

TM-0750

プロセッサ間ソフトウェア割り込み処理を
高速化するスリットチェック機構

中川貴之, 後藤厚宏, 近山 隆

July, 1989

©1989, ICOT

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

プロセッサ間ソフトウェア割り込み処理を高速化するスリットチェック機構

中川 貴之^{*} 後藤 厚宏 近山 隆

新世代コンピュータ技術開発機構[†]

スリットチェックとは、通常のハードウェア割り込みよりも大きな粒度で多種多様なイベントを処理するソフトウェア割り込みである。ソフトウェア割り込みにより、コンテキストの save/restore を抑えるなど、柔軟な処理が実現できる。しかし、ソフトウェア割り込みには、プログラムイベントの検出のためのポーリングによるオーバーヘッド、イベント情報の送受信によるオーバーヘッドが伴う。特に、密結合マルチプロセッサのプロセッサ間通信ではこれらはキャッシュミスとしてバストラフィックの増加をもたらす。本報告は、特にスリットチェック処理のプロセッサ間通信を高速化する簡単なハードウェア機構について提案し、コヒーレントキャッシュの拡張として考察する。また、密結合マルチプロセッサにおける自動負荷分散、プライオリティ制御について、使用例をあげ、有効性を検証する。

Slit-Check Features to Speed Up Interprocessor Software

Interruption Handling

Takayuki NAKAGAWA

Atsuhiko GOTO

Takasi CHIKAYAMA

Institute for New Generation Computer Technology (ICOT)

Slit-Check is a software interruption to handle many kinds of events in larger grains compared with conventional hardware interruptions. With software interruption, we can realize flexible handlings, such that saves context manipulation. On the other hand, software interruption accompanies polling overhead to detect events, and send/receive overhead to know detailed information about events. Especially handling interprocessor communications in tightly-coupled-multiprocessors(so-called TCMP), these overheads cause increase in bus-traffics because of cache-miss. This paper proposes a simple hardware feature to speed-up interprocessor communication related to Slit-Check handling. Next, we consider this feature as a extension of a coherent cache scheme. Then we verify the efficiency of this feature using examples, such as automatic load ballancing and priority handling in TCMP, which will be installed with this feature.

^{*} JUNET:nakagawa@icot.jp CSNET:nakagawa@icot.jp@relay.cs.net UUCP:ihnp4!kddlab!icot!nakagawa

[†] 108 東京都港区 三田 1 丁目 4 28 三田国際ビルディング 21F Phone: 03(456)3193 Telex: ICOT J32964

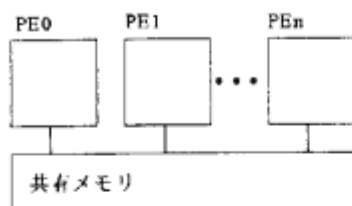


図1: 密結合マルチプロセッサ

1 背景

割り込みは計算機システム内の構成要素を有機的につなげるインタフェースとして、広く使われている。ハードウェアのタイマや周辺機器の機能拡張に際しては外部割り込み等の割り込みクラスを設けることにより対応してきた歴史も有る。しかし、並列処理環境で一つのプログラムを処理するには、プログラム実行以外に、多種多様なシステム事象が発生し、その処理がシステム性能のボトルネックともなりうる。

論理型言語のシステムを開発する過程において問題となったこの種の事象の刈り取りを、スリットチェック (Slit Check) と名付けて、高速化の検討を行った。

図1のように、共有メモリを複数のプロセッサエレメント (PE) で共有する密結合マルチプロセッサでは、ネットワークメッセージによる場合よりも早いレスポンスの処理が期待出来る。一方で、共有メモリ上での相互干渉が結合形態に関わらずオーバーヘッドをもたらす。

本報告では、一般に密結合マルチプロセッサ上で小粒度のプロセス制御を行うことを前提とするが、開発ならびに評価の基盤としては並列論理型言語 KLI (Kernel Language I) および、その処理系 PIM-FIRM、並列マシン PIM (Parallel Inference Machine) の密結合部分 (クラスタ) を仮定している¹⁾。

本報告ではまず、ポーリングや負荷分散、プライオリティ制御の中心であるスリットチェックの概念及びハードウェアについて述べる。次に、PE間通信方式を分類し、本機構を利用する具体的な例として自動負荷分散方式、プライオリティ制御方式について述べる。

2 SlitCheck

2.1 SlitCheck とは

SlitCheck は、ポーリング及びコンテキストの save/restore を抑えるソフトウェア割り込み機構である。PIM-FIRM ではリダクションというマシン命令よりも大きな粒度でのみイベントを検出してオーバーヘッドを抑える。

一方、多種多様なプログラムイベントを処理する場合、一般のマシンではハード機構としての割り込み機能を用い、プログラムマスクを ON/OFF する。この方式では、以下の不都合が生じる。

- PSW の save/restore やマスクの ON/OFF が無駄である。
- PE 間通信では Sigs (Signal Processor: 対象となる PE の肩を叩き、1 バイト程度のコマンドと、マシンによっては 1 word 程度のデータを送る) 命令²⁾ や、共有のセマフォレジスタ等の (かなり重そうな) 特殊な命令を必要とする。
- プログラマが使える割り込みクラスが少なく、やはり、ポーリングが必要。
- マシン依存のイベントハンドラとなるため、ある時は Timer 割り込み、ある時はプログラム割り込みとなり、統合した扱いが出来ない。
- マスクとイベントの対応が論理的でなかったり、マシン依存で混乱を招く。³⁾

そこで、このようなリダクション間でのみ行いうイベント処理をソフトウェア割り込みとして統合してスリットチェックと名付け、高速に処理する機構を考える。

3 PE 間スリットチェックとコヒーレントキャッシュの相性

コヒーレントキャッシュはエン트리毎に、共有の有無を状態として持つことにより、非共有エン트리への書き込みを局所化し、変更の有無を状態として持つことにより、未変更エントリの書き戻しを削減する。

コヒーレントキャッシュの一番のメリットは共有バスのトラフィックを削減するところにある。単一代入言語のように書き込み率の高い処理系には、従来のストアスルーキャッシュに比べて、書き込みによる無効化信号の削減効果が大きく、コヒーレントキャッシュは必須である。

以下の表はコヒーレントキャッシュの方式を、共有データに対する書き込み処理方式と、共有データをメモリに書き戻すタイミングから、分類したものである¹⁾。

共有時 / 書き込み時	放送	無効化
書き戻しあり	FIRELY	DRAGON
書き戻しなし	BERKLEY	ILLINOIS

¹⁾例えば、数値計算用マシンでは除算例外等を引き起こして別種の例外処理にあてる事が一般に行われている。

一方、アクセスには本質的に物理的ローカリティが高い(特定番地にアクセスする PE があまり変わらない)ものと、物理的ローカリティが本質的に低い(特定番地にアクセスする PE が頻繁に変わる)ものがある。一般にはキャッシュは不可視なのでこれらを区別出来ない。無効化と放送の基準のどちらがこれらに適合するか見てみる。

まず、物理的ローカリティが高いデータに対するバスコマンドの発行の度合いを示す。以下の例では、共有状態にある同一キャッシュブロックに対する各 PE のメモリアクセス履歴により、何回のバスコマンドが発行されるかを考える。放送タイプによると変更エントリが共有状態で残り続けるので放送自体のコストが大きくなる。

例 1) 物理的ローカリティが高いメモリアクセスに対するキャッシュの挙動

CPU コマンド	バスコマンド	
<pe0>	無効化	放送
write	(無効化)	(ワード転送)
read		
write		(ワード転送)
read		

次に、物理的ローカリティが低いデータに対するバスコマンドの発行の度合いを示す。n 台のマルチプロセッサで、本質的に全員が共有しているデータを更新すると、無効化タイプでは放送タイプの n 倍のバストラフィックが生じる。

例 2) 物理的ローカリティが低いメモリアクセスに対するキャッシュの挙動

CPU コマンド	バスコマンド	
<pe0><pe1><pe2>	無効化	放送
write	(無効化)	(ワード転送)
read	(ブロック転送)	
read	(ブロック転送)	

KEI 処理系の一般のメモリアクセスは、物理的ローカリティが高く、無効化タイプが望ましい。

他方で、スリットチェックによるプロセッサ間のデータ転送には、本質的に物理的ローカリティがなく、放送タイプが望ましい[3]。

そこで、無効化タイプのキャッシュにおける、スリットチェックのための高速化機能としてつぎのようなハードウェア機構を提案する。

3.1 スリットチェック機構

スリットチェック機構の要件として以下の 2 つがある。

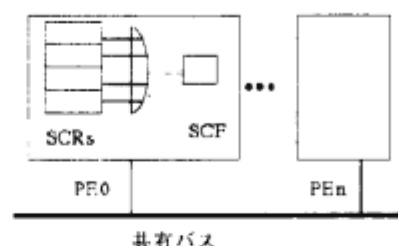


図 2: SCRs and SCF

- イベント検出時のポーリングオーバーヘッドの削減。
- 要因分析時のキャッシュミスヒットによる共有バスのバストラフィックの削減。

まず、イベントの統合環境としてそれぞれが 1 ビット幅のデータを保持する複数のスリットチェックレジスタ (SCR) を導入し、メモリ参照を削減する。また、イベント検出の高速化のために 1 ビット幅のスリットチェックフラグ (SCF) を導入し、PE 内の全てのスリットチェックレジスタのデータを OR した結果を常時保持することとする。

一般に PE 間通信は対象 PE の用途別 SCR をセットすることにより行う。SCR のデータ幅及び SCR を指定するアドレス幅が小さく出来ることから、無効化タイプのキャッシュでは実現が難しかった放送機能も容易に盛り込むことが出来る。その場合、クラスタ内の全 PE の SCR を共有レジスタのようにも set/reset 出来る。

スリットチェック機構として以下の SCR 操作命令を設ける。

ResetScrBit	SignalPosition, PEnumberReg
特定の SCR をリセットする	
ResetScrBitAll	SignalPosition
特定の SCR のリセットを放送する	
SetScrBit	SignalPosition, PEnumberReg
特定の SCR をセットする	
SetScrBitAll	SignalPosition
特定の SCR をセットを放送する	
IfSCFON	gotoLabel
イベントの検出を行う	
GetSCR	DestReg
イベントの要因分析のために全 SCR を汎用レジスタへ転送する	

以下の命令は、割り込み要因の分析に用いる。

PriorityEncoding	SrcReg, DestReg
イベントの存在を示す SCR 群の内最も若い番号を得る	

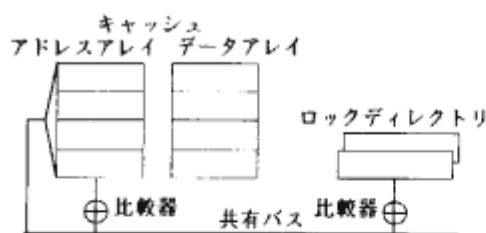


図 3: メモリのロック機構

ここで注意して頂きたいのはSCRはtest&set命令のような排他制御機構を持たない事である。PIM FIRMではSCRにより通知されたイベントとメモリに書き込んだ詳細メッセージの対応は、後述のメモリのロック機構で補うことを前提としている。

3.2 排他制御

SCRが放送機能を持つ限り共有レジスタの利用を考える必要が有る。つまり、排他制御がないと、イベントの通知と刈り取りの対応が保証出来ない。SCRに対するtest&set命令を設けるのが一番単純な結論であるが、ハードウェア機構としては重くなってしまふ。そこで、メモリのロック機構で補うことを考える。

PIMでは図3のようなメモリのロック機構を考え、以下のような命令で操作する2)。

```
Read&Lock    regsiter, address
Write&Unlock regsiter, address
Unlock       address
```

PIM-FIRMでロック機構を使う上で設けた制限事項について説明する。図3では、キャッシュのアドレスアレイにロック情報を保持せず、別にロックディレクトリを設けている。これは、一つには情報ビット数が節約出来るからであり、もう一つにはブロックアドレスよりも詳細な情報を保持出来ることにより、きめ細かなチェックが可能だからである。一方、この構成では、一つのPEからロックディレクトリのエントリ数以上のアドレスを同時にロック出来ないという制限も生じる。そこで、PIM-FIRMではこれを一般化して以下のような制限を設けた。

- ハードウェアロックはCompare&SwapやFetch&Addの形でのみ使用し、同時に2箇所以上をロックしない(6行)。

また、ハードウェアだけでロック中のアドレスに対する書き込みを防ぐには、書き込み毎に他のPEがロック中でないことをチェックする方式と、ロック中のキャッシュブロックに対し他

のPEは読みだしも出来ないようにする方式が有る。ソフトウェアの助けを借りる場合、test&setのようにロック状態をデータとして持てばチェックは出来る。より高速なハードウェアロックとしては、読みだしも出来ないようにするのが簡単である。一方、デッドロックを回避するには一般にアドレス比較を行うが、読みだしについてもアドレス比較を行うのは重い処理なので、以下の制限を設けた。

- ハードウェアロック中は原則として他のアドレスのメモリアクセスを行わない。

このようなロック機構をSCRの排他制御に使うには、

- 1) 個々のSCRと特定のメモリ番地を対応させ、
- 2) そのメモリ番地のロック中にSCRをset/resetすればよい。

但し、SCRに排他制御がないことによる問題の内、セットしたSCR情報を使う前に消してしまうケース(例えば、ゴール送信処理では、ゴールが“迷子”になるケース)は、以下の順番にすることにより回避出来る。

例1) SCRに排他制御が無くともゴールが迷子にならない方式

```
イベントのセット    イベントの刈り取り
-----
push_to_memory
SetScrBit
ResetScrBit
pop_all_from_memory
```

この場合、1 to 1の通信でも以下のような“空振り”のケースが発生する。

例2) SCRに排他制御が無い場合イベント処理が空振りするケース

```
イベントのセット    イベントの刈り取り
-----
push_to_memory
SetScrBit
push_to_memory
ResetScrBit
pop_all_from_memory
SetScrBit
ResetScrBit
(pop_none)
```

そこで、以下のようにメモリのロック機構を使うと、メモリのロック中に他のPEはSCRを変更しないので、空振りを回避出来る

例3) SCRにメモリの排他制御を流用して空振り
を無くした方式
イベントのセット イベントの刈り取り

```
-----
Read&Lock
  SetScrBit
Write&Unlock

                Read&Lock
                ResetScrBit
                Write&Unlock
```

4 PE間通信のタイプ

PIM-FIRMではPE間通信形態を以下の3つのタイプに分類する。

- 1 to 1 通信：相手先プロセッサを一つ指定したメッセージの送信
(シグナルは一つのプロセッサのみに送る)
- 1 to any 通信：クラスタ内の任意のプロセッサへのメッセージの通信
(シグナルは全プロセッサへ送り、任意のプロセッサが受信する)
- 1 to all 通信：クラスタ内の全てのプロセッサへのメッセージ通信
(シグナルは全プロセッサに送り、それらが全て受信される)

これらにおいて、同種の事象が起きた場合に望まれる処置により、対応するメモリ上に置く詳細情報の形態が選べる。それぞれの通信形態における、用途、詳細情報形態、排他制御との関連は以下になる。

4.1 1 to 1 通信

- 自動負荷分散要求、高プライオリティゴール要求、リジュームによるゴール送信処理に用いる。
- PIM-FIRMでは複数PEからゴールが来た場合、対応する共有メモリ領域は、受信PE毎にゴール環境レコードをリスト構造で保持するための受け入れ1(ポスト)とする。すなわち複数のイベントを1回のハンドリングで処理する。
- SCRの操作とメモリ操作のタイミングのずれにより、ゴールが迷子になったり、要求処理が空振りに終わるのを防ぐ為、SCRのset/resetは前述のようにメモリのロック期間内で行う。

4.2 1 to any 通信

- 自動負荷分散要求やハイプライオリティゴール要求に用いる。
- PIM-FIRMでは対応する共有メモリ領域は、クラスタ内PEで共有のビットマップ情報を保持する。ビットマップには要求の有無が0/1情報で記録される。従って、同じPEについての2回目以降の要求は無視される(出さなかったことと同じになる)。
- メモリ領域を用いた排他制御は、1要求に対し1ゴールのみを送るのに用いる。但し、メモリを更新する権利は全員にあるので空振りは回避出来ない。

例) メモリの排他制御機構を流用しても1 to any通信では空振りを回避出来ないケース
イベントのセット イベントの刈り取り

```
-----
Read&Lock
  SetScrBitAll
Write&Unlock

                Read&Lock
                ResetScrBitAll
                Write&Unlock

                Read&Lock(none)
                ResetScrBitAll
                Write&Unlock
```

4.3 1 to all 通信

- 一括GCの起動など、全てのPEに対する同期操作に用いる。
- PIM-FIRMでは対応する共有メモリ領域は、メッセージ種類をタグとして持つ、カウンタとして用いる。
- 各PEはカウンタをFetch&Decrementし、カウンタは0になるまで他の目的には使えない。

5 SlitCheckの使用例

スリットチェックではネットワークメッセージの着信に代表されるクラスタ間通信および、PE内最高プライオリティゴールの発生やメモリページ要求などのPE内部イベントもサポートしている。しかし、以下では主に、クラスタ内PE間通信に話題を絞って、自動負荷分散および、プライオリティ制御について述べる。

5.1 自動負荷分散

自動負荷分散は以下の過程を踏む。

1) 実行すべきゴールを持たないプロセッサがゴール要求割り込みを他の全てのプロセッサに送り、

2) 最初に割り取ったプロセッサがゴールを返送し要因を消す。

ゴール要求を行ったプロセッサも他の割り込みを受け付けられるように、ゴール返送をビジーウェイトしない。つまり、ゴール要求と、ゴール転送の、2種類のイベントとして処理する。また、ゴール要求はメモリに対する書き込みを伴うので1回しか行わない。また、ゴール転送はメモリの排他制御機構を使うので、1台のプロセッサのみが行う。

ゴール要求は LtoAny の通信であり、以下の2つの情報が操作される(図4)。

- 要求の存在を伝えるスリットチェックレジスタの1ビット
- プロセッサの要求状況を示すビットマップを格納するクラスタ内 PE 共通の1ワード

ゴール要求イベント送信手順は以下の通り。

a) Read&Lock: ビットマップをロックして読む

b) SetSerBitAll: 全てのプロセッサのスリットチェックレジスタをセットする

c) Write&Unlock: ビットマップの自分のプロセッサ番号のデータをセットしてロックを外す

ゴール要求イベント受信手順は以下の通り。

d) Read&Lock: ビットマップをロックして読む

e) 自分のスリットチェックレジスタをリセットする

f) 要求プロセッサが既に存在しない場合、ロックを外して処理を終える。

g) 自分が譲れるゴールを持っていない場合、ロックを外して処理を終える。

h) ResetSerBitAll: 全てのプロセッサのスリットチェックレジスタをリセットする

i) Write&Unlock: ビットマップデータをリセットしてロックを外す

j) 読み込んだビットマップをもとにゴール転送を行う

ゴール転送は LtoAny の通信であり、以下の3つの情報が操作される(図4)。

- ゴール転送を伝えるスリットチェックレジスタの1ビット

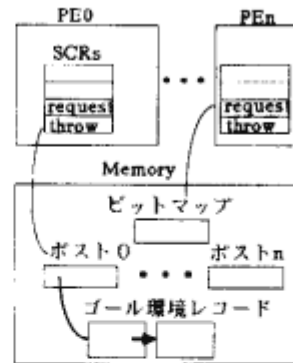


図4: 自動負荷分散方式

- 受信ゴールをスタックするポストとなる受信プロセッサ対応の1ワード
- 送信するゴール環境を保持するゴールレコード

ゴール転送イベント送信手順は以下の通り。

a) ポストをロックして読む

b) 相手のプロセッサのスリットチェックレジスタをセットする

c) 相手のポストに転送するゴールレコードアドレスをセットしてロックを外す

ゴール転送イベント受信手順は以下の通り。

d) 自分のポストをロックして読む

e) 自分のスリットチェックレジスタをリセットする

f) 自分のポストをクリアしてロックを外す

g) 読み込んだスタックの要素を順次処理する

5.2 プライオリティ制御とは

プライオリティの制御は次の理由から必要になる

- KLIで記述したOSの処理の一環としてキーボード割り込み等を速やかに反映する
- コーサプログラムのプライオリティ指定により、例えばアルファベータ枝割りにより消えそうな枝は後回しにして、無駄な処理を減らす

PE内では、図5のようなプライオリティ付きゴールスタックテーブルにより、プライオリティの大きい順番に実行する制御が実現出来る。

大前提としてクラスタ内で共有のグローバルスタックテーブルは考えない。しかし、PE毎のローカルスタックテーブルを用いても、以下の3種のデータのグローバルな更新がバストラフィック上の問題となると考えられる。

- cluster_max_priority

クラスタ内ゴールの最高プライオリティ。

- pe_max_priority(s)

PE 内ゴールの最高プライオリティ。プライオリティ付きゴールスタックテーブルにより、発生してからリダクション以内に実行に入る。

- pe_next_priority(s)

PE 内ゴールの 2 番目に高いプライオリティ。pe_max_prio ゴールが実行を終えると直ちに実行に入る。

PIM-FIRM では以下の 2 つの要件を満たすものとして、プライオリティ制御を考えている 5)。

- クラスタ内の最高プライオリティゴールが実行されずにいるのはなんとかしたい。

例 1) pe0 の pe_next_prio ゴールは pe1 か pe2 に投げたい

	<pe0>	<pe1>	<pe2>
cluster_max_prio	10	5	5
pe_max_prio	10		
pe_next_prio	10		

- クラスタ内の 2 番目以降のプライオリティゴールも面倒をみたい。

例 2) pe0 の pe_next_prio ゴールは pe1 か pe2 に投げたい

	<pe0>	<pe1>	<pe2>
cluster_max_prio	10	5	5
pe_max_prio	10		
pe_next_prio	7		

高、高プライオリティゴール要求と自動負荷分散のゴール要求はイベントとして一括に統合出来る。その場合以下のような弊害が伴うが、オーバーヘッドが増えるのみでプライオリティ制御は守れるので、問題とは考えていない。

例 3) pe0 の pe_next_prio ゴールが pe2 に投げられてしまう

	<pe0>	<pe1>	<pe2>
cluster_max_prio	10	10	5
pe_max_prio	5		
pe_next_prio	3	10	



図 5: プライオリティ付きゴールスタックテーブル

5.3 プライオリティ制御の圧力モデル

以上のような要求を満たすものとして、PIM-FIRM では圧力モデルと言う以下のパラメータを有する制御を考えている (図 6)。

- pressure : 各 PE のゴール要求及び cluster_max 変更要求の強さを表す変数。
- 圧力上限 : 圧力がこれを超えると、PE はゴール要求を出し、圧力をリセットする。ゴール要求の発行を間引く。
- 圧力下限 : 圧力がこれを超えると、PE は cluster_max を更新する。cluster_max の増加更新を間引く。
- ビットマップ : 全 PE からのハイプライオリティゴール要求の有無を表す。自分以外の全ての PE が要求を出すと、PE は cluster_max を更新する。PIM-FIRM においては、cluster_max の減少更新として、一般に全ての PE が同一プライオリティになった場合に起きると考えられる。
- Δt : 圧力を更新するサンプリングタイム。PIM の性能を 200KLIPS として、ここでは 2000 リダクション = 10msec を目安とする。更新は以下のように行う。

$$pressure += (cluster_max - pe_max)$$

cluster_max の更新までの最長時間は以下である。

$$(圧力上限 - 圧力下限) / (TrueMax - cluster_max) * \Delta t$$

要求を出すまでの最長時間は以下である。

$$(圧力上限 - 圧力下限) / (cluster_max - pe_max) * \Delta t$$

圧力上限 = 2000, 圧力下限 = -1000, cluster_max との差分 = 1 として、ゴール要求まで最長で 60 秒かかる。

期待値では圧力上限 = 2000, 圧力下限 = -1000, cluster_max との差分 = 100 として、ゴール要求までにかかる時間の期待値は 600msec。

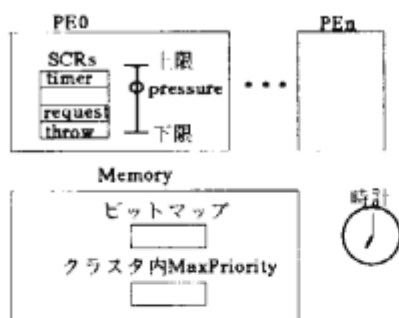


図 6 プライオリティ制御の圧力モデル

6 課題

本報告で述べたスリットチェック機構及びその応用には以下の課題がある。

- 放送タイプのキャッシュを備えたマシンでは必然性が薄い。
- プライオリティの圧力モデルによる制御はかなり粗く、しかもパラメタが多いので評価が難しい。

7 結論

スリットチェックとは、通常のハードウェア割り込みよりも大きな粒度で多種多様なイベントを処理するソフトウェア割り込みである。

本報告は、特にスリットチェック処理のプロセッサ間通信を高速化する簡単なハードウェア機構について提案し、コヒーレントキャッシュの拡張として考察した。また、密結合マルチプロセッサにおける自動負荷分散、プライオリティ制御について、使用例をあげた。

並列環境におけるプロセス制御は今後も有効性を発揮すると考えられ、更なるインタフェースの標準化および効率化が必要と考える。

8 謝辞

本報告は第5世代コンピュータの研究に基づく成果の一部である。新世代コンピュータ技術開発機構研究所の、測所長ならびに内田第4研究室長の日頃のご指導に感謝致します。

9 参考文献

- 1) Archibald and Baer: "An Evaluation of Cache Coherence Solutions in Shared-bus Multiprocessors"; Technical Report 85-10-05, University of Washington
- 2) 松本 他: "KLI のメモリ参照特性に適した並列キャッシュ機構"; データフロッワーワークショップ; 1987-10
- 3) 西田 他: "KLI 擬似並列処理系における実時間 GC 方式のキャッシュ特性の評価"; 情報処理学会コンピュータアーキテクチャシンポジウム; 1988-5
- 4) Goto et al: "Overview of the Parallel Inference Machine Architecture(PIM)"; Proceedings of the International Conference on the Fifth Generation Computer Systems 1988
- 5) Chikayama et al: "Overview of the Parallel Inference Machine Operating System(PIMOS)"; Proceedings of the International Conference on the Fifth Generation Computer Systems 1988
- 6) Stone: "Database Applications of the FETCH-AND-ADD Instruction"; IEEE Transactions on Computers vol.C-33,no.7; 1984-7
- 7) IBM System/370 Extended Architecture "Principles of Operation"; IBM manual