Vol.2013-ARC-206 No.16
2013/8/1

OSCAR API標準解釈系を用いた階層グルーピング対応ハードウェアバリア同期機構の評価

川島 慧大[1]　金羽木 洋平[1]　林 明宏[1]　木村 啓二[1]　笠原 博徳[1]

概要：1 チップ内に搭載されるコア数の増加に伴い，アプリケーションからより多くの並列性を抽出し，低オーバーヘッドで利用することがこれらのコアを有効利用するために重要となっている．OSCAR コンパイラによる自動並列化ではより多くの並列性を利用するため，ループやサブルーチン内部の粗粒度並列性を解析し，階層的にタスク定義を行う．この階層的に定義されたタスクをコアを階層的にグルーピングし，コアグループに対して割り当てることにより並列処理を実現する．この階層的なグループ間で独立かつ低コストでバリア同期を実現できるハードウェアが提案され，SH4A プロセッサ 8 コア搭載の情報家電用マルチコア RP2 に実装されている．本稿では，OSCAR API 標準解釈系の階層グループバリア同期 API を RP2 のハードウェアバリア同期機構に対応し評価を行った結果について述べる．8 コアを使用した SPEC CPU 2000 の ART による評価ではソフトウェアでのバリア同期に対し 1.16 倍の性能向上が得られた．

1. はじめに

近年，あらゆる分野においてコンピュータシステムが使用されており，高性能かつ省電力なプロセッサとしてマルチコアプロセッサが普及しており，中でもコア数が多いもので 64 コアや 80 コアが 1 チップに搭載されているものも開発されている [7][8]．マルチコアプロセッサでは，複数のプロセッサでプログラムを並列実行することで, 低周波数・低電力でも高性能を引き出すことが可能である．バリア同期は並列処理を実現するための基本要素であるが，コア数の増加と共にそのオーバーヘッドが並列処理性能向上の妨げとなる．さらに，今後のメニーコアの各コアを有効利用するために，各コアを任意のコア数で階層的にグルーピングし，各グループで独立的に並列処理を行う必要があると考えられる．

現在の主記憶共有型のマルチコアシステム上でバリア同期を実現する場合，メモリ上にバリア同期用のフラグ変数を配置し，排他制御を使用しつつフラグ変数を操作することによって実現する方法が主流である [9]．しかし，この手法の場合排他制御を使用するためコア数の増加への対応が困難である．またハードウェアでのバリア同期として，バリア同期に到達したフラグを書き込む 1 ビットのレジスタを用意し，バリア同期に参加するコアに対応したレジスタの全てのビットの論理積をとることでチェックする手法が提案されている [10]．しかし，この手法においてもビットを集約し論理積の演算を行う専用のハードウェアが必要になり，コア数増加への対応が困難である．

一方，筆者等は OSCAR マルチグレイン自動並列化コンパイラの開発を行っており，これによりプログラムを並列化しコア数に応じたスケーラブルな性能向上を得ることが可能となっている [1]．このマルチグレイン並列処理では，プログラム全域の並列性を利用するためループやサブルーチン内部の粗粒度並列性を解析し，階層的にタスク定義を行う．このタスクを階層的にグルーピングされたコアに割り当てることで並列処理を実現する [5]．そのため，グルーピングされたコア間でバリア同期を行う必要がある．この階層グルーピングに対応したハードウェアバリア同期機構が提案され，情報家電用マルチコア RP-2 に実装されている [6]．

また，OSCAR コンパイラによる並列化を様々な主記憶共有型マルチプロセッサ及びマルチコアシステムで実現するために OSCAR API[2] が提案されている．さらに OSCAR API のコンパイラディレクティブが挿入された並列化プログラムを，ターゲットプラットホームのランタイムライブラリコール入りプログラムに変換する OSCAR API 解釈系 [3] が開発されている．

本稿では，OSCAR API 標準解釈系の同期 API を用いたハードウェアバリア同期機構への対応を RP-2 のハードウェアバリア同期機構を用いて，評価を行った結果について述べる．

[1] 早稲田大学
Waseda Universoty

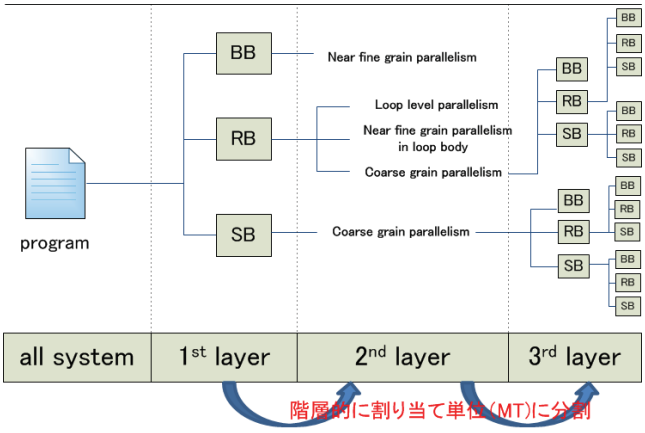

図 1 マクロタスクの階層的な定義
Fig. 1 A hierarchical definition of macro task

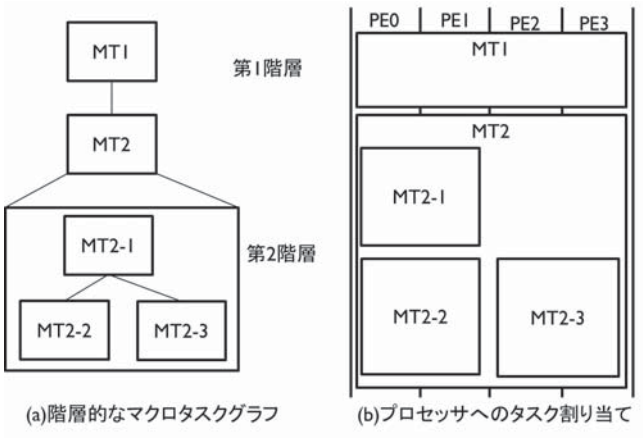

図 2 階層的なコード生成イメージ
Fig. 2 An image of hierarchical code generation

以下，第 2 章で OSCAR コンパイラと OSCAR API 標準解釈系の概要，第 3 章でハードウェアバリア同期機構，第 4 章で OSCAR API 標準解釈系によるバリア同期のコード変換，第 5 章で情報家電用マルチコア RP-2 の概要，第 6 章で評価結果を述べ，第 7 章で本稿のまとめとする．

2. OSCAR マルチグレイン自動並列化コンパイラ

OSCAR マルチグレイン自動並列化コンパイラは，基本的に逐次的に記述された C および Fortran のソースコードから並列化された C および Fortran を出力する Source to Source のコンパイラとして動作する．本章では OSCAR コンパイラの主要構成要素であるマルチグレイン並列処理についてと，OSCAR コンパイラで使用しているバリア同期，および OSCAR API 標準解釈系について説明する．

2.1 マルチグレイン並列処理

マルチグレイン並列処理とは，粗粒度タスクレベル並列性，ループイテレーションレベル並列性，ステートメントレベル近細粒度並列性の 3 つの並列性を効果的に組み合わせた並列処理手法 [4] である．

OSCAR コンパイラでは，ソースプログラムを基本ブロック（BB），繰り返しブロック（RB），サブルーチンブロック（SB）の 3 種類のマクロタスク（MT）に分割する．これを複数のプロセッサエレメント（PE：コアと同義）から構成されるプロセッサグループ (PG) に割り当てて実行することにより，MT 間の並列性を利用している．

ある RB や SB の内部にさらに粗粒度並列性が存在する場合，**図 1** に示すように MT 内部で階層的にさらに MT への分割を行う．このように階層的に定義された MT を PG を階層的に定義し（階層グルーピング），PG 単位で割り当てて実行する．

上記の階層的粗粒度タスク並列処理を行った後，RB にループの並列性が存在する場合，PG 内でループ分割を行う．また，ソースコードのステートメントレベルの並列性が利用可能な場合，近細粒度並列処理を施す．

図 2 に階層的なマクロタスクグラフと PE へのタスク割り当てのイメージを示す．図 2(a) のマクロタスクグラフを説明すると，上下に並ぶ MT1 と MT2 には依存関係が存在する．また，MT2 内部にさらに粗粒度並列性が存在し，階層的なマクロタスクが定義されている．この内部は MT2-1 と MT2-2，MT2-1 と MT2-3 には依存が存在し，MT2-2 と MT2-3 で粗粒度タスクレベル並列性が利用可能となっている．

これを 4 コアのシステムに対して生成したコードのイメージが図 2(b) である．この例の場合，まず第 1 階層において PE0-PE3 の 4 コアを 1 つのグループとしてグルーピングされ，これに対し，MT1 および MT2 が割り当てられる．このとき MT1 に対してループ並列化および近細粒度並列化を適用し，タスク内処理を 4 コアに割り当てる．一方 MT2 では，内部の粗粒度並列性が利用可能なため，第 2 階層として PE0-PE1 と PE2-PE3 の 2 コアずつのグルーピングを行い，それぞれのグループに対してタスク割り当てを行う．その後，タスク内処理を 2 コアに割り当てる．

このように生成されたコードではタスク内部において，必要に応じてバリア同期が行われる．この際，MT2-2 と MT2-3 の粗粒度並列性を阻害しないために，PG 内のバリア同期が他の PG に影響を与えてはならない．また，MT2-2 と MT2-3 の処理が同時に終了するとは限らないため，一方が第 2 階層脱出の同期を待っている状態で，もう一方が処理を実行しているという状況が考えられる．そのため，第 1 階層と第 2 階層のバリア同期は独立していなければならない．

2.2 OSCAR コンパイラにおけるバリア同期手法

OSCAR コンパイラではコア数に応じたスケーラブルな性能向上を目指すため，コアのロックを伴わなず，低レイテンシでバリア同期を処理する必要がある．そのため，ソ

```
//バージョンナンバインクリメント
barrier_flg [ 0 ] [ 0 ] [ 1 ] ++ ;
//PE1 の到達フラグ確認
while ( barrier_flg [ 0 ] [ 1 ] [ 1 ] <
        barrier_flg [ 0 ] [ 0 ] [ 1 ] ) { }
//PE2 の到達フラグ確認
while ( barrier_flg [ 0 ] [ 2 ] [ 1 ] <
        barrier_flg [ 0 ] [ 0 ] [ 1 ] ) { }
//PE3 の到達フラグ確認
while ( barrier_flg [ 0 ] [ 3 ] [ 1 ] <
        barrier_flg [ 0 ] [ 0 ] [ 1 ] ) { }
//解放フラグ送信
barrier_flg [ 1 ] [ 0 ] [ 1 ] ++ ;
```

(a) Master

```
//到達フラグ送信
barrier_flg [ 0 ] [ 1 ] [ 1 ] ++ ;
//解放フラグ確認
while ( barrier_flg [ 1 ] [ 0 ] [ 1 ] <
        barrier_flg [ 0 ] [ 1 ] [ 1 ] ) { }
```

(b) Worker

図 3 バリア同期コードの例
Fig. 3 A example of barrier synchronization code

フトウェアバリア同期手法として Master-Worker 方式を採用し，バージョンナンバを用いたバリア同期コードを生成する．この方式は，バリアを管理する 1 つの PE（Master）が他の PE のバリアへの到達を確認し，バリア解放通知を行う．バリア同期に参加する Master 以外の PE（Worker）はバリアに到達するとそれを Master に通知し，Master のバリア解放通知を待つ．バリア同期は現在の階層の PG ごとに行われ，Master は PG 内の PE のみ監視する．

PE0 から PE3 がグルーピングされている場合の PE0（Master）と PE1（Worker）のバリア同期コードの例を**図 3** に示す．図 3（a）は Master 用の同期コードであり，PE1 から PE3 のバリア同期到達フラグを busy wait ループにより待っている．全 PE からのフラグ到着を確認後，フラグ変数への代入によりバリア開放を各コアに通知する，一方図 3（b）は Worker 用のコードであり，フラグ変数への代入により Master にバリア同期到達を通知した後，busy wait ループにより Master からのバリア開放フラグを待つ．

各バージョンナンバ変数およびフラグ変数はプログラム実行時に 0 に初期化される．これらの変数はバリア同期ごとに別の変数を参照するため，リセットが不要であり，オーバーヘッドの小さなバリア同期を行うことが可能となっている．

2.3 OSCAR API 及び標準解釈系

OSCAR コンパイラによる並列化を様々な主記憶共有型マルチプロセッサ及び，マルチコアシステム上で実現するために，OSCAR API が提案されている．OSCAR API は OpenMP のサブセットをベースにしたディレクティブベースの API であり，スレッド制御，データ転送，タイマ，電力制御，アクセラレータ及び同期の各 API で構成されている．OSCAR コンパイラは，OSCAR API のディレクティブを挿入し並列化された C あるいは Fortran プログラムを生成する．

```
#pragma oscar groupbarrier vpcs(vpcgroups) [id(barrierid)]
{
        バリア等価コード
}
```

図 4 groupbarrier 指示文の文法
Fig. 4 Grammer of groupbarrier direction sentence

表 1 vpcs 指定例
Table 1 An Example of vpcs specification.

指定例	指定されるプロセッサエレメンツ
vpcs(1)	1
vpcs(2-4)	2,3,4
vpcs(3-5,7)	3,4,5,7

本稿が対象とする階層グルーピング対応バリア同期は，`groupbarrier` 指示文として OSCAR API では定義されている．`groupbarrier` 指示文の文法を**図 4** に示す．`groupbarrier` 指示文は vpcs と id の 2 つの値を指定して使用する．vpcs はバリア同期に参加する PE を指定する．指定例については**表 1** で示す．複数のコアを指定する場合，「-」を用いることで範囲指定が可能で，また連続でない PE を指定する場合は「,」で区切り指定する．id ではバリア同期に使用するハードウェア資源を指定する．API を使用しない場合は括弧内に記述されたバリア等価コードを使用してバリア同期を行う．

環境に依存しない解釈系として，OSCAR API をランタイムライブラリに変換する OSCAR API 標準解釈系が提供されている [3]．このランタイムライブラリは開発が比較的容易に可能となっており，プラットホーム毎に用意することにより，マルチプラットホームを実現している．

OSCAR API 標準解釈系と開発環境の構成を**図 5** に示す．OSCAR API 標準解釈系は，OSCAR API を含む C あるいは Fortran プログラムを入力とし，設定ファイルの設定に従いランタイムライブラリ関数を含む C あるいは Fortran プログラムを出力する．設定ファイルにアーキテクチャごとの設定を記述することで，各 API をランタイムライブラリ関数以外にネイティブコンパイラで利用可能な指示文への変換も行うことが可能となっている。

3. ハードウェアバリア同期機構

本章では階層グルーピングに対応したハードウェアバリア同期機構について説明する．

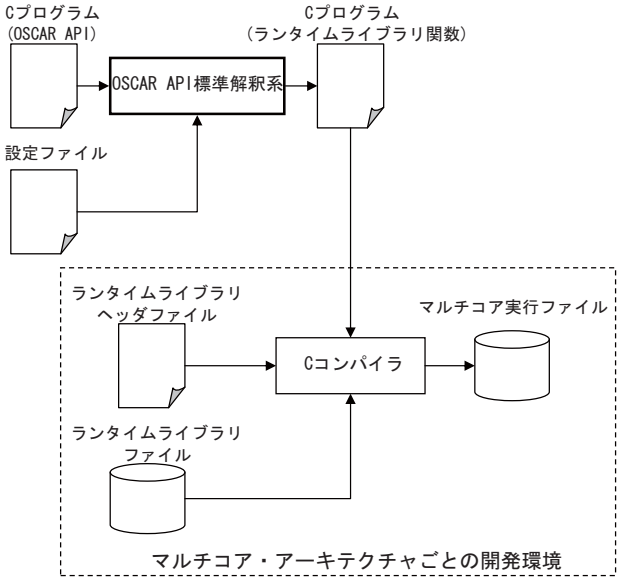

図 5 OSCAR API 標準解釈系と開発環境の構成
Fig. 5 Structure of OSCAR API standard translator and Development environment

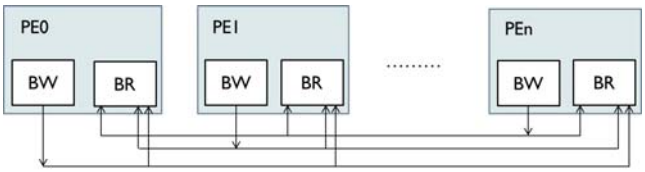

図 6 階層グルーピング対応バリア同期ハードウェア
Fig. 6 A hardware of barrier synchronization supporting hierarchical grouping

3.1 ハードウェアの概要

バリア同期用ハードウェアの概念図を**図 6** に示す．図では PE0-PEn の各コアに 1 ビットの書き込み専用レジスタ BW と，n ビットの読み込み専用レジスタ BR が設置され，配線されている．なお，この図では 1 セット分のみ表示している．各コアが自コアの BW レジスタに書き込むと，その情報はブロードキャストされ全コアの BR レジスタの該当ビットに反映される．この図の BR レジスタでは下位ビット側から順に PE0，PE1 と並んでいき，最上位ビットが PEn に該当している．

OSCAR コンパイラのコード生成ではコア数 n の場合 $\log_2 n$ セットレジスタを用意すればハードウェア資源として十分である．

3.2 ハードウェアによるバリア同期手法

バリア同期用ハードウェアを用いたバリア同期コードの例を**図 7** に示す．このコードは OSCAR コンパイラのバリア同期で用いられている Master-Worker 方式とは異なり，すべてのコアで共通のものとなっている．また，レジスタ以外の変数は各コアが独自に管理する．BW レジスタは 1 ビットのみ書き込みを行えるため，0 と 1 を交互に書き込む必要がある．そのため 1 ビットの変数 VERSION_NUMBER を用意し，毎回ビットを反転させることで交互に書き込むことができる．同様の理由で到達フラグ確認もビットを交互に反転する必要がある．変数 VCGROUPS は現在の階層でグルーピングされているコアの記録に用いられ，対応するコアのビットのみが 1 となる数字が代入される．この変数を用いることで，比較用の変数 VERSION_NUMBER_for_cmp のビットを交互に反転することができる．また，BR から同期に参加するコアのみのビットを取り出すことにも用いられる．これにより，粗粒度タスクレベル並列性を阻害することなく PG 内での同期を行うことが可能となっている．

```
//バージョンナンバインクリメント
VERSION_NUMBER ^= 1 ;
//到達フラグ送信
BW = VERSION_NUMBER ;
//比較用バージョンナンバインクリメント
VERSION_NUMBER_for_cmp ^= VCGROUPS ;
//到達フラグ確認
while ( ( BR & VCGROUPS ) != VERSION_NUMBER_for_cmp )
    ;
```

図 7 バリア同期ハードウェアによるバリア同期イメージ
Fig. 7 An image of barrier synchronization with hardware

また，複数階層でのバリア同期では，バリア同期用ハードウェアのレジスタセットと変数を変えることにより階層間でのバリア同期が干渉することなく対応可能となる．

4. OSCAR API 標準解釈系によるバリア同期コード変換

本章では OSCAR API の同期 API と標準解釈系によるコード変換について説明する．**図 8** に同期 API とコードの変換例を示す．図 8 (a) では `groupbarrier` 指示文によりバリア同期を生成している．この例は PE0 または PE1 のコードで，第 1 階層では PE0 から PE3 の 4 コアのグルーピングが行われており，id が 0 のハードウェア資源を指定する．第 2 階層では PE0 から PE1 と PE2 から PE3 の 2 コアのグルーピングが行われており，id が 1 のハードウェア資源を指定する．

OSCAR API 標準解釈系では，変換を行う場合バリア同期の各指示文を図 8 (b) のように `oscar_groupbarrier` というランタイムライブラリ関数に変換し，指示文で与えた情報を引数とするようにしている．変換を行わない場合は括弧内で指定されたバリア等価コードのみ出力する．バリア同期に参加する PE の情報は対応したビットが 1 になった変数としてグローバル変数の配列 vcgroups に保存される．具体的には 0-3 と指定されていた場合 0xf, 0-1 と指定されていた場合 0x3 となる．`oscar_groupbarrier` 関数は各プラットホームの仕様に基づいて実装すればよい．

```
//第 1 階層
#pragma oscar groupbarrier vpcs(0-3) id(0)
{
  //バリア等価コード
}
//第 2 階層
#pragma oscar groupbarrier vpcs(0-1) id(1)
{
  //バリア等価コード
}
```

(a) OSCAR API 指示文

```
//第 1 階層
oscar_groupbarier(vcgroups[0], 1, 0) ;
//第 2 階層
oscar_groupbarier(vcgroups[1], 1, 1) ;
```

(b) ランタイムライブラリ関数

図 8 同期 API とランタイムライブラリへの変換
Fig. 8 Synchronizetion API and translate to runtime library

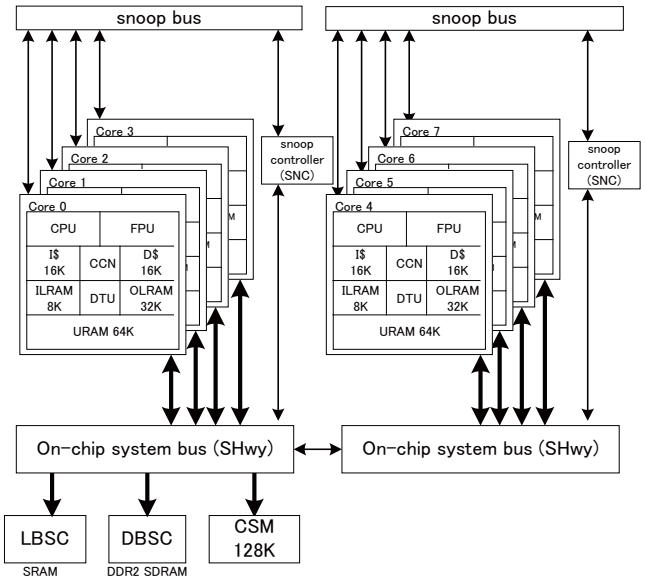

図 9 情報家電用マルチコア RP-2
Fig. 9 Multi-core RP-2

5. 情報家電用マルチコア RP-2

本章では情報家電用マルチコア RP-2 の概要について説明する．RP-2 は NEDO“リアルタイム情報家電用マルチコア技術の研究開発”プロジェクトの一環で日立製作所・ルネサステクノロジ・早稲田大学が共同開発したもので，3.1 章で述べた階層グルーピング対応ハードウェアバリア同期機構が実装されている．

RP-2 のアーキテクチャを**図 9** に示す．RP-2 は SH4A（SH-X3）コアを 8 コア搭載したホモジニアスマルチコアとなっている．各コアは命令キャッシュおよびデータキャッシュを持ち，SMP モードではスヌープコントローラが専用のスヌープバスを介して 4 コア毎にデータキャッシュの一貫性を保証する．そのため 5 コア以上使用時にはスヌープコントローラを無効にしてソフトウェアで一貫性の保証を行う必要がある．また各コアはローカルプログラムメモリ（ILRAM），ローカルデータメモリ（OLRAM），および分散共有メモリ（URAM）の 3 種のローカルメモリがある．集中共有メモリは on-chip のもの（CSM）と off-chip のもの (DDR2SDRAM) があり，各コアとスプリットトランザクションバス（SHwy）で接続されている．ハードウェアバリア同期機構に関しては各コアが書き込み用と読み込み用のレジスタを 3 セットずつ搭載しており，あるコアが書き込み用レジスタに書き込んだ値（1 or 0）は即座に全てのコアの読み込み用レジスタの対応する領域に反映される．

6. 性能評価

本章では OSCAR API 標準解釈系の同期 API のランタイムライブラリを RP-2 の階層グルーピング対応ハードウェアバリア同期機構に対応し，評価を行った結果について説明する．

6.1 評価環境

評価は OSCAR コンパイラによるソフトウェアバリア同期と比較して行う．なお，ソフトウェアバリア使用する変数はメモリアクセスの想定と，8 コア時のデータの一貫性保障ため，DDR2SDRAM 上のキャッシュ無効領域に配置し，ランタイムライブラリで使用する変数は OLRAM に配置した．

評価に用いたプログラムは以下の 2 つである．

- □ バリア同期テストコード
  ループによりバリア同期のみを 10000 回行う．これにより同期の正常動作と同期コスト測定を行える．
- □ ART
  SPEC CPU2000 のベンチマークアプリケーション．ニューラルネットワークを使った画像認識を行う．OSCAR コンパイラによって並列化を行い評価．

6.2 評価結果

バリア同期テストコードの評価結果を**図 10**，ART の評価結果を**図 11** に示す．各グラフの横軸は使用コア数を示している．また図 10 の縦軸は実行クロック数，図 11 の縦軸は速度向上率となっている．図中，SW は図 3 に示したソフトウェアによるバリア同期，API は OSCAR API 標準解釈系の同期 API のランタイムライブラリにより対応した階層グルーピング対応ハードウェアバリア同期機構によるバリア同期をそれぞれ表す．

図 10 を見ると，ソフトウェアバリア同期はコア数の増加に伴いメモリウォールにより同期コストが増加する一方，API の場合はすべてのコア数でほぼ同一の同期コストで同期を処理していることがわかる．また，ソフトウェアバリア同期が平均 390 クロックから 1160 クロック必要な

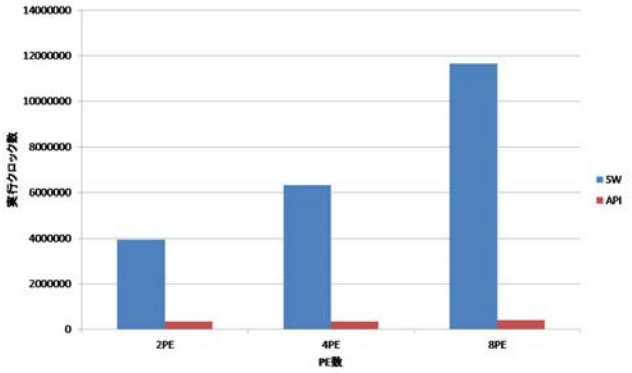

図 10 テストコードの評価結果
Fig. 10 Result of barrier synchronization test code

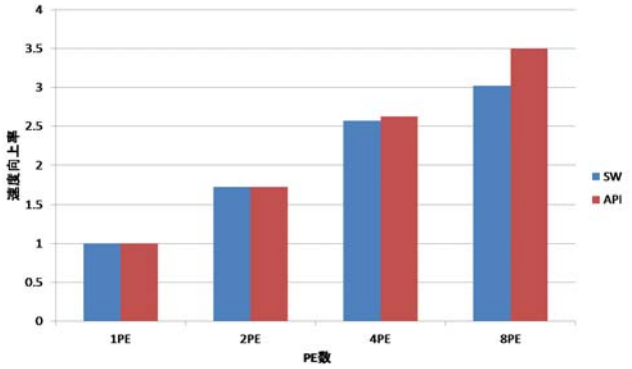

図 11 ART の評価結果
Fig. 11 Result of ART

のに対し，API では 33 クロックから 38 クロックで処理が終了し，2 コア時で比較してわずか 8.5%と非常にオーバーヘッドの小さいものとなっている．

また図 11 の ART は，並列処理において粒度の小さいタスクに対してバリア同期を行う必要があるプログラムとなっている．評価の結果，コア数の増加に応じたハードウェアによるバリア同期の効果が顕著になっており，8 コア時で 1.16 倍の性能向上が得られた．

7. まとめ

本稿では OSCAR マルチグレイン自動並列化コンパイラによる階層グルーピングに対し，動作可能なハードウェアバリア同期機構への対応方法として OSCAR API 標準解釈系での対応を行い，RP-2 上で評価を行った．その結果 SPEC CPU2000 の ART において最大 1.16 倍の性能向上が得られ，本手法によるバリア同期の有効性を示すことができた．

参考文献

[1] H. Kasahara, H. Honda, A. Mogi, A. Ogura, K. Fujiwara, and S. Narita: A Multi-grain Parallelizing Compilation Scheme for OSCAR (Optimally Scheduled Advanced Multiprocessor), 4th Intl. Workshop on LCPC, pp. 283-297, (August, 1991).

[2] Keiji Kimura, Masayoshi Mase, Hiroki Mikami, Takamichi Miyamoto, Jun Shirako and Hironori Kasahara: OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers, Proc. of The 22nd International Workshop on Languages and Compilers for Parallel Computing (LCPC2009), (October, 2009).

[3] 佐藤卓也，見神広紀，林明宏，間瀬正啓，木村啓二，笠原博徳：OSCAR API 標準解釈系を用いた Parallelizable C プログラムの評価，情報処理学会研究報告, (2011).

[4] 本多弘樹，岩田雅彦，笠原博徳：Fortran プログラム粗粒度タスク間の並列性検出手法，電子情報通信学会論文誌，(1990).

[5] 小幡元樹, 白子準, 神長浩気, 石坂一久, 笠原博徳：マルチグレイン並列処理のための階層的並列処理制御手法，情報処理学会論文誌，(2003).

[6] 山田海斗, 間瀬正啓, 白子準, 木村啓二, 伊藤雅之, 服部俊洋, 水野弘之, 内山邦男, 笠原博徳：階層グルーピング対応バリア同期機構の評価，計算機アーキテクチャ研究会報告，(2008).

[7] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal: On-Chip Interconnection Architecture of the Tile Processor, IEEE MICRO, vol. 27, no 5, pp.15-31, (September/October 2007).

[8] A. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar: A 5-GHz Mesh Interconnect for a Teraflops Processor, IEEE MICRO, vol. 27, no 5, pp.51-61, (September/October 2007).

[9] D. E. Culler, J. P. Singh, and A. Gupta.: Parallel Computer Architecture - A Hardware/Software Approach, Morgan Kaufmann Pub.,(1999).

[10] C. J. Beckmann, and C. D. Polychronopoulos: Fast Barrier Synchronization Hardware, Proc. of Supercomputing ’90, pp. 180-189,(November, 1990).