

# Introduction of High-Performance Computing for Neuroinformatics

山崎 匡\*

電気通信大学 大学院 情報理工学研究科

2017 年 11 月 21 日

## 概要

神経回路の数値シミュレーションは、大規模かつ精緻になるに従い、莫大な計算時間がかかるようになる。本チュートリアルでは、神経回路シミュレーションの基本事項をまずおさらいし、次いで様々な並列化によって計算を高速化する手法を学ぶ。単なる座学ではなく、実際に手を動かしてコードを書き実行させる、ハンズオンの形式を取る。

## 目次

1	はじめに	1
2	開会式	2
3	クラスタマシンへのログイン	2
4	一時間目: 神経回路シミュレーション事始め	2
4.1	ニューロン 1 個のシミュレーション	2
4.2	ニューロン 2 個のシミュレーション	4
4.3	ネットワークのシミュレーション	6
5	二時間目: OpenMP による計算の並列化	8
5.1	ランダムネットワークのシミュレーション	8
5.2	ニューロンの計算の並列化	9
6	閉会式	9

## 1 はじめに

生命維持から知的活動まで、脳は様々な機能を担っているが、その計算原理は未だに明らかになっていない。一方、脳の構造はそれに比べるとよく分かっており、ニューロンと呼ばれる神経細胞が複雑に繋がりあったネットワークである。一個のニューロンの挙動は具体的に数式で記述できるので、ニューロンの個数分そのような数式をプログラムし、コンピュータで数値シミュレーションを行うことで、原理的には脳の活動をコンピュータ上に再現することが可能になる。

ヒトの脳は約 1000 億個のニューロンからなると言われている。その全てを現実的な時間でシミュレートすることは、現在の最高性能のスパコンをもってしても難しい。しかしより小規模な動物の脳や、脳の一部を現実的な時間でシミュレートすることは十分可能になってきている。

---

\* Email: [aini17@numericalbrain.org](mailto:aini17@numericalbrain.org), Webpage: <http://numericalbrain.org/>

その際に本質的なのは、どのようにして計算を速くするか？ である。並列計算の技法を駆使することで、計算時間を数十倍から数百倍短縮することが可能になる。本チュートリアルではそのような手法を、典型的なランダムネットワークのシミュレーションを題材にして紹介する。

大体のスケジュールは以下の通り。

13:00–13:10 開会式  
13:10–13:30 クラスタマシンへのログイン  
13:30–14:30 一時間目: 神経回路シミュレーション事始め  
14:30–14:40 休憩  
14:40–15:40 二時間目: OpenMP による計算の並列化  
15:40–15:50 休憩  
15:50–16:50 三時間目: MPI による計算の並列化とハイブリッド並列  
16:50–17:00 閉会式

このチュートリアルは、文部科学省 ポスト「京」萌芽研究「思考を実現する」の協賛でお送りしています。

## 2 開会式

試合開始。まあ軽く自己紹介とか？

## 3 クラスタマシンへのログイン

各自 ssh の公開鍵を作って私に下さい。登録して引き替えにユーザ名をお渡しします。

公開鍵の作り方は、

```
$ ssh-keygen -t rsa
```

です。パスフレーズを決めて入力すると、`~/.ssh/id_rsa` と `~/.ssh/id_rsa.pub` ができるので、`id_rsa.pub` を下さい。

ユーザ名をもらったら、ログインしてみてください。マシン名は `pluto.sim.neuroinf.jp` です。

```
$ ssh pluto.sim.neuroinf.jp -l<username>
```

として、`<username>` のところに指定されたユーザ名を記載すれば、ログインできるはず。

## 4 一時間目: 神経回路シミュレーション事始め

### 4.1 ニューロン 1 個のシミュレーション

まず 1 個のニューロンのシミュレーションから始めよう。ニューロンの代表的なモデルは Hodgkin-Huxley モデルだが、本チュートリアルではより簡単な積分発火型モデル (Leaky integrate-and-fire model, LIF) を用いる。

カレントベースの LIF モデルは以下の微分方程式で記述される。

$$\begin{aligned}\tau \frac{dv}{dt} &= -(v(t) - V_{\text{leak}}) + I_{\text{ext}}, \\ v(t) > \theta &\Rightarrow \text{Spike}(t) = 1, v(t) \leftarrow V_{\text{reset}}, \\ v(0) &= V_{\text{init}}.\end{aligned}\tag{1}$$

ここで、 $v(t)$  は時刻  $t$  での膜電位、 $\tau$  ms は時定数、 $V_{\text{leak}}$  mV は静止電位、 $I_{\text{ext}}$  は外部電流、 $\theta$  mV はスパイク発射のための閾値、 $V_{\text{reset}}$  mV はリセット電位、 $V_{\text{init}}$  mV は膜電位の初期値である。

この微分方程式をコンピュータで数値的に解くために、差分方程式に変換する。具体的には十分短い時間間隔  $\Delta t$  を考え、

$$\frac{dv}{dt} \approx \frac{\Delta v(t)}{\Delta t} \quad (2)$$

と近似する。一方、 $v(t)$  を  $t$  の回りで  $\Delta t$  でテイラー展開すると、

$$v(t + \Delta t) = v(t) + \frac{dv}{dt} \Delta t + \frac{1}{2!} \frac{d^2v}{dt^2} \Delta t^2 + \dots \quad (3)$$

となり、 $\Delta t$  が十分小さいという仮定の下  $O(\Delta t^2)$  以降の項を無視すると

$$v(t + \Delta t) \approx v(t) + \frac{dv}{dt} \Delta t \quad (4)$$

となる。最後に上記 2 式を組み合わせると、

$$v(t + \Delta t) \approx v(t) + \Delta v(t) \quad (5)$$

となる。ここで

$$\Delta v(t) = \frac{\Delta t}{\tau} (- (v(t) - V_{\text{leak}}) + I_{\text{ext}}) \quad (6)$$

である。後は初期値  $v(0)$  さえ与えられれば、式 (5) を  $t = 0, \Delta t, 2\Delta t, \dots$  と逐次的に計算することで、 $v(t)$  の値を近似的に求めることができる。この数値解法には陽的オイラー法という名前がついている。

これを実際に試してみよう。次のコードを試す。

Listing 1 lif.c

```

1  #include<stdio.h>
2
3  #define TAU 20.0
4  #define V_LEAK -65.0
5  #define V_INIT (V_LEAK)
6  #define V_RESET (V_LEAK)
7  #define THETA -55.0
8  #define DT 1.0
9  #define T 1000.0
10 #define NT 1000 // ( T / DT )
11 #define I_EXT 12.0
12
13 void loop ( void )
14 {
15     double v = V_INIT;
16     for ( int nt = 0; nt < NT; nt++ ) {
17         printf ( "%f_%.f\n", DT * nt, v );
18         double i_ext = ( DT * nt < 100.0 || 900 < DT * nt ) ? 0 : I_EXT;
19         double dv = ( DT / TAU ) * ( - ( v - V_LEAK ) + i_ext );
20         v += dv;
21         if ( v > THETA ) {
22             printf ( "%f_0\n", DT * nt ); // print spike with membrane potential = 0 mV
23             v = V_RESET;
24         }
25     }
26 }
27
28 int main ( void )
29 {
30     loop ( );

```

```

31
32     return 0;
33 }

```

このコードでは、1000 ミリ秒 (=1 秒) 間のシミュレーションを  $\Delta t = 1$  ミリ秒で行う。100 ミリ秒から 900 ミリ秒までの間、外部電流として  $I_{\text{ext}} = 12$  nA を与える。このときの  $v(t)$  を、初期値  $v(0) = -65$  mV から  $\Delta t$  毎に逐次的に計算する。

コードの実行は 28 行目の `main` から始まり、関数 `loop` を実行するだけである (30 行目)。よって関数 `loop` (13–26 行目) がシミュレーションのコードそのものである。関数 `loop` の中身を詳しく見ていく。

15 行目で膜電位の変数 `v` を定義し、初期値として `V_INIT = -65` mV を代入する。`V_INIT` の定義は 5 行目である。

16 行目が時間に関するループである。シミュレートする時間を  $T = 1000$  ミリ秒間とし (9 行目)、それを  $\Delta t = 1$  ミリ秒の刻み (8 行目) で計算するので、ループの回数  $NT$  は  $NT = T/\Delta t = 1000$  回である。変数 `nt` を用意してカウントする。

17 行目で、今の時刻での  $v(t)$  の値を、時刻と共に表示する。

18 行目で、外部電流の値を設定する。3 項演算子を使って、100 ミリ秒から 900 ミリ秒までの間、`i_ext = I_EXT`、それ以外は 0 とする。`I_EXT` は 11 行目で定義されている。

19 行目で、式 (??) に従って  $\Delta v(t)$  を計算する。 $\tau$  は  $\text{TAU} = 20\text{ms}$  として 3 行目で定義されている。

20 行目で、式 (5) に従って  $v(t)$  を更新する。

21–24 行目はスパイク発射の判定である。もし  $v(t)$  が閾値 `THETA` を越えていたら (21 行目)、膜電位として 0 mV をその時刻と共に表示し (22 行目)、 $v(t)$  を `V_RESET` にセットする。`THETA = -55` mV は 7 行目で、`V_RESET = -65` mV は 6 行目でそれぞれ定義されている。

このコードをコンパイルして実行してみよう。コンパイルは以下のようにする。

```
[tyam ~/tutorial]$$ gcc -Wall -o lif lif.c
```

`-Wall` オプションは、全ての警告を表示するもので、超推奨。正常にコンパイルできると実行ファイル `lif` ができるので、以下のように実行する。

```
[tyam ~/tutorial]$ ./lif
```

実行すると、数字がどばつと表示されたと思うが、それが各時刻とその時の  $v(t)$  の値である。数字を眺めても何もわからないので、以下のようにリダイレクトしてファイルに出力し、

```
[tyam ~/tutorial]$$ ./lif > lif.dat
```

`gnuplot` で表示する。

```

[tyam ~/tutorial]$ gnuplot
      G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'lif.dat' with line

```

すると、図 1 のような膜電位の表示が得られるはずである。100–900 ミリ秒の間、一定の間隔でスパイクを発射している様子が確認できた。

ここで注意！ パラメータの単位には気をつけること。例えばもしこのプログラムで時間をミリ秒ではなく秒にして、 $T = 1.0$ ,  $\Delta t = 0.001$  とすると、正しい計算が行われない。このプログラムのように `Physiological Unit` を使うか、あるいは `SI Unit` を使うか、どちらかにすること。

## 4.2 ニューロン 2 個のシミュレーション

一番簡単なネットワークはニューロン 2 個からなるものなので、次はそれを作ろう。

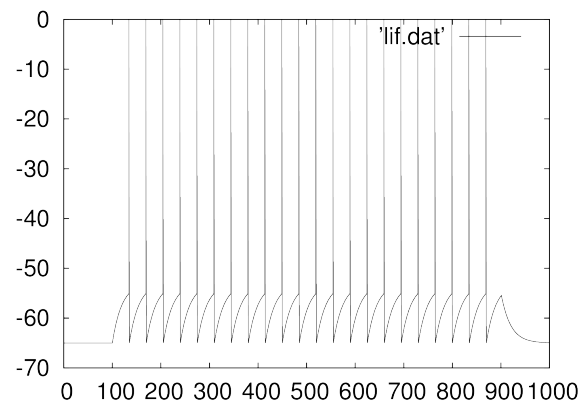


図1 1個のニューロンの膜電位のプロット

ニューロン同士がシナプスで繋がっておらず、完全に独立な場合は、lif.cをベースにしてほとんど自明に書ける。具体的には変数  $v$ ,  $dv$  を配列にして変数  $v[2]$ ,  $dv[2]$  とすれば良い。ただしそれだけでは完全に同じ計算をするだけなので、 $v$  の初期値を片方は 10 mV 下げよう。コードは以下になる。

Listing 2 lif2.c

```

1  #include<stdio.h>
2
3  #define TAU 20.0
4  #define V_LEAK -65.0
5  #define V_INIT (V_LEAK)
6  #define V_RESET (V_LEAK)
7  #define THETA -55.0
8  #define DT 1.0
9  #define T 1000.0
10 #define NT 1000 // ( T / DT )
11 #define I_EXT 12.0
12
13 void loop ( void )
14 {
15     double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
16     double i_ext = I_EXT;
17     for ( int nt = 0; nt < NT; nt++ ) {
18         printf ( "%f_0%f\n", DT * nt, v [ 0 ], v [ 1 ] );
19         double dv [ 2 ];
20         dv [ 0 ] = ( DT / TAU ) * ( - ( v [ 0 ] - V_LEAK ) + i_ext );
21         dv [ 1 ] = ( DT / TAU ) * ( - ( v [ 1 ] - V_LEAK ) + i_ext );
22         v [ 0 ] += dv [ 0 ];
23         v [ 1 ] += dv [ 1 ];
24         if ( v [ 0 ] > THETA ) {
25             printf ( "%f_0%f\n", DT * nt, v [ 1 ] ); // print spike with membrane potential = 0 mV
26             v [ 0 ] = V_RESET;
27         }
28         if ( v [ 1 ] > THETA ) {
29             printf ( "%f_1%f\n", DT * nt, v [ 0 ] ); // print spike with membrane potential = 0 mV
30             v [ 1 ] = V_RESET;
31         }
32     }
33 }
```

```

34
35 int main ( void )
36 {
37     loop ( );
38
39     return 0;
40 }

```

コードの変更点は以下の通りである。 $v$  を配列にし (15 行目)、初期値を変更した。外部電流は 0 ミリ秒から入れることとした (16 行目)。 $dv$  も配列にした (19 行目)。 $dv$ ,  $v$  の計算は添字を変えて 2 回計算した (20-23 行目)。閾値を超えたときの表示の仕方を変えた (25,29 行目)。

本来であれば 2 個のニューロンの計算は `for` ループで回すべきであるが、自明さを示すためにわざとループで書かなかった。

これをコンパイルして実行し、結果をプロットする。

```

[tyam ~/tutorial]$$ gcc -Wall -o lif2 lif2.c
[tyam ~/tutorial]$ ./lif2 > lif2.dat
[tyam ~/tutorial]$ gnuplot
      G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'lif2.dat' using 1:2 with line, 'lif2.dat' using 1:3 with line

```

図 2 のような膜電位のプロットが得られるはずである。初期状態が異なるのでスパイクのタイミングはズれるが、その他は同じなので同じ波形がシフトするだけとなる。

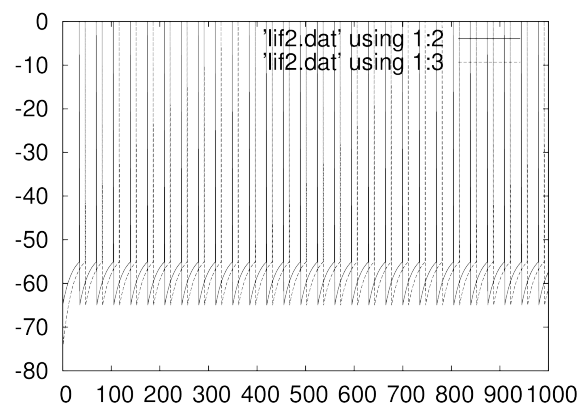


図 2 2 個の独立なニューロンの膜電位のプロット

### 4.3 ネットワークのシミュレーション

一時間目の最後に、2 個のニューロンをシナプスで結合してちゃんとしたネットワークにしよう。ここでは一番簡単な exponential synapse を導入する。

$$\tau_{\text{syn}} \frac{dg_i}{dt} = -g_i(t) + w \cdot \text{Spike}_{(i+1)/2}(t) \quad (7)$$

膜電位の式の右辺にシナプス後電位  $g(t)$  を追加する。ここで、 $i$  はニューロンの番号 ( $i \in \{0, 1\}$ )、 $\tau_{\text{syn}}$  は時定数、 $g_i(t)$  はシナプス後電位、 $w$  は結合重み、 $\text{Spike}_{(i+1)/2}(t)$  はもう片方のニューロンのスパイク発射 (0 または 1) を表

す。これも同様に差分化し、陽的オイラー法で解く。以下のようにすればよい。

$$g_i(t + \Delta t) = g_i(t) + \frac{\Delta t}{\tau_{\text{syn}}} \left( -g_i(t) + w \cdot \text{Spike}_{(i+1)/2}(t) \right) \quad (8)$$

ただし、初期値  $g_i(0)$  は 0 mV とする。この  $g_i(t)$  を、膜電位の式の右辺に追加する。

Listing 3 lif2net.c

```

1  #include<stdio.h>
2
3  #define TAU 20.0
4  #define V_LEAK -65.0
5  #define V_INIT (V_LEAK)
6  #define V_RESET (V_LEAK)
7  #define THETA -55.0
8  #define DT 1.0
9  #define T 1000.0
10 #define NT 1000 // ( T / DT )
11 #define I_EXT 12.0
12 #define TAU_SYN 5.0
13 #define W 10.0 //-10.0
14
15 void loop ( void )
16 {
17     double v [ 2 ] = { V_INIT, V_INIT - 10. };
18     double i_ext = I_EXT;
19     double g [ 2 ] = { 0., 0. };
20     int spike [ 2 ] = { 0, 0 };
21     for ( int nt = 0; nt < NT; nt++ ) {
22         printf ( "%f_0%f\n", DT * nt, v [ 0 ], v [ 1 ] );
23         double dv [ 2 ];
24         dv [ 0 ] = ( DT / TAU ) * ( - ( v [ 0 ] - V_LEAK ) + g [ 0 ] + i_ext );
25         dv [ 1 ] = ( DT / TAU ) * ( - ( v [ 1 ] - V_LEAK ) + g [ 1 ] + i_ext );
26         double dg [ 2 ];
27         dg [ 0 ] = ( DT / TAU_SYN ) * ( - g [ 0 ] + W * spike [ 1 ] );
28         dg [ 1 ] = ( DT / TAU_SYN ) * ( - g [ 1 ] + W * spike [ 0 ] );
29         v [ 0 ] += dv [ 0 ];
30         v [ 1 ] += dv [ 1 ];
31         g [ 0 ] += dg [ 0 ];
32         g [ 1 ] += dg [ 1 ];
33         spike [ 0 ] = ( v [ 0 ] > THETA );
34         if ( v [ 0 ] > THETA ) {
35             printf ( "%f_0%f\n", DT * nt, v [ 1 ] ); // print spike with membrane potential = 0 mV
36             v [ 0 ] = V_RESET;
37         }
38         spike [ 1 ] = ( v [ 1 ] > THETA );
39         if ( v [ 1 ] > THETA ) {
40             printf ( "%f_0%f\n", DT * nt, v [ 0 ] ); // print spike with membrane potential = 0 mV
41             v [ 1 ] = V_RESET;
42         }
43     }
44 }
45
46 int main ( void )
47 {
48     loop ( );

```

```

49
50     return 0;
51 }

```

コードの変更点は以下の通りである。

これをコンパイルして実行し、結果をプロットする。

```

[tyam ~/tutorial]$$ gcc -Wall -o lif2net lif2net.c
[tyam ~/tutorial]$ ./lif2net > lif2net.dat
[tyam ~/tutorial]$ gnuplot
      G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'lif2net.dat' using 1:2 with line, 'lif2net.dat' using 1:3 with line

```

図3のような膜電位のプロットが得られるはずである。今度はスパイクのタイミングが徐々に揃って行くことがわかる。

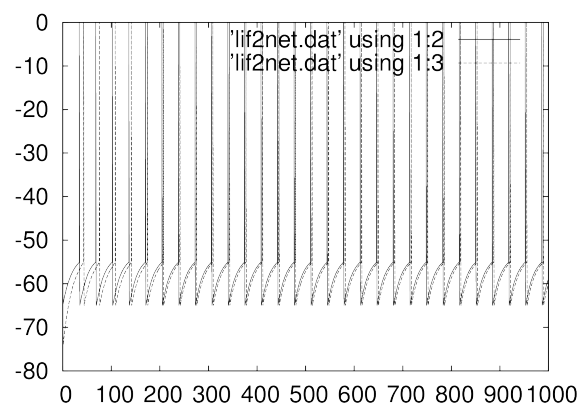


図3 互いに興奮性で接続された2個のニューロンの膜電位

#### 課題 1.

互いを抑制性で繋ぐと何が起こるか試して確認せよ。具体的には  $W$  の値の符号を負にすればよい。

#### 課題 2.

なぜこうなるのかを考察せよ。

一時間目はここまで。休憩！

## 5 二時間目: OpenMP による計算の並列化

### 5.1 ランダムネットワークのシミュレーション

一時間目にやった2個のニューロンからなるネットワークは小さすぎて、計算があっという間に終わってしまった。これでは面白くないので、もう少し大きなネットワークを考えよう。具体的には4000個のニューロンを4:1で興奮:抑制に振り分け、確率  $p = 0.02$  でランダムに結合させた、ランダムネットワークを考える。このネットワークは様々な神経回路シミュレータのベンチマークとしても利用されている、スタンダードなものである。



コードは付録にある。コンパイルして実行すると、`spike.dat` というファイルが生成されて、`gnuplot` で表示すると図??のようなラスタプロットが得られる。4000 個のニューロンの膜電位を一度にプロットしてもまともに見えないので、以降はこのようにスパイクだけをプロットする。

コードの概要は以下の通りである。

```
1 void loop ( void )
2 {
3     for ( int nt = 0; nt < NT; nt++ ) {
4         for ( int i = 0; i < N; i++ ) {
5             calculateSynapse ( i );
6             updateMembranePotential ( i );
7         }
8         outputSpike ( nt );
9     }
10 }
```

時間に関するループ (3 行目) はこれまで通り。ニューロンの数が増えたので、今回からちゃんとループにする (4 行目)。ループの中では、まずシナプス入力 の計算をし (5 行目)、ついで膜電位の値を更新する (6 行目)。ニューロンの計算が終わったら、スパイクの情報をファイルに出力する (8 行目)。関数 `calculateSynapse` および `updateMembranePotential` の中身はご想像の通りである。

本チュートリアルで使うクラスタマシンを普通に使って計算すると、1 回のシミュレーションに XXX 秒かかる。各自試してみよ。

## 5.2 ニューロンの計算の並列化

# 6 閉会式

試合終了。乙。