平成 12 年度

学士学位論文

DDMP による SOM の並列化に関する研究

A Study on Parallel Algorithms of SOM for DDMP

1010439　　福永　諭

指導教員　　Ruck Tawonmas 助教授

2001 年 2 月 5 日

高知工科大学 情報システム工学科

要 旨

DDMP による SOM の並列化に関する研究

福永　諭

Kohonen によって提唱された自己組織化 (Self-Organizing Map,SOM) は，クラスタリング等の様々な分野での応用が提案されている．しかし，ユニットの数が増えるにつれて，計算量が膨大になるため，並列処理による高速化が求められる．本研究では，入力層分割という手法をとりそのアルゴリズムにホストの有無という形で差をつけ，DDMP(Data Driven Media Processer) 上で実装して評価する．

キーワード　　自己組織化マップ,SOM,DDMP, 並列処理, 入力層分割法

Abstract

A Study on Parallel Algorithms of SOM for DDMP

Satoshi Fukunaga

Self-Organizing Map (SOM) proposed by Kohonen can be applied to various fields such as clustering However, since the amount of computation becomes huge as the number of unite increases, it is crucial to improve the speed by parallel processing We take the technique of Input Layer Parallel Model in this study. We compare two configurations of this model, one with host and the other without host, on Data Driven Media Processor (DDMP).

key words SOM, Parallel Algorithms, DDMP

目次

図目次

第 1 章

まえがき

Kohonen により提案された自己組織化マップ（Self-Organizing Map,SOM）は，クラスタリング等の様々な分野での応用が提案されている [1]．しかしながら，SOM は入力層，競合層の 2 層を用いており，各層間は完全結合で結ばれるため，ユニット数を増やすと計算量の爆発が起こる．そのため，並列処理による高速化手法が提案されている．しかしながら，従来の並列学習法は，競合層を分割して各プロセッシングエレメント (PE) に割り当てる競合層分割法を基本としており，SOM の学習アルゴリズムが持つ特性のために学習後半で PE の負荷が不均一となり効率的ではない．

そこで，入力層分割法 [2] という手法を用いる．入力層分割法は，各 PE が入力層のユニット一つを担当し，競合層の各ユニットに対して重みを持つ．したがって，競合層のどのユニットが勝利，あるいは近傍となって重みを更新してもすべての PE が同等の処理をするため負荷は均一となる．さらに，入力層のユニット数が競合層のユニット数より多くなるにつれて，入力層分割法のほうが競合層分割法と比較して高速となることが報告されている．

Data Driven Media Processer(DDMP)[3] という，データ駆動原理を採用したプロセッサがある．データ駆動処理型の大きな特徴は個々の演算に必要なデータが揃った時点でそれぞれの演算が起動され，一旦起動された演算相互間には，何らの依存関係がないことにある．すなわち，一旦起動された演算は，他の演算とまったく無関係に実行される．また，自己タイミング方式の特徴によって，信号を処理していない場合の消費電力は，ほぼ零になる．このように，電力対性能比の向上が達成されている．

本研究は入力層分割 SOM を DDMP 上で実装し，その性能を評価する．

第 2 章

Self-Organizing Map

2.1 SOM のアルゴリズム

SOM は，入力信号の持つ位相を教師信号を用いることなく競合層上へ写像することが出来るニューラルネットワークの一種である．SOM は入力層と競合層の 2 層からなり，各層間は完全結合で結ばれている．層間の結合には重みが与えられており，学習前に乱数で初期化される．SOM を以下に述べる競合層学習で学習させることにより，入力信号の位相を写像したトポグラフィックマップを得ることができる．

ある入力信号が$\boldsymbol{x}$が与えられたとき，競合層のユニット i は自分が持つ重み $\boldsymbol{w_i}$ と入力信号間の距離を計算する．多くの場合はユークリッド距離が用いられるが，この距離が最小の競合層ユニット c を勝利ユニット (ウィナー) とし，(2.1) の式で表す．

$$
\begin{aligned}
c &= argmin_i\{||\boldsymbol{x} - \boldsymbol{w_i}||\} \\
||\boldsymbol{x} - \boldsymbol{w_c}|| &= min_i\{||\boldsymbol{x} - \boldsymbol{w_i}||\}
\end{aligned} \tag{2.1}
$$

競合層上で c の近傍に存在するユニットは，c からの距離に反比例した強度で反応し，重みを式 (2.2) により，更新する．

$$
\boldsymbol{w_i}(t+1) = \boldsymbol{w_i}(t) + h_{ci}(t)(\boldsymbol{x}(t) - \boldsymbol{w_i}(t)) \tag{2.2}
$$

ここで，$t = 0, 1, 2, ...$ は離散時間座標である．更新過程で $h_{ci}(t)$ は近傍関数と呼ばれ，式 (2.3) によって定義される．

$$
h_{ci}(t) = h(||\boldsymbol{r_c} - \boldsymbol{r_i}||, t) \tag{2.3}
$$

ここで,$\boldsymbol{r_c}$,$\boldsymbol{r_i}$はそれぞれ競合層におけるユニット c と i の位置ベクトルであり，$||\boldsymbol{r_c} - \boldsymbol{r_i}||$ が増加するにつれて $h_{ci} \to 0$ とする．

必要な数，十分な時間の入力信号$\boldsymbol{x}$を用いて競合学習を行うことにより，競合層の各ユニットが持つ重みを参照ベクトルとするボロノイ・モザイク領域が形成される．図 (2.1) に 200 個の乱数を学習パターンとして，1000 回の学習を行ったときのトポグラフィックマップの生成過程を示す．図 (2.1) 中のマップに見られる頂点は，競合層ユニットの重みを 2 次元上にプロットしたものであり，ユニット間の空間的隣接関係を，各頂点を結ぶ線で表している．重み空間に一様に分布する乱数を用いて SOM の学習を行った場合，競合層ユニットの重みは重み空間に一様に広がり，最終的に格子状のトポグラフィックマップが得られる．図 2.1(a) に示すように，学習初期では重みはランダムに初期化されているため，重み空間における重みの分布とユニットの空間的な配置には規則性が見られない．しかし，(b) → (c) と学習が進むにつれて，重みは重み空間に一様に拡散していき，同時に各ユニットの空間的隣接関係を表すトポグラフィックマップが形成され始める．1000 回学習後では，図 2.1(d) のようなトポグラフィックマップが得られる．

2.2　近傍

勝利ユニットの周りには近傍領域が定義される．近傍サイズは最初大きくとっておき，時間軸とともに減少させていく．ノード c の周りのユニットを近傍集合とし,N_c と定義する (これによって時間の関数としての $N_c = N_c(t)$ が定義できる)．もし，$i \in N_c$(i が N_c 内のノード) なら $h_{ci} = \alpha(t)$，$i \notin N_c$(i が N_c 外のノード) なら，$h_{ci} = 0$ である．この時，$\alpha(t)$ の値を学習率係数と言う $(0 < \alpha(t) < 1)$．$\alpha(t)$ は普通，時間軸上において単調減少させる (式 2.4)．α_0 は，α の初期値であり，一般的に 0.2〜0.5 の値を選ぶ．T は，行われるべき学習での予定された全更新学習回数である．

$$\alpha(t) = \alpha_0(1 - \frac{t}{T}) \tag{2.4}$$

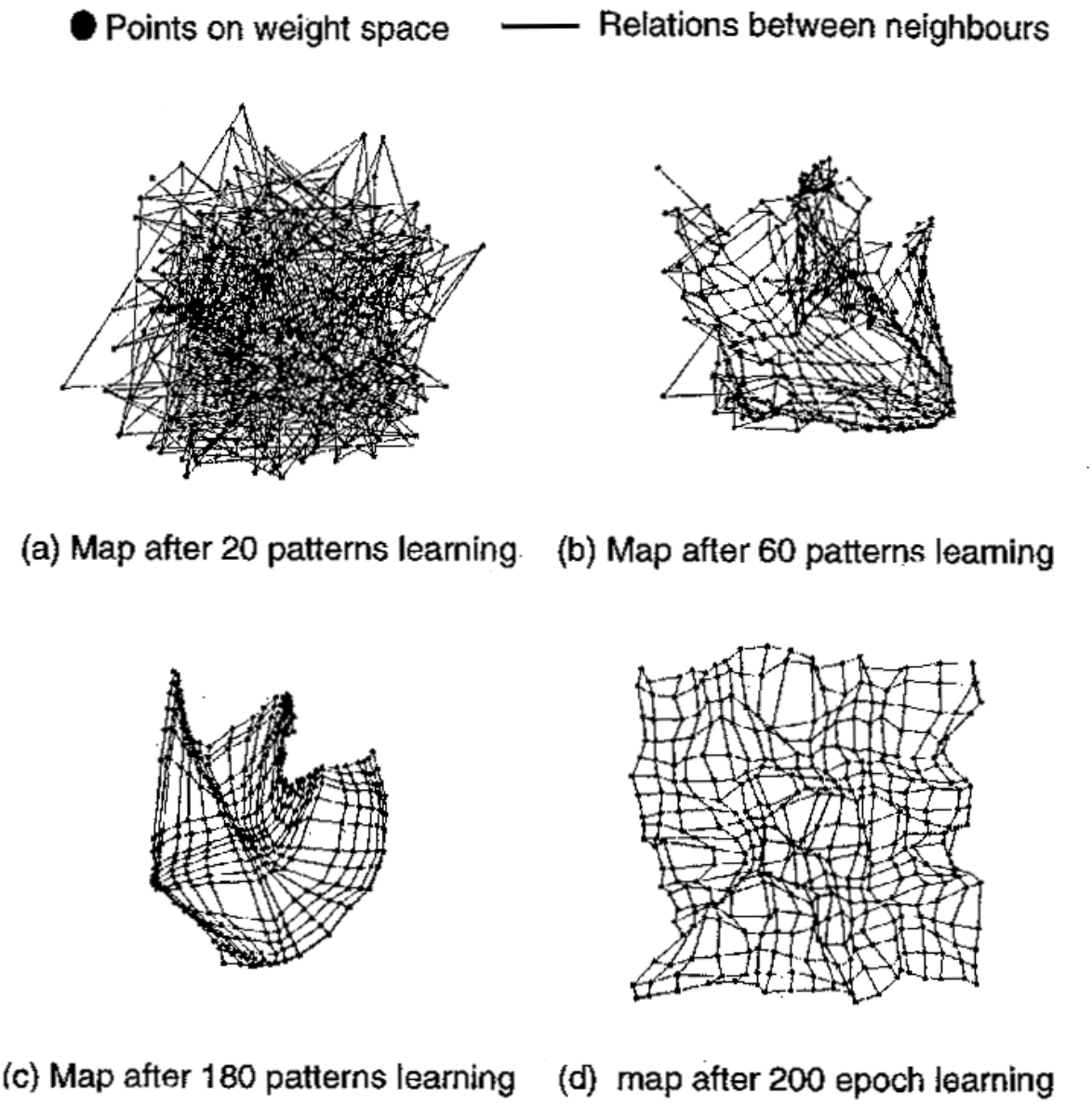


図 2.1　トポグラフィックマップの生成過程

また，近傍領域 $N_c = N_c(t)$ についても同様に，

$$N_c(t) = N_c(0)(1 - \frac{t}{T}) \tag{2.5}$$

と表すことが出来る．$N_c(0)$ は初期値である．

2.3　並列 SOM

SOM は入力層，競合層を 2 次元上に配置するため，ユニット数の増加は大幅な計算量増加につながる．そこで並列化による処理の高速化が求められる．この節では，従来の並列手法である，競合層分割と本研究の実装で取り入れた入力層分割について説明する．

2.3.1　競合層分割法

競合層分割法は従来行われてきた SOM の並列化の手法である．競合層分割法は SOM アルゴリズムの中で最も時間を要する更新部分の並列化を考え，競合層を分割して各 PE に割り当てる．こうすることによって，更新は勝利ユニットとその近傍ユニットを割り当てられた PE だけが行い，ほかの PE はアイドル状態で処理の終了を待つ．しかし，この方法では勝利ユニットとその近傍の割り当てられた PE のみに負荷がかかり，全体的に負荷の不均一となり，効率的ではない (図 2.2)．

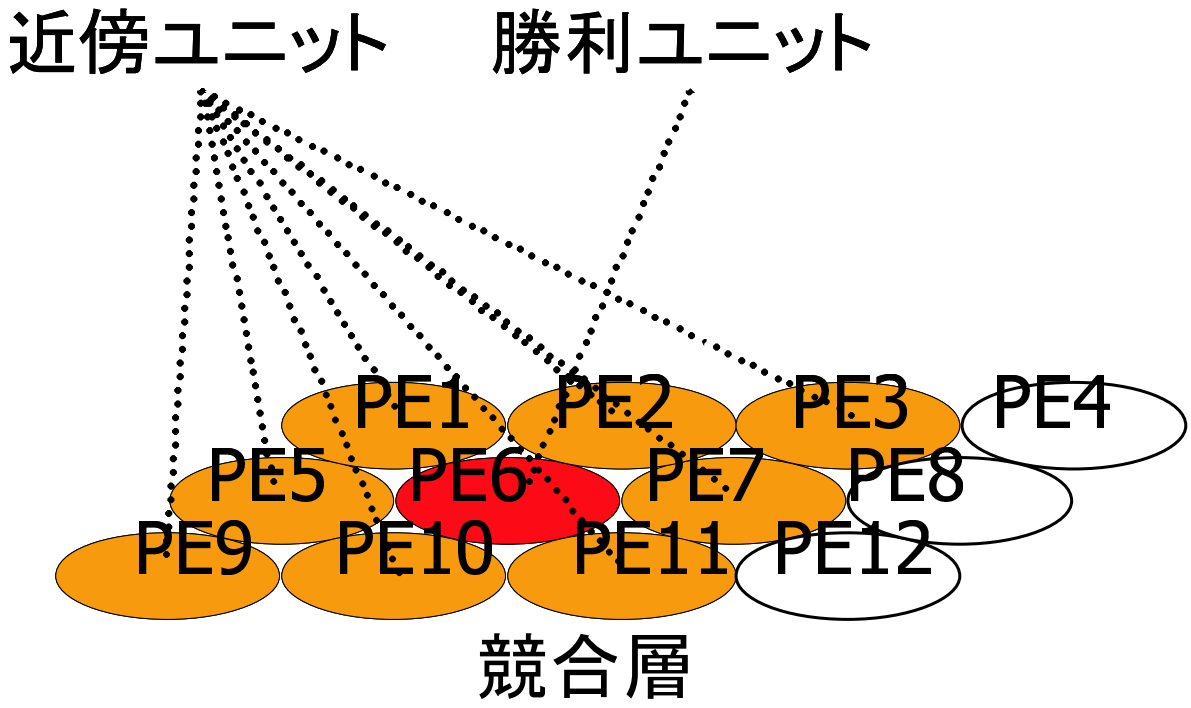


図 2.2　競合層分割法の PE 割り当て

2.3.2　入力層分割法

これまで行われた並列手法である競合層分割法による並列学習では，PE 間の負荷が不均一であったり，画像などの高次元入力では十分な高速化が行えないなどの問題が存在する．以下では，PE 間の負荷の不均一が原理的に発生しない入力層分割による並列化を解説する．

入力層分割 SOM の各 PE は，入力層のユニット一つを担当し，競合層の各ユニットへの重みを持つ．(図 2.3)．したがって，競合層のどのユニットがウィナー，あるいは近傍となっ

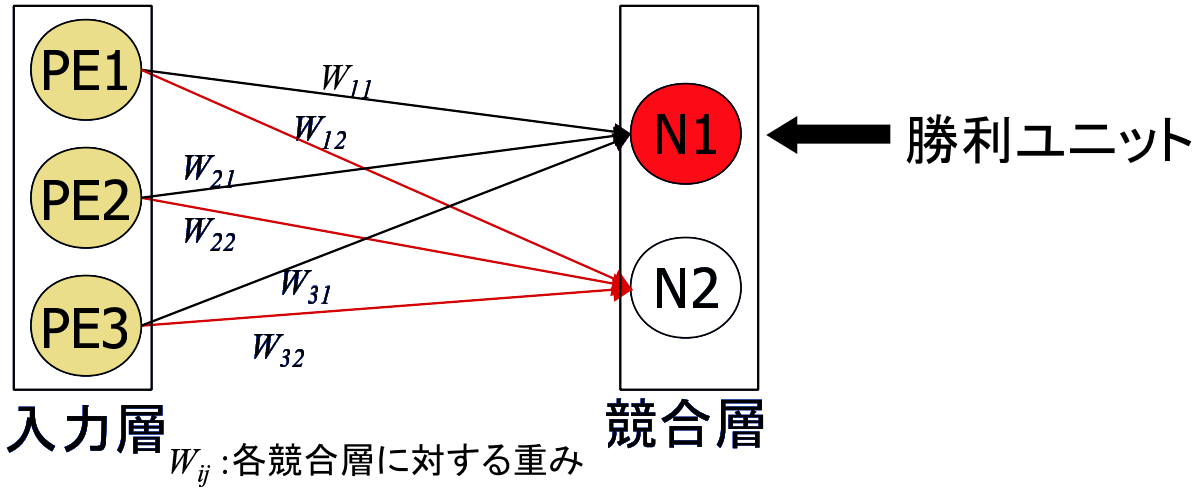


図 2.3　入力層分割の概要

て重みの更新を行っても，すべての PE が同等の処理をする為に各 PE の負荷は均一になる．アルゴリズムを示すにあたって対象とする SOM は入力層，競合層の 2 層とし，それぞれ，M×M，N×N 個のユニット数からなるとする．ある時刻 t における入力ユニット i と出力ユニット j 間の重みを $W_i^j (i = 1, \cdots, M^2, j = 1, \cdots, N^2)$ とする．この SOM を M×M 個の PE に分割し，各 PE は N^2 次元の重みベクトルを持つ．学習パターン $R = \{t_1, \cdots, t_{[}M^2]\}$ が入力層ユニットに入力された場合を考える．各 PE は学習パターンで対応する要素を用い，入力ベクトル要素 t_i と重みベクトル W_i^j との間のユークリッド距離 d_i^j を式 (2.6) に基づき計算する．

$$d_i^j = \sqrt{(t_i - W_i^j)^2} \qquad j = 1, \cdots, N^2 \tag{2.6}$$

各 PE_i が d_i^j を計算した後で，i について d_i^j を式 (2.7) により，集計し，結果を全 PE にブ

ロードキャストする．

$$d^j = \sum_{i=1}^{M^2} d_i^j \tag{2.7}$$

その結果，競合層ユニット j と学習パターン間の距離 d^j をすべての PE が持つことになる．各 PE は独立に d^j が最も小さいものをウィナー c として選び，式 (2.8) に表される近傍関数 h_{cj} を用いて c とその近傍 $N_c(t)$ の重みを更新する．

$$h_{ci} = \begin{cases} \alpha(t) & i \in N_c \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

したがって，各 PE は同一個数の重み要素を更新することになり，負荷の不均一は原理的に発生しないことになる．

入力層分割法による SOM の並列学習アルゴリズムを以下に示す．

1. Initialize $W_i^j(0)$.
2. Distribute t_i to each PE i.
3. **do** on each PE i,
4. **for** $j = 0$ to N^2,

   $d_i^j = ||W_i^j(t) - T_i||$.

   broadcast d_i^j and caluclate $d^j = \sum_i d_i^j$,then send it to PE i.

   decide winner $c = arg\min_j d^j$ and neighbours $N_c(t)$ on each PE.

   $W_i^j(t+1) = W_i^j(t) + h_{cj}(t)(W_i^j(t) - t_i) \qquad j \in N_c(t)$
5. **while** $d^c < LIMIT$, or learn for max iterations.

第 3 章

Data Driven Media Processer

DDMP はマルチメディア信号処理向けに開発された，データ駆動型信号処理プロセッサである．主な特徴として，データ駆動処理方式の採用があげられる．パイプライン処理方式は，ハードウェアの処理能力向上手段として，理想的であると考えられている．しかし，パイプライン処理方式が最大の効果を発揮するのは，パイプラインの中に，一定のデータ流量が定常的に確保されていることが大きな要件となる．

さらに，パイプラインの複数の段からの外部資源へのアクセス競合などによって，パイプライン段相互間のインターロッキングが要求される環境では，いわゆるバブルの発生によってパイプライン処理の効果が減殺されるばかりではなく，パイプライン制御機能のいたずらな複雑化を招く．これらの要請をシステム的に保証する最も明快な手段は，パイプライン内を流れる複数のデータの組の処理が相互に完全に独立であるような処理原理の導入である．

データ駆動型の大きな特徴は，個々の演算に必要なデータがそろった時点でそれぞれの演算が起動され，いったん起動された演算相互間には，何ら依存関係がないことにある．すなわち，いったん起動された演算は，ほかの演算とまったく無関係に実行される．. また，自己タイミング方式の特徴によって，信号を処理していない場合の消費電力はほぼ零となり，電力対性能比の向上が達成されている．

DDMP の専用型プロセッサの構成は，OCP(Operation and Control Procesor) と VMP(VideoMimoryControlProcesser) が各 2 つ，計 4 つのプロセッサからなっており (図 3.1)，シミュレーション時には設定ファイルを用いて，この構成は任意に変更できる．本研究では，OCP を 4 つ接続したプロセッサ構成で入力層分割 SOM の実装を行った．

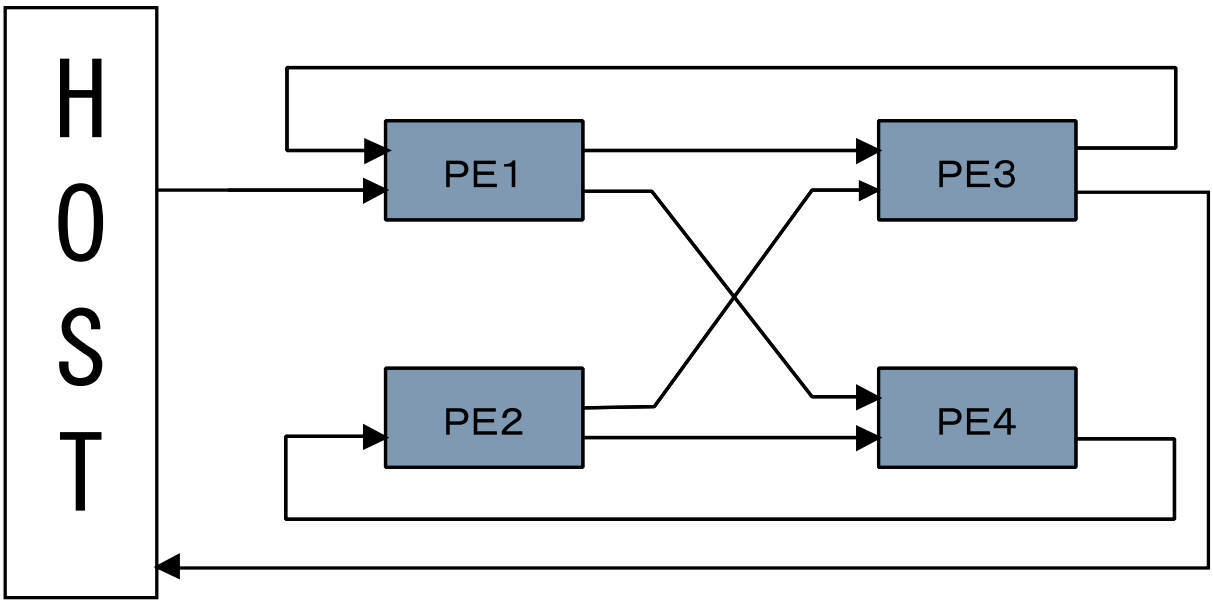


図 3.1　DDMP 内部の各プロセッサの接続図

第 4 章

実装

4.1 実装するアルゴリズム

入力層分割 SOM を DDMP のフローグラフシミュレータ上で実装し，その学習時間を評価する．SOM を実装する最初の試みとして，DDMP の 4 つのプロセッサのうち，一つをホストとして使用する場合と，そうでない場合の比較を行う (図 4.1)．このとき，入力層，競合層の数を共に 2，4，6 個とし，計 18 個の学習時間を計測し評価する．近傍は無し，学習回数は 1 回とする．以下にアルゴリズムを示す．

ホストあり

1. 各ＰＥで入力層のユニットと競合層のユニットとの誤差をとる
2. 1 で求めた誤差をホストに送信
3. ホストで競合層の各ユニットについての誤差を集計し，勝利ユニット及びその近傍ユニットを決定
4. 3 で得たデータを各プロセッサに送信し，各プロセッサで勝利ユニット及びその近傍ユニットに関連した重みを更新

ホストなし

1. 各ＰＥで入力層のユニットと競合層のユニットとの誤差をとる
2. 1 で求めた誤差を各プロセッサに送信
3. それぞれのプロセッサで競合層の各ユニットについての誤差を集計

4. 各プロセッサで勝利ユニット及びその近傍ユニットを決定し，それらに関連した重みを更新

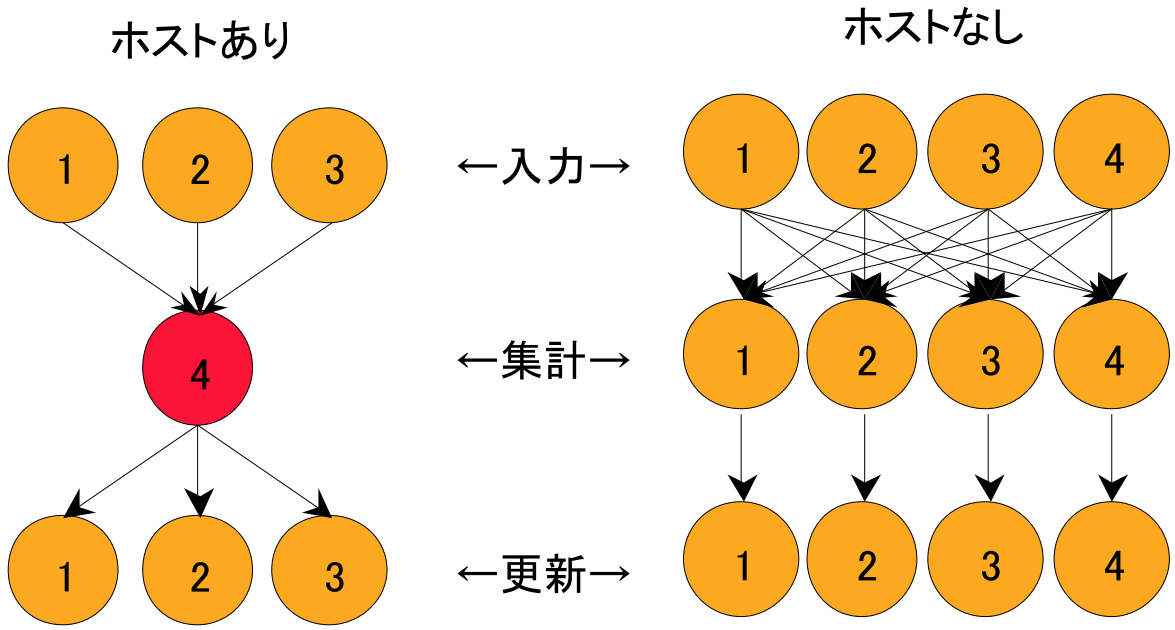


図 4.1　ホストの有無によるアルゴリズムの比較

4.2　準備

拡張子	用途	起動アプリ
*.fg *.mod	フローグラフエディタで作成したプログラム	フローグラフエディタ fed 600.exe
*.fe	アセンブラ出力ファイル（オブジェクトファイル） データフローグラフ（DFG）実行形式	シミュレーター fsim 4g
*.fs	フローグラフエディタ中間ファイル（アセンブラソース）	テキストエディタ
*.inp	テキスト形式で記述 入力パケット用ソースファイル	テキストエディタ
*.pkt	標準型入力パケットファイル	

図 4.2　DDMP で使用するファイルの一覧

プログラムの実装，実行に必要なファイルを図 (4.2) に記す．この他に，OCP を 4 つ接続したプロセッサで評価するためのコンフィグファイル (config.emu) も用意した．また，DDMP のプログラムを作成するにあたって，各試行ごとにパケットファイルを作成する必要がある．テキストエディタで拡張子 inp のファイル (図 4.3) を作成し，入力パケット生成ツール (mkpkt_4g.exe) を実行すれば拡張子 pkt のファイルを作ることが出来る．最上行の #4 は, パケットの投入感覚を表し，この場合，投入感覚は 4 サイクルである．その次の行の %1 は，一度に投入するパケット数である．この場合は，1 度に一つずつのパケットを投入することになる．

```
#10↓
%1↓
!        FD#      LN#      PX#      PE#      ENT#     DATA.
! N1↓
         0        0        0        0        1        10↓
         0        0        0        0        3        17↓
         0        0        0        0        5        19↓
         0        0        0        0        7        13↓
         0        1        0        0        1        26↓
         0        1        0        0        3        34↓
```

図 4.3　*.inp ファイルの例

フローグラフのプログラムは，入力ユニット，出力ユニット，宣言文，演算ユニット，モジュールの 5 つからなる (図 4.4)．データはまず入力ユニットへ入力され，それから，各演算ユニットへと流れていく．そして，モジュール (ここで言うモジュールとは，演算ユニットの集合のことである) などを通り，最終的には，出力ユニットから出力される．宣言文は，プログラム中で#define 以下の右の文字列を左の文字列で代用すると言う意味を持つ．図 (4.4) 中で説明すると，プログラムの中では，30 という数値よりも，OUT という表現のほ

うがわかりやすいのである．

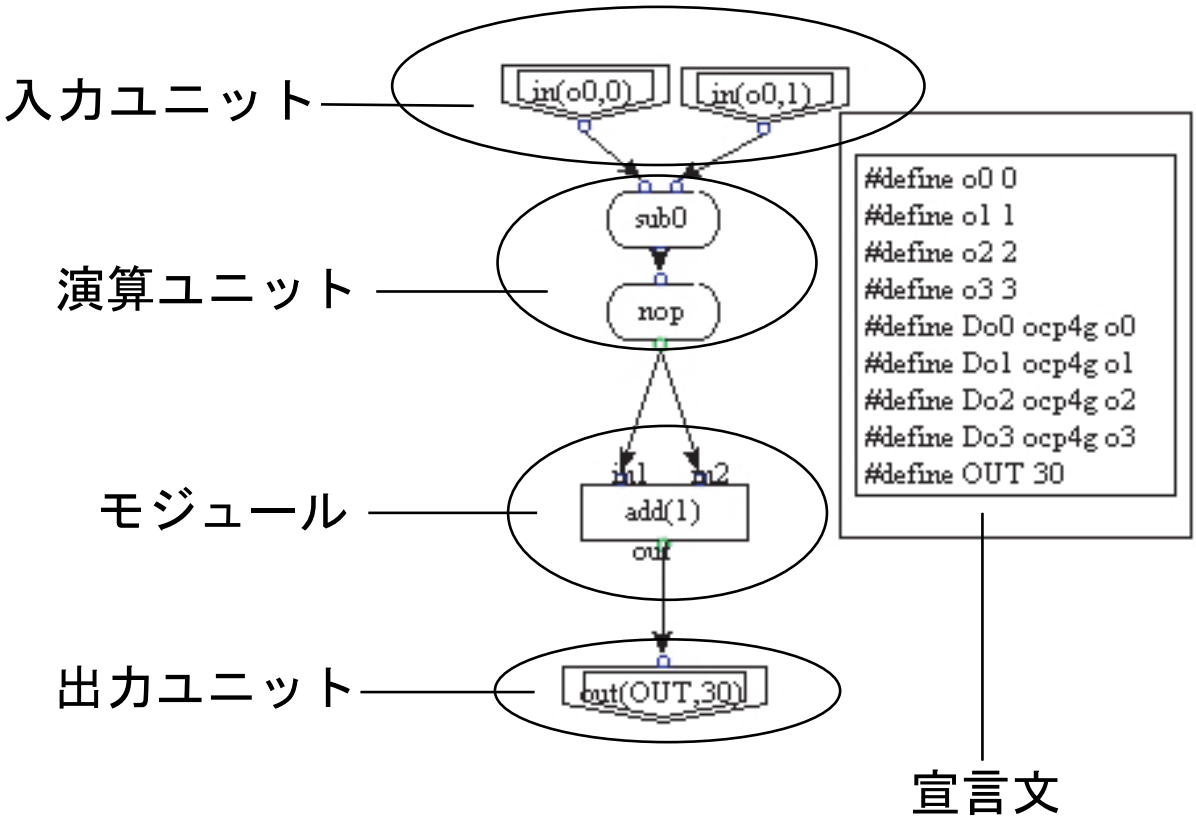


図 4.4　フローグラフの例

フローグラフエディタで実装していく場合に気をつけなければならないのは，データ駆動処理方式の特長によって，ある演算ユニットにおいて，流れてくる 2 つのデータの条件が一致したときに演算を開始すると言う点である．この条件は，演算ユニットによって様々であるが，パケットの持つ世代番号情報である，フィールド (FD)，ライン (LN)，ピクセル (PX) の 3 つが完全に一致しないことには，演算は行われない．また，フローグラフ中のそれぞれのユニットには，必ず一つの PE が割り当てられていなくてはならない．

4.3　プログラム

実際に実装された，DDMP のフローグラフがどのような構成になっているか，入力層のユニット数 6 個，競合層のユニット数 6 個のホスト有りプログラムと (図 4.5)，ホスト無し

4.3 プログラム

プログラム (図 4.6) の入力層分割 SOM のプログラムを例にとって説明していく．それぞれの図に見るように，入力部，集計部，更新部に分けることが出来る．

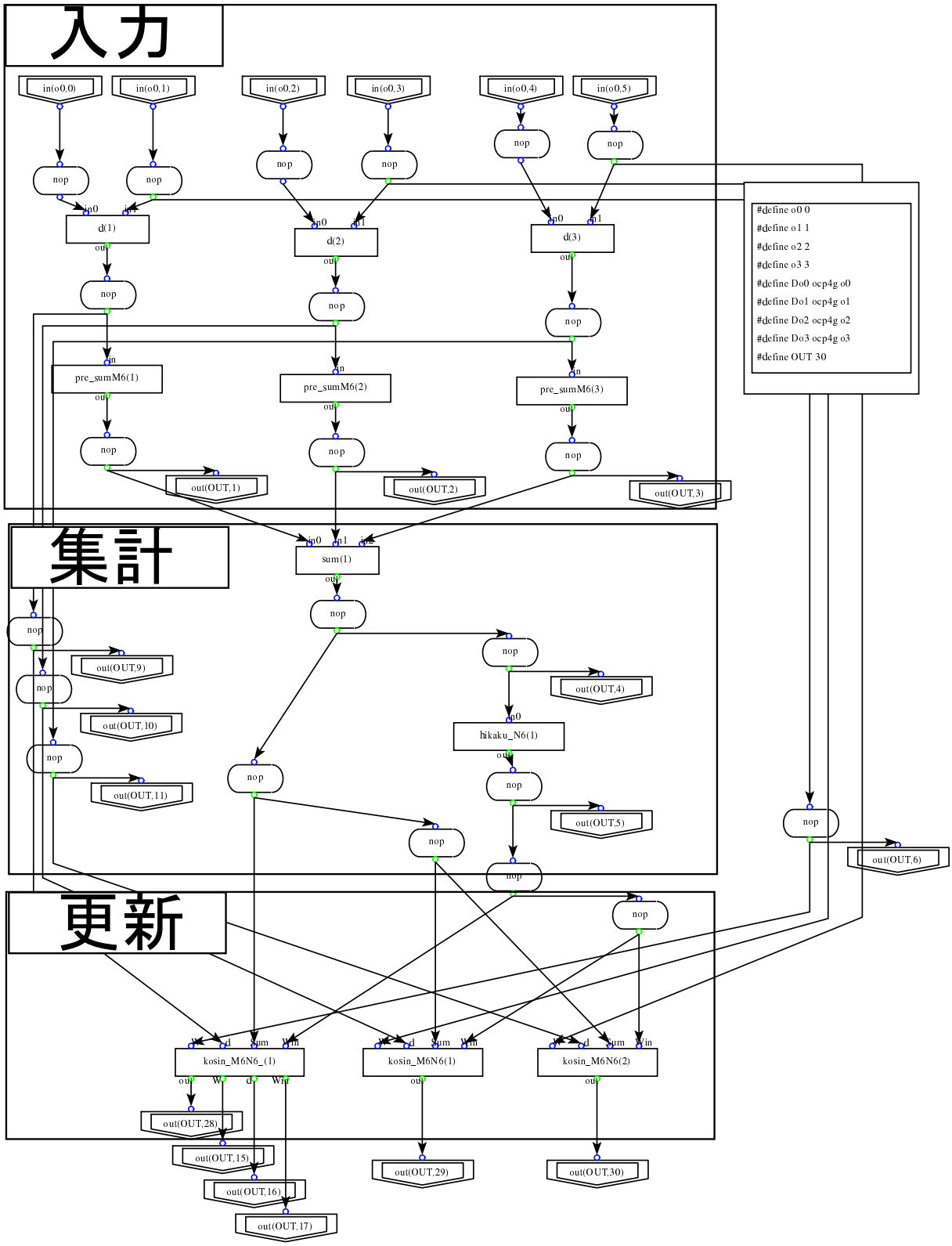


図 4.5　入力層及び競合層のユニット数 6 個，ホスト有りのプログラム

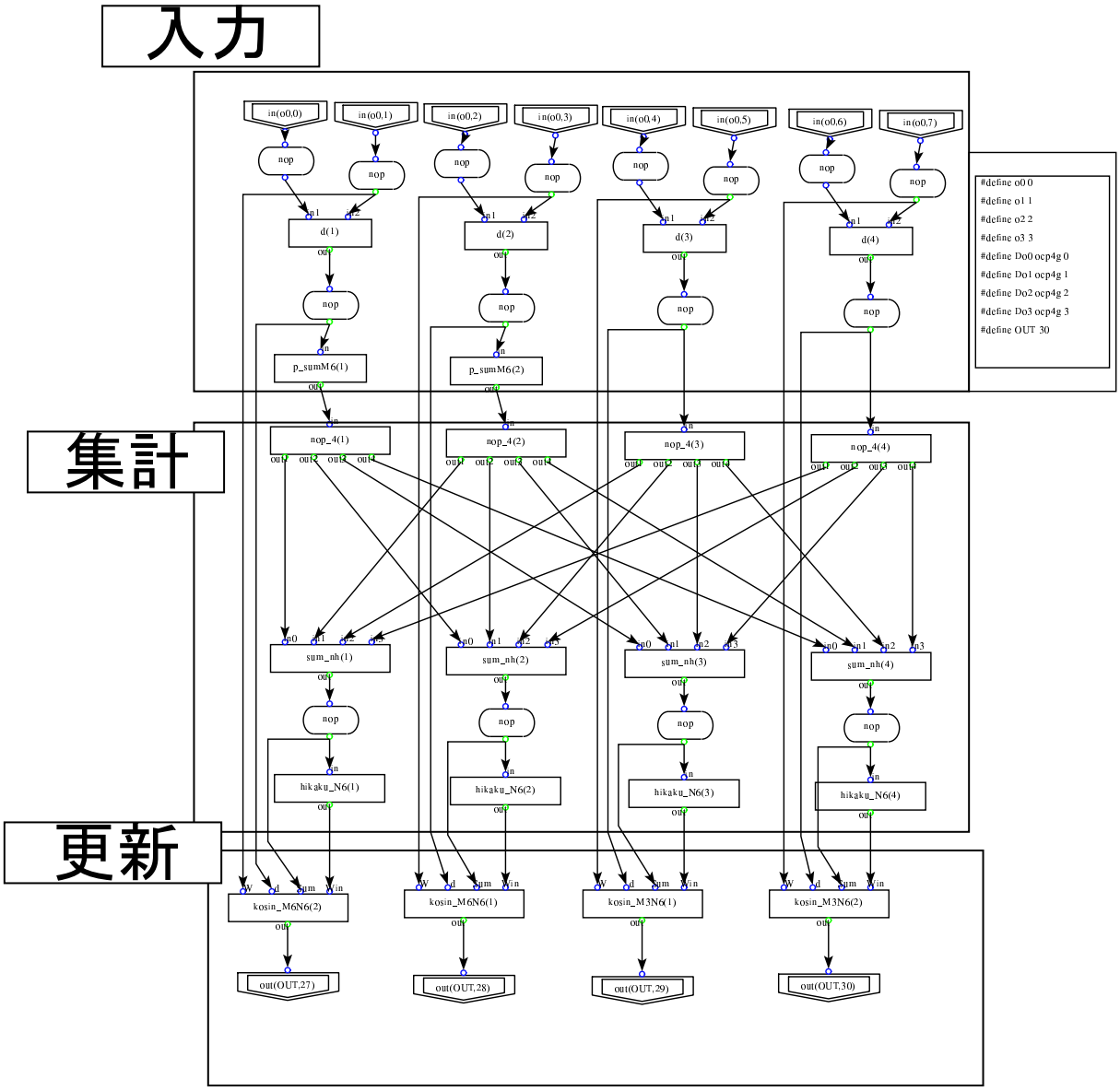


図 4.6　入力層及び競合層のユニット数 6 個，ホスト無しのプログラム

次にこのプログラムの各パーツの役割について説明していく．

4.3.1　入力ユニット

このユニット (図 4.7) は DDMP の内部プロセッサの特性上，全て PE0 に割り当てられている．また，エントリー番号の偶数には，SOM アルゴリズムにおける入力データが，奇数部分には，競合層の重みデータが入力される．

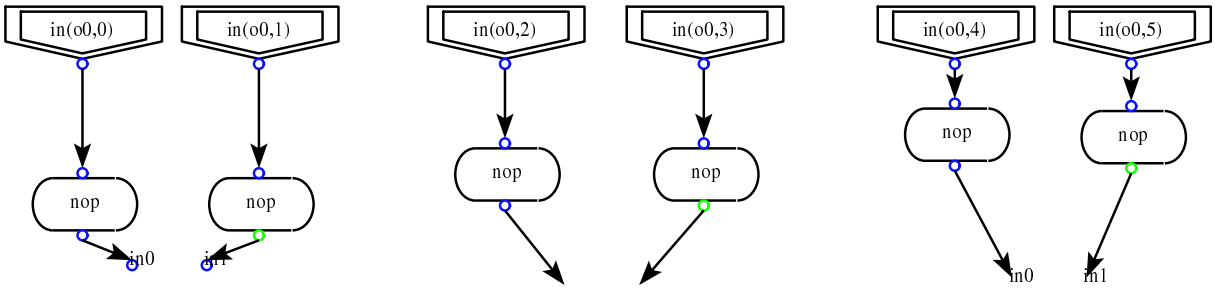


図 4.7 入力

4.3.2 誤差を取る

入力されたデータがまず初めに出会うモジュール d は，入力データと各競合層のユニットとの誤差の絶対値をとる．モジュールの内部構造はただの減算であるが，出力させたいデータは絶対値なので，減算結果が負になった場合は，-1 を乗じてやる．入力層の数が多い場合，PE によっては世代番号 LN がカウントされ，データが一巡多く流れる．これを次の処理である SUM に送る前に，あらかじめ集計しておこうと言うのが，モジュール pre_sum(p_sum) である．

4.3.3 集計，比較

次に先ほど求めた d を集計する．それが，モジュール sum の働きである．ただし，ホスト無しのアルゴリズムの場合，集計する前にモジュール nop_4 を使い，いったん他の PE にデータを分配してやる必要がある．誤差 d を集計した後は，勝利ユニットを決定するために比較を行う. このプログラムでは，競合層のユニットの識別を世代番号 PX を用いて行っている．hikaku.mod では，ループ操作を使い，PX での比較を可能にした．図 (4.10) にそれぞれのモジュールを示す．

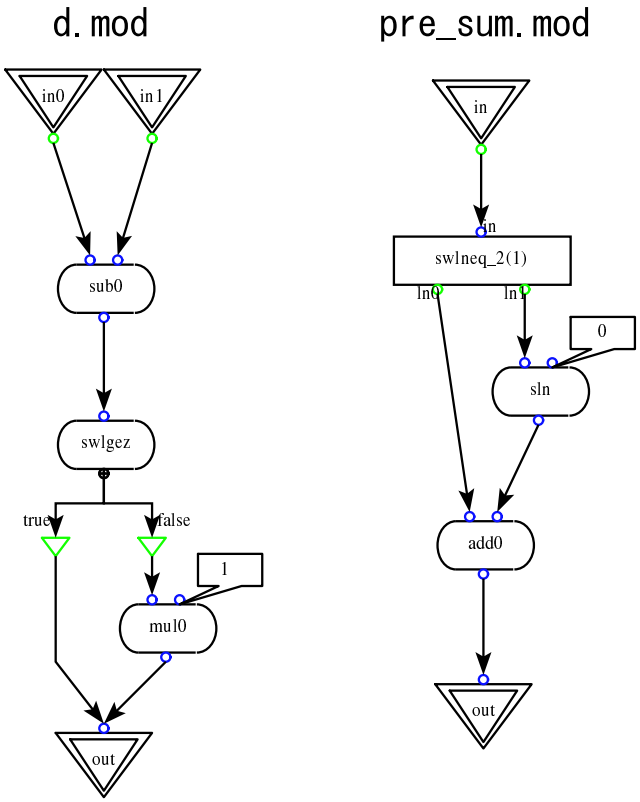


図 4.8 sum と pre_sum のモジュール

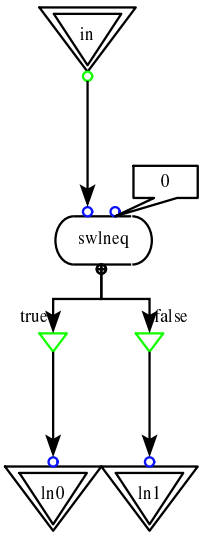


図 4.9 swlneq_2 のモジュール

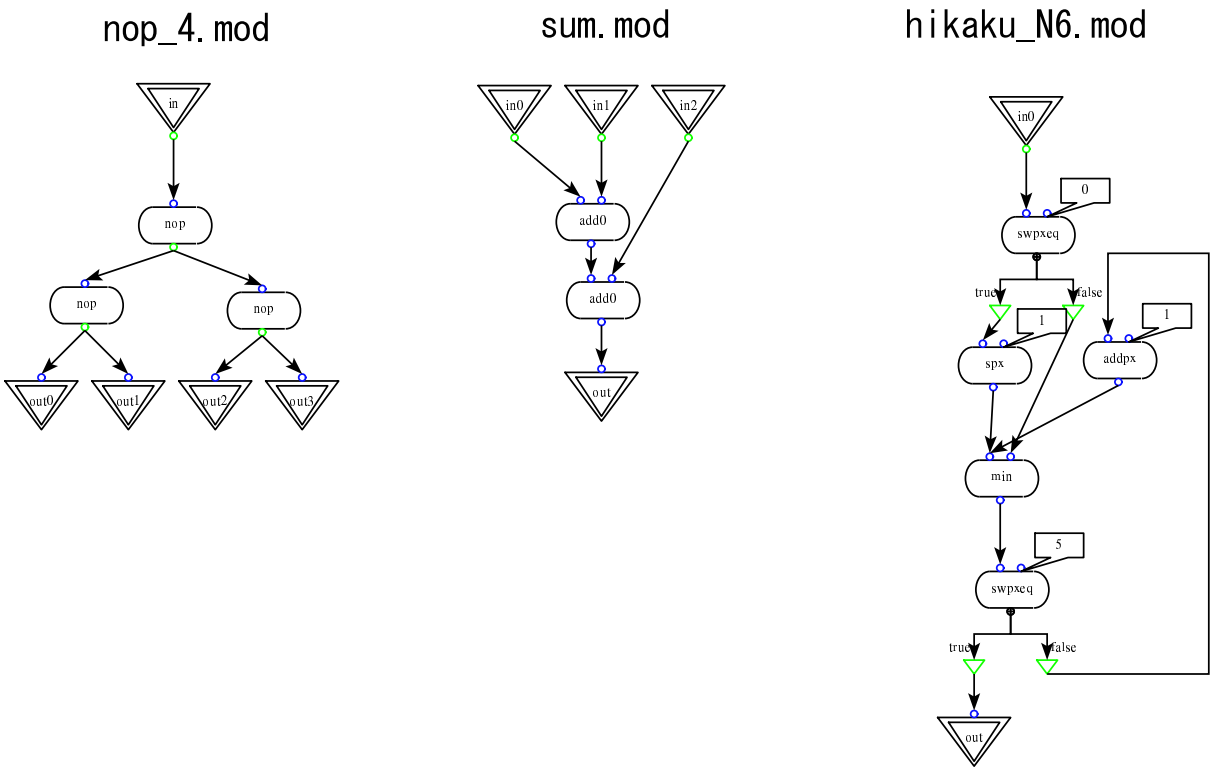


図 4.10　sum と hikaku，nop_4 のモジュール

4.3.4　更新

更新のモジュールは，比較で得た勝利ユニットのデータと，その PE で求められた誤差データ，誤差の集計データ，そして競合層の重みデータと 4 つのデータを使う (図 4.11)．モジュール hikaku でループ処理を行ってきたデータは，PX が競合層の数と同じになっているので，sum で得た合計データと比較するために，PX の値を更新する必要がある．sum の値と一致したデータは，そのまま更新作業に移ることが出来るが，それ以外のデータは，Absorb で廃棄される．本質的な更新はモジュール mulladd で行われる．競合層の重みの PX が勝利ユニットの PX と一致したならば，その競合層の重みは更新され，それ以外の競合層のデータは，そのまま出力される．

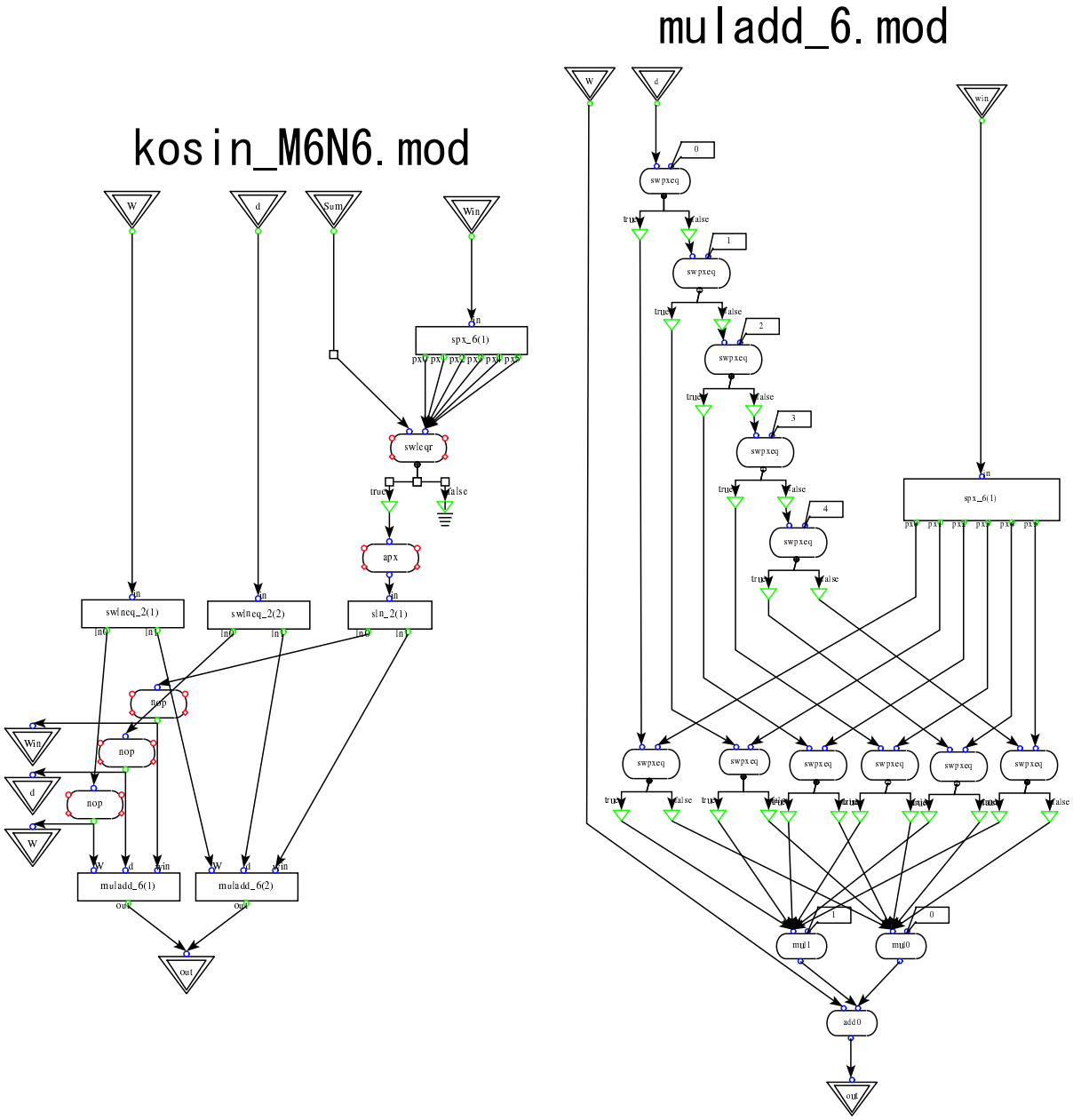


図 4.11　kosin と muladd のモジュール

4.3.5　シミュレート

フローグラフが出来上がったら，シミュレータ上で実行する．この時に表示されるサイクル数がこのプログラムの学習時間となる．図 (4.13) は，フローグラフシミュレータ (fsim_4g.exe) のスクリーンショットである．

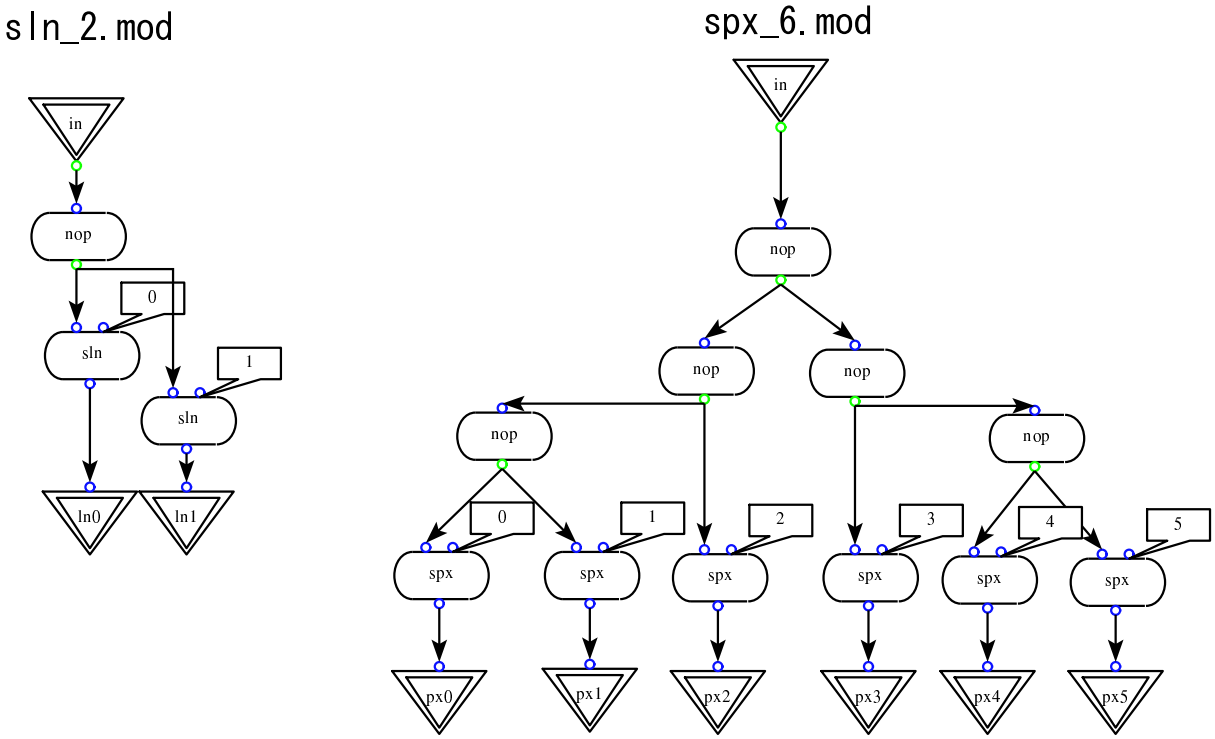


図 4.12 sln と spx のモジュール

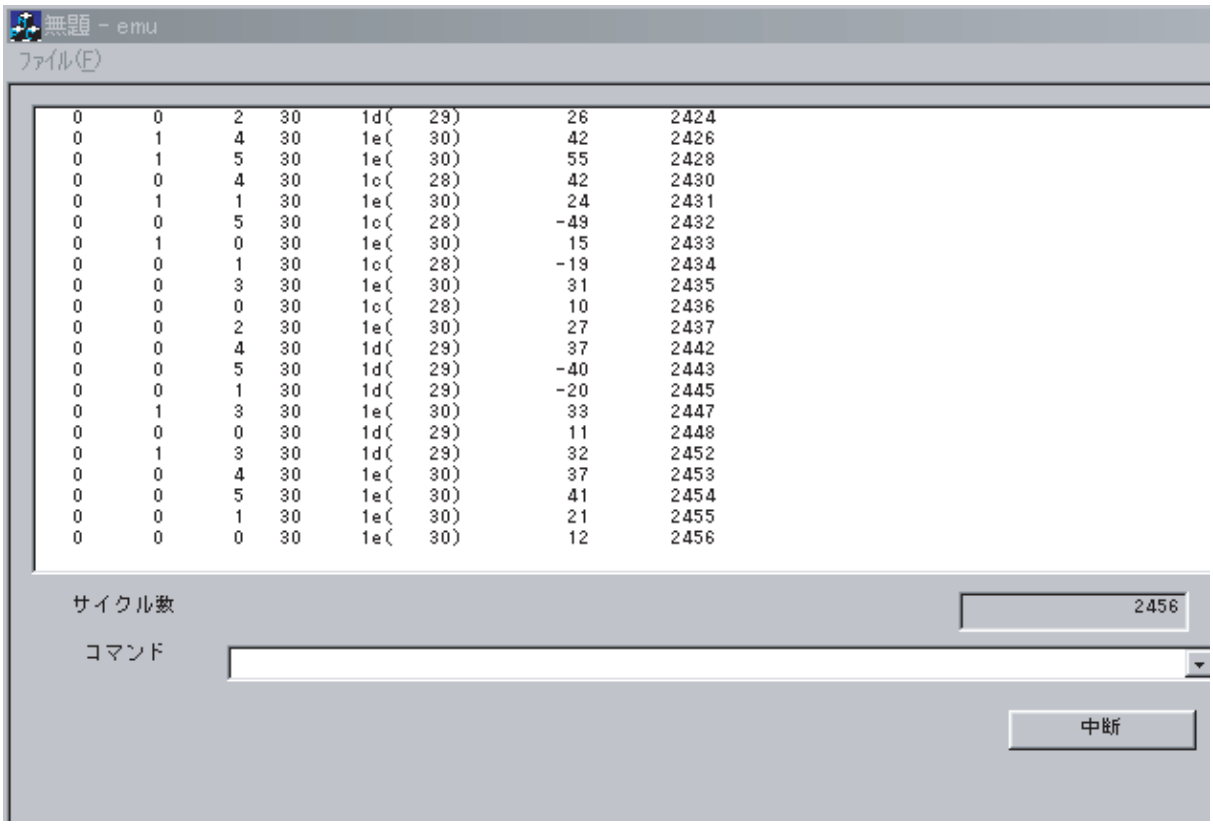


図 4.13　フローグラフシミュレータのスクリーンショット

第 5 章

結果

入力層，競合層，ともに 2,4,6 個の SOM を実装し，ホストの有無で比較したところ，下記のような結果が得られた．図 (??) は，実験した結果の学習時間の一覧である．これを，図

HOSTあり

競合層＼入力層	2	4	6
2	698	917	1041
4	953	1202	1434
6	1216	1520	2456

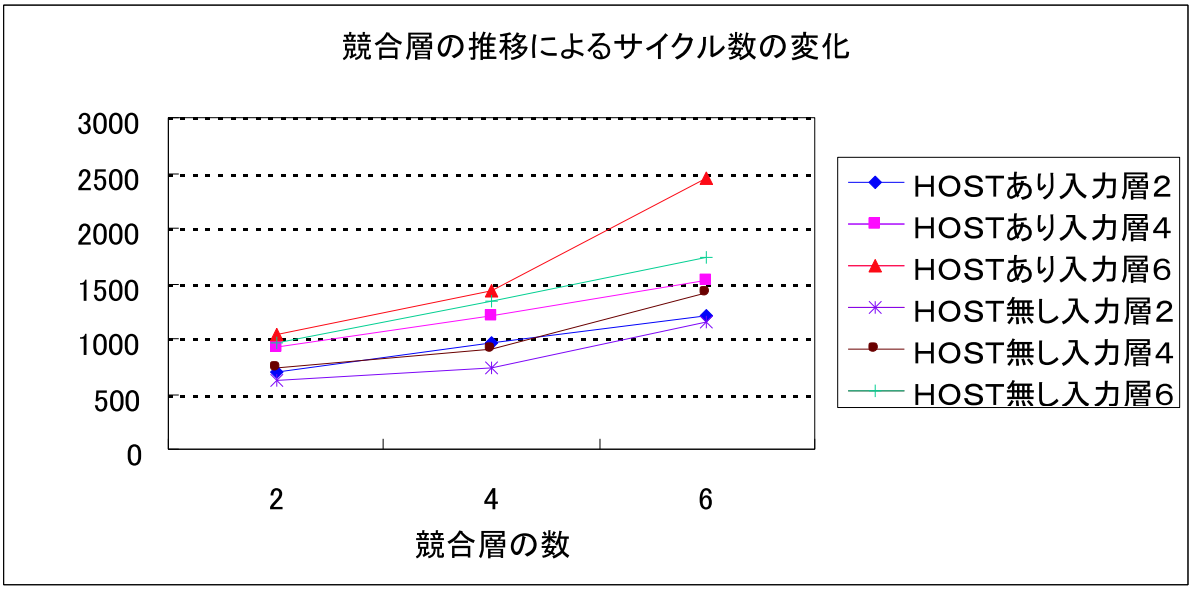
HOSTなし

競合層＼入力層	2	4	6
2	628	731	962
4	733	915	1345
6	1145	1408	1729

単位：サイクル

図 5.1　入力層分割 SOM 実装結果

(??) に競合層と入力層の推移で表す．全体的には，ホストなしのアルゴリズムのほうが学習時間は短い．これは，4 つのプロセッサを同時に使うことにより，ホストありのアルゴリズムより並列性が高くなることに由来すると考えられる．しかし競合層ユニットの数が 2，４個の場合，入力層の数が増えるにつれて，学習時間の長さは逆転する傾向にある．にもかかわらず，競合層のユニット数が 6 個の場合，入力層の数が増えるにつれて，ホストの有無で学習時間に大きな差が出てくる．ホストありのアルゴリズムの欠点は，一旦，ホストプロセッサに各プロセッサのデータを集計する点にある．その間，他の 3 個のプロセッサはアイドル状態になる．これに対して，ホストなしのアルゴリズムは，図 4 で見るように，一旦相



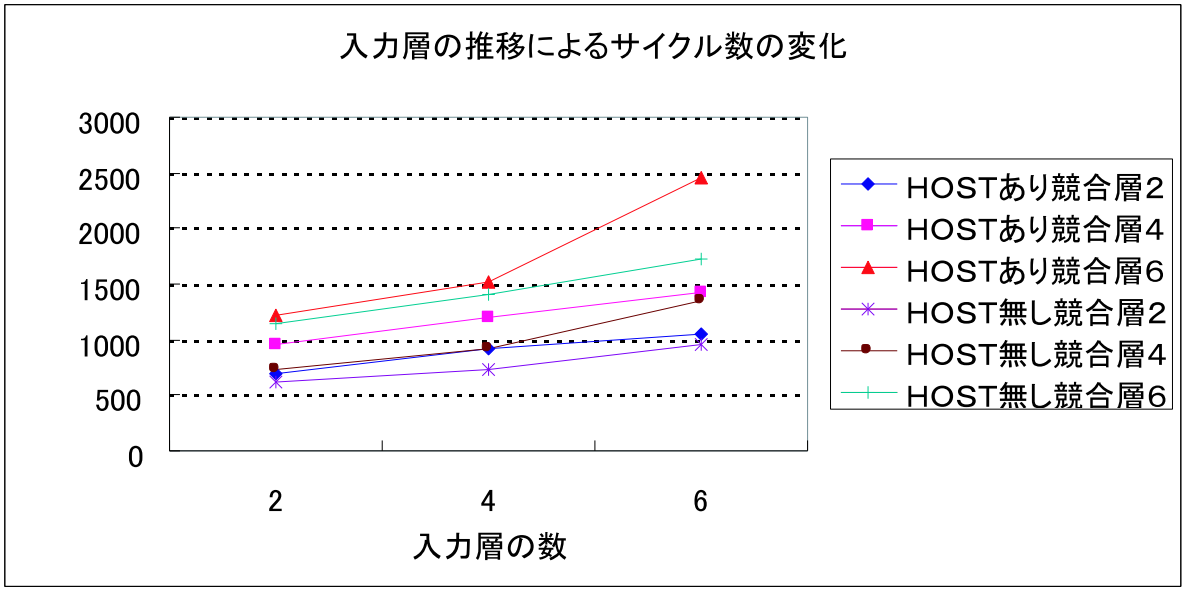


図 5.2　入力層の推移による DDMP の結果

互通信してデータのやり取りを終えてしまえば，あとは 4 個のプロセッサを同時に働かせることができる．このグラフより，入力層及び競合層のユニット数が少ないうちは，並列処理による学習時間短縮の効果は出ないが，入力層及び競合層のユニット数が多くなるにつれて，並列による高速化が期待できることがわかる．

第 6 章

まとめ

本研究では，各ＰＥの負荷が不均一にならない入力層分割法を用い，データ駆動処理型の DDMP で SOM を実装し，入力層，競合層，ともに 2,4,6 個の SOM の学習時間を評価した．その結果，全体的にホストなしのアルゴリズムの方の学習時間が短く，競合層，入力層のユニット数が多くなるほど並列処理による高速化が期待できることがわかった．しかし，今回の実験では DDMP の性質を理解しきれなかったために，競合層，入力層の数とも，非常に少ない数での研究となってしまった．PE の割り当てに関しても，どうも今の実装法では DDMP には，向いていない．今後の課題として，入力層と競合層の数を増やしての徹底的な実験と，DDMP に最も適した並列 SOM の実装法の提案があげられる．

謝辞

本研究を実施するにあたって，指導教員のラック先生には大変お世話になりました．また，DDMP のことで，色々と相談に乗ってくれた岩田先生，大森先生にも感謝の意を示したいと思います．その他，煮詰まったときに，いろいろとかまってくれた研究室のみんなにもありがとう♪

参考文献

[1] T. コホネン “自己組織化マップ”，シュプリンガー・フェアラーク東京 (1999)．

[2] 山森一人, 堀口進 “自己組織化ニューラルネットワークの並列学習シミュレーション” 北陸先端科学技術大学院大学 リサーチリポート IS-RR-97-0006, pp.1-16 (1997 Mar).

[3] 岩田誠, 宮田宗一, 寺田浩詔 “自己タイミング・スーパパイプライン型データ駆動プロセッサ” 電子情報通信学会論文誌 D-I Vol.J81, No.2, pp.62–69 (Feb. 1998).