Vol.2013-ARC-203 No.14
2013/2/1

コンパイラと協調したシミュレーション精度切り換え可能なマルチコアアーキテクチャシミュレータ

田口学豊[†1]　阿部洋一[†1]　木村啓二[†1]　笠原博徳[†1]

概要：本稿では，コンパイラと協調してシミュレーション精度を相互に切り替えることができるマルチコアアーキテクチャシミュレータによってシミュレーション速度を高速化する枠組みを提案する．本提案では，コンパイラを介して，対象プログラムにおける詳細シミュレーションを行うサンプリング量の決定や，並列化プログラムに対する精度切り換えコードの自動生成を行う．本手法を SPEC CPU 2000 の EQUAKE に適用したところ，誤差 1.6 パーセント以内で 50 倍～500 倍の高速化が可能であることを示した．

A Parallelizing Compiler Cooperative Multicore Architecture Simulator with Changeover Mechanism of Simulation Modes

GAKUHO TAGUCHI[†1]　YOUICHI ABE[†1]
KEIJI KIMURA[†1]　HIRONORI KASAHARA[†1]

A parallelizing compiler cooperative multicore architecture simulation framework, which enables reducing simulation time by a flexible simulation-mode changeover mechanism, is proposed. A multicore architecture simulator in this framework has two modes; namely, functional-and-fast simulation mode and cycle-accurate-and-slow simulation modes. This framework generates appropriate sampling points for cycle-accurate mode and runtime for mode changeover of the simulator depending on a parallelized application by cooperating with a parallelizing compiler. The proposed framework is evaluated with EQUAKE from SPEC2000. The evaluation result shows 50 times to 500 times speedup can be achieved within 1.6% error.

1. はじめに

アーキテクチャシミュレータは，コンピュータアーキテクチャの開発の補助や実機製作のコスト削減などにおいて非常に大きな役割を担っている．しかしながらソフトウェア上でのシミュレーションであるアーキテクチャシミュレータの動作は，実機に比べて約 5000～10000 倍の多大な時間を要する．特に複数のプロセッサコアを有するマルチコアアーキテクチャシミュレーションを行う際に，このシミュレーション時間の増大がさらに顕著になり，現在のコンピュータアーキテクチャ研究の大きな障害となっている．

そのためこのような課題を克服するために，高速で精度の高いシミュレーション手法についての研究が進められている．このような研究の，プログラムの一部分のみ詳細なシミュレーションを行うサンプリング実行により高速化を図る手法として，SimFlex[1)]と SimPoint[2)]が挙げられる．

一方，筆者等はこれまでに，シミュレーションに用いる並列化プログラムの構造に注目する手法を提案してきた[3,4)]．すなわち，並列化可能ループ及び並列化対象部分を囲むループのイタレーションの一部を，実機上の逐次実行プロファイルに基づいた統計的手法を用いて期待する誤差に収まる範囲で詳細実行するというものである．

本手法では，まず並列化前の逐次プログラムを任意の実機上で実行し，サンプリング対象となるループの 1 イタレーション毎の実行サイクル数を計測する．計測したイタレーション毎のコストから統計的手法により，総実行サイクルの推定値が期待する誤差に収まる，最小のイタレーション数（サンプリングサイズ）及びサンプリングするイタレーションを算出する．その後，算出したイタレーション回数だけサンプリング対象のループを詳細にシミュレーションし，その他のイタレーションは命令実行のみの簡易で高速なシミュレーションを行うことで，実行結果の確認を行う．この時，イタレーション毎の実行サイクル数の挙動が大きく異なる部分がある場合，コストが同程度のイタレーションをクラスタリングして、各クラスタに対してサンプリングを行うため，シミュレーションを行う際にはシミュレーション精度を任意のポイントで切り換えながら実行する必要がある．

本稿ではプログラム中のサンプリング対象となる箇所といったヒント情報を予めプログラムに与えることで，コンパイラと協調してサンプリングサイズと制度切り替えタイミングの算出を行い，並列化されたプログラムを算出されたタイミングに従ってシミュレーション精度を切り替えながらシミュレーションを行うことができるフレームワークを提案する．

以下，2 章ではシミュレーション高速化手法について，3 章ではシミュレーション精度切り換え機能について，4 章では評価結果，最後の 5 章でまとめをそれぞれ述べる．

†1 早稲田大学
WASEDA UNIVERSITY

2. シミュレーション高速化手法

本章では，本稿で提案するシミュレーション高速化手法と，それを実現するコンパイラと協調したシミュレーションのフレームワークについて述べる．

2.1 実行サイクル数推定手法 [3,4)]

本手法では並列化可能ループ及び並列化対象部分を内包するループに注目する．このループに対して全イタレーション数のうち一部を詳細にシミュレーションし，残りを高速かつ簡易なシミュレーションを行うことによって高速化を図る．そして，一部の詳細なシミュレーションにより得られた実行サイクル数から，ループ全体の実行サイクル数を推定する．

この時，詳細にシミュレーションするイタレーション数を決める必要がある．本章ではまず，ループのイタレーション毎のサイクル数の変化の挙動はプログラム，及び入力依存であり，命令セットも含めたアーキテクチャの違いに依る差異は小さいという前提をおく．その上で，並列化前の逐次プログラムのサンプリング対象ループに 1 イタレーション毎の実行サイクル数を計測するコードを挿入し，任意の実機上で実行する．実機で取得した実行サイクル数より統計的手法を利用し，期待する誤差の範囲で全実行サイクル数が推定可能な詳細シミュレーションを行うべきサンプル数を決める．この算出には，取得したイタレーション毎の実行サイクル数の標準偏差と平均値，標準偏差の上側 P%点，許容する誤差（信頼度）を用いる．このうち，標準正規分布の上側 P%点は，P=2.5 の時の 1.96，許容する誤差は 5%，すなわち信頼度 0.05 として以下の式で計算を行う．

$$\text{サンプル回数} \geq \left(\frac{\text{上側 P\%点}}{\text{信頼度}} \times \frac{\text{標準偏差}}{\text{平均値}}\right)^2 \qquad (1)$$

実機実行で得たプロファイルにおいて，イタレーション毎のサイクル数の挙動が大きく異なる範囲がある場合は，コストが同程度のイタレーションをクラスタリングし，各クラスタに対してサンプル回数を決定する．

このように決定したサンプルの回数だけ詳細にシミュレーションを行い，得られた実行サイクル数から，以下の式を用いて，全実行サイクル数を推定する．

$$\text{推定全実行サイクル数} = \text{詳細シミュレーションサイクル数} \times \frac{\text{全イタレーション回数}}{\text{サンプル数}} \qquad (2)$$

2.2 シミュレーションモード

サンプリング実行には，詳細シミュレーションと機能シミュレーションの，2 種類のシミュレーションモードを利用する．それぞれのシミュレーションモードについて解説する．

- 詳細シミュレーション

キャッシュやパイプライン及び相互接続網といったアーキテクチャ構造を詳細に再現する一方，処理時間が非常に長い．サンプル部分のみ実行する．

- 機能シミュレーション

アーキテクチャ構造の再現はせず，命令実行のみのシミュレーションを行う．一方，詳細シミュレーションに比べて 100～130 倍ほど処理時間が短い．サンプル部分以外で実行する．

2.3 サンプリング実行のコンパイルフロー

本節では，並列プログラムをコンパイラと協調してサンプリング実行するフレームワークについて述べる．

サンプリング実行は，サーバー評価から詳細シミュレーションすべきイタレーション数（サンプル数）と精度切り替えタイミングを決定する第一段階と，算出されたサンプル数を基に，実際に精度切り換えシミュレーションを行う第二段階の，二つの手順を踏む必要がある．これら一連の流れのフロー図を図 1 に示す．

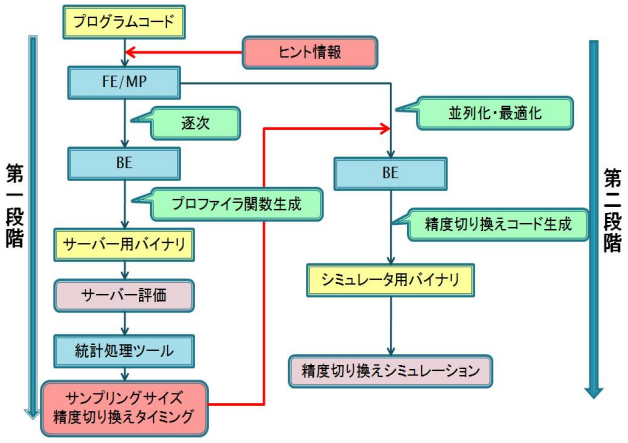

図 1　コンパイラと協調したサンプリング実行のフレームワーク

Figure 1 A compilation flow of the proposing sampling based architecture simulation

ここで，コンパイラは字句解析及び構文解析を行うフロントエンド（FE），並列化等の最適化及びヒント情報の処理を行うミドルパス（MP），及びターゲットアーキテクチャ用コード生成を行うバックエンド（BE）から構成される．また，本稿では並列化コンパイラとして OSCAR 自動並列化コンパイラを用いた [5)]．

まず，元プログラムにサンプリング実行するループの箇所といったヒント情報を与える．第一段階では逐次コンパイルを行うが，このときコンパイラによってヒント情報を基に実機上で 1 イタレーション毎の実行サイクル数を計測するプロファイラ関数が挿入される．その後，実機でプログラムを実行し，取得したプロファイル結果を統計処理ツールに通すことで，詳細シミュレーション，機能シミュレ

ーションのそれぞれ行うべきイタレーション数（サンプリングサイズ）を算出する．ここで，プロファイル結果において 1 イタレーション毎の実行サイクル数の挙動が大きく異なっている場合，コストが同程度のイタレーションにクラスタリングし，各クラスタでサンプリングサイズを算出する．そのため，それぞれループのイタレーションの何回転目で精度切り替えを行うかといった精度切り替えタイミングの情報もここで出力する．また，サンプリング対象ループは並列化可能ループ及び並列化可能部分を囲むループであるため，ここで得られたサンプリングサイズはコア数，キャッシュサイズ，及びキャッシュ最適化の有無に変化があっても適用可能なことが分かっている [3]．

次に第二段階では，同様のヒント情報が与えられたプログラムをコンパイラにより並列化する．同時にヒント情報を基に対象ループに精度切り換えコードが挿入される．その後，第一段階で得たサンプリングサイズと精度切り替えタイミングをフィードバックし，精度切り換えシミュレーションを行うことでサンプリング実行を実現する．

3. 精度切り換え機能

本稿で提案するシミュレーション高速化手法では，プログラム全体実行時間の推定誤差を最小化するような任意のイタレーションがサンプリング対象となる．そのため，実際のシミュレーションではループのイタレーション全体を通して詳細シミュレーションと機能シミュレーションを任意のタイミングで相互に切り替えながら実行する必要がある．

本章では，詳細シミュレーション，機能シミュレーションの 2 つのシミュレーションモードを相互に切り換える精度切り換え機能について述べる．

3.1 精度切り換え機能の概要

本手法におけるサンプリング実行において，サンプリングはプログラムのループ構造に着目し，ループのイタレーション単位で行われる．よって，精度切り換えも同様にループのイタレーション単位で行う．精度切り換えの様子を図 2 に示す．

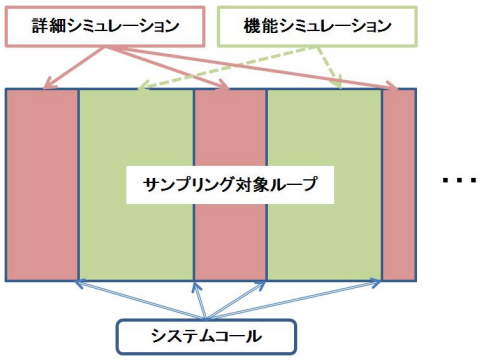

図 2　精度切り換えの様子

Figure 2　An image of simulation-mode changeover

図 2 のように，シミュレーション精度の切り替えはシステムコールを呼び出すことで行う．具体的には，シミュレータの想定する OS に精度切り替えのためのシステムコールを新設し，インラインアセンブラでシステムコールを呼び出すコードを直接プログラム中に挿入することにより，シミュレーション精度を切り替える．システムコールに相当する処理シミュレータ内部の処理によりシステムコールを発行した PE は待ち状態になり，全 PE が待ち状態になったときにシミュレータの実行モードが切り替わりシミュレーションを継続する．

インラインアセンブラによりシステムコールを呼び出すタイミングの指定やループの回転数計算は，コンパイラによりプログラム中に挿入されたランタイムにより行う．

3.2 精度切り換え機能のインターフェース

精度切り換えは対象ループの冒頭に精度切り換えコードを関数の形で挿入することで行う．この関数は精度切り替えタイミングの情報を受け取ってループの回転数の計算を行い，指定された回転数でシステムコールの呼び出しを行う．また，精度切り換えタイミングは，配列の形で詳細シミュレーションを行う回転数と機能シミュレーションを行う回転数を交互に設定することによって指定する．

図 3 に OSCAR コンパイラによる並列化プログラムを基にした例を示す [6]．図中，配列 sim_count が精度切り替えタイミングの回転数をしている．図の例では，最初の 12 回転を詳細モード，次の 238 回転を機能モード，さらに次の 3 回転を詳細モードで実行することをそれぞれ示す．また，sim_change 関数で実際の精度切り替えを行う．sim_change の第一引数が切り替えを行う PE 番号，第二引数が精度切り替えタイミング情報を格納している配列となる．

```
int sim_count[] = {12，238，3，…}　/* 精度切り換えタイミング
の回転数を指定 */
MAIN_PE0{　/*PE0 の処理*/
        for(…，…，…){　/*サンプリング実行対象ループ*/
            sim_change(0，sim_count);　/* 回転数の計算，シス
テムコールの呼び出しを行うユーザー関数 */
                        …
        }
}
MAIN_PE1{　/*PE1 の処理*/
        for(…，…，…){
            sim_change(1，sim_count);
                        …
        }
}
```

図 3 精度切り替えコード挿入のイメージ

Figure3 An image of code for changeover of simulation modes

4. 評価

本章では,まず本手法による精度切り替えのオーバーヘッドやキャッシュへの影響を測定し，シミュレーション精度にどの程度の影響を及ぼし得るか調査する．次に，アプリケーションを用いて本シミュレーション手法により得られる推定実行サイクル数の精度と，シミュレーション時間の速度向上率を評価する．

4.1 精度切り換えコードのプログラムへの影響

本来は精度切り替えの処理はシミュレータ側に隠蔽し，対象プログラムには影響を与えないというのが理想である．しかし，任意のタイミングでシミュレーションモード切り換えを可能とし，またアセンブラ埋め込みによるシステムコール呼び出しや回転数計算を行うといった性質上，精度切り替えの処理を第三節で述べたようにユーザーコードとして置かざるを得ない．そこで，本節では精度切り替えコードがプログラムや推定実行サイクル数にどのように影響を及ぼし得るかを評価する．

4.1.1 精度切り替えコードのオーバーヘッド

まず，精度切り替えコードのオーバーヘッドがどのように推定サイクル数に影響を及ぼすか調査した．

精度切り替えコードは回転数の計算や条件分岐，システムコールの呼び出しなどを含み，これらの処理にかかるオーバーヘッドが推定サイクル数の誤差の拡大につながることが考えられる．そこで，精度切り替えコードのサイクル数を計測したところ,一回当たり約 20 サイクル程度であった．本手法はサンプリング対象イタレーションの実行時間がある程度大きいプログラムを想定しているため，20 サイクル程度のオーバーヘッドは実行サイクル数の推定にほとんど影響を与えないと考えられる．

4.1.2 精度切り替えコードのキャッシュへの影響

次に，精度切り替えコードのキャッシュへの影響を調査する．

精度切り替えコードでは，精度切り替えタイミングの指定や回転数の計算時などで配列を用いるため，キャッシュに影響が及んでしまうことが考えられる．そこで，精度切り替えコードを挿入したプログラムと挿入していないプログラムそれぞれを，キャッシュサイズの異なるアーキテクチャで評価をとり，キャッシュミスヒット率のパラメータを比べることで，精度切り替えコードのキャッシュへの影響を調べる．L1 キャッシュサイズを変更する場合のキャッシュサイズを表 1 に，L2 キャッシュサイズを変更する場合のキャッシュサイズを表 2 に示す．

表 1　L1 キャッシュの評価のキャッシュサイズ

L1 cache size	32kB，16kB
L2 cache size	512kB

表 2　L2 キャッシュの評価のキャッシュサイズ

L1 cache size	32kB
L2 cache size	64kB，256kB，512kB

評価に使用したアプリケーションは SPEC CPU 2000 ベンチマークの EQUAKE とし,シミュレーションアーキテクチャの仕様を表 3 に示す．

次に,L1 キャッシュサイズを変更した時のキャッシュミス率を図 4 に,L2 キャッシュサイズを変更した時のキャッシュミス率を図 5 にそれぞれ示す．

図 4，図 5 を参照すると，キャッシュサイズが小さくなっても，精度切り換えコードはキャッシュミス率にほとんど影響を与えないことが分かる．

以上より，本手法における精度切り替えコード挿入は，シミュレーションによる実行コストやキャッシュの挙動にほとんど影響がないことが確かめられた．

表 3　キャッシュ評価における
シミュレーションアーキテクチャの仕様

命令セット	SPARC V9
コア数	8
L1 cache latency	1
L2 cache latency	4
memory latency	60
キャッシュ構成	L2 スヌープ

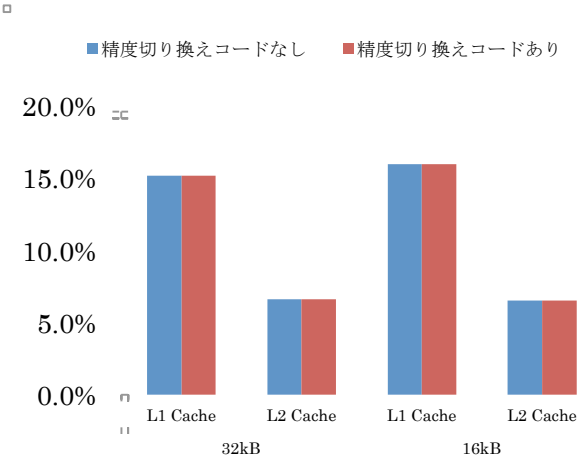

図 4　L1 キャッシュサイズを変更した時の
キャッシュミス率

Figure 4 L1 Cache-miss rate with and without runtime overhead

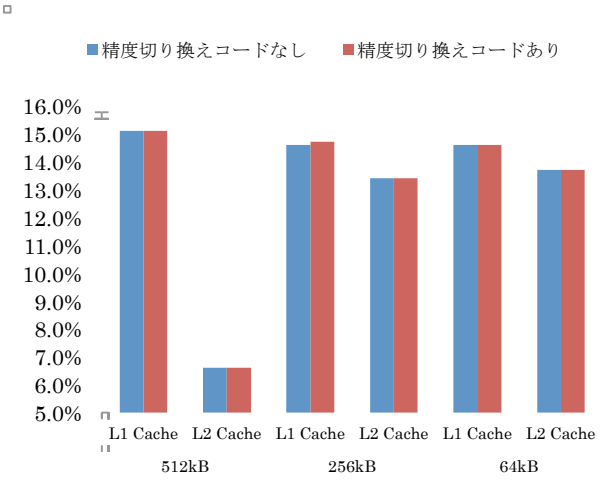

図 5　L2 キャッシュサイズを変更した時の
キャッシュミス率

Figure 5 L2 Cache-miss rate with and without runtime overhead

4.2 アプリケーション評価

本節では，アプリケーションを用いて本稿で提案するシミュレーション高速化手法により得られる推定実行サイクル数の精度と，シミュレーション時間の速度向上率を評価する．

評価アプリケーションは SPEC CPU 2000 のベンチマークの一つである，EQUAKE を用いる．これは，盆地のような地形を伝わる地震波の影響をシミュレーションするプログラムである．EQUAKE の構造を図 6 に示す．

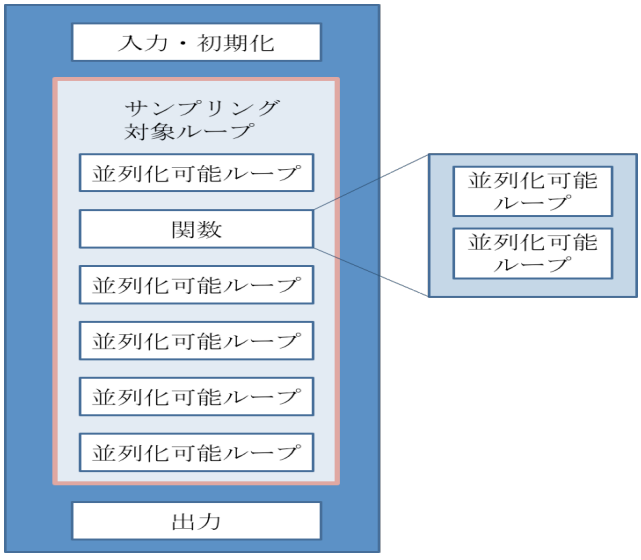

図 6　EQUAKE のプログラム構造

Figure 6　Program structure of EQUAKE

EQUAKE は，並列化可能ループを内包する一つのメインループを持つため，このメインループをサンプリング実行対象ループとする．

逐次プロファイル採取に使用したサーバーの仕様を表 4 に，採取したメインループのプロファイル結果を図 7 に示す．

表 4　Intel Xeon E5506 の仕様

CPU	Xeon
CPU 数	8
CPU Clock	2．83GHz
L1 Cache(I/D)	32KB/32KB
L2 Cache	6．0MB
Main Memory	7．8GB

図 7 より，250 回転目とそれ以降で，プログラムの挙動が大きく異なっていることがわかる．そのため本評価では，サンプリングは 250 回転以前と以後で分けて行う．また，125 回転目と 250 回転目に突出して挙動が異なるイタレーションがあるため，その部分を除外してサンプリングを行う．除外した 125 回転目と 250 回転目は，詳細シミュレーションを行うものとする．

以上の考察を基に，それぞれの部分のプロファイル結果を式(1)の計算を行う統計処理ツールに通すことで，詳細シミュレーションを行うべきイタレーション数(サンプル数)を得る．実機プロファイル結果から得られたループ回転数，標準偏差，平均値，サンプル数を表 5 に示す．

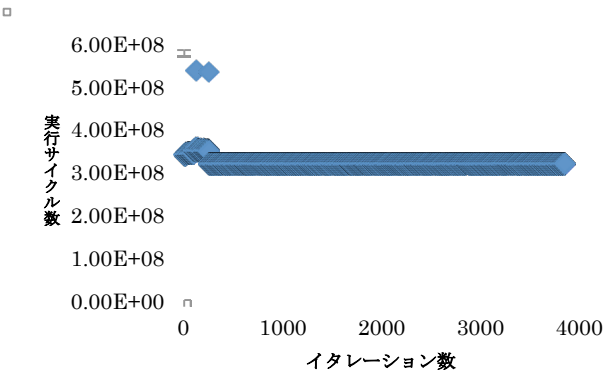

図 7　実機プロファイル結果

Figure 7　Execution cost of each iteration in a main loop of EQUAKE on a real server

表 5　EQUAKE の回転数，標準偏差，平均値，サンプル数

	回転数	標準偏差	平均値	サンプル数
250 回転以前	250	1．72E+07	3．49E+08	4
250 回転以後	3605	7．22E+05	3．19E+08	1

以上の結果を元に得られる精度切り替えタイミングの情報は次のような配列になる．

sim_count[] = {4, 120, 1, 124, 1, 1, 1}

次に，任意のコア数で並列化した EQUAKE の，精度切り替えシミュレーションの結果を示す．推定実行サイクル数は，精度切り替えシミュレーションを行い出力された実行サイクル数と式(2)によって算出する．また，詳細シミュレーションを行うイタレーション数を，算出されたサンプル数よりも増やしてシミュレーションを行い，全サイクル数との誤差率について調査する．

今回，評価時間の制約から全イタレーションの詳細実行を行うことができなかったため，できるだけ多くのイタレーション数をシミュレーションした結果から算出した推定実行サイクル数を全実行サイクル数として，誤差率を算出した．誤差率の算出式を以下に示す．

$$\text{誤差} = \frac{\text{推定全実行サイクル数} - \text{全実行サイクル数}}{\text{全実行サイクル数}} \times 100 \quad (3)$$

また，今回シミュレーションするアーキテクチャの仕様を表 6 に示す．

表 6　サンプリング実行評価における
シミュレーションアーキテクチャの仕様

命令セット	SPARC V9
コア数	1，2，4，8
L1 cache size	32kB
L1 cache latency	1
L2 cache size	512kB
L2 cache latency	4
memory latency	60
キャッシュ構成	L2 スヌープ

250 回転以前の部分の推定実行サイクル数と誤差率を図 8 に，250 回転以後の部分の推定実行サイクル数と誤差率を図 9 にそれぞれ示す．

図 8 と図 9 を参照すると，算出されたサンプル数以上の詳細シミュレーションを行うことで，本評価で設定した誤差率 5%以下に収まった実行サイクル数が推定できることが分かる．また，詳細シミュレーションを行うイタレーション数を増やすと，誤差率が低くなっていくことが分かる．

次に，250 回転以前の部分のシミュレーション時間の速度向上率を図 10 に，250 回転以後の部分のシミュレーション時間の速度向上率を図 11 にそれぞれ示す．全イタレーションのシミュレーション時間は，できるだけ多くのイタレーション数をシミュレーションした結果から推定されるシミュレーション時間を用いた．

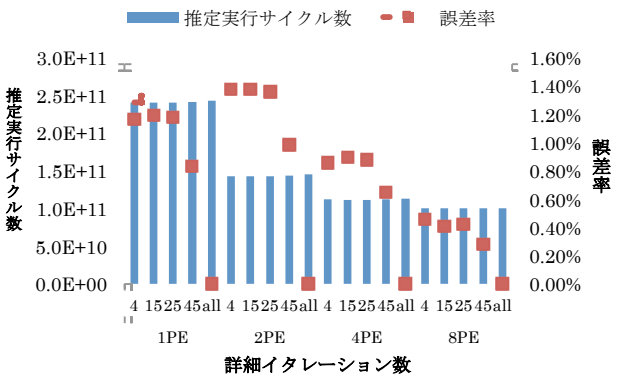

図 8　250 回転以前の部分の推定実行サイクル数と誤差率
Figure 8　The number of presumed execution cycles and error rate of a portion before 250 iterations

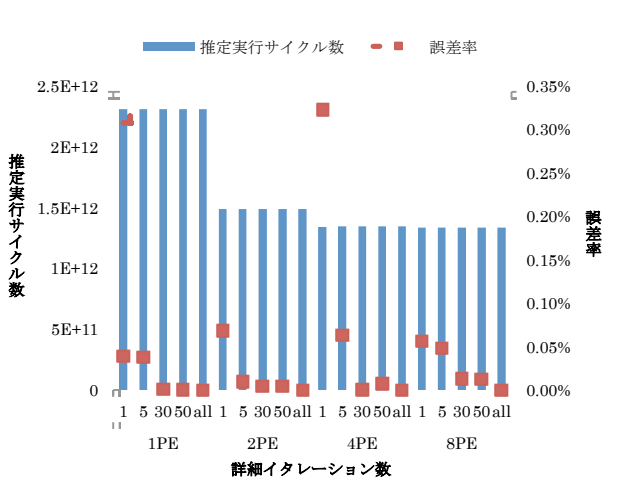

図 9　250 回転以後の部分の推定実行サイクル数と誤差率
Figure 9　The number of presumed execution cycles and error rate of a portion after 250 iterations

図 10 では全てのイタレーションを詳細実行する場合に比べて，4 イタレーション分の詳細シミュレーションでは約 54 倍，15 イタレーションで約 16 倍，25 イタレーションで約 10 倍，45 イタレーションで約 5 倍の高速化が得られることが分かった．また同様に図 11 では，1 イタレーションで約 558 倍，5 イタレーションで約 345 倍，30 イタレーションで約 102 倍，50 イタレーションで約 65 倍の高速化が得られることが分かった．

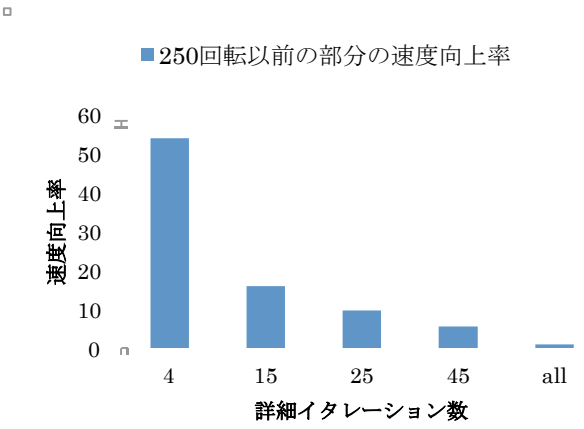

図 10　250 回転以前の部分の速度向上率
Figure 10　The speedup rate of a portion before 250 iterations

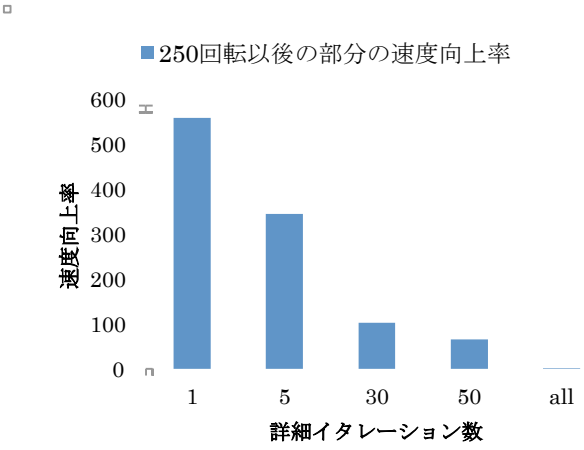

図 11　250 回転以後の部分の速度向上率
Figure 11　The speedup rate of a portion after 250 iterations

5. まとめ

本稿では，コンパイラと協調して精度の高いシミュレーションを高速に行うフレームワークを提案した．本フレームワークでは，予めプログラムにヒント情報を与えることで，コンパイラと協調してサンプリングサイズや精度切り替えタイミングの決定や，精度切り替えコードの生成を行い，サンプリング実行を行うことが可能となっている．

また，EQUAKE に本フレームワークを用いることで，期待する誤差の範囲で実行サイクル数を推定でき，さらにサンプル数を増やすことで誤差が少なくなっていくことを示した．

また，EQUAKE の全てのイタレーションを詳細実行した場合，一年以上の時間を要するが，本高速化手法を用いることで，誤差 1.6%以内で 50 倍～100 倍の高速化を実現できることが確かめられた．

謝辞　本研究の一部は科研費若手研究（B）23700064 の助成及び，経産省グリーンコンピューティングシステム研究開発により行われた．

参考文献

1)　Thomas F. Wenishch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Bavak Falsafi, and James C. Hoe, “Sim-Flex: Statistical Sampling of Computer System Simulation” Micro IEEE, Volume 26, Issue 4, pp.32-42, July-Aug, 2006

2)　Erez PerelmanGreg HamerlyMichael Van Biesbrouck Timothy SherwoodBrad Calder “Using SimPoint for Accurate and Efficient Simulation” SIGMETRICS ’ 03, San Diego, California, USA. ACM 1-58113-664-1/03/0006, June 10—14, 2003

3)　石塚亮，阿部洋一，大胡亮太，木村啓二，笠原博徳，“科学技術計算プログラムの構造を利用したメニーコアアーキテクチャシミュレーション高速化手法の評価”，情報処理学会研究報告．計算機アーキテクチャ研究会報告 2011-ARC-196(14), 1-11, 2011-07-20

4)　阿部洋一，石塚亮，大胡亮太，田口学豊，木村啓二，笠原博徳，“並列メディアアプリケーションを対象としたメニーコアアーキテクチャシミュレーションの高速化の検討”，情報処理学会第 191 回計算機アーキテクチャ研究会報告 Vol. 2012-ARC-199, No.3, 2011-07-20

5)　Hironori Kasahara, Motoki Obata, Kazuhisa Ishizaka, “Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP”, Proc. of 13th International Workshop on Languages and Compilers for Parallel Computing (LCPC’00), Aug., 2000

6)　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”, Lecture Note in Computer Science, Springer, Vol.5898, pp.188-202, 2010