

先行評価に適した並列計算機の ネットワーク構成

石崎一明 安江俊明 山名早人 村岡洋一

早稲田大学理工学部

先行評価は、制御依存を越えて命令を実行する方式である。実行される命令の中には、最終的には無駄になる命令の実行も含まれている。そこで、先行評価に関する処理やプロセッサ間通信が本来の実行に影響を及ぼさないようにする必要がある。並列計算機上で複数のプロセッサを用いて先行評価を行う場合、プロセッサのスケジューリングが影響を減少させるための重要な要因となる。

本稿では、1) 並列計算機上で先行評価を行う場合のプロセッサのスケジューリング方式、2) 先行評価を行うために並列計算機のネットワークに必要な機構、について述べる。

A Network Construction of Parallel Processor for Eager Evaluation

Kazuaki Ishizaki, Toshiaki Yasue, Hayato Yamana, Yoichi Muraoka

School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjiku-ku, Tokyo, 169 Japan

e-mail: ishiz@muraoka.info.waseda.ac.jp

On a parallel processor, when a program is executed using eager evaluations, the effect of executing codes with eager evaluation is not executed fully. Because eager evaluation codes include meaningless codes at the end of the execution. Therefore, processor scheduling is the key to executing full advantage of eager evaluation.

This paper describes 1) a scheduling scheme of processors for eager evaluation on parallel processors, 2) a network mechanism of parallel processor for eager evaluation.

1. はじめに

本報告では、先行評価を行う並列計算機のスケジューリングとネットワーク構成について述べる。

ここで対象とするネットワークは、スカラデータのみが流れるネットワークである。つまり、ターゲットとする並列計算機は、スカラデータ用のプロセッサ間通信路の他に配列データ用の通信路を持つ並列計算機である。スカラデータと配列データにそれぞれ専用のネットワークを用意する理由は第4節で述べる。ただし、スカラデータと配列データの分類は絶対的なものではなく、定義後すぐに参照される単一配列要素はスカラデータとしてネットワークに流す。

ここでは並列計算機として共有メモリ型の1000台規模の並列計算機（例：並列処理システム—晴—[YMHM88]）を対象とする。

n 段の条件分岐文を先行評価することで実行時間を短縮する方式は、従来 [RiFo72] [BaGa84] [Uht88] で提案されている。しかし、[Uht88] ではループ内に 1 つのみ条件分岐が存在する場合である。また、[RiFo72] では実行方式についての記述はない。[BaGa84] では、実現方式を提案しているが、専用ハードウェアを用いた単一プロセッサ上での実装手法である。このように、並列計算機上への実装手法の論議はこれまでされていない。

本報告では、n 段の条件分岐文の先行評価を行う [YIYM91] 際のスケジューリング手法を提案する。ここではスカラデータ依存関係に着目し、通信遅延（レイテンシ）を最小にして、プロセッサの稼働率を最大することを目的とした。その後、プロセッサに割り付けられるタスクを、その特徴から先行評価、高並列度ループ、低並列度部分に分類する。それぞれの場合にネットワーク上を転送されるデータの特徴を述べ、ネットワークに必要な要件を述べる。さらに、その要件を満たすネットワークを示す。

以下、第2節では先行評価方式の概要について述べる。第3節では、先行評価を行うタスクをプロセッサにスケジューリングするための方式を述べる。第4節では、データ転送の特徴を抽出し、ネットワークに必要なと考えられる要件について述べる。第5節では、第4節で述べた要件を満たすネットワーク構成と、その例を示す。第6節では、本報告のまとめを述べる。

2. 先行評価方式

2.1 n 段の条件分岐文を先行評価する方式 [YIYM91]

図1に実行の概要を示す。まず、図1に示すように、ループ内で同一の制御を受ける基本パス同士を融合し、タスクを作成する。このタスクの各段毎にゲートを挿入して、n 段先行評価するための制御を行う。

実行時には、実際に制御が確定した命令の実行によ

って、挿入したゲートのコントロールを行いながら、常に n 段の条件分岐文の先行評価を行う。

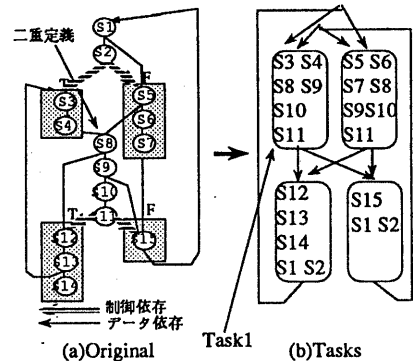


図1 n 段先行評価の概要

3. スケジューリング

3.1 定義

まず、本節で述べるスケジューリングの目的を以下のように定義する。

条件(1) 先行評価を行うタスク集合をプロセッサに割り付けたとき、実行時に同じプロセッサで異なるタスクが同時に実行されないこと

条件(2) 使用プロセッサ数を有限個にするためにプロセッサが再利用できること

という条件の元でプロセッサの稼働率（＝プログラムを実行中のプロセッサ数／先行評価に必要な全てのプロセッサ数）を最大にすることが、スケジューリングの目的である。また、プロセッサへのスケジューリングの単位は、1つのタスクを1つのプロセッサに割り当てるものとする。

ここで扱うタスク集合は、タスクをノード、タスク間のデータ依存関係をアークで示したときに、グラフが完全 n 分木で表されるものとする。

以下では、先行評価を行う際のプロセッサのスケジューリング手法として、静的スケジューリング、半動的スケジューリング、動的スケジューリングの3つを示す。その後、これらのスケジューリング手法を用いる方針について述べる。

3.2 静的スケジューリング

静的スケジューリングとは、プログラムのコンパイル時に実行するタスクとプロセッサの割付けが静的に決定できるとともに、データ通信を行うプロセッサ対も静的に決定できるスケジューリングをいう。

図2に、2分岐で3段の先行評価を行う場合を例に取りプロセッサの静的スケジューリングを示す。ノードがプロセッサで、ノードの中の番号がプロセッサ番号を示す。アークがデータ依存関係のあるプロセッサ

対を示す。また、網掛けが稼働中のプロセッサ、白が稼働していないプロセッサを表す。

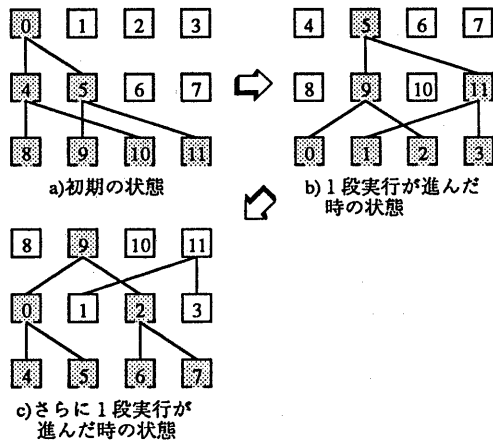


図2 静的スケジューリング

図2(a)に示すように、2分岐で3段の先行評価を行うタスクグラフがスケジューリングされている。このとき、図の1番上の段のプロセッサ番号0以外のタスクは条件分岐文を越えて先行評価を行っている。

以下、次の手順で実行を進める。

- 1) プロセッサ0で条件分岐文が実行されると、その結果に従って先行評価を行っているプロセッサ集合がその実行を停止する。

この場合、プロセッサ集合 {4, 8, 9} もしくは {5, 9, 11} がその実行を停止する。

- 2) ここでは、プロセッサ集合 {4, 8, 9} が実行を停止したとして、次段の実行を続ける。

次に、プロセッサ集合 {5, 9, 11} の根であるプロセッサ5が先行評価を行うプロセッサ集合の制御を行う。

また、プロセッサ集合 {5, 9, 11} の葉であるプロセッサ9, 11では先行評価段数を常に3段に保つために、プロセッサ0, 1, 2, 3にデータを転送し先行評価を行う(図2(b))。

- 3) プロセッサ5で条件分岐文が実行されると、その結果に従って先行評価を行っているプロセッサ集合が実行を停止する。

この場合、プロセッサ集合 {11, 1, 3} がその実行を停止したとする。

- 4) 次に、プロセッサ集合 {9, 0, 2} の根であるプロセッサ9が先行評価を行うプロセッサ集合の制御を行う。

また、プロセッサ集合 {9, 0, 2} の葉であるプロセッサ0, 2では先行評価段数を常に3段に保つために、プロセッサ4, 5, 6, 7にデータを転送し先行評価を行う(図2(c))。

この静的スケジューリングでは、先行評価が1段進んだ時に、同じプロセッサが同時に別のタスクを実行することはない。また、図2(a)と図2(c)を比較するとわかるように、プロセッサ {0, 4, 5} の結合形態は同一であるからプロセッサの再利用を行うことができる。

以下では、図2のスケジューリングが3.1項で述べた条件を満たすことを式によって示す。

まず、3.1項の最初で定義したスケジューリングの条件は、以下のような条件に置き換えることができる。

- 1) プロセッサが同時に別タスクを実行することはない
→ i 段目を実行しているプロセッサの出力と $i+1$ 段目を実行しているプロセッサの入力は1対1対応である
- 2) プロセッサが再利用できること
→ あるプロセッサの出力がプロセッサ間結合をたどった結果、同じプロセッサの入力に戻ることができるプロセッサ間結合であること

それぞれのプロセッサからの出力が2分岐の場合、各プロセッサが2入力2出力のポートを持つと考える。すると、図3のようにプロセッサのポートにそれぞれ2進表示で(000)から(111)までの番号を付与することができる。

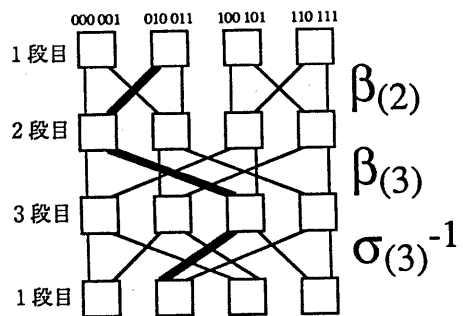


図3 プロセッサ間結合

以下、ある1段目のポート(010)を例に取り、このポートがプロセッサ間結合をたどった結果、もとのポートに戻ることによってプロセッサの再利用ができることを示す。

1段目のプロセッサと2段目のプロセッサの結合は2-サブバタフライであるから、1段目のポート(010)は2段目のポート(001)に結合される。2段目のプロセッサと3段目のプロセッサの結合は3-サブバタフライであるから、2段目のポート(001)は3段目のポート(100)に結合される。3段目のプロセッサの出力は3-逆サブシャフル網に接続されているから、3段目のポート(100)はポート(010)に結合される。ここで、ポート(010)というのは最初の1段目のポ

ト番号と同じである。従って、このポート(010)を1段目のプロセッサのポートに接続することによってプロセッサを再利用することができる。

この手順を一般化して表すと以下ようになる。まず、各ポートを $(a_n, \dots, a_2, a_1)$ とする。

- 1) あるポートの初期位置を $(a_n, \dots, a_2, a_1)$ とする。
- 2) 最初の置換操作 $\beta_{(2)}$ ($=2$ -サブバタフライ) によってポート $(a_n, \dots, a_2, a_1)$ は、次段のポート $(a_n, \dots, a_3, a_1, a_2)$ に結合される。
- 3) 出力ポート $(a_n, \dots, a_3, a_1, a_2)$ は、置換操作 $\beta_{(3)}$ によって次段のポート $(a_n, \dots, a_4, a_2, a_1, a_3)$ に結合される。
- 4) ポート $(a_n, \dots, a_4, a_2, a_1, a_3)$ は、置換操作 $\sigma_{(3)}^{-1}$ ($=3$ -逆サブシャフル) によって次段のポート $(a_n, \dots, a_4, a_3, a_2, a_1)$ に結合される。
- 5) 上で示されたポート $(a_n, \dots, a_4, a_3, a_2, a_1)$ は初期位置と同じである。

よって、このポート $(a_n, \dots, a_4, a_3, a_2, a_1)$ を初期位置のポートに接続することでプロセッサを再利用することができる。また、置換操作 $\beta_{(n)}$ 、 $\sigma_{(3)}^{-1}$ は1対1対応であることは明らかであるから [Tomii86]、プロセッサが同時に別のタスクを実行することはない。

一般に、 n 段先行評価をする場合、置換操作 $\beta_{(2)}$ 、 $\dots$ 、 $\beta_{(n)}$ 、 $\sigma_{(3)}^{-1}$ を行うことで、同時にプロセッサが2重使用されることなくプロセッサの再利用を行うことができる。

ここで、先行段数を n 、分岐数を b とおく。すると、静的スケジューリングを行う場合、 n 段目で必ず $b^n - 1$ 個のプロセッサが必要となるので、必要なプロセッサ数は $n \times b^n - 1$ で求めることができる。このとき、タスクを実行中のプロセッサ数は常に $b^n - 1$ 個である。よって、プロセッサの稼働率は

$$\frac{b^n - 1}{n \times b^n - 1}$$

で求めることができる。

3.1 半動的スケジューリング

半動的スケジューリングとは、プログラムの実行時に実行するタスクとプロセッサの割付けが計算によって決定できるとともに、データ通信を行うプロセッサ対も計算によって決定できるスケジューリングをいう。

図4に、2分岐で3段の先行評価を行う場合を例に取り、プロセッサの半動的スケジューリングを示す。ノードがプロセッサで、アークがデータ依存関係のあるプロセッサ対を示す。ノードの中の番号 (R, C)

(R :行番号、 C :列番号)は、現在のプロセッサのタスクグラフ上の論理的な位置を示す。また、 $[r, c]$ は1段前の実行状態の時に位置したプロセッサのタスクグラフ上の論理的な位置を示す。

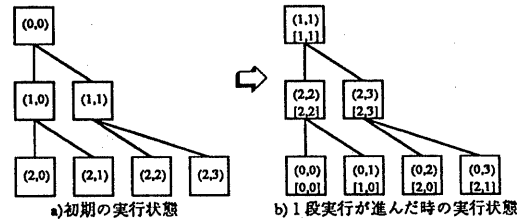


図4 半動的スケジューリング

図3(a)に示すように、2分岐で3段の先行評価を行うタスクグラフがスケジューリングされている。このとき、図の1番上の段の論理位置(0,0)以外のタスクは条件分岐文を越えて先行評価を行っている。

以下、次の手順で実行を進める。

- 1) タスク(0,0)で条件分岐命令が実行されると、その結果に従って先行評価を行っているタスク集合がその実行を停止する。

この場合、タスク集合 $\{(1,0), (2,0), (2,0)\}$ もしくは $\{(1,1), (2,2), (2,3)\}$ がその実行を停止する。

- 2) ここでは、タスク集合 $\{(1,0), (2,0), (2,1)\}$ が実行を停止したとして、次段の実行を続ける。

次に、タスク集合 $\{(1,1), (2,2), (2,3)\}$ の根であるタスク5が先行評価を行うタスク集合の制御を行う。

また、タスク集合 $\{(1,1), (2,2), (2,3)\}$ の葉であるタスク(2,2)、(2,3)では先行評価段数を常に3段に保つ必要がある。そこで、実行を停止したタスク(0,0)、(1,0)、(2,0)、(2,1)に新しいタスク番号(0,0)、(0,1)、(0,2)、(0,3)を割当て、データを転送し先行評価を行う(図2(b))。

先行評価が1段進んだ時の、プロセッサの論理的な位置の再割当ては以下の式によっておこなわれる。

$$(R, C) = d, \begin{cases} 0 & r=0 \\ t + |d_t| & r \neq 0 \end{cases} \quad t = b^{r-d+n-1}$$

R : 行番号

C : 列番号

b : 1つのプロセッサからの分岐数

n : 先行評価段数

d : 実行が1段進む前のタスクグラフの根の R

r : 1段前の実行状態の時の R

c : 1段前の実行状態の時の C

プロセッサ間のデータ通信を行うためには、データの送り先のプロセッサの物理的な番号を指定して送る必要がある。この式は物理的なプロセッサ番号と論理的なプロセッサ位置の対応を与えるものではない。従って、現在の論理的なプロセッサ位置と物理的なプロセッサ番号との対応を、なんらかの方法で各プロセッサが知らなければならない。そこで、以下に述べる方法でこの対応を知る。

あらかじめ各プロセッサは、物理プロセッサ番号と初期の論理的なプロセッサ位置を対応づけた図5(a)のような表を持つ。実際に先行評価が1段進んだ時に、プロセッサの論理的な位置の再割当の式に基づいて、物理プロセッサ番号と論理プロセッサ位置の対応を変更し表の書き換えを行う(図5(b))。

R\C	0	1	2	3
0	0			
1	1	2		
2	3	4	5	6

a) 初期の対応状態

R\C	0	1	2	3
0	0	1	3	6
1		2		
2		4	5	

b) 1段実行が進んだ時の対応状態

太字はプロセッサ番号、空白は対応するプロセッサがないことを示す

図5 物理プロセッサ番号の対応

半動的スケジューリングを行う場合、先行評価に必要なプロセッサ数は実行中のプロセッサ数と同じであり b^{n-1} 個である。よって、プロセッサの稼働率は1である。

3.4 動的スケジューリング

動的スケジューリングとは、プログラムの実行時に実行するタスクとプロセッサが動的に決定されるとともに、データ通信を行うプロセッサ対も動的に決定されるスケジューリングをいう。

図6に、2分岐で3段の先行評価を行う場合を例に取り、プロセッサの動的スケジューリングを示す。ノードがプロセッサで、ノードの中の番号がプロセッサ番号を示す。アークがデータ依存関係のあるプロセッサ対を示す。また、網掛けが稼働中のプロセッサ、白が稼働していない空きプロセッサを表す。

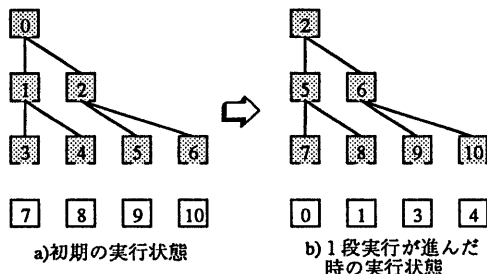


図6 動的スケジューリング

図6(a)に示すように、2分岐で3段の先行評価を行うタスクグラフがスケジューリングされている。このとき、図の1番上の段のプロセッサ番号0以外のタスクは条件分岐文を越えて先行評価を行っている。

以下、次の手順で実行を進める。

- 1) プロセッサ0で条件分岐文が実行されると、その結果に従って先行評価を行っているプロセッサ集合がその実行を停止する。

この場合、プロセッサ集合 {1, 3, 4} もしくは {2, 5, 6} がその実行を停止する。

- 2) ここでは、プロセッサ集合 {1, 3, 4} が実行を停止したとして、次段の実行を続ける。

次に、プロセッサ集合 {2, 5, 6} の根であるプロセッサ2が先行評価を行うプロセッサ集合の制御を行う。

また、プロセッサ集合 {2, 5, 6} の葉であるプロセッサ5, 6では先行評価段数を常に3段に保つために、空きプロセッサ7, 8, 9, 10にタスクを割り付けて先行評価を行う(図2(b))。

プロセッサ集合 {0, 1, 3, 4} は実行を完全に停止すると、空きプロセッサとなってさらに先行評価が進んだときにタスクが割り付けられる。

図6のように、常に 2^{n-1} 個の空きプロセッサを用意しておけば、次に割り付けるタスクをあらかじめ決めることができるので、タスクを動的に割り付けるオーバーヘッドを少なくすることができる。

ここで、先行段数を n 、分岐数を b とおく。すると、動的スケジューリングを行う場合、タスクを実行中のプロセッサ数は常に b^{n-1} 個である。また、必要なプロセッサ数は、実行中のプロセッサ数に加えて、先行評価が1段進んだ時に実行される空き状態の b^{n-1} 個となる。よって、 $b^n + b^{n-1} - 1$ 個となる。従って、プロセッサの稼働率は

$$\frac{b^n - 1}{b^n + b^{n-1} - 1}$$

で求めることができる。

3.5 各スケジューリングの使用方針

プロセッサの使用効率から考えると、半動的スケジューリングが最も優れている。しかし、この方式ではプログラムの実行時にデータを転送するプロセッサを、プログラムから明示的に変更できる命令アーキテクチャを持たなければならない。このような命令を持たない場合でも、関数を動的に生成できる機構を持つならば動的スケジューリングを使用することができる。ど

ちらもない場合には、プロセッサの使用効率が最も悪い静的スケジューリングを使用するしかない。

次に、プロセッサの通信時間から考える。2つのプロセッササブグループを用いて先行評価を行っていた場合を考える。この場合、サブグループ間のプロセッサ通信と、サブグループ内のプロセッサ通信にかかる時間には当然差がでる。

動的スケジューリングでは実際にタスクがプロセッサに割り付けられるまで、通信が行われるプロセッサ対が決定できない。すると、上で述べたようにプロセッサ間通信にかかる時間に偏りがある場合には、通信遅延が大きいプロセッサ間通信を用いて実行することがあり、プロセッサの通信時間が大きくなる。

半動的スケジューリングも、ほぼ同様である。プログラムの実行前にプロセッサ通信が起こるプロセッサ対を求めることはできるが、実際にはほとんど全てのプロセッサ対で通信が起きることになり、通信遅延が大きいプロセッサ間通信を用いて実行することがあり、プロセッサの通信時間が大きくなる。

静的スケジューリングでは、プログラムの実行前にプロセッサ通信が起こるプロセッサ対を特定することができる。よって、通信遅延が大きいプロセッサ間通信がある場合には、そこを避けるようなプロセッサ間結合を組み合わせることで、プロセッサの通信時間を短縮することができる。

よって、複数のプロセッササブグループを用いて先行評価を行う場合には、サブグループ間の通信遅延が大きい部分に対しては静的スケジューリングを用いて、サブグループ内の通信遅延が小さい部分には半動的または動的スケジューリングを用いる、というように複数のスケジューリング手法を用いるのがよいと考えられる。

4. ネットワークに必要な要件

本節では、最初に対象とする並列計算機にスカラデータの通信路と配列データの通信路をそれぞれ用意することを仮定した理由を述べる。

次に、プロセッサに割り当てられるタスクの種類によってプロセッサ間で通信を行うデータの特徴を分類し、必要とされる要件を明らかにする。

4.1 スカラデータネットワーク

対象とする並列計算機に、スカラデータの通信路と配列データの通信路をそれぞれ用意することを仮定した理由は以下のとおりである。

スカラデータでは、データ通信は散発的である。このデータ通信は、1つのタスクが複数のプロセッサに割り当てられた場合にプロセッサ間で起きる通信や、タスク間のデータ通信が該当する。一般に、このデー

タ通信が実行時間に影響する。

配列データは、プロセッサと共有メモリ間である一時期に大量のデータ通信が発生する。その結果、ネットワークの特定のリンクを占有することになる。ネットワークが閉塞網であった場合は、ホットスポットが発生する可能性が大きくなる。ホットスポットとは、ある一箇所に通信が集中するか継続することで他の通信をブロッキングし、その結果他の部分にも波及してネットワークの処理能力が大幅に減少してしまう現象である。

このように、配列データとスカラデータでは全く通信の性格が違う。さらに、配列データとスカラデータを同じネットワーク上で通信を行うと、配列データの転送によって引き起こされたホットスポットによって、スカラデータの転送に影響を及ぼす可能性がある。その結果、プログラムの実行時間が増加することになる。

従って、プログラムの実行時間に大きく影響するスカラデータの転送を優先するためには、スカラデータと配列データを別のネットワークで転送することは妥当であると考えられる。

4.2 転送データの分類

ネットワークの構成を示す前に、プロセッサ間で通信を行うデータの特徴を把握し、必要とされる要件を明らかにする。

プロセッサ間で通信を行うデータの特徴は、プロセッサに割り当てられるタスクの種類によって大きく異なる。以下では、タスクを先行評価、DOALL型及びDOACROSS型のループを実行する高並列度ループ、低並列度部分（前記の2つの場合以外のタスク）の3つに分類し、それぞれの場合についてデータ転送の特徴と要件を述べる。

4.3 先行評価のための要件

先行評価は、制御依存を越えて命令を実行する方式である。実行される命令の中には、最終的には無駄になる命令の実行も含まれている。そこで、先行評価に関する処理やプロセッサ間通信が本来の実行に影響を及ぼさないようにする必要がある。

また、同じ段数を実行している各タスクは、先行評価が1段進むとほぼ同一のタイミングで後続のタスクの転送を開始する。

さらに、静的なスケジューリングを行った場合はプロセッサ間通信の経路は静的に決まるが、その他のスケジューリングではプロセッサ間通信の経路が実行時にならないと決定しない。

従って、ネットワークは任意のプロセッサ間の通信が別のプロセッサ間の通信に影響を与えず独立に行うことができる非閉塞網であることが望まれる。特に、先行評価が複数のサブネットワークにまたがって行わ

れる場合、他のサブネットワーク間の通信になるべく影響を与えないことが必要である。

4.4 高並列度ループのための要件

マルチプロセッサにおける高並列度ループ処理方法の1つに、ループの各イテレーションを別々のプロセッサに静的に割り当てて行う方法がある。この場合、各プロセッサではループの繰り返し回数を決定するためにループの終値パラメータを必要とする。

1対1のプロセッサ間の通信を行うネットワークにおいて、ループを処理する多数のプロセッサに同一のループの終値パラメータを送る場合を考える。データをプロセッサ台数の回数だけ転送することになるので、最後の方に通信を行うプロセッサではデータの到着が遅れる。従って、ループ処理を開始するまでのオーバーヘッドが増加する。さらに、ネットワーク上にプロセッサ台数分のバケットが流れることにより通信遅延を増加させる。

従って、ネットワークには多数のプロセッサに対して同一のデータを高速にかつ通信遅延をなるべく少なく転送することができる機構が必要である。

4.5 低並列度部分のための要件

プログラムのうち、高並列ループと先行評価を行うことができなかったループ以外の部分がここに含まれる。この部分ではタスク間の並列性があまり得られないためシリアルな処理が多く、同期やデータ転送のオーバーヘッドが実行時間に占める割合が大きくなる。

また、実行がデータ依存に従って進むため通信が局所的になる傾向がある。

従って、近接のプロセッサ間通信が高速に行われることが必要である。

5. ネットワークの実現方式

4節で述べた、必要とされる要件を満たすネットワークに必要な機能と、その実現例を示す。

5.1 ネットワークの機能

4.3で述べたように、ネットワークの構成は非閉塞網であることが望ましい。最も強力な非閉塞網として、クロスバ網がある。しかし、1000台規模のプロセッサをクロスバ網で接続することは、ハードウェア量が $O(N^2)$ となるため物理的に不可能である。

別の非閉塞網として3ステージClos網がある。ハードウェア量が $O(N^{3/2})$ でありクロスバ網に比べて大幅にハードウェア量を削減することができる。しかし、どのターミナル間でも通信距離が3で一樣ある。これでは、4.5で述べた近接のプロセッサ間通信が高速であるという要件を満たすことができない。

そこで、最も強力な非閉塞網であるクロスバ網をローカルネットワークに用いる。ローカルネットワークにクロスバ網を用いることで、4.5で述べた近接のプロセッサ間通信が高速であるという要件を満たすことができる。また、先行評価時もローカルネットワーク内ではクロスバ網を用いることができるので、先行評価を実行するタスクは非閉塞網を用いた通信も行うことができる。

さらに、複数のローカルネットをグローバルネットを用いて接続する2階層のネットワークから構成する。

先行評価は複数のローカルネットワークを使用して行われることがある。よって、ローカルネットワーク間の距離がなるべく短くなるグローバルネットワークの構成を取らなければならない。4.3で述べたように、このグローバルネット上の先行評価のためのデータは他のローカルネットワーク間のデータ通信を妨げないようにしなければならない。

また、多数のプロセッサに対して同一のデータを高速に転送することができる機構としてマルチキャストを機構を実装する必要がある。クロスバ網では、マルチキャストは簡単に行うことができるので、グローバルネットワークの検討の際に念頭に置く必要がある。

5.2 ネットワーク構成

従来、base-m n-cube [TSNO88] ネットワークが提案されている。このネットワークは、隣接PEが $(m-1) \times n$ 個と多い、不規則・非局所的な通信に強い、ハードウェア量が比較的小さい、という特徴を持つ。

以下では、この特徴を生かしながら、1000台規模のプロセッサを接続するために、よりハードウェアを簡略化する方針でネットワーク構成を検討する。

具体的には、ハードウェアを削減する方法として、プロセッサをいくつかまとめたものを1つのグループと見なし、このグループに対してグローバルなネットワークの接続の配線をおこなう。このようにプロセッサをグループ化することで、実装を困難にする原因であるグローバルな配線、つまり隣接しないグループ間の配線を減らすことができる。

以上の方針を、base-m n-cubeに適応したネットワークの接続例を図7に示す。

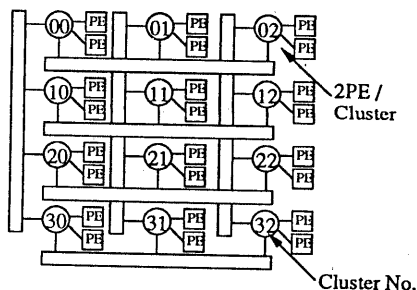


図7 ネットワークの構成

ここでは、5.1で述べたような理由でクロスバ網でプロセッサグループを形成している。クロスバ網は通信距離は1であるから、同じグループに属するプロセッサ同士は、距離1でお互いに影響されことなく通信を行うことができる。

このネットワークでは、 $(m-1) \times n$ 個の隣接プロセッサグループとの距離もbase-m n-cubeと同様に3である。よって、従来のbase-m n-cubeの特徴をそのまま受け継いでいる。この隣接プロセッサグループはお互いにクロスバ網で接続されているので、番号が異なるグループ同志の通信であれば、干渉することはない。

また、base-m n-cubeもクロスバ網から構成されているのでブロードキャスト機構を実装することは可能である。さらに、そのときの距離も高々 $2n$ である。

欠点としては、階層構造のネットワークであるため、グローバルネットワークとローカルなグループのネットワークを結ぶ接続路が、データ通信のボトルネックとなる可能性があることである。

5.3 実装面の評価

このネットワークを用いて、1024台のプロセッサを接続する場合を実装面から考察する。

ここでは、ローカルネットワークとして16台のプロセッサをクロスバ網で接続する。グローバルネットワークは64個のローカルネットワークをbase-8 2-cubeで接続する。ローカルネットワークとグローバルネットワークの通信路は1ポートとする。また、グローバルネット、ローカルネット共にバンド幅は16ビットとする。

現在、容易に手に入れられる大規模クロスバスイッチとしてT I社のAS8841がある。このLSIは、4ビット幅で 8×8 のクロスバとして使用することができる。このLSIを用いてバンド幅16ビットで、 8×8 のクロスバ網を構成するためには4個、 18×18 のクロスバ網を構成するためには36個のLSIが必要になる。

グローバルネットワークは、 8×8 のクロスバ網を16個使用する。1組のローカルネットワークは 18×18 のクロスバ網を1つ使用する。よって、ネットワーク全体で $4 \times 64 + 64 \times 36 = 2560$ 個のLSIが必要になる。

これまで実現されている4ビット幅 10×10 クロスバスイッチや、8ビット幅 8×8 クロスバスイッチを使用できると仮定する。この場合、それぞれ $4 \times 64 + 64 \times 16 = 1280$ 個、 $2 \times 64 + 64 \times 18 = 1280$ 個のLSIが必要になり、半分のLSI数で実現できる。

もし、1000台のプロセッサをbase-10 3-cubeで接続すると、4ビット幅 10×10 クロスバスイッチが1200個、ルータ用クロスバスイッチが1000個必要に

なる。

よって、base-10 3-cubeと比較しても必要なIC数を約半分まで減らすことができる。

6. おわりに

本報告では、 n 段の条件分岐文の先行評価を行う際のスカラデータ依存関係に着目し、プロセッサの稼働率を最大にするスケジューリング手法を提案した。さらに、実行を終えたプログラムを再利用するルーティングを行うことで、先行評価に使用するプロセッサ数を減少させた。

今後は、ネットワークの構成をさらに具体化して、シミュレーションによる動的なパラメータから最適なネットワーク構成を決定したい。

謝辞

本稿作成にあたり、御討論いただきました村岡研究室諸氏に感謝いたします。

参考文献

- [YMHM88] H. Yamana, T. Marushima, T. Hagiwara, Y. Muraoka: "System Architecture of the 'Parallel Processing System -Haray-', Proc. of ICS'88, pp. 76-89, 1988
- [BaGa84] Utpal Banerjee, D. D. Gajski: "Fast Execution of Loop with IF statements", IEEE Trans. on Computers, Vol. C-33, pp. 1030-1033, 1984
- [RiFo72] E. M. Riseman, C. C. Foster: "The Inhibition of Potential Parallelism by Conditional Jumps", IEEE Trans. on Computers, pp. 1405-1411, 1972
- [Uht88] A. K. Uht: "Requirements for Optimal Execution of Loops with Tests", Proc. of ICS88, pp. 230-237, 1988
- [Cytr86] Ron Cytron: "Doacross: Beyond Vectorization for Multiprocessors", Proc. of ICPP'86, pp. 836-844, 1986
- [YIYM91] 山名、石崎、安江、村岡: "並列処理システム一暗一における条件分岐の先行評価制御方式", 本研究会投稿予定
- [Tomi86] 富田: "並列計算機構成論", 昭晃堂, 1986
- [TSNO88] 田辺、鈴岡、中村、藤田、小柳: "並列A IマシンProdigyのプロセッサ間結合網", 信学技報, CPSY88-30, 1988