NAIST-IS-DT9561018

博士論文

並列計算機の解析モデルに関する研究

城 和貴

1996年6月25日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科において
博士 (工学) 授与の要件として提出された博士論文である。

提出者：　城 和貴

審査委員：　福田 晃 教授
渡邉 勝正 教授
尾家 祐二 教授

並列計算機の解析モデルに関する研究*

城 和貴

内容梗概

キャッシュ・メモリを有した共有メモリ型並列計算機に対する、セミマルコフ過程を利用した解析モデルを提案する。提案する SMCI (Semi-markov Memory and Cache coherence Interference) モデルは、評価対象の並列計算機の構成方式を与えることにより、キャッシュ・コヒーレンス制御命令や通常のデータ・リクエストに関する，ネットワーク競合やメモリ競合による待ち状態を始めとする、プロセッサの各状態を解析する。また、その上で稼働する並列プログラムに特有のパラメータも入力パラメータとして与えることができるため、多様なシステム構成を持つ並列計算機の性能予測や、それらのシステム上で実行されるアプリケーションの性能予測を、容易かつ詳細に得ることができる。これまでのマルコフ連鎖等を利用した並列計算機の解析モデルと異なり、本モデルにおける状態数は、プロセッサ台数やメモリ・サービス時間に依存せず、評価対象とする並列計算機の採用しているキャッシュ・コヒーレンス・プロトコルによってのみ決まる。従って、本モデルの計算時間は、本質的にシステムの規模に依存しない。本論文では、コヒーレンス・プロトコルとして、最も基本的な Synapse と、最も複雑な Dragon を対象とした。本モデルを用いて，プロセッサ利用率を評価した結果、本モデルによる計算結果は、実際のシミュレーションで求めた結果の極めて良好な近似になっており、しかも、計算時間もシミュレーション時間に比較して大幅に短縮された。

さらに、SMCI モデルを拡張して、スケーラブル共有メモリ型並列計算機に対する解析モデルも提案する。提案する階層型 SMCI モデルは、SMCI モデル同

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DT9561018, 1996 年 6 月 25 日.

様、プロセッサの取りうる各状態の極限確率を、安価な計算コストで算出する。プロセッサの取りうる状態には、ローカルおよびグローバルなデータ・リクエストやキャッシュ制御リクエスト状態、それらの待ち状態、計算状態等があり、当該アーキテクチャに対する詳細な評価尺度として十分なものである。本論文では、スケーラブル共有メモリ型並列計算機の例として、プロセッサ・クラスター型をベースとし、クラスタ内にはローカル・メモリとローカル・キャッシュ、クラスタ間には分散グローバル・メモリとグローバル・キャッシュを持つような階層的な構成の仮想的なアーキテクチャを示し、それに対する実際の階層型 SMCI モデルも構築する。

キーワード

解析モデル、並列計算機、セミマルコフ過程、キャッシュ・コヒーレンス、性能評価

Analytic Modeling for Parallel Computers*

Kazuki Joe

Abstract

In this paper, we propose an analytic model using a semi-markov process for shared memory parallel computers with cache memory. The proposed model, SMCI (Semi-markov Memory and Cache coherence Interference) Model, can be used for the performance evaluation or analysis of shared memory parallel computers with cache memory because it can be easily applied to descriptions of waiting states for network arbitration or memory contentions of both normal data accesses and cache coherence requests. Furthermore the model is so flexible that it allows various types of system configurations and offers efficient kinds of input parameters. Differing from conventional analytic models such as by a markov chain, the number of states in the SMCI model does not depend on the number of processor nor memory service time but on the kind of cache coherence protocol of the target architecture. Thus the computational complexity of the SMCI model is not affected by the system scale but by the kind of the protocol. In this paper, two protocols are reported for implemetation: 1) the Synapse protocol which is known to be the primitive one and 2) the Dragon protocol which is known to be the most complex one. Using the proposed analytic model, we investigate some comparative experiments with widely known simulation. The result of our analytic model shows very close pattern to the actual simulation model while the execution time of the model is extremely smaller than simulation.

Furthermore, we propose another analytic model for a scalable shared memory parallel computer. The proposed model is a hierarchical extention of the

*Doctral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9561018, June 25, 1996.

SMCI model. It can be used for the predictive performance evaluation of a cluster based parallel computer with hierarchical memory system, which is known to be a scalable shared memory parallel computer, by describing cache coherence mechanism and waiting states for network arbitrations or memory contentions as well as processor's ordinal behaviors. In order to apply the model to an actual parallel computer architecture, we introduce an example architecture of a cluster based parallel computer with hierarchical memory system connected by a hierarchical network. Then we show the way to construct the model for the example architecture.

Keywords:

Analytic Modeling, Parallel Computer, Semi-Markov Process, Cache Coherence, Performance Evaluation

目次

図目次

表目次

第 1 章

序論

本論文は、キャッシュを有するメモリ共有型の並列計算機アーキテクチャの性能予測を行なうための手段として、計算コストの安い解析モデルに着目し、その構築方法について行なった研究に関してまとめたものである。

本章では本研究の背景、目的、概要および本論文の構成について述べる。

1.1 本研究の背景と目的

並列計算機の評価基準には、与えられたハードウェア構成における、与えられた並列プログラムを実行させた時の、プロセッサ利用率やネットワークの利用率、それらに対する同期や共有データの一貫性を保証するためのオーバーヘッドの割合等が考えられる。

特定の並列計算機上で、特定の並列プログラムを稼働させた場合の、これらの性能評価は、実際の計測以外にも、アドレス・トレースを用いたシミュレーションで可能である。

一方、並列計算機アーキテクチャの研究は、この四半世紀に渡って、システムの統合方式や、キャッシュ制御方式を始めとした基礎研究がなされてきており、現在、スケーラブル共有メモリ型並列計算機という新しい方向に向かっている。また、並列プログラムは、その採用する並列アルゴリズムや並列化手法が、実行性能に大きな影響を与えることは、広く知られている。

しかし、アーキテクチャの大規模化複雑化、並列アルゴリズムや並列化手法の増加に対応して、パラメータの数や各パラメータの値域が増大し、結果としてそれぞれのパラメータ・チョイスに対応した評価すべき予測の組合わせ総数が爆発的に増えるため、並列計算機開発に先立つアーキテクチャの性能予測や、特定のアーキテクチャがどのような並列プログラムに適しているかを事前に知るための稼働予測は、極めて困難になりつつある。

例えば、プロセッサ数、キャッシュ・サイズ、メモリ・サービス時間、キャッシュ・コヒーレンス・プロトコルの種類というアーキテクチャ・パラメータに対して、それぞれ 4 種類の値を考えた場合、アーキテクチャの性能予測は 4^4 通り行なわなければならない。さらに、各アーキテクチャ・モデルにおいて、例えば 3 種類のアプリケーションを 10 種類の並列化手法で並列化して実行した場合の稼働予測は、$4^4 * 3^{10}$通り行なわなければならない。

シミュレーションは、与えられた少数のパラメータ・チョイスでの評価には有効である反面、このように組合せ的に増大するパラメータ・チョイス全てに対する評価には、計算コストの面で問題があることは明白である。

並列計算機アーキテクチャ研究において、提案されるアーキテクチャの性能予測が安価な計算コストで与えられれば、研究開発のターンアラウンド・タイムが

縮小され、より良いアーキテクチャの出現が期待できる。よって、このような組合せ的に増大するパラメータ・チョイス全てに適用できるほどに、計算コストの安い評価手法が切望されている。

並列計算機の解析モデルの中には、確率過程等を用いて安価な計算コストを実現しているものもある。特にマルコフ連鎖を利用した解析モデルは、キャッシュを持たない小規模な共有メモリ型アーキテクチャに数多く適用され、その状態数を減少させるための近似手法に関する研究が、精力的に行なわれてきた。

しかしながら、アーキテクチャ研究の関心が、キャッシュを持たない並列計算機から、キャッシュを持ちキャッシュ・コヒーレンス制御をハードウェアでサポートする並列計算機へと移行した時点で、システムの複雑化が、マルコフ連鎖を利用した解析モデルの構築を困難としていった。そのため、最近10年程は確率過程を利用した並列計算機の解析モデルに関する研究は極めて少なくなり、それに代わって、キューイング・ネットワークを利用した解析モデルが主流となっていた。

ところで、現在注目されているスケーラブル共有メモリ型並列計算機は、数万プロセッサ、数十万プロセッサの結合を意図したものであり、その圧倒的に大規模なシステムの性能予測をどのように実現するかが、現在の並列計算機アーキテクチャ性能評価研究の中心課題の一つである、と言っても過言ではないであろう。現在の主流となっている、キューイング・ネットワークを利用した解析モデルでは、サーバーの数、すなわちプロセッサ、キャッシュ、メモリ、ネットワークの数が計算コストに直結するため、近い将来必要となるであろう大規模かつ複雑なシステムの評価予測や、それらのアーキテクチャ上で稼働する並列アプリケーションの稼働予測に有効であるかどうかは定かでない。

本研究は、大規模な並列計算機アーキテクチャに対する解析モデルとして求められるのは、キューイング・ネットワークに代表されるような、個々の構成要素とその通信に着目する方式ではなく、その構造の規則性に着目した定式化を行なう方式という着眼点から出発する。なぜならば、前者は本質的に構成要素数が計算コストに反映されるのに対し、後者はシステムの複雑さのみが計算コストに反映されるため、その定式化がひとたび完成すれば、システムの規模の大きさに関係しない評価技術が確立するからである。

このような背景を踏まえて、本研究では、

1) 現在では一般的な技術であるキャッシュ・コヒーレンス制御が、並列計算機システムの中で、どのような影響を与えているかを解析するための定式化を図り、

2) その定式化を基に、システムの挙動を、プロセッサ数や共有データの数等に依存しない、少数の状態数で定義される確率過程を使ってモデル化を行ない、

3) それをさらに大規模なシステムに適用するための手法を確立する

ことを目的とするものである。

1.2 本論文の概要

本論文は 7章から成る。本章では、並列計算機アーキテクチャの性能予測を、システムの規模に左右されない計算コストで実現する技術が必要とされる背景について述べ、本研究の目的および概要について述べた。

第 2章では、過去四半世紀に渡って行なわれてきた並列計算機に対する解析モデルの研究について、対象とするアーキテクチャを大まかに四種類に分類した上で、順次解説する。

第 3章では、本論文にて用いられる既存の技術、すなわち、セミマルコフ過程、キャッシュ・コヒーレンス・プロトコル、階層的なキャッシュ・システムの無矛盾化技術について、それぞれ説明する。

第 4章では、キャッシュ・メモリを有し、特定のプロトコルに従ったキャッシュ・コヒーレンス制御をハードウェアでサポートする共有メモリ型並列計算機に対する、セミマルコフ過程を利用した解析モデルの提案と、その構築手法ならびに構築されたモデルの検証について論ずる。

第 5章では、第 4章で構築された解析モデルが、キャッシュ・コヒーレンス・プロトコルの種類に依存しないことを示すために、他の代表的なプロトコルにも同解析モデルが適用可能であることを示し、その構築手法ならびに構築されたモデルの検証を行なう。

1.2. 本論文の概要

第6章では、大規模なシステムとしてスケーラブル共有メモリ型並列計算機を想定し、その簡略化されたアーキテクチャ・モデルを定義した上で、第4章で論じた解析モデルの適用手法についてまとめる。

最後に第7章において、本研究の成果をまとめると共に、今後の課題を明らかにする。

なお、各章に関連した筆者自身の参考文献は以下の通りである。

本論文での章	筆者研究業績における文献番号
第2章	[30]
第4章	[2],[7],[9],[67],[68],[71],[72],[81]
第5章	[20],[84]
第6章	[10],[70]

第 2 章

関連研究

本章では、過去四半世紀に渡って行なわれてきた並列計算機に対する解析モデルの研究についてのサーベイを行なう。並列計算機の解析モデルの研究がいつ始まったのかは正確には不明であるが、過去の文献を調べる限り、当該内容で CMU より 1973 年に学位を取得した Bhandarkar の研究が最初であると思われる [9]。従って、本章では Bhandarkar の研究を出発点にして、以後四半世紀の間に研究されてきた解析モデルを、対象とするアーキテクチャを大まかに四種類に分類した上で、順次解説する。

2.1 はじめに

並列計算機はこの四半世紀の間にさまざまなアーキテクチャが提案され、現在も進化しつつある。このように次々と提案されるアーキテクチャに、適切な性能予測を与えることは、アーキテクチャの提案自体に優るとも劣らない重要な研究課題であろう。通常、並列計算機アーキテクチャ研究のために必要とされる性能予測は、シミュレーションや解析モデルで行なわれる。この時、評価対象が何であるかによって、その評価をもっとも効果的に行なえる、シミュレーション・モデルや解析モデルが選ばれる。例えば、クロスバー・ネットワーク等で結合されたキャッシュを持たない並列計算機の理論的なメモリバンド幅を求めるだけならば、確率と組合わせによる解析モデルで瞬時に答が得られる。一つの並列プログラムを全プロセッサで実行させた時の、プロセッサ利用率やネットワーク使用率を求めるには、与えられたプログラム・モデルが単純なものであれば、確率過程等を利用した解析モデルや、簡略化されたシミュレーション等が適当である。そのプログラムが非均等な同期命令を含むのであれば、時間ペトリ・ネットのような高性能の解析モデルや、詳細なシミュレーションに頼らざるをえないであろう。本章では、このような並列計算機アーキテクチャの性能予測を行なうために使われる、解析モデルについてのサーベイを行なう。

並列計算機の解析モデルは多岐に渡り、本章で説明する性能予測以外にも、ハードウェアの信頼性予測、並列プログラムや OS の評価等に利用されるが、本章では与えられた条件のもとでの、メモリ・バンド幅やプロセッサ利用率等の、主に共有メモリ型並列計算機の性能予測に関する解析モデルのサーベイを行なう。

まず 2.2節では共有バスやクロスバーで結合された、キャッシュを持たない小規模な構成の並列計算機に対する解析モデルについて解説する。次に 2.3節では、キャッシュを持たないが、クラスタ方式によって大規模な構成を可能とした並列計算機に対する解析モデルを紹介する。2.4節では、キャッシュを有し、ハードウェアによってキャッシュ・コヒーレンス制御を行なう、小規模な並列計算機の解析モデルについて説明する。2.5節では、最近注目されている、スケーラブル共有メモリ型並列計算機に対する解析モデルについて概説する。2.6節では、並列計算機に対する解析モデルの日本での現状について触れ、まとめとする。

2.2 小規模並列計算機の解析モデル

資源の共有問題に対する解析モデルの利用の歴史は古いが、並列計算機に対する解析モデルの本格的な利用は、1970 年代からである。本節では、クロスバー・ネットワークや、マルチプル・バス・ネットワーク等によって接続された、キャッシュを持たない共有メモリ型の小規模な構成の並列計算機アーキテクチャに対する解析モデルについて概説する。

計算機システムのモデル化には、マルコフ・モデルや待ち行列理論がよく用いられる。並列計算機に対するこれらの初期の適用例としては、Bhandarkar[10] や McCredie[45] らの研究が知られている。特に Bhandarkar の研究は以後の解析モデル研究に大きな影響を与え、文献 [10] は現在に至るまで最も数多く参照されている論文の一つである。Bhandarkar のモデルは、n 台のプロセッサと m 個のメモリ・モジュールをクロスバー結合したシステムに対して、メモリのサイクル時間を単位とした離散時間マルコフ連鎖を適用し、メモリ競合を解析する。この時の状態定義は、各メモリ・モジュールに対してリクエストの発行されているプロセッサ数の組で記述され、状態数の爆発的増加を防ぐために、さらにアクセス・パターンのクラス分けをしている。また、各プロセッサの処理時間は幾何分布を想定しており、モデルの妥当性はシミュレーションとの比較でなされている。一方、McCredie は 同様のアーキテクチャに対して、各プロセッサの処理時間とメモリのサービス時間の両方に指数分布を仮定することにより、Jackson のキューイング・ネットワーク（例えば文献 [40]）を適用し、メモリ・モジュール数とプロセッサ利用率との関係を解析している。

Hoogendoorn は Bhandarkar のシステム・モデルのもとで、ビジー状態にあるメモリ・モジュールの数（メモリ・バンド幅）をシステム・パフォーマンスと定義し、プロセッサがメモリを参照している確率の定式化を行ない、逐次的に解を求める手法を提案している [26]。Rau は同様のシステム・モデルにおけるメモリ・バンド幅を、マルコフ連鎖の定常分布を導出して求めている [60]。ここで、マルコフ連鎖の状態定義は Bhandarkar と同様であるが、近似解を与えることで計算コストの低い解析モデルを実現している。

Bhandarkar らのシステム・モデルでは、各プロセッサは各メモリ・モジュールに同じ確率でリクエストを行なうという一様参照モデルを仮定しているが、Sethi

らはこの仮定に対し、各プロセッサが特定のメモリ・モジュールにリクエストを集中する局所参照モデルに対応した解析モデルを提案している [61]。このモデルは McCredie と同様、メモリのサービス時間に指数分布を仮定し、Jackson のキューイング・ネットワークを利用して、定常状態の解を求めている。1983 年に Siomalas らはこの局所参照モデルに対するマルコフ連鎖を用いた解析モデルを提案しているが、近似解を導出するには至っていない [62]。なお、一様参照モデルはロー・ビットの、局所参照モデルはハイ・ビットのメモリ・インタリーブに対応していると考えられる。

これらの研究からは、システムの定常状態における特定の項目に関する近似解が何種類か得られている。Yen らは 1982 年にメモリ・バンド幅に対するより正確な近似解を導出するのに、リジェクトされたメモリ・リクエストが捨て去られるもの、リクエスト先を限定しないで再発行されるもの、といったそれまでの近似解を見直し、リクエスト先がリジェクトされたメモリ・モジュールと同じになるようにリクエストを再発行する確率モデルを提案している [71]。

Marsan らは、1982 年にマルチプル・バスによって結合した、並列計算機の正確なモデル化を、マルコフ連鎖を用いて表している [44]。マルコフ連鎖は、モデル化する対象が大きくなれば、状態数が爆発的に増大し、評価のための実際の計算が困難になる。そこで、Marsan らはマルコフ連鎖によるモデル化を簡略化する手法についていくつか提案している。正確なモデルでは各プロセッサやメモリ・モジュールごとに状態を割り当てていたが、プロセッサの取り得る各状態（共有メモリにアクセス／バスやメモリ競合）に対して、いくつのプロセッサがその状態にあるかで表したり、アクティブなプロセッサの数だけで状態を表したりしている。特に後者の場合、出生死滅過程（例えば [55]）に帰着できるため定常状態の解を求めることができる。これらの簡略化モデルに対する評価実験結果は、正確なモデルの結果に対して数パーセントの誤差であった。これらの簡略化モデルの一つは、単一バスによる 4 種類のアーキテクチャに適用され [41]、実際の並列計算機 TOMP [16] 設計の指針となった。さらに Marsan らは TOMP のシステム・モデルに、確率ペトリネット、マルコフ連鎖、BCMP キューイング・ネットワーク（例えば [40]）という 3 種類の解析モデルを適用し、実機との比較を行ない、これら解析モデルの妥当性を確認している [43]。

同じ時期に研究されたIraniとÖnyüksel らのアプローチは、マルチプル・バスで構成される並列計算機に対するキューイング・ネットワーク・モデルを、それと等価な出生死滅過程で表すことであった[27, 54]。これらのモデルは、キューイング・ネットワークで問題となる、計算量の爆発を抑えるのに有効であるが、シミュレーションや他の正確なモデルとの比較はなされていない。

BCMPキューイング・ネットワークを用いたマルチプル・バス結合型並列計算機の解析モデルはTowsleyによっても提案されている[63]。ここでは各プロセッサを同一のワークロード（平均リクエスト間隔が同じで、一様参照）でとらえる場合と、独立したワークロードでとらえる場合に対する、二種類の近似解を与える手法が述べられている。

Langらは、1982年に共有メモリ型並列計算機において、クロスバー・ネットワークとマルチプル・バス・ネットワークの、メモリ・バンド幅の評価を行なうために、正確な解析モデルを使う代わりに、シミュレーションを用いた[31]。その結果、マルチプル・バスの本数が、プロセッサ数の半分あれば、クロスバー・ネットワークとほとんど変わらないメモリ・バンド幅が得られることが報告されている。

1984年にBhuyanは、シミュレーションは勿論、簡略化されたキューイング・モデルでさえ、巨大な連立方程式を解く必要があることを指摘し、確率と組合せを用いて、マルチプル・バスとクロスバーによる、共有メモリ型並列計算機のメモリ・バンド幅の定式化を行なっている[11]。Bhuyanらは、さらにこの研究を発展させ、SethiやSiomalasらの局所参照モデルと同様のFavorite Memoryの概念を導入している[12]。このモデルはクロスバーおよびデルタ・ネットワークによって結合された共有メモリ型並列計算機に適用され、（特にデルタ・ネットワークの場合）Favorite Memoryが存在する方が、全体のバンド幅が大きいことを示し、それをシミュレーション結果と比較確認している。

Mudgeらは、1984年にBhuyanと同様、確率と組合せによる簡単なモデルを、マルチプル・バスによる共有メモリ型並列計算機に対して提案している[52, 51]。Bhuyanのモデルとの違いは、Mudgeらのモデルは、メモリ競合部分とネットワーク競合部分に分けて記述されている所にある。また、Bhuyanのモデルでは、リクエストがリジェクトされた場合、そのリクエストは捨て去られてしまうが、

Mudge らのモデルでは、Yen らのモデル同様、リクエストの再発行について考慮している。このため、解は逐次的な手法によって計算されている。Mudge らはさらに同年、クロスバーによる共有メモリ型並列計算機に対して、Equivalent Rate モデル（ER モデル）とマルコフ連鎖モデル（MC モデル）の、2 種類の解析モデルを提案している [50]。これまでのモデルがいずれもメモリ・バンド幅を予測するために提案されていたのに対し、Mudge らのこの 2 種類のモデルは、さらにリクエストのアクセプト率と、プロセッサ利用率を予測することが可能である。これは、一回のリクエストに必要とする時間を、ある確率変数を用いて表すことで実現されている。また、プロセッサからのリクエストがリジェクトされた場合、待ち状態に入る様子もモデル化される。

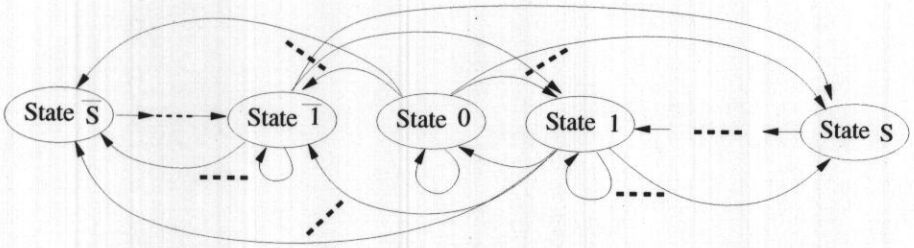


図 2.1 Mudge によるプロセッサの状態遷移を表すマルコフ連鎖

ER モデルは、プロセッサのリクエスト率を、平均通信時間/(平均通信時間＋平均計算時間) で表し、確率および組合せによるモデル化の手法に適用した簡単なものである。これに対し、MC モデルは、プロセッサの状態を図 2.1のように表し、各状態の定常状態から性能予測を行なう。図 2.1の各 State i は、あとどれだけの時間（サイクル）メモリ・アクセスを続けるかを示しており、従って、State 0 は計算状態を表す。また、State $\bar{i}$ は、メモリ・アクセスを行なうために、あとどれだけの時間待たされるかを示す。これらのモデルをシミュレーションと比較した結果、ER モデルでは、通信時間に対する変動係数が小さい場合 5%以内、大きい場合には 50%以内、MC モデルでは変動係数の値に左右されず 5%以内の誤差が報告されている。

マルコフ連鎖を利用した解析モデルは、適当な近似を与えることにより、どれだけ状態数の少ないものを構築できるかが重要である。Mudge らの MC モデル

2.2. 小規模並列計算機の解析モデル

以前のマルコフ連鎖を用いた解析モデルでは、状態数はプロセッサやメモリ・モジュールの数に依存しており、良いものでもプロセッサと同数程度の状態数が必要であった。これに対し、Mudge らの MC モデルはプロセッサやメモリ・モジュールの数に依存せず、(メモリ・アクセスのサイクル数) $*2+1$ 個の状態数で構築可能であり、マルコフ連鎖による並列計算機の解析モデルに対する新しい方向性を示したと言えよう。

さらに 1985 年に Mudge らは、マルコフ連鎖の代わりにセミマルコフ過程（例えば [55] 参照）を利用して MC モデルを改良し、マルチプル・バスによる共有メモリ型並列計算機の解析モデルとして Semi-markov Memory Interference（SMI）モデルを提案している [49]。MC モデルでは、状態遷移はシステム・サイクルごとに起こるため、メモリ・アクセスやその待ち状態を記述するのに、それらが必要とするサイクル数と同じ状態数が必要であった。そこで Mudge らは、各状態に任意の時間滞在できる、セミマルコフ過程を利用して、状態数を激的に減らすことに成功した。この SMI モデルは、図 2.2で示すように、メモリ・アクセス時間に関係なく、プロセッサの状態は 4 個だけで記述される。

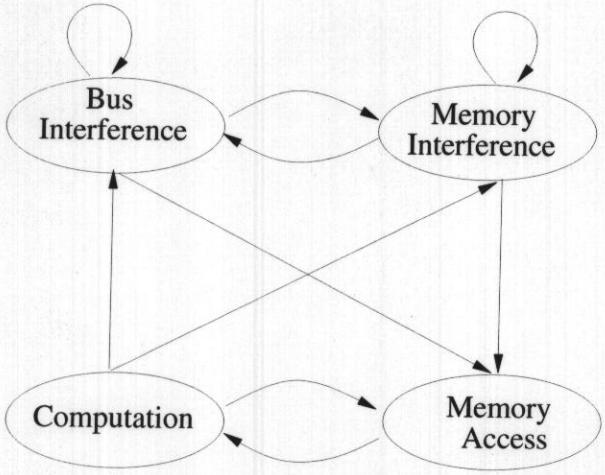


図 2.2 Mudge によるセミマルコフ過程を利用したプロセッサの状態遷移

SMI モデルでは、プロセッサは状態 0（計算中）、状態 1（メモリ・アクセス中）、状態 2（メモリ競合のための待ち状態）、状態 3（バス競合等のための待ち

状態）の 4 状態で表される。状態 0 では、プロセッサはある確率変数に従った時間計算を行ない、リクエストを発行して状態 1、2、3 のどれかに遷移する。競合が起こらなかった場合、状態 1 に遷移し、ある確率変数に従った時間メモリ・アクセスを行なって、状態 0 に戻る。リクエストを発行し、他のリクエストとのメモリ競合が起こった場合、状態 2 に遷移する。この時、状態 1 の時間と同じだけ待ち、再びリクエストを発行する。リクエストを発行し、ビジー状態のメモリ・モジュールをアクセスしようとしたり、バス競合が起こった場合、状態 3 に遷移する。状態 3 からは、状態 1 の時間を表す確率変数に対する 2 次モーメントに従った時間待ち、再びリクエストを発行する。この SMI モデルから定常状態を計算することにより、メモリ・バンド幅、プロセッサ利用率、メモリ・モジュール利用率、リクエスト待ち状態でのキューの長さ、待ち時間等が得られる。実際の評価に対しては、リクエスト率が他との相互作用（待ち状態）によって変動するので、Mudge らはこれに逐次近似法を適用して解を得ている。その結果、シミュレーションと比較して 6%以内の誤差であることが報告されている。この SMI モデルが、Bhandarkar や Marsan らのマルコフ連鎖をベースにしたモデルに対して、どの程度正確な解析能力を持つかということは研究されていないようだが、その状態数の少なさ（＝計算量の少なさ）は特筆に値するものであろう。

1992 年に Ramani らは Mudge の SMI モデルを拡張し、リクエストにプライオリティを持った並列計算機のモデル化を試みている [59]。対象となるアーキテクチャは、クロスバーによって接続された共有メモリ型並列計算機で、タスクごとに異なるプライオリティのリクエストを発行することができるので、OS や並列コンパイラ等、スケジューリングを考慮しなければならないシステムの性能予測に有効であろう。

前述したように確率ペトリネットによる解析モデルを提案した Marsan らは、さらに 1983 年に一般確率ペトリネットを提案し、並列計算機の解析モデルに利用した [42]。一方、Holliday らは一般時間ペトリネットを提案し、1987 年にマルチプル・バスによる並列計算機の正確な解析モデルを構築した [25]。このモデルは本節で紹介した他のモデル（[10, 63, 11, 50, 42]）と同一拘束条件のもとでの比較がなされ、極めて優れた解析能力が報告されている。

本節で説明した解析モデルは、メモリのサービス時間に一定値、幾何分布、指

数分布を仮定している。より一般的な分布に関しては、文献 [53, 33] を参照されたい。

2.3 クラスタ型並列計算機の解析モデル

並列計算機の性能向上のために、大規模化は当然の要求であるが、単一のネットワークに接続可能なプロセッサ数には限界がある。クラスタ型並列計算機アーキテクチャは、簡単な構造の並列計算機を一つのクラスターと見なし、それらを複数個ネットワーク結合したものである。本節では、これらクラスタ型の並列計算機に対する解析モデルについて概説する。

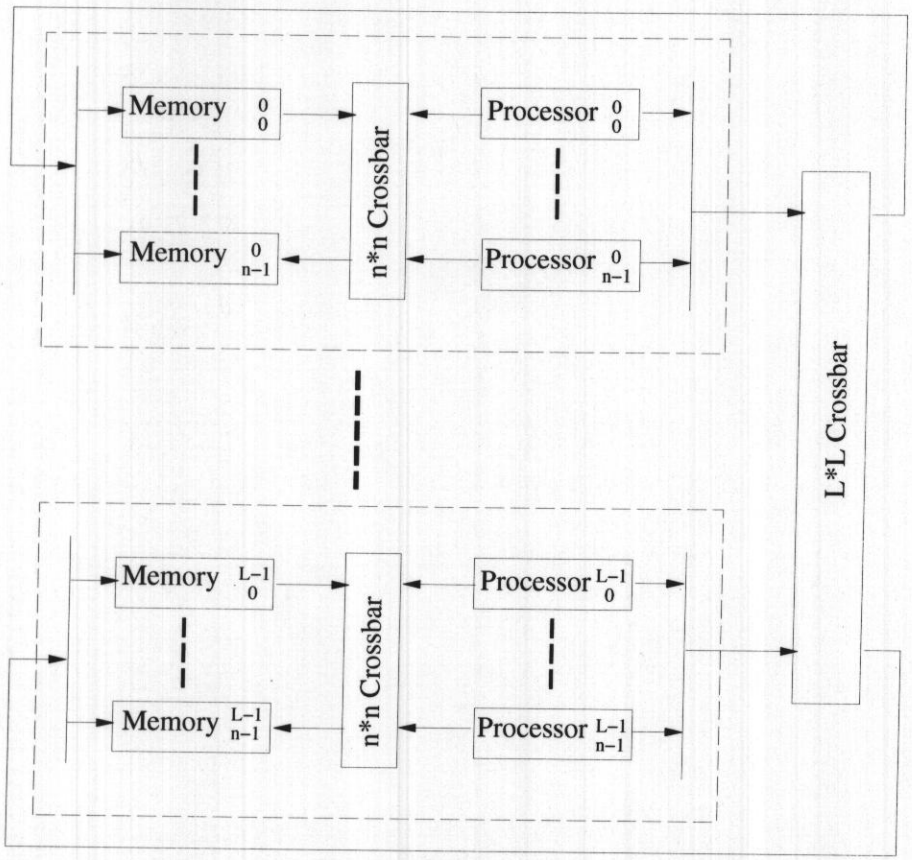


図 2.3 Agrawal のクラスタ型並列計算機アーキテクチャ

1985 年に Agrawal らは、図 2.3で示されるようなクラスタ型並列計算機に対する解析モデルを提案している [4]。対象とするアーキテクチャは、クラスタ内部では $n*n$ のクロスバーによって、n 個のプロセッサと、同数のメモリ・モジュールを結合し、クラスタ間では $L*L$ のクロスバーによって、L 個のクラスタを結合したものである。各クロスバーの評価は、簡単な確率組合わせモデルで表されており、プロセッサが発行するリクエストを、クラスタ内かクラスタ外かに分類している。リクエスト先のメモリ・アーキテクチャは、1) 単なるクラスタ型、2)Favorite Memory 付きのクラスタ型、3) プロセッサごとのローカル・メモリを持ち、かつ Favorite Memory を持ったクラスタ型、の 3 種類に分類され、それぞれのシステム全体のメモリ・バンド幅を確率組合わせを用いて定式化している。ただし、この文献では、提案されたモデルの妥当性を示す比較実験はなされていない。

Abraham らが、1986 年に発表したクラスタ型並列計算機の解析モデル [1] は、他のモデルと大きく異なる。Abraham らの対象とする、クラスタ型並列計算機アーキテクチャは、イリノイ大の Cedar [30] に相当し、モデル化の目的は、与えられたパフォーマンス・ゴールを達成するための、クラスタの構成方法である。これは、各プロセッサとネットワークに対し、スピードとコストを適当な変数で与え、それから全体のパフォーマンス（プロセッサの計算能力と通信遅延）をモデル化することで実現している。他の大部分の解析モデルが、プロセッサのリクエスト率をモデル化のベースとしているのに対し、Abraham のモデルはコストをベースとして、アプリケーションごとのクラスタ構成方法の妥当性を議論している点が評価されるが、コストの与え方の根拠が詳しく述べられておらず、モデル全体の正当性も証明されていない。

Agrawal らは 1987 年に、文献 [4] の成果を発展させ、マルチプル・バスによる並列計算機との比較を行ない、Favorite Memory のアクセス・パターンを持つ場合は、マルチプル・バス（ただしバスの本数はプロセッサ台数の半分）と同程度のパフォーマンスが得られることを報告している。さらに、Abraham のコストによるモデルの考えを取り入れ、Favorite Memory のアクセス・パターンを持つ場合は、単一のクロスバーによる結合より、コストあたりのメモリ・バンド幅が優れていることを示している [5]。

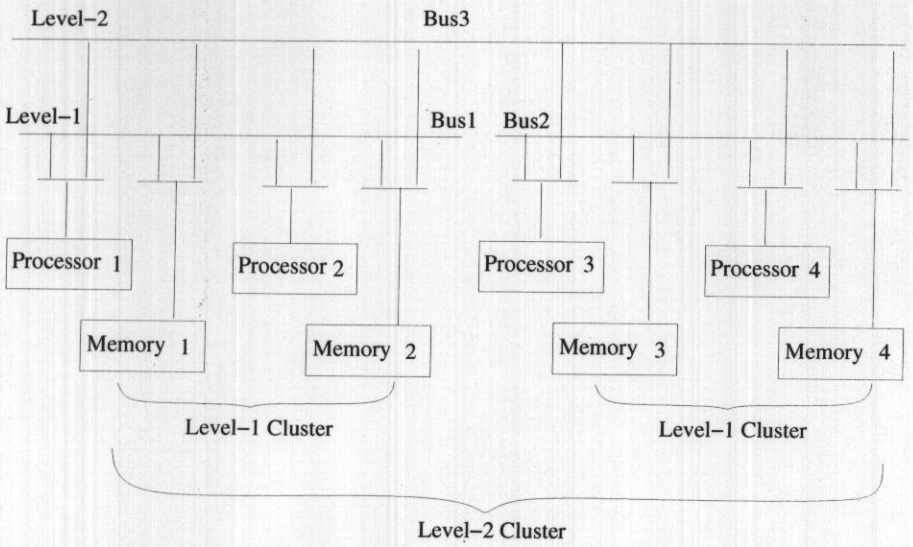


図 2.4 Mahmud の階層型バスによるクラスタ型並列計算機アーキテクチャ

Mahmud は 1990 年に、図 2.4で示されるような階層的にバスを結合したクラスタ型並列計算機に対する、解析モデルおよびシミュレーションによる性能分析を行なっている [38]。解析モデルとしては、Irani らと同様、クローズド・キューイング・ネットワークを用いて、積形式の解から逐次的に計算することにより解を求めている。また、システム比較のため、同数のプロセッサからなるマルチプル・バスによる並列計算機の解析モデルも構築され、その結果、マルチプル・バスに対する、階層型バスの優位性が示されている。なお、分析結果はメモリの平均アクセス・タイムとスループット時間で、階層バス型に関してはさまざまな構成での分析が報告されている。Mahmud はさらに 1994 年に同様の問題に対するクローズド・キューイング・ネットワークを MVA （Mean Value Analysis）法（例えば [32]）を用いて解いている [37]。

Mahgoub らは、1992 年に Agrawal のクラスタ型アーキテクチャを一般化した、m-レベル階層型並列計算機アーキテクチャ（図 2.5参照）を提案し、その解析モデルを文献 [5] と同様の手法で表している。メモリのアクセス・パターンに関しては、一様参照モデルと、 局所参照モデルを階層化した、Hierarchically Nonuniform Reference （HNR）モデルを採用している。m-レベル階層型ネット

ワークは、$m = 0$ の時は単一クロスバー、$m = 1$ の時はクラスタ型にそれぞれ等しい。性能予測は、単一のクロスバーのシステムとの比較でなされ、HNR モデルの場合、m-レベル階層型はクロスバーに迫る性能が可能なことを示している。

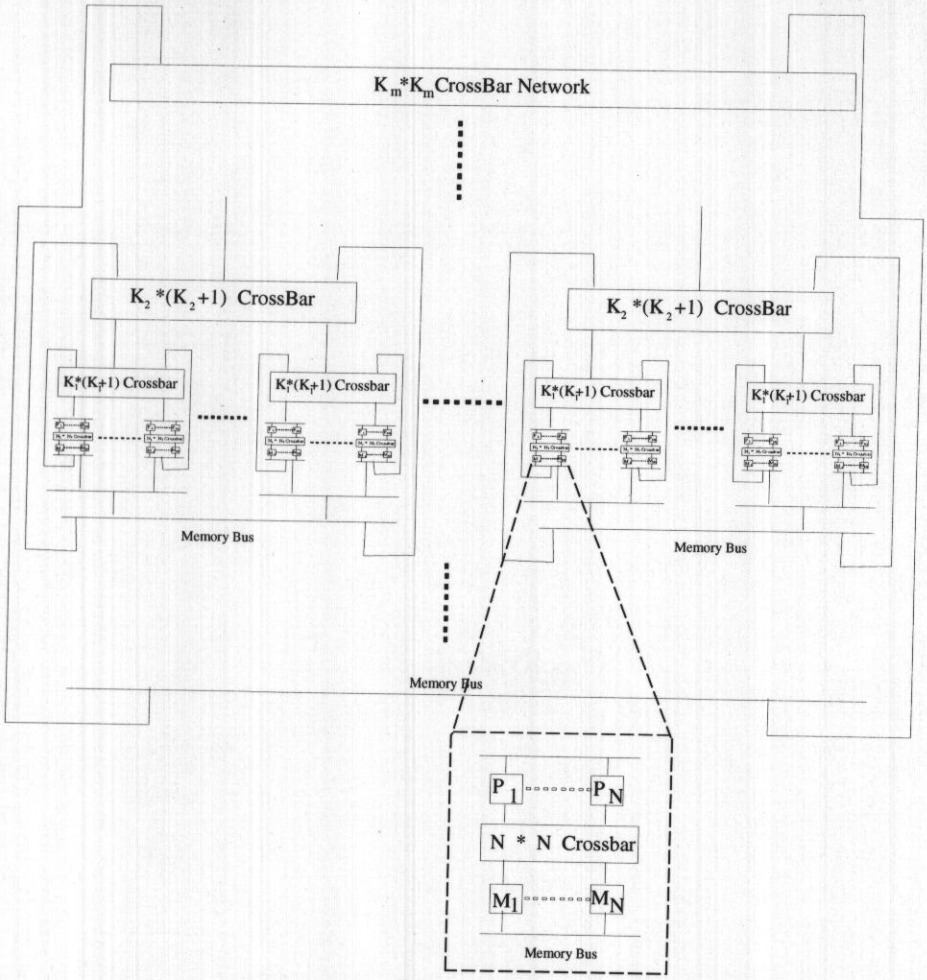


図 2.5 Mahgoub のm－レベル階層型並列計算機アーキテクチャ

Mohapatra らは、1994 年にクラスタ型並列計算機に対する階層的なキューイング・モデルによる解析を行なっている [47]。対象としているアーキテクチャは、ローカルおよびグローバルのメモリやネットワーク（MIN）、クラスタ内外を接続するバスといった階層的な構成になっており、それぞれの構成要素に対してキューイング・モデルを適用し、それの結果をもとに、逐次的な計算を繰り返してシス

テム全体の解析を行なっている。なお、各キューは M/D/1/L（指数分布到着、サービス時間は一定値）で表されている。

2.4 キャッシュ付き小規模並列計算機の解析モデル

並列計算機の各プロセッサに固有のキャッシュ・メモリを配備し、ネットワークのトラフィック軽減を図ることは、システムの性能向上に対する自然なアプローチであろう。一方、並列計算機におけるプロセッサ固有のキャッシュ・メモリは、キャッシュ・コンシステンシを保証してやる必要があり、キャッシュのない並列計算機システムに比べて、複雑なアーキテクチャを必要とする。この複雑さは、キャッシュ付き並列計算機の解析モデルの構築を遅らせた直接の要因と思われる。本節では、キャッシュを持つ小規模な並列計算機（主にバス結合）に対する解析モデルについて解説する。

他の何人かの研究者が、キャッシュを持たない並列計算機に対する解析モデルの研究を行なっていた 1982 年、Patel はキャッシュを持つ並列計算機（デルタ・ネットワーク結合）の数学的な解析を試みている [56]。ただし、キャッシュ・コンシステンシは考慮していない。この解析モデルはプロセッサのリクエスト率と、キャッシュとメモリの間のブロックの転送時間をもとに構築され、プロセッサの利用率を求めている。この解析モデルの検証は、同じワークロード・モデルのもとで構築されたシミュレーション結果との比較でなされており、解析モデルによる評価結果は、シミュレーション結果に対して 2%以内の差異しかなかったことが報告されている。

同年、Dubois らは既にバス結合型並列計算機における、キャッシュ・コヒーレンス制御の解析モデルを提案している [20]。提案されたモデルでは、具体的なプロトコルが採用されているわけではないが、キャッシュ制御のワークロードを与え、マルコフ連鎖を用いて、コヒーレンス制御を行なう場合と行なわない場合について、キャッシュ・サイズがヒット率に与える影響の定式化を行なっている。

この研究において注目すべきところは、並列プログラムにおいて、データ領域を共有ブロックと、プライベート・ブロックに分け、キャッシュの動きを、それぞれのブロックに応じた形でモデル化したことである。つまり、図 2.6で示される

ように、データ・アクセスは確率 q_s で共有ブロックに、$1-q_s$ でプライベート・ブロックに対してなされる。共有／プライベート・ブロックの概念は、以後のキャッシュ付き並列計算機（バス結合に限らない）に対する解析モデルに、大きな影響を与えている。

コヒーレンス制御を行なわない場合のモデル化では、キャッシュ・サイズと、キャッシュ・ヒット率に関する定式化を行ない、プロセッサからのリクエストが起きるたびに状態を変えるマルコフ連鎖で、システム全体が簡単に表されている。ここでの目的は、共有ブロックへのアクセスが、全体のキャッシュ・ヒット率にどのような影響を与えるかを分析することである。

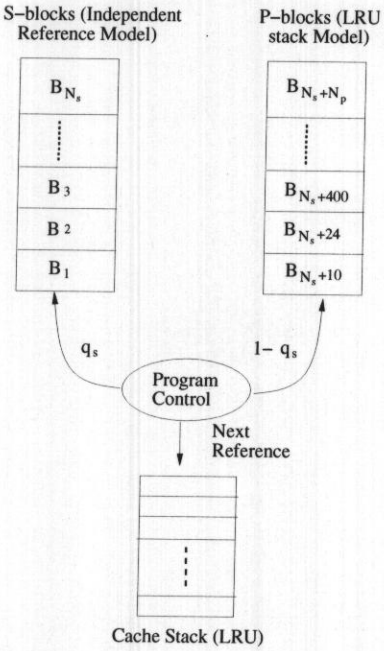


図 2.6 Dubois による共有ブロックとプライベート・ブロックの概念

コヒーレンス制御を行なう場合のモデル化（図2.7参照）では、特定のプロセッサに着目し、連続して起こる 2 回のリクエストの間に、他のプロセッサはそれぞれ 1 回ずつリクエストを起こすという仮定を与え、インバリデーションや、それに伴うリプレースを考慮にいれた定式化について報告している。このモデルは、

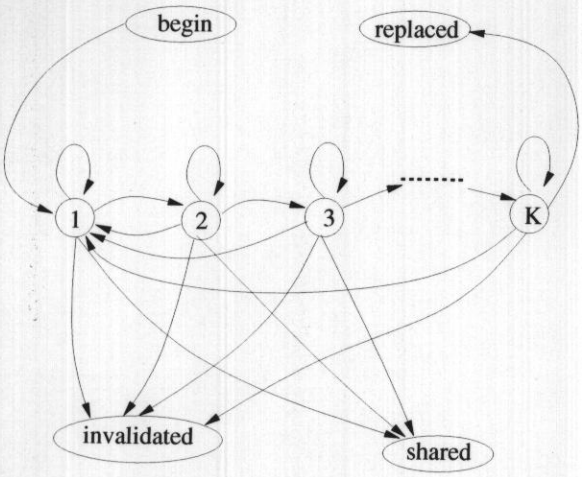


図 2.7 Dubois によるマルコフ連鎖を用いたコヒーレンス・チェック付き並列計算機中のキャッシュ・ブロックの状態遷移

仮想時間[1]で表されているため、プロセッサの利用率や、リクエストの待ち時間等の解析が不可能であり、かつ、各プロセッサのリクエストの出し方に制限が大きいため、キャッシュ付き並列計算機を正しくモデル化しているかどうかは疑問が残る。しかしながら、キャッシュ・コヒーレンス問題に対する研究がホットな時期のモデルであることを考慮すると、Dubois の研究は特筆に値するものであろう。

1988 年に Dubois らは、具体的な並列プログラムに対する解析モデルの提案を行なっている [22]。これは文献 [17] で提案したプログラム・ビヘイビア・モデルの、具体的な並列プログラムへの対応を許すもので、アクセス・バーストという新しい概念を導入している。ある並列プログラムが無限長のプライベート・キャッシュを持つ（プロセッサ数 J の）並列計算機で安定稼働している時、特定の共有ブロックは図 2.8で示すように、ダーティ（1_RW）もしくは k 個のプロセッサによって共有（k_RO）されている状態にある。状態遷移は、プログラムにおいて、クリティカル・セクション（CS）もしくはセミ・クリティカル・セクション（SCS）を脱出する瞬間に起きるものとする。あるプロセッサが CS や SCS を実行中の時、そのプロセッサはアクセス・バーストを行なっていると考える。このモデル

[1]これに対して、例えばネットワークのサイクルごとに状態を変えるようなモデルは、実時間で表されていると言う。

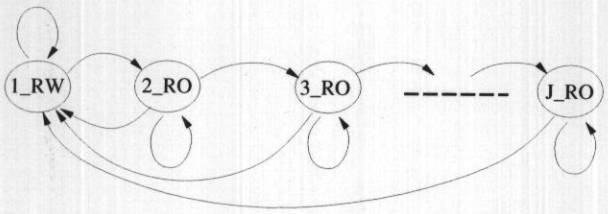


図 2.8 Dubois のアクセス・バースト・モデルによる共有ブロックの状態遷移

により、与えられたプログラムのアクセス・バースト長（CS や SCS におけるリクエストの数）、アクセス・バーストにおけるリード・ライトの順番等の情報を、パラメータとして与えてやると、共有ブロックのミス率およびコヒーレンス制御のためのオーバーヘッドを計算することができる。Dubois らはさらに、実際の並列プログラム（SOR、FFT、クィック・ソート）に対してこの解析モデルを適用し、文献 [19] で報告しているシミュレーション結果との比較を行ない、提案した解析モデルの妥当性を示している。このアクセス・バースト・モデルは、文献 [20] 同様、仮想時間により遷移するマルコフ連鎖を用いているため、プロセッサの待ち状態や利用率を求めることができず、無限長のキャッシュを想定しているためプライベート・ブロックがキャッシュ・ミスしたために引き起こされるリプレースの対象として、共有ブロックが選ばれる様子がモデル化されない。しかしながら、実際の並列プログラムの性能予測を可能とした初めての解析モデルとして、高い評価に値する研究と言えよう。

Wang らは、1990 年にアクセス・バースト・モデルを Write-Once、Synapse、Illinois、Berkeley 等、さまざまなコヒーレンス・プロトコルに拡張し、SOR の並列プログラムに適用した結果、Archibald のシミュレーション結果と同様の傾向が得られたことを報告している [68]。さらに翌年、Wang らはアクセス・バースト・モデルを有限長のキャッシュで構成されるシステムに拡張し、実際の並列プログラム（ヤコビ法、SOR、クイック・ソート、バイトニック・マージ・ソート、FFT、画像のラベリング）に適用している [69]。

Dubois は一方で非常に簡単な解析モデルの提案も行なっている。1988 年に提案された手法では、マルチプル・バスによって結合された並列計算機のスループッ

トの最大値を求めている [18]。これは、システム構成やプログラムのワークロードから決定的に得られる値をもとに計算するものであり、他の解析モデルと違って複雑な計算はほとんど必要としない。この手法によりライト・スルーとライト・バックの比較、SOR 法の並列化についての分析が報告されている。

Dubois らの一連の研究は、並列計算機の解析モデルと言うよりは、キャッシュ・コヒーレンス制御と並列プログラムの解析を行なうモデルと言えよう。キャッシュを持たない並列計算機に対する解析モデルでは、マルコフ連鎖等の確率過程を利用したものが多数を占めていたのに対し、キャッシュを持ちコヒーレンス制御を行なう並列計算機に対する解析モデルでは、マルコフ連鎖だけを用いたのでは状態数の爆発的増大等の問題が付随するため、システムの特定の部分のみを解析対象にしたものと思われる。

1986 年に Vernon らは、一般時間ペトリネットを用いて、バス結合型並列計算機のキャッシュ・コヒーレンス・プロトコルの性能評価を試みている [64]。ここでのプログラミング・モデルは、Dubois らの共有／プライベート・ブロックを利用しており、Archibald らのシミュレーション [7] と同様のシステム・モデルで、一般時間ペトリネットでのモデル化を行なっている。評価対象には、Write-Once プロトコルと、その改良版 5 種類のプロトコルを採用している。ただし、ここでの拡張版プロトコルとは、実際には拡張 Write-Once、Synapse、Berkeley、Illinois、Dragon に相当する。一般時間ペトリネットによる評価では、詳細な分析が可能であるが、プロセッサ数が増大するにつれて状態数が爆発的に増加するため[2]大規模な並列計算機システムの解析モデル化には適さない。

さらに 1988 年に Vernon らは、同様のシステム・モデルに対して、MVA 法を用いたキューイング・ネットワークを利用した解析モデルを提案している [67]。この解析モデルは近似であるにもかかわらず、一般時間ペトリネットによる解析結果と 3%程度の差異しか検出されなかった。計算時間は、プロセッサ数 10 のシステムを解析するのに、通常の WS で一般時間ペトリネットでは約 1 時間を要するが、MVA 法では約 1 秒しか必要としない。

MVA 法によるキューイング・ネットワークの解析は、その高い解析能力と安

[2]Write-Once プロトコルの場合、プロセッサ数 1 の場合、状態数 33 が、プロセッサ数 10 の場合、状態数 41,159 となる。

価な計算コストのために、並列計算機の解析モデルとして現在最も利用されている手法の一つである。バス結合による並列計算機への適用例としては、商用マシンに対するものや [29]、数千種類におよぶシステム構成の解析 [15] が報告されている。また、次節で述べるスケーラブル共有メモリ型並列計算機の解析モデルとしても広く利用されている。

2.5 スケーラブル共有メモリ型並列計算機の解析モデル

プロセッサのクラスタ化、キャッシュ・コヒーレンス制御、Multistage Interconnection Network （MIN）といったさまざまな技術を統合したスケーラブル共有メモリ型並列計算機（例えば [21]）が注目されている。次々と考案されるアーキテクチャに対し、例えば MVA 法によるキューイング・ネットワークを利用した解析モデルを適用することで、考案されたアーキテクチャの性能予測を行ない、システム改善の指標とすることができる。本節ではスケーラブル共有メモリ型並列計算機に対する解析モデルについて解説する。

1989 年に Bhuyan らは、MIN とバスの 2 系統のネットワークで結合された並列計算機を提案し、キューイング・ネットワークによる解析モデルによって性能予測を行なっている [13]。このアーキテクチャの特徴は、プロセッサからのリクエストが通常のリクエストである場合は、MIN を用いてデータ・アクセスを行ない、キャッシュ・コヒーレンス制御に関わるリクエストである場合は、スヌーピング・バスと呼ばれる、全プロセッサに接続しているコヒーレンス制御専用のバスを用いてアクセスを行なうことである。なお、コヒーレンス・プロトコルには、Write-Once を一部修正したものを用いている。図 2.9は、このアーキテクチャに対するキューイング・ネットワーク・モデルを示している。このキューイング・ネットワークでは、通常のリクエストはクローズド・クラスの、コヒーレンス制御に関するリクエストはオープン・クラスのカスタマーとして、それぞれ解釈される。ここで、オープン・クラスの到着率（インバリデーション、バス、ライトバックの 3 種類）は、共有ブロックに着目したマルコフ連鎖の定常確率から導出しており、このキューイング・ネットワークを MVA 法を使って解析する際の入

力パラメータとしている。

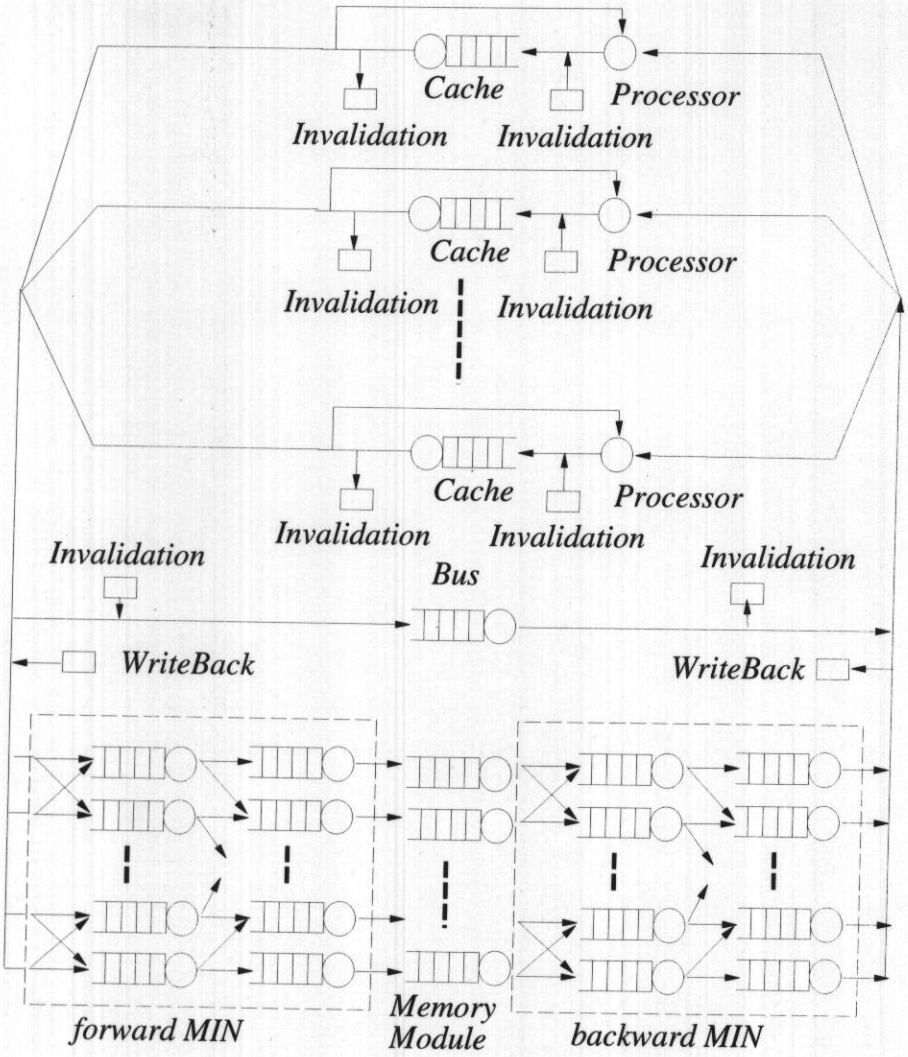


図 2.9 Bhuyan のキューイング・ネットワーク・モデル

ただし、プロセッサ利用率は容易に求めることができないので、逐次的に求めている。この解析モデルの評価結果はシミュレーションとほとんど一致しており、また、スヌーピング・バスの利用率が予想したほどではなく、実際、バスのサイクル・タイムを2倍にしても、ほとんど性能に影響しないことが報告されている。この研究は、アーキテクチャ設計に解析モデルが有効である良い事例と言えよう。

同年に Vernon らは、ウィスコンシン大で提案された TreeBus アーキテクチャ [65] の性能予測に、MVA 法によるキューイング・ネットワークを適用している [66]。この解析モデルではキャッシュ・コヒーレンス制御は考慮に入れられていないが、最大 2,048 までの、プロセッサ数に対するシステムの実行性能向上比率について報告されている。

1990 年に Mahmud らは、文献 [46] で提案されている、MIN 結合型並列計算機アーキテクチャに対する解析モデルを提案している [39]。評価対象のアーキテクチャは、プロセッサごとにキャッシュを持つだけではなく、MIN の中継スイッチにもキャッシュを持つ。Mahmud らは、この $(1+\frac{\log N}{2})N$ 個のキャッシュと、共有ブロックとの関係をマルコフ連鎖を用いて表している。このモデルに Dubois の提案したアクセス・バースト長と、ライト・アクセス・バースト率をパラメータとして与えると、全体でのキャッシュのヒット率やコミュニケーション・コストが得られる。

同じく 1990 年に Algudady らは、MIN で結合された各プロセッサが、プライベート・ブロックおよび共有ブロック用に、独立した二つのキャッシュを持つような並列計算機アーキテクチャに対する、キャッシュ・コヒーレンス・プロトコルの提案を行ない、MVA 法によるキューイング・ネットワークを使って性能を予測している [6]。ここで、プライベートおよび共有ブロックの区別は、アドレス空間で厳密に定義されている。共有ブロック用のキャッシュでは、MIN に起因する遅延問題が生じるが、これはコヒーレンス・プロトコル自体にトランジエント状態を付加することで対処している。システムの評価はプロセッサ利用率、バス利用率と共有データの割合いで分析され、63 プロセッサの場合、共有データの割合いが 2 割程度まではコヒーレンス・バスに余裕があることが報告されている。

Adve らは 1991 年に、同一のアーキテクチャのもとで、キャッシュ・コヒーレンス制御方式がシステムの性能にどのような影響を与えるかを、MVA 法によるキューイング・ネットワークで分析している [2]。ここでは MIN で結合され、ディレクトリ方式のキャッシュ・コヒーレンス制御を行なうアーキテクチャのもと、与えられた並列プログラム中の共有データから Weber の分類（read only, Migratory, Synchronization, Mostly-read, Read-Write）[70] に従って、ハイレベルのワークロードをもとめ、それをキューイング・ネットワークへのパラメータとなるよう

にローレベルのワークロードに変換して、ハードウェアによるコヒーレンス制御を行なう場合と、ソフトウェアによるコヒーレンス制御を行なう場合の性能比較を行なっている。

2.6 おわりに

並列計算機の解析モデル研究は、マルコフ連鎖でシステム全体を正確に記述するところから始まり、状態数を減らすための近似手法が研究された。しかしながら、キャッシュ技術やクラスタ化のようなアーキテクチャの複雑化・大規模化に対応した有効なマルコフ連鎖による解析モデルはあまり研究されておらず、現在、MVA 法を利用したキューイング・ネットワーク・モデルが、スケーラブル共有メモリ型を始めとした、多くの並列計算機の性能予測に利用されている。

一方、国内における並列計算機の解析モデルとしては、バス結合型のアーキテクチャに関するものが多く、待ち行列理論を用いたもの [78]、正確なマルコフ連鎖を用いたもの [74]、キューイング・ネットワークを MVA 法ではなくモーメント解析で解いたもの [72] 等があり、メモリ競合やバス競合に関しての解析を行なっている。また、マルコフ連鎖の状態遷移を生成してくれる並列システムの解析ツール [77] や、その実際の適用例 [75] についての研究もある。

一昔前までは、実用に耐え得る並列計算機の解析モデルは、シミュレーションと変わらない程計算量を必要としていたが、MVA 法によるキューイング・ネットワークの解析技術や、セミマルコフ、マルコフ・リワード・モデル等の利用により、システム設計の指針として実用に耐え得るようになってきている。また、4.4.3節で考察するように、これらの解析モデルはアーキテクチャの性能予測だけではなく、並列プログラムや並列化コンパイラ、並列 OS といったものにも適用可能である。今後はハードウェア・ソフトウェア・アプリケーションの全てを含む統合的な並列システムの性能予測を可能とする解析モデルの研究が期待される。

第 3 章

準備

本章では、本論文にて用いられる既存の技術に関する説明を行なう。3.1節では本論文で提案する解析モデルの中核をなすセミマルコフ過程について、3.2節では本論文を通して用いられる Synapse キャッシュ・コヒーレンス・プロトコルについて、3.3節では第 5章で用いられる Dragon キャッシュ・コヒーレンス・プロトコルについて、3.4節では第 6章での仮想マシン・アーキテクチャ構築に必要な階層的なキャッシュ・システムに対する MultiLevel Inclusion Properties の保証について、それぞれ説明する。

3.1 セミマルコフ過程

マルコフ連鎖による並列計算機システムのモデル化は、モデルの簡潔さという利点がある反面、各状態の時間が一定であるという制約があるため、プロセッサからのリクエストに対する待ち状態を記述しにくいという欠点がある。これに対し、セミマルコフ過程は各状態の時間を任意に設定できるため、キャッシュ・コヒーレンス制御を含む複雑なリクエストが発生しうる共有メモリ型並列計算機のモデル化に適する。セミマルコフ過程（以後 SMP と呼ぶ）に関する詳細は文献 [76, 55] に譲るとして、ここでは本論文で用いる SMP についての簡単な説明を行う。

本論文においての SMP は、K個の状態からなる離散的確率過程である。ここでパラメータ t_nはシステム・サイクルを単位とする離散値を取るものとする。状態 i においては時間分布関数 $F_{ij}(t)$ に従って滞在した後、状態 jに p_{ij}の確率で遷移する。ただし、分布関数 $F_{ij}(t)$ に対する、確率関数[1]を、$f_{ij}(t)$[2]とする。この時、状態 i での平均滞在時間η_iは、

$$\eta_i = \sum_{j=1}^{K} p_{ij} \sum_{n=1}^{\infty} t_n f_{ij}(t_n) \tag{3.1}$$

で表される。

例えば、現在プロセッサが計算状態にある場合、ある確率に従ってキャッシュ・リード・ヒット状態に遷移し、アーキテクチャ・モデルにより与えられる時間だけ滞在した後、再び計算状態に戻る。もし SMP がエルゴード的で既約なエンベデッド・マルコフ連鎖（以後 EMC と呼ぶ）を持つなら、状態 i の極限確率 P_iは次式で表される。

$$P_i = \frac{\pi_i \eta_i}{\sum_{j=1}^{K} \pi_j \eta_j} \tag{3.2}$$

ただし、$\{\pi_i\}$ は EMC の定常分布である。本論文におけるモデルは、後に述べるようにエルゴード的な成分が 1 つだけからなる EMC で表されているので、常に式 (3.2) が利用できる。

---

[1] Probability Mass Function

[2] 一般に、離散型確率変数 Xの確率関数は、$p_X(x_i) = P\{X = x_i\}$ で、分布関数は、$F_X(x) = \sum_{y \le x} p_X(y)$ で定義される。

3.2 Synapse プロトコル

Synapse プロトコル [7] は、Synapse N+1 並列計算機に採用されたメモリ・キャッシュ間転送／無効化型のキャッシュ・コヒーレンス・プロトコルである。第 4章で提案する解析モデルの対象となる並列計算機アーキテクチャでは、キャッシュ・コヒーレンス・プロトコルとして、Synapse プロトコルを想定するため、本節ではこのプロトコルの概説を行なう。

Synapse プロトコルのもとでは、任意のキャッシュ・ブロックは、図 3.1で示されるように、インバリッド（そのブロックは無効である）、バリッド（有効なブロックが 1 つ以上のキャッシュにある）、ダーティ（あるキャッシュに有効なブロックがあり、しかも既に書き込みが行なわれている）の三状態のうちの一状態にある。キャッシュ・ブロックがダーティ状態にある場合、そのブロックはリプレースされる時にメモリに書き戻される。

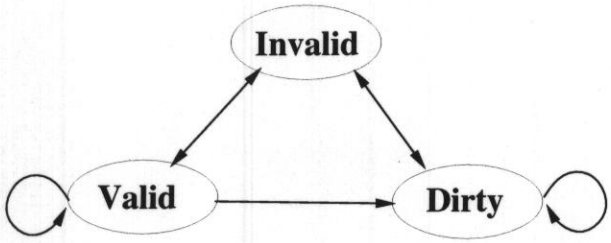


図 3.1 Synapse プロトコルの状態遷移

あるキャッシュがあるブロックに対しリード・ミスを起こし、他のキャッシュが当該ブロックのコピーをダーティ状態で持つ時、そのダーティ状態のブロックはメモリに書き戻され、インバリッドの状態に変更された後、リード・ミスを起こしたキャッシュが、書き戻されたブロックを、バリッド状態に変更してメモリから読み込む。同様にリード・ミスが発生し、当該ブロックがダーティ状態ではない場合は、リード・ミスを起こしたキャッシュが、当該ブロックをバリッド状態に変更してメモリから読み込む。

あるキャッシュがあるブロックに対しライト・ミスを起こし、他のキャッシュが当該ブロックのコピーをダーティ状態で持つ時、そのダーティ状態のブロックは

メモリに書き戻され、インバリッドの状態に変更された後、ライト・ミスを起こしたキャッシュが、書き戻されたブロックを、ダーティ状態に変更して書き込みを行なう。同様にライト・ミスが発生し、当該ブロックがダーティ状態ではない場合、インバリデーション命令が発行され、当該ブロックのコピーを持つキャッシュはそのブロックをフラッシュし、ライト・ミスを起こしたキャッシュが、当該ブロックを、ダーティ状態に変更して書き込みを行なう。

あるキャッシュがあるブロックに対しライト・ヒットを起こし、当該ブロックがダーティである場合は、即座に書き込みが行なわれる。同様にライト・ヒットが発生し、当該ブロックがバリッド状態の場合、ライト・ミスと同様である。

このように、Synapse プロトコルはインバリッド、バリッド、ダーティの三状態を持つが、本論文ではバリッド状態をクリーン状態と表現する。これは本研究の行なわれてきた経緯によるものである。

表 3.1は Synapse プロトコルの動作をまとめたものである。表中、C,D,I はそれぞれクリーン、ダーティ、インバリッド状態を示す。また、IV、MC、Ld、St はそれぞれ、無効化、メモリ・キャッシュ間転送、ロード、ストアを示す。

表 3.1 Synapse プロトコル概要

	cache	other caches	bus	memory	remark
Write Hit	D →D	not exist	no use	no use	バスは使わない
	C →D	C →I	IV,MC	Ld	メモリから読み込む
Read Miss	→C	not exist	MC	Ld	
	→C	D →I	IV,MC	Ld,St	無効化後読み込み
	→C	C →C	MC	Ld	
Write Miss	→D	not exist	MC	Ld	
	→D	C,D →I	IV,MC	Ld,St	無効化後読み込み
Replace	D →	no change	MC	St	メモリに書き戻し
	C →	no change	no use	no use	フラッシュ

3.3 Dragon プロトコル

Dragon プロトコル [7] は、ゼロックス社の Dragon 並列計算機に採用されたキャッシュ間転送／書き込み放送型のキャッシュ・コヒーレンス・プロトコルである。第5章では、SMCI モデルの書き込み放送型プロトコルへの適用について論じているが、その一例として Dragon プロトコルを想定するため、本節ではこのプロトコルの概説を行なう。

Dragon プロトコルのもとでは、任意のキャッシュ・ブロックは、図 3.2で示されるように、Clean-Exclusive（*CE*、そのブロックを一つのキャッシュだけが保持しており、モディファイしていない。）、Clean-Shared（*CS*、そのブロックを一つ以上のキャッシュが保持しており、モディファイしていない。）、Dirty-Exclusive（*DE*、そのブロックを一つのキャッシュだけが保持しており、モディファイされている。）、Dirty-Shared（*DS*、そのブロックを当該キャッシュがモディファイした状態で保持しているが、他のキャシュもそのブロックを *CS* 状態で持っているかもしれない。）の四状態のうちの一状態にある。キャッシュ・ブロックが Dirty-Exclusive 状態にある場合、そのブロックはリプレースされる時にメモリに書き戻される。

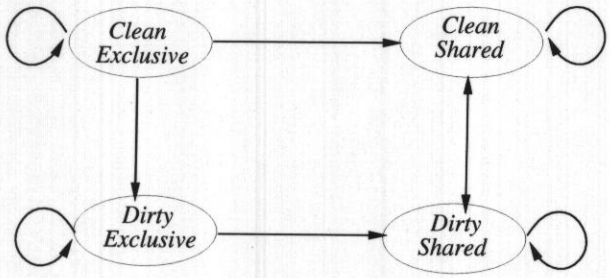


図 3.2 Dragon プロトコルの状態遷移

あるキャッシュがあるブロックに対しリード・ミスを起こし、他のキャッシュが当該ブロックのコピーを *DE*あるいは *DS*状態で持つ時、そのキャッシュは当該ブロックの内容を、キャッシュ・ミスを起こしたキャッシュに転送した後、当該ブロックを *DS*状態にする。リード・ミスを起こしたキャッシュでは、当該ブロック

は *CS*状態になる。リード・ミスの時に、他のキャッシュが当該ブロックのコピーを *CE*や *CS*で持つ場合は、キャッシュ間転送の後、*CS*状態になる。

あるキャッシュがあるブロックに対しライト・ミスを起こし、他のキャッシュが当該ブロックのコピーを *DE*あるいは *DS*状態で持つ時、そのキャッシュは当該ブロックの内容を、ライト・ミスを起こしたキャッシュに転送した後、当該ブロックを *CS*状態にする。ライト・ミスを起こしたキャッシュでは、当該ブロックをロード後書き込みを行い、当該ブロックを *DS*状態にして、書き込まれたワードを他のキャッシュにブロードキャストする。

表 3.2 Dragon プロトコル概要

	cache	other caches	bus	memory	remark
Write Hit	CE,DE →DE	not exist	no use	no use	バスは使用しない
	CS →DS	CS,DS →CS	CCW	no use	書き込み放送
	DS →DS	CS →CS	CCW	no use	書き込み放送
Read Miss	→CE	not exist	MCB	Load	メモリから読み込み
	→CS	CE,CS →CS	CCW	no use	キャッシュ間転送
	→CS	DE,DS →DS	CCW	no use	キャッシュ間転送
Write Miss	→DE	not exist	MCB	Load	メモリから読み込んだ後 キャッシュに書き込み
	→DS	CE,CS,→CS DE,DS	CCW	no use	キャッシュ間転送後 キャッシュに書き込み 書き込み放送
Replace	DE,DS →	no change	CMB	Store	メモリに書き戻し
	CE,CS →	no change	no use	no use	ブロックはフラッシュ

あるキャッシュがあるブロックに対しライト・ヒットを起こした時、そのブロックが *CE*や *DE*の場合には状態が *DE*に変わるだけで、バスもメモリもアクセスされない。そのブロックが *CS*や *DS*の場合、他のキャッシュでもそのブロックを保持しているかもしれないため、他のキャッシュでは *CS*に、ヒットしたキャッシュ

では書き込み後 *DS* にして書き込み放送を行なう。

表 3.2は Dragon プロトコルの動作をまとめたものである。表中、CCW、MCB はそれぞれキャッシュ間ワード転送、メモリ／キャッシュ間ブロック転送を意味する。

3.4 MultiLevel Inclusion Properties の保証

第6章における解析対象のアーキテクチャの特徴の一つとして、階層的なキャッシュが挙げられる。この時、上位のキャッシュと下位のキャッシュとの間で、データ・ブロックの無矛盾化が保証されなければならない。本節では階層的キャッシュ・システムにおける、データ・ブロックの無矛盾化を保証する、すなわち、包含属性を持たせるための技術に関する説明を行なう。

一般に、階層的なキャッシュは k レベルのキャッシュ[3]からなり、木構造を構成する。つまり、レベル $i+1$ のキャッシュによって保持されるデータ・ブロックは、レベル i のキャッシュによって共有され、逆にそれらレベル i のキャッシュの唯一の親がそのレベル $i+1$ のキャッシュである。このような多層レベルのキャッシュが包含属性を持つ（MultiLevel Inclusion Properties）とは、レベル $i+1$ のキャッシュが保持するデータ集合が、その直接の子供であるレベル i のキャッシュの保持するデータの集合の超集合である場合を言う。

第6章の並列計算機モデルは、6.2.1節で述べるように、キャッシュの階層度は 2（ローカルおよびグローバル・キャッシュ）であり、それぞれの階層のキャッシュはライトバック方式である。この階層的なキャッシュに MLI を保証させるには、以下の条件が必要となる [8]。ただし、対象となるブロックはグローバル・メモリに対するものだけである。

1) ローカル・キャッシュでミスが発生し、グローバル・キャッシュに当該ブロックが存在する場合、ローカル・キャッシュはグローバル・キャッシュからブロックを受けとる。

[3]ここでは、レベル 1 のキャッシュが一番プロセッサに近い位置にあるとする。

2) ローカル・キャッシュでライトバックが発生した場合、グローバル・キャッシュに対してライトバックがなされる。

3) グローバル・キャッシュでブロックのリプレースが発生した場合、その情報はローカル・キャッシュに伝えられ、ローカル・キャッシュも当該ブロックを保持している場合はそのブロックも同時にリプレースされる。

4) ローカル・キャッシュでブロックのリプレースが発生した場合、グローバル・キャッシュには影響を与えない。

5) ローカル・キャッシュでリード・ミスが発生し、同じグローバル・キャッシュを親に持つ別のローカル・キャッシュが当該ブロックをダーティ状態で保持している場合、そのローカル・キャッシュがグローバル・キャッシュに対して当該ブロックのライトバックを行なった後、通常のリード・ミスの処理がなされる。

6) ローカル・キャッシュでライト・ヒットが発生し、当該ブロックがクリーン状態であった場合、それに対応するグローバル・キャッシュでのブロックもダーティとなる。

7) ローカル・キャッシュでライト・ミスが発生し、グローバル・キャッシュでヒットした場合、必要に応じて他のローカル・キャッシュにインバリデーション命令やライトバック命令が発行された後、通常のライト・ミスの処理がなされる。

第 4 章

SMCI モデル

本章では、キャッシュ・メモリを有した共有メモリ型並列計算機に対する、セミマルコフ過程を利用した解析モデルである SMCI（Semi-markov Memory and Cache coherence Interference）モデルを提案する。本モデルは、評価対象の並列計算機の構成方式と、その上で稼働する並列プログラムに特有のパラメータを、入力パラメータとして受けとり、並列計算機を構成するプロセッサの取りうる各状態の極限確率を、安価な計算コストで算出する。プロセッサの取りうる状態には、キャッシュやメモリに対するデータ・リクエスト、キャッシュ・コヒーレンス制御リクエストや、それらに対するネットワーク競合やメモリ競合による待ち状態、計算状態等があり、当該アーキテクチャに対する詳細な評価尺度として十分なものである。これまでのマルコフ連鎖等を利用した並列計算機の解析モデルと異なり、本モデルにおける状態数は、プロセッサ台数やメモリ・サービス時間に依存せず、評価対象とする並列計算機の採用しているキャッシュ・コヒーレンス・プロトコルによってのみ決まる。従って、本モデルの計算時間は本質的にシステムの規模に依存しない。本モデルを用いて、プロセッサ利用率について実際の評価を行なった結果、本モデルは実際のシミュレーションで求めた結果に対する良好な近似を与えることが判明し、さらに、計算時間もシミュレーション時間に比較して大幅に短縮された。

4.1 はじめに

並列計算機はキャッシュ・コヒーレンス制御 [7]、クラスタ方式による大規模化 [30]、スケーラブルな共有メモリ方式 [34, 48] 等、さまざまな研究がなされ、より大規模に、より複雑に進化しつつある。このような大規模で複雑な並列計算機アーキテクチャの妥当性を確かめるのに、シミュレーションは有効な手法である反面、数万数十万規模のプロセッサの動きを検証するのに要する莫大なコストは、新たな問題になるであろう。

一方、並列計算機の性能予測を行なうのに、確率過程等を用いた解析モデルは低コストで有効な手法である。第2章で概説したように、確率過程、特にマルコフ連鎖を利用した並列計算機の解析モデルは、キャッシュを持たない小規模な共有メモリ型アーキテクチャに数多く適用され、その状態数を減少させるための近似手法に関する研究が精力的に行なわれてきた。

キャッシュを持たない並列計算機から、キャッシュを持ちキャッシュ・コヒーレンス制御をハードウェアでサポートする並列計算機への変革は、確率過程を利用した解析モデル構築という観点から見た場合、それまでのモデルを単純に拡張するという方針を実現不可能とした。

例えば、N個のプロセッサがクロスバー・ネットワーク等によって共有メモリにキャッシュを介さずに接続された場合、大雑把に考えると、各プロセッサの取りうる状態は、

1) 計算状態

2) メモリ・アクセス状態

3) メモリ競合による待ち状態

4) ネットワーク競合による待ち状態

の 4 種類であるため、正確なモデル化を行なうには全体で N^4個の状態数が必要となる。これは、マルコフ連鎖により解析することが事実上不可能な数であったため、Bhandarkar 以後の研究者は、様々な仮定や近似を与えることにより、これを計算可能な状態数にまで減らすことを試みた。

4.1. はじめに

キャッシュを持ちキャッシュ・コヒーレンス制御を行なう並列計算機アーキテクチャにおいて、プロセッサの状態を厳密に定義する場合、単にメモリ・アクセスに加えてキャッシュ・アクセスが増えるだけではなく、プロセッサの発行するリクエストの種類や、共有ブロックの状態[1]も、その定義の仕方に関係する。例えば、あるキャッシュに対してライト・リクエストが発行された時、リクエストされたキャッシュ・ブロックの状態に応じて、キャッシュ制御リクエストを引き起こす場合もあれば、通常のライト・リクエストに終る場合もある。このような厳密な状態定義は可能ではあるが、状態数が爆発的に増加するため、これを用いた実際の解析は不可能である。このため、Dubois らの研究のように、キャッシュ・ブロックの状態のみに着目した、マルコフ連鎖を利用した解析モデルはあるものの、システム全体の解析には MVA 法を利用したキューイング・ネットワーク・モデルが多く、マルコフ連鎖等の確率過程を利用した事例は報告されていない。すなわち、キャッシュを持つシステムの複雑さが、少ない状態数による確率過程の構築を困難としているのである。

そこで本章では、キャッシュを有する共有メモリ型並列計算機に対する、確率過程を利用した解析モデルとして、**SMCI**（Semi-markov Memory and Cache coherence Interference）モデルを提案する。SMCI モデルは、Mudge らの SMI（Semi-markov Memory Interference）モデルを拡張し、Dubois らの共有／プライベート・ブロックの考えを導入することにより、システムの規模に左右されない少ない状態数と、詳細な解析能力の両方を実現したものである。状態数はキャッシュ・コヒーレンス・プロトコルにより決まる。例えば、本章を通して用いられる Synapse プロトコルの場合は 20 である。また、システム全体の解析としては、並列計算機の各プロセッサの挙動を、通常のデータ・アクセスおよびキャッシュ・コヒーレンス命令と、それらの待ち時間をも考慮に入れた視点から予測する。この SMCI モデルの妥当性は、同じワークロードを用いたシミュレーション結果との比較によってなされる。

本章では、まず本解析モデル構築の基本方針を述べ、その対象となる並列計算機モデルを、アーキテクチャおよび並列プログラミング・モデルの観点から説明し、本解析モデルの満たすべき要件を与える。次に、本解析モデル構築のための

[1]キャッシュ・コヒーレンス・プロトコルに依存する。

仮定を示し、プロセッサ・リクエスト率の分類の諸定義を与える。これらの準備の後、モデル化のための状態定義と各状態間の遷移確率を与え、内部変数の導出を示し、具体的な計算手順を示した後、実際の性能評価実験を行なう。さらに、既存のシミュレーション結果 [7] との比較を行い、得られた解析結果を考察する。

4.2 並列計算機モデルと解析モデル

4.2.1 解析モデル化の基本方針

キャッシュのない並列計算機の解析モデルを設計する場合、あるプロセッサに着目して状態遷移を考え、それと等価なものが複数あると仮定して、全体のモデルを構築する場合がある。2.2節で概説した Mudge の解析モデルでは、各プロセッサは独立に稼働するが、その挙動は統計的に同じであるという仮定のもとで、大胆な簡略化を行なった。

ところが、図 4.1のように、キャッシュを持ちキャッシュ・コヒーレンス制御を行なう並列計算機に対しては、このようなモデル化の試みは難しい。なぜなら、システム全体の動作は、各プロセッサやネットワークの動作の影響を受けるからである。例えば、注目しているあるプロセッサが、あるサイクルにおいて、ライト・リクエストを発行したとすると、それに対応するシステム全体の挙動は、そのサイクルにおける、ライト要求されたデータ・ブロックとネットワークの状況に応じて異なる。要求されたデータ・ブロックが、

1) 他のどのキャッシュにも存在しなければ、通常のライト・ミスに、

2) 複数のキャッシュにモディファイされていない状態で存在すればキャッシュ・コヒーレンス制御命令を伴う、ライト・ヒットもしくはライト・ミスに、

3) 既に自分のキャッシュ中でモディファイされている場合は単なるライト・ヒットに、

4) 以上に加えてキャッシュ・ミスでかつ、ネットワークがビジーの場合、待ち状態に、

それぞれ遷移する。

ここで、キャッシュを持ちキャッシュ・コヒーレンス制御を行なう並列計算機を表す系の状態を考えてみよう。ただし、目的とする計算機はN個の（プライベートにキャッシュを持つ）プロセッサが単一バスによって共有メモリに接続され、E個のデータ・ブロックを共有するような並列プログラムが稼働しているものとする。各プロセッサの取りうる状態は、計算状態、各種データ・リクエスト状態に加えて、キャッシュ・コヒーレンス制御のためのさまざまなリクエスト状態、それらの待ち状態で表され、その状態数をAとする。各共有ブロックの状態は、使用しているキャッシュ・コヒーレンス・プロトコルによって定まるが、どのキャッシュにも保持されていない状態、1つ以上のキャッシュにモディファイされることなく保持されている状態、あるキャッシュに保持されており既にモディファイされた状態で表され、この状態数をBとする[2]。この時、システムの挙動全体を表す系の状態数は、$N^A E^B$となり、実際に平衡状態の方程式を計算して解くことは物理的に不可能となる場合が多い[3]。従って、本章の主眼である、極めて計算コストの低い解析モデルを構築するためには、大幅に簡略化されたモデルを考えなければならない。本章の主目的を達成するには、この簡略化は、例えば、状態数がプロセッサ数や共有ブロック数に依存するものでは、不十分であり、そのどちらにも依存しないものを考える必要がある。言い換えれば、プロセッサ数や共有ブロック数に状態数が依存しないような解析モデルを構築するのが、本章の主目的である。ところが、このような状態遷移を記述するには、各サイクルにおいて共有ブロックの状態（キャッシングされている／いない、モディファイされている／いない）や、他のプロセッサの状態（リクエスト実行中／リクエスト待ち）等を定式化しなければならず、システム全体のモデル化は極めて困難である。

そこで本章では、

1) 各プロセッサは同じ確率過程に従って独立して動作し、

2) 各共有ブロックがどのようにキャッシングされているかという状態は、同じ

[2]例えば本章における並列計算機モデルでは、$A = 20$, $B =$ プロセサ数 $+ 2$ となる。

[3]例えば、Eは与えられる並列プログラムによって決定されるが、数百から数千の値を取ることは珍しくない。また、Aの値も本論文での解析目的に十分な状態定義を行なうと、数十のオーダーになる。

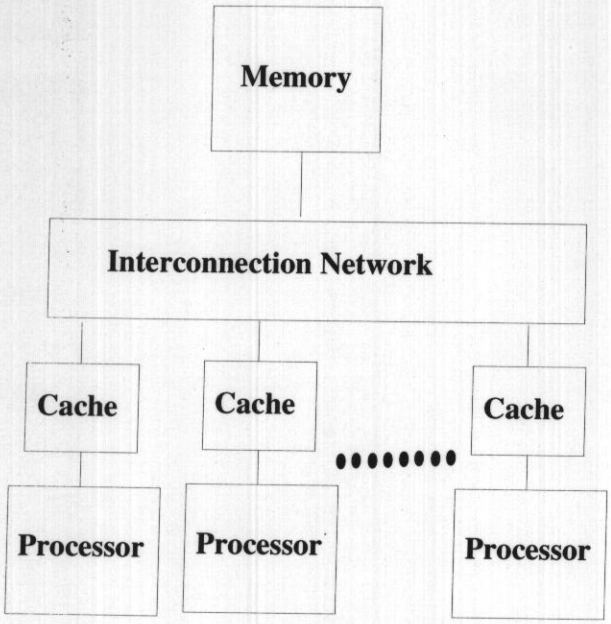


図 4.1 キャッシュを持つ並列計算機

確率で表される、

という仮定のもとで、キャッシュを持つ並列計算機に対する、簡略化された解析モデル構築を行なう。

まず、各共有ブロックの状態を確率的に表すために、Wang らのモデルにより導き出される、共有ブロックに対するキャッシュ・ミス率 [68] を利用する。次にこの確率を用いて、セミマルコフ過程を利用したプロセッサの定常状態と、その状態確率に基づいたネットワークのビジー率やリクエスト率（4.2.3節参照）[4]との関係を定式化する。その上で、適当な初期状態（ネットワークのビジー率とリクエスト率の初期値）からプロセッサの極限確率を計算し、それを基にネットワークのビジー率、リクエスト率を計算する。以下、この繰り返しを行ない、リクエスト率に関して収束点を求め、システム全体の定常状態を求める。得られたモデルの正当性は、Archibald のシミュレーション結果 [7] との比較によってなされる。

[4]ここで、ネットワークのビジー率やプロセッサのリクエスト率は、他のプロセッサの状態予測を意味する。

4.2.2 対象とする並列計算機モデル

本節では、SMCI モデルの対象となる並列計算機モデルについて、アーキテクチャおよびプログラミング・モデルの両側面から説明する。

4.2.2.1 アーキテクチャ・モデル

対象となる並列計算機モデルは、ハードウェア的に共有メモリをサポートしており、各プロセッサにはプライベートなキャッシュ・メモリが備わっているものとする。また、各プロセッサおよびキャッシュは、ネットワークを通してメイン・メモリと通信するものとする。なお、簡単のためプロセッサのサイクル・タイムとネットワークのサイクル・タイムは同じとする。

ネットワークのサイクル・タイムがプロセッサのサイクル・タイムの k 倍であるようなシステムの場合には、プロセッサの 1 サイクルあたりの、プロセッサからのリクエスト率を φ とすると、ネットワークの 1 サイクルあたり、ネットワークが受けとるリクエスト率は $\sum_{i=1}^{k}(1-\varphi)^{i-1}\varphi$ と考えることにより、本モデルに適用可能である。

プロセッサには、スーパースカラーやスーパーパイプライン、VLIW と言った高性能のものは想定しない。すなわち、単一のデコーダ、演算ユニット、ロード・ストア・ユニットを考える。プロセッサの状態は、計算を行なっている状態、メモリやキャッシュに対するリクエストを実行している状態、その状態への待ち状態の三種類とし、複数の状態間のオーバーラップ[5]はないものとする。

次に、キャッシュはライトバックによるフルアソシアティブ方式で、ブロックのリプレース・アルゴリズムは LRU とする。また、ライト・バッファ等の高速処理手法は用いないものとする。さらに、複数のキャッシュおよびメモリ間の無矛盾化を保証するために、キャッシュ・コヒーレンス制御を行なうものとする。この時のプロトコルとして、主記憶書き戻し型の **Synapse** [7] プロトコルを使う。多くのプロトコルの中から Synapse プロトコルを選んだ理由は、プロトコル自体の状態数の少なさだけである。従って、他プロトコルでも同様の SMCI モデルを構築可能である。なお、Synapse プロトコルについては、3.2節を参照されたい。

[5]例えば、リクエスト結果を待ちながら、計算を実行しているような状態。

ネットワークには単一の共有バスを用いる。本モデルは、単一共有バス以外のネットワークにも適用可能であるが、その詳細は第6章を参照されたい。

4.2.2.2 並列プログラム・モデル

本並列計算機モデルは、前節で述べたアーキテクチャ・モデルのもとで、特定の並列プログラム・モデルに従った並列プログラムが稼働している状態として定義される。その並列プログラム・モデルとは、**SPMD**（Single Program Multiple Data）モデルに、2.4節で述べた共有／プライベート・ブロックの概念を付加したものである。

各プロセッサは、等価なコード列を、異なるデータを使ってそれぞれ実行する。ここで、データは複数のプロセッサから読み書きされる共有ブロックと、それ以外のプライベート・ブロックに分けられる。全体のデータに対する共有ブロックの比率と、データ・アクセス全体に対するリード・アクセスの比率を指定することにより、実際のアプリケーションに対応した並列プログラム・モデルが構築できる。

ただし、本論文で考える並列プログラムとは、実際のアプリケーションのうち、完全に並列化がなされている部分を言い、逐次実行部分は考慮しない。

4.2.2.3 SMCI モデルの要件と仮定

本研究の主目的は、キャッシュを持つ並列計算機アーキテクチャの性能予測や、その上で稼働する並列プログラムの稼働予測を、プロセッサ数やデータ転送速度、キャッシュ制御の方式等のアーキテクチャ・パラメータと、データのリード・リクエスト率や共有ブロック参照率、共有ブロック数等の並列プログラム・パラメータとを入力として行なうものである。これらの具体的な予測対象は、プロセッサの各状態（計算状態、各種データ・リクエスト状態、キャッシュ・コヒーレンス制御リクエスト状態、リクエスト待ち状態）の割合であり、異なるパラメータ・チョイスでの各状態の相対的な比較を行なうことが主眼である。従って、特定のパラメータに対する絶対的な評価[6]を行なうことは主眼ではない。選ばれるパラ

[6]例えば、FLOPS 値等。

メータの組合せは、かなりの数に上るため、評価一回あたりの計算量は極めて小さくなければならない。以上、要件をまとめると以下のようになる。

（要件１）アーキテクチャ・モデルと並列プログラミング・モデルの詳細については、入力パラメータとして指定できる。

（要件２）実際の評価を行なうのに計算量が少ない。

（要件３）異なるパラメータ・チョイスに対する詳細かつ相対的な比較が可能。

上の要件を実現するために、本モデルに次のような仮定を導入する。

(仮定 1) 各プロセッサの挙動は同一の確率過程でモデル化され、各プロセッサは独立に動作する。

(仮定 2) 各共有ブロックの状態は、同じ確率で表される。

(仮定 3) 各プロセッサからの通常のデータ・リクエスト（キャッシュ・コヒーレンス制御のためのリクエストを除く）は、独立した分布に従う。

(仮定 4) プロセッサは与えられた時間パラメータによる幾何分布に従ってリクエストを発行する。また、リクエストの実行時間は、その種類ごと[7]にアーキテクチャ・モデルによって決まる一定分布に従う。

(仮定 5) 各待ち状態での滞在時間分布は、一括して一つの幾何分布で近似する。

(仮定 6) プログラムは安定稼働している状態、すなわち、起動時のページングやキャッシュ・ヒット率の悪さは考慮しない状態、を仮定する。またキャッシュは既に使い切った状態で、それに伴うキャッシュ・ブロックのリプレースについては考慮する。

仮定 (1),(2),(3) は本解析モデル構築に、仮定 (3),(4) は 4.4節で述べるシミュレーションとの比較のために、それぞれ必要なものである。仮定 (5) はモデル構築の簡略化のために採用した。また、仮定 (6) は前節で述べた極限確率の利用のために必要である。仮定 (1) は、本研究の出発点となるものである。各プロセッサが

[7]同一種類のリクエストに対して異なるアクセス時間が考えられる場合（例えば、同じ読み込み命令についてもバイト・リード、ワード・リード、ブロック・リード等が与えられるアーキテクチャの場合）、適当な分布関数を与える。

互いに影響し合う様子を厳密にモデル化すれば、その解析が事実上不可能な計算量になってしまうため、SPMD プログラム・モデルを前提に、各プロセッサは、『統計的に見て独立しており、かつ、同一の確率過程でモデル化可能』と考える。なお、(2) は一般には正しくないが、本モデルは SPMD プログラム・モデルを仮定しているため、(2) を仮定することができる。

4.2.3 各種リクエスト率の定義

並列計算機においては、プロセッサのリクエスト率は、システムの稼働時に決まるものである。例えば、同じプログラムを走らせたとしても、ネットワークが飽和している状態でのプロセッサ・リクエスト率は、そうでない状態でのリクエスト率とは異なってくる。文献 [71, 52] では、並列計算機の解析モデル中でのリクエスト率の修正方法について述べられている。これらの文献では、プロセッサ・リクエスト率は、実際のプログラムから決まるリクエスト率に加えて、プロセッサが待ち状態に陥った時に発行する再リクエストも、その割合に含めるとしている。

本章では、対象とするアーキテクチャが複雑なため、これまでのような簡単なプロセッサ・リクエスト率の概念では、その解析モデル化が困難である。そこで、本節では本モデル構築のために必要な、4 種類のリクエスト率の定義を行う。

- **ノーマル・リクエスト率**、φ_{nor}、とは、プロセッサが計算状態の時に引き続いて発行する、単位時間あたりのデータ・リクエストの割合である。

- **メモリ・リクエスト率**、φ_{mem}、とは、メモリ中のデータに対して発行される単位時間あたりのリクエストの割合である。(ノーマル・リクエスト率) × (キャッシュ・ミス率) + (その待ち状態からのリクエスト率) で定義される。

- **コヒーレンス・リクエスト率**、φ_{coh}、とは、キャッシュ・コヒーレンス制御のためのリクエスト率およびその待ち状態からのリクエスト率の和で定義される。ただし、本章では便宜上、通常のキャッシュ・ミスに伴うブロックのリプレースも、キャッシュ・コヒーレンス制御の一部と考える。

- **ネットワーク・リクエスト率**、φ_{net}、とは、ネットワークに対して発行されるリクエストの率である。メモリ・リクエスト率とコヒーレンス・リクエスト率の和で定義される。

以上の各種リクエスト率は、4.3節で構築するモデルの内部変数として使用される。本章においては、それぞれのリクエスト率は、一台のプロセッサが発行する、単位時間（サイクル・タイム）あたりの、リクエスト数である。

4.3 SMCI モデル

4.3.1 プロセッサの状態定義

SMCI モデル構築のために、プロセッサの状態定義を行う。この状態定義はキャッシュ・コヒーレンス・プロトコルに従ってなされる。本章では、Synapse プロトコルに沿ったものである。表 4.1は SMCI モデルにおけるプロセッサの状態定義を表す。また、表 4.2に同状態の集合を示す。これは本節における内部変数導出の際の表記を見やすくするための集合であり、モデル構築自体に必要なものではない。

4.3.2 状態遷移

図 4.2は定義された状態間の遷移を示す。それぞれの状態間の遷移は、ネットワークもしくはキャッシュに対して、データ・リクエスト、もしくはキャッシュ・コヒーレンス制御のリクエストがなされた場合、もしくはそれらのリクエストが完了した場合、他のプロセッサからのコヒーレンス制御命令を受けとった場合に起きる。図 4.2において、h はプライベート・ブロックに対するキャッシュのヒット率、Hは共有ブロックに対するキャッシュのヒット率、rはプロセッサからのデータ・リクエストに対するリード・リクエストの割合（従って、$1-r$はライト・リクエストを意味することになる。）、d はリクエスト要求のあるデータを含むブロックがダーティである確率、c はリクエスト要求のあるデータを含むブロックがダーティではない確率、w は（ネットワークに対して）リクエストを出した時に待ち状態に陥る確率（この時、4.2.3節で定義したネットワーク・リクエスト率

表 4.1 SMCI モデルの状態定義

COM	計算状態（**COM**putation）
Rh	キャッシュからの読み込み（**R**ead **h**it）
Wh	キャッシュへの書き込み（**W**rite **h**it）
HI	キャッシュ・ヒットにより発生するインバリデーション（**H**it & **I**nvalidate）
$\overline{HI}$	HIへの待ち状態
Rc	そのブロックの属性がダーティではないデータのキャッシュ・ミスによる読み込み（**R**ead from **c**lean block）
$\overline{Rc}$	Rcへの待ち状態
Rd	そのブロックの属性がダーティであるデータのキャッシュ・ミスによる読み込み（**R**ead from **d**irty block）
$\overline{Rd}$	Rdへの待ち状態
Wc	そのブロックの属性がダーティではないデータのキャッシュ・ミスによる書き込み（**W**rite to **c**lean block）
$\overline{Wc}$	Wcへの待ち状態
MI	キャッシュ・ミスにより発生するインバリデーション（**M**iss & **I**nvalidate）
$\overline{MI}$	MIへの待ち状態
Wd	そのブロックの属性がダーティであるデータのキャッシュ・ミスによる書き込み（**W**rite to **d**irty block）
$\overline{Wd}$	Wdへの待ち状態
WB	他のプロセッサから受けとったインバリデーション命令に対して発生するライトバック（**W**rite**B**ack）
$\overline{WB}$	WBへの待ち状態
RP	キャッシュ・ミスの結果生じるブロックのリプレースに付随するライトバック（**R**e**P**lace）
$\overline{RP}$	RPへの待ち状態
FL	ブロックのフラッシュ（**FL**ush）

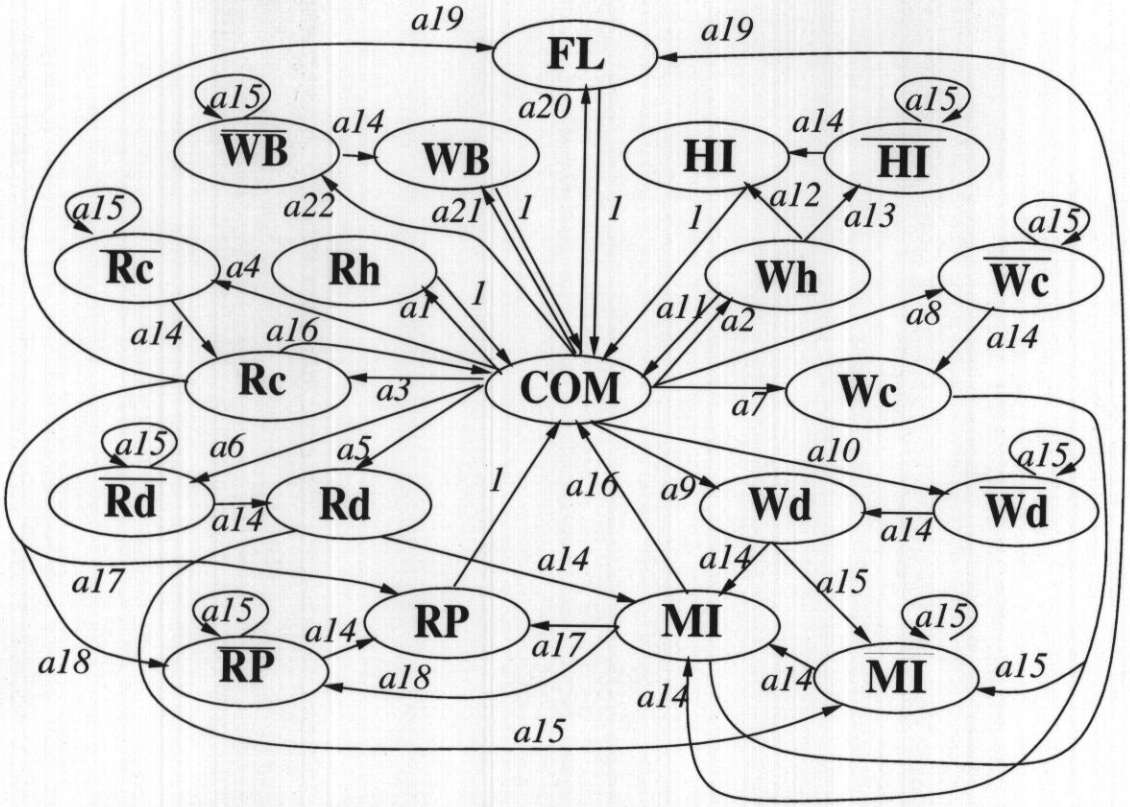
表 4.2 SMCI モデルの状態の集合

Q	SMCI モデル全状態の集合
Q_{mem}	$= \{Rc, Rd, Wc, Wd\}$
Q_{mem_w}	$= \{\bar{i} \mid i \in Q_{mem}\}$
Q_{coh}	$= \{HI, MI, WB, RP\}$
Q_{coh_w}	$= \{\bar{i} \mid i \in Q_{coh}\}$
Q_{net}	$= Q_{mem} \cup Q_{coh}$
Q_{net_w}	$= \{\bar{i} \mid i \in Q_{net}\}$

φ_{net}が必要となる。)、x はキャッシュが既に一杯である確率、u はデータ・リクエストが共有ブロックへのアクセスである割合（従って、$1-u$ がプライベート・ブロックへのアクセス率となる。)、m はリプレース対象のブロックが既にモディファイされている確率、yは COM状態からの全ての遷移に対する、同状態からのデータ・リクエストに起因する遷移の割合を示す。

後で述べるように、H, c, d は Wang らのモデル [68] により導き出され、x,w,y は、プロセッサの状態の極限確率から導き出され、その他は入力パラメータとして与えられる。

状態COMから遷移するのは、プロセッサがデータ・リクエストを発行した場合と、インバリデーション・リクエストを受信した場合のみである。後者の場合、プロセッサの状態は無効化の対象となるブロックがモディファイされているかどうか、ネットワークは利用可能かどうかによって FL, WB（WBと FL からはCOMのみに遷移する。）もしくは$\overline{WB}$に遷移する。前者の場合、そのデータ・リクエストがキャッシュ・ヒットするかしないか、キャッシュ・ミスした場合にそのブロックがダーティか否か、ネットワークが空いているかどうか、リクエストはリードかライトか、によってそれぞれ、Rh（この状態からはCOMにのみ遷移する。)、Rd, Rc, $\overline{Rd}$, $\overline{Rc}$, Wh, Wd, Wc, $\overline{Wd}$, $\overline{Wc}$に遷移する。Wh からはライト・ヒットしたブロックがクリーンである場合はインバリデーション命令を発行しなければならないので、ネットワークの状態によって HI（この状態からはCOMにのみ遷移する。）もしくは$\overline{HI}$に、クリーンでない場合は COM に遷移



a1=y(h(1−u)+H u)r , a2=y(h(1−u)+H u)(1−r) , a3=y((1−h)(1−u)+(1−H)u(1−d))r(1−w) ,
a4=y((1−h)(1−u)+(1−H)u(1−d))rw , a5=y(1−H)urd(1−w) , a6=y(1−H)urdw ,
a7=y((1−h)(1−u)+(1−H)u(1−d))(1−r)(1−w) , a8= y((1−h)(1−u)+(1−H)u(1−d))(1−r)w ,
a9=y(1−H)u(1−r)d(1−w) , a10=y(1−H)u(1−r)dw , a11=1−c , a12=c(1−w) , a13=cw ,
a14=1−w , a15=w , a16=1−x , a17=mx(1−w) , a18=mxw , a19=x(1−m), a20=(1−y)(1−m),
a21=(1−y)m(1−w), a22=(1−y)mw

図 4.2 Synapse プロトコルに従ったプロセッサの状態遷移

する。状態 Rc からはキャッシュに空きがあれば COM に遷移するが、空きがなくてリプレースされるブロックがダーティでなければ FL に、ダーティーであれば当該ブロックのライトバックによるリプレースを行なうためにネットワークの状態によって、RP（この状態からは COM にのみ遷移する。）もしくは $\overline{RP}$ に遷移する。状態 Rd, Wc, Wd からはインバリデーション命令を発行しなければならないので、ネットワークの状態によって MI もしくは $\overline{MI}$ に遷移する。状態 MI からは Rc 同様、COM, FL, RP, $\overline{RP}$ のいずれかに遷移する。待ち状態 $\overline{HI}$, $\overline{Rc}$, $\overline{Rd}$, $\overline{WB}$, $\overline{Wc}$, $\overline{MI}$, $\overline{Wd}$, $\overline{RP}$ はリクエストが受理されるとそれぞれ、HI, Rc, Rd, WB, Wc, MI, Wd, RP に遷移する。

図 4.2の遷移確率において、例えば $a1 = y(h(1-u)+Hu)r$ は状態 COM が Rh に遷移する確率を表す。なぜなら、状態 COM において状態遷移が発生すると、そのリクエストは y の割合でデータ・リクエストであり、$1-y$ の割合でキャッシュ・コヒーレンス制御のためのライトバックやフラッシュである。もしそれがデータ・リクエストの場合、r の割合でリード、$1-r$ の割合でライトである。それがリード・リクエストであるなら、そのリクエストは、キャッシュに向かって発行され、確率 $1-u$ でプライベート・ブロックのデータをリードすることになる。この時、プライベート・ブロックに対するキャッシュ・ヒット率は h である。また、発行されたリクエストは割合 u で共有ブロックのデータをリードすることになる。この時の共有ブロック・キャッシュ・ヒット率は H である。よって、$a1 = y(h(1-u)+Hu)r$ が示される。また、$a5 = y(1-H)urd(1-w)$ は状態 COM が Rd に遷移する確率を表す。なぜなら、リード・リクエストがキャッシュ・ミスを起こす時、もしリクエスト先のデータがプライベート・ブロック中に存在すれば、定義よりそのブロックはダーティになりえないからである。他の遷移確率も同様に導かれる。

4.3.3 滞在時間分布

先に述べた状態遷移はプロセッサがリクエストを発行したり、リクエストが完了した時に起きる。本節では各遷移間の滞在時間について説明する。

まず、状態 COM からは、厳密には次に遷移する状態（Rh, Rc, $\overline{Rc}$, Rd, $\overline{Rd}$, Wh, Wc, $\overline{Wc}$, Wd, $\overline{Wd}$, WB, $\overline{WB}$, FL）に従ってそれぞれ滞在時間分布関

数を与えるべきであるが、そのような関数を導き出すのは困難[8]であるため、遷移先の状態に関わらず、データ・リクエストに起因する遷移の場合はパラメータλ_{nor}の、キャッシュ制御に起因する遷移の場合はパラメータλ_{coh}の幾何分布に従うものとする。次に、キャッシュやメモリのアクセスを行なう状態 i （$i \in \{Rh, Rc, Rd, Wh, HI, Wc, MI, Wd, RP, WB\}$）から次の状態 j への遷移は、下記の一定分布に従う。

$$F_{ij}(t) = \begin{cases} 1 & \text{if } t \geq \text{状態 i でのアクセス・タイム} \\ 0 & \text{otherwise} \end{cases}$$

それぞれのアクセス・タイムは対象とする並列計算機アーキテクチャの仕様によって与えられる。待ち状態（$\overline{Rc}, \overline{Rd}, \overline{Wc}, \overline{Wd}, \overline{HI}, \overline{MI}, \overline{RP}, \overline{WB}$）から次の状態への遷移は平均 Wt の幾何分布に従うとする。ただし、後で述べるように Wt は初期値 1 から出発し、プロセッサの状態の極限確率と、それをもとに計算されるネットワークのビジー率から逐次的に計算される。

4.3.4 セミマルコフ過程の構築

先に述べた状態空間と遷移確率により定義される確率過程 $X = \{X_n\}_{n=0}^{\infty}$は、プロセッサの状態を表現する。4.2.2節で示した本並列計算機モデルの定義と、4.2.2.3節で与えた仮定から明らかなように、この確率過程はマルコフ連鎖である。

また、Xの各状態の滞在時間は、前節で与えた時間分布関数に従うので、その各滞在時間を $T = \{T_n\}_{n=0}^{\infty}$で与えられる確率変数列と考えると、確率過程 $(X, T) = \{X_n, T_n\}_{n=0}^{\infty}$は SMP（Semi-Markov Process）を形成する。

4.2.1節で述べたように、本 SMCI モデルは、適当な初期値から出発して、この (X, T) からプロセッサの状態の極限確率を求め、得られた結果からリクエスト率やネットワークのビジー率を修正し、以後繰り返し処理を行なうことにより、リクエスト率の収束点を求めて、システム全体の定常状態を求める。

(X, T) の EMC（Embedded Markov Chain）である X は、図4.2からも明らかなように、エルゴード的であるので、その定常分布を求めることができる。よっ

[8]例えば、与えられた並列プログラムから、状態 COMから状態 Rh に遷移するまでの滞在時間分布等を求めるのは極めて困難である。

て、EMC の定常分布 $\{\pi_i\}$ と、各状態の平均滞在時間 $\{\eta_i\}$ を式 (3.2) に適用して、(X,T) の極限確率を求めることができる。

ここでは、式 (4.1) で、EMC の定常分布のみを示しておく。ただし、gは正規化係数である。マルコフ連鎖の遷移確率から定常分布を求める手法については、文献 [76, 55] 等を参照されたい。

$$
\begin{pmatrix}
\pi_{COM} \\ \pi_{Rh} \\ \pi_{Wh} \\ \pi_{HI} \\ \pi_{\overline{HI}} \\ \pi_{Rc} \\ \pi_{\overline{Rc}} \\ \pi_{Rd} \\ \pi_{\overline{Rd}} \\ \pi_{Wc} \\ \pi_{\overline{Wc}} \\ \pi_{Wd} \\ \pi_{\overline{Wd}} \\ \pi_{MI} \\ \pi_{\overline{MI}} \\ \pi_{RP} \\ \pi_{\overline{RP}} \\ \pi_{WB} \\ \pi_{\overline{WB}} \\ \pi_{FL}
\end{pmatrix}
= g \times
\begin{pmatrix}
1-w \\
yr(h(1-u)+Hu)(1-w) \\
y(1-r)(h(1-u)+Hu)(1-w) \\
yc(1-r)(h(1-u)+Hu)(1-w) \\
yc(1-r)(h(1-u)+Hu)w \\
yr(\kappa_1+\kappa_2(1-d))(1-w) \\
yr(\kappa_1+\kappa_2(1-d))w \\
ydr(1-H)u(1-w) \\
ydr(1-H)uw \\
y(1-r)(\kappa_1+\kappa_2(1-d))(1-w) \\
y(1-r)(\kappa_1+\kappa_2(1-d))w \\
yd(1-r)(1-H)u(1-w) \\
yd(1-r)(1-h)uw \\
y((1-r)(\kappa_1+\kappa_2)+rd(1-H)mux)(1-w) \\
y((1-r)(\kappa_1+\kappa_2)+rd(1-H)mux)w \\
y((\kappa_1+\kappa_2)-d(1-H)ru(1-mx))mx(1-w) \\
y((\kappa_1+\kappa_2)-d(1-H)ru(1-mx))mxw \\
(1-y)m(1-w) \\
(1-y)mw \\
(1-y+(\kappa_1+\kappa_2(1+dmrx))xy)(1-m)(1-w)
\end{pmatrix}
\tag{4.1}
$$

ただし、$\kappa_1=(1-h)(1-u)$、$\kappa_2=u(1-H)$。

4.3.5 入力パラメータと共有ブロックの状態予測

SMCI モデルを計算するには、対象とするハードウェア・アーキテクチャ・モデルと並列プログラム・モデルを表した入力パラメータが必要である。この入力パラメータには、N（プロセッサ数）、h, u, r, E（共有ブロックの数）、m, η_i

($i \in Q_{net} \cup \{Rh, Wh\}$)、λ_{nor}がある。これらのパラメータの多くは、Archibald のシミュレーション [7] に対するパラメータ（システム・モデル）と同じである。なお、SMCI モデル構築に必要な入力パラメータとの一覧を表 4.3に示しておく。

表 4.3 SMCI モデルで使用される入力パラメータ

N	プロセッサ数
h	プライベート・ブロックに対するキャッシュ・ヒット率
u	共有ブロックへのアクセス率
r	リード・リクエスト率
E	プログラムで使われる共有ブロックの数
m	リプレース対象のブロックが既にモディファイされている確率
η_i	キャッシュやメイン・メモリのアクセス時間 ($i \in Q_{net} \cup \{Rh, Wh\}$)
λ_{nor}	COM状態からデータ・リクエストに起因する遷移を行なう際の滞在時間分布パラメータ

また、H, d, c は 4.2.1節で述べたように、Wang らのモデル [68] から導き出される。共有ブロックに対するキャッシュ・ヒット率 H と、キャッシュに保持されている共有ブロック数Ψ に関しては付録 A.1を参照されたい。

確率 c は、ライト・ヒット時にインバリデーションを引き起こすかどうかを決定する。当該ブロックが共有ブロックの場合は、そのブロックが非ダーティ状態である時に、当該ブロックがプライベート・ブロックの場合は、そのブロックがまだモディファイされていない時に、インバリデーションが発生する。よって確率 c は以下で近似される。

$$c = (1-u)u_{md} + ur\left(1 - \left(1 - \frac{\Psi r}{E}\right)^{N-1}\right) \tag{4.2}$$

上式の詳細な説明を行なう。まず、ライト・ヒットが起きた時、ヒットしたブロックがプライベート・ブロックなら、上式第 1 項の確率でインバリデーションが発生する。次に、共有ブロックの場合、当該ブロックが、自分のキャッシュ中

でモディファイされていない場合にのみ、インバリデーションは発生する。すなわち、最初にリード・ミスでメモリから読み込んできて、当該ライト・ヒットまでの間、一度もライトされていない場合にインバリデーションが発生する[9]と考える。もし、一度でもライトされていると、インバリデーションがその時点で発生して、他のキャッシュには当該ブロックのコピーは存在しない。従って、上式第2項は、ライト・ヒットした時に、そのブロックは共有ブロックで（u）、最初にリード・ミスによってキャッシングされており（rで近似）、他のキャッシュに当該ブロックの（クリーン状態の）コピーが存在する（$1-(1-(\Psi r)/E)^{N-1}$）という導出になる。

u_{md}はプライベート・ブロックのライト・ヒット時に当該ブロックがまだモディファイされていない確率を示すが、これは他の入力パラメータ m, r, h を使って $u_{md} = 1-(1-h)(m+r-1)/((1-r)h)$ で与えられる。式の導出は文献 [7] による。

あるキャッシュでキャシュ・ミスが起こったとき、当該共有ブロックがダーティ状態である確率 d は、そのブロックが自分以外の $N-1$ 個のキャッシュのうちの一箇所で、ライト・リクエストによってキャッシングされており、それ以外の $N-2$ 個のキャッシュではキャッシングされていない確率で近似する。

$$d = \begin{pmatrix} N-1 \\ 1 \end{pmatrix} \left(\frac{\Psi(1-r)}{E}\right) \left(1 - \frac{\Psi(1-r)}{E}\right)^{N-2} \tag{4.3}$$

4.3.6 リクエスト率とネットワークのビジー率の導出

(X,T) を解析するには、4.3.5節で与えられたパラメータ以外に、x, y, w, Wt の値が必要である。しかしながら、これらの内部変数を求めるには、各種リクエスト率とネットワークのビジー率が必要となる。これらは、(X,T) の極限確率から導出されるので、本節ではこの導出過程について説明する。なお、SMCI モデル構築に必要な内部変数の一覧を表4.4に示しておく。

確率 x は、あるプロセッサからのリクエストがキャッシュ・ミスを起こし、データ・ブロックがキャッシュ・インした時に、キャッシュに空きがあるかどうかを決

[9]最初にライト・ミスでメモリから読み込んできた場合は、既にモディファイされた状態なのでインバリデーションは発生しない。

表 4.4 SMCI モデルで使用される内部変数

H	共有ブロックに対するキャッシュ・ヒット率
Ψ	あるキャッシュに保持されている共有ブロックの数
d	リクエスト要求のあるデータ・ブロックがダーティである確率
c	リクエスト要求のあるデータ・ブロックがダーティでない確率
x	キャッシュが既に一杯である確率
inv_issue	あるプロセッサが任意のプロセッサに対してインバリデーション命令を発行する率
inv_arrive	あるプロセッサがインバリデーション命令を受信する確率
u_{md}	プライベート・ブロックのライト・ヒット時それがまだモディファイされていない確率
w	リクエストを出した時に待ち状態に陥る確率
y	状態 COM からの遷移に対して、それがデータ・リクエストに起因する遷移である割合
Wt	待ち状態の平均時間
λ_{coh}	COM 状態からキャッシュ制御リクエストに起因する遷移を行なう際の滞在時間分布パラメータ
φ_{nor}	ノーマル・リクエスト率
φ_{mem}	メモリ・リクエスト率
φ_{coh}	コヒーレンス・リクエスト率
φ_{net}	ネットワーク・リクエスト率
$\{\pi_i\}$	EMC の定常分布
$\{P_i\}$	SMP の極限確率

定する。ここで、キャッシュに空きができるのは、直前のキャッシュ・ミスから、今回のキャッシュ・ミスが発生するまでの期間、少なくとも一度はインバリデーション命令を受けた場合と考える。キャッシュ・ミスは、各サイクルあたり

$$cache\_miss = ((1-u)(1-h) + u(1-H))\varphi_{nor} \tag{4.4}$$

の確率で発生すると考えて、この逆数を二回のキャッシュ・ミスの平均間隔時間と見なす。

各プロセッサは、各サイクルあたりφ_{nor}の率でリクエストを発行しており、状態Whからcの確率、状態Rd, Wc, Wdからは確率1でインバリデーションを発行するので、あるプロセッサがインバリデーションを発行する率inv_issueは、

$$\begin{aligned} inv\_issue &= ((1-r)(c(h(1-u)+Hu) \\ &+ (1-h)(1-u)+(1-H)u(1-d)) \\ &+ u(1-H)d)\varphi_{nor} \end{aligned} \tag{4.5}$$

となる。よって、あるプロセッサがあるサイクルにおいて、自分以外の$N-1$のプロセッサの発行するインバリデーションを少なくとも一つ受けとる確率inv_arriveは、

$$inv\_arrive = 1 - \left(1 - \frac{\alpha(E,N)}{N-1} inv\_issue\right)^{N-1} \tag{4.6}$$

となる。ただし、$\alpha(E,N)$は共有ブロックの数とプロセッサ数を引数として、0から$N-1$までの値をとる関数で、一回のインバリデーション命令あたりの、無効化される平均ブロック数を表す。4.4.3節で考察するように、これは並列プログラム・モデルに強く依存したものである。

よって、キャッシュ・ミスが発生した時に、キャッシュに空きがない確率xは、以下のように表される。

$$x = (1 - inv\_arrive)^{\frac{1}{cache\_miss}} \tag{4.7}$$

ところで、状態COMからインバリデーション命令を受け取って状態遷移する場合の滞在時間分布パラメータλ_{coh}は、上で求めた確率inv_arriveをインバリデーション受信率と考えると、その逆数で定義される。

$$\lambda_{coh} = \frac{1}{inv\_arrive} \tag{4.8}$$

ネットワークに対してリクエストを出した時、待ち状態に陥る確率wとは、そもそもネットワークが空いてなかった場合と、ネットワークは空いているが他のプロセッサからのリクエストとの競合に負けた場合とに分けて考えられる。まず、ネットワークが空いていない確率は次のようにして求める。まず、あるプロセッサから見てネットワーク・ビジーである確率$Busy$は、

$$Busy = \begin{pmatrix} N-1 \\ 1 \end{pmatrix} \left(\sum_{i \in Q_{net}} P_i \frac{\eta_i - 1}{\eta_i} \right) \left(1 - \sum_{i \in Q_{net}} P_i \frac{\eta_i - 1}{\eta_i} \right)^{N-2} \tag{4.9}$$

となる。なぜなら、SMP によってプロセッサの極限確率が与えられている時、各プロセッサは極限確率$P_i (i \in Q_{net})$でネットワークを握っており、それぞれの状態において、当該状態に遷移してから$\eta_i - 1$サイクルの間は状態を変えない、すなわちネットワークを離さない。また、各サイクルにおいてネットワークを握ることのできるプロセッサは一つしかないからである。

次に、他のプロセッサからのリクエストとの競合に勝つ確率Winは

$$Win = \left(1 - (1 - \varphi_{net})^N\right) \frac{1}{N\varphi_{net}} \tag{4.10}$$

で求まる。なぜなら、リクエストが来ているサイクルにおいて、競合に勝つ確率はその時のリクエスト数の逆数であり、あるサイクルにおいて、リクエストが来ている確率は$1 - (1 - \varphi_{net})^N$であるからである。ここで、式(4.10)と式(4.9)のみがアーキテクチャに依存することに注意されたい。本章では4.2.2節で述べたように、ネットワークとして単一共有バスを用いているが、これは4.4節でのシミュレーション結果との比較のためである。

以上のことより、待ち状態に陥る確率wは次式で表される。

$$w = Busy + (1 - Busy)(1 - Win) \tag{4.11}$$

サイクル毎の各種リクエスト率φ_iは、次のようにして求める。まず、ノーマル・リクエスト率は定義より、

$$\varphi_{nor} = \frac{1}{\lambda_{nor}} \tag{4.12}$$

と表される。

メモリ・リクエスト率は、単位時間あたりの、COM状態からキャッシュ・ミスのリクエストを出す回数と、メモリ・アクセスの待ち状態にあるときのリクエスト率の和に等しい。待ち状態は、前述したようにネットワーク競合のための待ち状態と、ネットワーク・ビジーのための待ち状態とに分けられる。ネットワーク・ビジーのための待ち状態の場合、待ち状態の平均滞在時間 Wt に一回だけ、ネットワーク競合による待ち状態の場合サイクル毎に、それぞれリクエストを出すと考える。この時、メモリ・リクエスト率φ_{mem}は次式で求まる。

$$\begin{aligned}\varphi_{mem} &= ((1-u)(1-h) + u(1-H) \\ &+ \ c(1-r)(h(1-u) + Hu))\varphi_{nor} \\ &+ \ Busy \sum_{i \in Q_{mem\_w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{mem\_w}} P_i\end{aligned} \tag{4.13}$$

コヒーレンス・リクエスト率φ_{coh}は、インバリデーション発行率 inv_issue と、インバリデーション・リクエストを受け取った時に当該ブロックが既にモディファイされている場合に発行するライトバック命令の率 ($m\ inv\_arrive$)、キャッシュ・ミス時にキャッシュ・エントリーに空きがなく、リプレース対象のブロックがモディファイされているときに発行するライトバック命令の率、の三つの率をもとにして導出する。

$$\begin{aligned}\varphi_{coh} &= inv\_issue + inv\_arrive \times m + cache\_miss \times x \times m \\ &+ \ Busy \sum_{i \in Q_{coh\_w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{coh\_w}} P_i\end{aligned} \tag{4.14}$$

定義により、ネットワーク・リクエスト率φ_{net}は、次のようになる。

$$\varphi_{net} = \varphi_{mem} + \varphi_{coh} \tag{4.15}$$

Wt は待ち状態の平均滞在時間であるから

$$Wt = \frac{\sum_{i \in Q_{net\_w}} P_i}{\sum_{i \in Q_{net\_w}} \frac{P_i}{\eta_i}} \tag{4.16}$$

で求められる。

状態 COM から発するリクエストに対して、それがデータ・リクエストである割合 y は、

$$y = \frac{\frac{1}{\lambda_{nor}}}{1 - \left(1 - \frac{1}{\lambda_{coh}}\right)\left(1 - \frac{1}{\lambda_{nor}}\right)} \tag{4.17}$$

である。

最後に、状態 COM での平均滞在時間は、式 (3.1) を適用して、

$$\eta_{COM} = y\lambda_{nor} + (1-y)\lambda_{coh} \tag{4.18}$$

で与えられる。

4.3.7 SMCI モデルの計算手順

SMCI モデルの実際の計算手順は、4.3.4節で述べたように、入力パラメータと適当な初期値から (X, T) の極限確率を求め、その結果を元にリクエスト率とネットワーク・ビジー率等の修正をし、以後収束点に至るまで繰り返しを行なう。適当な初期値としては、

$$\begin{aligned} \varphi_{coh} &= 0 \\ y &= 1.0 \\ x &= 1.0 \\ w &= 1 - (1 - \varphi_{nor})^N \\ Wt &= 1 \end{aligned}$$

で、φ_{net} が収束するまで計算をおこなう。

4.4 評価と考察

4.4.1 シミュレーションとの比較

SMCI モデルの正当性を証明するため、Archibald のシミュレーション [7] 結果との比較実験を行なった。対象とするアーキテクチャ・モデルは、彼のシミュレーションと同じものを採用した。また、入力パラメータも同じものを用いた。キャッシュやメモリをアクセスする状態の平均滞在時間 $\{\eta_i \mid i \in Q_{net}\}$ は、アーキテクチャ・モデルに従い、入力パラメータとして与えられる。本評価実験では、表 4.5 で示される値を使用した。この値を変更することにより、異なるスペックのアーキテクチャを、容易に解析することができる。また、状態 COM の滞在時間分布を決めるパラメータ λ_{nor} は、文献 [7] に従って、3 とした。

表 4.5 キャッシュやメモリ・アクセスを行なう状態の平均滞在時間

Rh	Wh	HI	Rc	Rd
1	1	4	16	16
Wc	MI	Wd	WB	RP
16	4	16	16	16

図 4.3、図 4.4、図 4.5は、それぞれ共有ブロック数 E が 1024, 128, 16 である時の、プロセッサ（キャッシュ）台数に対するシステム・パワー [7] を表している。与えたパラメータは、共有ブロック・アクセス率 u は 0.001 と 0.05、プライベート・ブロックに対するキャッシュ・ヒット率 h は 0.95 と 0.98、リード・リクエスト率 r は 0.85 と 0.7、ブロックのリプレース時のモディファイ率 m は 0.3 と 0.4 である。システム・パワーは、各プロセッサ利用率の総和に 100 を掛けたものである。ただし、インバリデーション発生時に、無効化される平均ブロック数を決める関数 $\alpha(E, N)$ は、以下のように近似した。

$$\alpha(E, N) = 2N\sqrt{\log_2 E} \tag{4.19}$$

また、共有ブロックに対するキャッシュ・ヒット率に関係するアクセス・バー

スト長、$ls()$ を、共有ブロック数とプライベート・ブロックに対するキャッシュ・ヒット率を引数として、次のような近似関数で表した。

$$ls(E,h) = \sqrt{\log_2 E}(\frac{a}{h} - b) \tag{4.20}$$

ただし、a と b は定数で、本実験では $a = 279.3$, $b = 284$ を採用した。

関数$\alpha()$ および $ls()$ に関しては、その近似性妥当性を議論しなければならないが、本論文ではこれらの議論は行なわない。

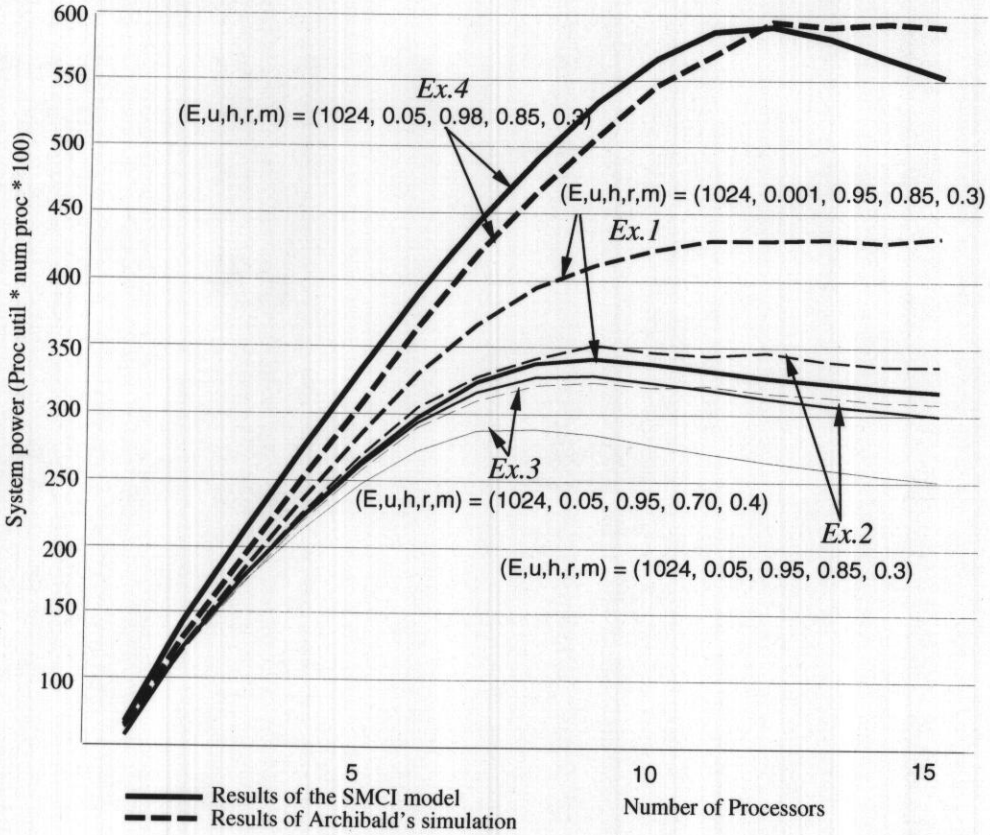


図 4.3 共有ブロック数１０２４での比較

この実験において、SMCI モデルは文献 [7] のシミュレーションと極めて良く似た結果を出している。シミュレーションとの誤差自体は、平均 8.0%とあまり良好な精度とは言えないが、各入力パラメータに対するシステム・パワーの相対的な比較が正しくなされている。本比較実験では、評価対象はシステム・パワー

（状態 COM）のみとしたが、これ以外にも表4.1で示されるすべての状態での同様の比較が可能である。すなわち、異なるパラメータ・チョイスに対する詳細かつ相対的な比較が可能であり、4.2.2.3節の（要件3）を満たしている。また、本実験は与えられたアーキテクチャ・モデルに対して、E, u, h, r, m で規定される並列プログラムを実行させた時の稼働予測である。表4.5を指定することにより、さまざまなアーキテクチャ・スペックの性能予測を行なうことができる。従って、（要件1）を満たしている。

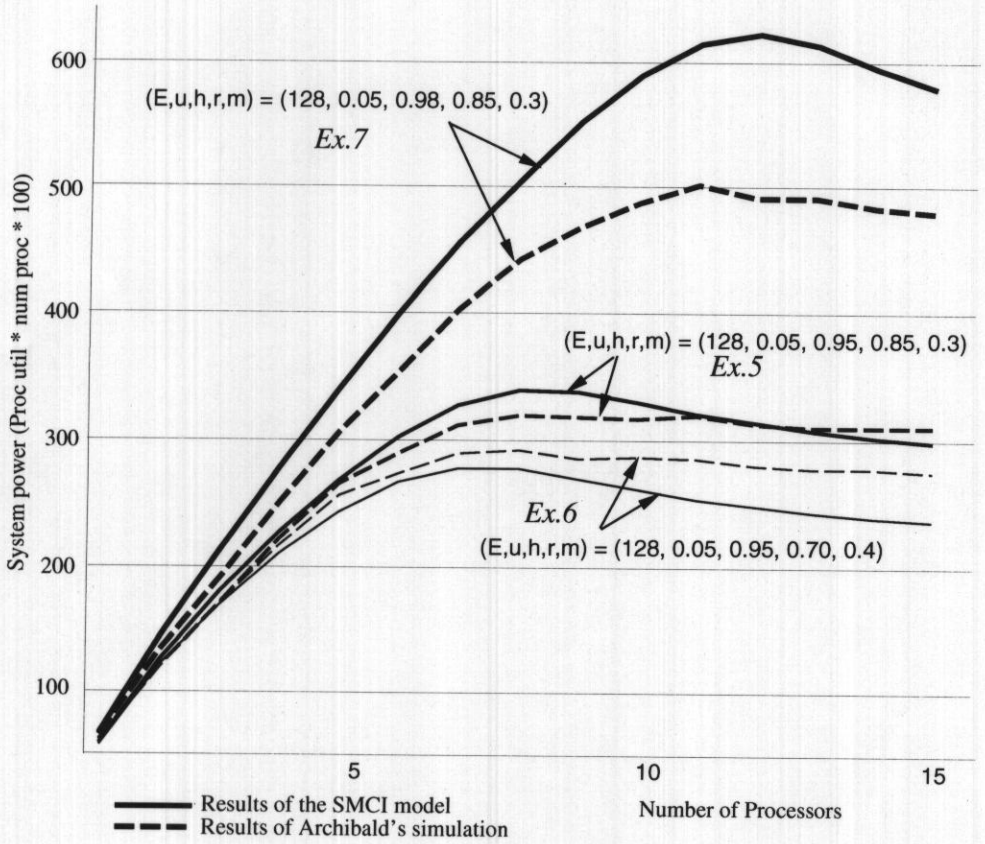


図 4.4 共有ブロック数１２８での比較

次に各実験結果から観測される点について説明する。共有ブロック数が16, 128, 1024、に対する平均誤差はそれぞれ、6.0%, 7.0%, 10.0% であったことから、SMCIモデルは共有ブロック数が少ない時、すなわち、共有ブロックが実際に共有される確率が高い時の方が、よりシミュレーションに近い予測をしていることが観測

される。

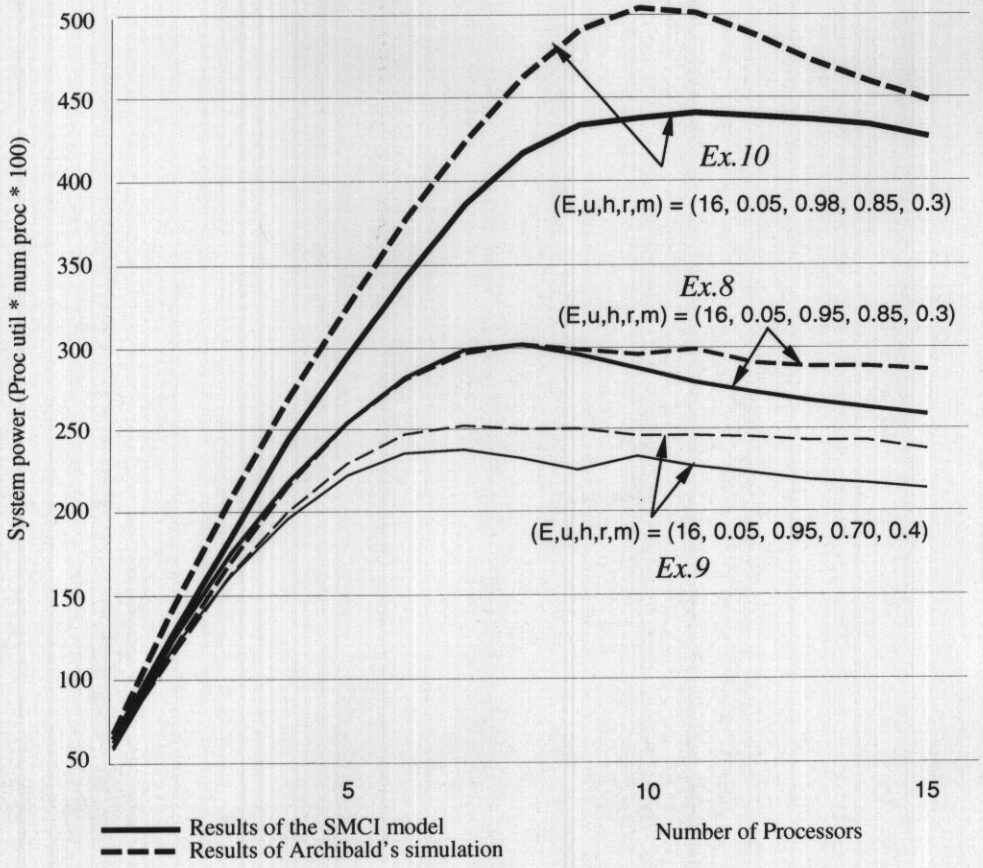


図 4.5 共有ブロック数１６での比較

また、共有ブロック・アクセス率が 5%で、プライベート・ブロック・キャッシュ・ヒット率が 95%の時、リード・リクエスト率が 70%, 85%に対して、平均誤差はそれぞれ 6.5%, 5.5%と精度が良いが、ヒット率が 98%の時には、共有ブロック数 16 に対して 14.5%, 128 に対して 8.5%と誤差が大きく、共有ブロック・アクセス率 0.1 の時には 18.0% と最大の誤差を記録している。このことから、本実験環境における SMCI モデルは、キャッシュ・コヒーレンス制御命令が多発しない場合の精度が劣るものと考えられる。また、本実験にて注目すべきことは、プロセッサ・パワーの最大値を与える入力パラメータ $E = 1024$, $u = 0.05$, $h = 0.98$, $r = 0.85$ と、最小値を与える入力パラメータ $E = 16$, $u = 0.05$, $h = 0.95$, $r = 0.7$ に対して、誤差はそれぞれ 3.0%と 2.5%であったことである。すなわち、一連の

実験における上限値と下限値を、SMCI モデルは極めて正確に予測していることになる。

文献 [7] のシミュレーションは、各実験とも 25,000 システム・サイクル、実験全部で $10*15*25,000 = 3,750,000$ システム・サイクル走らせなければならない[10]のに対し、SMCI モデルでは、実験全部を行なうのに、卓上のワークステーションで約 0.0001 秒しか必要としなかった。しかも、並列計算機のシミュレーションは、プロセッサ台数が増えるに従って実行時間が長くかかるのに対し、SMCI モデルでは、プロセッサ台数と実験時間はほぼ無関係である。実際、SMCI モデルで 1,024 台のプロセッサによるシステム・パワーを評価するにも、計算時間の大部分を占める極限確率の計算に 0.6 マイクロ秒[11]しか必要としない。従って、（要件 2）を満たしている。

これらの実験結果から、SMCI モデルが本論文の要件を全て満たすことが示され、SMCI モデルの妥当性が検証された。

4.4.2 システムの分析

図 4.6 および 図 4.7は、プロセッサ数がシステムの内部状態に与える影響を示している。いずれも、前節で行なわれた実験と同様のパラメータ（$E = 128$, $u = 0.05$, $h = 0.95$, $r = 0.85$, $m = 0.3$）で計算された結果である。

図 4.6は計算およびデータ・リクエストに関する状態 $\{COM, Rc, \overline{Rc}, Wc, \overline{Wc}\}$ の極限確率を示している。ここでは、プロセッサ数が 12 より多くなったあたりで、$\overline{Rc}$が極限確率のかなりの部分を占めていることが観測される一方、$\overline{Wc}$はさほどでもないことが示されている。これに対して、実際のデータ通信に関わる状態（Rc や Wc）は、あまり大きな値を取っていない。

図 4.7は、キャッシュ制御命令に関する状態 $\{WB, \overline{WB}, RP, \overline{RP}, HI, \overline{HI}, MI, \overline{MI}\}$ の極限確率を示している。待ち状態以外の状態では、WBと HIの割合が他に比べ大きいことがわかる。また、WBはプロセッサ数の増大に対して増加しているのに対し、RP, HI, MIは逆に減少している。これは、プロセッサ数の増加が、

[10] Archibald のシミューレションとほぼ同じ条件でシミュレーターを開発し、DEC 3000 ワークステーション上で同じ実験を行なったところ、全実験を行なうのに 1 時間以上かかった。

[11] DEC 3000 上で

インバリデーション命令の多発に直結しているためと考えられる。待ち状態に関しては、プロセッサ数が 7 になるのを境にして、$\overline{WB}$が急激に増えていることが観測される。

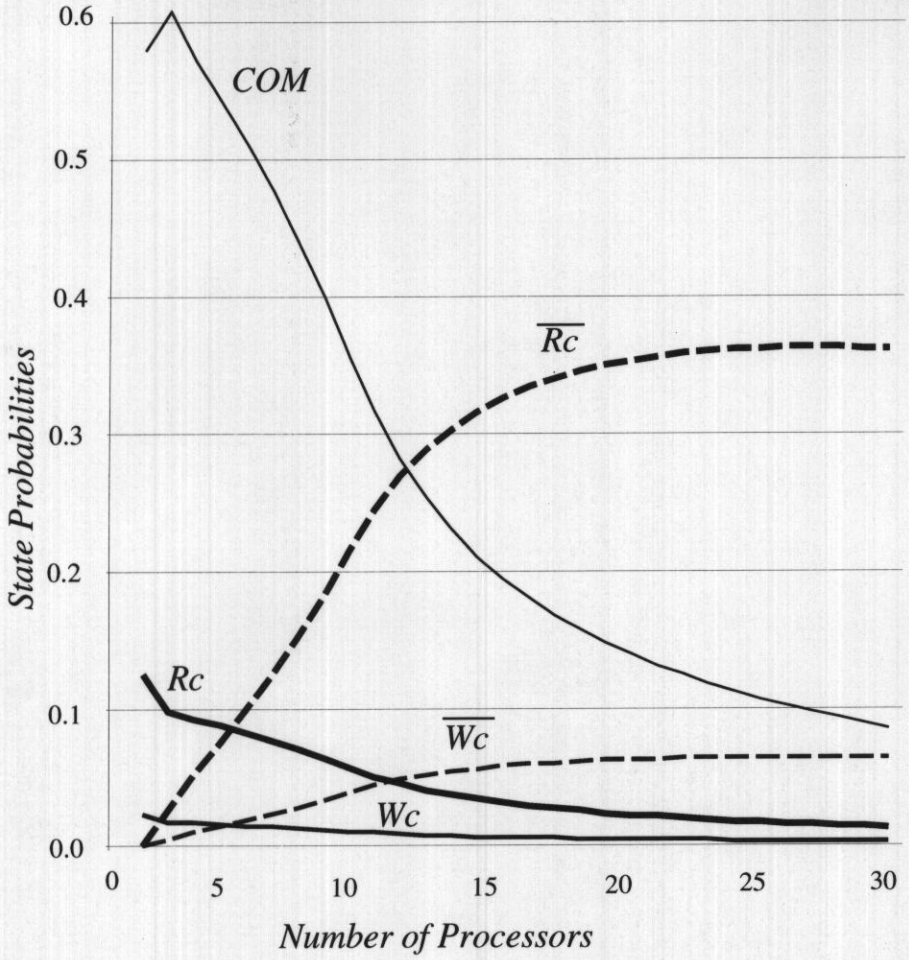


図 4.6 計算状態およびデータ・リクエストに関する状態を示す極限確率

これらの実験から、与えられたアーキテクチャ仕様では、プロセッサ数 7 でキャッシュ・コヒーレンス制御のオーバーヘッドが顕著となり、12 でバスが飽和していることがわかる。

さらに、別のパラメータを用いた二種類の実験を行なった。A）プロセッサ数は 12 に固定し、リード・リクエスト率を 0.7 から 1.0 まで変化させたものと、B）

4.4. 評価と考察

同じくプロセッサ数は12で、共有ブロック参照率を0.0から0.1まで変化させたものである。他のパラメータは先の実験と同じである。

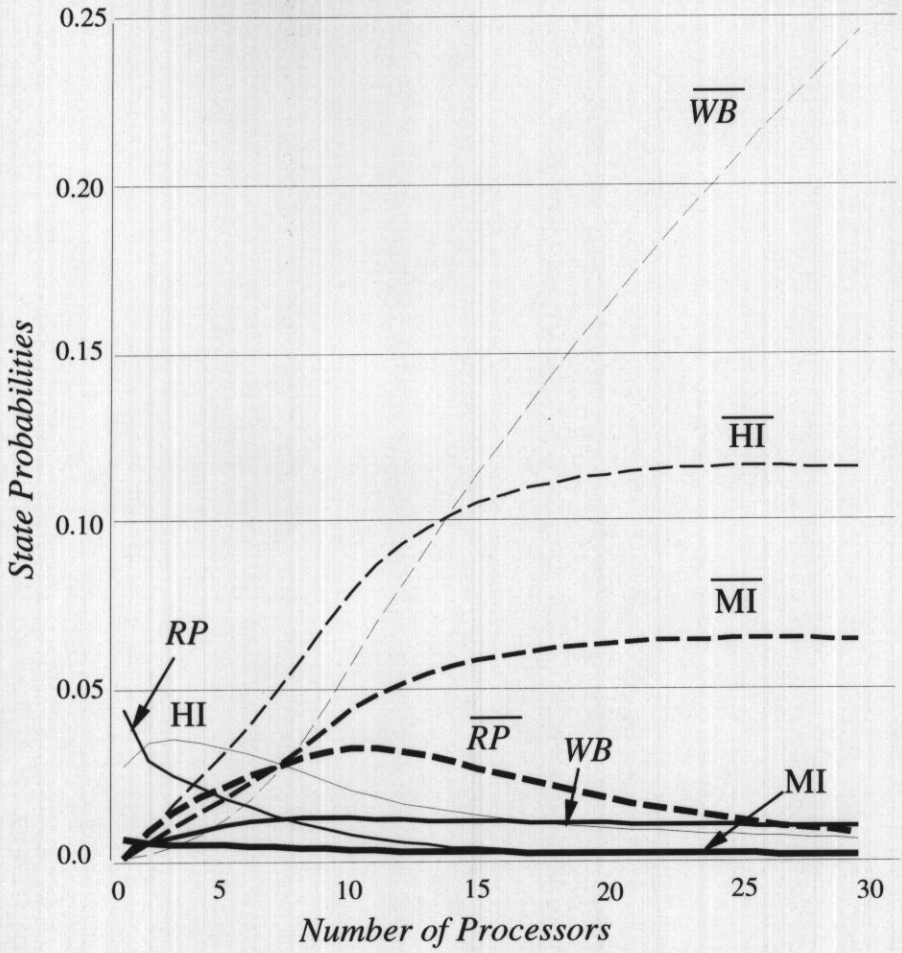


図 4.7 キャッシュ制御命令に関する状態を示す極限確率

図4.8は、パラメータA）の場合の状態$\{COM, Rc, \overline{Rc}, HI, \overline{HI}, MI, \overline{MI}, WB, \overline{WB}\}$の極限確率を表している。図から明らかなように、$COM$, Rc, $\overline{Rc}$はリード・リクエスト率に伴って増加している。一方、MIと$\overline{MI}$は逆に減少し、リード・リクエスト率が1.0の時には、0になっている。しかしながら、同じインバリデーション発行に関する状態のHIと$\overline{HI}$はリード・リクエスト率に伴って増加しており、一見矛盾しているように思える。この理由は、入力パラメータm（リプレース対

象のブロックが既にモディファイされている確率）がリード・リクエスト率の影響を受けない設定になっているからである。実際、リード・リクエスト率が 1 の時、m は明らかに 0 のはずである。従って、m は他の入力パラメータから導出されるべきである。

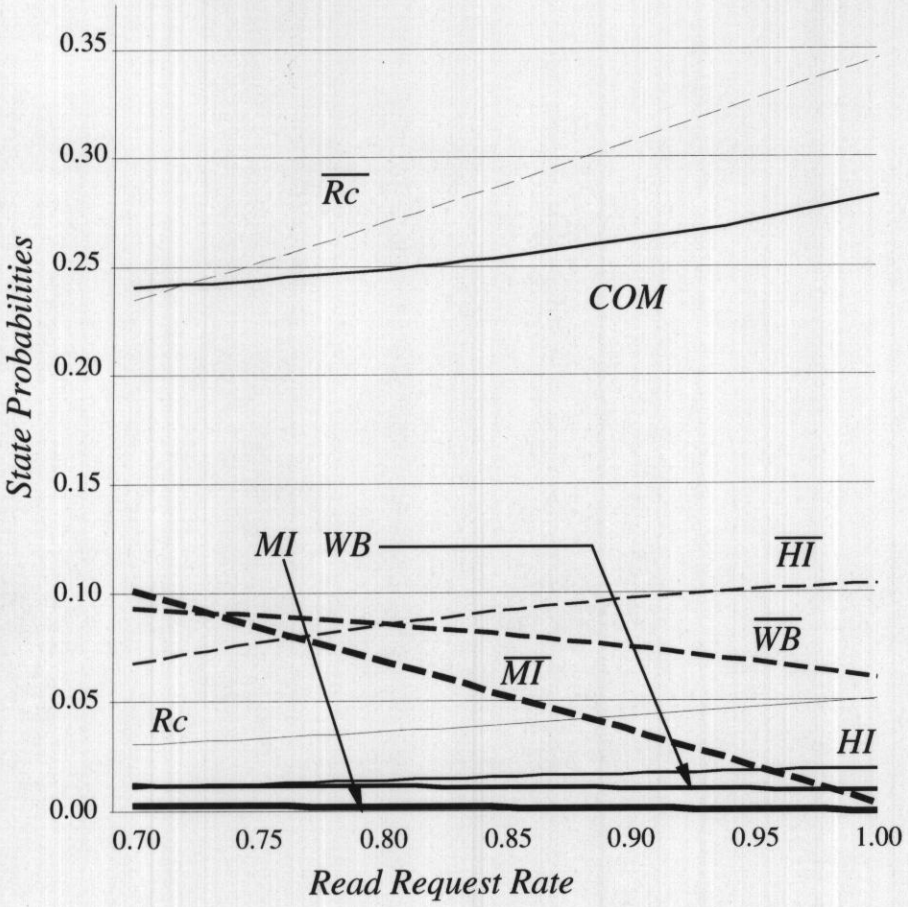


図 4.8 リード・リクエスト率と極限確率

図 4.9は、同様に、パラメータＢ）でのインバリデーションに関わる状態の極限確率を表している。インバリデーションに関連する状態、MI, HI, WBは、大きい値を取っていないが、それらに対する待ち状態はシステム性能に大きく影響を与えていることがわかる。従って、本実験において与えられた仕様のアーキテクチャにおいて、システム性能を向上させるキーポイントは、インバリデーショ

ン命令の削減、すなわち、効率的な並列プログラムが与えられることであると結論づけられる。また、興味深い現象としては、共有ブロック参照率が0である時でさえ、WBと$\overline{WB}$が正値を取っていることがあげられる。この理由は、Synapse

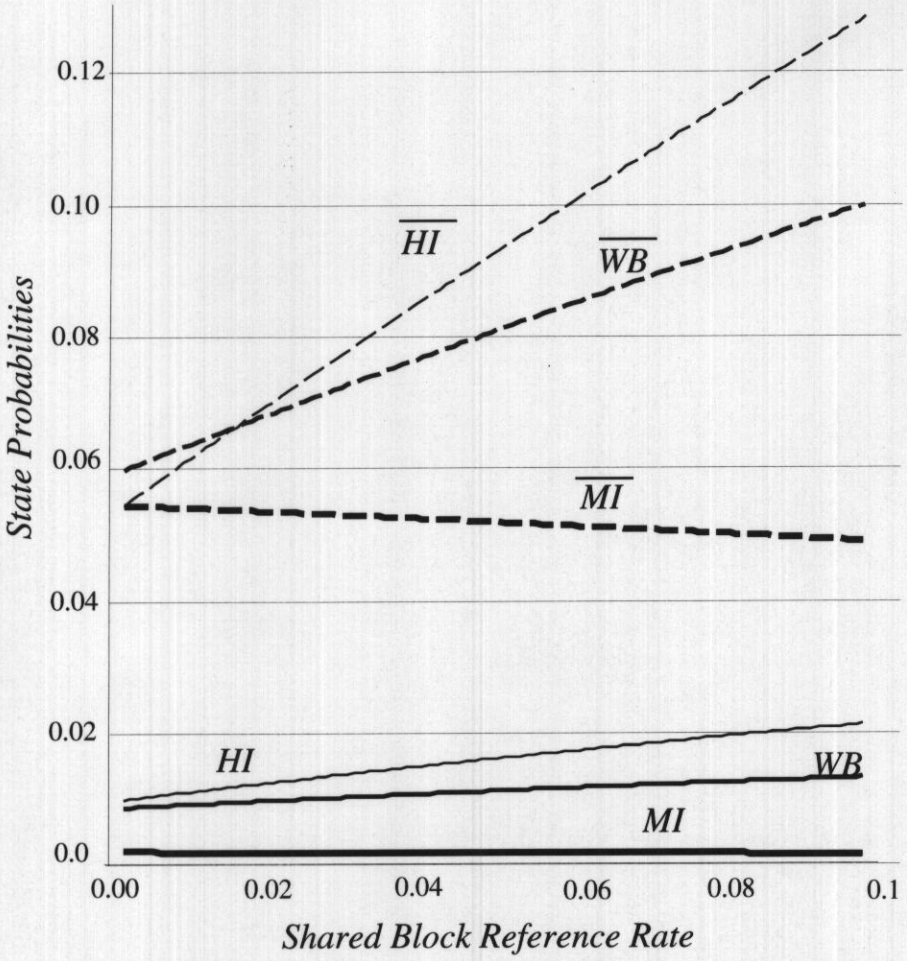


図 4.9 共有ブロック参照率と極限確率

プロトコルの定義によって、プライベート・ブロックに対するライト・ヒットもインバリデーション命令の発行を引き起こし、本来その命令は、他のどのプロセッサも受ける必要はないにもかかわらず、本章で用いられている$\alpha()$（インバリデーション発生時に、無効化される平均ブロック数を決める関数）が、それを考慮していないからである。また、これは前節での実験結果である、『キャッシュ・コヒー

レンス制御命令が多発しない場合の精度が劣る』という現象を裏付けている。

4.4.3 考察

本評価実験において、Archibald のシミュレーション結果と生じた差異について考察する。4.3.6節において、一回のインバリデーション命令あたりの、無効化される平均ブロック数$\alpha()$ と、アクセス・バースト長 $ls()$ の簡単な近似を行なったが、この関数を適切に定義することによって、当該シミュレーションとの差異は減少する。しかしながら、この値は与えられる並列プログラムによって大きく変動するものであることが知られており [24]、与えられた並列プログラムと、それを実行する並列計算機アーキテクチャから、その数を解析的に予測する研究は行なわれていない。Archibald のシミュレーションにおいても、一回のインバリデーション命令に対する平均無効化ブロック数や、アクセス・バースト長については報告はされていない。従って、この平均無効化ブロック数と、アクセス・バースト長に関する解析的なモデルが研究されない限り、当該シミュレーションにのみ有効な関数$\alpha()$ や $ls()$ を提案しても意味がないため、今回の実験では簡単な近似のみで実験を行なった。この平均無効化ブロック数に対する実用的なモデル化は、今後の重要な研究課題である。

既存の逐次プログラムを入力すると、自動的に並列性を抽出し、並列プログラムに変換する自動並列化コンパイラは、既存のソフトウェア資産を、今後より一般的になるであろう並列計算機で稼働可能にするという意味において、非常に重要な研究である。自動並列化コンパイラ研究の骨格は、さまざまな並列性の抽出手法 [57] であり、これらを実装した実際の自動並列化コンパイラも既に開発されている。例えば、イリノイ大の Parafrase-2 [58] は、40 種類以上の並列性抽出手法を提供する自動並列化コンパイラである。ユーザーはパスと呼ばれるこれらの並列性手法をコンパイラに指定し、プログラムの並列化を行なわせる。ユーザーの指定する各パスは、単体ではよく研究された並列化手法ではあるが、他のさまざまなパスとの組合せ[12]や、生成された並列プログラムを実行する並列計算機アーキテクチャが、実行性能にどのような影響を与えるかに関しては十分な研究がなされておらず、ユーザーが経験と勘に頼って並列化のパスの選択を行なっている

[12] しかも順序も関係することが知られている。

のが現状である。これを自動化するには、計算量の少ない並列計算機の性能評価手法が必要となる。

例えば、MVA 法を利用したキューイング・ネットワークは計算量の少ない並列計算機の解析モデルとして利用可能なことが知られているが、通常は実際の評価にワークステーション上で秒単位の計算時間が必要である。これはアーキテクチャ設計の指針として利用するには十分コストの安い手法であるが、自動並列化コンパイラの最適パス選択に用いるには難しい。なぜなら、前述 Parafrase-2 の 40 種類のパスから、例えば 5 種類のパスを順序づけて選ぶだけでも、78,960,960 通りの候補が存在するため、一つの候補をテストするのに秒単位の計算時間を必要とするのでは、全ての候補をテストするのに、数年を要してしまうからである。本論文で提案した SMCI モデルは、4.4.1節で示したように、一回の評価あたり、マイクロ秒単位しか必要とせず、自動並列化コンパイラの最適パス選択のための評価モジュールとしての応用が極めて有望である。SMCI モデルの、自動並列化コンパイラ最適パス選択への適用も今後の重要な研究課題である。

4.5 結論

キャッシュを持つ共有メモリ型並列計算機に対する、セミマルコフ過程を利用した解析モデルとして、SMCI モデルを提案した。SMCI モデル構築にあたっては、まず、対象となる並列計算機モデルを明確にした後、SMCI モデルの満たすべき要件と必要な仮定を述べ、プロセッサ・リクエスト率の分類の定義を与えた。これらの準備の後、既存のシミュレーション・モデルと同じ入力パラメータを利用できるよう考慮しつつ、プロセッサの各状態の定義とそれらの遷移確率の導出および各状態間の滞在時間分布関数を定義し、SMCI モデルを完成させた。構築された SMCI モデルは、通常のメモリ・アクセスはもとより、キャッシュ・コヒーレンス制御のためのリクエストや、キャッシュ・ミスによって引き起こされるキャッシュ・ブロックのリプレースも記述可能なものであり、さらにそれらの待ち状態もモデル化できる。

SMCI モデルを用いて Synapse キャッシュ・コヒーレンス・プロトコルに従う並列計算機の性能評価を、Archibald のシミュレーションと同じパラメータで行

なったところ、プロセッサ数や共有ブロック数がプロセッサ・パワーに与える影響が、シミュレーションの結果と同様であることが確認できた。計算コストに関しては、SMCI モデルでの一回の極限確率計算にかかる時間は卓上ワークステーションでわずか 0.6 マイクロ秒であった。この実行時間は、典型的なシミュレーション手法に対しておよそ$\frac{1}{10^9}$、ＭＶＡ手法を利用したキューイング・ネットワークによる解析モデルに対しておよそ$\frac{1}{10^6}$であり、SMCI モデルの計算量が極めて少ないことを示している。

第 5 章

書き込み放送型プロトコル対応 SMCI モデル

本章では、前章で提案した SMCI（Semi-markov Memory and Cache coherence Interference）モデルを、より複雑なキャッシュ・コヒーレンス・プロトコルである、Dragon に対応させるための考察を行なう。Synapse は無効化／主記憶転送型の、最も単純なコヒーレンス・プロトコルであり、SMCI モデルの汎用性を主張するためには、より複雑なコヒーレンス・プロトコルにも同モデルが同様に適用可能なことを示さなければならない。そこで本章では、同モデルの、書き込み放送／キャッシュ間転送型の代表的なプロトコルである Dragon への適用を行なう。この適用は、リクエスト率と状態定義の簡単な修正で可能であり、構築された SMCI/Dragon モデルは、SMCI/Synapse モデルと同程度の計算量で、それ以上に正確な性能予測を可能とすることを示す。これによって、SMCI モデルはコヒーレンス・プロトコルの種類に依存しない、汎用的な解析モデルであると言える。

5.1 はじめに

第4章で述べたSMCIモデルは、最も簡単な無効化／主記憶転送型のキャッシュ・コヒーレンス・プロトコルである、Synapseをベースにしたものである。本研究の目的は、キャッシュ・コヒーレンス制御を前提にした共有メモリ型並列計算機に対する、安価な計算コストと詳細な分析能力を兼ね備えた解析モデル化手法の体系を構築することである。このため、その基本となるSMCIモデルが特定のキャッシュ・コヒーレンス・プロトコルに依存してはならない。

一方、キャッシュ・コヒーレンス・プロトコルは大雑把に分類すると、

- 無効化型／書き込み放送型
- 主記憶転送型／キャッシュ間転送型

の組合せが考えられる[1]。ここで、転送型の分類は、データ・ブロックの供給元がどこかという分類 [23] に従うのではなく、物理的にどこからデータを転送するかという分類 [73] に従う。前者の意味ではSynapseプロトコルはキャッシュ間転送に分類されるが（メモリ経由のキャッシュ間転送）、後者の意味では主記憶転送（キャッシュ間の直接転送がない）である。

無効化／主記憶転送型の代表的なプロトコルとしては、Symmetry[73] が知られているが、Synapseプロトコルは Symmetry のサブセットと考えられる。実際、Symmetry の状態（Clean-Exclusive, Clean-Shared, Dirty, Invalid）から Clean-Exclusive の状態を除いたものが Synapse の状態と考えられ、ライト・ヒット時に当該ブロックが Clean である時にメモリ・ロードを行なうという Synapse の無駄な動作も Symmetry では解消されている。

筆者は Synapse を、キャッシュ・コヒーレンス・プロトコルのプリミティブなものと位置付けている。なぜなら Synapse は、最小の状態数（状態数３）、最大のバス・トラフィック [7]、最小のコスト（メモリ上の各ブロックにモディファイ・ビットが付加されているだけ）でコヒーレンス制御を可能としており、他のコヒーレンス・プロトコルは、結果的に、これらの基本要素を改善する形で実装されてい

[1]ただし、書き込み放送をサポートするがキャッシュ間転送をサポートしないアーキテクチャはコスト的に問題があるため、提案されていない。

るからである。これに対して、Dragon プロトコル [7] は、書き込み放送／キャッシュ間転送型のプロトコルであるが、最大の状態数（Invalid を含めると 5 ）、最小のバス・トラフィック [7]、最大のコスト（共有されているかどうかを検出する付加的なバス・ライン、キャッシュ間のワードおよびブロック転送）と、Synapse に対峙する位置付けにある。

本研究の中核となる SMCI モデルが、キャッシュ・コヒーレンス・プロトコルに依存していないことを示すために、本章では Synapse と正反対の性格を持つ、Dragon プロトコルに対する SMCI モデルの適用について論じる。SMCI/Dragon モデル構築の基本方針は SMCI/Synapse モデルと同じであるが、無効化と書き込み放送の表し方、内部変数の導出のしかた、それらに対応した別種のリクエスト率の定義等が SMCI/Dragon モデル構築に必要となる。

本章では第 4章と同様の基本方針、並列計算機モデル、状態記述定義手法、実際の計算手順を仮定する。

5.2 モデル化の方針および仮定と準備

SMCI/Synapse モデル同様、SMCI/Dragon モデルでは、まず、プロセッサの各状態の極限確率とリクエスト率に関する定式化を行なう。すると、適当な初期値を与えることで、極限確率が得られる。これを基にして、リクエスト率の修正を行なう。以後、この繰り返しを行なうことにより、リクエスト率における収束点（すなわちシステムの定常状態）を見つけ出す。SMCI/Dragon モデルの妥当性は、SMCI/Synapse モデル同様、文献 [7] のシミュレーション結果との比較でなされる。

5.2.1 仮定

モデル化のための仮定は、一部を除いて第 4章と同様である。

(仮定 1) 各プロセッサの挙動は同一の確率過程でモデル化され、各プロセッサは独立に動作する。

(仮定 2) 各共有ブロックの状態は、同じ確率で表される。

(仮定 3) 各プロセッサからの通常のデータ・リクエスト（キャッシュ・コヒーレンス制御のためのリクエストを除く）は、独立した分布に従う。

(仮定 4) リクエストを発行したり受けとったりする時間間隔は独立で、それぞれ、平均λ_{nor}（リクエストを発行）、λ_{CT}（キャッシュ間転送リクエスト要求）、λ_{Upd}（書き込み放送要求）の離散時間幾何分布に従う。リクエストのサービス時間はハードウェア構成により定まる一定分布に従う。

(仮定 5) 各待ち状態での滞在時間分布は、一括して一つの幾何分布で近似する。

(仮定 6) プログラムは安定稼働している状態、すなわち、起動時のページングやキャッシュ・ヒット率の悪さは考慮しない状態、を仮定する。またキャッシュは既に使い切った状態で、それに伴うキャッシュ・ブロックのリプレースについては考慮する。

5.2.2 リクエスト率の定義

SMCI/Synapse モデルにおける各種リクエスト率（4.2.3節参照）は、通常のリクエストに加え、インバリデーション・リクエストと、それに起因するライトバックのためのリクエストが考慮されていた。これに対して、SMCI/Dragon モデルでは、書き込み放送およびキャッシュ間転送のためのリクエストが対象となる。

- **ノーマル・リクエスト率**、φ_{nor}、とは、プロセッサが計算状態の時に引き続いてデータ・アクセスのために発行する単位時間あたりのリクエストの率である。

- **メモリ・リクエスト率**、φ_{mem}、とは、メモリ中のデータに対して発行される単位時間あたりのリクエストの率である。本章では便宜上、通常のキャッシュ・ミスに伴うブロックのリプレースも、メモリ・リクエストの一部と考える。

- **キャッシュ間通信リクエスト率**、φ_{cc}、とは、キャッシュ・ミスが発生した時のキャッシュ間転送や、ライトが行なわれた時の書き込み放送を行なうためのリクエスト率である。ただし、プロセッサAがプロセッサBにキャッシュ

間転送を要求した時、当該リクエストに加えて、実際のデータ転送に対するプロセッサBのオペレーションもリクエストと考えることができるが、本章では一連の動作は連続して行なわれるものとし、プロセッサAのリクエストのみがφ_{cc}を規定するものとする。

- **ネットワーク・リクエスト率**、φ_{net}とは、ネットワークに対して発行されるリクエストの率。メモリ・リクエスト率とキャッシュ間通信リクエスト率の和で定義される。

5.3 SMCI/Dragon モデル

5.3.1 プロセッサの状態定義

Dragon キャッシュ・コヒーレンス・プロトコルに基づいた SMCI モデルを構築するために、まず、プロセッサの状態定義を行なう。表 5.1は SMCI/Dragon モデルにおけるプロセッサの状態定義を表す。また、表 5.2に同状態の集合を示す。これは本節における内部変数導出の際の表記を見やすくするための集合であり、モデル構築自体に必要なものではない。

5.3.2 状態遷移

図 5.1は定義された状態間の遷移を示す。それぞれの状態間の遷移は、ネットワークもしくはキャッシュに対して、リクエストが発行／完了した場合、ブロックのキャッシュ間転送や、書き込み放送要求が発生／完了した場合に起きる。図 5.1において、h はプライベート・ブロックに対するキャッシュのヒット率、Hは共有ブロックに対するキャッシュのヒット率、rはプロセッサからのデータ・リクエストに対するリード・リクエストの割合、t は自分以外のキャッシュに目的とするブロックのコピーが存在しない確率、wは（ネットワークに対して）リクエストを出した時に待ち状態に陥る確率、u はリクエストが共有ブロックへのアクセスである割合、m はリプレース対象のブロックが既にモディファイされている確率、yは COM状態からの遷移確率に対するデータ・リクエスト（を確率と考

表 5.1 SMCI/Dragon モデルの状態定義

COM	計算状態（**COM**putation）
Rh	キャッシュからの読み込み（**R**ead **h**it）
WhE	キャッシュの Clean/Dirty-Exclusive ブロックへの書き込み（**W**rite **h**it **E**xclusive）
WhS	キャッシュの Clean/Dirty-Shared ブロックへの書き込み（**W**rite **h**it **S**hared）
WhB	WhSに起因する書き込み放送（**W**rite **h**it & update by **B**roadcast）
$\overline{WhB}$	WhBへの待ち状態
Rm	キャッシュ・ミスによる主記憶読み込み（**R**ead from **m**emory）
$\overline{Rm}$	Rm への待ち状態
Rc	他のキャッシュからの、キャッシュ・ミスによる読み込み（**R**ead from another **c**ache）
$\overline{Rc}$	Rc への待ち状態
Wm	キャッシュ・ミスによる主記憶書き込み（**W**rite to **m**emory）
$\overline{Wm}$	Wm への待ち状態
Wc	キャッシュ・ミスのため、他のキャッシュからロードされたブロックへの書き込み（**W**rite to a block loaded from another **c**ache）
$\overline{Wc}$	Wc への待ち状態
WcB	Wc に起因する書き込み放送（**W**rite miss & update by **B**roadcast）
$\overline{WcB}$	WcBへの待ち状態
RP	キャッシュ・ミスの結果生じるブロックのリプレースに付随するライトバック（**RP**lace）
$\overline{RP}$	RPへの待ち状態
FL	キャッシュ・ミスの結果生じるブロックのリプレースに付随するフラッシュ（**FL**ush）
CT	キャッシュ間転送要求のためのバス・ロード（**C**ache **T**ransfer）
Upd	書き込み放送要求によるブロックの更新（**Upd**ate）

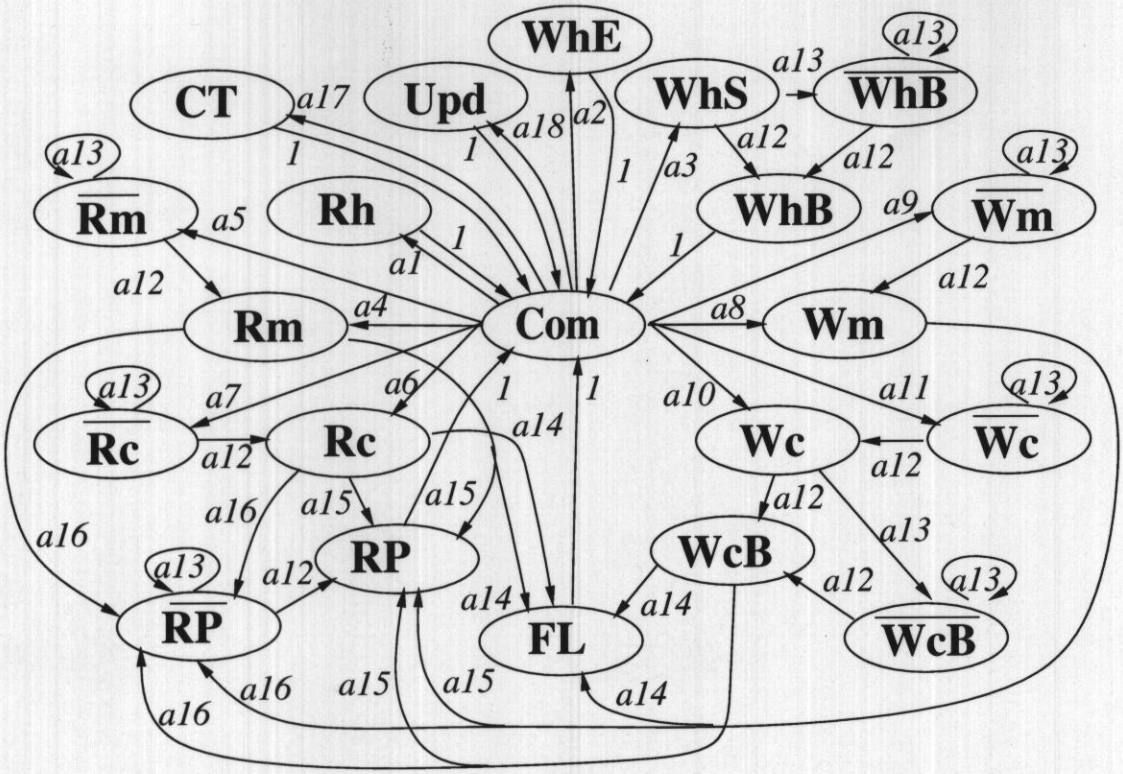
表 5.2 SMCI/Dragon モデルの状態の集合

Q	SMCI/Dragon モデル全状態の集合
Q_{mem}	$= \{Rm, Wm, RP\}$
Q_{mem_w}	$= \{\bar{i} \mid i \in Q_{mem}\}$
Q_{cc}	$= \{Rc, Wc, WhB, WcB\}$
Q_{cc_w}	$= \{\bar{i} \mid i \in Q_{cc}\}$
Q_{net}	$= Q_{mem} \cup Q_{cc}$
Q_{net_w}	$= \{\bar{i} \mid i \in Q_{net}\}$

えた時）の割合、zは状態COMからのキャッシュ間通信に起因する遷移に対するキャッシュ間転送要求に起因する遷移の割合、を意味する。

第4章同様、H、tは Wang らのモデル [68] により導き出され、w、y、zは、プロセッサの状態の極限確率から導き出され、その他は入力パラメータとして与えられる。

状態COMからの遷移は、データ・リクエストが発生した場合と、他のキャッシュからの要求に応じた場合の二種類がある。後者の場合、自プロセッサの能動的な遷移ではなく、他のプロセッサ（キャッシュ）からの、キャッシュ間転送要求に対するブロックのロード（CTに遷移）、書き込み放送要求に対するブロックの更新（Upd に遷移）で、いずれもネットワークへのリクエストは発行せず、これらの状態からは、直接COMに遷移する。前者の場合、そのデータ・リクエストがリードかライトか、キャッシュ・ヒットするかしないか、対象となるブロックのコピーが他のキャッシュに存在するか否か、ネットワークが空いているかどうか、によってそれぞれ、Rh、Rm、Rc、$\overline{Rm}$、$\overline{Rc}$、WhE、WhS、Wm、Wc、$\overline{Wm}$、$\overline{Wc}$に遷移する。RcおよびWcでは、データは主記憶から供給されるのではなく、他のキャッシュから転送されることに注意されたい。WhSからはキャッシュ・コヒーレンス制御命令である書き込み放送を行なうために、WhBもしくは$\overline{WhB}$に、WhEからはCOM に遷移する。Wcでは定義により他のキャッシュにコピーが存在するため、WcBもしくは$\overline{WcB}$のみに遷移する。Rm、Rc、Wm、WcBからは、仮定により他のブロックのリプレースが必要であるため、リプレー



a1=y(h(1-u)+H u)r , a2=y(h(1-u)+H u t)(1-r) , a3=y H u(1-t)(1-r) ,
a4=y((1-h)(1-u)+(1-H)ut)r(1-w) , a5=y((1-h)(1-u)+(1-H)ut)rw , a6=y(1-H)ur(1-t)(1-w) ,
a7=y(1-H)ur(1-t)w , a8=y((1-h)(1-u)+(1-H)ut)(1-r)(1-w) ,
a9=y((1-h)(1-u)+(1-H)ut)(1-r)w , a10=y(1-H)u(1-r)(1-t)(1-w) , a11=y(1-H)u(1-r)(1-t)w ,
a12=(1-w) , a13=w , a14=1-m , a15=m(1-w) , a16=mw , a17=(1-y)z , a18=(1-y)(1-z)

図 5.1 Dragon プロトコルに従ったプロセッサの状態遷移

ス対象のブロックが $Dirty-Exclusive/Shared$ 状態である場合は主記憶に書き戻しを行なうために RP もしくは $\overline{RP}$ に、$Clean-Exclusive/Shared$ 状態である場合は単にフラッシュを行なうために FL に遷移する。

5.3.3 滞在時間分布

前節ではプロセッサの状態遷移に関して述べたが、本節では各遷移間の滞在時間について説明する。

まず、状態 COM からは、厳密には次に遷移する状態（Rh、Rm、$\overline{Rm}$、Rc、$\overline{Rc}$、WhE、WhS、Wm、$\overline{Wm}$、Wc、$\overline{Wc}$、CT、Upd）に従ってそれぞれ分布関数を与えるべきであるが、そのような関数を導き出すのは困難であるため、遷移先の状態に関わらず、データ・リクエストに起因する遷移の場合はパラメータ λ_{nor} の、ブロックのキャッシュ間転送や書き込み放送要求に起因する遷移の場合はパラメータ λ_{CT} および λ_{Upd} の幾何分布に従うものとする。次に、キャッシュやメモリのアクセスを行なう状態（Rh、Rm、Rc、Wh、WhB、Wm、Wc、WcB、RP、FL、CT、Upd）から次の状態への遷移は、4.3.3節同様の一定分布に従う。待ち状態（$\in Q_{net\_w}$）から次の状態への遷移も、平均 Wt の幾何分布に従うとする。

5.3.4 セミマルコフ過程の構築

先に述べた状態空間と遷移確率により定義される確率過程 $X=\{X_n\}_{n=0}^{\infty}$ は、プロセッサの状態を表現する。5.3.1節で示した本並列計算機モデルの定義と、5.2.1節で与えた仮定から明らかなように、この確率過程はマルコフ連鎖である。

また、X の各状態の滞在時間は、前節で与えた時間分布関数に従うので、その各滞在時間を $T=\{T_n\}_{n=0}^{\infty}$ で与えられる確率変数列と考えると、確率過程 $(X,T)=\{X_n,T_n\}_{n=0}^{\infty}$ は SMP（Semi-Markov Process）を形成する。

(X,T) の EMC（Embedded Markov Chain）である X は、図5.1からも明らかなように、エルゴード的であるので、その定常分布を求めることができる。よって、EMC の定常分布 $\{\pi_i\}$ と、各状態の平均滞在時間 $\{\eta_i\}$ を式 (3.2) に適用して、(X,T) の極限確率を求めることができる。

ここでは、式 (5.1) で、EMC の定常分布のみを示しておく。ただし、g は正規

化係数である。

$$
\begin{pmatrix}
\pi_{COM} \\ \pi_{Rh} \\ \pi_{WhE} \\ \pi_{WhS} \\ \pi_{WhB} \\ \pi_{\overline{WhB}} \\ \pi_{Rm} \\ \pi_{\overline{Rm}} \\ \pi_{Rc} \\ \pi_{\overline{Rc}} \\ \pi_{Wm} \\ \pi_{\overline{Wm}} \\ \pi_{Wc} \\ \pi_{\overline{Wc}} \\ \pi_{WcB} \\ \pi_{\overline{WcB}} \\ \pi_{RP} \\ \pi_{\overline{RP}} \\ \pi_{FL} \\ \pi_{CT} \\ \pi_{Upd}
\end{pmatrix}
= g \times
\begin{pmatrix}
1-w \\
r\left(h(1-u)+Hu\right)(1-w)y \\
(1-r)\left(h(1-u)+Htu\right)(1-w)y \\
(1-r)H(1-t)u(1-w)y \\
(1-r)H(1-t)u(1-w)y \\
(1-r)H(1-t)uwy \\
r\left((1-h)(1-u)+(1-H)tu\right)(1-w)y \\
r\left((1-h)(1-u)+(1-H)tu\right)wy \\
(1-H)r(1-t)u(1-w)y \\
(1-H)r(1-t)uwy \\
(1-r)\left((1-h)(1-u)+(1-H)tu\right)(1-w) \\
(1-r)\left((1-h)(1-u)+(1-H)tu\right)wy \\
(1-H)(1-r)(1-t)u(1-w)y \\
(1-H)(1-r)(1-t)uwy \\
(1-H)(1-r)(1-t)u(1-w)y \\
(1-H)(1-r)(1-t)uwy \\
m\left((1-h)(1-u)+(1-H)u\right)(1-w)y \\
m\left((1-h)(1-u)+(1-H)u\right)wy \\
(1-m)\left((1-h)(1-u)+(1-H)u\right)(1-w)y \\
(1-w)(1-y)z \\
(1-w)(1-y)(1-z)
\end{pmatrix}
\tag{5.1}
$$

5.3.5 極限確率とリクエスト率

第4章同様、SMCI/Dragon モデルを計算するには、対象とするハードウェア・アーキテクチャ・モデルと並列プログラム・モデルを表した入力パラメータが必要である。この入力パラメータは第4章で説明した SMCI/Synapse モデルのそれと全く同じものである。また、H および Ψ は Synapse/SMCI モデル同様、付録 A.1 を参照されたい。

5.3. SMCI/DRAGON モデル

Dragon プロトコルの特徴として挙げられるのは、キャッシュ・ミスが発生した時、当該ブロックのコピーを他のキャッシュが保持する場合には、主記憶に対す

表 5.3 SMCI/Dragon モデルで使用される内部変数

H	共有ブロックに対するキャッシュ・ヒット率
Ψ	あるキャッシュに保持されている共有ブロックの数
t	特定のブロックが他のキャッシュにない確率
w	リクエストを出した時に待ち状態に陥る確率
y	状態 COM からの遷移に対して、それがデータ・リクエストに起因する遷移である割合
z	状態 COM からのキャッシュ間通信に起因する遷移に対する、キャッシュ間転送要求に起因する遷移の割合
Wt	待ち状態の平均時間
λ_{CT}	COM 状態からブロックのキャッシュ間転送要求に起因する遷移を行なう際の時間分布パラメータ
λ_{Upd}	COM 状態から書き込み放送要求に起因する遷移を行なう際の時間分布パラメータ
φ_{nor}	ノーマル・リクエスト率
φ_{mem}	メモリ・リクエスト率
φ_{cc}	キャッシュ間通信リクエスト率
φ_{net}	ネットワーク・リクエスト率
$\{\pi_i\}$	EMC の定常分布
$\{P_i\}$	SMP の極限確率

るアクセスがなされるのではなく、そのコピーをキャッシュ間転送することである。この様子を本モデルで定式化するために、特定のブロックが自分以外のキャッシュに保持されていない確率 t を導入する。この t は SMCI/Synapse モデルで用いられた確率 d と同様の考え方により、次のように導出される。

$$t = \left(1 - \frac{\Psi}{E}\right)^{N-1} \tag{5.2}$$

なお、SMCI/Synapse モデルで用いられた内部変数 u_{md}は、Dragon プロトコルの定義により必要ではない。

さて、(X,T) を解析するには、これらのパラメータ以外に、w、y、z、Wt の値が必要である。しかしながら、これらの内部変数を求めるには、各種リクエスト率とネットワークのビジー率が必要となる。これらは、(X,T) の極限確率から導出されるので、本節ではこの導出過程について説明する。なお、SMCI/Dragon モデル構築に必要な内部変数の一覧を、表 5.3に示しておく。

ネットワークに対してリクエストを出した時、待ち状態に陥る確率 wは、下に示すように第 4章と全く同じ形で得られる。

$$w = Busy + (1 - Busy)(1 - Win) \tag{5.3}$$

ただし、$Busy$はあるプロセッサから見てネットワーク・ビジーである確率、Win は他のプロセッサからのリクエストとの競合に勝つ確率で、以下のように表される。

$$Busy = \begin{pmatrix} N-1 \\ 1 \end{pmatrix} \sum_{i \in Q_{net}} P_i \frac{\eta_i - 1}{\eta_i} \left(1 - \sum_{i \in Q_{net}} P_i \frac{\eta_i - 1}{\eta_i}\right)^{N-2} \tag{5.4}$$

$$Win = (1 - (1 - \varphi_{net})^N) \frac{1}{N \varphi_{net}} \tag{5.5}$$

SMCI/Synapse モデルでの各種リクエスト率の定義は、データ・リクエストとキャッシュ制御のためのリクエストに大別された。これは、データ・リクエストに対して、例えばインバリデーション・リクエストは明らかに異なる分布に従ったものである、という着眼点からの定義であった。しかしながら、Dragon プロトコルでは、Synapse プロトコル同様のリクエスト率の定義は無意味であるため、5.2.2節のような定義を採用したのである。

まず、ノーマル・リクエスト率は定義よりそのまま、

$$\varphi_{nor} = \frac{1}{\lambda_{nor}} \tag{5.6}$$

で表される。次にメモリ・リクエスト率は、状態が Q_{mem} に含まれる場合のリクエストであるから、次のように導出される。

$$\begin{aligned}\varphi_{mem} &= ((1-h)(1-u)+(1-H)tu)\,\varphi_{nor} \\ &+ \; m\,((1-h)(1-u)+(1-H)u)\,\varphi_{nor} \\ &+ \; Busy \sum_{i \in Q_{mem\_w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{mem\_w}} P_i \end{aligned} \tag{5.7}$$

キャッシュ間通信リクエスト率は、状態が Q_{cc} に含まれる場合のリクエストであるから、

$$\begin{aligned}\varphi_{cc} &= (1-H)(1-t)u\varphi_{nor} \\ &+ \; u(1-r)(1-t)\varphi_{nor} \\ &+ \; Busy \sum_{i \in Q_{cc\_w}} \frac{P_i}{\eta_i} + (1-Busy) \sum_{i \in Q_{cc\_w}} P_i \end{aligned} \tag{5.8}$$

で表される。定義により、ネットワーク・リクエスト率は、次のようになる。

$$\varphi_{net} = \varphi_{mem} + \varphi_{cc} \tag{5.9}$$

Wt は待ち状態の平均滞在時間であるから、第4章と同様、

$$Wt = \frac{\sum_{i \in Q_{net\_w}} P_i}{\sum_{i \in Q_{net\_w}} \frac{P_i}{\eta_i}} \tag{5.10}$$

で求められる。

状態 COM からの遷移に対して、その遷移がデータ・リクエストに起因する割合 y は、キャッシュ間通信要求の頻度から導出される。前述のキャッシュ間通信リクエスト率は、キャッシュ間転送リクエスト率、$(1-H)(1-t)u\varphi_{nor}$、書き込み放送リクエスト率、$u(1-r)(1-t)\varphi_{nor}$、と、それらの待ち状態からのリクエスト率に分けて考えられる。よって、あるプロセッサがキャッシュ間転送リクエストを受けとる確率 z_{CT} は、

$$z_{CT} = 1 - \left(1 - \frac{1}{\alpha(N,E)}(1-H)(1-t)u\varphi_{nor}\right)^{N-1} \tag{5.11}$$

書き込み放送リクエストを受ける確率 z_{Upd}は、

$$z_{Upd} = 1 - \left(1 - \frac{\alpha(N,E)}{N-1}u(1-r)(1-t)\varphi_{nor}\right)^{N-1} \tag{5.12}$$

となる。一方、状態 COMからデータ・リクエストに起因する遷移を起こす確率は、φ_{nor}を確率と見なしてφ_{nor}で表される。

よって、yは以下で表される。

$$y = \frac{\varphi_{nor}}{1-(1-\varphi_{nor})(1-z_{CT})(1-z_{Upd})} \tag{5.13}$$

同様に、状態 COMからキャッシュ間通信に起因する遷移に対する、キャッシュ間転送要求に起因する遷移の割合 zは以下のとおりである。

$$z = \frac{z_{CT}}{(1-(1-z_{CT})(1-z_{Upd}))} \tag{5.14}$$

z_{CT}と z_{Upd}は、状態 COMからの遷移に関する時間分布パラメータλ_{CT}とλ_{Upd}を導出する。

$$\lambda_{CT} = \frac{1}{z_{CT}} \tag{5.15}$$

$$\lambda_{Upd} = \frac{1}{z_{Upd}} \tag{5.16}$$

最後に、状態 COMでの平均滞在時間は、式 (3.1) を適用して、

$$\eta_{COM} = y\lambda_{nor} + (1-y)z\lambda_{CT} + (1-y)(1-z)\lambda_{Upd} \tag{5.17}$$

で与えられる。

5.3.6 SMCI/Dragon モデルの計算手順

SMCI/Dragon モデルの実際の計算手順は、SMCI/Synapse モデルのそれと同じである。すなわち、入力パラメータと適当な初期値から (X,T) の極限確率を求め、その結果を元にリクエスト率とネットワーク・ビジー率等の修正をし、以後収束点に至るまで繰り返しを行なう。適当な初期値としては、

$$\begin{aligned} \varphi_{cc} &= 0 \\ y &= 1.0 \\ z &= 0.0 \\ w &= 1-(1-\varphi_{nor})^N \\ Wt &= 1 \end{aligned}$$

で、φ_{net}が収束するまで計算をおこなう。

5.4 評価

SMCI/Synapse モデル同様、SMCI/Dragon モデルの正当性を証明するため、Archibald のシミュレーション結果 [7] との比較実験を行なった。実験環境は 4.4 節での SMCI/Synapse モデル評価実験と同じである。ただし、メモリやキャッシュのサービス時間は表 5.4の通りである。

表 5.4 キャッシュやメモリ・アクセスを行なう状態の平均滞在時間

Rh	WhE	WhS	WhB	Rm	Rc	
1	1	1	4	16	16	
Wm	Wc	WcB	RP	FL	CT	Upd
16	16	4	16	1	16	1

また、状態 COMの滞在時間分布を決めるパラメータλ_{nor}は、文献 [7] に従って、3 とした。

図 5.2、図 5.3、図 5.4は、それぞれ共有ブロック数 E が 1024、128、16 である時の、プロセッサ台数に対するシステム・パワー [7] を表している。入力パラメータは、4.4節での SMCI/Synapse モデル評価実験と同じである。

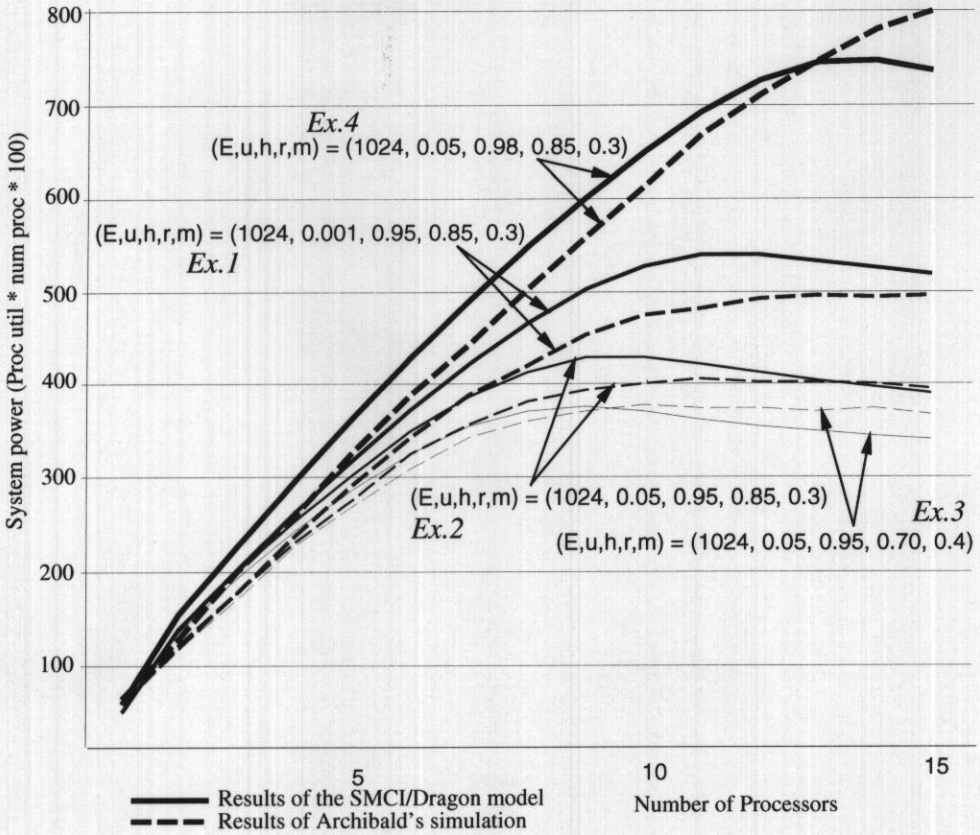


図 5.2 共有ブロック数１０２４での比較

本評価実験では、SMCI/Dragon モデルと Archibald のシミュレーション結果との差異は 5.4% であった。SMCI/Synapse モデルの評価実験における差異が 8.0% であったことを考えると、SMCI/Dragon モデルの解析能力の高さが示されただけではなく、キャッシュ・コヒーレンス・プロトコルに左右されない SMCI モデルの構築という、本研究の主目的の一つが達成されたと言えよう。

各実験結果をさらに分析すると、以下のようなことが判明した。まず、共有ブロック数 1024、128、16 でのシミュレーション結果との差異は、それぞれ 6.6%、4.7%、4.5%であった。このうち、共有ブロック数 1024 の実験には、最も精度の

5.4. 評価

劣る実験1（共有ブロック・アクセス率が0.1のケース）を含んでいるため、適当な共有ブロック・アクセス率が与えられた実験2から実験9までは、共有ブロック数に関係なく、正確な性能予測が得られたと言えよう。実際、実験1を含めた

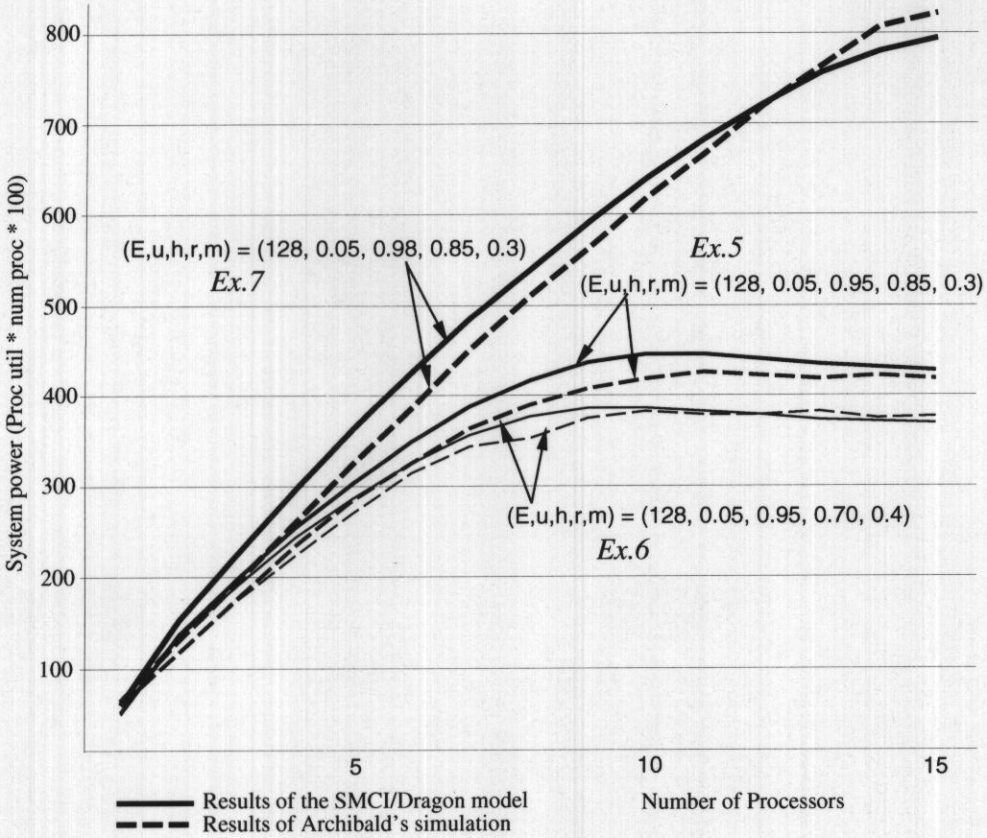


図 5.3 共有ブロック数１２８での比較

全実験結果の分散は1.85であるのに対し、実験1を除いた（$u = 0.1$ 以外の）実験結果の分散は0.73であった。従って、本論文で提案したSMCI/Dragonモデルは、適当なデータ共有が行なわれるような並列計算機モデルの性能予測は、かなりの精度で可能な反面、データ共有のほとんど行なわれない並列計算機モデルの予測は、正確さに問題があると言える。

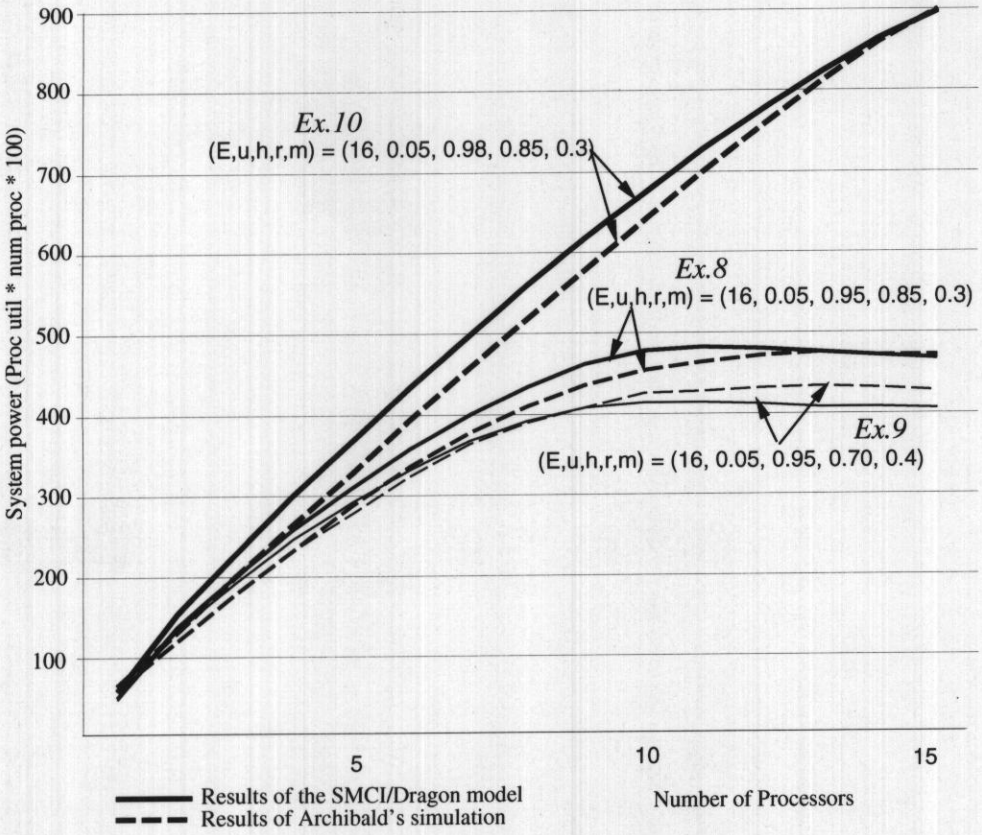


図 5.4 共有ブロック数１６での比較

5.5 結論

本研究は、将来の大規模かつ複雑な並列計算機の開発研究に先立って、その指針を与えるための性能予測技術となる、並列計算機の解析モデルを構築することにある。そのような解析モデルに必要とされる要件は、計算コストの安さと汎用性、拡張性であると筆者は考えている。

第4章では、上記要件を満たすモデル構築の基本モジュールとして、SMCI/Synapseモデルを提案したが、モデルの構築にあたっての状態定義がSynapseキャッシュ・コヒーレンス・プロトコルを前提としていたために、SMCIモデルの汎用性を主張するためには、他のコヒーレンス・プロトコルへの適用も示されなければならなかった。

キャッシュ・コヒーレンス・プロトコルには、(無効化型、書き込み放送型)／(主記憶転送型、キャッシュ間転送型)の組合せが考えられるが、Synapseプロトコルは無効化／主記憶転送型で、各種コヒーレンス・プロトコル中のプリミティブな性質を持つものと考えられる。これに対し、より高性能かつ高コストな書き込み放送／キャッシュ間転送型のプロトコルであるDragonは、Synapseに対峙するコヒーレンス・プロトコルと考えることができる。

そこで本章では、SMCIモデルのDragonプロトコルへの適用について考察した。その結果、プロトコルの違いはリクエスト率の修正で簡単に対応できることが判明した。また、状態定義も若干の変更で対応でき、SMCI/Synapseモデルと全く同様の手順でSMCI/Dragonモデルを構築することができた。構築されたモデルの検証は、SMCI/Synapseモデル同様、Archibaldのシミュレーション結果との比較が可能であった。その結果、SMCI/Synapseモデルよりも、より正確に性能予測が行なえるにもかかわらず、計算時間はほぼ同じであった。

以上のことより、SMCIモデルはキャッシュ・コヒーレンス・プロトコルに依存しない、並列計算機の汎用的な解析モデルと結論付けることができた。

なお、SMCI/Synapseモデルでのプロセッサの状態数は20で、SMCI/Dragonモデルでのそれは21であった。状態数だけの比較では、SMCI/Dragonモデルの方が複雑であったため、内部変数の導出等も複雑になるものと予想されたが、実際にはSMCI/Dragonモデルでの内部変数の導出は極めてすっきりした形で行なえたばかりでなく、実際の評価性能もSMCI/Synapseを上回るものであった。

第 5 章 書き込み放送型プロトコル対応 *SMCI* モデル

第 6 章

階層型 SMCI モデル

本章では、スケーラブル共有メモリ型並列計算機に対する解析モデルを提案する。提案される階層型 SMCI（Semi-markov Memory and Cache coherence Interference）モデルは、第 4章で提案した、キャッシュを持つ共有メモリ型並列計算機に対する解析モデルである SMCI モデルの自然な拡張であり、最近注目されているスケーラブル共有メモリ型並列計算機に対する性能予測を与える。階層型 SMCI モデルは SMCI モデル同様、データ・リクエスト、キャッシュ制御リクエストおよびそれらの待ち状態、計算状態等に関する詳細な分析を安価な計算コストで実現する。特に、リクエストは局所（クラスタ内通信）と広域（クラスタ間通信）を独立して扱うだけではなく、局所リクエストと広域リクエストの相互作用も定式化される。本章では、スケーラブル共有メモリ型並列計算機の例として、プロセッサ・クラスター型をベースとし、クラスタ内にはローカル・メモリとローカル・キャッシュ、クラスタ間には分散グローバル・メモリとグローバル・キャッシュを持つような階層的な構成の仮想的なアーキテクチャを示し、それに対する実際の階層型 SMCI モデルを構築する。

6.1 はじめに

プロセッサ・クラスタ方式の並列計算機は、将来のスーパーコンピューティングへの有力な一方式として、研究開発が推進されてきた。特に共有メモリを持つもの [30] は、自動並列化コンパイラ開発の観点からも望ましいものである。さらにキャッシュ制御方式の研究 [3] に伴い、キャッシュを持ったクラスタ方式の共有メモリ型並列計算機が精力的に研究されている [14, 34, 48]。いずれもネットワークの性能を補うものとして、キャッシュを有効利用しており、スケーラブル共有メモリ型並列計算機 [21] という新しい方向に向かっている。

一方、計算機アーキテクチャ研究において、提案されるアーキテクチャの性能予測が安価な計算コストで与えられれば、研究開発のターンアラウンド・タイムが縮小され、より良いアーキテクチャの出現が期待できる。実際、過去におけるスーパーコンピュータの利用事例の一つが、『次世代のスーパーコンピュータのシミュレーション』であったことは周知のことであろう。

これまで、スーパーコンピュータや高性能 RISC プロセッサ・システムにおける、次世代機開発にかかる期間は数年で、速度向上率も数倍に留まっていた。従って、次世代機開発のための現行機によるシミュレーション実行環境は、さほど問題にはならなかったものと思われる。しかしながら、超並列計算機の研究開発を考えた場合、プロセッサの速度向上率が数倍に留まっても、効率の良いネットワークやメモリ・システムを付加することで、全体として数百倍数千倍の速度向上率が可能である[1]。このような速度向上率の大きい並列計算機アーキテクチャ研究において、次世代機のシミュレーションを現行機で行なうという従来の手法では、研究のターンアラウンド・タイムが極めて悪くなることが予想される。

本研究の目的は、次世代のスーパーコンピューティングに対する有力なアーキテクチャである、スケーラブル共有メモリ型並列計算機の性能予測を、既存のシミュレーション技術による解析能力に迫る、計算コストの安い技術で実現することである。

[1]例えば、nCUBE 社の nCUBE1 は最大 1,024 プロセッサ構成で数十 MFLOPS の性能であったのが、nCUBE2 では最大 8192 プロセッサ構成で数十ギガ FLOPS の性能に、最新の nCUBE3 では最大 65,536 プロセッサ構成で数十テラ FLOPS の性能を持つと言われているが、nCUBE1 の発表から nCUBE3 の発表までの期間はわずか１０年足らずである。

6.1. はじめに

一般に、確率過程等を用いた並列計算機の解析モデルは、低コストで有効な手法である。第2章で概説したように、キャッシュを持たない簡単な構造の並列計算機に対する確率過程、特にマルコフ連鎖を利用した解析モデルの研究は、黎明期における Bhandarkar [10] に始まり、Marsan [44] や Mudge [50] らによって、当該モデルで問題となる状態数の爆発的増加を抑えるための試みがなされてきた。この一連の研究は、Mudge のセミマルコフ過程を利用した SMI（Semi-markov Memory Interference）モデル [49] により、一応の完成を見ることとなる。

しかしながら、並列計算機アーキテクチャ研究はその当時、既にキャッシュを持ちキャッシュ・コヒーレンス制御を行なうものが主流となりつつあり、その解析モデルとして、確率過程を用いたものは少なかった。Dubois らの研究は、システム全体の解析を行なうものではないが、キャッシュ・コヒーレンス・プロトコルの解析をマルコフ連鎖を利用して行なったものである [68]。

キャッシュを持ち、コヒーレンス制御を行なう並列計算機に対する解析モデルとしては、Vernon らの一般時間ペトリネットを利用したもの [64] があるが、計算時間に問題があったため、Vernon らはさらに同様のシステム・モデルに対して、MVA（Mean Value Analysis）法を利用したキューイング・ネットワークによる解析モデルを提案している [67]。MVA 法によるキューイング・ネットワークの解析は、その高い解析能力と安価な計算コストのために、キャッシュを持つ並列計算機の解析モデルとして現在最も利用されている手法の一つである [29, 15]。

これに対して、第4章で述べたように、筆者は **SMCI**（Semi-markov Memory and Cache Coherence Interference）モデルと呼ばれる、確率過程を利用した解析モデルを提案した。SMCI モデルは、Mudge らの SMI モデルを、キャッシュ・コヒーレンス制御を行なう並列計算機に適用するように拡張したものであり、その安価な計算コストにおいて、MVA 法によるキューイング・ネットワークの解析モデルを凌駕する。

先に述べたスケーラブル共有メモリ型並列計算機に対する解析モデルとしては、MVA 法を利用したキューイング・ネットワークによるものが極めて多い。Vernon らは、[67] の手法を逸早く大規模なシステムに適用している [35]。また、Bhuyan [13]、Algudady [6] らはデータ・ブロックの共有部分と非共有部分をハードウェア的に分離して操作するスケーラブル共有メモリ型のアーキテクチャを提案し、

MVA 法によるキューイング・ネットワークの解析モデルで評価を行なっている。

一方、第4章で提案した SMCI モデルの問題点は、大規模な並列計算機への適用であった。そこで本章では、スケーラブル共有メモリ型並列計算機に対する解析モデルとして、SMCI モデルを拡張した、**階層型 SMCI** モデルを提案する。本モデルは SMCI モデル同様、データおよびキャッシュ制御のリクエストとそれらの待ち状態、計算状態等に関する解析を、安価な計算コストで行なうことができる。

本モデルを、スケーラブル共有メモリ型並列計算機に、実際に適用するための仮想マシンとして、プロセッサ・クラスタを単位としたキャッシュを持つ分散共有メモリ型並列計算機アーキテクチャを導入する。この仮想マシンは、文献 [48] で示されるような階層的なメモリ・システムを持ち、文献 [36] で提案された階層型ネットワークを単純化したネットワークによって結合されたものである。

本章では、まず本解析モデルの対象となる並列計算機モデルについて、アーキテクチャならびに並列プログラミング・モデルの観点から説明し、本解析モデルの基本構築方針を述べる。次に、本解析モデル構築のための仮定を示し、プロセッサ・リクエスト率の分類の諸定義を与える。これらの準備の後、モデル化のための状態記述の定義を行い、内部変数の導出を示し、具体的な計算手順を与える。

6.2 並列計算機モデルと解析モデル

本節では、本章で提案する階層型 SMCI モデルの解析対象となる並列計算機モデルについて、アーキテクチャならびにプログラム・モデルの両側面から説明する。また、本章で提案する解析モデルの基本設計方針と、その満たすべき要件についても説明する。

6.2.1 アーキテクチャ・モデル

スケーラブル共有メモリ型並列計算機は数多く提案されている [21, 34, 48] が、これら実際のアーキテクチャに本解析モデルをいきなり適用すると、具体的な構築方法そのものが複雑となり、本章の主眼から離れたものとなる恐れがある。そこで、本章では本解析モデルを適用するための仮想的かつ簡略化されたアーキテクチャ・モデルを想定し、その並列計算機モデル上での解析モデル構築方法につ

いて論じることとする。以下、この仮想的なスケーラブル共有メモリ型並列計算機のことを、仮想マシンと呼ぶ。

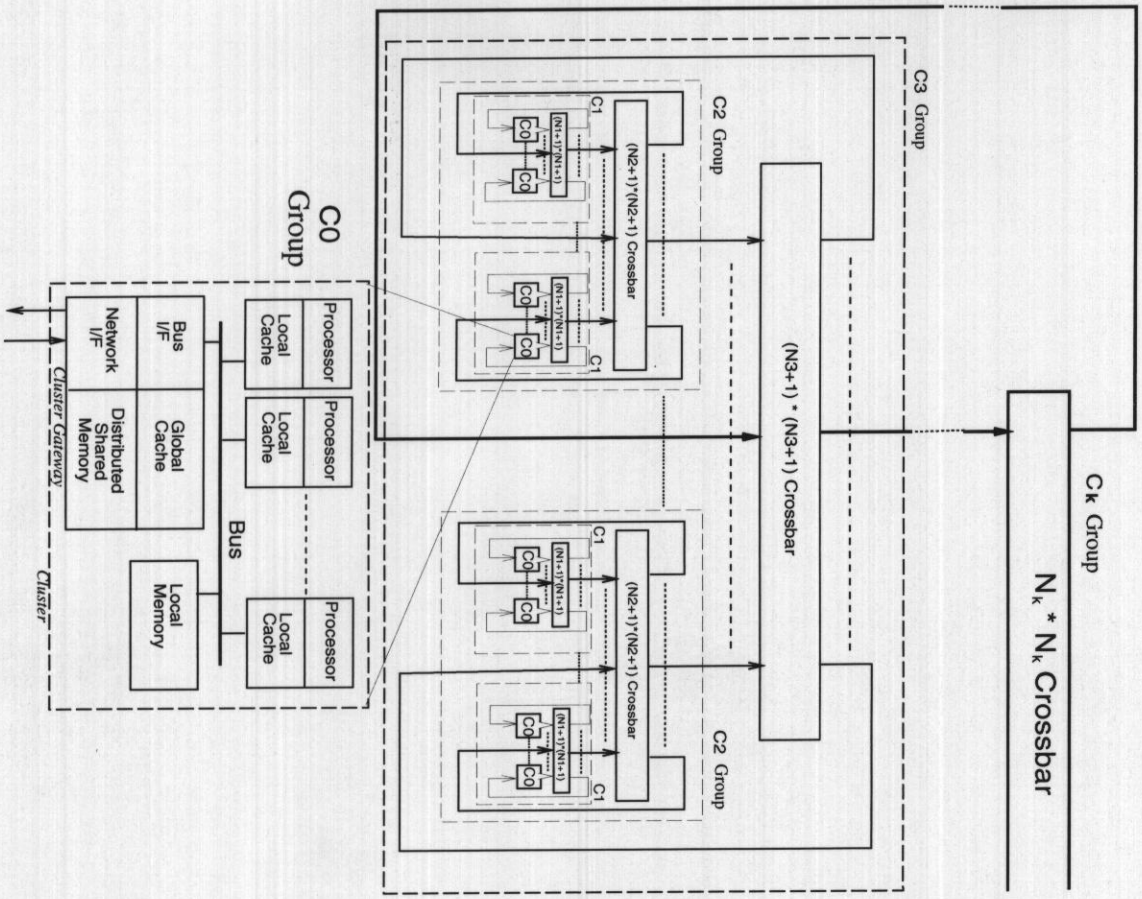


図 6.1 仮想マシンのアーキテクチャ

本解析モデルの解析対象である仮想マシンは、プロセッサ・クラスタ方式（以後、プロセッサ・クラスタをクラスタと呼ぶ。）と、階層的なメモリ・システム、階層型相互通信網により構成される。

クラスタは、プロセッサ固有のローカル・キャッシュを持つ、バスによって結合された、小規模共有メモリ型計算機である。階層的なメモリ・システムとしては、文献 [48] に示されるような、プロセッサ固有のローカル・キャッシュ、同一

クラスタ内のプロセッサからのみアクセス可能なローカル・メモリ、クラスタ固有のグローバル・キャッシュ、すべてのプロセッサからアクセス可能な各クラスタに分散配置されたグローバル・メモリによって構成されるものを考える。

各クラスタは、図 6.1に示されるように、**階層的クロスバー・ネットワーク**によって結合される。階層型クロスバー・ネットワークは、小規模なクロスバー・スイッチを階層的に結合配置したもので、通信方式は同期式とする。このように、kレベルの階層型クロスバー・ネットワークを使って、仮想マシンは$k+1$階層のグループで構成されることになる。

最下位のグループ C_0（クラスタ）は、N_0個のプロセッサとそれらに対応したローカル・キャッシュを、ローカル・メモリ、クラスタ・ゲートウェイとでバス結合したもので構成される。クラスタ・ゲートウェイは、バスとのインタフェイス、上位ネットワークとのインタフェイス、グローバル・キャッシュ、およびグローバル・メモリから構成される。本章では、ローカル・メモリおよびグローバル・メモリのバンク数は 1 を仮定する。これは、本解析モデル構築の簡略化のためである。

クラスタ内の各プロセッサはローカル・メモリを参照できるが、他のクラスタ内のローカル・メモリを参照することはできない。また、各クラスタに分散配置されたグローバル・メモリは全てのプロセッサによって参照される。プロセッサには最近のスーパースカラ方式等の高性能のものは仮定せず、その状態は、計算を行っている状態、ローカル／グローバルのキャッシュやメモリに対してリクエストを発行している状態、その状態への待ち状態の三状態からなるとする。

ローカル・キャッシュはローカル・メモリおよびグローバル・キャッシュの両方のデータ・ブロックをキャッシングし、Synapse プロトコル [7] によりスヌーピング方式のキャッシュ・コヒーレンス制御を行なう。グローバル・キャッシュは各クラスタに分散配置されたグローバル・メモリをキャッシングし、ローカル・キャッシュと同じく Synapse プロトコル [7] によりフルマップ・ディレクトリ方式 [3] のキャッシュ・コヒーレンス制御を行なう。さらに、ローカル／グローバル・キャッシュ間のコンシステンシは **MultiLevel Inclusion Property** [8] の保証（以後 MLI 保証と呼ぶ。3.4節参照）により実現する。また、両キャッシュはライトバックによるフルアソシアティブ方式で、ブロックのリプレース・アルゴリズ

ムはLRUとする。なお、Synapseプロトコルを選んだ理由は、モデル簡略化のための状態数の少なさである。また、ローカル・キャッシュおよびグローバル・キャッシュのブロック・サイズは、一般的には数倍から数十倍異なる方が効率が良いとされているが、本章では、モデル簡略化のために同一とする。

各プロセッサからのデータ・リクエストは、そのデータのアドレスがローカル・メモリに割り当てられている場合は、ローカル・キャッシュもしくはローカル・メモリに対して発行され、グローバル・メモリに割り当てられている場合は、ローカル・キャッシュ、グローバル・キャッシュ、クラスタ内のグローバル・メモリ（以後ローカルなグローバル・メモリと呼ぶ）もしくは階層型クロスバー・ネットワークを介して他のクラスタ内にあるグローバル・メモリ（以後リモートのグローバル・メモリと呼ぶ）のいずれかに対して発行される。あるプロセッサがリモートのグローバル・メモリにアクセスを行った場合、そのプロセッサの含まれるクラスタにおけるグローバル・キャッシュのみならず、リクエストを発行したプロセッサに固有のローカル・キャッシュにも、当該データ・ブロックがキャッシュ・インする。

C_1グループは、N_1個の最下位のグループC_0と、N_1+1入力N_1+1出力のクロスバー・ネットワークによって構成される。あるC_0グループ（に属するプロセッサ）からのリクエスト先が、同じC_1グループ内の他のC_0グループ（にあるリモートのグローバル・メモリ）にある場合は、そのリクエストは上位のグループに渡されずに、目的とするC_0にN_1+1入力N_1+1出力のクロスバー・ネットワークを介して渡される。一方、そのリクエスト先が同じC_1グループ内のC_0グループにない場合、そのリクエストは上位のC_2グループに渡される。

一般に、C_iグループ（ただし、$0<i<k$）はN_i個のC_{i-1}グループとそれらを結合するN_i+1入力N_i+1出力のクロスバー・ネットワークによって構成される。ここでも同様に、あるC_{i-1}グループからのリクエストは、そのリクエスト先が同一C_iグループ内の他のC_{i-1}グループにあるかないかに従って、当該C_{i-1}グループに渡されたり、上位のC_{i+1}グループに渡されたりする。

最上位のグループC_kは、N_k個のC_{k-1}グループとそれらを結合するN_k入力N_k出力のクロスバー・ネットワークによって構成される。従って、下位のC_{k-1}グループからのリクエストは別のC_{k-1}グループに渡される。

このようにして、仮想マシン全体では $N = N_1 \cdots N_k$ 個のクラスタ、NN_0個のプロセッサからなる並列計算機を形成することになる。

6.2.2 並列プログラム・モデル

本並列計算機モデルは、前節で述べたアーキテクチャ・モデルのもとで、一般的な **SPMD**（Single Program Multiple Data）型の並列プログラムが稼働しているものとする。ただし、データの属性として、広域ならびに局所に対応した共有／プライベート・ブロックを新たに定義する。ここで、広域とは本アーキテクチャ・モデルにおけるグローバル・メモリを、局所とはローカル・メモリを意味する。

あるデータ・ブロックが**広域共有ブロック**であるとは、そのブロックがプログラム実行中に、相異なるクラスタの複数のプロセッサからアクセスされることを意味し、同じく**広域プライベート・ブロック**であるとは、同一クラスタ内のプロセッサからしかアクセスを受けないことを意味する。

あるデータ・ブロックが**局所共有ブロック**であるとは、そのブロックがプログラム実行中に、同一クラスタ内の複数のプロセッサのみからアクセスされることを意味し、同じく**局所プライベート・ブロック**であるとは、唯一のプロセッサからしかアクセスを受けないことを意味する。

各プロセッサは、等価なコード列を、異なるデータを使ってそれぞれ実行する。ここで、データは上記分類に従い、グローバル・メモリにアドレス付けされる広域共有／プライベート・ブロック、ローカル・メモリにアドレス付けされる局所共有／プライベート・ブロックに分けられる。それぞれ広域ならびに局所データに対して、リード・アクセスの割合、共有ブロックの総数とそのアクセス比率を指定することにより、実際のアプリケーションに対応した並列プログラム・モデルの指定ができる。

ただし、本章で考える並列プログラムとは、実際のアプリケーションのうち、完全に並列化がなされている部分を言い、逐次実行部分は考慮しない。

6.2.3 解析モデル化の基本方針

第4章では、共有メモリ型並列計算機の解析モデル、SMCI モデルを提案したが、本章の主目的は、それを最近のスケーラブル共有メモリ型のアーキテクチャに拡張適用することである。

前節までに述べたような、大規模で複雑なスケーラブル共有メモリ型のアーキテクチャに、SMCI モデルをそのまま適用することは実質上不可能である。なぜなら単一のプロセッサに焦点を置いてその状態遷移を与え、システム全体の挙動を表現しようとすると、状態数が爆発的に増え、さらにそれらの状態間の遷移確率を与えることが極めて困難だからである。

そこで本章では、上記仮想マシン・アーキテクチャに SMCI モデルを拡張適用するために、アーキテクチャ同様、解析モデルにも階層的な概念を導入する。クラスタ間の通信は、リモートのグローバル・メモリのアクセス、グローバル・キャッシュのコヒーレンス制御、MLI 保証のためのキャッシュ制御によって引き起こされる。今、これらのリクエストがクラスタ内のいずれかのプロセッサから発せられたのではなく、一つの仮想的なプロセッサから発せられたと考えると、クラスタを単位と考えた SMCI モデルが適用可能である。

以下に、本章で提案する階層型 SMCI モデル構築の基本方針を示す。まず、クラスタ内はプロセッサを単位とした SMCI モデルを適用し、同時にクラスタ外へのアクセス（グローバル・キャッシュやローカルなグローバル・メモリ, リモートのグローバル・メモリのアクセス）や、クラスタ外からのアクセス（MLI 保証のためのキャッシュ制御命令）を、それぞれプロセッサの一つの状態として定義し、SMCI モデルを拡張する。これを **Processor モデル**（以後 P モデルと略記する。）と呼ぶ。次に、クラスタを単位とした別の SMCI モデルを構築する。クラスタ内外からのグローバル・メモリやグローバル・キャッシュに対するリクエスト状態を、P モデルにおけるプロセッサのリクエスト状態に、クラスタ内外のリクエストがない状態を、同じくプロセッサの計算状態に、リクエスト待ち状態は同じくリクエスト待ち状態に対応させることで、クラスタを単位とした SMCI モデルの拡張を図る。これを **Cluster モデル**（以後 C モデルと略記する。）と呼ぶ。このようにして拡張構築された二対の SMCI モデルが、本章で提案する階層型 SMCI モデルとなる。

SMCI モデルの構築に際しては、特定の共有ブロックの状態予測（そのブロックがダーティ状態／非ダーティ状態である確率）が必要であった。同様に、P モデルに対しては局所共有、C モデルに対しては広域共有ブロックの状態がどのようになっているかの予測が必要となる。本章では第4章同様、Wang らのモデル [68] により導き出される共有ブロックに対するキャッシュ・ミス率を利用することにより、上記局所／広域共有ブロックに対する状態予測を行う。

次に局所ブロックの状態予測を用いて、セミマルコフ過程を利用したプロセッサの極限確率と、その状態確率に基づいたクラスタ内でのプロセッサのリクエスト率との関係を定式化する。その上で、適当な初期状態（リクエスト率の初期値）からプロセッサの極限確率を計算し、それを基にリクエスト率を修正する。以下、この繰り返しを行ない、リクエスト率に関して収束点を求め、クラスタ単体におけるプロセッサの定常状態を求める。

この定常状態、すなわち P モデルは、グローバル・キャッシュやグローバル・メモリに対するリクエスト状態を定義する。これらプロセッサの諸状態と広域共有ブロックの状態予測を用いて、別のセミマルコフ過程を利用したクラスタ単位の極限確率と、その状態確率に基づいたクラスタ単位のリクエスト率との関係を定式化する。ここでも同様に、適当な初期状態（P モデルによって定まる、クラスタを単位としたリクエスト率）からクラスタの状態の極限確率を計算し、それを基にリクエスト率を計算し、以下この繰り返しを行い、リクエスト率に関して収束点を求め、クラスタを単位としたシステム全体の定常状態、すなわち C モデルを求める。

この時、通信遅延を含む広域データのサービス時間と、MLI 保証のためのローカル・キャッシュに対するキャッシュ制御命令の頻度が導き出されるので、この結果を P モデルに再帰的に適用し、以下、P モデルと C モデルを繰り返し計算し、C モデルより算出されるローカル・キャッシュに対するキャッシュ制御リクエスト率について収束点を求め、システム全体の定常状態を求める。

6.2.4 各種リクエスト率の定義

本章では、第4章同様、これまでの単純なプロセッサ・リクエスト率の概念で解析モデルを構築することが困難なため、4.2.3節で提案された 4 種類のリクエス

ト率を利用する。以下に、各種リクエスト率についての再記を行う。以後本節において、括弧内はCモデルに適用する場合に読み変えられたい。

- **ノーマル・リクエスト率**、$\varphi_{nor}^{(p)}$（$\varphi_{nor}^{(c)}$）、とは、プロセッサ（クラスタ）が計算状態（アイドル状態）の時に引き続いて発行される通常のデータ・リクエストの率である。

- **メモリ・リクエスト率**、$\varphi_{mem}^{(p)}$（$\varphi_{mem}^{(c)}$）、とは、ローカル・メモリ、グローバル・キャッシュ、グローバル・メモリ（グローバル・メモリ）中のデータに対して発行されるリクエスト率であり、ノーマル・リクエスト率とキャッシュ・ミス率の積に、その待ち状態からのリクエスト率を足した値で定義される。

- **コヒーレンス・リクエスト率**、$\varphi_{coh}^{(p)}$（$\varphi_{coh}^{(c)}$）、とは、ローカル・キャッシュ（グローバル・キャッシュ）の制御のためのリクエスト率およびその待ち状態からのリクエスト率の和で定義される。

- **ネットワーク・リクエスト率**、$\varphi_{net}^{(p)}$（$\varphi_{net}^{(c)}$）、とは、バス（階層型クロスバー・ネットワーク）に対して発行されるリクエスト率で、メモリ・リクエスト率とコヒーレンス・リクエスト率の和で定義される。

以上の各種リクエスト率は、6.3節で構築するモデルの内部変数として使用される。本章においては、それぞれのリクエスト率は、一台のプロセッサ（クラスタ）が発行する、ネットワーク・サイクル時間あたりの、リクエスト数である。

6.2.5 階層型SMCIモデル構築のための仮定

本節では、解析モデル構築を簡略化するために、次のような仮定を導入する。

(仮定1) 各プロセッサ（クラスタ）の挙動は同一の確率過程でモデル化され、それぞれ独立に動作する。

(仮定2) 各局所（広域）共有ブロックの状態は、同じ確率で表される。

(仮定3) 各プロセッサ（クラスタ）からの通常のデータ・リクエストは、独立した分布に従う。

(仮定 4) P モデルでは、プロセッサは与えられた時間パラメータによる幾何分布に従ってリクエストを発行する。C モデルでは、P モデルからの入力値を時間パラメータとする幾何分布に従ってリクエストを発行する。また、両モデルとも、リクエストの実行時間は、その種類ごとにアーキテクチャの仕様によって決定される一定分布に従う。

(仮定 5) P モデル（C モデル）における、各待ち状態での滞在時間分布は、一括して一つの幾何分布で近似する。

(仮定 6) プログラムは安定稼働している状態、すなわち、起動時のページングやキャッシュ・ヒット率の悪さは考慮しない状態を仮定する。またキャッシュは既に使い切った状態で、それに伴うキャッシュ・ブロックのリプレースについては考慮する。

(仮定 7) グローバル・キャッシュでインバリデーションやリプレースに起因するライトバックやフラッシュが発生した時、無条件にそのクラスタ内のローカル・キャッシュに対してインバリデーションを発行する。

6.3 階層型 SMCI モデル

階層型 SMCI モデルは、6.2.3節で述べたように、SMCI モデルの拡張である、P モデルと C モデルの二対によって構築される。仮想マシン全体の状態を反映させた階層型 SMCI モデルを完成させるためには、P および C モデルの相互作用を表現しなければならない。P モデルの場合、グローバル・メモリやグローバル・キャッシュに対してなされるべきリクエスト（グローバル・メモリにアドレス付けされている通常のデータ・アクセスや、MLI を保証するためのキャッシュ制御命令）、C モデルの場合、クラスタ内のローカル・キャッシュに対するキャッシュ制御命令がこれに相当する。

本節では、P および C モデル構築のための、SMCI モデル拡張部分の説明と、それらの相互作用の定式化についての説明を行う。

6.3.1 P モデル

6.3.1.1 状態定義と状態遷移

P モデル構築のために、まずプロセッサの状態定義を行なう。表 6.1は P モデルにおける、プロセッサのローカル・キャッシュ、ローカル・メモリ、バスに対する各状態を表している。この状態定義は Synapse プロトコルに沿って定義される。また、表 6.2 に同状態の集合を示す。これは本節における内部変数導出の際の表記を見やすくするための集合であり、モデル構築自体に必要なものではない。

表 6.1 P モデルの状態定義

COM_p	計算状態
Rh_p	ローカル・キャッシュからの読み込み
Wh_p	ローカル・キャッシュへの書き込み
HI_p	ローカル・キャッシュ・ヒットにより発生するインバリデーション
Rc_p	非ダーティな局所ブロックのキャッシュ・ミス・リード
Rd_p	ダーティな局所ブロックのキャッシュ・ミス・リード
Wc_p	非ダーティな局所ブロックのキャッシュ・ミス・ライト
MI_p	ローカル・キャッシュ・ミスにより発生するインバリデーション
Wd_p	ダーティな局所ブロックのキャッシュ・ミス・ライト
WB_p	クラスタ内外からのインバリデーションまたはグローバル・キャッシュでのリプレースによる、ローカル・キャッシュでのライトバック
RP_p	ローカル・キャッシュでのリプレースによるライトバック
Grw_p	広域ブロックへの、ローカル・キャッシュでのミスによる、アクセス
FL_p	ローカル・キャッシュでのリプレースによるフラッシュ、またはグローバル・キャッシュでのリプレースに起因するローカルでのフラッシュ
$\bar{i}$	状態 i に対する待ち状態。($i \in Q_{net}^{(p)}$)

図 6.2 は定義された状態間の遷移を示している。状態遷移は各種リクエスト(6.2.4節参照)が発行されたり完了したりする時に起きる。図中の変数の説明は

表 6.2 P モデルの状態の集合

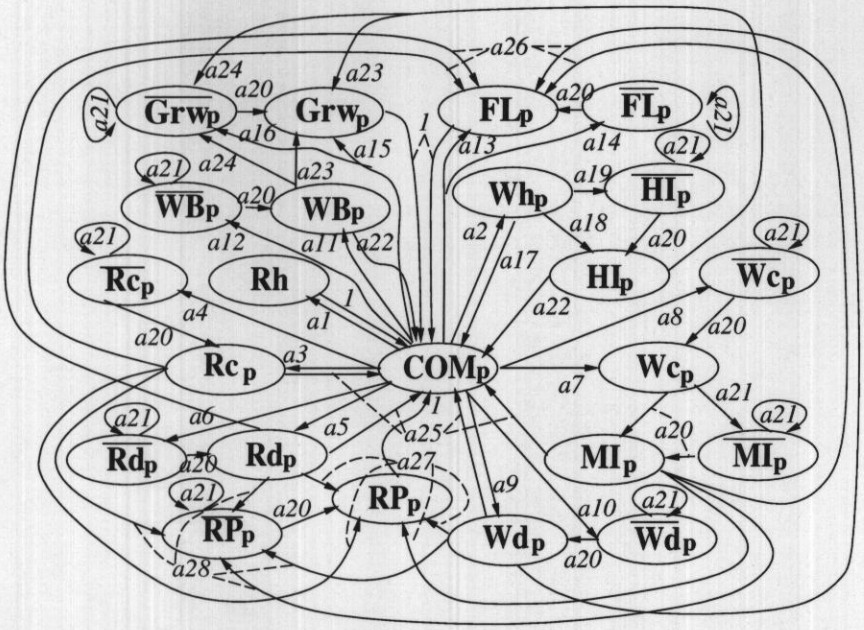
$Q^{(p)}$	P モデル全状態の集合
$Q^{(p)}_{mem}$	$= \{Rc_p, Rd_p, Wc_p, Wd_p, Grw_p\}$
$Q^{(p)}_{mem\_w}$	$= \{\bar{i} \mid i \in Q^{(p)}_{mem}\}$
$Q^{(p)}_{coh}$	$= \{HI_p, MI_p, WB_p, RP_p, FL_p\}$
$Q^{(p)}_{coh\_w}$	$= \{\bar{i} \mid i \in Q^{(p)}_{coh}\}$
$Q^{(p)}_{net}$	$= Q^{(p)}_{mem} \cup Q^{(p)}_{coh}$
$Q^{(p)}_{net\_w}$	$= \{\bar{i} \mid i \in Q^{(p)}_{net}\}$

表 6.4を参照されたい。

COM_pからGrw_p、$\overline{Grw_p}$への遷移は、1）COM_pから広域ブロックに対するローカル・キャッシュ・ミスが発生した場合と、2）ローカル・キャッシュにおいて広域ブロックのライトバックや、クリーン状態の広域ブロックに対するライト・ヒットが発生した場合である。本章では後者の場合、グローバル・キャッシュに対するライト・ヒットと解釈する。WB_pやHI_pからGrw_pや$\overline{Grw_p}$への遷移確率は、$1-L$で近似する。ただし、Lはリクエストされたデータが局所ブロックにある確率で、P モデルに対する入力パラメータの一つである。

FL_pへの遷移は、ローカル・キャッシュがインバリデーションやリプレース要求に従って、ブロックをフラッシュする場合である。グローバル・キャッシュからMLI 保証のためにローカル・キャッシュに対してインバリデーションやリプレース要求が発行された時、当該リクエストはクラスタ・ゲートウェイが発行しているのであって、プロセッサが発行しているのではないが、モデル簡略化のために、このリクエストはプロセッサによって発行されたものと考える。この時、バスの状態によっては$\overline{FL_p}$にも遷移しうる。なお、キャッシュ・ブロックのフラッシュを表す状態FL_pは、MLI の保証のための制御命令を表現するためにも必要である。

これ以外の遷移に関しては、4.3.2節を参照されたい。



$a1=Y(H_p(1-U)+H_sU)R$, $a2=Y(H_p(1-U)+H_sU)(1-R)$, $a3=Y\,A\,R(1-W)$, $a4=Y\,A\,R\,W$,
$a5=Y\,B\,R(1-W)$, $a6=Y\,B\,R(1-W)$, $a7=Y\,A(1-R)(1-W)$, $a8=Y\,A(1-R)W$,
$a9=Y\,B(1-R)(1-W)$, $a10=Y\,B(1-R)W$, $a11=(1-Y)M(1-W)$, $a12=(1-Y)M\,W$,
$a13=(1-Y)(1-M)(1-W)$, $a14=(1-Y)F\,W$, $a15=Y(1-H_s)U(1-L)(1-W)$,
$a16=Y(1-H_s)U(1-L)W$, $a17=1-C$, $a18=C(1-W)$, $a19=C\,W$, $a20=1-W$, $a21=W$,
$a22=L$, $a23=(1-L)(1-W)$, $a24=(1-L)W$, $a25=1-X$, $a26=(1-M)X$, $a27=M\,X(1-W)$,
$a28=M\,X\,W$ where $A=(1-H_p)(1-U)+(1-H_s)U\,L(1-D)$, $B=(1-H_s)U\,L\,D$

図 6.2 P モデルで使われる状態遷移

6.3.1.2 滞在時間分布

以上のように定義されたプロセッサの各状態では、その滞在時間が状態によって異なる。ここで各状態に対する滞在時間分布を与える。

まず、状態COM_pからは、厳密には次に遷移する状態（Rh_p, Rc_p, $\overline{Rc_p}$, ..., FL_p, $\overline{FL_p}$）に従ってそれぞれ分布関数を与えるべきであるが、そのような関数を導き出すのは困難であるため、データ・リクエストに起因する遷移の場合はパラメータ$\lambda_{nor}^{(p)}$の、キャッシュ制御に起因する遷移の場合はパラメータ$\lambda_{coh}^{(p)}$の幾何分布に従うものとする。ここで、$\lambda_{nor}^{(p)}$は与えられたプログラム・モデルによって決まるものであり、$\lambda_{coh}^{(p)}$はMLI保証のためのキャッシュ制御命令を含むため、C モデルとの相互作用で決まるものであることに注意されたい。

次に、ローカル・キャッシュやローカル・メモリのアクセスを行なう状態 i（$i \in \{Rh_p, Rc_p, Rd_p, Wh_p, HI_p, Wc_p, MI_p, Wd_p, RP_p, WB_p, FL_p\}$）から次の状態 j への遷移は、下記の一定分布に従う。

$$F_{ij}(t) = \begin{cases} 1 & \text{if } t \geq \text{状態 i でのサービス時間} \\ 0 & \text{otherwise} \end{cases}$$

それぞれのサービス時間は、対象とする仮想マシンの具体的な仕様によって与えられる。

グローバル・キャッシュやグローバル・メモリのアクセスを行う状態 Grw_p の滞在時間は、平均 $\eta^{(p)}_{Grw_p}$（C モデルから導出される）の幾何分布に従うものとする。また、待ち状態から次の状態への遷移は平均 WT の幾何分布に従うものとする。

ただし、後で述べるように WT は初期値 1 から出発し、プロセッサの状態の極限確率から逐次的に計算される。$\eta^{(p)}_{Grw_p}$ も同様に初期値 1 から出発し、C モデルにおけるクラスタの状態の極限確率から逐次的に計算される。

6.3.1.3 セミマルコフ過程の構築

先に述べた状態空間と遷移確率により定義される確率過程 $X = \{X_n\}_{n=0}^{\infty}$ は、クラスタ内プロセッサの状態を表現する。6.2.1節で示した本仮想マシンの定義と、6.2.5節で与えた仮定から明らかなように、この確率過程はマルコフ連鎖である。

また、X の各状態の滞在時間は、前節で与えた時間分布関数に従うので、その各滞在時間を $T = \{T_n\}_{n=0}^{\infty}$ で与えられる確率変数列と考えると、確率過程 $(X, T) = \{X_n, T_n\}_{n=0}^{\infty}$ は SMP（Semi-Markov Process）を形成する。

(X, T) の EMC（Embedded Markov Chain）である X は、図 6.2からも明らかなように、エルゴード的であるので、その定常分布を求めることができる。よって、EMC の定常分布 $\{\pi_i^{(p)}\}$ と、各状態の平均滞在時間 $\{\eta_i^{(p)}\}$ を式 (3.2) に適用して、(X, T) の極限確率を求めることができる。

6.3.1.4 極限確率とリクエスト率

P モデルの極限確率を計算するには、対象とするクラスタのアーキテクチャ・モデルと並列プログラム・モデルを表した入力パラメータが必要である。この入

力パラメータは、表6.3で示されている。与えられた入力パラメータとリクエスト率（初回は入力パラメータとして、2回目以後は極限確率から導出）からPモデルの遷移確率に使われる内部変数を導出し、それによって新しい極限確率を求める。

表 6.3 P モデルで使用される入力パラメータ

N_0	クラスタ中のプロセッサ数
H_p	局所プライベート・ブロックに対するローカル・キャッシュのヒット率
U	局所ブロック参照に対する局所共有ブロック参照の割合
R	局所ブロック参照に対するリードの割合
L	リクエストされたデータがローカル・メモリにアドレス付けされている確率
E	並列プログラム中の局所共有ブロックの総数
M	リプレース対象のブロックが既にモディファイされている確率
$\eta_i^{(p)}$	P モデルの SMP の状態 i における滞在時間（COM_p, Grw_pと待ち状態を除く）
$\lambda_{nor}^{(p)}$	COM_p状態からのデータ・リクエストに起因する遷移に対する滞在時間分布のパラメータ

内部変数（表6.4参照）のうち H_s とΨは6.2.3節で述べたように、Wang らのモデル [68] から導き出される。付録A.1を参照されたい。Dと C は4.3.5節と同様に導出される。

$$C = (1-U)U_{MD} + UR\left(1-\left(1-\frac{\Psi R}{E}\right)^{N_0-1}\right) \tag{6.1}$$

$$D = \begin{pmatrix} N_0-1 \\ 1 \end{pmatrix}\left(\frac{\Psi(1-R)}{E}\right)\left(1-\frac{\Psi(1-R)}{E}\right)^{N_0-2} \tag{6.2}$$

また、U_{MD}も4.3.5節の u_{md}と同様に、$U_{MD} = 1-(1-H_p(M+R-1))/((1-R)H_p)$ で与えられる。

表 6.4 P モデルで使用される内部変数

H_s	局所共有ブロックに対する、ローカル・キャッシュのヒット率
Ψ	あるローカル・キャッシュ に保持されている局所共有ブロック数
D	リクエストされた局所共有ブロックがダーティである確率
C	リクエストされた局所ブロックがダーティでない確率
X	ローカル・キャッシュ が既に一杯である確率
Inv_p	ローカル・キャッシュによるインバリデーション発行率
Inv_m	MLI 保証のためのローカル・キャッシュに対するインバリデーション発行率
Inv_arv	あるプロセッサがインバリデーション命令を受信する確率
U_{MD}	局所プライベート・ブロックがライトヒットした時にまだモディファイされていない確率
W	クラスタ内のバスにリクエストを出した時、待ち状態に陥る確率
Y	COM_pから発するリクエストに対して、それがデータリクエストである確率
WT	平均リクエスト待ち時間
$\lambda_{coh}^{(p)}$	COM状態からキャッシュ制御リクエストに起因する遷移を行なう際の滞在時間分布パラメータ
$\varphi_{nor}^{(p)}$	ノーマル・リクエスト率
$\varphi_{mem}^{(p)}$	メモリ・リクエスト率
$\varphi_{coh}^{(p)}$	コヒーレンス・リクエスト率
$\varphi_{net}^{(p)}$	ネットワーク・リクエスト率
$\{\pi_i^{(p)}\}$	P モデルの SMP における EMS の定常分布
$\{P_i^{(p)}\}$	P モデルの SMP の極限確率

以上の内部変数に関しては入力パラメータから直接計算できるが、残りに関しては極限確率やリクエスト率を必要とするので、実際の計算の際にはこれらに適当な初期値（6.3.3節参照）を与えて逐次的に計算を行なう。

ローカル・キャッシュが既に一杯である確率 X に関しては、4.3.6節と同様に導出されるが、インバリデーション命令は同一クラスタ内の他のキャッシュから来る以外にも、グローバル・キャッシュからも MLI 保証のために来ることに注意されたい。

ローカル・キャッシュに空きができるのは、直前のキャッシュ・ミスから、今回のキャッシュ・ミスが発生するまでの期間、少なくとも一度はインバリデーション命令を受けた場合と考える。ローカル・キャッシュ・ミスは、各サイクルあたり

$$Cache\_miss = ((1-U)(1-H_p) + U(1-H_s))\varphi_{nor}^{(p)} \tag{6.3}$$

の確率で発生すると考えて、この逆数を二回のキャッシュ・ミスの平均間隔時間と見なす。

Inv_p は、広域および局所共有ブロックへのライトのリクエストが発行された際に、そのブロックが非ダーティ状態である場合、もしくは、広域および局所共有ブロックへのリードもしくはライトのリクエストがローカル・キャッシュ・ミスに終り、そのブロックがダーティ状態にある場合の確率である。よって、あるプロセッサがインバリデーションを発行する率 Inv_pは、

$$\begin{aligned} Inv_p &= ((1-R)(C(H_p(1-U) + H_sU) \\ &+ (1-H_p)(1-U) + (1-H_s)U(1-D)) \\ &+ U(1-H_s)D)\varphi_{nor}^{(p)} \end{aligned} \tag{6.4}$$

となる。ただし、式 (6.4) の Cと Dは局所ブロックに対する確率であるため、式 (6.4) は近似である。

同一クラスタ内のグローバル・キャッシュが MLI 保証のためにインバリデーション命令を発行する率を Inv_mとすると、あるプロセッサがあるサイクルにおいて、自分以外の $N_0 - 1$ のプロセッサ、もしくは、グローバル・キャッシュの発行するインバリデーションを少なくとも一つ受けとる確率 Inv_arv は、

$$Inv\_arv = 1 - \left(1 - \frac{\alpha_p(E, N_0) Inv_p}{N_0 - 1}\right)^{N_0 - 1} \left(1 - \frac{\alpha_p(E, N_0) Inv_m}{N_0 - 1}\right) \tag{6.5}$$

となる。

ただし、$\alpha_p(E, N_0)$ も 4.3.6節同様、共有ブロックの数とプロセッサ数を引数として、一回のインバリデーション命令あたりの、無効化される平均ブロック数を返す関数である。Inv_mの導出は、式 (6.41) で示されている。

よって、ローカル・キャッシュ・ミスが発生した時に、ローカル・キャッシュに空きがない確率 Xは、以下のように表される。

$$X = (1 - Inv\_arv)^{\frac{1}{Cache\_miss}} \tag{6.6}$$

COM_p状態からの、キャッシュ制御リクエスト発行による遷移に関する時間分布パラメータ$\lambda_{coh}^{(p)}$は、インバリデーション受信率 Inv_arvを使って、次のように定義される。

$$\lambda_{coh}^{(p)} = \frac{1}{Inv\_arv} \tag{6.7}$$

クラスタ内のバスにリクエストを出した時、待ち状態に陥る確率 Wの導出も、4.3.6節と同様である。まず、あるプロセッサから見てネットワーク・ビジーである確率 $BUSY$は、

$$BUSY = \begin{pmatrix} N_0 - 1 \\ 1 \end{pmatrix} \left(\sum_{i \in Q_{net}^{(p)}} P_i^{(p)} \frac{\eta_i^{(p)} - 1}{\eta_i^{(p)}}\right) \left(1 - \sum_{i \in Q_{net}^{(p)}} P_i^{(p)} \frac{\eta_i^{(p)} - 1}{\eta_i^{(p)}}\right)^{N_0 - 2} \tag{6.8}$$

となり、他のプロセッサからのリクエストとの競合に勝つ確率 WINは、

$$WIN = \left(1 - (1 - \varphi_{net}^{(p)})^{N_0}\right) \frac{1}{N_0 \varphi_{net}^{(p)}} \tag{6.9}$$

で与えられるので、待ち状態に陥る確率 wは次式で表される。

$$W = BUSY + (1 - BUSY)(1 - WIN) \tag{6.10}$$

ノーマル・リクエスト率$\varphi_{nor}^{(p)}$、メモリ・リクエスト率$\varphi_{mem}^{(p)}$、コヒーレンス・リクエスト率$\varphi_{coh}^{(p)}$、ネットワーク・リクエスト率$\varphi_{net}^{(p)}$も 4.3.6節と同様に、以下のように表される。

$$\begin{aligned}
\varphi_{nor}^{(p)} &= \frac{1}{\lambda_{nor}^{(p)}} && (6.11)\\
\varphi_{mem}^{(p)} &= ((1-U)(1-H_p)+U(1-H_s)\\
&+ C(1-R)(H_p(1-U)+H_sU))\varphi_{nor}^{(p)} && (6.12)\\
&+ BUSY \sum_{i\in Q_{mem_w}^{(p)}} \frac{P_i^{(p)}}{\eta_i^{(p)}} + (1-BUSY) \sum_{i\in Q_{mem_w}^{(p)}} P_i^{(p)}\\
\varphi_{coh}^{(p)} &= Inv_p + Inv_m + M \times Inv_arv + X \times M \times Cache_Miss\\
&+ BUSY \sum_{i\in Q_{coh_w}^{(p)}} \frac{P_i^{(p)}}{\eta_i^{(p)}} + (1-BUSY) \sum_{i\in Q_{coh_w}^{(p)}} P_i^{(p)} && (6.13)\\
\varphi_{net}^{(p)} &= \varphi_{mem}^{(p)} + \varphi_{coh}^{(p)} && (6.14)
\end{aligned}$$

状態COM_pから発するリクエストに対して、それがデータ・リクエストである割合Yは、

$$Y = \frac{\frac{1}{\lambda_{nor}^{(p)}}}{1-\left(1-\frac{1}{\lambda_{nor}^{(p)}}\right)\left(1-\frac{1}{\lambda_{coh}^{(p)}}\right)} \tag{6.15}$$

で、待ち状態の平均滞在時間WTは

$$WT = \frac{\sum_{i\in Q_{net\_w}^{(p)}} P_i^{(p)}}{\sum_{i\in Q_{net\_w}^{(p)}} \frac{P_i^{(p)}}{\eta_i^{(p)}}} \tag{6.16}$$

で、それぞれ導出される。

状態COM_pの平均滞在時間は、式 (3.1) を適用して、

$$\eta_{COM_p} = Y\lambda_{nor}^{(p)} + (1-Y)\lambda_{coh}^{(p)} \tag{6.17}$$

で与えられる。

6.3.2 C モデル

P モデルでは、プロセッサの状態遷移を、ローカル・キャッシュの動きに注目してモデル化したが、C モデルでは、プロセッサの代わりにクラスタ・ゲートウェイの状態遷移を、グローバル・キャッシュの動きに注目してモデル化する。従って、P モデルでの計算状態の代わりに、C モデルではリクエストがない状態、アイドル状態、を状態遷移の中心に考える。なお、C モデルでの階層型クロスバー・ネットワークの階層度は k とする。よって、クラスタの数 N は、$N_1 * N_2 * \cdots * N_k$ となる。

6.3.2.1 状態定義と状態遷移

C モデルの状態定義も、P モデル同様に行なわれる。表 6.5は C モデルにおける、クラスタ・ゲートウェイのグローバル・キャッシュ、グローバル・メモリ、階層型クロスバー・ネットワークに対する各状態を表している。この状態定義も P モデル同様、Synapse プロトコルに沿って定義されている。また、表 6.6に同状態の集合を示す。これは本節における内部変数導出の際の表記を見やすくするための集合であり、モデル構築自体に必要なものではない。

図 6.3は定義された状態間の遷移を示している。状態遷移は各種リクエストが発行されたり完了したりする時に起きる。

プロセッサからデータ・リクエストが発生した時、確率 $1 - L$ でそれは広域ブロックに対するものであり、さらに確率 $1 - H_s$ で、そのリクエストは、クラスタ・ゲートウェイに到達する。このリエストの行き先は、付録 A.2 で考察するように、S_i の確率で C_i グループ内にある。よって、$L + S_0 + S_1 + \cdots + S_k = 1$ が成り立つ。ただし、S_0 はリクエスト先がローカルなグローバル・メモリであるため、階層型クロスバー・ネットワークを使用しない。

C モデルでのデータ・リクエストは、プロセッサからの広域データへのローカル・キャッシュのキャッシュ・ミスによる通常ものと、ローカル・キャッシュでのライトバックやクリーン・ブロックに対するライト・ヒットを、グローバル・キャッシュへのライト・ヒットと解釈したものの 2 種類からなる。後者は厳密にはグローバル・キャッシュに対する MLI 保証のための制御命令であるが、モデル構築を簡単にするため、このように仮定する。

表 6.5 C モデルの状態定義

$Idle_c$	アイドル状態
Rh_c	グローバル・キャッシュからの読み込み
Wh_c	グローバル・キャッシュへの書き込み／ローカル・キャッシュのライトバックに伴う書き込み
HI_c	グローバル・キャッシュ・ヒットにより発生するインバリデーション
Rc_c	非ダーティな広域ブロックのキャッシュ・ミス・リード
Rd_c	ダーティな広域ブロックのキャッシュ・ミス・リード
Wc_c	非ダーティな広域ブロックのキャッシュ・ミス・ライト
MI_c	グローバル・キャッシュ・ミスにより発生するインバリデーション
Wd_c	ダーティな広域ブロックのキャッシュ・ミス・ライト
WB_c	ダーティな広域ブロックへの読み書きにより発生するライトバック
RP_c	グローバル・キャッシュでのリプレースによるライトバック
FL_c	グローバル・キャッシュでのリプレースによるフラッシュ
LFL_c	MLI 保証のためのローカル・キャッシュへのフラッシュ要求
LWB_c	MLI 保証のためのローカル・キャッシュへのライトバック要求
$\bar{i}$	状態 i に対する待ち状態。($i \in Q_{net}^{(c)}$)

グローバル・キャッシュでは、リプレースやインバリデーションの要求に対して、当該ブロックがモディファイされている場合、ライトバックを行なうだけであるが、そうではない場合、フラッシュが行なわれるだけではなく、MLI 保証のためにローカル・キャッシュに対してもキャッシュ制御命令が発行される。当該ブロックが、ローカル・キャッシュにおいてダーティの場合（確率 b）、ローカル・キャッシュに対するライトバック要求（LWB_c）を、ダーティでない場合（確率 $1-b$）、フラッシュ要求（LFL_c）を行なう。これ以外の状態遷移は P モデルと同様である。

表 6.6 C モデルの状態定義

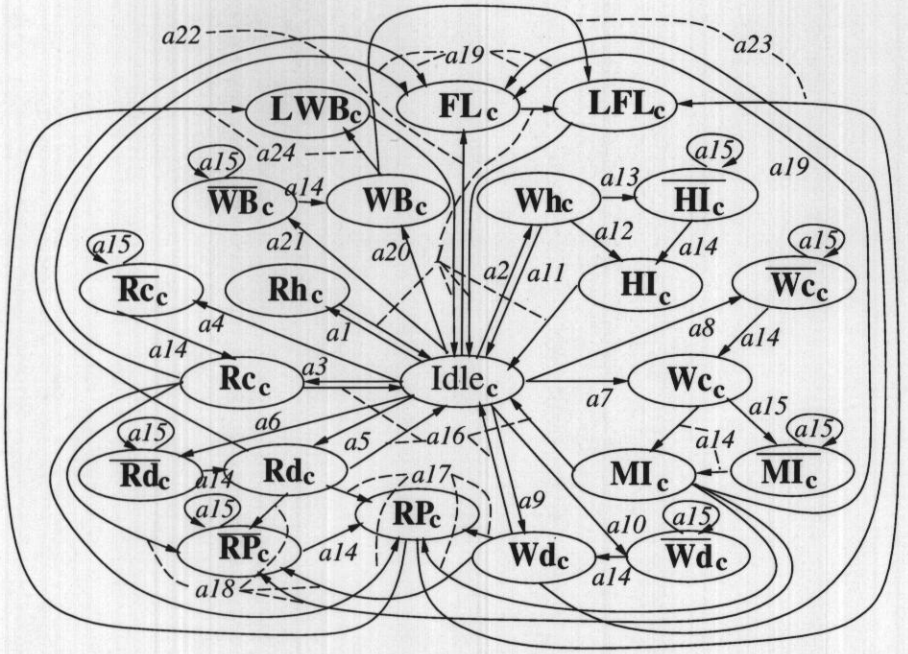
$Q^{(c)}$	C モデル全状態の集合
$Q^{(c)}_{mem}$	$= \{Rc_c, Rd_c, Wc_c, Wd_c\}$
$Q^{(c)}_{mem\_w}$	$= \{\bar{i} \mid i \in Q^{(c)}_{mem}\}$
$Q^{(c)}_{coh}$	$= \{HI_c, MI_c, WB_c, RP_c\}$
$Q^{(c)}_{coh\_w}$	$= \{\bar{i} \mid i \in Q^{(c)}_{coh}\}$
$Q^{(c)}_{net}$	$= Q^{(c)}_{mem} \cup Q^{(c)}_{coh}$
$Q^{(c)}_{net\_w}$	$= \{\bar{i} \mid i \in Q^{(c)}_{net}\}$

6.3.2.2 滞在時間分布

$Idle_c$の滞在時間分布に関しては、P モデル同様、データ・リクエストに起因する遷移の場合はパラメータ$\lambda^{(c)}_{nor}$の、キャッシュ制御に起因する遷移の場合はパラメータ$\lambda^{(c)}_{coh}$の幾何分布に従うものとする。

次に、グローバル・キャッシュやグローバル・メモリのアクセスを行なう状態の滞在時間分布は、リクエスト先がどのC_iグループに属するかによって異なるため、正確にモデル化を行なうにはリクエストの種類（階層型クロスバー・ネットワークの階層度）に応じた状態を定義しなければならない。これは、プロセッサ数やクラスタ数に対して増えるのではないため、状態数の爆発的増大というマルコフ連鎖を用いた並列計算機の解析モデルにおける致命的な欠点には直結しないが、本章では簡単のために各リクエストに対する滞在時間の平均値によって近似する。

すなわち、グローバル・メモリやグローバル・キャッシュのサービス時間（ネットワーク遅延を除く）を Λ_j（$j \in Q^{(c)}_{net}$）、C_i グループの $N_i + 1$ 入力 $N_i + 1$ 出力クロスバーの遅延時間をΔ_i、全体のアービトレーションに要する時間をΓ とすると、グローバル・キャッシュやグローバル・メモリのアクセスを行なう状態 j（$j \in Q^{(c)}_{net}$）から次の状態 kへの滞在時間分布 F_{jk}は、P モデル同様、次式で与えられる一定分布を用いる。



$a1=y\,A\,r,\ a2=y\,A\,(1-r),\ a3=y\,B\,r(1-w),\ a4=y\,B\,r\,w,\ a5=y(1-h_s)u\,r\,d(1-w),$
$a6=y(1-h_s)u\,r\,d\,w,\ a7=y\,B(1-r)(1-w),\ a8=y\,B(1-r)w,\ a9=y(1-h_s)u(1-r)d(1-w),$
$a10=y(1-h_s)u(1-r)d\,w,\ a11=1-c,\ a12=c(1-w),\ a13=c\,w,\ a14=1-w,\ a15=w,$
$a16=1-x,\ a17=m\,x(1-w),\ a18=m\,x\,w,\ a19=(1-m)x,\ a20=(1-y)m(1-w),$
$a21=(1-y)m\,w,\ a22=(1-y)(1-m),\ a23=1-b,\ a24=b$
$\text{where } A=(h_p(1-u)+h_s u),\ B=((1-h_p)(1-u)+(1-h_s)(1-d)u)$

図 6.3 C モデルで使われる状態遷移

$$F_{jk}(t) = \begin{cases} 1 & \text{if } t \geq \text{状態 i でのサービス時間} \\ 0 & \text{otherwise} \end{cases}$$

状態 i でのサービス時間 $Serv(i)$ は、次式で与えられる。

$$Serv(i) = \sum_{j=1}^{k} \frac{S_j}{\sum_{l=1}^{k} S_l} \left(\Delta_j + 2 \sum_{q=1}^{j-1} \Delta_q \right) + \Gamma + \Lambda_i \tag{6.18}$$

ただし、インバリデーション・リクエストに関しては、グローバル・キャッシュのキャッシュ・コヒーレンス制御がフルマップ・ディレクトリ方式であるため、以下のような近似を行なう。

インバリデーションが発生した時、ディレクトリに示される当該ブロックを持つ他のグローバル・キャッシュに対して、インバリデーション・リクエストが発行されるが、この時当該ブロックを持つグローバル・キャッシュの数、$\alpha_c(e,N)$、だけリクエストが繰り返し発行される。この様子を、一回あたりのインバリデーション状態（HI_c、MI_c）と、その待ち状態（$\overline{HI_c}$、$\overline{MI_c}$）の滞在時間を$\alpha_c(e,N)$倍したもので近似する。

また、インバリデーション状態以外の待ち状態から次の状態への遷移は平均 wt の幾何分布に従う。ただし、$\overline{HI_c}$、$\overline{MI_c}$に関しては、平均$\alpha_c(e,N) * wt$ の幾何分布に従う。ただし、$\alpha_c(e,N)$ も P モデルでの$\alpha_p(E,N_0)$ 同様、共有ブロックの数とクラスタ数を引数として、一回のインバリデーション命令あたりの、無効化される平均ブロック数を返す関数であるが、対象とする共有ブロックは、広域に限る点が異なる。

LFL_cと LWB_cは、MLI 保証のためにローカル・キャッシュに対してコヒーレンス制御命令を実行するための状態である。まず、LFL_cに関しては、先に述べたように、ローカル・キャッシュに対するフラッシュ要求は、実際はグローバル・キャッシュが行なうが、P モデルにおいてグローバル・キャッシュとバスとの間の直接の関係を定式化するのは困難なため、この要求はプロセッサが行なったものと解釈し、待ち時間を含めたこの要求の処理時間を LFL_cの平均滞在時間と解釈する。LWB_cも同様に、P モデルにおける WB_pと$\overline{WB_p}$の滞在時間の和を、LWB_cの平均滞在時間と考える。これらの分布はそれぞれの平均滞在時間をパラメータとした幾何分布と考える。

このように定義された状態と状態遷移、滞在時間分布により導き出される確率過程は、P モデル同様、SMP を形成する。

6.3.2.3 極限確率とリクエスト率

P モデル同様、C モデルの内部変数（表 6.8参照）も、モデルに対する入力パラメータ（表 6.7参照）と、他の内部変数から導出される。

P モデル同様、内部変数（表 6.8参照）のうち h_sとψは 6.2.3節で述べたように、Wang らのモデル [68] から導き出される。これらを用いて、d と c は、次のように表される。

表 6.7 C モデルで使用される入力パラメータ

N	クラスタの数
N_i	C_iグループに属するC_{i-1}グループの数
h_p	クラスタ単位の、広域プライベート・ブロックに対する、 グローバル・キャッシュのヒット率
u	広域ブロック参照に対する広域共有ブロック参照の割合
r	広域ブロック参照に対する リードの割合
S_i	広域ブロックに対してリクエストが発行された時、 そのブロックがC_iグループにある確率
e	並列プログラム中の広域共有ブロックの総数
m	リプレース対象のブロックが既にモディファイされている確率
$\eta_i^{(c)}$	C モデルの SMP の状態 i における滞在時間 ($Idle_c$、LWB_c、LFL_cと待ち状態を除く)
$\lambda_{nor}^{(c)}$	$Idle_c$からの広域データ・リクエストに起因する遷移に 対する滞在時間分布パラメータ
Γ	階層型クロスバー・ネットワーク 全体での調停に要する平均時間
Δ_i	C_iグループの N_i+1 入出力クロスバの遅延時間
Λ_j	状態 $j \in Q_{net}^{(c)}$での上記遅延を除いたサービス時間

$$c = (1-u)u_{md} + ur\left(1-\left(1-\frac{\psi r}{e}\right)^{N-1}\right) \tag{6.19}$$

$$d = \binom{N-1}{1}\left(\frac{\psi(1-r)}{e}\right)\left(1-\frac{\psi(1-r)}{e}\right)^{N-2} \tag{6.20}$$

また、u_{md}も 6.3.1.4節のU_{MD}と同様に、$u_{md} = 1-(1-h_p(m+r-1))/((1-r)h_p)$で与えられる。

特定の広域ブロックがローカル・キャッシュでダーティである確率 b の導出は、P モデルにおける d の導出と同じように考える。

表 6.8 C モデルで使用される内部変数

h_s	クラスタ単位の、広域共有ブロックに対する、グローバル・キャッシュのヒット率
ψ	あるグローバル・キャッシュに保持されている広域共有ブロック数
d	リクエストされた広域共有ブロックがダーティである確率
c	リクエストされた広域ブロックがダーティでない確率
u_{md}	広域プライベート・ブロックがライトヒットした時にそれがまだモディファイされていない確率
x	グローバル・キャッシュ が 既に一杯である確率
inv	グローバル・キャッシュによるインバリデーション発行率
inv_arv	グローバル・キャッシュでのインバリデーション受信率
b	特定の広域ブロックがローカル・キャッシュでダーティである確率
w	階層型クロスバー・ネットワークにリクエストを出した時、待ち状態に陥る確率
y	$Idle_c$から発するリクエストに対して、それがデータリクエストである確率
wt	平均リクエスト待ち時間
$\lambda_{coh}^{(c)}$	$Idle_c$からキャッシュ制御リクエストに起因する遷移に対する滞在時間分布パラメータ
$\varphi_{nor}^{(c)}$	ノーマル・リクエスト率
$\varphi_{mem}^{(c)}$	メモリ・リクエスト率
$\varphi_{coh}^{(c)}$	コヒーレンス・リクエスト率
$\varphi_{net}^{(c)}$	ネットワーク・リクエスト率
$\{\pi_i^{(c)}\}$	C モデルの SMP における EMS の定常分布
$\{P_i^{(c)}\}$	C モデルの SMP の極限確率

$$b = \begin{pmatrix} N_0 \\ 1 \end{pmatrix} \left(\frac{\Psi(1-R)}{E}\right) \left(1 - \frac{\Psi(1-R)}{E}\right)^{N_0-1} \tag{6.21}$$

グローバル・キャッシュが既に一杯である確率 x に関しては、P モデルでの X と同じように導出する。

グローバル・キャッシュに空きができるのは、直前のキャッシュ・ミスから、今回のキャッシュ・ミスが発生するまでの期間、少なくとも一度はインバリデーション命令を受けた場合と考える。グローバル・キャッシュ・ミスは、各サイクルあたり

$$cache\_miss = ((1-u)(1-h_p) + u(1-h_s))\varphi_{nor}^{(c)} \tag{6.22}$$

の確率で発生すると考えて、この逆数を二回のキャッシュ・ミスの平均間隔時間と見なす。

inv は、広域共有ブロックへのライトのリクエストが発行された際に、そのブロックが非ダーティ状態である場合、もしくは、広域共有ブロックへのリードもしくはライトのリクエストがグローバル・キャッシュ・ミスに終り、そのブロックがダーティ状態にある場合の確率である。よって、あるクラスタがインバリデーションを発行する率 invは、

$$\begin{aligned} inv &= ((1-r)(c(h_p(1-u) + h_s u) \\ &+ (1-h_p)(1-u) + (1-h_s)u(1-d)) \\ &+ u(1-h_s)d)\varphi_{nor}^{(c)} \end{aligned} \tag{6.23}$$

となる。

この invを使って、あるクラスタがあるサイクルにおいて、自分以外の $N-1$ のクラスタの発行するインバリデーションを、少なくとも一つ受けとる確率 inv_arv は、

$$inv\_arv = 1 - \left(1 - \frac{\alpha_c(e,N)inv}{N-1}\right)^{N-1} \tag{6.24}$$

となる。

よって、グローバル・キャッシュ・ミスが発生した時に、キャッシュに空きがない確率 x は、以下のように表される。

$$x = (1 - inv\_arv)^{\frac{1}{cache\_miss}} \tag{6.25}$$

inv_arvを使って、クラスタの $Idle_c$状態からキャッシュ制御リクエストによる遷移に関する時間分布パラメータ$\lambda_{coh}^{(c)}$が、次のように導出される。

$$\lambda_{coh}^{(c)} = \frac{1}{inv\_arv} \tag{6.26}$$

階層型クロスバー・ネットワークにリクエストを出した時、待ち状態に陥る確率 wは、クエスト先のクラスタがビジー状態である確率 $busy$と、階層型クロスバー・ネットワークでの競合に勝つ確率 win により計算される。

$$w = busy + (1 - busy)(1 - win) \tag{6.27}$$

$busy$と win は P モデルの $BUSY$、WINの導出と大きく異なるので、詳細な説明を要する。

あるクラスタ A が、自分から見て、C_iグループに属するクラスタ B にリクエストを出そうとした時、A 以外の $N-1$ 個のクラスタのうちの一つが、B と通信している確率は、次のようにして求める。

あるクラスタがある瞬間において任意のクラスタと通信しており、次のサイクルもその通信が終了しない確率 $\mathcal{P}$は、C モデルの SMP の極限確率を使って次のように表される。

$$\mathcal{P} = \sum_{i \in Q_{net}^{(c)}} P_i^{(c)} \frac{\eta_i^{(c)} - 1}{\eta_i^{(c)}} \tag{6.28}$$

クラスタ B から見て、C_jグループに属するが、それより下位のグループには属さない、クラスタ A 以外のクラスタの数は

$$N_{C_j} = \begin{cases} (N_j - 1)N_{j-1}N_{j-2} \cdots N_1 - 1 & \text{if } j = i \\ (N_j - 1)N_{j-1}N_{j-2} \cdots N_1 & \text{otherwise} \end{cases} \tag{6.29}$$

である。よって、クラスタ B から見て、C_jグループに属するが、それより下位のグループには属さない、クラスタ A 以外の任意のクラスタから、クラスタ B がアクセスされている確率 $\mathcal{A}(j)$ は、次式で表せる。

$$\mathcal{A}(j) = \begin{pmatrix} N_{C_j} \\ 1 \end{pmatrix} \frac{1}{N_{C_j}} \frac{S_j}{\sum_{l=1}^{k} S_l} \mathcal{P} \left(1 - \frac{1}{N_{C_j}} \frac{S_j}{\sum_{l=1}^{k} S_l} \mathcal{P}\right)^{N_{C_j}-1} \tag{6.30}$$

従って、あるクラスタがC_iグループの特定のクラスタをアクセスしようとした時に、それがビジー状態である確率 $busy$は、次のように表される。

$$busy = \mathcal{A}(1) + \sum_{i=2}^{k} \mathcal{A}(i) \prod_{j=1}^{i-1} (1 - \mathcal{A}(j)) \tag{6.31}$$

次に、クラスタがリクエストを出して、ネットワーク競合に勝つ確率 win は、付録 A.2の式 (A.15) を使って、

$$win = \sum_{i=1}^{k} \frac{S_i}{\sum_{l=1}^{k} S_l} ARB(i) \tag{6.32}$$

で表される。

ノーマル・リクエスト率$\varphi_{nor}^{(c)}$は、定義により、$Idle_c$状態からのデータ・リクエストによる遷移に関する時間分布パラメータ$\lambda_{nor}^{(c)}$によって、

$$\varphi_{nor}^{(c)} = \frac{1}{\lambda_{nor}^{(c)}} \tag{6.33}$$

と表される。$\lambda_{nor}^{(c)}$は P モデルでの各種リクエスト率等を用いて、次のように導出される。

$$\lambda_{nor}^{(c)} = 1 - \Big(1 - \Big(((1 - H_s) + (1 - R)H_s C)\, \varphi_{nor}^{(p)} + Inv_p\, M\Big)\, U(1 - L)\Big)^{N_0} \tag{6.34}$$

ただし、$Inv_p U(1-L)M$は、広域ブロックに対するライトバックの発生率の近似である。

メモリ・リクエスト率$\varphi_{mem}^{(c)}$、コヒーレンス・リクエスト率$\varphi_{coh}^{(c)}$、ネットワーク・リクエスト率$\varphi_{net}^{(c)}$も P モデルと同様に、以下のように表される。

$$
\begin{aligned}
\varphi_{mem}^{(c)} &= ((1-u)(1-h_p)+u(1-h_s) \\
&+ c(1-r)(h_p(1-u)+h_s u))\varphi_{nor}^{(c)} \qquad (6.35) \\
&+ busy \sum_{i \in Q_{mem\_w}^{(c)}} \frac{P_i^{(c)}}{\eta_i^{(c)}} + (1-busy) \sum_{i \in Q_{mem\_w}^{(c)}} P_i^{(c)} \\
\varphi_{coh}^{(c)} &= inv + m \times inv\_arv + x \times m \times cache\_miss \qquad (6.36) \\
&+ busy \sum_{i \in Q_{coh\_w}^{(c)}} \frac{P_i^{(c)}}{\eta_i^{(c)}} + (1-busy) \sum_{i \in Q_{coh\_w}^{(c)}} P_i^{(c)} \\
\varphi_{net}^{(c)} &= \varphi_{mem}^{(c)} + \varphi_{coh}^{(c)} \qquad (6.37)
\end{aligned}
$$

待ち状態の平均滞在時間 wt も、P モデル同様、次のように表される。

$$
wt = \frac{\sum_{i \in Q_{net\_w}^{(c)}} P_i^{(c)}}{\sum_{i \in Q_{net\_w}^{(c)}} \frac{P_i^{(c)}}{\eta_i^{(c)}}} \qquad (6.38)
$$

状態 $Idle_c$から発するリクエストに対して、それがデータ・リクエストである割合 Yは、P モデルでの Y同様、

$$
y = \frac{\frac{1}{\lambda_{nor}^{(c)}}}{1-\left(1-\frac{1}{\lambda_{nor}^{(c)}}\right)\left(1-\frac{1}{\lambda_{coh}^{(c)}}\right)} \qquad (6.39)
$$

で表される。

状態 $Idle_c$の平均滞在時間は、式 (3.1) を適用して、

$$
\eta_{Idle_c} = y\lambda_{nor}^{(c)} + (1-y)\lambda_{coh}^{(c)} \qquad (6.40)
$$

で与えられる。

P モデルにおいて、あるプロセッサが MLI 保証のためのリプレース情報をグローバル・キャッシュから受けとる率は、次のように表される。

$$
Inv_m = y((1-h_p)(1-u)+(1-h_s)u)x\varphi_{nor}^{(c)} + (1-y)inv\_arv \qquad (6.41)
$$

式 (6.41) の第 1 項は、グローバル・キャッシュでのキャッシュ・ミスに関与するリプレースを、第 2 項は他のクラスタからのインバリデーション命令に関与するリプレースを表す。

最後に、残された平均滞在時間の導出を行なう。P モデルのプロセッサの状態 Grw_p の平均滞在時間は、C モデルでの通常のデータ・アクセスに要する時間であるから、次のように近似する。

$$\eta_{Grw_p}^{(p)} = \frac{\sum_{i \in Q_{net}^{(c)} \cup Q_{net\_w}^{(c)}} P_i^{(c)}}{\sum_{i \in Q_{net}^{(c)} \cup Q_{net\_w}^{(c)}} \frac{P_i^{(c)}}{\eta_i^{(c)}}} \tag{6.42}$$

同様に、C モデルの状態 LWB および LFL の平均滞在時間は、それぞれ P モデルにおけるライトバックおよびフラッシュに要する時間である。

$$\eta_{LWB_c}^{(c)} = \frac{\sum_{i \in \{WB_p, \overline{WB_p}\}} P_i^{(c)}}{\sum_{i \in \{WB_p, \overline{WB_p}\}} \frac{P_i^{(c)}}{\eta_i^{(c)}}} + W \times WT \tag{6.43}$$

$$\eta_{LFL_c}^{(c)} = \eta_{FL_p}^{(p)} + W \times WT \tag{6.44}$$

6.3.3 階層型 SMCI モデルの計算手順

階層型 SMCI モデルの実際の計算手順は、SMCI モデル同様、イタレーションによる逐次近似法を用いる。以下にその手順を示す。

1) P モデル、C モデルに対する入力パラメータ（表 6.4,6.8参照）を与える。

2) P モデル、C モデルの内部変数のうち、初期値を必要とするものに、表 6.9 で示されるような初期値を与える。

3) P モデルの SMP の極限確率を求める。

4) P モデルの内部変数の更新を行なう。

5) $\varphi_{net}^{(p)}$ について収束していなければ 3) に戻る。

6) C モデルの SMP の極限確率を求める。

7) C モデルの内部変数の更新を行なう。

8) $\varphi_{net}^{(c)}$について収束していなければ 6) に戻る。

9) Inv_mについて収束していなければ 3) に戻る。

表 6.9 階層型 SMCI モデルの内部変数に対する初期値例

P モデル内部変数の初期値例		C モデル内部変数の初期値例	
X	1	x	1
W	$1-(1-\varphi_{net}^{(p)})^{N_0}$	w	$1-(1-\varphi_{net}^{(p)})^{N}$
Y	1	y	1
Wt	1	wt	1
$\varphi_{mem}^{(p)}$	$(1-h_p)\varphi_{nor}^{(p)}$	$\varphi_{mem}^{(c)}$	$(1-H_p)\varphi_{nor}^{(c)}$
$\varphi_{coh}^{(p)}$	0	$\varphi_{coh}^{(c)}$	0
Inv_p	0	inv	0
Inv_m	0	η_{Idol_c}	$\lambda_{nor}^{(c)}$
η_{COM_p}	$\lambda_{nor}^{(p)}$	η_{LFL_c}	η_{FL_p}
η_{Grw_p}	η_{Rh_c}	η_{LWB_c}	η_{WB_p}

6.4 結論

スケーラブル共有メモリ型並列計算機に対するセミマルコフ過程を利用した解析モデルとして、階層型 SMCI モデルを提案した。本モデルは、第 4章で提案された共有メモリ型並列計算機に対する解析モデルである SMCI モデルを、スケーラブル共有メモリ型アーキテクチャの特質を生かして、階層的に統合構築したものである。

本モデル構築に先立ち、対象とするアーキテクチャ・モデルとして、階層的なメモリ・システム、階層的な相互結合網という特徴のある、仮想的かつ簡略化されたアーキテクチャを提案した。このアーキテクチャは実際のマシンとしては簡

単すぎるものではあるが、MLI の保証を行なう等、並列計算機アーキテクチャ研究の観点からも興味深いものであろう。さらに、第4章で提案した、各種リクエスト率を、本アーキテクチャ・モデルに対応するように拡張した。また、対象とするプログラム・モデルとして、システムの階層性を考慮に入れた、広域および局所に対応した、共有および非共有のデータ・ブロックの概念を提案した。

実際の構築にあたっては、第4章に準拠した形で入力パラメータを利用できるよう考慮しつつ、プロセッサとクラスタの各状態の定義と、それらの遷移確率の導出、および各状態間の滞在時間分布関数を定義し、本モデルを完成させた。また、本モデルを用いて、対象とするアーキテクチャの予測評価を計算する際の、内部変数の初期値例および計算手順も示した。

本モデルの妥当性の証明は、実際のシステムにおけるベンチマークや、シミュレーション結果との比較でなされるべきであり、今後の重要研究課題ではあるが、大規模なスケーラブル共有メモリ型並列計算機の開発に先だって、異なる仕様の相対的かつ大まかな予測性能比較は可能であり、これは本論文の主眼の一つでもある。

第 7 章

結論

本章では本研究の成果についてまとめ、今後の課題を明らかにする。

7.1 本研究の成果

本研究では、並列計算機の予測性能評価技術として、計算コストが安価であるにもかかわらず詳細な解析を可能とした解析モデルの理論体系を構築し、その妥当性を既存シミュレーション結果との比較により検証した。本研究で得られた成果を以下にまとめる。

7.1.1 計算量の極めて少ない解析モデルの必要性

並列計算機アーキテクチャ研究において、何らかの性能評価基準は必要不可欠のものであり、これまでに多くのメトリクスが提案されてきている。一方、現在の、そして近い将来の多種多様に渡るアーキテクチャの性能予測や、それらの上で稼働するアプリケーションの稼働予測を可能とするためには、アーキテクチャ・モデルや並列プログラム・モデルを詳細に記述する豊富なパラメータ・セットが必要となる。また、近年の素子技術の大幅な進展により、並列計算機構成要素（プロセッサ、キャッシュ、ネットワーク、メモリ）の速度や容量のレンジは飛躍的に増加しており、これに対応する上記パラメータの値域も広がっている。このため、それぞれのパラメータ・チョイスに対応した評価すべき予測の組合わせ総数が爆発的に増え、その結果として、並列計算機開発に先立つアーキテクチャの性能予測や、特定のアーキテクチャがどのような並列プログラムに適しているかを事前に知るための稼働予測が、極めて困難になりつつある。

本研究では、並列計算機研究の進展に付随して、必然的に発生する上記問題点を明確にした上で、この問題を解決するには、例えば確率過程の定常状態解等を利用した手法が有効であると考えた。

確率過程は、並列計算機研究の黎明期より、その性能予測を与える解析モデルの一手法として利用されてきたが、近年の複雑なアーキテクチャに対応したものは報告されていなかった。しかしながら、システム構成の規則性に着目した確率過程によるモデル化手法が一度確立されれば、実際の解析に要する計算量は激的に減少し、上記予測評価の組合せ的爆発にも耐え得るものと考えられる。このような解析モデルは、対象とするアーキテクチャ・モデルと並列プログラム・モデルに適当な仮定を与えることで可能であり、実際の評価を行なうのに通常のワー

クステーション上で、マイクロ秒のオーダーで計算可能なことが望ましい。

このような計算量の極めて少ない解析モデルの必要性が、本論文の主要な問題提起部分である。

7.1.2 SMCI モデルの提案

キャッシュを持つ共有メモリ型並列計算機に対する、セミマルコフ過程を利用した解析モデルとして、Semi-markov Memory and Cache coherence Interference（SMCI）モデルを提案した。SMCI モデル構築にあたっては、まず、対象となる並列計算機モデルを明確にした後、SMCI モデルの満たすべき要件と必要な仮定を述べた。状態数をシステムの規模に左右されないものにするための仮定として、1) 各プロセッサの挙動は同一の確率過程でモデル化され、各プロセッサは独立に動作し、2) 各共有ブロックの状態は、同じ確率で表される、ことが必要となった。また、キャッシュ・コヒーレンス制御命令を通常のデータ・リクエストと分離して考えるために、プロセッサ・リクエスト率に関するいくつかの新しい定義を与えた。このリクエスト率は、プロセッサ状態の極限確率から導出される。これらの準備の後、プロセッサの各状態の定義とそれらの遷移確率の導出および各状態間の滞在時間分布関数を定義し、SMCI モデルを完成させた。

SMCI モデルには、次のような利点がある。

- プロセッサ数やメモリのサービス時間、キャッシュ制御時間等のアーキテクチャ・パラメータと、共有ブロック参照率やアクセス・バースト長といった並列プログラムによって定まるソフトウェア・パラメータを有する。

- 定義されたプロセッサの各状態の極限確率を利用することにより、上記パラメータ・チョイスに対する詳細な性能比較が可能である.

- 極限確率計算にかかる時間は、卓上ワークステーションでわずか 0.6 マイクロ秒と、極めて安価である。

7.1.3 SMCI モデルの汎用性

第4章で提案したSMCIモデルは、キャッシュ・コヒーレンス・プロトコルとして、無効化主記憶転送型のSynapseを前提としており、プロセッサの状態定義はSynapseによって決定されていた。一方、本研究は、将来の大規模かつ複雑な並列計算機の開発研究に先立って、その指針を与えるための性能予測技術となる、並列計算機の汎用解析モデルを構築することである。そのため、SMCIモデルの汎用性を主張するためには、他のコヒーレンス・プロトコルへの適用も示されなければならない。

そこで第5章では、SMCIモデルの汎用性を示すために、SMCIモデルの、書き込み放送キャッシュ間転送型コヒーレンス・プロトコルDragonへの適用について考察した。その結果、プロトコルの違いはリクエスト率の修正で簡単に対応できることが判明した。また、状態定義も若干の変更で対応でき、SMCI/Synapseモデルと全く同様の手順でSMCI/Dragonモデルを構築することができた。構築されたモデルの検証は、SMCI/Synapseモデル同様、Archibaldのシミュレーション結果との比較が可能であった。その結果、SMCI/Synapseモデルよりも、より正確に性能予測が行なえるにもかかわらず、計算時間はほぼ同じであった。

以上のことより、SMCIモデルはキャッシュ・コヒーレンス・プロトコルに依存しない、並列計算機の汎用的な解析モデルと結論付けることができた。

7.1.4 SMCI モデルの拡張性

共有メモリ型並列計算機の拡張性は、スケーラブル共有メモリ型というアーキテクチャに示される。従って、SMCIモデルの拡張性を論ずるためには、同モデルのスケーラブル共有メモリ型アーキテクチャの適用について考察しなければならない。

そこで第6章では、SMCIモデルを、スケーラブル共有メモリ型アーキテクチャの特質を生かして階層的に統合構築した、階層型SMCIモデルの提案を行なった。

階層型SMCIモデル構築に先立ち、対象とするアーキテクチャ・モデルとして、階層的なメモリ・システム、階層的な相互結合網という特徴のある、仮想的かつ簡略化されたアーキテクチャを提案した。また、対象とするプログラム・モデル

として、システムの階層性を考慮に入れた、広域および局所に対応した共有および非共有のデータ・ブロックの概念を提案した。

実際の構築手法は、アーキテクチャ・モデルのプロセッサ・クラスタを一つの仮想的なプロセッサと見なし、広域データ・アクセスとグローバル・キャッシュに関するコヒーレンス制御をSMCIモデルで記述し、プロセッサ・クラスタ内は通常のSMCIモデルで記述した。また、グローバルおよびローカルのキャッシュ間の整合性を保つためのMLI保証も、これら二つのSMCIモデル間の相互作用として記述した。

7.2 今後の課題

今後の研究課題は、SMCIモデルのさらに複雑な並列分散システムへの拡張、一回のインバリデーション命令に対する平均無効化ブロック数を与える関数のモデル化、アクセス・バースト長の定式化、キャッシュ・ミス時にリプレース候補のブロックが既にモディファイされている確率の定式化等の、モデル自体の改善があげられる。

4.4.3節でも論じたように、SMCIモデルは、自動並列化コンパイラの最適パス選択のための評価モジュールとしての応用が極めて有望である。従って、最適の並列化手法を自動的に選択してくれる、性能指向型自動並列化コンパイラの研究が、本研究に直結し、かつ、本研究の研究成果を最も活かせる今後の研究課題と言えよう。

第 7 章 結論

謝辞

まず、本研究を進めるにあたり、始終あたたかい御指導を賜わりました本学福田晃教授に、心から感謝の意を表します。論文について御査読頂き、適切な御教示を賜わりました本学渡邉勝正教授ならびに尾家祐二教授に深く感謝致します。

筆者は、株式会社クボタ在職中、京都大学工学部との共同研究である分散共有メモリ型並列計算機『ASURA』プロジェクトの仕様検討会議において、京都大学工学部富田眞治教授より、『大規模な並列計算機アーキテクチャ設計に先立ち、シミュレーション等の計算コストの高い評価手法ではなく、もっと迅速に性能予測を行なうことはできないのか？』という指摘を承りました。これが本研究の発端であり、以後、今日に至るまで本研究は5年に渡って続けられてきました。このようにして、本学入学以前に本研究に携わるきっかけを頂いた京都大学工学部富田眞治教授、名古屋大学工学部阿草清滋教授、株式会社クボタ山口宗之部長、同山田清弘副部長（当時）に心から感謝の意を表します。また、当時さまざまな議論につき合ってくれた京都大学工学部森眞一郎助教授ならびに齋藤秀樹君（現在イリノイ大学シャンペイン校CSRD）に感謝致します。

本研究は、筆者の本学博士課程入学にともない、特定のアーキテクチャを対象としたものから、より一般的なものへと進展しました。本学での研究を進めるにあたり、多くの研究討議をして頂いた本学荒木啓二郎教授（現在九州大学大学院システム情報科学研究科教授）に感謝の意を表します。

本研究のアイデアの多くは、さまざまな研究討議からヒントを得たものです。

キャッシュ・ブロックに注目した解析的手法については、MIT の Prof. Anant Agarwal と同研究室の Dr. Dan Nussbaum らとの研究討議からヒントを得ました。

スイス連邦工科大 IPS 所長代理の Dr. Hans Lüthi と Dr. Peter Arbenz には、

分散メモリ環境における実アプリケーションの問題点について研究討議をして頂きました。

解析モデルの並列化コンパイラへの応用は、イリノイ大学シャンペイン校CSRDの Prof. Constantine Polychronopoulos と同研究室の Dr. Jose Moreira（現在ＩＢＭワトソン研究所研究員）および、カリフォルニア大学アーバイン校の Prof. Alex Nicolau らとのディスカッションから生まれました。特に、Prof. Polychronopoulos には、公私にわたっていろいろ助けて頂きました。

イリノイ大学シカゴ校の Prof. Alex Veidenbaum、ダートマス大の Prof. George Cybenko、Prof. Carl Beckman、コンベックス・コンピュータの Dr. Patrick McGehearty、アルゴンヌ研究所の Dr. Haghigha Mohammad 、ニューメキシコ大 Prof. Briant Smith のインテルの Dr. Utpal Banerjee 諸氏からは有益なコメントを頂きました。

ところで、筆者は、ＡＴＲ視聴覚機構研究所視覚機構研究室から、研究者としての活動を始めましたが、当時の淀川英司所長（現在工学院大学大学院工学研究科教授）、梅田三千雄室長（現在大阪電気通信大学工学部教授）ならびに三宅誠主幹研究員（現在ＮＨＫ放送技術研究所研究企画室長）には、適切なる御指導を賜わりました。特に、同研究室の佐藤隆夫主幹研究員（現在東京大学文学部助教授）には、研究者としての物の見方考え方を始めとして、実に多くのことを教えて頂きました。また、同研究所聴覚研究室の平原達也主任研究員（現在ＮＴＴ基礎研究所主幹研究員）には、連日のように深夜に及ぶ研究討議をして頂きました。

奈良女子大学理学部の落合豊行教授、ＡＴＲ人間情報通信研究所の東倉洋一社長、片桐滋主幹研究員、佐藤雅昭主任研究員、和歌山大学システム工学部の國枝義敏教授、ＲＷＣ超並列応用研究室の秋山泰室長、筑波大学電子・情報工学系の朴泰佑助教授、神戸大学工学部の中條拓伯助手の皆様方には、公私にわたっていろいろ助けて頂きました。また、義父の永岡道雄氏には有形無形の援助をして頂きました。

これらの人々の御協力に、深く感謝いたします。

最後に、経済的に困難な三年三カ月を共に乗り切ってくれた妻 江美と、その笑顔で、ふがいない父親を励まし続けてくれた長女 冴香に感謝の言葉を送りたいと思います。

参考文献

[1] Abraham, S. G. and Davidson, E. S.: A Communication Model for Optimizing Hierarchical Multiprocessor Systems, *In Proceedings of International Conference on Parallel Processing*, pp. 467–474 (1986).

[2] Adve, S. V., Adve, V. S., Hill, M. D. and Vernon, M. K.: Comparison of Hardware and Software Cache Coherence Schemes, *In Proceedings of International Symposium on Computer Architecture*, pp. 298–308 (1991).

[3] Agarwal, A., Simoni, R., Hennessy, J. and Horowitz, M.: An Evaluation of Directory Schemes for Cache Coherence, *In Proceedings of International Symposium on Computer Architecture*, pp. 280–289 (1988).

[4] Agrawal, D. P. and Mahgoub, I. O.: Performance Analysis of Cluster-Based Supersystems, *In Proceedings of International Conference on Supercomputing Systems*, pp. 593–602 (1985).

[5] Agrawal, D. P. and Mahgoub, I. O.: Analysis of a Class of Cluster-Based Multiprocessor Systems, *Information Science International Journal*, Vol. 43, pp. 85–105 (1987).

[6] Algudady, M. S., Das, C. R. and Thazhuthaveetil, M. J.: A Write Update Cache Coherence Protocol for Min-based Multiprocessors with Accessibility-based Split Caches, *In Proceedings of Supercomputing 90*, pp. 544–553 (1990).

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

[8] Baer, J. and Wang, W.-H.: Architecutural Choices for Multilevel Cache Hierarchies, *In Proceedings of the 1987 International Conference on Parallel Processing*, pp. 258–261 (1987).

[9] Bhandarkar, D. P.: *Analytic Models for Memory Interference in Multiprocessor Computer Systems*, PhD Thesis, CMU E.E. (1973).

[10] Bhandarkar, D. P.: Analysis of Memory Interference in Multiprocessors, *IEEE Trans. on Computer*, Vol. C-24, No. 9, pp. 897–908 (1975).

[11] Bhuyan, L. N.: A Combinatorial Analysis of Multibus Multiprocessors, *In Proceedings of International Conference on Parallel Processing*, pp. 225–227 (1984).

[12] Bhuyan, L. N.: An Analysis of Processor-Memory Interconnection Networks, *IEEE Trans. on Computer*, Vol. C-34, No. 3, pp. 279–283 (1985).

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

[14] Cheriton, D. R. and A.Goosen, H.: Paradigm: A Highly Scalable Shared-Memory Multicomputer Architecture, *IEEE Computer*, Vol. 24, No. 2, pp. 33–46 (1991).

[15] Chiang, M. and Sohi, G.: Experience with Mean Value Analysis Models for Evaluating Shared Bus, Throughput-Oriented Multiprocessors, *In Proceedings of ACM SIGMETRICS*, pp. 173–182 (1990).

[16] Conte, G. et al.: TOMP80–A Muptiprocessor Prototype, *In Proceedings of EUROMICRO81* (1981).

[17] Dubois, M.: Effect of Invalidations on the Hit Ratio of Cache-based Multiprocessors, *In Proceedings of International Conference on Parallel Processing*, pp. 255–257 (1987).

[18] Dubois, M.: Throughput Analysis of Cache-Based Multiprocessors with Multiple Buses, *IEEE Trans. on Computer*, Vol. C-37, No. 1, pp. 58–70 (1988).

[19] Dubois, M., Briggs, F., Patil, I. and Balakrishnan, M.: Trace-driven Simulations of Parallel and Distributed Algorithms in Multiprocessors, *In Proceedings of International Conference on Parallel Processing*, pp. 909–916 (1986).

[20] Dubois, M. and Briggs, F. A.: Effects of Cache Coherency in Multiprocessor, *IEEE Trans. on Computer*, Vol. C-31, No. 11, pp. 1083–1099 (1982).

[21] Dubois, M. and Thakkar, S. S.(eds.): *Scalable Shared Memory Multiprocessors*, Kluwer Academic (1992).

[22] Dubois, M. and Wang, J.: Shared Data Contention in a Cache Coherence Protocol, *In Proceedings of International Conference on Parallel Processing*, pp. 146–155 (1988).

[23] Flynn, M. J.: *Computer Architecture: Pipelined and Parallel Processor Design*, JONES and BARTLETT, chapter 8 (1995).

[24] Gupta, A. and Weber, W.: Cache Invalidation Patterns in Shared-Memory Multiprocessors, *IEEE Trans. on Computers*, Vol. 41, No. 7, pp. 794–810 (1992).

[25] Holliday, M. A. and Vernon, M. K.: Exact Performance Estimates for Multiprocessor Memory and Bus Interference, *IEEE Trans. on Computer*, Vol. C-36, No. 1, pp. 76–85 (1987).

[26] Hoogendoorn, C. H.: A General Model for Memory Interference in Multiprocessors, *IEEE Trans. on Computer*, Vol. C-26, No. 10, pp. 998–1005 (1977).

[27] Irani, K. B. and Onyuksel, I. H.: A Closed-Form Solution for the Performance Analysis of Multiple-Bus Multiprocessor Systems, *IEEE Trans. on Computer*, Vol. C-33, No. 11, pp. 1004–1012 (1984).

[28] Joe, K. and Fukuda, A.: An Analytic Model for a Parallel Computer – Prediction of a Shared Block Behavior –, *In Proceedings of the 1st workshop on Performance Evaluation of Parallel Systems*, pp. 67–74 (1993).

[29] Jōg, R., Vitale, P. L. and Callister, J. R.: Performance Evaluation of a Comercial Cache-Coherent Shared Memory Multiprocessor, *In Proceedings of ACM SIGMETRICS*, pp. 173–182 (1990).

[30] Kuck, D. J., Davidson, E. S., Lawrie, D. H. and Sameh, A. H.: Parallel Supercomputing Today and the Cedar Approach, *Science*, Vol. 231, No. 2, pp. 967–974 (1986).

[31] Lang, T., Valero, M. and Alegre, I.: Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors, *IEEE Trans. on Computer*, Vol. C-31, No. 12, pp. 1227–1234 (1982).

[32] Lazowska, E. D., Zahorjan, J., Graham, G. S. and Sevcik, K. C.: *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall (1984).

[33] Lefvoort, A. and Subramanian, N.: A New Approach for the Performance Analysis of a Single-Bus Multiprocessor System with General Service Times, *IEEE Trans. on Computer*, Vol. C-42, No. 3, pp. 358–362 (1993).

[34] Lenoski, D. E.: *The Design and Analysis of Dash: A Scalable Directory-based Multiprocessor*, PhD Thesis, Stanford University, CSL (1992).

[35] Leutenegger, S. and Vernon, M.: A Mean-Value Performance Analysis of a New Multiprocessor Architecture, *In Proceedings of SIGMETRICS*, ACM, pp. 167–176 (1988).

[36] Mahgoub, I. O. and Elmagarmid, A. K.: Performance Analysis of a Generalized Class of m-Level Hierarchical Multiprocessor Systems, *IEEE Trans. on Parallel and Distributed System*, Vol. 3, No. 2, pp. 129–138 (1992).

[37] Mahmud, S.: Performance Analysis of Multilevel Bus Networks for Hierarchical Multiprocessors, *IEEE Trans. on Computer*, Vol. C-43, No. 7, pp. 789–805 (1994).

[38] Mahmud, S. M.: Performance Analysis of Asynchronous Hierarchical-Bus Multiprocessor Systems Using Closed Queuing Network Moddels, *In Proceedings of International Symposium on Circuits and Systems*, pp. 2689–2692 (1990).

[39] Mahmud, S. M. and Tiruveedhula, V.: Hit Ratio and Communication Cost of Shared Data in a Cache-Based System with Multistage Interconnection Network, *In Proceedings of International Conference on Parallel Processing*, pp. 173–176(I) (1990).

[40] Marsan, M. A., Balbo, G. and Conte, G.: *Performance Models of Multiprocessor Systems*, MIT Press (1983).

[41] Marsan, M. A., Balbo, G. and Conte, G.: Comparative Performance Analysis of Single Bus Multiprocessor Architectures, *IEEE Trans. on Computer*, Vol. C-31, No. 12, pp. 1179–1191 (1982).

[42] Marsan, M. A., Balbo, G. and Conte, G.: A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems, *ACM Trans. on Computer System*, Vol. 2, No. 2, pp. 93–122 (1984).

[43] Marsan, M. A., Balbo, G., Conte, G. and Gregoretti, F.: Modeling Bus Contention and Memory Interference in a Multiprocesor System, *IEEE Trans. on Computer*, Vol. C-32, No. 1, pp. 60–72 (1983).

[44] Marsan, M. A. and Gerla, M.: Markov Models for Multiple Bus Multiprocessor Systems, *IEEE Trans. on Computer*, Vol. C-31, No. 3, pp. 239–248 (1982).

[45] McCredie, J. W.: Analytic Models as Aids in Multiprocessor Design, *In Proceedings of Ann. Princeton Conference on Information Science and System*, pp. 186–191 (1973).

[46] Mizrahi, H. E., Baer, J., Lazowska, E. D. and Zahorjan, J.: Extending the Memory Hierarchy into Multiprocessor Interconnection Networks: A Performance Analysis, *In Proceedings of International Conference on Parallel Processing* (1989).

[47] Mohapatra, P., Das, C. R. and Feng, T.: Performance Analysis of Cluster-Based Multiprocessors, *IEEE Trans. on Computer*, Vol. C-43, No. 1, pp. 109–114 (1994).

[48] Mori, S., Saito, H., Goshima, M., Yanagihara, M., Tanaka, T., Joe, K., Fraser, D., Nitta, H. and Tomita, S.: A Distributed Shared Memory Multiprocessor: ASURA – Memory and Cache Architectures–, *In Proceedings of Supercomputing 93*, pp. 740–749 (1993).

[49] Mudge, T. N. and Al-Sadoun, H. B.: A Semi-Markov Model for the Performance of Multiple-Bus Systems, *IEEE Trans. on Computer*, Vol. C-34, No. 10, pp. 934–942 (1985).

[50] Mudge, T. N. and Al-Sadoun, H. B.: Memory Interference Models with Variable Connection Time, *IEEE Trans. on Computer*, Vol. C-33, No. 11, pp. 1033–1038 (1984).

[51] Mudge, T. N., Hayes, J. P., Buzzard, G. D. and Winsor, D. C.: Analysis of Multiple-Bus Interconnection Networks, *J. of Parallel and Distributed Computing*, Vol. 3, No. 3, pp. 328–343 (1986).

[52] Mudge, T. N., Hayes, J. P., Buzzard, G. D. and Winsor, D. C.: Analysis of Multiple Bus Interconnection Networks, *In Proceedings of International Conference on Parallel Processing*, pp. 228–232 (1984).

[53] Onyuksel, I. H. and Irani, K. B.: Markovian Queueing Network Models for Performance Analysis of a Single-Bus Multiprocessor System, *IEEE Trans. on Computer*, Vol. C-39, No. 7, pp. 975–990 (1990).

[54] Onyuksel, I. H. and Irani, K. B.: A Markovian Queueing Network Model for Performance Evaluation of Bus-Deficient Multiprocessor Systems, *In Proceedings of International Conference on Parallel Processing*, pp. 437–439 (1983).

[55] Osaki, S.: *Applied Stochastic System Modeling*, Springer-verlag (1992).

[56] Patel, J. H.: Analysis of Multiprocessors with Private Cache Memories, *IEEE Trans. on Computer*, Vol. C-31, No. 4, pp. 296–304 (1982).

[57] Polychronopoulos, C. D.: *Parallel Programming and Compilers*, Kluwer Academi (1988).

[58] Polychronopoulos, C. D. et al.: Parafrase-2: An Environment for Compiling, Partitioning,Synchronizing, and Scheduling Parallel Programs, *In Proceedings of International Conference on Parallel Processing*, pp. 39–48(II) (1989).

[59] Ramani, A. K., Chande, P. K. and Sharmma, P. C.: A General Model for Performance Investigations of Priority Based Multiprocessor System, *IEEE Trans. on Computer*, Vol. C-41, No. 6, pp. 747–754 (1992).

[60] Rau, B. R.: Interleaved Memory Bandwidth in a Model of a Multiprocessor Computer System, *IEEE Trans. on Computer*, Vol. C-28, No. 9, pp. 678–681 (1979).

[61] Sethi, A. S. and Deo, N.: Interference in Multiprocessor Systems with Localized Memory Access Probabilities, *IEEE Trans. on Computer*, Vol. C-28, No. 2, pp. 157–163 (1979).

[62] Siomalas, K. O. and Bowen, B. A.: Performance of Cross-Bar Multiprocessor Systems, *IEEE Trans. on Computer*, Vol. C-32, No. 7, pp. 689–695 (1992).

[63] Towsley, D.: Approximate Models of Multiple Bus Multiprocessor Systems, *IEEE Trans. on Computer*, Vol. C-35, No. 3, pp. 220–228 (1986).

[64] Vernon, M. K. and Holliday, M. A.: Performance Analysis of Multiprocessor Cache Consistency Protocols Using Generalized Timed Petri Nets, *In Proceedings of ACM SIGMETRICS*, pp. 9–17 (1986).

[65] Vernon, M. K., Jōg, R. and Sohi, G. S.: The TREEBus Architecture and its Analysis, Technical Report TR-747, U.Wisconsin-CS (1988).

[66] Vernon, M. K., Jōg, R. and Sohi, G. S.: Performance Analysis of Hierarchical Cache-Consistent Multiprocessors, *Performance Evaluation*, Vol. 9, pp. 287–302 (1989).

[67] Vernon, M. K., Lazowska, E. D. and Zahorjan, J.: An Accurate and Efficient Performance Analysis Technique for Multiprocessor Snooping Cache-Consistency Protocols, *In Proceedings of International Symposium on Computer Architecture*, pp. 308–315 (1988).

[68] Wang, J. and Dubois, M.: Performance Comparison of Cache Coherence Protocols Based on the Access Burst Model, *Computer System Science and Engeneering*, Vol. 5, No. 3, pp. 147–158 (1990).

[69] Wang, J., Dubois, M. and Brigs, F. A.: Analytical Modeling for Finite Cache Effects, *In Proceedings of International Conference on Parallel Processing*, pp. 287–291(I) (1991).

[70] Weber, W. and Gupta, A.: Analysis of Cache Invalidation Patterns in Multiprocessors, *In Proceedings of Architecture Support for Programing Language and Operating System*, pp. 243–256 (1989).

[71] Yen, D. W., Patel, J. H. and Davidson, E. S.: Memory Interference in Synchronous Multiprocessor Systems, *IEEE Trans. on Computer*, Vol. C-31, No. 11, pp. 1116–1121 (1982).

[72] 趙永健, 岡田博美, 前川禎男: マルチプロセッサシステムにおける共用メモリアクセス競合の解析, 電子情報通信学会論文誌, Vol. J70-D, No. 5, pp. 870–879 (1987).

[73] 高橋義造 (編): 並列処理機構, 丸善, chapter 5 (1989).

[74] 池原悟: マルチプロセッサ方式における共用メモリアク セス競合の解析, 電子通信学会論文誌, Vol. J63-D, No. 4, pp. 334–341 (1980).

[75] 鳥居淳, 竹本卓, 天野英晴, 小椋里: バス結合型並列計算機の交信用メモリの性能評価, 情報処理学会論文誌, Vol. 33, No. 3, pp. 307–319 (1992).

[76] 小和田正: 確率過程とその応用, 実教出版 (1983).

[77] 木村哲朗, 山本欧, 天野英晴, 竹本卓: 並列システム解析モデル：STMT ネット, 電子情報通信学会論文誌, Vol. J75-D, No. 8, pp. 654–663 (1992).

[78] 古谷立美: バス結合マルチプロセッサシステムの解析モデ ルと解析, 情報処理学会論文誌, Vol. 17, No. 5, pp. 394–401 (1976).

参考文献

付録

A.1　特定の共有ブロックのキャッシング確率

以下に、あるプロセッサが特定の共有ブロックをアクセスしようとしている時に、それ以外のプロセッサに対応するキャッシュが当該ブロックをキャッシングしている確率を導出する過程について述べる。

プロセッサからのデータ・リクエストが、キャッシュ・コヒーレンス制御命令を引き起こすかどうかを知るには、リクエストされたブロックが他のキャッシュに保持されているかどうかを予測しなければならない。一般に、リクエストされたデータ・ブロックが、自分のキャッシュに保持されているかどうかは、ＬＲＵスタック・モデルを用いて近似されることが知られている [20]。しかしながら、リクエストされた共有ブロックが、他のキャッシュに保持されているかどうかを示す分布は、明らかに単純なスタック・モデルでは近似できない。なぜなら、SPMD 並列プログラミング・モデルで示されるように、並列プログラムの多くは、コントロール・シーケンスは各プロセッサとも同じような処理がなされるのに対し、必要とされるデータ・アクセスのシーケンスは異なるからである。

本論文では、文献 [20] と同様、リクエストされた共有ブロックが、他のキャッシュに保持されているかどうかを示す分布として、一様分布を仮定する。この仮定に従うと、リクエストされた共有ブロックが、他のキャッシュに保持されている確率は、共有ブロックの総数 E と、その時点で自分のキャッシュに保持されている共有ブロック数 Ψ を使って、Ψ/E で表すことができる。ただし、E は並列プログラムに依存する変数であり、本モデルでは入力パラメータとして与えられる。また、Ψ を求めるには、共有ブロックに対するキャッシュ・ヒット率が必要であ

る。4.2.1節で述べたように、本論文では、共有ブロックに対するキャッシュ・ヒット率Hに関しては、Wang らの文献 [68] で与えられる キャッシュ・ミス率を使って、Synapse の場合、

$$H = 1 - \frac{1}{ls}\frac{N(N-1)(1-r)}{(N-r)(1+(N-1)(1-r))} \tag{A.1}$$

で、Dragon の場合

$$H = 1 - \frac{1}{ls}\frac{(N-1)(1-r)}{1+(N-1)(1-r)} \tag{A.2}$$

で近似する。(Nはプロセッサ台数、rはリード・リクエスト率、ls はアクセス・バースト長。) ただし、後者は Dragon に対して求められたものではなく、Berkeley、Illinois に対するものである。Dragon の性質上、共有ブロックに対するキャッシュ・ヒット率は、Berkeley、Illinois に対するものより若干良いと思われるが、Dragon に対するキャッシュ・ヒット率の定式化がなされていないため、上記で代用する。

ここで、共有ブロックに対するキャッシュ・ヒット率Hは近似であることを明確にしておこう。Wang らのアクセス・バースト・モデルは第2章で概説したように、無限個のキャッシュ・エントリーを仮定している。これに対して本モデルでは有限個のキャッシュ・エントリーを仮定しており、厳密には文献 [28] 等で報告されている、有限個のキャッシュ・エントリーを持つ並列計算機のキャッシュの解析モデルを利用して、共有ブロックに対するキャッシュ・ヒット率を与えるべきである。しかしながら、文献 [28] によれば、共有ブロックに対するキャッシュ・ヒット率に影響を与えるプライベート・ブロックに対するヒット率は、かなり低い値 (90%以下) で顕著になるのであって、本論文での評価対象である 95% 以上 (4.4節参照) のヒット率では、その影響は極めて少ない。また、文献 [28] では、Wang らのようにキャッシュの状態遷移の定常分布を式で表せていないため、定常分布を求めるために大量の計算が必要となる。以上のことを鑑みて、本論文では共有ブロックに対するキャッシュ・ヒット率を近似値で与えることにした。

次にキャッシュに保持されている共有ブロック数Ψを求める。先に述べたように、リクエストされたデータ・ブロックが、自分のキャッシュに保持されているかどうかは、LRU スタック・モデルを用いて近似されることと、本論文での評価対象を考えて、このアクス・パターンの分布は文献 [7] と同じものを使うことにする。すなわち、ある共有ブロックが、LRU スタック上において、i 番目にある

確率は、

$$g \times (\frac{1}{5+i} - \frac{1}{6+i}) \tag{A.3}$$

である。ただし、gは正規化係数である。よって、Ψは次の方程式で求まる。

$$\frac{\int_0^{\Psi} g \times (\frac{1}{5+x} - \frac{1}{6+x})dx}{\int_0^{E} g \times (\frac{1}{5+x} - \frac{1}{6+x})dx} = H \tag{A.4}$$

式を簡単にして、Ψは次で求められる。

$$\Psi = \frac{6 \times \frac{5}{6}\left(\frac{6(5+E)}{5(6+E)}\right)^H - 5}{1 - \frac{5}{6}\left(\frac{6(5+E)}{5(6+E)}\right)^H} \tag{A.5}$$

よって、特定の共有ブロックのキャッシング確率は、

$$\frac{\Psi}{E} = \frac{\left(\frac{6(5+e)}{5(6+E)}\right)^H - 1}{1 - \frac{5}{6}\left(\frac{6(5+E)}{5(6+E)}\right)^H}\frac{5}{E} \tag{A.6}$$

で与えられる。

A.2 階層型クロスバー・ネットワークのリクエスト受理確率

プロセッサが別のクラスタにリクエストを発行するとき、階層型クロスバー・ネットワークの構成上、行き先のクラスタがどのレベルのグループに含まれているかによって、通信遅延時間もリクエスト受理確率も異なってくる。例えば、リクエスト先のクラスタがC_1グループに属する場合とC_2グループに属する場合を考えてみると、前者はC_1グループ内のクロスバーを一度通過するだけなのに対し、後者はC_1グループのクロスバーを二度、C_2のクロスバーを一度通過しなければならない。

ここでは、各C_iグループでのリクエスト発生率を与えることにより、プロセッサが任意のクラスタに対してリクエストを発行した時の受理確率を、簡単な確率モデルを用いて表す。ただし、このリクエストは通常のデータ・アクセスだけではなく、キャッシュ・コヒーレンス制御のためのリクエストも含まれるものとする。

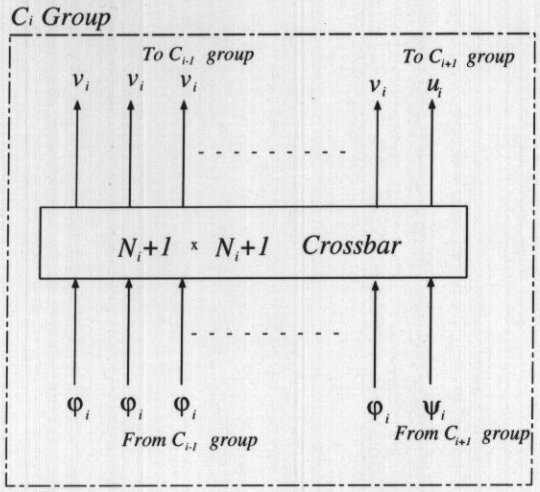


図 A.1 C_iグループにおける入力と出力

まず、kレベルの階層型クロスバー・ネットワークにおいて、C_iグループ $(1 \leq i < k)$ の N_i+1 入力 N_i+1 出力クロスバーに対するリクエスト率を定義する。

C_iグループにおいて、C_{i-1}グループからのリクエスト率をφ_i、そのリクエスト先が同じ C_iグループに存在する確率を S_i、C_{i+1}グループに対するリクエスト率を u_i、C_{i+1}グループからのリクエスト率をψ_i、C_{i-1}グループへのリクエスト率をv_iとする。つまり、図A.1で示されるように、C_iグループのクロスバーの入力は、リクエスト率φ_iであるものが N_i個と ψ_i であるものが 1 個の計 N_i+1 入力となり、出力はリクエスト率 v_iであるものが N_i個と u_iであるものが 1 個の計 N_i+1 出力となる。各リクエスト率は次のようにして求められる。

$$\varphi_i = u_{i-1} \tag{A.7}$$

$$u_i = 1-(1-\frac{S_i}{\sum_{j=i}^{k} S_j}\varphi_i)^{N_i} \tag{A.8}$$

$$\psi_i = v_{i+1} \tag{A.9}$$

$$v_i = 1-(1-\frac{\varphi_i}{N_i}\frac{S_i}{\sum_{j=i}^{k} S_j})^{N_i}(1-\frac{\psi_i}{N_i}) \tag{A.10}$$

ただし、φ_1はクラスタからの階層型クロスバー・ネットワークに対するリクエスト、すなわち、ネットワーク・リクエスト率を意味する。次に最上位のグルー

プ C_kについては、上位グループとの通信は存在しないので、

$$\varphi_k = u_{k-1} \tag{A.11}$$

$$v_k = 1 - (1 - \frac{\varphi_k}{N_k})^{N_k} \tag{A.12}$$

によって求められる。

それでは階層型クロスバー・ネットワークにおけるリクエスト受理確率を考えてみよう。$ArbUp(i)$ を C_iグループにおいて、C_{i-1}グループから、自分より上位のグループ（S_j $(j > i)$）に対してリクエストが発行された時に、そのリクエストが N_i+1 入力 N_i+1 出力クロスバー・ネットワークで受理される確率とする。同様に $ArbDown(i)$ を C_iグループにおいて、自分もしくは C_{i+1}グループから、自分より下位のグループ（S_j $(j < i)$）に対してリクエストが発行された時に、そのリクエストが N_i+1 入力 N_i+1 出力クロスバー・ネットワークで受理される確率とする。この時、$ArbUp(i)$、$ArbDown(i)$ はそれぞれ次のように表される。ただし、C_k グループにおけるψ_kを、形式的に$\psi_k = 0$ と定義しておく。

$$ArbUp(i) = \frac{u_i}{N_i \frac{\sum_{j=i+1}^{k} S_j}{\sum_{j=i}^{k} S_j} \varphi_i} \tag{A.13}$$

$$ArbDown(i) = \frac{v_i}{N_i \frac{S_i}{\sum_{j=i}^{k} S_j} \varphi_i + \psi_i} \tag{A.14}$$

よって、あるプロセッサから S_iのリクエストが発せられたときに、そのリクエストが階層型クロスバー・ネットワークによって受理される確率 $ARB(i)$ は次のように表される。

$$ARB(i) = \left(\prod_{j=1}^{i-1} ArbUp(j) \right) \prod_{j=i}^{1} ArbDown(j) \tag{A.15}$$

付 録

筆者研究業績

学術論文

1 Joe, K., Mori, Y. and Miyake, S.: Construction of a Large-scale Neural Network: Simulation of Handwritten Japanese Character Recognition on NCUBE, Concurrency:practice and experience, Vol.2, No.2, pp.79–107 (1990).

2 Joe, K. and Fukuda, A.: Analytic Modeling of Cache Coherence Based Parallel Computers, 電子情報通信学会英文論文誌, Vol.E79-D, No.7（採録決定）(1996).

国際会議

3 Joe, K., Mori, Y., and Miyake, S.: Simulation of a Large-Scale Neural Network on a Parallel Computer – An Application for Handwritten Japanese Character Recognition–, In Proceedings of 4th Conference of Hypercube Concurrent Computers and Applications, Vol.2, pp.1111-1118 (1989).

4 Mori, Y. and Joe, K.: A Large-Scale Neural Network Which Recognizes Hanwritten Kanji Characters, In Proceedings of 2nd Neural Information Processing System, pp.415–422 (1990).

5 Sato, M., Joe, K. and Hirahara, T.: APOLON Brings Us to the Real World: Learning Nonlinear Dynamics and Fluctuations in Nature, In Proceedings of International Joint Conference on Neural Network, pp.581–587 (1990).

6 Sato, M., Murakami, Y. and Joe, K.: Learning Chaotic Dynamics by Recurrent Neural Networks, In Proceedings of International Conference on Fuzzy Logic and Neural Networks, pp.601–604 (1990).

7 Joe, K. and Fukuda, A.: An Analytic Model for a Parallel Computer, – Prediction of Shared Block Behavior –, In Proceedings of 1st International Workshop on Performance Evaluation on Parallel Systems, pp.67–74 (1993).

8 Mori, S., Saito, H., Goshima, M., Yanagihara, M., Tanaka, T., Joe, K., Fraser, D., Nitta, H., and Tomita, S.: A Distributed Shared Memory Multiprocessor: ASURA –Memory and Cache Architectures–, In Proceedings of IEEE Supercomputing93, pp.740–749 (1993).

9 Joe, K. and Fukuda, A.: The Semi-Markov Memory and Cache Coherence Interference Model, In Proceedings of High Performance Computing Conference'94, pp.254–265 (1994).

10 Joe, K. and Fukuda, A.: An Analytic Model for a Hierarchical Parallel System, In Proceedings of 2nd International Workshop on Massive Parallelism: Hardware, Software and Applications, pp.287–304 (1994).

11 Omori, Y., Joe, K., Fukuda, A. and Araki, K.: A Parametric Simulator For a Parallel Computer Using Address-Trace, In Proceedings of International Computer Symposium, pp.979–985 (1994).

12 Nakanishi, T., Joe, K., Polychronopoulos, C., Fukuda, A. and Araki, K.: The Data Partitioning Graph: Extending Data and Control Dependencies for Data Partitioning, In Lecture Notes of Computer Science 892: 7th International Wokshop on Languages and Compilers for Parallel Computing, pp.170–185 (1994).

13 Kitasuga, T., Joe, K., Schouten, D., Fukuda, A. and Araki, K.: A Loop Parallelization Technique for Linear Dependence Vector, In Proceedings of

International Conference on Parallel Architectures and Compilation Techniques, pp.285–289 (1995).

14 Nakanishi, T., Joe, K., Saito, H., Fukuda, A. and Araki, K.: CDP^2 Algorithm: Combined Data and Program Partitioning, In Proceedings of International Conference on Parallel Processing, pp.177–181(II) (1995).

15 Saito, T., Joe, K., Lüthi, H., Arbenz, P., Fukuda, A. and Araki, K.: An Analysis of the SCIDDLE Library on a Workstation Cluster: A Step Towards a Globally Distributed Locally Parallel Processing Environment, In Proceedings of International Symposium on Parallel and Distributed Supercomputing, pp.104–111 (1995).

16 Nakanishi, T., Joe, K., Polychronopoulos, C.D., Araki, K. and Fukuda, A.: Estimating Minimum Execution Time of Perfect Loop Nests with Loop-Carried Dependences, In Proceedings of 9th International Workshop on Languages and Compilers for Parallel Computing, （採録決定）(1996).

17 Sasakura, M., Kiwada, S., Joe, K. and Araki, K.: 3D Visualization of Program Structure and Data Dependence for Parallelizing Compilers and Parallel Programming, In Proceedings of 9th International Workshop on Languages and Compilers for Parallel Computing, （採録決定）(1996).

18 Nakanishi, T., Joe, K., Polychronopoulos, C. D., Araki, K. and Fukuda A.: Estimating Parallel Execution Time of Loops with Loop-Carried Dependences, In Proceedings of International Conference on Parallel Processing, (採録決定), (1996).

19 Koita, T., Joe, K. and Fukuda, A.: A Data Block Mapping Method to Reduce Cache Coherence Overhead, In Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, （採録決定） (1996).

20 Joe, K. and Fukuda, A.: Applying the Semi-Markov Memory and Cache Coherence Interference Model to an Updating Based Cache Coherence Protocol, In Proceedings of Euro-Par96 (採録決定) (1996).

著書

21 Fukuda, A. and Joe, K.: Toward the Single and Multiple Programmed Environments for the Future Large-Scaled Parallel Systems, (ed. A.Nicolau and M.Sato: Parallel Language and Compiler Research in Japan, pp.325–347, 分担共著, Kluwer Academic Publishers (1995).

特許

22 アレイ・キャッシュ方式, 出願 (1987).

23 ニューラル・ネットワークの学習アルゴリズム 1 , 出願 (1989).

24 ニューラル・ネットワークの学習アルゴリズム 2 , 出願 (1989).

25 ニューラル・ネットワークの学習アルゴリズム 3 , 出願 (1989), 受理 (1995).

26 ニューラル・ネットワークの学習アルゴリズム 4 , 出願 (1989).

27 ノン・ブロッキング・キャッシュ方式, 出願 (1992).

雑誌記事

28 三宅誠, 城和貴: ノイマン型からニューロコンピュータへ, 遺伝別冊, Vol. 2, pp.96–101 (1989).

29 城和貴: ImageHoiHoi –Lisp 環境下での画像処理システム–, 映像情報, Vol. 22, pp.27-32 (1990).

30 城和貴, 福田晃: 並列計算機の解析モデル, 情報処理学会誌, Vol. 37, No. 6, pp.528-535 (1996).

筆者研究業績

招待講演

31 Alliant, Convex, nCUBE の視聴覚情報処理への応用, 情報処理学会関西支部ソフトウェア研究会第１回講習会, 招待講演 (1988).

32 Lisp マシンによる画像処理, セコムＩＳ研, 招待講演 (1989).

33 ミニスーパーコンピュータの視聴覚情報処理への応用, 第７３回情報通信システム分科会, 招待講演 (1990).

34 The Overview of ASURA, Seminars at the University of Illinois, the University of Wisconsin, DEC-Cambridge and OSF (1991).

35 並列計算機と解析モデル, ＡＴＲ人間情報通信研究所, セミナー (1993).

36 並列計算機の解析モデル, 九州大学工学部, セミナー (1993).

37 An Analytic Model for parallel computers, Seminar at the University of Bristol (1993).

38 並列計算機アーキテクチャ, 奈良女子大学理学部, セミナー (1994).

39 An Analytic Model for a Hierarchical Parallel System, Invited talk at ETHZ (1994).

40 An Analytic Modeling Technique for a Scalable Shared Memory Multiprocessors, Seminor at Convex Computer, Dallas (1994).

41 The Semi-markov Memory and Cache coherence Interference Model, Seminor at Dartmouth Colledge (1994).

42 Towards a Globally Distributed Locally Parallel Processing Environment, Invited talk at Maui High Performance Computing Center (1995).

シンポジウム

43 笹倉万里子, 木和田智子, 城和貴, 荒木啓二郎: 並列化支援視覚化システム NaraView におけるプログラム情報の 3 次元表示法, 情報処理学会, 並列処理シンポジウム JSPP'96, pp.267–274(1996).

44 中西 恒夫, 城 和貴, Polychronopoulos, C., 福田 晃, 荒木 啓二郎: ループ並列実行時間の下限を算出する一手法, 情報処理学会, 並列処理シンポジウム JSPP'96, pp.57–64(1996).

口頭発表等

45 城和貴, 梅田三千雄: ＡＣＢ（アレイ・キャッシュ・バッファ）による高速アレイ・アクセス, 情報処理学会第 35 回全国大会講演論文集, 5C-7, pp.177–178 (1987).

46 城和貴, 梅田三千雄: リスプ環境下における高速画像処理, 電子情報通信学会情報システム部門全国大会講演論文集、pp.129 (1987).

47 城和貴: リスプ・マシン用イメージ・スキャナ接続プログラム「ＫＡＯＲＵ」, ＡＴＲテクニカル・レポート, TR-A-0007 (1987).

48 城和貴: Ｕｎｉｘのセキュリティに関する考察, ＡＴＲテクニカル・レポート, TR-A-0008 (1987).

49 城和貴, 梅田三千雄: リスプによる高速画像処理環境, 情報処理学会第 36 回全国大会講演論文集, pp.1757–1758 (1988).

50 城和貴, 佐藤嘉伸, 梅田三千雄: 解像度ピラミッドを用いた輪郭抽出, 電子情報通信学会春季全国大会, 1D-515, pp.258 (1988).

51 城和貴, 梅田三千雄: リスプによる高速画像処理環境と処理システム【Ｉ】, 電子情報通信学会パターン認識と理解研究会, PRU87-86, pp.31–38 (1988).

52 城和貴: リスプによる高速画像処理環境と処理システム【ＩＩ】, 電子情報通信学会パターン認識と理解研究会, PRU88-7, pp.15–22 (1988).

53 城和貴, 緒方昌美: シンボリックス・ソフトウェア環境の内部構造と機能, ＡＴＲテクニカル・レポート, TR-A-0013 (1988).

54 森吉弘, 城和貴: 大規模ニューラルネットワークの構築, 第１０回神経情報科学研究会, ポスターセッション (1988).

55 積山香, 城和貴, 梅田三千雄: 読唇による単音節の認識, テレビジョン学会視覚情報処理研究会, IE87-113, pp.33–40 (1988).

56 積山香, 城和貴, 梅田三千雄: 単音節の読唇における混同行列の分析 –多次元尺度法による知覚属性の検討–, 電子情報通信学会パターン認識と理解研究会, PRU87-113, pp.29–36 (1988).

57 積山香, 城和貴, 梅田三千雄: 読唇における日本語単音節の心理空間, 日本心理学会第５２回大会, 感覚・知覚, 2-6, pp.515 (1988).

58 積山香, 城和貴, 梅田三千雄: 読唇における単音節の知覚的特徴, 日本教育心理学会第３０回総会, 学習, pp.694–695 (1988).

59 城和貴: Alliant, Convex, Ncube のアーキテクチャとパフォーマンス, ＡＴＲテクニカル・レポート, TR-A-0045 (1989).

60 城和貴, 森吉弘: ニューラルネットの学習における新しいタイプの教師, 電子情報通信学会パターン認識と理解研究会, PRU88-94, pp.23–30 (1988).

61 城和貴, 森吉弘: 階層型ニューラルネットの統合, 電子情報通信学会秋季全国大会シンポジウム, SD-11-4, pp.282–283 (1989).

62 森吉弘, 城和貴: 大規模ニューラルネットによる手書き漢字認識, 電子情報通信学会秋季全国大会シンポジウム, SD-11-5, pp.284–285 (1989).

63 佐藤雅昭, 城和貴, 平原達也: リカレントネットによる音声ゆらぎの学習, 電子情報通信学会ニューロコンピューティング研究会, NC90-140, pp.169–174 (1991).

64 森眞一郎, 齋藤秀樹, 五島正裕, 富田眞治, 田中高士, David Fraser, 城和貴, 新田博之: 分散共有メモリ型マルチプロセッサ「阿修羅」の概要, 情報処理学会計算機アーキテクチャ研究会, ARC94-6, pp.41-48 (1992).

65 城和貴, 柳原守, 田中高士, David Fraser, 新田博之, 齋藤秀樹, 森眞一郎, 富田眞治: 分散共有メモリ型マルチプロセッサ「ＡＳＵＲＡ」の階層性とその評価, 情報処理学会計算機アーキテクチャ研究会, ARC95-1, pp.1–8 (1992).

66 齋藤秀樹, 森眞一郎, 富田眞治, 田中高士, David Fraser, 城和貴: イベント対応型キャッシュ・コヒーレンス制御方式の応用事例とその基本性能, 情報処理学会計算機アーキテクチャ研究会, ARC95-2, pp.9-17 (1992).

67 城和貴, 内藤潤: セミ・マルコフ過程を用いたＡＳＵＲＡクラスタの解析モデル, 情報処理学会計算機アーキテクチャ研究会, ARC97-9, pp.65–72 pp. (1992).

68 内藤潤, 城和貴, 松野宏明, 新田博之: ＡＳＵＲＡクラスタの性能評価, 情報処理学会計算機アーキテクチャ研究会, ARC97-10, pp.73–80 (1992).

69 齋藤秀樹, 森眞一郎, 富田眞治, 田中高士, David Fraser, 城和貴, イベント対応型キャッシュ・コヒーレンス制御方式とそのバリア同期への応用, 情報処理学会計算機アーキテクチャ研究会, ARC98-17, pp.129–136 (1992).

70 城和貴: ＡＳＵＲＡの解析モデル, 電子情報通信学会コンピュータ・システム研究会, CPSY92-90, pp.81–88 (1993).

71 城和貴, 福田晃: 並列計算機の解析モデル –シミュレーションとの比較–, 情報処理学会計算機アーキテクチャ研究会, ARC100-3, pp.17-24 (1993).

72 城和貴, 福田晃: 並列計算機の解析モデル –共有ブロックの予測– , 情報処理学会ハイパフォーマンスコンピューティング研究会, HPC48-11, pp.81-88 (1993).

73 中西恒夫, 城和貴, 福田晃, 荒木啓二郎: ＤＰＧ:データ・パーティショニング・グラフ, 情報処理学会計算機アーキテクチャ研究会, ARC104-16, pp.121–128 (1993).

74 大森洋一, 城和貴, 福田晃: 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, 情報処理学会ハイパフォーマンスコンピューティング研究会, HPC48-15, pp.113–120 (1993).

75 笹倉真理子, 中西恒夫, 城和貴, 荒木啓二郎: 形式化に基づく並列性抽出－既存逐次プログラムを並列実行するためのパラダイム, 情報処理学会ソフトウェア工学研究会, SE99-3, pp.17–24 (1994).

76 北須賀輝明, 城和貴, 福田晃, 荒木啓二郎: 線形依存ベクトルのループの並列性抽出法, 情報処理学会プログラミング—言語・基礎・実践— 研究会, PRG18-4, pp.25-32 (1994).

77 中西恒夫, 城和貴, Polychronopolous, D., 福田晃, 荒木啓二郎: ＨＤＰＧ：階層データ分割グラフ, 情報処理学会ハイパフォーマンスコンピューティング研究会, HPC51-15, pp.89–94 (1994).

78 笹倉万里子, 木和田智子, 城和貴, 荒木啓二郎: 複合グラフを用いた階層タスクグラフの視覚化, 情報処理学会プログラミング研究会, PRO2-4, pp.25-32 (1995).

79 大森洋一, 城和貴, 福田晃, 荒木啓二郎: OMT 法による並列化コンパイラ中間言語フレームワークの構築, 情報処理学会プログラミング研究会, PRO2-5, pp.33-40 (1995).

80 丸川一志, 城和貴, Craig, D., Polychronopoulos, C., 福田晃: HTG の最適化手法への適応に関する考察, 情報処理学会ハイパフォーマンスコンピューティング研究会, HPC58-1, pp.1–8 (1995).

81 Joe, K. and Fukuda, A.: Analytic Modeling of Cache Coherence Based Parallel Computers, 情報処理学会数理モデル化と問題解決研究会, MPS3-3, pp.13-18 (1995).

82 中西恒夫, 城和貴, Polychronopoulos, C., 福田晃, 荒木啓二郎: 整数計画問題としてのループ並列実行時間の下限算出問題, 情報処理学会数理モデル化と問題解決研究会, MPS5-3, pp.13-18 (1996).

83 中西恒夫, 城和貴, Polychronopoulos, D., 福田晃, 荒木啓二郎: ループによって運ばれる依存を有するループの並列実行時間の見積り, 情報処理学会計算機アーキテクチャ研究会, ARC117-5, pp.25-30 (1996).

84 Joe, K. and Fukuda, A.: Applying the Semi-Markov Memory and Cache Coherence Interference Model to an Updating Based Cache Coherence Protocol, 情報処理学会数理モデル化と問題解決研究会, MPS6-2, pp.2-7 (1996).