プログラミング 2−12
（1995. 8. 24）

OR並列Prolog処理系のための分散管理方式による負荷分散機構のAP1000上での実現

川畑 徹[1]，内垣 雄一郎[2]，
松田 秀雄[3]，金田 悠紀夫[1]

[1]神戸大学自然科学研究科
[2]三菱電機
[3]大阪大学基礎工学部

本稿では，OR並列Prolog処理系を疎結合型超並列計算機上に構築する場合の負荷分散方式を提案し，AP1000上で実現した処理系について性能評価を行なっている．負荷分散方式にはプロセッサ台数の増加による影響を受けにくい分散管理方式を採用した．また，スコープの概念を取り入れ，その決定にfibonacci数列を用いることにより負荷状況把握の時間を抑え，効果的な負荷分散を実現した．性能評価の結果，511台のプロセッサで約30倍の並列効果があった．また，プロセッサ台数の増加に対しても逆に実行時間が長くなることはなかった．

Implementation of Dynamic Load Balancing with Distributed Control for OR-parallel Prolog on AP1000.

Toru KAWABATA[1], Yuichiro UCHIGAKI[2],
Hideo MATSUDA[3] and Yukio KANEDA[1],

[1]The Graduate School of Science and Technology, Kobe University
[2]Mitsubishi Electric Corporation
[3]Faculty of Engineering Science, Osaka University

In this paper we present a method for performing dynamic load balancing with distributed control on a loosely-coupled massively multicomputer system. In our method, each processor is responsible for load balancing. To reduce the communication overhead, we introduce scope for deciding the range of load balancing for each processor. Each scope is overlapped to others. We utilize a fibonacci progression of processor ids to decide which processors are included in a scope. Experimental results for load balancing of OR-parallel Prolog execution show that our method can achive proper load balancing on hundreds of processors in the Fujitsu AP1000.

1 はじめに

論理型言語であるPrologは，知識情報処理分野において有効なプログラミング言語である．知識情報処理は近年盛んに研究されている分野であり，これらが実用段階に入れば扱う問題は飛躍的に大きくなることが予想される．従って，論理型言語の処理系にはより一層の実行速度の向上が望まれている．

速度向上の要求に答えるものとして並列処理が研究されている．並列処理による速度向上では，解くべき問題の構造と，それを実行する並列計算機のアーキテクチャを考慮に入れた問題のマッピングを行なうことが重要である．

以上の点から，著者らは疎結合型の高並列計算機AP1000においてOR並列Prolog処理系を実装した．本稿では，本処理系における負荷分散機構ついて考察する．負荷分散機構とは並列処理を行なう各計算機の負荷（計算量）を均等化する機構であり，高効率で均等な負荷分散を行なうことが処理系の速度の向上につながる．

疎結合型の計算機システムは，密結合型の計算機システムに比べて台数拡張性に富むという利点があるが，計算機内の処理速度に比べてプロセッサ間の通信にかかる時間が大きいという問題点があり．通信コストを考慮した負荷分散を行う必要がある．

2 プライオリティ制御機構つきOR並列Prolog

2.1 OR並列Prolog

OR並列は，OR関係にある節を並列に実行するもので，バックトラックの先取りと言える．例えば，図1の例では，節qの呼びだしが並列に行なわれる．(図1)

このようにOR並列Prologでは実行中に並列実行可能なプロセスが生成されるので，実行開始時に問題を静的に部分問題に分割することはできない．そこで，実行中に負荷の均等化を行なう動的負荷分散が必要となる．

2.2 プライオリティ制御

効率の良い探索を行なうために，本処理系で

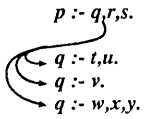

図1: PrologのOR並列実行

はプロセスの粒度や実行順序を制御する機構を備えている．

プログラム中でユーザが並列実行する節を宣言し，処理系は宣言された節についてのみプロセスを生成することにより，粒度の調節を可能にした．

また，実行順序の制御のために個々のプロセスにプライオリティという整数値を与えた[1]．ユーザはプログラムの実行中に組み込み述語によりプライオリティ値を変更することができる．処理系はプライオリティの大きなプロセスから順に実行する．

3 実行環境

本研究では疎結合高並列計算機富士通AP1000上に処理系を実現する．各プロセッサはネットワークにより結合されている．システム上の各プロセッサのメモリは独立であり，共有メモリは持たない．各プロセッサは独立にプログラムを実行し，ネットワークを介してメッセージ通信を行なう．

著者らはこれまでにもワークステーションをイーサネットで結合したマルチコンピュータ環境での並列実行について扱ってきたが[2]，AP1000では，プロセッサ間の通信に特別なネットワークを用いており，通信速度がイーサネットなどに比較して速い．従って，通信速度によって抑えられるプロセッサ台数の実用的な上限が，イーサネットを用いた時のそれと比べて緩められている．

AP1000等の疎結合型の計算機システムでは計算機内部での演算速度に対して，通信に多くの時間がかかる．従って通信量を抑えた処理系の実装を行なうことが重要である．

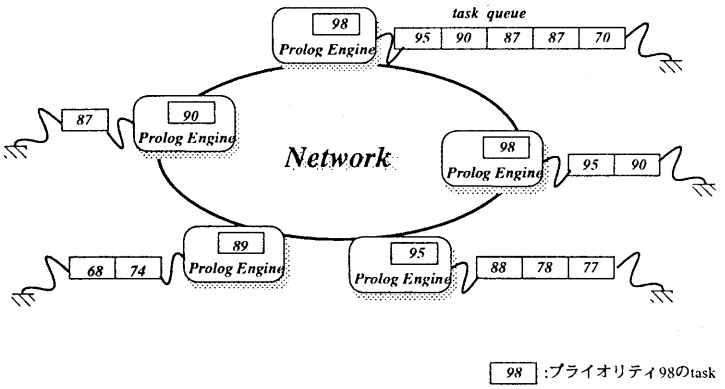

図 2: 疎結合型システム上での並列実行

4 疎結合型システムにおける負荷分散

4.1 Prolog Engine と task

本処理系では，Prolog プログラムを実行するプロセスを，**Prolog Engine(PE)** と呼ぶ．PE は，Prolog の逐次実行を行なう仮想マシンである．

PE が実行するのは task である．task は独立に実行可能な逐次的なゴールの実行である．task の実行が並列実行可能な節の呼び出しに到達すると，並列に呼び出す節の 1 つに対して 1 つの task が生成される．例えば，図 1 では，3 つの task が生成される．

PE は実行すべき task を保持するための待ち行列をもっている．この待ち行列をタスクキューと呼び，PE が実行中に生成した task はその PE のタスクキューにつながれる．PE が task をやりとりしながら，独立に task を実行することにより並列実行が行なわれる．

4.2 設計方針

負荷分散の目的は，全 PE を常に忙しく働かせることである．さらに本処理系ではプライオリティ制御を導入しているので，全体として，プライオリティの高い task から順に実行するように負荷分散を行なう必要がある．

本処理系の負荷分散機構は，各 PE が持つ局所的な負荷の情報をもとに task の移動を決定し，実行中に生じる PE 間の負荷の偏りを補正する．

本研究は，負荷情報や task の送受信にかかる通信量を抑え，実行台数を増加させた時に破綻しない負荷部分散機構を実現することを目標とする．

5 本研究で提案する負荷分散機構

5.1 集中管理方式と分散管理方式

これまでに様々な負荷分散方式が提案されてきたが [3, 4, 5]，負荷情報を管理するプロセッサに注目すると，集中管理方式と分散管理方式に分けられる．

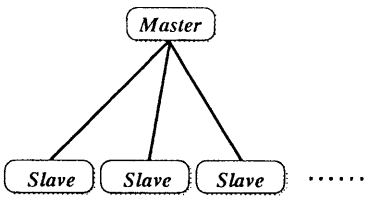

図 3: 集中管理方式

集中管理方式では，マスタという負荷情報を集中して管理するプロセッサが存在し，このマ

スタが処理系全体の負荷情報を管理し負荷分散を行なう．この集中管理方式では一つのマスタが処理系全体を見渡しているので，局所的な負荷の偏りが起こりにくい反面，マスタの処理能力が処理系の処理能力の上限を規定するため，多数のプロセッサによる並列実行には向かないと考えられる．(図 3)

マルチレベル動的負荷分散方式 [3] では負荷情報の管理を複数のプロセッサで分担して，段階的に階層化することにより，マスタのボトルネックを解消している．

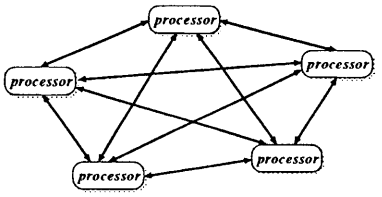

図 4: 分散管理方式

分散管理方式は，マスタが存在せず，各プロセッサが個別に負荷情報を管理し，負荷分散を行なう．プロセッサは他のプロセッサの情報を管理し，負荷の移動を決定する．拡散型動的負荷分散方式 [4] や，スマートランダム方式は分散管理方式 [5] の一種と考えられる．この方式ではプロセッサ台数が増加すると各プロセッサが負荷情報を把握するために行なう通信の量が爆発的に増加するという欠点があるが，マスタのようにボトルネックとなるプロセッサが存在しない．従って負荷情報の把握にかかる通信量を抑えることにより，プロセッサ数の増加に対応できるものと考えられる．(図 4)

本研究では分散管理方式をもとに，負荷分散機構を実現する．

5.2 スコープの概念の導入

分散管理方式の考えをもとに，負荷情報の把握にかかる通信量を抑えるためにスコープの概念を導入した．(図 5)

スコープとは PE の視野である．PE はスコープ内の PE の負荷情報をもとにして負荷分散を行なう．PE は，起動時にスコープに入れるべき PE を決定する方法を知っており，また，他のどの PE が自分をスコープに入れているかも知っている．

これらのスコープはお互いに重複する部分を持っているので．各 PE が自分のスコープに基づいて負荷分散を行なうことにより，それが相互に作用し処理系全体としての負荷分散が行なわれる．

各 PE は自分のスコープ内の負荷情報だけを知っていれば良いので，負荷情報把握のための通信量が抑えられる．

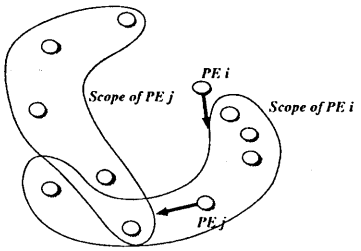

図 5: スコープ

5.3 スコープの決定法

通信量を抑えようとして PE の近傍だけをスコープに入れると，自分の回りだけ忙しく，離れた所ではあまり忙しくないというような負荷の偏りが生じる．逆に負荷の偏りをなくそうとしてスコープを単純に広げると，負荷情報の把握にかかる時間が爆発的に増加する．

スコープの決定にはスコープに入れる PE の数を抑えつつ，負荷の偏りを起こさないような工夫が必要となる．また，AP1000 では通信時間がプロセッサ間距離に依存しないという特徴を持つので，PE 間の距離を意識せずにスコープに入れる PE の選択を行なえる．

これらを考慮に入れて，本研究では自分のスコープに入れる PE の選択の方法に，fibonacci 数列を利用した．

fibonacci 数列は次のように定義される．

$$f(0) = 0, \qquad f(1) = 1$$
$$f(n) = f(n-1) + f(n-2) \quad (n \geq 2)$$

具体的には

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

といった数列である．

並列実行に使用されるPEの数は実行開始時に指定され，実行中PE台数の変更はない．実行開始時にn台のPEに1からnの整数値がidとして与えられ，PEはこれによってお互いを識別する．

各PEがスコープに入れるPEのidはfibonacci数列の各項の値に自分のidを加え，総PE台数で割った余りにより得られる．(図6)

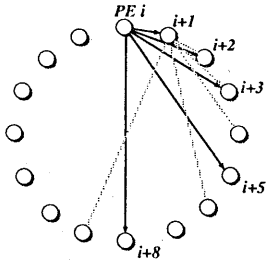

図6: スコープに入れるPEの決定

fibonacci数列を使用することの利点は次の3点である．

- fibonacci数列は始めの方の項では，各項の間が割合に詰まっているのに対して，先の方では項の間隔が急激に広がっている．この性質により，PEは自分の近くのPEには均等に負荷を分散し，離れたPEに対しても所々に負荷を送ることができる．送った先では，送り先のPEがその周辺のPEに負荷を広げてくれることを期待できる．
- fibonacci数列は，非常に早く増加し，PEを1000台用いた時でも，スコープに入れるPEの数はわずかに15である．従って，負荷情報の把握のために各PEが行なう通信量を抑えることができる．
- fibonacci数列は非常に簡単な数列である．このためPEは自分をスコープに入れたPEを知ることができ，他のPEの要求を待たずに自分の負荷の情報を送ることが出来る．

5.4 負荷分散のタイミング

本処理系では一定時間間隔で負荷分散を行なう．各PEは内部にタイマを持っており，一定の時間間隔毎に独立に負荷分散を行なう．PE間で同期を取らないので，同期にかかるコストを削減することが出来る．負荷分散の時間間隔は，ユーザが実行時にパラメタとして決定できる．

6 実行例による性能評価

実験により，本研究で提案した負荷分散機構の性能評価を行なった．本処理系において例題を実行し，実行時間と各PEが実行したtask数を測定した．また，集中管理方式や逐次処理系と実行時間を比較し，評価を行なった．

```
#para pselect/3.

?- queen_all( 10 ).

queen_all( N ) :- gen( N, L ),
                  pqueen( L, [], W, N ),
                  write( W ), nl, fail.

pqueen( S0, L, A, N ) :- =>( N, 8 ),
                         oneof( S0, X, S1 ),
                         safe( X, L, 1 ),
                         -( N, 1, N1 ),
                         pqueen( S1, [X|L], A, N1 ).
pqueen( S0, L, A, N ) :- <( N, 8 ),
                         queen( S0, L, A ).
                         pqueen( [], L, L, _ ).

oneof( [H|T], X, Others ) :- pselect( [H|T], X, Others ).

pselect( [A|L], A, L ).
pselect( [A|L0], X, [A|L1] ) :- oneof( L0, X, L1 ).

queen( S0, L, A ) :- select( S0, X, S1 ),
                     safe( X, L, 1 ),
                     queen( S1, [X|L], A ).
queen( [], L, L ).

select( [A|L], A, L ).
select( [A|L0], X, [A|L1] ) :- select( L0, X, L1 ).

safe( _, [], _ ).
safe( A, [X|L], Y ) :- +(X, Y, Z), =\=( A, Z ),
                       -(X, Y, W), =\=( A, W ),
                       +(Y, 1, S), safe( A, L, S ).

gen( 1, [1] ) :- !.
gen( Y, [Y|T] ) :- -(Y, 1, Y1), gen( Y1, T ).
```

図7: 10queen問題のプログラムリスト

6.1 Queen配置問題

組合せ問題の例として，Queen配置問題を解いた．Queen配置問題のプログラムリストを図

7に示す.

6.1.1 実行時間

本処理系では実行時のパラメタとしてPE台数と負荷分散を行なう時間間隔を設定できる. 実験ではPE台数は1, 31, 63, 95, 127, 191, 255, 319, 383, 447, 511とし, 負荷分散時間間隔を0.1秒, 0.01秒, 0.001秒, 0.0001秒とし, 全ての組合せで10queen問題を実行した. 実行結果を図8に示す. 縦軸は実行時間, 横軸はPE台数である.

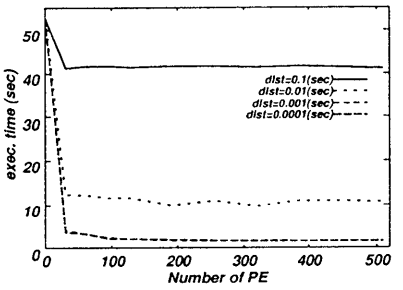

図8: 実行結果 : 10 queen 問題

負荷分散間隔が大きい時は実行時間の変化があまり見られないことから, 負荷分散時間間隔が長過ぎて負荷分散が十分に行なえないことがわかる. 一方, 負荷分散間隔を0.001秒, 0.0001秒と短くとった場合は, PE台数の増加に対して実行時間が短くなる傾向があり, 負荷分散がうまく行えていることがわかる.

負荷分散間隔0.0001秒の場合, PE31台による実行では約3.58秒となり, PE1台の場合の約52.52秒に対して約16.3倍の実行速度であった. PE511台による実行では約1.73秒となり, 約30.4倍の速度であった.

6.1.2 集中管理方式との比較

同じ10queen問題を集中管理方式で実行した. 実行結果を図9に示す.

負荷分散間隔を0.1秒とし, 31台のPEで実行した場合で約4.22秒となり既に十分な高速化がなされているが, PE台数が増加するにつれて, 処理時間が急速に大きくなっている.

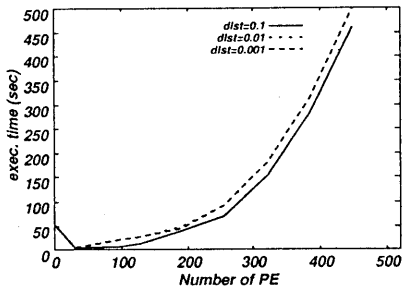

図9: 実行結果 : 10 queen 問題／集中管理方式での実行

集中管理方式では, マスタが全体の負荷情報を管理するためにPEからメッセージを受けとるが, PE台数が増加するとそのメッセージの処理にかかる時間が大きくなる. このようなマスタのボトルネックが実行速度低下の原因であると考えられる.

一方, 本研究で提案する方式では, 負荷分散間隔を0.0001秒とした場合, 集中管理方式で同じ台数のPEを使用した場合より実行時間が短くなっている. また, PE台数を増加させると, 実行時間は短くなる傾向が見られた.

このことから, 集中管理方式では問題に応じて適切なPEを与える必要があるのに対して, 本研究で提案する方式では問題を解く時点で使用可能な限りのPEを投入すれば良いといえる.

6.1.3 逐次処理系との比較

次に, Prologの逐次処理系であるSICStus Prologと本処理系との比較を行なう. AP1000の単体プロセッサ(クロック25MHzのSPARC)と性能の近いワークステーション東芝AS4060(SPARCstation 1のOEM機, CPUはクロック20MHzのSPARC)上で, SICStus Prologにより10queen配置問題を解いた.

実行時間はcompact code[1]で約20.67秒, fast code[2]では約7.56秒であった.

本処理系におけるAP1000のPE1台での実行時間は52.52秒であり, 東芝AS4060上のSIC-

[1]compact code:中間コードに変換されるコンパイルモード

[2]fast code:機械語にまで変換されるコンパイルモード

Stus Prolog の compactcode より 2.5 倍，fast code で 6.9 倍大きくなっている．このことから本処理系は単体性能においては改善の必要性がある．しかし，PE31 台以上の並列実行では本処理系による実行の方が実行時間が短くなっている．

6.1.4 負荷分散の状況

最後に，負荷が PE 全体に平均的に分散されているかを見る．図 10では，各台数において，各 PE が実行した task 数のうち最大のものをプロットしている．理想値は総 task 数を PE 台数で割った値である．

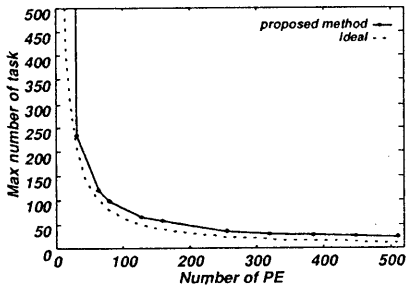

図 10: 負荷分散の状況：10 queen 問題

図から，測定値は理想値に近く，500 台を越える並列実行においても，各 PE に平均的に負荷が分散されていることがわかる．

7 結論

本稿では，疎結合型高並列計算機システム上での動的負荷分散機構について述べた．疎結合型のシステムでは，通信コストが大きいため，通信量を抑えた負荷分散機構を実現した．

本研究では分散管理方式を採用した．分散管理方式では各プロセッサが独立して負荷分散を行ない，その相互作用で全体としての負荷分散が進む．分散管理方式では集中管理方式のように全プロセッサを管理するプロセスが存在しないので，多数のプロセッサによる並列実行に向いている．

また，スコープの概念を導入し，各プロセッサの負荷分散の対象を制限することにより，通信量を抑えている．その際，スコープに入れるプロセッサは fibonacci 数列を用いて決定する．fibonacci 数列を用いることにより，近傍のプロセッサには均等に負荷を分散し，所々離れたプロセッサにも負荷を送るので全体として偏りのない負荷分散を行うことができる．

評価プログラムにより，性能評価を行なった．10Queen 問題では，プロセッサ 31 台で約 16.3 倍，511 台で約 30.4 倍の台数効果を得た，

この数字は台数効果としては大きくないが，集中管理方式ではプロセッサ台数が多過ぎると逆に実行速度が遅くなることがあるのに対し，本方式ではプロセッサ台数の増加に対して実行速度が低下することがないので，実行時に可能なだけプロセッサを使うことでその時点で最大の処理能力を発揮することを期待できるという点で優れている．

以上の結果から，本研究で提案する負荷分散方式が大規模なプロセッサ数での並列実行に有効であることがわかった．

今後の課題としては，更に大きなプログラムを実行できるようにするためにメモリ効率を改善することが挙げられる．

謝辞：本研究は富士通研究所（株）の AP1000 を用いて進められた．プログラムの作成にあたってマニュアル [6] を参考にした．

参考文献

1) 松田秀雄，鈴鹿重雄，金田悠紀夫: OR 並列 Prolog におけるプライオリティ制御機構とその応用，情報処理学会論文誌，Vol.34, No.4, pp.773–781(1991).
2) 内垣雄一郎，林彰吾，松田秀雄，金田悠紀夫: プライオリティ制御機構を有する OR 並列 Prolog における負荷分散方式とその評価，情報処理学会研究報告，PRG-18-9,1994.
3) 古市昌一，瀧和男，市吉伸行: 疎結合並列計算機上での OR 並列問題に適した動的負荷分散方式とその評価，情報処理学会研究報告，ARC79–10(1989).
4) 佐藤令子，佐藤裕幸，中島克人，田中千代治: 疎結合型マルチプロセッサ上の拡散型動的負荷分散方式，情報処理学会論文誌，Vol.35, No.4, pp.571–580(1994).

5) Sugie,M., Yoneyama,M., Ido,N. and Tarui,T.: LOAD-DISPATCHING STRATEGY ON PARALLEL INFERENCE MACHINES, *Proc. of the Int'l Conf. on Fifth Generation Computer Systems*, ICOT, pp.987–993(1988).
6) AP1000 マニュアル，（株）富士通研究所,1992.