ハイパフォーマンス
コンピューティング 48－15
（1993．8．20）

アドレストレースを利用した並列計算機のパラメトリックシミュレータ

大森洋一†, 城 和貴†,
福田 晃†, 荒木啓二郎†
†：奈良先端科学技術大学院大学 情報科学研究科

Abstract
従来、既存のマルチプロセッサ以上の規模のマシンの評価は、解析モデルもしくは確率によるシミュレーションで行なわれることが多かった。しかし、実マシンの動作にはこれらでは表現しきれない部分がある。これに対し、トレースドリブンなシミュレータは実機を忠実に表現できるが、実行時間・必要なデータ量から大規模マルチプロセッサへの適用が困難である。そこで、トレースドリブンを基本に、他のプロセッサと独立な部分は確率モデルを適用する。これにより、従来のトレースドリブン型のシミュレーションに比べ、大幅にメモリ使用量を減らすことができた。また、データ操作の省略により、シミュレーション時間も短縮された。これによる動作の相違は 5% 程度であった。

A Parametric Simulator for Multiprocessor Using Adress Trace

Youichi OMORI†, Kazuki JOE†, Akira FUKUDA†, and Keijirou ARAKI†

†: Department of Information Science Faculty of Engineering,
Advanced Institute of Science and Technology Nara Graduate school
Takayama-chou8916-5, Ikoma-city, Nara 630-01 Japan

E-mail: {youiti-o, kazuki-j, fukuda, araki}@is.aist-nara.ac.jp

Conventionally, an evaluation of a multiprocessor computer larger than existing computers has been usually performed by using analytic model or probabilistic simulation. But, in some cases, they don't have enough ability to stand for real machines. Trace-driven simulation results in very similar to real machines. It takes too long execution time and needs too much memory to stand for a large scale multiprocessor. So, based on the Trace-driven simulation, using probabilistic model in independent modules, we save not only memory but also execution time. And the difference of result from pure Trace-driven simulation is only 5 % or less.

1 はじめに

計算機の性能向上のために並列化技術が広く用いられるようになるにつれて、性能評価をどのようにして行なうかが重要な課題となっている。特にシステムが大規模になるにつれて、個別のモジュール性能から予測される理論性能に対し、実効性能が飽和してしまう現象が知られている。そこで、システムのどの部分を改良すべきであるのかを知るためにベンチマークテストが行なわれる。

しかし、設計時など実機が存在しない場合にはベンチマークテストは不可能であるので、システムパラメータや負荷パラメータを入力とし、性能予測を出力するような、抽象モデルによる性能評価を利用しなくてはならない。

対象がシングルプロセッサの場合、任意の事象に適切な生起確率を与えることにより、システムの実効性能の定量的な評価を低コストかつ短時間で行なう待ち行列を用いた解析モデルが有効であることが知られている [1][2][3]。

対象がマルチプロセッサの場合に、解析モデルを適用した例もある [4][5]。しかし、プロセッサ数が増加すると、図1に見られるようなプロセッサ間のメモリアクセス集中により、相互結合網のバンド幅を一時的に越えてしまうといった命令列の物理的局所性、あるいは図2に見られるような連続したメモリアクセスによるメモリアクセスパイプラインの乱れといった時間的局所性による問題が相対的に強く現れるため、待ち行列を用いた解析モデルを適用すると、さまざまな条件について分布が大きく変化する [6][7]。

このため解析モデルを、プロセッサ間通信の遅延の特定の問題に対する定量的評価に用いると、ごく簡単な場合にしか適用できないといった問題を生じる。

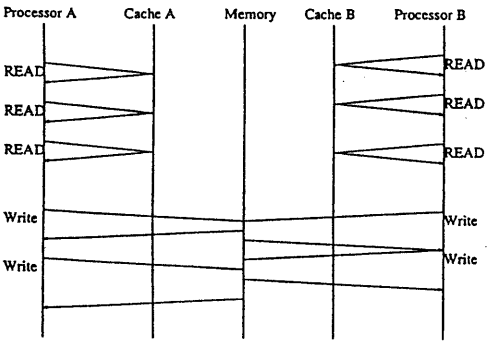

図 1: 空間的なメモリアクセス集中

シミュレーションモデルは現実のコンピュータと同じようなふるまいをするプログラムであり、システムを反映する意味のある予測情報を与えてくれる。

ただし、どのような性質の問題についてシミュレーションを行なうかというワークロードモデルを適切に選択しなければならない [8][9]。

もっとも簡単な方法は乱数により命令列を与えるものである。この時、時間的・空間的に適当な分布にしたがって、メモリアクセス命令を発行する。この手法ではシミュレーションの正当性はシミュレーションモデルのみならず、ワークロードの乱数の質、命令列の分布の正当性、初期条件などにより変化する [10]。

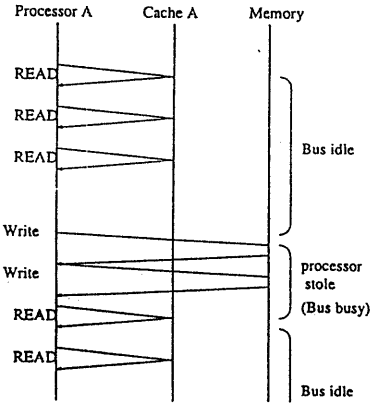

図 2: 時間的なメモリアクセス集中

より詳しい情報を得るためには、実際のマルチプロセッサ上で動作可能な命令列を、シミュレータ上で実行することができればよい [1][11]。

これは、並列化済みのプログラムを実際に並列実行して得られるアドレストレースを基にシミュレートする手法である [12]。これにより、少なくともプロセッサレベルではアルゴリズムの忠実な表現が可能である。これを実現するには、実際に動作する並列化済みのプログラム中にシミュレーションに必要なコードを埋め込んで実マシン上で動作させる手法 [13][14] や、シミュレーション対象のマシンのコードを実マシンの命令に変換する手法が挙げられる [15]。

しかしながら、これらの手法はシミュレーションを実行するマシンのアーキテクチャに依存しがちであり、共有メモリ型のシミュレーションモデルをメッセージパッシング型のマシン上で実行するといった、大きく異なったモデルを利用するのは難しい場合がある。また、実行マシンの性能を大幅に上回るマシンのシミュレーションを行なうためには何らかの仮想化機構を利用しなければならず、これは大幅な実行時間の低下を招く。また、シミュレーションモデルを作成するのに多くの開発工程を必要とし、シミュレーション実行時間も長くなる [1]。

本稿では、メモリアクセスの種類により確率を用いたシミュレーションとアドレストレースを行なうシミュレーションを切替えることによるシミュレーションの高速化手法を提案する。特に同期命令については、同期に参加しているプロセッサの数を表現するためにエントリを割り当てるだけでなく、同期変数を実際に操作する。

このような既存のマシンよりも大きなマシンのシミュレーションを実行するための手法として、市販の論理回路用CAD にみられるように、実行時にアドレストレースを作成することにより、メモリの使用量を節約する手法が挙げられる [9][16]。実行時にアドレストレースを作成する手法は、実際に即した柔軟なシミュレーションが可能であるが、着目する項目以外のシミュレートも行なわなくてはならな

表 1: 確率モデルの状態遷移

現状態 \ 入力	000	010	100	110
Hit	Hit / (Hit,ϕ)	Hit / (Hit,ϕ)	Hit / (ϕ, Invalidation) | Wait / (ϕ, Writeback)	Hit / (ϕ, ϕ) | Wait
Miss	Wait / (ϕ,Mreq)	Wait / (ϕ,Mreq)	Wait / (ϕ,Mreq)	Wait / (ϕ,Mreq)
Wait	--	--	--	--

現状態 \ 入力	Invalidate	ϕ	Special Lock	Release
Hit	Miss / (ϕ,Writeback)	--	Miss / (ϕ,Mreq)	Miss / (ϕ,Mreq)
Miss	Miss / (ϕ,ϕ)	--	Miss / (ϕ,Mreq)	Miss / (ϕ,Mreq)
Wait	--	Hit / (Hit,ϕ)	--	--

いため、実行時間が長くなる。

すなわち、シミュレーションに当たって、1 サイクルのシミュレーションに複数回の演算が必要となりプロセッサのコストが大きくなること、アドレストレースの過程を保持するための記憶装置のコストが大きいことが問題となる。これは、特に大規模なマルチプロセッサを対象とするシミュレーションでは致命的である。

本稿では特にシミュレーションを実行するマシンよりも 10 倍から 100 倍のメモリ、演算性能をもつマルチプロセッサのシミュレーションを行なう場合を考える。これは現在のマシン性能の向上速度を考えると不自然な仮定ではない。

以下、2章で既存のシミュレーションモデルをマルチプロセッサに適用する例、3章でトレースデータを用いたシミュレーションの改良、4章で本稿で対象とするワークロードモデルの性質、5章で結果および性能評価について述べる。

2 既存のシミュレーションモデル

2.1 アドレストレースを利用したシミュレーション

論理共有メモリシステムにおいては、プロセッサの次状態はメモリシステムの状態に依存し、メモリシステムの次状態もプロセッサに依存する。このような場合、システムの状態数はプロセッサとメモリシステムの状態数の直積となるので、システムが大規模になると待ち行列等を用いた解析モデルでは状態数が爆発する。このため、定量評価のためにはアドレストレースを用いたシミュレーションが行なわれることが多い。

トレースドリブン型シミュレーションではアドレストレースを保持するためのメモリの確保が問題となる [17]。

ここで

プロセッサ数 P_n
命令語長 OC_n
オペランド長 OR_n
シミュレーション実行ステップ数 S_n
キャッシュライン数 C_n
キャッシュエントリの大きさ C_e
ネットワークバッファ量 N_b
メモリエントリ数 M_n
メモリエントリの大きさ M_e

とすると必要なメモリ量は次のようにして求められる。

$$P_n \times ((OC_n + OR_n) \times S_n + C_n \times C_e) + N_b + M_n \times M_e \quad (1)$$

具体的な値としてプロセッサ数 64 個, 1 プロセッサ当たりキャッシュ 4M, メモリ 16M × 64 = 1G のシステムを 1000 ステップシミュレートすることを考えると、

$$P_n = 64, OC_n = 1[B], OR_n = 1[B], \quad (2)$$
$$S_n = 1000, C_n = 4M, C_e = 1[B],$$
$$N_b = 64[B], M_n = 32M, M_e = 2[B]$$

となり、データ部分だけでメモリ使用量は約 318M[B] となるので実質的にシミュレート不可能である。

2.2 確率を用いたシミュレーション

次に確率を用いた状態遷移を考える。これはモンテカルロシミュレーションともいわれる手法である [18]。

各モジュールは非決定性オートマトンで表現され、キャッシュ部分は次の 4 字組

$$(S, I, T, O) \quad (3)$$

ただし、

S = { Hit, Miss, Wait },
I = { Load/Store, Shared/Private, Special/Normal, Invalid
O = { ϕ, Memreq, Invalidate, Hit, Writeback}

となる。ここで、

({ 状態 }, { 入力 }, { 状態遷移関数 }, { 出力 })

である。T は入力を

$Load/Store, Shared/Private, Special/Normal$

の順に 3 ビットで表現すると 表 1 のような規則によって次状態を与えるものとする。ただし、表 1 の 1 要素は

{ 次状態 / (プロセッサへの出力, メモリへの出力)}

である。ここでは、同期変数はキャッシングされないものとし、メモリ上で同期操作が行なわれるものと仮定している。

表1からわかるように、確率モデルでは遷移する次状態の選択は高々4 ビットで表現されるので、トレースドリブンに対してデータ量が非常に少ないことが特徴である。

シミュレーションは命令の割合に応じてプロセッサが出力することで行なう。これに対し、メモリシステムは状態遷移関数を実際の分布と一致するような確率分布で与える。

この手法で必要なメモリ量は、状態の記憶がいらないので2.1節と同様の表現で

$$P_n \times ((OC_n + OR_n) \times S_n + 1[B]) + 1[B] + 1[B] \quad (4)$$

2.1節と同じパラメータで約 128 K [B] となり、大幅にメモリ使用量を削減できる。計算量はトレースドリブンの場合、現状態の読み出しに一回、次状態の計算に一回、次状態の書き込みに一回かかるとすると、マシンサイクルのシミュレーションにキャッシュヒット時で 6 回、キャッシュミス時で 18 回の計算が必要である。これに対し、確率を用いたシミュレーションではキャッシュヒット時で 2 回、キャッシュミス時で 4 回の計算ですむので、計算回数を削減できる。

3 確率を併用したシミュレーション

3.1 対象とするアーキテクチャ

本稿ではシミュレーション対象とするマルチプロセッサのアーキテクチャとして、待ち行列モデルでの表現が複雑になる物理分散論理共有メモリをもつマシンを考察する。論理共有メモリ型では、同期操作はメモリシステムへのアクセスとして表現され、マルチプロセッサの演算性能を低下させる要因は、メモリシステムへのアクセス待ちに相当する。対象となるモデルは図3のようなものである。

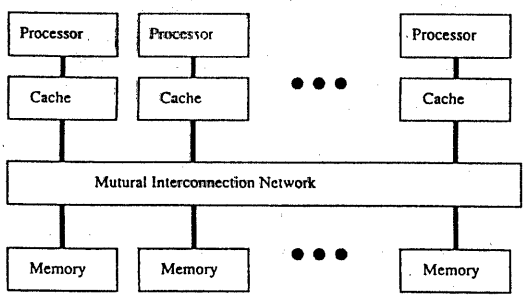

図 3: ハードウェア概念図

各プロセッサはあらかじめ与えられた並列化済みのコードをプロセッサコンシステンシ [19] を保って実行する。並列化済みとはコード、データの分割がなされ、同期命令が埋め込まれた実行可能な系列を指す。

メモリシステムはプロセッサ外部に 2 階層記憶を持つ現在では標準的なものを考える。キャッシュはブロッキング[1]するものとし、各プロセッサに個別につき、他のプロセッサの干渉はないものとする。

ネットワークはキャッシュとメモリの間にあり、FCFS の順に要求を処理する。

メモリは論理的に共有されており、プログラムはすべてメモリ上にあるものとする。すなわち、スワップアウトによる遅延や他のプロセッサから見えないローカルメモリは考えないものとする。

3.2 確率の併用

式1を見ると、アドレスドリブン型シミュレーションで必要なメモリの大半はメモリ部分に使われていることが分かる。そこで、メモリ部分のシミュレーションにかかるメモリ量を減らすことを考えなくてはならない。

一方、モンテカルロシミュレーションのような手法においては、シミュレーション対象上で具体的なアルゴリズムの動作を表現するのは困難である。なぜなら正確な表現をしようとすれば、そのための確率モデルの生成、すなわち状態遷移関数の作成に手間がかかり、確率モデルの長所である低コストという良さが失われるからである。

また、確率を用いたモデル化においては問題の局所性の十分な表現が不可能であり、特に時間軸へのメモリアクセスの分布を表現するのは困難である。

そこで、本研究では1章で述べたシステム評価手法の長所を組み合わせる手法を提案する。まず、実行時にアドレストレースを作成する方式を基本とし、対象のアドレスをそのプロセッサのみがアクセスする場合は、メモリシステムの動作を確率により決定する。

このようにシングルプロセッサと等価な場合の動作決定に確率を用いることにより、メモリの使用量を節約することができる。また、データの操作を行なわなくてすむため演算時間も小さくなる。この場合の誤差はシングルプロセッサについてのシミュレーションから、システムの性能に比べ小さいことが分かっている [2][3]。複数のプロセッサにより共有されるデータの部分については、トレースデータを用いることにより、ワークロードに対して正確な結果を得ることができる。

具体的には、各プロセッサから見たブロックを自プロセッサからのみアクセスがある非共有ブロックと他のプロセッサからのアクセスがある共有ブロックに分類し、モデルを切替える。

非共有ブロックについては確率モデルとして扱う。この時の処理は、ミスヒット時のメモリアクセスがブロックされる可能性を除き、シングルプロセッサの場合と全く同じである。キャッシュミス時にはメモリアクセスを行なう。この場合、キャッシュにエントリを確保し、ネットワークにおけるアービトレーション、メモリアクセスの遅延を考慮する。

共有ブロックに対してはプロセッサ間のブロック依存関係を表現するために、アドレストレースにしたがった処理を行なう。これはデータ処理を行なわない点を除けば、実際のシステムと等価な手順で処理を行なうことを意味する。

[1]ブロッキングキャッシュとはキャッシュミス時にメモリへのアクセスを行なう間、プロセッサへのサービスを停止するようなキャッシュである。

3.3 本研究で用いるシミュレーションモデル

本シミュレーションモデルモデルはプロセッサ、キャッシュ、バス、メモリを表すモジュールからなる。

各プロセッサは並列化済みのコードにしたがって ω サイクルのプロセッサ内部のデータ操作命令のシミュレーションを実行する。その後、メモリリクエストを発行し、キャッシュのサービスキューに入れる。そして、何らかの仕事がなされるまで応答を待つ。プロセッサ利用率は総実行時間中の有効な仕事の実行時間の割合で、システム性能はプロセッサ利用率の総和でそれぞれ計測される。

各キャッシュは受けとった命令の対象アドレスにより、動作を変える。非共有ブロックについてはキャッシュにヒットしたかどうかの判定を確率で行なう。ヒットした場合は1サイクル後キャッシュはプロセッサに Ack の信号を送る。キャッシュミスした場合は、バスリクエストを生成し、バスのサービスキューに入れる。

共有ブロックについては、そのブロックがキャッシュにエントリを持たない場合、初めて共有ブロックとしてアクセスされたと考え、エントリを割り当てる。以降のキャッシュにヒットしたかどうかの判定はこのエントリを用いて行なう。ヒットした場合は1サイクル後、プロセッサに Ack の信号を送る。キャッシュコヒーレンスの制御はイリノイ・プロトコルによる。したがって、ストア命令がヒットした場合は他のキャッシュへ Invalidation 命令を送る。キャッシュミスした場合は、バスリクエストを生成し、バスのサービスキューに入れる。プロセッサ間の時刻のずれから矛盾を生じた場合のロールバックに備えて、キャッシュ内のエントリは動作時刻の履歴をもっている必要がある。

他のプロセッサの動作の対象となるブロックのコピーを持っているキャッシュは、バスからの Invalidation 命令を受けとる場合がある。これらのコマンドはキャッシュにおいて、プロセッサのメモリリクエストよりも高い優先度を持つ。そのキャッシュコヒーレンスのための動作が完了してはじめて、キャッシュはプロセッサのリクエストに応答できる。もし、ブロックの状態が Dirty であれば、そのブロックは Writeback 命令によりメモリへ書き戻される。そうでなければそのブロックは無効化される。

バスを表すモジュールはキャッシュモジュールから要求を受け取り、FIFO の順に従って要求を処理する。要求は次の三つのうちのいずれかに属する。メモリリクエスト、Dirty block のライトバック、Invalidation 命令である。バスモジュールはバッファリングにより、シミュレーション時間上の動作誤差をある程度吸収し、要求を非同期に処理する。このためにはバスを要求する可能性のあるモジュールのバッファリングが必要である。バッファ内の要求は時刻の小さい順に処理される [20]。

ここで、バッファで吸収できない大きさの誤差を生じた時は、その時刻以降のシミュレーションを無効化する。この命令はバスからプロセッサ、キャッシュ、メモリに対して発行され、指定時刻の状態を回復する [21][22]。

メモリモジュールは、そのブロックがエントリを持たない場合、初めてアクセスされたと考え、エントリを割り当てる。

以降のメモリ状態の判定はこのエントリを用いて行なう。メモリはキャッシングされているかどうか、キャッシングされているブロックの位置、Dirty かどうかといった状態をタグにもつ。バスからの要求を処理するに当たって、データの供給を必要とする他のキャッシュが決定される。これは、そのブロックが他で要求されているか、Dirty であるかによらない。また、共有データへの書き込み時にそのブロックを無効化するかといった、状態変化も決定され、適切なキャッシュへ適切な命令が送られる。処理が完了した時点で、バスはキャッシュに続行の信号を送る。

シミュレーションモデルは実際のトレースではなく、合成された参照命令の列によって動作する。これは、特に今回対象とするような大規模な問題についてのマルチプロセッサのトレースが存在しないからである。

具体的にはビンゴゲームの並列実行をマルチプロセスで行ない、その結果を n プロセッサによる実行に変換した。この変換はシミュレータで実行するための全く形式的なものであり、元のプログラムと意味的に等価である。

各シミュレーションに対して、プロセッサにおける参照命令列は、確率モデルの部分における乱数依存部分以外は同一とした。すなわち、ワークロードは各プロセッサについて同一である。

図4にシミュレータのモジュール図を示す。プロセッサはキャッシュのみと通信し、他のモジュールからはキャッシュがメモリアクセスをするかしないかだけが見える。この意味で、キャッシュからプロセッサという視点とおなじであり、階層性が表現される。

4 ワークロードモデル

プログラムの量的特性はブロックの分割配置をさまざまに変更することにより変化する。すなわち問題に対し固有のものではなく、アルゴリズムによって変化する。

逆にいうと、問題の種別によらずアドレストレースのみに注目することで、記憶階層のシステムパラメータの性能に与える影響を知ることができる。

ワークロードモデルによりデータ共有の性質が定まり、すべてのコヒーレンス問題の解決策の実効性はデータ共有の度合に強く依存することが知られているので、ワークロードの選択は重要である。

本稿では Archibald と同様のモデルを採用する [8]。Archibald のモデルは Dubois のモデル [23] を共有データの参照の局所性を反映して拡張したものである。本モデルでは、これをさらに、プロセッサの発行する命令は実行命令列に従い、同期変数へのアクセスは実際に同期操作を行なうという 2 点について拡張する。以下にその詳細を述べる。

あるプロセッサの主記憶参照命令列はオペランドの値により、共有ブロックに対するものか、非共有ブロックに対するものか決定される。

要求が非共有ブロックに対するものであれば、確率 h でヒットし、確率 1 - h でミスする。同様に読みだし命令を確率 rd 書き込み命令を確率 1 - rd で生成する。h,rd はあらかじめパラメータとして与えられた値とする。

ただし、非共有ブロックに対するワークロードモデルは平衡時の動作を反映したもので、初期キャッシュミスは無視するものとする。

共有ブロックに対しての要求は必ずすぐメモリモジュールに伝えられる。メモリモジュールではグローバルディレクトリを用いて、今回のメモリリクエストがキャッシュにヒットしたかミスしたかを知る。

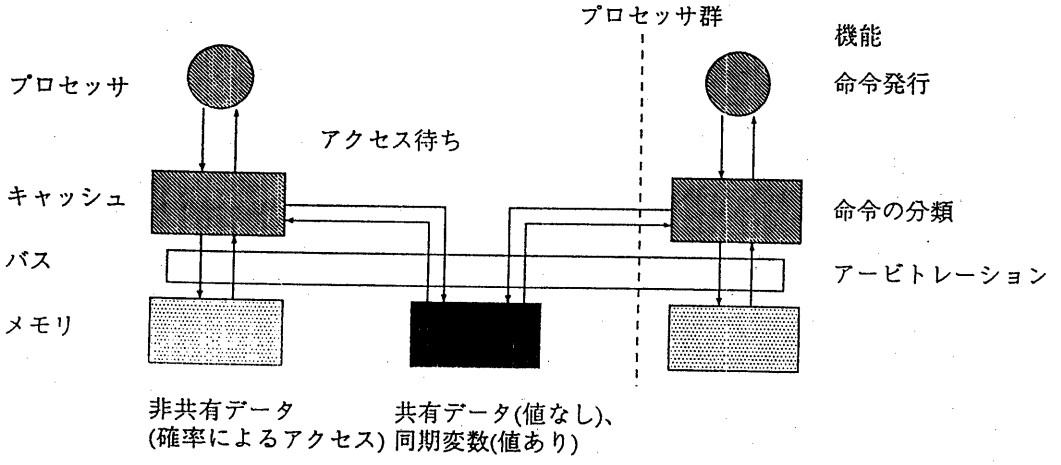

図 4: シミュレーション概念図

同時にメモリモジュールはその時点でのブロックのキャッシュタグに基づいたコヒーレンス動作を行なう。これは、キャッシュコヒーレンスプロトコルにもよる。

共有ブロックへの参照命令は、実際に他のキャッシュに存在するブロックに対するものであるとは限らない。共有ブロックとその時点で共有されるブロックは、異なるからである。

キャッシュミスが生じた場合、共有ブロックであっても、非共有ブロックであっても新しいブロックを置く場所を作る必要がある。

選択されたブロックが非共有ブロックであった場合、確率 md で内容がメモリと異なり、書き戻しが必要となる。また、確率 1 - md でメモリと内容が同じであり、書き戻さなくてよい。ただし、md はパラメータとして与えられた値とする。

共有ブロックが選択された場合、共有ブロックのタグによって、書き戻すかどうかが決定される。書き戻しは共有ブロックに対しても、非共有ブロックに対してもロードの前に行なわれる。

さらに重要な命令として同期命令がある。対象となるアーキテクチャの同期命令は、共有メモリ上の同期変数を用いて行なわれる。

hold 命令は同期変数の確保に成功するまで繰り返され、確保に成功したらクリティカルセクションに入る。クリティカルセクションを終了したら release 命令を発行し、同期変数を解放する。

この同期動作はシミュレータ内に同期変数を用意し、hold, release の両命令に対して、実際に同期変数の確保、解放を実行する。すなわち、本シミュレータは同期命令に関してはアドレストレースのみならず、データを操作する。

これにより、マルチプロセッサの依存関係を反映したシミュレーションが実行可能である。

5 考察と性能評価

以上の方針にしたがって実際にシミュレータを作成し、ワークステーション上で簡単な実験を行なった。

実験に用いたワークロードについて、ひとつのプロセッサについてのロード/ストア命令の時間的な分布の様子を図 5、図 6に示す。これらのグラフから、大域的に一様にみえる命令についても、時間的な局所性が見られる場合があることが分かる。

これらを基に 16 プロセッサの場合について、プロセッサ稼働率を調べるシミュレーションを確率モデル、トレースドリブン、本手法の三者について比較を行なった。プログラムはトレースドリブン型のシミュレータのモジュールをモデルに応じて、確率を組み込んだものに交換することで対応した。

アドレストレースはデバッガによるトレース情報をもとに作成した。全体で先頭から 200000 ステップ作成し、そのうちの中央のステップを 10000 シミュレーションサイクル利用した。これは主として、出力結果の保持、実行時間の制限という実現上の制限による。

各プロセッサモジュールに動作時刻をファイルに出力させ、その結果を集計することにより、システムの性能を調べた。

まず、この結果は確率モデルの 5 秒以下には及ばないが、まずまず我慢できる時間内に得られたことを表 2 に示す。

表 2:10000 シミュレーションサイクルの実行時間

	確率モデル	本方式 (単位は秒)
実行時間	4.8	35

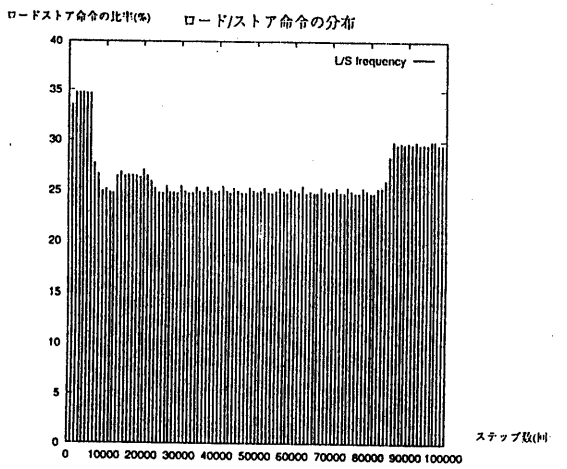

図 5: ロード/ストア命令の頻度 (粗粒度)

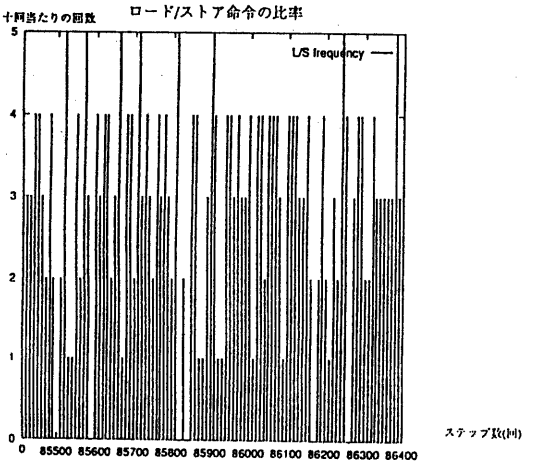

図 6: ロード/ストア命令の頻度 (細粒度)

本手法による結果を図 7に示す。実行範囲におけるこのワークロードに対するプロセッサ稼働率は 68.8%であり、トレースと実行結果を組み合わせることにより、プロセッサのストールしている状況がおよびその原因を知ることができる。

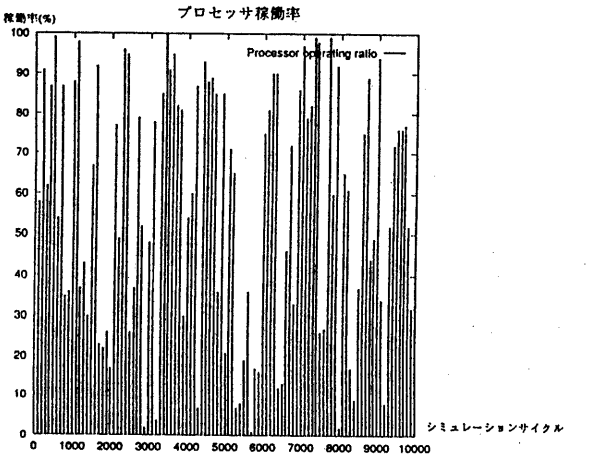

図 7: プロセッサ稼働率

動的なメモリ使用量については図 8のような結果を得た。トレースドリブン型と比較すると、両者ともプロセッサ数が増加するとともにメモリの使用量は増加する傾向にあるが、本方式では予想どおり大幅なメモリ使用量の節約が可能となったことが分かる。

小規模の場合でも本方式の有用性は認められるが、両者の差は、プロセッサレベルでのデータの扱い方の違いによる。すなわち、プロセッサ毎のデータ量の大きさの違いによることが推測されるので、シミュレーション対象のプロセッサ数が増加するにつれて、絶対量の違いが大きくなり、シミュレーションの実現可能性にも影響することが予測される。

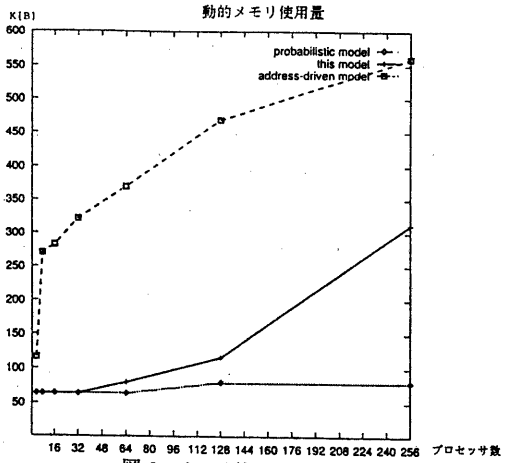

図 8: メモリ使用量の比較

参考に確率モデルの場合も示す。確率モデルはプロセッサ数の差があまりメモリ使用量に反映されない。

なお、結果はプロセッサ数に対する効果を強調するために横軸を対数化してある。

今後の課題として、既存のマルチプロセッサ上に本シミュレータを実装し、それを利用して適当なマルチプロセッサの性能評価を行ない、実機との比較を行なう必要があると考えられる。

大規模な問題への適用を考える上でも、マルチプロセッサへの対応が必要となるであろう。

謝辞

日頃より有益な御議論をいただく奈良先端科学技術大学院大学情報処理学専攻荒木研究室の皆様に感謝します。

参考文献

[1] Maekawa, M., Oldhoeft, A. E., Oldhoeft, R. R., 前川守(監訳): オペレーティングシステムの先進的概念, 丸善 (1990).

[2] Denning, P. and Buzen, J.: The Operational Analysis of queueing network models, *ACM Computing surveys*, Vol. 10, No. 3, pp. 225–261 (1978).

[3] Reiser, M. and Lavenberg, S.: Mean value analysis of closed multichain queueing network, *Journal Of the ACM*, Vol. 27, No. 2, pp. 313–322 (1975).

[4] Bhuyan, L. N., Liu, B.-C. and Ahmed, I.: Analysis of MIN Based Multiprocessors With Private Cache Memories, in *Proceedings of the 1989 International Conference on Parallel Processing*, Vol. I, pp. 51–58 (1989).

[5] 城和貴, 福田晃:並列計算機の解析モデルーシミュレーションとの比較ー, Technical Report 93-ARC-100-3, 情報処理学会ＡＲＣ (1993).

[6] 柴村英智, 久我守弘, 末吉敏則:超並列計算機のための相互結合網シミュレーション, Technical Report 92-ARC-97-16, 情報処理学会ＡＲＣ (1992).

[7] Dally, W. J.: Performance Analysis of k-ary n-cube Interconection Networks, *IEEE Trnsaction on Computer*, Vol. 39, No. 6, pp. 775–785 (1990).

[8] Archibald, J. and Baer, J.: Cache Coherence Protocols:Evaluation Using a Multiprocessor Simulation Model, *ACM Transactions on Computer Systems*, Vol. 4, No. 4, pp. 273–298 (1986).

[9] Smith, A. J.: Cache Evaluation and the Impact of Workload Choice, in *Proceedings of the International Symposium on Computer Architecture*, pp. 64–73 (1985).

[10] Jain, R.: *The Art of Computer Systems Performance Analysis*, John Wiley & Sons (1991).

[11] Arvind, and Ianucchi, R. A.: A Critique of Multiprocessing Von Neumann Style, in *IEEE Proceedings 10th Annual of International Symposium on Computer Architecture*, pp. 426–436 (1983).

[12] O'Krafka, B. W. and Newton, R.: An Empirical Evaluation of Two Memory-Efficient Directory Methods, in *IEEE Proceedings 17th Annual of International Symposium on Computer Architecture*, pp. 138–140 (1990).

[13] 堀江健志, Chuang, I., 井川英子, 林憲一:メッセージレベルシミュレータ, Technical Report 92-ARC-95-10, 情報処理学会ＡＲＣ (1992).

[14] Malony, A. D., Larson, J. L. and Read, D. A.: Tracing Application Program Execution on the Cray X-MP and Cray 2, in *IEEE Proceedings Supercomputing '90*, pp. 60–71 (1990).

[15] B.Stunkel, C. and Fuchs, W.: Analysis of Hypercube Cache Performance Using Address Traces Generated by TRAPEDS, in *Proceedings of the 1989 International Conference on Parallel Processing*, Vol. I, pp. 33–40 (1989).

[16] Heidelberger, P. and Lavenberg, S. S.: Computer Performance Evaluation Methodology, *IEEE Transaction on Computers*, Vol. C-33, pp. 1195–1220 (1984).

[17] Sherman, S. W. and Browne, J. C.: Trace-Driven Modeling:Review and Overview, in *Proceedings Sympsium on Simulation of Computer Systems*, pp. 201–207 (1973).

[18] Bratley, P., Fox, B. L. and Schrage, L. E.: *A Guide to Simulation*, Springer-Verlag, New York (1986).

[19] Lamport, L.: How to make a Multiprocessor Computer That Correctly Executes Multiprocess Diagram, *IEEE Transaction on Computers*, Vol. C-28, No. 9, pp. 241–248 (1979).

[20] Misra, J.: Distributed Discrete-Event Simulation, *ACM Computing surveys*, Vol. 18, No. 1, pp. 39–66 (1986).

[21] 松本幸則, 瀧和男:バーチャルタイムによる並列論理シミュレーション, 情報処理学会論文誌, Vol. 33, No. 3, pp. 387–395 (1992).

[22] 石川貴史:イベント時刻の下限値を用いた並列論理シミュレーション, 並列処理シンポジウム ＪＳＰＰ'９２, pp. 461–468 (1992).

[23] Dubois, M., Briggs, F., Patil, I. and Balakrishnan, M.: Trace-driven simulations of parallel and distributed algorithms in multiprocessors, in *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 909–916 (1986).