プログラムフェーズを考慮した SMTプロセッサ上でのスレッドスケジューリング手法

市川 雄二郎† 有次 正義‡

†群馬大学大学院工学研究科情報工学専攻 〒376-8515 群馬県桐生市天神町 1-5-1
‡群馬大学工学部情報工学科 〒376-8515 群馬県桐生市天神町 1-5-1
E-mail: †ichikawa@dbms.cs.gunma-u.ac.jp, ‡aritsugi@cs.gunma-u.ac.jp

あらまし 複数のスレッドコンテキストから同時にインストラクションを発行し，実行できる，SMT プロセッサでは，同時実行されるスレッドの組合せがシステム利用効率に影響する．従来研究では，より適切なスケジュールを探索するためにスケジューリングを２つのフェーズに分けた．1 つ目は，いくつかのスケジュールを評価するための情報を集める，2 つ目は，得られたサンプリング結果からその時点において最適と予期されるスケジュールを決定するフェーズである．本稿では，各ジョブの振る舞いを表すプログラムフェーズを用いることで 1 つ目のフェーズにおけるサンプリング処理を効率化し，スレッドの実行パフォーマンスが向上することを示す．
キーワード SMT，スレッドスケジューリング

A Thread Scheduling Method for SMT Processor considering Program Phase

Yujiro ICHIKAWA† and Masayoshi ARITSUGI‡

†Department of Computer Science, Graduate School of Engineering, Gunma University
1-5-1 Tenjin-cho, Kiryu, Gunma, 376-8515 Japan
‡Department of Computer Science, Faculty of Engineering Gunma University
1-5-1 Tenjin-cho, Kiryu, Gunma, 376-8515 Japan
E-mail: †ichikawa@dbms.cs.gunma-u.ac.jp, ‡aritsugi@cs.gunma-u.ac.jp

Abstract How to combine threads to be executed simultaneously must effect the performance of SMT processor systems in which multiple instructions can be issued from multiple thread contexts and executed. A conventional research divides a scheduling method into two phases for generate a good schedule: one is to collect information in order to evaluate candidate schedules, and the other is to decide the best schedule among them at that moment. In this paper, we extend the former phase to make it efficiency by exploiting program phase, which represents the behavior of each job. We also report some simulation results showing that our proposal can improve the performance of an SMT processor system.
Keyword SMT, Thread Scheduling

1. はじめに

Simultaneous Multithreading（SMT）[1,2]は，各サイクルで複数の独立するジョブの実行単位，すなわち，スレッドから同時にインストラクションを発行し，スーパースカラプロセッサに実行させることを可能にした技術である．SMT プロセッサを用いたシステムでは，スレッドレベル並列性（TLP）が命令レベル並列性（ILP）に転換され，命令実行スループットが増大し各スレッドの実行速度が向上する．

図 1 に，複数のスレッドが存在する状況での SMT および既存アーキテクチャの振る舞いを示す．この図では，システム上に実行可能スレッドが 5 つ存在し，SMT プロセッサが同時実行をサポートする数，すなわち，ハードウェアコンテキスト数が 4 つある場合での，サイクルが進むにつれて各スレッドからインストラクションが発行されることでスレッドの同時実行が行なわれる様子を示している．このような，実行可能スレッドがハードウェアコンテキスト数を超える状況では，スレッドスケジューラは，ハードウェアコンテキストの数だけを実行可能スレッドから選択し，選択されたスレッドにインストラクションを発行させることで同時実行を行なっている．SMT プロセッサ上で同時実行を行なう場合，ハードウェアリソースが選択されたスレッド集合間で共有されるため，実行スレッドの組合

せによってシステムの利用効率に違いが生じることになる．そのため，SMT システムではスケジューラが適切な実行スレッドの集合を決定できるかが重要となる．

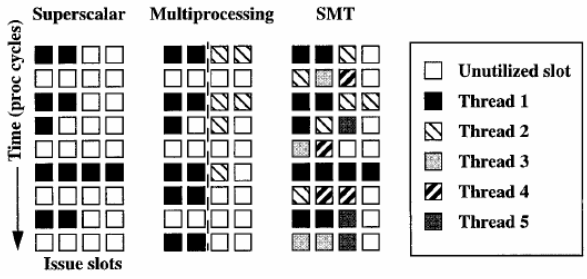

図 1　複数の実行スレッドが存在する環境での
各アーキテクチャの振る舞い[1]

Fig1. Behavior of each architecture in an environment where two or more execution threads exist [1]

このような課題に対し，従来研究ではあらかじめいくつかの候補スケジュールを比較するための情報をサンプリングすることで，より適切なスケジュールの探索を行い，結果として応答速度の向上に繋がったことを示している[3]．しかし，この研究ではサンプリングに必要となるサイクル数が過多にならないようサンプル数を少ない数に抑えている．そのため，実行スレッドの組合せが多くなる環境では，サンプル数が候補スケジュール数に比べてわずかなものとなる．もし，比較する候補スケジュール数を全候補数に近づけたならば，より適切なスケジュールを探索することができるといえる．本研究では，サンプリングに必要とするサイクル数を増加せずに，これを行う方法について考える．

本研究では，従来手法のスケジューリングにプログラムフェーズを用いて，比較する候補スケジュールの数を増加し，より性能の高いスケジュールの実行をねらうことで，スレッドの実行パフォーマンスを向上することを目的とする．プログラムフェーズとは，等しい命令間隔で各ジョブの振る舞いを分類したものである．プログラムフェーズが同一の場合は，該当する範囲間でその性能が類似すると考えられている[5]．この性質を利用し，提案手法では，サンプリング結果をシステム上に保持し以降のスケジューリングで使用する．

本手法の特徴として，比較する候補スケジュールの数の増加が挙げられる．従来手法では，ある時点における適切なスケジュールの探索は，その直前に得られた比較対象の候補スケジュールのサンプリング結果に基づいて行われる．それに対し，本手法ではプログラムフェーズの性質により過去に取得した結果も利用することができる．スケジューラは，サンプリング処理の開始時に，システムが保持する過去のサンプリング結果を参照する．もし，開始されるサンプリング処理に関するプログラムフェーズが，ある過去の結果に関するものと同一であるならば，これから行われるサンプリング処理の結果もそれと類似すると考えることができその処理を省略することができる．同一のものが存在しない場合は，サンプリング処理を行いその結果をシステムに格納する．省略された処理の数が増加すれば，サンプリングに用いるサイクル数を増やさずに，従来と比べて多くの候補スケジュールを比較することができる．これにより，スケジューラがより完全に適切なスケジュールを探索することができ，各スレッドのシステム利用効率が向上することが期待できる．実験では，従来手法と同一のサンプリングに使用するサイクル数で，平均の候補スケジュール比較数が増大したことが確認できた．

以下，2 章で関連研究について述べ，3 章で提案手法について説明する．4 章で提案手法の性能評価を行ない，最後に 5 章で本研究のまとめを行なう．

2. 関連研究

Symbiotic Jobscheduling [3]は，SMT プロセッサで発生する実行スレッド間でのハードウェアリソースに対する相互作用に着目したスケジューリング手法である．この研究では，システム上にハードウェアコンテキストの数を超えるスレッドが存在し，それぞれが公平にサイクルを割り当てられ実行される環境を想定している．また，スケジューリングを 2 つのフェーズに分けることでより適切な候補スケジュールを探索しジョブ実行効率を向上させている．

候補スケジュールは，タイムスライスが複数集まることで構成される．タイムスライスは，ハードウェアコンテキストの数だけスレッド識別子を持ち，ある時点においてどのスレッドを同時実行するかを示す．タイムスライスが開始されるとき，それが持つ識別子に該当するスレッドにインストラクションを発行させることでスレッドの同時実行が行なわれる．同時実行の対象であるスレッドをアクティブスレッド，それ以外のものをノンアクティブスレッドと呼ぶ．候補スケジュールの生成は，スレッド数，ハードウェアコンテキスト数，また，タイムスライス終了時にアクティブ・ノンアクティブを切り替えるスレッド数，というパラメータで候補スケジュールの生成が行なわれる．

図 2 に，スレッド数を 8，ハードウェアコンテキスト数を 4，切り替えスレッド数を 1 としたときの候補スケジュールの例と SMT 上での実行の様子を示す．まず，候補スケジュールの生成は，それがいくつのタイムスライスを持つか決定することから始まる．図 2 では，最も左の初期タイムスライスがスレッド識別子，0，

2，3，5，を持つ．これらは，8 個のスレッドの識別子 0～7 よりランダムに 4 つ選択したものである．次のタイムスライスでは，選択されているスレッド識別子を，切り替えスレッド数の数，ここでは 1 つだけを選択されていないスレッド識別子と入れ替える．図 2 では，2 番目のタイムスライスで，スレッド識別子 3 の代わりに 4 が含まれている．このようにして，各スレッドがアクティブになる回数が同一になるまでタイムスライスを増加する．

ある候補スケジュール内で各スレッドがアクティブになる回数を同じにするのは，システムが各スレッドに対して公平に同時実行のためのサイクルを割り当てることを考えなければならないからである．候補スケジュールに与えられるサイクルは，それを構成するタイムスライス間で等しく分割される．図 2 では，候補スケジュールに n サイクル与えられたとき，各タイムスライスは 1n/8 サイクルの間だけ開始されることが示されている．また，ここでは各スレッドが 4 回ずつアクティブになる．よって，それぞれが 4*1n/8 サイクルだけ同時実行のためのサイクルが与えられることになる．また，候補スケジュールは複数存在する．候補スケジュールは，各スレッドがアクティブになるタイミングによって異なる．例えば，図 2 で示した候補スケジュールは，初期タイムスライスのアクティブスレッドが，0，2，3，5，であるが，別の候補スケジュールでは，これが，1，2，4，7，になる場合もある．

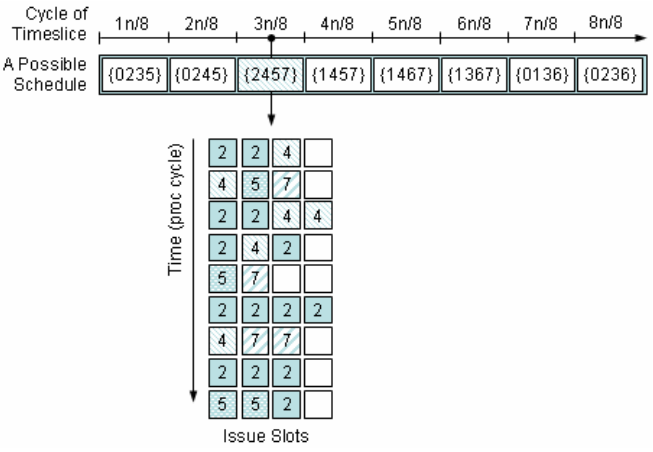

図 2　候補スケジュールの例と SMT 上での実行
Fig2. Example of a possible schedule and execution on SMT

タイムスライスでは，アクティブスレッドがハードウェアリソースを競合する可能性がある．競合の度合いは，同時実行される各スレッドの動作よって変化する．例えば，ある実行スレッドがメモリアクセス待ち状態で SMT プロセッサに対しインストラクションを発行しないとき，その分別の実行スレッドがハードウェアリソースを利用することができる．従って，候補スケジュール間でシステムの利用効率に違いが生じることになる．

スケジューリングは Sample Phase，Symbiosis Phase という 2 つのフェーズで行なわれる[3]．これらは，各自設定されたサイクル数に基づいて順番に切り替えていく．初期状態は Sample Phase であり，ここではスケジューラがより適切なスケジュールを探索するための情報を集める．まず，候補スケジュールをいくつかランダムで選択する．選び出された候補スケジュールはサンプルスケジュールとして扱い，それぞれ順番に同じサイクル数を割り当てタイムスライス内のスレッドを実行する．その際，スケジューラは CPU のハードウェアパフォーマンスカウンタより候補スケジュールを比較するための情報を集める．これは，IPC やキャッシュミス率を計算することで行なわれる．Sample Phase 終了後，スケジューラは Symbiosis Phase に切り替わる．Symbiosis Phase では，直前に得られたサンプルスケジュールの内，最も良い結果を出したものをその時点における最適なスケジュールとして考え，タイムスライス内のスレッドを実行する．

Symbiotic Jobscheduling[3]によって応答速度の向上が示されている，しかし，サンプリングに使用するサイクル数が過多にならないよう，サンプル数を少ない数に抑えている．[3]では，候補スケジュール数が 2520 に対してサンプル数が 10 である実験が行なわれていた．この場合，サンプリングによって判断したスケジュールが全スケジュールの中でどのくらい適切であるかが疑問となる．本研究では，この問題に着目し，Symbiotic Jobscheduling[3]を従来手法とすることで，新しい SMT 上でのスレッドスケジューリング手法を提案する．

[4]では，[5]で示されているプログラムフェーズを利用して，SMT 上で 2 つのプログラムを動かした場合での性能評価を行う研究を行っている．この研究では，プログラムフェーズの性質を利用することで，プログラム同時実行を終始行うことなく，同時実行全体の性能を計測する．[4]と本研究の違いは，まず，本研究ではハードウェアコンテキストの数を超えるスレッドが存在する環境を想定している．そして，同時実行全体の性能を計測することではなく，多数の実行スレッドの組合せが存在する状況で適切なスケジューリングを行うためにプログラムフェーズを用いることを考える点である．

3. プログラムフェーズを考慮したスレッドスケジューリング

本研究では，従来手法[3]のスケジューリングにプログラムフェーズを用いて，比較する候補スケジュールの数を増加し,より性能の高いスケジュールの実行をねらうことで，スレッドの実行パフォーマンスを向上させることを目的とする．ここでは，まず，プログラムフェーズを用いた場合の特徴を述べる．続いて，2で説明した従来手法を拡張し，プログラムフェーズを考慮したスレッドスケジューラの実装に関して具体的に説明する．

3.1. プログラムフェーズ

プログラムフェーズとは，あるジョブを等しい命令間隔で分割し，それぞれをその振る舞いに関して分類したものである.プログラムフェーズが同一の場合は，該当する範囲間でその性能が類似すると考えられている[5]．各ジョブのプログラムフェーズは，SimPoint[6]によって得ることができる．

プログラムフェーズの性質を利用することで，スケジューラは過去に得られたサンプリング結果を利用することができる．つまり，あるタイムスライスのサンプリング開始時に，システムが保持する過去のサンプリング結果を参照し可能ならばサンプリング処理を省略することができる．その結果，従来手法[3]に比べ多くの候補スケジュールに対して比較を行うことができる．

サンプリング処理を省略できるかどうかは，スケジューラが実行中のタイムスライスに関するプログラムフェーズ，すなわち，そのタイムスライスを構成するスレッドの ID とプログラムフェーズの集合と同一のものがシステム上に保持されているかで判断する．このために，本研究では，システム上にサンプリング結果を保持する仕組みを用意し，スケジューラは必要に応じてこれにアクセスする処理を追加することを考える．具体的には，TaskPhaseTable，CoPhaseTable という 2 つのテーブルを追加し，スケジューラがこれらを利用して目的を達成するよう処理を拡張する．

3.2. スレッドスケジューラの実装

本手法では，スケジューラが過去に取得したサンプリング結果を利用するための仕組みとして，まず，TaskPhaseTable, CoPhaseTable という 2 つのテーブルを追加する．図 3，図 4 に各テーブルの例を示す．

TaskPhaseTable は，SimPoint[6]によって得られたあるジョブに関するプログラムフェーズ識別子を保持するものである．本研究ではその識別子として整数を使用した．TaskPhaseTable の各レコード間には等しい命令間隔があり，スケジューラは，あるスレッドが現在のプログラムフェーズ識別子を取得する．図 3 では，あるジョブが 10M のインストラクション間隔で分割され，各間隔が整数値，例えば，0～10M の範囲ではプログラムフェーズ識別子が 10，を持つことが示されている．そして，TaskPhaseTable を作成する際は，ジョブ開始方向から終了方向に向けて，各インストラクション間隔のプログラムフェーズ識別子を格納していく．スケジューラが，TaskPhaseTable よりスレッドの現在のフェーズを取得するときは，そのスレッドが実行したインストラクション数を引数にする．図 3 を例に取ると，スレッドが 11M インストラクション実行したならば，11/10=1.1≦2 より TaskPhaseTable の 2 番目のレコードに格納されているプログラムフェーズ識別子 8 が得られる．

CoPhaseTable は，サンプリング処理が行われたときの，タイムスライスが持つスレッド識別子と，処理が開始された時点で TaskPhaseTable より取得した各スレッドのプログラムフェーズ識別子からなる CoPhase エントリと，サンプリング終了時に計算する性能結果 SCORE で構成される．図 4 を例に取ると，ある候補スケジュールの，左より 3 番目のタイムスライスに関してサンプリング処理が行われるとき，まず，スレッド識別子，2，4，5，7，のスレッドそれぞれに関して TaskPhaseTable よりプログラムフェーズ識別子を取得する．ここでは，得られたプログラムフェーズ識別子をそれぞれ，16，6，14，3，とする．これら 2 種類の識別子により，このタイムスライスに関する CoPhase エントリは，

{{ThreadID}{ProgramPhaseID}}
= {{2,4,5,7}{16,6,14,3}}

となる．そして，サンプリング終了時に計算した性能結果，例えば，IPC=2.13，を開始時に作成した CoPhase エントリと共に CoPhaseTable に格納する．

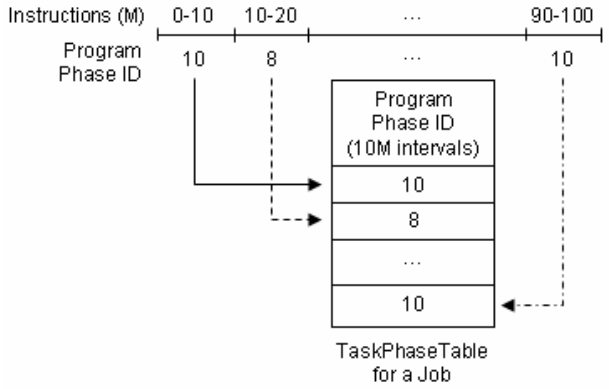

図 3　TaskPhaseTable の例
Fig3. Example of a TaskPhaseTable

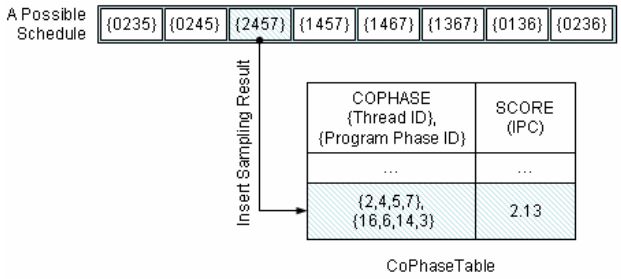

図 4　CoPhaseTable へのサンプリング結果の格納
Fig4. Example of inserting a sampling result into CoPhaseTable

本研究では，従来手法[3]のスケジューリング処理に対し，上の 2 つのテーブルにアクセスする処理を追加し，拡張する．図 5，図 6 に，本手法におけるスケジューリング処理の実行の流れを示す．

Sample Phase では，あるタイムスライスのサンプリング開始時に，まずそれを構成するスレッド ID およびプログラムフェーズの集合から CoPhase エントリを生成し，それと同一のものが CoPhaseTable 内に存在するか調べる．同一のエントリが存在した場合，スケジューラは，そのレコードに含まれるサンプリング結果値を，開始したタイムスライスの結果値として扱い以降のサンプリング処理を省略する．エントリが存在しない場合は，従来通りそのタイムスライスのサンプリング処理を行う．そして，得られた結果を開始時に作成した CoPhase エントリと共に CoPhaseTable に格納する．このとき，CoPhaseTable に設定されている最大レコード数を上回る場合は，既存のレコードを上書きする．本実験では，最後に参照されたレコードを上書きする方式で更新することとした．

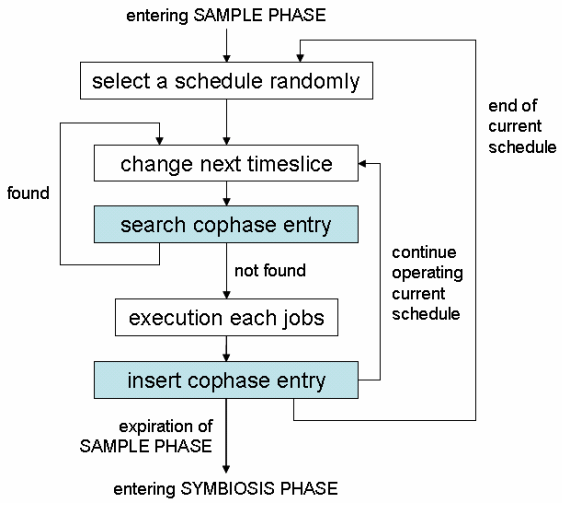

図 5　Sample Phase の実行の流れ
Fig5. Flow of execution in Sample Phase

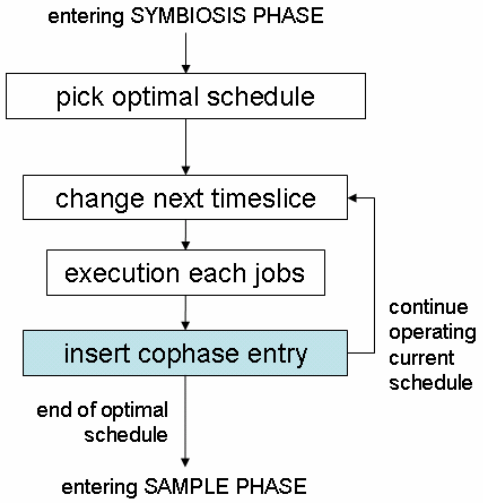

図 6　Symbiosis Phase の実行の流れ
Fig6. Flow of execution in Symbiosis Phase

従来手法に対し，本手法では，Symbiosis Phase でもスケジュールの性能に関する情報の収集を行なう．これは，Sample Phase で得られたサンプリング結果値と Symbiosis Phase における実測値の違いを考慮するためである．従来手法では，両方の値に違いが見られても以降のスケジューリングにそれが活かされることはなかった．それに対し，本手法では Symbiosis Phase で取得したサンプリング結果値を CoPhaseTable に反映することができる．Symbiosis Phase において取得したサンプリング結果は，CoPhaseTable 内に同一の CoPhase エントリが存在する場合でも結果値を上書きさせる．これにより，より精度の高い適切な候補スケジュールの探索を行うことができると考えられる．

4. 実験

4.1. 実験環境

本実験では，システム上にハードウェアコンテキスト数を超えるジョブが存在する状況における複数のスケジューリング手法の比較実験を行った．実験対象となるスケジューリング手法には，本手法，従来手法[3]，およびラウンドロビンスケジューリングを選択した．ラウンドロビンでは，スケジューラはサンプリング処理を行わず，一定のサイクル間隔でランダムに候補スケジュールを 1 つ選択し各ジョブを実行する．このサイクル間隔は，他の手法における Symbiosis Phase で使用するサイクル数と同一とする．

本実験を行った環境について説明する．SMT プロセッサのシミュレータとして SMTSIM[7,8]を選択し，実行ジョブには SPEC ベンチマーク[9]より，crafty, equake, gcc-166, gcc-200, gcc-expr, gzip-log, gzip-graphic, gzip-source，を使用した．また，実験を行う上で設定

する必要があるパラメータがいくつか存在する．まず，スケジュールを生成するための，ジョブ数，ハードウェアコンテキスト数，および，タイムスライス終了時にアクティブ・ノンアクティブを切り替えるスレッド数である．また，Sample Phase，Sample Phase で取得するサンプルの最小数，Symbiosis Phase で使用するサイクル数，TaskPhaseTable の各レコード間にある命令間隔，そして CoPhaseTable の最大レコード数が挙げられる．表 1 に今回設定したパラメータとその値を示す．

実験では，各スケジューリング手法を比較するための指標として IPC を選択した．そして，各スケジューリング手法のスレッド実行パフォーマンスを求めた．スレッド実行パフォーマンスは，以下の式によって計算する．

$$\text{Performance of } Schedule_t = \sum_{i=1}^{n} \text{IPC of } timeslice_i \quad (1)$$

この式におけるスケジュールは，本手法および従来手法[3]では Symbiosis Phase で選択された候補スケジュール，ラウンドロビンスケジューリングではランダムに選択された候補スケジュールとなる．

また，本手法の性能を確認するために，Sample Phase で比較した候補スケジュールの数と，他の手法に対する性能の増加率を次の式を用いて求めた．

$$\text{Performance Improvement rate of } Schedule_t = \frac{\text{Performance of } Schedule_t \text{ in Our Method}}{\text{Performance of } Schedule_t \text{ in Other Method}} \quad (2)$$

表 1　本実験で設定したパラメータ

Table1. Parameters set by our experiment

Number of Jobs	8
Multithreading	4
Number of Swapped Threads	1
(Possible Schedules)	(2520)
Number of Samples	10
CYCLE of Sample Phase	100000
CYCLE of Symbiosis Phase	1000000
Instruction Interval of TaskPhaseTable	10000000
Max Records of CoPhaseTable	5000000

4.2. 実験環境

本研究の目的は，従来手法のスケジューリングにプログラムフェーズを用いることで，比較する候補スケジュールの数を増加し，より性能の高いスケジュールの実行をねらうことでスレッドの実行パフォーマンスを向上させることである．それを確認するために 4.1 で説明した実験の結果を示す．

まず，図 7 に本手法における各 Sample Phase で比較した候補スケジュールの数を示す．図 7 では確認できないが，実験開始直後ではシステムが保持するサンプリング結果が少ないため，比較した候補スケジュールが 15 前後といった数であった．この数は，スケジューラが，サンプリング結果を CoPhaseTable に格納するにつれて増加し，実験終了時まで従来手法より多くの候補スケジュールを比較したことが分かる．

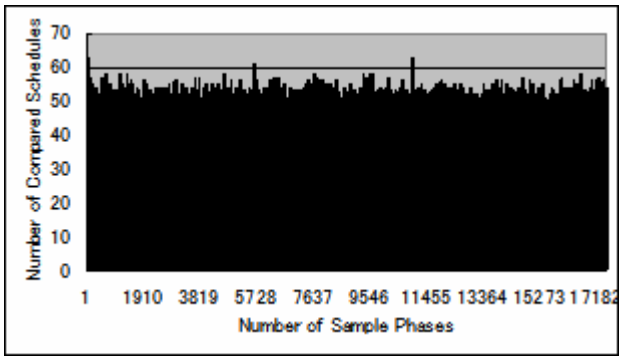

図 7　Sample Phase で比較した候補スケジュールの数

Fig7. Number of possible schedules that compared in Sample Phase

続いて，各スケジューリング手法のスレッド実行のパフォーマンスを図 8，図 9，図 10 に示す．ラウンドロビン形式では，スケジュールの実行パフォーマンスにばらつきが見られた．これは，ランダムに選択された候補スケジュールの中に適切でないものが含まれることで，実行スレッド間でのハードウェアリソース競合の度合いが高くなる状況が発生したからだと考えられる．それに対して，本手法および従来手法[3]は，スケジュールの実行パフォーマンスが安定する結果となった．これにより，スケジューラはサンプリングを行うことでスレッドに安定したシステム利用効率を提供することができるといえる．

最後に，図 11，図 12 で，ラウンドロビンスケジューリング，および従来手法に対するに本手法の性能増加率を示す．本手法では，ラウンドロビンスケジューリングに対しては大幅にスケジュール実行パフォーマンスを伸ばすことができた．従来手法に対しても，実行したスケジュールの 50.7%が 1 を超える性能増加率

を示し，その内 12.5%が 1.3 を超える結果となった，また，0.7 を下回る性能増加率を示したものは全体の 6.32%であった．以上より，本手法は，従来手法に対して，スレッド実行パフォーマンスを向上させることができるといえる．

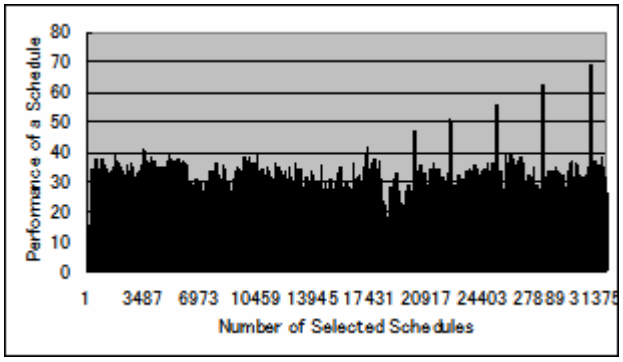

図 8　ラウンドロビンスケジューリングの
スレッド実行パフォーマンス
Fig8. Execution performance of threads
for Round-Robin Scheduling

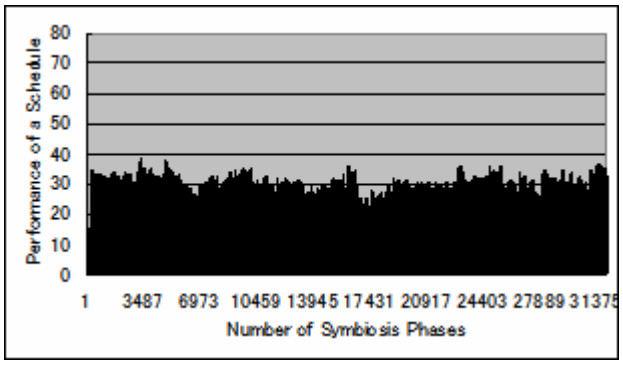

図 9　従来手法のスレッド実行パフォーマンス
Fig9. Execution performance of threads
for the conventional method

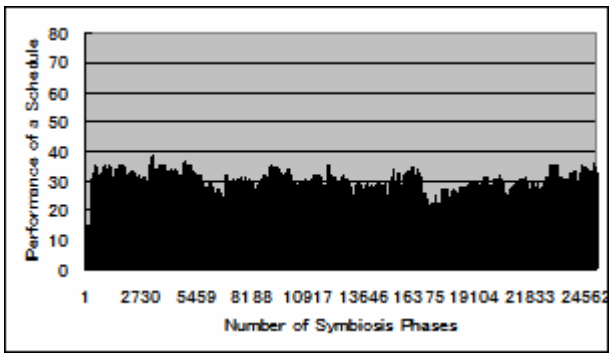

図 10　本手法のスレッド実行パフォーマンス
Fig10. Execution performance of threads for our method

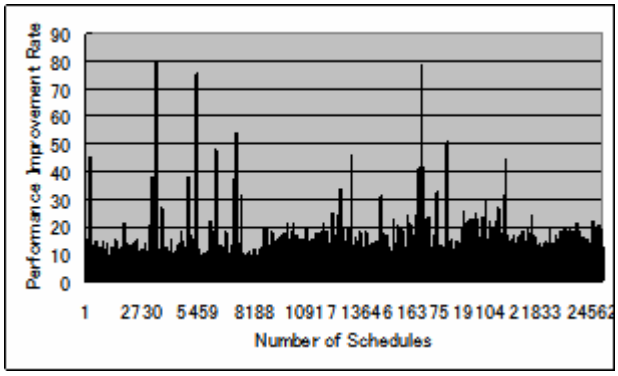

図 11 ラウンドロビンスケジューリングに対する
本手法の性能増加率
Fig11. Performance improvement rate of our method
against Round-Robin Scheduling

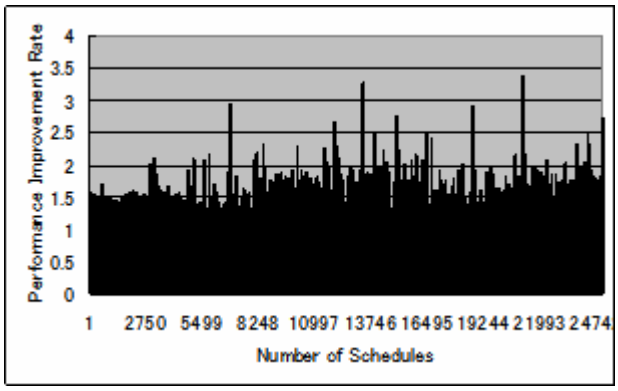

図 12　従来手法に対する本手法の性能増加率
Fig12. Performance improvement rate of our method
against the conventional method

5. まとめ

本稿では，比較する候補スケジュールの数を増加し，より性能の高いスケジュールの実行をねらうことで，スレッドの実行パフォーマンスを向上させることを目的とし，従来手法にプログラムフェーズを用いたスレッドスケジューリング手法を提案した．本手法では，TaskPhaseTable, CoPhaseTable という 2 つのテーブルを追加し，スケジューラがこれらにアクセスするよう処理を拡張する．スケジューラは，サンプリング処理の開始時に，システムが保持する過去のサンプリング結果を参照する．もし，開始されるサンプリング処理に関するプログラムフェーズが，ある過去の結果に関するものと同一であるならば，これから行われるサンプリング処理の結果もそれと類似すると考えることができその処理を省略することができる．実験によって，本手法は，比較する候補スケジュールの数が増え，他の手法に対してスケジュール実行性能の増加したことが確認できた．

今後の課題として，使用したジョブの性質と候補ス

ケジュールを比較するための指標との関係に着目し，本手法の性能評価を行なうことが挙げられる．今回は，指標としてIPCを選択したが，もし使用するジョブがデータ集中的なものであるならば，IPCではなくキャッシュミス率で候補スケジュールの比較を行なった方が，スケジューラが判断した適切なスケジュールによるスレッド実行パフォーマンスが高くなるのではないかと考えられる．

謝　辞

本研究の一部は，科学研究費補助金基盤研究(C)(18500073)により行なわれた．

文　献

[1] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In ISCA96, pages 191-202, May 1996.

[2] J. L. Lo, S. J. Eggers, J. S. Emer, H. M. Levy, R. L. Stamm, and D. Tullsen. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. ACM Transactions on Computer Systems, Volume 15, Issue 3, Pages 322 - 354, Aug. 1997.

[3] A. Snavely and D. Tullsen. Symbiotic jobscheduling for a simultaneously multithreading processor. In Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, Pages 234 - 244, Nov. 2000.

[4] M. V. Biesbrouck, T. Sherwood, B. Calder. A Co-Phase Matrix to Guide Simulation Multithreading Simulation. In IEEE International Symposium on Performance Analysis of Systems and Software, Pages 45 - 56, March 2004.

[5] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In 10th International Conference on Architectural Support for Programming, Pages 45 - 57, October 2002.

[6] G. Harmerly, E. Perelman, J. Lau, B. Calder. SimPoint 3.0: Faster and More Flexible Program Analysis.

[7] SMTSIM, <http://www-cse.ucsd.edu/~tullsen/smtsim.html>

[8] D. M. Tullsen, S. Eggers, and H. Levy. Simulation and modeling of a simultaneous multithreading processor. In 22nd Annual Computer Measurement Group Conference, Dec. 1996.

[9] SPEC Benchmarks, <http://www.spec.org/>