, S., Umezawa, K., Yokokawa, M. and Kitawak, S.: Hardware system of the Earth Simulator, *Parallel Computing*, Vol.30, No.12, pp.1287–1313 (2004).
6) Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6d mesh/torus interconnect for exascale computers, *Computer*, Vol.42, No.11, pp. 36–40 (2009).
7) Hoefler, T., Mehlan, T., Mietke, F. and Rehm, W.: Adding Low-Cost Hardware Barrier Support to Small Commodity Clusters, *19th International Conference on Architecture and Computing Systems - ARCS06*, pp.343–350 (2006).
8) Villa, O., Palermo, G. and Silvano, C.: Efficiency and scalability of barrier synchronization on NoC based many-core architectures, *Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems* (Altman, E., ed.), Atlanta, GA, USA, ACM, ACM, pp.81–90 (2008).
9) Ito, M., Hattori, T., Yoshida, Y., Hayase, K., Hayashi, T., Nishii, O., Yasu1, Y., Hasegawa, A., Takada, M., Ito, M., Mizuno, H., Uchiyama, K., Odaka, T., Shirako, J., Mase, M., Kimura, K. and Kasahara, H.: An 8640 MIPS SoC with Independent Power-Off Control of 8 CPUs and 8 RAMs by An Automatic Parallelizing Compiler, *2008 IEEE International Solid-State Circuits Conference Digest of Technical Papers* (Fujino, L.C., ed.), San Francisco, CA, IEEE, S Digital Publishing Inc, pp.90–598 (2008).