計算機アーキテクチャ 80－2
（1990．1．25）

マルチプロセッサキャッシュの実験的評価
～小規模プロトタイプ機によるシミュレーション

砂田 佳久† 相原 玲二†† 山下 雅史† 阿江 忠†

† 広島大学工学部第二類
†† 広島大学集積化システム研究センター

マルチバス・マルチプロセッサは、バスを複数設けてバス競合の起こる確率を減少させることにより性能向上をはかる構成法である。本報告では、マルチバス・マルチプロセッサ上でキャッシュプロトコルとしてドラゴンプロトコルを用いた場合の性能評価を、小規模プロトタイプマルチプロセッサUNIP上のシミュレーションにより実験的に行なった。評価は、マルチバス・マルチプロセッサをメモリバンクとバスが一対一対応になるような構成について行ない、シミュレーション手法は、各事象をヒット率などにより確率的に発生させる合成駆動法をとった。

An Experimental Evaluation of Multiprocessor Cache Protocol by Simulation on a Small-scale Prototype Machine

Yoshihisa SUNADA† Reiji AIBARA†† Masafumi YAMASHITA† Tadashi AE†

† Department of Electrical Engineering,
Faculty of Engineering, Hiroshima University
Saijo, Higashi-Hiroshima, 724 Japan

†† Research Center for Integrated Systems,
Hiroshima University
Saijo, Higashi-Hiroshima, 724 Japan

Multiple-bus multiprocessors are the architectures that provide the high performance by reducing the contention on the shared bus. This paper presents an experimental evaluation of a multiprocessor cache protocol on multiple-bus multiprocessor by simulation. The simulation is implemented on a small-scale multiprocessor UNIP and the model is the synthesis driven scheme in which various events are decided by some probablistic parameters. In the evaluation, among some possible architectures of multiple-bus multiprocessors, we focus on the one that restricts the connectivity of each bus to only one memory bank.

1. はじめに

近年のＶＬＳＩ技術の進歩により高性能プロセッサや大容量メモリを小型かつ低価格で実現できるようになった。これを背景にして複数のプロセッサからなるマルチプロセッサシステムが注目されている。マルチプロセッサシステムのなかでも、プロセッサ間の通信を共有メモリを通して行なう共有メモリ型マルチプロセッサは従来の単一プロセッサ上でのソフトウェアとの整合性がとれていることなどから、スーパミニコンとして商用化がおこなわれ[1]、次代を担う高性能ワークステーションとしても研究が行なわれている。[2-4]。

共有メモリ型マルチプロセッサでは、システムの構築が容易であることなどから結合ネットワークとしてシングルバスが用いられることが多い。共有メモリ型バス結合マルチプロセッサには、このようなシステムの構築容易性などの利点がある反面、プロセッサ数が増えてくるとバス競合を起こす確率が高くなりあるプロセッサ数以上で性能が飽和してしまうという欠点がある。この欠点を解消するアプローチとして、

1) 各プロセッサの共有バス使用頻度を出来るだけおさえる、

2) 共有バスの容量を増やす、

の二つが考えられる。

バスの使用頻度をおさえる前者のアプローチは各プロセッサが図１のようにローカルキャッシュを持つことにより実現される。バス結合マルチプロセッサでは、このようなローカルキャッシュを設けることが一般的となっているが、この際に問題となるのは異キャッシュ間での同一データに対する一貫性をいかにして保つかということである。この問題はキャッシュコンシステンシー問題 (Cache Consistency Problem)、もしくはキャッシュコヒーレンス問題(Cache Coherence Problem) と呼ばれ、その解決策として数々のキャッシュプロトコルが提案されている[5]。

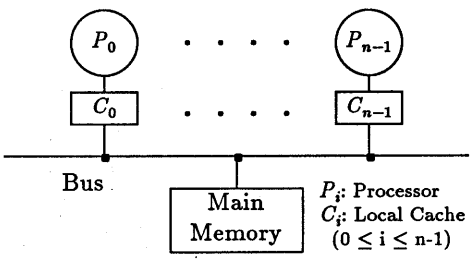


図1 ローカルキャッシュを持ったバス結合マルチプロセッサ

しかし、このようなローカルキャッシュを持ったとしてもプロセッサ数が十台を超えるとバスがボトルネックになり性能が飽和してしまうというのが現状である[5]。そこで、より高性能なマルチプロセッサを実現しようとすると前者のようにバスの使用頻度をおさえた上でバスの容量そのものを増やしてバスがボトルネックにならないようにする必要がある。

バスの容量を増やす後者のアプローチとして考えられるもののうち最も自然なものは、バスを複数にして複数のプロセッサが同時にネットワークを使用できるようにするアプローチである[6]。このアプローチをとるマルチプロセッサをここではマルチバス・マルチプロセッサ(multiple-bus multiprocessors)と呼ぶ。マルチバス・マルチプロセッサは一般に図2のようにn個のプロセッサ、 m本のバス、k個のメモリバンクから構成されるマルチプロセッサである。

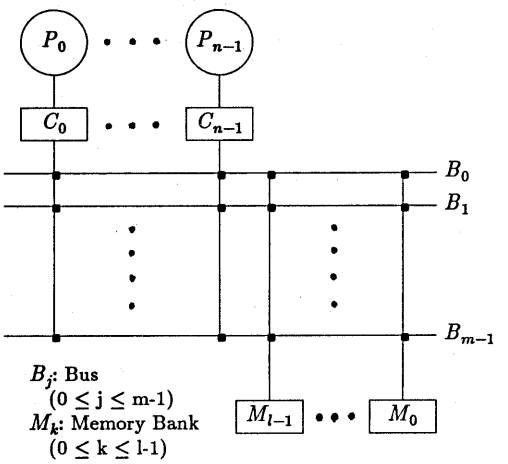


図２ マルチバス・マルチプロセッサ

従来、マルチバス・マルチプロセッサの研究は、性能解析による評価がいくつか行なわれている。これらの評価の中でもキャッシュプロトコルを考慮して評価したものは、ライトワンス・プロトコルを考慮した[7]ぐらいであり、マルチバス・マルチプロセッサの性能評価をキャッシュプロトコルを考慮して行なうことは重要と思われる。

我々は、キャッシュプロトコルとしてドラゴンプロトコルを用いたマルチバス・マルチプロセッサを小規模プロトタイプマルチプロセッサUNIP[13]上のシミュレーションにより実験的に評価する。以下、従来のキャッシュプロトコルの研究、マルチバス・マルチプロセッサについて、マルチバス・シミュレーション、

マルチバス・マルチプロセッサシミュレーションについて述べる。

2. マルチプロセッサキャッシュプロトコルに関する従来の研究

2.1. マルチプロセッサキャッシュプロトコル

従来、キャッシュコヒーレンス制御に関していくつかの手法が提案されている。それらの手法は大きく分けるとソフトウェア制御法とハードウェア制御法に分かれる。

ソフトウェア制御法[8]はプログラム中でユーザが記述するかもしくはコンパイル時に制御コードを生成することによりデータの一貫性を保つ手法である。この手法にはキャッシュにロード可能なデータをプライベートデータに限るShared Noncachable 手法や、必要に応じて共有データを含むブロックをキャッシュからフラッシュしてしまう Software Flush手法などがある。

ハードウェア制御法には、集中管理を行なうディレクトリ法と分散管理を行なうスヌープキャッシュ法がある。ディレクトリ法[9]は、主記憶の各ブロックがが現在どのキャッシュに存在しているかというディレクトリ情報を主記憶制御装置に持たせる方法でコヒーレンスを保つ手法である。この手法は多段ネットワークで結合されたマルチプロセッサで用いられており、本稿で対象としているバス結合マルチプロセッサでは用いられず、むしろ次に挙げるスヌープキャッシュ法が用いられる。

スヌープキャッシュ法[5]は、各プロセッサに付随しているキャッシュコントローラがネットワーク上を流れる情報を監視して(snoop)その情報により、一貫性を保つ手法である。スヌープキャッシュ法には、各プロセッサがキャッシュ上の共有ブロックを更新する際に他のプロセッサにそのことを放送して他のプロセッサのキャッシュ上の同一ブロックを無効化する無効型と、それを更新する更新型とがある。前者には、バークレイプロトコル[3]、イリノイプロトコル[10]などがあり、後者にはドラゴンプロトコル[5]、ファイアフライプロトコル[2]などがある。 Archibaldらは、文献[5]においてこれらのスヌープキャッシュプロトコルの性能評価をシミュレーションにより行なっている。この結果、様々な負荷に対して更新型のドラゴンプロトコルが最もよい性能を示すことが分かっている。我々はこの結果からマルチバスマルチプロセッサの評価をドラゴンプロトコルを用いて行なった。

2.2. ドラゴンプロトコル

ドラゴンプロトコルはXerox PARCで開発されたマルチプロセッサDRAGONに用いられているキャッシュプロトコルである。このプロトコルにおける各ブロックの状態は四通りで、それぞれ以下のように定義される。

1) Valid-Exclusive
同一ブロックが他のキャッシュ上には存在しない状態でかつキャッシュにロードされてから更新されてない状態

2) Dirty
他のキャッシュには同一ブロックが存在しない状態でかつキャッシュにロードされてから更新されている状態

3) Shared Clean
他のキャッシュ上に同一ブロックが存在する可能性があり、かつ他のブロックと置き換えられる際にライトバックを必要としない状態

4) Shared Dirty
他のキャッシュ上に同一ブロックが存在する可能性があり、かつ他のブロックと置き換えられる際にライトバックを必要とする状態

ドラゴンプロトコルでは、あるローカルキャッシュ上の共有ブロック(Shared Dirty もしくは Shared Cleanの状態であるブロック）に対して書き込みが起こった場合、書き込まれたワードのアドレス及び内容がバスを通して放送される。他のプロセッサでは、放送されたデータを自キャッシュが持っているかどうかを判断して持っていればそのワードを書き換える。また、同じ更新型のファイアフライプロトコルと異なり書き込み放送が行なわれる際に主記憶の内容は更新されない。

3. マルチバス・マルチプロセッサ

マルチバス・マルチプロセッサで、シングルバス・マルチプロセッサの場合に加えて考慮しなければならない点は、複数のプロセッサが、異なるバスを用いて同一ブロックを相互に書き換えることがないように制御する必要があるということである。この点を考慮するとマルチバス・マルチプロセッサの構成法は以下のような三通りになる。

第一の構成法は、図２でバスの調停を行なう前に、各ブロックが含まれるメモリバンク毎に調停を行ない、複数のプロセッサが同時に同一ブロックに対して参照することのないように構成する方式である。第二の構

成法は、図3のように書き込みバスをある一本のバスだけに制限して、書き込みを行なう際他のバスが使用されないように構成する方式である。第三の構成法は、図4のように各メモリバンクを各バスに一対一対応するように構成し、それに合わせてローカルキャッシュも複数のモジュールで構成する方式である[11]。

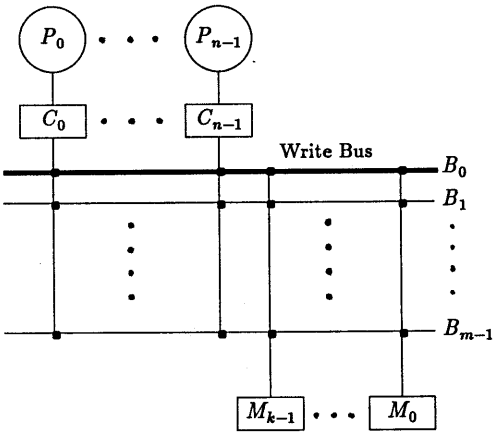
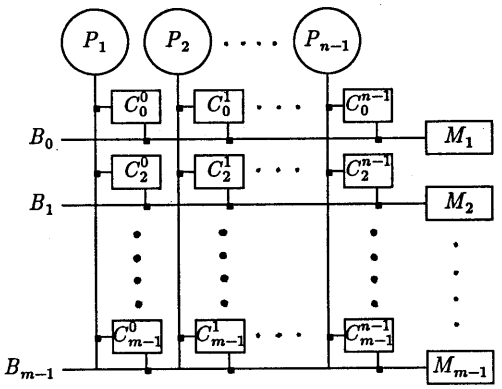


図3　書き込みバスを制限したマルチバス・マルチプロセッサの構成法



C_j^i: Cache Memory Bank of P_i for B_j

図4　メモリバンクとバスを一対一対応させたマルチバス・マルチプロセッサの構成法

以上の三つの方式を比較すると第一の構成法は、バス選択に自由度があるが調停回路を構成するためのハードウェア量が多くなるという欠点がある。また、コヒーレンス制御としてスヌープキャッシュプロトコルを採用した場合、キャッシュコントローラが、全てのバスを統括して管理する必要がありこの点でもハードウェアが複雑化する。第二の構成法は、バス構成が簡単で書き込みバス以外ではバスを自由に使えるので書き込みが多くないときは有利であるが、書き込みが多くなると書き込みバスにバスのトラフィックが集中してボトルネックになる。第三の構成法は、バス選択が限定されるという欠点があるが、各バスの管理を独立して行なうことが出来るのでシングルバスマルチプロセッサで用いられた技術を応用することが容易であるという利点がある。本報告では、第三の構成法でキャッシュプロトコルとしてドラゴンプロトコルを採用した場合の評価を行なう。

4. マルチバスネットワークの評価

マルチバス・マルチプロセッサの評価を行なう前に、マルチバスのネットワークとしての特性を見るためにマルチバスネットワークの評価を行なった。マルチバスネットワークのバスプロトコルとしては、シングルバスのCSMA / CD方式をマルチバスに拡張したプロトコル[12]を用いた。評価は本研究室で開発されたマルチプロセッサUNIP上でシミュレーション実験により行なった。

4.1. マルチバスプロトコル

評価したマルチバスネットワークは図5のようにn個の局と m本のバスからなるネットワークで各局は全バスに接続されている。

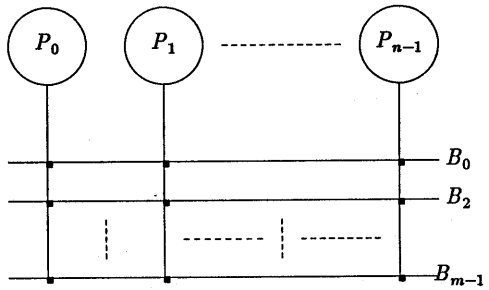


図5 マルチバスネットワーク

本稿ではマルチバスネットワークを以下のようなプロトコルで評価した。

1) パケットの送信を試みる前に全てのバスをセンスする。

2) 1)の結果アイドルなバスがあれば、その中から1つをランダムに選択してパケット送信を試みる。また、1)の結果アイドルのバスがなければ1-persistent方式によりどこかのバスがアイドルになるまで待つ。

3) 2)で衝突が検知された際は、バイナリ・エクスポーネンシャルプロトコルによりランダム時間待って、その後1)から繰り返す。

4.2. シミュレーションモデル

シミュレーションモデルの概要は以下のとおり。

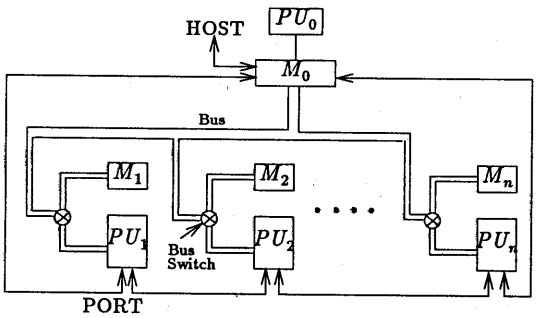
1) 各パラメータ
γ:最遠局の伝搬遅延(sec)、
L_p:パケット長(bit)、
τ:単位スロット時間
λ:パケット発生間隔（1局当たり）（スロット）

2) 時間軸を$\tau(=2\gamma)$でスロット化し、各局は、スロット毎に同期して上記のプロトコルの手順にしたがって動作する。ここで、衝突はあるスロットで2つ以上の局が同一バスに対してパケット送信を試みた際に生じるとする。

3) パケットの発生間隔は平均λの一様分布とし、1～2λのランダムな数を発生する。

4) 送信バッファ長は無限長とする。

5) パケット長L_pは一定とする。



PU_i: Processor, M_i: Memory
(i=0 Master, 1≤ i ≤ n Slave)

図6 マルチプロセッサUNIPの構成

4.3. シミュレーション

上記のモデルをマルチプロセッサUNIP[13]上に実装し実験を行なった。マルチプロセッサUNIPは図6のようにマスタ1個、スレーブ31個のプロセッサからなるマスタースレーブ方式のマルチプロセッサで、各プロセッサはZilog Z-80A(4MHz)を使用し、プロセッサ間は単一バス結合とパラレルポート結合を併用している。マルチバスネットワークのシミュレーションを行なう際は、ネットワーク上の各局をUNIP上の各スレーブプロセッサに、バスをマスタプロセッサに対応させて実装を行なった。

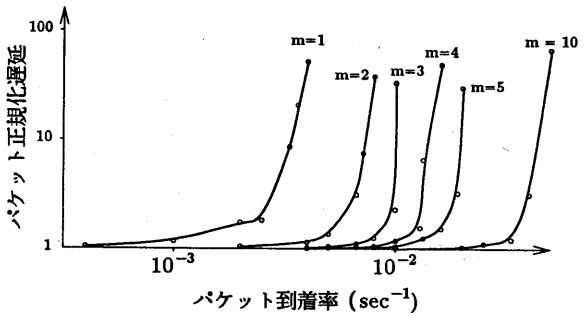


図7 マルチバスネットワークのパケット到着率対パケット長正規化遅延特性

4.4. シミュレーション結果

バスのバンド幅を100Mbps、単位スロット時間τを10^{-5}、L_pを10000bitとしてマルチバスネットワークのシミュレーションを行なった。その結果をパケット発生確率（1/λ）対パケット長正規化遅延特性として図7に示す。この図より、正規化遅延がバスの本数が増えるとそれだけ改良されているところからバスをマルチ化する効果が十分期待できることが分かる。

5. マルチバス・マルチプロセッサのシミュレーション

マルチバス・マルチプロセッサの評価をマルチバスネットワークと同様にマルチプロセッサUNIP上のシミュレーションにより行なった。ここでのシミュレーション手法としてはトレース駆動法(trace driven scheme)と合成駆動法(synthesis driven sheme)が考えられ、前者は、ベンチマークプログラムのメモリ参照をトレースしてそれをもとにしてシミュレーションを行なう手法で、後者は、メモリ参照を確率的なパラメータにより発生させてシミュレーションを行なう手法である。ここでは、システムの一般的な評価を目的としているので、後者のアプローチをとるものとする。

5.1. シミュレーションモデル

マルチバス・マルチプロセッサをモデル化する際に、プロセッサ、キャッシュ、バスを表すハードウェアモデルと、メモリ参照やキャッシュヒット／ミスを表すソフトウェアモデルの二つのサブモデルに分けて考える。

5.1.1. ハードウェアモデル

ハードウェアモデルを図8のように待ち行列を使ってモデル化した。モデルでは各プロセッサとローカルキャッシュを n個のプロセッサモジュール(PM_j, 0 ≤ j ≤ n-1)で、各バスをバスモジュール(BM_j, 0 ≤ j ≤ m-1)で表している。プロセッサモジュールは、図9のようにそれぞれプロセッサを表す処理ユニット(PU)とローカルキャッシュおよびキャッシュコントローラを表すキャッシュユニット(CU) から構成される。また、ドラゴンプロトコルのスヌープ機構をモデル化するために、処理モジュールとバスモジュールの間で情報の交換を行なうようにした。

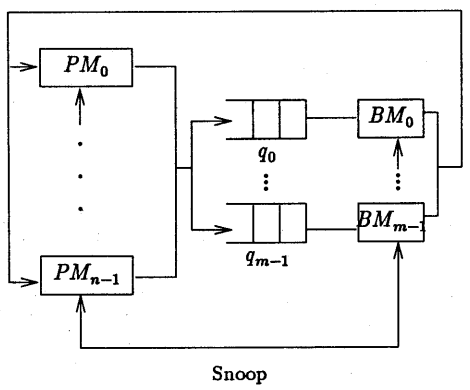


図8　マルチバスマルチプロセッサのシミュレーションモデル (全体図)

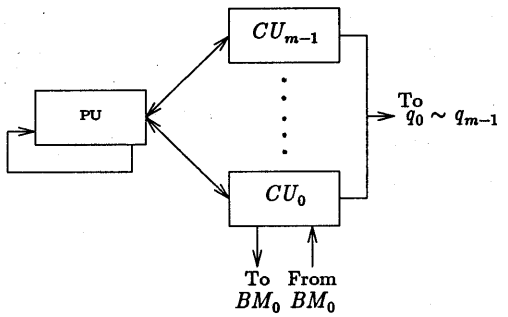


図9　マルチバスマルチプロセッサのシミュレーションモデル (プロセッサモジュール))

処理ユニットはRISC型[14]のプロセッサを仮定し、命令フェッチ(IF)、命令実行(EXE)、オペランドフェッチ(OF)の三つのフェーズから構成される2段の命令パイプラインを持っているとする。処理ユニットでは、5.1.2で述べるようなソフトウェアモデルに従ってメモリ参照が起こり、キャッシュユニットに対してメモリ参照要求を表すトークンを送り、キャッシュユニットから承認トークンを受けるまで待つ。処理ユニットからメモリ参照要求を受け取ったキャッシュユニットでは、キャッシュにヒットしているかどうか、キャッシュコヒーレンス制御のための処理が必要であるかどうかを調べ、バストランザクションを必要としない場合は、1キャッシュメモリサイクル後に処理ユニットに承認トークンを返す。バストランザクションを必要とする場合はバス使用要求をm個の待ち行列のうちのどれかに入れ、バスモジュールよりバストランザクション終了のトークンを受け取るまで待つ。また、キャッシュユニットで処理ユニットからのメモリ参照とバスライトによる書き込みが重複した際は、バスライトによる書き込みが優先され、プロセッサに割り込みが掛けられて書き込みが終了するまで待たされるとする。

バスモジュールでは、バス使用要求のトークンの内容などからバスサービス時間を決定し、その時間を消費した後承認トークンをプロセッサモジュールに送る。バス使用要求トークンは、(バストランザクションの種類、ブロックID、ブロックの属性)の三項組で表され、スヌープ機構をモデル化するためにこの三項組が全プロセッサモジュールに放送されるとする。また、バストランザクションの種類はライトミス、リードミス、ライトバック、ライトブロードキャストの4種類とする。

5.1.2. ソフトウェアモデル

ソフトウェアモデルで用いるパラメータを表1に示す。ソフトウェアモデルの特徴は、データブロックを共有データを含む共有ブロックと共有データを含まないプライベートブロックに分けてモデル化していることである。プライベートブロックに対して参照する場合は個々のブロックを意識せずに起こる事象を全て確率的に決定し、共有ブロックに対して参照する場合は個々のブロックを意識して各ブロックの状態により起こる事象を決定する。

プロセッサがメモリ参照を行なう命令は Load、Store命令のみとし、その確率を *ls* とする。ここでメモリ参照するデータが共有ブロックに含まれる確率は *shd* で、プライベートブロックに含まれる確率は *1-shd* であるとする。プライベートブロックの場合、キャッシュヒットは確率的に決まり、命令フェッチでは *1-msins* の確率でキャッシュにヒットし、オペランドフェッチの場合は *1-msdat* の確率でキャッシュにヒットする。共有データの場合は、個々のブロックを意識して

各ブロックの状態によってキャッシュヒット及びミスを決定する。この際、参照されるブロックはランダムに選択されるとする。

表1　ソフトウェアモデルで用いられるパラメータ

パラメータ	意味
ls	メモリ参照率
shd	共有データ参照率
msdat	オペランドミス率
msins	命令ミス率
md	プライベートデータ・ダーティ率
wr	書き込み確率
nshd	共有ブロック数
c_size	キャッシュサイズ(ブロック)

また、ブロックの置換を行なう際も、置換されるブロックが共有ブロックである場合と個別ブロックである場合に分けて考える。この際、置換されるブロックとして個別ブロックが選ばれる確率は、その時点でキャッシュ上にある共有ブロック数をac_blkとすると、プライベートブロックが選ばれる確率は 1 - ac_blk / c_sizeで、共有ブロックが選ばれる確率は ac_blk / c_sizeとする。プライベートブロックが選ばれた場合、メモリに対してライトバックする必要がある確率はmd、必要がない確率は1-mdとする。共有ブロックが選ばれた場合は、ブロックの状態によりライトバックが必要かどうかを決める。

5.2. シミュレーション結果

表2に示すようなパラメータでキャッシュプロトコルとしてドラゴンプロトコルを用いたマルチバス・マルチプロセッサの性能評価を行なった。バスの本数、及びプロセッサ数はそれぞれ、1－3本、1－30台の範囲でシミュレーションを行なった。図10にその結果としてプロセッサ数とシステム性能（一秒間に実行された命令数の合計[MIPS値]）の関係を示す。

この結果を、まずシステム性能が飽和するプロセッサ数について比較する。バスが1本の時にシステム性能が飽和するプロセッサ数は10程度であるが、バスが二本、三本に増えるとそれぞれ、15、25にまで伸びる。また、飽和時のシステム性能を比較するとほぼバスの本数に比例して性能が伸びていることがわかる。

次に、プロセッサ数を固定して性能向上比を比較する。バスを複数にすることによる性能向上比は、比較的少ないプロセッサ数では余り高くない。例えば、プロセッサ数が5台の時を比較すると、バスの本数にかかわらずほぼ同じ性能を示し、バスを複数にした効果はほとんど見られない。しかし、プロセッサ数が10を超えると、バスが1本の時と2、3本の時との差は顕著であり、2本、3本になると1本の時の2倍以上の性能が得られる。

表2　シミュレーション実験で用いたパラメータ値

パラメータ	値
ls	30(%)
shd	5(%)
msdat	5(%)
msins	1(%)
md	40(%)
wr	30(%)
nshd	16 (ブロック)
c_size	2K (ワード)
ブロックサイズ	4 (ワード)
プロセッササイクル	100 (n秒)
キャッシュサイクル	100 (n秒)
主記憶サイクル	200 (n秒)

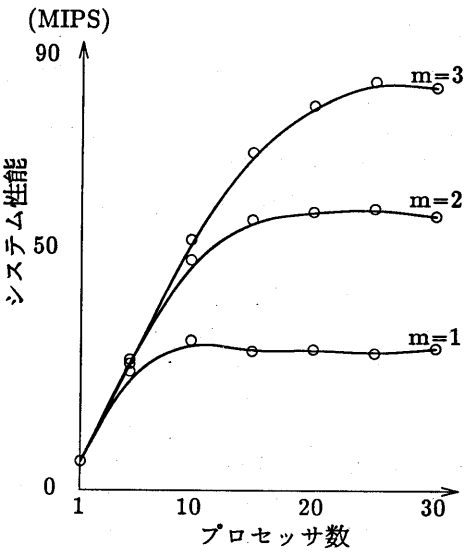


図10マルチバス・マルチプロセッサシミュレーション結果

6. おわりに

本報告では、マルチバス・マルチプロセッサでキャッシュプロトコルとしてドラゴンプロトコルを用いた場合の評価について述べた。今後の課題としては、マルチバス・マルチプロセッサの構成法およびキャッシュプロトコルに関する総合的評価などが挙げられる。

参考文献

[1] G.Fielland and D.Rodegers, "32-bit computer system shares load equally among up to 12 processors," *Electronic Design*, Sep. 6, 1984, pp.153-168.

[2] C. P. Thacker, L. C. Stewart, E. H. Satterthwaite, Jr., "Firefly:A Multiprocessor Workstation", *IEEE Trans. Computer* , vol.37, No.8, Aug.1988, pp.909-920.

[3] M.Hill, et al. "Design Decision in SPUR", *Computer, Nov. 1986,* pp.8-22.

[4] 清水、大庭、森脇、中田、小原、" 高性能マルチプロセッサ・ワークステーション TOP-1", JSPP '89, Feb. 1989, pp.155-162.

[5] J.Archibald and J.Baer, "Cache Coherence Protocols:Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. Computer and Systems* , vol.4, No.4, Nov. 1986, pp.273-298.

[6] T.N. Mudge, J.P. Hayes, and D.C.Winsor, "Multiple Bus Architecture," *Computer*, June 1987, pp.42-48.

[7] Q.Yang and L.N. Bhuyan, "A Queing Network Model for a Cache Coherence Protocol on Multiple-bus Multiprocessors", *In Proceedings of International Conference of Parallel Processing,* Aug. 1988, pp.130-137.

[8] S. Owicki, A. Agarwal, "Evaluating the Performance of Software Cache Coherence," *Proceedings of 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, April, 1989, pp.230-242.

[9] A. Agarwal, R. Simoni, J. Hennessy and M. Horowitz "An evaluation of Directory Schemes for Cache Coherence," *In Proceedings of 15th International Conference on Computer Architecture,* June 1988, pp.280-289.

[10] M. S. Papamarcos and J.H. Patel, "A Low-overhead Coherence Solution for Multiprocessors with private cache memories," In Proceedings of 11th International Conference on Computer Architecture, June 1984, pp.349-354.

[11] D.C.Winsor and T.N.Mudge, "Crosspoint cache architecture," *In Proceedings of International Conference of Parallel Processing,* Aug. 1987, pp.266-269.

[12] V.V.Karmarkar, J.G.Kuhl,"An Integrated Approach to Distributed Demand Assignment in Multiple-Bus Local Networks", *IEEE Trans. Computers*, vol. 38, No.5, May 1989, pp.679-695.

[13] 相原、阿江、" マルチマイクロプロセッサによるソート／サーチ エンジンの試作"、情報処理学会論文誌、vol.26, No.2, Mar. 1985, pp.349-355.

[14] W.Stallings, "Reduced Instruction Set Computer Architecture," *Proceedings of IEEE* vol.76, No.1, Jan. 1988, pp.38-55.