計算機アーキテクチャ 95−16
(1992. 8. 20)

スーパーデータベースコンピュータ (SDC) のバケット平坦化ネットワークにおける縮退動作支援アルゴリズムとその評価

田村孝之 中村 稔 喜連川 優 高木幹雄
東京大学 生産技術研究所

概要

並列計算機システムにおいては, 処理要素間での通信オーバーヘッドや負荷の偏りを解決し, 大きな性能向上を達成することと共に, 1つ1つの処理要素の故障に対するロバスト性が要求される. 現在東京大学で開発中のスーパーデータベースコンピュータ SDC では, 前者の問題を解決するために“バケット平坦化機能”をネットワークのハードウェアに付加している. しかし, これまでのアルゴリズムによると2の冪乗以外の処理モジュール数は許されないため, 障害からの回復に難点があった. そこで本論文では, 縮退動作を支援する任意数の処理モジュール間での新しいバケット平坦化手法を提案し, そのアルゴリズムとシミュレーションによる評価結果について述べる.

An algorithm for reduced configuration support on bucket flattening network of the Super Database Computer and its evaluation

Takayuki TAMURA Minoru NAKAMURA
Masaru KITSUREGAWA Mikio TAKAGI
Institute of Industrial Science, University of Tokyo
7-22-1, Roppongi, Minato-ku, Tokyo 106, JAPAN.

Abstract

In parallel computing system, It is needed to achieve robustness for the failure of each processing elements, as well as to improve the performance significantly by overcoming communication overheads and load unbalance among them. In *the Super Database Computer (SDC)*, which is under developing at University of Tokyo, we have introduced *the bucket flattening omega network* to satisfy the latter requirements. In this paper, we propose a new bucket flattening algorithm which allows system to consist of arbitrary number of processing elements. The mechanism and the performance evaluations are discussed.

1 はじめに

並列処理において，処理要素数の増加に見合った性能向上を得るには，I/O ボトルネックや通信オーバーヘッド，各処理要素間の負荷の偏りなどを解決する必要がある．また，ネットワークを介した疎結合型のアーキテクチャは密結合方式に比べてスケーラビリティの点で優るが，システムが大規模化していった時に処理要素間でのランダムな通信が増加し，ネットワークのスループットが低下してしまうという傾向がある．そこで，現在東京大学で開発中の高並列関係データベースサーバ“スーパーデータベースコンピュータ (SDC)”では，処理モジュール間の相互結合ネットワークとして，処理要素間のランダムな通信の発生を抑えつつ負荷を均等に分散させるために，ネットワークの各スイッチ素子自体が局所的な履歴に基づいて適応的なルーティングを行なう“パケット平坦化”オメガネットワークを採用している．

一方，システムの並列度が増すにつれ，信頼性を向上させ，実用的なシステムを作るには各処理要素の故障を考慮に入れること欠かせなくなってくる．ことに各処理要素が大容量の2次記憶系を含む並列データベースシステムにおいては，一部の処理要素の故障は単なる性能の低下に留まらず記憶されていた大量のデータを失うことにもつながるので，故障に対するロバスト性は重要である．そこで問題となるのは，故障した処理要素をいかに検出するか，故障を検出した後でどのようにして処理を続行するかということである．ネットワークの大きさは通常 2 の冪乗に限られており，正常な処理要素だけで縮退運転を再開するにはネットワークの大きさと異なる任意数の処理要素でシステムを構成できなければならない．また，たとえ障害の発生を別にしても，取扱う処理の規模や蓄積されたデータ量が増すのにつれて徐々にシステムを拡張していこうとする場合などには，2 の冪乗以外の処理要素数が許されることが望ましい．

ところが，SDC のパケット平坦化ネットワークでは各スイッチ素子は局所的な情報しか持っていないため，これまでのアルゴリズムでは故障した処理要素を除外して残りの処理要素間で負荷を分散することはできない．そこで，故障した処理要素に関する大域的な情報を局所的な情報に変換して各スイッチ素子に与えるためのアルゴリズムが必要になってくる．

本論文では，SDC のパケット平坦化ネットワークにおいて，任意数の処理モジュール間での負荷分散を可能にする新しいアルゴリズムについて述べる．このアルゴリズムは，正常な処理モジュールを数え上げ，それぞれのスイッチから到達可能な処理モジュール数を設定する部分と，その値を重みとした出力タプル数の分散に基づいて経路の決定を行なう部分とからなる．

以下，第2章で SDC についてそのアーキテクチャと主要な関係演算アルゴリズムを述べ，パケット平坦化機能について説明する．次に第3章で，これまでのパケット平坦化アルゴリズムにおける縮退動作時の問題点を指摘し，これを解決する新たなアルゴリズムを提案する．第4章ではシミュレーションによりこのアルゴリズムの有効性を示し，さらに第5章で得られた結果をまとめると共に，この機能を実現するための効率的なハードウェア実装法について考察する．

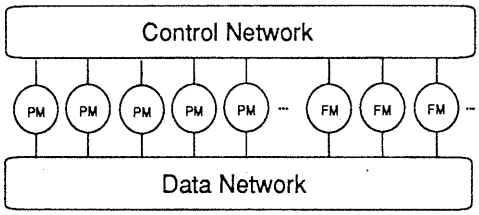


図 1: SDC のアーキテクチャ

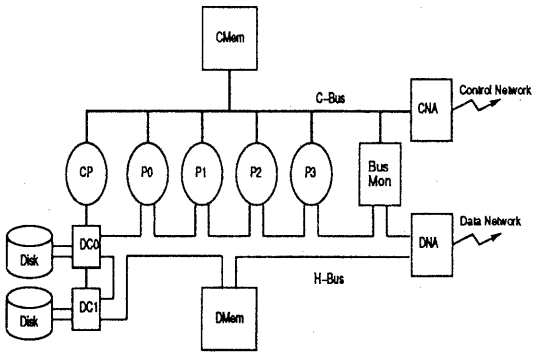


図 2: 各処理モジュールの構成

2 スーパーデータベースコンピュータ SDC の概要

2.1 SDC のアーキテクチャ

スーパーデータベースコンピュータ SDC は，現在我々が開発中の高並列関係データベースサーバである．SDC は，数台 (4～6台) のプロセッサと磁気ディスク装置とを共有バスで密に結合した処理モジュールをネットワークで疎に結合したハイブリッドアーキテクチャをとる [1]．また，データベース処理における大量のデータの流れとコントロール用の通信とが衝突してデータ転送性能を低下させるのを防ぐために，処理モジュール内の共有バスおよび処理モジュール間のネットワークには各々に対してデータ系とコントロール系との2系統を用意している．

SDC 全体のアーキテクチャを図1に，また各処理モジュール内の構成を図2に示す．ただし，図中の PM は処理モジュール，FM はフロントエンドモジュールを表す．フロントエンドモジュールはユーザインタフェースを司る部分であり，ホストから SQL のコマンドを受けとってその実行を各処理モ

ジュールに割り付ける機能を受け持つ.

SDC のデータネットワークの特徴としては, 単なるデータ転送機能に止まらず次節で述べる並列関係演算アルゴリズムを支援する負荷分散機能を有していることが挙げられる. この機能により, 処理モジュール数を増加させた時にも特定の処理モジュールに負荷が偏ることなく線形な性能向上を期待することが出来る.

2.2 SDC における並列関係演算アルゴリズム

関係データベース処理の中でも特に負荷の重い演算として結合演算が挙げられる. 結合演算のアルゴリズムには様々なものが提案されているが, SDC においてはハッシュ分割法の一種である "バケット分散 GRACE ハッシュ" アルゴリズムを採用している [2][4].

バケット分散結合法は以下のようなフェーズにしたがって実行される.

- **分割フェーズ**
  リレーション R と S について, ディスクから読み出したタプルのキー属性にハッシュ関数 (スプリット関数) を適用してバケット ID を決定し, ネットワークに送出する. 従来のハッシュ結合アルゴリズムではバケット ID から転送先のモジュールを静的に決定する "バケット集中方式" を採っていたが, 本処理方式においてはバケットを一旦全ての処理モジュール間に渡って水平に分散する "バケット分散方式" を採用する.
  この際, 各タプルの転送先はあらかじめ決まっていないため, ネットワーク上で動的に行き先を決定することができ, 競合によるスループットの低下を避けることができる. また同時に, 各処理モジュール毎に格納されるバケットが全て均等な大きさになるように制御することにより, 後の演算フェーズを効率的に実行することができる. このような分布をバケット平坦分布と呼ぶ。
- **bucket size tuning**
  処理モジュール内で複数のバケット (サブバケット) をほぼ一定の大きさにまとめあげる.
- **演算フェーズ**
  バケットを各処理モジュールに割り当て, 各処理モジュール毎に独立に結合演算を行なう. 処理すべきバケットはサブバケットとして処理モジュール間に分散しているが, これを巡回的に読み出すことにより (図 3), アクセス競合を起すことなく収拾することができる.

このように, バケット分散ハッシュ法では分割フェーズにおける,

1. ネットワーク上で各タプルの転送先を動的に決定する.
2. バケット平坦分布を実現する.

という機能が重要である. そこで, 我々はこの機能を "バケット平坦化機能" と呼び, ネットワークのハードウェアで実現することを提案している.

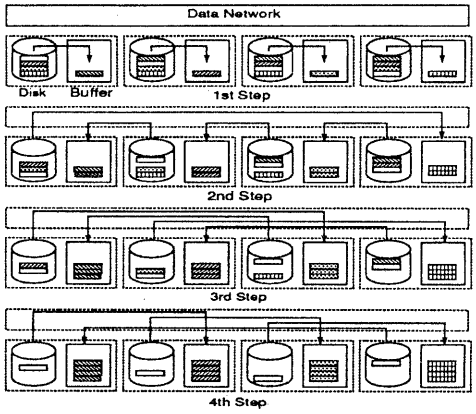


図 3: 巡回的バケット収集

2.3 従来のバケット平坦化アルゴリズム

最も基本的なバケット平坦化は, 各バケットの分布の平坦さの尺度として, 処理モジュール間でのバケットの大きさ (属するタプルの数) の分散を考え, そのバケット毎の総和が最小になるようにネットワークに入力されたタプルの行き先を決定することとして表現することができる. すなわち, N を全処理モジュール数 (ネットワークの大きさ), $C_i(x,t)$ を時刻 t までに処理モジュール i に送られたバケット x に属するタプル数の合計とすると, 時刻 t におけるバケット x の分布の分散 $V_N(x,t)$ は,

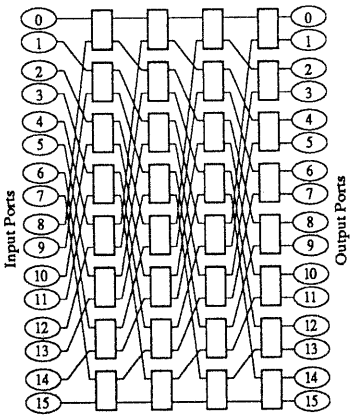
$$V_N(x,t) = \frac{1}{N}\sum_{i=0}^{N-1}\left\{C_i(x,t) - \frac{1}{N}\sum_{j=0}^{N-1}C_j(x,t)\right\}^2 \quad (1)$$

で与えられ, 各バケット毎の分散の総和 (ただし全バケット数を B とする) を,

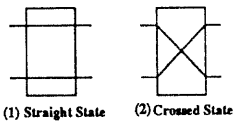
$$F(t) = \sum_{x=0}^{B-1} V_N(x,t) \quad (2)$$

で表すことにすると, 時刻 t において入力された全てのタプルについて接続後のコストの変化 $\Delta F = F(t+1) - F(t)$ が最小になるように接続状態を決定する, ということである.

しかし, 実際には N 個の入力と N 個の出力の間を接続する組合せは $N!$ 通りあるので, 全ての可能な接続状態について ΔF を算出し, 最小値を見つけるのは実現性に乏しい. そこで現在, SDC ではバケット平坦化機能を実現するのに図 4(a) に示されるような間接多段網 (オメガネットワーク) を用いている. オメガネットワークでは小規模のクロスバースイッチを多段にシャッフル接続して入出力間での経路を設定する. 各スイッチング素子に 2 入力 2 出力のものを用いることにすると, スイッチの接続状態には図 4(b) に示すように straight と crossed の 2 通りがあり, N 個の処理モジュー



(a) オメガネットワークの構成



(b) 各スイッチの接続状態

図 4: オメガネットワーク

ルがある時にはスイッチを 各段あたり $N/2$ 個, $\log_2 N$ 段接続する. 逆に, スイッチの段数を n とすると, ネットワークの大きさ (入出力ポートの数) は $N = 2^n$ となる.

このような多段構成のネットワークではネットワーク全体を集中して制御するのではなく, $N/2\log_2 N$ 個のスイッチそれぞれにおいて独立に ΔF を最小化するように経路を分散制御によって決定することができる. この場合, スイッチの 2 つの出力ポートに送られたパケットの分散は次のようになる.

$$V_2(x,t) = \frac{1}{4}\{C_0(x,t) - C_1(x,t)\}^2 \qquad (3)$$

ここで, $C_i(x,t)$ は時刻 t までに各スイッチの出力ポート $i(=0,1)$ に送られたパケット x に属するタプルの総数である. また, 時刻 t において入力ポート $j(=0,1)$ から入力されたタプルの属するパケット ID を $b_j(t)$ とすると, 接続状態が s に決定した時の C の増分は,

$$\begin{aligned} \Delta C_0(b_0(t),s) &= \Delta C_1(b_1(t),s) = 1-s \\ \Delta C_1(b_0(t),s) &= \Delta C_0(b_1(t),s) = s \end{aligned} \qquad (4)$$

となる. ただし, straight のとき $s=0$, crossed のとき $s=1$ である.

このとき, ΔF の値は,

$$\Delta F(s) = (\frac{1}{2} - s)\{D(b_0(t),t) - D(b_1(t),t)\} + \frac{1}{2} \qquad (5)$$

ただし,

$$D(x,t) \equiv C_0(x,t) - C_1(x,t) \qquad (6)$$

となるので, ΔF を最小にする接続状態は以下のように決定できることになる.

$$s = \begin{cases} 0 & \cdots \quad D(b_0(t),t) - D(b_1(t),t) < 0 \\ 1 & \cdots \quad \text{otherwise.} \end{cases} \qquad (7)$$

つまり各スイッチにおいて, 全てのパケット x について $D(x,t)$, つまりポート 0 に出力されたタプル数とポート 1 に出力されたタプル数との差をカウントしておき, 新たなタプルが入力された時にその 2 つのタプルの属するパケットに対応するカウンタの値を比較してその大小関係で経路を決定すればよい [3][5].

また, 状態が決定した後では式 (4) に従って $C_i(x,t)$ の値が変化するのでカウンタ $D(x,t)$ の値を更新する必要がある. つまり, ポート 0 から入力されたタプルの属するパケット b_0 に関しては straight ならインクリメント, crossed ならデクリメントを行ない, ポート 1 から入力されたタプルの属するパケット b_1 に関してはその逆を行なう.

以上が, これまでに提案されてきたパケット平坦化手法である.

3 任意数の処理モジュールに対するパケット平坦化

3.1 従来のパケット平坦化の問題点

従来のパケット平坦化アルゴリズムにおいては, 過去に出力されたタプル数のカウントに基づいてデータの行き先が動的に決まるということが特長であったが, 逆にこの方法ではデータの行き先をあらかじめ指定できないためにネットワークに接続されている全ての処理モジュールは常にデータを受けとれる状態になっていなければならない. SDC のように 1 つ 1 つの処理モジュール自体が大きく, 大容量の 2 次記憶系を含むシステムでは, 処理モジュール数が常に 2 の冪乗でなければならないという制限は非常に厳しいものである.

以上の観点から, 従来のパケット平坦化アルゴリズムが抱える問題点をまとめると次のようになる.

- どれか 1 つでも処理モジュールが故障すると, 運転を再開するには処理モジュール数を元の半分にして 2 の冪乗に合わせなければならない. その結果, 正常な処理モジュールの約半分がアクセス不能になるのでデータの冗長性を大きくする必要があり, 障害時の縮退運転には大きな犠牲が伴う.
- システムを拡張するためには常に現在と同じだけの処理モジュールがもう 1 組必要になり, 処理モジュール数を増やすことの負担が急激に大きくなる. そのため, 蓄積されたデータ量の増加に応じて徐々にシステムを大きくしていくことができない.
- 処理モジュール全体ではなく数台ある内の 1 部の磁気ディスク装置が故障した場合や, 処理モジュール毎のバージョンが異なる場合には性能の異なる処理モジュールが混在することになる. ところが, 従来のアルゴリズ

ムでは平坦化されるのは各処理モジュールに分配されるデータ量なので,結果として処理時間が処理モジュール毎に異なってしまい,真の意味での負荷分散ができない.

- システム全体をいくつかのパーティションに分けてそれぞれ独立に処理をさせたい時に,2の冪乗以外のパーティショニングが行なえないため,データの分布が一部の処理モジュール群に偏っている時には処理モジュールの使用効率が低下してしまう.

以上のような問題点を解決するために,ネットワークに接続された一部の処理モジュール間でパケット平坦化を行なう機能が必要である.

3.2 パケット平坦化アルゴリズムの拡張

処理モジュールの一部だけを用いてパケット平坦化を行なう場合には,各処理モジュール毎に送られるデータの量を変えなくてはならない.したがって,処理モジュールiに送られるべきデータ量をn_iで表すことにすると,平坦度の評価に使う(1)式の分散の定義を次のように変更する必要がある.

$$V'_N(x,t) = \frac{1}{N_a}\sum_{i=0}^{N-1}\left\{\frac{C_i(x,t)}{n_i} - \frac{1}{N_a}\sum_{j=0}^{N-1}\frac{C_j(x,t)}{n_j}\right\}^2 \tag{8}$$

ここで,n_iは処理モジュールiが アクティブな時 1となり,そうでない時は 0となる.また,N_aはアクティブな処理モジュールの総数で,

$$N_a = \sum_{i=0}^{N-1} n_i \tag{9}$$

である.$n_i = 0$となる処理モジュールiに対してはタブルを全く送らない,すなわち$C_i(x,t) = 0(^\forall x, ^\forall t)$とするような制御が必要である.

ここでも,従来のパケット平坦化と同様に各スイッチ毎の分散制御を行なうので,式(3)に対応するパケットの分散を示すと,

$$V'_2(x,t) = \frac{1}{4}\{w_0C_0(x,t) - w_1C_1(x,t)\}^2 \tag{10}$$

となる.ただし,$w_i(i=0,1)$は,第m段目のスイッチに対して,

$$w_i = \frac{2^m}{\sum n_j} \tag{11}$$

で定義される重み付けであり,和は出力ポートiから到達可能な全ての処理モジュールについてとる(3.3節参照).2^mは正規化のための係数で,全てのモジュールがアクティブな時は$w=1$となるが,一般には$w \geq 1$である.

パケットの分散の定義が変更されると,当然コスト関数Fが変わってくるので式(5)と同様にΔFを求めると以下のようになる.

$$\Delta F(s) = -\frac{w_0+w_1}{2}s\{D'(b_0(t),t) - D'(b_1(t),t)\} + const. \tag{12}$$

ここでD'の定義は,

$$D'(x,t) \equiv w_0C_0(x,t) - w_1C_1(x,t) \tag{13}$$

である.ΔFを最小にする接続状態は従来と同様,以下のように決定できる.

$$s = \begin{cases} 0 & \cdots \quad D'(b_0(t),t) - D'(b_1(t),t) < 0 \\ 1 & \cdots \quad \text{otherwise.} \end{cases} \tag{14}$$

つまり,メモリに記憶する内容を式(6)で表されるポート間での出力タブル数の単純な差から,式(13)で表される重み付けした差に置き換えることで一部の処理モジュール間でのパケット平坦化が実現できることになる.ただし,例外としてw_iが有限でなくなるスイッチ,つまり,出力ポートiの先にアクティブな処理モジュールが全くないスイッチでは,出力ポートiにはタブルを送れないのでどちらかの入力をブロックする必要がある.

また,状態決定後には式(4)と式(13)に基づき,カウンタD'の値に,パケットb_0に関してはstraightなら$+w_0$,crossedなら$-w_1$を加え,パケットb_1に関してはstraightなら$-w_1$,crossedなら$+w_0$を加える必要がある.そのため,各スイッチ素子ではカウンタ$D'(x,t)(x=0,1,...,B-1)$に加えて重み$w_i(i=0,1)$も保持する必要がある.

3.3 各スイッチに対する重みの決定法

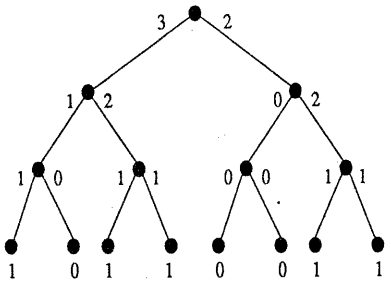
次に問題になるのは,任意のアクティブな処理モジュールの集合が与えられた時に各スイッチング素子の出力ポートに対する重み付けw_iを設定する方法であるが,直観的には図5(a)に示すようなバイナリツリーの数え上げアルゴリズムで説明できる.この場合の手順は,

1. 全てのリーフについて,アクティブなら 1,そうでなければ 0を マークする.
2. リーフ側のノードから順に,右側の子ノードの値と左側の子ノードの値とを加えて 自分のノードの値とし,その値を親ノードに伝える.
3. ルートノードに達するまで 2.を繰り返す.

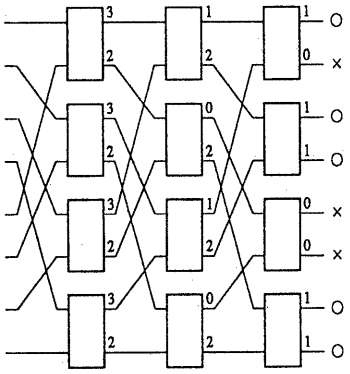
となる.

オメガネットワークのような間接多段網は入力側をルートとするバイナリツリーを少しずつずらしながら重ね合わせたものと考えられるので,上記のアルゴリズムにおいて“ノード”を“スイッチング素子”に置き換え,値を伝搬する際に常に 2つの親に同じ値を伝えるようにすれば,各スイッチの出力ポートから到達可能な処理モジュールの数を一意に決定することができる(図5(b)).

ネットワークの大きさが2^nのとき,第m段(出力側から順に$0,1,...,n-1$)のスイッチの出力ポートから到達可能なアクティブな処理モジュールの数を$N_m(i)$(iはポートの通し番号とする)で表すことにすると,オメガネットワークで



(a) バイナリツリーの場合



(b) オメガネットワークの場合

図 5: アクティブな処理モジュールの数え上げ

は,

$$N_m(i) = \sum_{k=0}^{2^m-1} n_{(i_{n-m-1}\cdots i_0 0\cdots 0)_2+k} \quad (15)$$
$$(\text{ただし } i=(i_{n-1}i_{n-2}\cdots i_0)_2 \text{ とする})$$

となる. つまり, 式(11)における和は, スイッチのポート番号の下位 $n-m$ ビットとモジュール番号の上位 $n-m$ ビットが一致する 2^m 個の処理モジュールについてとればよいことになる.

また, 式(11)によると重み w_i は N_m の逆数に比例することになるが, 実際にはスイッチの2つのポート間での比のみが関係する. したがって, スイッチの通し番号が j の時,

$$w_0 : w_1 = \frac{2^m}{N_m(2j)} : \frac{2^m}{N_m(2j+1)} = N_m(2j+1) : N_m(2j) \quad (16)$$

であるから, w_0 には ポート 1 から到達可能な処理モジュール数, w_1 には ポート 0 から到達可能な処理モジュール数を設定すればよく, 逆数を計算する必要はない.

実際に上で述べた方法にしたがって重みを決定するには, スイッチ自身に設定を行なわせる機能を持たせるか他のプロセッサで計算してスイッチに与えるかの2通りが考えられる. スイッチ自身に値を設定させる場合は, ネットワークの大きさを N とした時に $O(\log N)$ で決定できるという利点があり, ダイナミックな設定も可能になる. しかし, スイッチ間のリンクを双方向性にしなければならないためハードウェアが複雑化し, 転送スピードが低下するおそれがある. 単一のプロセッサで重みを計算するとしても, 同じ設定値が何度も現れるため $N/2\log N$ 回の繰り返しは必要なく, 実際には1種類のツリーについて計算すればよい. したがって, 加算の実行は $N-1$ 回で十分である.

3.4 重みの効果とバイアス値について

スイッチのどちらの出力ポートからもアクティブな処理モジュールに到達できるスイッチ素子においては, タブルを通した後のカウントの仕方が重みによって異なるだけで, 出力タブル数を2つのポート間で変化させるような効果はあまり大きくないように思われる. しかし, 実際にはアクティブでない処理モジュールに至る経路において必ず片方のポートにしかタブルを出力できないスイッチが存在し, そのスイッチでは2つのポートに同時にタブルが入力された場合は, 片方のタブルをブロックすることになる. そのため, そのスイッチからさらに前段にさかのぼってブロックが発生し, 長い時間で平均すると出力されたタブル数の差となって現れる.

このように, 2の冪乗でない処理モジュール間でパケット平坦化を行なう際には常にブロックが多く発生する. そのため, スイッチの両方のポートに同時にタブルが入力される割合は通常動作時に比べて小さくなり, 一方のポートのみにタブルが到着した場合の振舞いが相対的に重要になってくる. そこで, 式(12)の ΔF をタブルがポート 0 にのみやってきた場合について求めてみると,

$$\Delta F(s) = -\frac{w_0+w_1}{2}s\left\{D'(x,t)+\frac{1}{2}(w_0-w_1)\right\} + const. \quad (17)$$

となり, カウンタ D' の値そのものではなく, それに定数を加えたものを比較の対象とする必要があることが分かる.

この定数の値は, スイッチの2つのポートの重みの差に比例するので, そのスイッチの不平衡度を表すと考えられ, あらかじめ重みの軽い方のポートに出力されやすくなるようバイアスする働きがあると考えられる. このときの 1/2 という係数は, あくまでも静的な解析から得られたものなので実際のタブルの発生の仕方に応じて最適な値は変化するものと考えられる. そこでシミュレーションを行なう際にはこの係数をパラメータとし,

$$D'(x,t)+M(w_0-w_1) \quad (18)$$

を実際の比較に用いることとした.

このようにバイアス値を考慮に入れることにすると，経路決定にかかるステップ数が増加することになり，好ましくないように思われる．しかし，読み出されたカウンタの値に毎回一定の値が加算されるので，実際にはカウンタの初期値を0でなく $M(w_0 - w_1)$ とすることで余分なステップを隠すことができる．したがって，新たなタブルが到着した時の経路決定までの状態遷移は基本的には以下のようになる．

1. タブルの先頭に書かれているパケットIDをレジスタにラッチする．
2. パケットIDをアドレスとしてカウンタ値 (D') をメモリから読み出す．
3. カウンタ値を比較する．
4. 比較器の出力に基づいてスイッチの接続状態を決定．
5. 転送開始．
6. カウンタを更新 $(+w_0, -w_1)$ する．
7. カウンタをメモリに書き戻す．

4 シミュレーションによる評価

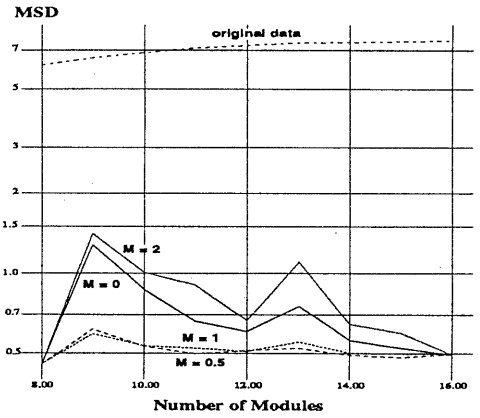
ここでは，ネットワークのサイズを固定し，アクティブな処理モジュールの数を1つずつ減らしていった時のパケット分布の平坦度と処理時間についてシミュレーションした結果を述べる．

1タブルは10ワードの固定長データとし，発生間隔は指数分布に従うものとした．また，処理モジュールはネットワークの端から1つずつ，ネットワークの大きさの半分になるまで減らしていった．

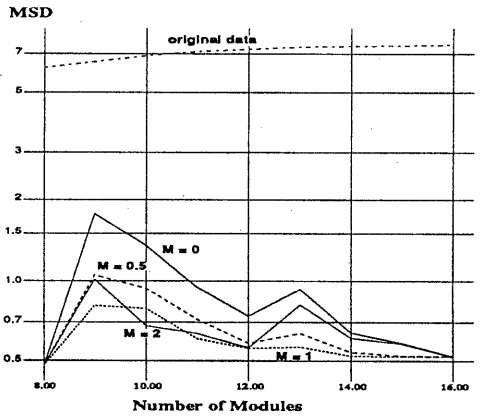
図6と図7は $N = 16$, $B = 128$, 処理モジュールあたりのタブル数 $T = 1024$ とした時のパケット分布の平均標準偏差と処理時間をそれぞれ示している．ただし，任意の時点でのタブルの発生確率 $L = 0.05, 0.1$, 前節で述べたバイアス値の係数 M は 0, 0.5, 1, 2 とした．

全体の傾向として $N/2 + 1$ 台のときは平坦度が悪化するが，$\frac{3}{4}N$ 台以上用いる場合は若干処理時間が増えるものの，ほぼ通常時 (処理モジュール数が16，または8台の時) に比べて遜色のない性能を示していることが分かる．また，$M = 0$ のときはその他の場合と比べて平坦度，処理時間ともかなり劣っていることが分かる．従って，バイアス値の導入は2の冪乗でない処理モジュール間での負荷分散においては，必須であるといえる．

さらに，タブルの発生率が変化すると同じ M の値であっても平坦度の特性が変わってくる．発生率が小さい時は，ほぼ前節の理論値 $M = 1/2$ で良い特性が得られているが，発生率が少し大きくなると M の値が大きい方が特性が良くなってくる．したがって，バイアス値はタブルの発生率 (結合演算の分割フェーズにおける選択率) に基づいて，あらかじめ最適な値を予想して設定しておく必要がある．



(a) タブル発生率 $L = 0.05$



(b) タブル発生率 $L = 0.1$

図 6: 処理モジュール数対平均標準偏差

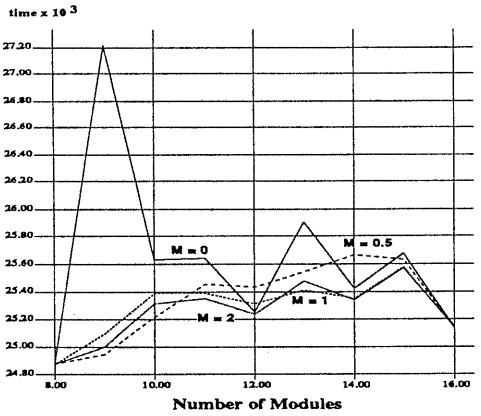
5 まとめ

以上，パケット平坦化機能を持つオメガネットワークにおいて任意数の処理モジュール間での負荷分散を可能にするために，従来のアルゴリズムの拡張を行なった．また，シミュレーションによりこのアルゴリズムが処理モジュール数の変化に対してほぼ一定の負荷分散能力を有することを確認した．

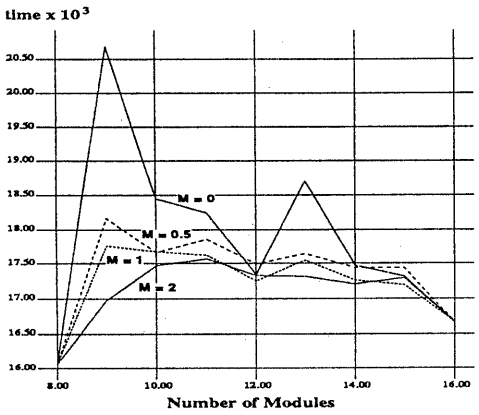
その結果，

- 処理モジュールの故障に対するロバスト性
- システム構成の柔軟性
- 各処理モジュールの性能の差への対応

などの目的が達成された．ただし，3. については述べなかったが，処理モジュールを数える時に n_i として小数も許し，$0 \leq$



(a) タプル発生率 $L = 0.05$



(b) タプル発生率 $L = 0.1$

図 7: 処理モジュール数対全処理時間

$n_i \leq 1$ とすることで簡単に実現できる. この場合は, 重み w_i の精度を必要に応じて数ビット増やす必要がある.

また, 平坦化能力を高めるためには, タプル数のカウント値に対して一定のバイアス値を加えることが有効であることも分かった. この値は, スイッチの不平衡度に比例し, タプルの発生確率によりその効果が変化するが, この値の持つ働きと最適値の決定法についてはさらに今後の詳細な解析が必要であると思われる.

拡張バケット平坦化アルゴリズムをハードウェアで実現するためには, 各スイッチに次のようなものが必要となる.

- カウンタの値 D' を記憶するためのメモリ.
- w_i を保持するレジスタ.
- カウンタの値を比較する比較器.
- カウンタの値を更新するための加算器.

接続状態の制御には比較器からの出力がそのまま使えるのでそれほど大規模な回路は必要としない. この内, カウンタメモリと比較器についてはこれまでの平坦化アルゴリズムでも必要とされていた [7]. また, インクリメンタ / デクリメンタも従来から含まれていた. 新たに必要となったのは重みを保持するレジスタと, より精度の高い加算器およびカウンタメモリである. カウンタの更新は, 状態決定後にデータの転送とオーバーラップして行なえるので, 加算器の性能はそれほど問題にならない. したがって, 新たなアルゴリズムを実現するにはわずかな変更しか要求されないと言える. 重要なのはカウンタの精度の向上に伴う外部ピン数の増加への対応である. なるべく高速でスケーラビリティ (転送データ幅方向への) に優れたネットワーク系を作るには, スイッチのビットスライス化も検討する必要がある.

今後は, このアルゴリズムを実現するスイッチ素子を実装し, その性能評価を行ないたい. また, このアルゴリズムを応用してネットワークを任意の大きさのパーティションに分割することも検討し, そのときの動作特性などについて調べていきたい.

参考文献

[1] 平野, 原田, 中村, 小川, 楊, 喜連川, 高木: "スーパーデータベースコンピュータ SDC のアーキテクチャ", 並列処理シンポジウム, pp.137, 1990.

[2] 平野, 原田, 中村, 楊, 喜連川, 高木: "スーパーデータベースコンピュータ SDC のソフトウエア", 電子情報通信学会技報, pp.7-12, 1990.

[3] 喜連川, 小川: "バケット平坦化機能を有するオメガネットワーク", 情報処理学会論文誌, 30(11) pp.1494, 1989.

[4] 小川, 喜連川: "スーパーデータベースコンピュータ SDC におけるバケット分散方式による並列ハッシュ結合演算法", 電子情報通信学会技報, 1989.

[5] Kitsuregawa, M. and Ogawa, Y.: "A New Parallel Hash Join Method with Robustness for Data Skew in Super Database Computer (SDC)", *Proc. of 16th Int. Conf. on VLDB*, pp. 210-221, 1990.

[6] 相場, 喜連川, 平野, 高木: "スーパーデータベースコンピュータにおけるバケット平坦化オメガネットワークの動作特性", 電子情報通信学会技報, 1991.

[7] 田村, 原田, 平野, 中村, 喜連川, 高木: "スーパーデータベースコンピュータ (SDC) におけるデータネットワークの論理設計", 情報処理学会第 44 回全国大会講演論文集, pp. 4-207, 1992.