計算機アーキテクチャ　63－20
(1986・11・21)

データ駆動計算機における命令水準の静的負荷分散

Static Load Allocation in Dataflow Machines

大塚　喜久† 坂井　修一†† 弓場　敏嗣††
Yoshihisa OTSUKA Shuich SAKAI Toshitsugu YUBA

† （株）神戸製鋼所
KOBE STEEL, LTD.

†† 電子技術総合研究所
Electrotechnical Laboratory

1．はじめに

現実のプログラムには命令水準の並列性の多い部分と少ない部分が混在し、並列計算機では並列性の少ない部分が実行性能の低下の原因となる。また基本プロセッサ（ＰＥ）の台数が多い場合、プログラムの並列性は多くてもＰＥ１台あたりの並列性は少なくなり、負荷のばらつきが処理時間に大きな影響を及ぼす。データ駆動方式は命令水準の並列性を自然に抽出できることが特徴であるが、上記の状況に対しては得られた並列実行可能な命令をＰＥに効率良く分散する必要がある。

負荷分散[1]は各命令のＰＥへの割り付けを実行時に行う動的なもの[2]と、実行以前にコンパイラによって行う静的なもの[3,6]とに分けられる。

まず、命令水準の負荷分散を考える。各プロサッサに命令水準の待ち合わせ機構を持つデータ駆動計算機では、各命令を発火させるためのトークン対は同じＰＥに向かわねばならない。この管理を行いつつ動的に負荷を分散するにはスケジューラの負荷が重くなり、実行時のオーバーヘッドが増す。また、動的な負荷分散では実行時の局所的な情報のみを利用するのに対し、静的な負荷分散ではプログラム全ての情報を利用することができる。

一方、関数水準の負荷分散では、静的な負荷分散を行うためには各関数の依存関係、実行回数、実行時間を実行以前に把握する必要がある。しかし、プログラム中に再帰呼び出し、ループ、条件分岐が存在すれば、これらの把握は困難である。

これらの点から、静的、動的な負荷分散は共に必要であり、命令水準では静的な負荷分散が、また関数水準では動的な負荷分散が有効である。本稿では、データ駆動計算機上で各関数を効率よく実行させるための命令水準の静的負荷分散方式について論じる。

本問題と類似した問題としてマルチプロセッサシステムにおけるスケジューリング問題がある。これらの問題はほとんどの場合ＮＰ完全な問題に属し、現実的な時間内に最適解が求まるのは特殊なケースに限られる[1]。そのため、一般的な問題に対しては近似解を求めるヒューリスティックなアルゴリズムが研究されている[3]-[5]。これらのアルゴリズムは、各タスクにレベル（優先度）を付け、実行可能となったタスクの中で最もレベルの高いタスクにプロセッサを割り当てる方式をとっている。笠原ら[4]により提案されている CP/MISF法では、レベルを各タスクから終了までの最短時間として定義し、レベルが同じ場合には直接の後続タスク数の多いタスクを優先させている。また所ら[5]はデータ駆動計算機に対してワーキングセットの概念を導入し、プログラムの１次記憶への読み込み戦略として類似した方法を用いている。

従来の負荷分散（スケジューリング）問題は各タスクを実行するプロセッサと実行順序を決定するものである。しかしデータ駆動計算機では、発火条件によりタスクに対応する命令の実行順序が動的に定まり、実行順序の指定をスケジューラは行わない。また、タスクグラフに対応するデータフローグラフにはループが存在し、さらに命令水準のデータ駆動計算機ではＰＥ間の通信時間も無視できない。これらの相違により、データ駆動計算機では上記スケジューリン

グアルゴリズムとは異なったアルゴリズムが必要である。

本報告では、ループを含むプログラムに対し、ＰＥ間の通信時間を考慮に入れた静的負荷分散方式の提案及びその評価を行う。２章ではまず問題の設定を行う。３章ではデータフローグラフにおける各ノードのレベルを、また４章では通信コストを定義する。５章ではレベル、通信コストを用いた静的負荷分散方式を提案し、その評価を行う。

２．問題の定義

対象とするデータ駆動計算機は図１に示すようにｎ台のＰＥがクロスバーネットワークで結合されており、下記の条件を満たすものとする。これらの条件は電総研で開発中のSIGMA-1[7]のアーキテクチャをもとに設定した。

1)命令水準のデータ駆動方式
2)タグ付きトークン方式
3)異なるＰＥ間のデータ転送時間は一定
4)各ＰＥは待ち合わせ機構および実行部を持つ
5)プログラムは全てのＰＥに重複して格納

プログラムの並列性がＰＥ台数に比べ十分に多い部分では全てのＰＥはアイドルとはならず最大効率が発揮できるが、並列性の少ない部分では並列実行可能な命令を各ＰＥに分散し、かつＰＥ間の通信コストを少なくする必要がある。また、データ駆動計算機では実行順序の指定をスケジューラが行わない。これらの状況から、データ駆動計算機の命令水準の負荷分散、即ちＰＥ台数およびデータフローグラフが与えられた場合、各ノードを処理するＰＥを決定する問題を考える。

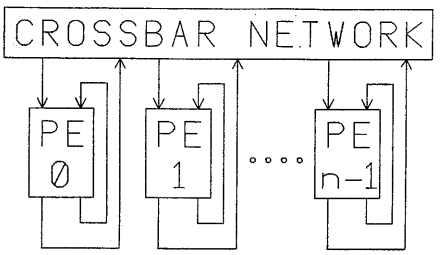


図１　データ駆動計算機

ここで本報告では解析、評価を行うにあたり、次の仮定を置いた。

1)各命令の実行時間は等しい
2)データフローグラフは論理的に多重ループ含まない有向グラフ

上記2)は高級言語で記述したプログラムに多重ループが含まれないことを意味する。（高級言語で記述した）プログラムが一重ループであっても、対応するデータフローグラフ上では多重ループとなる（図６、７を参照）。今後はプログラムとデータフローグラフを区別して用いる。

３．レベル付け

3.1.レベルの種類

負荷分散に先立ちデータフローグラフの状況を把握するために、各ノードに対して下記のレベルを付ける。レベルの名前および概要は所ら[5]と同じであるが、ループに含まれるノードをスカラー値で定義した。これらの例を図２に示す（図２は図６のプログラムのデータフローグラフである）。

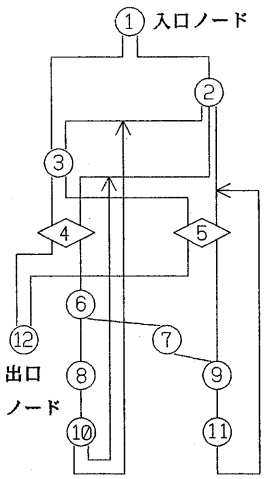
1)E-level　:入口ノードから各ノードを実行するまでの最短時間をあらわす。ＰＥ台数が無限にあり、通信時間を無視した場合の各ノードの実行時刻となる。
2)L-level1 :各ノードから出口ノードを実行するまでの最短時間をあらわす。ただし下位に存在するループを回らない場合の時間とする。このレベルが大きなノードは後続のノード数が多く、有利な状況にＰＥを割り付ける必要がある。
3)L-level2 :L-level1と同様。ただし、下位のループは１回回る場合時間とする。
4)Loop-Flag:このフラグでループに含まれるノードと含まれないノードを区別する。また、各繰り返しで独立に実行可能なノードを区別する。

これらのレベルの計算方法は次節以降に示す。これらのレベル付けはノード数をｎとしてＯ（ｎ²）の計算時間で処理可能である。

3.2. E-level(Execution Level)

E-levelは入口ノードから当該ノードを実行するまでの最短距離をあらわす。ただし上位にループがある場合にはループを回らない場合の距離とする。データ駆動計算機では、当該命令に必要なデータがそろうことが実行制御メカニズムであるので、概ねこの E-level順に実行は進む。

E-levelの計算方法は入口ノードのレベルをＯとし、その他のノードのレベルは上位のノードのレベルの最大値に１を加えた値とする。計算アルゴリズムを図３に示す。



ノード	E	L1	L2	FLG
1	0	4	9	N
2	1	3	8	N
3	2	2	7	G
4	3	1	6	G
5	3	1	6	G
6	4	5	5	G
7	5	4	4	P
8	5	4	4	G
9	6	3	3	G
10	6	3	3	G
11	7	2	2	G
12	4	0	0	N

図２　レベルの例

1)入力のないノードのレベルＥを０とする
2) i ＝ 1
3)実行のために必要な上位ノードに全てｉ未満のレベルが付いているノードのレベルＥをｉとする。
4) i ＝ i ＋ 1
5)全てのノードにレベルが付くまで 3)、4)を繰り返す。

図３ E-levelの計算アルゴリズム

3.3. L-level1(Latest Execution Level 1)

L-level1は当該ノードから出口ノードを実行するための最短距離をあらわす。ただし下位にループがある場合にはループを回らない場合の距離とする。L-level1の計算方法は出口ノードのレベルを０とし、その他のノードのレベルは下位のノードのレベルの最大値に１を加えた値とする。ただしループが存在する場合には工夫を要する。計算アルゴリズムを図４に示す。

1)出力のないノードのレベルL1を０とする
2) i ＝ 1
3)出力先のノードに全てｉ未満のレベルが付いているノードのレベルをｉとする。ただしループを構成するためのSWITCHノードに関しては、片方の出力先のレベルが付いていればレベルを付ける。
4) i ＝ i ＋ 1
5)全てのノードにレベルが付くまで 3)、4)を繰り返す。

図４ L-level1の計算アルゴリズム

1)出力のないノードのレベルL2を０とする
2) i ＝ 1
3)出力先のノードに全てｉ未満のレベルが付いているノードのレベルをｉとする。ただしループを構成しているswitchノードに関しては片方の出力先のレベルが付いていれば予備的にレベルを付ける。予備的にレベルを付けているノードからループの外には飛び出さない。ループ内において、 E-levelが大きくなるノードには通常通りにレベルを付け、また E-levelが小さくなるノードには予備的にレベルを付ける。レベル付けを続ける際に、予備的にレベルを付けたノードには上書きの形でレベルを付ける。
4) i ＝ i ＋ 1
5)全てのノードにレベルが付くまで 3)、4)を繰り返す。

図５ L-level2の計算アルゴリズム

3.4. L-level2(Latest Execution Level 2)

L-level2はL-level1とほぼ同じであるが、下位にループがある場合にはループを１周回る場合の距離とする。L-level2の計算方法もL-level1の計算方法とほぼ同じであるが、ループの部分ではループの路長を計算する必要がある。計算アルゴリズムを図５に示す。

3.5. Loop-Flag

ループを含むプログラムの負荷分散には、各ノードに対し、ループに含まれるか否かの区別、および各繰り返し毎に独立に演算できるか否かの区別が必要となる。

まず（高級言語で記述された）プログラムにおけるループがデータフローグラフではどの様に記述されるかを考えよう。例として図６のプログラムは図７のようにデータフローグラフとして記述される。図６におけるループのアンフォールディング[8]の部分および和の計算をする部分はグラフ上でもループとなるが、積の計算はグラフ上ではループとならない。グラフ上でループを形成した部分は前回の繰り返しとデータ依存関係があるが、ループを形成しない積のノードは繰り返し毎に独立に実行可能である。繰り返し毎に独立に実行可能なノードが複数ある場合には、これらのノード列上をパイプライン的に処理が進むことによってループ毎の並列性が得られる。

これらの状況をLoop-Flagによって判別する。

Flag=N ： プログラムにおけるループに属さないノード

Flag=G ： グラフにおけるループに属するノード

Flag=P ： プログラムにおけるループに属するがグラフ上のループに属さないノード

図７の例では、斜線をいれたブロックのノードがＧ、積のノードがＰ、その他のノードがＮである。

```
for(s=0,i=1;i<=n;i++) s=s+2*i;
```

図６　和のプログラム

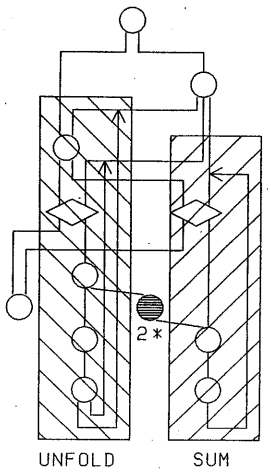


図７　和のデータフローグラフ

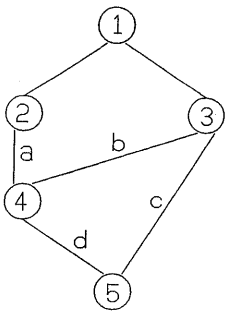


図８　通信コストの定義

４．通信コスト

図１のデータ駆動計算機では異なるPE間でのデータの転送には一定の時間を要する。通信コストは、各ノードを処理するＰＥと上位のノードを処理するＰＥとの関係によって定義する。

この定義の概念を図８のグラフを用いて説明する。図８のノード２、３は並列に処理されるとする。ノード４はトークンａ，ｂの到着によって発火するが、どちらかのトークンの転送に時間を要しても発火の時刻が遅れる。一方、ノード５はトークンｃ，ｄの到着で発火するが、ｃのトークンの転送に時間を要してもその間にノード４の実行が必要なので転送の遅れは問題とはならない。図８のノードが全て同一ＰＥによって処理されれば、全てのノードに対し通信コストは０とする。ノード４に関しては、処理するＰＥとノード２、３を処理するＰＥが共に異なれば通信コストは２とし、ノード２、３の片方を処理するＰＥと異なれば通信コストを１とする。ノード５に関しては、処理するＰＥとノード４を処理するＰＥが異なる場合のみ通信コストを１とする。

この定義の詳細を図９に示す。ただしこの定義では上位のノードを処理するＰＥの情報が必要なので、各ノードをＰＥに割り付ける際にこの定義を用いる。

5．負荷分散方式

まずループの無いデータフローグラフに対する負荷分散を行う基本方針について検討し、次にループに対する拡張方法を検討する。

5.1.基本方針

ＰＥ間のデータ通信時間を考慮にいれた負荷分散を行うために、データフローグラフの入口ノードから順に下位のノードに向かってＰＥの割り付けを行う。ここで、通信時間を最小にすることのみを考えると、全てのノードを１台のＰＥに割り付けることになるので、可能な限り多くのＰＥに割り付けることを最優先する。即ち、上位の全てのノードにＰＥが割り付けられているノード群に対し、ある選択基準によりノードと割り付けるＰＥを選択する。このアルゴリズムを図１０に示す。

ここで、選択基準として、下記の３方法が考えられる。

Ｔ）　通信コストを最小
Ｅ）　E-level を最小
Ｌ）　L-level を最大

T)は通信コストを抑えることを目的とする。

E)は、実行は E-level順に行われることが期待されるので、各時刻において最大の並列性を引き出すことを目的とする。

L)は割り付け可能なノードの内、L-level の大きなノード、即ち出口ノードとの距離が最も長いノードを有利な状況に割り付けることを目的とする。

１）全てのノードに対して　　cost=0
２）当該ノードの全ての入力アークに対して次の演算を行う
　当該ノードとアークの出側ノードとの E-level の差が１で、かつ
　当該ノードとアークの出側ノードを処理するＰＥが異なれば
　cost=cost+1

図９　通信コストの定義方法

これらの方法単独では、ノードは一意的には選択されない。よって、上記のある基準を用いて複数のノードが選択された場合、これらのノードに対し他の基準を適応することができる。即ち上記の順に適用する場合を TELと書くと、基本方針として TEL, TLE, ETL, ELT, LTE, LETの６通りの組み合わせが考えられる。この中でＬレベルを最優先する考え方は従来研究されているマルチプロセッサシステムにおけるリストスケジューリング[2,3] の基本的な考え方と一致する。

5.2.基本方針の検討

検討には電総研で開発中のSIGMA-1[7]のソフトウェアシミュレータを用いた。説明のためにＰＥ台数を２台としたがこれ以上のＰＥ台数でも同じ議論ができる。下記のベンチマークプログラムを用い5.1.節の６通りの基本方針による実行時間を測定した。結果を表１に示す。

ﾌﾟﾛｸﾞﾗﾑ1：１３個のノードを逐次的に並べたグラフ逐次処理の典型例（逐次）
ﾌﾟﾛｸﾞﾗﾑ2：二進木状に並列度を１から３２ま

１）全ＰＥをidleとする。
２）全ての割り付け可能なノード*)を検出する。もし無ければ、終了。
３）割り付け可能なノードからノードを１つ選択する。
４）選択されたノードをidle状態ＰＥに割り付ける。選択されたノードを実行可能なノードから除く。割り付けられたＰＥをbusyとする。
５）idle状態のＰＥがない場合は１）へ
６）割り付け可能なノードがない場合は２）へ
７）３）へ
（*:ＰＥの割り付けがされていないノードの内、入力アークの無いノード又は発火に必要なトークンの送り元のノードにＰＥが割り付けられているノード）

図１０　静的負荷分散の基本アルゴリズム

で広げ、さらに折り返して並列度を１まで戻したグラフ。デバイドアンドコンカー法に代表される並列プログラム（二進木）

ﾌﾟﾛｸﾞﾗﾑ3：比較的大きなサブグラフに分割できるプログラム（図１１）

ﾌﾟﾛｸﾞﾗﾑ4：並列性の変化の大きなプログラム（図１２、破線のアークを含まず）

ﾌﾟﾛｸﾞﾗﾑ5：並列性の変化の大きなプログラム（図１２、破線のアークを含む）

これらのプログラムは人工的で規模の小さなものであるが、本方法はプログラム中の並列度の少ない部分を対象としているので、これらのプログラムによる検討は十分に意味があると考えられる。比較対象として、ＰＥが１台の場合（1PE)、及び乱数を用いて負荷分散を行った場合(ran)の結果も併せて示す。

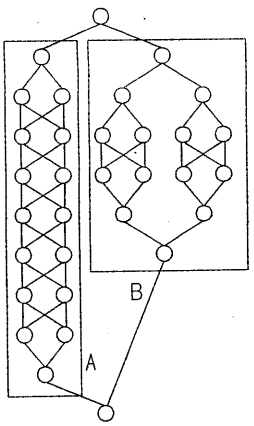


図１１　プログラム３

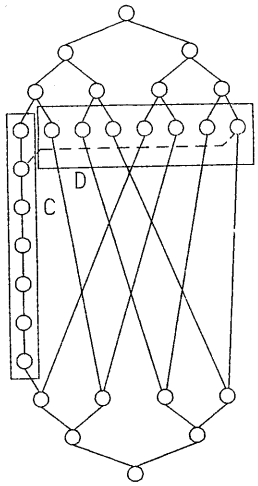


図１２　プログラム４，５

表１　各基本方針による実行時間

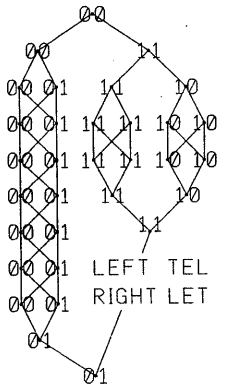
Program	1PE	ran	TEL	TLE	ETL	ELT	LTE	LET
1逐次	26	34	26	26	26	26	26	26
2二進木	135	94	78	78	78	78	78	78
3図11	64	53	44	44	47	46	47	47
4図12	50	44	39	39	39	39	35	37
5図12+破線	52	46	41	41	41	41	45	46

ランダムハッシュでは、プログラム１のような並列性のないプログラムはもちろん、プログラム２、３のような並列性のあるプログラムに対してもかなり悪い結果となっている。

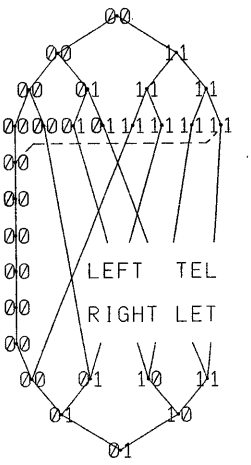
プログラムが比較的大きなサブグラフに分割できる例としてプログラム３を考える。TEL,TLE では各ＰＥの受け持つノードがサブグラフ（図１１におけるＡとＢのブロック）に対応し、処理も速い。各ノードの割り付け結果例として、TEL,LET による結果を図１３に示す。ノードの左側の数字はTEL により割り付けられたＰＥの番号を、また右側の数字はLET により割り付けられたＰＥの番号を示す。この例から通信コスト（Ｔ）を最優先するとサブグラフへの分割が行えることがわかる。

次にプログラム４、５について考える。 TEL,LETによる割り付け結果を図１４に示す。 L-levelを最優先した場合(LTE,LET) は、これらのプログラムのように並列度の変化の大きな例ではプログラムの微妙な差異で結果は大きく異なる。プログラム４の場合には図１２におけるＣとＤのノード群は独立に処理でき、Ｌレベルを最優先した場合には処理は速い。しかし、プログラム５の場合には破線のアークの出側ノードがＤのノード群の中で最も速く実行されるとは限らないので、処理は速いとは限らない。

プログラム４、５の例を最適化するには、並列実行可能なノードの実行順序を完全に把握す



プログラム３の
図１３　割り付け結果



プログラム４，５の
図１４　割り付け結果

る必要があるが、発火条件のみを実行制御メカニズムとするデータ駆動計算機では実行以前に実行順序を把握するのは困難である。実行はE-levelの順に進むと考えるのは自然であり、サブグラフへの対応のとれるＴＥＬが基本方針として良いと考えられる。以下では基本方針としてＴＥＬを用いる。

5.3. ループの処理

ループの処理方法としては次の３点に着目した。

5.3.1. L-levelの選択

ループが存在する場合、L-level1,L-level2は異なるので、どちらを使用するかで結果が異なる。

5.3.2.ループの区別

図１５のプログラムを２台のＰＥに割り付ける場合を考える。ループを含むパス（図中のLOOPのブロック）と含まないパス（図中のSEQUENTIALのブロック）は並列に処理可能であるので、これらに含まれるノードが同数であれば、ループを回るパスを１ＰＥが受け持ち、ループを回らないパスを他のＰＥが受け持つ。しかし、ループを複数回回るとループを回るパスを受け持ったＰＥの負荷が重くなる。

このような事態を回避するために、ループをに属するノードは属さないノードと区別して割り付けを行う必要がある。グラフ上のどの部分をループとして区別するかは下記の３通りが考えられる。

L-N) ループを区別しない
L-G) FlagがＧであるノード、即ちグラフ上でループを形成しているノードのみをループとして区別する
L-P) FlagがＧまたはＰであるノード、即ちプログラムでループとして記述されているノードをループとして区別する

このようにループとループ以外を区別することにより、ループを回らない場合でも効率良い割り付け方法となっている。

5.3.3.FlagがPであるノードへの割り付け

ループにおける並列性には２種類ある。

1)ループ内並列性：ループボディに存在する並列性で、データフローグラフ上に静的に記述される。
2)ループ間並列性：ループが回る度に生成される並列性で、Ｐであるノード列がパイプライン的に処理可能となる。

前者の並列性は基本方針で対応可能であるので、後者の並列性について考える。

Ｐであるノード列はパイプライン的に処理可能となるので、これらのノード列を１つのＰＥに割り付けると割り付けられたＰＥに負荷が集中する。一方、各ノードを異なるＰＥに割り付けると、通信コストが増す。特定のＰＥに負荷が集中せずに、かつ通信コストの少ない方法が必要となる。

また、図１６のグラフを考えた場合、基本方針によって割り付けを行うと図中のＡのブロックのノードとＢのブロックのノードは同じＰＥに割り付けられる。しかし、これらのノードは並列に処理可能であるので、基本方針だけでは効率の良い割り付けとはならない。

これらの要求を満足する方法として、次の方法を提案する。

概要はＰであるノードを実行するＰＥをループカウンタによりラウンドロビン式に変更することにより、ループ間並列性のあるノードが特定のＰＥに集中することを防ぐ。ただしこの場合、Ｐであるノードを処理するＰＥはループ毎に変わるのでＰであるノードとＣであるノード

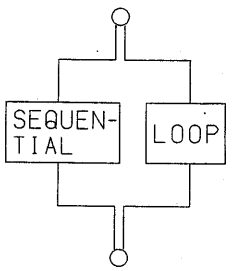


図１５　ループを含むプログラム

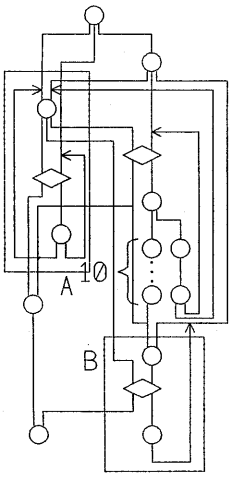


図１６　プログラム６

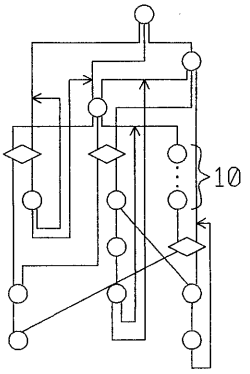


図１８　プログラム７

割り付け方法

1) 全てのPEに対応する値を０とする。
2) 基本方針を基にノードとPEを選択する
3) 選択したPEの候補が一意的に決まれば9)へ。
4) 選択したノードがＮ、Ｐならば8)へ。
5) 直上のノードにＧでないノードがあれば7)へ。
6) 直上のどれかのノードに割り当てられたPEに候補のPEがある場合はそのPEを新たに候補とて8)へ。
7) 候補のPE中、対応する値の最も小さなPEの１つを選び、そのPEの対応する値をインクリメントし、9)へ。
8) 候補のPE中の任意のPEを選択する。
9) 選択したノードに選択したPEを割り付ける。
10) 全てのノードへPEが割り付ければ終了。それ以外の場合は2)へ。

実行方法

1) Ｐであるノードに割り付けられたPE番号は予備的に付けられたものとする。
2) Ｐであるノードは次式で計算されるPEで実行する。

((PE番号)+(ループカウンタ値))
mod　(PE数)

図１７ループの処理アルゴリズム

との間の通信コストを少なくすることはできない。また、基本方針によって選択するＰＥが一意に決まらない場合には、受け持っているループ中のパス数の最も少ないＰＥを選択する。この方法を図１７に具体的に示す。

評価の際にはこの方式をとらない場合と比較する。

5.4. ループ処理方法の評価

ＰＥは４台とし、図１６、図１８、階乗の計算の３種類のプログラムによりループ長を１０として評価を行った。ここで、図１６、１８はループ間並列性のあるノードを持つ例であり、階乗はループ間並列性のない例である。これらの表１、２とに示す。

これらの結果より、L-level2,L-Pおよび Pであるノードへの割り付けを行った場合の組み合わせが良いことがわかる。

5.5. 提案方法の評価

図１９に示す熱伝導方程式[9] および図２０に示す総和により上記の割り付け方法の評価を行った。熱伝導方程式はループ内並列性の多い

表２　L-level1による実行時間

Prog	Loop	<-- N ->		<-- G ->		<-- P ->	
	P_node	off	on	off	on	off	on
6 図16		205	181	232	183	204	181
7 図18		202	171	210	185	201	167
8 階乗		165	165	163	163	163	163

表３　L-level2による実行時間

Prog	Loop	<-- N ->		<-- G ->		<-- P ->	
	P_node	off	on	off	on	off	on
6 図16		194	183	252	151	195	150
7 図18		202	171	210	185	201	167
8 階乗		125	125	124	124	124	124

例であり、総和はループ間並列性のある例である。ＰＥが１台に対する速度比を図２１、２２に示す。また比較のために、乱数により割り付けを行った結果もあわせて示す。これらの結果より、ＰＥ台数が多い場合、即ちＰＥ１台あたりの並列度が少ない場合は、乱数による割り付けに比べ高速に実行できることがわかる。

次に、資源が十分にある場合の最適な割り付けによる実行時間を考える。

乱数を用いた割り付けで速度が飽和した状態は、並列実行できるノードは全て並列実行されかつ全てのトークンはネットワークを経由して送られている。この状態では、クリティカルパス上のノードは１つのノードが到着することにより発火する場合と、２つのノードが同時に到着することにより発火する場合が起こりうる。すなわち、

(1)データ転送、待ち合わせ、実行

または

(2)データ転送、待ち合わせ、待ち合わせ、実行

の状態が繰り返される。一方、十分多くのＰＥに最適な割り付けがなされていれば、上記の状態中データ転送の状態が省かれる。データ転送、待ち合わせ、実行、に要する時間はシミュレータでは同一としている。よって、資源が無限にある場合の最適割り付けによる実行時間は

（乱数による割り付けで飽和した実行時間）

＊　３／４　or　２／３

によって推定できる。これらの値を図２１、２２ではハッチをいれて示した。

ループ内並列性のある例である熱伝導方程式ではＰＥ台数が平均並列度である１２で速度はほぼ飽和し、最大並列度である２１を越せば完全に飽和している。また飽和した値は最適割り付けによる推定値の領域内に入っている。これらから非常に良い割り付けが行われていることがわかる。

ループ間並列性のある例である総和では熱伝導方程式の場合ほど並列度との関係が顕著ではないが、最大並列度である７でかなり飽和値に近い値となっている。

これらの例から本方法は精度の良い負荷分散方式であることがわかる。

６．あとがき

ループを含むプログラムに対し、ＰＥ間の通信時間を考慮に入れた静的な負荷分散方式の提案を行った。本方法は、リストスケジューリングを基本方針としているが、データ駆動計算機では実行順序をスケジューラが指定できないた

```
for(i=1;i<=100;i++){   /* loop for time  */
    z1 = u1 +alfa*(u0-2*u1 +u2 );
       :            :
    z10= u10+alfa*(u9-2*u10+u11);
    z0 = u0;           /* boundary : x=0 */
    z11= z9;           /* boundary : x=1 */
    u1 = z1; ...; u11 = z11;
}
```

図１９　熱伝導方程式

```
for(s=0, i=1;i<=100;i++)
    s = s + ((...(a10*i+a9)*i+a8)...)*i+a0;
            ←－－－－－１０次式－－－－→
```

図２０　総和

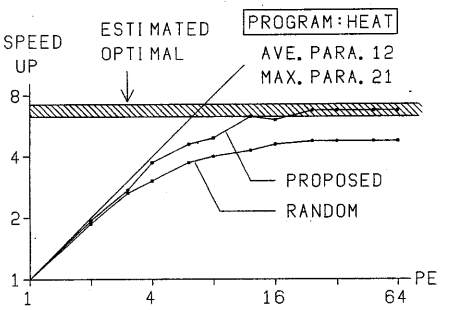


図２１　熱伝導方程式の例による評価

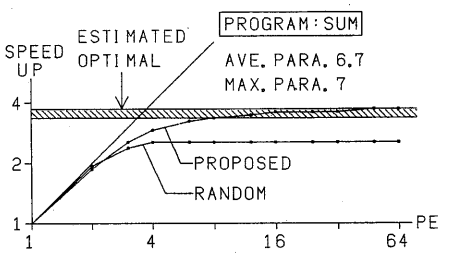


図２２　総和の例による評価

め、従来研究されているマルチプロセッサシステムにおけるものとはプライオリティの設定方法が異なる。また、ループ間に存在する並列性も考慮に入れた。

本方法の採用により各ＰＥあたりの並列度が少ない場合には乱数による負荷分散に比べ20%-30% 速度が向上することをシミュレーションにより確認した。

また、本方法を SIGMA-1のコンパイラへの実装を検討している。しかし本方法では下記の事項に対する考慮はなされておらず今後の課題として残されている。

1)条件分岐
2)多重ループ
3)stickyトークンの処理
4)サーキュラパイプラインの活用

研究の遂行にあたり、御指導、御討論いただいた電子技術総合研究所柏木寛電子計算機部長および計算機方式研究室の方々に感謝致します。

文献

[1] 坂井修一:"並列計算機におけるスケジューリングと負荷分散",情報処理,Vol.27,No.9,pp.1031-1038(1986)

[2] 平木敬,関口智嗣,島田俊夫:"並列計算機におけるネットワークを用いた動的負荷分散機構",電子通信学会論文誌,Vol.J69,No.2,pp.180-189(1986)

[3] E.G.Coffman,Jr. and P.J.Denning: "Operating Systems Theory", Prentice-Hall series in Automatic Computation (1973)

[4] H.Kasahara and S.Narita: "Practical Multi-Processor Sheduling Algorithms for Efficient Parallel Processing", IEEE Trans. Comput., Vol.C-33,No.11, pp.1023-1029 (1984)

[5] M. Tokoro, J. R. Jagannathan, and H.Sunahara:"On the Working Set Concept for Data-Flow Machines", Proc. of 10th Annu. Symp. on Comput. Arch., pp.90-97 (Jun.1983)

[6] 大塚喜久,坂井修一,弓場敏嗣:"データ駆動計算機における静的負荷分散方式の検討",情処学会第33全大,5C-6(1986)

[7] T.Shimada, K.Hiraki, and K.Nishida: "Evaluation of a Prototype Data Flow Processor of the SIGMA-1 for Scientific Computation", Proc. of 13th Annu. Symp. on Comput. Arch., pp.226-234 (Jun.1986)

[8] Arvind and K. P. Gostelow: "The U-interpreter", IEEE Comput., Vol.10, No.2, pp.42-49 (1982)

[9] 大塚喜久,関口智嗣,平木敬,島田俊夫: "科学技術用データ駆動計算機SIGMA-1 における偏微分方程式の解法の評価",情処学会第32全大,6R-4(1986)