神経網の離散事象シミュレーションの並列化
Parallel Discrete Simulation of a Nerve Network

三輪 忍

平成12年2月15日

Abstract

現在、神経網の離散事象シミュレーションを並列化することが望まれている。その理由として、大規模な神経網を逐次的にシミュレートしたのでは時間がかかり過ぎてしまうことが挙げられる。そのようなものをシミュレートする際、並列化を行い、高速にシミュレートする必要がある。なお本シミュレーションにおいては、発火の情報を事象として離散事象シミュレーションを行っている。

神経網の離散事象シミュレーションを並列化するにあたり問題となるのは、極端に負荷がばらついているという性質である。これは、実際の脳におけるニューロンの活動にばらつきがあるためである。実際の脳においては、1つの動作の処理に全体の約1%の神経細胞しか使われない、という状態が頻繁に現れる。そして我々は、厳密に脳の状態をシミュレートすることを目的としているため、シミュレーションにおける各ニューロンもこのような状態にあると考えられる。

離散事象シミュレーションを行うのであるが、本シミュレーションにおいては論理回路の離散事象シミュレーションの並列化において有効とされてきた手法を適用することが難しい。それは神経網と論理回路という異なる対象をシミュレートしようとするからである。対象が異なれば並列化の際に考慮されるべき特徴も当然異なるため、神経網において考慮されるべき特徴が考慮されていない可能性がある。

つまり、神経網の離散事象シミュレーションが有する特徴を考慮した何らかの手法により、並列化する必要がある。神経網の離散事象シミュレーションは現実の脳の活動にもとづき、1) 最近発火したニューロンの周辺のニューロンが次に発火する確率が高い、2) 最近活動したニューロンは次に活動する確率が高い、という特徴を有している。

そこで、発火履歴をもとにした負荷分散を用いて並列化する。ここで発火の情報を負荷分散の参考にしたのは、発火が負荷の目安となるからである。よって、発火履歴から各プロセッサの負荷を推定し、負荷が均等になるように分散を行うことにする。

各ニューロンの過去の発火情報は、2次元平面上の1点として表現する。これを、ネットワークを2次元平面に射影することにより実現している。

このようにして全ニューロンの発火履歴を反映した平面を、プロセッサ台数分の領域に分割する。分割は、各領域の負荷が均等になるように行われる。分割によって生成された個々の領域が、各プロセッサが担当するニューロンを表すことになる。

次に、分割された領域はそれぞれプロセッサに割り当てられる。各プロセッサはその割り当てられた領域に従い、処理を進めることになる。つまり、領域内のニューロンに対しては計算を行い、領域外のニューロンに関しては行わない。

さらに、負荷分散処理が全シミュレーション時間に占める割合を軽減することを考える。そのため、この処理を毎時行うのではなく、ある一定の時間間隔を空けて行うようにする。このように時間間隔をある程度空けても負荷の分散に影響はほとんどない。それは、短い時間間隔においては、活動しているニューロンの分布が

ほとんど変化しないためである。

このような負荷分散を用いてNECの並列計算機Cenju-4上で並列化を行い、その結果を測定した。ニューロン数約15000、全接続枝数約100000という環境でシミュレーションを行った。平均発火率26.7%の状況で、100回に1回負荷分散を行った場合の台数効果を測定したところ、32台のプロセッサを用いた時に約11倍の速度向上が得られることが判明した。また静的な割り当て、すなわち負荷分散を行わない場合との実行時間の差も測定したところ、32台のプロセッサを用いた時に、最大で約22%の速度向上が得られることが判明した。

Parallelizing the discrete simulation of a nerve network is coming important. Because the cost of simulating a large network sequentially is too much. When simulating such a network, fast simulating in parallelizing is required. Besides, our simulation is a discrete simulation, whose events are the fires of nerve cells.

When parallelizing a discrete simulation of a nerve network, the most critical problem is load balancing. This is caused from the brain's aspect that the actions are different among the nerve cells. In the brain, there is a state that about 1 % of the nerve cells is only used when calculating 1 input. We want to simulate brain strictly, so each nerve cell in the simulation is in such a state.

However simulating discretely, our simulation is difficult to be adapt the methods that is valid in the field of the discrete simulation of a logic circuit. Because there are differences of the object simulated between logic circuit and neural net. So, there are differences of the aspect required to consider among them, and the aspect required in after one may not be considered in before one.

As a result, the method is required which is considered the aspect of a discrete simulation of a nerve network, and it must be used in parallelizing. Such simulation has the aspect: 1)the rates of the neighborhood of the fire cells are higher than those of the others, 2)the rates of the recent fire cells are higher than those of the others.

So, our discrete simulation of a nerve network was parallelized with the fire result. The reason of using the fire result is an aim to consider the load. The processors assume the load from the fire result, and achieve load balancing.

Each fire result is represented by 1 point on the square. By a projection of the network, this is actualized.

Such square of the fire results of all cells is divided by the number of the processors. The division makes the load of each area almost same Each area created by the division represents the cells each processor is in charge.

Next, the divided areas are assigned to each processors. Each processor calculate in the rule of its area. In the other words, the cells in its area is calculated and the cells out of its area is not calculated.

The percentage of the total time of the simulation is not decreased to execute the operation of load balancing at every clock time, but to execute it on every period that is constant time. Such method on every period doesn't affect the load balancing so match. Because, in the short time, the distribution of the fire cells is not so changed.

We implemented such a method on the parallel computer Cenju-4 made in NEC, and evaluated the performance. Performance evaluation of this simulator used a nerve network model that have about 15000 nerve cells and about 100000 connection in total. In the situation, with a rate of fire nerve cell of 26.7%, when we use 32 processors and load balancing once a hundred iteration, we get about 11 times speed up. With 32 processors, we get about 22 % speedup compared to the case of a static assignment in best cases.

目次

1 はじめに 3

2 神経網シミュレーション 3
  2.1 ニューロンモデル 4
    2.1.1 従来のニューロンモデル 4
    2.1.2 パルス動作型ニューロンモデル 4
  2.2 離散事象シミュレーション . . . 5
  2.3 並列処理から見た神経網シミュレーションの特徴 7
  2.4 関連研究 8

3 シミュレータの並列化 9
  3.1 Cenju-4 9
    3.1.1 データ通信 9
  3.2 並列処理方式 9
    3.2.1 シミュレーションの流れ 11
    3.2.2 負荷分散の概念 12
  3.3 実装 12
    3.3.1 負荷分散方式 13

4 評価 14

5 おわりに 16

1 はじめに

以前から神経網シミュレーションに用いられてきたニューロンモデルでは、各ニューロンの出力値は、シグモイド関数を用いて 0 から 1 のアナログ値によって表現されている。しかしこのモデルにおいては、実際の神経細胞の間での信号伝達に関与していると見られる、パルス情報が失われていた。そこでパルス情報も考慮に入れた、パルス動作型ニューロンモデルが考案された。

このモデルにおいては全ニューロンの計算を毎時行う必要はなく、特定のニューロンのみ計算をすれば正しい結果を与える。この「特定のニューロンのみを計算」する方法は、論理回路シミュレーションで用いられる離散事象シミュレーションにより実現されている [1]。すなわち、逐次レベルでの高速化は達成されている。

実際の脳には百数十億個もの神経細胞が存在する。このような大規模なものを逐次的にシミュレートしたのでは、時間が掛かり過ぎてしまい現実的でない。そこで並列にシミュレートすることが望まれる。

離散事象シミュレーションの並列化の研究は、論理回路シミュレーションを主として様々な研究が成されている。それにより、Time Warp・Virtual Time といった並列化手法が提案された。しかし神経網という異なる対象をシミュレートする場合には、これらの手法を適用することが困難である。そのため、神経網の有する特徴を考慮した並列化を行う必要がある。

本論ではまず次章において、従来のニューロンモデルについて述べ、続いてパルス動作型ニューロンモデルについて述べる。次にそのパルス動作型ニューロンモデルが離散事象によるシミュレーションが可能であることに触れた後、実際にどのように離散事象シミュレーションを行っているかについて述べる。さらに、並列処理の観点から見た特徴を、離散事象シミュレーションの代表例である論理回路シミュレーションと比較しながら述べる。3 章では、その特徴を考慮して NEC の並列計算機 Cenju-4 上並列化を行ったので、その並列処理方式について述べる。最後に 4 章でその実現結果を示し、今後の課題を挙げる。

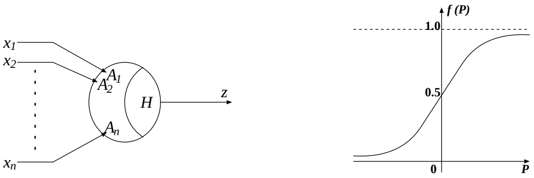
2 神経網シミュレーション

本章ではまず、従来のニューロンモデルについて述べ、続いて、本シミュレーションにおいて採用したパルス動作型ニューロンモデルについて述べる。さらに、このモデルが離散事象シミュレーションが可能であることについて触れた後、離散事象の神経網シミュレーションについて述べる。その後、この離散事象の神経網シミュレーションの特徴を並列処理の観点から、離散事象シミュレーションとして代表的な論理回路シミュレーションと比較しながら述べる。

2.1 ニューロンモデル

本節では、従来のニューロンモデルについて触れた後、実際に今回採用したパルス動作型ニューロンモデルについて述べる。

2.1.1 従来のニューロンモデル



<単純なニューロンモデル>　<シグモイド関数>

図 1: 従来のニューロンモデルとシグモイド関数

最も単純かつよく用いられるモデルはMcCulloch & Pitts による形式ニューロンモデルである (図 1)。このモデルにおいては、1 つのニューロンは複数の入力 $x_1, x_2, \cdots, x_n$を受け、膜電位 $P = \sum_i^n A_i x_i$を計算する。そしてその出力 zを $z = f(P - H)$ によって判断する。ニューロンに伸びる入力/出力枝をシナプスと呼ぶ。Hは閾値と呼ばれ、$P \geq H$となる時そのニューロンは発火すると言う。

このようなニューロンを、複数の出力も許し (その際の出力値は同じ) 複数結合したものがニューラルネットである。

当初は評価関数fとして、$x \geq 0$の時1、$x < 0$の時 0 となる単位ステップ関数 $f(x)$ を用いてきた。一般には、シグモイドのような単調増加関数や単調非減少関数を用いる。この場合の出力値は 0 から 1 のアナログ値を取り、これによりニューロンの平均発火頻度を表現する。

しかしこのモデルにおいては、実際の脳内の神経細胞における信号伝達において観測されるパルス情報などは考慮されていない。つまり細胞間を伝わるパルスの到着時刻や到着間隔といった時間的情報がこのモデルでは失われてしまい、そのことがモデル全体の表現能力に限界を与えてしまっている可能性がある。

2.1.2 パルス動作型ニューロンモデル

実際の神経網におけるパルスの到着時刻や到着間隔といった時間的情報までも考慮したモデルにパルス動作型ニューロンモデルがある。

時刻 $t \geq t_0$ におけるニューロンの膜電位を $P(t)$、閾値電位を $H(t)$ とした場合、それぞれは以下のような式で与えられるとする。ここで、各変数/定数の添字 E, I はそれぞれ、興奮性、抑制性を表している。

$$
\begin{aligned}
P(t) &= P_\infty + (P_0 - P_\infty) e^{-(t-t_0)/T_p} \\
&\quad + \sum_{j=1}^{K} A_j \sum_{i=1}^{n_j} e^{-(t-t_{ji})/T_p} \\
H(t) &= H_\infty + (H_0 - H_\infty) e^{-(t-t_0)/T_h}
\end{aligned}
$$

- P_0, H_0 初期リセット膜電位、初期リセット閾値電位
- P_∞, H_∞ 静止膜電位、静止閾値電位
- A_j j番目のシナプスにおける単一パルスあたりのニューロンの膜電位の変化量
- K 入力線路数
- n_j j番目の入力線路への入力パルス数
- t_{ji} j 番目の入力に i 個目の入力パルスが印加された時刻
- t_0 発火後、不応期τを経た時刻
- T_p, T_h 膜電位、閾値電位の時定数

膜電位 $P(t)$ はパルスが到着しない間は静止膜電位に指数関数的に収束するが、パルスが到着するとその電位を急激に変化させ、そののちまた静止膜電位に収束し始める。これに対し、閾値電位 $H(t)$ は単に減衰を続け、静止閾値電位 H_∞ に指数関数的に収束する。

時刻 t において膜電位 $P(t)$ の値が閾値電位 $H(t)$ の値を上回ると、その神経細胞は発火する。出力は発火を $0/1$ の 2 値によって表現する。この際、$P(t)$, $H(t)$ は初期値 P_0, H_0 にリセットされる。これをそれぞれ初期リセット膜電位、初期リセット閾値電位 と呼ぶ。

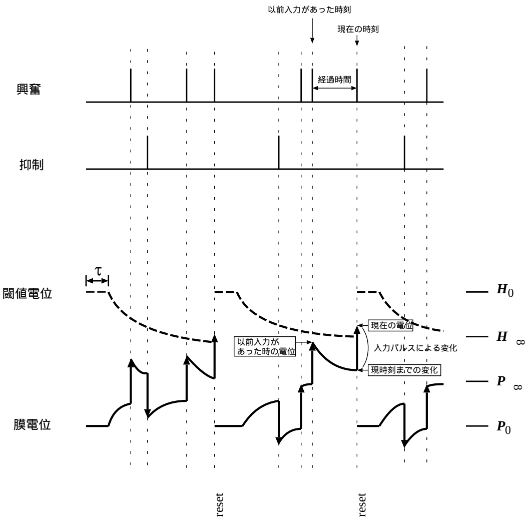


図 2: 入力の変化と膜電位の変化

現実の脳には発火後のニューロンは、しばらくの間活動しないという性質がある。不応期とはこのニューロンが活動しない期間のことであり、従ってこの期間は膜電位・閾値電位の変化はない。リセット後は不応期τを経て、ニューロンは活動を再開する。(図 2)

シミュレーションの際には、このような計算が毎時全ニューロンに対して行われるわけだが、本シミュレーションでは入力が変化したニューロンのみ計算を行っている。以下にその正当性を述べる。

計算の目的というのは、その時刻において各ニューロンの膜電位が閾値電位を越えるかどうか確認することである。膜電位 $P(t)$ と閾値電位 $H(t)$ はそれぞれ静止膜電位と静止閾値電位に収束しようとするため、[静止膜電位 $P_\infty <$ 静止閾値電位 H_∞] であれば、前時刻に発火しないニューロンに対し現時刻で入力が変化しなければ、図 2からも分かるように、$P(t)$、$H(t)$ はそれぞれ P_∞、H_∞ に近づくだけである。つまり、$P(t)$、$H(t)$ の描く軌跡が交わる、すなわち入力が変化することなくそのニューロンが発火することは決してない。そのため、膜電位の計算を行うまでもなくそのようなニューロンは現時刻では発火しないことが分かる。よって $P_\infty < H_\infty$ であるならば、膜電位が閾値電位を越えるのは入力パルスが変化した時のみである、と言える。

また前述の膜電位の計算式は、以下の様に変形が可能である。

$$
\begin{aligned}
P(t) &= P_\infty + (P(t_{before}) - P_\infty)e^{-(t-t_{before})/T_p} \\
&\quad + \sum_{j=1}^{K} A_j \\
P(t_0) &= P_0
\end{aligned}
$$

・t_{before} 以前パルスが印加された時刻

つまり、現時刻の膜電位は現在の時刻 t、以前発火した時刻 t_0、以前パルスが印加された時刻 t_{before} とその時の膜電位 $P(t_{before})$ から計算可能である。従って現時刻の膜電位の算出に、1 つ前の時刻の電位などの情報は必要ない。よってこのように入力が変化したニューロンのみ計算を行っても、シミュレーションは正しい結果を与える。

2.2 離散事象シミュレーション

実際に計算機で行うことは各ニューロンに対し、入力信号から膜電位・閾値電位を計算し、出力すなわち発火の判断をすることである。本来であれば全てのニューロンに対し、そのニューロンの全ての入力を見て電位の計算と発火の判断を行うべきである。しかし、2.1.2で述べたように、全ての入力が変化しないニューロンについては計算の必要がない。よって入力が変化したニューロンのみ計算する、すなわちあるニューロンが発火した時に、次にその出力が繋がる先のニューロンのみを計算するという処理を繰り返せば、正しい結果を与えることになる。これは、ニューロンの発火を事象とする離散事象シミュレーションが可能であることを意味する。シミュレーションの時刻管理には Time Table を用いた。その処理は次の様なものである。

事象は Time Table の処理すべき時刻に登録される。登録される場所をスロットと呼ぶ。計算機は各時刻において、Time Table のその時刻のスロットに登録されている事象を全て処理し、その後時刻を進める。

各ニューロンは、

1. そのニューロンが以前発火した時刻
2. そのニューロンの入力が以前変化した (そのニューロンが以前計算された) 時刻
3. そのときに変化した膜電位
4. 膜電位、閾値電位の時定数
5. 出力枝それぞれに対する荷重
6. 出力枝それぞれに対する遅延時間
7. そのニューロンの出力先ニューロン

などを情報として持つ。

1 ~ 5 の値は、電位の計算に用いている。まず直前の信号到着時刻と現時刻から経過時間を割出す。その経過時間を用いて以前の膜電位の値からの変化を計算し、それに現時刻の入力パルスの変化量 (入力枝に設定されている荷重) を加えることで現時刻の膜電位を計算する。(2.1.2 の変形後の式)

6、7 は事象の処理に必要である。なぜなら、発火したニューロンが分かり事象が生成されても、その出力先がなければ計算すべきニューロンが分からず、また遅延がなければ、そのニューロンを計算すべき時刻が分からないからである。

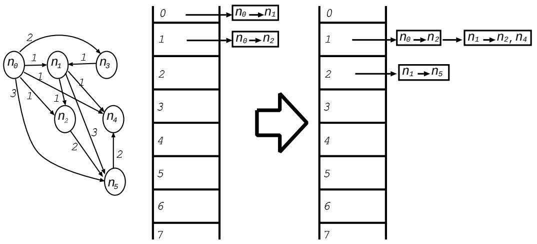


図 3: Time Table による事象管理

例として、図 3のようなネットワークを考える。矢印はニューロンの出力枝 (パルス信号の伝わる方向) を表し、矢印に付加された数字は遅延時間を表す。この遅延時間というものは、実際の神経網における神経細胞間の物理的距離をパラメータ化したものである。あるニューロンが発火した時、そのニューロンの出力先ニューロンの計算は、現時刻から遅延時間経過後に行われる。

今 $t = 0$ で n_0からの入力が変化し、電位計算の結果、n_1が発火したとする。このとき n_1が発火したという信号が伝わるニューロンは n_2、n_4、n_5の 3 つであり、これらの情報を Time Table に登録しなければならない。$n_1 \to n_2$間の枝の遅延時間は 1 であるから、n_1の出力が n_2に到達するという事象 ($n_1 \to n_2$) を Time Table の $t = 1$ の箇所に登録する。n_4、n_5についても同様に、各出力枝の遅延時間を考慮しながら Time Table への事象の登録を行う。(図 3右)

各事象は以下のような構成をしている。

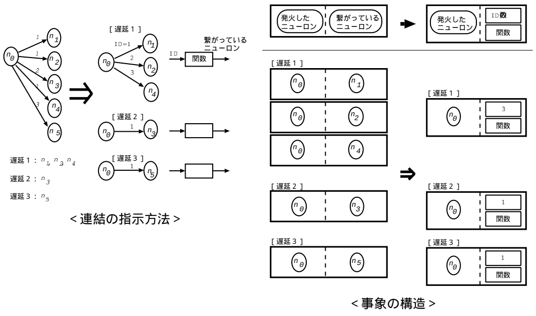


図 4: 枝の構成と事象の構造

あるニューロンが発火し、その出力先ニューロンの計算が行われる。出力先ニューロンの計算は、現時刻から各枝の遅延時間経過後に行われる。換言すれば、Time Table のその時刻のスロットに、そのニューロンを計算するという事象が登録されている。遅延が同じである出力先ニューロンは、同じ時刻に計算が行われる。すなわち、あるニューロンの発火により生成される事象の内、同一遅延の出力先ニューロンに関する事象は、Time Table の同一時刻のスロットに登録される。

図 4の例では、ニューロン n_0の出力先ニューロンである n_1、n_2、n_3、n_4、n_5を遅延によって分類すると、遅延 1 の組 (n_1, n_2, n_4)、遅延 2 の組 (n_3)、遅延 3 の組 (n_5) のようになる。ニューロン n_1、n_2、n_4に対する計算は同時刻に行われる必要があり、従って n_1の計算を行う事象、n_2の計算を行う事象、n_4の計算を行う事象は、全て Time Table の同一スロットに登録される。そこで Time Table に登録される事象の数を減ら

すために、遅延が同じである出力先ニューロンの計算を示す事象は、1 つにまとめて構成する。

事象が指示するニューロンに対し行うべき処理は、以下のようなものである。

1. 現時刻がそのニューロンの不応期を経過しているか否かを判断する。ニューロンには一度発火するとしばらくは発火しないという不応期が存在し、これを考慮する必要がある。(2 へ)
2. 膜電位及び閾値電位の計算
    - (a) 不応期期間中であった場合は何も行わずに、事象が指示する次のニューロンに処理が移る。
    - (b) 不応期が経過した場合は、そのニューロンの膜電位及び閾値電位を計算する。(3 へ)
3. 膜電位と閾値電位の比較 (発火の判断)
    - (a) 膜電位 $<$ 閾値電位 の場合、発火は起こらず次の処理に移る。
    - (b) 膜電位 $>$ 閾値電位 の場合は発火が起こり、以下の処理を行う。
        - i. そのニューロンの出力先である全てのニューロンに対して、対応する出力枝の遅延時間が同一であるもの毎にまとめ、事象を生成し、現時刻からその遅延時間経過後の時刻に対応するスロットに登録する。

発火したニューロンの膜電位と閾値電位はそれぞれ静止膜電位、静止閾値電位にリセットされる。電位計算と発火判断を合わせて行った場合、同時に複数の入力がある時は、全ての入力に対する計算を終えないうちに発火することが分かるかもしれない。すると、電位はそこでリセットされてしまい、その後の入力に対する計算はリセットされた値を用いることになり、正しい結果を与えない。

そこで、一度その時刻において計算されるべき全てのニューロンの膜電位 閾値電位の計算を行った後に、計算を行った全ニューロンに対して発火の判断を行うことにする。すなわち Time Table の各時刻のスロットは、電位計算のために 1 回、発火判断のために 1 回の計 2 回走査されることになる。

2.3 並列処理から見た神経網シミュレーションの特徴

現実の脳は、次のような特徴を有している。

空間的局所性 現在発火しているニューロンの周辺のニューロンは、次に発火する確率が高い。

時間的局所性 最近活動したニューロン群は次に活動する確率が高い。

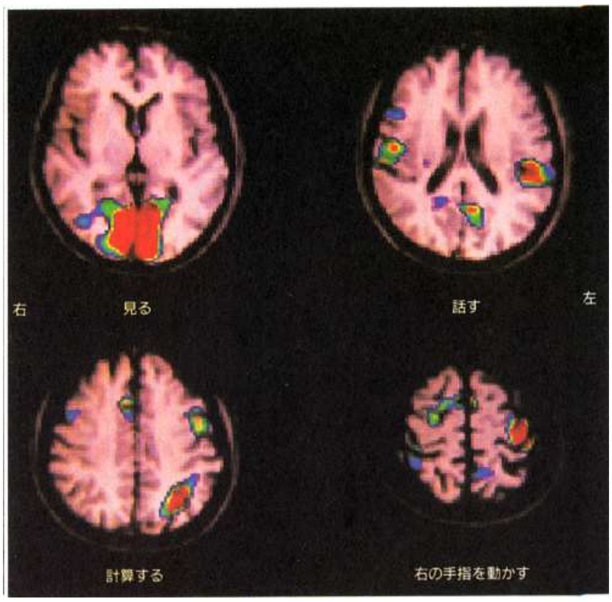


図 5: 現実の脳におけるニューロンの発火の様子

実際の脳におけるニューロンの発火の様子を図 5に示す。実際の脳においては 1 つの動作で使われるのはそのほんの一部分であり、その動作の間中その部分のニューロンのみ活動し、他の部分のニューロンは活動していない。

離散事象シミュレーションの代表的なものに論理回路シミュレーションがあり、論理回路シミュレーションの並列化手法として Virtual Time ・ Time Warp に代表されるような様々な手法が

考案されている。離散事象シミュレーションの並列化を行うのであるから、これらの手法を用いることはできないだろうか、と考えるのは当然のことである。しかし、これらの手法は論理回路のような特徴を有するものをシミュレートする場合に有効なのであり、今回のような異なる特徴を有するものをシミュレートする場合には有効とは限らない。

論理回路と脳との大きな違いは、素子(ニューロン)の活動のばらつきの程度にある。論理回路においては、素子間で多少の活動のばらつきはあるだろうが、例えば脳のように全体の1%のニューロンしか活動しておらず、他の部分は休止しているといった図5のような状態は起こらない。しかし、実際の脳はそのような論理回路では起こり得ない状態であり、本シミュレータはそのような状態をシミュレートすることを目指している。そのため、論理回路シミュレーションにおいて最良とされてきた手法を適用することが困難である。

2.4　関連研究

採用しているモデルは異なるが、神経網シミュレーションの並列化を行った例[2][3][4][5]もいくつか存在する。これらのシミュレーションにおいて採用されているモデルの中には、本シミュレーションにおいて採用されているモデルと同様、必要なニューロンのみ計算を行えば正しい結果を与えるモデルも存在する。しかし並列化の際に、計算の不要なニューロンに対しても計算を行うコストを問題としてこなかったため、これらのシミュレーションでは毎時全ニューロンに対し計算を行っている。

神経網シミュレーションにおける負荷とは、計算を行うニューロンの数と考えることができる。これは、ニューロンが異なっても、行われる計算量に大きな差がないためである。従って全ニューロンに対し計算を行うのであれば、全てのプロセッサに同数のニューロンを割り当てれば負荷のバランスはとれることになる。これは脳のような一部分しか使われていない状態をシミュレートする場合でも、ニューロンの計算がその活動状態に依存しないため問題ない。

このような毎時全ニューロンに対して計算を行うシミュレーションを、高速化のために必要なニューロンのみ計算するようにしたらどうなるであろうか。負荷とは計算を行うニューロンの数であるため、以前のように各プロセッサに同数のニューロンを割り当てたままでは、計算されるニューロンを多く担当しているプロセッサと計算されないニューロンを多く担当しているプロセッサとでは負荷にばらつきが発生してしまう。

これに対し、伊藤らが見解を述べている[2]。1つは、あらかじめ各入力データに対する負荷を予測し、予測される負荷が小さいものと大きいものの処理をそれらの間で協同して処理することによって、各ノードの負荷を平均化する方法である。従ってこの方法は、入力に対しどのニューロンが反応するか予測可能な場合に有効である。しかし我々のモデルにおいては、予測は困難である。全入力パターンに対する負荷のパターンをテーブルに記録しておき、入力の度にそれを呼出すという手段も考えられるが、そのような非常に大きなテーブルを用意することは物理的に不可能である。また全入力パターンというものは、入力を0/1の2値としても2の入力線数乗存在する。よって入力線の数を増やした時に、そのような全入力パターンを記録する作業に費す時間ははかりしれないものとなる。

また別の手法として、マスター・スレーブ方式による実行を彼らは挙げている。あるプロセッサを処理の管理用とし残りを計算用として、管理用のプロセッサが計算用のプロセッサにタスクを割り当てる。この方式は処理するタスクのサイズと数により効率は異なる。タスクが大きく管理プロセッサが管理するタスクの数が少なければ効果はあるが、タスクが小さく管理プロセッサが管理するタスクの数が多ければ効率が悪い。仮に百億個のニューロンをシミュレートする場合、発火率が1%としても1億個のニューロンが発火し、同数の事象が生成されることになる。本モデルにおいてタスクを事象と考えると、十億個の事象を1プロセッサで管理することになり、これは明らかに効率が悪いであろう。

以上に述べたように、論理回路シミュレーションとは特徴が明らかに異なるために、論理回路

の離散事象シミュレーションの並列化で用いられてきた手法を用いることはできない。また、毎時全ニューロンに対して計算を行う神経網シミュレーションとは根本的に異なるため、これらの研究成果に対し負荷分散の答えを求めることもできない。さらに、伊藤らの提案する方法も本シミュレーションでは採用できない。従って、前節の冒頭に述べたような特徴を考慮した、効果的な負荷分散を用いた並列化手法が求められる。

3　シミュレータの並列化

2.4で述べたように、何らかの効果的な負荷分散を用いて並列化する必要がある。そこで発火履歴をもとに、各プロセッサが担当するニューロンの過去の発火回数の総数が等しくなるような負荷分散を行うことにする。これは神経網には時間的局所性が存在するため、前回発火したニューロンの次に発火する確率が高いためである。さらに、実際に計算されるニューロンではなく発火するニューロンを負荷の目安としたのは、あるニューロンが発火するためには計算される必要があること、また、実際に計算の行われた(入力が変化した)ニューロンの数を調べる作業に手間がかかるためである。

このような負荷分散を用いて、NEC の並列計算機 Cenju-4 上で並列化を行った。本章ではその並列化方式について述べる。

3.1　Cenju-4

Cenju-4 は多数のプロセッサを多段結合網により接続した構造をしている。今回使用したシステムは 32 台のプロセッサをもつものである。

ネットワークは 4*4 のクロスバースイッチを多段結合することによりオメガネットを構成しており、マルチキャスト、ギャザーをネットワークで実現していることが特徴である。このような結合網のため、各プロセッサ間の通信時間に差はない。一段のレイテンシは 125nsec である。

Cenju-4 は SPMD(Single Program Multiple Data stream) 方式を採用し、分散メモリ、分散共有メモリの 2 つの環境を提供している。プログラミングライブラリとして、Cenju-4 上での並列プログラミングライブラリである Paralib/CJ4、Cenju-4 上の共有メモリライブラリ DSM/CJ4 の他に、MPI、PVM が用意されている。

今回は分散メモリ上で MPI を用いて並列化を行ったが、分散共有メモリ上での並列化、また Paralib/CJ4 などの他のプログラミングライブラリでの実行は、今後の課題として挙げられる。

3.1.1　データ通信

本シミュレーションにおけるデータ通信の指示は、MPI を用いて行われる。ブロードキャスト以外は、1 対 1 通信であるノンブロッキングバッファ通信`MPI_Ibsend`と`MPI_Irecv`を用いて行われる。通信がノンブロッキングであるため、送信は対応する受信が発行される前に完了し後続の命令を実行することができ、高速に動作することが可能である。1 対 1 の送信には、送り先のプロセッサを指定する必要がある。

3.2　並列処理方式

並列化に際し、各プロセッサに Time Table を 1 つずつ持たせる。しかし、Time Table の内容は全てのプロセッサで同じ訳ではなく、プロセッサによって Time Table に登録されている事象は異なる。それは、基本的に自分のプロセッサで計算され発火したニューロンの事象しか、自分の Time Table には登録されないためである。このことは、3.3で詳しく述べる。各プロセッサは自分の Time Table に登録されている事象についてのみ処理を行い、同期を取りながら時刻を進める。

各プロセッサで行われる処理は以下の通りである。

- 電位計算と発火判断及び事象の登録(2.2節)
- 負荷分散処理
- 各時刻における事象の送受信処理

本シミュレーションにおいて事象は、発火したニューロンとその出力先ニューロンの内で同一遅延のものの集合となっている。従って、ある 1 つのニューロンに繋っている複数のニュー

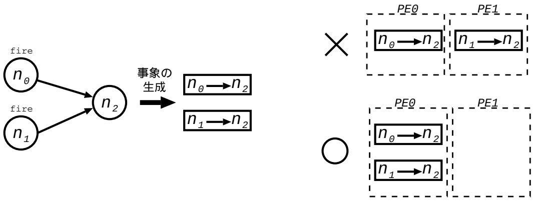


図 6: 事象の生成とプロセッサへの割り当て

ロンが同時に発火した場合、そのニューロンの計算を指示する事象は複数生成されることになる。(図 6左)

しかし、その処理 (ある 1 つニューロンにおける電位計算と発火判断) は 1 つのプロセッサにおいてなされなければならない。1 つのプロセッサで処理されない、すなわちある 1 つのニューロンの計算を指示する複数の事象を複数のプロセッサで処理した場合、その結果は同時に複数の入力がある状態を正しく反映しない。それは、個々のプロセッサではその処理すべき複数の事象の内の一部しか処理していない (図 6右) ためである。正しく反映するためのつじつまあわせの処理を挿入するならば、全体の処理が複雑になる。

従って、ある 1 つのニューロンの処理というものは 1 つのプロセッサでなされる必要があり、それはある 1 つのニューロンの計算を指示する全ての事象は同一プロセッサの Time Table に登録されている必要がある、ということを意味する。そのため、各プロセッサは自分の担当しているニューロンに関する事象全てを、他のプロセッサから集めなければならない。その処理が事象の送受信処理である。

各プロセッサで行う計算すなわち 2.2で述べた処理は、ニューロン単位の処理である。一方、事象はニューロンの結合の遅延が単位として構成されている。そのためこのような構成を採る限り、同じ事象に複数のニューロンの処理を含みそれらのニューロンは別々のプロセッサの担当である、という図 7のような状態が起こり得る。これが正しく処理されるためには、その事象を必要とする全てのプロセッサに登録されている必要がある。つまり、同じ事象のコピーが複数のプロセッサに存在することになる。

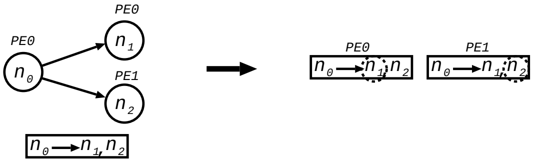


図 7: 同一事象が複数のプロセッサに登録されるケース

同じ事象を複数のプロセッサで持つと言っても、各プロセッサはその事象が表す全てのニューロンを計算する訳ではない。計算が行われるのは自分の担当しているニューロンだけである。自分の担当でないニューロンの計算を行っても意味がないばかりか、時間の無駄である。なぜなら、前述のように複数入力がある場合はそれに関連する事象が全て 1 ヶ所に集まっていなければならず、自分の担当でないニューロンに関する事象はその全てが集まっている訳ではないため、計算を行っても正しい結果を与えないからである。

このような事象の登録の規則が存在するため、事象の送受信処理が必要となる訳だが、一般的にこのような処理を行う場合は 2 つの方法が考えられるであろう。

1 つは事象を登録する際、その事象が処理されるべきプロセッサを調べてそのプロセッサに登録する方法である。この方法は動的に負荷分散をしない (各ニューロンを担当するプロセッサが変わらない) ならば、事象を登録しようとした時にそれが処理されるべきプロセッサが直ちに判明するため、問題はない。しかし動的に負荷分散を行った場合、登録時にはその事象が処理されるべきプロセッサが分からない可能性がある。負荷分散処理による各プロセッサの担当ニューロン変更に伴い、ある事象を処理すべきプロセッサも変更されるためである。このような場合の対処法として、負荷分散後に実行される事象はどこかに溜めておき、負荷分散が終わったら処理すべきプロセッサに登録する、という方法が挙げられる。しかし、この方法では事象の管理が複雑になってしまうため好ましく

ない。

もう1つの方法は、どのTime Tableでもよいから一旦登録をしておき、その事象を処理する際にどのプロセッサに登録しなければならないのか判断する方法である。この方法ならば動的負荷分散を用いた場合でも問題なく、事象の管理も容易である。従ってこちらの方法を採ることにする。

全てのニューロンは2.1.2の式より、現在の時刻、以前発火した時刻、以前パルスが印加された時刻、その時の電位から、現在の電位が計算できる。これらの値はシミュレーションを通して変わる値であり、現在の時刻以外の値はそのニューロンの計算を行ったプロセッサにおいて変更される。全プロセッサは全ニューロンのこれらのシミュレーションによって変わる値を知っている必要はなく、これらの情報はそのニューロンを担当しているプロセッサだけが知っていればよい。なぜなら、担当でないプロセッサにおいてはそのニューロンが計算されることはないからである。よって、担当するニューロンが変わった際に、これらのニューロン固有の情報を新しい担当プロセッサに送れば十分であることが分かる。

以上並列化の際に求められることをまとめると、次の様になる。

- ある1つのニューロンの処理は同一プロセッサで処理される必要がある。
- 従って、ある1つのニューロンの処理を表す事象が複数存在する時は、1つのプロセッサに集めるようにする。
- このことは同一の事象が複数のプロセッサに存在することを許す。
- ニューロン固有の情報は、担当するニューロンが変わった際すなわち負荷分散処理時に新しい担当プロセッサに送る。

以上の点を踏まえ、以下では本章の冒頭で述べた負荷分散を用いた並列処理方式について述べる。まずシミュレーションの流れについて説明を行い、各箇所における動作について述べた後、負荷分散法について述べる。

3.2.1 シミュレーションの流れ

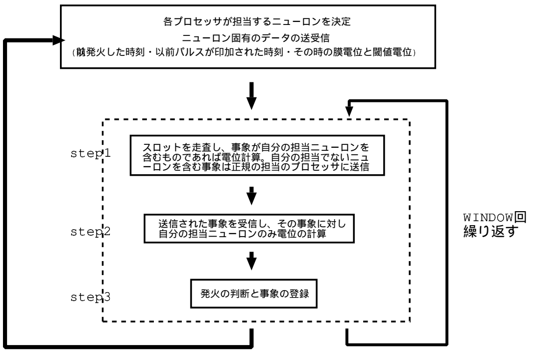


図 8: シミュレーションの流れ

図8にシミュレーションの流れを示す。

負荷分散のための操作はある程度の時間間隔を空けて行うようにする。なぜなら2.3で述べたように時間的局所性が存在するため、ごく短い時間においては発火が集中的に起こる場所にそれほど変化がない。従って負荷のばらつきもそれほど変化がないと考えられ、返って負荷分散をし直す処理に費す時間の方が多いだろう、と予想されるからである。

ここで言う「ある程度の時間」というのは、シミュレーション上では時間をステップ値として表現しているが、そのステップ値の一定の幅を意味する(例えば500ステップのシミュレーションにおいて5回負荷分散を行うのであれば「ある程度の時間」は100ステップとなる)。以下、この「ある程度の時間」をWINDOW幅と呼ぶことにする。WINDOW幅毎に[現時刻 − WINDOW幅]から現時刻までの発火履歴を用いて、各プロセッサが担当するニューロンを決める。

電位の計算と発火の判断の2つに分けたために、Time Tableのスロットを各時刻2回づつ走査する必要性があることは2.2で述べた。もし、事象の送受信操作のため(送る事象をチェックするため)スロットを1回走査するならば、電位計算で1回、発火判断で1回と計3回もスロットを走査することになる。登録されている事象の数が少なければ問題はないであろうが、数が多くなってくると3回もスロットを走査することが全シミュレーション時間に及ぼす影響は大

きくなる。そこで、電位計算のための走査に事象の送受信操作を埋め込むことにする。電位計算を行うためにスロットに登録されている事象を調べながら、他のプロセッサにも必要な事象はそれらのプロセッサにその事象のコピーを送る (step1)。受け取った事象の処理がまだ行われていないためこの処理 (step2) を行い、再度スロットを走査して現時刻に電位の計算が行われたニューロンに対し発火の判断を行う (step3)。

3.2.2　負荷分散の概念

今回は簡単のために 2 次元平面の分割として扱うことにする。

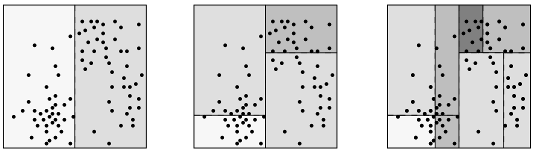

[2 PE]　[4 PE]　[8 PE]

図 9: 負荷分散の概念

発火履歴を用いた負荷分散の概念的な図を図9に示す。図中の黒点は、以前ニューロンの担当割り当てを行ってから WINDOW が経った現在までの間に発火したニューロンを示している。WINDOW 内に同一のニューロンが複数回発火した場合は、発火した回数分プロットされている。

このような有限個の点がプロットされた平面をプロセッサ台数分の領域に分割し、各領域の点の数が等しくなるようにしたい。そこでそのための計算に重心を用いることにする。

まず発火の重心により 2 つの領域に分割する。プロセッサ台数が 2 台より多いならば、各々の領域をそれぞれの重心によりさらに 2 つに (先程とは異なる軸方向で) 分割する。プロセッサが 8 台、16 台、…、の場合も以下同様にして領域の分割を行う。

3.3　実装

各 step における動作を図 10に示す。図中の drive1 は電位の計算処理、drive2 は発火の判断と事象の登録処理を表す。

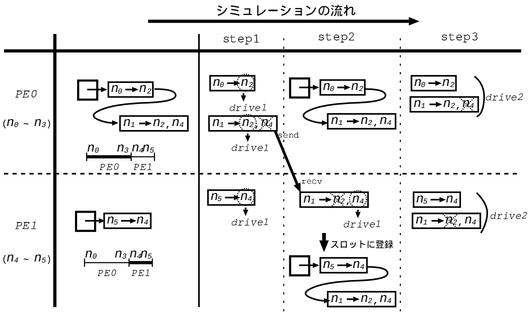


図 10: 各 step の動作

step1 では、電位計算と同時に前述の事象の送受信処理 (送信のみ) を行っている。図中の PE0 では 1 つめの事象 ($n_0 \to n_2$) は自分の担当ニューロンである n_2しか含んでいないため、他のプロセッサの Time Table に登録する必要はない。従って、n_2の電位計算をこのプロセッサで行うことで、この事象に対する処理は完了である。次の事象 ($n_1 \to n_2, n_4$) は、自分の担当ニューロンである n_2を含むが、同時に自分の担当でないニューロン n_4をも含んでいる。このような自分の担当でないニューロンをも指示する事象は、それを担当するプロセッサの Time Table のスロットにも登録される必要がある。なぜなら、このケースでは PE1 は n_4の計算を行う必要があり、現在その n_4には複数の入力 (n_1 と n_5から) がある状態で $n_5 \to n_4$という事象しか PE1 で処理されないならば、このような複数の入力がある状態を正しく反映しないからである。よって PE0 ではこの事象を発見した時点で、この事象のコピーを PE1 に送るようにする。

step2 では step1 で送信された事象に対する受信を行う。事象を受信した後、その事象が計算を指示するニューロンの内で自分の担当であるニューロンについてのみ電位計算を行う。図中の PE1 では PE0 から事象 ($n_1 \to n_2, n_4$) を受信している。そして、PE1 の担当であるニューロン n_4についてのみ、電位の計算を行う。その後、これらの新しく送られてきた事象を自分のスロットに登録する。

step3 では発火の判断と事象の登録を行う。この処理もやはり、スロットに登録されている事象の指すニューロンの内、自分が担当しているニューロンについてのみ行う。図中の PE0 においては n_2に対し、PE1 においては n_4に対し処理を行う。

発火判断の結果、膜電位が閾値電位を越えるすなわち発火した場合、2.2で述べたように発火したニューロンとその出力先ニューロンを同一遅延毎にまとめて、事象として登録しなければならない。現在用いている、処理時に事象の送受信を行う方法においては、どのプロセッサの Time Table に登録しても問題ない。なぜなら、その事象を処理する際に登録されるべき場所 (その事象が指すニューロンを担当するプロセッサ) に再登録されるからである。そのため各プロセッサは、最も速く登録できる自分の Time Table に登録するようにする。

また発火履歴をもとに負荷分散を行うためには、どこかに発火履歴を記録しておく必要がある。よって発火した際、後述する発火テーブルへの登録も行う。発火テーブルへの登録も、そのニューロンを担当しているプロセッサのテーブル対してのみ行われる。換言するならば、あるプロセッサの発火テーブルに登録されている情報は、そのプロセッサが担当しているニューロンの発火情報だけである。

3.3.1　負荷分散方式

図 11に負荷分散方式を示す。

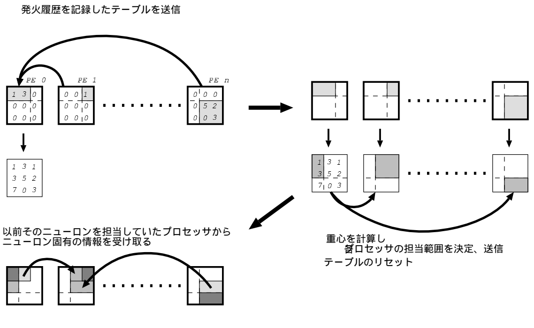


図 11: 負荷分散方式

各ニューロンは (colummn, rank, file) という 3 つの値により 3 次元空間内での位置を表現している。今回は問題を 2 次元として扱うため、(rank, file) で構成される 2 次元平面で考える。そこで、各プロセッサに rank と file を引数とした発火履歴を記録するための 2 次元テーブルを用意する。テーブルの各要素は、対応する (rank, file) のニューロンが WINDOW 内に発火した回数を示している。

3.3で述べたように、各プロセッサの発火テーブルには、そのプロセッサがそれまで担当していたニューロンの発火情報しか登録されていない。よって負荷分散を行うためには、まず全ニューロンの発火履歴を反映したテーブルを作成する必要がある。そこで全プロセッサは WINDOW 毎に、あるプロセッサ (ここでは PE0) に発火履歴を記録したテーブルを送信する。PE0 では送られてきたテーブルを 1 つのテーブルにまとめ、全ニューロンの発火履歴を反映したテーブルを作成する。

次に PE0 では、領域の分割を行う。領域の分割には重心を用いている。まず、前述の 1 つにまとめられたテーブルの全要素を調べ重心を計算し、求めた重心の値 (rank の値または file の値) を用いて 2 つの領域に分割する。プロセッサ台数が 2 台ならば、処理はここで終了する。そうでないならば先程 2 つに分割した各領域に対し、それぞれの重心を求め、今度は先程とは異なる方向で (先程重心の rank の値を用いて分割したのであれば今度は重心の file の値を用いて) それぞれの領域を分割する。以下プロセッサ台数が 8 台、16 台、…、の時も同様である。

このように領域の分割を進めているわけだが、このような分割方法ではプロセッサ台数が 2^n の時しか分割できない。例えば図 9のように点がプロットされた空間を 3 つの領域に分割しようとした場合、全ての領域で点の数がほぼ均等になるように分割するとアルゴリズムが複雑になるため、このような分割は行わない。従ってこの分割方式を用いた本シミュレータは、プロセッサ台数が 2^n の時にのみ対応してる。

さて、分割した領域を担当するプロセッサの決定方法であるが、PE0 では領域の分割と同時に、その領域を担当するプロセッサも決めている。その様子を図 12に示す。

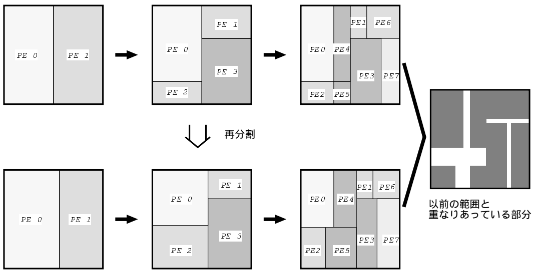


図 12: 領域分割と担当プロセッサの決定

最初に二分した際の領域の仮の担当を PE0、PE1 とする。次にその PE0、PE1 の担当することになっていた領域を二分した際、二分した領域の決まった一方 (図中では二分した領域の上の領域) を今まで担当とされていた PE0、PE1 の担当とし、残り領域の仮の担当をそれぞれ PE2、PE3 とする。さらに PE0、PE1、PE2、PE3 の持っている領域を二分する際には、やはり二分した領域の決まった一方 (図中では二分した領域の左の領域) を今まで担当することになっていた PE0、PE1、PE2、PE3 の担当とし、残りの領域 (二分した右の領域) をそれぞれ順に PE4、PE5、PE6、PE7 に割り当てる。以下この繰り返しにより、プロセッサ台数が 16 台、32 台の場合も同様に領域の分割と担当プロセッサを同時に決定している。この方法で再分割を行えば、図 12からも明らかなように、発火するニューロンの位置が大きく変わらない限り、新たに決定した領域が以前そのプロセッサが担当していた領域を多く含んでいることが分かる。ニューロン固有のデータの送受信は、負荷分散処理の結果、担当となるプロセッサが変更されたニューロンに対して行われる。従って、このような割り当てを用いて担当となるプロセッサが変更されるニューロンの数を抑えることにより、ニューロン固有のデータの転送量を削減することができる。

担当領域の計算を行っていた PE0 は担当領域の決定後、各プロセッサにそのプロセッサの担当領域だけを送信するのではなく、各プロセッサに全プロセッサの担当領域をブロードキャストする。これは各プロセッサが事象及びニューロン固有のデータを送信する際、3.1.1で述べたように送信先すなわち当該ニューロンを現在担当しているプロセッサがどのプロセッサであるかという情報が必要となるためである。

各プロセッサは今までの担当領域を保存し、新たな担当領域を PE0 から受け取り、担当領域の更新を行う。このように以前の担当領域を保存しておくのは、ニューロン固有データ送信の際に、送信先決定のためにその情報が必要となるからである。

その後、発火テーブルをリセットする。

全節の冒頭で述べた様に、負荷分散後はニューロン固有の情報の更新が必要となる。そこで各プロセッサは、自分の新しい担当領域と以前の担当領域に含まれる全てのニューロンについて、以下のことを行う。

- そのニューロンが新しい領域にも以前の領域にも含まれるならば、なにもしない。
- そのニューロンが新しい領域には含まれないが以前の領域には含まれるならば、新しくこのニューロンを担当することになったプロセッサに、このニューロン固有の情報を送信する。

各プロセッサは先程までそのニューロンを担当していたプロセッサからそのニューロンの以前発火した時の時刻、以前パルスが印加された時刻、その時の電位の値を受け取り、自分のプロセッサのそのニューロンの持つ各値を更新する。

4 評価

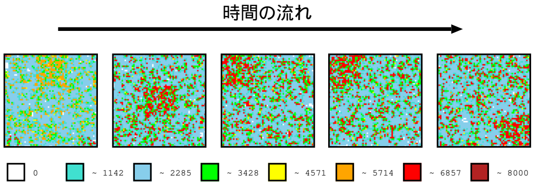


図 13: 発火の様子

column∗rank∗file を 4∗64∗64 としたネットワークでシミュレーションを行った。全接続枝数を約 100000、接続枝の重み (その枝を通って信

号が伝達される際の膜電位の変化量) をランダムに設定し、不応期は一律0に設定した。以下ではシミュレーションを行う時間のステップ値をEPOCHと呼ぶことにする。

図13のような発火パターンを示すニューラルネットを、前章で述べたシミュレータを用いて並列実行した。EPOCHを10000としてWINDOWを100に固定し、プロセッサ台数を変化させた。ニューロンの平均発火率は26.7%であった。そのときの実行結果を図14に示す。

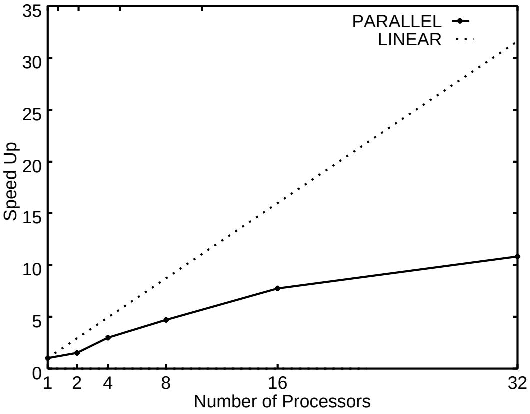


図 14: プロセッサ台数の変化による実行時間の速度向上率 (WINDOW=100)

グラフからもわかるようにプロセッサ32台を用いた時に、逐次タイプの約11倍の速度向上が得られた。

次にプロセッサ台数を32に固定し、WINDOWを変化させてみた。そのグラフを図15に示す。

WINDOW=0のときの値は、静的に(各プロセッサで担当するニューロンの数が等しくなるように) プロセッサの担当を決定した時の実行時間を示している。グラフからもわかるようにWINDOWが100の時が最速であるという結果が得られ、負荷分散を行わなかった時よりも約22%の速度向上が得られた。

理想としては、32台のプロセッサを用いた時に32倍の速度向上が得られることが望ましい。しかし図14が示すように、実際には11倍程度しか速度向上は成されていない。そこで、WINDOWを100に固定しプロセッサ台数を変化させた時の各箇所の時間の掛かり具合を調査した。その結果を表1に示す。

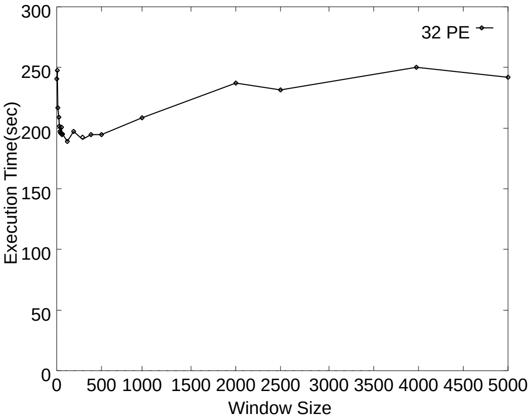


図 15: WINDOW 幅の変化による実行時間の変化 (PE=32)

負荷分散処理においては、全てのプロセッサから発火履歴を保存したテーブルを集める。よってプロセッサ台数が多くなれば当然通信も増えるため、処理に時間がかかるようになる。表1の負荷分散処理の欄にも、それは現れている。しかし、これはトータルのシミュレーション時間から見れば微々たるものである。

表より同期の待ち時間が、32PEともなるとかなりのウェイトを占めていることが分かる。これについて、各プロセッサにおいて処理されるニューロンの計算にばらつきがあるのではないかと考えた。そこで、各プロセッサがニューロンが計算をし判断を行った回数を、プロセッサ32台の場合について500EPOCH毎に調べた。

調査の結果、最悪の場合であるプロセッサが全ニューロン計算の6.7%のニューロン計算を請け負い、別のプロセッサでは1.5%のニューロン計算しか請け負っていないことが判明した。これは、その期間において大幅な発火の推移が起こったためであり、そのため以前の発火情報により割り振った担当では対応できなくなったことによる。よってニューロンの計算の負荷という観点から、これを如何に軽減するかという問題が課題として挙げられる。

理想としては表の各列が、発火の判断の列のようなプロセッサ台数が倍になれば所要時間が

表 1: 並列処理 (WINDOW=100 の時) における各箇所の平均所要時間 (単位 :sec)

	負荷分散処理	電位計算と事象の送信	事象の受信と電位計算	発火の判断	同期の待ち時間	その他 (初期設定など)
2PE	2.60	733.40	35.02	516.92	38.16	0.62
4PE	3.28	353.24	38.83	257.62	34.90	0.55
8PE	4.91	210.44	66.34	128.35	27.81	0.52
16PE	5.90	117.63	51.99	66.21	24.66	0.52
32PE	7.90	80.88	35.99	34.11	29.93	0.57

半減するという変化をすることが望ましい。発火の判断処理においては通信は行われていないため、純粋なニューロンの計算時間であると言える。一方、電位計算と事象の送信、事象の受信と電位計算の欄においては、純粋なニューロンの計算時間に、通信のための時間が加わってしまっている。

今回、計算を行うニューロンの数だけを負荷と考えて、負荷分散を考えた。そのため、通信量 (通信時間) の削減という点をあまり考慮せずに並列化を行った。その結果、通信を行っている処理 (特に受信処理) において、台数効果が得られることができていない。今後は、通信量という負荷も考慮に入れた負荷分散の手法を考える必要がある。

5　おわりに

本論では、神経網の離散事象シミュレーションの並列化を行い、その評価を行った。

まず、本シミュレーションにおいて採用されているパルス動作型ニューロンモデルについて述べた。神経網の離散事象シミュレーションについて説明を行い、それを並列化するにおいて問題となってくる点を、離散事象シミュレーションの代表例である論理回路シミュレーションと比較しながら明らかにした。それらの問題を踏まえながら、今回発火履歴をもとに発火の重心によって各プロセッサが担当するニューロンを決定する負荷分散法用いて並列化を行った。

このシミュレータを用いて評価を行った結果、発火率 26.7%で 32 台のプロッセッサを用いた時に約 11 倍の速度向上が得られた。また、発火履歴を用いた負荷分散法を用いず、静的に担当を割り振った場合と比較して、32 台のプロセッサを用いた時に約 22%の速度向上が得られることが判明した。さらにシミュレーションにおける各箇所の所要時間を測定し、今後の課題を提示した。

謝辞

本研究の機会を与えて下さり、適切な御指導を賜わりました富田眞治教授に深甚な謝意を表します。

また、貴重な御助言をいただいた森眞一郎助教授、五島正裕助手、津邑公暁氏に深く感謝致します。

さらに、日頃暖かく御鞭撻下さった京都大学工学部情報学科富田研究室の諸兄に感謝致します。

ありがとうございました。

参考文献

[1] 津邑公暁:神経網の高速シミュレーション手法, 修士論文 (1998)

[2] 伊藤崇之, 福島邦彦, 三宅誠:並列計算機による神経回路モデルの実現手法の検討-ネオコグニトロンの NCUBE 上での実現-, 電子情報通信学会論文誌,D-1,Vol.J73-D-1,No.1(1990)

[3] 池田孝利, 川人光男, 三宅誠, 乾敏郎, 淀川英司, 鈴木良次: 初期視覚の計算理論と神経回路による解法, 第2回生体・生理工学シンポジウム論文集,pp.137-140(Nov.1987)

[4] M.Drumheller and T.Pggio: On parallel stereo, Proc. Intl. Conf. on Robotics and Automation, IEEE(1986)

[5] 大森隆司:神経回路網シミュレーションのための並列計算機, 信学論(D),J71-D,8,pp.1461-1466(1988)