TR-0841

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（三菱電機）

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

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

三菱電機 (株) 情報電子研究所

E-mail: {ezaki,hiroyuki,nak}@isl.melco.co.jp

概要

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。LLS-G 方式では隣接したプロセッサと忙しさの予測情報を交換しつつ負荷の分散・均等化、及び、その間隔の調整を行なう。予測には部分問題の並列度の増減を用いている。並列推論マシン PIM/m 上での評価により、従来の動的負荷分散方式よりも優れた絶対性能と台数効果を確認した。

Abstract

A dynamic load balancing scheme called LLS-G (Local Load Spreading method using Generation Information) is proposed for large scale loosely coupled multi-processors. In this scheme, each processor exchanges with its neighbors information on the predicted workload derived from the number of sub-tasks generated, and distributes tasks to lower-loaded processors. The interval of load balancing is also managed according to the degree of the workload. Evaluation on the parallel inference machine, PIM/m, shows better performance and speed-up than other dynamic load balancing schemes.

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一つであると考えられている。

もともと、負荷を「分散」することと、それによって消費される資源を「均等利用する」ことは、密接に関係はしているが異なった問題である。どのような情報に基づいて分散を行なえば、各プロセッサの資源を均等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毎の固有の問題であるが、一般的には、実行時間がその対象として考えられることが多い。

現在、ユーザが実際に並列マシンを用いて問題を解く際に用いている負荷分散の戦略は、とりあえずCPU を遊休させず「稼働率」を 100%に維持することである。しかし、これ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だけで、「均等化」を行なっているわけではなく、ある時間断面での各プロセッサの持つ仕事量は一般に不均等である。

しかし、このような方式でも充分効果的である場合が多いため、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。また、このような「均等化」が、「均等化」を行なうために各プロセッサから得るべき基準情報が明確でないこと、「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あることなどにより、現実に適用しにくいことも確かである。

本稿は、このような負荷の動的な均等化問題に対する一つの解決策として、均等化を行ないながら負荷を分散し、「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が可能な「世代別の拡散型動的負荷分散方式」の提案を行なうものである。

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

A diffusional load balancing scheme on loosely coupled multi-processors =LLS-G=
Reiko Satoh　Hiroyuki Sato　Katsuto Nakajima
Mitsubishi Electric Corporation

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[1]。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 では、仕事の分割と供給を行な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なう。

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、仕事の分割及び供給を行うプロセッサを特定せず、全プロセッサが仕事の分割及び供給を行なう方式である。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及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なっていない。

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なった、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、各々の近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、それに基づく仕事の分配を行なう方式である。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なうことにより、プロセッサ全体の仕事量の均等化を行なうことができる。

LLS で言う「忙しさ」とは、プロセッサがある個数 (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である。すなわち、LLS においては、「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う。基準情報は、各プロセッサからの自発的な報告によってなされるので、何らかの値を用いて正規化する必要がある。このため、この時間の計測はすべて、報告を受けた側のプロセッサの仕事の個数を用いて行なっている。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[2]。

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は自動的に調節される。

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な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) リニア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

[1] MLB 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時には仕事を実行する。

[2] 実際にこの値は「今センチネル個数の仕事を実行した」という情報のみでよく、ハードウエア上で信号線を一本追加するだけで実現できる、という利点があった。

式では、現在の所その性能はSTB方式の約$\frac{1}{2}$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なう間隔が細か過ぎ、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過ぎ、仕事のたらい廻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ない、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[3]。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なわれてしまうと、それ以降、各プロセッサに充分な量の仕事があり、かつある程度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。

一方、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均等化に成功した場合には不必要な均等化処理を行なう回数が増える。

一般にプログラムの実行を負荷の均等化の立場から見た時、次の3つのフェーズに分けることができる。

1. プログラムの立ち上がり → 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. 各プロセッサに充分たくさんの仕事があり、均等化がある程度できている → 細か過ぎないある程度の間隔を維持したい
3. プログラムの立ち下がり → 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

[3]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時に行なわれる。

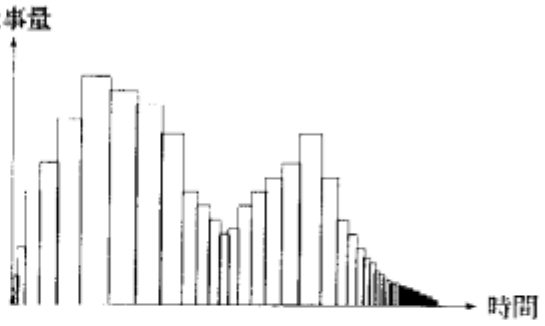

図1: 仕事量と負荷の均等化の間隔

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、大雑把に言うならば、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、現在立ち上がり時か立ち下がり時か、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及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多いため、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、基準情報の報告及び均等化を「世代」別に行なうことにした。

ここでいう「世代」は、探索木で言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なう。このため、均等化は各世代で一度しか行なわれず、同様に基準情報の報告も各「世代」に関して一度しか行なわない。従って、ある世代の均等化はその親世代の基準情報を用いて行なわれ、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、短冊の横幅が均等化の間隔を表しており、仕事

量が充分にある場合には、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。実際には LLS 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なうことができる。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基準情報を算出する[4]。

- 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) にかかると予想される時間 (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なう。

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想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。

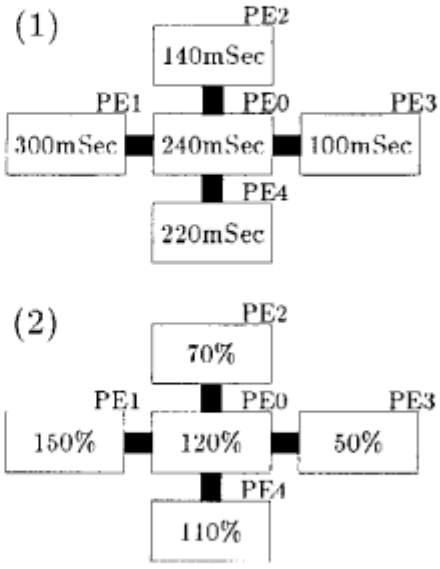

図 2: 近傍の忙しさ

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

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、

$$(240 + 300 + 140 + 100 + 220) \div 5.0 = 200$$

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。

図 2(2) でわかるように、PE0(プロセッサ 0) を中心とする近傍での忙しさ 100% に対し、PE0 の相対的な忙しさは 120% であるため、PE0 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。この時 PE0 が分配を行なう総仕事量 (N_{dist}) は、その近傍において、親世代と同じベースで仕事の実行がなされた場合に、現在 PE0 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

[4] 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。

間内には実行しきれないと予想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。

$$N_{dist} = N_c \times \frac{T_g - T_{ave}}{T_g} \quad (\text{但し} \quad T_g > T_{ave})$$

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(と予想される)プロセッサに対して、各々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計る。例えば、図2の例では、PE2とPE3に対して、各々

$$PE2 : N_{dist} \times \frac{100-70}{(100-70)+(100-50)} = N_{dist} \times \frac{3}{8}$$
$$PE3 : N_{dist} \times \frac{100-50}{(100-70)+(100-50)} = N_{dist} \times \frac{5}{8}$$

ずつ仕事を投げることで、次世代の仕事量の均等化を計ることができる。

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が、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なっている。

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、投げるべき仕事量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他ならない。

4 評価

4.1 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を用いて実装した。その結果を表1に、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。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。

問題D

5	6	11	3
13	2	4	8
14		7	12
9	10	1	15

問題E

5	6	11	3
14	13	4	8
9		7	12
10	2	1	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

図3: 15パズル(問題D, 問題E)

ここでは、ある一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及びE)を用いて計測を行なった(図3)。また、比較のためSTB方式で同じ問題の計測を行なった。

表1からわかるように、プロセッサ台数が少ない場合にはLLS-GはSTB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、プロセッサ台数の多い場合にはLLS-Gの方がSTB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Eの台数効果をみると、STBではプロセッサ台数が増えると共に台数効果が頭打ちとなっているのに対し、LLS-G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、プロセッサ台数の増加について考える。

LLS方式では、一台あたりの分散コストがプ

表 1　LLS-G と STB の実行時間

(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

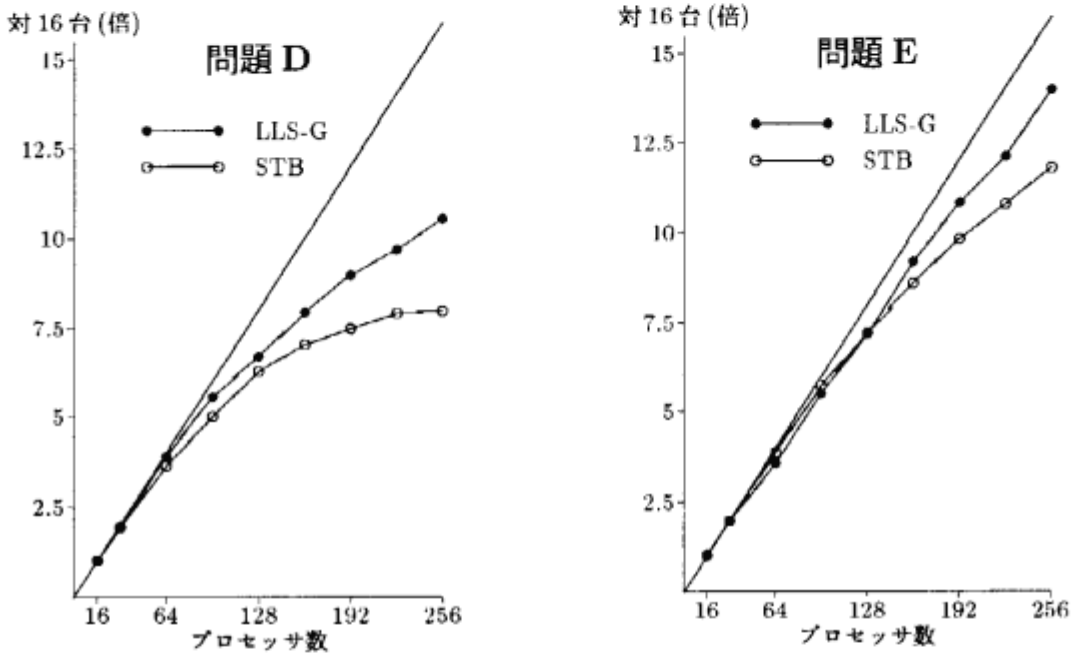

図 4:　LLS-G と STB の台数効果

ロセッサ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 (図 5)。

この二つのコストが交差する点 P が実際にプロセッサ台数が何台のところにあるかを知ることが、本研究における一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二つのコストが交差しないように思われた [3]。これは、STB 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 では実行時間に依存する関数になってしまうため、台数に応じた大きな問題を扱うほど、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なった LLS-G 方式の提案を行なった。

LLS-G 方式では、明らかに STB が有利である少ない台数の場合にも、最悪 1.25 倍程度の実行時間で抑えることができるようになっており、かつ、マシン台数がある程度を越えると逆に実行時間が速くなっている (256 台で 1.18 倍)。

また、問題 D と問題 E の違いは主にその問題サイズ (OR 並列型探索問題では、そのまま問題の並列度となって反映される) にあるが、この台数効果の違いについては、次のように考えることができる。

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「問題の並列度」と「負荷分

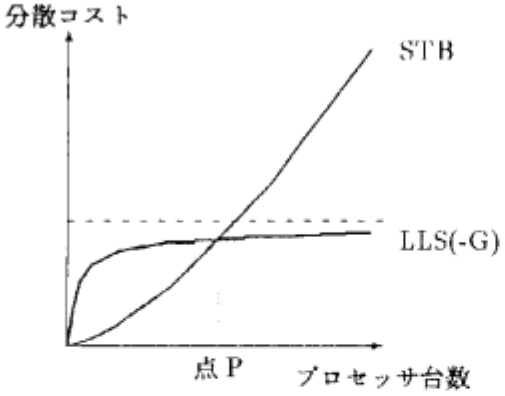

図 5: LLS(-G) 方式と STB 方式の分散コスト

散オーバヘッド」である。問題 D においては、問題の並列度がある程度少ないため、マシン台数が少ない間は問題の並列度が支配項となり、実行時間に占める負荷分散オーバヘッドが目立たない。しかし、ある程度以上に台数が増えると負荷分散オーバヘッドが支配項となり、実行時間が削減できなくなる。すなわち、問題 D における台数効果で、STB の台数効果が LLS-G の台数効果に較べて早く頭打ちとなるのは、問題の並列度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。

一方問題 E では、問題の並列度が充分あるため、並列度と負荷分散オーバヘッドの割合の逆転が起こりにくく、STB でも頭打ちになる台数がかなり 256 台に近くなっており、LLS-G の場合は、ほとんど、リニアな台数効果を維持することができている。

すなわち、LLS-G 方式では「世代」の導入により、負荷分散のオーバヘッドをある程度小さく抑えることで、図5の点 P が、問題 D では 64 台のところに、また問題 E では 128 台のところにあるということを、示すことを可能とした。

このように、LLS-G 方式は、LLS だけでなく、STB 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、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、LLS-G 方式の効果はいっそう期待できると考えられる。

5 おわりに

本稿では、「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することにより、その性能評価などを行なってきた。

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。

今後、LLS-G 方式について、本方式がどのような範囲の問題に適用できるかを明確にしていくと共に、さらに方式そのものに対する解析・考察を行ない、より、効率の良い動的負荷分散方式の研究・開発を行なっていきたい。

参考文献

[1] 古市昌一、磯和男、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」 *KL1 Programming Workshop '90*, pp. 1-9, 1990 年 5 月 .

[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『人沼』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 年 7 月 .

[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, CPSY92 9, 1992 年 8 月 .

[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 年 8 月 .

[5] H. Nakashima, K. Nakajima, S. Kondo, Y. Takeda, Y. Inamura, S. Onishi and K. Masuda. "Architecture and Implementation of PIM/m" In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pp. 425-435, June 1992 .