TR-0794

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

佐藤　令子、佐藤　裕幸（三菱）

August, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

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
A scheme for diffusional load balancing on loosely coupled multi-processors

佐藤令子、佐藤裕幸
Reiko Satoh, Hiroyuki Sato
三菱電機(株) 情報電子研究所
MITSUBISHI ELECTRIC Corp. Computer & Information Systems Laboratory
E-mail: {ezaki,hiroyuki}@isl.melco.co.jp Tel: 0467-46-3665

疎結合型大規模マルチプロセッサにおける負荷の分散と均等化を行なう方法として、マルチレベル動的負荷分散方式(MLB)、スタック分割動的負荷分散方式(STB)などの方式が発表されている。これらの方式は、負荷の分散を行なうための基本的な機構が要求駆動型であり、各プロセッサの忙しさを現在プロセッサ内に仕事があるかどうかで判断する方式である。本論文ではこれに対して、動的負荷浸透方式と名付けた、「要求駆動型でない」負荷分散方式を提案する。本方式は、各プロセッサの忙しさをある一定のタイミングで計測し、その結果をもとにより負荷の大きいプロセッサが負荷の小さいプロセッサに対して自発的に仕事を分散、プロセッサ全体の負荷を徐々に均等化していく方式である。本方式においては、少ない台数構成のマルチプロセッサの場合、負荷の均等化に用いるプロセッサ間通信のコストが全体として高くつきあまり良い性能が期待できないが、このコストの増加はプロセッサ台数の増加に対して緩やかであるため、より大規模なマルチプロセッサになるに従い、性能の改善が期待できる。本方式をICOTで開発した並列推論マシン PIM/m 上に実現し、黙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、128台以上のプロセッサを用いた場合、ユーザによるチューニングをまったく行なわずに STB 方式とほぼ同程度の絶対性能が得られた。

1 はじめに

個々のプロセッサを意識できないような大規模な並列マシンで処理を行う際に最も重要な課題の一つとなるのが、負荷の分散とその均等化である。

理想的には、このような負荷の分散と均等化は、システムあるいはマシン自身が自動的におこなうべきものであり、ユーザはあくまで、プロセッサの台数を意識しないで並列マシンを『効果的に』使用できるべきである。しかし現状では「どのような問題に対しても有効な負荷の自動的な分散と均等化」は、通信のコストが比較的高くつく疎結合型並列マシンにおいてはかなり難しく、問題を解くユーザ自身が問題毎に最適と思われる負荷分散を行う場合がほとんどである。

このような負荷の分散及び均等化を疎結合型並列マシンの上でプログラムの実行時に動的に行う方法としては、既にマルチレベル動的負荷分散方式(Multi-Level Dynamic Load Balancing : MLB 方式) [2, 3] 及びスタック分割動的負荷分散方式(Stack Splitting Dynamic Load Balancing: STB 方式) [4] の2方式が提案されている。

本稿では、これらの方式に対し新たなメカニズムで負荷の分散と均等化を行なう動的負荷浸透方式(Dynamic Local Load Spreading Method : LLS 方式) の提案を行なう。

なお、本研究は、ICOT[1]の再委託研究の一環として実施したものである。

2 従来の方式

2.1 MLB 方式

マルチレベル動的負荷分散方式(MLB 方式)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[2]である。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分割するプロセッサがボトルネックになるが、 MLB では、これを解消するために仕事の分割と供給を行なうプロセッサを階層的に(すなわち、マルチレベルに)複数台割り付けることによってこの解消を行なっている。

しかし、このような階層化を行なうことによって、何段の階層にするか、その階層をそれぞれどの深さに設定するか、それぞれの段における供給側のプロセッサを何台に設定するかなどのパラメタの調整が必要となる。さらに、全体のプロセッサ台数を変更するたびにこのような調整を度々行なう必要があり、仕事の供給を行なうプロセッサを特定したことが、 MLB のユーザプログラム

[1](財) 新世代コンピュータ技術開発機構
[2]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。

への適用を複雑にしていると言うことができる。

また、各プロセッサにおける、負荷分散の基準となる「忙しさ」は、

仕事を持っていない ＝ 暇である

仕事を持っている ＝ 忙しい

のように定義されており、事実上「忙しさ」の定義は行なわれていない。

2.2 STB 方式

スタック分割動的負荷分散方式 (STB 方式) はミネソタ大学の V. Kumar らによって考案された並列深さ優先探索アルゴリズム (Parallel Depth First Search : PDFS) [6, 7] の中で用いられている動的負荷分散方式で、各種の MIMD 型並列計算機上に実現されて良い台数効果が得られると同時に、台数拡張性に優れていることが報告されているものである。

STB では、MLB と同様、仕事のなくなったプロセッサが仕事を要求するという、要求駆動型の負荷分散を行なうが、特定のプロセッサが仕事の分割及び供給を行うのではなく、全てのプロセッサが仕事を供給すると同時に仕事を実行する。従って、MLB のように仕事の供給がボトルネックとなることはない。

しかし、逆に、仕事を供給するための特定のプロセッサを決めないことにより、供給を行なうプロセッサを選択する戦略が調整の必要なパラメタとなるだけでなく、結果的に自分以外の全プロセッサに対して仕事の要求をする可能性があるのでローカリティが失われてしまう。このようなローカリティの欠如は、大規模な並列プロセッサを扱う場合には通信距離や通信量の点で大きな問題となる。

また、各プロセッサにおける「忙しさ」は MLB と同じであり、事実上定義されていない。

以下では、前述のような MLB 方式、STB 方式の特徴を踏まえ、新たに設計を行なった LLS 方式について、その詳細なアルゴリズム及び PIM/m[9] 上での実装とその結果について詳述する。

3 動的負荷浸透方式 (LLS 方式)

MLB と STB はいずれも要求駆動型の負荷分散方式であり、この 2 方式の間には、対象ハードウェアとして 64 プロセッサ構成の Multi-PSI を用いた場合には、ほとんど性能差が認められないこと、またどちらの方式の場合にも線形に近い台数効果が得られることが報告されている [4]。

ただし、この二つの方式の場合、

- 「忙しさ」の定義がない。すなわち、負荷を「分散」しているが、「均等化」しているわけではない。従って、例えばプロセッサごとのメモリの使用量の偏りを緩和できない。
- プロセッサ間通信のローカリティがない (STB) か、ローカリティを得るために、多数のパラメータを調節する必要がある (MLB)。

ことがわかっている。

これらの点を考慮し、筆者らは、新たな動的負荷分散及び均等化方式として、動的負荷浸透方式 (LLS 方式) の設計を行なった。この方式は、MLB 及び STB のような要求駆動ではなく、各プロセッサが近接するプロセッサ群 (これを「近傍」と呼ぶ) に対する自分の相対的な忙しさを動的に計算し、その値により自プロセッサを近傍内では忙しいと判断した場合にのみ、自発的に仕事をわけていく方式である。

本章では、このような動的負荷浸透方式 LLS の考え方について「分散」と「均等化」の観点から述べ、さらに LLS でどのように実現しているかについて詳述する。

3.1 仕事の分割

負荷分散の対象となる「仕事」は、幾つもの「子供の仕事」に分割できることが前提となる。この時、各プロセッサでは「親」の世代の仕事に対して、一段処理を進めた「子供」の世代の仕事を生成するという処理が行なわれる。

LLS では (また MLB や STB でも) このように分割された粒度の均質でない「仕事」が分散の対象であり、このため、親の仕事から子供の仕事が比較的単純に生成できるような問題、すなわち各解が同時独立に求められる OR 並列型探索問題などに対しては、非常に自然に適用することができる。

3.2 分散の戦略

仕事の分散の戦略に関しては、MLB、STB は共に要求駆動型であった。この戦略はすなわち、仕事のなくなったプロセッサが他のプロセッサに仕事の要求を行なうというものであり、MLB と STB のこの面における差は、仕事を要求されるプロセッサがマネージャという形である程度固定されているか、自分以外の全員であるか、という点に集約されるだろう。

一方、LLS は、「要求駆動型でない」分散方式であり、仕事を余分に持っていると判断したプロセッサが自発的に近傍プロセッサに対して仕事を分配していく方式である。図 1 に LLS で初期状態から仕事が次第に浸透していく様子を示す。

仕事は、あるプロセッサが近傍の他のプロセッサよりも「自分は忙しい」と判断した場合にのみ分散される。例えば図 1 において、もっとも初期の状態 (1) にはすべての仕事は中心のプロセッサに集中している。これに対し、その近傍のプロセッサはまったく仕事をもっていないので、中心のプロセッサは自分が忙しいと判断し仕事を分配する。(2) はこの分配がなされた後の状態を示す。

(2) で新たに仕事の分配を受けたプロセッサは、その各々の近傍に仕事を持っていないプロセッサがいるので、やはり自分が忙しいという判断の結果、それぞれの近傍

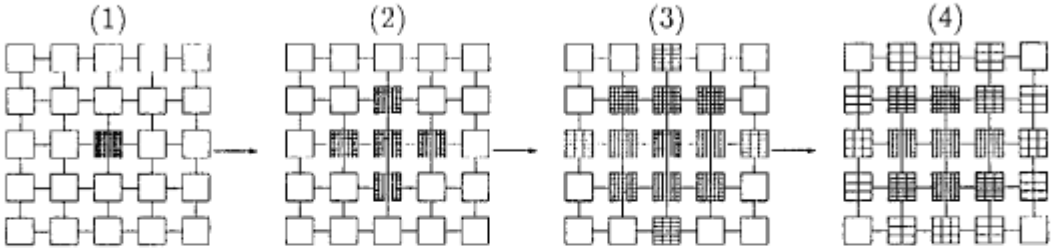

図 1: 仕事の分散と均等化

のプロセッサに仕事を分配する。この時中心のプロセッサとその近傍の間に「忙しさ」の差がなければ、これらのプロセッサの間では、どちら向きにも新たな分配は起こらない。

さらに、すべてのプロセッサでその近傍に対する自プロセッサの忙しさの評価が行なわれ、仕事の分配がすすむ (3) が、この時、中心のプロセッサで再度、近傍に対する仕事の分配が行なわれることに注意する必要がある。LLS では、近傍のプロセッサがある程度の仕事を既に持っている場合でも、自プロセッサが近傍に対して「忙しい」と判断すれば、仕事の分配は行なわれる。 LLS はこの点で MLB や STB と大きく異なっている。

LLS では、各プロセッサの「忙しさ」は、ある一定のタイミングで常時近傍のプロセッサに報告される。忙しさを報告する相手、及び自プロセッサの忙しさの評価の基準、として近傍のプロセッサのみを用いたのは、自プロセッサから遠距離にあるプロセッサの「忙しさ」の報告が、この報告自身のコストや遅れ、負荷の計算や遠距離への仕事の分散にかかるコストなどにより、ほとんど意味を持たない (すなわち、「報告の内容が古過ぎ」たり「そうまでして頼むよりも自分でやった方が安くつく」) ことが容易に想像できるからである。このため、プロセッサの全体の構成に対して「近傍」の概念を導入し、「分散」と「均等化」が近傍でのみ行なわれるような戦略とした。

なお、本論文での評価の際には、ハードウェアのローカリティを活かすという意味もあり、自プロセッサと物理的に直結しているプロセッサを近傍として扱うことにしている。

また、図 1の (4) で四隅のプロセッサに対する分配がなかなか起こらないことからもわかるように、分散する相手を近傍のプロセッサに限る LLS 方式では、(多くの場合立ち上がりにおける) 負荷の伝搬の遅れが問題となり得る。プロセッサ台数が何台以上になるとこの遅れが実際に問題となってくるかは、ハードウェアに依存する部分でもあり、今回の評価でも最大の関心事のひとつであった。

3.3 忙しさの定義と均等化

MLB や STB では、各プロセッサに仕事がなくなることにより仕事の要求がおき、この要求に対する仕事の分配を「均等化」と呼んでいる。このため、「均等化」は、あるプロセッサに仕事のなくなった時点で起きる。一方、LLS では仕事の均等化は「忙しさの報告」に基づいているため、仕事のあるなしにかかわらず「忙しさの報告」がなされる度に起きる可能性がある。従って、LLS では、MLB や STB と比較して早いサイクルで均等化を行なっていることになり、仕事の分散が早めに起こると考えられる。また、このような均等化を行なうと、ある時間断面における各プロセッサの持っている仕事が均等化されており、それらの消費するメモリもある程度分散されるという長所がある。

このような「均等化」をおこなうための方法として、例えば、各プロセッサ上に存在する仕事のキューの長さを平均化する、といった方式が考えられる。これは、「キューの長さ」すなわち「仕事の個数」を「忙しさ」の基準として採用していることになるが、「各仕事の粒度」がまったく考慮されないため、粒度が非常に不均質な問題の場合には、このような「キューの長さ」はプロセッサの忙しさを評価するためには不適切であると思われる。

これに対し LLS では、各プロセッサの「忙しさ」は、ある一定個数の仕事を実行するのに要する時間で定義する。この一定個数を示す初期値はすべてのプロセッサで同じであるが、実行時には動的に調整され、各プロセッサで異なった値となる。この値をセンチネル値と呼ぶことにすると、

各プロセッサの「忙しさ」 =

センチネル値 × 1個あたりの仕事に要する時間

のように定義できる。

LLS では、このような忙しさの定義を用いて、負荷の均等化を次のようなメカニズムで実行する。

各プロセッサには、実行可能なジョブのキューが存在しており、そのジョブのキューの中には、ある一定個数ごとに、自分の忙しさを近傍に報告するための区切り (タイミング) を示す「センチネル」がおかれている。

各プロセッサでは、ジョブ・キューからジョブが順次とりだされて実行される。実行が進み、「センチネル」がジョブ・キューからとれると、近傍のプロセッサに対して自分の「忙しさ」、すなわち、センチネル個数の仕事の実行が終ったことを報告する。

この近傍からの報告は、一台のプロセッサにおいては、ジョブの実行中に順次受け付けられる。図 2は、 PE1~PE4

の4台の近傍プロセッサを持つあるプロセッサ(PE0とする)が実行中に「忙しさの報告」を受け、それを自分の内部に記録し、それを元に自分の報告のタイミングを調整する様子を示したものである。

図2の(1)では、PE0はPE4からの報告を受けとる。この時この報告値は、PE0自身のジョブの実行時間に対する相対的な時間で充分であり、PE0でその報告を受けるまでに実行できたジョブの個数(すなわち実行ジョブ数1)とすることができる。同様に、(2)~(4)ではそれぞれ、PE1からの報告が値3で、PE2からの報告が値4で、PE3からの報告が値5で記録できる。

(5)では、PE0自身でセンチネルが検出され、近傍のプロセッサに対するPE0からの「忙しさの報告」が起こる。これによって、全員の「忙しさ」が揃うので、自プロセッサを含む近傍の「忙しさ」を計算することができる。この「忙しさ」は、自分を含む近傍プロセッサからの報告に刻印された値を平均することで求められ、この場合、$(7+3+4+5+1)\div 5=4$であるから、この近傍の忙しさはPE0の尺度で「4」と定義することができる。

(6)は次のサイクルの開始を示しており、PE0で自分の忙しさをその近傍の平均(すなわちこの場合は4)に合うよう、センチネル値を変更してジョブの実行を再開する直前の様子である。

なお、近傍からの報告が揃う前に自プロセッサでセンチネルが検出された場合には、近傍からの報告が揃うまで負荷の均等化は行なわれない。

近傍に属する各プロセッサの忙しさは平均値に対するそれぞれの忙しさで表すことができる。図2に示した例では、平均値4に対して、自プロセッサの忙しさは7であり、自プロセッサが近傍の中で「より」忙しいことがわかる。この時PE0は、自分が保持しているジョブの7分の4を残し、7分の3を近傍のプロセッサに分配する。

この場合に分配の対象となるのは平均よりも忙しくなかったPE1、PE2、PE4の3台であり、それぞれの忙しさの割合に応じて分配される個数が計算される。一方PE3はこの近傍では忙しい方に属するので、仕事の分配の対象からははずされる。

LLSでは、このようなメカニズムを用いて、各プロセッサに現在存在するジョブをある個数実行するのにかかる時間を一定に揃えていくことで全体の負荷の均等化をはかっている。これは、現時点のジョブの実行に要する時間から、残りのジョブ(すなわち新たに生成される子供のジョブ)の実行に要する時間を推測していることに他ならない。このため、LLSでは仕事の分配を行なう際に、自プロセッサのキューに存在するジョブの性質が子供のジョブに遺伝する可能性(すなわち、同じ個数を実行するのに同じだけの時間がかかる可能性)を期待し、自プロセッサ内にできるだけ子供の世代のジョブを残す、というヒューリスティクスを導入している。すなわち、現在実行しているジョブと同世代のジョブと子供の世代のジョブを持っている時、同世代のジョブから先に分配

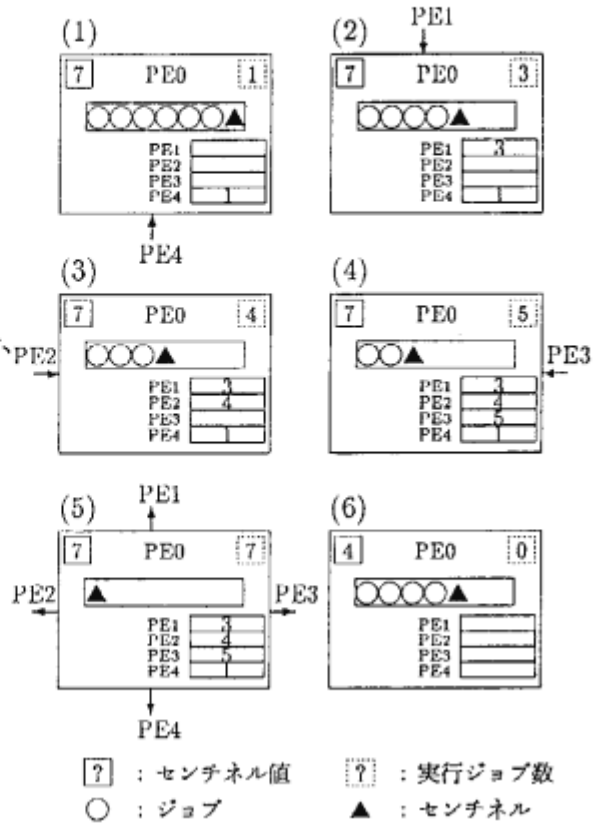

図2: 忙しさの報告

が行なわれることになる。

各プロセッサが、それぞれ上記のメカニズムに基づいて近傍との忙しさの均等化をはかることにより、(MLBやSTBと比較して)比較的ゆっくりとではあるが、プロセッサ全体の「忙しさ」の均等化をはかることが可能である。

4 PIM/m上での実現と詰込みパズルへの適用

本節では、PIM/mでのLLSの実現方式と詰込みパズルへの適用例を述べる。

4.1 PIM/mとKL1

計測に用いたPIM/m[9]は疎結合型の大規模マルチプロセッサマシンであり、最大256台のプロセッサ(PE: Processing Element)を高速な専用ネットワークで二次元格子状に結合した並列推論マシンである。PIM/m上でのプログラミングは、並列論理型言語KL1[1]を用いて行なわれ、KL1自身はマイクロプログラムにより実行される。1PEあたりのKL1の性能は約530KRPS[3][4]で

[3]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。

[4]最近、約610KRPSにチューンアップされた。

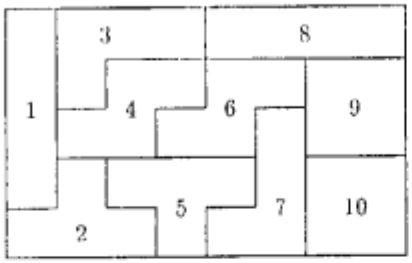

図 3: 詰込みパズル

あり、最大規模の構成の場合、マシン全体で約130MRPSの性能が得られることになる。

並列論理型言語 KL1 は Flat GHC をベースとした comitted choice language であり、OS の記述が可能なように、莊園(メタプログラム)機能、プラグマ(ユーザ・プログラムへの付加による実行制御)機能などの拡張がなされている。PIM/m 上では実際に KL1 で記述された OS・PIMOS[1] が動作しており、今回の計測は、PIMOS 及び KL1 の計測機能を利用して行なったものである。

4.2 詰込みパズル

詰込みパズルは、4つの正方形を組み合わせた様々な形のピースを長方形のケースに詰め込む問題である(図3)。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、このパズルの原型は、各ピースが5つの正方形から生成され得るすべての異なった形からなり、「ペントミノ」と呼ばれている。)

詰込みパズルを解く手順は、次のようになる。まずケースの隅にピースを置く。次に別のピースをその隣に置く。ピースは向きによって異なった形をしているので複数の置き方がある場合がある。その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となる(すなわち、ここで OR 木が生成される)。ピースが置けた時には、別のピースに対して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ていき、ケースが全てピースで埋まったらそれが解となる。また、途中で置けるピースがなくなった場合にはその探索枝は終了であり、解はない。ここでは、ピースをひとつケースに詰める事が、探索を一段深めることに相当する[5]。

詰込みパズルにおいては、ピースを置いた個々の状態が分散の対象となる。すなわち、ピースを i つ置いた状態のケースがピースを 2 つ置いた状態のケースにあたる子供データを生成する処理を繰り返しつつ探索が進むが、この子供データの生成処理が分散の対象である。

[5]詳細な問題の特徴、及び解析は [4] に詳しい。LLS では、問題の特徴が及ぼす影響をできるだけ無視できるよう努力しているため、詳細は言及しない。

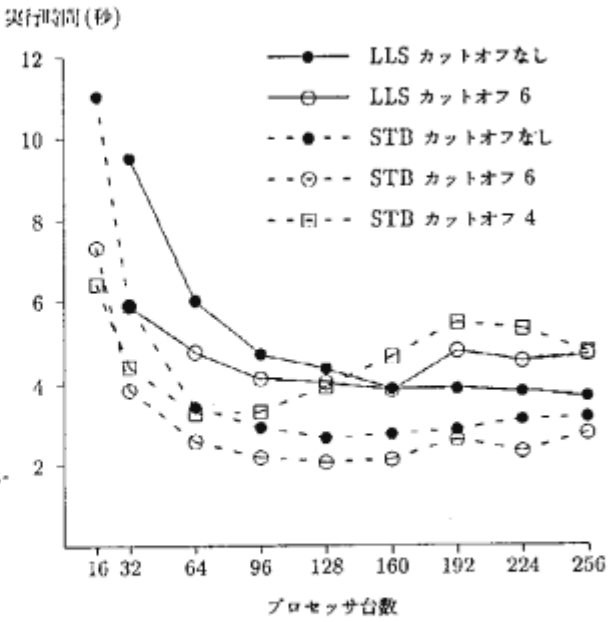

図 4: 各方式の実行時間

4.3 LLS の実現

LLS は PIM/m 上において、各 PE のそれぞれに分散と均等化を行なうプロセスを、ハードウェアのコンフィギュレーションに従って、二次元格子状に手を繋いだ状態で生成する。「忙しさの報告」と「仕事の分散」は、この手を通じて行なわれる。LLS のプログラム自身は並列論理型言語 KL1 を用いて記述を行なっているため、分散その他を行なうためのプロセスは KL1 のプロセスに、手は KL1 のストリームにマッピングしている。

LLS の長所のひとつは負荷分散方式自身にかかわるパラメタが少ないことであり、この時点で特にユーザが調節を考えなければならないパラメタや戦略は存在しない。

5 性能の測定

図4で示した実行時間は、詰込みパズルの全解探索問題の実行を開始してから全ての解が一台のプロセッサに返されるまでのものである。MLB や STB で実際に計測して得られた結果 [4] によれば、実行時間はあきらかに問題の最小粒度に依存している。すなわち、ある程度以上粒度の小さな問題を分散しようとすると、分散するためのコストの方が高くついてしまう。このため、探索木の深さがある程度深くなると、そこから先は分散の対象となる子供データを生成しない方が効率がよい場合があり、この分散を行なわない深さのことをカットオフと呼んでいる。

Multi-PSI の 64PE での最適のカットオフは、MLB、STB の場合には、共に実測の結果 4 (すなわち探索木のルートから数えて 4 段目まで) であることがわかっており、本詰め込みパズルの場合最大の深さは 10 であるか

ら[6]、カットオフがかなり浅い所にあることがわかる。このような分散対象である問題の粒度への依存は、LLS を用いた場合にも充分考えられるので、評価のための計測はカットオフ 6 とカットオフなしで行なうことにした。また、比較のため、STB 方式で同様の計測 (すなわち、最適にチューニングを行なった場合と、チューニングを行なわなかった場合についての計測) を行なった。

6 考察

本節では、前節での結果に基づき、STB との比較を行ないながら LLS の評価を行なう。

6.1 実行時間と通信量

前節の計測結果からわかるように、LLS の絶対性能は、あるパラメタにおける STB のものより劣っている。しかし、プロセッサの構成台数が増えるに従ってこの差は狭まり、同一条件 (128PE、カットオフ 6) の下では、1.96 倍程度となっている。 そこでここでは、両方式において、『プロセッサの構成台数が増える』ということが何を意味するかについて考察する。

LLS と STB における、負荷バランスのための通信を単純に比較すると、LLS では常時「忙しさの報告」を行なう分だけ通信が多いことが直観的に理解できる。一方、実際に仕事を分配するための通信は、LLS では、近傍のプロセッサに対するジョブの引渡しだけであるが、STB では、仕事を分配してくれるプロセッサが見つかるまでの数回 (または数十回か数百回かもしれない) の通信になる。すなわち、STB 方式を用いた時に、仕事の要求に要する一台あたりの通信は、構成台数 N に対して $O(N)$ であるが、LLS では、近傍プロセッサの数は一定であるため $O(1)$ であり一定である。

実際に、STB においては、プロセッサ台数がある程度をこえると、プロセッサ一台あたりの通信量が急激に増えることで全体の実行時間が遅くなり、あるピークを境に絶対性能が逆に悪くなるという現象が見られる。一方 LLS ではこのような急激な増加は起こらない。

この観点から、少なくとも、LLS で STB 以上の効果をあげるためには、ある一定台数以上 (本論文で行なった評価によれば 128PE 以上) の構成のマルチ・プロセッサである必要があり、LLS はより大規模マシンに有効な負荷分散方式であるということができる[7]。

なお、MLB の場合は、負荷分散のための通信を論理的に構成したプロセッサ・ツリー上で行なうが、プロセッサ結合の物理的な結合からくるローカリティを用いない負荷分散方式であるため、スケーラビリティの限界がある。

6.2 パラメタチューニングの容易さ

MLB、STB いずれの方式においても、ユーザはより良い性能を得るために幾つかのパラメタのチューニングを行ない、「最適な」パラメタを得る必要があった。このようなチューニングは、その問題が既に解けることがわかっていてそれを繰り返し行なえる場合にのみできることであり、初見の問題や、「一度しか解かない問題」に対してこのような細かいチューニングを行なうことは難しい。

MLB 方式の場合には、仕事の供給を行なうプロセッサの台数、階層化をする際に階層を何段にし、それぞれの階層の深さを幾つにするかなどのパラメタがあり、STB の場合にはカットオフと仕事の要求先の決定方式というパラメタがあった。また、64PE 構成程度までは現れなかった PE 数の増加に伴う実行性能の低下などがあることから、実際にはプロセッサ台数もパラメタの一つであることが判明した。

LLS は本来、ユーザを上記のようなチューニングの負担から解放し、初見の問題に対してある程度の (できれば『ほぼ最適な』) パフォーマンスを得られる負荷分散を行なうための方式である。よって、このようなチューニングは一切不要のはずであった。

しかし残念なことに、実際には少ない PE 数で LLS を用いた場合には、分散する問題の粒度が分散するコストに較べて小さくなり過ぎて分散のコストが高くつき、部分問題の最小の粒度を考慮することなしに『最適な』パフォーマンスを得ることができない。すなわち、128PE 以下の場合には、カットオフなしとした場合にくらべるとカットオフを 6 に設定した場合の方が 20% 程度絶対性能がよく (64PE の場合)、プロセッサの台数に対して問題のカットオフを調節することでも、より良い性能を得られるという点に関しては他の方式とまったく同様であることがわかる。ただし、128PE 以上の場合にカットオフなしとすると、絶対性能が悪くならないという利点があり、このことから、LLS においては、充分なプロセッサ台数を得られることを前提とすれば、問題の最小粒度を考慮する必要がない、という長所があると思われる。

この 128PE という台数は、STB 方式を用いた場合にピーク性能が得られる台数 (STB では 128PE を境に絶対性能が低下し始める) と一致しているため、PIM/m ではこの台数を、STB のような「要求駆動型」の負荷分散方式を用いるか、LLS のような「要求駆動型でない」方式を用いるか、の判断の基準となる台数と考えることができるかもしれない。

7 おわりに

本稿では、拡散型の動的負荷均等化の一方式である LLS 方式の提案と実装、および簡単な評価を行なった。本方式は要求駆動型でない動的負荷分散を行なうものであり、ハードウェア結合のローカリティを活かし、ユーザが適用する際のパラメタの調整を極力不要とした方式である。

---

[6]本詰め込みパズルのピースの数は 10 であるので、これより深くなることはない

[7]大規模なマシンで台数効果をあげるためには、もちろん、対象とする問題自身も充分大規模である必要がある。

本方式を PIM/m 上で詰込みパズルの全解探索問題に適用したところ、構成プロセッサ数が 128 台以上の場合には、ユーザによるパラメタのチューニングなしに、従来の方式とほぼ同程度の性能が得られることがわかった。

本方式は、MLB 方式及び STB 方式に比べると負荷の均等化を行なうためのコストが高くつくと予想され、当初その有用性が疑問視されていたが、実際に STB に対して 2 倍程度の絶対性能が得られることがわかったことは、大きな収穫であった。

なお、今回本文中には述べなかったが、要求駆動型でない負荷分散方式におけるもうひとつの大きな問題として、仕事の振動が起こりやすい、ということがある。仕事を投げ合うことにより、各プロセッサでは一見充分に仕事をしているように見えながら、実際の処理がほとんど進行しない、といった現象が LLS のデバッグの過程でも度々見られ、この現象は処理すべき部分問題数が増えるに従って、さらに顕著になると考えられる。

このような仕事の投げ合いを防止するためのアルゴリズムの再検討を行ない、LLS 方式をより大規模な並列プロセッサに対して実用に耐え得る方式とすることが、今後の大きな課題である。

8 謝辞

本研究において、MLB 方式及び STB 方式の詳細な評価結果を提供いただくと同時に、本方式の評価に対する御助言をいただきました三菱電機・情報電子研究所の古市昌一氏、また、数々の貴重な御助言をいただきました同研究所の中島克人氏、中島浩氏らに深く感謝致します。

参考文献

[1] 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*, pp. 230–251, 1988.

[2] 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. of the 2nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 50-59, Mar. 1990.

[3] 古市昌一、齋和男、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

[4] 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 年 7 月

[5] K. Kimura and N. Ichiyoshi "Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme." *to appear at DMCC6*, 1991.

[6] V. Kumar and V. N. Rao. "Parallel Depth-First Search, Part I: Implementation." In *International Journal of Parallel Programming*, 16(6), pp. 479–499, 1988.

[7] V. Kumar and V. N. Rao. "Scalable Parallel Formulations of Depth-First Search" In *Parallel Algorithms in Machine Intelligence and Vision*", Springer-Verlag, pp. 1-41, 1990.

[8] K. Kumon, H. Masuzawa, A. Itashiki, K. Satoh, and Y. Sohma. "Kabuwake: A new parallel inference method and its evaluation" In *Proceedings of COMPCON 86*, March 1986.

[9] 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.