スパコンで脳を再現する *

山﨑 匡 †
電気通信大学 大学院 情報理工学研究科

2018 年 3 月 28, 29 日

概要

脳の仕組みはまだよく分かっていませんが、その構造は非常によく分かっています。脳はニューロンと呼ばれる神経細胞が複雑に繋がったネットワークであり、一つのニューロンのことは数式で書けます。よってそのような式をニューロンの数だけ記述し、プログラムしてスパコンで計算することで、原理的には脳の活動をスパコン上で再現できます。本テーマでは実際にスパコンを使って、そのような計算を行います。

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ますが、その計算原理は未だに明らかになっていません。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す。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ります。

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ます。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まだ難しいです [1, 2]。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ます。

また、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？ です。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ります。

そこで、本テーマでは、(1) 単一ニューロンのシミュレーションからネットワークのシミュレーションまでを実際にコードを書いて試す、(2) さらに高性能計算の手法を駆使して計算を高速に実行する、の 2 つを行います。

大体のスケジュールは以下の通り。

13:00–13:30　開会式＆研究室に移動
13:30–13:45　自己紹介
13:45–14:00　クラスタマシンへのログイン
14:00–14:50　一時間目: 高校生のための計算神経科学入門
14:50–15:00　休憩
15:00–15:50　二時間目: 神経回路シミュレーション事始め
15:50–16:00　休憩
16:00–16:50　三時間目: ランダムネットワークの並列計算
16:50–17:15　まとめ＆クラスタマシン見学＆閉会式へ移動
17:15–17:45　閉会式

このスプリングスクールは、文部科学省 ポスト「京」萌芽的課題 4 思考を実現する神経回路機構の解明と人工知能

* 当日配布版。事前配布版から大幅に加筆修正してあり、これなら読めると思います。

† Email: blsc-ss17@numericalbrain.org, Webpage: http://numericalbrain.org/

への応用「脳のビッグデータ解析、全脳シミュレーションと脳型人工知能アーキテクチャ」[*1]、文部科学省 高性能汎用計算機高度利用事業「ヘテロジニアス・メニーコア計算機による大規模計算科学」[*2]、ならびにニューロインフォマティクス国際統合機構日本ノード事業「Simulation Platform の開発」[*3]の協賛でお送りしています。

2 クラスタマシンへのログイン

まず最初にクラスタマシンのアカウントをお配りします。

1 人 1 台ずつ MacBook Air を用意しました。すでに利用可能になっていて、1 つ大きなウィンドウが開いていると思います。これを端末エミュレータと呼びます。この端末エミュレータに様々な命令を入力して、計算機に指示を与えます。

まずその Mac から、今回使うクラスタマシンにログインします。やりかたは、

```
user@MacBookAir$ ssh -Y <hostname> -l guestXX
```

です。name@MacBookAir$はプロンプトと呼ばれる部分で、入力する部分ではありません。ssh 以降を入力して下さい。入力したらリターンキーを押すのを忘れずに。

<hostname>というのが、このスプリングスクールで使うクラスタマシンの名前で、guest##というのが最初にお配りした自分のユーザ名です。うまくログインできると、

```
user@MacBookAir$ ssh -Y <hostname> -l guest##
:
guest##@node00:~$
```

となり、新しいプロンプトが表示されます。これでクラスタマシンを使う準備が整いました。例えば sl と入力してみてください。

```
guest##@node00:~$ sl
```

蒸気機関車はちゃんと走って行ったでしょうか？

3 一時間目: 高校生のための計算神経科学入門

3.1 ニューロンの計算

脳はニューロンと呼ばれる神経細胞が複雑に繋がったネットワークである。ニューロンのネットワークなので神経回路 (ニューラルネットワーク) と呼ぶ (図 1)。

ニューロンは、(1) 他のニューロンから入力をもらう樹状突起、(2) 入力を加算する細胞体、(3) 他のニューロンに出力をする軸索、からなる。ニューロンは電気的素子であり、膜電位と呼ばれる電圧のパラメータを持っていて、その値を上下させる。

ニューロン同士の情報のキャリアは、スパイクと呼ばれる短い電気パルスである。送り手のニューロンの軸索は、受けてのニューロンの樹状突起にシナプスという構造を介して結合している。スパイクがシナプスに到達すると、受け手のニューロンの膜電位が変化する。送り手のニューロンには興奮性・抑制性の 2 種類があり、興奮性 (抑制性) ニューロンからのスパイクは受け手の膜電位を上昇 (下降) させる。何発かの興奮性スパイクが同時に受け手のニューロンの到達すると、受け手のニューロンの膜電位が大きく上昇し、閾値を超える。膜電位が閾値を超えると受け手のニューロンはスパイクを発射し、次のニューロンへと情報を伝える。

[*1] https://brain-hpc.jp/

[*2] https://hetero-manycore.riken.jp/symposium2018/

[*3] https://sim.neuroinf.jp/

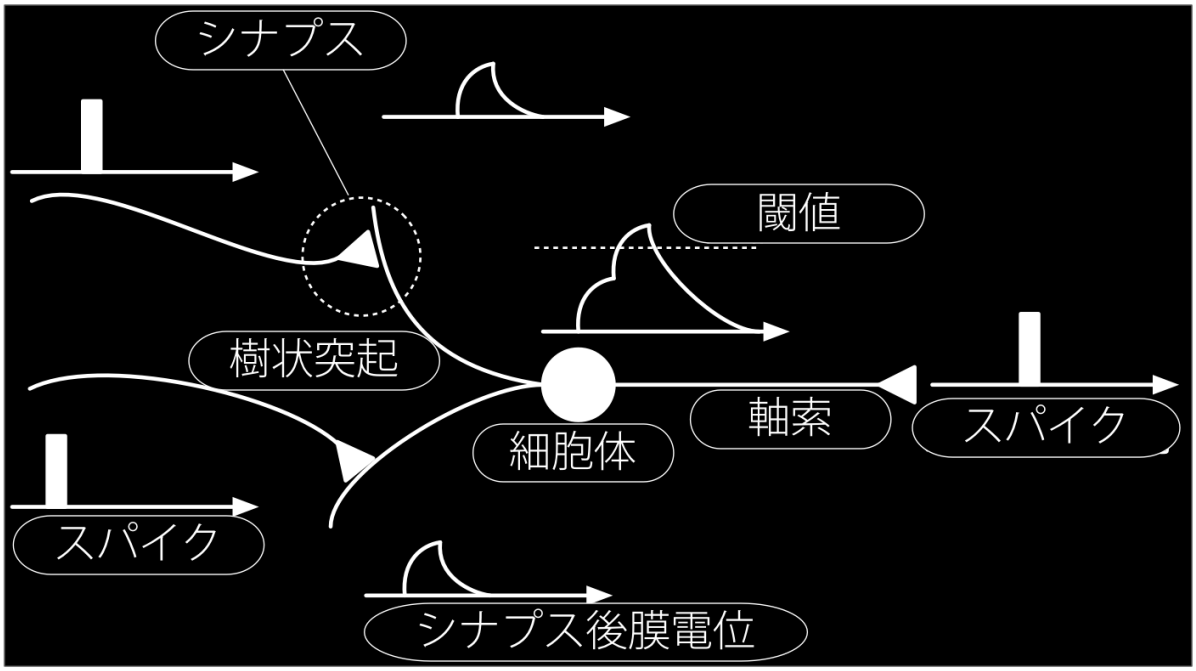


図 1　ニューラルネットワークの模式図

ニューロンがやっていることは、原理的にはこれだけである[*4]。後はたくさんのニューロンがネットワークで繋がったときに、どのように振る舞うのか？ が大事になってくる。

ニューロンの計算もシナプスの計算も、原則として数式[*5]で記述でき、それを解くことで、脳全体のネットワークの計算が (原理的には[*6]) 可能になる。その数式の解き方をまず学ぼう。

3.2　まずは: 高校物理のおさらい

脳の活動は一つの物理的な現象なので、まずは超簡単な物理のおさらいから始めよう。

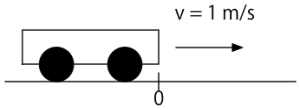
[*4] もちろん本当はこれだけでは無い

[*5] 微分方程式 (後述)

[*6] この原理的には、っていう物の言い方は注意が必要で、可能か不可能かで言えば可能だけど、実際のところは非常に非常に困難である、という意味。

問題 1

時刻 0 秒で原点 ($x = 0$) に静止している車が、速度 $v = 1$ メートル/秒で右に移動を開始した (以下の図)。



車の位置のプロット。横軸が時間 (秒)、縦軸は原点からの距離 (メートル) である

問 a　移動を開始してから 1 秒後の位置 $x(1)$ を答えよ。
問 b　2 秒後、3 秒後、$\cdots$、a 秒後の位置 $x(2), x(3), \cdots, x(a)$ を答えよ。

解答:

問 a　$x(1) = 1$ メートル
問 b　$x(2) = 2$ メートル, $x(3) = 3$ メートル, $\cdots$, $x(a) = a$ メートル

問題 2

右に一定速度 v メートル/秒で走っている車があり、時刻 1 秒で原点から位置 $x(1)$ メートル、1 秒後の時刻 2 秒で位置 $x(2)$ メートルにいたとする。v を x で表せ。
解答: $v = (x(2) - x(1))/1$ メートル/秒

問題 3

右に一定速度 v メートル/秒で走っている車があり、時刻 t 秒で原点から位置 $x(t)$ メートル、Δt 秒後の時刻 $t + \Delta t$ 秒で位置 $x(t + \Delta t)$ メートルにいたとする。v を x で表せ。
解答: $v = (x(t + \Delta t) - x(t))/\Delta t$ メートル/秒

ここまではいいですかね。では次。

問題 4

問題 3 の設定の元で、$x(t + \Delta t)$ を $x(t)$, v, Δt で表せ。
解答: $v = (x(t + \Delta t) - x(t))/\Delta t$ より、$x(t + \Delta t) = x(t) + v \times \Delta t$ ($\times$ は通常のかけ算の記号)。

この最後の式:

$$x(t + \Delta t) = x(t) + v \times \Delta t \tag{1}$$

がとっても重要で、右辺第一項の今の位置 $x(t)$ と、右辺第二項に含まれる今の速度 v がわかれば、Δt 後の位置 $x(t + \Delta t)$ (左辺) が計算できる、ということ表している。同様にして、そのさらに次の時刻での位置 $x(t + 2\Delta t)$ も

$$x(t + 2\Delta t) = x(t + \Delta t) + v \times \Delta t \tag{2}$$

として計算できる。要するに最初の位置 $x(0)$ と v が決まれば、その先の未来の位置は全て順番に計算できる。

余談: もちろん v は定数である必要は無く、一般に $v(t)$ として時間的に変化しても良い。

さて、これを実際にプログラムを組んで試してみよう。`car.c` というプログラムが準備されている。エディタを起動して中身を確認してみよう。エディタというのは、プログラムを見たり書いたり書き直したりするために使うアプリ。

端末エミュレータで

```
tyam@node00:~$ nano car.c
```

と入力する。プログラムはこんな風になっている。

ソースコード 1 car.c

```
1  #include<stdio.h>
2
3  int main ( void )
4  {
5    double t = 0; // 時刻: 最初は 0秒から
6    double x = 0; // 位置: 最初は原点（0メートル）から
7    double v = 1.0; // 速度: 1メートル/秒
8    double dt = 1.0; // 時間の刻み幅: 1秒ずつ進める
9
10   while ( t < 10.0 ) { // 10 秒間繰り返し
11
12     printf ( "%f␣%f\n", t, x ); // 今の時刻と位置を表示
13
14     x = x + v * dt; // 次の時刻の位置を計算
15     t = t + dt; // 時間をdt 秒進める
16   }
17
18   return 0;
19 }
```

C 言語がわからなくても心配不要。そんな大したことはしていない。

- 1–4 行目: おまじない[*7]。
- 5–8 行目: 変数の設定。時間 (t)、位置 (x)、速度 (v)、時間のステップ幅 (Δt) をそれぞれ変数 t, x, v, dt に保存することとし、初期値を代入しておく。時間は 0 秒から、位置は原点から 0 メートルから、速度は 1 メートル/秒、ステップ幅は 1 秒とした。
- 10–16 行目: 実際の計算部分。
  - 10 行目: 時間が 10 秒進むまで繰り返す。
  - 12 行目: 今の時刻と位置を画面に表示する。
  - 14 行目: 式 (1) に従って次の時刻の位置を計算する。
  - 15 行目: 時間を 1 秒進める。
  - 16 行目: 10 行目に戻る。
- 17–19 行目: おまじない。

中身を確認したら、エディタを終了して端末エミュレータに戻ろう。^X(コントロールキーを押しながら x を押す)で戻る。戻ったら、このプログラムをコンパイル[*8]して、実際に走らせてみよう。コンパイルの仕方はこう:

```
guest##@node00:~$ gcc -std=c99 -O3 -o car car.c
```

gcc というのはコンパイラの名称。-std=c99 はどういう C 言語の仕様に従うか示すおまじない。-O3 は最適化オプションというおまじないで、これをつけると速く計算が終わるようになる。-o car はコンパイル結果である実行形式のファイル名で、car.c が今のプログラムのファイル名。

何もメッセージが表示されなければ正常にコンパイルできた証。実行してみよう。

```
guest##@node00:~$ ./car
0.000000 0.000000
1.000000 1.000000
2.000000 2.000000
3.000000 3.000000
```

[*7] C 言語の規則に従って書く部分であり、計算の本質ではない。
[*8] プログラムのテキストを計算機が解釈・実行できるようにするための操作。

```
4.000000 4.000000
5.000000 5.000000
6.000000 6.000000
7.000000 7.000000
8.000000 8.000000
9.000000 9.000000
tyam@node00:~$
```

./はおまじないだと思って。この通り、左側に時刻 t、右側に位置 $x(t)$ の値が計算され表示される。この場合単位はそれぞれ秒とメートルね。

このデータをグラフにして表示してみよう。以下のようにすると:

```
tyam@node00:~$ ./car > car.dat
```

数値データをファイル car.dat に出力してくれる。これを、

```
guest##@node00:~$ gnuplot
       G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'car.dat'
```

とすると、図 2 のようなグラフが表示される。ちゃんと表示されましたかね。ここまでできたらニューロンの計算も

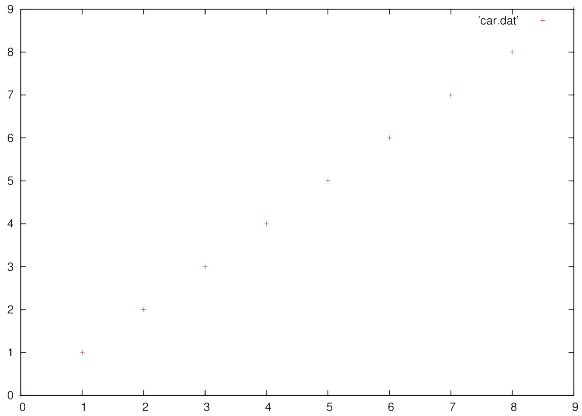


図 2　車の位置のプロット。横軸が時間 (秒)、縦軸は原点からの距離 (メートル) である

(一応は) できる。

もし時間があったら、速度 v を t にしてみよう。これは加速度 1 $\mathrm{m/s^2}$ の等加速度運動に対応する。

最後にもう一つだけ。

$$\frac{x(t+\Delta t)-x(t)}{\Delta t} \tag{3}$$

は、Δt を十分小さく取ったときに $\dfrac{dx}{dt}$ と書く。これを x の (時間に関する) 微分と言う。もちろん $v(t)=\dfrac{dx}{dt}$ である。微分を含む方程式を微分方程式と呼び、様々な現象が微分方程式で記述できる。

一時間目はここまで。休憩！

4 二時間目: 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 [3]。ニューロンの代表的なモデルは Hodgkin-Huxley モデルだが、本スプリングスクー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 モデルは次の式で記述される。

$$\tau \frac{dv}{dt} = -\left(v(t) - V_{\mathrm{leak}}\right) + RI_{\mathrm{ext}}(t), \tag{4}$$

$$v(t) > \theta \Rightarrow \mathrm{Spike\ (t)} = 1, v(t) \leftarrow V_{\mathrm{reset}}, \tag{5}$$

$$v(0) = V_{\mathrm{init}}. \tag{6}$$

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

式 (4) が基本的な膜電位のダイナミクスを記述する。V_{leak} を平衡点とし、外部入力 $RI_{\mathrm{ext}}(t)$ に時定数 τ で漸近する挙動を示す。式 (5) はスパイク発射の条件である。膜電位が閾値を超えると、その時刻でスパイクを発射したものとし (Spike (t) = 1)、かつ膜電位をリセットする。式 (6) は膜電位の初期値を与える。

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 Δt を考え、

$$\frac{dv}{dt} \approx \frac{\Delta v(t)}{\Delta t} \tag{7}$$

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

$$v(t + \Delta t) = v(t) + \frac{dv}{dt}\Delta t + \frac{1}{2!}\frac{d^2v}{dt^2}\Delta t^2 + \cdots \tag{8}$$

となり、Δt 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

$$v(t + \Delta t) \approx v(t) + \frac{dv}{dt}\Delta t \tag{9}$$

となる。最後に上記 2 式を組み合わせると、

$$v(t + \Delta t) \approx v(t) + \Delta v(t) \tag{10}$$

となる。ここで

$$\Delta v(t) = \frac{\Delta t}{\tau}\left(-\left(v(t) - V_{\mathrm{leak}}\right) + I_{\mathrm{ext}}\right) \tag{11}$$

である。後は初期値 $v(0)$ さえ与えられれば、式 (10) を $t = 0, \Delta t, 2\Delta t, \cdots$ 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。

ソースコード 2 neuron1.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 R 1.0
```

```
 9  #define DT 1.0
10  #define T 1000.0
11  #define I_EXT 12.0
12
13  void loop ( void )
14  {
15    double t = 0;
16    double v = V_INIT;
17    int spike = 0;
18
19    while ( t < T ) {
20      printf ( "%f␣%f\n", t, (spike ? 0.0 : v ) );
21      double dv = ( DT / TAU ) * ( - ( v - V_LEAK ) + R * I_EXT );
22      spike = ( v > THETA ) ? 1 : 0;
23      v = ( v > THETA ) ? V_RESET : v + dv;
24      t = t + DT;
25    }
26  }
27
28  int main ( void )
29  {
30    loop ( );
31    return 0;
32  }
```

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。外部電流として $I_{\rm ext} = 12$ nA を与える。膜抵抗は簡単のために 1 MΩ とする。このときの $v(t)$ を、初期値 $v(0) = -65$ mV から Δt 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 行目で時間の変数 `t`、16 行目で膜電位の変数 `v` を定義し、初期値として `t=0` ミリ秒, `V_INIT`$= -65$ mV を代入する。`V_INIT` の定義は 5 行目である。

19 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 (10 行目)、それを $\Delta t =$ `DT` $= 1$ ミリ秒の刻み (24 行目) で計算する。

20 行目で、今の時刻での $v(t)$ の値を、時刻と共に表示する。

21 行目で、式 (11) に従って $\Delta v(t)$ を計算する。τ は `TAU` $= 20$ms として 3 行目で、`R` $= 1$ MΩ は 8 行目でそれぞれ定義されている。

22 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、変数 `spike` に 1 を、そうでなければ 0 をセットする。この書き方は三項演算子という。`THETA` $= -55$ mV は 7 行目で定義されている。

23 行目は膜電位の更新である。もし $v(t)$ が閾値 `THETA` を越えていたら、$v(t + \Delta t)$ として `V_RESET` を代入し、そうでなければ $v(t + \Delta t) = v(t) + \Delta v(t)$ とする。`V_RESET` $= -65$ mV は 6 行目で定義されている。

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。

```
guest##@node00:~$ gcc -std=c99 -O3 -o neuron1 neuron1.c
```

`-std=c99` オプションは、C99 の仕様でコンパイルするもの。`-O3` は一番エグい最適化をするオプション。正常にコンパイルできると実行ファイル `neuron1` ができるので、以下のように実行する。

```
guest##@node00:~$ ./neuron1
```

実行すると、数字がどばっ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、

```
guest##@node00:~$ ./neuron1 > neuron1.dat
```

gnuplot で表示する。

```
guest##@node00:~$ gnuplot
       G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'neuron1.dat' with line
```

すると、図 3 のような膜電位の表示が得られるはずである。1 秒間の間、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。

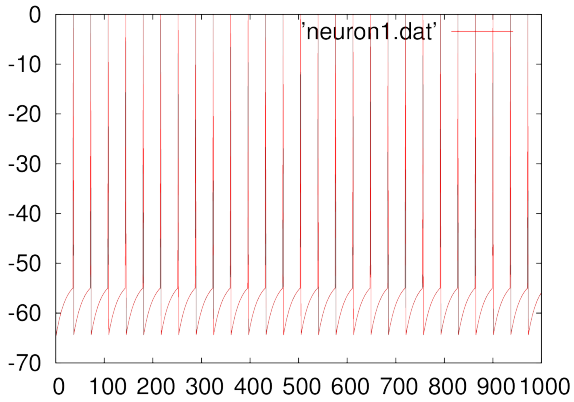


図 3　1 個のニューロンの膜電位のプロット

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 Physiologial Unit を使うか、あるいは SI Unit を使うか、どちらかにすること。

課題 1.

外部電流 I_{ext} の強さを色々変えて試してみよ。例えば 10nA, 15nA をそれぞれ試し、発火頻度 (発射されたスパイク数) を調べよ。

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。

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、`neuron1.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`, `spike`, `dv` を配列にして変数 `v[2]`, `spike[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。コードは以下のようになる。

ソースコード 3　`neuron2.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 R 1.0
9  #define DT 1.0
10 #define T 1000.0
11 #define I_EXT 12.0
12
13 void loop ( void )
14 {
15   double t = 0;
16   double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
17   int spike [ 2 ] = { 0, 0 };
18
19   while ( t < T ) {
20     printf ( "%f␣%f␣%f\n", t, ( spike [ 0 ] ? 0.0 : v [ 0 ] ), ( spike [ 1 ] ? 0.0 : v [ 1 ] )
         );
21     double dv [ 2 ] = { 0.0, 0.0 };
22     for ( int i = 0; i < 2; i++ ) {
23       dv [ i ] = ( DT / TAU ) * ( - ( v [ i ] - V_LEAK ) + R * I_EXT );
24     }
25     for ( int i = 0; i < 2; i++ ) {
26       spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
27       v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv [ i ];
28     }
29     t = t + DT;
30   }
31 }
32
33 int main ( void )
34 {
35   loop ( );
36   return 0;
37 }
```

コードの変更点は以下の通りである。v を配列にし (16 行目) 初期値を変更した。spike と dv も配列にした (17, 21 行目)。dv, v の計算は添字を変えて 2 回計算した (21–24 行目)。dv は添字を変えて 2 回計算した (22–24 行目)。v, spike も添字を変えて 2 回計算した (25–28 行目)。膜電位の表示の仕方を変えた (200 行目)。

これをコンパイルして実行し、結果をプロットする。

```
guest##@node00:~$ gcc -std=c99 -O3 -o neuron2 neuron2.c
guest##@node00:~$ ./neuron2 > neuron2.dat
guest##@node00:~$ gnuplot
       G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'neuron2.dat' using 1:2 with line, 'neuron2.dat' using 1:3 with line
```

図 4 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ずれるが、その他は同じなので同じ波形がシフトするだけとなる。

4.3 ネットワークのシミュレーション

二時間目の最後に、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。

$$\tau_{\mathrm{syn}} \frac{dg_i}{dt} = -g_i(t) + w \cdot \mathrm{Spike}_{(i+1)\%2}(t) \tag{12}$$

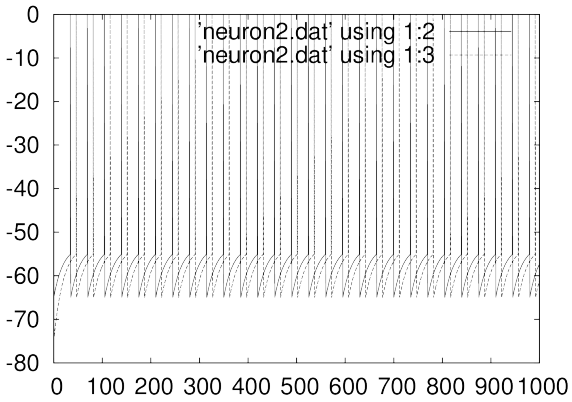


図 4　2 個の独立なニューロンの膜電位のプロット

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 ($i \in \{0, 1\}$)、τ_{syn} は時定数、$g_i(t)$ はシナプス後電位、w は結合重み、$\mathrm{Spike}_{(i+1)\%2}(t)$ 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[*9]。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。

$$g_i(t + \Delta t) = g_i(t) + \frac{\Delta t}{\tau_{\mathrm{syn}}} \left(-g_i(t) + w \cdot \mathrm{Spike}_{(i+1)\%2}(t) \right) \tag{13}$$

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。

ソースコード 4　network2.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 R 1.0
9  #define DT 1.0
10 #define T 1000.0
11 #define I_EXT 12.0
12 #define TAU_SYN 5.0
13 #define W 10.0
14
15 void loop ( void )
16 {
17   double t = 0;
18   double g [ 2 ] = { 0.0, 0.0 };
19   double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
20   int spike [ 2 ] = { 0, 0 };
21
22   while ( t < T ) {
23     printf ( "%f␣%f␣%f\n", t, ( spike [ 0 ] ? 0.0 : v [ 0 ] ), ( spike [ 1 ] ? 0.0 : v [ 1 ] )
         );
24     double dg [ 2 ] = { 0.0, 0.0 };
25     double dv [ 2 ] = { 0.0, 0.0 };
26     for ( int i = 0; i < 2; i++ ) {
```

[*9] $i = 0$ のとき $(i+1)\%2 = 1$、$i = 1$ のとき $(i+1)\%2 = 0$ なので。

```
27      dg [ i ] = ( DT / TAU_SYN ) * ( - g [ i ] + W * spike [ ( i + 1 ) % 2 ] );
28      dv [ i ] = ( DT / TAU ) * ( - ( v [ i ] - V_LEAK ) + g [ i ] + R * I_EXT );
29    }
30    for ( int i = 0; i < 2; i++ ) {
31      spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
32      g [ i ] = g [ i ] + dg [ i ];
33      v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv [ i ];
34    }
35    t = t + DT;
36  }
37 }
38
39 int main ( void )
40 {
41   loop ( );
42   return 0;
43 }
```

コードの変更点は以下の通りである。まずシナプス入力の変数 g, dg(18, 24 行目) を定義する。27 行目で式を計算し、28 行目で膜電位の式にシナプス入力を加える。32 行目で値を更新する。

これをコンパイルして実行し、結果をプロットする。

```
guest##@node00:~$ gcc -std=c99 -O3 -o network2 network2.c
guest##@node00:~$ ./network2 > network2.dat
guest##@node00:~$ gnuplot
       G N U P L O T
(...snip...)
Terminal type set to 'x11'
gnuplot> plot 'network2.dat' using 1:2 with line, 'network2.dat' using 1:3 with line
```

図 5 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。

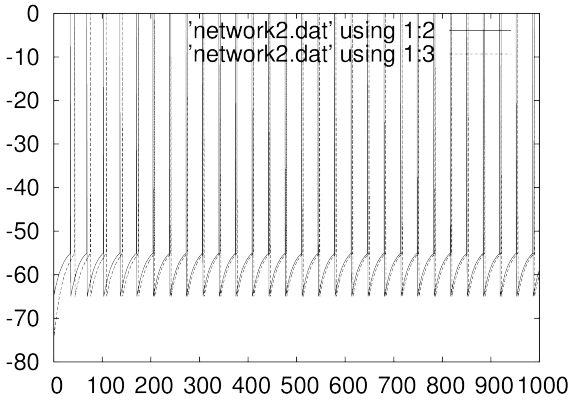


図 5　互いに興奮性で接続された 2 個のニューロンの膜電位

課題 2.

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。

二時間目はここまで。休憩！

5 三時間目: ランダムネットワークの並列計算

5.1 ランダムネットワーク

二時間目にやった 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 [4]。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 [5]。

膜電位の式は式 (4)–(6) と同様である:

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

ここで、$v(t)$ は時刻 t での膜電位、$\tau = 20$ ms は時定数、$V_{\text{leak}} = -49$ mV は静止電位、$ge(t)$, $gi(t)$ はそれぞれ興奮性、抑制性のシナプス電流、$\theta = -50$ mV はスパイク発射のための閾値、$V_{\text{reset}} = -60$ mV はリセット電位、$V_{\text{init}} = -60 + 10 \times \text{rand}(t)$ は膜電位の初期値、$\text{rand}(t)$ は $[0, 1)$ の一様乱数である。一方、シナプス電流は以下の式で計算する。

$$\begin{aligned} \tau_e \frac{dge}{dt} &= -ge(t) + \sum_{j \in \text{Exc}} w_e \cdot \text{Spike}_j(t), \\ \tau_i \frac{dgi}{dt} &= -gi(t) + \sum_{j \in \text{Inh}} w_i \cdot \text{Spike}_j(t). \end{aligned} \tag{14}$$

ここで、$\tau_e, \tau_i = 5, 10$ ms はそれぞれ時定数、Exc, Inh はそれぞれ興奮性、抑制性のニューロン集団、$w_e, w_i = +1.62, -9$ mV はそれぞれスパイク入力 1 発あたりのシナプス後電位の変化量、$\text{Spike}_j(t) \in \{0, 1\}$ はニューロン j が時刻 t でスパイクを発射した場合 1, そうでなければ 0 である。

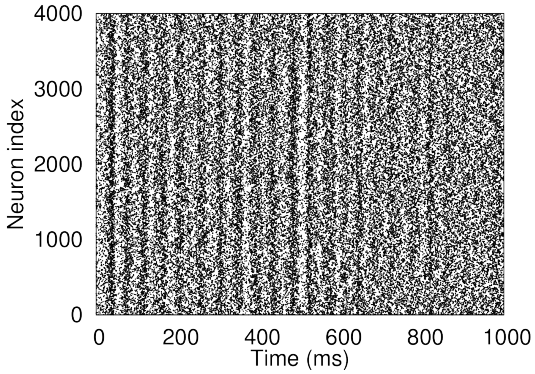


図 6 ランダムネットワークのラスタープロット。横軸は時間 (ms)、縦軸はニューロン番号である。1 つのドットが 1 つのスパイクを表す。

コードは `random.c` である。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 6 のようなラスター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。

ソースコード 5 `random.c`

```
1 void loop ( void )
```

```
2  {
3    double t = 0.;
4    timer_start ();
5    while ( t < T ) {
6      for ( int i = 0; i < N; i++ ) {
7        calculateSynapse ( i );
8        updateMembranePotential ( i );
9      }
10     outputSpike ( t );
11     t = t + DT;
12   }
13   double elapsedTime = timer_elapsed ();
14   printf ( "Elapsed␣time␣=␣%f␣sec.\n", elapsedTime);
15 }
```

時間に関するループ (5 行目) とニューロンに関するループ (6 行目) はこれまで通り。ループの中では、まずシナプス入力の計算をし (7 行目)、ついで膜電位の値を更新する (8 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 (10 行目)。関数 `calculateSynapse` および `updateMembranePotential` の中身はご想像の通りである。

本スプリングスクー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 16 秒かかる。各自試してみよ。コンパイルは `make random` を実行すると良い。自動的にコンパイルされる[*10]。

5.2 MPI による計算の並列化

本スプリングスクールで使うクラスタマシンは 8 ノード 192CPU コアからなり、これを全て使えば理論上は 192 倍速く計算が完了するはずである。そのような並列計算は MPI (Message Passing Interface) を用いて行う。MPI は複数の計算ノードを用いた並列計算のための標準規格であり [7]、フリーなものからネットワーク機器ベンダの独自のものまで、様々な実装がある。MPI には同期/非同期の送信/受信、全体の制御の同期、リダクションなど様々な命令が用意されていて、非常に低レベルの記述ができる分、コードは複雑になる傾向がある。ここでは最も容易な `MPI_Allgather` 命令を用いた並列化を紹介する。

並列化されたコードを次に示す。

ソースコード 6 `random_mpi.c`

```
1  void loop ( const int mpi_size, const int mpi_rank )
2  {
3    const int n_each = ( N + mpi_size - 1 ) / mpi_size;
4    int spike_local [ n_each ];
5    double t = 0.;
6    timer_start ();
7    while ( t < T ) {
8      for ( int n = 0; n < n_each; n++ ) {
9        calculateSynapse ( n, n_each * mpi_rank );
10       updateMembranePotential ( n, n_each * mpi_rank, spike_local );
11     }
12     MPI_Allgather ( spike_local, n_each, MPI_INT, spike, n_each, MPI_INT, MPI_COMM_WORLD );
13     if ( mpi_rank == 0 ) { outputSpike ( t ); }
14     t = t + DT;
15   }
16   double elapsedTime = timer_elapsed ();
17   if ( mpi_rank == 0 ) { printf ( "Elapsed␣time␣=␣%f␣sec.\n", elapsedTime); }
```

[*10] Makefile を用意してあるため

```
18 }
```

まず、関数に引数 mpi_size, mpi_rank が渡されている (1,2 行目)。これは MPI を初期化したときに得られる値であり、それぞれコア[*11]の総数と、自分自身のコア番号を表す。大きな変更点はニューロンに関するループ (8 行目) である。全ニューロン数を全コア数で割り、各コアは n_each 回ループして、スレッド毎に n_each 個のニューロンのみを計算する[*12]。そのため、関数 calculateSynapse と updateMembranePotential の引数に、計算すべきニューロン群の先頭の番号 (n_each * mpi_rank) を加える。計算されたスパイク発射の情報は大きさ n_each の配列 spike_local に格納する。spike_local の定義は 4 行目である。MPI_Allgather の実行は 12 行目である。これを実行すると、各スレッドが保持している spike_local の内容を全スレッドで共有し、結果を大きさ N の配列 spike に格納する (図 7)。よって命令の実行後は逐次版と同様に、ニューロンのスパイクの情報が復元され、全スレッドで共有されることになる。

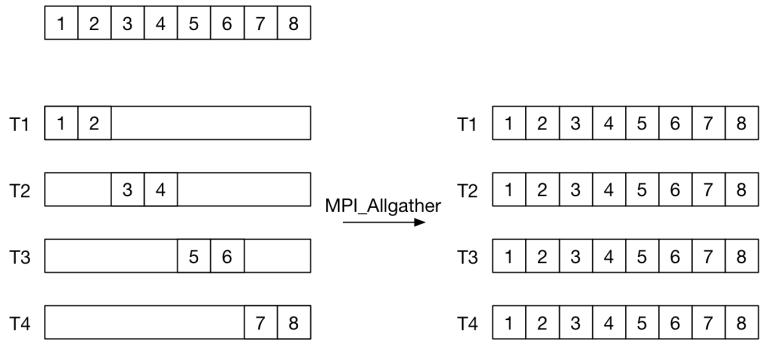


図 7 MPI_Allgather による計算の並列化。4 スレッド (T1–4) で 8 ニューロン (四角) の計算をする例を考える。各スレッドは 2 個のニューロンの計算だけを行い、スパイク発射の有無を保持する。MPI_Allgather を実行するとスパイクの情報が交換され、全スレッドで共有される。実行後は全てのスレッドが逐次計算版と同じ状態になり、計算を継続できる。

MPI を使ったコードは mpicc でコンパイルし、mpirun で以下のように実行する。コンパイルは make random_mpi とする。

```
guest##@node00:~$ make random_mpi
guest##@node00:~$ mpirun -hostfile hostfile -np 16 ./random_mpi
```

hostfile は利用可能な計算ノード名を記載したテキストファイル、-np の引数は実際に計算に使う CPU のコア数である。hostfile の内容は例えば次のようになる。

```
1 node01:24
2 node02:24
3 node03:24
4 node04:24
5 node05:24
6 node06:24
7 node07:24
8 node08:24
```

計算ノード名とノード当たりの CPU 数をコロン (:) で繋いだものを列挙する。

ここで注意！ MPI による並列シミュレーションでは一人で全ノードを占有することになるので、一度に一人しか試すことができない。声をかけあって順番を守ること。

[*11] より正確にはスレッド

[*12] 端数がでることがあるので、実際には n_each の計算は少しトリッキーである (3 行目)

本スプリングスクールで使うクラスタマシンでは`-np 192`までは速くなる。ということで次の課題。

課題 3.
`-np`の値をいくつか試して計算時間がどう変わるかを調べよ。理想的には例えば`-np`の値を 2 倍にすると計算時間は 1/2 になり、一般に n 倍すると $1/n$ になる。このような理想的な状態を強スケーリングと言う。

5.3 その他

並列計算用のハードウェアとしてグラフィクスプロセッシングユニット (GPU) が良く用いられる。NVIDIA 社が自社の GPU 用に開発している並列計算ライブラリ CUDA (Compute Unified Device Architecture)[8]、OpenCL Working Group が仕様策定しているライブラリ OpenCL[9] で並列化されたコードが書けるが、本スプリングスクールの範囲を超えるので、今回はやらない。

また今回はニューロンモデルとして LIF だけを考えたが、スパイク生成のメカニズムを研究するなら Hodgkin-Huxley 方程式を解く必要があるし、ニューロンの形状を研究するならマルチコンパートメントモデルにする必要があり、その場合はケーブル方程式を解く必要がある。それらについても本スプリングスクールの範囲を超える。

6 まとめ

そういうわけで、脳は作れます。でも計算機を上手に使う必要があります。I 類ではそのような使い方をきっちり学ぶことができます。

ではまたどこかでお会いしましょう。

参考文献

[1] GIGAZINE (2013). 人間の脳の活動でわずか 1 秒間はなんとスーパーコンピュータ「京」の 40 分に匹敵することが判明 `http://gigazine.net/news/20130806-simulating-1-second-of-real-brain/`. (最終アクセス 2017 年 10 月 14 日)

[2] Exascale Computing Project.
`https://exascaleproject.org/` (最終アクセス 2017 年 10 月 20 日)

[3] Gerstner W, Kistler W. Spiking Neuron Models. Cambridge (2001).

[4] Example: CUBA. `http://brian2.readthedocs.io/en/stable/examples/CUBA.html` (最終アクセス 2017 年 10 月 17 日)

[5] Brette R et al. Simulation of networks of spiking neurons: A review of tools and strategies. J Comp Neurosci 23:349-398, 2007.

[6] Goodman DF, Brette R. The Brian simulator. Front Neurosci 10:3389, 2009.

[7] Gropp W, Lusk E, Skjellum A. Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface. MIT Press (1999).

[8] NVIDIA. CUDA C Programming Guide. `http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html` (最終アクセス 2017 年 10 月 27 日)

[9] OpenCL Working Group. OpenCL Overview. `https:// www.khronos.org/opencl/` (最終アクセス 2017 年 10 月 27 日)