計算機アーキテクチャ 148－3
（2002．5．13）

# SMPマシン上での粗粒度タスク並列処理オーバーヘッドの解析

和田康孝[†] 中野啓史[†] 木村啓二[†]
小幡元樹[†,††] 笠原博徳[†,††]

E-mail: {yasutaka,hnakano,kimura,obata,kasahara}@oscar.elec.waseda.ac.jp

マルチプロセッサシステムの実効性能を今後さらに高めていくためには，ループ並列処理に加え，ループ・サブルーチン・基本ブロック間の並列性を利用する粗粒度タスク並列処理の利用が重要である．この粗粒度タスク並列処理をより効果的に実現するためには，各種オーバーヘッドを定量的に解析する必要がある．本稿では，Sun Ultra80，IBM RS/6000 7044 Model 270，SGI Origin2000の各プラットフォーム上に用意された測定機能により，L2キャッシュメモリのミスペナルティ，バリア同期およびスレッド生成のオーバーヘッドについて解析を行い，その結果をもとにOSCAR Fortran並列化コンパイラによる性能向上の要因について述べる．

# Evaluation of Overhead with Coarse Grain Task Parallel Processing on SMP Machines

Yasutaka Wada,[†] Hirofumi Nakano,[†] Keiji Kimura,[†]
Motoki Obata[†,††] and Hironori Kasahara [†,††]

E-mail: {yasutaka,hnakano,kimura,obata,kasahara}@oscar.elec.waseda.ac.jp

Coarse grain task parallel processing, which exploits parallelism among loops, subroutines and basic blocks, is getting more important to attain performance improvement on multiprocessor architectures. To efficiently implement the coarse grain task parallel processing, it is important to analyze various processor overhead quantitatively. This paper evaluates overheads of barrier synchronization, thread fork/join and L2 cache miss penalty are using performance measurement mechanisms to analyze the performance improvements by OSCAR Fortran compiler on Sun Ultra80, IBM RS6000 and SGI Origin2000.

## 1. はじめに

マルチプロセッサシステムにおいて従来から行われてきたループ並列処理は，データ依存解析及びリストラクチャリング手法の開発により成熟期を迎え，今後劇的な性能向上は難しいと考えられている．そのため，マルチプロセッサシステムの実効性能をさらに高めるためには，ループ並列処理に加え，ループ・サブルーチン・基本ブロック間の並列性を利用する粗粒度タスク並列処理を併用するなど，プログラム全域より並列性を抽出するマルチグレイン並列化が必要となっている．

従来の研究から，この粗粒度タスク並列処理やデータの局所性を利用したキャッシュ最適化手法を適用することにより大きな性能向上が得られることが確認されているが[1)2)3)]，その性能向上の要因を明らかにしていかなければならない．

パフォーマンス解析に関連して，パフォーマンス測定のための統一的インタフェースであるPAPI[4)]や，OpenMPディレクティブ[5)]に関わるオーバーヘッドを測定するMicrobenchmark[6)]等が提供されている．本研究の測定では，各種プラットフォームのメーカーが提供する自動並列化コンパイラ及びOSCAR並列化コンパイラの挙動をクロックサイクル単位で測定するため，各メーカーの提供するパフォーマンス測定APIを使用した．具体的には，Sun Ultra80[7)]，IBM RS/6000 7044 Model 270[8)]，SGI Origin2000[9)]の各アーキテクチャ上に用意された測定機能[10)11)12)]により，L2キャッシュメモリのミスペナルティ，バリア同期およびスレッド生成のオーバーヘッドについて解析を行い，その結果を用いてOSCAR Fortran並列化コンパイラ[13)]による性能向上の要因について述べる．

以降，第2章では粗粒度タスク並列処理，第4章で

† 早稲田大学理工学部電気電子情報工学科
Dept. of Electrical, Electronics and Computer Engineering, Waseda University

†† アドバンスト並列化コンパイラ研究体
Advanced Parallelizing Compiler Reserch Group
http://www.apc.waseda.ac.jp/

は測定を行った環境，第5章では測定・解析方法，第6章では解析結果について述べる．

## 2. 粗粒度タスク並列処理

### 2.1 粗粒度タスク並列処理の概要

粗粒度タスク並列処理とは，ソースプログラムを疑似代入文ブロック (BPA)，繰り返しブロック (RB)，サブルーチンブロック (SB) の3種類ののマクロタスク (MT) に分割し，その MT を複数のプロセッサエレメント (PE) を持つプロセッサクラスタ (PC) に割り当てて実行することによって MT 間の並列性を利用する並列処理手法である．

OSCAR Fortran 並列化コンパイラにおける粗粒度タスク並列処理の手順は次のとおりである．

(1) プログラムを階層的に3種類の MT に分割
(2) 各階層の MT 間のコントロールフローやデータ依存を解析してマクロフローグラフ (MFG) を生成
(3) MFG 上の制御依存とデータ依存を考慮して MT 間の並列性を抽出する最早実行可能条件解析を行い，マクロタスクグラフ (MTG) を生成
(4) MTG がデータ依存エッジのみを持つ場合には，MT はスタティックスケジューリングによって PC または PE にコンパイル時に割り当てられる．一方，MTG が条件分岐等の実行時不確定性を持つ場合には，コンパイラがユーザコード中に生成したダイナミックスケジューリングルーチンによって，MT を PC または PE に実行時に割り当てる．

以上のような粗粒度並列性の解析及び抽出は OSCAR Fortran 並列化コンパイラにより実現されている．

## 3. OSCAR Fortran 並列化コンパイラ

### 3.1 構　　成

OSCAR Fortran 並列化コンパイラはフロントエンド，ミドルパス，バックエンドから構成される．

フロントエンドは Fortran77 のソースコードの字句解析・構文解析を行い，ミドルパスは，コントロールフロー解析，データ依存解析を行い，プログラムのリストラクチャリング，マクロタスクの生成，およびこれらを用いた並列性の抽出を行う．

OSCAR Fortran 並列化コンパイラは OSCAR マルチプロセッサシステムや OpenMP を始めとするさまざまなターゲット用のバックエンドを持ち，ミドルパスが出力した中間言語から各ターゲット用のアセンブリコードや並列化 Fortran コードを出力する．

今回は OpenMP バックエンドを用い，生成した OpenMP Fortran プログラムを各アーキテクチャ向けのネイティブコンパイラを用いてコンパイルし，実行させた．つまり OSCAR Fortran 並列化コンパイラはプリプロセッサとして動作することになる．OpenMP バックエンドにより生成される OepnMP Fortran プログラムの実行イメージを図1に示す．

### 3.2 粗粒度タスク並列処理の実行モデル

OpenMP による粗粒度タスク並列処理の実行モデルを図1を用いて説明する．本モデルでは図1に示すように，"!$OMP PARALLEL SECTIONS" ディレクティブを用いてスレッドの fork/join を一度のみで済ませることでオーバーヘッドを最小限に抑えるワンタイム シングルレベル スレッド生成法を用いている．

図1の例では，プログラム開始時に8スレッドが fork される．図の例では，第1階層 (プログラム全体) の内部に MT1_1，MT1_2，MT1_3 が定義されている．生成された8スレッドは第一階層では4スレッドずつにグループ化 (PC) されており，グループ単位で MT を実行する．この例では第1階層にはスタティックスケジューリングが適用され，スレッド 0 - 3 のグループに MT1_1，MT1_3 が，スレッド 4 - 7 のグループに MT1_2 がそれぞれ割り当てられており，各セクション毎に割り当てられた MT 用のコードがコンパイラによって生成される．

SB である MT1_2 の内部には第2階層が定義され，内部をさらに MT に分割し，集中ダイナミックスケジューリングを適用し，グループ内の 4 スレッドによって粗粒度タスク並列処理が階層的に行われる．集中スケジューリングの場合，スレッド 7 のように1つのスレッドをスケジューラのコードが占有する．

一方，RB である MT1_3 の内部にも第2階層が定義され，グループ内の4スレッドをさらに2スレッドずつにグループ化して，分散ダイナミックスケジューリングによる粗粒度タスク並列処理が行われる．分散ダイナミックスケジューリングでは，各スレッドがスケジューリングと MT の実行を行うので，各セクションには MT のコードとスケジューリングルーチンが生成される．

並列処理可能ループである MT1_1 の処理は，グループ内の 4 スレッドに分割されている．

また，粗粒度タスク並列処理では任意のスレッド集合内でバリア同期をとることが必要となるため，"!$OMP BARRIER" ディレクティブではなく図2のようにスレッド共有変数を用いてバリア同期を行う．

図2では，まず各セクションは同期部分に差し掛かった時点で，共有変数を用いたフラグをインクリメントして1番目のセクション (PE0) に通知し，同期成立待ち状態にはいる．1番目のセクションは，そのフラグを参照してグループ内の全てのセクションが同期に差し掛かったことを確認し，再び共有変数によるフラグをインクリメントして全セクションが同期されたことを伝える．1番目のセクションによりインクリメントされたフラグをみて，他の各セクションは待ち状態から抜け出し，同期がとられる．OpenMP バックエンドによる同期は以上の様な実装となっている．

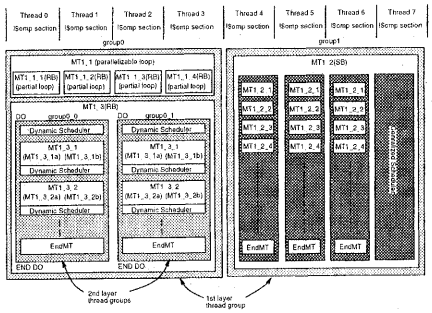


図 1　OSCAR コンパイラによる並列化コードイメージ

```
!$OMP PARALLEL SECTIONS
C=== SECTION for PE0 ===C
!$OMP SECTION
      ****PE0 での処理****
      FLG_100 = FLG_100 + 1
   10 CONTINUE
!$OMP FLUSH
      IF(FLG_110.LT.FLG_100) GOTO 10
      FLG_101 = FLG_101 + 1
!$OMP FLUSH
      ****PE0 での処理****
!$OMP END SECTION
C=== SECTION for PE1 ===C
!$OMP SECTION
      ****PE1 での処理****
      FLG_110 = FLG_110 + 1
!$OMP FLUSH
   20 CONTINUE
!$OMP FLUSH
      IF(FLG_101.LT.FLG_110) GOTO 20
      ****PE1 での処理****
!$OMP END SECTION
!$OMP END PARALLEL SECTIONS
```

図 2　OSCAR Fortran コンパイラによる同期の例 (2PE 時)

## 4. 測定環境

本研究では，Sun Ultra80，IBM RS/6000 7044 Model 270 及び SGI Origin2000 の 3 つのマシン上で測定を行った．各マシンの詳細は表 1，表 2，表 3 のとおりである．

各アーキテクチャには各種イベントを測定するための機能およびこれを用いるための API が用意されている．Sun Ultra80 および IBM RS6000 では C 対応の API のみが提供されているため，Fortran に対応したラッパーを作成してこれを用いた．

## 5. 測定・解析方法

本研究では，OpenMP による粗粒度タスク並列処理の実行に特に大きな影響を持つと考えられる L2 キャッシュのミスペナルティ，及びバリア同期やスレッド生成に関わるオーバーヘッドについて測定を行った．特に，L2 キャッシュのミスペナルティとバリア同期に関しては，実プログラムでの挙動に関するデータがコンパイラの開発に有意であると考え，SPEC95fp の 101.tomcatv，102.swim を用いて測定した．

具体的には，OSCAR Fortran コンパイラによる並列処理のオーバーヘッドと，各プラットフォームにおける OpenMP による並列処理のオーバーヘッドの比較を行っている．

### 5.1 L2 キャッシュのミスペナルティ

ここでは，L2 キャッシュのミスペナルティの測定方法について説明する．

表 1　Sun Ultra80

| Vender | Sun Microsystems |
|---|---|
| CPU | 450MHz UltraSPARC-II<br>4 台からなる SMP |
| L1 Instruction Cache | 16Kbyte<br>Pseudo 2-Way Set Associative<br>line size: 32byte |
| L1 Data Cache | 16Kbyte, Direct-Map<br>line size: 32byte<br>write-through, non-allocating |
| L2 Unified Cache | 4Mbyte, Direct-Map<br>line size: 64byte<br>write-back, allocating |
| Main Memory | 1024Mbyte |
| OS | Solaris8 |
| 使用コンパイラ | Forte for HPC<br>Ver.6 Update1 |
| 1PE 実行に対する<br>コンパイルオプション | -fast |
| OpenMP 並列処理に対する<br>コンパイルオプション | -fast<br>-mp=openmp<br>-explicitpar<br>-stackvar |

表 2　IBM RS/6000 7044 Model 270

| Vender | IBM |
|---|---|
| CPU | 375MHz POWER3-II<br>4 台からなる SMP |
| L1 Instruction Cache | 32Kbyte<br>128-Way Set Associative<br>2-Way Interleaved<br>line size: 128byte |
| L1 Data Cache | 64Kbyte<br>128-Way Set Associative<br>8-Way Interleaved<br>line size: 128byte |
| L2 Unified Cache | 4Mbyte, Direct-Map<br>line size: 64byte<br>write-back |
| Main Memory | 1024Mbyte |
| OS | AIX 4.3.3 |
| 使用コンパイラ | XL Fortran Compiler<br>Ver.7.1 |
| 1PE 実行に対する<br>コンパイルオプション | -O3<br>-qhot<br>-qarch=pwr3 |
| OpenMP 並列処理に対する<br>コンパイルオプション | -O3<br>-qsmp=noauto<br>-qhot<br>-qarch=pwr3 |

表 3　SGI Origin2000

| Vender | SGI |
|---|---|
| CPU | 250MHz MIPS R10000<br>8 台からなる ccNUMA |
| L1 Instruction Cache | 32Kbyte<br>2-Way Set Associative<br>line size: 128byte |
| L1 Data Cache | 32Kbyte<br>2-Way Set Associative<br>line size: 64byte<br>write-back |
| L2 Unified Cache | 4Mbyte<br>2-Way Set Associative<br>line size: 128byte |
| Main Memory | 2048Mbyte |
| OS | IRIX 6.5 |
| 使用コンパイラ | MIPSPro 7 Fortran 90 |
| 1PE に対する<br>コンパイルオプション | -Ofast |
| OpenMP 並列処理に対する<br>コンパイルオプション | -Ofast<br>-mp<br>-mpio |

測定の対象としたのは，OSCAR Fortran 並列化コンパイラを用いて L2 キャッシュメモリに対するキャッシュ最適化を適用したプログラムとキャッシュ最適化は適用せずループ並列性のみを利用したプログラムそれぞれの実行時間および L2 キャッシュのミスヒット

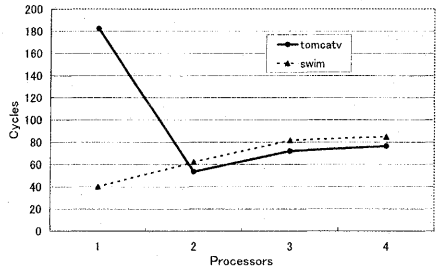

図 3 Sun Ultra80 における L2 キャッシュミスペナルティ

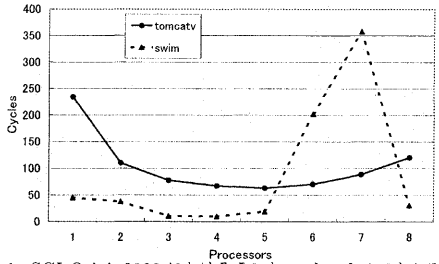

図 4 SGI Origin2000 における L2 キャッシュミスペナルティ

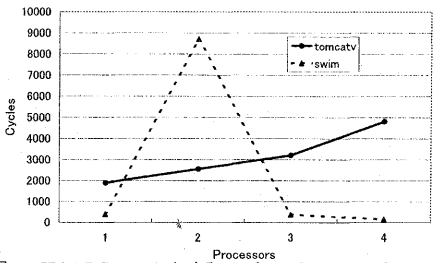

図 5 IBM RS6000 における L2 キャッシュミスペナルティ

回数である．ここでは，キャッシュ最適化適用の有無による実行時間の差は L2 キャッシュミスヒットのみに依存するという仮定を行い，L2 キャッシュメモリのミスペナルティ[cycle] は，測定した値を用いて以下のような式で求める．

$$L2\ Cache\ miss\ penalty\ = \frac{T \times f}{M}\ [cycle]\ (1)$$

ただし，T は実行時間差，f はプロセッサ動作周波数，M は L2 キャッシュミスヒット回数差である．

また，IBM RS6000 (POWER3 プロセッサ[14]) では厳密に L2 キャッシュミスの回数を測定することが出来ないため，PAPI において “Level2 Load/Store Misses” として定義されているものを計測結果として用いた．

### 5.2 並列処理時のバリア同期オーバーヘッド

本研究では，プログラム内でバリア同期を実現している部分の前後にパフォーマンス測定ルーチンを挿入することで，バリア同期のオーバーヘッドを測定した．本測定では，バリア同期を 2 回連続して行わせ，2 回目のバリア同期の測定を行うことで，純粋な同期オーバーヘッドに対して測定を行った．

OSCAR Fortran 並列化コンパイラでは，OpenMP バックエンドによって生成される OpenMP Fortran プログラムに対して測定を行った．また，比較対象として OpenMP による並列プログラムにおいて “!$OMP BARRIER” ディレクティブを用いた物を計測した．

### 5.3 スレッド生成のオーバーヘッド

本研究では，単純なループに対して OpenMP ディレクティブを用いて並列処理を行い，その処理に対してサイクル数の測定を行った．測定方法としては，Microbenchmark[6] と同様の手法を利用した．

比較対象として，スレッド生成を伴う “!$OMP PARALLEL DO” とすでに生成されたスレッド間でワークシェアリングのみを行う “!$OMP DO” ディレクティブを用いた場合を計測した．

## 6. 測定結果

本章では，第 5 章に示した手法に従って各オーバーヘッドを測定した結果について述べる．

### 6.1 L2 キャッシュミスペナルティ

各アーキテクチャ上での L2 キャッシュのミスペナルティの計測結果を，図 3，図 4，図 5 にそれぞれ示す．各図に示されているのは，各プラットフォームにおいて L2 キャッシュミスヒット 1 回あたりにかかるクロックサイクル数を測定・計算した結果であり，縦軸はクロックサイクル数 [cycle]，横軸はプロセッサ数を表している．なお，値の算出には 3 回測定を行った平均値を採用した．

Sun Ultra80 での計測結果 (図 3) を見ると，swim に対する計測値はプロセッサ数と共に増加しているが，tomcatv に対する計測値は 2 プロセッサのときに最小値を示している．なお，計測された平均値は約 66[cycle] であった．

SGI Origin2000 での計測結果 (図 4) では，swim の 6 プロセッサ時及び 7 プロセッサ時，tomcatv の 1 プロセッサ時に大きい値が算出されていることを除けば，安定した値を示している．計測された平均値は約 97[cycle] であった．

IBM RS6000 での計測結果 (図 5) では，swim の 2 プロセッサ時の値が約 8700[cycle] と大きくなっており，全体としても平均約 2760[cycle] という結果が得られた．このように他のプラットフォームに比べて大きな値が得られた原因としては，POWER3 プロセッサのハードウェアプリフェッチ機構が影響しているものと考える．このハードウェアプリフェッチ機構は，連続したデータ領域に対してキャッシュミスが発生したときに，上位のメモリ階層にあらかじめデータをフェッチしておくという機能である．そのため，式 (1) では，純粋な L2 キャッシュメモリのミスペナルティを算出することが出来なかった．

上述のように，L2 キャッシュメモリのミスペナルティが Sun Ultra80 では 平均 66[cycle]，SGI Origin2000 では平均 97[cycle] という大きな数値を示しており，L2 キャッシュを対象としたキャッシュ最適化が重要であることが確認出来た．

### 6.2 バリア同期オーバーヘッド

各アーキテクチャ上でのバリア同期オーバーヘッドの計測結果を，図 6，図 7，図 8 にそれぞれ示す．各図に示されているのは，各プラットフォームにおいて測定した OSCAR コンパイラおよび OpenMP ディレクティブ “!$OMP BARRIER” によるバリア同期 1 回あたりのオーバーヘッドであり，縦軸はクロックサイクル数 [cycle]，横軸はスレッド数を表している．ただし，採用した値は 3 回測定を行った平均値である．

Sun Ultra80 における計測結果 (図 6) をみると，OSCAR コンパイラによるバリア同期の方が，OpenMP

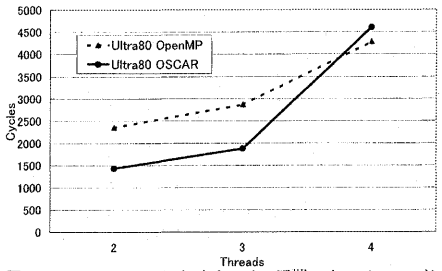


図 6　Sun Ultra80 におけるバリア同期のオーバーヘッド

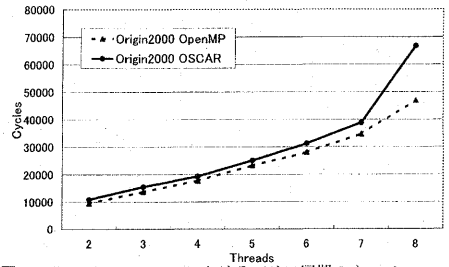


図 7　SGI Origin2000 におけるバリア同期のオーバーヘッド

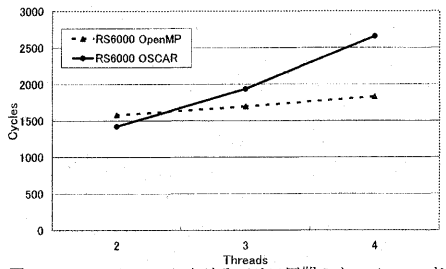


図 8　IBM RS6000 におけるバリア同期のオーバーヘッド

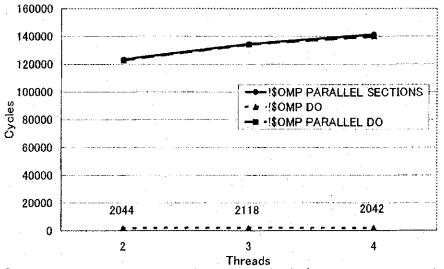


図 9　Sun Ultra80 におけるスレッド生成のオーバーヘッド

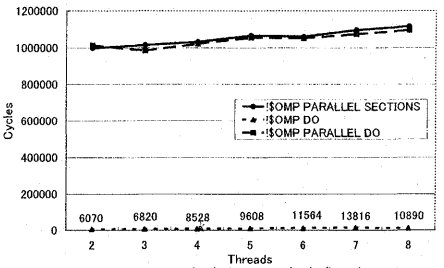


図 10　SGI Origin2000 におけるスレッド生成のオーバーヘッド

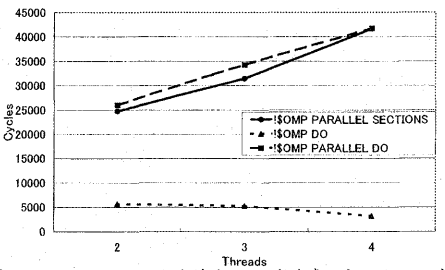


図 11　IBM RS6000 におけるスレッド生成のオーバーヘッド

ディレクティブ同期に比べ，スレッド数が 2 のときには約 920[cycle]，スレッド数が 3 のときには約 990[cycle] コストが小さい．それに対して，スレッド数が 4 になると逆に OSCAR コンパイラの方が約 300[cycle] コストが大きくなっている．

SGI Origin2000 での計測結果 (図 7) では，スレッド数が 7 までであれば，OSCAR コンパイラによる同期コストと OpenMP ディレクティブによる同期コストがほぼ同様に推移しており，OSCAR コンパイラの方が平均 2300[cycle] コストが大きいだけであるが，スレッド数が 8 になると，OSCAR コンパイラの方が約 20000[cycle] コストが大きくなっている．

IBM RS6000 での計測結果 (図 8) では，スレッド数が 2 のときには OSCAR コンパイラによるバリア同期の方が OpenMP ディレクティブによる同期に比べて約 150[cycle] コストが小さいが，スレッド数が 3 のときには約 140[cycle]，スレッド数が 4 のときには約 820[cycle] コストが大きくなっている．

また全体としては，Ultra80 では OSCAR コンパイラによる同期の方が平均約 530[cycle] コストが小さく，IBM RS6000 では平均約 300[cycle]，SGI Origin2000 では平均約 4800[cycle] OSCAR コンパイラによる同期の方がコストが大きいという結果が得られた．

以上のことから，OpenMP ディレクティブによるバリア同期と OSCAR コンパイラによるバリア同期のオーバーヘッドは，スレッド数が 4 あるいは 8 程度までであれば，1 回あたりのコストに大きな差は無いということがいえる．ただし，OSCAR コンパイラのとっている共有変数を用いたバリア同期の方式の場合，OpenMP ディレクティブを用いた場合と比べて，スレッド数の増加に伴うコストの増分がより大きくなる傾向にある．このことから，今後 OSCAR コンパイラでバリア同期コードを生成する際には，少スレッドの可変グループバリアには共有変数を用いたバリアを，全スレッドが参加するバリアには OpenMP ディレクティブによるバリアを使用する方が良いことが分かった．

### 6.3　スレッド生成のオーバーヘッド

各プラットフォーム上でのスレッド生成に関するオーバーヘッドの計測結果を，図 9，図 10，図 11 にそれぞれ示す．各図に示されているのは，各プラットフォームにおいて OpenMP ディレクティブを用いてスレッド生成あるいはワークシェアリングを行った際のオーバーヘッドを測定した結果であり，縦軸はクロックサイクル数 [cycle]，横軸はスレッド数をそれぞれ表している．ただし，システムの状態によって測定値が上下したので，ここでは 20 回測定したうち最小値を理想状態での値として採用した．

計測結果を見ると，いずれのプラットフォームにおいても “!$OMP PARALLEL SECTIONS” ある

いは "!$OMP PARALLEL DO" によるスレッドの生成にかかるオーバーヘッドはほぼ同じで，Sun Ultra80 では平均約 $3.00 \times 10^4$[cycle]，SGI Origin2000 では平均約 $1.04 \times 10^6$[cycle]，IBM RS6000 では平均約 $1.33 \times 10^5$[cycle] と大きい値を示している．一方，"!$OMP DO" によるワークシェアリングにかかるオーバーヘッドに注目すると，Sun Ultra80 では平均 2000[cycle]，IBM RS6000 では平均 4700[cycle]，SGI Origin2000 では平均 7200[cycle] 程度のコストがかかっている．これらのことから，一度のみスレッドを fork/join させることで階層的なマルチグレイン並列処理が可能で，ループを並列処理する際にスレッド間でワークシェアリングの手続きを必要としない OSCAR Fortran コンパイラのワンタイム シングルレベル スレッド生成法はオーバーヘッドを最小化するために有効であることが確かめられた．

## 7. まとめ

本論文では，SMP マシン上で粗粒度タスク並列処理を行う際に問題となるバリア同期やスレッド生成のオーバーヘッドについて解析を行い，OSCAR Fortran 並列化コンパイラによる性能向上の要因について述べた．その結果，OSCAR Fortran 並列化コンパイラは，バリア同期 1 回当りのオーバーヘッドでは OpenMP ディレクティブによる同期オーバーヘッドに対して，Sun Ultra80 では平均約 530[cycle] コストが小さく，IBM RS6000 では平均約 300[cycle]，SGI Origin2000 では平均約 2300[cycle] 程度コストが大きいことが確認され，バリア同期 1 回あたりのオーバーヘッドでは大きく引けをとらないことが分かった．また，OSCAR コンパイラによる同期方式は，スレッド数の増加に伴って，同期コストの増分がより大きくなる傾向にある．このことから，少スレッドの可変グループバリアには共有変数を用いたバリアを，全スレッドが参加するバリアには OpenMP ディレクティブによるバリアを使用する方が良いことが分かった．

一方，スレッド生成の点では，"!$OMP DO" ディレクティブによるワークシェアリングに関わるオーバーヘッドが Sun Ultra80 では平均約 2000[cycle]，IBM RS6000 では平均約 4700[cycle]，SGI Origin2000 では平均約 7200[cycle] であることと，スレッドの生成にかかるオーバーヘッドは Sun Ultra80 では平均約 $3.00 \times 10^4$[cycle]，SGI Origin2000 では平均約 $1.04 \times 10^6$[cycle]，IBM RS6000 では平均約 $1.33 \times 10^5$[cycle] と大きい値を示していることから，スレッドの fork/join を一度のみ行うだけで階層的なマルチグレイン並列処理が可能な OSCAR コンパイラの ワンタイム シングルレベル スレッド生成法がオーバーヘッドを最小化するために有効であることが確かめられた．

また，L2 キャッシュのミスペナルティが Sun Ultra80 で平均約 71[cycle]，SGI Origin2000 では平均約 97[cycle] 程度と大きいことから，L2 キャッシュメモリを対象としたキャッシュ最適化が重要であることが確認された．

今後の課題として，今回測定したデータを元に，実アプリケーションにおける OSCAR コンパイラと各メーカーの提供する自動並列化コンパイラの性能差の分析，及び OSCAR コンパイラのチューニング等が挙げられる．

**謝辞** 本研究の一部は，STARC「自動並列化コンパイラ協調型シングルチップマルチプロセッサの研究」及び経済産業省/NEDO ミレニアムプロジェクト「アドバンスト並列化コンパイラ」により行われた．

## 参考文献

1) Ishizaka, K. et al.: Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multiprocessor, *Proc. 14th Workshop on Languages and Compilers for Parallel Computing* (2001).
2) 吉田明正ほか: SMP 上でのデータ依存マクロタスクグラフのデータローカライゼーション手法, Arc2001-141-6, 情報処理学会 (2001).
3) 中野啓史ほか: キャッシュ最適化を考慮したマルチプロセッサシステム上での粗粒度タスクスタティックスケジューリング手法, Arc2001-140-11, 情報処理学会 (2001).
4) London, K. et al.: End-User Tools for Application Performance Analysis Using Hardware Counters (2001). International Conference on Parallel and Distributed Computing Systems.
5) OpenMP Architecture Review Board: *OpenMP Fortran Application Program Interface* (2000).
6) Bull, J.M. and O'Neill, D.: A Microbenchmark Suite for OpenMP 2.0 (2001). Proceedings of the Third European Workshop on OpenMP.
7) サン・マイクロシステムズ株式会社: Sun Ultra80 ワークステーション Just the Facts (1999).
8) Andersson, S. et al.: *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, IBM Corp (1998).
9) Laudon, J. and Lenoski, D.: *The SGI Origin: A ccNUMA Highly Scalable Server*, Sillicon Graphics Inc.
10) Sun Microsystems: *UltraSPARC User's manual* (1997).
11) Sillicon Graphics Inc.: *Origin2000 and Onyx2 Performance Tuning and Optimization Guide* (2001).
12) Smolders, L.: *System and Kernel Thread Performance Monitor API Toolkit Guide*, IBM Corp (2000).
13) 岡本雅巳ほか: マルチグレイン並列化 FORTRAN コンパイラ, 情報処理学会論文誌, Vol. 40, No. 12 (1999).
14) O'Connell, F. P. and White, S. W.: *POWER3: The next generation of PowerPC processors*, IBM Corp (2000).