TR－H－058

CAM-Brain シミュレータの高速化
プログラムの改良と CM-5 への移植

関　努
（豊橋技術科学大学）
邊見 均

1994. 2. 24

ATR 人間情報通信研究所

〒619-02 京都府相楽郡精華町光台 2-2　☎07749-5-1011

ATR Human Information Processing Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan

Telephone: +81-7749-5-1011
Facsimile: +81-7749-5-1008

CAM-Brain シミュレータの高速化

プログラムの改良と CM-5 への移植

関　努

(豊橋技術科学大学　知識情報工学課程)

邊見　均

(ATR 人間情報通信研究所　第 6 研究室)

Abstract

CAM-Brain は、Cellular Automata Machine(CAM) 上で、自ら成長し進化するニューラルネットを作ることを目的としたプロジェクトである。しかし CAM-Brain で使われる必要な状態遷移ルールはかなり多く、そのシミュレーションは、ワークステーション SPARC station 10 を用いても遅かった。

そこで、シミュレーションの高速化のために 3 種類のプログラムを作成した。まず、元のシミュレーションをプログラム本体に改良を加えた。次に改良を加えたプログラムを並列処理計算機である CM-5 に、MIMD プログラムとして移植を行った。このプログラムは、CAM ブレーンをいくつかに分割し、それぞれを並列に処理することによって高速化をはかった。その結果、シミュレーション速度は同 CPU を持つワークステーションの約 4 倍になった。これら 2 つのシミュレーションプログラムは、CAM-Brain の状態遷移ルールを開発時に使用するために作成した。3 つめに作成したプログラムは、将来状態遷移ルールが完成したときに、CAM-Brain を成長／進化させるためのものである。これは異なる遺伝子を並列に処理するプログラムであり、CAM-Brain が進化する速度が約 30 倍になった。

1　はじめに

CAM-Brain は、Cellular Automata Machine(CAM) 上で、遺伝的アルゴリズムを利用し、自ら成長し進化するニューラルネットを作ることを目的としたプロジェクトである。現在、このプロジェクトの動きのひとつとして、CAM-Brain の状態遷移ルールの開発を行っている。その開発のために CAM-Brain シミュレータが使われているが、このシミュレータは WS (SPARC station 10) を用いても遅く、開発の支障となっていた。

そこで、この CAM-Brain シミュレータの高速化をはかるために、シミュレーションプログラムを改良を加えた。また同時に汎用化のための改造をいくつか行った。その後、さらに高速化をはかるために並列処理計算機 CM-5 への移植を行った。またそれとは別に、ニューラルネットの成長 / 進化する状況を高速にシミュレーションするシミュレータを同計算機上で作成した。

2　CAM-Brain シミュレータの改良、改造

2.1　CAM-Brain シミュレータが遅い理由

CAM-Brain シミュレータの大きな動作は次の三つである。

1. 状態遷移ルールの適用における状態の更新
2. 更新後の状態の出力
3. 次世代への遺伝子の進化

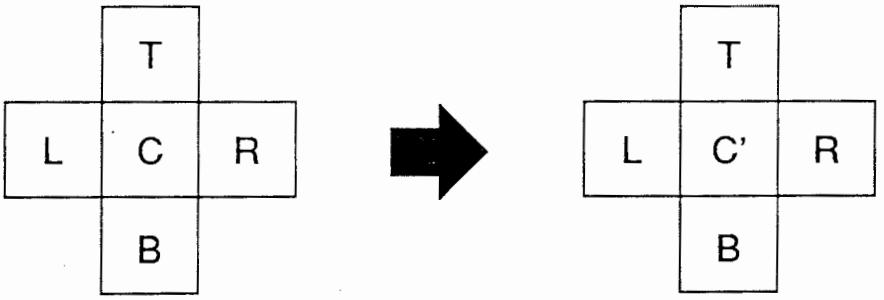


図 1: 状態遷移ルール

この中でシミュレータが遅くなるもっとも大きな原因は1である。その理由として次の二つが挙げられる。

a. 状態遷移ルールの多さ

b. 状態を示すセルマトリックス (プレーン) の大きさ

b.の理由は、2の処理が遅くなる理由でもあるが、これはやむを得ないことである。

ここで、簡単に a.について説明をする。状態遷移ルールとは図 1のようになっている。 C と C に隣接する TRBL の 5 つの状態によって、C から C' へ状態が変化することをこのルールは示している。もし仮に CTRBL のそれぞれの状態が 10 通りあるならばルール数は 10 万、 100 通りならば 100 億個となる。つまりこのような膨大なルールの中から、ひとつの適用すべきルールを捜し出して状態を更新する処理を、プレーン全体について行わなければならない。ここにシミュレータの遅さの最大の原因がある。

2.2 CAM-Brain シミュレータの変更点

ここで、大きな変更点を列挙する。

1. 改良点
   (a) 状態遷移ルールの最小化
   (b) プレーンに対する処理の最小化
2. 改造点
   (a) 必要な初期データのファイル化
   (b) 保存ファイルの一般化

1.a は、状態遷移ルール TRBL の並びが同じ (回転しても良い) であればルールは同じであることに着目した。つまり 4 つのうちのひとつだけテーブルに登録すればよいことになる。利点はルール数が 1/4 になることであり、欠点は登録するときも検索するときも TRBL の並び方を常に一定にしなければならないことである。この欠点は高速化の妨げになるが、そのことよりもむしろ将来性を考えるとこの方法を導入しないわけにはいかなかった。それは、記憶領域を減らすためであり、将来三次元の状態遷移ルールになると並び方が同じになるルールが 10 個になるからである。 1.b は、状態遷移ルールの性質上、更新が必ず行われない場所がプレーンに存在することから、更新される可能性がある場所のみに、状態遷移ルールの適用を行い、更新された場所のみを出力するように変更した。 2. は、拡張性を考慮して行った。

2.3 変更結果

いくつかの改良を加えてみたが、そのことによる高速化の成果は期待したほどには得られなかった。しかし、今後の拡張性の面においては充分に強化した。

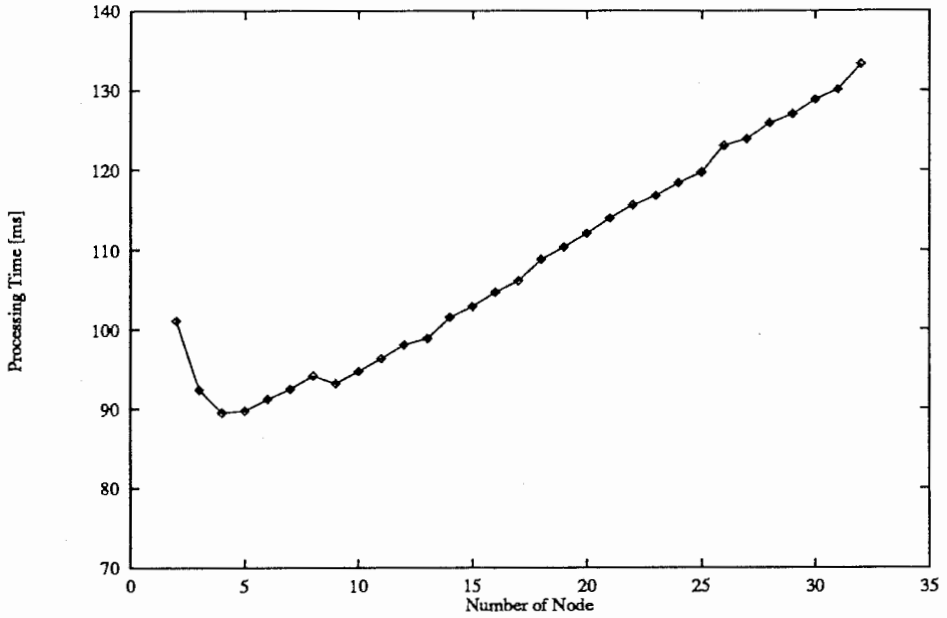


図 2: CM-5 版シミュレータの処理時間

3 並列処理計算機 CM-5 への移植

シミュレーションプログラムを改良したことによる高速化の成果が期待したほど得られなかったので、通常の WS では限界と考えた。理由は状態遷移ルールの適用と状態の出力というシミュレーションの大半をしめる処理は、常にどちらか一方しか行えないからである。そこでこの問題を解決すべく並列処理計算機 CM-5 への行った。ここで並列処理計算機 CM-5 について簡単に説明すると、この計算機は現時点で 32 個のプロセッサを同時に計算させることができる計算機である。この計算機上ならばより速くシミュレーションできると考えた。

3.1 移植の際の変更点

まず処理の分割について充分に考慮した。それは処理によっては分割できない、効果が少ない、または逆効果と考えられるからである。結論としては大きく分けて次の 2 点である。

- 状態遷移ルール適用の処理と出力処理
- ブレーンの分割

前者の理由は、前述のとおりである。後者は理由は、状態遷移ルール適用の処理は、出力処理よりもずっと処理が多いと考えたからである。つまりブレーンを分割することで出力処理の間に、状態遷移処理を終えることができることを目標とした。

3.2 移植結果

実際に移植を行った後、テストを行った。その結果を図2に示す。横軸がブレーンの分割数、つまり並列に計算させたプロセッサ (以後、ノード) の数である。もっとも速かったノード数 4 での処理時間は、各々のノードが持つ CPU と同じ CPU を持つ WS (SPARC station 2) でシミュレーションしたときの 1/4、SPARC station 10 でシミュレーションしたときの 1/2 となった。

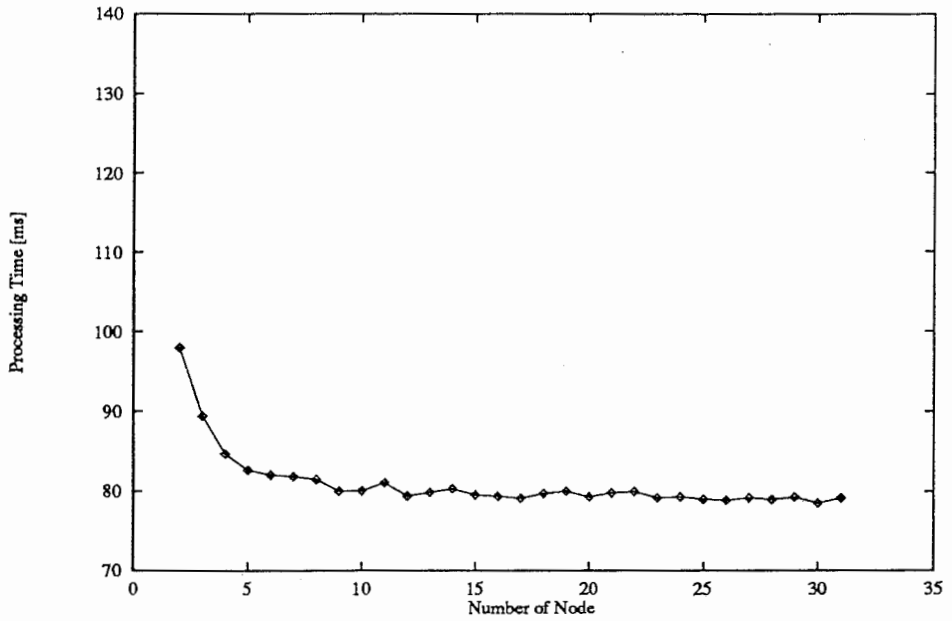


図 3: 改良型 CM-5 版シミュレータの処理時間

しかし、ノードをすべて使うともっとも時間がかかったのは予想外であった。これはノードの通信によるオーバーヘッドが大きくなったからだとわかった。

3.3 CM-5 版シミュレータの改良

この問題点は些細なことではないと考え、改良を加えた。変更点はデータ通信もできる限り出力処理時間の間に行うようにしたことである。具体的にはある特定のノードをデータ通信専用にしたことである。変更後、同様のテストを行った。結果は図 3である。このようにノード数 8 以上でほぼ安定した処理時間となり、わずかではあるがさらに処理時間が短縮した。(図 4参照)

4 CM-5 版遺伝子進化用シミュレータ

CM-5 にシミュレータを移植したことで表示を高速に行う目標は達成した。しかし並列処理計算機をより活かすならば、個々の遺伝子を別々に処理させてネットワークの進化を速くするほうがより適していると考えた。そこでシミュレータをそのように変更すると、遺伝子を進化させるシミュレーション速度は約 30 倍となった。また実際にシミュレーションを 2000 世代行った結果が、図 5である。この結果はニューラルネットが実際に進化することを示した、はじめての結果である。

5 今後の課題

- 現在は初期データが固定されている。初期データを可変型にすることができれば、汎用性はかなり向上し CAM-Brain 開発の大きな援助になると考えられる。
- シミュレーション速度は向上したとはいえ、まだまだかなりの時間を要することが実際に行ったシミュレーションからわかった。このことからある程度の処理時間を犠牲にしても、不測の事態に対する措置を取ったほうが良いと思った。

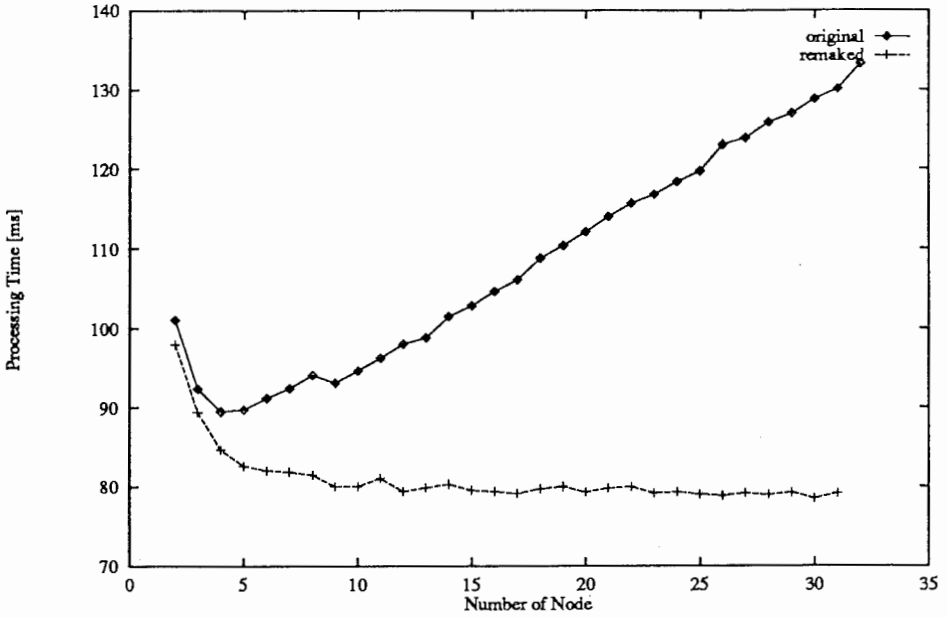


図 4: 改良前後の比較

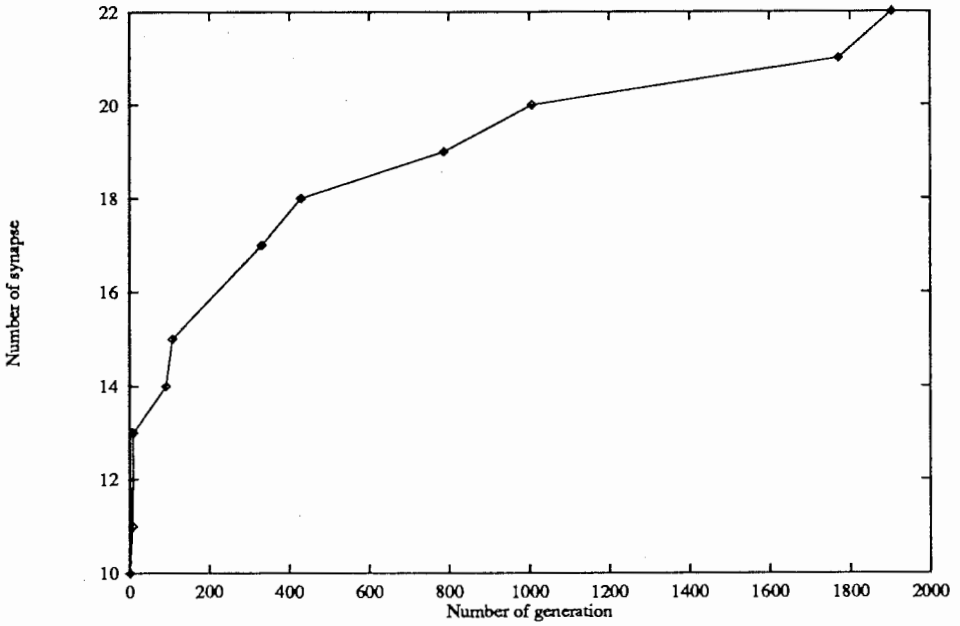


図 5: CAM-Brain シミュレーション結果

darwins について (Wed, 16 Feb 1994)

darwins には4つのプログラム(lhaで圧縮されています)があります。
それらは以下に示す4つです。

darwins/NdwV16.lzh
darwins/NdwV232.lzh
darwins/NdwV252.lzh
darwins/growth/NdwV3.lzh

それぞれのプログラムの実行ファイルは、

darwins/ndarwin
darwins/ndw_cm5
darwins/ndw_cm5V25
darwins/grow/ndw_grow

です。使い方などは、それぞれ別のドキュメントを用意していますので、それらを参照してください。

darwins/NdwV16.lzh -> darwins/doc/darwins.doc1.j
darwins/NdwV232.lzh -> darwins/doc/darwins.doc2.j
darwins/NdwV252.lzh -> darwins/doc/darwins.doc3.j
darwins/growth/NdwV3.lzh -> darwins/doc/darwins.doc4.j

なお、以上のプログラムの変更、改良などは自由に行ってくださってかまいません。ただ変更したことによる責任は負いかねます。

＊＊＊＊ 実行上の注意 ＊＊＊＊
この2点は重要です。実行前の確認と注意を忘れないでください。
1) 環境変数 DISPLAY の設定
(作者自身、このミスでかなりの迷惑を引き起こしました)
2) バックグラウンド処理ができません
(プログラムの構造上です。今後の方に期待します)

それぞれのプログラムについて、私が作ったのはCAM-Brainに関するところのみで、表示ルーチンは別の方に依存しています。表示に関することはその方にたずねてください。
いずれのプログラムも作者自身は完動すると思っていますが、もしかするとバグが残っているかもしれません。問題が起こった時は使用者の方にお任せします。
もしこれらのプログラムに関しての質問があれば下記にE-mailしてください。

豊橋技術科学大学 知識情報工学系 吉田研究室
(Human Interface Laboratory,
Course of Knowledge Based Information Engineering,
Toyohashi University of Technology)

関 努 (Tsutomu Seki)
(E-mail: seki@alice.tutkie.tut.ac.jp)

NdwV16.lzh (ndarwin) について

＜概要＞
ndarwin はもとになった darwin を大幅に修正を加えたものです。大きな変更点は次のとおりです。

1) 初期データの形式、ルールの定義の形式、セーブデータの形式が変わった。
2) メモリの使用方法が変わった。(それにともなって使用するメモリが大幅に減っている)
3) ファイルを分割している。

注意する点は、darwinで使っていたルール、セーブデータは使えないことです。それ以外の点(プログラムの流れなど)は、変更していません。が、たぶんdarwinよりは、速くなっていると思います。Xの使えるマシン(BSDのみ)なら大抵動くと思います(開発はSPARC station 2で行いました)。SPARC station 10 くらいがあれば、それほど苦痛なく使えると思います。

＜実行方法＞
まずカレントにndarwinと、以下のファイルをおいてください。

Colorfile
growthdata
initdata
my_rules_r
populations
rgb.txt

これらのファイルは、darwins/Ndw_needed.lzhで圧縮されています。lhaで展開してください。それから、そのカレント上でndarwinを実行すればＯＫです。引数は何も受け付けていない(はず)です。
あとは、画面のメッセージに従ってください。

(ファイルの説明)

Colorfile
rgb.txt

この2つのファイルは、カラーパレットを設定しています。この2つを変更しますと、作者の手にも負いかねます。そのまま使ってください。

growthdata

このファイルの中は、次の様になってます。

ex)
2.0 0.6 0.02

これらの値は、前から順にスケーリング値、遺伝子をクロスさせる確率、遺伝子が突然変異を起こす確率です。(float型であることに注意してください)
さらに詳しいことについては原作者であるHugo de GARIS さんに尋ねてください。

initdata

このファイルは、デフォルトにおける初期データです。(2次元です)
アスキーコードですから見れば想像がつくと思います。

populations

このファイルの中は、次の様になってます。

ex)
10 1

前の値は、1世代の(遺伝子の)人数です。

後の値は、次の世代に優秀な遺伝子を何人残すかを示しています。

  my_rules_r

これが状態遷移ルールです。中は次のようになっています。

ex)
12 3 12 12 12 12
78 77 3 75 2 81
        .
        .
        .

これは、1行のデータが次のように並んでいます。

<C> <T> <R> <B> <L> <Cnew>

これは、　　<T>　　の状態の時、<C> が <Cnew> に遷移することを
　　<L> <C> <R>
　　<B>
示すルールです。
(これも、詳しいことは、Hugo de GARISさんに尋ねてください)

それぞれの数字や文字の区切りは、スペース(またはそれに準ずるもの)にしてください。(つまりfscanf()使ったとき、区切れるコードです)

<実行上の注意(darwinとの相違点)>
  状態遷移ルールのファイルは上記したフォーマットのものを使ってください。
また、セーブファイルのフォーマットも変わりましたから、注意してください。
セーブファイルのフォーマットの概要は

<1世代の人数> <次世代に残される人数>
<遺伝子のデータ>
<次世代に残される遺伝子のデータ>
<実行した進化世代数> <より優秀な遺伝子が見つかった時の世代の数>
<より優秀な遺伝子が見つかった時の世代ナンバー>
<より優秀な遺伝子が見つかった時の評価値>

になってます。実際のセーブファイルをみてもらい、プログラムのセーブルーチンを見るとすぐにわかると思います。

また環境変数 DISPLAY の確認を忘れないでください。

<コンパイルの仕方>
  コンパイルの前に次のことを確認してください。

1)/home/gauss/degaris/Xdemos2 に 以下のファイルがあるか。
  classhnt.c color.c connect.c gc.c oval.c sizehint.c
  visual.c window.cw mhints.c wmname.c
2)カレントに以下のファイルがあるか。
  chap4.c privatecolormap.c
  (なければ、Ndw_forX.lzh を展開してください)

上の2点が大丈夫であれば、カレントに NdwV16.lzh を展開してください。
(コマンドライン上で、lha x NdwV16.lzh を実行すればＯＫです)
そのあと、そのカレント上で、makeを実行すれば、ndarwinが作られます。

<プログラムについて>
  NdwV16.lzhを展開すると、darwin/の下にソースができます。以下にこれらのファイルの関数単位での繋がりとそれぞれの関数、使用している構造体について説明します。(明らかに説明不用と思われるものは省略します)

(関数の相関図)

```
main -+- set_populations
      +- darwin_malloc
      +- get_rules        --- insert_hashtable
      +- (chap4main)
      +- genalg           -+- set_growthdata
      |                     +- initpopln
      |                     +- set_initfilename
      |                     +- popqual          -+- converttape
      |                     |                     +- loadcells
      |                     |                     +- showcells
      |                     |                     +- cells_update
      |                     |                     +- findstates       -+- set_target
      |                     |                     |                     +- set_cell
      |                     |                     |                     +- get_cell
      |                     |                     +- showcells_update
      |                     |                     +- findfitness
      |                     |                     +- savecells
      |                     +- scale
      |                     +- repro
      |                     +- unifcross
      |                     +- mutate
      |                     +- elite
      +- lookcells
      +- darwin_free
```

(関数の略説)

set_populations	: 1世代の遺伝子の人数の設定と次世代に残す人数の設定
darwin_malloc	: 必要なメモリの確保
get_rules	: 状態遷移ルールのハッシュテーブルを作成する
+insert_hashtable	: ある1つのルールをテーブルに挿入する
(chap4main)	: (表示用のウィンドウを用意する)
genalg	: CAM-Brainシミュレーション本体
+set_growthdata	: 必要な設定値の設定
+initpopln	: 遺伝子の初期化
+set_initfilename	: 初期状態を示すファイルの設定
+popqual	: ひとりの遺伝子をシミュレーションする本体
+converttape	: 遺伝子をCAM-Brain用にデータ変換を行う
+loadcells	: 初期状態をファイルから読み込み設定する
+showcells	: フレーム(セル全体を示す)をウィンドウに表示する
+cells_update	: 遺伝子を順番に取り出す
+findstates	: フレームを1状態成長させる
+set_target	: 1組のセルのセットを取り出す (<実行方法>の my_rulrs_r のところを参照)
+set_cell	: 1組のセルのセットを1状態更新する
+get_cell	: 1つのセルの値を取り出す
+showcells_update	: showcells と同じ(内部処理が少し異なる)
+findfitness	: 最終状態を評価する
+savecells	: フレームの状態を保存する
+scale	: 遺伝子全員の最終状態の評価値をスケーリングする
+repro	: 次世代の移る遺伝子の前処理
+unifcross	: 遺伝子をクロスさせる
+mutate	: 遺伝子の突然変異
+elite	: 優秀な遺伝子を次世代に残す
lookcells	: savecells で保存したデータを表示する
darwin_free	: 使用したメモリを開放する

注) scale repro unifcross mutate elite は CAM-Brain の内部にかかわることです。詳細は、原作者に尋ねてください。

(構造体などの仕様)

1)注意すべき変数の型について

・POSN 型

この変数は、次に示す値とそれらのORのみをしめします。

CENTER	: 中央
TOP	: CENTER の対して上

```
  RIGHT      : CENTER の対して右
  BOTTOM     : CENTER の対して下
  LEFT       : CENTER の対して左
  NOCHANGE   : 上記5つに属さない
(実際には、CENTER から順に LEFT まで1ビットずつ割り当てられてます)

2)使用している構造体について
・フレームを構成するセル
typedef struct cel {
    CHROM status;          : セルの状態
    POSN upd;              : セルに対するフラグ(ルールを適用するのに使用)
} Cel;

・フレーム更新のとき、更新される可能性のあるセルの位置
(フレームは左上角を(0,0)とし、右にx軸、下にy軸の＋方向としている)
typedef struct chcel {
    int x, y;              : フレームでの位置
} Chcel;

・フレーム
typedef struct frame {
    Cel **cell;            : フレームを構成する2次元セルマトリックス
    Chcel *chcell;         : 更新時の更新の可能性がある位置
    int pos;               : chcell の個数
} Frame;

・プログラムレベルでのフレーム全体を構成する構造体
typedef struct cels {
    Frame frm[2];          : フレーム(2面を交互に使用している)
    STAGE s;               : 現在、処理対象となっているフレーム番号
    Chcel *chd;            : 更新されたセルの位置(表示するのに使用)
    int chd_p;             : chd の個数
} Cels;

・状態遷移ルールを構成する共用体
typedef union neighbor {
    struct {
        CELL_T T, R, B, L;  : 中央のセルに対する近傍
        CELL_T C;           : 中央のセル
    } Sepa;                 : (separation の意味です)
    struct {
        NBR N;              : 中央のセルに対する近傍のひとまとめ
        CELL_T C;           : 中央のセル
    } Conb;                 : (これは、combinationを勘違いしました)
} Neighbor;

・状態遷移ルールのハッシュテーブルを構成する構造体
typedef struct lnktbl {
  int next;             : 次に検索すべきルールの位置
                          (MAXHASH ならば、そこは空であることを示す)
  Neighbor Nbr;         : 一組のセルの繋がり
  CELL_T Cnew;          : 中央のセルの次の状態
} Lnktbl;

・遺伝子の評価を示す構造体
typedef struct quality {
    float q;            : 遺伝子の評価値
    int num;            : 遺伝子の番号
} Quality;

・1世代全員の遺伝子を示す構造体
typedef struct chromos {
    STAGE stg;              : 現在、処理対象となる遺伝子
    CELL_T **data[2],       : 1世代全員の遺伝子 (2面を交互に使用)
           **elt;           : 次世代に残る遺伝子
```

```
    CELL_T **tape;          : 遺伝子を処理用に展開する領域
    int fitgen[FITSIZE];    : 過去に優秀な遺伝子を残すことになった世代
    float fitval[FITSIZE];  : 上の時のその評価値
    Quality *qual;          : 遺伝子全員の評価値
    int pop,                : 1世代の遺伝子の人数
        elites,             : 次世代に残される遺伝子の人数
        leng,               : 1人の遺伝子の長さ
        nrgen,              : 処理を開始した世代
        fitp;               : fitgen(fitval) の個数
} Chromos;
```

・セル更新用の構造体
(この構造体の仕組みに関しては、CAM-Brainの問題に関するもので、正確なこと、詳しいことは書けません。原作者に尋ねてください)

```
typedef struct positions {
    FLAG flag1,             : 重要な信号があったか否か？
         type1;             : Axon か Dendrite か？
    POSN sig1,              : 1つめの重要な信号の位置
         sig2,              : 2つめの重要な信号の位置
         ax,                : Axon の位置
         den,               : Dendrite の位置
         V;                 : 移動すべき信号の元の位置
} Positions;
```

＜補足＞
状態遷移ルールのハッシュテーブルについて

上記した状態遷移ルールは、次の法則があります。

```
<C> <T> <R> <B> <L> <Cnew>
<C> <R> <B> <L> <T> <Cnew>
<C> <B> <L> <T> <R> <Cnew>
<C> <L> <T> <R> <B> <Cnew>
```

これらは、すべて同じものです。もう少しわかりやすく示しますと

```
    <T>             <R>             <B>             <L>
<L> <C> <R> == <T> <C> <B> == <R> <C> <L> == <B> <C> <T>
    <B>             <L>             <T>             <R>
```

このように、あるルールに対して、隣接するものは並び方(便宜上時計周りとします)が変わらなければルールとしては同じわけです。原作者のプログラムではひとつのルールに対して上のように4つテーブルに登録していましたが、いくつかの点から、本プログラムからひとつしか登録しないようにしました。そこで、<T> <R> <B> <L> の並び方をひとつの数字と見なし最小になる並び方だけ登録する方法を採用しました。このため当然、検索するときにも <T> <R> <B> <L> の並び方を前もって最小にしておかねばなりません。(今回はその部分をマクロにしてます)
処理速度が低下するにもかかわらず、このような方法を採用した理由は将来性からです。現時点では2次元ですからルールは見かけ上4倍にしかなりません。しかし将来的には3次元に拡張することを考えますと10倍になるからです。つまり、

```
<C> <N> <E> <S> <W> <U> <D> <Cnew>
<C> <E> <S> <W> <N> <U> <D> <Cnew>
<C> <S> <W> <N> <E> <U> <D> <Cnew>
<C> <W> <N> <E> <S> <U> <D> <Cnew>
<C> <N> <D> <S> <U> <E> <W> <Cnew>
<C> <N> <W> <S> <E> <D> <U> <Cnew>
<C> <N> <U> <S> <D> <W> <E> <Cnew>
<C> <D> <E> <U> <W> <N> <S> <Cnew>
<C> <S> <E> <N> <W> <D> <U> <Cnew>
<C> <U> <E> <D> <W> <S> <N> <Cnew>
```

これだけ、同じルールが出てくるわけです。わかりやすく並べますと、

```
<U> <N>         <U> <E>         <U> <S>         <U> <W>
<W> <C> <E> == <N> <C> <S> == <E> <C> <W> == <S> <C> <N>
    <S> <D>         <W> <D>         <N> <D>         <E> <D>

               <E> <N>         <D> <N>         <W> <N>
           == <U> <C> <D> == <E> <C> <W> == <D> <C> <U>
                   <S> <W>         <S> <U>         <S> <E>

               <N> <D>         <D> <S>         <S> <U>
           == <W> <C> <E> == <W> <C> <E> == <W> <C> <E>
                   <U> <S>         <N> <U>         <D> <N>
```

と、なるわけです。この場合、ルールを適用するときの前処理に時間がかかるので全ルールを登録したほうがいいかも知れませんが、私には予想ができませんから、メモリ領域が少なくてすむほうを採用しました。(また全てのルールを登録するようにプログラムを変更するのは容易ですし)

以上、NdwV16.lzh (ndarwin) について
Written by Tsutomu Seki (18. Feb. 1994)

NdwV232.lzh (ndw_cm5) について

＜概要＞
これは、ndarwin(NdwV16.lzh)を並列処理計算機CM-5に移植したものです。CAM-Brainシミュレータが遅くならざるを得ないもっとも大きな原因は、状態遷移ルールの多さです。この問題を解決する方法として考案したのが、フレームをいくつかに分割し並列に処理を行うことによる処理時間の短縮です。実際のプログラムではフレームを表示する計算機とフレームを更新する計算機を別にし、またフレームを分割し、それぞれを別の計算機で処理するようになってます。この結果、フレームの分割数を考慮して実行すると約40%の処理時間の短縮に成功しました。

＜実行方法＞
カレントに、ndw_cm5と以下のファイルをおいてください。

```
Colorfile
growthdata
initdata
my_rules_r
populations
rgb.txt
```

これらのファイルについては、NdwV16.lzh の方で説明してますから省略します。次に計算機をCM-5のパーティション・マネージャー(ここでは"mozart"もしくは"brahms")に移って、カレントをndw_cm5のあるところにしてください。あとは、

```
% ndw_cm5 [-node number]
```

で、実行できます。number は、フレームの分割数(使用するノード数)を示します。たとえば

```
% ndw_cm5 -node 10
```

ならば、フレームを10個に分割して演算を行います。
オプションを付けなければ、デフォルトの分割数(現在は5)で行います。

＜実行上の注意＞
実行は、かならずパーティション・マネージャーで行ってください。
また環境変数 DISPLAY の確認を忘れないでください。

＜コンパイルの仕方＞
まず、計算機を ＩＯコントロール・プロセッサ (ここでは"ligeti") もしくは、ダイアグノスティック・プロセッサ(ここでは"webern")に移ってください。あとは、ndarwinの場合と同じで、NdwV16.lzh の代わりに NdwV232.lzh を展開して make すればＯＫです。

＜プログラムについて＞
NdwV232.lzh を展開すると、NdwV16.lzh の場合と同じようにdarwin/の下にソースができます。ただしNdwV16.lzh の場合とは違い、ファイルが *.cp.c と *.pn.c の2種類あります。これは前者が、ホストプログラム(本プログラムではデータの収集やフレームの表示を行う)で、後者がノードプログラム(本プログラムでは分割したそれぞれのフレームを計算する)です。詳細はCM-5のマニュアル(CMMD User's Guide Version 3.0 など)を見てください。
ここでは、プログラムの関数単位での相関図と ndarwin との相違点(変更点)について書いておきます。

(関数の相関図)
・ホストプログラム

```
main -+- set_populations
      +- darwin_malloc
      +- get_rules        --- insert_hashtable
```

```
      +- (chap4main)
      +- genalg          -+- set_growthdata
      |                    +- initpopln
      |                    +- set_initfilename
      |                    +- popqual          -+- loadcells        --- sendcells
      |                    |                    +- showcells
      |                    |                    +- findstates
      |                    |                    +- showcells_update
      |                    |                    +- findfitness
      |                    |                    +- savecells
      |                    +- repro
      |                    +- unifcross
      |                    +- mutate
      |                    +- elite
      +- lookcells
      +- darwin_free

・ノードプログラム
main -+- darwin_malloc
      +- converttape
      +- cells_update
      +- local_findstates       -+- set_target
      |                          +- set_cell
      |                          +- get_cell
      +- darwin_free
```

(ndarwinと異なる関数の略説)

・ホストプログラム

```
main          : ルールのテーブルをノードに送る
+genalg
 +popqual     : 遺伝子をノードに送る
  +loadcells
   +sendcells : 初期フレームデータをノードに送る
  +findstates : 各ノードから更新したフレームデータを集め、フレーム全体
                を更新する
```

・ノードプログラム

```
main             : ルール、遺伝子、初期フレームデータをホストから受け取る
                   更新したフレームデータをホストへ送る
                   フレームデータの更新前後に隣会うフレームのノードと
                   データを送り合う
local_findstates : 各々のノードが持つフレームを更新する
```

＜補足＞

このプログラムは高速化の為にノードを半分以上スリープさせます。本来ノードをいくつ使ってもデータの通信量は変わらないのですが、ホストとノードがコネクションするときのオーバーヘッドが実際の処理時間に影響します。この問題はある種の最適化問題で、ノードを少なくし過ぎると、最初の問題がそのまま残ります。何回かテストした結果、現在の段階では、ノード数を4もしくは5にしたときがほぼ最速でした。この理由からデフォルトのノード数を5にしています。

しかし32個も使えるノードの5/6は使っていないわけですから、かなりもったいない使い方と思います。そこでこれを解決したものが、NdwV252.lzh(ndw_cm5V25) であり、さらに5%から10%の高速化に成功しています。

しかしこのバージョンを残したのは理由があります。それは、このプログラムの方が数段わかりやすいからです。理由は、NdwV252.lzh(ndw_cm5V25)のドキュメントを読んでもらえばわかると思います。また実際にNdwV3.lzhはこのバージョンから作成しました。

ですから、実際は、ndw_cm5V25(NdwV252.lzh)を使い、プログラムの変更、改良には、こちらを使うことをお勧めします。

以上、NdwV232.lzh (ndw_cm5) について

Written by Tsutomu Seki (18. Feb. 1994)

NdwV252.lzh (ndw_cm5V25) について

＜概要＞
ndarwin を CM-5 に移植した ndw_cm5 は高速化に成功しましたが、別の問題が出てきました。それは分割数を増やす、つまりノード数を増やすとホストとノードとの間に無視できないオーバーヘッドが生じることです。これはノード数を減らす(ノード数を最適化する)ことである程度解決できますが、将来性を考えると、なんらかの対応策を講じたがよいと思い、さらに改良を加えたものがこの ndw_cm5V25 (NdwV252.lzh) です。
改良点は、ノードが余る(つまり、使わない)のであれば、1つくらい別の目的で使用してもかまわない、という発想から、ホストと各ノードのデータの通信を直接行うのではなく、あるノードにバッファの役割をさせたことです。
具体的には、ノード番号0がこの役割をし、ホストがデータを受け取るのはここからのみで、各々のフレームを処理しているノードは、結果を0番に送るわけです。(プログラム的にホストが2つになったと考えてください)
こうすることで、ノード数がある程度以上(現時点では5以上)であれば処理時間は安定する、つまりノードが増えてもオーバーヘッドは増加しないようにすることに成功しました。また全体として処理時間は5%から10%短縮しました。

＜実行方法＞
ndw_cm5 と同じです。そちらを参照してください。

＜実行上の注意＞
これも ndw_cm5 と同じです。かならずパーティション・マネージャーで実行してください。また、環境変数 DISPLAY の確認を忘れないでください。

＜コンパイルの仕方＞
NdwV232.lzh を NdwV252.lzh に差し替えて行ってください。手順は NdwV232.lzh と同じです。

＜プログラムについて＞
NdwV232.lzh との相違点を中心に示しておきます。

(関数の相関図)
・ホストプログラム

```
main -+- set_populations
      +- darwin_malloc
      +- get_rules        --- insert_hashtable
      +- (chap4main)
      +- genalg          -+- set_growthdata
      |                   +- initpopln
      |                   +- set_initfilename
      |                   +- popqual          -+- loadcells        --- sendcells
      |                   |                    +- showcells
      |                   |                    +- findstates
      |                   |                    +- showcells_update
      |                   |                    +- findfitness
      |                   |                    +- savecells
      |                   +- repro
      |                   +- unifcross
      |                   +- mutate
      |                   +- elite
      +- lookcells
      +- darwin_free
```

・ノードプログラム(1)

```
main -+- darwin_malloc
      +- converttape
      +- cells_update
      +- local_findstates       -+- set_target
      |                          +- set_cell
```

```
      |                          +- get_cell
      +- darwin_free
```

・ノードプログラム(2)
main --- subhost

(ndw_cm5 と異なる関数の略説)
・ホストプログラム
main
+genalg
 +popqual
  +findstates : subhost(ノード0)から更新したフレームデータを集める。

・ノードプログラム(1)
main   : 更新したフレームデータは、subhost(ノード0)に送る

・ノードプログラム(2)
main
+subhost : それぞれのフレームを更新してるノードから更新データを集め、
           一まとめにしてからホストプログラムに送る。

以上、NdwV252.lzh (ndw_cm5V25) について
       Written by Tsutomu Seki (21. Feb. 1994)

NdwV3.lzh (ndw_grow) について

＜概要＞
ndarwin (ndw_cm5 あるいは ndw_cm5V25) は、状態遷移ルールを開発するために、ひとつの遺伝子において高速処理、高速表示を目的としました。しかしある程度、状態遷移ルールの開発が終われば、CAM-Brain シミュレーションの成長/進化のプロセスが速くなる方が好ましくなると考えました。つまり、遺伝子の世代がより速く進化するほうが役立つと考えたわけです。
そこでプログラムを、各々のノードがひとつの遺伝子を処理するようにしました。その結果、世代の進化する速度は、以前のシミュレータの約30倍(推定)になりました。

＜実行方法＞
必要なファイルなどは ndw_cm5 と同じですが、いくつかのオプションを用意しています。

% ndw_grow [-nonstop] [-best] [-nonshow]

-nonstop : 途中の中断を一切なくします。
(遺伝子の保存は最後のみ、フレーム状態の保存はできません)
-best : 以前よりもよい評価を得た遺伝子を発見したときの世代数とその評価値を起動画面上に表示します。
-nonshow : 遺伝子が成長/進化する過程を表示しません。
(このオプションは、-nonstop -best を包括しています)

＜実行上の注意＞
ndw_cm5 と同じくパーティション・マネージャーで実行してください。また、環境変数 DISPLAY の確認を忘れないでください。

＜コンパイルの仕方＞
NdwV232.lzh を NdwV3.lzh に置き換えて行ってください。手順は NdwV232.lzh と同じです。

＜プログラムについて＞
プログラムは、NdwV232.lzh と同じくホストプログラムとノードプログラムがあります。関数単位での相関図と mdarwin (ndw_cm5) と異なっている関数を示します。

(関数の相関図)
・ホストプログラム

```
main -+- set_populations
      +- darwin_malloc
      +- get_rules        --- insert_hashtable
      +- (chap4main)
      +- genalg           -+- set_growthdata
      |                    +- initpopln
      |                    +- set_initfilename
      |                    +- popqual          -+- loadcells        --- sendcells
      |                    |                    +- showcells
      |                    |                    +- findstates
      |                    |                    +- showcells_update
      |                    |                    +- savecells
      |                    +- repro
      |                    +- unifcross
      |                    +- mutate
      |                    +- elite
      +- lookcells
      +- darwin_free
```

・ノードプログラム

```
main -+- darwin_malloc
```

```
      +- local_popqual        -+- local_findstates -+- cells_update
      |                        |                    +- set_target
      |                        |                    +- set_cell
      |                        |                    +- get_cell
      |                        +- findfitness
      +- local_popqual_node0  -+- local_findstates -+- cells_update
      |                        |                    +- set_target
      |                        |                    +- set_cell_node0
      |                        |                    +- get_cell
      |                        +- findfitness
      +- converttape
      +- darwin_free
```

(関数の略説)
・ホストプログラム
main
+genalg
 +popqual : 各々の遺伝子の評価は各々のノードで行う
  各々のノードから受け取るのはその評価値のみ
  +loadcells
   +sendcells : すべてのノードに同じ初期データを送る
  +findstates : 1つ前の世代での最良遺伝子による更新データだけ受け取る
  (ノード0で処理されるものが前世代での最良遺伝子であり、
   ノード0だけが、ndw_cm5 と同様にホストに更新データを送る)
・ノードプログラム

main	: フレームの隣合うノードとのデータの受渡しを行わなくなった。(各々のノードが別々の遺伝子を処理するため)
+local_popqual	: 遺伝子シミュレーション本体(ノード0以外)
+local_findstates	: フレームを1状態更新する
+findfitness	: 最終状態を評価する
+local_popqual_node0	: 遺伝子シミュレーション本体(ノード0) 更新したフレームデータをホストに送る
+local_findstates	: 上の local_findstates と同上
+set_cell_node0	: set_cell の処理とホストに送るべきデータの更新を行う。
+findfitness	: 上の findfitness と同上

＜今後の課題＞
今回、時間の都合上できなかったことを、思い付く限り示しておきます。今後のCAM-Brainシミュレータの改良の参考にしてください。

1) バックグラウンド可能型
stdin での入力をコマンドラインからの引数にする、ファイルから読み出す(コマンドラインでファイル名を示す)、デフォルトを持たせるなどの処理を加えることで可能になると思います。

2) 初期データを可変にする
現在は、初期データ、遺伝子の出現数、出現位置など固定です。(部分的には、プログラムがそうなっています)これを可変にすることで汎用性があがると思います。(これは、結構面倒な作業になると思います)

3) ルールの追加を随時出来るようにする
これは、プログラムの構造上それほど難しくはないと思います。ただ作者自身が追加したいルールの作成方法がわからなかったのでこのルーチンは今回のプログラムでは割愛しました。

さらに細かいことはいろいろとあるかも知れませんが大まかにはこのくらいです。作者は、表示部分には関与してませんから何もいう権利はありませんが、少し話を聞いた限りでは、パレット数を減らすと、現時点で少々見づらいことが直るそうです。

最後になりますが、今回作成した4つのプログラムは、基本的に他の人に見て

もらってもわかるようになるよう努力しました。これ今後のCAM-Brain(シミュレータ)の発展に役に立てれば幸いです。

以上、NdwV3.lzh (ndw_grow) について
Written by Tsutomu Seki (21. Feb. 1994)

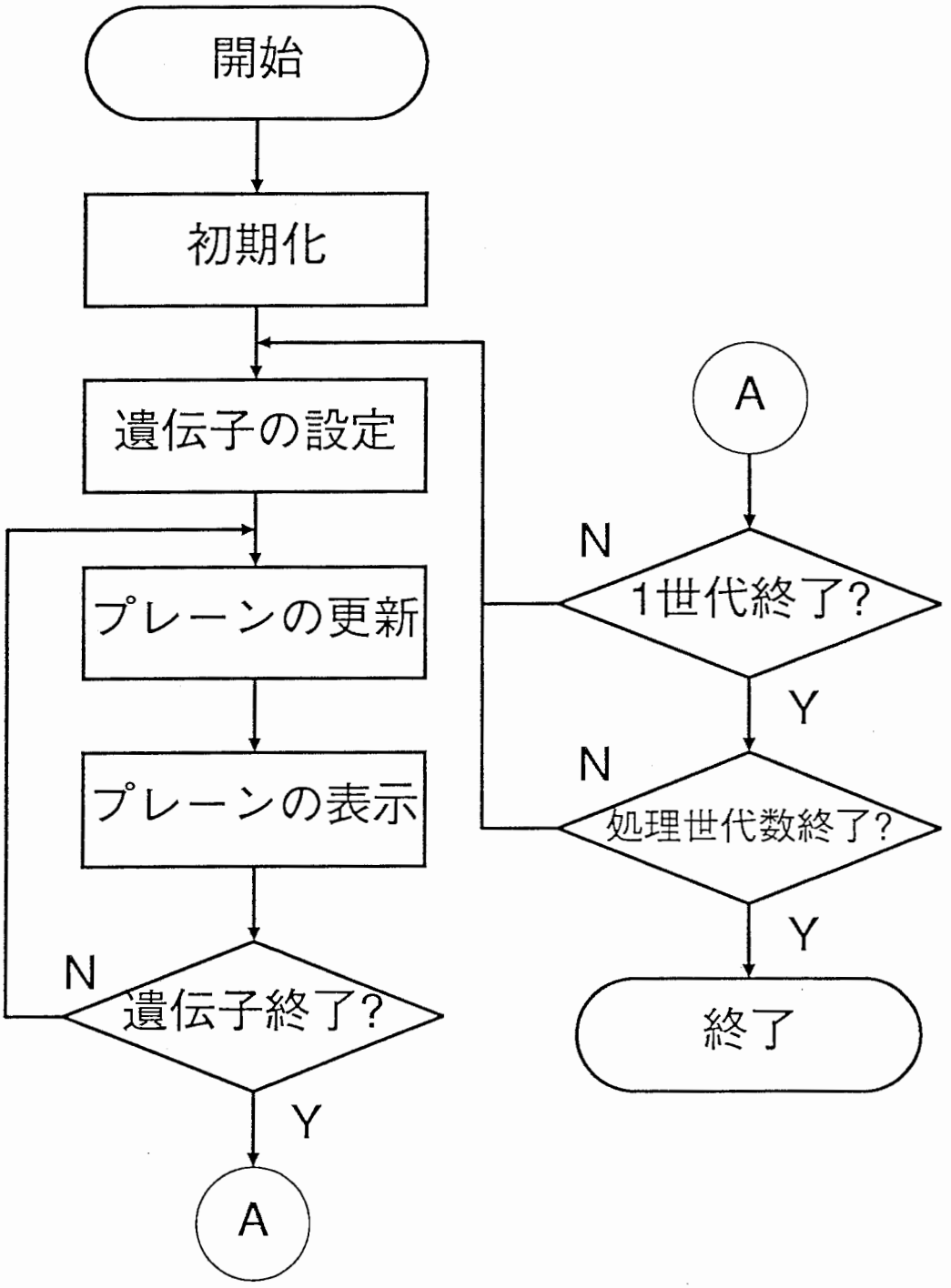
ＣＡＭ－Ｂｒａｉｎ
シミュレータの高速化

プログラムの改良とＣＭ－５への移植

第六研究室　実習生

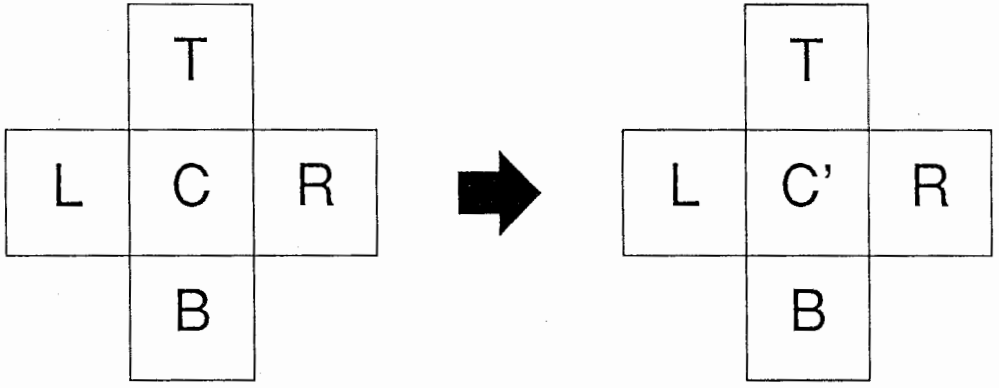
関　努

CAM-Brainシミュレータの流れ図



CAM-Brainシミュレータが遅い理由

1) 状態遷移ルールの多さ



状態遷移ルールの個数 ＝ 状態数の5乗

2) プレーンの大きさ

CAM-Brainシミュレータの改良点

1) 状態遷移ルールの最小化

2) プレーンに対する処理、操作の最小化

3) カラーパレットのテーブル化

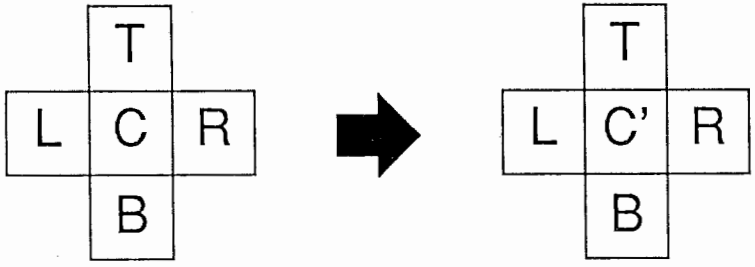
CAM-Brainシミュレータの改造点

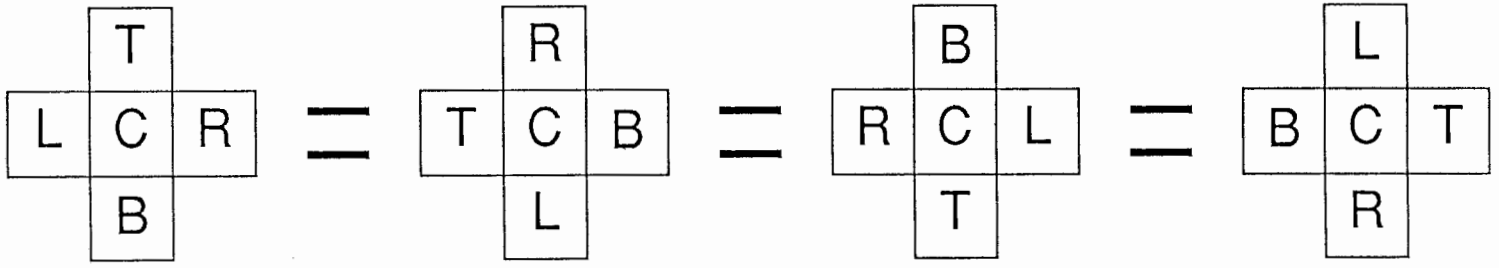
1) 初期データのファイル化

2) 必要設定値のファイル化

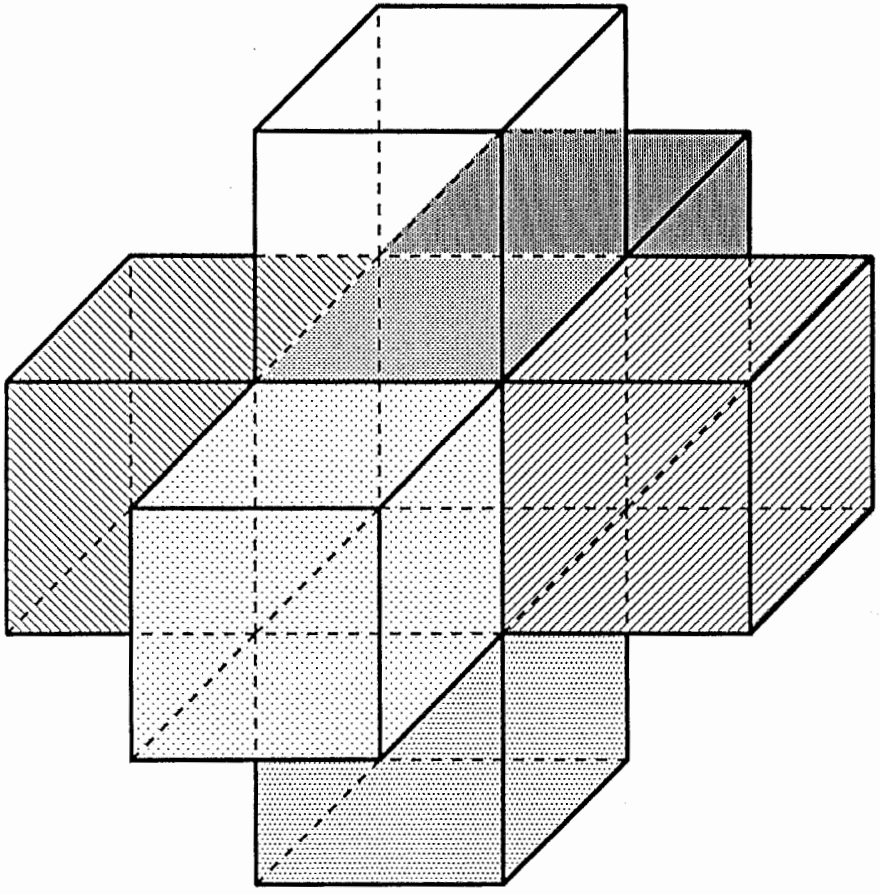
3) 保存ファイルの変更

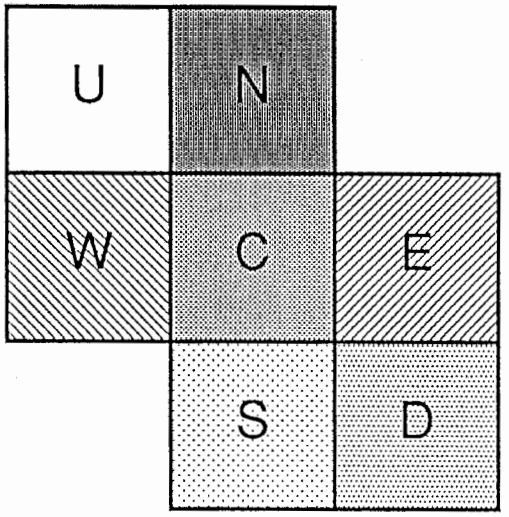
状態遷移ルール(二次元)



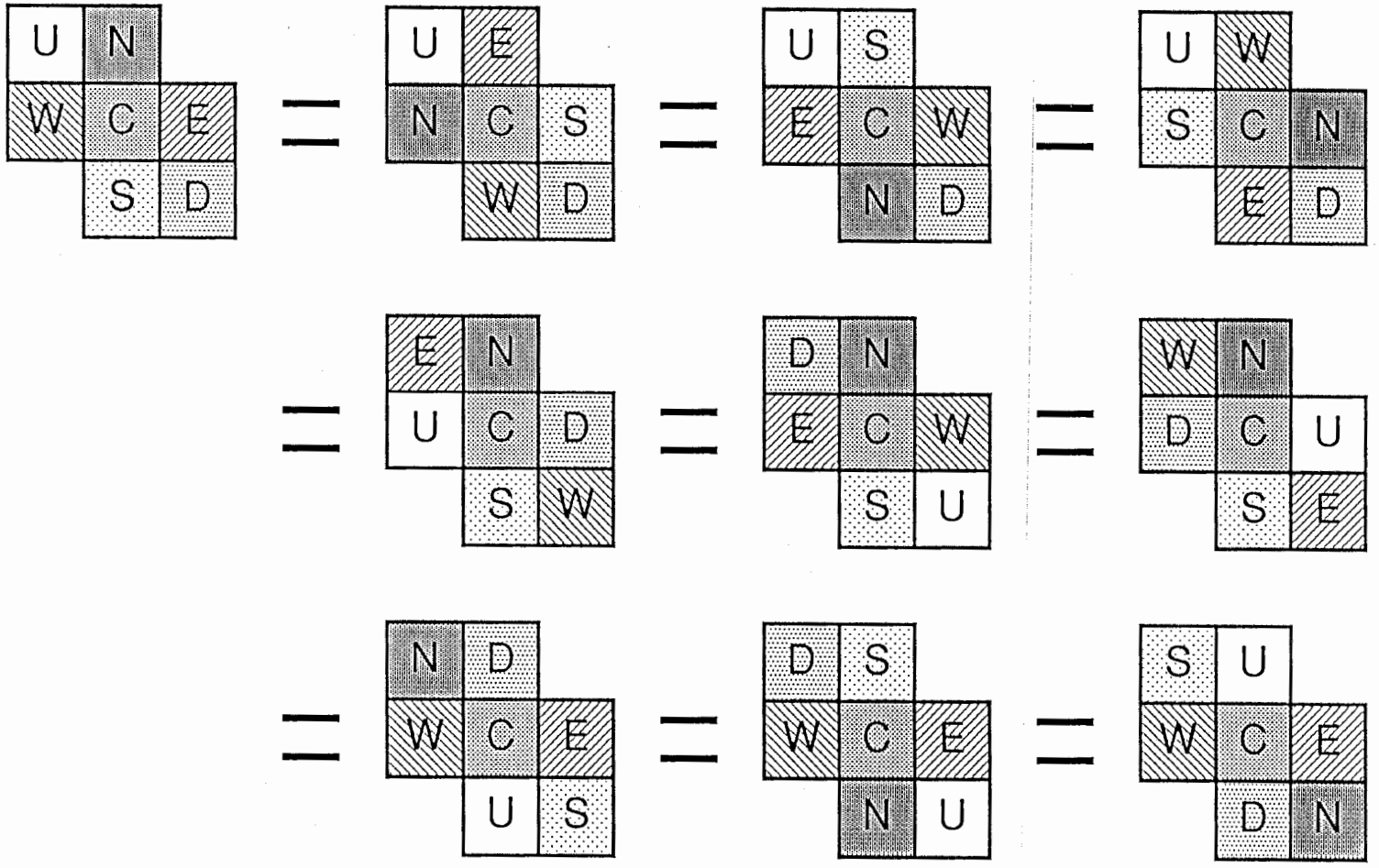


状態遷移ルール(三次元)





状態遷移ルール(三次元) (つづき)



CM-5への移植

プログラムの改良により
ある程度の高速化に成功

⬇

しかし、通常のWSでは限界がある

理由) シミュレーションの主な動きは2つあり、
(状態遷移ルールの適用と画面の表示)
これらの処理は同時に行えない

⬇

もし2つが同時に行うことができれば
処理時間は短縮できる

⬇

並列処理計算機ならば同時に行える

CM-5への移植 (つづき)

CM-5への移植による変更点

1) ホスト/ノード プログラムに分離

・ホスト プログラム

状態遷移ルールテーブルの作成

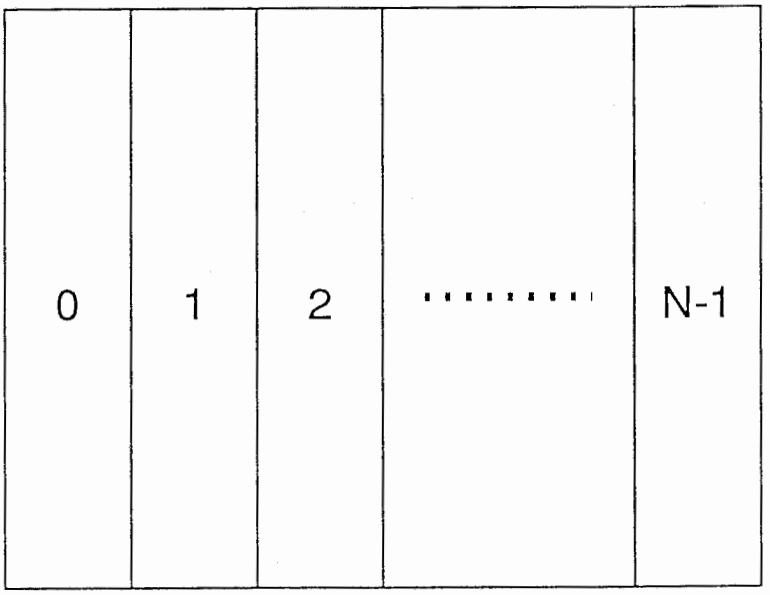
プレーンの表示

遺伝子の次世代への進化

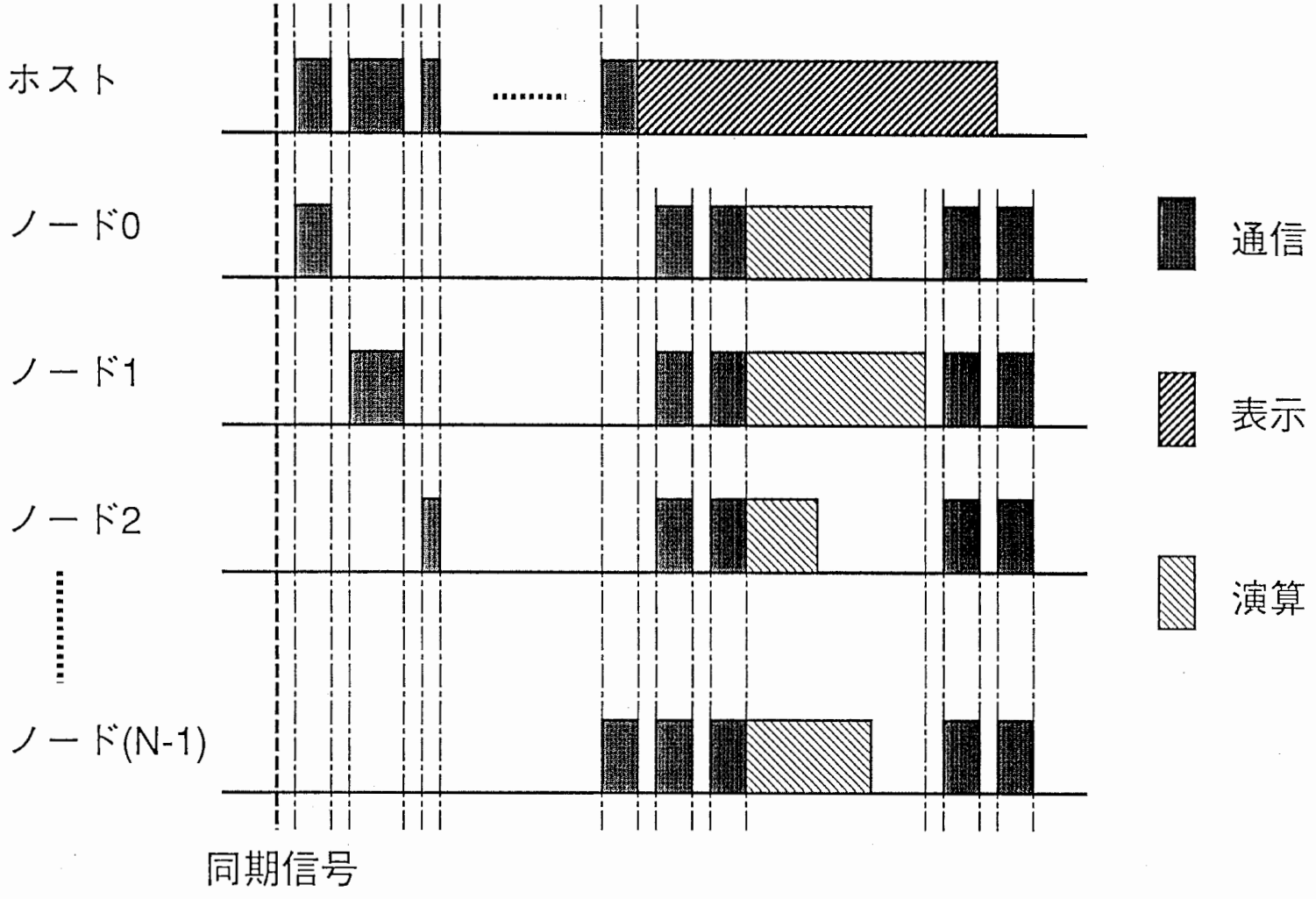
・ノード プログラム

状態遷移ルールの適用による
プレーンの更新

2) プレーンの分割処理



CM-5へ移植したシミュレータのタイミングチャート



CM-5へ移植した結果

効果

1) 状態遷移ルールの適用と表示を
分割することによる高速化

2) プレーンを分割することによる
状態遷移ルールの適用時間が短縮

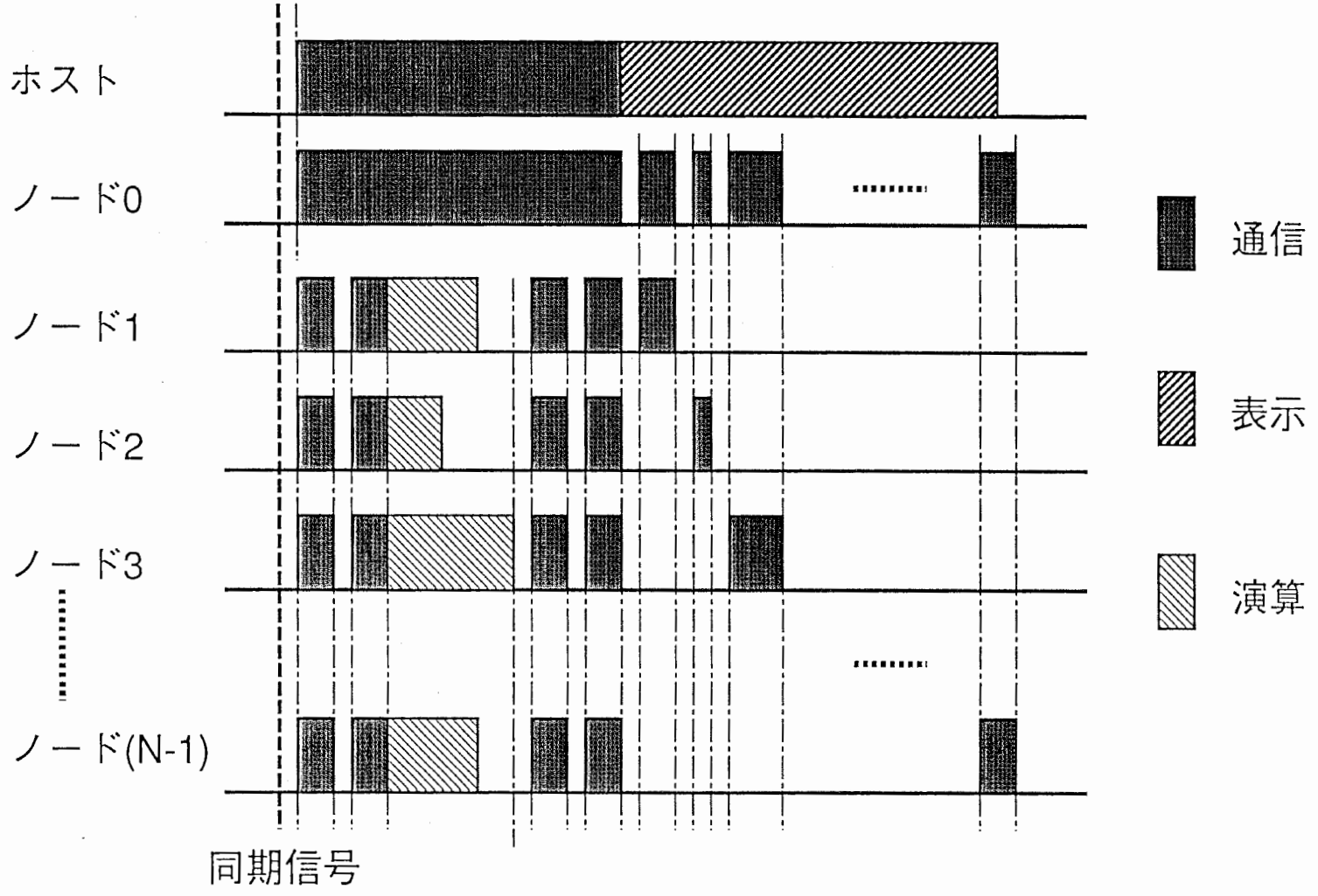
問題

データの通信時間には
オーバーヘッドがあり
ノード数が増えると無視できない

解決方法

1) ノード数の最適化
2) データの通信のみ行うノードを用意

改良を加えたシミュレータのタイミングチャート



CAM-Brainシミュレータの用途

現在、状態遷移ルールの開発が中心

↓

状態遷移ルールを適用した時の
状況を早く知ることが重要

↓

状態遷移ルールの完成

↓

遺伝子の進化する状況を
知ることがより重要

↓

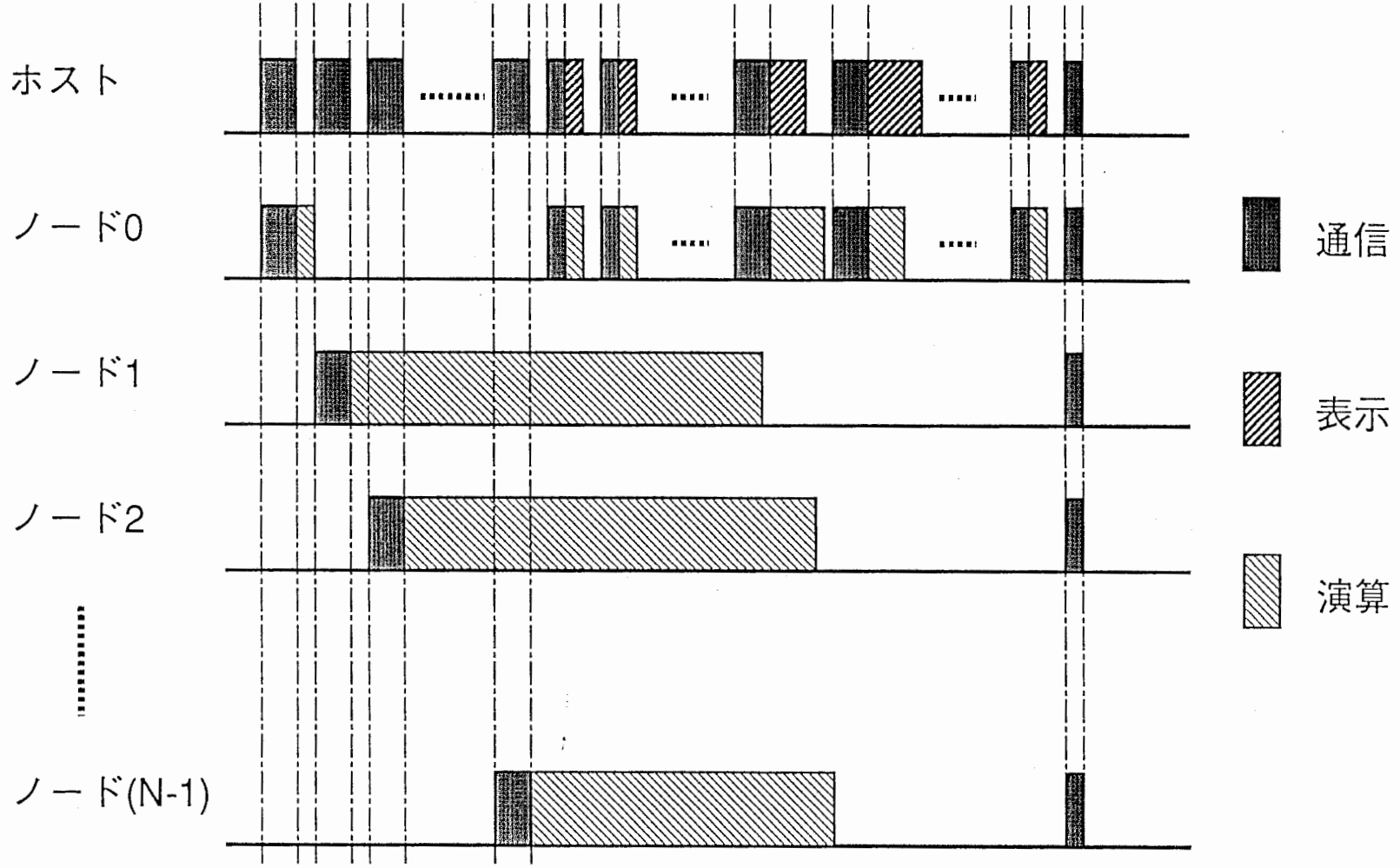
1世代のそれぞれの遺伝子を
同時に並列に処理させる

CAM-Brain シミュレータの変更点

1) ホスト プログラム
   - 最良遺伝子のプレーンの表示
   - 各々のノードに別の遺伝子を配布

2) ノード プログラム
   - ひとつのノードがひとつの遺伝子を処理
     (プレーンの分割処理の廃止)
   - ホストに送るデータは遺伝子の評価値のみ
   - プレーンの更新状況をホストに送るノードは
     (前世代での)最良遺伝子を担当したノードのみ

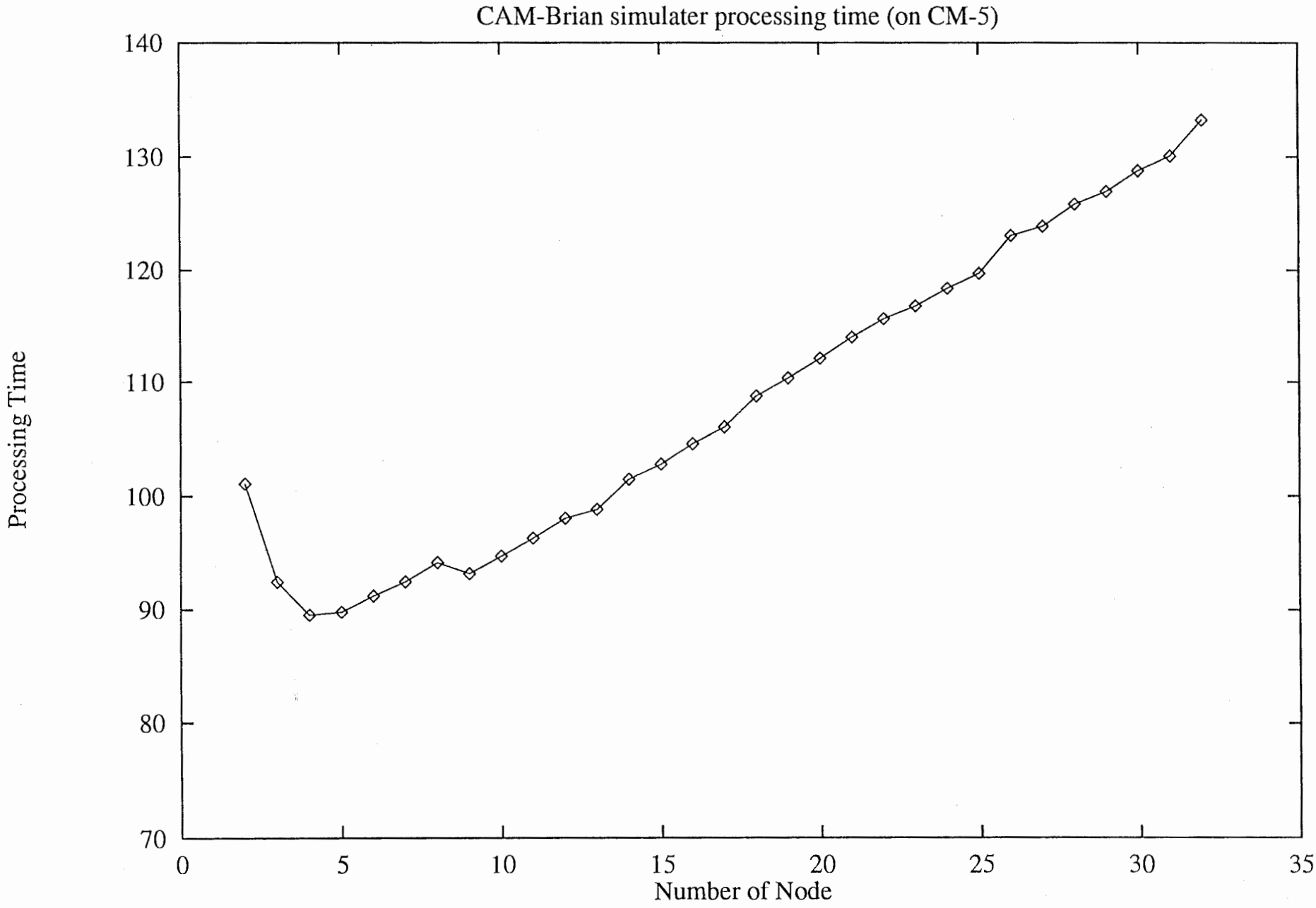
遺伝子進化型シミュレータのタイミングチャート

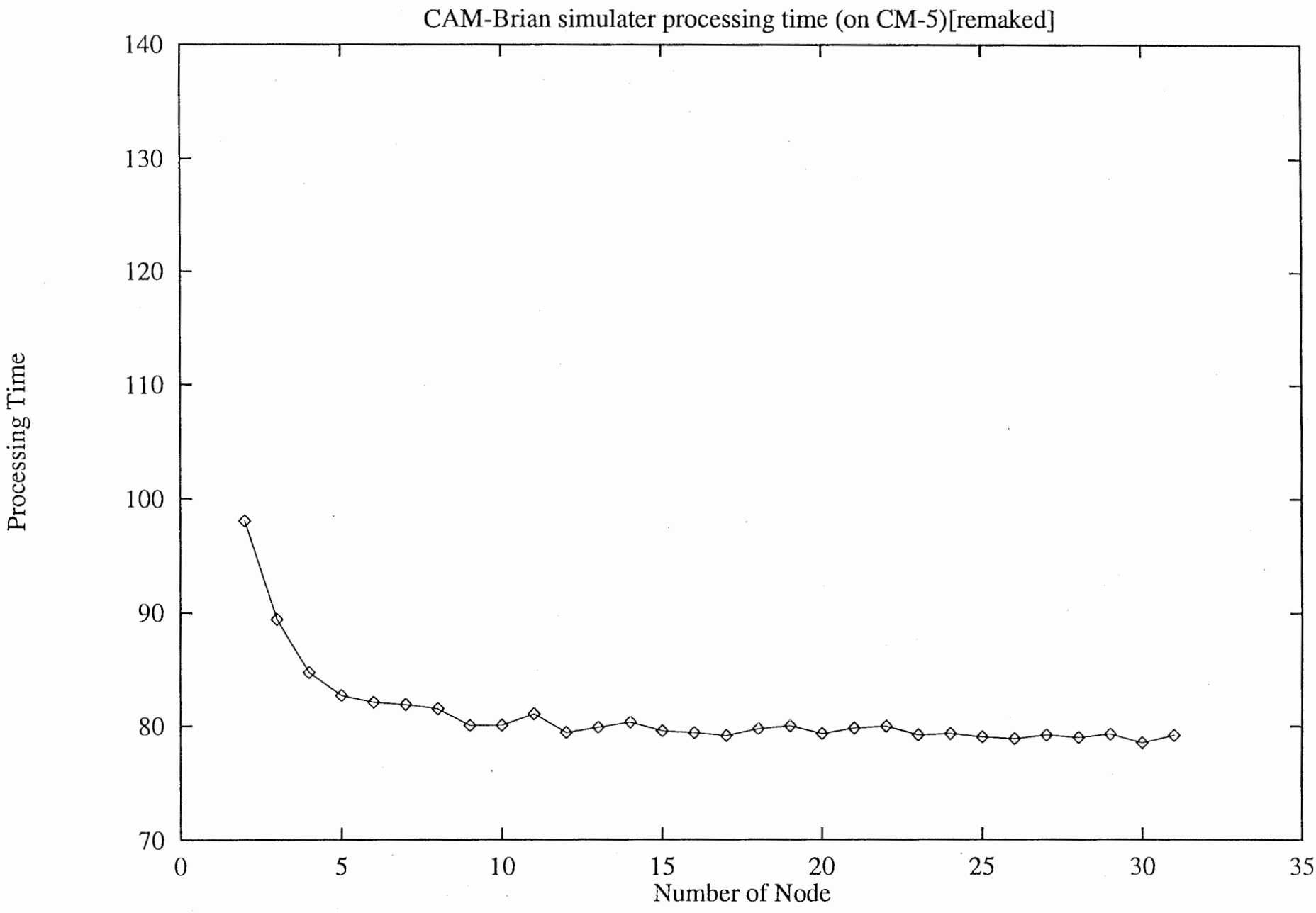


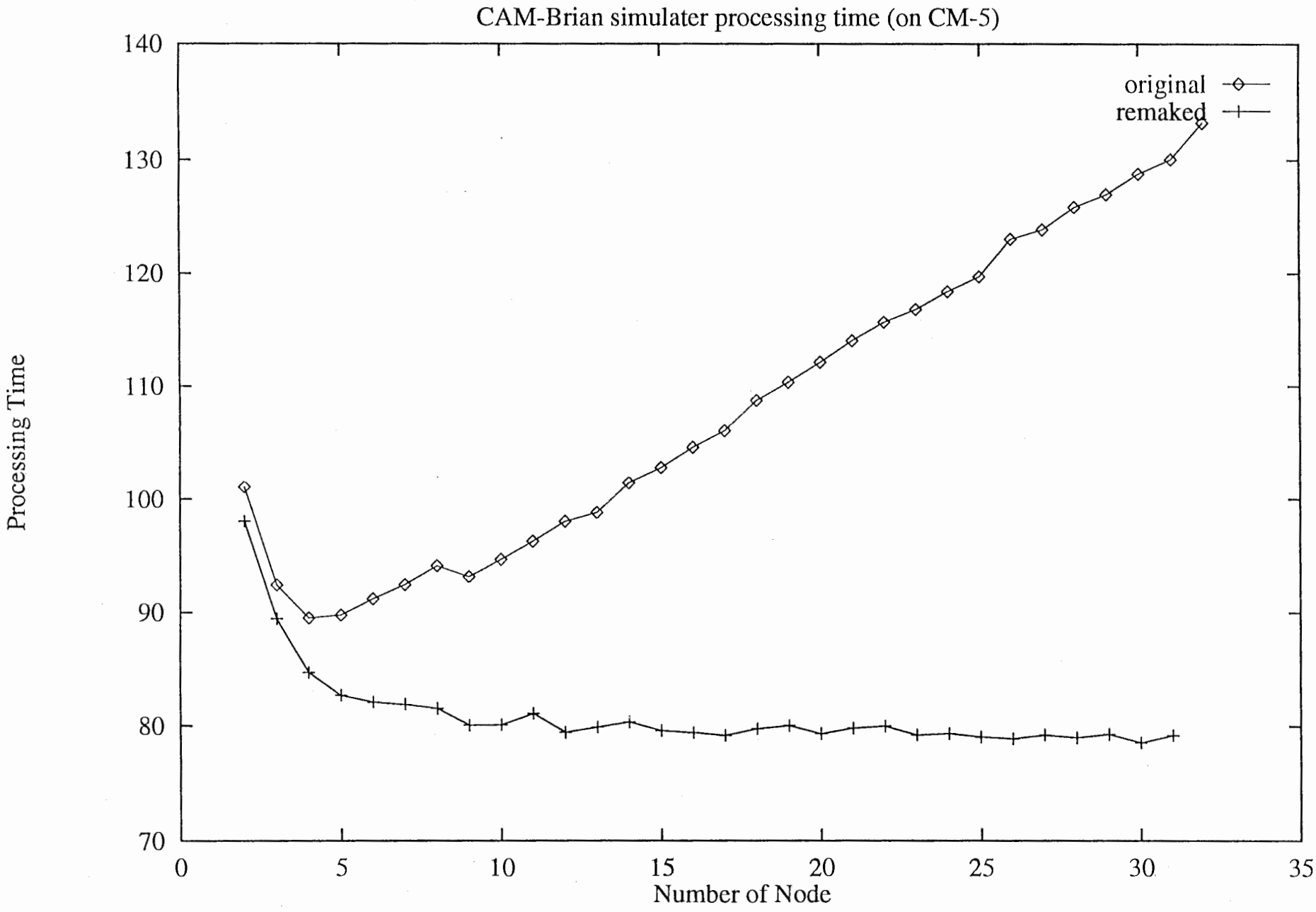
今後の問題

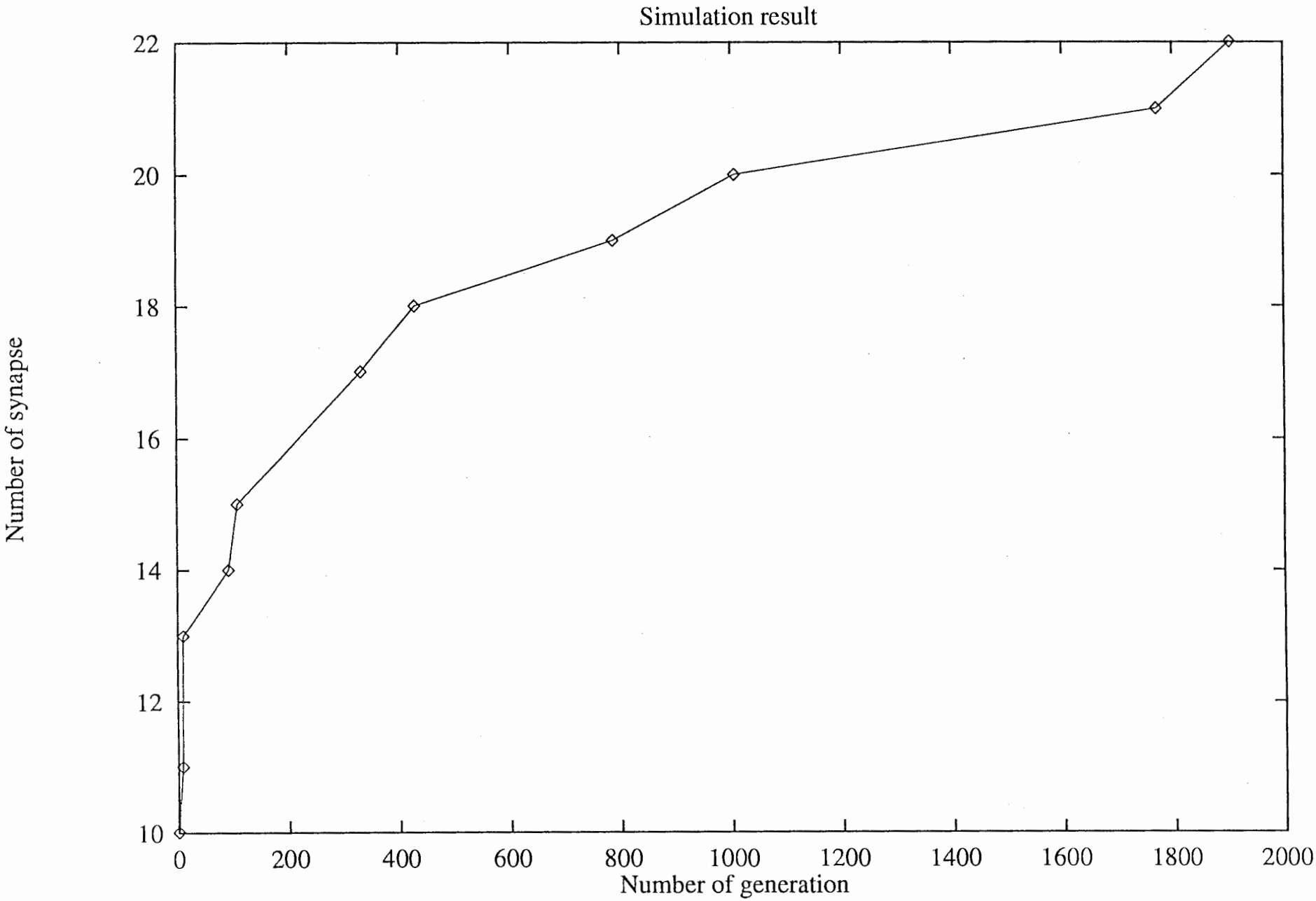
・バックグラウンド処理を可能にする

・初期データを可変にする


CAM-Brian simulater processing time (on CM-5)
Processing Time
Number of Node
140
130
120
110
100
90
80
70
0
5
10
15
20
25
30
35


CAM-Brian simulater processing time (on CM-5)[remaked]
Processing Time
Number of Node
140
130
120
110
100
90
80
70
0
5
10
15
20
25
30
35


CAM-Brian simulater processing time (on CM-5)
Processing Time
Number of Node
original
remaked
140
130
120
110
100
90
80
70
0
5
10
15
20
25
30
35


Simulation result
Number of synapse
Number of generation
22
20
18
16
14
12
10
0
200
400
600
800
1000
1200
1400
1600
1800
2000