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

——LLS-G 方式——

佐藤令子† 佐藤裕幸†
中島克人† 田中千代治†

疎結合型大規模マルチプロセッサに適した，局所情報に基づく動的負荷分散の一方式として，世代別動的負荷浸透方式（LLS-G 方式）を提案する．本方式では，親タスクから生成される子タスクと，この生成期間中に他プロセッサから受け取ったタスクを1つの「世代」として管理し，各プロセッサはおのおのの隣接するプロセッサ群と，各世代の実行ごとに次の世代の実行時間，すなわち忙しさの予測情報を交換し，より低い負荷が予想されるプロセッサにタスクを分散する．本方式は積極的に負荷を均等化させようとするものであり，メモリ等の使用資源の均等化やタスクの分配遅れの隠蔽等の効果が得られると共に，仕事が十分にある状態での負荷分散オーバヘッドを抑制するという特長も有する．タスク分散の時間間隔が各世代の実行時間に応じて調整されるためである．本方式は問題を互いに独立な多くの部分問題に分割し，繰り返す手続きを「世代」として定義できるならば，種々の問題に適用可能である．本方式を並列推論マシン PIM/m（最大構成時プロセッサ数 256）上で実装し，IDA*アルゴリズムに基づく15パズルの2つの初期パターンに適用して評価を行った．この結果，要求駆動型の動的負荷分散方式であるスタック分割動的負荷分散（STB）方式を適用した場合に比べ，プロセッサ数が多い場合に，絶対性能および台数効果が上回ることを確認した．

A Diffusional Load Balancing Scheme on Loosely Coupled Multi-Processors

——LLS-G——

Reiko Satoh,† Hiroyuki Sato,† Katsuto Nakajima† and Chiyoji Tanaka†

We propose a dynamic load balancing scheme called LLS-G (Local Load Spreading scheme using Generation Information) for large scale loosely coupled multiprocessors. In this scheme, subtasks spawned by parent task(s) or received from other processors during the spawn are classified as one "generation." Each processor exchanges information with its neighbors on the predicted workload derived from the execution time of the former generation, and distributes tasks to processors with lower loads. Results of load balancing using this scheme include the balanced use of memory and the hiding of load distribution delays. In addition, because the interval of load balancing is synchronized with the execution time of each generation, the overhead of load balancing can be made smaller when each processor has sufficient tasks. This load balancing scheme is applicable to various types of problems as long as an iterative procedure can be defined to yield the next "generation" subtasks. We evaluated this scheme on the mesh-connected parallel inference machine, PIM/m (up to 256 processors). When run on a large number of processors, a 15-piece puzzle program based on the IDA* algorithm shows better performance and speed-up with this scheme than with another dynamic load balancing schemes called STB method.

1. はじめに

大規模な疎結合型のマルチプロセッサにおいて，負荷の動的な分散と均等化は，プロセッサ資源の有効利用の観点から非常に重要な問題の1つである．

どのような情報に基づいて負荷の分散を行えば各プロセッサの資源を有効利用したことになるのかは，そのシステムあるいはアーキテクチャごとの固有の問題であるが，一般的には CPU の稼働率をその対象として考えることが多い．すなわち，とりあえず CPU を遊休させず稼働率を100%に維持する，という戦略である．このような戦略では，結果としてすべてのプロセッサでの実行時間が同じくらいになる，という効果はあるが，「均等化」をしているわけではないので，ある時間断面での各プロセッサの持つ負荷量は一般に不均等となり，ある特定のプロセッサでメモリのワー

† 三菱電機(株)情報システム研究所
Mitsubishi Electric Corporation

キングセットが大きくなりキャッシュ・メモリのミスヒットが多発する，といった現象が防止できない．

また，従来の方式の多くは単にCPUを遊休させないということを目標としているため，負荷分配の遅れの防止が困難であると同時に，自プロセッサが所有して実行すべき負荷の適正量を求める手段がなく，負荷の自発的な分配ができない，という問題点がある．

しかし，実用上はこのような戦略でも十分効果的な場合が多く，また，負荷の適正量を算出するために各プロセッサから得るべき情報が明確にできないこと，「均等化」のメカニズムをソフトウェアで実装した際のオーバヘッドが「均等化」の効果を上回る可能性が否定できないことなどにより，複雑な「均等化」のメカニズムを実際の問題に適用しようという努力はほとんど払われていない．

本論文は，このような負荷の動的な均等化問題に対する1つの解決策として，均等化を行いながら自発的に負荷を分散し，「世代」という概念を取り入れることで均等化のオーバヘッドを小さく抑えることを可能とした「世代別の拡散型動的負荷分散方式」の提案を行うものである．

2. 従来の方式

2.1 MLB 方式と STB 方式

マルチレベル動的負荷分散方式（Multi-Level Dynamic Load Balancing: MLB 方式）[1]は，特定のプロセッサが，問題を互いに独立な仕事に分割する作業を行い，仕事を要求してきたプロセッサに対して割り付ける方式（このような方式を『要求駆動型』の負荷分散方式と呼ぶ）である*．一般にこのような方式ではプロセッサの台数が多くなると仕事を供給するプロセッサがボトルネックになるが，MLB 方式では仕事の分割と供給を行うプロセッサを階層的に複数台割り付けることによってこのボトルネックの解消を行う．このため，階層の構成（何層にするか，各層における仕事の供給を行うプロセッサを何台にするか，など）をユーザが問題ごとに調整する必要が出てくる．

また，スタック分割動的負荷分散方式（Stack Splitting Dynamic Load Balancing: STB 方式）[2]は，MLB 方式と同様に要求駆動のメカニズムを用いながら，仕事の分割および供給を行うプロセッサを特定せずどのプロセッサも仕事の分割および供給の対象とする方式である．各プロセッサは仕事の供給が受けられるまで，ある戦略に従い自分以外のすべてのプロセッサに順番に仕事を要求する．このため，MLB 方式のような階層化によるボトルネック対策は不要であるが，自分以外の全プロセッサに対して仕事を要求および供給する可能性があるため，大規模な並列マシンにおいてはローカリティが失われ，通信距離や通信量が大きな問題点となる可能性を持っている．

この2つの方式は，実際に幾つかの応用問題に適用されて相当の台数効果をあげているが，いずれの方式においても均等化に相当することは特に行っていない．

2.2 LLS 方式

動的負荷浸透方式（Dynamic Local Load Spreading Method: LLS 方式）[3]は筆者らが提案を行った，均等化メカニズムを含む動的でかつ『要求駆動型でない』負荷の分散方式である．

LLS方式は，マルチプロセッサを構成する各プロセッサがおのおのの隣接するプロセッサ群（これを「近傍」と呼ぶ）からの「忙しさ」の報告を受け，近傍の平均的な忙しさに対する自プロセッサの忙しさを動的に計算し，その結果に基づいて仕事の分配を行う方式である．仕事の分配は自プロセッサが近傍内で「比較的忙しい」と判断した場合に起こり，分配量は近傍の各プロセッサの「忙しさ」ができるだけ均等となるよう決定される．すべてのプロセッサが自分を中心とする近傍に対して同様の処理を行うことにより，プロセッサ全体の仕事量の均等化を行うことができる．

LLS方式でいう「忙しさ」とはプロセッサがある個数（ジョブキューにセンチネルを挟んで数えることからこれをセンチネル個数と呼んでいる）の部分問題の実行に費やす時間，すなわち，「センチネル個数の仕事の実行にどのくらい時間を要したか」を負荷均等化のための基準情報として扱っている．この基準情報は近傍からおのおののプロセッサの基準で自発的に報告されるので，この「時間」の正規化を，報告を受けた側のプロセッサの仕事の個数を用いて行う．これにより，本方式では，タイマが不要であると同時に，基準情報のデータ量が最小限に抑えられる*．

また，センチネル個数は，均等化の度に近傍での平均の忙しさに合わせられるので，近傍の忙しさに応じて均等化の間隔が自動的に調節されるという利点があ

* MLB 方式では仕事を供給するプロセッサも他のプロセッサからの仕事の要求がないときには仕事を実行する．

* 実際にこの情報は「今センチネル個数の仕事を完了した」という通知のみでよく，ハードウェア上で信号線を一本使用するだけで実現できる，という利点があった．

るが，一方，センチネル個数の値によっては，基準情報の報告を行う回数が非常に増える可能性を持っており，同時に均等化の回数も増えるため，負荷分散オーバヘッドが大きくなるという問題点を持っている．

3. 世代別動的負荷浸透方式（LLS-G 方式）

MLB 方式と STB 方式ではパラメタが最適に調節された場合，性能差はほとんど見られず（64 台以下の場合）線形に近い台数効果が得られることが報告されている[2]．一方，LLS 方式では現在のところその性能は STB 方式の約 1/2 であり[3]，その原因として大きく以下の3つの要素が考えられる．

（1）均等化に用いる基準情報の報告を行う間隔が細かすぎ，オーバヘッドが大きい．
（2）均等化に用いる基準情報が不十分である．
（3）均等化の間隔が細かすぎ，仕事のたらいまわしが起こりやすくなっている．

そこで，これらの要素について考察を行い，世代別動的負荷浸透方式（Dynamic Local Load Spreading Method-Generation：LLS-G 方式）を新たに提案する．

3.1 基準情報の報告と均等化の間隔

LLS 方式では，基準情報報告の間隔は各プロセッサに設定されたセンチネル値に従う*．このセンチネル値の変更は近傍における各プロセッサの「忙しさ」が異なっている場合にのみ起こるので，たまたま小さなセンチネル値で均等化が行われてしまうと，それ以降，各プロセッサに十分な量の仕事がありかつ均等化された状態においても，（不要な）基準情報の報告を細かい間隔で実施してしまう．また，均等化の間隔も基準情報の報告と同様に各プロセッサのセンチネル値に従うので，小さなセンチネル値で一度均等化に成功した場合には，以後不必要な均等化処理を行う回数が増えることになる．

一般にプログラムの実行は，負荷の均等化の立場から見たとき次の3つのフェーズに分けることができる．

（1）プログラムの起動時→できるだけ早く問題を分散するために細かい間隔で均等化を実施する
（2）各プロセッサに十分たくさんの仕事があり，均等化がある程度できているとき→細かすぎないある程度の実施間隔を維持したい
（3）プログラムの終了時→すべてのプロセッサができるだけ同時に終了できるように，細かい間隔で均等化を実施する

このようなプログラム全体の実行状況は，問題の（例えば，OR 並列型探索問題を対象とするならば探索木の）形状からある程度予測することが可能である．つまり「問題の（探索木の）形状」は，大雑把にいうならば，各深さにおける部分問題群がその子供として生成した部分問題群の総数に基づいた関数であらわすことができるので，各深さにおける部分問題群ごとの均等化を実現することにより，プログラム全体の実行状況（すなわち，現在起動時か終了時か，あるいは中間か）をある程度考慮した均等化が可能である．

LLS 方式では，基準情報の報告と均等化の間隔およびそれらの回数については，センチネル個数を用いて制御することで仕事量の増減に関し追従性のよい値が得られると考えていた．しかし実際には基準情報の報告にかかるコストが予想外に大きいため，このコストを削減する必要がでてきた．そこで LLS-G 方式では LLS 方式のセンチネルを廃止し，基準情報の報告および均等化を「世代」別に行うことにした．

ここでいう「世代」は，探索木でいえばルートからの深さである*．LLS-G 方式での均等化は各「世代」の実行が終了した時点で行う．このため，均等化は各世代で一度しか行われず，同様に基準情報の報告も各「世代」に関して一度しか行わない．したがって，ある世代の均等化はその親世代の基準情報を用いて行われ，次の均等化のタイミングはその世代の部分問題の量に依存する．**図1**に仕事量と負荷の均等化の間隔の例を示す．1つの短冊は1世代を表し，短冊の横幅は均等化の間隔を表しており，仕事量が十分にある場合

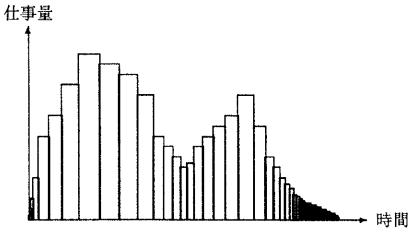


図 1 仕事量と負荷の均等化の間隔
Fig. 1 Relation of processor workload and interval of load balancing.

* 基準情報の報告はセンチネル個数の仕事の実行が完了したときに行われる．

* ただし後述のように，実装上は各プロセッサで一段探索木を降りる間に分配される仕事も次の世代に含めるため，『同世代』には兄弟だけでなく「叔父」「姪」なども含まれることになる．

には均等化の間隔は大きくなる．このように世代ごとの仕事量の増減に応じて自動的に均等化の間隔を調整することで，ほぼ最適な均等化のタイミングを得ることが可能である．

3.2 基準情報

LLS 方式では，基準情報の報告の頻度が高いためできるだけ情報の内容を軽減しておく必要があった．LLS 方式での基準情報の内容は，『今，センチネル個数の仕事の実行が終わった』という信号のみであり，それがどれくらいの時間間隔と評価されるかは，その信号を受けとったプロセッサの側にまかされていた．

一方，LLS-G 方式では，基準情報の報告回数が必要最小限に留められているため，もう少し詳細な情報の報告を行うことができる．と同時にセンチネルを廃止したため，近傍での比較の基準となる数値を新たに設定する必要がでてきた．そこで LLS-G 方式では，次のような情報を用いた基準情報を算出する*．

- 親世代が実行した仕事の個数（N_p）
- 親世代の仕事の実行を開始した時刻（t_0）
- 親世代の仕事の実行を終了した時刻（t_1）
- 親世代の実行により生成された子世代の仕事の個数（N_c）

これらの情報から，近傍のプロセッサ群における子世代の仕事の実行（孫世代の生成）にかかると予測される時間（T_g）を算出することができる．

$$T_g=(t_1-t_0)\times\frac{N_c}{N_p}$$

LLS-G 方式における均等化は，ここで求められる T_g を基準情報とし，この値を近傍で平均化することで行う．

3.3 分配量の決定

基準情報 T_g は，子世代の仕事の実行にかかると予測される時間を示している．この値はプロセッサごとに異なっており，この値を各プロセッサの忙しさと定義することで，近傍の忙しさ（T_{ave}）は以下のように算出できる．

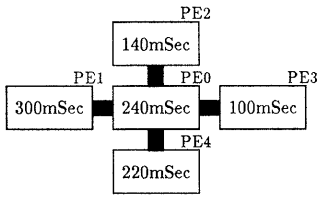
$$T_{ave}=\frac{1}{n}\sum_{i=1}^{n}T_{gi}$$

ただし n は近傍に属するプロセッサの台数である．

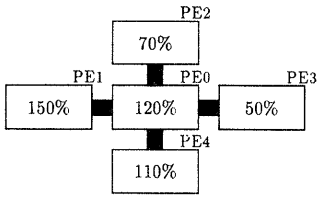
例えば，**図 2**（1）において，近傍の忙しさ T_{ave} は，

$(240+300+140+100+220)\div 5=200$

となる．この T_{ave} を用いて各プロセッサの忙しさを正規化すると，図 2（2）に示す値が得られる．これが近傍における各プロセッサの相対的な忙しさを表している．



（1）



（2）

図 2 近傍の忙しさ
Fig. 2 Workload of neighborhood.

図 2（2）でわかるように，PE 0（プロセッサ 0）を中心とする近傍での忙しさ 100% に対し PE 0 の相対的な忙しさは 120% であるため，PE 0 は「近傍の他のプロセッサより 20% 余計に仕事を抱えている」と考えることができる．このとき PE 0 が分配を行う総仕事量（N_{dist}）は，その近傍において親世代と同じペースで仕事の実行がなされた場合に，現在 PE 0 が持っている子世代の仕事の個数（N_c）のうち他のプロセッサと同一時間内には実行しきれないと予測される仕事量である．この値は次の計算式により，簡単に算出できる．

$$N_{dist}=N_c\times\frac{T_g-T_{ave}}{T_g}\qquad(\text{ただし } T_g>T_{ave})$$

この余剰すると予測される仕事量を，平均よりも負荷が軽い（と予測される）プロセッサに対しておのおのの忙しさに応じて比例配分することで，子世代の仕事量の均等化を図る．例えば図 2 の例では，PE 2 と PE 3 に対して，おのおの

$$PE2:\ N_{dist}\times\frac{100-70}{(100-70)+(100-50)}=N_{dist}\times\frac{3}{8}$$

$$PE3:\ N_{dist}\times\frac{100-50}{(100-70)+(100-50)}=N_{dist}\times\frac{5}{8}$$

ずつ仕事を分配することで次世代の仕事量の均等化を図ることができる．

一方，64 台のシステムにおいても 256 台のシステムにおいても，ある一定時間内ですべてのプロセッサ

* 時刻を計測するために各プロセッサが独立したタイマを持つことを前提としている．

に仕事を拡散できるようにすることを考えると，システム半径が大きくなり拡散を『早めたい』場合には，仕事の粘性（拡散のしにくさ）を下げてやる必要があることがわかる．この粘性Dはシステム半径に応じて設定されるべきハードウェア上のパラメータであり，0から1の間の値をとると考えている．しかし現在は定式化できていないため，暫定的な式を用いて調整を行っている．近傍の忙しさの基準値 T_{ave} は，実際にはこの粘性 D を乗じて 100% よりも小さな値として計算されるので，分配すべき仕事量 N_{dist} も多めに計算されることになる．これは，1ホップよりも遠隔にあるプロセッサ群の忙しさを，隣接するプロセッサと同様であると予測していることにほかならない．

4. 評　　価

4.1 実装と計測

LLS-G 方式を並列推論マシン PIM/m[5]上に並列論理型言語 KL1 を用いて実装し計測を行った．対象とした問題は，Iterative-Deepening A* アルゴリズムに基づく 15 パズル問題[4]である．また，比較のため STB 方式を用いて同様の計測を行った．

ここでは，ある1つの初期状態から最終状態を導く15パズルについて文献4)に掲載された2種類の問題（問題DおよびE）を用いて計測を行った（**図3**）．15パズルは，15枚のタイルのある状態から次にどのタイルを動かすかにより，幾つかの独立した状態をつくるので，基本的な OR 並列型探索問題として考えることができる．

ここで計測に用いた問題DおよびEは，タイルを1

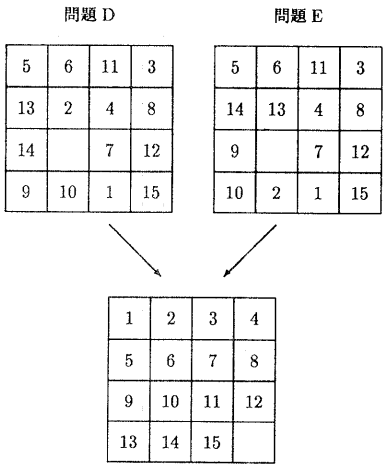


図 3 15パズル（問題D，問題E）
Fig. 3 15 puzzle (Problem D and Problem E).

枚動かすことを1世代として，どの1枚を動かすかである状態から子世代にあたる異なった状態を作ると仮定したとき，それぞれ，

問題D 総状態数約150万，世代数38，最大状態数約17万（第20世代）

問題E 総状態数約590万，世代数42，最大状態数約62万（第23世代）

となり，いずれの場合も，各世代の状態数の推移は最大状態数を持つ世代を中心とした山なりの形状を示す．

ユーザプログラムと LLS-G 方式を用いた負荷バランサの基本的な動作は

- ユーザプログラムは実行を1世代進める，すなわち，本 15 パズルの場合は，ある状態を与えられるとその続きとなるすべての「タイルを1枚動かした状態」を作って返す．
- LLS-G 方式を用いた負荷バランサは，各プロセッサごとにユーザプログラムを用いてある世代（幾つかの状態）から次の世代（幾つかの状態）を得，各世代ごとに他のプロセッサと負荷分散を行う．

である．本負荷バランサを他の問題に対して適用するユーザは，「1世代」を定義し，「ある状態から1世代進めた状態を生成する」特定の名前の述語を設けるだけでよい*．前述した各問題の「世代数」は探索木のルートからの距離を示す値であり，いわば，「論理的な世代」である．これに対し，実装上は各プロセッサにおける世代ごとの実行は同期せずに進むため，並列実行した場合の均等化の回数は世代数を越える数値となるし，各プロセッサが1世代と見なす仕事は，異なった世代の仕事の混在状態となる．

表1に各問題の実行時間を，また，16台版の実行時間を基準とした台数効果を**図4**に示す．

表1からわかるように，マシンを構成するプロセッサ台数（以下，「構成台数」と呼ぶ）が少ない場合には，LLS-G 方式は STB 方式より実行時間が遅いが，問題Dでは160台以上の場合に，また，問題Eでは256台の場合に実行時間の逆転が起こっている．このことから，構成台数がある程度多い場合には LLS-G 方式の方が STB 方式よりも速くなる傾向にあると考えられる．また，図4で問題Dの台数効果をみると，STB 方式では構成台数が増えるとともに台数効果が

* このような負荷バランサとユーザプログラムの切り分けは STB 方式[2]で用いられているものである．

表 1 LLS-G と STB の実行時間
Table 1 Execution Time of LLS-G and STB.

(単位: ミリ秒)

プロセッサ数	16	32	64	96	128	160	192	224	256
問題 D									
LLS-G	136,140	69,731	34,807	24,473	20,339	17,137	15,181	14,099	12,919
STB	121,594	63,143	33,243	24,297	19,394	17,265	16,258	15,400	15,276
問題 E									
LLS-G	539,895	271,696	151,238	98,405	75,227	58,863	49,865	44,594	38,710
STB	469,451	236,317	121,390	82,110	65,055	54,681	47,849	43,518	39,821

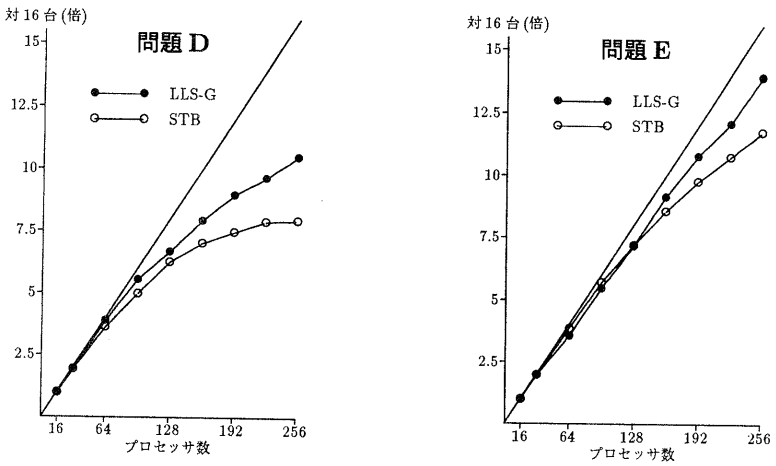


図 4 LLS-G と STB の台数効果
Fig. 4 Performance of LLS-G and STB.

頭打ちとなっているのに対し，LLS-G 方式では傾斜が緩やかになってはいるが，依然として上昇傾向を示していることがわかる．

4.2 考　　察

LLS 方式の長所は，構成台数の増加による影響を受けにくいことにある．例えば，分散のコストを「仕事を貰う1回あたりのコスト」として構成台数の増加について考える．

LLS 方式では，1台あたりの分散コストが構成台数の増加によらず各プロセッサが近傍と接続する手の本数で抑えられる．一方，STB 方式では分散のための論理的なネットワークが完全結合である必要があり，そのコストは（プロセッサ総数×システムの半径）のオーダになると考えられる．この2つのコストが交差する点（点Pとする）が実際に構成台数が何台のところにあるかを知ることが，本研究における1つの大きな課題であった．

PIM/m 上の LLS 方式では，少なくとも 128 プロセッサまででではこの2つのコストが交差しないように思われた[3]．これは STB 方式が要求駆動型であり，1回の負荷分散のためのメッセージ数が1～2に抑えられているのに対し，LLS 方式では実行時間に依存する関数になってしまうため，構成台数に応じた大きな問題を扱い実行時間が伸びるほど負荷分散のためのメッセージが増加し，最終的な負荷分散コストを引き上げているためと考えられた．そこでこのメッセージ数の削減を行った LLS-G 方式の提案を行った．

表 2 に，LLS-G 方式および STB 方式で問題の実行時に実際に受信されたすべての通信メッセージの数を示す．数値をみただけでも，明らかに構成台数に対する増加率が異なっていることがわかる．

図 5 は，表2のメッセージ数をもとに1ミリ秒あたりに1台のプロセッサが受信したメッセージの数を示したものである．特に問題Dの場合に LLS-G 方式と STB 方式の差が顕著である．STB 方式では構成台数に対してメッセージ数が線形程度かまたはそれ以上に伸びているのに対し，LLS-G 方式ではほぼ一定の値を保っているので，2つのコストが逆転する点Pが，

表 2 LLS-G と STB のメッセージ数
Table 2 Number of messages for LLS-G and STB.

(単位: × 1000 メッセージ)

プロセッサ数	4	8	16	32	64	96	128	160	192	224	256
問題 D											
LLS-G	1,387	2,073	2,026	2,976	2,915	3,135	3,482	3,928	3,842	4,264	4,737
STB	490	379	340	1,237	1,337	2,626	3,994	6,227	7,894	14,510	23,518
問題 E											
LLS-G	9,001	7,216	9,072	10,303	12,869	11,115	12,026	12,968	13,303	13,782	14,462
STB	1,667	1,136	1,266	1,745	3,890	5,751	7,451	8,729	11,251	14,868	18,938

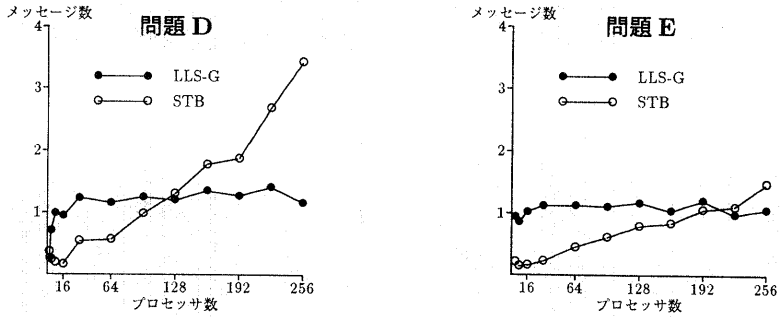


図 5 LLS-G と STB の通信メッセージ数
Fig. 5 Number of messages for communication of LLS-G and STB.

問題Dでは128台のところに，問題Eでは224台のところにあることがわかる．すなわち，LLS-G 方式では各プロセッサが分散に費やすメッセージ数は構成台数によらず一定であり，分散コスト≃メッセージ数と考えると，より大規模なマルチプロセッサ向きの動的負荷分散方式である，ということができる．問題の実行時間の観点から見ても，分散コストの点からは明らかに STB 方式が有利である少ない構成台数の場合にも，LLS-G 方式では最悪 1.25 倍程度の実行時間で抑えることができ，かつ構成台数がある程度を越えると逆に実行時間が速くなる（256 台で 1.18 倍）ことから，構成台数によらず LLS-G 方式を用いる利点は十分にあると思われる．

また，図4，図5に共通の傾向として，問題Eに比べ問題Dの結果に STB 方式と LLS-G 方式の差が顕われやすい．この理由として，問題サイズ（これは15パズルの場合，探索木上の総状態数を指す）が考えられる．ある問題を並列実行する場合に，その実行時間を支配するのは「プロセッサあたりの仕事量*」と「負荷分散オーバヘッド」である．プロセッサあたりの仕事量が少ない問題では，多い問題に比べて負荷分散オーバヘッドが支配項になりやすく，仕事の分散による実行時間の短縮の効果が顕れにくい．

図4の問題Dにおける台数効果で STB 方式の台数効果が LLS-G 方式の台数効果に比べて早く飽和するのは，プロセッサあたりの仕事量と負荷分散オーバヘッドの実行時間に占める割合の逆転が比較的少ない台数で起こるためであると考えられる．これは図5の STB 方式による問題Dの通信メッセージ数の増加率が LLS-G 方式に比べてかなり大きいことからも予測できる．これに対し問題Eでは，プロセッサあたりの仕事量が比較的多いため仕事量と負荷分散オーバヘッドの割合の逆転が起こりにくく，基本的に高い台数効果を得やすい．このため負荷分散オーバヘッドの大小によらず，比較的良い台数効果を維持できると考えられる．

また，LLS-G 方式に関して各問題の台数効果と負荷分散オーバヘッドを見たとき，通信メッセージ数がほぼ一定であるにもかかわらず台数効果の低下が見られる．これはプロセッサの稼働率の低下によるものであり，「負荷の分散状態が理想的な均質状態からどのくらい遠いか」を示しているものと考えられる．

* ここでは，世代ごとに存在する仕事量を構成台数で除したものと考えられる．

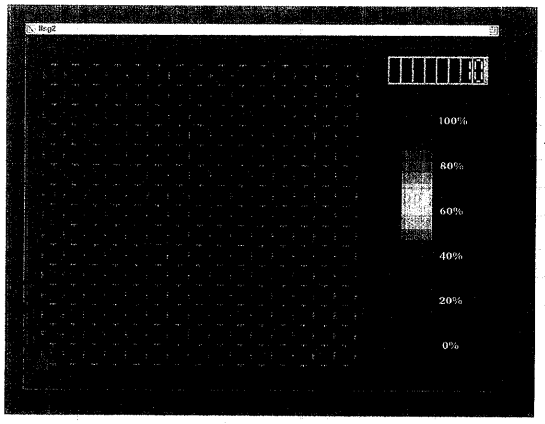


図 6(1) 稼働率モニタ (1)
Fig. 6(1) Performance Monitor (1).

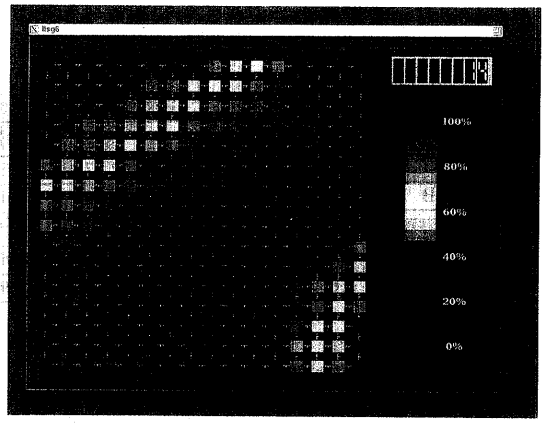


図 6(3) 稼働率モニタ (3)
Fig. 6(3) Performance Monitor (3).

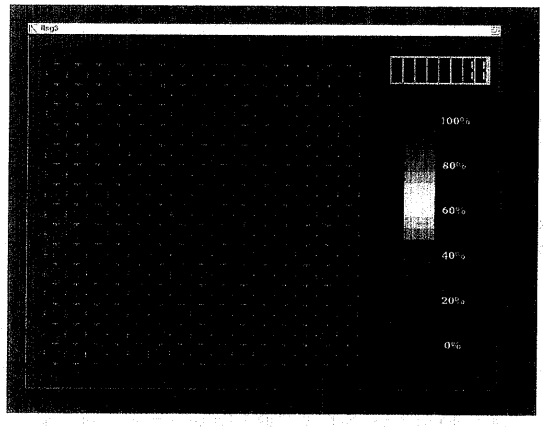


図 6(2) 稼働率モニタ (2)
Fig. 6(2) Performance Monitor (2).

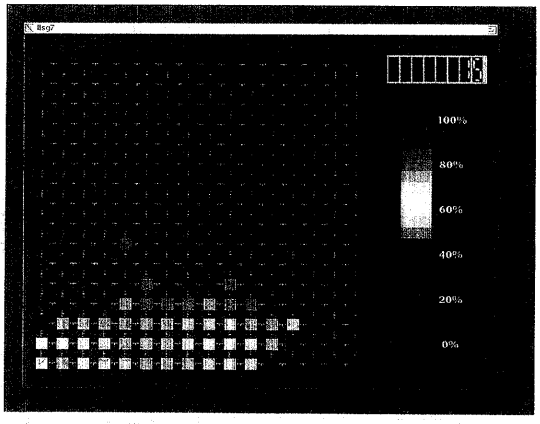


図 6(4) 稼働率モニタ (4)
Fig. 6(4) Performance Monitor (4).

図 6 はパフォーマンス・メータと呼ぶプロセッサの稼働率モニタ[7]である．1つの四角が1プロセッサを表し，画面右のスケールに合わせて色で各プロセッサの稼働率が示される．図6は問題Dの実行時の稼働状況を示したもので，(1)は 16×16 の二次元格子に構成されたプロセッサ群の（ほぼ）中心に仕事を投入してから次の2秒間で仕事が行き渡った様子，(2)は全プロセッサの稼働率がほぼ 100% に達した様子である．その後4秒間この状態が続く．(3)は実行が収斂し始めた瞬間を示しており，正方形に配置したプロセッサの左上および右下から仕事がなくなってくる様子である．同様に次の2秒では仕事のなくなったプロセッサが増え(4)，さらに2秒後には仕事は終了する．図6(1)および(4)はともかく，(3)に示す収斂開始時には遊休プロセッサと稼働率100%のプロセッサが同時に存在する様子が見られる．このような状況に，完全に線形な台数効果を得ることのできない理由の一端があり，さらに理想的に負荷が均等化された状態が存在する可能性は十分に考えられる．より精度の高い立ち上がり，収斂状況を得るためには，マシンごとの粘度Dを定式化するための尺度を確定し，「仕事の拡散しにくさ」を正確に定義可能とする必要がある．

いずれにしろ，考察を行ってきたように，LLS-G方式は LLS 方式だけでなく STB 方式のような従来方式と比較しても，疎結合型の大規模マルチプロセッサにおいて，かなり有望な動的負荷分散方式であるといえる．

また，計測に用いた並列推論マシン PIM/m においては，ネットワークを渡るメッセージの送受信に要する時間のうち，通信パケットのエンコード/デコードに要する時間がかなりの部分を占めており[6]，ネットワーク・ハードウェア上のホップ数の影響がほとんど

現れない．換言すれば，メッセージの送受信時間に対してシステム半径が支配項となるようなタイプの大規模並列マシンにおいては，局所情報のみにより負荷分散を行う LLS-G 方式の効果は一層期待できると考えられる．

5. おわりに

本論文では「世代別動的負荷浸透方式 (LLS-G 方式)」について詳細に述べ，並列型推論マシン上に実装し OR 並列型探索問題に適用することにより，その性能評価などを行った．

拡散型の動的負荷分散方式は，もともとその負荷分散オーバヘッドの大きさから実現が難しいと考えられていたが，「世代」概念の導入により，より大規模なマシンに対して有効な方式であることが実証できたと考えている．

「世代」という概念を導入するにあたり，対象問題を OR 並列型探索問題とすることで問題に対する世代の効果の解析を行いやすくなった．しかし「世代」は，もともとこのような問題自身の性格とは異なった概念であり，OR 並列型でない問題，例えばデータ並列型の問題に対してもデータを適当にパーティショニングしてそれぞれのパーティションを「世代」として扱う，などのように「世代」を想定したプログラミングは可能であり，本方式を適用することができると思われる．

ただし，本方式では負荷均等化の精度が「親世代」と「子世代」の負荷量 (T_g) の増減の類似性に依存しているので，世代ごとに常に増加傾向と減少傾向が切り換わるような問題があるとすると，「負荷の予測」が外れ続け，仕事のたらいまわしが起きるだけで実行がまったくなされない，ということも考えられる．

今後，LLS-G 方式を用いた負荷バランサをさまざまな問題に適用することで，本方式がどのような範囲の問題に適用可能であるかを明確にしていくと共に，方式そのものに対する解析・考察を行い，より効率の良い動的負荷分散方式の研究・開発を行っていきたい．

謝辞 本研究は(財)新世代コンピュータ技術開発機構 (ICOT) の研究成果の検証の一環として実施したものである．

本研究の遂行にあたり，ご指導，ご協力いただいた ICOT の近山隆・第一研究部長，和田正寛研究員（現シャープ(株)情報技術研究所）ならびに積極的に討論していただいた三菱電機(株)情報システム研究所・並列処理第2グループの各位に感謝いたします．

参考文献

1) 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，*KL 1 Programming Workshop '90*, pp. 1-9 (1990. 5).
2) 古市昌一，中島克人，中島 浩，市吉伸行：スタック分割動的負荷分散方式とマルチ PSI 上での評価，1991 年並列/分散/協調処理に関する『大沼』サマー・ワークショップ，コンピュータシステム研究会，pp. 33-40 (1991. 7).
3) 佐藤令子，佐藤裕幸：疎結合型マルチプロセッサ上の拡散型負荷分散の一方式，1992 年並列/分散/協調処理に関する『日向灘』サマー・ワークショップ，コンピュータシステム研究会，CPSY 92-9 (1992. 8).
4) 和田正寛，市吉伸行，六沢一昭：Iterative-Deepening A^* アルゴリズムのスタック分割動的負荷分散方式による並列化と並列推論マシン PIM/m 上の性能評価，1992 年並列/分散/協調処理に関する『日向灘』サマー・ワークショップ，プログラミング―言語・基礎・実践―研究会，pp. 195-202 (1992. 8).
5) Nakashima, H., Nakajima, K., Kondo, S., Takeda Y., Inamura, Y., Onishi, S. and Masuda, K.: Architecture and Implementation of PIM/m, *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 425-435 (June 1992).
6) Nakajima, K.: Distributed Implementation of KL 1 on the Multi-PSI, *Implementation of Distributed Prolog*, John Wiley & Sons, pp. 311-332 (1992).
7) Aikawa, S., Kamiko, M., Kubo, H., Matsuzawa, F. and Chikayama, T.: ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems, *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 286-293 (June 1992).

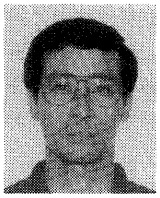
(平成5年9月14日受付)
(平成6年1月13日採録)

佐藤　令子（正会員）

1961年生．1984年九州工業大学情報工学科卒業．同年三菱電機(株)入社．(財)新世代コンピュータ技術開発機構の委託により，並列論理型言語の処理系，並列推論マシンのフロントエンド機能の開発などに従事．現在，同社情報システム研究所にて，並列および分散環境における負荷の解析と分散方式の研究に従事．

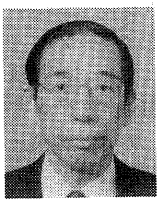
佐藤　裕幸（正会員）

昭和34年生．昭和57年筑波大学第3学群情報学類卒業．同年三菱電機(株)に入社．昭和60年6月～平成元年4月，(財)新世代コンピュータ技術開発機構に出向し，推論マシン用プログラミング環境，並列オペレーティング・システムの研究開発に従事．その後，並列データベース管理システムの開発を経て，現在，三菱電機情報システム研究所にて，並列プログラミング言語の研究開発に従事．



中島　克人（正会員）

1953年生．1977年京都大学工学部電気第二工学科卒業．1979年同大学院修士課程修了．同年三菱電機(株)に入社し，汎用・専用計算機の開発に従事．1982年より第五世代コンピュータ・プロジェクトに参画し，逐次型および並列型推論マシンのアーキテクチャ/言語処理系などの研究開発に従事．1985年～1989年(財)新世代コンピュータ技術開発機構（ICOT）に出向．現在，同社情報システム研究所並列処理第2グループマネージャ．1993年よりリアルワールド・コンピューティング（RWC）プロジェクトに参画し超並列マシン向き計算モデルの研究に従事．



田中千代治（正会員）

1961年大阪大学工学部通信工学科卒業．同年三菱電機(株)入社．1968～1970年イリノイ大学計算機工学部に留学，修士課程（MS）修了．現在，同社情報システム研究所勤務．計算機アーキテクチャ，人工知能，第五世代コンピュータプロジェクト等の研究に従事．工学博士．電子情報通信学会，人工知能学会各会員．