7F-7

並列計算機 $(SM)^2$-II への汎用ニューラル・ネットワーク・シミュレータの実装

森下 明, 新海 浩, 天野 英晴
慶応義塾大学

1 はじめに

近年、ニューラル・ネットワークの研究が盛んになり、それに伴って数多くの仮想モデルが提案されている。この為、手軽にモデルの正当性の検証ができる汎用シミュレータに対する需要がますます高まっていくと思われる。このようなシミュレーションには膨大な処理が必要であるが、現存のワーク・ステーションでは能力不足である。一方、最近、汎用並列計算機が普及し、これを用いることによりある程度高速で柔軟性の高いシミュレータが実現できる可能性が現れた。そこで、汎用並列計算機の為のニューラル・ネットワーク・シミュレータに関して検討し、これを本理工学部で開発した$(SM)^2$-II に実装する事にした。本報告は、このシミュレータの概要と実装方針に関するものである。

2 シミュレータ概要

2.1 現存するシミューレータとの関係

現在ワーク・ステーション上で稼働しているシミュレータとしては、「Rochester Connectionist Simulator」,「SunNet」などがある。前者は汎用性において非常に優れておりニューロン構造の定義法や、ネットワークの構成法などにも学ぶべき点が多い。そこで、これを拡張し並列計算機上で、より柔軟な処理と複雑な同期管理を実現する事にした。記述言語はCである。ユーザプログラム例を図1に示す。

2.2 システムの規定するユニット構造

システム上では、ニューロンはユニットとして表現され、図2に示す様に、1つのユニットは複数の「**メインサイト**」「**インサイト**」「**アウトサイト**」「**リンクサイト**」より構成される。ユニット生成は、ユニットという器に各サイトを次々に付加していくという形式で行う事にする。

2.3 同期管理について

- 同期管理の主体は**メインサイト**であり、同一ユニット内のメインサイトでも異なる動作タイミングで動作しうる。これによって、Back-Propagation 等のニューロンをも1ユニットで構成できる様になる。
- 同期/非同期動作の2タイプのメインサイトを想定し、それらには以下の相違点がある。
  非同期動作メインサイト
  確率的動作タイプ：単位時間毎に確率 P で動作する
  定間隔動作タイプ：定時間隔 T 毎に動作する
  同期動作メインサイト
  指定されたリンクサイトの全てに入力が揃った時点で動作する

Implementation of general neural network simulator on parallelcomputer $(SM)^2$-II

Akira Morishita Hiroshi Sinkai Hideharu Amano
Keio University

3 ターゲット・マシン$(SM)^2$-IIの構造

$(SM)^2$-II は、相磯研究室で1984年に開発した「大規模一般疎行列計算」用のマルチキャスト可能なメッセージ・パッシング型の並列計算機である。シングル・バス結合されたPE群がクラスタを構成し、リンク結合したクラスタ群がシステムを構成する。プロセス生成に関してはスタティックである。

4 実装方針

4.1 PEにマッピングするプロセスに関して

- プロセス生成方針としては、図3の2通りが考えられる。

  1プロセス = 1ユニット /1PE = n プロセス方式
  プロセス管理を全てOSに任せられる反面、「プロセススイッチのオーヴァーヘッドが大きい」、「同期/非同期の柔軟な管理がやり難い」、「OSによっては、メモリ効率が非常に悪くなる」等の欠点が伴う。
  1プロセス = n ユニット /1PE = 1 プロセス方式
  上記の欠点が無い反面、システムが複雑になるが、当シミュレータはこの方式を採用する事にした。アーキテクチャへの依存性を極力排除する為である。この場合、「プロセス間通信 = PE間通信」となる。

- 各プロセスの受け持ちユニットの決定
  図1で示したユーザ・プログラムは全てのPEに転送される。そして、各PEは実行開始時に共通のアルゴリズムで自分の担当するユニットを決定し、その構造体をローカル・メモリ内に生成する事にした。

  この方式ならば、ダイナミック・マッピングを行う拡張をした時にも少数のパラメータを転送するだけで良いので効率がいいと思われる。

4.2 ユニットとネットワークの表現

- 上記の方式を実現する為に、各ユニットを図4の様な構造体で表現する。

4.3 各ユニットの処理方法

- 以下の3つのターンの実行で1イテレーション・サイクルが終了する。
  インサイト・ターン
  全インサイトでPE間通信を担当するシステムルーチンが実行され、次いでユーザ関数が実行される。この際、同期動作のメインサイトで動作条件が満たされたものに対して、動作可能フラグを立てる。
  メインサイト・ターン
  同期動作メインサイトは動作可能フラグの参照によって、非同期動作メインサイトは前述の方式に従って動作の可否を決定し、ユーザ関数を実行する。
  アウトサイト・ターン
  全アウトサイトでユーザ関数が実行され、次いでPE間通信を担当するシステムルーチンが実行される。

コントロール・ターン
マスタープロセスからの種々の制御信号を受信したり、現在の状態などを送信したりする。

ここで、各サイトの実行に関しては、構造体リストを辿りながら各 func メンバーの示す関数を実行するだけであり非常に簡単である。また、一般には各ターン毎に全 PE 間で終了の同期を取る事になるが、全ユニットが非同期動作だったならば、各 PE を勝手に動作させる事も可能であり、この時はイテレーション・サイクルなる概念は消滅する。

5 おわりに

本シミュレータは、$(SM)^2$-II 上に実装するが、その特殊なアーキテクチャへの依存性は極力抑えて作成されている。また、高速性の追究よりも高い汎用性の方に重点を置いている。それ故、他の汎用の並列計算機、或はワークステーション上にも容易に移植可能である。今後は、汎用シミュレータに最適なアーキテクチャなどについての研究も並行して行うつもりである。

[参考文献]

[1] N.H.Goddard,et al.,「Rochester Connentionist Simulator」 Univ. of Rochester Computer Science
[2] H.Amano,et al.,「$(SM)^2$-II: The new version of the Sparse Matrix Solving Machine」,Proc.of 12th Ann.ISCA.June 1985

```
USERPROG()
  {
                    |
  MakeUnit("exor","1Dvec",5,0,0,"cal",1,&uid);
  for ( i=0;i<5;i++ )  {
      buf=MakeName("exor[%d]",i);
      AddMsite(buf,"EXOR",MS_FUNC,&mid);
      AddIsite(buf,"EXOR","IS",IS_FUNC,&isid);
      AddOsite(buf,"EXOR","OS",OS_FUNC,&osis);
    }
  for ( i=2;i<4;i++)  {
      buf1=MakeName("exor[%d]",i);
      for ( j=0;j<2;j++)  {
          buf2=MakeName("exor[%d]",j);
          Makelink(buf2,"EXOR","OS",
                   buf1,"EXOR","IN",WGT_FUNC);
                   |
```

図 1: ユーザプログラム例 (exor 演算)

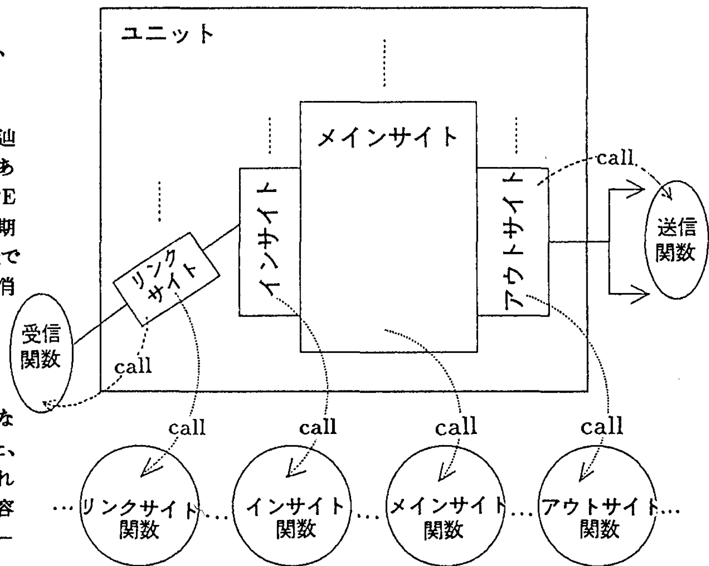
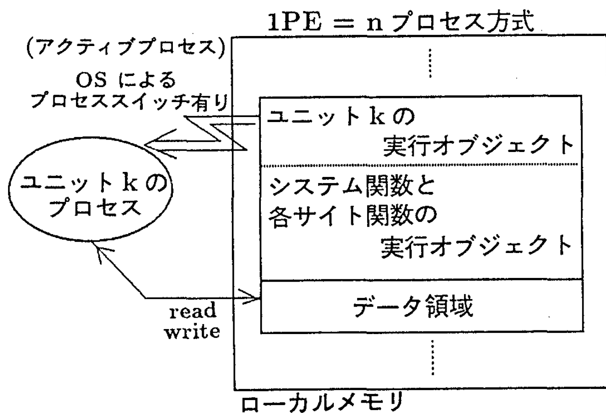


図 2: ユニットの構造

```
/* - - - - -  ユニット構造体  - - - - - */
struct  UNIT {
        char    name[20];       /* ユニット名                          */
        char    *type;          /* ユニットのタイプ名                  */
        int     index;          /* ユニットID                          */
        char    state;          /* ユニットの状態                      */
        char    flag;           /* システム用フラグ                    */
        struct  Msite   *msite; /* 所有メイン・サイトのアドレス        */
        struct  Unit    *next;  /* 構造体リストの次のアドレス          */
        };
/* - - - - -  メイン・サイト構造体  - - - - - */
struct  MSITE {
        double  value;          /* ユーザー用変数                      */
        double  data1;          /* ユーザー用変数                      */
        double  data2;          /* ユーザー用変数                      */
        char    *name;          /* メイン・サイト名                    */
        int     index;          /* メイン・サイトID                    */
        int     *func;          /* メイン・サイト関数のアドレス        */
        char    info;           /* 処理する情報タイプ                  */
        char    flag;           /* システム用フラグ                    */
        short   event;          /* 現時点での入力イベント数            */
        short   evmax;          /* サイト駆動に必要なイベント数        */
        struct  Isite   *isite; /* 所有イン・サイトのアドレス          */
        struct  Osite   *osite; /* 所有アウト・サイトのアドレス        */
        struct  Msite   *next;  /* 構造体リストの次のアドレス          */
        };
/* - - - - -  イン・サイト構造体  - - - - - */
struct  ISITE {
        double  data1;          /* ユーザー用変数                      */
                  S
```

図 4: ユニット / サイトの構造体リスト



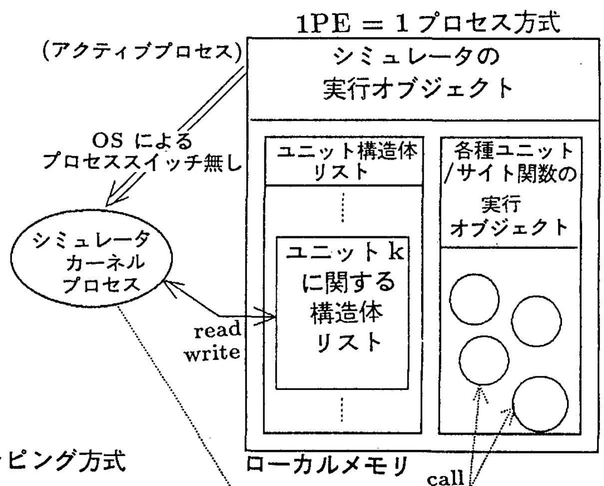


図 3: ユニットのマッピング方式