スパコンで脳を再現する

山﨑 匡 (@neuralgorithm)

電気通信大学 大学院 情報理工学研究科
NumericalBrain.Org

BLSCスプリングスクール 2018年3月28,29日 電気通信大学

ようこそ！

満員御礼

スケジュール

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

資料配布

・授業テキスト (当日配布版)
・スライドの縮小コピー

を配ります。

ノートを取る必要はありません。メモはテキストやスライドに書き込んで下さい。話を聞く方が重要。

今日やらないこと

・人工知能
・深層学習

もしこういうのを期待して来てたらごめんなさい

自己紹介

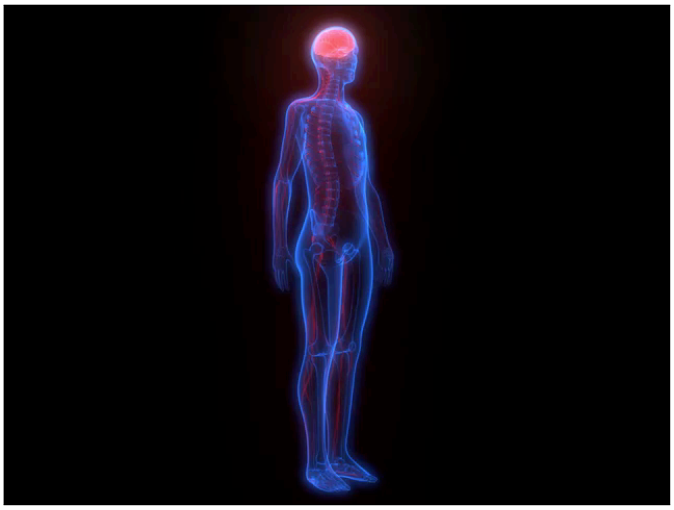
クラスタマシンへのログイン

クラスタマシンへのログイン

1. 各自のユーザ名を配ります (guest##, ##は数字)
2. 端末エミュレータで
   `ssh -Y <hostname> -l guest##`
   と入力してください
3. ログインできると
   `guest##@node00:~$ □`
   と表示されると思います
4. プロンプトに続けて
   `sl`
   と入力して何が起きるかを確認してください

一時間目

高校生のための計算神経科学入門



ニューロンの数式

$$C\frac{dV}{dt} = -g_{\mathrm{leak}}\left(V(t) - E_{\mathrm{leak}}\right) + I_{\mathrm{syn}}(t) + I_{\mathrm{ext}}(t)$$

$$I_{\mathrm{syn}}(t) = -g_{\mathrm{syn}}(t)\left(V(t) - E_{\mathrm{syn}}\right)$$

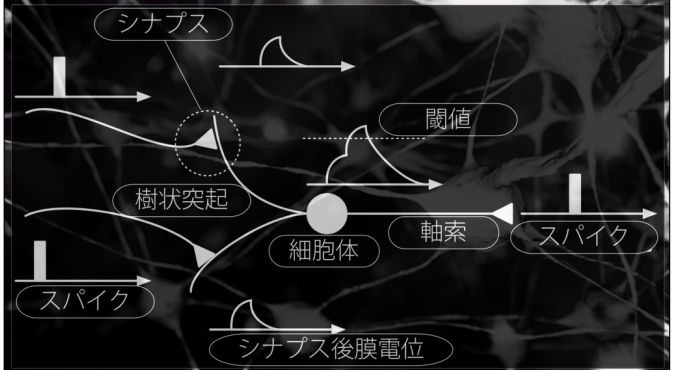
$$\tau_{\mathrm{syn}}\frac{dg_{\mathrm{syn}}}{dt} = -g_{\mathrm{syn}}(t) + \sum_j w_j \delta_j(t)$$

入力スパイク

シナプス結合重み

$$V(t) > \theta \Rightarrow \delta(t) = 1,\ V(t) = V_{\mathrm{reset}}$$

ニューロンの計算

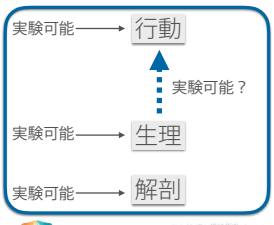


神経回路シミュレーションとは

人工の脳を作ることに相当

脳のシステム的理解
- 神経回路の挙動の**再現**
- 仮説の**検証**
- 実験結果の解釈と**予言**

実験ではできない操作が可能



EUのThe Human Brain Project

ヒト全脳シミュレーション
10年で10億ユーロ

JUQUEEN (IBM, 5.9PFs)

Markram et al. Cell (2015)

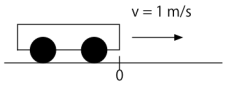
何をどう計算するか？

1. 何を計算するか？ → 先ほどの式を計算する
2. どう計算するか？ → 数値計算法が必要

高校物理の超簡単なおさらい

問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。



問1a: 移動を開始してから1秒後の位置 x(1) を答えよ。
問1b: 2秒後、3秒後、... a秒後の位置をそれぞれ答えよ。

答え

x(1) = 1 メートル、x(a) = a メートル

高校物理の超簡単なおさらい

問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。

答え

v = (x(2) - x(1)) / 1 メートル/秒

高校物理の超簡単なおさらい

問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+Δt秒で位置x(t+Δt)メートルにいたとする。vをxで表せ。

Δ:「デルタ」と読む

答え

v = (x(t+Δt) - x(t)) / Δt メートル/秒

高校物理の超簡単なおさらい

問4

問3の設定の元で、x(t+Δt)をx(t), v, Δtで表せ。

v = (x(t+Δt) - x(t)) / Δt

答え

x(t+Δt) = x(t) + v × Δt　　(×は普通のかけ算)

この式の意味を考えよう

$$x(t+\Delta t) = x(t) + v \times \Delta t$$

・時刻tでの位置 x(t)
・そのときの速度 v

がわかれば、Δt 後の位置 x(t+Δt) がわかる

ということを言ってるんだけど、わかりますか？

もう一歩進めて考えよう

$$x(t+\Delta t) = x(t) + v \times \Delta t$$

・ x(0)とvからx(Δt)がわかる
・x(Δt)とvからx(Δt+Δt) = x(2Δt)がわかる
・x(2Δt)とvからx(2Δt+Δt) = x(3Δt)がわかる
・ :

つまり、最初の位置x(0)と速度vがわかれば、
未来の位置 x(t) は(Δt毎に)全て順番にわかる

ということも言えるんだけど、わかりますか？

数値計算法を導入する

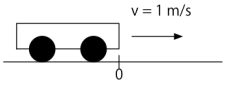
$$x(t+\Delta t) = x(t) + v \times \Delta t$$

x(0)とvからx(Δt), x(2Δt), x(3Δt), …を順番に求める
→ 前方オイラー法 (大学2年後期の「数値計算」で学ぶ)

ここからクラスタマシンで実際にプログラムを実行します

ちなみに v は一般に v(t) でかまわない

問1の答えを数値的に求めよう



以下はクラスタマシンで作業する

・**ls**
と入力するとファイルの一覧が表示される
・**car.c** というファイルがあるはず
・**nano car.c**
として、ファイルを開く(**nano**はエディタ)

car.cの説明

```
#include<stdio.h>

int main ( void )
{
  double t = 0; // 時刻：最初は0秒から
  double x = 0; // 位置：最初は原点から0メートルから
  double v = 1.0; // 速度：1メートル/秒
  double dt = 1.0; // 時間の刻み幅：1秒ずつ進める

  while ( t < 10.0 ) { // 10 秒間繰り返し

    printf ( "%f %f\n", t, x ); // 今の時刻と位置を表示

    x = x + v * dt; // 次の時刻の位置を計算
    t = t + dt; // 時間をdt秒進める
  }

  return 0;
}
```

プログラムを実行する

以下はクラスタマシンで作業する

- nanoで**^X** (Controlを押しながらXを押す)

  → nano が終了する

- **gcc -std=c99 -O3 -o car car.c**

  (TODO: おまじないの説明をする)

- **./car**

  計算結果の数字列が表示されるはず

計算結果を表示する

以下はクラスタマシンで作業する

- **./car > car.dat**

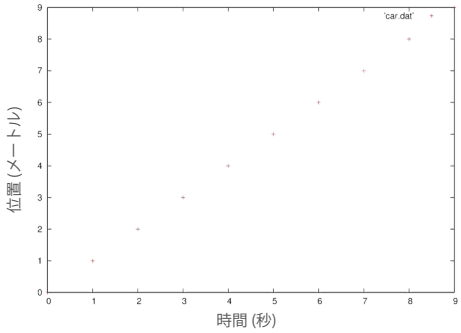
  計算結果をファイルに保存

- **gnuplot**

  gnuplotはグラフを表示するアプリ

- **plot 'car.dat'**

こんなのが表示されるはず



- 終わったら Ctrl + D でgnuplotを終了する
- 時間があったら、**v**を**t**に書き換えて等加速度運動にする
  書き直したら**^X** → **Y** → リターン

一時間目まとめ

- 脳はニューロンがシナプスで繋がったネットワークである
- 1個のニューロンとシナプスの振る舞いは数式で書ける
- その数式は数値的に答えを求めることができる
- ので、全ニューロン・全シナプスの数式を全てプログラムして数値的に解けば、(原理的には)脳と同じ動作をするプログラム (いわばコンピュータ上の脳) を作成できる

二時間目は実際にニューロンとシナプスの式を解きます

一時間目まとめ

- 脳はニューロンがシナプスで繋がったネットワークである
- 1個のニューロンとシナプスの振る舞いは数式で書ける
- その数式は数値的に答えを求めることができる
- ので、全ニューロン・全シナプスの数式を全てプログラムして数値的に解けば、(原理的には)脳と同じ動作をするプログラム (いわばコンピュータ上の脳) を作成できる

ちなみに、(x(t+Δt) - x(t)) / Δtは、Δtが十分小さいとき dx/dt (もしくは x')と書く。これをxの (時間に関する) **微分**という。もちろん v(t) = dx/dt である。

10分休憩！

二時間目

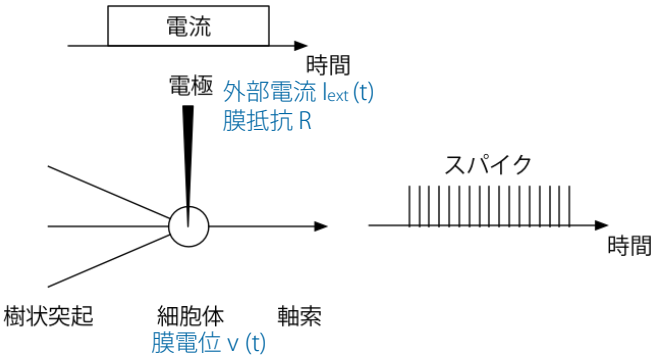
神経回路シミュレーション事始め

内容

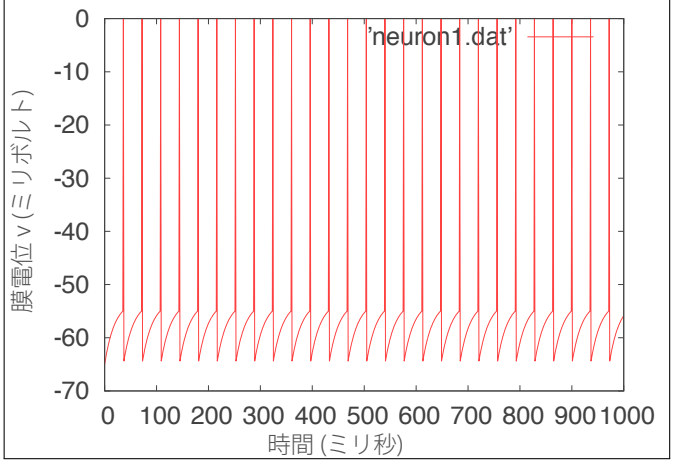
・ニューロン1個のシミュレーション
・ニューロン2個 (繋がない) のシミュレーション
・ニューロン2個 (繋ぐ) のシミュレーション

をやります。

ニューロン1個のシミュレーション



ニューロン1個のシミュレーション



積分発火型ニューロンモデル

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

何だこれ？と思っても心配無用！説明します！

積分発火型ニューロンモデル

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

積分発火型ニューロンモデル

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

τ (タウ), V_{leak}, R は定数

$$\frac{v(t + \Delta t) - v(t)}{\Delta t}$$

Δt も定数

式変形をすると、

$$v(t + \Delta t) = v(t) - \frac{\Delta t}{\tau}(v(t) - V_{leak}) + \frac{\Delta t}{\tau}RI_{ext}(t)$$

v (0) と $I_{ext}(t)$ を決めればあとは順番に計算できる

積分発火型ニューロンモデル

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

何だこれ？と思っても心配無用！説明します！

積分発火型ニューロンモデル

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

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

$$v(0) = V_{init}.$$

θ (シータ): 定数(閾値)

V_{reset}: 定数

何だこれ？と思っても心配無用！説明します！

積分発火型ニューロンモデル

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

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

$$v(0) = V_{init}.$$

初期値の設定

何だこれ？と思っても心配無用！説明します！

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.    while ( t < T ) {
19.     printf ( "%f %f\n", t, (spike ? 0.0 : v ) );
20.     double dv = ( DT / TAU ) * ( - ( v - V_LEAK ) + R * I_EXT );
21.     spike = ( v > THETA ) ? 1 : 0;
22.     v = ( v > THETA ) ? V_RESET : v + dv;
23.     t = t + DT;
24.   }
25. }
26.
27. int main ( void )
28. {
29.   loop ( );
30.   return 0;
31. }
```

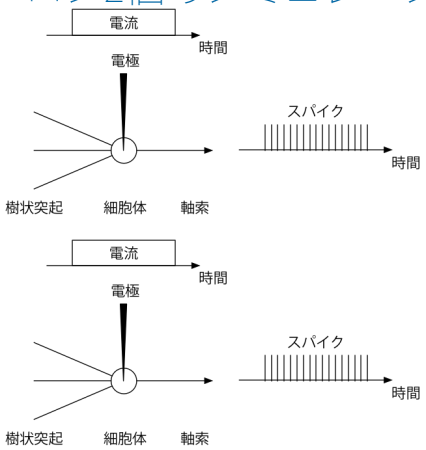
要説明

コードを試す

- `gcc -std=c99 -O3 -o neuron1 neuron1.c`
- `./neuron1`

  数字が表示されるのを確認
- `./neuron1 > neuron1.dat`
- `gnuplot`
- `plot 'neuron1.dat' with line`
- [課題] 外部電流の強さを10.0 (ナノアンペア)や15.0 (ナノアンペア)に変更して、発射されるスパイク数の変化を調べてください

ニューロン2個のシミュレーション



neuron2.c
(抜粋)

```
1.  void loop ( void )
2.  {
3.    double t = 0;
4.    double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
5.    int spike [ 2 ] = { 0, 0 };
6.     while ( t < T ) {
7.      printf ( "%f %f %f\n", t, ( spike [ 0 ] ? 0.0 : v [ 0 ] ),
8.                                ( spike [ 1 ] ? 0.0 : v [ 1 ] ) );
9.      double dv [ 2 ] = { 0.0, 0.0 };
10.     for ( int i = 0; i < 2; i++ ) {
11.       dv [ i ] = ( DT / TAU ) * ( - ( v [ i ] - V_LEAK ) + R * I_EXT );
12.     }
13.     for ( int i = 0; i < 2; i++ ) {
14.       spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
15.       v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv [ i ];
16.     }
17.     t = t + DT;
18.   }
19. }
```

要説明

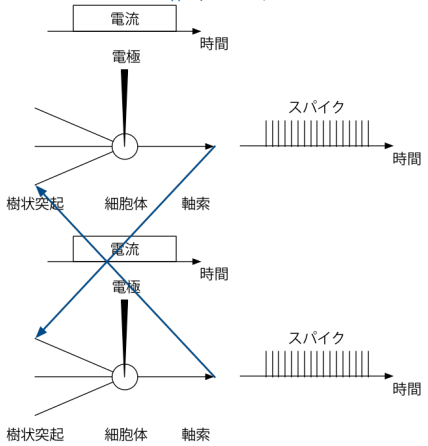
コードを試す

- `gcc -std=c99 -O3 -o neuron2 neuron2.c`
- `./neuron2`

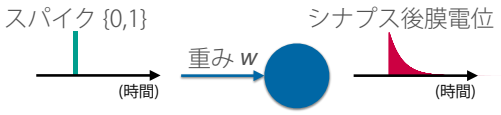
  数字が表示されるのを確認
- `./neuron2 > neuron2.dat`
- `gnuplot`
- `plot 'neuron2.dat' using 1:2 with line,`

  `'neuron2.dat' using 1:3 with line`

ニューロン2個のネットワーク



指数関数型シナプス



$$\tau_{\mathrm{syn}} \frac{dg}{dt} = -g(t) + w \cdot \mathrm{Spike}(t)$$

network2.c
(抜粋)

```
1.  void loop ( void )
2.  {
3.    double t = 0;
4.    double g [ 2 ] = { 0.0, 0.0 };
5.    double v [ 2 ] = { V_INIT, V_INIT - 10.0 };
6.    int spike [ 2 ] = { 0, 0 };
7.
8.    while ( t < T ) {
9.      printf ( "%f %f %f\n", t, ( spike [ 0 ] ? 0.0 : v [ 0 ] ),
10.                              ( spike [ 1 ] ? 0.0 : v [ 1 ] ) );
11.     double dg [ 2 ] = { 0.0, 0.0 };
12.     double dv [ 2 ] = { 0.0, 0.0 };
13.     for ( int i = 0; i < 2; i++ ) {
14.       dg [ i ] = ( DT / TAU_SYN ) * ( - g [ i ] + W * spike [ ( i + 1 ) % 2 ] );
15.       dv [ i ] = ( DT / TAU ) * ( - ( v [ i ] - V_LEAK ) + g [ i ] + R * I_EXT );
16.     }
17.     for ( int i = 0; i < 2; i++ ) {
18.       spike [ i ] = ( v [ i ] > THETA ) ? 1 : 0;
19.       g [ i ] = g [ i ] + dg [ i ];
20.       v [ i ] = ( v [ i ] > THETA ) ? V_RESET : v [ i ] + dv [ i ];
21.     }
22.     t = t + DT;
23.   }
24. }
```

要説明

課題

- `gcc -std=c99 -O3 -o network2 network2.c`
- `./network2`
  数字が表示されるのを確認
- `./network2 > network2.dat`
- `gnuplot`
- `plot 'network2.dat' using 1:2 with line,`
  `'network2.dat' using 1:3 with line`
- [課題] wの符号を負にして再度試してください

二時間目まとめ

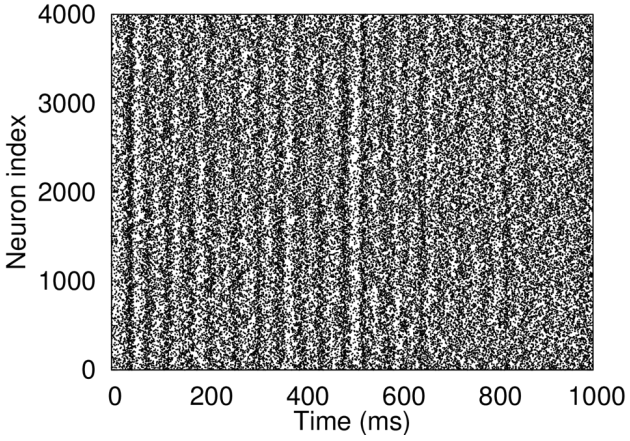
- ニューロン1個のシミュレーションから始めて、ニューロン2個のネットワークまで作成した
- シナプスの結合重みを変えるだけでネットワークの挙動は変化した
- 脳はネットワークの繋ぎ方がとても重要である

10分休憩！

三時間目

ランダムネットワークの並列計算

ニューロン4000個のネットワーク



random.c
(抜粋)

```
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. }
16.
```

ニューロン数 (4000)

要説明

プログラムを実行してみる

- **make random**

  自動的にコンパイルされる
- **./random**
- **spike.dat**ができる
- **gnuplot**
- **plot 'spike.dat' with dots**

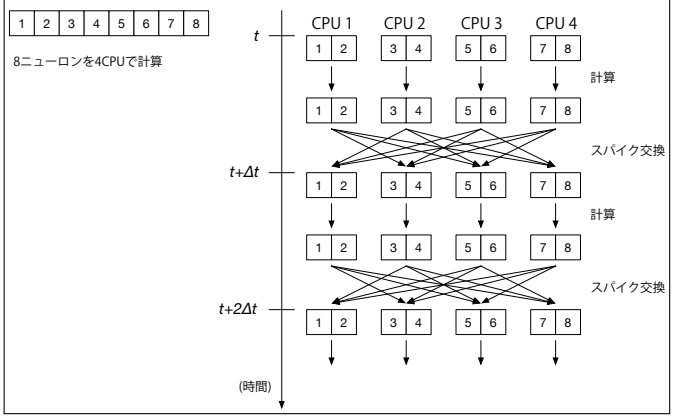
  計算時間も測定する

計算が遅すぎる問題とその解決策

- 計算時間が結構かかりますよね (16秒くらい)
- ここまで1個のCPU (コア) しか使っていません
- 今使っているクラスタマシンは全部で192コアです
- ので、192コア全部使わないともったいない

  → 並列計算

例: 8ニューロンを独立の4CPUで計算

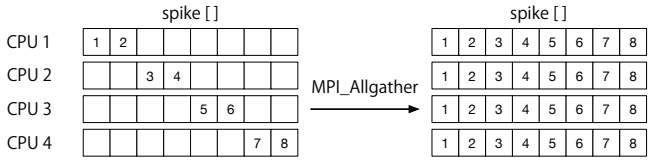


スパイク交換 = MPIによる通信

ネットワークを介して、異なるCPU間で情報を転送する

MPI (Message Passing Interface) というライブラリがある

今日は一番簡単なMPI_Allgatherを使います



random_mpi.c (抜粋)

```
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); }
18. }
```

4000 / コア数
(例えば100コアあれば40)

要説明

コードを試す

- **make random_mpi**
- **mpirun -hostfile hostfile -np 16 ./random_mpi**
- 計算時間を測定する
- **-np** の後ろの数字を変えて測定する

  10人いるから、1, 2, 4, 8, 16, 32, 64, 128, 160, 192をそれぞれ

  1人ずつ順番に試そう

計算時間はどう変化するか？

もし、CPU数を2倍にすると計算時間が1/2になったら、

理想的な並列化がなされている → 強スケーリング

プログラムの全ての箇所が並列化できるわけではない

並列化できない箇所に計算時間がかかっていたら、

できる箇所を頑張っても意味が無い → アムダールの法則

なので、並列化は必ずしもあらゆる問題に対する解決策

(いわゆる銀の弾丸) ではない

三時間目まとめ

- 4000ニューロンからなるランダムネットワークのシミュレーションをおこなった
- MPIを使った並列計算で、計算を高速化した

全体まとめ

まとめ

・脳は作れる！
・ただし、計算機を上手に使う必要あり
・I類ならそのような使い方をばっちり学べます

質問？

おしまい！

閉会式に移動