計算機アーキテクチャ 79－10
（１９８９．１１．２１）

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

古市 昌一
三菱電機株式会社
情報電子研究所

瀧 和男，市吉 伸行
(財) 新世代コンピュータ技術開発機構

概要

疎結合並列計算機上での OR 並列型全解探索問題に適した動的負荷分散方式とその評価について述べる．負荷分散で一番重要なのは，負荷の均等化である．密結合並列計算機上ではこれまでに幾つかの動的負荷均等化方式が成功しているが，疎結合並列計算機上ではプロセッサ間の通信がオーバヘッドとなるために，動的な負荷の均等化は難しい．ここでは，OR 並列型全解探索問題一般に適用可能な動的負荷分散方式を提案する．本方式は，動的に検出した暇なプロセッサに対して仕事を割り付けるものである．また，プロセッサをグループ化してプロセッサグループに対する負荷分散とグループ内での負荷分散を行なう事により，階層的に負荷の均等化を行ない，更に階層を増やす事も可能なため，プロセッサの台数拡張性が高い．本方式を ICOT で開発した並列推論マシン・マルチ PSI/V2 上に実現し，詰込みパズルの全解探索問題に適用して評価を行なったところ，32 台プロセッサで 28.4 倍，64 台で 50 倍の台数効果が得られた．

An Efficient Dynamic Load Balancing Scheme for OR-Parallel Problems on Loosely-Coupled Multiprocessor and its Evaluation

Masakazu Furuichi
Computer Systems Development Dept.
Information Systems and Electronics Development Lab.
Mitsubishi Electric Corporation
5-1-1 Ofuna, Kamakura, 247 JAPAN
Kazuo Taki, Nobuyuki Ichiyoshi
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo, 108 JAPAN

Abstract

Good load balancing is a key to derive maximal performance from multiprocessors. Several successful dynamic load balancing techniques on tightly-coupled multiprocessors have been developed. However, load balancing is more difficult on loosely-coupled multiprocessors because inter-processor communication overheads takes more cost. We have developed a dynamic load balancing scheme which is applicable to OR-parallel programs in general. Processors are grouped, and work loads of groups and processors are balanced hierarchically. Moreover, it is scalable to any number of processors because of this multi-level hierarchical structure. The scheme is tested for the all-solution exhaustive search Pentomino program on the mesh-connected loosely-coupled multiprocessor Multi-PSI/V2, and speedups of 28.4 with 32 processors and 50 with 64 processors have been attained.

1 はじめに

並列計算機の性能を最大限に引き出すためには，負荷の均等化が最も重要である．負荷の均等化は，与えられたプログラムを互いに独立なサブタスクに分割し，各要素プロセッサ (PE) の負荷が均等となるようにそれらを割り付けることで行なわれる．負荷の均等化方式は，様々な分野でこれまでに多くの研究がなされており [1, 2, 4, 7, 10, 13]，特に密結合並列計算機においては，幾つかの動的負荷均等化方式が成功している [7]．しかし疎結合並列計算機では，プロセッサ間通信のコストが高いために動的な負荷の均等化がより難しい．

数値演算を主体とする応用プログラムにおいては，タスクの粒度やタスク間の依存関係，及び通信のパターンが実行前に予測可能な場合が多く，負荷の均等化を静的に行なう事も可能である．しかし，多くのプログラムではこれらが不可能な場合が一般的であり，動的な負荷の均等化が不可欠である．疎結合並列計算機における動的負荷均等化方式の成功例はこれまで幾つか紹介されているが [5, 6]，個々のプログラム毎に作られており，また台数の拡張性はあまり考慮されていない．

本論文は，OR 並列型探索問題一般に適用可能な動的負荷均等化方式 (動的負荷分散方式) について述べたものである．本方式はプロセッサをグループ化して，グループのレベルとプロセッサのレベルで動的に負荷の均等化を行なうもので，多階層構造のために台数拡張性がある．本方式を疎結合並列計算機マルチ PSI/V2 上で詰込みパズルの全解探索問題に適用したところ，ほぼ線形的な台数効果が得られた．

以下，第 2 章では要求駆動型動的負荷分散方式について述べ，第 3 章では本論文で提案するマルチレベル動的負荷分散方式について述べる．第 4 章では性能の測定結果とその評価を行ない，第 5 章では負荷分散を行なう際に要求されるタスクの粒度に関する一般的議論を行ない，測定結果の解析を行なうとともに，本方式を他の応用プログラムに適用する方法について述べる．最後に，第 6 章にてまとめを行なう．

2 要求駆動型動的負荷分散方式

2.1 サブタスクの生成

サブタスクの生成は，特定の PE (マスター PE と呼ぶ) 上で行ない，その様子を図 1 に示す．ここで，大きな三角形は OR 木で表わされる探索空間を示す．縦棒で塗り潰された小さな三角形はサブタスクの生成を行なうサブタスクジェネレータを示し，探索の深さがあるレベル (負荷分散 (開始) レベル) に達するまでマスター PE 上で実行される．円は生成されたサブタスクを示し，その大きさが粒度を示す．従って，サブタスクの粒度は負荷分散レベルを深くするに従って小さくなる．ただし，粒度にはばらつきがあるものと仮定する．各サブタスクは，負荷を均等化するための戦略に従ってプロセッサに

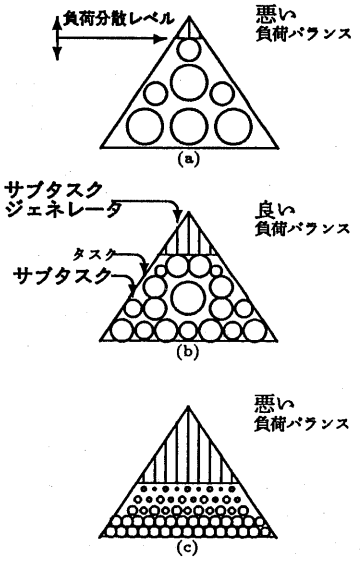

図 1: サブタスクの生成

割り付けられる．

サブタスクの粒度には，互いに反する 2 つの条件が要求される．一方では，多くの PE を全て忙しく働かせるためにサブタスクの数は多くなければならない．他方，負荷分散の手間と比較して充分粒度を大きくしなければ，負荷分散オーバヘッドにより性能が抑えられる．例えば，図 1(a) の場合は負荷分散のオーバヘッドは小さいが，サブタスクの数が少なく粒度のばらつきにより負荷は均等化されない．また，図 1(c) の場合はサブタスクの数は充分多いが，負荷分散のオーバヘッドが大き過ぎる．従って良い台数効果を得るためには，図 1(b) に示されるように，チューニングによって最適な負荷分散レベルを見つけねばならない．

2.2 サブタスクの割り付け

2.1のようにして生成したサブタスクは，各 PE の負荷が均等化するように暇な PE に割り付けられる．これは，ある PE が暇になるとマスター PE に対して新たなサブタスクの割り付けを要求するメッセージを送る事によって行なわれるので，要求駆動による動的負荷分散と呼ばれる．図 2はその方式を示したものであり，詳細は次の通りである．

1. 初期状態では全 PE は暇であり，マスター PE に要求メッセージを送る．
2. マスター PE は暇な PE にサブタスクを割り付ける．

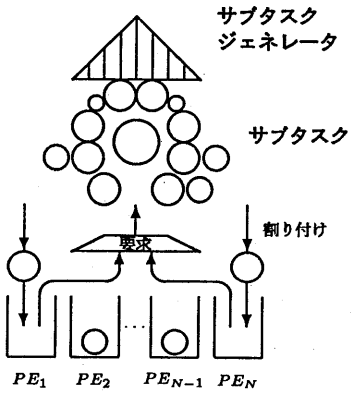

図 2: 要求駆動による動的負荷分散

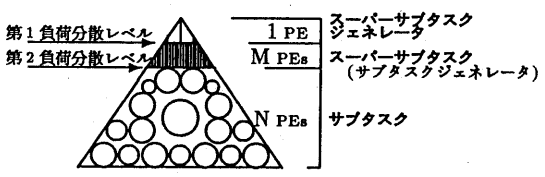

図 3: 二段階のサブタスク生成

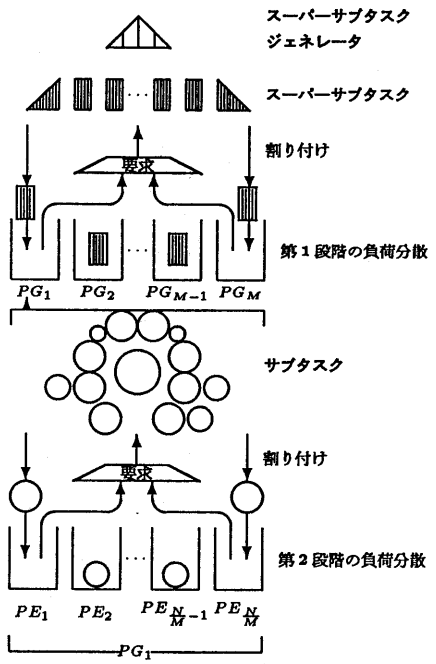

図 4: 二段階の動的負荷分散

3. 各 PE は, サブタスクの実行を終了すると新たなサブタスクを要求するメッセージを送る.

PE が暇になってからサブタスクが割り付けられるまでの間には遅延が生ずるが, これがサブタスクの平均実行時間と比較して無視できない場合には, サブタスクのダブルバッファリングを行なえば良い.

3 マルチレベル動的負荷分散方式

3.1 サブタスク生成ボトルネック

前章で示した要求駆動方式による動的負荷分散方式には, 台数の拡張性がないという問題点がある. 即ち, プロセッサの台数が増えるに従って単位時間当たりに解かれるサブタスクの数が, 単位時間当たりに生成及び供給するサブタスクの数を越えた時, サブタスクの生成がボトルネックとなる.

3.2 マルチレベル動的負荷分散方式

このボトルネックを解消するためには, サブタスクの生成を行なうプロセッサの数を増やせばよい. そのために, サブタスクの生成を図 3に示すように二段階で行ない, 負荷分散を図 4に示すように行なうマルチレベル動的負荷分散方式を提案する.

まず, スーパーサブタスクジェネレータはマスター PE に割り付けられ, 探索が第 1 負荷分散レベルに達するまでタスクをスーパーサブタスクに分割する. これを第一段階の負荷分散と呼ぶ. ここで, N 台の PE は M 個のプロセッサグループ (PG) にグループ化され, スーパーサブタスク (サブタスクジェネレータ) は PG に割り付ける. 各 PG は固定台数の PE ($\frac{N}{M}$ 台) から構成される. また, PG 中の特定の PE はグループマスター PE と呼ぶ. 第一段階の負荷分散では, スーパーサブタスクは要求駆動方式で検出した暇なグループマスター PE に割り付け, グループ間の負荷の均等化が行なわれる.

次に, M 個の PG に割り付けられたサブタスクジェネレータは, 第 2 負荷分散レベルに達するまでスーパーサブタスクをサブタスクに分割し, グループ内の PE に割り付けを行なう. これを第二段階の負荷分散と呼ぶ. 第二段階の負荷分散では, 要求駆動方式で検出した暇な PE にサブタスクを割り付け, グループ内の負荷の均等化が行なわれる.

なお, ここでは簡単のため 2 段階で負荷分散を行なった場合について説明したが, プロセッサ台数が更に多く二段階でサブタスクを生成してもボトルネックとなる場合には, 更に多段階で行なえば良い.

3.3 グループマージ

マルチレベル動的負荷分散方式は台数拡張性がある

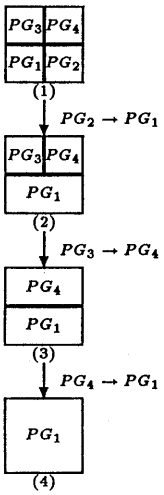

図 5: グループマージ手法

が，スーパーサブタスクの数が充分多くない時には，グループ間の負荷の不均衡が生ずる場合がある．この問題を部分的に解決するために，我々はグループマージ手法を取り入れた (図 5)．

図 5(1) は N 台の PE が 4 つのプロセッサグループに分けられている事を表わす．ここでスーパーサブタスクが 5 個あったとすると，最初の 4 個は PG_1, PG_2, PG_3,PG_4 に割り付けらる．次に PG_1 が全て仕事を終えると，5 番目のスーパーサブタスクは PG_1 に割り付けられる．この時点で 4 つの PG は全て忙しい状態である．次に PG_2 が仕事を終えて暇になると，割り付けるスーパーサブタスクがないので PG_2 は PG_1 にマージされる (図 5(2))．同様にしてグループがマージされていき，最終的には全ての PG が PG_1 にマージされる (図 5(4))．

このグループマージ手法では，グループのマージが進んでグループを構成する PE 台数が多くなるにしたがって，再度サブタスク生成がボトルネックになるという問題点がある．我々の実験例では本手法はうまく働いているが，ボトルネックとならないような手法の研究を今後行なう必要があろう．

4 性能の測定とその評価

前章で示したマルチレベル動的負荷分散方式は，OR 並列型の全解探索問題である詰込みパズルに適用して，疎結合並列計算機マルチ PSI 上で性能評価を行なった．

4.1 問題の説明

詰込みパズルは，様々な形をしたピースを長方形のケースに詰め込むパズルである (図 6)．ここでは，ピースをケースに詰め込む全ての方法を求める全解探索問題として計測に用いた．なお，このパズルはピースが全て 5 つの四角からなる時にはペントミノとして知られている．

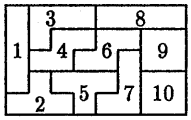

図 6: 詰込みパズル

このパズルを解くには，まずケースの隅の方のある位置にピースを置くための置き方を求める．ピースを 1 つケースに詰める事が，探索を 1 段深める事に相当する．ここで，複数の置き方がある場合にはそれぞれの置き方に対する部分探索空間は互いに独立であり，OR 木で表わされる．ピースを一つ置くと，順次その隣の空いている位置に置くための置き方を求め，ケースが全てピースで埋まったらそれが解である．また，途中で置けるピースがなくなったら，その探索枝は終了である．なお，探索のレベルを深めるにつれて OR 木の数は次第に増加する．ただし，置けるピースがなくなった時に OR 木は枝刈りされるため，ある探索レベルを境にして OR 木の数は次第に減少する．

4.2 マルチ PSI/V2 と並列言語 KL1

計測は，ICOT で開発した並列推論マシン実験機であるマルチ PSI/V2 [9] の上で行なった．マルチ PSI/V2 は，並列論理型言語 KL1 [3] を高速に処理する疎結合並列計算機で，最大 64 台の PE から構成される．各 PE は逐次型推論マシン PSI-II の CPU であり，8 × 8 の格子状の高速なネットワークで接続されている．並列言語処理系はマイクロプログラムで記述されており，1 プロセッサ当たりの append 実行性能は 130KRPS[1]である．

並列論理型言語 KL1 は Flat GHC をベースとしており，メタプログラム機構，プラグマ機構等が拡張されている．これらの機構は，OS の記述及び負荷分散の指定のために利用される．マルチ PSI の OS は PIMOS [3] で，全て KL1 で記述されており，OS の基本機能を全て提供している．なお，計測は OS 及び KL1 の計測機能を利用して行なった．

4.3 暇なプロセッサの検出方法

暇なプロセッサの検出は，KL1 が言語レベルで提供する優先度制御機能を利用した．本機能は，プログラムの実行効率を高めるために提供されているものであり，

[1]Reduction Per Second: 一秒当たりに実行されるリダクション (推論ステップ) の数であり，大雑把に言って 1 リダクションは手続き型言語の数十命令に相当する

	サブタスクの数 (N)	総リダクション数 (R) (×1,000)	サブタスクの総リダクション数 (S) (×1,000)	ジェネレータの総リダクション数 (G) (×1,000)	$\frac{G}{R}$ (%)	サブタスクの平均リダクション数 (×1,000)
L1	13	8,269	8,267	1.7	0.0	636.0
L2	118	8,273	8,267	5.4	0.0	70.1
L3	485	8,289	8,253	35.9	0.4	17.0
L4	1,583	8,321	8,201	120.6	1.4	5.2
L5	5,625	8,446	8,049	397.5	4.8	1.4
L6	16,124	8,854	7,774	1,080.4	12.2	0.5
L7	38,105	-	-	-	-	-
L8	64,980	-	-	-	-	-
L9	44,560	-	-	-	-	-
L10	3,106	-	-	-	-	-

表 1: サブタスクの数と粒度

4,096 段階の優先度を用いて細かくプログラムの実行を制御できる．ここでは，問題を解くためのサブタスクには高い優先度を与え，サブタスクの要求を行なうためのメッセージを送るタスクには低い優先度を与えた．従って，プロセッサが暇になった時にのみ優先度の低いタスクが実行される．優先度制御機能を利用した暇なプロセッサの検出方法は大変簡単であり，またインプリメンテーション上のオーバヘッドは大変小さく，また OS の機能を必要としないのを特徴とする．

4.4 サブタスクの粒度の計測

表 1 は，負荷分散レベルを変えた時のサブタスクの数とその粒度に関して計測した結果である．表中，L1~L10 は負荷分散レベルを示す．各 Ln に対応するサブタスクの数 (N) は，探索のレベルが n の時に生成されるサブタスクの数，即ち OR ノードの数を示す．L2~L10 の総リダクション数 (R) は，二段階の負荷分散 L1-Ln を行なった時の総リダクション数を示す．L1 の総リダクション数 (R) は，1 レベルの負荷分散 L1 を行なった時の総リダクション数を示す．サブタスクの総リダクション数 (S) は，生成されたサブタスクのリダクション数の総和である．ジェネレータの総リダクション数 (G) は，スーパーサブタスク及びサブタスクの生成に要するリダクション数の総和であり，$R-S$ によって計算される．$\frac{G}{R}$ は，総リダクション数 (R) 中サブタスクの生成に要するリダクション数の占める割合である．サブタスクの平均リダクション数は，生成されたサブタスクの平均粒度であり，$\frac{S}{N}$ によって計算される．

図 7 はサブタスクの粒度の分布を示すグラフである．X 軸はサブタスクの粒度 (リダクション数)，Y 軸はサブタスクの数を示す．

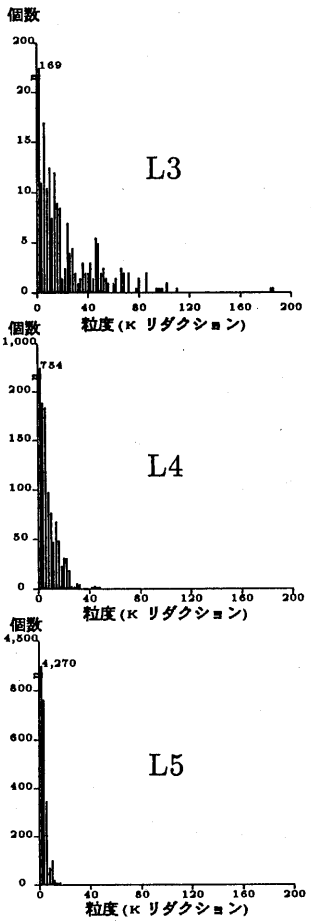

図 7: サブタスクの粒度分布

台数		1台	8台	16台	32台	64台
実行時間 (秒)	L3	260.4	36.7	22.4	16.8	13.2
台数効果	L3	1	7.1	11.6	15.5	19.7
台数効果率 (%)	L3	100	88.8	72.5	48.4	30.8

表 2: 一段階負荷分散を行なった時の実行性能

台数	1台	8台	16台	32台	64台
実行時間 (秒)					
L1-L3	259.2	33.7	18.8	10.4	7.8
L1-L4	261.2	34.2	17.5	9.2	5.3
L1-L5	262.4	37.8	18.7	9.8	5.8
L1-L6	278.4	46.9	26.5	17.8	14.2
L2-L3	260.3	34.5	19.7	10.9	6.6
L2-L4	264.8	34.4	17.6	9.4	5.3
L2-L5	265.1	37.3	18.8	9.7	5.3
L2-L6	273.7	50.8	26.7	15.5	11.2
台数効果					
L1-L3	1	7.7	13.8	24.9	33.2
L1-L4	1	7.6	14.9	28.4	49.3
L1-L5	1	6.9	14.0	26.8	45.2
L1-L6	1	5.9	10.5	15.6	19.6
L2-L3	1	7.5	13.2	23.9	39.4
L2-L4	1	7.7	15.0	28.2	50.0
L2-L5	1	7.0	14.1	27.3	50.0
L2-L6	1	5.4	10.3	17.7	24.4
台数効果率 (%)					
L1-L3	100	96.3	86.3	77.8	51.8
L1-L4	100	95.0	93.1	88.8	77.0
L1-L5	100	86.3	87.5	83.8	70.6
L1-L6	100	73.8	65.6	48.8	30.6
L2-L3	100	93.8	82.5	74.7	61.6
L2-L4	100	96.3	93.8	88.1	78.1
L2-L5	100	87.5	88.1	85.3	78.1
L2-L6	100	67.5	64.3	55.3	38.1

表 3: 二段階負荷分散を行なった時の実行性能

4.5 実行性能の計測

実行性能の計測は，64台構成のマルチ PSI/V2 上で行なった．一段階の負荷分散を行なった時の性能は，プロセッサ台数を 1, 8, 16, 32, 64 台とした時のそれぞれについて計測し，一番性能が良かった負荷分散レベル L3 のデータを表2に示す．二段階の負荷分散を行なった時の性能は，第一負荷分散レベルを L1, L2 に設定した時のそれぞれについて，第二負荷分散レベルを L3, L4, L5, L6 に設定した時の性能を測定し，その結果を表3に示す．

なお，二段階の負荷分散はプロセッサグループ (PG) の大きさを4台のプロセッサ (PE) としたが，これは 8PE の性能を計測するためである．従って，64PE の時

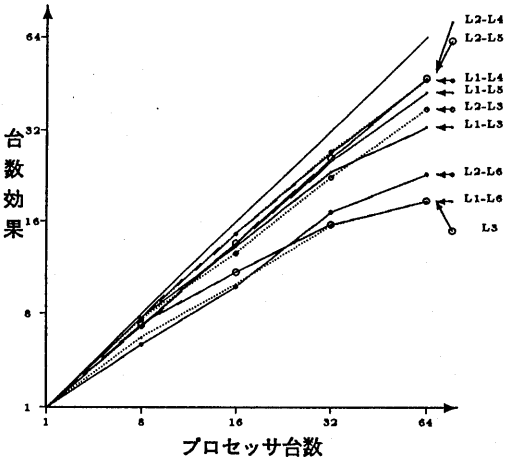

図 8: 台数効果

には 16PG から構成され，32PE の時には 8PG, 16PE の時には 4PG, 8PE の時には 2PG から構成される．また，1PE の時の性能は全てのサブタスクを同一 PG，及び同一 PE に割り付けた場合について計測した．表中の各項目は次の通りである．また，台数効果をあらわすグラフを図8に示す．

実行時間 (T_N) はマルチ PSI/V2 のシステムクロックを用いて計測した．台数効果 (S_N) は1台で実行した時の実行時間 (T_1) と N 台で実行した時の実行時間 (T_N) との比で表わされ，$\frac{T_1}{T_N}$ にて計算される．台数効果率 (SR_N) は $\frac{S_N N}{\times}100$ にて計算される．

4.6 計測結果

表1中で，負荷分散レベル L1 では 13 個のサブタスクが生成され，L8 までその数は増えて，その後 OR ノードが枝刈りされるために数は減っている．L10 の時には 3,106 個であるが，これは解の数を示している．

総リダクション数 (R) は負荷分散レベルを深くするに従って増えている．ここで，1つのサブタスクを分散するのに要するコストはほぼ一定であり，約35リダクションである．従って，サブタスクの数が増えるに従って総リダクション数も増える．

表2からわかるように，一段階の負荷分散を行なった時にはプロセッサの台数を増やすにつれて実行時間は小さくなっているが，32PE 或いは 64 PE の時には台数効果率は 50% 以下になっており，頭打ちとなっている．これは，サブタスクの生成ボトルネック或いは負荷の不均衡が原因である．この事に関しては，次章にて詳しく述べる．

表3からわかるように，二段階の負荷分散を行なった時には Li-Lj の全ての組で実行時間は小さくなってお

り，ほぼ線形に近い台数効果が得られている．64台で一番性能が良いのはL2-L4及びL2-L5の組であり，50倍の台数効果が得られている．一段階と二段階の負荷分散を比較すると，特に32台と64台の時の性能向上が著しい．なお，表中で四角で囲まれた性能は，それぞれのプロセッサ台数の時一番性能が良かったものと二番目に良かったものを示す．

5 考察

第2章で述べたように，サブタスクの数と粒度は互いに相反する2つの要求条件を満たさねばならない．一方では負荷の均等化のためにサブタスクの数は充分多くなければならず，他方では生成がボトルネックとならないように粒度は充分大きくなければならない．本章では，これらの要求条件を式とグラフを用いて明確にし，計測結果の解析を行ない，更に新たな問題が与えられた時に本方式を適用するための指針を示す．

5.1 サブタスクの数に対する要求条件

ここでは，プロセッサの平均稼働率がある値以上になる事を保証するために必要なサブタスクの数の下限を求めてみる．ただし，以下に示すようなやや乱暴な仮定を置く．

与えられたOR並列問題とプロセッサの台数は固定とする．また，暇なプロセッサがサブタスクを要求してからサブタスクが割り付けられるまでの時間は，各サブタスクの実行時間と比較すると無視できるもので，サブタスクの生成はボトルネックになっていないものとする．

ここでプロセッサの台数をN_{PE_s}とし，$N_{PE_s}-1$台の各プロセッサは平均粒度と等しい大きさのサブタスクをK個ずつ実行するものとする．残りの1台のプロセッサは，同じ粒度のサブタスクをK個実行した後，最後に最大粒度のサブタスクを1個実行するものとする．この時，平均粒度のサブタスクの個数は次式であらわされる．

$$\text{サブタスクの数} = K \times N_{PE_s} \tag{1}$$

最後のサブタスクを実行中，$N_{PE_s}-1$台のプロセッサは暇にしているので，この時に平均稼働率は一番悪くなる．

$$\text{平均稼働率} = \frac{K \times \text{平均粒度}}{K \times \text{平均粒度} + \text{最大粒度}} \tag{2}$$

この最悪ケースにおいて期待稼働率を達成する，即ち平均稼働率 $\geq$ 期待稼働率 を満たすためには，上の二式から導かれる次式のように，充分多くのサブタスクの数が必要である．

$$\text{サブタスクの数} \geq N_{PE_s} \times \frac{\text{期待稼働率}}{1-\text{期待稼働率}} \times \frac{\text{最大粒度}}{\text{平均粒度}} \tag{3}$$

ここで，期待稼働率はユーザによって与えられるものであり，最大粒度と平均粒度はプログラムに依存する．例えば64PEのシステムを考えた時，最大粒度が平均粒度の10倍大きいと仮定し，平均稼働率として80%を期待したとすると，次の条件

$$\text{サブタスクの数} \geq 64 \times \frac{0.8}{1-0.8} \times 10 = 2,560 \tag{4}$$

を満たせば平均稼働率80%が保証され，負荷分散レベルをチューニングしてサブタスクの数を決める際のガイドラインとなるものである．なお，式(3)は最悪ケースの場合に期待稼働率を保証するものであり，一般的には更にサブタスクの数を少なくしても，期待する平均稼働率を達成できる場合が多い．

5.2 サブタスクの粒度に関する要求条件

次に，サブタスクの生成がボトルネックとならないために必要なサブタスクの粒度の下限を，次のような制約条件の下で求めてみる．

サブタスクの生成に要するコスト(生成コスト)とその分散に要するコスト(分散コスト)は，マスターPEのみにかかるものとし，マスターPEはサブタスクの生成と分散のみに専念し，他のPEがサブタスクを実行するものとする．この時，サブタスク生成がボトルネックになっていなければ，N_{PE_s}台のプロセッサで$(N_{PE_s}-1)$倍の台数効果が得られる．ここで，平均粒度が(生成コスト+分散コスト)の$(N_{PE_s}-1)$倍以上大きければ，サブタスクの生成はボトルネックとならない．

$$\text{平均粒度} \geq (\text{生成コスト}+\text{分散コスト}) \times (N_{PE_s}-1) \tag{5}$$

この条件は，平均粒度のサブタスクが生成される時のものである．従って，例えば平均より小粒度のサブタスクが最初に連続的に生成されるような場合には，ボトルネックとなる場合もある．従って，平均粒度は条件式(5)よりも大きいのが望ましい．

5.3 問題の規模と台数効果

条件式(3)と(5)はそれぞれサブタスクの数の下限と上限，或いは粒度の上限と下限を与えたものである．ここでは，これらの条件を問題の規模及び台数効果の関係を示したグラフ(図9)を用いて明確にする．図中，3本の実線で示されたグラフは，それぞれ異なる規模の問題をあらわしている．

領域Aは，条件式(5)を満たしていないためにサブタスクの生成がボトルネックとなり，粒度が小さくなるに従って台数効果率が悪くなっている．この付近では3本のカーブは接近している．領域Bは，条件式(3)を満たしていないために負荷の不均衡が生じて，粒度が大きくなるに従って台数効果率が悪くなっている．この付近では3本のカーブは離れているが，これは同じ粒度であっても問題の規模が違うとサブタスクの数が違うからであり，次の関係式から明らかである．

$$\text{サブタスクの数} = \frac{\text{問題の規模}}{\text{平均粒度}} \tag{6}$$

大規模の問題の場合には，領域AとBの間に広い平坦な領域があり，その部分では高い台数効果率を得るこ

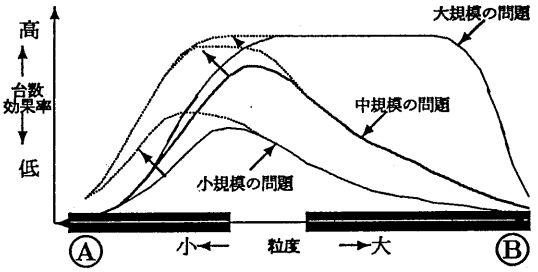

図 9: 問題の規模と台数効果率

とができる．このことは，大規模の問題の方が小規模の問題よりも負荷の均等化が易しいことを示している．中規模の問題では，領域 A と B の間は狭くなり，平坦な領域はないので，最適な粒度を見付けるのは難しい．小規模の問題では，領域 A と B はオーバーラップしており，サブタスクの生成ボトルネックと負荷の不均衡の双方が原因となって良い台数効果を得るのは難しい．

ここで，図 9 を用いて我々のマルチレベル負荷分散を考察する．本方式は，領域 A におけるサブタスク生成ボトルネックを解消したものであり，条件式 (5) 中のプロセッサ台数 (N_{PEs}) を，プロセッサグループ中のプロセッサ台数 (N_{PEs_inPG}) に置き換えたものとみなすことができる．すなわち，ボトルネックを起こさないための粒度の下限を小さくする効果がある．これは A の領域がグラフ中の左側に移動する事に対応し，図中点線の曲線のように領域 A の台数効果率のカーブを上方に持ち上げる効果がある．

5.4 計測結果の考察

ここでは，詰込みパズルの解法プログラムに条件式 (3) と (5) を適用して最適なサブタスクの数と粒度を求め，計測結果の考察を行なう．

まず，負荷の均等化に必要なサブタスクの数を条件式 (3) から求める．ここで，最大粒度は図7中の L3 のデータ (180) を用い，平均粒度は表 1の L3 のデータ (17.0) を用い，期待稼働率を 80% と決め，プロセッサ台数は 64 とすると，式 (3) から 2,710 個が求まる．表 1でサブタスクの数をみると，L5 と L6 がこの要求条件を満たしており，L4 では要求の 60% 程度のサブタスク数である．表 3をみると，L1-L5 と L2-L5 については良い台数効果が得られているが，Ln-L6 に関しては後に述べる原因で性能は良くない．また L4 はこの条件を満たしてはいないが，負荷の不均衡は表面化せずに良い性能が得られている．これは，条件式 (3) が最も厳しい場合の要求条件を与えているためである．

次に，サブタスクの生成がボトルネックとならないために必要な粒度を求める．ここで，サブタスクの生成コストは詰込みパズルの場合は約 40 リダクションで，マルチレベル動的負荷分散を適用した時に負荷分散に要するコストは約 35 リダクションである．また，プロセッサグループの大きさは 4 台とすると，必要な粒度は式 (5) から 225 リダクションである．表 1をみると，L1 から L6 の全ての負荷分散レベルでこの条件を満たしているが，Ln-L6 は表 3 からわかるように良い性能は得られていない．

L6 は条件 (3) と (5) の両者を満たしているのに関わらず良い性能を得られていないが，この原因はグループマージの問題であると考えられる．即ち，プロセッサグループがマージされて次第に大きくなるにつれて，条件式 (5) 中のプロセッサ台数は多くなる．例えば，実行が開始された当初はプロセッサ台数が 4 台であっても，グループマージが行なわれて最終的に 64 台になった時には，要求される平均粒度は 4,625 リダクションとなって L6 はこの条件を極端に満たさない．

以上の結果より，L3, L4, L5, L6 が図 9の中規模の問題のグラフ中のどこに位置するかをみてみる．まず二段階の負荷分散の適用により，A の領域はより左に移っている．L3 は負荷の不均衡を生じているので領域 B に属すると考えられる．L4 と L5 は良い性能が得られており，点線で示された平坦な領域に属すると考えられる．また L6 は，実行の初期では平坦な領域に属するが，グループマージにともなって二段階の負荷分散が一段階の負荷分散と同等になってしまうため，A の領域が右側に移動した事になり，最終的に A 領域に属するものと考えられる．

5.5 最適な負荷分散レベルの決定方法

OR 並列問題にマルチレベル動的負荷分散方式を適用する際には，最適なサブタスクの数と粒度を得るために負荷分散レベルをチューニングする必要があるが，ここではそのための指針を示す．

(a) 推測或いは実測によって問題の規模を求める．

(b) 期待稼働率を決め，条件式 (3) より要求されるサブタスクの数を求める

(c) (a) と (b) からサブタスクの平均粒度を求める

(d) サブタスクの生成コストを実測或いは推測によって求め，サブタスクの分散コスト (固定) からプロセッサグループの大きさを条件式 (5) から導く．

(e) 問題の規模が大きい場合には，条件式 (5) 中のプロセッサの数は条件式 (3) 中のものと同じか或いは大きくなるため，1 レベルの負荷分散が効果的である．

(f) $2 \leq N_{PEs}$(条件式 (5)) $\leq N_{PEs}$(条件式 (3)) を満たす時には，マルチレベルの負荷分散が効果的である．

(g) N_{PEs}(条件式(5)) ≤ 2 の時, 問題の規模が小さ過ぎるために, そのプロセッサ台数では良い台数効果は得られない. この場合は使用するプロセッサ台数を少なくしてサブタスクの数を減らし, 1 レベルの負荷分散を試みてみる.

以上の手順で本方式を適用することによって, 多くの試行錯誤を繰り返すことなく, 与えられた問題に対して最大の台数効果を得ることができる.

6 おわりに

以上, 疎結合並列計算機マルチ PSI/V2 上での, OR 並列型全解探索問題に適したマルチレベル動的負荷分散方式について述べた. 本方式はプロセッサをグループ化し, グループレベルの負荷とプロセッサレベルの負荷の均等化を階層的に行なうものである. 階層構造をなしているため, プロセッサの台数拡張性がある. ただし, グループマージ手法には改良の余地がある.

本方式は詰込みパズルの全解探索問題に適用し, メッシュ結合型の疎結合並列計算機であるマルチ PSI 上に実現して計測を行なったところ, 次のようにほぼ線形に近い台数効果が得られた: 8 台で 7.7 倍, 16 台で 15 倍, 32 台で 28.4 倍, 64 台で 50 倍.

またサブタスクの数と粒度についての定式化を行なったが, これは疎結合並列計算機上での OR 並列問題一般の負荷分散を考える際の指針となるものである. 本方式は OR 並列問題に限らず, アルファベータ枝刈り問題等プロセッサ間通信があまり頻繁に起こらない木構造の探索型問題一般に適用可能である. 各種応用プログラムへの本方式の適用が, 今後の課題である.

7 謝辞

本研究の機会をいただいた ICOT の内田俊一第 4 研究室室長, データの計測にあたって多くの測定用ツールを整備していただいた中島克人氏 (現在 三菱電機 (株) 情報電子研究所), 及び様々な助言をいただいた各氏に感謝致します.

参考文献

[1] K. M. Baumgartner, and B. W. Wah. "GAMMON: A Load Balancing Strategy for Local Computer Systems with Multiaccess Networks". In *IEEE Transactions of Computers*, Vol. 38, No.8, pages 1,098–1,109, Aug. 1989.

[2] T. Chikayama. "Load balancing in a very large scale multi-processor system". In *Proceedings of Fourth Japanese-Swedish Workshop on Fifth Generation Computer Systems*. SICS, 1986.

[3] T. Chikayama, H. Sato, and T. Miyazaki. "Overview of the parallel inference machine operating system (PIMOS)". In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 230–251, 1988.

[4] T. C. K. Chou, and J. A. Abraham. "Load Balancing in Distributed Systems". In *IEEE Transactions of Computers*, Vol. SE-8, No.4, pages 401–412, Jul. 1982.

[5] E. W. Felten and S. W. Otto. "A highly parallel chess program". In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 1001–1009, Dec. 1988.

[6] C. Ferguson, and R. E. Korf. "Distributed Tree Search and its Application to Alpha-Beta Pruning". In *Proceedings of the Seventh National Conference on Artificial Intelligence 1988*, Vol. 1, pages 128–132, Aug. 1988

[7] A. George, M. T. Heath, J. Jiu, and E. Ng. "Solution of Sparse Positive Definite Systems on a Shared-Memory Multiprocessor". In *International Journal of Parallel Programming*, Vol. 15, No. 4, pages 309–325, 1986.

[8] S. Hiroguchi, and Y. Shigeki. "Optimal Number of Processors for Finding the Maximum Value on Multiprocessor Systems". In *Proceedings of The Twelfth Annual International Computer Software and Applications Conference 1988*, pages 308–315, Oct. 1988

[9] K. Nakajima, Y. Inamura, N. Ichiyoshi, K. Rokusawa, and T. Chikayama. "Distributed implementation of KL1 on the Multi-PSI/V2". In *Proceedings of the Sixth International Conference on Logic Programming*, pages 436–451, 1989.

[10] L. M. Ni, and Kai Hwang. "Optimal Load Balancing Strategies for A Multiple Processor System". In *Proceedings of 10th International Conference of Parallel Processing 1981*, pages 352–357, Aug. 1981 .

[11] D. M. Nicol, and F. H. Willard. "Problem Size, Parallel Architecture, and Optimal Speedup". In *Journal of Parallel And Distributed Computing*, 6, pages 404-420, 1988

[12] S. Pulidas. "Imbedding Gradient Estimators in Load Balancing Algorithms". In *Proceedings of 8th International Conference on Distributed Computing Systems 1988*, pages 482–490, Jun. 1988

[13] A. N. Tantawi. "Optimal Static Load Balancing in Distributed Computer Systems". In *Journal of the Association for Computing Machinery*, Vol. 32. No. 2, pages 445-465, April 1985.

[14] E. Tick. "Compile-Time Granularity Analysis for Parallel Logic Programming Languages". In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, ICOT, 1988.

[15] J. Schaeffer. "Distributed Game-Tree Searching". In *Journal of Parallel And Distributed Computing*, 6, pages 90-114, 1989