ICOT Technical Memorandom: TM-1257

TM-1257

並列推論マシン PIM/p における クラスタ内自動負荷分散方式の評価

畑澤　宏善、新井　進（富士通）

April, 1993

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5

Institute for New Generation Computer Technology

並列推論マシンPIM/pにおける クラスタ内自動負荷分散方式の評価

畑澤 宏善[†]　　新井 進
(株) 富士通ソーシアルサイエンスラボラトリ　富士通株式会社

概　要

並列推論マシンPIM/pのKL1処理系におけるクラスタ内自動負荷分散方式について評価した。また、性能向上のためにKL1のストリーム通信に着目した改良、PIM/p固有の命令を利用した改良について検討、評価した。その結果、ランダムな動的負荷分散ではデータの局所性が低下し、実行命令数も増加して台数効果が上がらないことが分かった。そこで、静的な解析や動的負荷分散の改良で、PE間の通信を減らすことによって負荷分散の効果を改善できることを示した。

Abstract

We evaluate the automatic load balancing method on the parallel inference machine, PIM/p. Inter-process communications through the 'stream' is improved not to use shared bus so often. It is concluded that the efficiency of a random load balancing method is not so good because of the low data locality, the large bus traffic and the increasing of instructions executed by multiple PEs. But static analysis of the KL1 programs and our improvements applied to the stream communications may better the efficiency.

1　はじめに

ICOTでは、並列論理型言語KL1を実行するための並列推論マシンPIM(Parallel Inference Machine)の開発を行っている。PIMにはアーキテクチャの異なる複数のモデルがあり、PIM/p[1, 2]は、8台の要素プロセッサ(PE)が並列キャッシュメモリを介してメモリ共有結合されたクラスタを、ネットワークで結合する二階層構成をとっている。PIM/pのKL1処理系[3]においては、KL1プログラムに内在する高い並列性を引き出すと同時にプログラマの負担を軽減するために、クラスタ内の自動負荷分散機構を実装している。

マルチプロセッサシステムにおける動的負荷分散については、これまでにもさまざまな方式の研究がなされてきた[4, 5, 6, 7]。文献[4]では、GHC処理系におけるエキストラ・キューを使った動的負荷分散方式の有効性が示唆された。また、最短経路問題[5]やOR並列型探索問題[6]など、特定の性質を持つ問題に対して有効な方式も示されている。

本研究では、PIM/pのKL1処理系に実装した自動負荷分散方式について、いくつかのベンチマークプログラムに対する負荷分散の状況をハード、ソフトの両面から分析し、改良方式の検討および評価を行う。特にハードウェアの視点からは、負荷分散と共有バスを介したプロセッサ間通信との関係について検討する。さらにこれらの結果をもとに、プログラムの静的解析による負荷分散と動的負荷分散の併用の可能性についても検討し、共有メモリを介したマルチプロセッサシステムの負荷分散のあり方につ

---

†現在: (財)新世代コンピュータ技術開発機構
Evaluation of Automatic Load Balancing Method on the Parallel Inference Machine PIM/p,
Hiroyoshi HATAZAWA [1], Susumu ARAI [2].
1: Fujitsu Social Science Laboratory Ltd. (Presently, ICOT)
2: Fujitsu LTD.

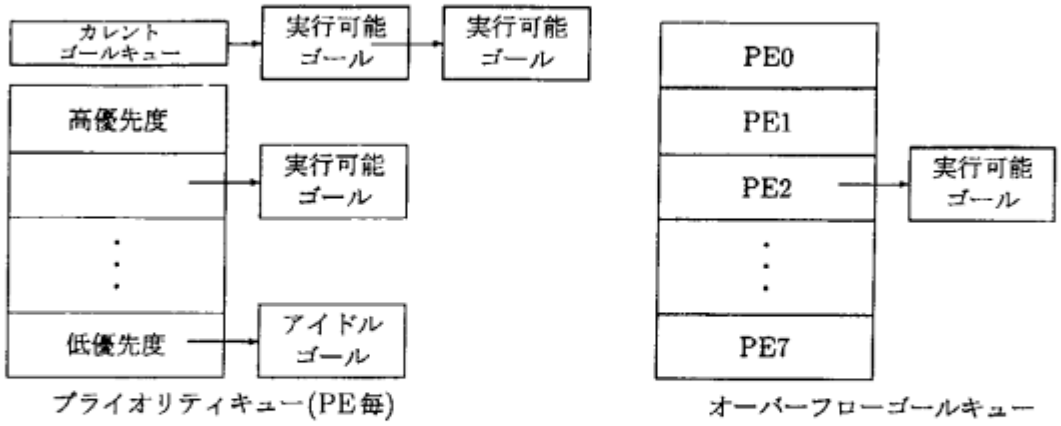


図 1: PIM/pのゴールキュー

いて考察する。

本論文では、まずPIM/pの自動負荷分散方式について説明する(2節)。3節では、現状の負荷分散効率について分析し問題点を絞る。その改善のために、ストリーム通信に注目した改良(4節)、およびPIM/p固有の命令を利用した改良(5節)について検討、評価する。最後の節でまとめを述べる。

2 クラスタ内自動負荷分散の方式

PIM/pのKL1処理系では、細粒度のプロセスであるゴール単位で負荷分散を行う。ゴールはゴールレコードと呼ばれる構造体で実現される。KL1のゴールには、ソースプログラム中でプラグマを付加することによって、実行優先度や実行プロセッサを指定することができる。ただし、実行プロセッサを指定されたゴールは自動負荷分散の対象外となるので、以下では特に触れない。

各PEは優先度別のゴールキュー(プライオリティキュー)をもっており、このうち現在実行中のゴールと同じ優先度のゴールキューは、カレントゴールキューとしてキャッシュしている。この他に各PE毎にオーバーフローゴールキューがあり、カレントゴールキューに一定数以上のゴールが溜ると、オーバーフローゴールキューに移される(図1参照)。クラスタ内の各PEのオーバーフローゴールキューは、PE間で相互にアクセス可能となっている。この方式は、文献[4]の方式を改良したもので、PE毎にオーバーフローキューを持つことによってゴールキューへのアクセス競合を起こりにくくしている。

本処理系でゴールをPE間で送受するのは、次の二つの場合である。

get-G: 実行可能なユーザゴールがないPEは、優先度最低のアイドルゴールを実行する。アイドルゴールでは、クラスタ内の他PEのオーバーフローゴールキューから、ゴールを取り出してくる。

throw-G: 実行中断(サスペンド)状態にあるゴールは、中断の要因となった未定義変数の具体化によって実行可能状態にされる(リジュームされる)。ゴールをリジュームさせたPEがサスペンドさせたPEと違う場合、そのゴールはサスペンド元のPEに投げ返される。この方式は、サスペンド元PEのキャッシュにはゴールを実行するための命令コードが乗っていると期待して採用された。

以下では、これら二つのケースをget-G, throw-Gと略記する。これらの仕組みによって、クラスタ内各PEで自動負荷分散を行っている。

表 1: 各KL1プログラムについての測定結果

プログラム	PE数	リダクション数 (x1000)	台数効果	サスペンド率	get-G 発生率	throw-G 発生率	備考
tsumego	1	250	1.00	0.186			詰め碁
	8	906	1.44	0.200	0.003	0.033	
	8*	860	2.38	0.187	0.003	0.000	
life	1	360	1.00	0.572			ライフゲーム†
	8	360	3.77	0.625	0.016	0.529	
	8*	360	5.44	0.588	0.064	0.000	
zebra	1	404	1.00	0.000			
	8	404	5.01	0.003	0.091	0.003	
	8*	404	4.48	0.008	0.123	0.000	
mastermind	1	1737	1.00	0.000			
	8	1737	7.66	0.001	0.001	0.000	
	8*	1737	7.68	0.001	0.001	0.000	

†オリジナルのライフゲームと異なり4方向からの情報のみを使う。

台数効果は、1PEで実行した場合に対する速度向上比。

サスペンド率, get-G, throw-G発生率は、1リダクションあたりの発生回数。

PE数が8* の欄は、4.2節の改良Bを施した場合の値。

3 現状の負荷分散方式の評価

3.1 ゴール実行状況の分析

始めに、規模や性質の異なる複数のKL1プログラムを使って現状の負荷分散方式について評価を行った。表1に、そのうちの4ケースについて収集したデータを示した。

KL1ゴールの実行状況から各プログラムの挙動を分析したところ、問題自体の並列度があるにも関わらず台数効果の低いプログラムには、以下のような問題のあることが分かった。

冗長な計算: KL1プログラムでゴールに優先度を指定して枝刈りを行う場合、PE毎にプライオリティキューを持っているため、クラスタ内全体では優先度が厳密に守られない。1PEでは枝刈りの対象となる計算を、8PEでは行ってしまう。この場合、PE数の増加に応じてリダクション数も増加する。(例えば tsumego)

サスペンドの増加: KL1プログラムを複数PEで実行した場合、自動負荷分散によってゴールが並列に実行されるため、1PEではサスペンドしなかったゴールがサスペンドする。表1に示したように、一般に8PEで実行した場合の方が1PEの場合よりサスペンド率が高い。

アイドルゴール実行コスト: アイドル状態になったPEはユーザゴールの環境を退避してアイドルゴールを実行するため、切替のコストが大きい。get-G発生率がその切替頻度の指標になり、zebraでは、これが台数効果を落す要因になっている。

ゴール送受オーバヘッド: throw-Gにともなうゴール送受の操作のオーバヘッドが大きい。lifeでは、これが台数効果を落す要因になっている。

PE数が増すとこれらがオーバヘッドとなり実行命令数を増して、台数効果を低下させる。

さらに、表1にあげたプログラムのうち台数効果の低いlifeについてプログラムの挙動をハードウエアの面からも分析した。その結果、8PEで実行した場合の性能低下の要因は、表2のとおりであった。この表より、1命令を実行するのに要したクロック数(以下CPIと略す)の増加が、台数効果低下の最大の原因になって

表 2: lifeプログラムの性能低下の要因

アイドル時間	3%
実行命令数の増加	22%
CPI(1命令実行当たりのクロック数)の増加	75%

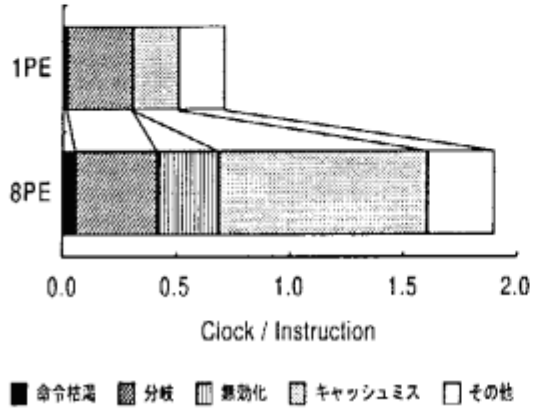


図 2: lifeプログラムのCPI増加の要因

いることがわかる。図2には、このプログラムを1PEで実行した場合と、8PEで実行した場合のCPI増加の状況を要因ごとに示す。

このグラフより、CPI増加の大半は、並列キャッシュの無効化ペナルティ(複数のPEで共有されている可能性のあるキャッシュブロックへの書き込みに伴い、他PEの該当ブロックを無効化するのに要した時間)とキャッシュメモリのミスペナルティによるものであることが分かる。この現象は、以下のように説明される。つまり、lifeプログラムでは、メッシュ状に配置されたゴール間でデータをやりとりするが、現在の自動負荷分散方式では、各ゴールのPEへの割り付けは、ほぼランダムに行われている。そのために、ゴール間の通信がPE間の通信になる可能性が非常に高くなり、データを受けるPEでのキャッシュヒット率が悪化する。また、PIM/pでは、更新無効化方式の並列キャッシュメモリを採用しているため、PEを跨いだゴール間通信は、キャッシュメモリの無効化をも多発させ、そのためのペナルティも増大することになる。

3.2 ランダムな負荷分散の問題点

3.1節の結果から、PIM/pで採用しているようなランダムな自動負荷分散には次のような問題のあることが分かった。

ゴールの並列実行によってデータの同期待ちによるサスペンドが増え、リジューム時のthrow-Gオーバヘッドによって実行命令数が増える。また、頻繁にアイドル状態になる場合(get-G発生率が高い)には、ユーザゴールとアイドル状態との切替コストが高く、台数効果を低下させる。

更にハードウェアの視点からは、細粒度のプロセスであるゴールが各PEにランダムに分散されるため、ゴール間の通信がPE間の通信になり易く、

- 共有バスの負荷が高くなる
- キャッシュミスが多発する
- キャッシュメモリの無効化が多発する

のような現象が観測される。

4 ストリーム通信に着目した改良

この節では、KL1で多用されるストリーム通信に着目して負荷分散方式の改良について検討する。ストリーム通信の問題として、メッセージ待ちのためのサスペンド/リジュームの多発、ゴール間の通信がPE間に跨る場合の共有バスの負荷増大とデータ局所性の低下などが考えられるが、これらはいずれも3節で問題となった事項である。また、ストリーム通信は再帰呼出による一連のゴール実行(以下この一連のゴールを単に"プロセス"と呼ぶ)によって行われることが多いため、再帰呼出についても検討する。

なお、検証のための実験では、3.1節で取り上げたlifeを使って主な評価を行う。このプログラムは、プロセス構成が単純なため評価しや

表 3: lifeプログラムに対する各改良方式の効果

	台数効果	性能向上	Suspend 率	Bus 使用時間
改良前	3.77	1.00	0.62	303
改良 A	4.97	1.51	0.62	113
改良 B	5.44	1.44	0.59	180
改良 C	2.85	1.03	0.40	249
改良 D	4.27	1.20	0.63	251
改良 E	4.10	1.01	0.63	288

* 台数効果: 同一の改良を施して 1PE, 8PE で実行した場合の速度比
* 性能向上: 8PE で実行した場合の、改良前後の速度比
* Bus 使用時間: 単位は 100万クロック

すく、メッシュ状に配置されたノード間でストリーム通信を行うため典型的な結果が得られると期待される。

4.1 静的プロセス割り付け

ストリーム通信を多用するKL1プログラムでデータ局所性を増して台数効果を上げるためには、頻繁に通信を行うプロセス同士を同じPE上に割りつけることによって、プロセス間の通信がなるべくPE間に出ないようにするべきである。このための静的なプロセス割り付けとしては、以下のような方法が考えられる。

[改良A] コンパイル時(あるいはプログラム時)の解析によって、生成するプロセスをあらかじめグルーピングし、頻繁に通信を行うプロセス同士は同じPEに割り付ける。

lifeにおいて、プログラム中で静的にプロセッサを指定した場合の台数効果を表3の改良Aの欄に示した。この例では、lifeプログラムで生成するメッシュ状に結合されたノードを2×4のブロックに分割し、各ブロック毎にPEに割り付けることによってPE間に出る通信を少なくした。この結果、共有バス使用時間は6割以上減少し、台数効果は約1.2ポイント向上した。また、実行速度も改良前の1.5倍と大きく向上している。

このような割り付けができれば、理想的な台数効果を得られると期待できるが、一般に推論問題等の不規則な問題ではこのような解析を、コンパイル時(あるいはプログラム時)に行うことは難しい。

4.2 動的プロセス割り付け

改良Aのような静的割り付けが難しい場合でも、動的な手法によってデータ局所性を高める方法がある。

[改良B] サスペンドしているゴールがストリームからのデータ到着によってリジュームされた時に、このゴールをリジュームしたPE上で実行する。

lifeによる実験結果を、表3の改良Bの欄に示した。共有バス使用時間は、改良前に比べて4割程度減少し、台数効果は 1.7 ポイント向上した。台数効果が改良Aの結果よりも良くなっているのは、改良Aでは1PEの場合にも実行速度が向上したためである。8PEでの性能を比較すると、改良Aに比べて若干劣るが、静的な解析が出来ない場合にも動的な割付けを工夫することによって、かなりの程度台数効果を上げられることが示唆された。

現在の処理系では、オブジェクトコードの局所性を優先して、変数の具体化によってゴールをリジュームする際に、そのゴールをサスペンドさせたPEに戻すという方式を採用している。改良Bでは、オブジェクトコードの局所性よりも、ゴール間で通信するデータと、ゴールレコードの局所性が優先される。lifeのように、すべてのノードが同じコードを実行している場合には、改良Bによって大きくデータの局所性が増し、台数効果を高めることが出来る。さらに、その他のベンチマークの例でも改良Bを施すことによって、性能の向上したものが多かった。(表1参照)

この方式の利点をまとめると:

- 初期にランダムな負荷分散を行っても、動的な処理によってデータの局所性を高めることが可能である。
- 3.1節で述べたリジューム時の throw-G オーバヘッドを0にする。これによって、複数PEで実行した場合の実行命令数の増大を押えることが出来る。

しかし、この方式には次のような問題もある。ストリーム通信型以外のプログラムで、例えば、ある一つのデータで多数のプロセスが一斉同期をとるような場合には、リジュームを行った1PEにゴールが溜り過ぎる。この場合には、その他のPEでは、get-Gが多発し台数効果が落ちる場合がある。また、この改良Bを行った結果、lifeでは1世代を計算する毎に各ノードの処理を実行するPEが変わっていくという問題も生じた。この場合には、PE間のストリーム通信は減少しても、ゴール自体がPE間を移動することによって共有バスの負荷を十分に下げることが出来ない。この問題については、4.4節で触れることにする。

4.3 サスペンド回数の削減

ストリーム通信のメッセージ待ちなど、KL1における同期機構の実現手段であるサスペンド／リジューム方式の改善手法として、大きく二つに分けて、サスペンド／リジューム時のコスト削減と、サスペンド回数そのものの削減とが考えられる。

コスト削減は、基本的には処理系のチューニングによって行うべきだが、先に述べた、4.2節の改良Bの方式でもリジューム処理のコスト削減につながる。

サスペンド回数の削減については、次のように考えられる。

[改良C] この改良は、基本的にプログラムの静的解析に依存する。つまり、コンパイル時にゴール実行順を操作するとか、文献[7]や[8]に述べられているプログラム変換(ソースコードレベルでサスペンドしない方向に書き換えてしまう)のような手法を採り入れるべきである。

特にKL1では、ゴール実行優先度のプラグマ指定を行うことによって、ゴールの実行順序を制御することが可能である。

lifeプログラムにおいて、1世代毎に実行優先度を下げる指定を行い、ある世代の計算が全ノードで終るまで次の世代の実行を始めないようにして負荷分散の状況を調べてみた。結果を、表3の改良Cの欄に示す。サスペンド率が2/3に下がったにも関わらず、台数効果が改良前よりも悪くなっているのは、1PEで実行した場合にもサスペンド率が下がって性能が大幅に向上したためである。また、優先度を指定したゴール操作のオーバヘッドがあるため、8PEの実行速度での性能向上も改良前に比べてわずかであった。

これらは直接的には負荷分散の範囲ではないが、負荷分散によってサスペンド率が増えることを考えると、間接的に台数効果の向上に寄与すると思われる。

4.4 再帰呼出処理の改良

ストリーム通信を受けてメッセージを処理するタイプのプロセスは、KL1の再帰呼出として記述されることが多く、このようなプロセスの寿命は一般に長い。このようなプロセスがPE間を移動すると、データの局所性が落ち、共有バスの負荷を高める(4.2節参照)。

一般に、再帰呼出によるプロセスの寿命と負荷分散の関係は、以下のように考えられる。

- **負荷分散するべきでないプロセス**: 寿命の長いプロセスはオブジェクトコードおよびデータの局所性を高めるために、一度あるPEに割り付けた後、他のPEに移動(分散)するべきではない。
- **負荷分散の好ましくないプロセス**: 寿命の長いプロセスから派生する寿命の短いプロセスは、派生元と同じPEで実行した方が、データの局所性を高める。このようなプロセスを頻繁に負荷分散すると、データ局所性が悪くなるばかりでなく、get-G発生率が高まり実行命令数も増大する。
- **負荷分散の好ましいプロセス**: 途中から派生してくる寿命の長いプロセスがあれば、それを他のPEに割りつけることによって負荷分散効果を増し得る。ただし、この場合でも派生元との通信量が多い場合には、PE間通信の問題が生じる。

[改良D] そこで、寿命の長いプロセスを他PEに負荷分散させないために、再帰呼出を行うゴールを自PEに割りつけるという実験を

```
List 1
 dispatch([message(M)|S],Stat) :- true |
     m_handler(M,Stat,Stat2),        % (A)
     dispatch(S,Stat2).              % (B)
 dispatch([],_) :- true | true.

List 2
 make_node(N,Stream) :- N > 0, N1 := N-1 |
     Stream = [S1|S2],
     node(N,S1),                     % (C)
     make_node(N1,S2).               % (D)
 make_node(N,Stream) :- N =:= 0 |
     Stream = [].
```

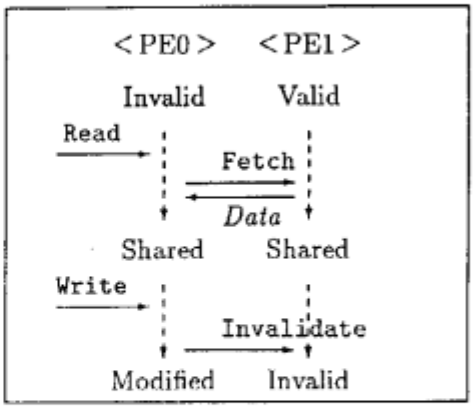
図 3: 再帰呼出の例

行った。結果を表3の改良Dの欄に示す。改良前に比べて、バス使用時間が減ったが、改良AやBほどではなかった。8PEの台数効果で0.5ポイント、実行速度で1.2倍の向上が見られたが、これらは主にget-Gに伴う負荷分散オーバヘッドの減少が原因と考えられる。

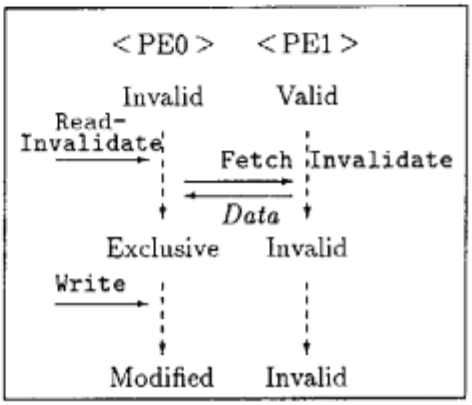
今回の実験では、動的な判定によって一律に自PEのゴールキューに再帰呼出ゴールを入れたが、実験の結果、再帰呼出を行うゴールはその種類により2通りに分類でき、それぞれ以下のように実行されるのが望ましいと分かった。

図3のList 1に示すような、ストリーム通信型のゴールでは、メッセージ処理のためのプロセス(A)を優先的に実行する。次のメッセージを受けるための再帰呼出(B)は自PEのゴールキューに入れることで、キューから取り出されるまでに、次のメッセージが到着していることを期待できる。

一方、KL1プログラムの初期段階で実行されるネットワーク生成型のゴール(List 2)では、生成される子プロセス(C)は一般に寿命が長く負荷分散の対象にすべきである。それよりも、再帰呼出によるプロセスの生成(D)を優先的に行う。PIM/pの自動負荷分散方式では、新たに生成されたプロセスがこの間にゴールキューに溜り、これがオーバーフローゴールキューへと移されて他PEへと負荷分散される。



(a) 純粋な更新無効化方式



(b) Read-Invalidate命令を利用した場合

図 4: 並列キャッシュメモリの状態遷移

5 PIM/p固有命令の利用

この節では、PIM/pに固有の命令を使うことによって、キャッシュメモリ無効化を削減する方式を検討する。

3.1節の測定では、同一アドレスに対するread & writeが頻繁に行われていることが分かった。このパターンで、最初のreadがミスし、他PEのキャッシュメモリからデータが供給されると、純粋な更新無効化方式の並列キャッシュでは、図4(a)のようにキャッシュメモリの状態が遷移する。この場合、read命令でキャッシュミスペナルティを、write命令で無効化ペナルティを要する。

一方、PIM/pでは、read-invalidate命令を

用意し、キャッシュブロックを排他的に読み出すことができるので、図4(b)のような状態遷移が可能である。これによって、write命令での無効化ペナルティを省ける利点がある。

[改良E] そこでread & writeのパターンをread-invalidate & writeに置き換えることで、無効化ペナルティをどれくらい削減できるか調べた。結果を表3の改良Eの欄に示す。ここまでの他の改良に比べると効果は小さいが、共有バス使用時間が改善された。台数効果の向上は0.3ポイントだったが、オブジェクトコードに分岐命令を埋め込んで実験したため、性能の向上は小さかった。KL1処理系やKL1コンパイラの出力するコードで、始めからread-invalidate命令を使えば効果が大きくなると期待できる。

6 結論

並列推論マシンPIM/pの自動負荷分散方式を評価した結果、現状のランダムな負荷分散では3.2節に示したような問題によって、台数効果が上がらないことが分かった。

そこで、KL1で多用するストリーム通信に着目し、データの局所性を高めるための改善方法を検討した。4.1節のようなプロセス割り付けが出来れば理想的だが、4.2節の改良Bでもそれに近い効果が得られた。また、静的解析によってサスペンド回数を減らす工夫や、寿命の長い再帰呼出ゴールを負荷分散の対象としないことも台数効果の向上につながった。これらの改良は、ストリーム通信の一般性から、共有メモリを介したマルチプロセッサシステムの負荷分散において有益なものと考えられる。

さらに、PIM/p固有の命令であるread-invalidate命令を使うことによって、キャッシュメモリの無効化ペナルティを削減できることが分かった。

今後は、より大きなプログラムを対象に、負荷分散の改善手法について調べたい。

謝辞

本研究の対象となった自動負荷分散方式の基本的な実装を行い、有益な助言をいただいたシャープ(株)技術本部 情報技術研究所の今井明氏、ならびにICOT、富士通、富士通SSLのPIMグループ諸氏に感謝する。

参考文献

[1] 服部, 篠木, 久門, 後藤: 並列型推論マシンPIM/pのアーキテクチャ, 情報処理学会論文誌 Vol.30, No.12, pp.1584-1592, 1989.

[2] K.Kumon et al.: Architecture and Implementation of PIM/p, In *Proc. of International Conference on the Fifth Generation Computer Systems* , pp.414-424, 1992.

[3] ICOT第1研究室編: VPIM処理方式解説書, TR-1044, ICOT, 1991.

[4] 安里, 岸本, 小沢, 細井, 林, 服部: GHC処理系における負荷分散方式の検討, 信学技法 Vol. 88, No. 156, CPSY88-46, pp. 49-54, 1988.

[5] K.Wada and N.Ichiyoshi: A Study of Mapping of Locally Message Exchanging Algorithms on a Loosely-Coupled Multiprocessor, Technical Report 587, ICOT, 1989.

[6] M.Furuichi, K.Taki, and N.Ichiyoshi: A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI, In *Proc. 2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 50-59, Mar. 1990.

[7] 瀧, 市吉: マルチPSIにおける並列処理とその評価 – 小粒度高並列オブジェクトモデルに基づくパラダイムについて –, 電子情報通信学会論文誌 Vol. J75-D-I, No.8, pp. 723-739, 1992.

[8] 久門, 平田: FGHCプログラムにおけるプロセスとメッセージの交換 – 実行スレッドに着目した効率改善手法, 日本ソフトウェア科学会第9回大会, A2-3, 1992.