Vol.2010-ARC-190 No.20
2010/8/4

プログラム構造に着目したメニーコアアーキテクチャシミュレータの高速化手法

石　塚　亮[†1]　大友俊也[†1,†2]　大胡亮太[†1]
木村啓二[†1]　笠原博徳[†1]

本稿ではキャッシュやパイプラインまでシミュレーションする詳細シミュレーションと命令実行のみの高速な機能シミュレーションの両方を用いたシミュレーション精度切り替えによるメニーコアシミュレータの高速化手法を提案する．本手法はメニーコアシミュレータ上で並列化プログラムを実行することを前提としており，このプログラムの一部のみを詳細シミュレーションを行うことにより高速化を図る．このとき，詳細シミュレーションを行うサンプリング部分をプログラム構造から判断し，その分量を統計的手法により決定する．本手法を SPEC95 の TOMCATV，SWIM で及びルネサステクノロジ（当時）提供の AAC エンコーダプログラムを用いて評価したところ，64 コアを想定したシミュレーションで，TOMCATV で 3%以下の誤差，SWIM で 6%以下の誤差，AAC エンコーダで 5%以下の誤差の実行サイクル数を 1/90〜1/8 のサンプリング実行で得ることができた．

An Acceleration Technique of Many Core Architecture Simulator Considering Program Structure

Ryo Ishizuka ,[†1] Toshiya Ootomo ,[†1,†2]
Ryota Daigo ,[†1] Keiji Kimura [†1]
and Hironori Kasahara[†1]

This paper proposes an acceleration technique of many core architecture simulator which dynamically changes the simulation mode. The detailed simulation mode considering architectual details, such as cache and pipeline, is used for some essential portion of the target program while the fast functional simulation mode which only simulates instruction execution is leveraged for the rest of the program. The key feature of the proposed technique is that the essensial portion of the program which should be precisely simulated, is analyzed from the program strutre as well as the appropriate sampling size for detail simulation for that portion are determined with statistical approach. The evaluation results show that the simulation method give us the within 3% error for TOMCATV, 6% error for SWIM, 5%error for AACencorder, of execution clock cycles by 1/90 - 1/8 of samplings in the simulation of 64 cores.

1. はじめに

コンピュータアーキテクチャの研究において, アーキテクチャシミュレータは非常に大きな役割を担っている．しかしながら, ソフトウェアによるシミュレーションは, 評価において多大な時間要する．特に複数のプロセッサコアを有するマルチコアのアーキテクチャシミュレーションを行う際にはコア数の増加に伴うシミュレーション時間の増加が深刻な問題となっており，メニーコア研究の障害になっている．

シングルプロセッサのシミュレーション評価に関しては，以前からプログラムの一部のみ評価し，その際の IPC を比較するという方法が採られてきた．しかしながら，マルチコアにおける並列アプリケーションの評価においてはプログラムの速度向上率が重要であり，プログラム部分の IPC 比較は適さず，シミュレーションそのものの高速化が求められる．

アーキテクチャシミュレーションの高速化に関しては，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注目されている．これらの研究には FPGA によりテストベッドを構築するもの[1] シミュレータを並列化するもの[2)3)4)] 統計的手法を用い，詳細シミュレーションを行う部分を限定するもの[5)6)7)8)] などがある．特に，プログラムの一部分を詳細にシミュレーションするサンプリング実行により高速化を図る手法として SimFlex[5] と SimPoint[6] が挙げられる．

SimFlex[5] は統計的手法により，詳細シミュレーションを行うサンプル量を決定することが特徴として挙げられる．まずは，評価アプリケーション全体を簡易なシミュレーションで実行する．この時に一定間隔でプロセッサの状態をチェックポイントという目印をサンプル数書き出す．次に書き出したチェックポイントから一定区間を詳細なシミュレーションで行う．この時に得られた評価結果を統合して全体の評価結果とする．

SimPoint[6] はプログラム構造に着目し，詳細にシミュレーションを行うプログラムを抽出する．Basic Block Vector(BBV) を利用し，プログラムの振る舞いをキャプチャする．キャ

†1 早稲田大学 基幹理工学部 情報理工学科
Dept. of Computer Science and Engineering, Waseda University
†2 現在，富士通株式会社

プチャ後，その BBV を比較し，類似している部分を解析し，クラスタ化する．それぞれのクラスタでその代表となるシミュレーションポイントをピックアップする．それらシミュレーションポイントのみを詳細にシミュレーションを行うことにより高速化を行っている．

これら２つの例のようにシミュレーション時間の短縮に関して様々な取り組みが行われてきた．しかし，SimFlex や SimPoint のようにサンプリング手法を用いることにより，メニーコア・アーキテクチャ上の並列プログラム実行に関して，シミュレーションの高速化に成功した例は未だ存在しない．

提案するシミュレータの高速化手法の特徴は，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ことにある．すなわち，並列化されたループ，あるいは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統計的手法を用いて期待する誤差に収まる範囲でサンプリング実行する．

本手法では，まず並列化する前の逐次プログラムを任意の実機上で実行する．その際，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を算出する．そのデータを基にサンプリング対象のループを算出したイタレーション回数だけ詳細にシミュレーションを行い，実行結果の確認を行うために残りのイタレーションは簡易なシミュレーションを行う．

以下，2 章では全て詳細にシミュレーションしたときの全実行サイクル数を推定する手法について，3 章では評価アプリケーションについて，4 章ではそれぞれの評価結果，最後の 5 章でまとめをそれぞれ述べる．

2. 実行サイクル数推定手法

本手法ではプログラムにおけるイタレーション回数のサンプリング対象とな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する．その際，詳細にシミュレーションするイタレーション数を明らかにする必要がある．

本章では並列化される前の逐次プログラムを任意の単一プロセッササーバ上で実行したデータを基に詳細にシミュレーションするイタレーション数を算出し，一部のみ詳細にシミュレーションした実行サイクル数から全て詳細にシミュレーションしたときの実行サイクル数を期待する誤差の範囲で推定する手法を述べる．

なお，本稿ではサンプリング対象ループを並列化ループを内包する最も回転数の多いループ，そのようなループがない場合は並列ループそのものとする．

2.1 イタレーション回数算出手法

まず，サンプリング対象のループに 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，任意の実機上で逐次実行させる．

今回はアーキテクチャの差異を確認するため，Intel 社の Xeon と IBM 社の Power5 上でそれぞれ 1 コアを使用し評価を行った．3.1 節で述べる，SPEC95 ベンチマークの TOMCATV のサンプリング対象ループのイタレーション毎の実行サイクル数を図 1，図 2 示す．

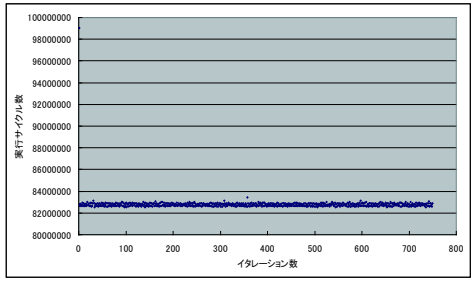

図 1 XEON で計測した TOMCATV のイタレーション毎の実行サイクル数

これらの図から，キャッシュやパイプラインの状態が安定していない最初の数イタレーションは実行サイクル数の変化は大きいが，すぐに，イタレーション毎の実行サイクル数の変化は小さくなる．また，実機の違いによってイタレーション毎の実行サイクル数に大きな差が無いことも確認できる．この結果から，サンプリング対象となるループの実行サイクル数はプログラム依存であり，一部の実行サイクル数から全体の実行サイクル数を推定できる可能性が高いことを示している．

このように実機から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となる，詳細にシミュレーションを行うイタレーション回数（サンプル回数）を決定する．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を算出し，許容する誤差（信頼度）を決定する．例えば誤差 5%以下と目的とするならば，信頼度は 0.05 となる．これらの値を用いて，以下の式で計算する．

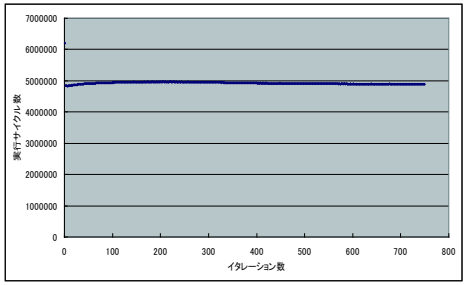

図 2 POWER5 で計測した TOMCATV のイタレーション毎の実行サイクル数

$$\lfloor \text{サンプル回数} \rfloor \geq (\frac{2.00}{\text{信頼度}} \times \frac{\text{標準偏差}}{\text{平均値}})^2 \quad (1)$$

この式から得られたサンプル回数の値が，詳細にシミュレーションするイタレーション回数となる．

2.2 推定実行サイクル数計算式

決定したサンプル回数の数だけ，サンプリング対象のループを詳細にシミュレーションする．その時の実行サイクル数を部分詳細シミュレーションサイクル数と呼称する．これから全て詳細にシミュレーションした際の実行サイクル数を以下の式から算出する．

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

3. 評価アプリケーション

本章では今回シミュレーションをするにあたって使用したアプリケーションの特徴と並列化した際のプログラムについて述べる．これらのプログラムの並列化には OSCAR 自動並列化コンパイラ[9] を用いた．

3.1 TOMCATV

tomcatv は SPEC95 に含まれるプログラムの 1 つであり，vectorized mesh を生成するプログラムである．tomcatv はサブルーチンや関数を持たないプログラムで，実行時間の大部分は 1 つのメインループによって占められている．このメインループにはイタレーション間の並列性はないが，内部は並列実行可能ループで構成される．TOMCATV のプログラム構造を図 3 に示す．本評価ではこのメインループをサンプリング対象とした．

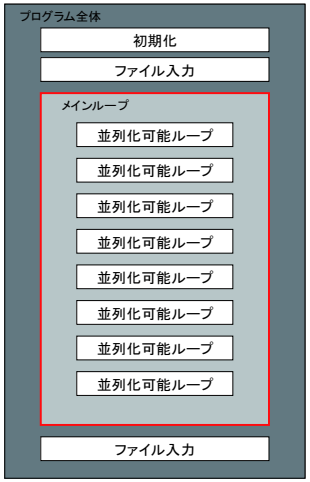

図 3 TOMCATV のプログラム構造

3.2 SWIM

swim は SPEC95 に含まれるプログラムの 1 つであり，Shallow water 方程式の求解プログラムである．swim は 1 つのメインループがあり，メインループから呼ばれるサブルーチンが CALC1，CALC2，CALC3，および CALC3Z の 4 つ存在する．また，これら 4 つのサブルーチンのうちの CALC3Z は初めの 1 回転目のみ呼ばれる．SWIM のプログラム構造を図 4 に示す．本評価ではこのメインループをサンプリング対象とした．

3.3 AAC エンコーダ

AAC エンコーダは，ルネサステクノロジ（当時）御提供による，オーディオ圧縮を行うマルチメディアアプリケーションである．非圧縮音声データの WAVE ファイルを入力して，AAC 形式の音声圧縮データを出力する．エンコーダ内部では WAVE ファイルをフレームと呼ばれる単位に分割し，このフレームごとに圧縮処理を行っており，またこのフレーム数

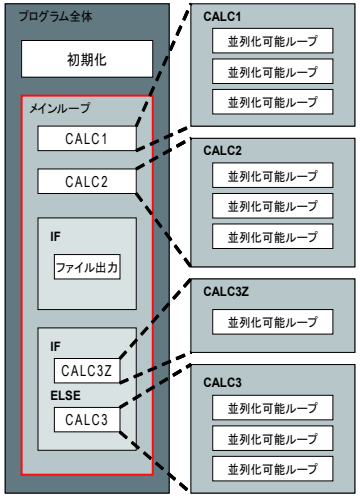

図 4 SWIM のプログラム構造

がメインループのイタレーション数となる．圧縮処理にはフレーム間の依存が無いため，この単位で並列化可能である．AAC エンコーダのプログラム構造を図 5 に示す．本評価ではこの並列ループをサンプリング対象とした．

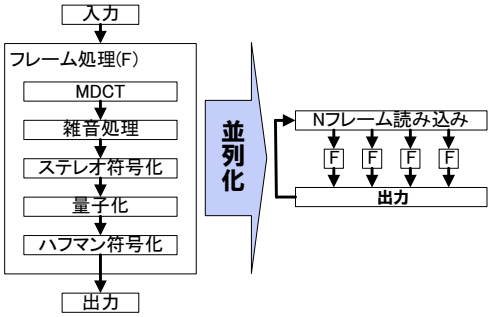

図 5 AAC エンコーダのプログラム構造

3.4 サンプリングするイタレーション回数

Xeon における各アプリケーションの対象ループの回転数，標準偏差，平均値，サンプル数を表 1 に示す．2 章で述べた，プログラムの逐次実行の値と，式 (1) よりサンプル数を算出した．

表 1 各アプリケーションの回転数，標準偏差，平均値，サンプル数

アプリケーション	回転数	標準偏差	平均値	サンプル数
TOMCATV	750	605.7×10^3	82.8×10^6	1
SWIM	900	362.2×10^3	82.8×10^6	1
AAC エンコーダ	8240	83.7×10^3	1.52×10^6	7

4. 評 価 結 果

本章では 3 章で挙げた各アプリケーションの推定実行サイクル数と全て詳細にシミュレーションした実行サイクル数の誤差を示す．なお，誤差の算出式を以下に示す．

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

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 2 に示す．

表 2 シミュレーションアーキテクチャ仕様

命令セット	SPARC V9
コア数	1，16，32，64
L1cache	32kB(I/D)
L1cache latency	1
L2cache	64kB，128kB，512kB
L2cache latency	2
memory latency	24
キャッシュ構成	L2 スヌープ

なお，L2cache サイズが 512kB の際かつ 32 コア以上のアーキテクチャで TOMCATV，SWIM のデータが全てキャッシュに乗るように設定されている．

4.1 TOMCATV

TOMCATV は本来，ループを 750 回転するプログラム構造であるが，全て詳細にシミュレーションしようと試みると 64 コアで推定 4，5ヶ月ほどかかり現実的に不可能である．よっ

Vol.2010-ARC-190 No.20
2010/8/4

てループのイタレーション数を変化させた時の推定実行サイクル数の誤差を評価の対象とする．本評価では，サンプル数の変化による推定誤差の影響を調査するため，イタレーション数は 5 回，10 回，20 回，40 回と変化させた．また，TOMCATV のプログラムをキャッシュ最適化[10] をかけた時の推定実行サイクル数も同様の方法で算出した．

TOMCATV のイタレーション数を変化させた時の推定実行サイクル数と実行サイクル数の誤差の評価結果を図 6，図 7 にそれぞれ示す．なお，40 回転の推定実行サイクル数を実行サイクル数と仮定し，算出した．

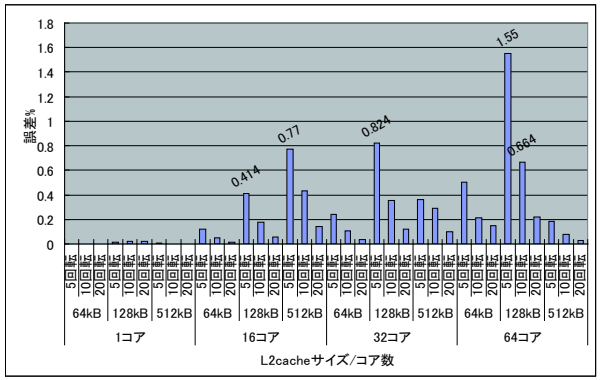

図 6 TOMCATV の推定実行サイクル数誤差

図 6，7 より，イタレーション数を変化させた場合でもその推定実行サイクル数の誤差はほとんど無い．最大で誤差 3 仮に回転数をこれ以上増やしたとしても結果は同様であると考えられる．

4.2 SWIM

SWIM は元のプログラムでは，ループを 900 回転するプログラム構造であるが，TOMCATV 同様，全て詳細にシミュレーションを行うことは現実的ではない．よってループのイタレーション数を変化させた時の推定実行サイクル数の誤差を評価の対象とする．4.1 節の TOMCATV と同様に，イタレーション数は 5 回，10 回，20 回，40 回と変化させた．な

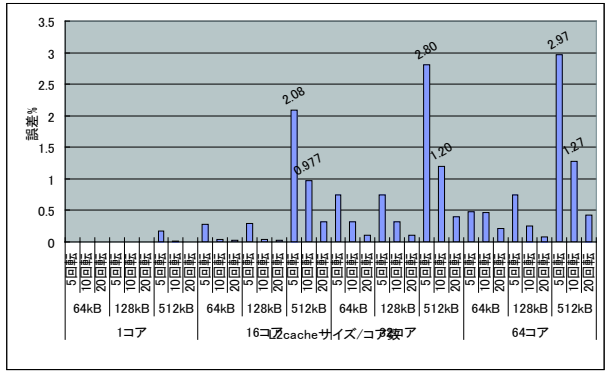

図 7 TOMCATV(キャッシュ最適化) の推定実行サイクル数誤差

お，第 3 章で述べたように SWIM は 1 回目だけ呼び出される関数が異なる．そこで，推定実行サイクル数を以下のように変形させた．

$$\begin{aligned}\text{推定実行サイクル数} = (&\text{詳細シミュレーションサイクル数} \\ &-1\text{ 回転目の詳細シミュレーションサイクル数}) \\ &\times \frac{899}{\text{サンプル数-1}} + 1\text{ 回転目の詳細シミュレーションサイクル数}\end{aligned} \tag{4}$$

また，の SWIM のプログラムをキャッシュ最適化[10] をかけた時の推定実行サイクル数も同様の方法で算出した．

SWIM のイタレーション数を変化させた時の推定実行サイクル数と実行サイクル数の誤差の評価結果を図 8，9 にそれぞれ示す．なお，40 回転の推定実行サイクル数を実行サイクル数と仮定し，算出した．

SWIM も TOMCATV 同様，イタレーション数を変化させてもその推定実行サイクル数の誤差は非常に小さい．コア数の変化やキャッシュサイズの変化，及びキャッシュ最適化の有無に対しても誤差は微少であり最大 6%程度に収まっている．回転数を増加させた場合，TOMCATV 同様に推定実行サイクル数の精度が高くなる．

Vol.2010-ARC-190 No.20
2010/8/4

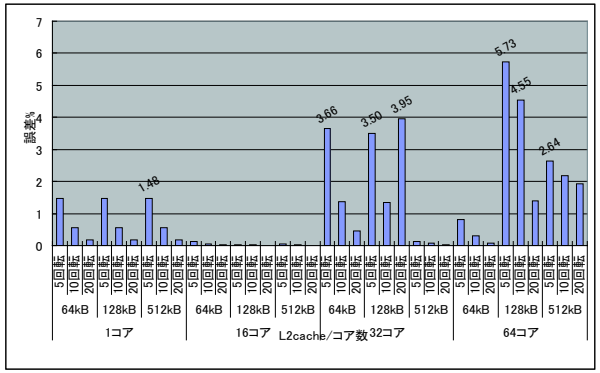

図 8 SWIM の推定実行サイクル数の誤差

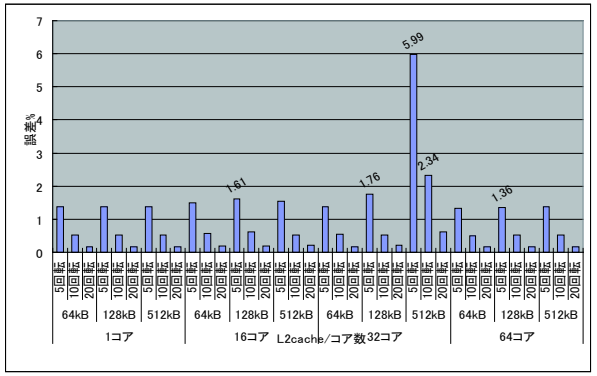

図 9 SWIM(キャッシュ最適化) の推定実行サイクル数の誤差

4.3 AAC エンコーダ

AAC エンコーダは全て詳細にシミュレーションした際のデータが取得出来ている．このデータと推定実行サイクル数の比較検討を行う．AAC エンコーダは先の 2 つのアプリケーションと比較して，イタレーション毎の実行サイクル数の変化が大きい．そのため，サンプル数が先のアプリケーションよりも大きく，キャッシュやパイプラインの状態が安定するまでのウォーミングアップのイタレーション数も大きい．本評価では並列ループ実行開始後各コアで，単一プロセッサ上で取得した，イタレーション毎の実行サイクル数から，キャッシュ，パイプラインの状態が落ち着くまでの 8 イタレーションが経過してから表 1 で示した 7 イタレーションを実行している．AAC エンコーダの推定実行サイクル数と実行サイクル数の誤差の評価結果を図 10 に示す．

図 10 より，目標とした誤差 5%を満たしており，本手法が標準偏差の比較的大きい AAC エンコーダプログラムに適応可能であるということがわかる．

4.4 シミュレータ高速化

以上の結果より，コア数，キャッシュサイズ，キャッシュ最適化の変化によらず，本稿によるサンプル対象ループのイタレーション数を縮減することにより，並列化プログラムの実行サイクル数を期待する誤差の範囲で見積もり可能なことが解った．

64 コアを想定したシミュレーションにおいて，TOMCATV のサンプリング対象ループ 40 回転の詳細シミュレーション時間は 216 時間であったが，機能シミュレーション時間は 1 時間 50 分と約 120 倍高速である．SWIM のサンプリング対象ループ 40 回転の詳細シミュレーション時間は 3 時間 37 分であり，機能シミュレーションでは 3 分と約 106 倍ほど高速である．また AAC エンコーダにおいて，フレーム処理を 8240 回転行う場合の詳細シミュレーション時間は 383 時間に対して，機能シミュレーションでは 2 時間 20 分と約 166 倍ほど高速である．

本手法で示した，サンプリングサイズのみ詳細シミュレーションを行い，その他の部分を機能シミュレーションに切り替えた際，その高速化率は数十倍〜百倍程度となることが期待できる．

5. ま と め

本稿では，キャッシュやパイプラインまでシミュレーションする詳細シミュレーションと命令実行のみで高速な機能シミュレーションのシミュレーション精度切り替えによる，メニーコアシミュレータの高速化手法について述べた．サンプリング対象ループに着目し，期待する誤差の範囲内で全実行サイクル数を推定可能なイタレーション数を特定し，そのイタレー

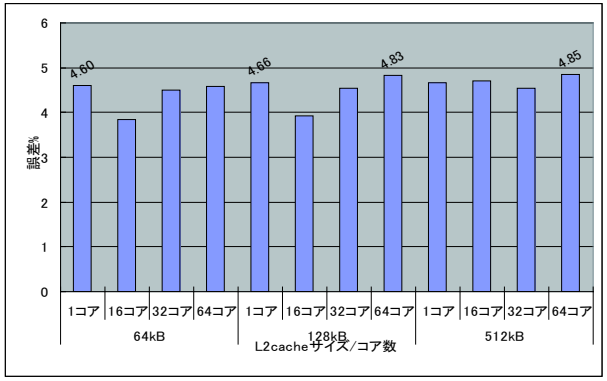

図 10 AAC エンコーダにおける推定実行サイクル数と実行サイクル数の誤差

ション数だけ詳細にシミュレーションを行うことにより，少ないシミュレーション時間で高精度の実行サイクル数推定が可能となる．今回評価したアプリケーションでは TOMCATV で誤差 3 また，ここで得られたサンプル数はコア数，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解った．さらに，機能シミュレーションは詳細シミュレーションの 106 倍〜166 倍ほど高速であり，サンプリング対象のループを全体の一部のみ詳細シミュレーションすることにより，数十倍〜百倍程高速化が見込めることが解った．

今後は，より複雑な構造を持つ様々なプログラムを用いた評価，及び，並列化コンパイラとの連動によるシミュレーションフレームワークの構築を行う予定である．

参 考 文 献

1) John Wawrzynek, Mark Oskin, Christoforos Kozyrakis, Derek Chiou, David A. Patterson, Shih-lien Lu, James C. Hoe, Krste Asanovic, “RAMP: Research Accelerator for Multiple Processors”, IEEE Micro, Vol. 27, Issue 2, March/April 2007
2) James Donald, Margaret Martonosi, “An Efficient, Practical Parallelization Methodology for Multicore Architecture Simulation”, IEEE Computer Architecture Letters, Vol. 5, 2006
3) 高崎透，中田尚，津邑公暁，中島浩 “時間軸分割並列化による高速マイクロプロセッサシミュレーション” 情報処理学会論文誌 Vol46 No.SIG 12 (ACS 11) pp.84-97 (2005)
4) 松尾治幸，今福茂，大野和彦，中島浩 “共有メモリマルチプロセッサの分散シミュレータ Shaman の設計と実装” 情報処理学会論文誌 Vol44 No.SIG 1 (HPS 6) pp.114-127 (2003)
5) 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
6) 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
7) Manu Shantharm, Padma Raghavan, Mahmut Kandemir, “Hybrid Techniques for Fast Multicore Simulation”, Lecture Notes In Computer Science, Vol. 5704 Proc. of 15th Intl Euro-Par 2009 Parallel Processing, Aug. 22, 2009
8) Davy Genbrugge, Lieven Eechkout, “Chip Multiprocessor Design Space Exploration through Statistical Simulation”, IEEE Trans. on Computers, Vol 58, No. 12, Dec. 2009
9) H.Kasahara，H.Obata，K.ishizaka “Automatic Cooanse Grain Task Parallel Processing on SMP using Open MP” Proc,of 13th Intl.workshop on language and Compilers for Parallel Processing(LCPC’00) Aug.2000
10) K.ishizaka, M.Obata, H.Kasahara, “Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multi Processor” Proc. of Intl.Workshop on Language an Compilers for Parallel Processing (LCTC2001) Aug.2001