解 説

神経系大規模シミュレーションのためのソフトウェア

～NEURON と NEST～

浦 久 保 秀 俊

京都大学大学院情報学研究科システム科学専攻*

Software for Large-scale Neural Simulation: NEURON and NEST

Hidetoshi Urakubo

Department of Systems Science, Graduate School of Informatics, Kyoto University*

概要

近年、多数の PC (計算ノード) を並列につなげたスーパーコンピュータを用いて、脳神経活動をより大規模に、あるいは精密にシミュレーションしようとする試みが行われるようになってきた。このようなシミュレーションを行うにあたっては、複数の PC の上で単一のプログラムを分散・並列に実行する「並列プログラミング」が必須になるが、汎用プログラム言語を用いて並列プログラミングを行うことは容易ではない。本稿では、並列プログラミングに対応した神経シミュレータである「NEURON」および「NEST」に注目し、その特長、使用方法、並列化方法について簡単に紹介する。

1. はじめに

脳の神経活動をコンピュータシミュレーションにより再現しようとする試みは、古くは Hodgkin と Huxley (1952) より始まり、人工ニューラルネットワークと関係しつつ 60 年以上の歴史を持って発展してきた¹⁾。脳は精密な「情報処理装置」と言える。無数の神経細胞 (ニューロン) の相互作用が織りなす複雑な情報処理過程を計算機シミュレーションにより理解しようとする試みは、間違いなく脳神経研究の一つの方向性である。

さらに近年は多数の PC (計算ノード) を並列につなげた「スーパーコンピュータ」を用いて、脳神経活動をより大規模に、あるいは精密にシミュレーションしようとする試みが行われるようになってきた^{2, 3)}。最大 1~10 億個ものニューロン、1~10 兆個ものシナプスを各計算ノードに割り当てて、ニューロンネットワークの並列計算が行われている^{3, 4)}。まさに現実の脳 (の一部) をまるごとスーパーコンピュータの中に再現しようというのである。

このようなシミュレーションを行うにあたっては、複数の PC の上で単一のプログラムを分散・並列に実行する「並列プログラミング」が必須になる。また、あえてスーパーコンピュータを使わずとも、PC の性能を効果的に活用しつつニューロンネットワークのシミュ

レーションを行うために、並列プログラミングは重要である。現在の PC は単一 CPU の速度を上げるというよりは、CPU 数あるいは CPU 内のコア数を増やす方向に進化を続けている。並列プログラミングにより複数の CPU・コアを協調して用いることにより、計算を加速することができる。

しかし、C/C++などの汎用言語を用いてニューロンネットワークの並列プログラミングを行うことは、次の 2 つの理由でお勧めできない。第一に、並列プログラミングはプロセス間の情報の受け渡し、同期など独特の技術を必要とし、素人が取り組むには困難を伴う。第二に、多くのモデル開発者が同じような目的を共有しており、すでに並列プログラミングに対応したシミュレーション環境 (シミュレータ) が存在する。まずは、これらのシミュレータを試すことが近道であろう。

本稿では、並列プログラミング・大規模シミュレーションに対応した神経シミュレータである「NEURON」⁵⁾および「NEST (NEural Simulation Tool)」⁶⁾の特長、使用方法、プロセス並列 (MPI 並列) 化法¹¹⁾について

¹¹⁾ 並列化にはプロセス並列とスレッド並列の 2 種類がある。スレッド並列は NEURON, NEST とユーザから見えない形ですで行っている。また、モデルパラメータのグリッドサーチなど、プロセス/スレッド間の通信のない、いわゆる「Embarrassingly Parallel」な並列計算は扱わない。

図1 NEURON を用いたシミュレーション。A: NEURON 起動。コマンドプロンプトより `nrngui` コマンドを実行すると、NEURON Main Menu ウィンドウが立ち上がる。B: サンプルプログラム 1 により定義されるニューロン “dendrite_neuron”。C: サンプルプログラム 2 により定義されるニューロンネットワーク。D: サンプルプログラム 2 の実行結果。各線は、ネットワーク中の各ニューロンの膜電位の時間変化を示す。E: サンプルプログラム 3 の実行結果。各 gid で指定されるニューロンの発火時刻がシミュレーション終了順に表示される。

簡単に紹介する。本稿は、あくまで利用者の視点から書かれた各シミュレータの日本語チュートリアルである。気軽に読んでいただけると幸いである。ただし、二つのシミュレータの使い方、設計思想について並べて紹介するだけで、現在の神経シミュレーションの現状について透けて見えてくるものがある。少し議論したい。

2. NEURON

2.1 特長と使用方法

NEURON とは、マルチコンパートメント Hodgkin-Huxley モデルニューロンのシミュレーションを第一の目的に、その他の抽象モデルニューロンやニューロンネットワークのシミュレーションを効率的に行うこともできる神経活動シミュレーションのための統合環境である⁵⁾。Nicholas T. Carnevale, Michael L. Hines らにより開発が続けられている。現状、NEURON は神経シミュレーションのための事実上の標準ツールで

ある。UNIX (Linux), Mac (OS X), Windows に対応し、近年は Python のモジュールとしても提供されている⁷⁾。NEURON Web サイト^{†2)}のインストラクションに従い NEURON のインストールに成功したら (付記参照)、コマンドプロンプトより “`nrngui`” コマンドを実行して、NEURON の立ち上げを確認されたい (図 1A)。

NEURON はモデル作成 & シミュレーション実行のために hoc (high order calculator) と呼ばれる独自のインタプリタ言語を持っている。モデル開発者は、hoc 言語を用いてモデルを定義して NEURON に読み込ませる。hoc 言語において、ニューロンやシナプスに対応するオブジェクトを生成することにより定義される。サンプルプログラム 1 “`dendritic_neuron.hoc`” (表 1) をご覧いただきたい。プログラムでは soma, dend という名前の円筒形の section (ニューロン外形) オブジェクトを生成し (7 行目)、オブジェクト

^{†2} <https://www.neuron.yale.edu/neuron/>

表 1 サンプルプログラム 1 “dendritic_neuron.hoc”

1	begintemplate Dendritic_Neuron
2	public init, soma, dend, syn, synlist, connect2target
3	create soma, dend
4	objectvar syn, synlist
5	
6	proc init() {
7	create soma, dend
8	soma {
9	nseg = 1
10	diam = 20
11	L = 20
12	insert hh
13	}
14	dend {
15	nseg = 5
16	diam = 3
17	L = 400
18	insert pas
19	}
20	connect soma(0), dend(1)
21	dend syn = new ExpSyn(0.5)
22	synlist = new List()
23	synlist.append(syn)
24	}
25	
26	obfunc connect2target() { localobj nc
27	soma nc = new NetCon(&v(0.5), \$o1, -20, 1, 0.5)
28	if (numarg() == 2) { \$o2 = nc }
29	return nc
30	}
31	endtemplate Dendritic_Neuron

変数「nseg：コンパートメント数」「diam：円筒直径」「L：円筒長さ」を定義している（8～19 行目）。soma に Hodgkin-Huxley チャンネル “hh” を導入し（12 行目）、dend には受動チャンネル “pas” を導入する（18 行目）。さらに、soma, dend 二つの section の端点 soma(0), dend(1) を電氣的に接続する^{†3}（20 行目）。dend の中央部（0.5）には指数関数減衰するシナプス入力部位を導入する（21～23 行目）。以上の手続きにより、図 1B

^{†3} section 円筒の長さ方向の相対位置は [0, 1] により指示される。

に示される “dendritic_neuron” が定義される。

NEURON においてニューロン外形を作成する section オブジェクトは、シミュレーションの際の実際の計算単位であるコンパートメントとは異なることに注意されたい。モデル開発者は、オブジェクト変数 nseg を変更するだけで section あたりのコンパートメント数（つまり空間粒度）を自由に設定することができる。顕微鏡観測で得られるニューロン、デンドライト、軸索の複雑な形状は、直径と長さの異なる円筒形の繋ぎ合わせとして表現される。NEURON はそのようなニュー

表 2 サンプルプログラム 2“main_serial.hoc”

1	// Definition of neurons
2	ncell = 4
3	load_file("dendritic_neuron.hoc")
4	objectvar N[ncell]
5	for i = 0, ncell-1 {
6	N[i] = new Dendritic_Neuron()
7	}
8	
9	// Synaptic connection
10	objectvar nc[ncell]
11	for i = 0, ncell-2 {
12	nc[i] = N[i].connect2target(N[i+1].syn)
13	}
14	
15	// Current injection
16	objectvar stim
17	N[0].soma {
18	stim = new IClamp(0.5)
19	stim.del = 7
20	stim.dur = 3
21	stim.amp = 1
22	}
23	
24	// Simulation and graph plot
25	tstop = 25
26	objectvar G
27	G = new Graph()
28	G.size(0, tstop, -80, 40)
29	G.addexpr("N[0].soma.v(0.5)",1,2)
30	G.addexpr("N[1].soma.v(0.5)",2,2)
31	G.addexpr("N[2].soma.v(0.5)",3,2)
32	G.addexpr("N[3].soma.v(0.5)",4,2)
33	graphList[0].append(G)
34	run()

ロン形状データと計算実体の間を仲介し、煩雑な計算を削減してくれる機能を持つ⁸⁾。Hodgkin-Huxley モデルは時間に関する常微分方程式 (ODE) である。NEURON が行うシミュレーションは、ODE をニューロンに最適な適応時間刻み幅でもって数値的に解いてゆくことである。

次に、複数の “dendritic_neuron” をシナプス結

合させてニューラルネットワークを作成し、シミュレーションを実行したい。サンプルプログラム 2“main_serial.hoc” (表 2) では、4 つの “dendritic_neuron” n[0]~n[3] を生成し (2~4 行目)、直列にシナプス結合を行っている (9~13 行目)。一つ目のニューロン n[0] に 7~10 msec の間 1 nA の電流入力を行いつつ (15~22 行目)、25 ms 間シミュレーショ

表 3 サンプルプログラム 3“main_parallel.hoc”

```

1 // Preprocessing
2 ncell = 4
3 load_file("netparmpi.hoc")
4 objectvar pnm
5 pnm = new ParallelNetManager(ncell)
6 pnm.round_robin()
7
8 // Definition of neurons and connections
9 load_file("dendritic_neuron.hoc")
10 for i=0, ncell-1 if (pnm.gid_exists(i)) {
11     pnm.register_cell(i, new Dendritic_Neuron())
12 }
13 for i=0, ncell-2 {
14     pnm.nc_append(i, i+1, 0, 0.5, 1)
15     /* (from, to, synlist id, weight, delay) */
16 }
17
18 // Current injection
19 objectvar stim
20 if (pnm.gid_exists(0)) pnm.pc.gid2cell(0).soma{
21     stim = new IClamp(0.5)
22     stim.del = 7
23     stim.dur = 3
24     stim.amp = 1
25 }
26
27 // Simulation and print
28 tstop = 25
29 pnm.set_maxstep(1)
30 pnm.want_all_spikes()
31 stdinit()
32 {pnm.psolve(tstop)}
33
34 for i=0, pnm.spikevec.size-1 {
35     print pnm.spikevec.x[i], "msec at gid ", pnm.idvec.x[i]
36 }
37
38 {pnm.pc.runworker}
39 {pnm.pc.done}
40 quit()

```

表 4 サンプルプログラム 4“addition.hoc”

1	// Recording
2	objectvar rect, recv
3	if (pnm.gid_exists(0)) pnm.pc.gid2cell(0).soma{
4	rect = new Vector()
5	recv = new Vector()
6	rect.record(&t)
7	recv.record(&v(0.5))
8	}
9	// Saving
10	objectvar savedata
11	if (pnm.gid_exists(0)) {
12	savedata = new File()
13	savedata.wopen("neuron0.dat")
14	for i = 0, rect.size()-1 {
15	savedata.printf("%g %g¥n", rect.x(i), recv.x(i))
16	}
17	savedata.close()
18	}

ンを実行して膜電位の時間変化をグラフ表示している (24~34 行目)。サンプルプログラム 1, 2 を同じディレクトリ上に作成し, “nrngui main_serial.hoc” コマンドを実行すると, ニューロン発火が伝わる様子をグラフ表示することができる (図 1D)。

以上のように, NEURON を利用することにより, C/C++などの汎用言語で記述するより圧倒的に短い記述でニューロン, ネットワークの定義を行い, 簡便にシミュレーションを実行することができる。

2.2 並列プログラム実行法

続いて, サンプルプログラム 2 で行ったシミュレーションを MPI 並列で行う方法について紹介する。NEURON では二つの方法により MPI 並列を行うことができる。一つは, 単一のマルチコンパートメントニューロンを分割し, 複数のプロセスに割り当てる方法⁹⁾, もう一つはネットワーク中の各ニューロンを各々別々のプロセスに割り当てる方法である¹⁰⁾。本稿では, ネットワーク中の各ニューロンを各々別々のプロセスに割り当てる方法をとる。

サンプルプログラム 3 “main_parallel.hoc” (表 3) は, サンプルプログラム 2 の MPI 並列版である。“mpirun -np 4 nrngui -mpi main_parallel.hoc” コマンドによりサンプルプログラム 3 を mpi 並列で実行すると,

MPI 設定の段階であらかじめ指定された (複数の) ホスト PC において, 同じプログラムが 4 つのプロセスで同時に実行される^{†4}。プリプロセスで 4 つのニューロンを生成することを宣言し (2, 5 行目), 各プロセスにニューロンインデックス gid を割り当てる。該当 gid を持つプロセスにおいて “dendritic_neuron” を生成し (9~12 行目), それらを直列にシナプス結合する (13~16 行目)。gid = 0 のニューロンの soma 中央部 (0.5) に 7~10 msec の間 1 nA の電流入力を行いつつ (18~25 行目), 並列にシミュレーションを実行し, 各ニューロンが発火した時刻を表示する (27~40 行目, 図 1E)。NEURON において, 並列プログラムを行うこと自体は意外と簡単であることがご理解いただけると思う。

並列プログラム実行時, NEURON は GUI を動作させることを想定していないが, 膜電位の時間変化などをファイルに保存することは可能である。サンプルプログラム 3 の 26 行目, 33 行目にサンプルプログラ

^{†4} 実は, 利用した ParallelNetManager class の仕様上の制限のため, サンプルプログラム 3 を実行可能なのは単一ホスト PC 上でのみである。もちろん, 複数のホスト PC 上で実行されるプログラムを書くことも容易である¹⁰⁾。プログラムは若干長くなる。

ム 4 (表 4) の 1~8 行目, 9~18 行目を差し込んでみていただきたい. `gid = 0` のニューロンの膜電位の時間変化が “neuron0.dat” というファイル名で保存される. 適当なグラフソフトで膜電位の時間変化を表示することができる.

2.3 プロセス間通信

複数のニューロンが各々別々のプロセスに割り当てられる並列シミュレーションを行う場合, プロセス間のイベント伝達 (シナプス伝達) を適切な方法で行うことが重要になる. NEURON では, シミュレーション実行前にすべてのシナプス伝達の遅延時間が比較され, もっとも短い伝達遅延がシナプス伝達のための単位時間 (*local_minimum_delay*) として設定される (29 行目). シミュレーションが実行されると, 時間積分は *local_minimum_delay* 時間ごとに一旦停止する. すべてのプロセスが停止時間に達すると, シナプス伝達の「送り手」ニューロンから「受け手」ニューロンへ積分時間内に発生したシナプス伝達 (強度, 遅延) がまとめて送信され, その後, 次の単位時間の積分が行われる. *local_minimum_delay* はシナプス伝達の遅延より必ず短いので, シナプス伝達が「受け手」ニューロンに到着する時刻より先に時間積分が進んでしまうことはない.

3. NEST

3.1 NEST とは

NEST とは, 積分発火ニューロンなどの比較的単純なモデルニューロンをノードとし, 複雑な結合パターンを持ったニューロンネットワークをシミュレーションすることを第一の目的に開発されたシミュレータである⁶⁾. NEST Initiative により活発に開発が進められている. UNIX (Linux), Mac (OS X), Windows (Cygwin) バージョンがあり, Python のモジュールとしても提供されている. 特に大規模ネットワークのシミュレーションを目的とした場合, 数あるシミュレータの中で, もっとも強力なものである. もちろん, Hodgkin-Huxley モデルニューロンなどの複雑なモデルニューロンのシミュレーションを行うこともできる.

NEST は, SLI という独自のインタプリタ言語を持つ. ただし, 入出力に関わる行列計算や結果のグラフ表示のことを考えると, Python モジュール版である PyNEST を用いてモデル開発を行うのが妥当であろう. 数値計算モジュール *numpy*, グラフ表示モジュール *Matplotlib* などと連携させることができる. インストールに成功したら (付記参照), Python のコマンドプロンプトより “import nest” コマンドを実行し,

初期画面が表示されることを確認されたい (図 2A).

NEST において, モデル開発者はニューロンに対応するオブジェクトを作成し, オブジェクト同士を関連づけることによりネットワークモデルを定義する. サンプルプログラム 5 “main_nest.py” (表 5) をご覧いただきたい. まず, 4 つのニューロンを作成することにして (`ncell=4`; 5 行目) NEST, *Numpy*, *Matplotlib* 各モジュールをインポートし, 初期設定を行っている (6~10 行目). 積分発火ニューロン “n” を 4 つ生成するとともに (13 行目). “n” 同士を直列にシナプス結合し (14~15 行目), 一つ目のニューロン “n[0]” に 400 pA の定常電流入力を行いつつ (18 行目), 200 ms 間シミュレーションを実行している (30 行目) (図 2B). また, 膜電位とニューロン発火イベントの記録のため, オブジェクト *memb*, *spik* を定義し (20~27 行目; 図 2B), ニューロン膜電位の時間変化と発火時刻をグラフに表示している (32~40 行目). サンプルプログラム 5 を “python main_nest.py” として実行すると, 図 2C に示されるように, 一つ目のニューロン “n[0]” の発火に励起されてニューロン発火が伝わる様をグラフ表示することができる.

より複雑なモデルを開発したい場合には, NEST の Web サイト¹⁵⁾に多様なサンプルプログラムが紹介されているので参照されたい. また, Python コマンドプロンプトより, `dir(nest)` を実行することにより *nest* モジュールが Python に提供する関数の一覧を得ることができる. さらに, 例えば `nest.GetDefaults("iaf_neuron")` を実行することにより *iaf_neuron* ニューロンモデルが持つモデルパラメータの種類と既定値を見ることができる. 各モデルの枠組みの詳細は, \$NEST_SOURCE/models/内の各 *cpp*, *h* ファイルを覗いてみるのが手取り早い. 非常に分かりやすく書かれており, c/c++ の知識が少しでもあれば, モデルの枠組みを理解することができる. もちろん, 独自のニューロンモデル, シナプスモデルを開発し, NEST に取り込むことも可能である.

サンプルプログラム 5 と 3 を比較すると, NEST のプログラムは NEURON のそれと大きく変わらないように見える. しかし, 内部的な構造は大きく異なる. NEURON は, 基本的には時間に関する常微分方程式 (ODE) を解く ODE ソルバである. それに対して, NEST は時間に関するイベント (シナプス伝達) をオブジェクト間で効率的に受け渡すことに主眼をおいて開発されている. NEST は, “resolution”¹⁶⁾で定めら

¹⁵⁾ <http://www.nest-initiative.org/>

¹⁶⁾ サンプルプログラム 5 の 10 行目で指定されている.

図2 NESTを用いたシミュレーション。A:NEST 起動. コマンドプロンプトより Python を実行し, “import nest” コマンドを実行すると, NEST のクレジットが表示される. B: サンプルプログラム 5 により定義されるニューロンネットワーク. C: 単一プロセスでの実行結果 (python main_nest.py). 各パネルは, ニューロン $n[0] \sim n[3]$ の膜電位の時間変化と発火時刻を示す. D: 2 プロセスでの実行結果 (mpirun -np2 python main_nest.py). Rank0 のプロセスには $n[1]$, $n[3]$ が, Rank1 のプロセスには $n[0]$, $n[2]$ が割り当てられた.

れる時間ステップを単位としてオブジェクト間のイベントの受け渡しをする。イベントを受け取ったニューロンオブジェクトは、次の単位時間までの内部状態の遷移を各々独立に計算して内部状態を更新し、必要であればイベントを送信する。イベントの送信には必要に応じて遅延や強度などの変数を伴う。

NEST において、イベント処理のための時間ステップは、ODE ソルバの時間ステップとは異なる。ニューロンオブジェクトが時間に関する ODE を数値的に解く必要がある場合は、独自に Gnu scientific library (GSL) の ODE ソルバを呼び出す仕様になっており、NEST 本体で ODE を解くことはない。このような仕様になっている第一の理由は、多くの抽象モデルニューロンは単に時間に関する指数関数減衰のダイナミクスしか持たない場合が多く、そもそも ODE ソルバを使用する必要がないからである。ODE ソルバを使わなくて良いなら使わない方が計算が速いので、強制的に ODE ソルバを使わなくても良いことになっている。第二の理由は、NEURON 同様、ODE ソルバとイベント処理の時間ステップを分離することにより、イベント

処理を効率的に行うことができるからである。NEST では、単一プロセスのシミュレーションであっても、“resolution”を変更することでシナプス伝達の「時間粒度」を自由に設定することができる。“resolution”を大きくとることでシミュレーションの実行速度は早くなるが、“resolution”より短い遅延のシナプス伝達を行うことはできない。

3.2 並列プログラム実行法

続いて、サンプルプログラム 5 で行ったシミュレーションを MPI 並列で行う方法について紹介する。NEST では、MPI 並列のために新たにプログラムを書く必要はなく、“mpirun -np 2 Python main_nest.py” コマンドを実行するだけで、MPI 並列で nest が実行される。ニューロンオブジェクトの各プロセスへの割り当ては自動で行われる。図 2D は、ニューロン “ $n[0]$ ”, “ $n[2]$ ” が Rank 1 のプロセスに、“ $n[1]$ ”, “ $n[3]$ ” が Rank 0 のプロセスに割り当てられた場合のシミュレーション結果である。

NEST におけるプロセス間のイベント伝達 (シナプス伝達) は、基本的に NEURON と同様である。複数

表 5 サンプルプログラム 5 “main_nest.py”

1	#!/usr/bin/env python
2	
3	# Preprocessing
4	runtime = 200
5	ncell = 4
6	import nest
7	import numpy as np
8	import pylab as pl
9	nest.ResetKernel()
10	nest.SetKernelStatus({"local_num_threads":1,"resolution":0.1})
11	
12	# Definition of neurons and connections
13	n = nest.Create('iaf_neuron', ncell)
14	nest.Connect(n[0:ncell-1],n[1:ncell], 'one_to_one',
15	syn_spec={'weight': 1500.0, 'delay': 5.0})
16	
17	# Current injection
18	nest.SetStatus([n[0]], {"I_e": 400.0})
19	
20	# Recording
21	memb = nest.Create('multimeter', ncell,
22	params = {'withtime': True,
23	'interval': 0.1,
24	'record_from': ['V_m']})
25	spik = nest.Create('spike_detector', ncell)
26	nest.Connect(memb, n, 'one_to_one')
27	nest.Connect(n, spik, 'one_to_one')
28	
29	# Simulation and graph plot
30	nest.Simulate(runtime)
31	fig, ax = pl.subplots(nrows=ncell)
32	tmp = "Rank: " + str(nest.Rank())
33	ax[0].set_title(tmp)
34	pl.xlabel('Time [ms]')
35	for k in range(ncell):
36	pot = nest.GetStatus([memb[k]])[0]['events']
37	ax[k].axis([0, runtime, -100, 0])
38	ax[k].plot(pot['times'], pot['V_m'], linewidth=2)
39	for s in nest.GetStatus([spik[k]], "events")[0]['times']:
40	ax[k].plot([s, s], [-75, -13], color='r', linewidth=2)
41	pl.show()

のプロセスに割り振られたニューロンオブジェクトは、“resolution”で指定された時間を単位に各々独立に処理を行う。すべてのプロセスが単位時間の処理を終了した後、シナプス伝達の「送り手」ニューロンから「受け手」ニューロンにまとめてシナプス伝達を送信される。複数のシナプス伝達の一つのニューロンに与えられる場合はバッファに蓄えられ、必要な時刻に呼び出される。特に大規模なネットワークシミュレーションの場合、シナプス伝達を保存するバッファが大きくなり、メモリ消費の大半を占めるようになる^{3, 11)}。大規模なシミュレーションにおいてメモリ消費を最小限にとどめるため、現在も Connect 関数の改良が行われている^{3, 11)}。

4. ま と め

本稿では、NEURON と NEST の特長、利用方法および並列プログラム方法を簡単なサンプルプログラムを添えて解説した。NEURON はニューロン形状データと実際の計算単位であるコンパートメントを別々に取り扱うことにより、効果的なシミュレーションを行っていた。NEST は、大規模なニューロンネットワークにおけるシナプス伝達の処理を効率的に行うメカニズムを持っていた。

NEURON と NEST、二つのシミュレータを並べて比較すると、恐らくは2つの発展の可能性に思い至る。発展1: NEURON と NEST のプログラムは外見的には似通っているのだから、共通に解釈可能な言語をつくれのだろうか？ 発展2: NEURON と NEST の好みとこ取りシミュレーションはできないだろうか？ 実は、両方ともすでに試みられている。発展1に対応するものとして、NEURON や NEST などの神経シミュレータ共通のニューロンネットワーク言語 PyNN の開発が進められている¹²⁾。Python モジュールとして提供されており、モジュールを import する段階で NEURON を使用するか NEST を使用するか指定するだけで、同じプログラムを各々のシミュレータ上で実行することができる。発展2に対応するものとして、MUSIC インターフェースの開発が進められている¹³⁾。プロセス間のイベント伝達（シナプス伝達）を、プロキシを介することで異なるシミュレータ間で可能にするもので、NEST のネットワークの中で、神経・細胞内シグナル伝達シミュレータ GENESIS^{14, 15)}のモデルニューロンをシミュレーションすることに成功している。ただし、本稿執筆時点で NEURON には実装されていない。また、XML 形式に基づく一般的なモデルニューロン・ネットワーク記述法の開発など、様々なレベルにおけ

るシミュレータ間の連絡の可能性が模索されている¹⁶⁾。

NEURON, NEST というシミュレータは計算論的神経科学という学問分野の二つの大きなニーズに根差しており、業界の縮図でもある。NEURON シミュレータはニューロンの物理的・電気生理学的特性に基づいたシミュレーションの指向を持ち、空間的広がりを持つデンドライトの動的特性をシミュレーションすることを得意とする¹⁷⁾。NEST は抽象ニューロンを用いて大規模ネットワークのシミュレーションを行い、可能であれば準解析的にでも動作原理を解明したい指向を持っている。どちらも、これまでに成功したシミュレーションを容易に再現することができ（例：Synfire chain^{18~20)}）、モデルニューロン・ネットワーク専用に作られたインタプリタ言語は、時に自然言語や数式よりも正確・的確にモデルの定義を伝える。一方、いずれのタイプのシミュレーションについても、批判も多い²¹⁾。ニューロン膜電位依存性チャネルのデータ不足や、ネットワーク結合についての情報の不十分さより、シミュレーションが現実を再現しておらず、実効的な予測性能を持たないのではないかと、とするのが主要内容である。この批判に対しては、明るいニュースがある。アメリカ²²⁾、ヨーロッパ¹⁷⁾、日本¹⁸⁾とも、脳神経系の大型研究プロジェクトが立ち上げられており、それらのなかには、コネクティクスなど脳神経の網羅的で大規模な実験観測を目的とするものがある。十分な実験データに基づいて、シミュレーションによる強い予測ができるようになった時、NEURON, NEST 各シミュレータの価値はさらに飛躍的に上がって行くかもしれない。

謝辞

本稿の執筆は JSPS 科研費 (26540154) および脳科学研究戦略推進プログラム（脳プロ）の助成を受けて行われております。本稿を執筆する機会をお与えくださいました、神経回路学会編集委員の稲垣圭一郎氏（中部大学）に感謝いたします。

付記

本稿で紹介した NEURON, NEST を利用するにあたって、第一の（そして大きな）関門になるのは各ソフトウェアの PC へのインストールである。もちろん NEURON, NEST の Web ページのインストラクションに従ってインストールするだけなのだが、ほとんどの場合において一筋縄ではいかない。注意点をまとめた。

¹⁷ <https://www.humanbrainproject.eu/>

¹⁸ <http://brainminds.jp/>

1. NEURON, NEST 共通

あらかじめ mpi 環境 (NEURON は MPICH2, NEST は openMPI) が正しくインストールされている必要がある。mpicxx コマンドにより、(ヘッダで mpi.h を include した) C/C++ プログラムがコンパイルされ、mpirun -np 2 a.out コマンドにより並列に実行されることを確認されたい。また、環境変数 \$PATH, \$LD_LIBRARY_PATH, \$PYTHON_PATH が正しく設定されていることを確認されたい。複数のコンパイラや、複数バージョンの Python へのパスが混在していたりすると、トラブルの原因となる。

2. NEURON

NEURON の Web サイト^{†9} より iv-VERSION.tar.gz (GUI), nrn-VERSION.tar.gz (本体) 二つのソースコードをダウンロードする。インストラクションにしたがって圧縮されたプログラムを展開し、iv, nrn の順で各々 configure, make, make install を行う。特に nrn の configure の際には "--with-iv=\$IV_DIR --with-paranrn" オプションを付加し、gui と mpi を実行可能な実行形式ファイルを作成する。

3. NEST

あらかじめ GNU scientific library, および Python2.X (2015.7 時点²³) と一連のモジュール numpy, scipy, matplotlib, IPython をインストールする。次に、NEST の Web サイト^{†10} より本体のソースコード nest-VERSION.tar.gz をダウンロードし、インストラクションにしたがって圧縮プログラムの展開、configure, make, make install, make installcheck を行う。configure の際には "--with-mpi" オプションをつけて、mpi で実行可能な実行形式ファイルを作成する。\$HOME/.nestrn ファイルを編集し、環境変数 \$PATH, \$PYTHONPATH を設定する。

インストールでつまづいた時は、NEST メールリストに入会し、アーカイブを参照すると良い。インストール時のトラブルに関するスレッドが多数ある。

なるべく手軽に NEST, NEURON を利用するために、NEST の Web サイトより NEST, NEURON があらかじめインストールされた Ubuntu linux の VirtualBox イメージが提供されている。1) Virtual box と extension pack のインストール, 2) NEST-Ubuntu イメージの起動, 3) Guest addition のインストール (Display 解像度の設定) & 日本語キーボードの設定, により mpi を使用しない設定で NEST, PyNEST,

NEURON, PyNN などを利用することができる。

参 考 文 献

- 1) Hodgkin, A.L., Huxley, A.F. (1952): A quantitative description of membrane current and its application to conduction and excitation in nerve, *J. Physiol.*, Vol.117, No.4, pp.500–544.
- 2) Markram, H. (2006): The blue brain project, *Nat. Rev. Neurosci.*, Vol.7, No.2, pp.153–160.
- 3) Helias, M., et al. (2012): Supercomputers ready for use as discovery machines for neuroscience, *Front. Neuroinform.*, Vol.6, No.26.
- 4) Ananthanarayanan, R., Esser, S.K., Simon, H.D., Modha, D.S. (2009): The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Article No.63.
- 5) Carnevale, N.T., Hines, M.L. (2006): *The NEURON Book*. Cambridge, UK: Cambridge University Press.
- 6) Gewaltig, M.-O., Diesmann, M. (2007): NEST (NEural Simulation Tool), *Scholarpedia*, Vol.2, No.4, p.1430.
- 7) Hines, M.L., Davison, A.P., Muller, E. (2009): NEURON and Python, *Front. Neuroinform.*, Vol.3, No.1.
- 8) 浦久保秀俊, 黒田真也, 相原 威 (2006): NEURON/GENESIS シミュレータから迫る神経科学へスパイクタイミング依存シナプス可塑性の計算論的解析～, *シミュレーション学会誌*, 25 巻 1 号, pp.4–12.
- 9) Hines, M.L., Markram, H., Schurmann, F. (2008): Fully implicit parallel simulation of single neurons, *J. Comput. Neurosci.*, Vol.25, No.3, pp.439–448.
- 10) Hines, M.L., Carnevale, N.T. (2008): Translating network models to parallel hardware in NEURON, *J. Neurosci. Methods*, Vol.169, No.2, pp.425–455.
- 11) Kunkel, S., et al. (2014): Spiking network simulation code for petascale computers, *Front. Neuroinform.*, Vol.8, No.78.
- 12) Davison, A.P., et al. (2008): PyNN: A common interface for neuronal network simulators, *Front. Neuroinform.*, Vol.2, No.11.
- 13) Djurfeldt, M., et al. (2010): Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics*, Vol.8, No.1, pp.43–60.
- 14) Bower, J.M., Beeman, D. (1998): *The Book of GENESIS: Exploring Realistic Neural Models with the General Neural Simulation System*. Second ed., New York: Springer-Verlag.

^{†9} <https://www.neuron.yale.edu/neuron/download/getstd>

^{†10} <http://www.nest-simulator.org/download/>

- 15) Urakubo, H., Honda, M., Froemke, R.C., Kuroda, S. (2008): Requirement of an allosteric kinetics of NMDA receptors for spike timing-dependent plasticity, *J. Neurosci.*, Vol.28, No.13, pp.3310–3323.
- 16) Cannon, R.C., et al. (2007): Interoperability of neuroscience modeling software: current status and future directions, *Neuroinformatics*, Vol.5, No.2, pp.127–138.
- 17) Urakubo, H., et al. (2004): Spatial localization of synapses required for supralinear summation of action potentials and EPSPs, *J. Comput. Neurosci.*, Vol.16, No.3, pp.251–265.
- 18) Diesmann, M., Gewaltig, M.O., Aertsen, A. (1999): Stable propagation of synchronous spiking in cortical neural networks, *Nature*, Vol.402, No.6761, pp.529–533.
- 19) Shinozaki, T., Cateau, H., Urakubo, H., Okada, M. (2007): Controlling synfire chain by inhibitory synaptic input, *J. Phys. Soc. Jpn.*, Vol.76, No.4, pp.44806-1–44806-6.
- 20) Schrader, S., Grun, S., Diesmann, M., Gerstein, G.L. (2008): Detecting synfire chain activity using massively parallel spike train recording, *J. Neurophysiol.*, Vol.100, No.4, pp.2165–2176.
- 21) Fregnac, Y., Laurent, G. (2014): Neuroscience: Where is the brain in the Human Brain Project?, *Nature*, Vol.513, No.7516, pp.27–29.
- 22) Insel, T.R., Landis, S.C., Collins, F.S. (2013): Research priorities. The NIH BRAIN Initiative, *Science*, Vol.340, No.6133, pp.687–688.
- 23) Zaytsev, Y.V., Morrison, A. (2014): CyNEST: a maintainable Cython-based interface for the NEST simulator, *Front. Neuroinform.*, Vol.8, No.23.