博士論文

次世代スーパーコンピュータ環境における
効率的かつ大規模な
詳細神経回路シミュレーション手法に関する研究

東京大学 大学院工学系研究科 先端学際工学専攻

宮本 大輔

2018/02/05

概要

近年，EU の Human Brain Project やアメリカの BRAIN Initiative など，世界的に大規模な全脳スケールの脳研究プロジェクトが進められている．これにより，全脳規模かつ単一神経細胞レベルの実験的データ基盤が整いつつあり，今後，これらの情報を統合的に理解するために，シミュレーションによる再構築や計算機実験が大きな役割を果たすことが期待されている．しかし，実際に全脳スケールでのシミュレーションを行うためには，実験データの整備だけでなく，スーパーコンピュータのような大規模計算資源と，そのような大規模並列環境で高速に動作するシミュレーション環境の構築が不可欠となる．そこで，本研究では，脳の情報処理機構について，単一神経細胞レベルからボトムアップに再構築した全脳スケールモデルを対象に，高効率・高並列なシミュレーションが可能なプラットフォームの構築を行った．

シミュレーションを行う対象としては，単一神経細胞レベルのデータが整備されている生物として，カイコガ (*Bombyx mori*) 及びショウジョウバエ (*Drosophila melanogaster*) を用い，これらに由来するデータベース上の情報を基づき，詳細な細胞形態を再現したマルチコンパートメント Hodgkin-Huxley 型モデルとして大規模神経回路を再構築した．この再構築された大規模神経回路モデルを用い，汎用的な神経細胞・回路シミュレータである NEURON をベースに，CPU 命令レベルの演算性能の高速化・高効率化による単体性能の向上，OpenMP/MPI ハイブリッド並列化による並列性能の向上，細胞分割による演算時間の短縮，細胞形態縮約による演算時間の短縮といった高速化手法の実装を行った．これにより，最終的に，京コンピュータの 663,552 CPU コアを用い，約 10,000 神経細胞の系において，リアルタイムスケールの計算速度でのシミュレーションを実現することに成功した．これは，詳細な細胞形態を有した神経回路シミュレーションとしては，世界最大級の規模となるものである．

また，同時に，次世代のスーパーコンピュタ環境を見据え，GPGPU や，Xeon Phi や PEZY-SC といったメニーコアアーキテクチャについても，本シミュレーション基盤を適用をし，いずれの環境でも高い性能を得ることができた．

本研究により，昆虫の脳情報処理機構解明に向けた全脳シミュレーションの計算機科学

的基盤が大きく整備された．また，本研究は実験データの入手性から昆虫脳を対象としているが，シミュレーション基盤としては，ゼブラフィッシュやヒトなどのより高次な生物についても適用可能であり，将来ヒト全脳シミュレーションに向けた共通のシミュレーション基盤を提供するものである．

目次

概要 1

第 I 部　序論 17

第 1 章　背景 19
1.1　大規模神経回路シミュレーションの意義 19
1.2　昆虫脳モデル 20
1.3　神経細胞形態を用いる意義 21
1.4　リアルタイムシミュレーション 21
1.5　神経回路モデル構築 22

第 2 章　大規模神経回路シミュレーション高速化の先行研究 25

第 3 章　本研究の目的・構成 27

第 4 章　本研究におけるソフトウェア構成 29

第 II 部　手法・計算機・ソフトウェア 31

第 5 章　神経細胞・シナプスモデル 33
5.1　マルチコンパートメント Hodgkin-Huxley 型モデル 33
5.1.1　ケーブル方程式 34
5.1.2　Hodgkin-Huxley 型方程式 34
5.1.3　マルチコンパートメント Hodgkin-Huxley 型モデル 34
5.2　シナプスモデル 35

第 6 章　計算環境 41

6.1 京コンピュータ 41
6.1.1 概要 41
6.1.2 CPU アーキテクチャ 41
6.1.3 ネットワーク構造 42
6.2 FX100 42
6.2.1 概要 42
6.2.2 CPU，メモリアーキテクチャ 42
6.2.3 ネットワーク構造 43
6.3 Suiren (PEZY-SC) 43
6.3.1 概要 43
6.3.2 CPU アーキテクチャ 43
6.4 Xeon Phi 43
6.5 louse (Intel Core i7 計測用マシン) 44

第 7 章 開発を行ったソフトウェア 45
7.1 NEURON K+ 45
7.1.1 概要 45
7.1.2 環境ごとのビルド手順 45
7.1.3 NEURON K+ 専用ビルドオプション 47
7.1.4 仮想化環境 48
7.2 NRC (NeuRal Circuit) Format 49
7.2.1 概要 49
7.2.2 フォーマット定義 50
7.2.3 NRC Format 例 50
7.3 NRC Generator 52
7.3.1 概要 52
7.3.2 使用方法 52
7.4 nrc-loader 52
7.5 細胞形態縮約ツール: reduct_neuron 53
7.5.1 概要 53
7.5.2 縮約手法 53
7.5.3 使用方法 53
7.6 SWC ファイルマッピングツール: NeuroRegister 54
7.6.1 概要 54
7.7 LAL-VPC 領域マッピングツール: LAL-VPC Mapping 54

7.7.1 概要 54
7.7.2 使用方法 55
7.7.3 領域ラベル一覧 55
7.8 カイコガ標準脳シミュレーション環境: SB Simulation 55
7.8.1 概要 55
7.8.2 使用方法 55
7.8.3 可視化 56
7.9 asm-neuron 56
7.9.1 概要 56
7.9.2 使用方法 56
7.10 細胞形態・シミュレーション結果可視化ツール: swc2vtk 57
7.10.1 概要 57
7.10.2 インストール 57
7.10.3 使用方法 57

第 8 章 利用した既存のデータベース・ソフトウェア等 63
8.1 ファイルフォーマット 63
8.1.1 SWC 形式 63
8.1.2 VTK 形式 64
8.2 神経細胞形態データベース 65
8.2.1 Bombyx Neuron Database 65
8.2.2 無脊椎動物脳プラットフォーム / 比較神経科学プラットフォーム 65
8.2.3 FlyCircuit 66
8.3 神経細胞形態抽出 69
8.3.1 KNEWRiTE 69
8.3.2 SIGEN 69
8.4 シミュレーション高速化・並列化 69
8.4.1 MPI 69
8.4.2 OpenMP 70
8.4.3 OpenCL/PezyCL 70
8.5 汎用ベンチマーク 70
8.5.1 STREAM 70
8.5.2 NAS Parallel Benchmark 70
8.6 可視化 71
8.6.1 Paraview 71

第 9 章 データベース情報を元にした神経回路モデルの構築 75
9.1 本研究で構築した神経回路モデル 75
9.2 カイコガ均一神経細胞モデル 77
9.2.1 概要 77
9.2.2 神経回路ネットワーク作成手法 77
9.3 カイコガ LAL-VPC 領域 86 細胞神経回路モデル 81
9.3.1 概要 81
9.3.2 LAL-VPC5 領域モデルによる神経細胞の入出力領域の同定 . . . 81
9.3.3 入出力領域に基づいた Peter's rule によるシナプス作成 81
9.3.4 LAL-VPC 領域 86 細胞モデルの可視化 82
9.4 ショウジョウバエ大規模モデルの構築 94
9.4.1 概要 94
9.4.2 nrc-gen による NRC ファイル生成 94
9.4.3 ショウジョウバエ大規模モデルの可視化 96

第 10 章 神経細胞・回路シミュレーションの高速化・高並列化手法 99
10.1 NEURON 標準の高速化手法の評価 99
10.2 NEURON の単体性能解析 99
10.2.1 ソースコードベースでの演算数・メモリ使用量算出 100
10.2.2 京コンピュータ上のプロファイラによる演算数算出 100
10.3 Hodgkin-Huxley 方程式計算部の単体性能の高速化 100
10.3.1 SIMD 演算の適用 100
10.3.2 配列構造の最適化 107
10.3.3 m, h, n 計算以外の領域の高速化 108
10.4 OpenCL/PezyCL の適用 112
10.5 OpenMP によるハイブリッド並列化 115
10.6 細胞分割による計算時間の短縮 120
10.7 細胞形態縮約による計算時間の短縮 123
10.7.1 枝刈り・短絡手法 123
10.7.2 電気生理学的特性を考慮した枝刈り・短絡手法 123
10.7.3 細胞形態縮約のシミュレーションに与える影響の評価 123
10.7.4 神経回路の縮約手法 125

第 III 部　結果 127

第 11 章　マルチコンパートメント Hodgkin-Huxley 型モデルの計算特性と高速化 129
11.1　マルチコンパートメント Hodgkin-Huxley 型モデルの数値計算手法の比較 129
11.2　NEURON におけるマルチコンパートメント Hodgkin-Huxley 方程式の計算構造と計算時間 130
11.3　NEURON built-in 高速化手法の検証 131
11.4　単体性能の高速化 131

第 12 章　均一な神経細胞による大規模シミュレーション 135
12.1　細胞分割 135
12.1.1　単一細胞の分割結果 135
12.1.2　大規模シミュレーションの細胞分割による計算時間短縮 135
12.2　細胞形態縮約 139
12.2.1　縮約時の形態変化 139
12.2.2　大規模シミュレーションの細胞形態縮約によるリアルタイムシミュレーションの実現 139

第 13 章　不均一な神経細胞による大規模シミュレーション 143
13.1　ショウジョウバエ大規模モデルの解析 143
13.2　ショウジョウバエ神経回路モデルの縮約 144
13.3　ショウジョウバエ神経回路モデルによるストロングスケーリング 147

第 IV 部　結論 151

第 14 章　考察 153
14.1　マルチコンパートメント Hodgkin-Huxley モデルの高速化 153
14.2　大規模計算機上での複数階層での並列化 153
14.3　データベースに基づいたシミュレーション 154
14.4　次世代スーパーコンピュータに向けて 154

第 V 部　業績・謝辞・参考文献 155

業績一覧 157

謝辞 163

参考文献 167

表目次

9.1 本研究で作成した神経回路シミュレーションモデル 76
9.2 Parameters for the benchmark simulation 77

11.1 Number of floating-point operations of the Hodgkin-Huxley part of the benchmark simulation 130

図目次

1.1 TOP500 (Jun. 2017) 23
1.2 CPUDB による CPU クロック性能 23
1.3 アムダールの法則 24

4.1 本研究におけるソフトウェア構成 30

5.1 細胞形態を再現したマルチコンパートメント Hodgkin-Huxley 型モデル . 37
5.2 マルチコンパートメントモデル 37
5.3 分枝がある場合のケーブル方程式 [50] 38
5.4 各シナプスコンダクタンスモデルにおけるコンダクタンス時間的変化 . . 39

7.1 swc2vtk による可視化例（draw_mode=0） 59
7.2 swc2vtk による可視化例（draw_mode=1） 59
7.3 swc2vtk による可視化例（draw_mode=2） 59
7.4 swc2vtk による可視化例（draw_mode=3） 60
7.5 swc2vtk による可視化例（draw_mode=3 with Gourand Shading） . . 60
7.6 swc2vtk によるカイコガ神経細胞可視化例 61

8.1 SWC 形式の例 64
8.2 Bombyx Neuron Database 67
8.3 無脊椎動物脳プラットフォーム (`https://invbrain.neuroinf.jp`) . . 67
8.4 比較神経科学プラットフォーム (`https://cns.neuroinf.jp`) 68
8.5 FlyCircuit (`https://http://www.flycircuit.tw/`) 68
8.6 KNEWRiTE による細胞形態抽出 73

9.1 カイコガ均一神経細胞モデルにおける基準細胞 (BN1056) 80
9.2 カイコガ均一神経細胞モデルにおける 3 種類のネットワークパターン . . 80
9.3 岩野らによる LAL-VPC 領域 Bilateral ニューロンの分類 [63] 83

9.4 岩野らによる LAL-VPC 領域 Unilateral ニューロンの分類 [63] 84
9.5 池野，並木，加沢らによる BoND データベース 8.2.1 上のニューロンと，岩野らによる 5 領域投射モデル [63] の対応 85
9.6 並木による LAL-VPC5 領域の塗り分け 86
9.7 LAL-VPC5 領域モデルによる神経細胞の領域分け結果 1 87
9.8 LAL-VPC5 領域モデルによる神経細胞の領域分け結果 2 88
9.9 Peter's rule によるシナプス作成例 1 89
9.10 Peter's rule によるシナプス作成例 2 90
9.11 Peter's rule によるシナプス作成結果 91
9.12 Peter's rule によるシナプス作成 + ランダムシナプス（$p = 0.1, N = 10$）作成結果 91
9.13 LAL-VPC 標準脳モデルの可視化（細胞ごとの色分け） 92
9.14 LAL-VPC 標準脳モデルの可視化（コンパートメント太さによる色分け） 93
9.15 オスショウジョウバエ大規模神経回路モデル 97
9.16 メスショウジョウバエ大規模神経回路モデル 97
9.17 オスショウジョウバエ大規模神経回路モデル（細胞種類ごと色分け） . . 98
9.18 メスショウジョウバエ大規模神経回路モデル（細胞種類ごと色分け） . . 98

10.1 NEURON の計算構成 101
10.2 Hodgkin-Huxley 方程式計算部の高速化 101
10.3 OpenCL/PezyCL 利用時の概念図 119
10.4 OpenMP によるハイブリッド並列 119
10.5 細胞分割概念図 122
10.6 コンパートメント数縮約手法 1: 末端枝の枝刈り 124
10.7 コンパートメント数縮約手法 2: 分枝のない中間点の短絡 124
10.8 縮約手法のシミュレーションへの影響評価 126

11.1 京コンピュータにおける演算時間評価 132
11.2 単体性能比較 132
11.3 京コンピュータでの Roofline 解析 133
11.4 FX100 での Roofline 解析 133
11.5 Intel i7-6700K での Roofline 解析 134

12.1 BN1056 137
12.2 単一細胞の分割結果 137
12.3 細胞分割による計算時間短縮 138

12.4 細胞縮約による形態変化 140
12.5 細胞分割による形態変化（元形態と手法 1+2 の比較） 141
12.6 細胞分割による形態変化（元形態と手法 1+2 の比較，詳細） 141
12.7 細胞形態縮約による計算時間短縮 142

13.1 オスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム . 145
13.2 メスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム . 145
13.3 オスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム（縮約処理後） 146
13.4 メスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム（縮約処理後） 146
13.5 オスショウジョウバエ大規模モデルのストロングスケーリング 148
13.6 メスショウジョウバエ大規模モデルのストロングスケーリング 148
13.7 オスショウジョウバエ大規模モデルでの Raster Plot 149
13.8 オスショウジョウバエ大規模モデルのストロングスケーリング（スレッドによる細胞分割） 150
13.9 メスショウジョウバエ大規模モデルのストロングスケーリング（スレッドによる細胞分割） 150

Listings

7.1 Build NEURON K+ for PC 45
7.2 Build NEURON K+ for K 46
7.3 Build NEURON K+ for K with Profiler 46
7.4 do_config_k1.sh 46
7.5 do_config_k2.sh 47
7.6 Dockerfile for NEURON K+ 48
7.7 ring.nrc 51
7.8 ring0.nrc 51
7.9 ring4.nrc 51
7.10 Example: nrc-gen.py 52
7.11 Example: swc_list.txt 52
7.12 reduct neuron 53
7.13 swc2vtk Example1 57
7.14 swc2vtk Example2 57
7.15 swc2vtk Example3 58
7.16 swc2vtk Example3 result.dat 58
7.17 swc2vtk Example3 HOC 58
8.1 Example: sample.swc 63
8.2 Example: sample.vtk 64
8.3 Paraview: loaditems.py 71
9.1 リングネットワークの作成 78
9.2 ランダムネットワークの作成 78
9.3 Watts and Strogatz ネットワークの作成 78
9.4 nrc-gen for flycircuit 94
9.5 visualization for flycircuit 95
10.1 `hh.mod` 100

10.2 hh.mod の主要計算部 104
10.3 hh.c の主要計算部 104
10.4 hh.c における各種変数の定義 105
10.5 Hodgkin-Huxley 方程式計算部擬似コード（最適化前） 105
10.6 Hodgkin-Huxley 方程式計算部擬似コード（くくりだしによる最適化） . 106
10.7 マスターテーブルの作成部の擬似コード（最適化前） 107
10.8 マスターテーブルの作成部の擬似コード（順序入れ替えによる最適化） . 107
10.9 Hodgkin-Huxley 方程式計算部擬似コード（配列順序の変更による最適化）107
10.10 `nrn_cur()` 108
10.11 `nrn_jacob()` 110
10.12 `nrn_state()` 110
10.13 Hodgkin-Huxley MOD with OpenCL 112
10.14 `nrn_fixed_step_thread()` 関数の擬似コード 115
10.15 `nrn_fixed_step_thread()` 関数の擬似コード（手法 1） 116
10.16 `nrn_fixed_step_thread()` 関数の擬似コード（手法 2） 116
10.17 `nrn_fixed_step_thread()` 関数の擬似コード（手法 3） 117
10.18 multisplit による細胞分割計算 120
11.1 Call Graph (チューニング前) 130

第 I 部

序論

第 1 章

背景

1.1　大規模神経回路シミュレーションの意義

近年，EU の進める Blue Brain Project[83]/Human Brain Project[21] やアメリカの主導する BRAIN Initiative[3] など，様々な国や研究機関によって全脳規模の脳・神経機能を解明するためのプロジェクトが進んでおり，これらの大規模プロジェクトにより，大量の実験的情報基盤が取得可能になることが予想されている．このような大量の実験的情報を統合的に理解するためには，ボトムアップ的に構築されたシミュレーション及びそのシミュレーションを用いた計算機実験が大きな役割を果たすことが期待されている．しかし，大規模なシミュレーションを行うためには，大量の計算資源が必要となる．この問題を解決するためには，スーパーコンピュータを用いる事が必要不可欠であるが，現在のスーパーコンピュータは，スーパーコンピュータ性能の世界ランキングである TOP500 (Fig. 1.1) に示されるように，依然，ムーアの法則に示されるような指数関数的性能向上を示しているが，性能の基本的な指標の一つである，CPU のクロック周波数については，2 - 4 GHz 程度で頭打ちになっている 1.2．そのため，スーパーコンピュタの性能を押し上げている要因は，命令レベル，CPU コアレベル，ノードレベルの複数の階層における並列度の向上に由来していることがわかる．例を挙げると，京コンピュータであれば，約 70 万 CPU コア，更に新しいアーキテクチャである Sunway TaihuLight では約 1000 万 CPU コアと，大量のコアを用いることで，劇的な性能向上を可能としている．

しかし，この複数の階層における並列性から，十分な性能を引き出すことは容易ではない．この困難さは，一般にアムダールの法則 [5] の拡張として表現される．アムダールの法則は，プログラムの中の一部（構成比率が $p < 1$）を S 倍高速化した際に，全体の性能向上度（SpeedUp）を表したもので，以下のように定式化される．

$$\mathrm{SpeedUp} = \frac{1}{(1-p) + \frac{p}{S}} \tag{1.1}$$

この時明らかなのは，部分的な高速化度 S がたとえ無限大であっても，全体としての性能向上は $\frac{1}{1-p}$ にしかならないと言う事である (Fig. 1.3).

並列計算において，並列度の向上というのは S の上昇に相当するが，プログラムには，逐次的に処理しなければならない部分や通信に関わる部分など，並列度の向上によっても速度が向上しない部分（$1-p$）というのが存在してしまう．そのため，大規模並列計算においては，この $1-p$ の部分をいかに減らすかという事が大きな課題となり，例えば 10,000 並列時に 9,000 倍の性能向上を得るためには，$p > 0.999989$ でなくてはならない．京コンピュータは 663,552 CPU コアを有しているため，p を極めて 1 に近づける必要がある事がわかる．

このような研究の例としては，著名な神経回路シミュレータである [34] を用いて，京コンピュータ上で，17 億神経細胞スケールのシミュレーションを行った例 [39] などがあるが，これらの研究は，細胞の形態を簡略化したシングルコンパートメントモデルが主であり，実際の神経細胞形態を再現できるようなモデルを採用した場合に，計算量がどの程度か，また並列化効率を上げるためにどのような手法が必要なのかについては，まだ未解明である．

そこで，本研究では詳細な神経細胞形態を再現可能なマルチコンパートメント Hodgkin-Huxley 型モデルを用い，モデルの計算的側面から解析を行うとともに，様々な高速化・並列化手法を用いて，10^4 神経細胞のスケールでリアルタイム相当の速度でシミュレーション可能な計算環境の構築を行った．

1.2　昆虫脳モデル

本研究では，高速化・高並列化のターゲットとして，カイコガ (*Bombyx mori*) や，ショウジョウバエ (*Drosophila melanogaster*) といった昆虫の脳モデルを利用することとした．昆虫脳は，神経細胞数 10^5〜10^6 個と，ヒトの 1000 億個に比べると非常に少なく，ある程度網羅的に調べることが可能であり，特にこれらの昆虫については，BoND[] や，FlyCircuit[] といったデータベースに，単一神経細胞レベルの情報が集積されており，全脳スケールのモデルを構築するための非常に良いテストケースとなりうる．

また，昆虫は，神経細胞数が少ないにも関わらず，自然の中で不確定な環境に適応しようとする高次な知能の特徴を多くを有している．一例を挙げると，カイコガにおいて，雄カイコガが，メスに向かっていくフェロモン源探索行動がある [68]．においは空気中には連続的にでなく，乱流などの流体力学的効果が原因で，におい断片の連なりとして存在し

ている．その様な環境の中でフェロモン情報を手掛かりに，左右にジグザグ様にターンを繰り返しながらメスの元に向かうにおい源探索行動は，不確定環境のなかで情報を更新しながらにおいの源に向かう高度な適応行動であると言える．このような匂い源探索行動は，工学的にも非常に高度であり，カイコガがどのように匂い源にたどり着くかというメカニズムについて，神経回路の情報処理の観点から明らかにすることは，工学的にも大きな意義があると考えられる．

1.3　神経細胞形態を用いる意義

神経細胞は，様々な形態を有しており，その形態と機能には大きな関連があると考えられている．特に昆虫においては，idenfied neuron[19] と呼ばれる，特有の形態を持つ巨大な神経細胞が存在しており，学習や情報伝達において，大きな役割を果たしている[37, 9]．このような働きを再現するためには，マルチコンパートメント Hodgkin-Huxley 型モデルのような，細胞内の電位伝搬を記述することの可能なモデルを用いることが必要不可欠となる．

また，それに加え，現在脳内を観測するための大きな手段である様々なイメージング技術では，電気生理学的手法に比べ時間的解像度は低いものの，空間的に広い領域の情報を同時に得ることが可能であり，このような実験結果とシミュレーションを対応させるためには，空間的な広がりを持ったシミュレーションを行うことが重要となる．

1.4　リアルタイムシミュレーション

本研究において，シミュレーション速度の向上を目指す上で，リアルタイムシミュレーションをひとつのランドマークとした．これは，1 秒間のシミュレーションを現実の 1 秒間相当で終えることを意味する．このような目標設定を行った理由としては主に 2 つ挙げることができる．ひとつ目は，脳は，外部からの情報を元に運動パターンを生成する情報処理機であり，このシミュレーションによる動作を検証するためには，実際に複雑な外部環境に晒す必要があるということである．そのため最終的には，シミュレーションをロボットに接続し，リアルタイムに演算を行ったうえで，実環境下で動作させることを想定している．ふたつ目は，シミュレーションを行う際のパラメータ推定について，神経細胞の持つパラメータ空間は非常に広大であるため，データ同化手法を用いて電気生理学的実験と共同させるような系を構築する必要があるということである．このような将来目標に適合させるためには，リアルタイムまたはそれ以上の速度でシミュレーションが動作することが非常に重要となる．

1.5　神経回路モデル構築

本研究では 3 種類の神経回路ベンチマーク用モデルの構築を行うため，既存の様々なソフトウェアやフレームワークを使用または参考にした．まず第一にはデータベースである．神経細胞・回路シミュレーションに関係したデータベースには，神経細胞の形態情報を収集しているものと，シミュレーションモデル等の収集を目的としているものがある．前者には FlyCircuit[16] (`http://www.flycircuit.tw/`)，NeuroMorph[7] (`http://neuromorpho.org`)，そして我々が開発を進めている無脊椎動物脳プラットフォーム (`https://invbrain.neuroinf.jp`) 及び比較神経科学プラットフォーム (`https://cns.neuroinf.jp`) などがあり，後者には，ModelDB[87] や，それをベースにクラウド環境でシミュレーションを実行可能な Simulation Platform[132] が存在しており，実際のモデル構築の活用が可能である．

共焦点レーザー顕微鏡などで撮像された三次元画像から，マルチコンパートメントモデルを構築するためには，まず，画像の中の神経細の領域かそれ以外の領域かを識別するセグメンテーションと呼ばれる工程と，識別された領域について，擬一次的なグラフ構造として抽出を行うトレーシングと呼ばれる工程が必要となる．この目的のために，複数のソフトウェアが制作されており，本研究では，SIGEN[129] と KNEWRiTE[60] を使用したが，Peng らによって開発が進められている Vaa3D[109] 上では，様々なアルゴリズムの実装の共通基盤として用いられており，今後も注目が必要である．

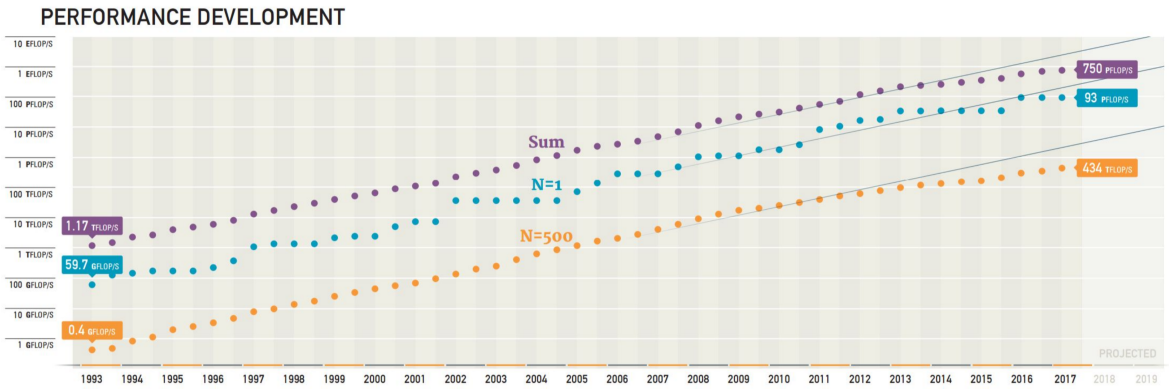


Fig.1.1　TOP500 (Jun. 2017)

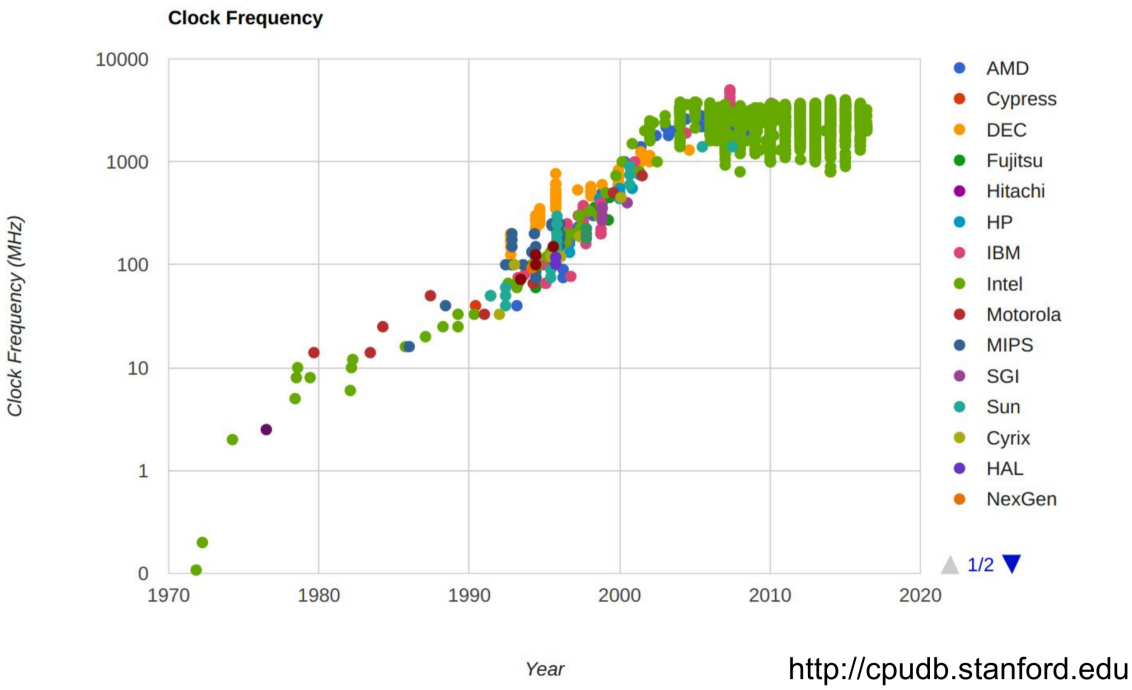


Fig.1.2　CPUDB による CPU クロック性能

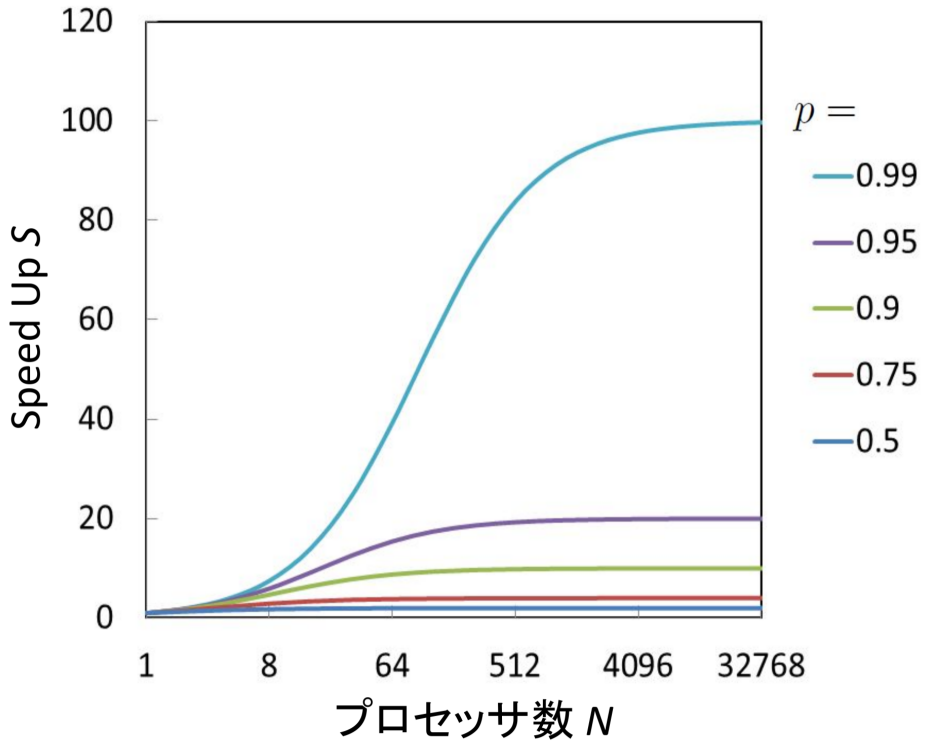


Fig.1.3　アムダールの法則

第 2 章

大規模神経回路シミュレーション高速化の先行研究

大規模神経回路や神経細胞のパラーメータ推定には多大な計算時間を必要とするため，これまでに様々な手法による高速化が提案されてきた．大規模化では，計算量や計算手法の複雑さの観点から，Integrate-and-Fire model や，Izhikevich model のような大幅に簡略化された神経細胞モデルを用いる事が多い．このようなモデルについて現在一般的なのは，MPI (Message Passing Interface) を用いて，PC クラスタやスーパーコンピュータ上で並列化を行うものであり，Diesmann らが開発を進めている NEST simulator (`http://www.nest-simulator.org/`) での実装 [77] がある．彼らのケースでは，メモリ容量がシミュレーション大規模化の阻害要因となっており，神経細胞やシナプスのデータ構造を工夫することで，京コンピュータ上で，10^9 神経細胞スケールのシミュレーションに成功している．また，GPU の高速化と，CUDA や OpenCL といった GPGPU 向けプログラムパラダイムの一般化により，複数のシミュレータが GPGPU での神経細胞シミュレーションを行っている [130, 31, 133]．特に，Nowotny らが開発を行っている GeNN (`https://genn-team.github.io/genn/`)[133] では，細胞モデルの定義を抽象化することで，様々な神経細胞モデルについての CUDA 実装を生成することが可能となっているが，その分実行効率は落ちることとなり，トレードオフの存在を見ることができる．また，GeNN は，Green Brain Project (`http://greenbrain.group.shef.ac.uk/`) として NVIDIA Tegra のような GPGPU の動作するモバイル CPU を利用してドローン上でのリアルタイム神経回路シミュレーションを目指しており，この点からも興味深い．さらに近年では次世代アーキテクチャを見据えたメニーコア CPU での実装も提案されており，小脳のリアルタイムシミュレーション等の事例がある [131]．この研究で使用されている PEZY-SC は，Green500 (`https://www.top500.org/green500/`) と呼ばれる HPL 実行時の消費電力効率のランキングにおいて，高い成績を残しており，今後の消費電力が

課題となるスーパーコンピュータ環境に向けた一つの指針を示していると言える．これらの他にも，FPGA による実装 [107] や ASIC による専用チップ化 [33, 88] が行われている．特に Fuber らの SpiNNaker では，計算部分については汎用の ARM CPU コアを使用しているが，シナプス伝達部について専用のルーティング構造を有しており，ハードウェア化における汎用と専用の切り分け方の一例を見ることができる．

同様に，マルチコンパートメントモデルについても MPI や GPGPU での実装が提案されている．特に Hines らによって開発が進められている NEURON シミュレータの MPI 並列実装については，多くの研究が存在しており，基本的にはスパイク伝達手法の効率化が重要となっている [91, 53, 82]．また彼らは 1 細胞の分割計算についても行っており [52]，同時に複数細胞の分割を行う事例は公開されていないものの，本研究においては大いに参考とした．また GPGPU の事例としては，GPU を用いて 1 細胞を高速に計算するための事例があり [11]，細胞のコンパートメントという階層について，どのように取り組むかが重要と言える．

本研究ではこれらの実装を踏まえ，更なる大規模化に向けたマルチパラダイムでの並列化について研究を行った．

第 3 章

本研究の目的・構成

本研究の最終的な目的は，昆虫全脳スケール (10^4 - 10^5) の神経回路モデルにおいて，リアルタイムと同等の速度でのシミュレーションを実現可能な大規模並列計算プラットフォームを構築することである．

この目的を達成するためのベンチマークモデルとして，以下の 4 種類の神経回路モデルを作成し，シミュレーションの高速化・高並列化のターゲットとした．

- カイコガ均一神経細胞モデル
- カイコガ LAL-VPC 領域 86 細胞モデル
- オスショウジョウバエ大規模モデル
- メスショウジョウバエ大規模モデル

また，シミュレーションの高速化・高並列化を行うためには，既存のプログラムの性能測定が重要となる．そこで，本研究では，マルチコンパートメント Hodgkin-Huxley 型モデルを対象に，浮動小数点演算数や要求メモリバンド幅について，詳細な検討を行った．

そして，これらの情報を元に，京コンピュータや，その後継機である FX100，GPGPU，メニーコアアーキテクチャである Intel Xeon Phi 及び PEZY-SC といった様々なアーキテクチャに対し，プラットフォームの適用を行い，優れたパフォーマンスを得ることを目指した．

さらに，京コンピュータにおいては，大規模シミュレーションを実現するため，OpenMP/MPI によるハイブリッド並列化や，神経細胞形態の複数ノードでの分割計算や，細胞形態の縮約による計算時間の短縮を行い，最終的に，10^4 スケールの神経回路モデルにおいて，リアルタイムスケールでのシミュレーションを達成した．

第 4 章

本研究におけるソフトウェア構成

本研究において開発・使用したソフトウェアの構成を Fig. 4.1 に示す．なお，各ソフトウェアの詳細については，7 章及び 8 章にて述べる．

モデル構築の元となる神経細胞形態データベースとしては，BoND[71] (カイコガ) 及び FlyCircuit[16] (ショウジョウバエ) を使用した．BoND には，共焦点レーザー走査型顕微鏡により取得された三次元画像が登録されているため，KNEWRiTE 及び SIGEN を用いて細胞形態抽出を行い，細胞形態ファイル (SWC) を生成した．FlyCircuit では，すでに細胞形態抽出済みのデータが登録されていたため，これをそのまま用いた．

形態抽出を行ったカイコガ神経細胞については，NeuroRegister (7.6) によりカイコガ標準脳座標系 [60] へマッピングを行った．また，LAL-VPC Mapping (7.7) を用い，カイコガ LAL-VPC5 領域モデル [63] に基づいて神経細胞の各コンパートメントが，どの領域に属しているかの判定を行い，入力領域，出力領域を定義した．この情報は，SB Simulation において，カイコガ LAL-VPC 領域 86 細胞モデルのシミュレーションを行う際に，Peter's rule に基づいて，シナプスの生成を行うのに使用され，実際にシミュレーションを行うこととなる．

また，ランダムなシナプス作成を行うカイコガ均一神経細胞モデル及び，ショウジョウバエ大規模モデルについては，NRC (7.2) という，神経回路定義ファイルを用いて構築を行った．ここでは，細胞数や，細胞リストを元に nrc-gen (7.3) により，NRC ファイルの生成を行う．この時，必要に応じて，細胞の分割や，reduct-neuron (7.5) による細胞形態の縮約を行った．生成された NRC ファイルは，NEURON K+ 用の nrc-loader (7.4) により読み込まれ，シミュレーションを行う．

なおいずれの場合も，シミュレーション結果については，swc2vtk (7.10) により細胞形態及び電位の情報について，VTK ファイルに変換を行い，最終的に Paraview を用いて，3 次元可視化を行った．

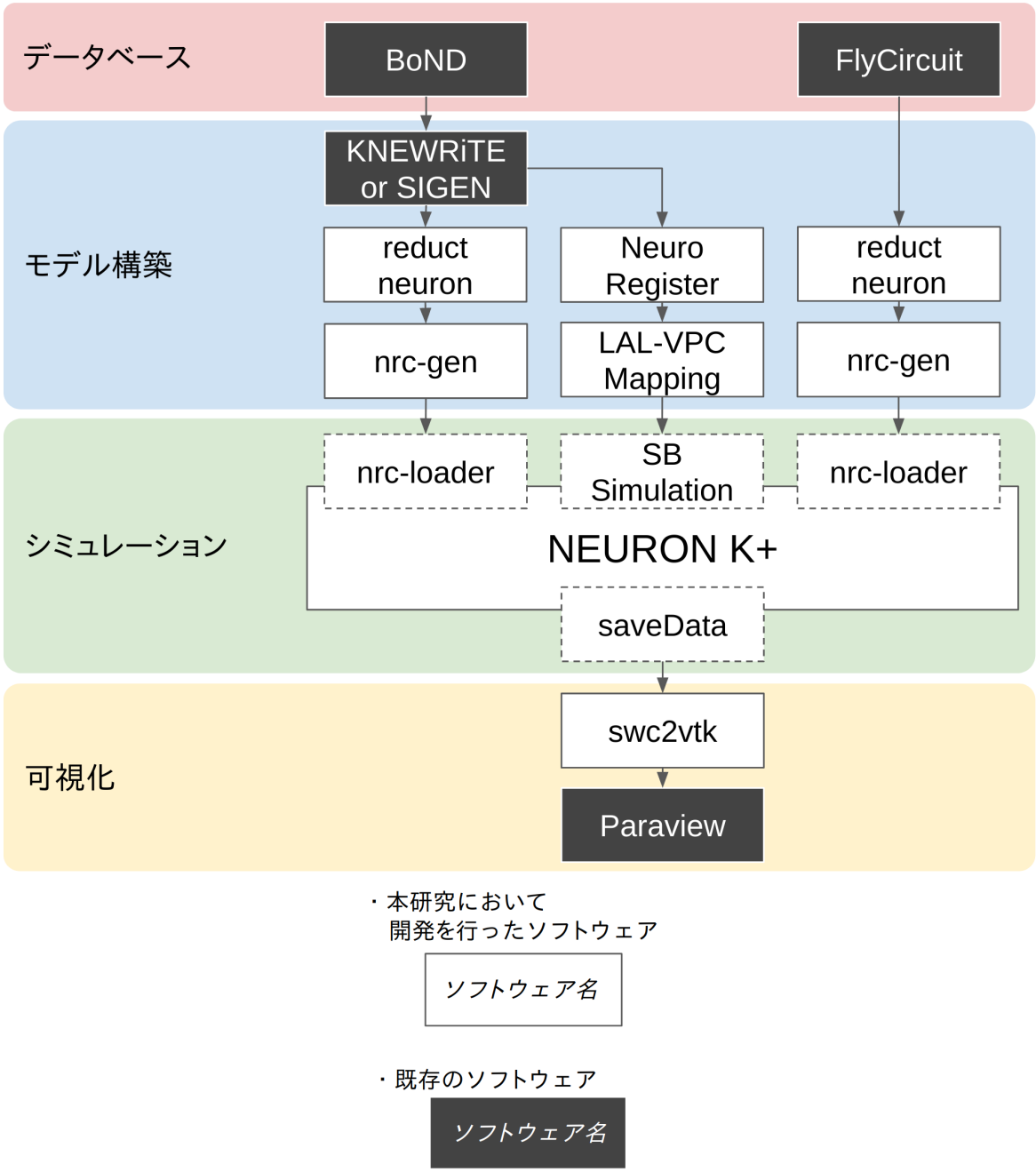


Fig.4.1　本研究におけるソフトウェア構成

第 II 部

手法・計算機・ソフトウェア

第 5 章

神経細胞・シナプスモデル

神経細胞・回路を表現する数理モデルには，様々なものがあり，対象とする現象により多くの選択肢が存在している．一般に神経細胞の形態とその機能には多くの関係があると言われているが，特に昆虫においては，哺乳類などに比べ神経細胞の数が少なく，それは逆に一つ一つの神経細胞の果たす役割が，より複雑で高度であるからだと考えられる．

その根拠の一つとして，昆虫において一つ一つの神経細胞の役割が大きい idenfied neuron[19] とよばれる個々に識別可能であり，多くの場合巨大で特有の形態をもつ神経が多く存在している事が挙げられる．これはカイコガにおいては，におい源探索行動と同期してステアリング信号を出すといわれる G1-G2 下降性神経が存在している [69, 126].

このような状況から，本研究では，細胞での樹状突起中の局所的な相互作用や，細胞内の電位伝播による信号の遅れといったより詳細な情報処理に伴う物理現象を記述できるマルチコンパートメント Hodgkin-Huxley 型モデルを採用する事とした．

またさらに加え，形態的効果を含んだシミュレーションでは，神経細胞を一つの点として表現するようなモデルに比べ計算量が莫大になるため，この様な計算量にどう取り組むべきかを示す事で，計算科学的側面においても意義があると考えた．

5.1 マルチコンパートメント Hodgkin-Huxley 型モデル

マルチコンパートメント Hodgkin-Huxley 型モデルは，電位の伝搬をコンパートメントモデルを使用したケーブル方程式で，各コンパートメントの能動的な電位変化を Hodgkin-Huxley 型方程式で記述したものである．

5.1.1　ケーブル方程式

ケーブル方程式は神経細胞の形態を電気的な等価回路として表現したものであり，ある時刻 t における各点での電位を V とした時に，古典的には以下の熱拡散方程式様の 2 階の偏微分方程式で表される [110].

$$\frac{1}{r_l}\frac{\partial^2 V}{\partial x^2} = c_m \frac{\partial V}{\partial t} + \frac{V}{r_m} \tag{5.1}$$

この式について，一般解を得る事はできないが，離散化し，コンパートメントの連なりとして表現することで，複雑な細胞形状に対しても電位の変化を数値的に求める事が可能となる.

また，細胞形態に分岐がある場合は，Fig. 5.3[50] のように，接続行列として表現することが可能であり，数値計算を行う場合は，ガウス消去法や，CG 法などを用いて解くこととなる.

5.1.2　Hodgkin-Huxley 型方程式

Hodgkin-Huxley 方程式は，1952 年の Hodgkin 及び Huxley の一連の論文 [54] でヤリイカの巨大軸索の電気的特性のモデル化として提案されたものであり，以下の 4 つの微分方程式により表される（この時，C_m は膜容量，$\overline{g_{\mathrm{Na}}}$ と $\overline{g_{\mathrm{K}}}$ はイオンごとの最大コンダクタンス，E_{Na} と E_{K} はイオンごとの平衡電位，g_{L} はリークチャネルのコンダクタンス，E_{L} はリークチャネルの平衡電位，α と β は電位 V の関数である）.

$$C_m \frac{dV}{dt} = I - \overline{g_{\mathrm{Na}}} m^3 h (V - E_{\mathrm{Na}}) - \overline{g_{\mathrm{K}}} n^4 (V - E_{\mathrm{K}}) - g_{\mathrm{L}}(V - E_{\mathrm{L}}) \tag{5.2}$$

$$\frac{dm}{dt} = \frac{m_{\mathrm{inf}}(V) - m}{\tau_m(V)} \tag{5.3}$$

$$\frac{dh}{dt} = \frac{h_{\mathrm{inf}}(V) - h}{\tau_h(V)} \tag{5.4}$$

$$\frac{dn}{dt} = \frac{n_{\mathrm{inf}}(V) - n}{\tau_n(V)} \tag{5.5}$$

5.1.3　マルチコンパートメント Hodgkin-Huxley 型モデル

上述のケーブル方程式について空間的な離散化を行い，コンパートメント化したものについて，各コンパートメント内の電位変化を Hodgkin-Huxley 型方程式で記述すること

で，マルチコンパートメント Hodgkin-Huxley 型モデルを定義することができる．この時，m, h, n のダイナミクスについては，Hodgkin-Huxley 方程式と同様である．

$$\frac{1}{R_a}\frac{\partial^2 V}{\partial x^2} = C_m \frac{\partial V}{\partial t} + I_{HH} \tag{5.6}$$

$$I_{HH} = \overline{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) + \overline{g}_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) + g_{\mathrm{L}} (V - E_{\mathrm{L}}) \tag{5.7}$$

5.2　シナプスモデル

シナプスモデルについては，シナプス情報を送る側であるプレシナプス部位と，シナプス情報を受け取る側であるポストシナプス部位に分けて考えることができる．

プレシナプス部位については，そのシナプスを含むコンパートメントにおいて，電位がある閾値を超えた際にシナプスでの開口放出が起きると定義できる．この閾値は，固定のモデルと可変のモデルがあるが，本研究においては可塑性を考慮せず，主に固定閾値のモデルを用いることとした．

また，ポストシナプス部位については，マルチコンパートメント Hodgkin-Huxley 型モデルにおいて，他のイオンチャネルと同様に，並列に挿入される電位と可変抵抗として表現することができる (式 5.2)．

$$I_{\mathrm{syn}} = g_{\mathrm{syn}} (V - V_{\mathrm{syn}}) \tag{5.8}$$

この時，g_{syn} の時間的変化については，以下のような複数のモデルが提案されている．

- アルファシナプス

$$g_{\mathrm{syn}} = w_1 \times t \exp\left(-\frac{t}{\tau}\right) \tag{5.9}$$

$$\tag{5.10}$$

- 指数減衰型シナプス

$$g_{\mathrm{syn}} = w_2 \times \exp\left(-\frac{t}{\tau}\right) \tag{5.11}$$

$$\tag{5.12}$$

- 二重指数減衰型シナプス

$$g_{\mathrm{syn}} = w_3 \times \left\{ \exp\left(-\frac{t}{\tau_2}\right) - \exp\left(-\frac{t}{\tau_1}\right) \right\} \tag{5.13}$$

本研究においては，主に Exponential Synapse を使用する事とした．
なお，それぞれの g_{syn} モデルにおける時間的変化のグラフを Fig. 5.4 に示す．

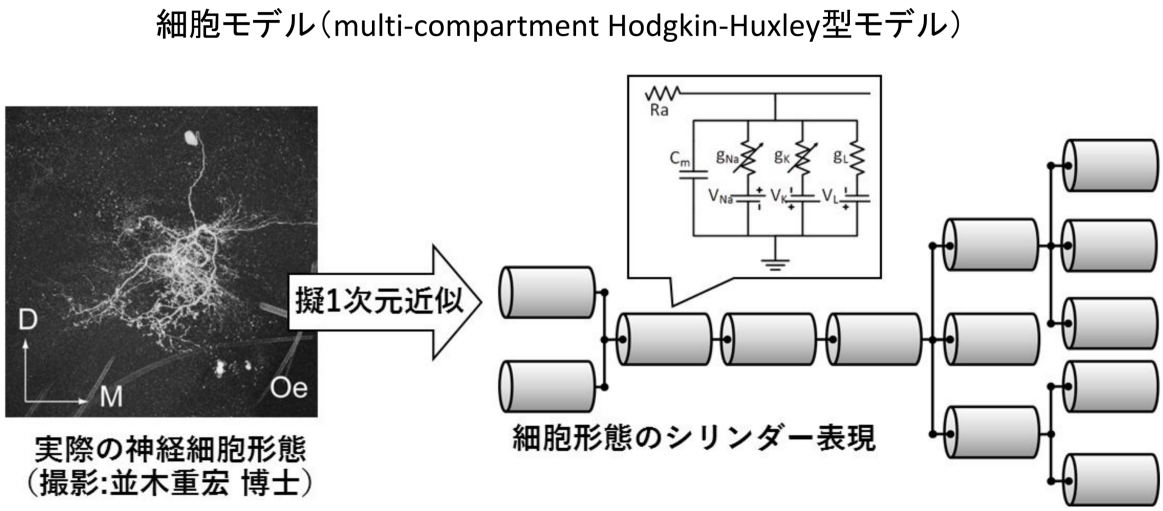


Fig.5.1　細胞形態を再現したマルチコンパートメント Hodgkin-Huxley 型モデル

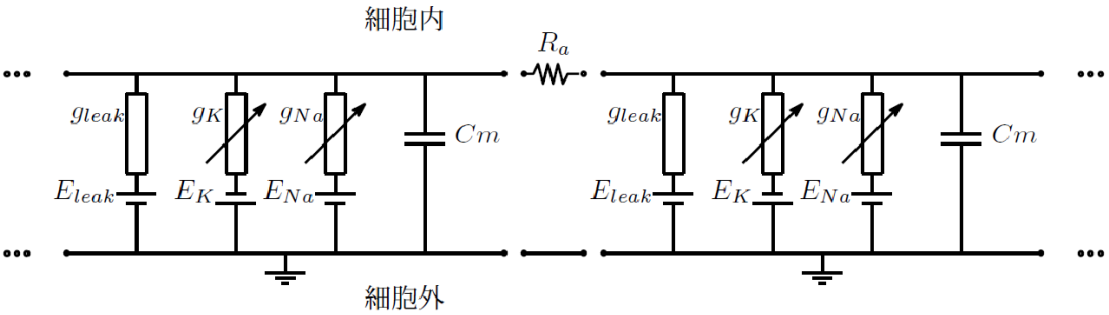


Fig.5.2　マルチコンパートメントモデル

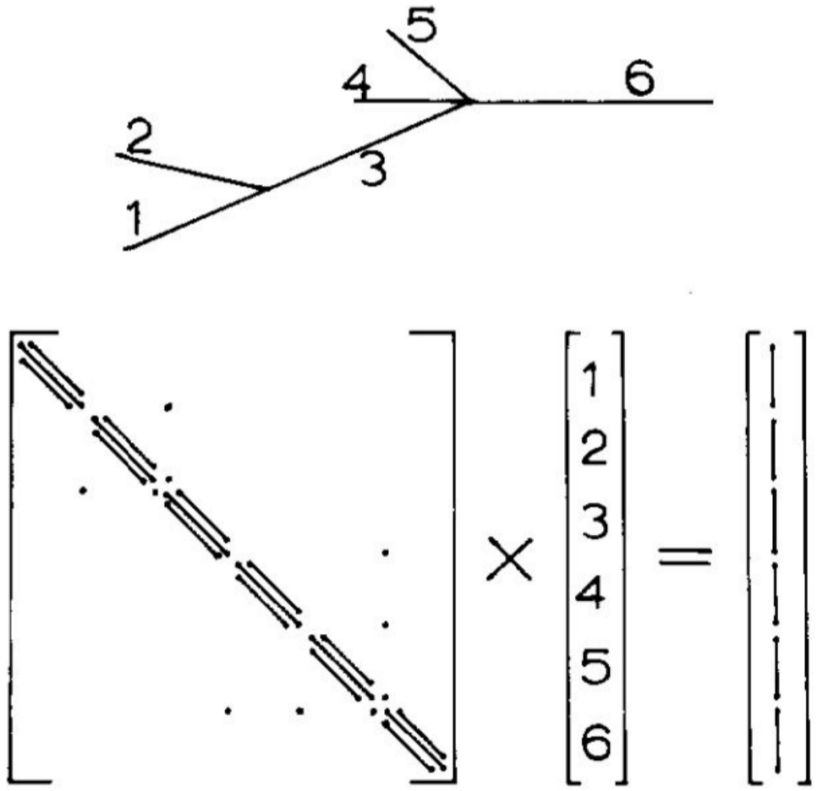


Fig.5.3　分枝がある場合のケーブル方程式 [50]

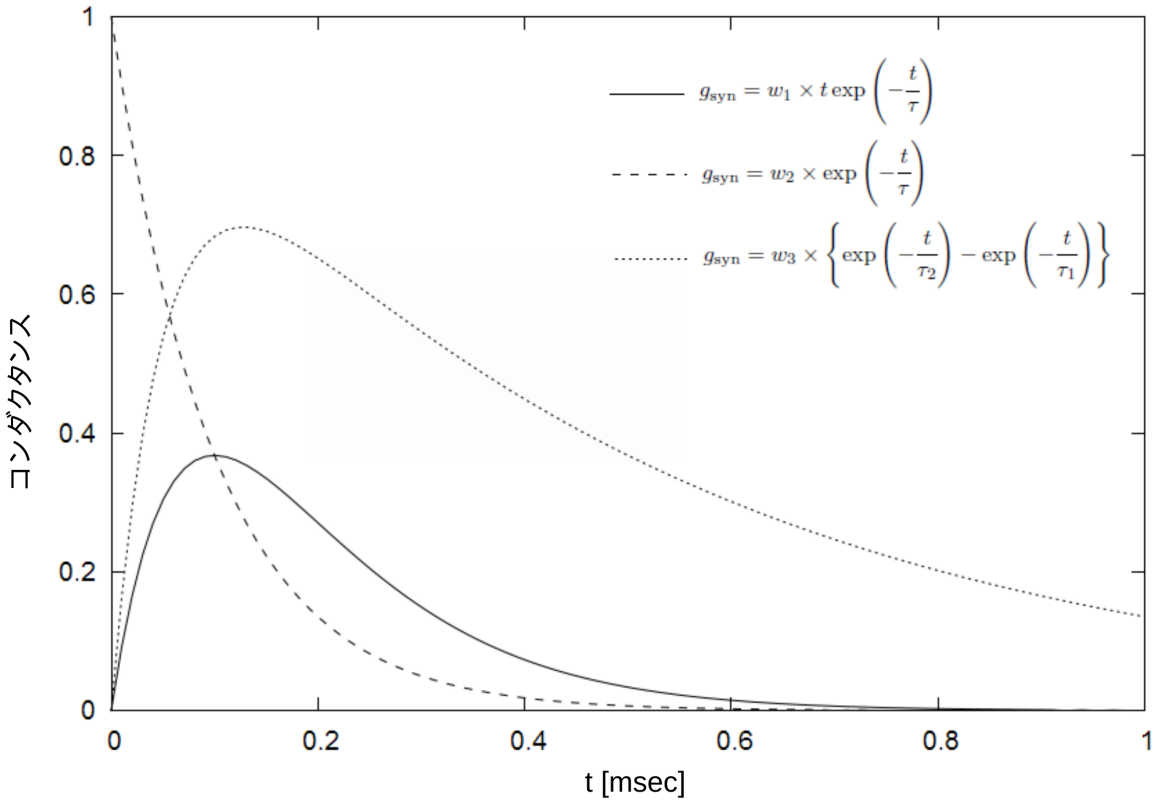


Fig.5.4　各シナプスコンダクタンスモデルにおけるコンダクタンス時間的変化

第 6 章

計算環境

6.1 京コンピュータ

6.1.1 概要

京コンピュータは，独立行政法人理化学研究所 計算科学研究機構（AICS）に設置され，2012 年より運用されているスーパーコンピュータである．

約 1 京 FLOPS（10^{16} FLOPS）の計算性能をもち，2011 年 6 月及び 2011 年 11 月の TOP500[124] では，世界 1 位の性能を記録したほか，2016 年 6 月においても Graph500 において 1 位になるなど，依然高い性能を誇っている．

6.1.2 CPU アーキテクチャ

以下に，京コンピュータで用いられている CPU である SPARC64VIIIfx[118] の概要を示す．着目すべき要素は，クロックが 2 GHz とそこまで高くない代わりに，SIMD (Single Instruction Multiple Data) により最大で 8 浮動小数点演算を同時に行うことで性能を上げていることである．このため，京コンピュータの性能を十分に発揮させるためには，SIMD に最適化された計算方式を用いる事が非常に重要となる．

キャッシュについては，各 CPU コアごとに割り当てられている 1 次キャッシュが 2 way と若干少ないのに対し，CPU コア間で共有されている 2 次キャッシュの分量が多い．そのため，各コアを独立ではなく，お互いに協調して動作させることで，キャッシュのヒット率を上げることができると考えられる．

- 1 チップ（1 ノード） = 8 CPU コア
- クロック：2GHz
- SIMD : 2 × FMA (Fused Multiply-Add) / core

- コアあたり演算性能: 16 GFLOPS/core = 4 Floating-point Operation × 2 SIMD Unit × 2 GHz
- ノードあたり演算性能: 128 GFLOPS/node = 16 GFLOPS/core × 8 core
- 1 次キャッシュ：32 KB 2way （CPU コアごと）
- 2 次キャッシュ：6 MB, 12 way （CPU コア間共通）
- SMP (Symmetric Multiprocessing) 方式
- メモリバンド幅: 64 GB/s (実効 46.6 GB/sec STREAM Triad)
- メモリ容量：16 GB/node

6.1.3　ネットワーク構造

京コンピュータは TOFU[2] と呼ばれるネットワークによって接続されており，ハードウェア的には 6 次元のトーラス構造を持っており，ソフトウェア（MPI）からは任意のサイズで，3 次元トーラスとして，切り出すことができるものである．

6.2　FX100

6.2.1　概要

京コンピュータをベースに，一般販売用に開発されたものであり，本研究では名古屋大学で運用されているものを利用した．アーキテクチャが近いため，京コンピュータと同様のチューニング手法が適用可能であるが，32 CPU コアは，16 + 16 の NUMA 構造となっており，スレッド並列化において注意が必要となる．

6.2.2　CPU，メモリアーキテクチャ

- 1 チップ（1 ノード） = 32 CPU コア
- クロック：2GHz
- SIMD : 2 × 256bit, FMA / core (8 double FMA)
- ノードあたり演算性能: 1.126 TFLOPS/node
- メモリバンド幅: Read: 240 GB/s + Write: 240 GB/s
- STREAM ベンチマーク: Triad 305 GB/s
- メモリ容量: 32 GB/node

6.2.3 ネットワーク構造

- ネットワーク構造: 6 次元メッシュ/トーラス
- インターコネクト: Tofu インターコネクト 2
- インターコネクトバンド幅: 12.5 GB/s × 2

6.3 Suiren (PEZY-SC)

6.3.1 概要

Suiren は，PEZY Computing と ExaScaler によって開発された，PEZY-SC というメニーコアプロセッサを採用したスーパーコンピュータである．液浸冷却システムを採用しており，消費電力性能比に優れているという特徴があり，2014 年 11 月の Green500 では，2 位を獲得した []．

MIMD 型のためコアごとの処理の自由度が高いが，分岐予測がない，In-Order 実行しかないなど，性能を出すためには細かいチューニングが不可欠となっている．

6.3.2 CPU アーキテクチャ

- 1 チップ = 1024 CPU コア
- クロック：733 GHz
- In-order 2way SuperScaler
- ノードあたり演算性能: 1.5 TFLOPS/node (倍精度)

6.4 Xeon Phi

Intel のメニーコアプロセッサである Xeon Phi (KNL) を採用した研究室内マシンである．

- CPU: Intel Xeon Phi(TM) CPU 7250 @ 1.40GHz
- MCDRAM: 16 GB = 2 GB x 8
- Main Memory: 96 GB, 2400 MHz
- Himeno Benchmark: 31 GFLOPS @ 68 core, XL
- STREAM: 40 GB/sec

6.5 louse (Intel Core i7 計測用マシン)

一般的な Intel Core i7 プロセッサのマシンとして，ベンチマークの基準や，一般環境での検証等に使用した．また，GPGPU における高速化についても本マシン上の NVIDIA GTX960 を中心に検証を行った．

- CPU: Intel Core i7-6700K 4.0 GHz
- Memory: DDR4 3000 MHz x 2 (Dual Channel)
- Bandwidth: 48 GB/s
- Himeno Benchmark: 13 GFLOPS @ 8 core , XL
- STREAM ベンチマーク: 26 GB/s (Triad)
- GPU: NVIDIA GTX960

第 7 章

開発を行ったソフトウェア

7.1 NEURON K+

7.1.1 概要

ソースコード: `https://github.com/sc4brain/neuron_kplus` (未公開)

NEURON K+ は，Yale 大学の Hines らによって開発されている神経回路・細胞シミュレーションソフトウェアである NEURON[] をベースに，京コンピュータのような大規模並列環境に適合させるために，著者らによって開発されているソフトウェアである．本論文における試行の多くは，NEURON の拡張機能として実装されており，既存の NEURON 用プログラムを活用することが可能である．

7.1.2 環境ごとのビルド手順

PC 環境でのビルド手順

Listing 7.1 Build NEURON K+ for PC

```
$ cd nrn-7.3
$ ../config/do_config_pc_gcc.sh
$ make
$ make install
$ ../exec/x86_64/bin/nrniv
$ cd ../specials
$ ../exec/x86_64/bin/nrnivmodl ../mod
$ ./x86_64/special
```

京でのビルド手順

クロスコンパイルを行うため，ローカルで動作させる必要のある MOD ファイルの変換用バイナリと，京で動作させる用のバイナリの作成の 2 段階に分けてビルドを行う必要がある．

Listing 7.2　Build NEURON K+ for K

```
$ cd nrn-7.3
$ ../config/do_config_k1.sh
$ make
$ make install
$ ../config/do_config_k2.sh
$ make clean
$ make
$ make install
$ cd ../exec
$ cp ./x86_64/bin/* ./sparc64/bin/
$ cd ./specials
$ ./make_special_sparc64.sh
```

また，NEURON K+ には，計算時間・演算量を測定するために，プロファイラ用のコードが挿入されている．これらを有効化するためには，`do_config_k2.sh` の代わりに，`do_config_k2_fapp.sh` を用いる．

Listing 7.3　Build NEURON K+ for K with Profiler

```
$ cd nrn-7.3
$ ../config/do_config_k1.sh
$ make
$ make install
$ ../config/do_config_k2_fapp.sh
$ make clean
$ make
$ make install
$ cd ../exec
$ cp ./x86_64/bin/* ./sparc64/bin/
$ cd ./specials
$ ./make_special_sparc64.sh
```

なお，この時使用している補助スクリプト（`do_config_k1.sh`，`do_config_k2.sh` は，以下の通りである．

Listing 7.4　do_config_k1.sh

```
#!/bin/bash

./configure --prefix=$(cd ../; pwd)/exec/       \
    --without-iv --without-x --with-nmodl-only  \
```

```
    linux_nrnmech=no \
    CC=gcc CXX=g++
```

Listing 7.5 do_config_k2.sh

```
FLAGS="-Kfast,openmp,ocl,optmsg=2"\
"␣-Nrt_tune,src,sta"\
"␣-DKPLUS␣-DKPLUS_FADVANCE"\
"␣-DKPLUS_USE_FADVANCE_OMP"\
"␣-DARCH_K␣-DKPLUS_SOLVE␣-DKPLUS_TREESET␣-DKPLUS_CAP_JACOB␣-DKPLUS_EION"
    \

./configure --prefix=$(cd ../; pwd)/exec/                              \
    --without-x --without-nmodl                                        \
    --without-nrnoc-x11 --without-x                                    \
    --host=sparc64-unknown-linux-gnu --build=x86_64-unknown-linux-gnu \
    --enable-shared=no --enable-static=yes                             \
    --with-paranrn --with-mpi --with-multisend                         \
    linux_nrnmech=no use_pthread=no                                    \
    CC=mpifccpx CXX=mpiFCCpx MPICC=mpifccpx MPICXX=mpiFCCpx            \
    CFLAGS="${FLAGS}" \
    CXXFLAGS="${FLAGS}" \
```

7.1.3 NEURON K+ 専用ビルドオプション

NEURON K+ では，NEURON にはない複数の機能を，コンパイル時のオプション (CFLAGS, CXXFLAGS) に指定することで，利用可能である．以下に主要なオプションについて記述する．

- KPLUS_K: メモリアロケーション量などを京コンピュータに適した値に設定する他，京コンピュータ向けのビルドであることを明示的に伝える．
- KPLUS_FADVANCE: fadvance() 関数内部での高速化機能を有効化する．
- KPLUS_SOLVE: solve() 関数内部での高速化機能を有効化する．
- KPLUS_TREESET: treeset() 関数内部での高速化機能を有効化する．
- KPLUS_CAP_JACOB: cap(), jacob() 関数内部での高速化機能を有効化する．
- KPLUS_EION: eion() 関数内部での高速化機能を有効化する．
- KPLUS_FADVANCE_OMP: fadvance() 関数内部での OpenMP 利用を有効化する．KPLUS_FADVANCE との併用が必須．
- KPLUS_MOD_OMP: MOD ファイル内部での OpenMP 利用を有効化する．
- KPLUS_USE_FIPP_RANGE: 京コンピュータの簡易プロファイラ (fipp) 用プロファイルレンジ設定．実計算部分のほぼすべてをカバー．

- KPLUS_USE_FAPP_RANGE: 京コンピュータの詳細プロファイラ (fapp) 用プロファイルレンジ設定．レベル 4 まで設定しており，その場合は単一関数レベルの解像度となる．
- KPLUS_GATHER_SCATTER: HOC より呼び出し可能な MPI_Gather，MPI_Scatter 関数の追加 (nrn-7.2 のみ，後藤昂彦氏により開発)
- KPLUS_SPAWN: MPI_Comm_spawn により呼びだされた際に，親ノードと通信するための，関数・変数のセットアップ (nrn-7.2 のみ, 福田哲也氏により開発)

7.1.4　仮想化環境

NEURON のビルドには必要な機能ごとに煩雑な手順が必要であり，また，NEURON 自体の依存関係が膨大であるため，環境によりビルドが失敗することがある．そこで，クラウド環境を含め，様々な環境で NEURON を容易に利用可能にするため，Docker イメージを作成した．

Docker[] インストール済みの環境であれば，以下のコマンドで，OpenMPI 環境でコンパイルされた NEURON K+ を利用することができる．

```
$ docker run -it dmiyamoto/neuron
```

参考のため，以下に Dockerfile を掲載する．

Listing 7.6　Dockerfile for NEURON K+

```
FROM ubuntu:16.04

MAINTAINER Daisuke Miyamoto <miyamoto@brain.imi.i.u-tokyo.ac.jp>

ARG NRN_VERSION="7.4"
ARG NRN_ARCH="x86_64"
ARG NRN_CONFIGURE_OPT="--without-iv␣--with-nrnpython=/usr/bin/python␣--
    with-paranrn␣--enable-static=yes"
ARG NRN_CFLAGS="-O3"
ARG NRN_CXXFLAGS="-O3"


RUN mkdir /work
WORKDIR /work

RUN apt-get update \
    && apt-get install -y \
        locales \
        wget \
        gcc \
        g++ \
        build-essential \
```

```
22          libncurses-dev \
23          python \
24          python-pip \
25          libpython-dev \
26          cython \
27          openmpi-bin \
28          openmpi-common \
29          libopenmpi-dev \
30      && localedef -i en_US -c -f UTF-8 -A /usr/share/locale/locale.alias
            en_US.UTF-8 \
31      && pip install --upgrade pip \
32      && wget http://www.neuron.yale.edu/ftp/neuron/versions/v${
            NRN_VERSION}/nrn-${NRN_VERSION}.tar.gz \
33      && tar xvzf nrn-${NRN_VERSION}.tar.gz \
34      && rm nrn-${NRN_VERSION}.tar.gz \
35      && cd nrn-${NRN_VERSION} \
36      && ./configure --prefix=`pwd` ${NRN_CONFIGURE_OPT} CFLAGS=${
            NRN_CFLAGS} CXXFLAGS=${NRN_CXXFLAGS} \
37      && make \
38      && make install \
39      && rm -rf /var/lib/apt/lists/* \
40      && apt-get autoclean
41
42  RUN cd /work/nrn-7.4/src/nrnpython \
43      && python setup.py install
44
45  RUN useradd -m neuron
46  USER neuron
47
48  ENV LANG en_US.utf8
49  ENV PATH $PATH:/work/nrn-7.4/x86_64/bin
```

7.2 NRC (NeuRal Circuit) Format

7.2.1 概要

NEURON は，データとプロセスが一体となっている言語のため，大規模なシミュレーションを行うためには，データ部分の分離が必要不可欠である．

近年では，NeuroML[35] のような XML をベースとした形式が用いられる事も増えてきており，NEURON にも ython ベースの，読み込み関数が提供されている．しかし，NeuroML はそのままでは，並列環境で細胞の CPU コアへの割り当てを指定することができない点や，CPU コアごとにファイルを分割する仕様がないためステージングで大量のデータ転送が発生してしまう，といった問題が存在する．

そこで，本研究では，新たに NRC (NeuRal Circuit) Format という神経回路記述フォー

マットを定義した．NRC Format は，大規模シミュレーションを想定し，細胞形態やシナプス情報などを記述するためのフォーマットであり，単一ファイルとしても利用できる他，京コンピュータの様なディスク分散環境向けに，各ノード用のファイルに分割して用いることも可能である．

7.2.2　フォーマット定義

NRC Format の各項目は以下の通りとなる．なお，コンパートメント位置について**-2** を指定した場合は細胞内のランダム位置となる．

- `#`: #以降はコメントとなる
- `CELL`: 細胞定義．引数は順に，ノード ID，細胞 ID，細胞形態ファイル名，NetCon のコンパートメント位置，3 次元空間上の位置 (シミュレーションには無関係) を指定する．
- `STIM`: 刺激定義．引数は順に，細胞 ID，コンパートメント位置，刺激数，刺激間隔，刺激開始時刻を指定する．
- `PRESYN`: プレシナプス定義．引数は順に，シナプス ID，細胞 ID，コンパートメント位置を指定する．同一シナプス ID を持つ `POSTSYN` と接続される．
- `POSTSYN`: ポストシナプス定義．引数は順に，シナプス ID，細胞 ID，コンパートメント位置，シナプス強度を指定する．同一シナプス ID を持つ `PRESYN` と接続される．
- `BARRIER`: 全てのノードで同期を行う．並列実行の際は `POSTSYN` を作成する前に同期を行う必要がある．

NRC ファイルを作成する際には，以下の点に注意が必要となる．

- 細胞 ID とシナプス ID は，内部では共通であり，重複はエラーとなる．
- 分割ファイルモードで利用する場合は，`STIM`，`PRESYN`，`STIM` は，指定した細胞 ID が存在しているのと同じファイルに記述する必要がある．
- シナプス定義は，全てのファイルにおいて，`PRESYN`，`BARRIER`，`POSTSYN` の順に記述する必要がある．並列出ない場合は `BARRIER` は不要である．

7.2.3　NRC Format 例

以下に，8 個細胞を定義し 8 ノード上に置く場合に，それぞれ順に 0 番細胞が 1 番細胞に，1 番細胞が 2 番細胞に，とリング状にシナプスを作成する NRC ファイルについて，

単一ファイルの場合と，ノードごとの分割ファイルとした場合の 0 番 4 番ノード向けファイルについて掲載する．

Listing 7.7 ring.nrc

```
CELL 0 0 test0.swc -2 0 0 0
CELL 0 1 test1.swc -2 0 0 0
CELL 0 2 test2.swc -2 0 0 0
CELL 0 3 test3.swc -2 0 0 0
CELL 0 4 test4.swc -2 0 0 0
CELL 0 5 test5.swc -2 0 0 0
CELL 0 6 test6.swc -2 0 0 0
CELL 0 7 test7.swc -2 0 0 0
STIM 0 -2 100 100 2.000000
PRESYN 8 0 -2
PRESYN 9 1 -2
PRESYN 10 2 -2
PRESYN 11 3 -2
PRESYN 12 4 -2
PRESYN 13 5 -2
PRESYN 14 6 -2
PRESYN 15 7 -2
BARRIER
POSTSYN 15 0 -2 0.050000
POSTSYN 8 1 -2 0.050000
POSTSYN 9 2 -2 0.050000
POSTSYN 10 3 -2 0.050000
POSTSYN 11 4 -2 0.050000
POSTSYN 12 5 -2 0.050000
POSTSYN 13 6 -2 0.050000
POSTSYN 14 7 -2 0.050000
```

Listing 7.8 ring0.nrc

```
CELL 0 0 5HT1A-M-100028.swc -2 0 0 0
STIM 0 -2 100 100 2.000000
PRESYN 8 0 -2
BARRIER
POSTSYN 15 0 -2 0.050000
```

Listing 7.9 ring4.nrc

```
CELL 4 4 5-HT1B-M-000001.swc -2 0 0 0
PRESYN 12 4 -2
BARRIER
POSTSYN 11 4 -2 0.050000
```

7.3 NRC Generator

ソースコード: `https://github.com/DaisukeMiyamoto/nrc-gen/` (未公開)

7.3.1 概要

任意の細胞数，細胞形態ファイル (SWC)，シナプス数，ノード数について，NRC ファイルを生成するための Python モジュールを作成した．

7.3.2 使用方法

以下に，Source Code 7.2.3, 7.2.3 の例を出力するようなサンプルを示す．この時，`swc_list.txt` は，定義する細胞数と同じ行数のテキストファイルであり，使用する細胞形態ファイル (SWC) 名を順に記述する．

Listing 7.10　Example: nrc-gen.py

```
nrcg = nrcgen.NrcGenerator(8)
nrcg.add_stim(1)
nrcg.add_ring_synapse(1)
nrcg.append_swc_file_list('./swc_list.txt')
nrcg.write('./ring8', 8)
```

Listing 7.11　Example: swc_list.txt

```
5HT1A-M-100028.swc
5HT1A-M-100029.swc
5HT1A-M-700001.swc
5-HT1B-M-000000.swc
5-HT1B-M-000001.swc
5-HT1B-M-000002.swc
5-HT1B-M-000003.swc
5-HT1B-M-000004.swc
```

7.4 nrc-loader

NEURON/NEURON K+ において，NRC フォーマットを取り扱うためのモジュールである nrc-loader を作成した．

7.5 細胞形態縮約ツール: reduct_neuron

7.5.1 概要

ソースコード: `https://github.com/DaisukeMiyamoto/reduct_neuron`

細胞の大域的な形態を維持しながら，コンパートメント数を縮約するためのツールである，reduct_neuron を開発した．

7.5.2 縮約手法

本ツールでは，

- 手法 1 枝刈り: 末端枝の削除
- 手法 2 短絡: 分枝のない中間点の短絡

という 2 種類のコンパートメント数縮約手法を繰り返して適用することで，大域的な形状を維持したまま，コンパートメント数を減らす実装になっている．

7.5.3 使用方法

Listing 7.12 reduct neuron

```
from swc import Swc

filename = 'hogehoge.swc'

swc = Swc(filename=filename, set_fingerprint=1)
print "%s : cmp=%d length=%.1f [um] vol=%.1f [um^3]" % (swc.get_filename
    (), swc.get_n_cmp(), swc.get_total_length(), swc.get_total_volume())


filename1 = filename.replace('.', '-1.', 1)
swc.reduct1()
swc.write(filename1)
print "%s : cmp=%d length=%.1f [um] vol=%.1f [um^3]" % (swc.get_filename
    (), swc.get_n_cmp(), swc.get_total_length(), swc.get_total_volume())

filename2 = filename1.replace('.', '-2.', 1)
swc.reduct2()
swc.write(filename2)
print "%s : cmp=%d length=%.1f [um] vol=%.1f [um^3]" % (swc.get_filename
    (), swc.get_n_cmp(), swc.get_total_length(), swc.get_total_volume())

filename3 = filename2.replace('.', '-1.', 1)
```

```
swc.reduct1()
swc.write(filename3)
print "%s␣:␣cmp=%d␣length=%.1f␣[um]␣vol=%.1f␣[um^3]" % (swc.get_filename
    (), swc.get_n_cmp(), swc.get_total_length(), swc.get_total_volume())

filename4 = filename3.replace('.', '-2.', 1)
swc.reduct2()
swc.write(filename4)
print "%s␣:␣cmp=%d␣length=%.1f␣[um]␣vol=%.1f␣[um^3]" % (swc.get_filename
    (), swc.get_n_cmp(), swc.get_total_length(), swc.get_total_volume())
```

7.6　SWC ファイルマッピングツール: NeuroRegister

7.6.1　概要

ソースコード: `https://github.com/sc4brain/neuroregister`

細胞の LSM 画像を標準脳座標系 [] にマッピングする際の変換情報を用いて，細胞形態ファイル (SWC) も同様の標準脳座標系にマッピングするための Fiji/ImageJ[] プラグインである．

開発には，細胞の LSM 画像を標準脳座標系 [] にマッピングするためのプラグインである，VIB[] プラグインセットの，`Name Landmarks and Register` をベースとしており，Affine 変換や Thin-Plate Spline などの複数のレジストレーション手法を用いて細胞形態ファイル (SWC)　を標準脳座標系にマッピングすることが可能となっている．

レジストレーション対象となるカイコガ標準脳については，池野らの作成したものを用いた [60].

7.7　LAL-VPC 領域マッピングツール: LAL-VPC Mapping

7.7.1　概要

ソースコード: `https://github.com/sc4brain/LAL-VPCmapping` (未公開)

カイコガ LAL-VPC5 領域モデルをベースに，ある細胞形態ファイル（SWC）の各コンパートメントが，どの領域に属しているかを判定し，SWC ファイル内の，ラベルカラムに書き込むツールの開発を行った．

立命館大学西川研究室 (当時) の加藤のプログラムをベースに，判定アルゴリズムの修正や他のソフトウェアとの連携，バグ修正等を行った．

7.7.2 使用方法

```
$ python ./coloring_swc.py [INPUT SWC FILENAME] [OUTPUT SWC FILENAME]
```

なお，領域塗り分け画像 `LAL12345.tif` を使用する．

7.7.3 領域ラベル一覧

- 0: not match any region
- 1: left side region1
- 2: left side region2
- 3: left side region3
- 4: left side region4
- 5: left side region5
- 6: right side region1
- 7: right side region2
- 8: right side region3
- 9: right side region4
- 10: right side region5

7.8 カイコガ標準脳シミュレーション環境: SB Simulation

7.8.1 概要

ソースコード: `https://github.com/sc4brain/standardBrainSimulation` (未公開)

LAL-VPC 領域マッピング情報を元に，東京大学神崎・高橋研究室 (当時) の森友亮のプログラム [] をベースに，同所属の福田哲也とともに開発を行った．なお，実行には NEURON K+ が必要となる．

7.8.2 使用方法

本プログラムは，シナプス生成と，実際のシミュレーションの二段階で実行を行う．

シナプス生成

```
$ nrnivmodl ./use_mod
```

```
$ python getSynList_LALVPC.py
```

シミュレーション

```
$ nrniv networkSimulation.hoc
```

7.8.3　可視化

本プログラムには，シナプス接続や，シミュレーション結果を可視化するためのスクリプトが含まれている．3 次元可視化には，`swc2vtk` (7.10 参照) を使用している．

- `utils/draw_synconnection.py`: synlist 内のシナプス強度をヒートマップ表示する．
- `utils/synapse2vtk.py`: 任意の synlist 内のシナプスファイルの標準脳上の位置について表示する
- `utils/visualize_simulation.py`: シミュレーション結果の表示

7.9　asm-neuron

7.9.1　概要

ソースコード: `https://github.com/DaisukeMiyamoto/asm-neuron`

ARM 環境において，NEON 命令のような SIMD 演算の効果を測定するため，アセンブラによる Izhikevich モデル及び hodgkin-Huxley モデルの実装を行った．

7.9.2　使用方法

x86 環境

```
$ cd src $ make $ ./iz.out $ ./hh.out
```

ARMv8 (aarch64)

```
$ cd src $ make arm $ ./iz_arm.out  $ ./hh_arm.out
```

7.10　細胞形態・シミュレーション結果可視化ツール: swc2vtk

7.10.1　概要

ソースコード: `https://github.com/DaisukeMiyamoto/swc2vtk`

ドキュメント: `http://daisukemiyamoto.github.io/swc2vtk/`

大規模な科学的可視化には，Paraview[] と呼ばれるソフトウェアが一般的に用いられる．このソフトウェアは，VTK 形式の入力ファイルであれば，汎用的に扱うことが可能なため，細胞形態ファイル (SWC) やシミュレーション結果について，VTK 形式に変換するためのツールの開発を行った．

7.10.2　インストール

本ソフトウェアは，PyPi (`https://pypi.python.org/`) により公開済みである．そのため，インストールには `pip` コマンドを利用可能である．

```
$ pip install swc2vtk
```

7.10.3　使用方法

単一ニューロンの VTK 化

Listing 7.13　swc2vtk Example1

```
import swc2vtk
vtkgen = swc2vtk.VtkGenerator()
vtkgen.add_swc('simple.swc')
vtkgen.write_vtk('simple.vtk')
```

複数ニューロンの VTK 化

Listing 7.14　swc2vtk Example2

```
import swc2vtk
vtkgen = swc2vtk.VtkGenerator()
vtkgen.add_swc('simple.swc')
vtkgen.add_swc('simple1.swc')
vtkgen.add_swc('simple2.swc')
vtkgen.write_vtk('simple.vtk')
```

シミュレーション結果の可視化

Listing 7.15　swc2vtk Example3

```
import swc2vtk
vtkgen = swc2vtk.VtkGenerator()
vtkgen.add_swc('simple.swc')
vtkgen.add_datafile('result.dat')
vtkgen.write_vtk('simple.vtk')
```

Listing 7.16　swc2vtk Example3 result.dat

```
-65.0
-65.0
-65.0
-65.0
```

また，このような電位情報を出力するための，NEURON K+ での HOC コードの例は以下のようになる．

Listing 7.17　swc2vtk Example3 HOC

```
proc saveData() { local i localobj outfile strdef filename
    filename = $s1

    printf("filename:␣%s\n", filename)
    outfile = new File()
    outfile.wopen(filename)
    for(i=0; i<SectionNum; i=i+1){
          access Dend[i]
        outfile.printf("%f\n", v)
    }
    outfile.printf("\n")
    outfile.close()
}
```

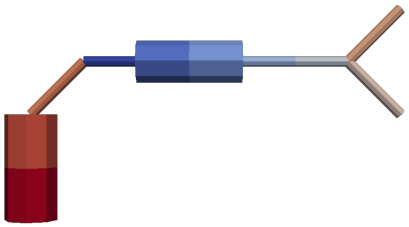


Fig.7.1　swc2vtk による可視化例（draw_mode=0）

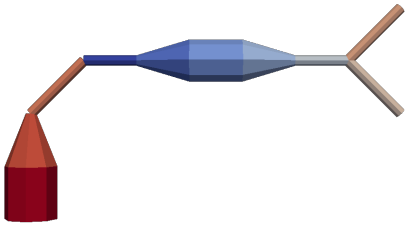


Fig.7.2　swc2vtk による可視化例（draw_mode=1）

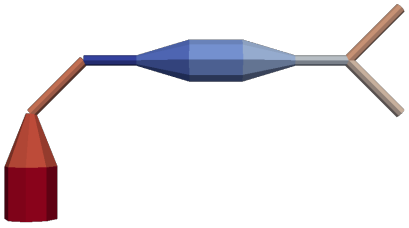


Fig.7.3　swc2vtk による可視化例（draw_mode=2）

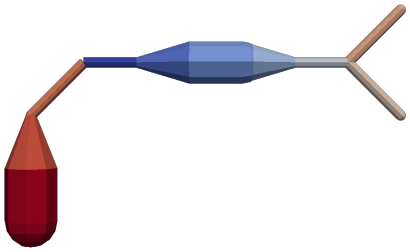


Fig.7.4　swc2vtk による可視化例（draw_mode=3）

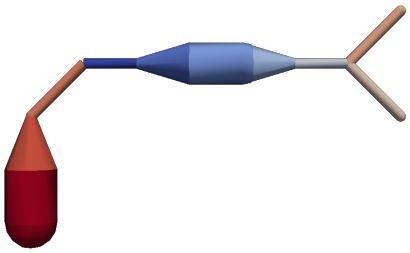


Fig.7.5　swc2vtk による可視化例（draw_mode=3 with Gourand Shading）

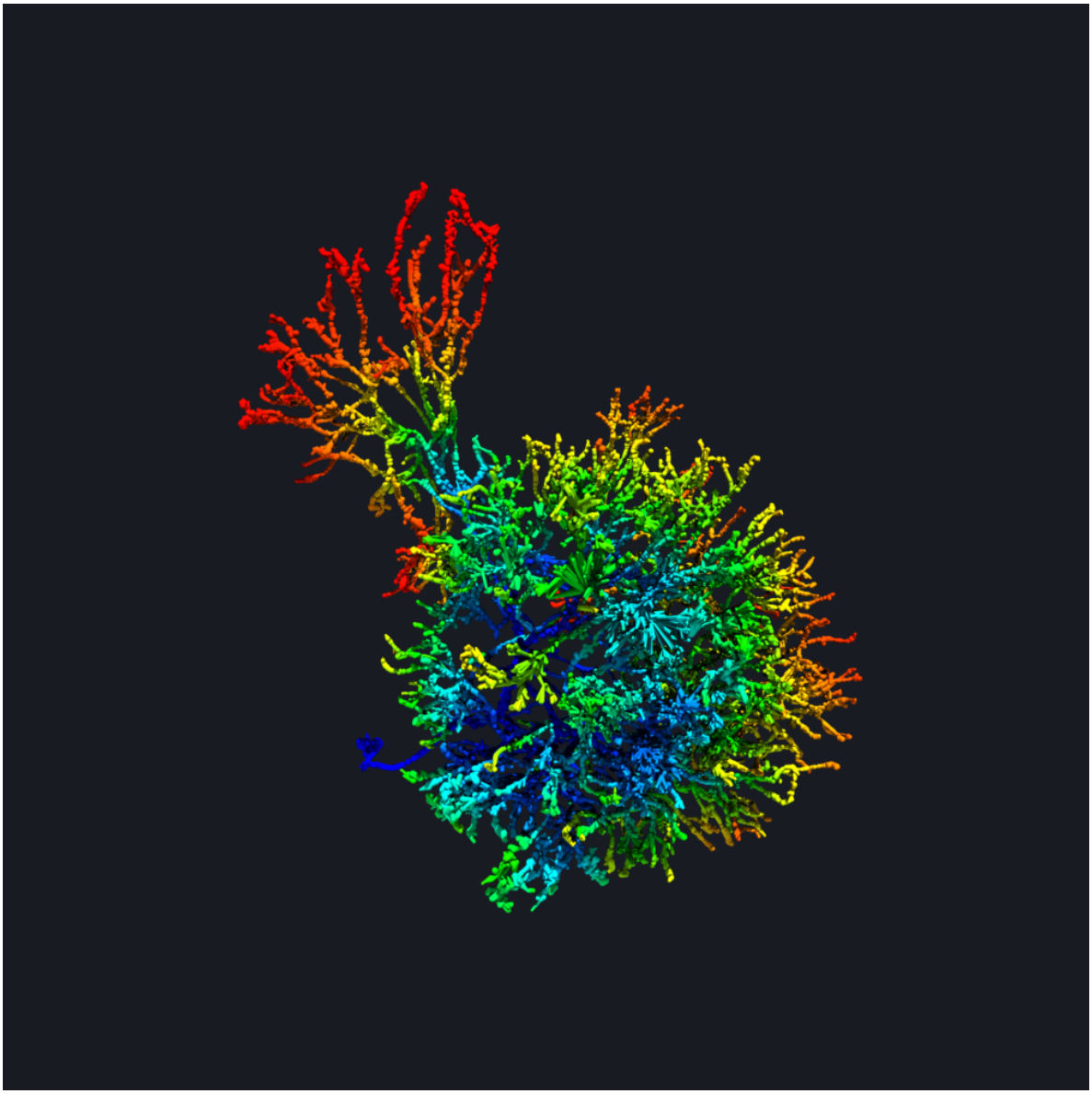


Fig.7.6　swc2vtk によるカイコガ神経細胞可視化例

第 8 章

利用した既存のデータベース・ソフトウェア等

8.1 ファイルフォーマット

8.1.1 SWC 形式

シミュレーションに用いる細胞形態の記述フォーマットとして，SWC 形式を使用した．

SWC は，提案者である，E.W. Stockley，H.V.Wheal，H.M. Cole の頭文字から名付けられ，細胞形態を円柱状のコンパートメントの連なりとして表現することができる．

以下に，SWC 形式の例を示す．

Listing 8.1 Example: sample.swc

```
#ORIGINAL_SOURCE IOSSIM
#CREATURE
#REGION
#FIELD/LAYER
#TYPE
#CONTRIBUTOR
#REFERENCE
#RAW
#EXTRAS
#SOMA_AREA
#SHINKAGE_CORRECTION 1.000000 1.000000 1.000000
#VERSION_NUMBER
#VERSION_DATE 2018-03-31
#SCALE 1.0 1.0 1.0

1 0 100.0 100.0 100.0 1.0 -1
2 0 200.0 100.0 100.0 1.0 1
3 0 300.0 150.0 100.0 1.0 2
4 0 300.0 50.0 100.0 1.0 2
```

この時，最初の#から始まる部位はヘッダーである．この中では#SCALE が特に重要であり，SWC ファイルで記述されている数字が，X 方向，Y 方向，Z 方向それぞれに対して何 μ 向に相当するかが記述されている．この例では，X，Y，Z 共に 1.0 なので，この後に記述されている数字は，そのまま μ のとして読むことができる．次にヘッダーが終わったあとのデータ列については，順に ID， タイプ， X 座標， Y 座標， Z 座標， 直径，親 ID となっている．この SWC ファイルの XY 空間状の形状は，Fig. 8.1 となる．

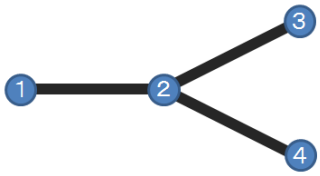


Fig.8.1　SWC 形式の例

8.1.2　VTK 形式

可視化用の汎用ファイルフォーマットとして，VTK を使用した．VTK には，複数のデータセット形式があるが，今回は非構造データを記述するのに向いた，UNSTRUCTURED GRID タイプを主に使用した．

以下に例を示す．

Listing 8.2　Example: sample.vtk

```
# vtk DataFile Version 3.0
SWC2VTK
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 56 float
1.000000 0.000000 0.000000
1.000000 0.000000 0.000000
1.000000 0.000000 0.000000
1.000000 0.000000 0.000000
1.000000 0.000000 0.000000
~~

CELLS 5 95
18 0 8 1 9 2 10 3 11 4 12 5 13 6 14 7 15 0 8
18 8 16 9 17 10 18 11 19 12 20 13 21 14 22 15 23 8 16
18 16 24 17 25 18 26 19 27 20 28 21 29 22 30 23 31 16 24
18 24 32 25 33 26 34 27 35 28 36 29 37 30 38 31 39 24 32
18 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 40 41

```

```
CELL_TYPES 5
6
6
6
6
6

CELL_DATA 5
SCALARS data float 1
LOOKUP_TABLE default
0.0
0.0
0.0
0.0
0.0
```

8.2 神経細胞形態データベース

8.2.1 Bombyx Neuron Database

Bombyx Neuron Database (BoND) [71] は，東京大学先端科学技術研究センター神崎・高橋研究室内で運用されている，カイコガの形態情報や各種実験結果を登録するためのデータベースである．当システムは，コンテンツマネジメントシステムである，XOOPS (`https://www.xoops.org/`) 上のモジュールである，Cosmo DB (`http://cosmodb.osdn.jp/`) を用いて開発されている．

登録されているデータには，共焦点レーザー走査型顕微鏡等によって得られた三次元画像や，細胞内記録，各種イメージングの結果などがあり，2017 年現在で，約 1600 件のデータが登録されている．

8.2.2 無脊椎動物脳プラットフォーム / 比較神経科学プラットフォーム

BoND は，研究室内でのみ利用可能となっているが，そのデータの一部は，無脊椎動物脳プラットフォーム (IVBPF: `https://invbrain.neuroinf.jp`) として，公開されている．

また，IVBPF は，現在，脊椎動物も含めた比較神経科学的観点からデータの収集をすすめている，比較神経科学データベース (CNSPF, `https://cns.neuroinf.jp`) に移行を行っている．

8.2.3 FlyCircuit

台湾の NCHC (National Center for High-performance Computing) 及び NTHU (National Tsing Hua University) によって運営されている，ショウジョウバエの単一神経細胞レベルのデータベースである．2017 年 10 月現在は，Version 1.1 となっており，メスショウジョウバエについては 18,161 ニューロン，オスショウジョウバエについては，5,422 ニューロンが登録されている．本研究では，Version 1.0 時点で登録されていた，メスショウジョウバエ 12,528 ニューロン，オスショウジョウバエ 3699 ニューロンを用いてモデル構築を行っている．

なお，FlyCircuit からの細胞形態ファイル (SWC) 取得については，並木重宏 博士にご協力を頂いた．

Fig.8.2　Bombyx Neuron Database

Fig.8.3　無脊椎動物脳プラットフォーム（`https://invbrain.neuroinf.jp`）

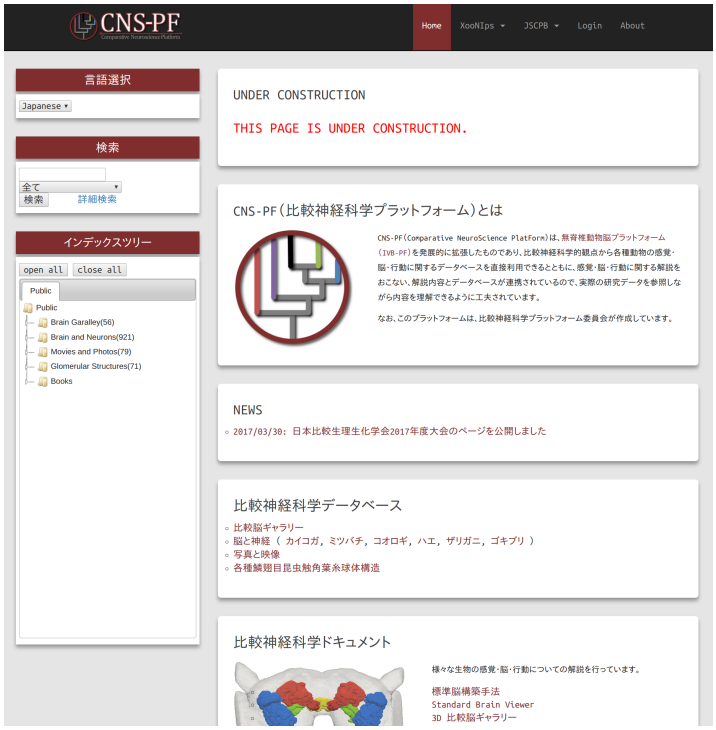


Fig.8.4　比較神経科学プラットフォーム (`https://cns.neuroinf.jp`)

Fig.8.5　FlyCircuit (`https://http://www.flycircuit.tw/`)

8.3　神経細胞形態抽出

神経細胞形態データベースには各神経細胞を共焦点レーザー走査型顕微鏡等により撮像した三次元画像が登録されているが，実際にシミュレーションを行うためにはこの三次元画像を元にマルチコンパートメント Hodgkin-Huxley 型モデルを構築する必要がある．そのためには，以下の様な作業が必要となる．

- 神経細胞の領域かそれ以外の領域かを識別するセグメンテーション
- 抽出された領域を元に，擬一次元的な近似を行いグラフとして再構築するトレーシング

本研究では，これらの作業を行うために，KNEWRiTE と SIGEN という二種類のソフトウェアを使用した．

8.3.1　KNEWRiTE

KNEWRiTE は，東京大学神崎・高橋研究室 (当時) の佐藤陽平により開発された細胞形態抽出のためのソフトウェアである [135] (Fig. 8.6) .

本ソフトウェアは，Rayburst 法 [113] と呼ばれる自動化手法と GUI によるインタラクティブな抽出結果の確認や修正，部分的に抽出したもの同士の接続を組み合わせたセミオートによる抽出を可能としている．これにより，全てマニュアルで抽出した場合に比べ，約 1/5 の時間で終える事ができるようになっただけでなく，抽出結果を元にシミュレーションを行った際の，抽出者によるばらつきも低減することに成功している [60].

8.3.2　SIGEN

兵庫県立大学 池野英利教授らにより開発されている，SIGEN[129] を，細胞の形態抽出のために使用した．

8.4　シミュレーション高速化・並列化

8.4.1　MPI

MPI (Message Passing Interface) は，並列で計算を行うための通信 (メッセージング) 手法に関する標準化規格である．神経回路シミュレーションにおいては，各細胞の並列性を元に別々のノードに配置するので，シナプス情報の伝達については，MPI のようなメッ

セージングを用いて実装することが一般的である [45].

なお，本研究においては，原則的に MPI2.0 の仕様に基づいて開発を行った.

8.4.2 OpenMP

OpenMP は，主にメモリを共有した計算環境において，スレッド単位の並列計算を容易に行うためのディレクティブベース並列計算環境である．プラットフォーム固有のスレッド API を用いずに暗黙にスレッドが生成されるため移植性は高くなるが，スレッド生成のオーバヘッドがかかる位置については，注意する必要がある．また，特に NUMA 型のプロセッサでは，事前にアロケーションしたメモリ領域へのアクセスが速い CPU コアで計算が行われるとは限らないため，メモリの割当については，'first touch' などの手法を用いて適切な場所になるよう配慮する必要がある.

本研究においては，ノード間は MPI により通信を行い，ノード内においては OpenMP によるスレッド並列化を行う，ハイブリッド化と呼ばれる手法を用いて開発を行った.

8.4.3 OpenCL/PezyCL

OpenCL (Open Computing Language) は，GPU や FPGA などのヘテロジニアス環境での高度計算を行うための標準化規格である．CUDA などベンダー固有の環境に比べると性能を出しにくいという欠点はあるが，複数のプラットフォームで共通のコードを利用することが可能なため，移植性は高くなる.

本研究においては，NVIDIA の `nvcc` を用いて GPU 向けの開発を行ったほか，PEZY-COMPUTING 社の OpenCL 互換環境である PezyCL を用いて，PEZY-SC 向けの開発を行った.

8.5 汎用ベンチマーク

計算機の実効性能を測定するため，汎用的なベンチマークを活用した.

8.5.1 STREAM

8.5.2 NAS Parallel Benchmark

NAS Parallel Benchmark[] は，NASA が開発している，並列計算ベンチマークスイートである．主に CFD などの流体シミュレーションを想定し，様々な計算カーネル（シミュレーションのコアとなる部分）や疑似アプリケーションでテストすることが可能と

なっている．

8.6 可視化

8.6.1 Paraview

Paraview[8] (http://www.paraview.org/) は，ロスアラモス国立研究所，Kitware 株式会社らによって開発が行われているオープンソース・マルチプラットフォームのの並列分散データ分析・可視化アプリケーションである．

本研究では，swc2vtk により，SWC を VTK 形式に変換し，最終的には Paraview を用いて可視化を行った．全脳規模の可視化では，2 億ポリゴン以上と非常に巨大となるが，Paraview は並列環境での可視化機能を有しており，高速に処理を行うことができた．

ただし，シミュレーションの可視化のような連番 VTK を用いる場合では，複数モデルの同時読み込みを行う方法がなかったため，プラグインスクリプトを作成し，レンダリングを行った．

以下に使用したデータ読み込みスクリプトの例を示す．

Listing 8.3 Paraview: loaditems.py

```
from paraview.simple import *

paraview.simple._DisableFirstRenderCameraReset()

filelist_small = [
    '0004_regist',
    '0005_regist',
    '0008_regist'
]

readers = []
displays = []
renderview = GetActiveViewOrCreate('RenderView')

for filename in filelist:
    reader = LegacyVTKReader(FileNames=[filepos + filename + '.vtk'])
    display = Show(reader, renderview)
    # ColorBy(display, None)
    RenameSource(filename, reader)

    renderview.ResetCamera()
    Render()
    ColorBy(display, ('CELLS', 'type'))
    display.RescaleTransferFunctionToDataRange(True, False)

    readers.append(reader)
```

```
    displays.append(display)
```

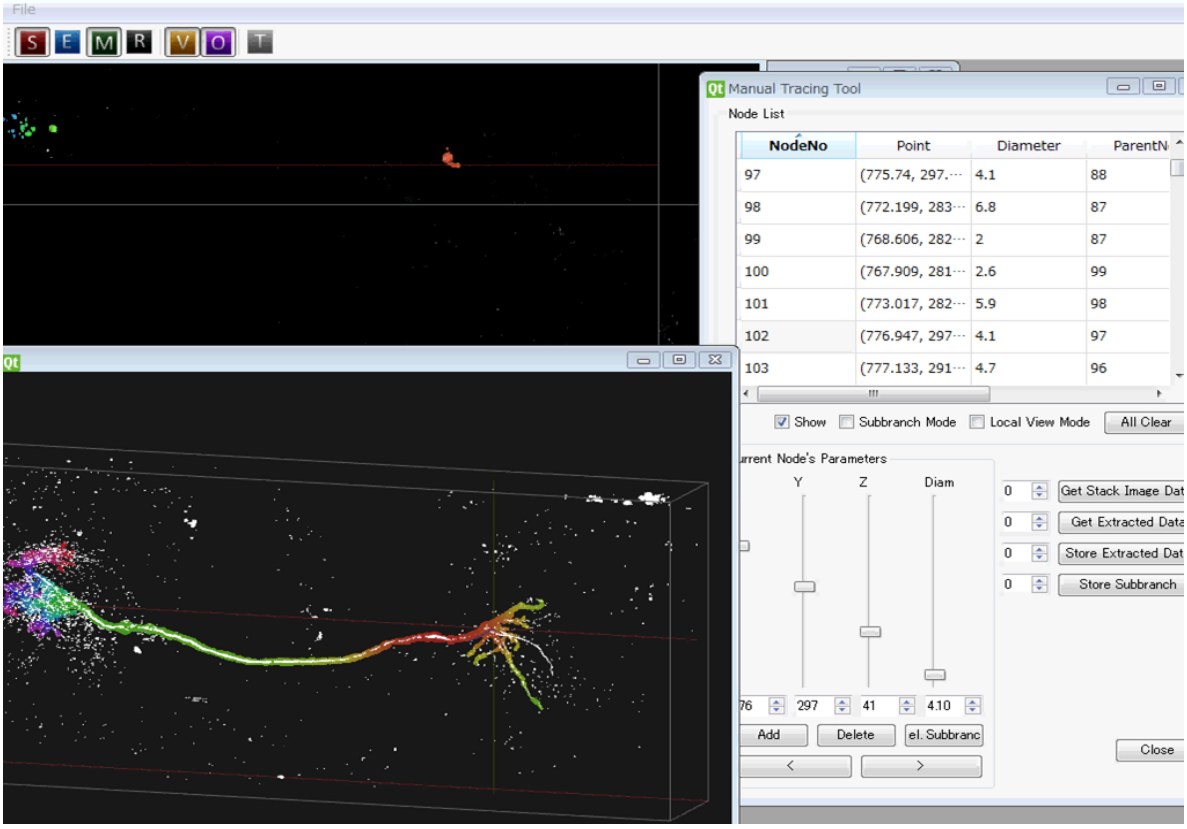


Fig.8.6　KNEWRiTE による細胞形態抽出

第 9 章

データベース情報を元にした神経回路モデルの構築

本研究では，BoND (8.2.1) や FlyCircuit (8.2.3) といった単一神経細胞スケールの実験結果データベースに登録されているデータに基づいて，神経回路シミュレーションモデルの構築を行った．本章では，そのモデルの構築手法について述べる．

9.1　本研究で構築した神経回路モデル

神経回路シミュレーションでは，細胞形態の複雑さやネットワーク構造などについて，対象ごとに非常に多様である．そこで，様々なモデルについて，シミュレーションの高速化・高並列化結果を検証するため，計算量や並列化の難易度の観点から，複数の神経回路モデルを構築した．なお，全てのモデルにおいて，神経細胞内のモデルとしては，マルチコンパートメント Hodgkin-Huxley 型モデル (5.1) を用い，シナプスとしては可塑性を考慮しない固定閾値の指数減衰型シナプス (5.2) を用いた (5 章参照) .

Table9.1 に，今回構築したモデルの一覧を示す．

Table9.1 本研究で作成した神経回路シミュレーションモデル

モデル名	細胞種	総細胞数	総コンパートメント数	平均コンパートメント数
カイコガ均一神経細胞モデル	1	任意	3888 × 細胞数	3,888
カイコガ LAL-VPC 領域 86 細胞神経回路モデル		86		
オスショウジョウバエ大規模モデル	3,649	3,649	1,871,937	513
メスショウジョウバエ大規模モデル	12,393	12,393	4,399,515	355

9.2　カイコガ均一神経細胞モデル

9.2.1　概要

カイコガ均一神経細胞モデルは，細胞数について，自由にスケーリングしてベンチマークを行うために作成したモデルである．

ここでは, カイコガの bilateral 神経細胞である BN1056 について, KNEWRiTE (8.3.1) により神経細胞形態抽出を行ったものを任意の個数並べ，それらを一定のルールに基づいてシナプス接続することでモデル構築を行った．また，実際の神経細胞では，マルチコンパートメント Hodgkin-Huxley 方程式における各種定数は様々な値をとるが，今回のベンチマークでは，これらの値が直接的に計算性能に影響を与える可能性は低いと考え，原則として，NEURON のデフォルト値（[54] がベース) を用いることとした．実際の値については，Table 9.2 に示す．

なお，モデル生成には，nrc-gen (7.3) を利用した．

Table9.2　Parameters for the benchmark simulation

	Variable Name	Value	Stored in
Δt	dt	0.025 [msec]	global
T	tstop	500.0 [msec]	global
	celsius	6.3 [degC]	global
C_m	cm	1.0 [μF/cm^2]	each compartment
$\overline{g}_{\mathrm{Na}}$	gnabar	120.0 [mS/cm^2]	each compartment
$\overline{g}_{\mathrm{K}}$	gkbar	36.0 [mS/cm^2]	each compartment
g_L	gl	0.3 [mS/cm^2]	each compartment
E_{Na}	ena	50.0 [mV]	each compartment
E_{K}	ek	-77.0 [mV]	each compartment
E_{L}	el	-54.3 [mV]	each compartment

9.2.2　神経回路ネットワーク作成手法

シナプス接続のルールは，全体のネットワークパターンを規定し，MPI 通信に大きく影響するため，非常に重要となる．そこで，リングネットワーク，ランダムネットワーク，Watts and Strogatz ネットワーク [127] の 3 種類を用意することとした．以下に，各ネッ

トワークの概要と擬似コードを示す．

尚，各種定数と関数については，

- NCELL：細胞数
- NSYNAPSE：細胞 1 個あたりのシナプス数
- RND：1 以上かつ細胞数より少ない整数乱数
- makeSynapse(X, Y)：シナプス前末端を細胞 X，シナプス後末端を細胞 Y に作成し，接続する．

とし，いずれの場合も，細胞内のシナプス位置はランダムとした．

リングネットワーク

隣り合った細胞にシナプスを作成する．

Listing 9.1　リングネットワークの作成

```
for(int i=0; i<NCELL; i++){
    for(int j=0; j<NSYNAPSE; j++){
        makeSynapse(i, i+1 mod NCELL);
    }
}
```

ランダムネットワーク

シナプス前末端の位置はリングネットワークと同様だが，シナプス後末端の位置がランダムなネットワーク．

Listing 9.2　ランダムネットワークの作成

```
for(int i=0; i<NCELL; i++){
    for(int j=0; j<NSYNAPSE; j++){
        makeSynapse(i, (i+RND) mod NCELL);
    }
}
```

Watts and Strogatz ネットワーク

実際の神経回路ネットワークは，リングでも完全なランダムでもないと考えられる．そこで，その 2 つの中間的なネットワークとして，Watts and Strogatz ネットワーク [127] についても考慮する．リングネットワークを作成した後に，確率 $p < 1$ で後末端をランダムに繋ぎ変えるものである．ここでは，$P = \mathrm{NSYNAPSE} * p$ とした．

Listing 9.3　Watts and Strogatz ネットワークの作成

```
for(int i=0; i<NCELL; i++){
    for(int j=0; j<P; j++){
        makeSynapse(i, (i+RND) mod NCELL);
    }
    for(int j=P; j<NSYNAPSE; j++){
        makeSynapse(i, i+1 mod NCELL);
    }
}
```

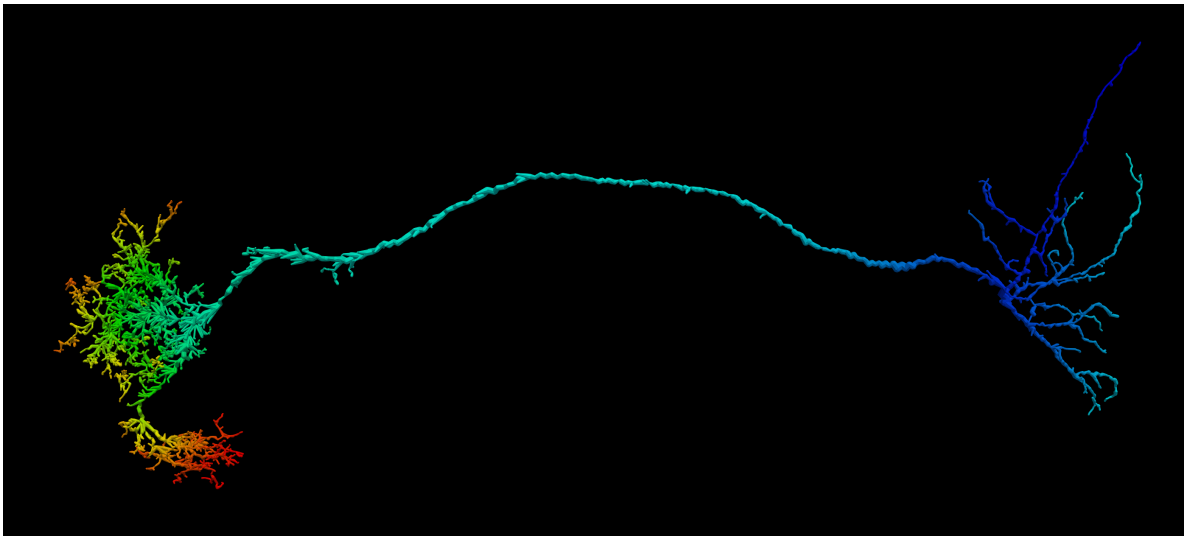


Fig.9.1　カイコガ均一神経細胞モデルにおける基準細胞 (BN1056)

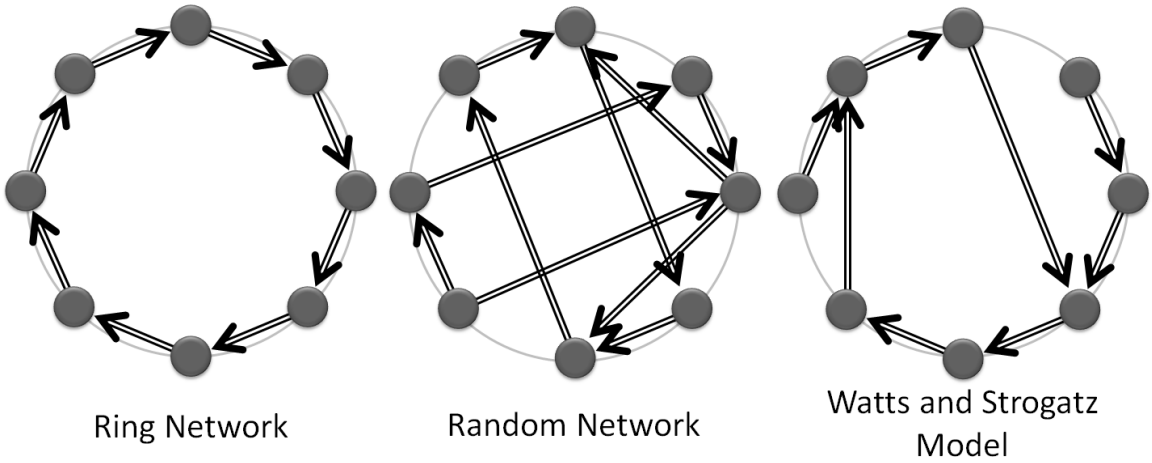


Fig.9.2　カイコガ均一神経細胞モデルにおける 3 種類のネットワークパターン

9.3　カイコガ LAL-VPC 領域 86 細胞神経回路モデル

9.3.1　概要

均一モデルに比べ，より生物学的に詳細なモデルとして，LAL-VPC 領域 86 細胞神経回路モデルの構築を行った．本モデルでは，カイコガ脳 LAL-VPC 領域について，岩野正晃らの分類した 5 つの領域，lower LAL (lLAL)，upper LAL (uLAL)，outer VPC (oVPC)，inner VPC (iVPC)，anterior inner VPC (aVPC) 及びそれぞれの領域への神経細胞の投射情報 (Fig.9.3, 9.4) [63] に基づき，池野，並木，加沢らが BoND データベース (8.2.1) より，対応する神経細胞を選び出した情報 (9.5) を元に，再構築を行っている．

なお，本モデルの構築は，東京大学神崎・高橋研究室 (当時) の福田哲也と共同で行った．

9.3.2　LAL-VPC5 領域モデルによる神経細胞の入出力領域の同定

特定の神経細胞において，シナプスの作成位置を決定するためには，まず入出力領域の同定が必要となる．そこで，BoND データベースより取り出した神経細胞について，KNEWRiTE (8.3.1) 及び SIGEN (8.3.2) による細胞形態抽出を行い，NeuroRegister (7.6) による標準脳マッピングを行った．この時，マッピング情報については，池野らの作成したものを利用した．

このマッピング済み神経細胞と，並木により作成された標準脳上の LAL-VPC5 領域の 3 次元塗り分け結果 (Fig.9.6) を比較するソフトウェアである LAL-VPC Mapping (7.7) を作成し，神経細胞の各コンパートメントがどの領域に属しているか，判別を行った．この結果を，Fig.9.7, 9.8 に示す．

9.3.3　入出力領域に基づいた Peter's rule によるシナプス作成

これまでの情報により，各神経細胞の入出力領域について判定ができたため，これを元に，同一の領域に入力部と出力部を持つ全ての神経細胞対について，Peter's rule に基づいてシナプスの作成を行った．これには，SB Simulation (7.8) ソフトウェアを用いた．

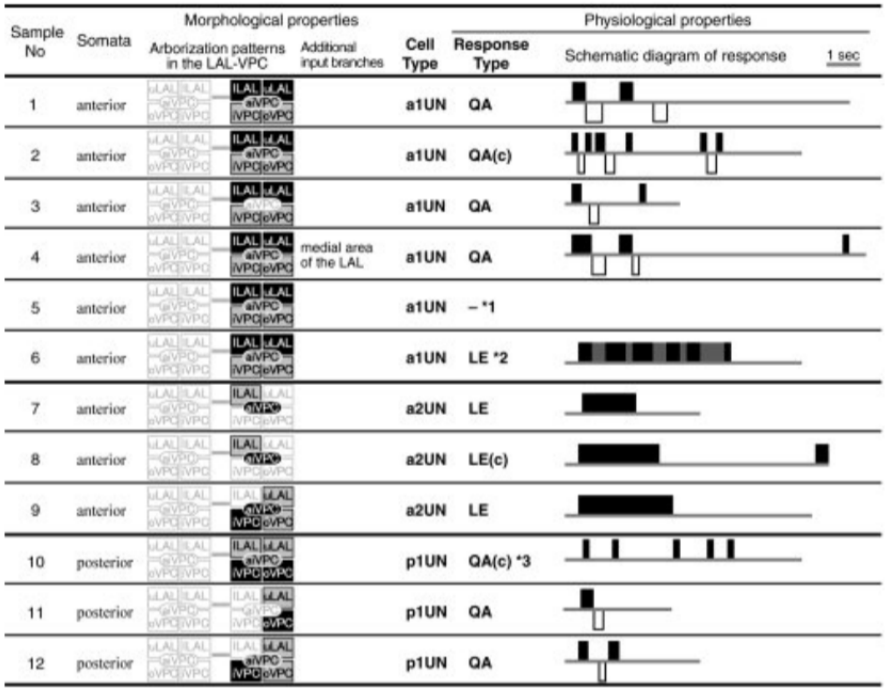
この時作成したシナプス位置の可視化結果について，Fig. 9.9, 9.10 に示す．また，各神経細胞対の合計シナプス強度について，Fig. 9.11 に示す．

9.3.4　LAL-VPC 領域 86 細胞モデルの可視化

以上により作成を行った，LAL-VPC 領域 86 細胞モデルについて，swc2vtk (7.10) を用いて，可視化を行った結果を，Fig. 9.13, 9.14 に示す．

Sample No	Somata	Morphological properties: Additional output branches	Arborization patterns in the LAL-VPC	Additional input branches	Cell Type	Physiological properties: Response Type	Schematic diagram of response (1 sec)
1	anterior	β-lobe of the MB	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	LE(c)	
2	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	LE(c)	
3	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	BI-LE(c)	
4	anterior	ventromedial to the LAL	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	LE(c)	
5	anterior	ventromedial to the LAL	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	BI-LE(c)	
6	anterior	anterior part of the SMPC	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	β-lobe of the MB	a1BN	BE	
7	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	anterior part of the SMPC and middle part of the IMPC	a1BN	BE	
8	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	BE	
9	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	BE	
10	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	α-lobe of the MB	a1BN	–	
11	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a1BN	–	
12	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a2BN	LE(c)	
13	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	anterior part of the SMPC	a2BN	BE	
14	anterior	a part of the CBL and anterior part of the SMPC	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	a part of the median olive	a2BN	BI	
15	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	β-lobe of the MB	a2BN	BI	
16	anterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		a2BN	NR	
17	posterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		p1BN	–	
18	posterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		p1BN	–	
19	posterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		p1BN	BE	
20	posterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	anterior part of the SMPC	p1BN	BE	
21	posterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		p1BN	BE	
22	posterior		uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC	middle part of the ILPC	p1BN	BE	
23	posterior	medial to the LAL	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		p1BN	LI(c)	
24	posterior	PSL and anterodorsol to the LAL	uLAL, ILAL, @VPC, vVPC, VPC / ILAL, uLAL, @VPC, lVPC, vVPC		p1BN	LI	

Fig.9.3　岩野らによる LAL-VPC 領域 Bilateral ニューロンの分類 [63]



Sample No	Somata	Morphological properties: Arborization patterns in the LAL-VPC	Additional input branches	Cell Type	Physiological properties: Response Type	Schematic diagram of response (1 sec)
1	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a1UN	QA	
2	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a1UN	QA(c)	
3	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a1UN	QA	
4	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC	medial area of the LAL	a1UN	QA	
5	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a1UN	– *1	
6	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a1UN	LE *2	
7	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a2UN	LE	
8	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a2UN	LE(c)	
9	anterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		a2UN	LE	
10	posterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		p1UN	QA(c) *3	
11	posterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		p1UN	QA	
12	posterior	uLAL, lLAL, aVPC, iVPC, lLAL, uLAL, aiVPC, iVPC, eVPC		p1UN	QA	

Fig.9.4　岩野らによる LAL-VPC 領域 Unilateral ニューロンの分類 [63]

BN

No	モデルNo	岩野氏論文での応答	使用したID	Neuronラベル
1	1	LLA	1020	091210_3_sn
2	2	LLA	1020(LLA代表データ)	091210_3_sn
3	3	BI-LLA	663(BI-LLA代表データ)	050324_4_sn
4	4	LLA	1020(LLA代表データ)	091210_3_sn
5	5	BI-LLA	663	050324_4_sn
6	6	BA	993(BA代表データ)	091216_3_sn
7	7	BA	661	050323_1_sn
8	8	BA	993(BA代表データ)	091216_3_sn
9	9	BA	993(BA代表データ)	091216_3_sn
10	10	不明	965(NR代表データ)	090617_1_sn
11	11	不明	965(NR代表データ)	090617_1_sn
12	12	LLA	1020(LLA代表データ)	091210_3_sn
13	13	BA	664	050612_8_sn
14	14	BI	655(BI代表データ)	050609_3_sn
15	15	BI	655(BI代表データ)	050609_3_sn
16	16	Non Response	965(NR代表データ)	090617_1_sn
17	17	不明	965(NR代表データ)	090617_1_sn
18	18	不明	965(NR代表データ)	090617_1_sn
19	19	BA	984	091208_8_sn
20	20	BA	984	091208_8_sn
21	21	BA	993(BA代表データ)	091216_3_sn
22	22	BA	993(BA代表データ)	091216_3_sn
23	23	LLI	661(LLI代表データ)	050323_1_sn
24	24	LLI	655	050609_3_sn

UN

No	モデルNo	岩野氏論文での応答	使用したID	Neuronラベル
1	25	QR	970	091122_1_sn
2	26	QR	970	091122_1_sn
3	27	QR	969	091119_1_sn
4	28	QR	1009(QR代表データ)	091229_6_sn
5	29	不明	970	091122_1_sn
6	30	LLA	970	091122_1_sn
7	31	LLA	1080	090603_7_sn
8	32	LLA	1080	090603_7_sn
9	33	LLA	986	091208_10_sn
10	34	QR	1009(QR代表データ)	091229_6_sn
11	35	QR	988	091209_5_sn
12	36	QR	1009(QR代表データ)	091229_6_sn

DN

タイプ	モデルNo	和田氏論文での応答	使用したID	Neuronラベル
GI A	37	FF	973	090924_4_sn
GI B	38	LLI	967	091124_1_sn
GI C	39	LLI	661(BNのLLI代表データ)	050323_1_sn
GII A	40	FF	1068	091209_9_sn
GII B	41	BI	663(BNのBI代表データ)	050324_4_sn
GII C	42	BA	993(BNのBA代表データ)	091216_3_sn
GII D	43	FF	966	090925_4_sn

Fig.9.5　池野，並木，加沢らによる BoND データベース 8.2.1 上のニューロンと，岩野らによる 5 領域投射モデル [63] の対応

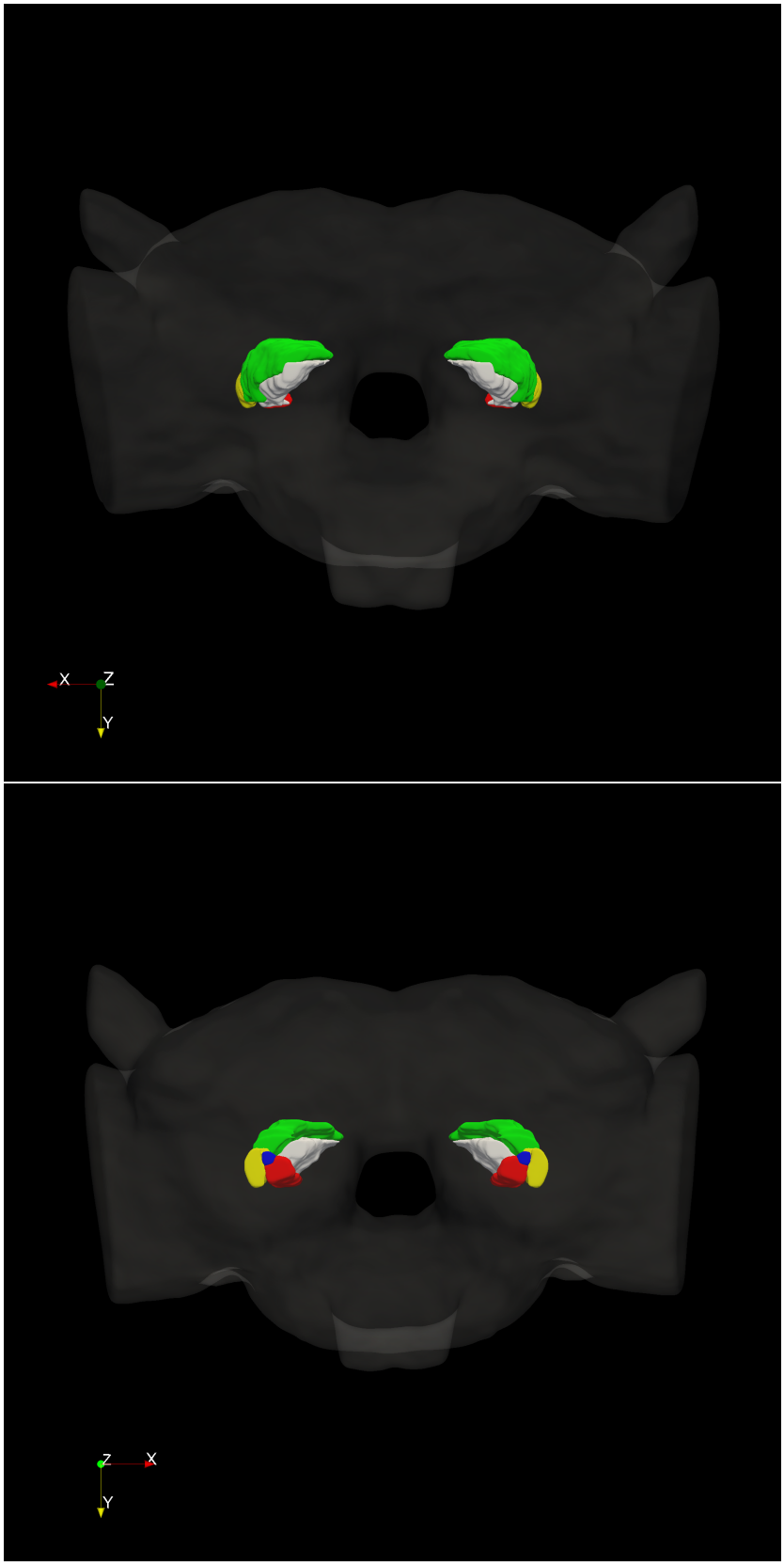


Fig.9.6　並木による LAL-VPC5 領域の塗り分け

			投射コンパートメント数				
Cell ID	BoND ID	Flip position	Region1	Region2	Region3	Region4	Region5
1	1020		4	0	8	42	4
2	1020		4	0	8	42	4
3	663		0	0	18	854	396
4	1020		4	0	8	42	4
5	663		0	0	18	854	396
6	993		0	0	0	22	0
7	661		0	287	7	285	2563
8	993		0	0	0	22	0
9	993		0	0	0	22	0
10	965		0	0	0	0	0
11	965		0	0	0	0	0
12	1020		4	0	8	42	4
13	664		0	0	0	1027	317
14	655		0	16	0	95	0
15	655		0	16	0	95	0
16	965		0	0	0	0	0
17	965		0	0	0	0	0
18	965		0	0	0	0	0
19	984		0	64	1	117	47
20	984		0	64	1	117	47
21	993		0	0	0	22	0
22	993		0	0	0	22	0
23	661		0	287	7	285	2563
24	655		0	16	0	95	0
25	970		0	8	0	1404	1214
26	970		0	8	0	1404	1214
27	969		123	742	419	2371	2159
28	1009		0	0	0	27	0
29	970		0	8	0	1404	1214
30	970		0	8	0	1404	1214
31	1080		0	0	0	0	0
32	1080		0	0	0	0	0
33	986		0	235	3	1574	463
34	1009		0	0	0	27	0
35	988		87	76	10	620	850
36	1009		0	0	0	27	0
37	973		0	65	9	78	208
38	967		103	163	145	152	337
39	661		0	287	7	285	2563
40	1068		0	0	0	138	143
41	663		0	0	18	854	396
42	993		0	0	0	22	0
43	966		0	0	0	0	0
44	1020		4	0	8	42	4

Fig.9.7　LAL-VPC5 領域モデルによる神経細胞の領域分け結果 1

45	1020	[redacted]	4	0	8	42	4
46	663	[redacted]	0	0	18	854	396
47	1020	[redacted]	4	0	8	42	4
48	663	[redacted]	0	0	18	854	396
49	993	[redacted]	0	0	0	22	0
50	661	[redacted]	0	287	7	285	2563
51	993	[redacted]	0	0	0	22	0
52	993	[redacted]	0	0	0	22	0
53	965	[redacted]	0	0	0	0	0
54	965	[redacted]	0	0	0	0	0
55	1020	[redacted]	4	0	8	42	4
56	664	[redacted]	0	0	0	1027	317
57	655	[redacted]	0	16	0	95	0
58	655	[redacted]	0	16	0	95	0
59	965	[redacted]	0	0	0	0	0
60	965	[redacted]	0	0	0	0	0
61	965	[redacted]	0	0	0	0	0
62	984	[redacted]	0	64	1	117	47
63	984	[redacted]	0	64	1	117	47
64	993	[redacted]	0	0	0	22	0
65	993	[redacted]	0	0	0	22	0
66	661	[redacted]	0	287	7	285	2563
67	655	[redacted]	0	16	0	95	0
68	970	[redacted]	0	8	0	1404	1214
69	970	[redacted]	0	8	0	1404	1214
70	969	[redacted]	123	742	419	2371	2159
71	1009	[redacted]	0	0	0	27	0
72	970	[redacted]	0	8	0	1404	1214
73	970	[redacted]	0	8	0	1404	1214
74	1080	[redacted]	0	0	0	0	0
75	1080	[redacted]	0	0	0	0	0
76	986	[redacted]	0	235	3	1574	463
77	1009	[redacted]	0	0	0	27	0
78	988	[redacted]	87	76	10	620	850
79	1009	[redacted]	0	0	0	27	0
80	973	[redacted]	0	65	9	78	208
81	967	[redacted]	103	163	145	152	337
82	661	[redacted]	0	287	7	285	2563
83	1068	[redacted]	0	0	0	138	143
84	663	[redacted]	0	0	18	854	396
85	993	[redacted]	0	0	0	22	0
86	966	[redacted]	0	0	0	0	0

Fig.9.8　LAL-VPC5 領域モデルによる神経細胞の領域分け結果 2

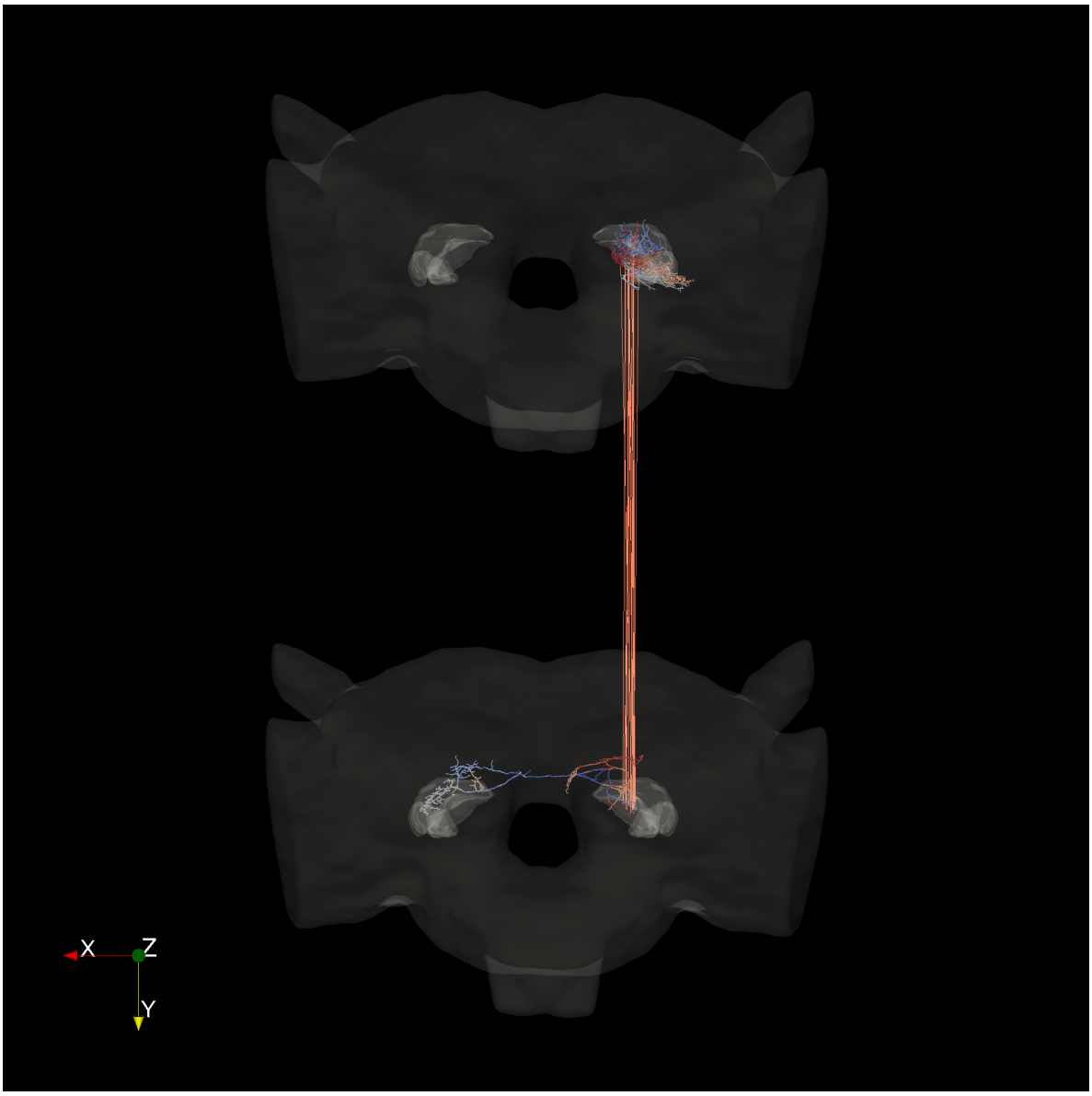


Fig.9.9　Peter's rule によるシナプス作成例 1

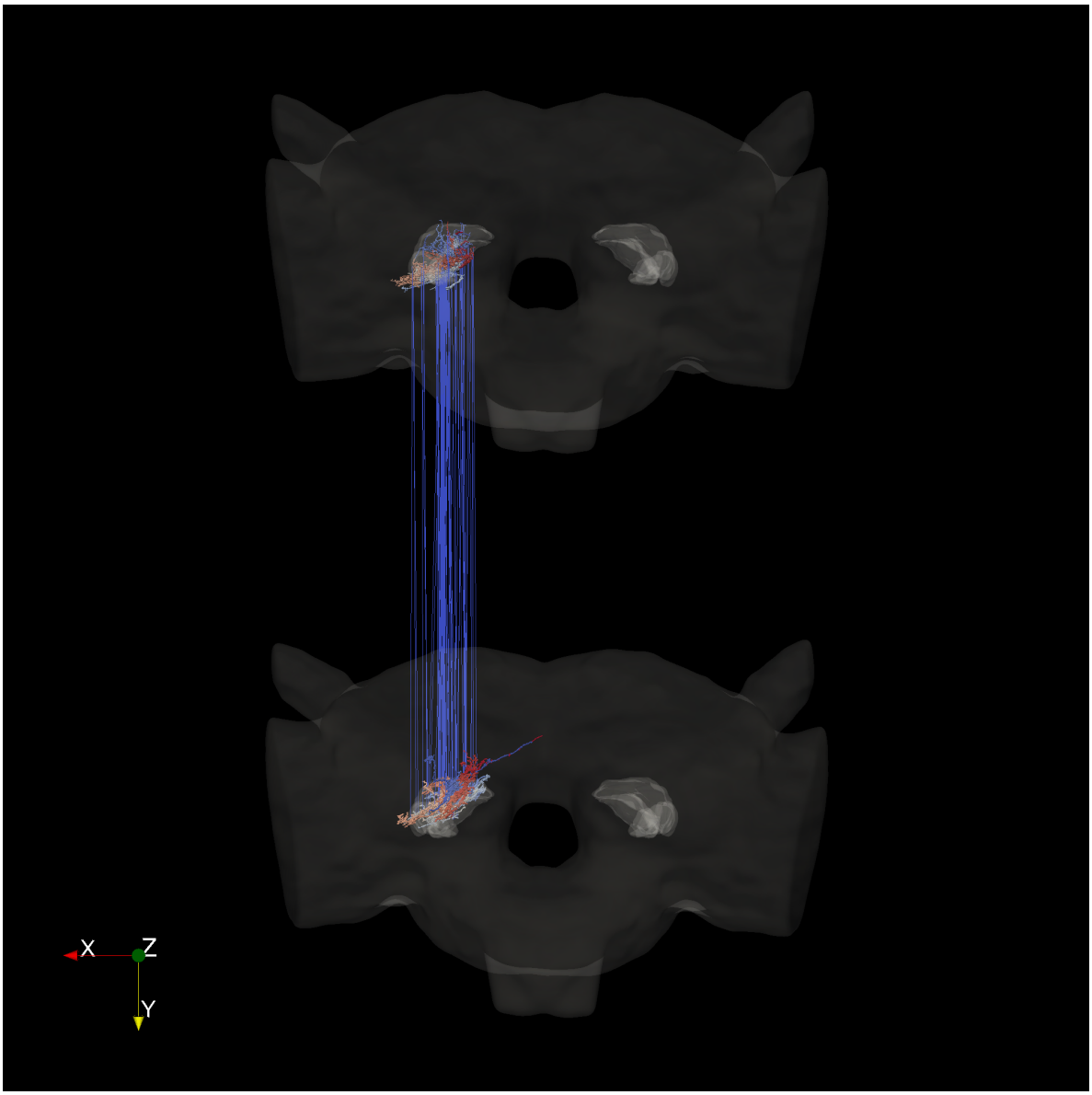


Fig.9.10　Peter's rule によるシナプス作成例 2

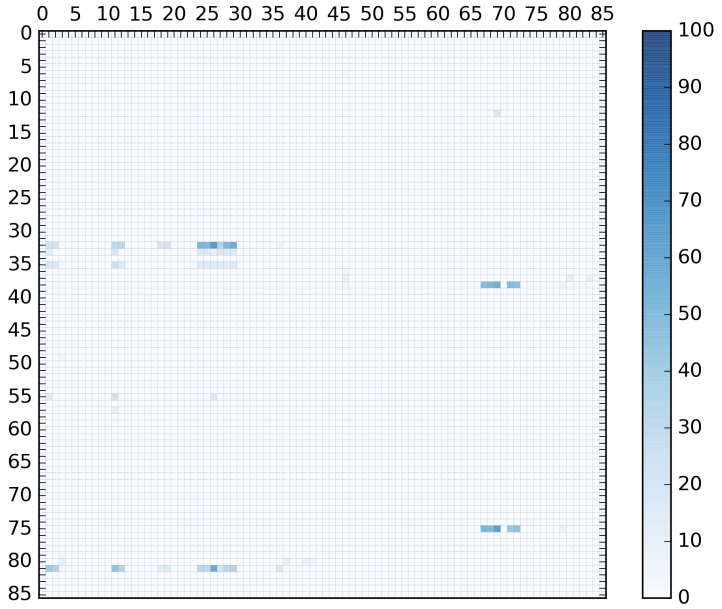


Fig.9.11　Peter's rule によるシナプス作成結果

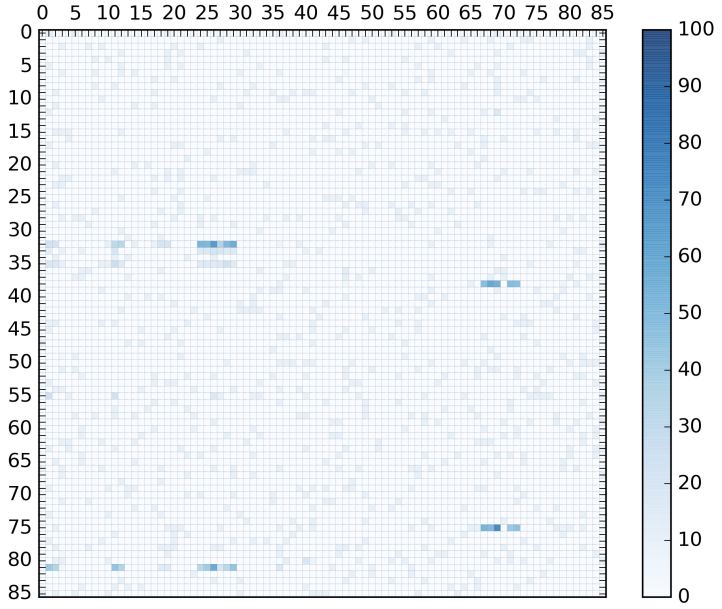


Fig.9.12　Peter's rule によるシナプス作成 + ランダムシナプス（$p = 0.1, N = 10$）作成結果

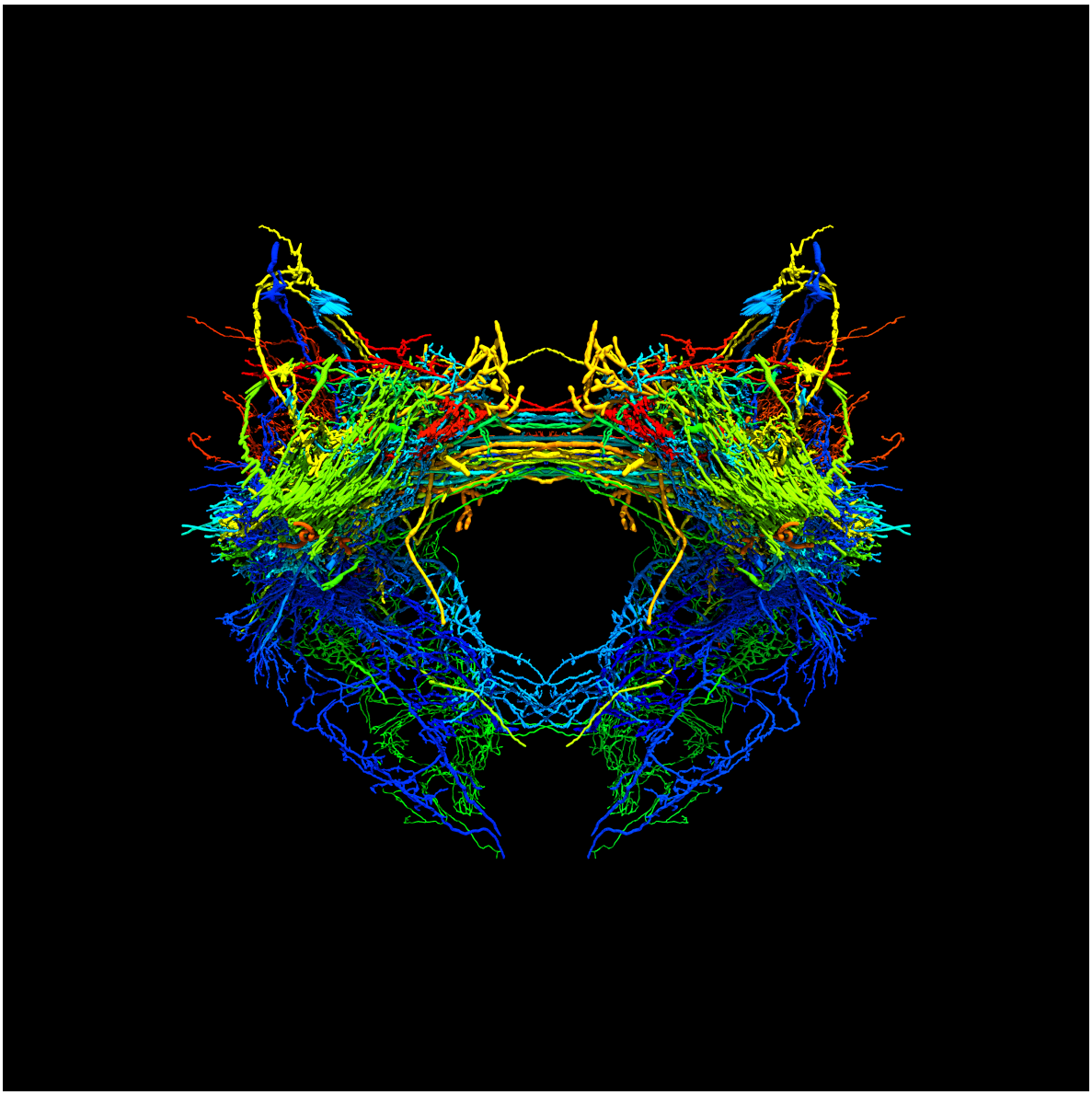


Fig.9.13　LAL-VPC 標準脳モデルの可視化（細胞ごとの色分け）

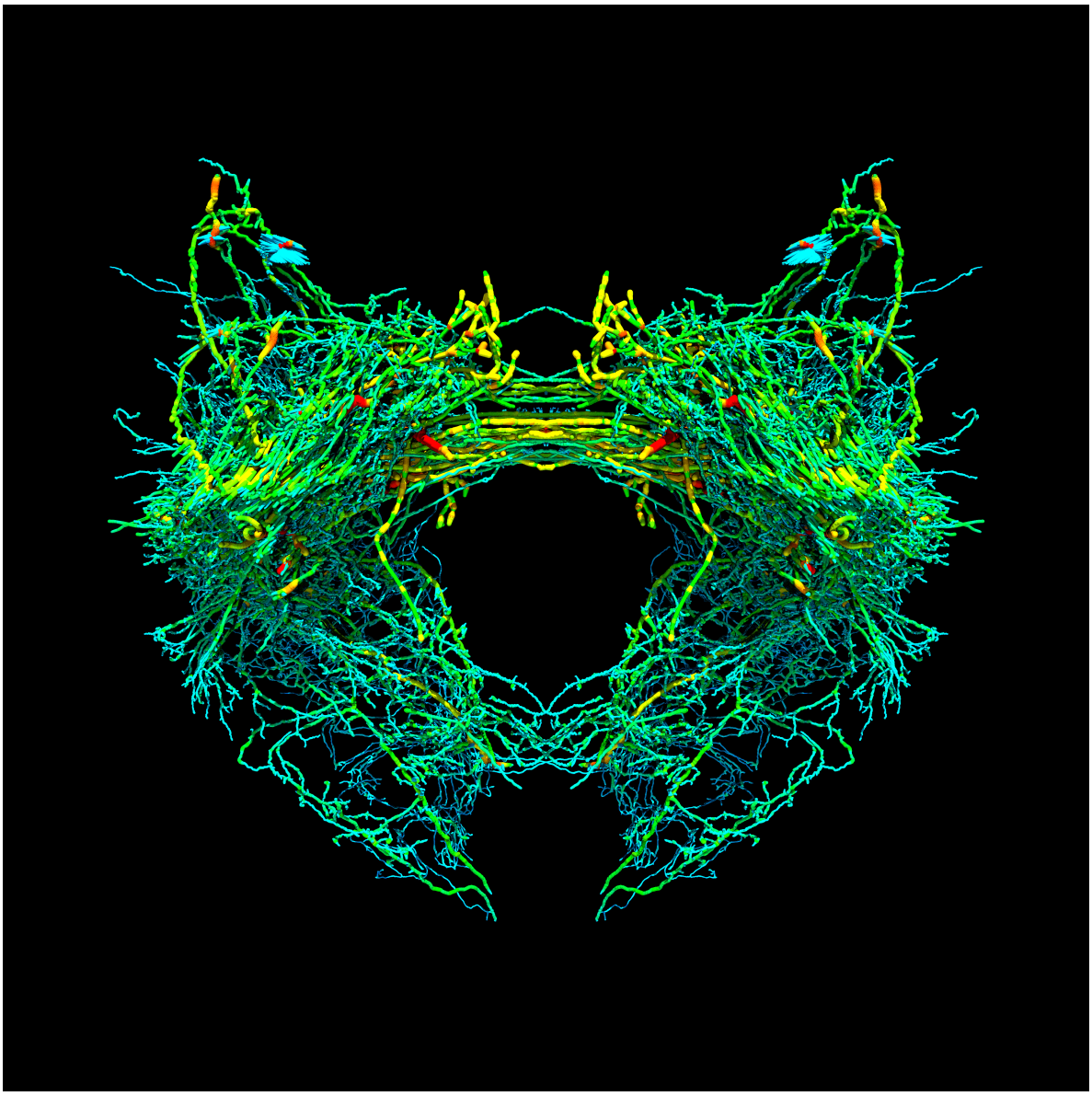


Fig.9.14　LAL-VPC 標準脳モデルの可視化（コンパートメント太さによる色分け）

9.4　ショウジョウバエ大規模モデルの構築

9.4.1　概要

大規模かつ多様な神経細胞形態を含むモデルとして，FlyCircuit データベースのデータを元にショウジョウバエ神経細胞の大規模神経回路モデルの構築を行った．シミュレーション速度・並列性能のベンチマークとしてのみ利用するため，初期刺激細胞，シナプスを生成する細胞対，シナプス位置等については，ランダムに作成を行う事とした．

9.4.2　nrc-gen による NRC ファイル生成

大規模並列環境においてスケーリングを行うためには，ノード数に合わせた細胞配置を事前に決めておく必要がある．ここでは，nrc-gen (7.3) を用いて，細胞数と同じ数のノード数を利用するものを基準に，$\frac{1}{2}$，$\frac{1}{4}$，$\frac{1}{8}$ のノード数を利用する NRC ファイルセットを生成した．

この時使用した，スクリプトについて，以下に掲載する．

Listing 9.4　nrc-gen for flycircuit

```
import nrcgen

# M

nrcg = nrcgen.NrcGenerator(3649)
nrcg.add_stim(500)
nrcg.add_ring_synapse(50)
nrcg.append_swc_file_list('./flycircuit_M_swc_list.txt')
nrcg.write('./flycircuit_M_3649', 3649)
nrcg.write('./flycircuit_M_1825', 1825)
nrcg.write('./flycircuit_M_913', 913)
nrcg.write('./flycircuit_M_457', 457)
nrcg.write('./flycircuit_M_229', 229)

nrcg = nrcgen.NrcGenerator(3649)
nrcg.add_stim(500)
nrcg.add_ring_synapse(50)
nrcg.default_swc_filename = 'fru-M-200331.swc'
nrcg.write('./flycircuit_M_3649_homo', 3649)
nrcg.write('./flycircuit_M_1825_homo', 1825)
nrcg.write('./flycircuit_M_913_homo', 913)
nrcg.write('./flycircuit_M_457_homo', 457)
nrcg.write('./flycircuit_M_229_homo', 229)

# F
```

```

nrcg = nrcgen.NrcGenerator(12393)
nrcg.add_stim(1000)
nrcg.add_ring_synapse(50)
nrcg.append_swc_file_list('./flycircuit_F_swc_list.txt')
nrcg.write('./flycircuit_F_12393', 12393)
nrcg.write('./flycircuit_F_6197', 6197)
nrcg.write('./flycircuit_F_3099', 3099)
nrcg.write('./flycircuit_F_1550', 1550)
nrcg.write('./flycircuit_F_775', 775)

nrcg = nrcgen.NrcGenerator(12393)
nrcg.add_stim(1000)
nrcg.add_ring_synapse(50)
nrcg.default_swc_filename = 'VGlut-F-300375.swc'
nrcg.write('./flycircuit_F_12393_homo', 12393)
nrcg.write('./flycircuit_F_6197_homo', 6197)
nrcg.write('./flycircuit_F_3099_homo', 3099)
nrcg.write('./flycircuit_F_1550_homo', 1550)
nrcg.write('./flycircuit_F_775_homo', 775)
```

Listing 9.5 visualization for flycircuit

```
# -*- coding: utf-8 -*-

import os
import sys
import utils
import swc2vtk


output_base_dir = '/home/nebula/work/vtk/flycircuit'
output_dir = os.path.join(output_base_dir, '20170830')
swcfilename_base = os.path.join('/home/nebula/IdeaProjects/flycircuit/
    swc/M', '%s')

cellname_list = utils.cellname_list_from_file('../M_swc_list.txt')

if not os.path.isdir(output_dir):
    os.mkdir(output_dir)

vtkgen = swc2vtk.VtkGenerator()
for filename in cellname_list:
    vtkgen.add_swc(swcfilename_base % (filename + '.swc'))

vtkgen.write_vtk(os.path.join(output_dir, 'M_all.vtk'), coloring=True,
    normalize_diam=False, radius_data=True)
```

9.4.3　ショウジョウバエ大規模モデルの可視化

以上により作成を行った，ショウジョウバエ大規模モデルについて，swc2vtk (7.10) を用いて，可視化を行った結果を，Fig. 9.15, 9.16 に示す．

また，各細胞種ごとに色分けした結果について，Fig. 9.17，9.18 に示す．

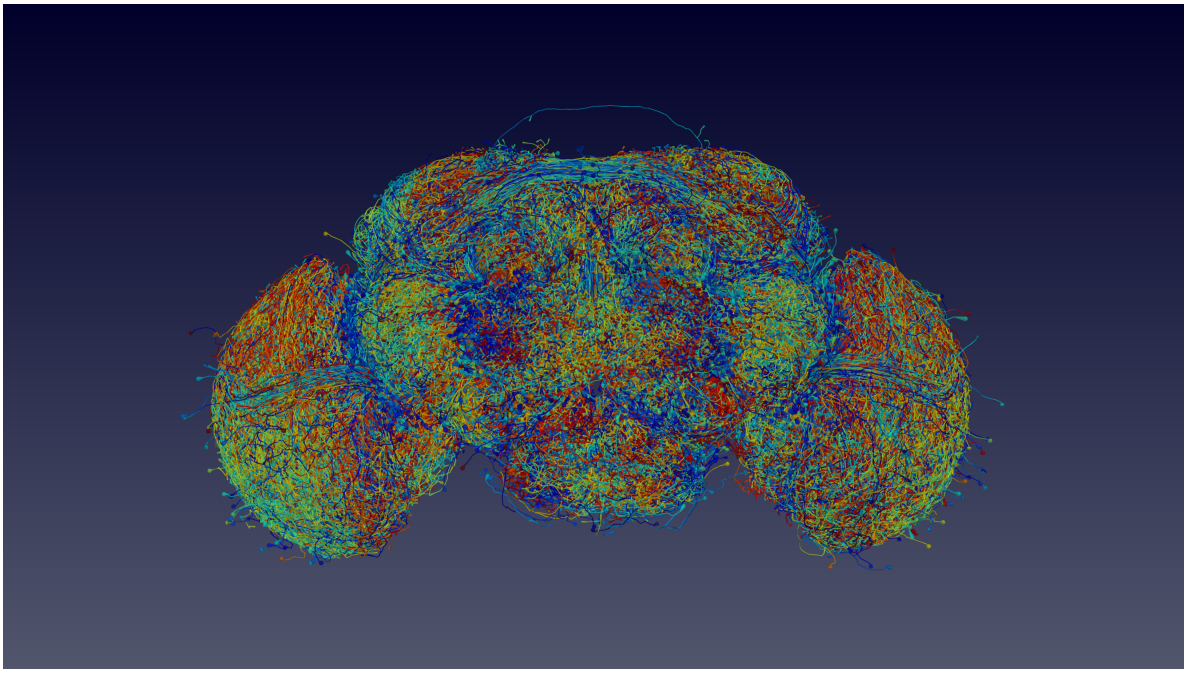


Fig.9.15　オスショウジョウバエ大規模神経回路モデル

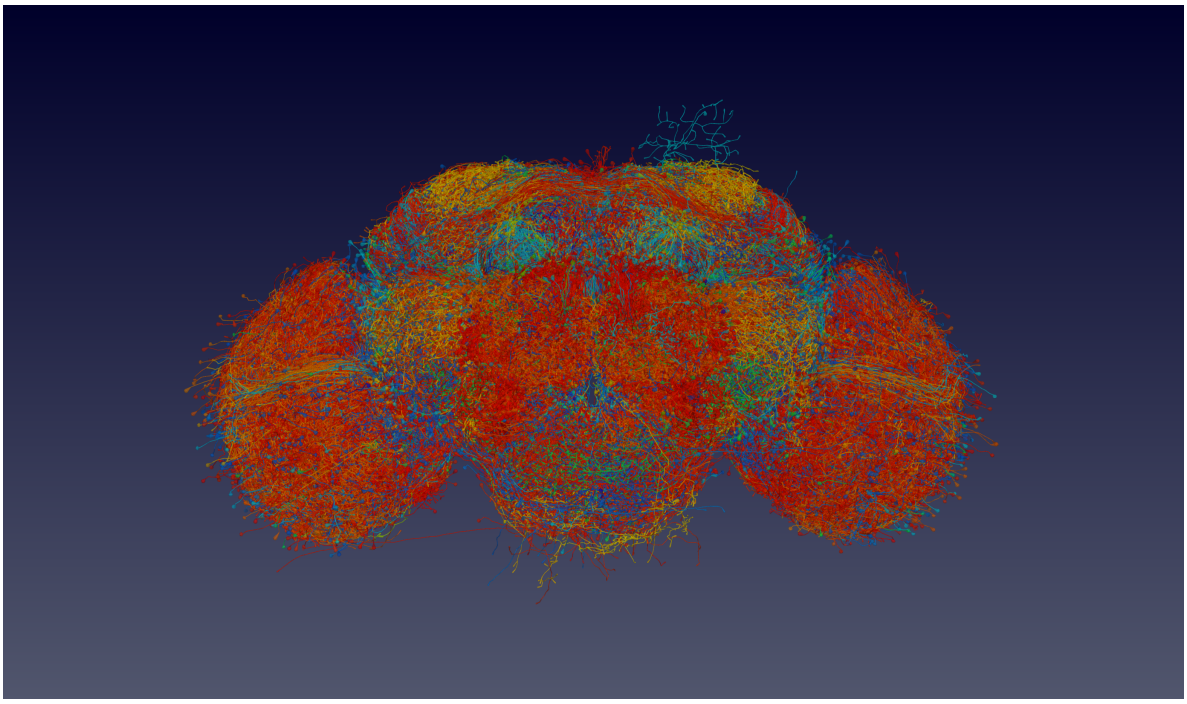


Fig.9.16　メスショウジョウバエ大規模神経回路モデル

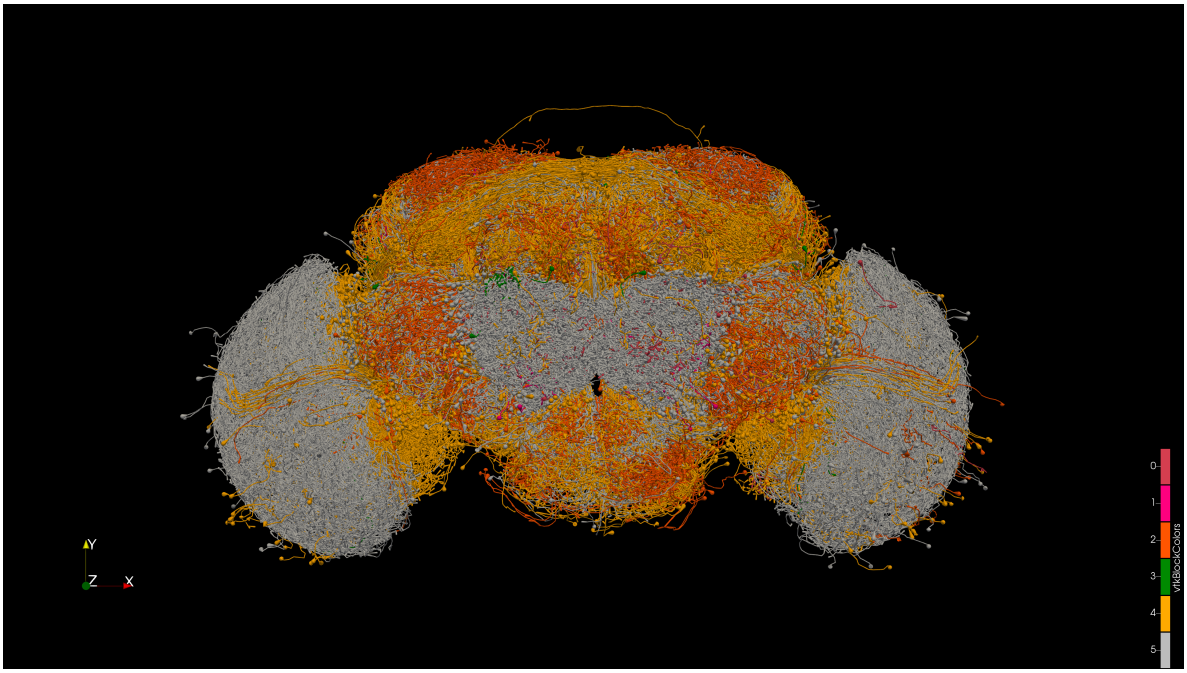


Fig.9.17　オスショウジョウバエ大規模神経回路モデル（細胞種類ごと色分け）

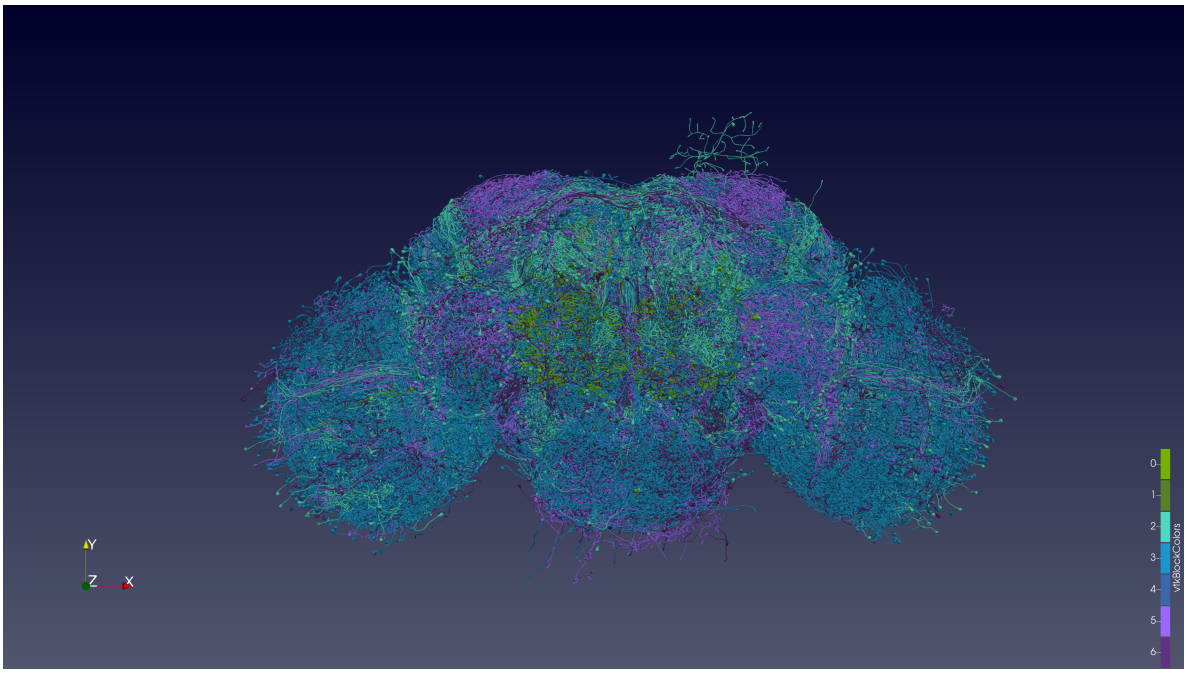


Fig.9.18　メスショウジョウバエ大規模神経回路モデル（細胞種類ごと色分け）

第 10 章

神経細胞・回路シミュレーションの高速化・高並列化手法

10.1 NEURON 標準の高速化手法の評価

NEURON には，計算を高速化するための複数の仕組みが標準で備わっている．一つは電位配列の括りだしによる高速化である．これは，各コンパートメントの V の変数について，別の配列として保持することで，メモリへのアクセスが連続になるようにし，プリフェッチを有効活用する事ができる．この機能を有効にするためには，`CVode` class の `cach_efficient()` というメソッドを計算開始前に呼ぶ必要がある．

もう一つは，Hodgkin-Huxley 方程式の計算における look-up table (LUT) の活用である．Hodgkin-Huxley 方程式の中でも m, h, n の更新に用いられる rate 方程式は非常に計算コストが大きいが，V の値ごとに値を事前に計算しておくことで，計算を短縮することが可能である．また，事前に計算した値のみでは，精度が充分でないため，実際に使用する際には線形補間を適用している．この機能は，NMODL 内で，テーブル化する変数について，`TABLE minf, mtau, hinf, htau, ninf, ntau DEPEND celsius FROM -100 TO 100 WITH 200` などと記述することによって有効にすることが可能である．

10.2 NEURON の単体性能解析

高速化を行うための準備段階として，NEURON K+ の主要関数における浮動小数点演算数と要求メモリアクセス量について，検討を行った．この時，演算数については，ソースコードを元にした計数と，京コンピュータのプロファイラをベースに実行時測定を行ったもののふた通り行った．メモリアクセス量については，ソースコードを元にした計数で行った．

なお，計数・測定を行った関数について，以下に挙げる．これらの関数の NEURON 内部での構成は Fig. 10.1 となっている．

- `bksub()`: ケーブル方程式におけるガウス消去法部
- `triang()`: ケーブル方程式におけるガウス消去法部
- `current()`: Hodgkin-Huxley 方程式において，コンパートメントを流れる電流を計算する部位
- `state()`: Hodgkin-Huxley 方程式において，チャネルのダイナミクスを計算する部位

10.2.1　ソースコードベースでの演算数・メモリ使用量算出

この時，`exp()` 関数については，計算手法が複数あり，一意に定まらないが，京コンピュータでのプロファイラによる結果を援用し，1 `exp()` 関数につき，25 浮動小数点演算として，カウントした．

10.2.2　京コンピュータ上のプロファイラによる演算数算出

京コンピュータ上の基本プロファイラ (`fipp`) により NEURON 全体のコールグラフを作成し，計算時間のおおまかな分布について，取得した．また，精密プロファイラ (`fapp`) により，各計算部位演算数を取得した．

10.3　Hodgkin-Huxley 方程式計算部の単体性能の高速化

NEURON では，コンパートメント内のモデルについて，NMOD[] という，特殊な形式で記述し，実行時に本体とリンクさせ，実行する形式となっている．そのため，Hodgkin-Huxley 型方程式の主要な計算部分については，基本的にこの NMOD ファイルから生成された C 言語ファイルに含まれているため，この生成された C 言語ファイルをチューニングすることで，計算性能の高速化を行った (Fig. 10.2)．

10.3.1　SIMD 演算の適用

Hodgkin-Huxley 方程式は，NEURON 内では，MOD 形式で次のように記述されている．

Listing 10.1　`hh.mod`

```
1 TITLE hh.mod   squid sodium, potassium, and leak channels
```

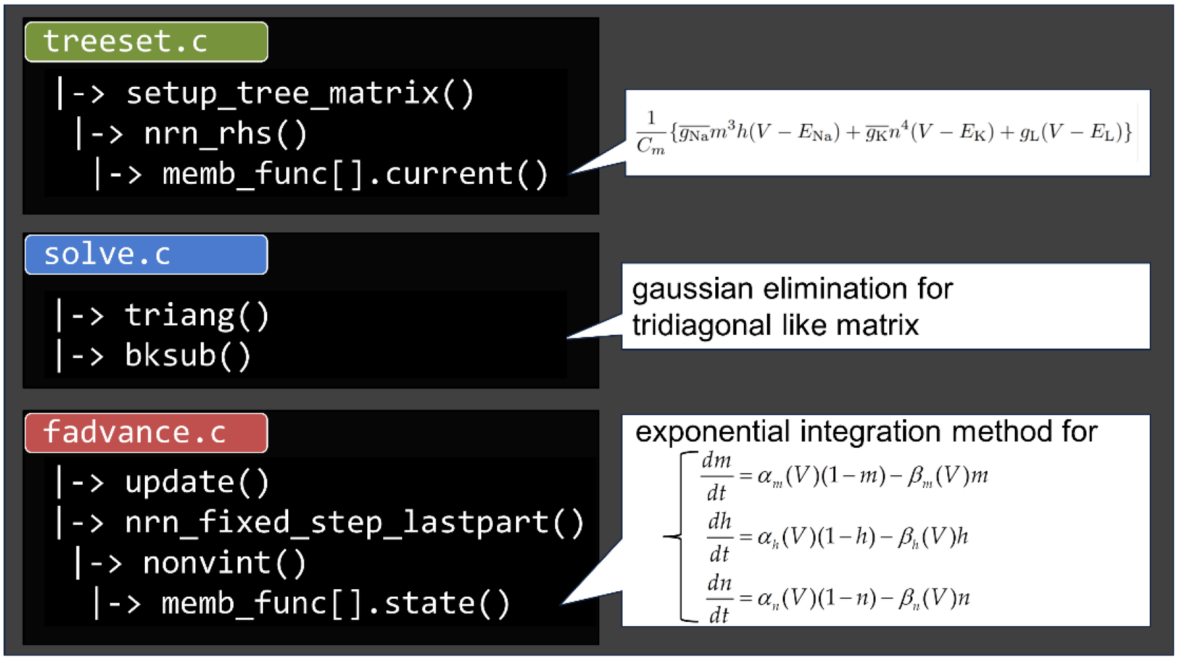


Fig.10.1　NEURON の計算構成

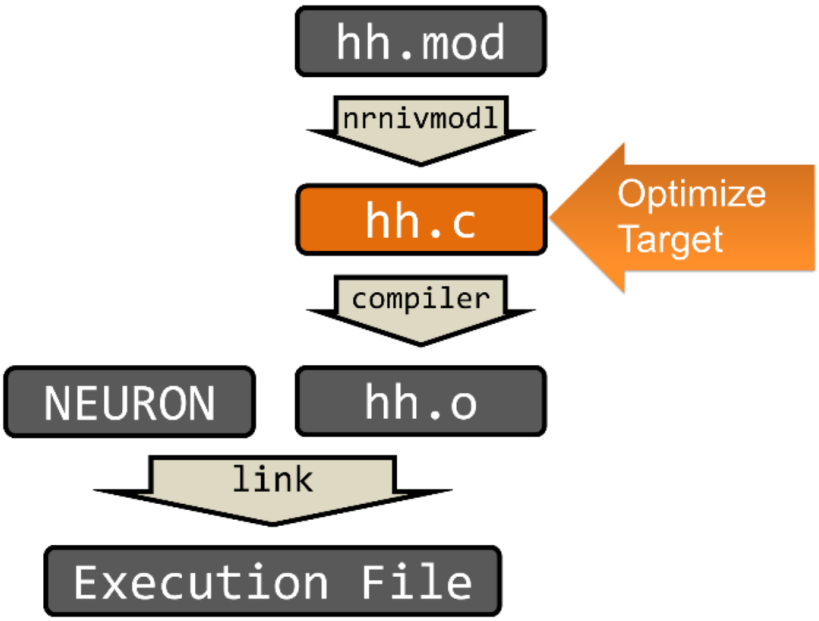


Fig.10.2　Hodgkin-Huxley 方程式計算部の高速化

```
2
3
4  UNITS {
5          (mA) = (milliamp)
6          (mV) = (millivolt)
7          (S) = (siemens)
8  }
9
10 ? interface
11 NEURON {
12         SUFFIX hh
13         USEION na READ ena WRITE ina
14         USEION k READ ek WRITE ik
15         NONSPECIFIC_CURRENT il
16         RANGE gnabar, gkbar, gl, el, gna, gk
17         GLOBAL minf, hinf, ninf, mtau, htau, ntau
18         THREADSAFE : assigned GLOBALs will be per thread
19 }
20
21 PARAMETER {
22         gnabar = .12 (S/cm2)    <0,1e9>
23         gkbar = .036 (S/cm2)    <0,1e9>
24         gl = .0003 (S/cm2)      <0,1e9>
25         el = -54.3 (mV)
26 }
27
28 STATE {
29         m h n
30 }
31
32 ASSIGNED {
33         v (mV)
34         celsius (degC)
35         ena (mV)
36         ek (mV)
37
38         gna (S/cm2)
39         gk (S/cm2)
40         ina (mA/cm2)
41         ik (mA/cm2)
42         il (mA/cm2)
43         minf hinf ninf
44         mtau (ms) htau (ms) ntau (ms)
45 }
46
47 ? currents
48 BREAKPOINT {
49         SOLVE states METHOD cnexp
50         gna = gnabar*m*m*m*h
51         ina = gna*(v - ena)
```

```
52          gk = gkbar*n*n*n*n
53          ik = gk*(v - ek)
54          il = gl*(v - el)
55  }
56
57
58  INITIAL {
59          rates(v)
60          m = minf
61          h = hinf
62          n = ninf
63  }
64
65  ? states
66  DERIVATIVE states {
67          rates(v)
68          m' =  (minf-m)/mtau
69          h' = (hinf-h)/htau
70          n' = (ninf-n)/ntau
71  }
72
73  :LOCAL q10
74
75
76  ? rates
77  PROCEDURE rates(v(mV)) {
78          :Computes rate and other constants at current v.
79          :Call once from HOC to initialize inf at resting v.
80
81          LOCAL  alpha, beta, sum, q10
82          TABLE minf, mtau, hinf, htau, ninf, ntau DEPEND celsius FROM
                -100 TO 100 WITH 200
83
84  UNITSOFF
85          q10 = 3^((celsius - 6.3)/10)
86                  :"m" sodium activation system
87          alpha = .1 * vtrap(-(v+40),10)
88          beta =  4 * exp(-(v+65)/18)
89          sum = alpha + beta
90          mtau = 1/(q10*sum)
91          minf = alpha/sum
92                  :"h" sodium inactivation system
93          alpha = .07 * exp(-(v+65)/20)
94          beta = 1 / (exp(-(v+35)/10) + 1)
95          sum = alpha + beta
96          htau = 1/(q10*sum)
97          hinf = alpha/sum
98                  :"n" potassium activation system
99          alpha = .01*vtrap(-(v+55),10)
100         beta = .125*exp(-(v+65)/80)
```

```
101         sum = alpha + beta
102         ntau = 1/(q10*sum)
103         ninf = alpha/sum
104 }
105
106 FUNCTION vtrap(x,y) {  :Traps for 0 in denominator of rate eqns.
107         if (fabs(x/y) < 1e-6) {
108                 vtrap = y*(1 - x/y/2)
109         }else{
110                 vtrap = x/(exp(x/y) - 1)
111         }
112 }
113
114 UNITSON
```

詳細に実効プロファイルをチェックしたところ，この内特に計算に時間がかかっているのは，66 行目 - 71 行目の

Listing 10.2　hh.mod の主要計算部

```
1 DERIVATIVE states {
2         rates(v)
3         m' =  (minf-m)/mtau
4         h' = (hinf-h)/htau
5         n' = (ninf-n)/ntau
6 }
```

に相当する計算であった．

この微分方程式の計算法は，49 行目の

```
1         SOLVE states METHOD cnexp
```

によって指定されており，cnexp では，Exponential Time Differencing (ETD) Crank-Nicolson 法 [] が使用される．これにより，MOD コンパイラによって C 言語に変換された後では以下の様なコードとなる．

Listing 10.3　hh.c の主要計算部

```
1 static int states (double* _p, Datum* _ppvar, Datum* _thread, _NrnThread
      * _nt) { {
2    rates ( _threadargscomma_ v ) ;
3     m = m + (1. - exp(dt*(( ( ( - 1.0 ) ) ) / mtau)))*(- ( ( ( minf ) )
          / mtau ) / ( ( ( ( - 1.0) ) ) / mtau ) - m) ;
4     h = h + (1. - exp(dt*(( ( ( - 1.0 ) ) ) / htau)))*(- ( ( ( hinf ) )
          / htau ) / ( ( ( ( - 1.0) ) ) / htau ) - h) ;
5     n = n + (1. - exp(dt*(( ( ( - 1.0 ) ) ) / ntau)))*(- ( ( ( ninf ) )
          / ntau ) / ( ( ( ( - 1.0) ) ) / ntau ) - n) ;
6    }
7   return 0;
8 }
```

この時，m，h，n は，

Listing 10.4 hh.c における各種変数の定義

```
#define gnabar _p[0]
#define gkbar _p[1]
#define gl _p[2]
#define el _p[3]
#define gna _p[4]
#define gk _p[5]
#define il _p[6]
#define m _p[7]
#define h _p[8]
#define n _p[9]
#define Dm _p[10]
#define Dh _p[11]
#define Dn _p[12]
#define ena _p[13]
#define ek _p[14]
#define ina _p[15]
#define ik _p[16]
#define v _p[17]
#define _g _p[18]
```

の様に定義されており，_p が全てのコンパートメントについて移動しながら states() 関数を実行するようなループ構造となっている．また，minf，mtau，hinf，htau，ntau，ninf といった変数は，事前に作られたマスターテーブルを元に，線形補間公式（$m_{\mathrm{Inf}} = m_{\mathrm{Inf_N}} + \theta(m_{\mathrm{Inf_{(N+1)}}} - m_{\mathrm{inf_N}})$）で計算されている．これらの部分をわかりやすくするため，以下のように擬似的に示す．

Listing 10.5 Hodgkin-Huxley 方程式計算部擬似コード（最適化前）

```
static int nrn_state() {

    for(int i = 0; i < _cntml; i++){
        minf = minf_master[N] + theta * (minf_master[N+1] - minf_master[N]);
        mtau = mtau_master[N] + theta * (mtau_master[N+1] - mtau_master[N]);
        hinf = hinf_master[N] + theta * (hinf_master[N+1] - hinf_master[N]);
        htau = htau_master[N] + theta * (htau_master[N+1] - htau_master[N]);
        ninf = ninf_master[N] + theta * (ninf_master[N+1] - ninf_master[N]);
        ntau = ntau_master[N] + theta * (ntau_master[N+1] - ntau_master[N]);

        array[i][7] = array[i][7]
```

```
12                      + (1.0 - exp( - dt / mtau) )) * (minf - array[i][7]);
                            // m
13          array[i][8] = array[i][8]
14                      + (1.0 - exp( - dt / htau) )) * (hinf - array[i][8]);
                            // h
15          array[i][9] = array[i][9]
16                      + (1.0 - exp( - dt / ntau) )) * (ninf - array[i][9]);
                            // n
17      }
18  }
```

この様に書き下すと明らかなように，このコードは，ループの中の計算が多くコンパイラが SIMD 化出来る箇所を認識しにくい，array 配列へのアクセスが非連続的でメモリフェッチに時間がかかる，といった問題が存在していることが予想できる．

そこで，minf，mtau，hinf，htau，ntau，ninf は，事前に計算し，テーブルに格納しておくことで，exp() の含まれる部分をくくりだし，また，m，h，n についても，他の部分からは参照されていないため，専用の配列に移し替える．

Listing 10.6　Hodgkin-Huxley 方程式計算部擬似コード（くくりだしによる最適化）

```
1   static int nrn_state() {
2
3       for(int i = 0; i < _cntml; i++){
4           minf_table[i] = minf_master[x]
5                + theta * (minf_master[x+1] - minf_master[x]);
6           mtau_table[i] = mtau_master[x]
7                + theta * (mtau_master[x+1] - mtau_master[x]);
8           hinf_table[i] = hinf_master[x]
9                + theta * (hinf_master[x+1] - hinf_master[x]);
10          htau_table[i] = htau_master[x]
11               + theta * (htau_master[x+1] - htau_master[x]);
12          ninf_table[i] = ninf_master[x]
13               + theta * (ninf_master[x+1] - ninf_master[x]);
14          ntau_table[i] = ntau_master[x]
15               + theta * (ntau_master[x+1] - ntau_master[x]);
16      }
17
18      for(int i = 0; i < _cntml; i++){
19          m[i] = m[i] + (1.0 - exp( - dt / mtau_table[i]) )) * (minf_table
                [i] - m[i]);
20          h[i] = h[i] + (1.0 - exp( - dt / htau_table[i]) )) * (hinf_table
                [i] - h[i]);
21          n[i] = n[i] + (1.0 - exp( - dt / ntau_table[i]) )) * (ninf_table
                [i] - n[i]);
22      }
23  }
```

これにより，m，h，n の計算がシーケンシャルかつ依存性の少ないものとなり，SIMD

化が促進される．

10.3.2 配列構造の最適化

計算を高速化させる上で，データを局在化させ，一度にアクセスするメモリの領域を出来る限り狭くする事は，重要なポイントの一つである．

ここでは，minf，mtau，hinf，htau，ntau，ninf のマスターテーブルへのアクセスが，単一コンパートメントでは常に同一の領域 x，x+1 であることに着目し，マスターテーブルの構造を変えることで，さらなる高速化を目指した．

マスターテーブルは，nrn_init() でコード 10.7 のように作成されている．なお

Listing 10.7 マスターテーブルの作成部の擬似コード（最適化前）

```
1 for (int i=0; i < 201; i++) {
2         float x = min_x + dx * i;
3       minf_master[i] = calc_minf(i);
4       mtau_master[i] = calc_mtau(i);
5       hinf_master[i] = calc_hinf(i);
6       htau_master[i] = calc_htau(i);
7       ninf_master[i] = calc_ninf(i);
8       ntau_master[i] = calc_ntau(i);
9 }
```

ここで，マスターテーブルの順序を入れ替えて，以下のようにした．

Listing 10.8 マスターテーブルの作成部の擬似コード（順序入れ替えによる最適化）

```
1 for (int i=0; i < 201; i++) {
2         float x = min_x + dx * i;
3       _t_master[i][0] = calc_minf(i);
4       _t_master[i][1] = calc_mtau(i);
5       _t_master[i][2] = calc_hinf(i);
6       _t_master[i][3] = calc_htau(i);
7       _t_master[i][4] = calc_ninf(i);
8       _t_master[i][5] = calc_ntau(i);
9 }
```

これに加え，m，h，n，及び minf，mtau，hinf，htau，ntau，ninf も単一の配列にし，ソースコード 10.6 を書き換えると，以下の用になる．

Listing 10.9 Hodgkin-Huxley 方程式計算部擬似コード（配列順序の変更による最適化）

```
1 static int nrn_state() {
2
3     for(int i = 0; i < _cntml; i++){
4         inftau_table[i][0] = minf_master[x]
5              + theta * (minf_master[x+1] - minf_master[x]);
6         inftau_table[i][1] = mtau_master[x]
```

```
7                + theta * (mtau_master[x+1] - mtau_master[x]);
8            inftau_table[i][2] = hinf_master[x]
9                + theta * (hinf_master[x+1] - hinf_master[x]);
10           inftau_table[i][3] = htau_master[x]
11               + theta * (htau_master[x+1] - htau_master[x]);
12           inftau_table[i][4] = ninf_master[x]
13               + theta * (ninf_master[x+1] - ninf_master[x]);
14           inftau_table[i][5] = ntau_master[x]
15               + theta * (ntau_master[x+1] - ntau_master[x]);
16       }
17
18       for(int i = 0; i < _cntml; i++){
19           mhn_table[i][0] = mhn_table[i][0] + (1.0 - exp( - dt /
                 inftau_table[i][1]) )) * (inftau_table[i][0] - mhn_table[i
                 ][0]);
20           mhn_table[i][1] = mhn_table[i][1] + (1.0 - exp( - dt /
                 inftau_table[i][3]) )) * (inftau_table[i][2] - mhn_table[i
                 ][1]);
21           mhn_table[i][2] = mhn_table[i][2] + (1.0 - exp( - dt /
                 inftau_table[i][5]) )) * (inftau_table[i][4] - mhn_table[i
                 ][2]);
22       }
23   }
```

10.3.3　m, h, n 計算以外の領域の高速化

これまで述べてきたように，hh.c の遅さは基本的に，データ構造がコンパートメント単位でまとめられており，メモリアクセスが非連続的になってしまっていることに起因しているものが多い．そこで，nrn_cur()，nrn_jacob() といった hh.c 内の他の関数にも，その関数で必要となる変数についてのみくくりだしたテーブルを用意し，同様の高速化を適用した．この時の，nrn_cur()，nrn_jacob() 及び nrn_state の実コードについて，コード 10.10, 10.11, 10.12 に示す．

Listing 10.10　nrn_cur()

```
1
2   static void nrn_cur(_NrnThread* _nt, _Memb_list* _ml, int _type)
3   {
4     Datum* _thread;
5     Node *_nd; int* _ni;
6     int _iml, _cntml;
7
8     if(!use_cachevec){
9       printf("ERROR␣:␣CACHEVEC␣not␣enable.\n");
10      return;
11    }
12
```

```
13    _ni = _ml->_nodeindices;
14    _cntml = _ml->_nodecount;
15    _thread = _ml->_thread;
16
17    {
18
19      // initialize
20      if(init_nrn_cur_check[_nt->_id] != 1){
21
22        for (_iml = 0; _iml < _cntml; _iml++) {
23          double *_p = _ml->_data[_iml];
24          Datum *_ppvar = _ml->_pdata[_iml];
25          int index = _iml+_nt->_id*BUFFER_SIZE;
26            _gnabar_table[index] = gnabar;
27            _gkbar_table[index] = gkbar;
28            _gl_table[index] = gl;
29            _ena_table[index] = ena;
30            _ek_table[index] = ek;
31            _el_table[index] = el;
32
33          _ion_dinadv_table[index] = _ion_dinadv;
34          _ion_dikdv_table[index] = _ion_dikdv;
35          _ion_ina_table[index] = _ion_ina;
36          _ion_ik_table[index] = _ion_ik;
37        }
38        init_nrn_cur_check[_nt->_id] = 1;
39      }
40
41
42      for (_iml = 0; _iml < _cntml; _iml++) {
43        double _gna, _ina, _gk, _ik, _il;
44        double _v = _nt->_actual_v[_ni[_iml]];
45        int index = _iml+_nt->_id*BUFFER_SIZE;
46
47          _gna = _gnabar_table[index] * _mhn_table[index][0] * _mhn_table[
              index][0] * _mhn_table[index][0] * _mhn_table[index][1];
48          _ina = _gna * ( _v - _ena_table[_iml] ) ;
49
50          _gk = _gkbar_table[index] * _mhn_table[index][2] * _mhn_table[
              index][2] * _mhn_table[index][2] * _mhn_table[index][2];
51          _ik = _gk * ( _v - _ek_table[_iml] );
52
53          _il = _gl_table[_iml] * ( _v - _el_table[_iml] );
54
55          _rhs_table[_iml] = _ina + _ik + _il;
56          _g_table[index] = _gna + _gk + _gl_table[_iml];
57
58          _ion_dinadv_table[index] += _gna;
59          _ion_dikdv_table[index] += _gk;
60
```

```
61
62        _ion_ina_table[index] += _ina;
63        _ion_ik_table[index] += _ik;
64
65      }
66
67      for (_iml = 0; _iml < _cntml; _iml++) {
68        _nt->_actual_rhs[_ni[_iml]] -= _rhs_table[_iml];
69      }
70    }
71  }
```

Listing 10.11　nrn_jacob()

```
1
2   static void nrn_jacob(_NrnThread* _nt, _Memb_list* _ml, int _type)
3   {
4     double* _p; Datum* _ppvar; Datum* _thread;
5     Node *_nd; int* _ni; int _iml, _cntml;
6
7     _ni = _ml->_nodeindices;
8     _cntml = _ml->_nodecount;
9     _thread = _ml->_thread;
10
11    for (_iml = 0; _iml < _cntml; ++_iml) {
12      _p = _ml->_data[_iml];
13      VEC_D(_ni[_iml]) += _g_table[_iml+_nt->_id*BUFFER_SIZE];
14    }
15  }
```

Listing 10.12　nrn_state()

```
1   static void nrn_state(_NrnThread* _nt, _Memb_list* _ml, int _type) {
2     int _i_table[BUFFER_SIZE];
3     double _xi_table[BUFFER_SIZE], _theta_table[BUFFER_SIZE];
4     double _inftau_table[BUFFER_SIZE*6];
5
6
7     int* _ni;
8     int _cntml;
9     Datum* _thread;
10    const double mfac_rates = _mfac_rates;
11    const double tmin_rates = _tmin_rates;
12
13    Node *_nd;
14    double _dt = dt;
15
16    _ni = _ml->_nodeindices;
17    _cntml = _ml->_nodecount;
18    _thread = _ml->_thread;
```

```
19
20    if(!use_cachevec){
21      printf("ERROR␣:␣CACHEVEC␣not␣enable.\n");
22      return;
23    }
24    if(_cntml > BUFFER_SIZE){
25      printf("ERROR␣:␣BUFFER␣OVER.\n");
26      return;
27    }
28
29  #undef exp
30    {
31      for (int _iml = 0; _iml < _cntml; _iml++) {
32        double* _p;
33        Datum* _ppvar;
34        double _xi;
35        double _v;
36
37        _p = _ml->_data[_iml];
38        _ppvar = _ml->_pdata[_iml];
39        _v = _nt->_actual_v[_ni[_iml]];
40        _xi = mfac_rates * (_v - tmin_rates);
41        _i_table[_iml] = (int) _xi;
42        _theta_table[_iml] = _xi - (double)_i_table[_iml];
43      }
44
45      for (int _iml = 0; _iml < _cntml; _iml++) {
46        int index = _i_table[_iml]*6;
47        _inftau_table[_iml*6+0] = _t_master[index+0] + _theta_table[_iml
            ]*(_t_master[index+6] - _t_master[index+0]);
48        _inftau_table[_iml*6+1] = _t_master[index+1] + _theta_table[_iml
            ]*(_t_master[index+7] - _t_master[index+1]);
49        _inftau_table[_iml*6+2] = _t_master[index+2] + _theta_table[_iml
            ]*(_t_master[index+8] - _t_master[index+2]);
50        _inftau_table[_iml*6+3] = _t_master[index+3] + _theta_table[_iml
            ]*(_t_master[index+9] - _t_master[index+3]);
51        _inftau_table[_iml*6+4] = _t_master[index+4] + _theta_table[_iml
            ]*(_t_master[index+10] - _t_master[index+4]);
52        _inftau_table[_iml*6+5] = _t_master[index+5] + _theta_table[_iml
            ]*(_t_master[index+11] - _t_master[index+5]);
53      }
54
55      for (int _iml = 0; _iml < _cntml; _iml++) {
56        _mhn_table[_iml+_nt->_id*BUFFER_SIZE][0] += (1.0 - exp(- _dt /
            _inftau_table[_iml*6+1]) ) * ( _inftau_table[_iml*6+0]  -
            _mhn_table[_iml+_nt->_id*BUFFER_SIZE][0]);
57        _mhn_table[_iml+_nt->_id*BUFFER_SIZE][1] += (1.0 - exp(- _dt /
            _inftau_table[_iml*6+3]) ) * ( _inftau_table[_iml*6+2]  -
            _mhn_table[_iml+_nt->_id*BUFFER_SIZE][1]);
58        _mhn_table[_iml+_nt->_id*BUFFER_SIZE][2] += (1.0 - exp(- _dt /
```

```
          _inftau_table[_iml*6+5]) ) * ( _inftau_table[_iml*6+4]  -
          _mhn_table[_iml+_nt->_id*BUFFER_SIZE][2]);
    }
  }
#define exp hoc_Exp

}
```

10.4　OpenCL/PezyCL の適用

GPU や PEZY-SC といったヘテロジーニアス環境で高速計算を行うため，Hodgkin-Huxley 方程式計算部位の OpenCL/PezyCL 化を行った．これらのデバイスは，CPU とメモリを共有していないため，計算に必要なデータについては，明示的にデバイスに送信する必要がある．OpenCL/PezyCL では，`clCreateBuffer()` といった関数をもちいて，転送用バッファの読み書き属性やサイズを定義することができる．

以下に，NEURON の MOD 内部にて，OpenCL/PezyCL 向けにメモリ転送を行う部分のコードについて載せる．

Listing 10.13　Hodgkin-Huxley MOD with OpenCL

```
#ifdef KPLUS_USE_OPENCL
#include "opencl_utils.cpp"

typedef double FLOAT;
typedef struct __hh_mem_dev
{
  unsigned long num;
  cl_mem _n;
  cl_mem _m;
  cl_mem _h;
  cl_mem _v;
  cl_mem _g_tmp;
  cl_mem _i;
  cl_mem table;
} HH_Mem_Dev;

CLInfo _cli;
CLInfo *cli;
HH_Mem_Dev _hh_mem_dev;
HH_Mem_Dev *hh_mem_dev;

cl_kernel hh_kernel;
double time_calc=0.0, time_copy=0.0;


```

```
26 void init_hh_mem_dev (CLInfo *cli, const unsigned int ndata, HH_Mem_Dev
       *hh_mem_dev)
27 {
28   FLOAT *n_table = &(_n_table[BUFFER_SIZE * 0]);
29   FLOAT *m_table = &(_m_table[BUFFER_SIZE * 0]);
30   FLOAT *h_table = &(_h_table[BUFFER_SIZE * 0]);
31
32   size_t memsize = ndata * sizeof(FLOAT);
33   size_t table_memsize = TABLE_SIZE * 6 * sizeof(FLOAT);
34
35   hh_mem_dev->num = ndata;
36   hh_mem_dev->_n  = clCreateBuffer(cli->context,  CL_MEM_READ_WRITE,
         memsize, NULL, NULL);
37   hh_mem_dev->_m  = clCreateBuffer(cli->context,  CL_MEM_READ_WRITE,
         memsize, NULL, NULL);
38   hh_mem_dev->_h  = clCreateBuffer(cli->context,  CL_MEM_READ_WRITE,
         memsize, NULL, NULL);
39   hh_mem_dev->_v  = clCreateBuffer(cli->context,  CL_MEM_READ_WRITE,
         memsize, NULL, NULL);
40   hh_mem_dev->_g_tmp  = clCreateBuffer(cli->context,  CL_MEM_WRITE_ONLY,
          memsize, NULL, NULL);
41   hh_mem_dev->_i  = clCreateBuffer(cli->context,  CL_MEM_WRITE_ONLY,
         memsize, NULL, NULL);
42   hh_mem_dev->table = clCreateBuffer(cli->context,  CL_MEM_READ_WRITE,
         table_memsize, NULL, NULL);
43
44   clEnqueueWriteBuffer(cli->queue, hh_mem_dev->_n,  CL_TRUE, 0, memsize,
          n_table,  0, NULL, NULL);
45   clEnqueueWriteBuffer(cli->queue, hh_mem_dev->_m,  CL_TRUE, 0, memsize,
          m_table,  0, NULL, NULL);
46   clEnqueueWriteBuffer(cli->queue, hh_mem_dev->_h,  CL_TRUE, 0, memsize,
          h_table,  0, NULL, NULL);
47   clEnqueueWriteBuffer(cli->queue, hh_mem_dev->table,  CL_TRUE, 0,
         table_memsize, hh_table, 0, NULL, NULL);
48
49   clFinish(cli->queue);
50 }
51
52
53 void set_kernel_hh(CLInfo *cli, const char *kernelname, HH_Mem_Dev *
       hh_mem_dev, cl_kernel *kernel)
54 {
55   *kernel = clCreateKernel(cli->program, kernelname, NULL);
56
57   clSetKernelArg(*kernel, 0, sizeof(unsigned long), &(hh_mem_dev->num));
58   clSetKernelArg(*kernel, 1, sizeof(cl_mem), &(hh_mem_dev->_n));
59   clSetKernelArg(*kernel, 2, sizeof(cl_mem), &(hh_mem_dev->_m));
60   clSetKernelArg(*kernel, 3, sizeof(cl_mem), &(hh_mem_dev->_h));
61   clSetKernelArg(*kernel, 4, sizeof(cl_mem), &(hh_mem_dev->_v));
62   clSetKernelArg(*kernel, 5, sizeof(cl_mem), &(hh_mem_dev->_g_tmp));
```

```
63    clSetKernelArg(*kernel, 6, sizeof(cl_mem), &(hh_mem_dev->_i));
64    clSetKernelArg(*kernel, 7, sizeof(cl_mem), &(hh_mem_dev->table));
65    //clSetKernelArg(*kernel, 1, sizeof(unsigned long), &i);
66  }
67
68  void calc_opencl_step (CLInfo *cli, cl_kernel *kernel, HH_Mem_Dev *
        hh_mem_dev)
69  {
70    FLOAT *v_table = &(_v_table[BUFFER_SIZE * 0]);
71    FLOAT *g_table   = &(_g_table[BUFFER_SIZE * 0]);
72    FLOAT *il_table  = &(_il_table[BUFFER_SIZE * 0]);
73
74    size_t global_item_size = hh_mem_dev->num;
75    cl_event ev_calc, ev_copy;
76    cl_int ret;
77
78    clEnqueueWriteBuffer(cli->queue, hh_mem_dev->_v,  CL_TRUE, 0,
          hh_mem_dev->num * sizeof(FLOAT), v_table,  0, NULL, NULL);
79    ret = clFinish(cli->queue);
80    ret = clEnqueueNDRangeKernel (cli->queue, *kernel, 1, NULL, &
          global_item_size, NULL, 0, NULL, &ev_calc);
81    if (ret != CL_SUCCESS)
82      {
83        printf ("Error␣:␣Enqueue␣kernel␣failed\n");
84        exit(-1);
85      }
86    ret = clFinish(cli->queue);
87    clEnqueueReadBuffer (cli->queue, hh_mem_dev->_i, CL_TRUE, 0,
          hh_mem_dev->num * sizeof(FLOAT), il_table, 0, NULL, NULL);
88    clEnqueueReadBuffer (cli->queue, hh_mem_dev->_g_tmp, CL_TRUE, 0,
          hh_mem_dev->num * sizeof(FLOAT), g_table, 0, NULL, NULL);
89    ret = clFinish(cli->queue);
90    if (ret != CL_SUCCESS)
91      {
92        printf ("Error␣:␣Enqueue␣kernel␣failed\n");
93        exit(-1);
94      }
95    //printf ("v=%f, g=%f, i=%f\n", v_table[0], g_table[0], il_table[0]);
96
97    //time_calc += get_opencl_time(ev_calc);
98    //time_copy += get_opencl_time(ev_copy);
99  }
100
101 static void myopencl_init (const unsigned int ndata)
102 {
103
104   cli = &_cli;
105   hh_mem_dev = &_hh_mem_dev;
106
107   init_cl (cli);
```

```
108   if (cli->state != 1)
109     {
110       printf ("No␣CL␣Device␣found.\n");
111   exit(-1);
112     }
113   print_cl_info (cli);
114
115 #ifdef USE_PZCL
116   make_cl_program_with_binary (cli, "./kernel.sc32/kernel.pz");
117 #else
118   make_cl_program_with_source (cli, "./cl/hh.cl");
119 #endif
120
121   init_hh_mem_dev (cli, ndata, hh_mem_dev);
122   set_kernel_hh (cli, "HH", hh_mem_dev, &hh_kernel);
123 }
124
125 #endif /* KPLUS_USE_OPENCL */
```

10.5 OpenMP によるハイブリッド並列化

NEURON には，POSIX Thread によるスレッド並列の機構が備わっているが，京コンピュータ上では，OpenMP によるスレッド並列化しかサポートされていない．また，スレッド並列の手法についても，NEURON が採用している細胞単位での並列化やり方以外にもコンパートメント単位で行うもの，その両方を兼ねるもの等が考えられた．そこで新たに OpenMP ベースのハイブリッド並列化機構を実装することとした．

NEURON のシミュレーションにおける 1 step の計算は，`src/nrnoc/fadvance.c` の `nrn_fixed_step_thread()` 関数が基盤となっている．そこでまずは，`nrn_fixed_step_thread()` 関数の概要を，擬似コードで示す（ここではかなり簡略化している）．この時，外側のループは元々の実装ではスレッドごとのループだが，ここでは，細胞ごとのループと読み替えている．

Listing 10.14 `nrn_fixed_step_thread()` 関数の擬似コード

```
1  nrn_fixed_step_group_thread()
2  {
3    deliver_net_event();        // Spike情報等の通信
4
5    for(int i; i<NCELL; i++){   // 全ての細胞についてのループ
6      setup_tree_matrix();      // 木構造のチェック
7      nrn_solve();              // Cable方程式の計算
8      nonvint();                // 下記
9    }
10 }
11
```

```
12 nonvint()
13 {
14   for(int i; i<NCOMP; i++){  // 全てのコンパートメントについてループ
15     calc_mechanism();        // MODファイルで記述されたメカニズムを呼び
         出す.
16   }
17 }
```

手法 1：細胞単位のスレッド並列化

通信以外の部分は，細胞ごとに全て独立に処理を行うことができる．そこで，第一の方法としては，全ての細胞について回っているループに対して OpenMP を適用することが考えられる．

これは，擬似コードで表すと以下のようになる．

Listing 10.15　nrn_fixed_step_thread() 関数の擬似コード（手法 1）

```
1  nrn_fixed_step_group_thread()
2  {
3    deliver_net_event();       // Spike情報等の通信
4
5  #pragma omp parallel for
6    for(int i; i<NCELL; i++){  // 全ての細胞についてのループ
7      setup_tree_matrix();     // 木構造のチェック
8      nrn_solve();             // Cable方程式の計算
9      nonvint();               // 下記
10   }
11 }
12
13 nonvint()
14 {
15   for(int i; i<NCOMP; i++){  // 全てのコンパートメントについてループ
16     calc_mechanism();        // MODファイルで記述されたメカニズムを呼び
         出す.
17   }
18 }
```

手法 2

CPU コア間でのキャッシュのヒット率を向上させるためには，OpenMP を適用するループはできるだけ，計算そのものの近くのほうが良いと考えられる．また，擬似コードの関数のうち，nonvint() の計算量が多いことは事前にわかっている．

そこで，次のようなスレッド並列化も考えられる．

Listing 10.16　nrn_fixed_step_thread() 関数の擬似コード（手法 2）

```
nrn_fixed_step_group_thread()
{
  deliver_net_event();        // Spike情報等の通信

  for(int i; i<NCELL; i++){   // 全ての細胞についてのループ
    setup_tree_matrix();      // 木構造のチェック
    nrn_solve();              // Cable方程式の計算
    nonvint();                // 下記
  }
}

nonvint()
{
#pragma omp parallel for
  for(int i; i<NCOMP; i++){   // 全てのコンパートメントについてループ
    calc_mechanism();         // MODファイルで記述されたメカニズムを呼び
        出す.
  }
}
```

手法 3

第 3 の手法としては，手法 1 と 2 の両方を組み合わせる事が考えられる．擬似コードで示すと以下のようになる．

Listing 10.17 `nrn_fixed_step_thread()` 関数の擬似コード（手法 3）

```
nrn_fixed_step_group_thread()
{
  deliver_net_event();        // Spike情報等の通信

#pragma omp parallel for
  for(int i; i<NCELL; i++){   // 全ての細胞についてのループ
    setup_tree_matrix();      // 木構造のチェック
    nrn_solve();              // Cable方程式の計算
  }

  for(int i; i<NCELL; i++){   // 全ての細胞についてのループ
    nonvint();                // 下記
  }
}

nonvint()
{
#pragma omp parallel for
  for(int i; i<NCOMP; i++){   // 全てのコンパートメントについてループ
    calc_mechanism();         // MODファイルで記述されたメカニズムを呼び
        出す.
  }
```

```
22 }
```

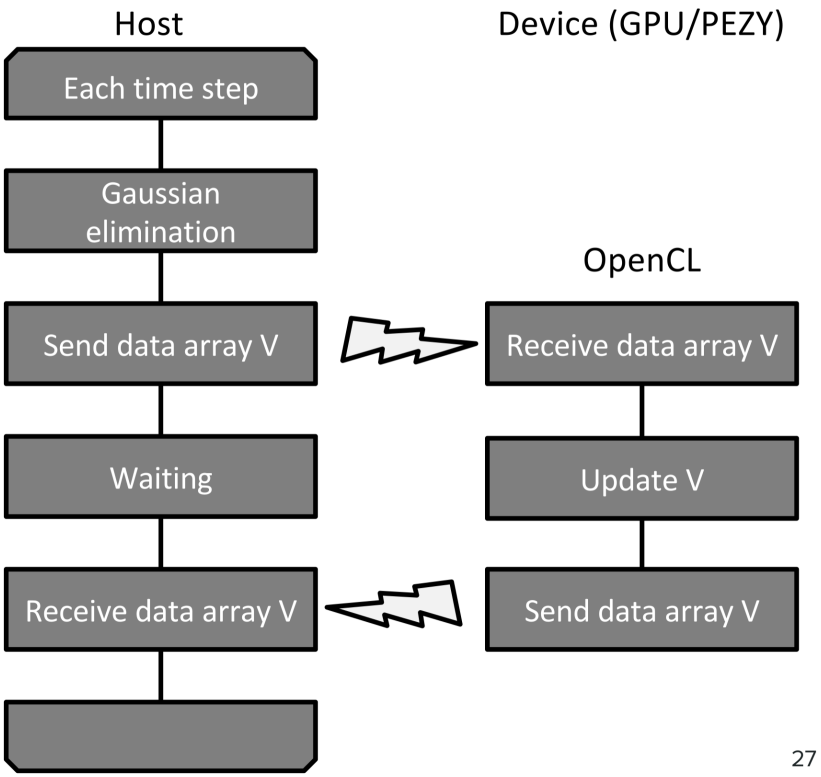


Fig.10.3　OpenCL/PezyCL 利用時の概念図

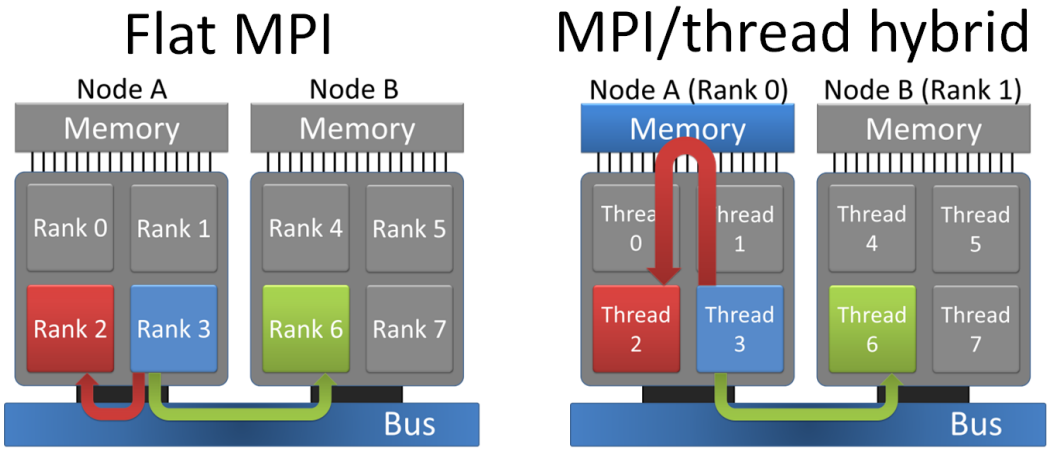


Fig.10.4　OpenMP によるハイブリッド並列

10.6　細胞分割による計算時間の短縮

計算速度の向上とは別の観点からの計算時間短縮の手法として，1 細胞を複数の領域に分割し，複数のノード上で計算を行う手法 (Fig. 10.5) の開発を行った．NEURON には，1 細胞を複数の領域に分割して計算を行う事のできる multisplit という関数の実装 [48] があったため，これを nrc-gen (7.3) に組み込み，複数細胞の同時計算にも対応できるように拡張を行った．

なお，細胞分割では，計算結果に対する影響はほぼないと考えられる．

以下に `multisplit()` の使用例について示す．

Listing 10.18　multisplit による細胞分割計算

```
{load_file("nrngui.hoc")}
{load_file("netparmpi.hoc")}
{load_file("binfo.hoc")}

{load_file("myloadbal.hoc")}
{load_file("loadSwc.hoc")}
{load_file("mcomplex.hoc")}

func multisplit() {\
    local c, cm, maxfactor\
    localobj b, ms, vs, bi, cb, nc, _pc, nil

    b = new MyLoadBalance()
    _pc = new ParallelContext()
    maxfactor = 0.3

    read_mcomplex(b)
    if (_pc.nhost > 1) {
        ms = new Vector(100)
        if (_pc.id == 0) {
            c = b.cell_complexity()
            cm = c * maxfactor / _pc.nhost
            //print "c = ", c, "  maxfactor = ", maxfactor, "  cm = ",
                cm
            b.multisplit(0, cm, ms)
            //bprint(ms)
        }
        _pc.broadcast(ms, 0)
        vs = new VectorStream(ms)
        bi = new BalanceInfo()
        bi.msgid = 1e6
        bi.nhost = _pc.nhost
        bi.ihost = _pc.id
        bi.bilist.append(new CellBalanceInfo(vs))
```

```
34          bi.mymetis2(_pc.nhost)
35
36          //if (_pc.id == 0) { bi.stat() }
37
38          cb = bi.bilist.object(0)
39          nc = new NetCon(&v(.5), nil)
40          cb.multisplit(nc, 100, _pc, _pc.id)
41      }
42      //printf("%d cpu complexity %g\n", _pc.id, b.cpu_complexity())
43      cplx = b.cpu_complexity()
44
45      return(cplx)
46  }
```

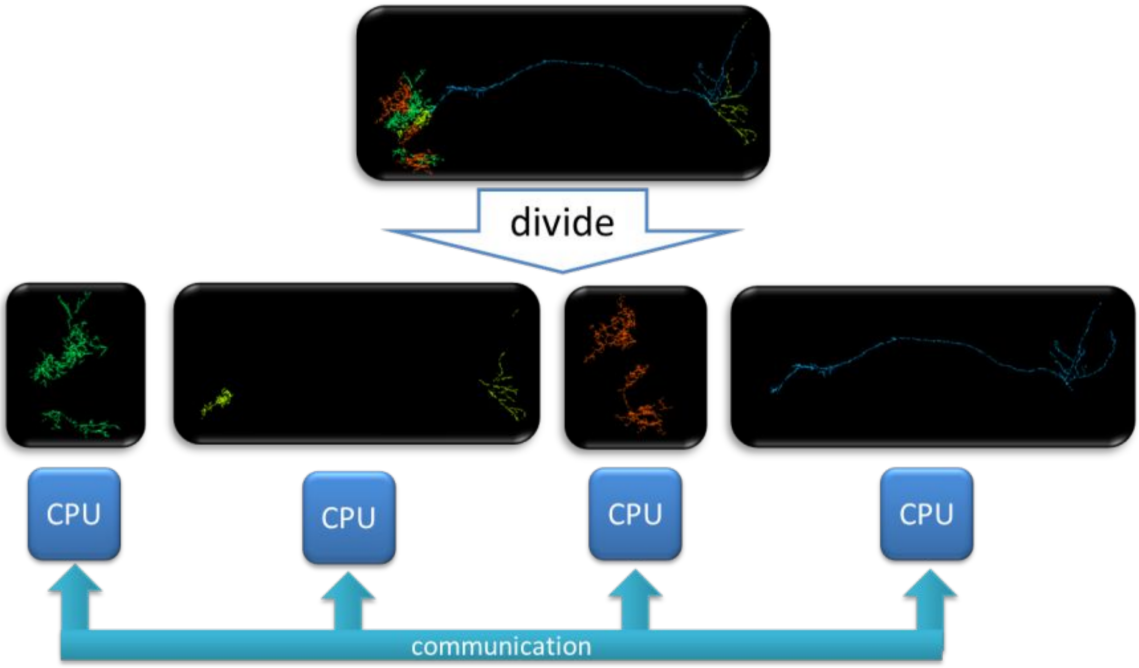


Fig.10.5　細胞分割概念図

10.7　細胞形態縮約による計算時間の短縮

10.7.1　枝刈り・短絡手法

計算速度の向上とは別の観点からの計算時間短縮の手法として細胞形態縮約手法を開発した．ここでは，枝刈りと短絡という二種類の手法を，繰り返し細胞形態ファイル (SWC) に適用することで，大まかな細胞形態を維持しながら，コンパートメント数を減らした細胞形態ファイル (SWC) を得ることができる (Fig. 10.6, 10.7).

- 手法 1 枝刈り: 末端枝の削除
- 手法 2 短絡: 分枝のない中間点の短絡

本手法を実装したツールとして，reduct_neuron(7.5) を開発した．

なお，この手法では，計算時間の低減が期待できるものの，縮約した分，シミュレーション結果に影響をあたえるため，どの程度の差が生まれるか，測定・検証する必要がある．

10.7.2　電気生理学的特性を考慮した枝刈り・短絡手法

細胞形態縮約を行った際の，シミュレーションへの影響を低減するため，電気生理学的特性を考慮した枝刈り・短絡手法の開発を行った．ここでは，手法 1,2 をそれぞれ以下のように修正した．

- 手法 1' 枝刈り: 末端枝の削除と，削除した枝の体積を親枝に半径増加として繰り込み
- 手法 2' 短絡: 分枝のない中間点を短絡し，新しいコンパートメントの長さについては元の 2 つの枝の合計と同じ値に伸長する

10.7.3　細胞形態縮約のシミュレーションに与える影響の評価

カイコガ 86 細胞モデルのような入出力領域が定義されている細胞について，縮約のシミュレーション結果に与える影響について評価する手法を構築した．ここではまず，入力領域と出力領域の中からランダムに刺激点と測定点を選択し (Fig. 10.8)，刺激点に sin 波を際の，測定点でのスパイク数について計測し，縮約前後で比較を行った．

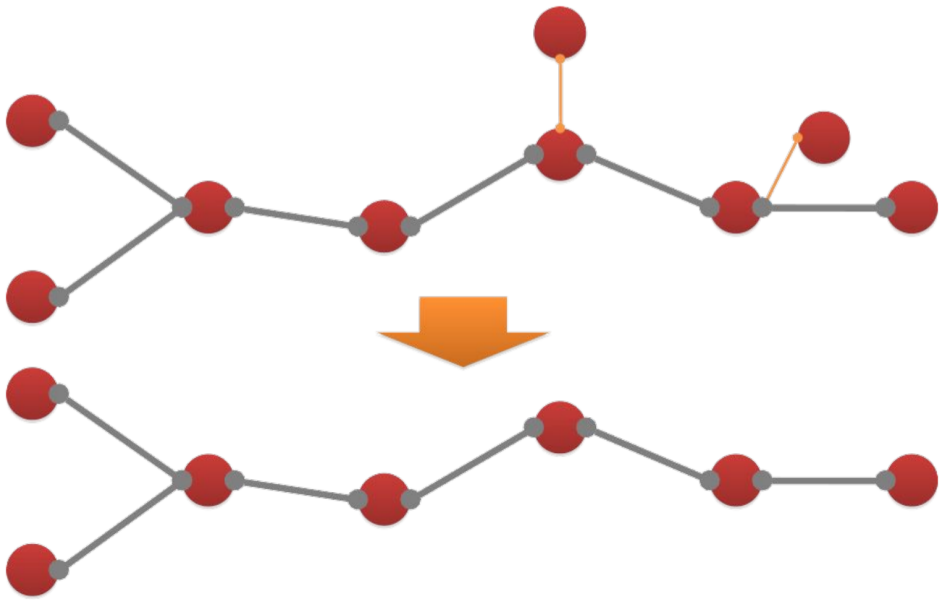


Fig.10.6　コンパートメント数縮約手法 1: 末端枝の枝刈り

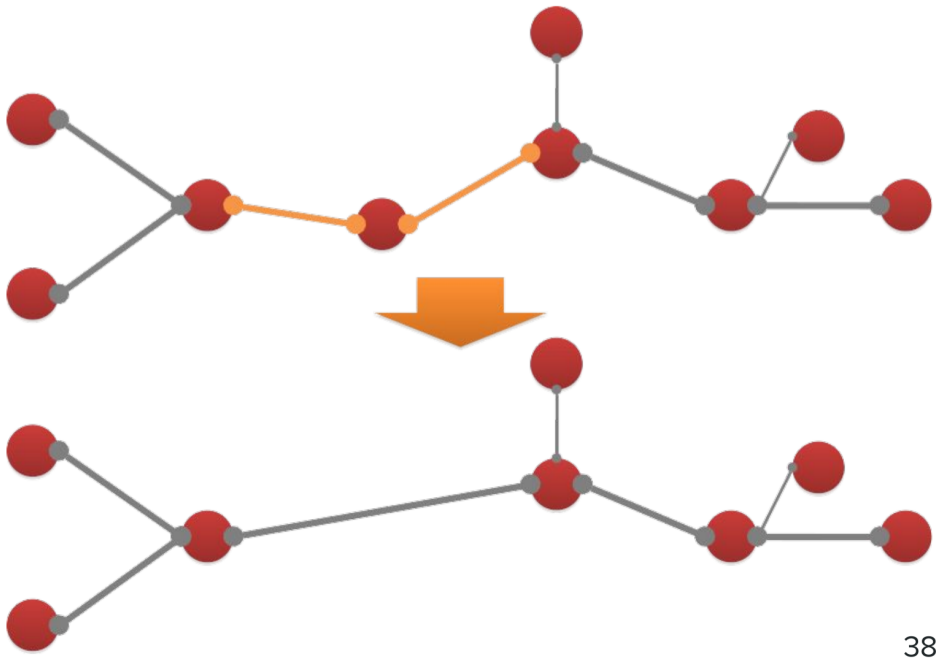


Fig.10.7　コンパートメント数縮約手法 2: 分枝のない中間点の短絡

10.7.4　神経回路の縮約手法

本縮約手法を，神経回路について適用するために，許容される最大縮約レベルを R とした際に，以下の手順で操作を行った．

1. 神経回路の中で最もコンパートメント数の大きい物を選択
2. 選択された細胞について，既に最大縮約レベルに達していたら，処理を終了する
3. 選択された細胞を 1 段階縮約し，1 に戻る

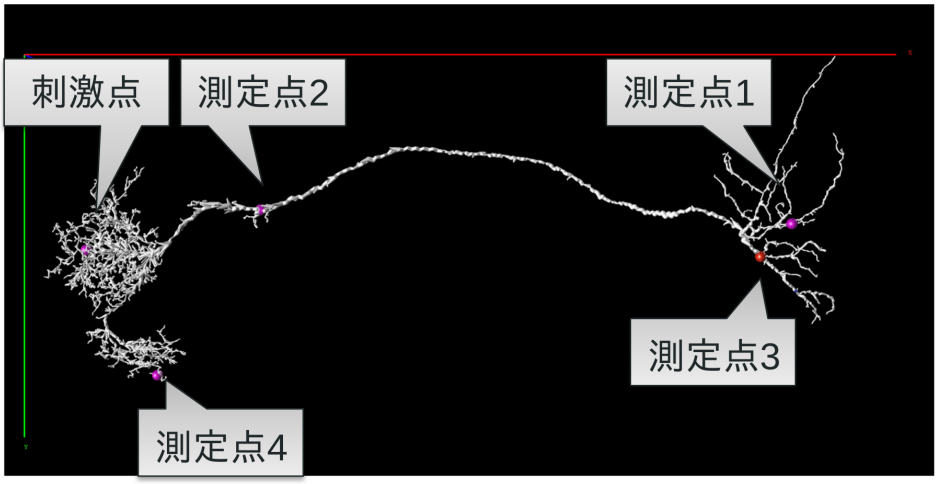


Fig.10.8　縮約手法のシミュレーションへの影響評価

第 III 部

結果

第 11 章

マルチコンパートメント Hodgkin-Huxley 型モデルの計算特性と高速化

マルチコンパートメント Hodgkin-Huxley モデルは，計算量が大きく，CPU 内での命令レベル並列化を活用することで大幅な高速化を行うことが可能である．本章では，カイコガ均一神経細胞モデル (Table 9.1) を使用し，MCHH モデルの演算数やメモリアクセス量といった計算プロファイルを明らかにし，その情報を元に複数の CPU/GPU 環境に向けて高速化を行った．

11.1 マルチコンパートメント Hodgkin-Huxley 型モデルの数値計算手法の比較

複数の計算機間で演算性能の比較を行ったり，高速化を行うためには，対象となる計算手法の必要とする演算量やメモリアクセス量といった計算特性の情報が必要不可欠である．

神経細胞モデルの数値計算手法としては，一般的な計算手法である Euler 法や 4th order Runge-Kutta 法などの他に，急激な変化にたいして安定と言われる Exponential Euler 法が用いられることがある [115, 12]．この計算法は NEURON では，MCHH に対するデフォルトの計算法として用いられる．

そこで，これらの手法について，NEURON ソースコード上の演算数及び配列変数へのアクセス量を計数した．その結果が，Fig. 11.1 である．

Table11.1 Number of floating-point operations of the Hodgkin-Huxley part of the benchmark simulation

	Conditions	Floating-point OPs	Memory Traffic	Arithmetic i
Hodgkin-Huxley eq.	Euler, without LUT	$50 + 6 \times \exp = 240$	120 byte	2.00
Hodgkin-Huxley eq.	Euler, with LUT	37	144 byte	0.25
Hodgkin-Huxley eq.	EI, without LUT	$44 + 9 \times \exp = 314$	120 byte	2.61
Hodgkin-Huxley eq.	EI, with LUT	$30 + 3 \times \exp = 130$	144 byte	0.90
Hodgkin-Huxley eq.	RK4, without LUT	$219 + 24 \times \exp = 915$	120 byte	7.63
Hodgkin-Huxley eq.	RK4, with LUT	103	144 byte	0.71
Cable eq.		16	88 byte	0.18

11.2 NEURON におけるマルチコンパートメント Hodgkin-Huxley 方程式の計算構造と計算時間

NEURON における計算構造を明らかにするため，京コンピュータ上のプロファイラにより，コールグラフの作成を行った．主要部分について，以下に示す．

Listing 11.1 Call Graph (チューニング前)

```
 0% <13> nrn_fixed_step_thread [1 / 5781]
 0%   <14> nrn_fixed_step_lastpart [0 / 4310]
 0%     <15> nonvint [0 / 4305]
74%       <16> nrn_state [75 / 4299]
 0%   <14> setup_tree_matrix [0 / 906]
 3%     <15> nrn_rhs [178 / 676]
 8%       <16> nrn_cur [246]
 2%     <15> nrn_lhs [119]
 1%     <15> nrn_cap_jacob [34]
 1%     <15> nrn_jacob [77]
 1%   <14> triang [82]
 6%   <14> bksub [368]
 1%   <14> nrn_capacity_current [32]
 1%   <14> update [81]
```

これより，計算構造としては，

- ケーブル方程式を連立一次方程式として解く，`triang()`, `bksub()`
- Hodgkin-Huxley 方程式の第一式である V についての式を計算する `nrn_cur()`
- Hodgkin-Huxley 方程式の m, h, n についての式を計算する `nrn_state()`

といった部分が，大部分を占めていることがわかった．また，このうち，`nrn_cur()` 及び `nrn_state()` は，NMODL 内部の関数である．

これらのうちでも特に `nrn_state()` の占める割合が大きく，特に，この中の `exp()` 関数が大きな時間を占めていることがわかった．

及びソースコード上のタイマーにより，計算時間について測定を行った．

11.3 NEURON built-in 高速化手法の検証

11.4 単体性能の高速化

10 章にて述べた手法に基づき，カイコガ均一神経細胞モデル (9.2) による小スケール神経回路シミュレーションでの単体ノードでの高速化を行った．この時，関数レベルでの性能向上について，Fig. 11.1 に示す．

また，複数の計算機について，Hodgkin-Huxley モデルを計算している部位である `state()` 関数について，Roofline 解析を行った (Fig. 11.3， 11.4， 11.5) ．いずれの計算機においても，単体性能高速化を行うことで，STREAM ベンチマークにより測定した実効メモリ速度をもとに算出した理論限界性能に近似した性能を得ることができた．

また，各計算機での最終的な単体性能について，Fig. 11.2 に示す．実行時間全体をみても，全ての環境において大きな（2 - 5 倍）性能向上を確認することができた．

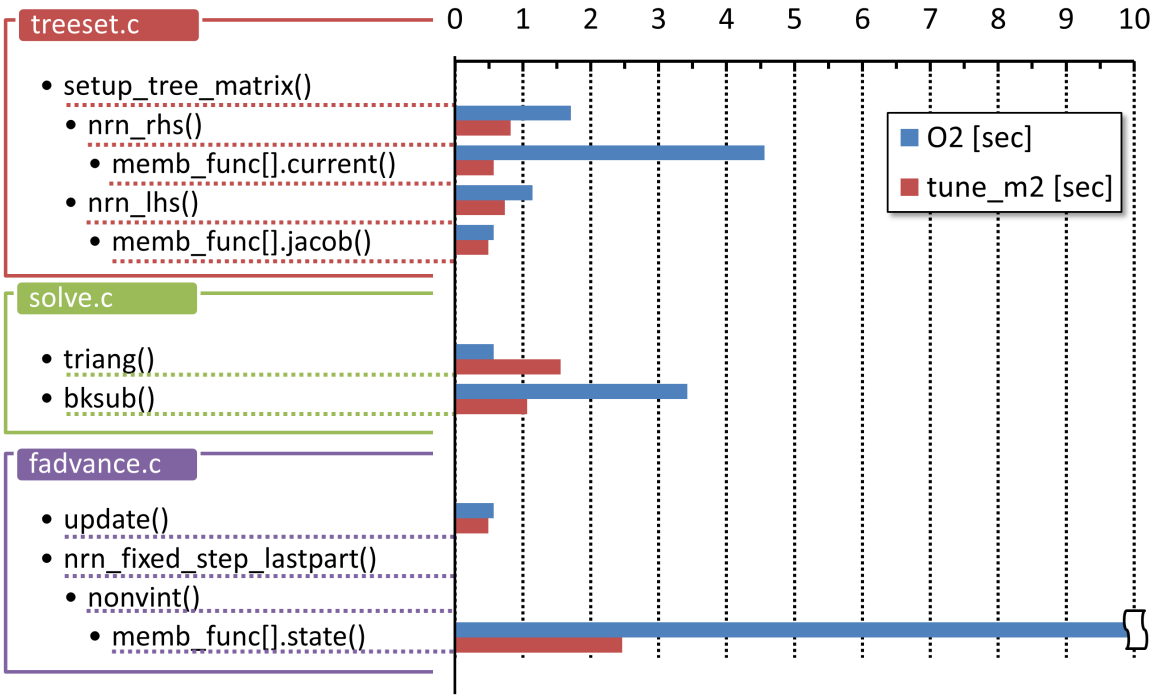


Fig.11.1　京コンピュータにおける演算時間評価

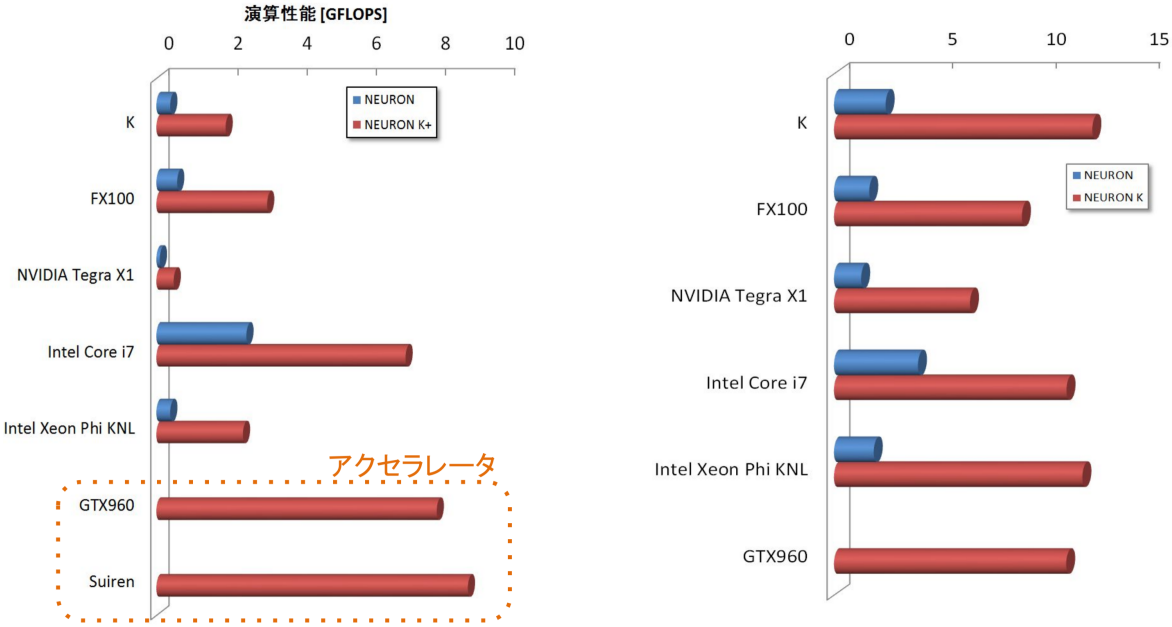


Fig.11.2　単体性能比較

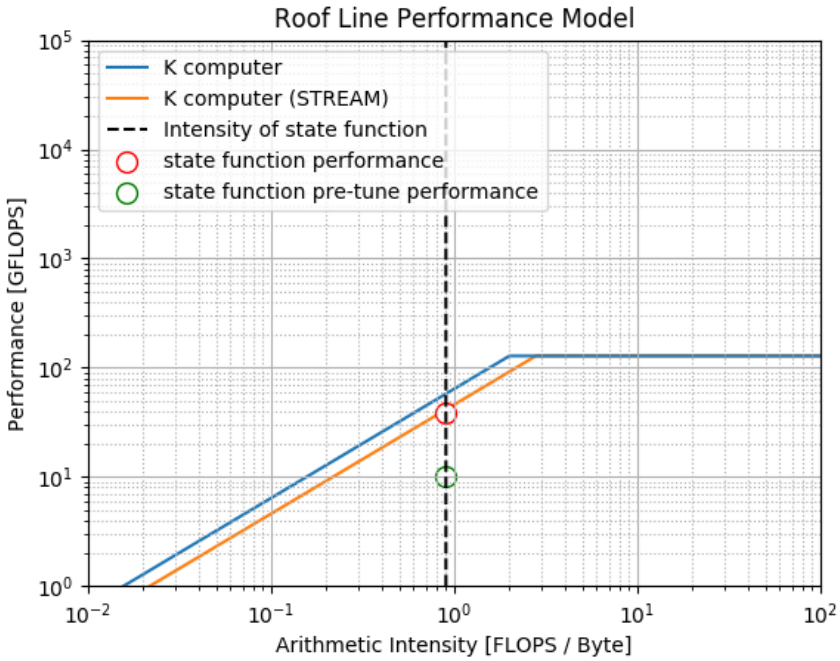


Fig.11.3　京コンピュータでの Roofline 解析

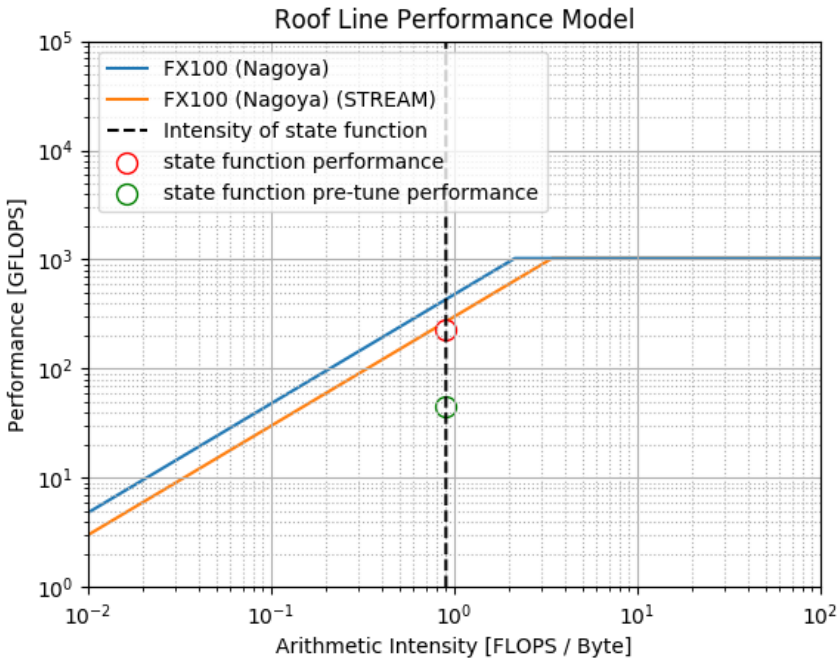


Fig.11.4　FX100 での Roofline 解析

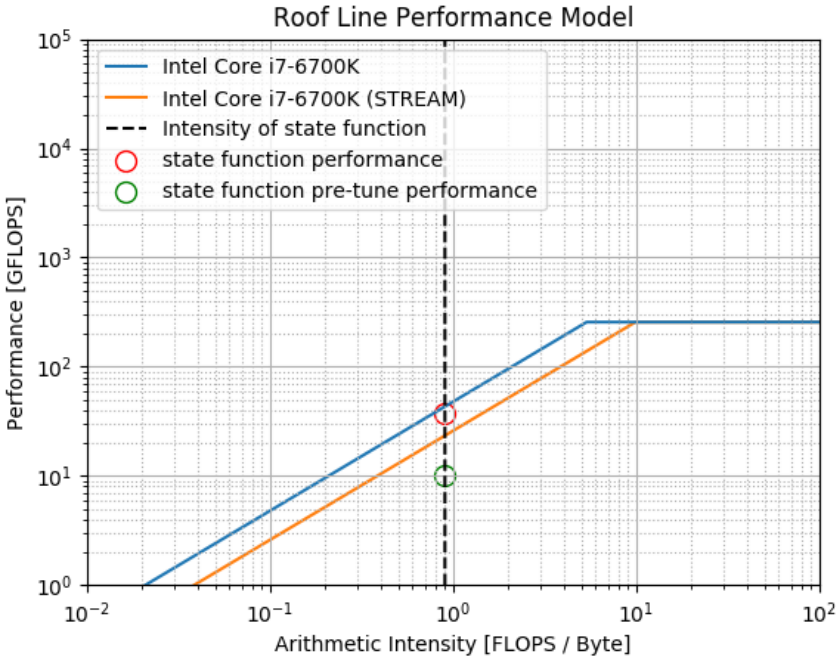


Fig.11.5　Intel i7-6700K での Roofline 解析

第 12 章

均一な神経細胞による大規模シミュレーション

スーパーコンピュータ環境における神経回路シミュレーションの並列化については，細胞ごとの並列性を利用し，別々の細胞をノードに割り振り，シナプス伝達においてのみ，MPI を用いて通信を行う事が一般的である．しかし，神経細胞数を固定したうえで，更なる大規模化を行うためには，1 細胞を複数のノードに分割して計算することが必要不可欠となる．また，リアルタイムでの計算を行うための手法として，1 細胞あたりのコンパートメント数を縮約する手法を開発し，その効果について測定を行った．

12.1 細胞分割

12.1.1 単一細胞の分割結果

カイコガ LAL-VPC 領域の神経細胞である Fig. 12.1 に対し，NEURON による細胞の分割手法である `multisplit` 法 [52] を用いて京コンピュータ上で細胞分割計算のスケーリングを行った．

この時，結果は Fig. 12.2 のようになり，実行速度では 64 分割時が最速となった．

12.1.2 大規模シミュレーションの細胞分割による計算時間短縮

単一細胞での結果を元に，京コンピュータ上で，カイコガ均一神経細胞モデル (Table 9.1) を用いて大規模スケーリングを行った．この時，NEURON では，multisplit 法の複数細胞への実装が提供されていなかったため，事前に分割済の細胞を用意し，NRC 上の命令として，`mutlsplit` の部分使用による細胞接続を用いることで，実現した．

結果を Fig. 12.3 に示す．この時，細胞分割を行わない場合では，CPU コア数に対し

て，同数または実行効率を上げるために数倍程度の細胞数を用いる必要があったが，細胞分割手法を導入することで，CPU コア数に対して，$\frac{1}{64}$ 個の細胞で計算を行うことが可能となった．その結果，663,552 CPU コア使用時に，10,368 細胞について，リアルタイムシミュレーションに対して約 2 倍遅い程度の時間でシミュレーションを行うことができた．

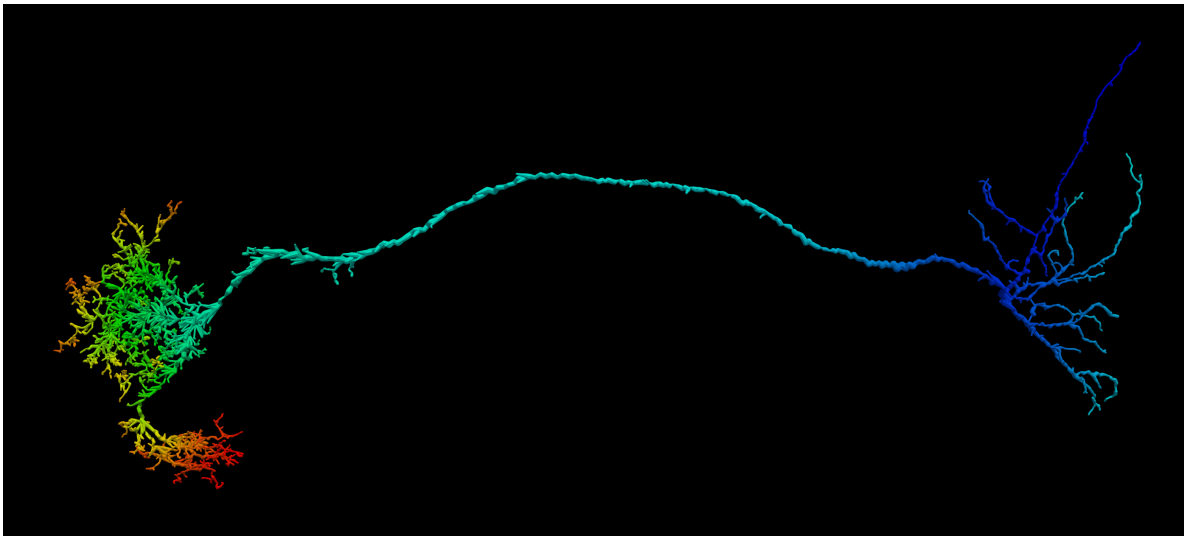


Fig.12.1　BN1056

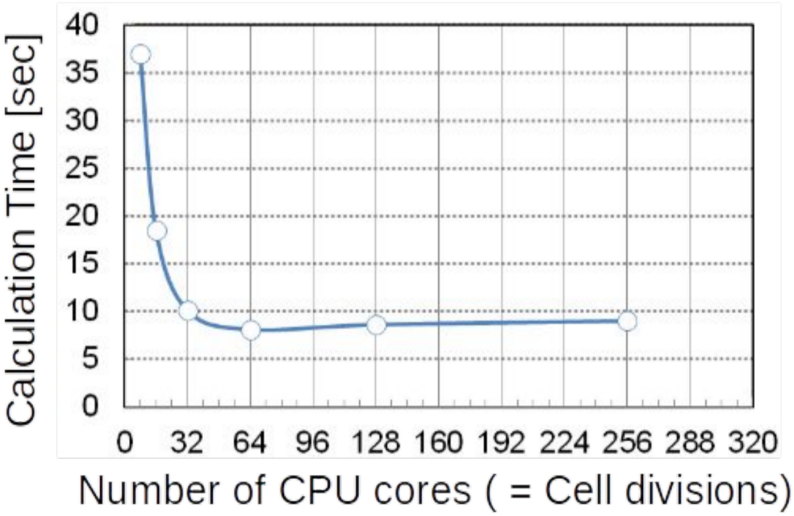


Fig.12.2　単一細胞の分割結果

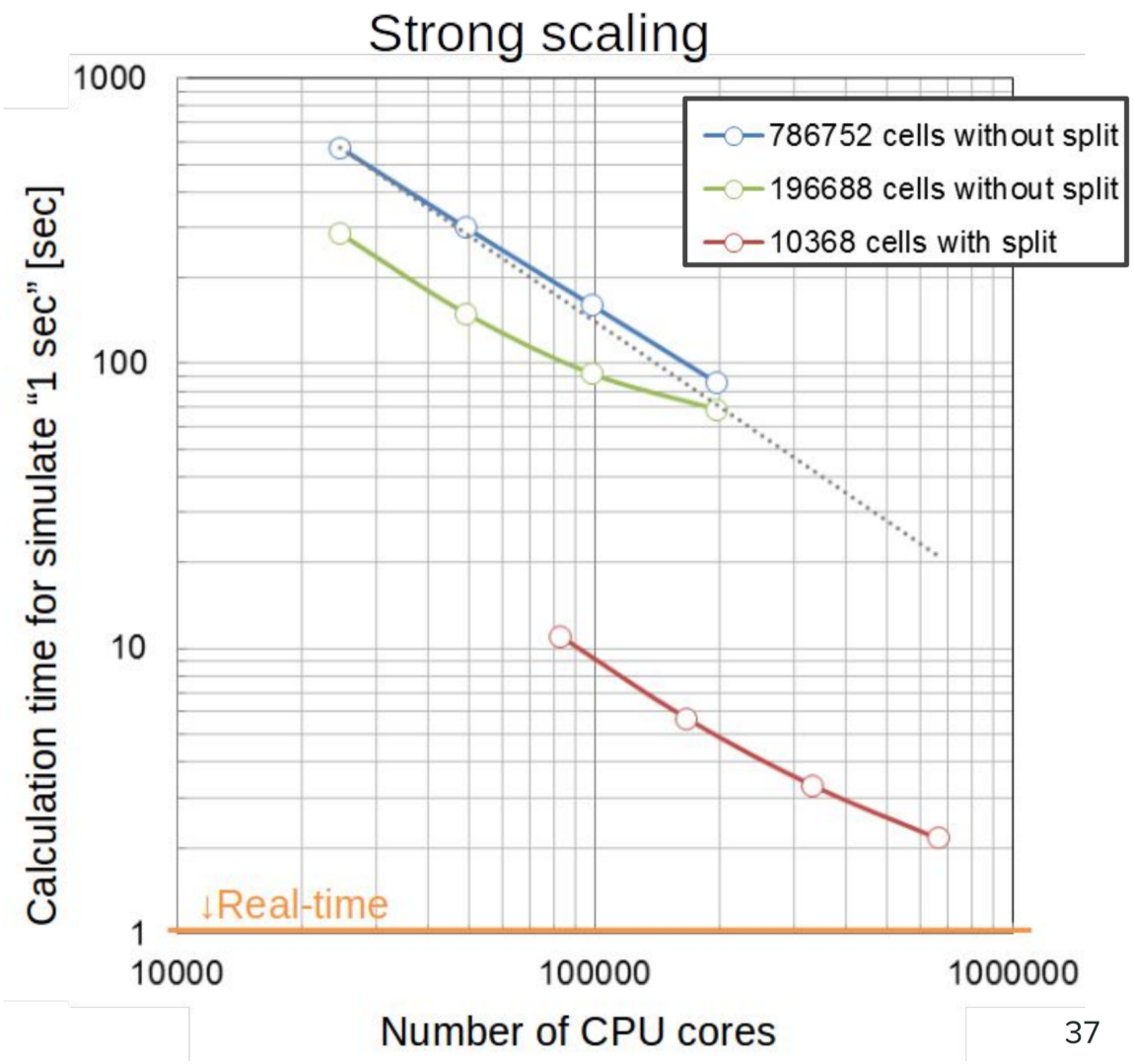


Fig.12.3　細胞分割による計算時間短縮

12.2　細胞形態縮約

12.2.1　縮約時の形態変化

リアルタイムでのシミュレーションを可能とするため，13.2 で述べた手法を用いて，細胞のコンパートメント数縮約を行った．

Fig. 12.1 について，手法 1, 手法 2 を繰り返し適用したところ，コンパートメント数は以下のようになった．

- 元形態: 3,890 コンパートメント
- 手法 1: 2,856 コンパートメント
- 手法 1+2: 859 コンパートメント
- 手法 1+2+1: 369 コンパートメント
- 手法 1+2+1+2: 186 コンパートメント

この時の形態変化を Fig. 12.2.2 に示す．

12.2.2　大規模シミュレーションの細胞形態縮約によるリアルタイムシミュレーションの実現

カイコガ均一神経細胞モデルを用いてストロングスケーリングを行った．結果を Fig. 12.7 に示す．この時，手法 1+2+1 を適用した 369 コンパートメントでの条件下において，10,368 細胞（382 万コンパートメント）について京コンピュータの 82,944 CPU コアを利用することで，リアルタイムシミュレーションを実現することができた．

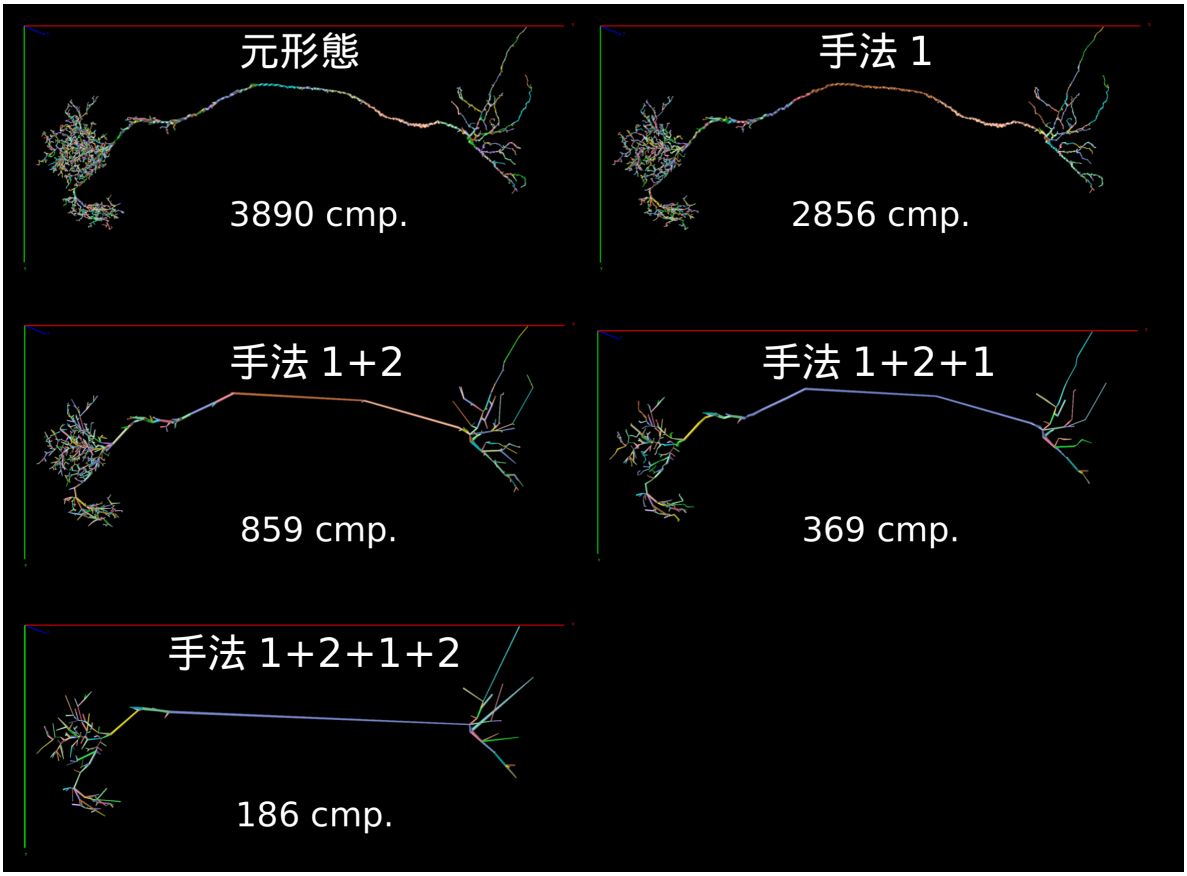


Fig.12.4　細胞縮約による形態変化

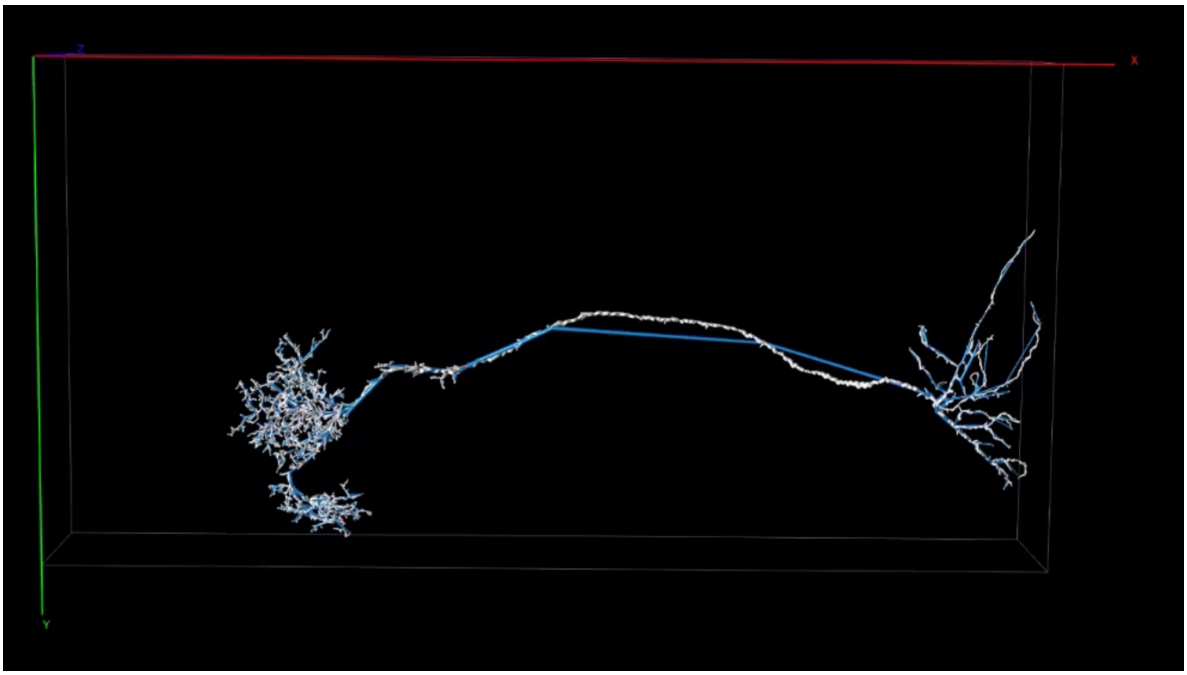


Fig.12.5　細胞分割による形態変化（元形態と手法 1+2 の比較）

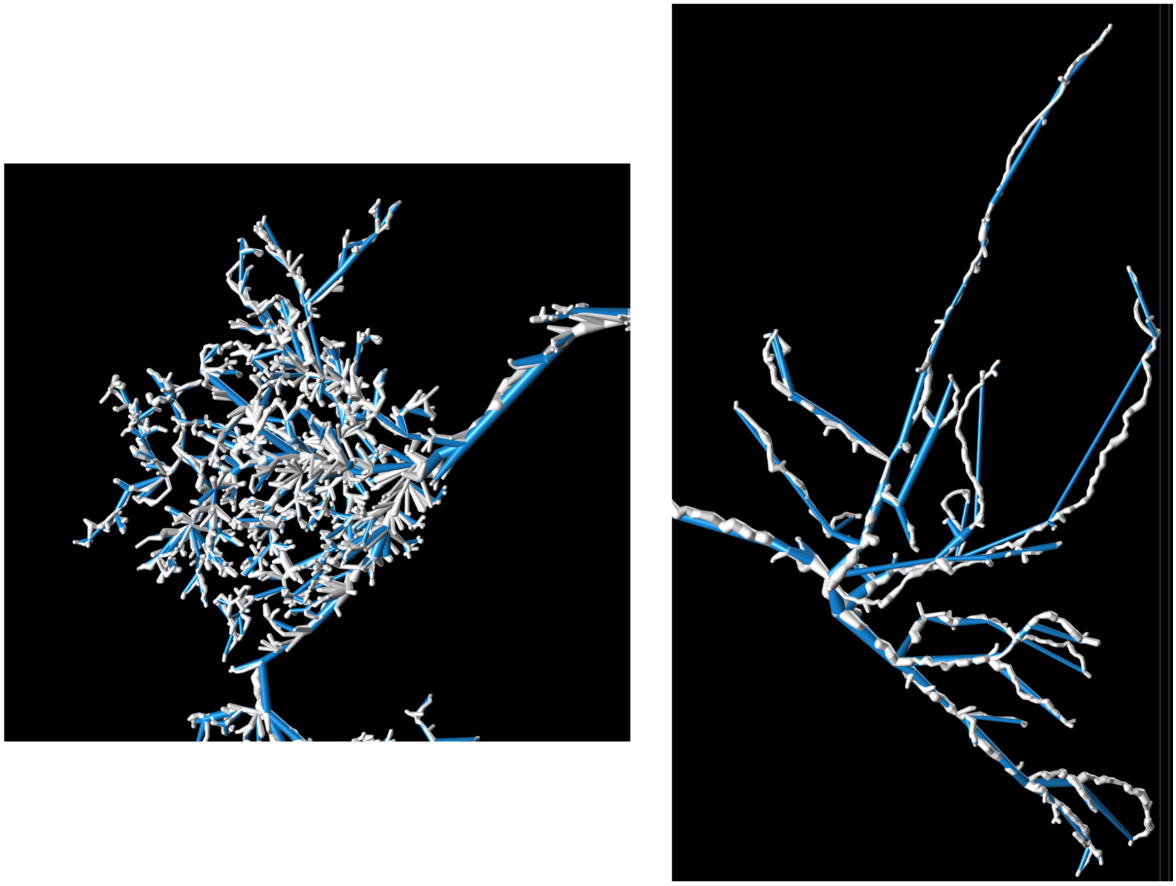


Fig.12.6　細胞分割による形態変化（元形態と手法 1+2 の比較，詳細）

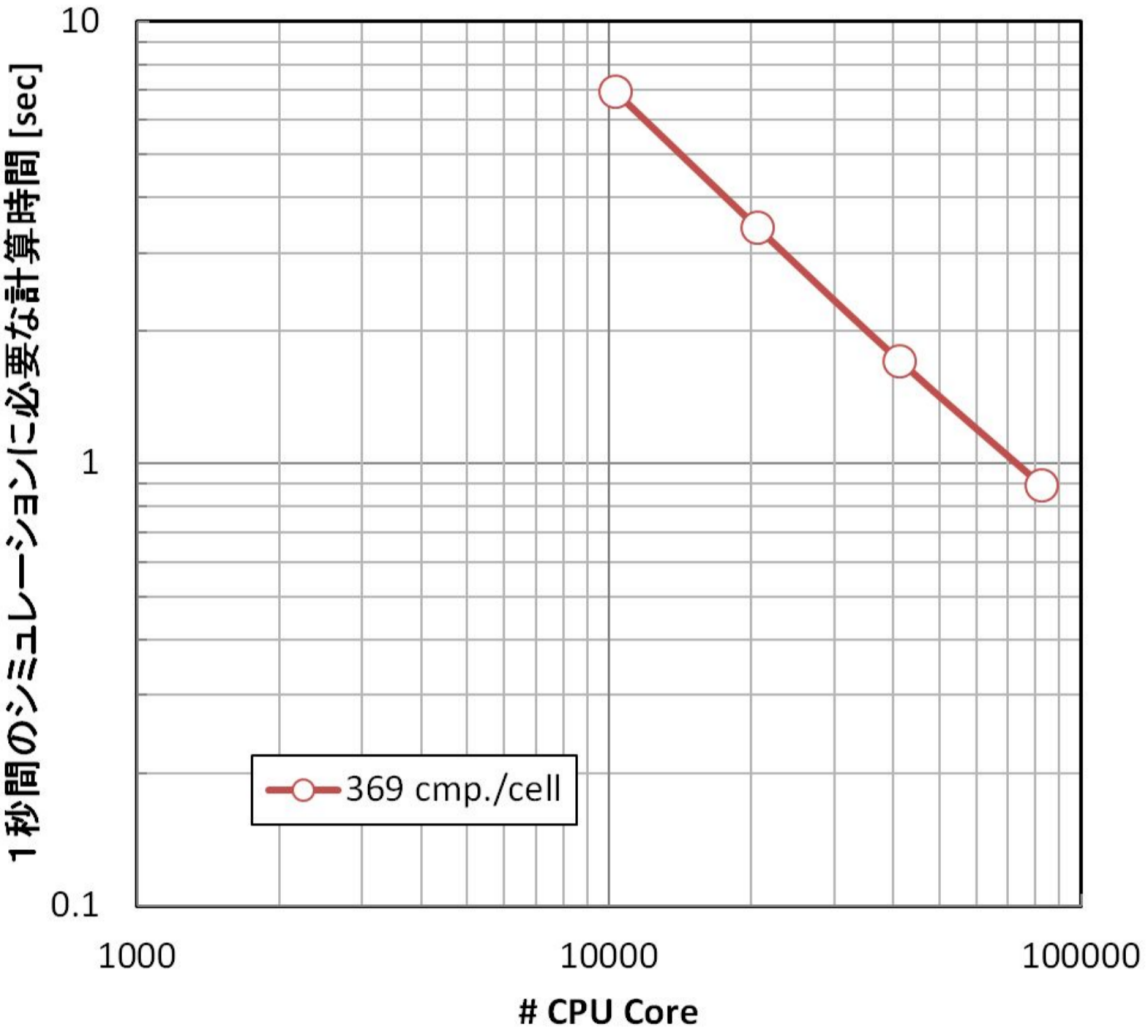


Fig.12.7　細胞形態縮約による計算時間短縮

第 13 章

不均一な神経細胞による大規模シミュレーション

不均一な細胞形態によって構成された神経回路の大規模シミュレーションを行うため，FlyCircuit[16] のデータベースに登録されているオス及びメスのショウジョウバエ大規模神経回路モデルについて，京コンピュータを用いて並列シミュレーションを行った．

13.1 ショウジョウバエ大規模モデルの解析

ショウジョウバエ大規模モデルの構成を把握するため，各神経細胞のコンパートメント数について，最大値，最小値，平均値を求めた．また，ヒストグラムを作成し，Fig. 13.1, 13.2 に示した．

- オスモデル (3,649 細胞)
  - 最大値: 3,911 コンパートメント
  - 最小値: 14 コンパートメント
  - 平均値: 513.8 コンパートメント
- メスモデル (12,393 細胞)
  - 最大値: 4,789 コンパートメント
  - 最小値: 16 コンパートメント
  - 平均値: 355.1 コンパートメント

その結果，いくつかの SWC ファイルについて，以下の示す不具合があったため，処理からは除外して扱うこととした．

- Cha-F-300116.swc: 空のデータが登録されている

- Gad1-F-200095.swc: 空のデータが登録されている
- Trh-M-400002.swc: SWC の根を表す-1 が複数存在している

13.2　ショウジョウバエ神経回路モデルの縮約

ショウジョウバエ大規模モデルを京コンピュータ上で並列計算を行ったところ，高並列時に性能が大幅に悪化することが測定された．これは，シナプス伝達の通信の際に同期を取るために，最も大きな細胞モデルの計算時間が律速となっているためと考えられる．

そこで，の手法を用いて，最大のコンパートメント数を持つものから順に縮約処理を行うことで，殆どの細胞において神経細胞の概形を保ったまま，大きな細胞においてのみ，コンパートメント数を縮約することができた．

この時のオス，メスそれぞれの神経回路における各縮約回数は以下の通りである．

- オスモデル (3,649 細胞)
  - 0 回: 2,973 細胞
  - 1 回: 273 細胞
  - 2 回: 322 細胞
  - 3 回: 81 細胞
  - 処理後最大値: 791 コンパートメント
  - 処理後平均値: 377.4 コンパートメント
- メスモデル (12,393 細胞)
  - 0 回: 11,581 細胞
  - 1 回: 350 細胞
  - 2 回: 388 細胞
  - 3 回: 74 細胞
  - 処理後最大値: 911 コンパートメント
  - 処理後平均値: 303.1 コンパートメント

また，その際のヒストグラムは，Fig 13.3, Fig 13.4 のようになり，コンパートメント数の大きい物が減っている事がわかる．

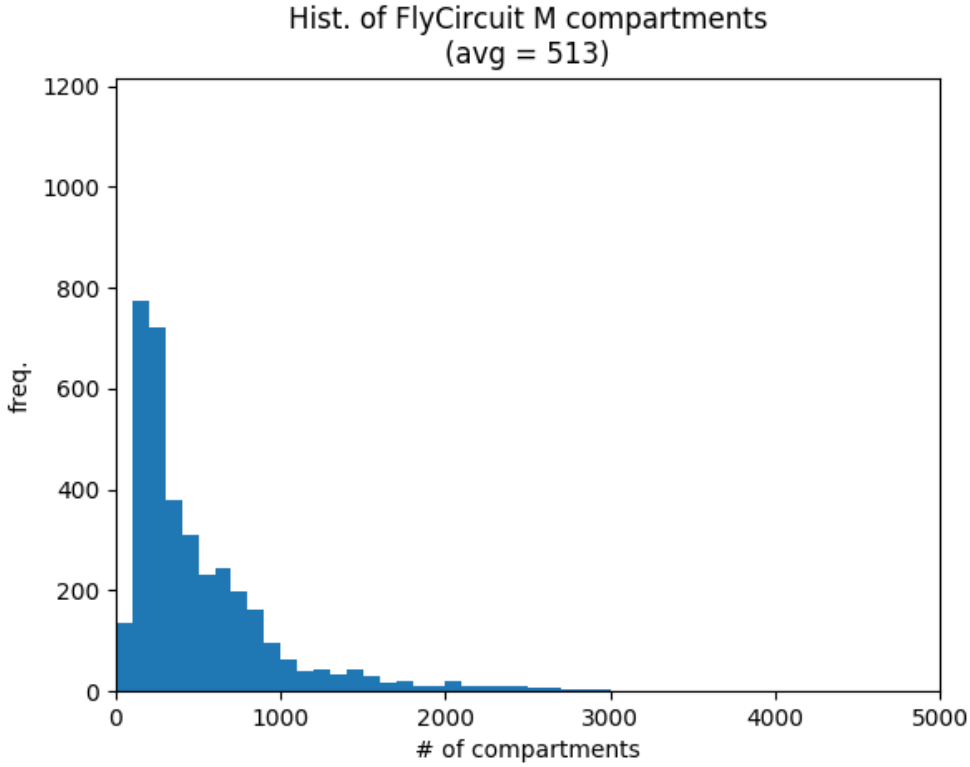


Fig.13.1　オスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム

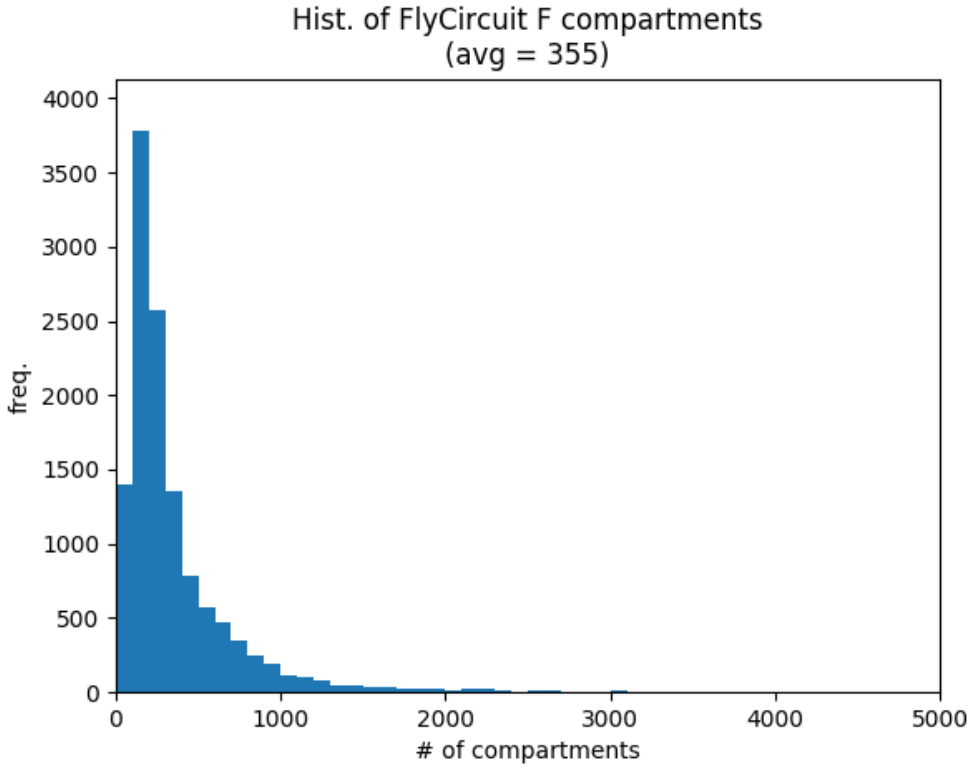


Fig.13.2　メスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム

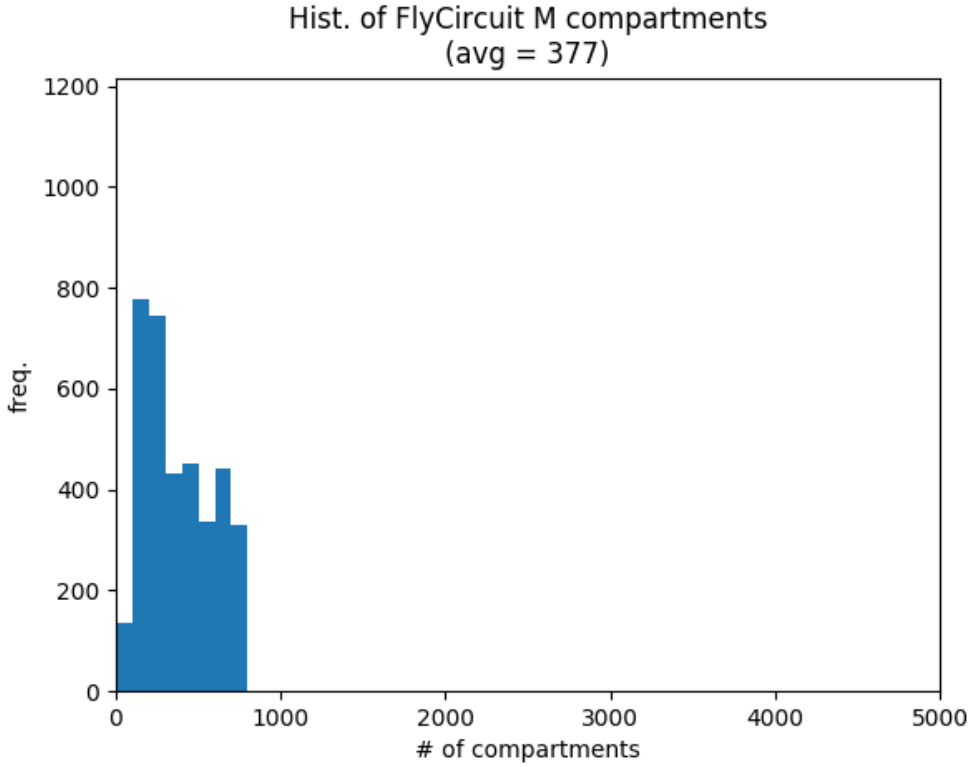


Fig.13.3　オスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム（縮約処理後）

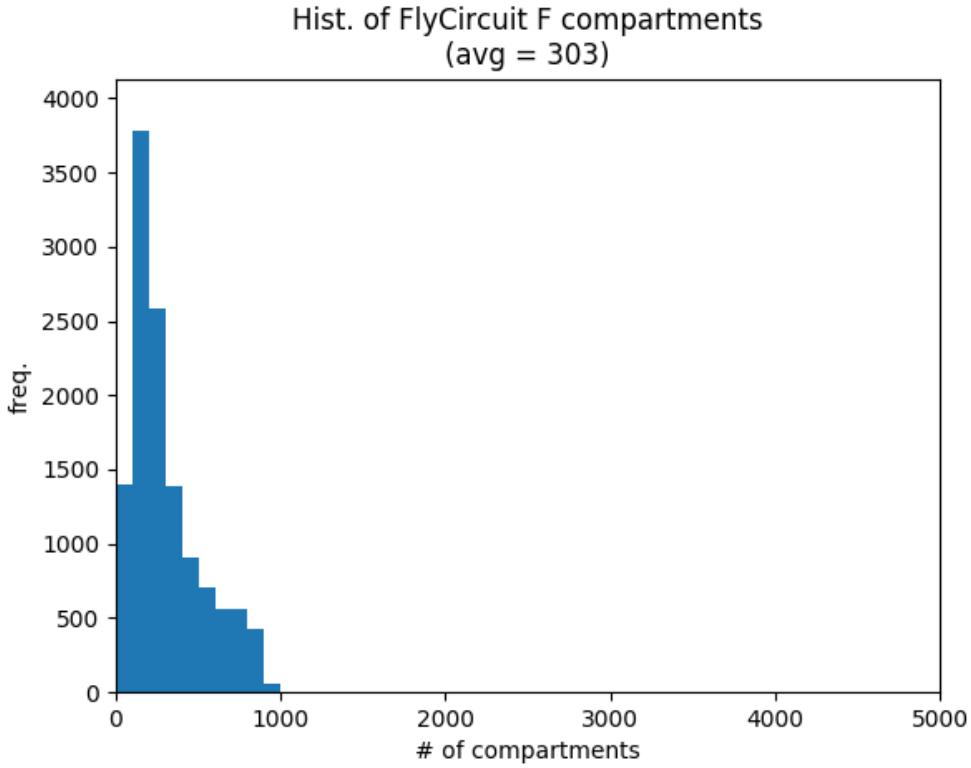


Fig.13.4　メスショウジョウバエ大規模モデルのコンパートメント数ヒストグラム（縮約処理後）

13.3　ショウジョウバエ神経回路モデルによるストロングスケーリング

ショウジョウバエ神経回路モデルについて，以下の 4 種類のパターンについてストロングスケーリングを行った (Fig. 13.8, Fig. 13.9).

- ショウジョウバエ大規模モデル (Variety ケース)
- ショウジョウバエ大規模モデルの神経細胞を，大規模モデルの平均コンパートメント数の神経細胞に置き換えたモデル (Average ケース)
- ショウジョウバエ大規模モデルの神経細胞を，大規模モデルの最大コンパートメント数の神経細胞に置き換えたモデル (Max ケース)
- ショウジョウバエ大規模モデルの神経細胞を，前述の手法で縮約を行ったモデル (Reduction ケース)

この時，ノード数の増大に伴い，ショウジョウバエ大規模モデル (Variety) は，Max ケースでのベンチマーク結果に近づくこととなった．また，縮約手法を用いることで，特に大規模並列時において，Average に近い演算速度を実現することができた．

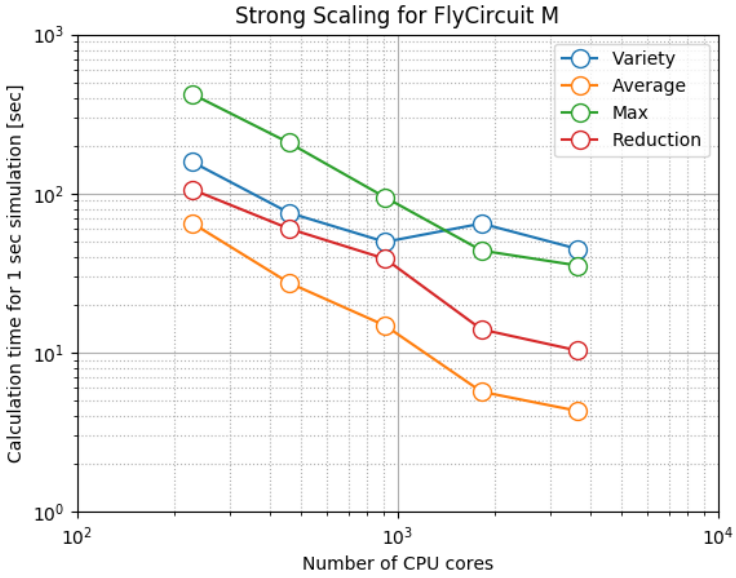


Fig.13.5　オスショウジョウバエ大規模モデルのストロングスケーリング

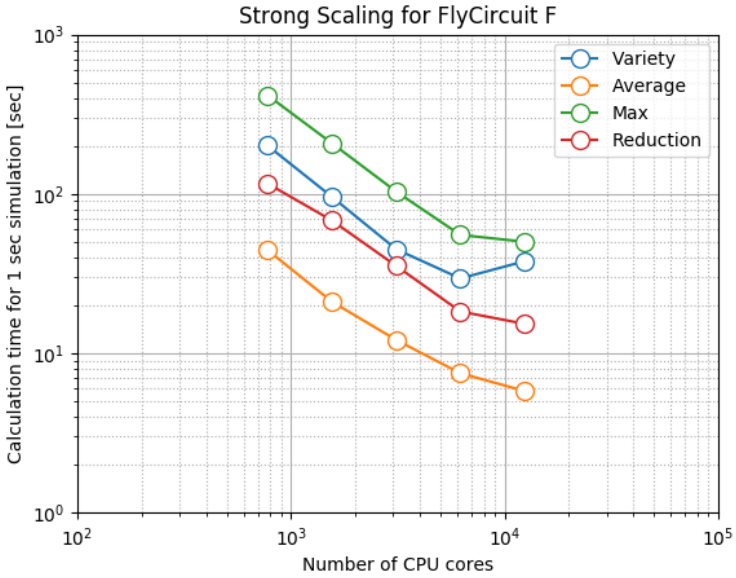


Fig.13.6　メスショウジョウバエ大規模モデルのストロングスケーリング

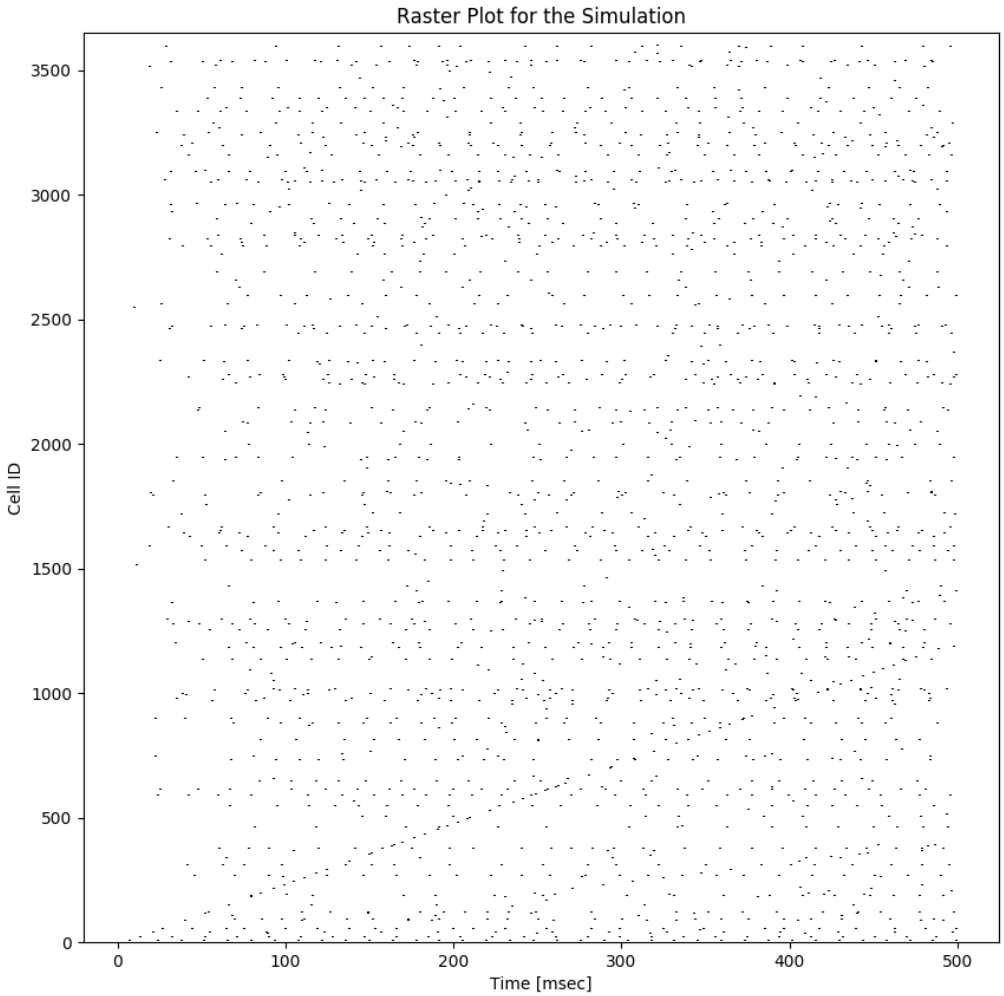


Fig.13.7　オスショウジョウバエ大規模モデルでの Raster Plot

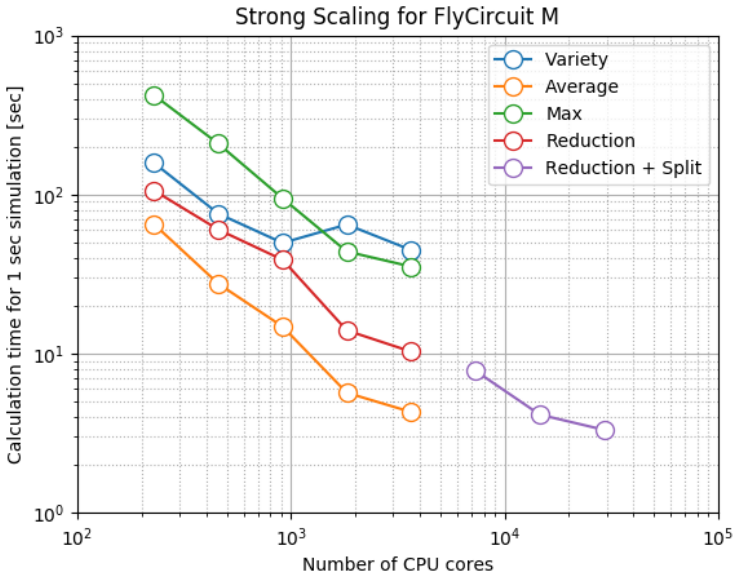


Fig.13.8　オスショウジョウバエ大規模モデルのストロングスケーリング（スレッドによる細胞分割）

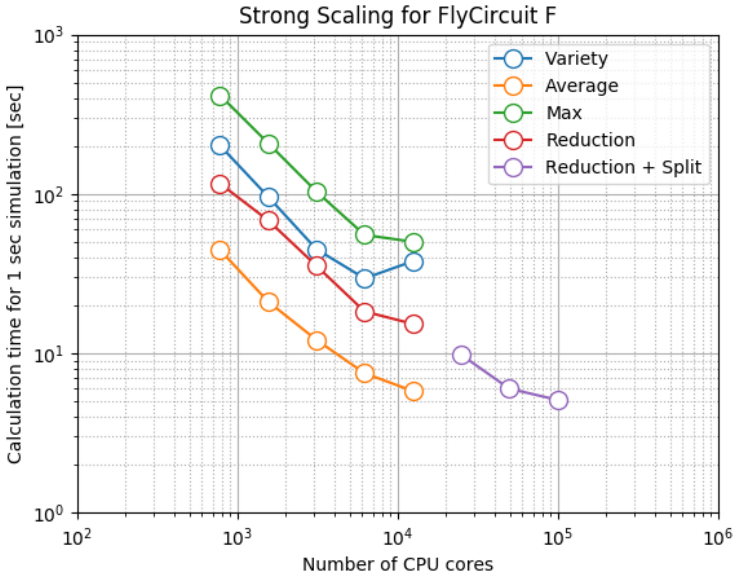


Fig.13.9　メスショウジョウバエ大規模モデルのストロングスケーリング（スレッドによる細胞分割）

第 IV 部

結論

第 14 章

考察

14.1 マルチコンパートメント Hodgkin-Huxley モデルの高速化

本研究では，NEURON シミュレータでのマルチコンパートメント Hodgkin-Huxley モデル計算におけるデータ構造や計算順序などについて検証，再構成を行い，最終的に，京コンピュータ上の SPARC64 VIIIfx において，元のシミュレーションに対し，6.4 倍高速化することができた．これは，メモリバンド幅律速と考えた場合のほぼ限界に近い値である．このような高速なシミュレーションは，全脳スケールの大規模シミュレーションや，進化計算を用いた神経細胞・回路パラメータ推定について，大幅な高速化を可能とし，これらの活用を大幅に進めるものである．

また，原理的にほぼ同じ計算であっても，NEURON の built-in 高速化手法も使用しなかった場合（これは，マルチコンパートメント Hodgkin-Huxley モデルの非常に素朴な実装となる）と，今回提案した高速化手法すべてを適用した場合とでは，LUT 使用による計算誤差はあるものの，同じマシンであっても最大で 200 倍近くの計算速度の差がある事がわかった．これは，数式だけでなく，ソースコード上の実装の重要性を示すものである．

14.2 大規模計算機上での複数階層での並列化

本研究では，マルチコンパートメント Hodgkin-Huxley モデルを高速化するため，SIMD 効率の上昇，OpenMP の導入と細胞分割計算，細胞形態縮約によるロードバランスの向上といった手法を適用してきた．これは，現在のスーパーコンピュータにおける，命令レベル並列性，CPU コアレベル並列性，ノードレベル並列性に対応しており，このようなマルチパラダイムの並列手法の活用は，現代のみならず次世代のスーパーコンピュータにおける計算性能の向上において，非常に重要になっていくと考えられる．

14.3　データベースに基づいたシミュレーション

本研究では，CNS-PF や FlyCircuit といったデータベース情報に基づいて，神経回路モデルの構築を行った．これらのデータベースは非常に有用だが，当然まだ課題も存在している．例えば，本研究の中で，これらのデータベースに入っている情報のうち，いくつか不具合があるものを発見したが，もしこれらの情報が修正されたとしても，どの程度自分のモデルを修正しなければいけないかが自明でないという問題がある．また情報が更新されていった場合に，どの状態のモデルに基づいてシミュレーションを行ったか検証が難しくなってしまう．このような問題の解決策として考えられるのが，OpenSourceBrain (`http://www.opensourcebrain.org/`) のような，バージョン管理システムとの統合である．このデータベースは，github をバックエンドとして利用しており，様々な人が自分のブランチを作成し，またそれが元のデータベースに差分としてアップデートされていく．このような仕組みを導入していくことで，よりデータベースの利用価値が上がっていくと考えられる．

また，これを更に発展させ，NeuroScienceGateway や，Simulation Platform[132] のようなクラウドコンピュータ環境で，すべてのシミュレーションを実行できる仕組みと連携することで，本研究のような高速化されたシミュレーションを，誰でもすぐに使える環境を作り出すことが可能になると考えられる．

14.4　次世代スーパーコンピュータに向けて

本研究ではマルチパラダイム並列手法として，命令レベル，CPU コアレベル，ノードレベルの階層について，高速化手法を提案し，大規模並列化を実現した．スーパーコンピュータにおける階層的並列性はしばらく続くと考えられ，大規模神経回路シミュレーションを行う上では，今後も対応を行っていく必要がある．その場合に課題となりうるのは，この階層の単位がマシンごとに固定されてしまっているという事である．例えば，京コンピュータでは，1 チップ辺りのスレッド数は 8 であり，今回はこの数で，細胞分割計算を行ったが，本来はこの値は細胞形態の複雑さに応じて，大きくも小さくもなりうる．そこで，ある程度の範囲で，この階層性のスケールを変更できるような仕組みがあれば，よりこのようなシミュレーションが行い易くなるのではないかと考えられる．

ただしその場合は，明示的に指定できる場合と比べ，実行効率は落ちる可能性が高く，トレードオフをいかに設定するかが非常に難しくなると考えられる．

第 V 部

業績・謝辞・参考文献

業績一覧

Fellowships and Awards

- JSPS Research Fellowship for Young Scientists (DC2), since April 2016.
- Excellent research project of the HPCI System for FY2014, The Second Project Report Meeting of the HPCI System including K computer, 2015.
- Poster Award, K computer Symposium 2012, 2012.

Full-papers (peer-reviewed)

- Takuma Iwamatsu, Daisuke Miyamoto, Hidefumi Mitsuno, Yoshiaki Yoshioka, Takeshi Fujii, Takeshi Sakurai, Yukio Ishikawa, and Ryohei Kanzaki, "Identification of repellent odorants to the body louse, Pediculus humanus corporis, in clove essential oil", Parasitology Research, 2016.
- Hidetoshi Ikeno, Tomoki Kazawa, Shigehiro Namiki, Daisuke Miyamoto, Yohei Sato, Stephan Shuichi Haupt, Ikuko Nishikawa, and Ryohei Kanzaki, "Development of a Scheme and Tools to Construct a Standard Moth Brain for Neural Network Simulations", Computational Intelligence and Neuroscience, vol. 2012, Article ID 795291, 10 pages, 2012.

Review papers (peer-reviewed)

- 宮本大輔, 加沢知毅, 神崎亮平, "昆虫嗅覚系全脳シミュレーションに向けて：スーパコンピュータによる大規模脳シミュレーションの現在とその展望", 人工知能学会誌, Vol. 30, No. 5, pp. 630-638, 2015 年 9 月.
- 加沢 知毅, 宮本 大輔, 後藤 晃彦, 朴 希原, 福田 哲也, 神崎 亮平, "昆虫全脳シミュレーションへむけて ―その構成技術と進展―", 日本神経回路学会誌, Vol.22,

No.3, pp. 89-102, 2015 年 9 月.

Invited Presentations

- Daisuke Miyamoto, “Database collaboration operated in the INCF J-node/NIJC hackathon”, 10th FENS Forum of Neuroscience, Copenhagen, Denmark, 4th July 2016, 2016.
- 宮本大輔, “京コンピュータを用いたマルチコンパートメント Hodgkin-Huxley 型モデルの高速化・高並列化”, 第 25 回 日本神経回路学会 全国大会 (JNNS 2015), 東京, 2015 年 9 月 2-4 日, 2015.
- 宮本大輔, ”リアルタイム神経回路シミュレーションに向けた計算高速化の必要性とその実際”, ESCAR, 2015 年 8 月 7-8 日, 2015.

International Conferences (peer-reviewed)

- Daisuke Miyamoto, Hidetoshi Ikeno, Yuko Okamura-Oho, Yoshihiro Okumura, Nobuyuki Munemura, Akira Sato, Yoko Yamaguchi, Ryohei Kanzaki, Teiichi Furuichi, “Database integration pipeline for highly reliable spatial gene expression patterns”, Front. Neuroinform. Conference Abstract: INCF Congress of Neuroinformatics 2016, 2016.
- Daisuke Miyamoto, Tomoki Kazawa, Stephan Shuichi Haupt, Masashi Tabuchi, Tomoaki Mori, Kei Nakatani and Ryohei Kanzaki, “Estimation method for biophysical properties of insect neurons in the combination of suitable stimulation and multi-compartment simulation with supercomputers”, Front. Neuroinform. Conference Abstract: 5th INCF Congress of Neuroinformatics., doi:10.3389/conf.fninf.2014.08.00011, 2014.
- Daisuke Miyamoto, Tomoki Kazawa, Ryohei Kanzaki, ”Neural Circuit Simulation of Hodgkin-Huxley Type Neurons Toward Peta Scale Computers”, SC12, Salt Lake City, Utah, November 10-16, 2012. (Selected as an oral presentation)
- Daisuke Miyamoto, Tomoki Kazawa, Stephan Shuichi Haupt, Hidehito Sato, Masashi Tabuchi, Kei Nakatani, Ryohei Kanzaki, “Automatic estimation of neural properties for Hodgkin-Huxley type models”, 5th International Symposium on Adaptive Motion of Animals and Machines, Hyogo, Japan, October 11-14, 2011.

- Heewon Park, Tomoki Kazawa, Daisuke Miyamoto, Akihiko Goto, Masashi Tabuchi, Ryohei Kanzaki, “Realistic neural circuit simulation of the moth antennal lobe that recognizes relative pheromonal concentration” , Front. Neurosci. Conference Abstract: Neuroinformatics 2015. doi: 10.3389/conf.fnins.2015.91.00044, 2015.
- Akira Takashima, Tomoki Kazawa, Shigehiro Namiki, Daisuke Miyamoto, Stephan Shuichi Haupt, Hidetoshi Ikeno, Ryohei Kanzaki and Shiro Usui, ” Interactive brain map in the Invertebrate Brain Platform (IVB-PF)” , Front. Neuroinform, Conference Abstract: 5th INCF Congress of Neuroinformatics., doi:10.3389/conf.fninf.2014.08.00082, 2014.

International Conferences

- Daisuke Miyamoto, Tomoki Kazawa, Hidetoshi Ikeno, Yoko Yamaguchi, Ryohei Kanzaki, The Invertebrate Brain Platform (IVB-PF) -updates of IVBPF in 2016-, Advances in Neuroinformatics IV. AINI2016 and INCF Nodes Workshop Abstract, doi:10.14931/aini2016.ps.15, 2016.
- Daisuke Miyamoto, Yuko Okamura-Oho, Teiichi Furuichi, Ryohei Kanzaki, Hidetoshi Ikeno, “An automatic registration method for ISH images into the 3D standard brain atlas” , Advances in Neuroinformatics 2015, Tokyo, Japan, Nov. 26-27, 2015.
- Daisuke Miyamoto, Tomoki Kazawa, Ryohei Kanzaki, “Analysis and Optimization of Multi-compartment Hodgkin-Huxley Type Model Simulations for Processing with SIMD Instructions” , Advances in Neuroinformatics 2015, Tokyo, Japan, Nov.26-27, 2015.
- Daisuke Miyamoto, Tomoki Kazawa, Akihiko Goto, Ryohei Kanzaki, “Acceleration of NEURON Simulator for Morphological Detailed Multi-Compartment Hodgkin-Huxley Type Simulation toward Real-time Speed” , Advances in Neuroinformatics 2014, Wako Japan, Sep. 25-26, 2014.
- Daisuke Miyamoto, Tomoki Kazawa, Akihiko Goto, Hidetoshi Ikeno, and Ryohei Kanzaki, “Constructing a massively parallelized morphological detailed neural circuit simulation of silkmoth brain with neuron database” , 2014 ICN/JSCPB, Sapporo Japan, July28-August 1st, 2014.
- Takuma Iwamatsu, Daisuke Miyamoto, Hidefumi Mitsuno, Yoshiaki Yoshioka,

Takeshi Sakurai, Ryohei Kanzaki, “Search for odorants inducing olfactory behavior on the body louse Pediculus humanus corporis based on the response of olfactory receptor” 8th Asia-Pacific Association of Chemical Ecologist (APACE) Conference, Sep. 23-26 , California, USA, 2015.

- Tomoki Kazawa, Daisuke Miyamoto, Akihiko Goto, Heewon Park, Hidetoshi Ikeno, Ikuko Nishikawa, Ryohei Kanzaki, “Constructing Multi-Compartment Parallelized Simulation from Olfactory Input to Premotor Command Generation of Silkmoth Brain”, Advances in Neuroinformatics 2014, Wako Japan, Sep.25-26, 2014.
- Akihiko Goto, Tomoki Kazawa, Daisuke Miyamoto, Masashi Tabuchi, Ryohei Kanzaki, “Examination of stimulus pattern and neuronal morphology for efficient biophysical property estimation of neurons in the silkmoth antennal lobe” , 2014 ICN/JSCPB, Sapporo, Japan, July 28 - August 1st, 2014.
- Tomoki Kazawa, Daisuke Miyamoto, Yohei Sato, Stephan Shuichi Haupt, Shigehiro Namiki, Akira Takashima, Hidetoshi Ikeno, Ikuko Nishikawa, Ryohei Kanzaki , “Towards the Whole Brain Simulation of the Insect Olfactory System” , 8th International Congress of Comparative Physiology and Biochemistry, Nagoya, Japan, May 31-June 5, 2011.

Japanese Conferences

- 宮本大輔, 加沢知毅, 神崎亮平, “OpenMP による細胞分割手法及び大規模並列計算機を用いた，詳細な形態を有する 10,000 神経細胞規模のリアルタイムシミュレーション”, 第 25 回 日本神経回路学会 全国大会 (JNNS 2015), 東京, 2015 年 9 月 2-4 日, 2015.
- Daisuke Miyamoto, Tomoki Kazawa, Stephan Shuichi Haupt, Shigehiro Namiki, Yohei Sato, Hidehito Sato, Yusuke Mori, Ryota Kobayashi, Hidetoshi Ikeno, Ikuko Nishikawa, Ryohei Kanzaki, ”IOSSIM: the framework for constructing multi-compartment neuron models and large scale simulations”, The 34th Annual Meeting of The Japanese Society for Comparative Physiology and Biochemistry 2012, Hayama, Japan, July 6-8, 2012.
- 宮本大輔, 加沢知毅, 神崎亮平, ”昆虫全脳リアルタイムシミュレーションに向けたペタフロップスコンピュータでの超並列計算の高速化”, 電子情報通信学会 2012 年総合大会, 岡山, 3 月 20〜23 日, 2012.

- 宮本大輔, 佐藤陽平, 加沢知毅, Haupt Stephan 周一, 並木重宏, 神崎亮平, 百田直也, 池野英利, 小林亮太, 西川郁子, “IOSSIM (Insect Olfactory System SIMulator) の開発”, 次世代生命体統合シミュレーションソフトウェアの研究開発（ライフ）公開シンポジウム, 神戸, 3 月 5〜6 日 2012.
- 宮本大輔，加沢知毅，Haupt Stephan 周一，田渕理史，中谷敬，神崎亮平，”マルチコンパートメントホジキンハクスレー型モデルを用いた昆虫神経細胞の膜情報自動推定”，バイオスーパーコンピューティングサマースクール 2011, 淡路島, 9 月 26〜27 日, 2011.
- Tomoki Kazawa, Daisuke Miyamoto, Akihiko Goto, Tetsuya Fukuda, Heewon Park and Ryohei Kanzaki, “Developing tools towards the olfactory system simulation of silkmoth”, CompBiol 2015 広島大会, Hiroshima, Japan, 2015.
- 福田哲也，加沢知毅，宮本大輔，神崎亮平, “マルチコンパートメント詳細モデルによる学習する神経回路の構築”, 第 25 回 日本神経回路学会 全国大会 (JNNS 2015), 東京, 2015 年 9 月 2-4 日, 2015.
- 岩松琢磨，宮本大輔，光野秀文，櫻井健志，神崎亮平, “匂い物質に対するコロモジラミの行動解析”, 第 59 回日本応用動物昆虫学会大会, 2015 年 3 月 26-28 日, 2015.
- 後藤昂彦, 加沢知毅, 宮本大輔, Haupt Stephan 周一, 森友亮, 神崎亮平, “大規模 RCGA による単一ニューロンに対する H-H 型モデルパラメータの推定”, 電子情報通信学会 NC 研究会, 1 月 20 日, 2014.
- 森友亮, 加沢知毅, 宮本大輔, 神崎亮平, “昆虫全脳シミュレーションに向けたスーパーコンピュータとのリアルタイム通信の検討”, 電子情報通信学会 NC 研究会, 9 月 24 日, 2013.
- 百田直矢，加沢知毅，Shuichi Haupt 周一，並木重宏，宮本大輔，神崎亮平，西川郁子，池野英利，”カイコガ標準脳データベース構築に向けた脳画像データの標準化と活用”，電子情報通信学会 NC 研究会 2011.

Others

- **Administrator and Co-developer:** Comparative Neuroscience Platform, `https://cns.neuroinf.jp` , since 2016.
- **Administrator and Co-developer:** Invertebrate Brain Platform, `https://invbrain.neuroinf.jp` , since 2012.
- **Demonstration:** Invertebrate neuron database and neural circuit simulation

based on the database, the 79th National Convention of Information Processing Society of Japan (IPSJ), https://www.neuroinf.jp/modules/news/index.php?page=article&storyid=269&ml_lang=en , 2017.

- **Design Co-operation:** AGE OF SUPER SENSING センシングデザインの未来, https://www.amazon.co.jp/dp/4822239713/ , 2017.
- **Organize Committee:** BrainHack Global 2017 Japan, https://www.neuroinf.jp/brainhack-global-2017/ , 2017.
- **Organize Committee:** Congress of Japanese Society for Comparative Physiology and Biochemistry 2016, Tokyo, Japan, https://cns.neuroinf.jp/jscpb/2016/ , 2016.
- **Travel Award:** International HPC Summer School 2016, Ljublijana, Slovenia, http://ihpcss2016.hpc.fs.uni-lj.si/ , 2016.
- **Travel Award:** 3rd Human Brain Project Education Workshop, Manchester, UK, 2016.
- **Travel Award:** The first BigNeuron algorithm porting hackathon kicked off at Beijing, China, 2015.
- **Organize Committee and Lecture:** Brain Atlas Ideathon/Hackathon 2015, Wako, Japan, https://www.neuroinf.jp/bah2015/ideathon.html , 2015.
- **Participation Report:** “日本神経回路学会「脳と心のメカニズム」第 12 回冬のワークショップ参加助成報告”, 日本神経回路学会誌, Vol. 19, No. 1, 2012.
- **Qualification:** AWS Certified Solution Architect - Associate, Aug 2017.
- **Qualification:** AWS Certified Developer - Associate, Aug 2017.

謝辞

本研究は，東京大学大学院工学系研究科先端学際工学専攻の神崎亮平教授のご指導のもと行われました．

東京大学先端科学技術センター 神崎亮平 教授には，研究指導だけでなく，生物，特に昆虫の賢さ・おもしろさについて多くの事を教えていただきました．また，常に前向きな姿勢と，周囲を奮い立たせる言葉には何度も助けられました．

東京大学先端科学技術センター 加沢知毅 特任研究員には大学院入学当初から最後に至るまで，直接の指導者として多くの助言をいただきました．議論においては必ずしも一致した見解に至るとは限りませんでしたが，多彩な知識を元にした洞察の数々には幾度も何度も目を見開かされる思いでした．

岩松琢磨 博士には研究のみならず，人生の様々な事象について議論をさせていただきました．あの日々がなければ，この論文は間違いなく完成しなかったと確信しております．また，研究遂行に必要な計算環境についてご支援頂いたことは今でも本当に感謝しております．困窮している時にご支援いただけるというのは希少かつ貴重であり，自分もそのようなことができるようになりたいと強く感じます．

佐藤陽平 氏, 佐藤秀仁 氏, 森友亮 氏, 李夏栄 氏, 後藤昂彦 氏, 朴希原 氏, 福田哲也 氏, 山崎寛明 氏, 米山兼治 氏, 韮沢拓也 氏，荒瀬晃介 氏, 角田颯飛 氏, 井上裕太 氏とは研究室の同チームのメンバーとして，ともに多くの議論をさせていただきました．昆虫全脳シミュレーションという目標に向けて協力しあえた事を誇りに思います．特に後藤昂彦 氏, 福田哲也 氏, 井上裕太 氏には，私が取り組んでいたテーマの一部を引き継いでいただき，自分では達し得ないような素晴らしい成果を残していただきました．とても感謝しております．

並木重宏 博士, Stephan Haupt 周一 博士, 藤原輝史 博士, 田渕理史 博士には多くの実験的結果やそれを元にした素晴らしい知見をご教授いただきました．本研究は実験的成果を基盤として初めて成立するものであり，特にデータベースの構築のような地道かつ重要な作業がなければ，モデルの構築には何倍もの年月を要したものと思います．また，実際に大規模シミュレーションとして成立させるために，多くの取捨選択を行ったことについ

てはお叱りを受ける点が多々あるかと思いますが，どうか今後の課題としてご容赦いただけましたら幸いです．

兵庫県立大学 池野英利 教授には，他大学にも関わらず，入学当初から神経細胞の情報の取り扱いについて多くのご指導をいただきました．また，落ち着いた議論の進め方や，研究を進める上での考え方について，様々な事を学ばせていただきました．

理化学研究所神経情報基盤センター 奥村嘉宏 主任, 山口陽子 センター長，臼井支朗 前センター長には，神経情報基盤センターと協力して進めているデータベース構築のみならず，ハッカソンや国際ワークショップなど様々な場で大変お世話になりました．特に奥村主任にはウェブ技術やネットワーク管理について様々な事を教えていただき，また悩んだ時にはいつでも気軽に相談にのっていただき，本当に感謝しております．

理化学研究所情報基盤センター 五十嵐潤 博士には，ISLiM プロジェクトの時期から現在に至るまで，大規模脳・神経系シミュレーションの第一人者として，多くのご指導をいただきました．対象とする神経回路モデルは異なるものの，抱えている課題としては共通する事も多く，特に ISLiM プロジェクト時期の AICS でのディスカッションでは，まだこの分野を学び始めたばかりの自分に，基本となる考え方をご教授くださり，本当にありがとうございました．

電気通信大学 山崎匡 准教授には，大規模脳・神経系シミュレーションの第一人者として，多くのアドバイスをいただき，また，脳・神経系シミュレーションがどうあらねばならないかということについて，様々なことを学ばせていただきました．また，大きなビジョンとそれを裏打ちするプログラミング力，文章記述力等から研究者のあるべき姿に触れさせていただき，強い刺激となりました．

実践女子大学 於保祐子 教授には，NIJC 主催のハッカソンなどを通じて大変お世話になりました．また脳に関する生物学的な知見だけでなく，論文の書き方についても多くの事を学ばせていただきました．

理化学研究所生命システム研究センター 大野洋介 博士，理化学研究所情報基盤センター 舛本現 博士，理化学研究所生命システム研究センター 泰地 真弘人 プロジェクトリーダーには，ISLiM プロジェクト高度化チームとして，京コンピュータの特性に関する非常に深い理解から来る多くのアドバイスをいただきました．それだけでなく，プログラムを高速化するという事について，多くの知見をいただきました．

理化学研究所情報基盤センター 姫野龍太郎 センター長には，ISLiM プロジェクトから始まり，多くのプロジェクトやバイオスーパーコンピューティング研究会などで大変お世話になりました．本研究はそれらのプロジェクトと深く関わっており，今回まとめることができましたのも，継続的なご支援があってこそだと強く感じております．

理化学研究所情報基盤センター 戎崎俊一 主任研究員には，高校時代のまだ右も左もわからない時期に，シミュレーションや可視化の面白さやを教えていただきました．また，

改めてアクセラレーション研究会などでお世話になり，PEZY-SC のような非常に興味深いアーキテクチャに触れる機会を頂いただけでなく，常に攻めの姿勢から多くの事を学ばせていただきました．

長崎大学 濱田剛 准教授には，FPGA や GPGPU といった特殊な計算資源による高度計算の面白さを教えていただきました．また，DEGIMA での HPL 測定について，微力ながらお手伝いさせていただいた経験は，大規模計算機の運用について多くの事を考える素晴らしい機会となりました．

高橋宏知 講師, 安藤規泰 特任講師, 櫻井健志 特任講師, 光野秀文 助教, 峯岸諒 博士，三觜裕之 氏, 渡辺慶 氏, 狩野竜示 氏, 徳江和典 氏，矢田祐一郎 博士, 塩田裕介 氏, 照月大悟 氏, 秋山義太郎 氏, 石津光太郎 氏, 和家尚希 氏, 村山裕哉 氏には，同じ研究室の指導者，同期，後輩として，多くのご指導を賜りました．個性豊かな方が多く，自分の想像もしていなかったような考え方に触れることが多く，非常に刺激を受けました．

木村立代 さん，岩月知香 さんには研究室を支える立場として，普段は優しく，時に厳しく多くのご支援をいただきました．切羽詰っている時には，多くのご心配をお掛けしたことと思います．

最後になりますが，妻である宮本由佳と両親には非常に心配をかけました．いつもあまり詳しく説明せず，また自分で勝手に決めてしまう性格のため，戸惑いもあったかと思いますが，常に支えてくださり，ありがとうございました．

本研究を進めるにあたっては，この他にも大変多くの方々のご支援をいただきました．全ての方のお名前を挙げることはできませんでしたが，本当に感謝しております．ありがとうございました．

データ利用

本研究を行うにあたっては，以下のデータベースを利用させていただきました．

- 東京大学先端科学技術研究センター神崎・高橋研究室: Bombyx Neuron Database [71]
- the NCHC (National Center for High-performance Computing) and NTHU (National Tsing Hua University): FlyCircuit [16]

計算機利用

本研究を行うにあたっては，以下の計算環境を利用させていただきました．

- 理化学研究所計算科学研究機構 京コンピュータ (hp120263, hp140151, hp150074, hp160269)
- 東京大学 FX10 Oakleaf-FX (hp150074)
- 九州大学 FX10 (hp140151)
- 名古屋大学 FX100 (hp160269)
- 高エネルギー加速器研究機構石川研究室 Suiren

研究助成

本研究を行うにあたっては，以下のプロジェクトより助成をいただきました．

- 日本学術振興会 特別研究員 (DC2, 2016 年度 - 2017 年度)
- ISLiM 次世代生命体統合シミュレーションソフトウェア (2008 年度 - 2012 年度)
- HPCI 一般利用 (2012 年度 - 2016 年度: hp120263, hp140151, hp150074)
- 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への応用 (2016 年度 - 2017 年度)

参考文献

[1] Sudhir Ahuja, Nicholas Carriero, and David Gelernter. Linda and friends. *Computer*, Vol. 19, No. 8, pp. 26–34, August 1986.

[2] Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. Tofu: A 6d mesh/-torus interconnect for exascale computers. *Computer*, pp. 36–41, 2009.

[3] A Paul Alivisatos, Miyoung Chun, George M Church, Ralph J Greenspan, Michael L Roukes, and Rafael Yuste. The brain activity map project and the challenge of functional connectomics. *Neuron*, Vol. 74, No. 6, pp. 970–4, jun 2012.

[4] Mara Almog and Alon Korngreen. Is realistic neuronal modeling realistic? *Journal of Neurophysiology*, Vol. 116, No. 5, 2016.

[5] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pp. 483–485, New York, NY, USA, 1967.

[6] Haroon Anwar, Christopher J Roome, Hermina Nedelescu, Weiliang Chen, Bernd Kuhn, and Erik De Schutter. Dendritic diameters affect the spatial variability of intracellular calcium dynamics in computer models. *Frontiers in cellular neuroscience*, Vol. 8, p. 168, 2014.

[7] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. NeuroMorpho.Org: a central resource for neuronal morphologies. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, Vol. 27, No. 35, pp. 9247–51, aug 2007.

[8] Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware, Inc., USA, 2015.

[9] J. P. Bacon and R. K. Murphey. Receptive fields of cricket giant interneurones are related to their dendritic structure. *J. Physiol. (Lond.)*, Vol. 352, pp.

601–623, Jul 1984.

[10] Roy Ben-Shalom, Amit Aviv, Benjamin Razon, and Alon Korngreen. Optimizing ion channel models using a parallel genetic algorithm on graphical processors. *Journal of Neuroscience Methods*, Vol. 206, No. 2, pp. 183–194, may 2012.

[11] Roy Ben-Shalom, Gilad Liberman, and Alon Korngreen. Accelerating compartmental modeling on a graphical processing unit. *Frontiers in neuroinformatics*, Vol. 7, p. 4, 2013.

[12] Christoph Börgers and Alexander R Nectow. EXPONENTIAL TIME DIFFERENCING FOR HODGKIN-HUXLEY-LIKE ODES. *SIAM journal on scientific computing : a publication of the Society for Industrial and Applied Mathematics*, Vol. 35, No. 3, pp. B623–B643, 2013.

[13] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M. Bower, Markus Diesmann, Abigail Morrison, Philip H. Goodman, Frederick C. Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P. Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, Vol. 23, No. 3, pp. 349–398, dec 2007.

[14] N Brunel and N Brunel. Dynamics of sparsely connected networls of excitatory and inhibitory neurons. *Computational Neuroscience*, Vol. 8, pp. 183–208, 2000.

[15] Leland Chang, David J. Frank, Robert K. Montoye, Steven J. Koester, Brian L. Ji, Paul W. Coteus, Robert H. Dennard, and Wilfried Haensch. Practical strategies for power-efficient computing technologies. *Proceedings of the IEEE*, Vol. 98, No. 2, pp. 215–236, 2010.

[16] Ann-Shyn Chiang, Chih-Yung Lin, Chao-Chun Chuang, Hsiu-Ming Chang, Chang-Huain Hsieh, Chang-Wei Yeh, Chi-Tin Shih, Jian-Jheng Wu, Guo-Tzau Wang, Yung-Chang Chen, Cheng-Chi Wu, Guan-Yu Chen, Yu-Tai Ching, Ping-Chang Lee, Chih-Yang Lin, Hui-Hao Lin, Chia-Chou Wu, Hao-Wei Hsu, Yun-Ann Huang, Jing-Yi Chen, Hsin-Jung Chiang, Chun-Fang Lu, Ru-Fen Ni, Chao-Yuan Yeh, and Jenn-Kang Hwang. Three-Dimensional Reconstruction of Brain-wide Wiring Networks in Drosophila at Single-Cell Resolution. *Current Biology*, Vol. 21, No. 1, pp. 1–11, jan 2011.

[17] Matteo Colombo. Why build a virtual brain? Large-scale neural simulations

as jump start for cognitive computing. *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 3079, No. 2, pp. 1–10, 2016.

[18] Matteo Colombo. Why build a virtual brain? Large-scale neural simulations as jump start for cognitive computing. *Journal of Experimental & Theoretical Artificial Intelligence*, Vol. 3079, No. 2, pp. 1–10, 2016.

[19] C. M. Comer and R. M. Robertson. Identified nerve cells and insect behavior. *Prog. Neurobiol.*, Vol. 63, No. 4, pp. 409–439, Mar 2001.

[20] Hermann Cuntz, Friedrich Forstner, Juergen Haag, and Alexander Borst. The morphological identity of insect dendrites. *PLoS Computational Biology*, Vol. 4, No. 12, 2008.

[21] Egidio DAngelo, Giovanni Danese, Giordana Florimbi, Francesco Leporati, Alessandra Majani, Stefano Masoli, Sergio Solinas, and Emanuele Torti. The Human Brain Project: High Performance Computing for Brain Cells Hw/Sw Simulation and Understanding. *2015 Euromicro Conference on Digital System Design*, pp. 740–747, 2015.

[22] Edoardo Datteri and Federico Laudisa. Large-scale simulations of brain mechanisms: beyond the synthetic method. *Paradigmi*, No. 3, pp. 23–46, 2016.

[23] E De Schutter and J M Bower. An active membrane model of the cerebellar Purkinje cell. I. Simulation of current clamps in slice. *Journal of neurophysiology*, Vol. 71, No. 1, pp. 375–400, 1994.

[24] E De Schutter and J M Bower. An active membrane model of the cerebellar Purkinje cell II. Simulation of synaptic responses. *Journal of neurophysiology*, Vol. 71, No. 1, pp. 401–19, jan 1994.

[25] Erik De Schutter. Why Are Computational Neuroscience and Systems Biology So Separate? *PLoS Computational Biology*, Vol. 4, No. 5, p. e1000078, may 2008.

[26] M. Djurfeldt, J. Hjorth, J. M. Eppler, N. Dudani, M. Helias, T. C. Potjans, U. S. Bhalla, M. Diesmann, J. H. Kotaleski, and O. Ekeberg. Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics*, Vol. 8, No. 1, pp. 43–60, Mar 2010.

[27] GNU Ed. `http://www.gnu.org/software/ed/`.

[28] Robert Egger, Vincent J Dercksen, Daniel Udvary, Hans-Christian Hege, and Marcel Oberlaender. Generation of dense statistical connectomes from sparse morphological data. *Frontiers in neuroanatomy*, Vol. 8, No. November, 2014.

[29] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, C. Tang,

and D. Rasmussen. A large-scale model of the functioning brain. *Science*, Vol. 338, No. 6111, pp. 1202–1205, Nov 2012.

[30] Micro EMACS. `http://www.aquest.com/emacs.htm`.

[31] Andreas K. Fidjeland, David Gamez, Murray P. Shanahan, and Edgars Lazdins. Three tools for the real-time simulation of embodied spiking neural networks using GPUs. *Neuroinformatics*, Vol. 11, No. 3, pp. 267–290, 2013.

[32] SWC File Format. `http://research.mssm.edu/cnic/swc.html`.

[33] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The SpiNNaker Project. *Proceedings of the IEEE*, Vol. 102, No. 5, pp. 652–665, may 2014.

[34] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, Vol. 2, No. 4, p. 1430, 2007.

[35] P. Gleeson, S. Crook, R. C. Cannon, M. L. Hines, G. O. Billings, M. Farinella, T. M. Morse, A. P. Davison, S. Ray, U. S. Bhalla, S. R. Barnes, Y. D. Dimitrova, and R. A. Silver. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput. Biol.*, Vol. 6, No. 6, p. e1000815, Jun 2010.

[36] Sten Grillner, Nancy Ip, Christof Koch, Walter Koroshetz, Hideyuki Okano, Miri Polachek, Mu-ming Poo, and Terrence J Sejnowski. Worldwide initiatives to advance brain research. *Nature Neuroscience*, Vol. 19, No. 9, pp. 1118–1122, 2016.

[37] M. Hammer. An identified neuron mediates the unconditioned stimulus in associative olfactory learning in honeybees. *Nature*, Vol. 366, pp. 59–63, Nov 1993.

[38] Park Heewon, Kazawa Tomoki, Miyamoto Daisuke, Goto Akihiko, Tabuchi Masashi, and Kanzaki Ryohei. Realistic neural circuit simulation of the moth antennal lobe that recognizes relative pheromonal concentration. *Frontiers in Neuroscience*, Vol. 9, , 2015.

[39] Moritz Helias, Susanne Kunkel, Gen Masumoto, Jun Igarashi, Jochen Eppler, Shin Ishii, Tomoki Fukai, Abigail Morrison, and Markus Diesmann. Supercomputers ready for use as discovery machines for neuroscience. *Frontiers in Neuroinformatics*, Vol. 6, p. 26, 2012.

[40] I. Hepburn, W. Chen, and E. De Schutter. Accurate reaction-diffusion operator splitting on tetrahedral meshes for parallel stochastic molecular simulations. *Journal of Chemical Physics*, Vol. 145, No. 5, 2016.

[41] M. L. Hines and N. T. Carnevale. The NEURON simulation environment. *Neural Comput*, Vol. 9, No. 6, pp. 1179–1209, Aug 1997.

[42] M L Hines and N T Carnevale. The NEURON simulation environment. *Neural computation*, Vol. 9, No. 6, pp. 1179–209, aug 1997.

[43] M. L. Hines and N. T. Carnevale. Expanding NEURON's repertoire of mechanisms with NMODL. *Neural Comput*, Vol. 12, No. 5, pp. 995–1007, May 2000.

[44] M L Hines and N T Carnevale. Expanding NEURON's repertoire of mechanisms with NMODL. *Neural computation*, Vol. 12, No. 5, pp. 995–1007, may 2000.

[45] M. L. Hines and N. T. Carnevale. Translating network models to parallel hardware in NEURON. *J. Neurosci. Methods*, Vol. 169, No. 2, pp. 425–455, Apr 2008.

[46] M. L. Hines, A. P. Davison, and E. Muller. NEURON and Python. *Front Neuroinform*, Vol. 3, p. 1, 2009.

[47] M. L. Hines, H. Eichner, and F. Schurmann. Neuron splitting in compute-bound parallel network simulations enables runtime scaling with twice as many processors. *J Comput Neurosci*, Vol. 25, No. 1, pp. 203–210, Aug 2008.

[48] M. L. Hines, H. Markram, and F. Schurmann. Fully implicit parallel simulation of single neurons. *J Comput Neurosci*, Vol. 25, No. 3, pp. 439–448, Dec 2008.

[49] M. Hines, S. Kumar, and F. Schurmann. Comparison of neuronal spike exchange methods on a Blue Gene/P supercomputer. *Front Comput Neurosci*, Vol. 5, p. 49, 2011.

[50] Michael Hines. Efficient computation of branched nerve equations. *International Journal of Bio-Medical Computing*, Vol. 15, No. 1, pp. 69–76, jan 1984.

[51] Michael Hines. A PROGRAM FOR SIMULATION OF NERVE EQUATIONS WITH BRANCHING GEOMETRIES. Vol. 24, pp. 55–68, 1989.

[52] Michael L Hines, Henry Markram, and Felix Schürmann. Fully implicit parallel simulation of single neurons. *Journal of computational neuroscience*, Vol. 25, No. 3, pp. 439–48, dec 2008.

[53] M.L. Hines and N.T. Carnevale. Translating network models to parallel hardware in NEURON. *Journal of Neuroscience Methods*, Vol. 169, No. 2, pp. 425–455, apr 2008.

[54] A. L. HODGKIN and A. F. HUXLEY. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol. (Lond.)*, Vol. 117, No. 4, pp. 500–544, Aug 1952.

[55] A. L. HODGKIN and A. F. HUXLEY. Currents carried by sodium and potas-

sium ions through the membrane of the giant axon of Loligo. *J. Physiol. (Lond.)*, Vol. 116, No. 4, pp. 449–472, Apr 1952.

[56] A. L. HODGKIN and A. F. HUXLEY. The components of membrane conductance in the giant axon of Loligo. *J. Physiol. (Lond.)*, Vol. 116, No. 4, pp. 473–496, Apr 1952.

[57] A. L. HODGKIN and A. F. HUXLEY. The dual effect of membrane potential on sodium conductance in the giant axon of Loligo. *J. Physiol. (Lond.)*, Vol. 116, No. 4, pp. 497–506, Apr 1952.

[58] A. L. HODGKIN, A. F. HUXLEY, and B. KATZ. Measurement of current-voltage relations in the membrane of the giant axon of Loligo. *J. Physiol. (Lond.)*, Vol. 116, No. 4, pp. 424–448, Apr 1952.

[59] Keiichi Ida, Yasuyuki Ohno, Shunsuke Inoue, and Kazuo Minami. Performance Profiling and Debugging on the K computer. *FUJITSU Science and Technology Journal*, Vol. 48, No. 3, pp. 331–339, 2012.

[60] Hidetoshi Ikeno, Tomoki Kazawa, Shigehiro Namiki, Daisuke Miyamoto, Yohei Sato, Stephan Shuichi Haupt, Ikuko Nishikawa, and Ryohei Kanzaki. Development of a scheme and tools to construct a standard moth brain for neural network simulations. *Computational intelligence and neuroscience*, Vol. 2012, p. 795291, 2012.

[61] Tammo Ippen, Jochen Martin Eppler, Hans Ekkehard Plesser, and Markus Diesmann. Constructing neuronal network models in massively parallel environments. *Frontiers in Neuroinformatics*, Vol. 11, No. May, p. 30, 2017.

[62] IVBPF. `http://invbrain.neuroinf.jp/modules/htmldocs/IVBPF/Top/index.html`.

[63] M. Iwano, E. S. Hill, A. Mori, T. Mishima, T. Mishima, K. Ito, and R. Kanzaki. Neurons associated with the flip-flop activity in the lateral accessory lobe and ventral protocerebrum of the silkworm moth brain. *J. Comp. Neurol.*, Vol. 518, No. 3, pp. 366–388, Feb 2010.

[64] E. M. Izhikevich and G. M. Edelman. Large-scale model of mammalian thalamocortical systems. *Proc. Natl. Acad. Sci. U.S.A.*, Vol. 105, No. 9, pp. 3593–3598, Mar 2008.

[65] E.M. Izhikevich. Which Model to Use for Cortical Spiking Neurons? *IEEE Transactions on Neural Networks*, Vol. 15, No. 5, pp. 1063–1070, sep 2004.

[66] Eric R Kandel, Henry Markram, Paul M Matthews, Rafael Yuste, and Christof Koch. Neuroscience thinks big (and collaboratively). *Nature reviews. Neuro-*

science, Vol. 14, No. 9, pp. 659–64, sep 2013.

[67] Svilen Kanev, Sam Likun Xi, Gu-Yeon Wei, and David Brooks. Mallacc: Accelerating Memory Allocation. *SIGOPS Oper. Syst. Rev.*, Vol. 51, No. 2, pp. 33–45, 2017.

[68] R. Kanzaki, A. Ikeda, and T. Shibuya. Morphological and physiological properties of pheromone-triggered flipflopping descending interneurons of the male silkworm moth, bombyx mori. *Journal of Comparative Physiology A*, Vol. 175, No. 1, pp. 1–14, 1994.

[69] Ryohei Kanzaki and Tatsuya Mishima. Pheromone-triggered'fiipflopping'neural signals correlate with activities of neck motor neurons of a male moth, bombyx mori. *Zoological science*, Vol. 13, No. 1, pp. 79–87, 1996.

[70] Narayanan Kasthuri, Kenneth Jeffrey Hayworth, Daniel Raimund Berger, Richard Lee Schalek, José Angel Conchello, Seymour Knowles-Barley, Dongil Lee, Amelio Vázquez-Reina, Verena Kaynig, Thouis Raymond Jones, Mike Roberts, Josh Lyskowski Morgan, Juan Carlos Tapia, H. Sebastian Seung, William Gray Roncal, Joshua Tzvi Vogelstein, Randal Burns, Daniel Lewis Sussman, Carey Eldin Priebe, Hanspeter Pfister, and Jeff William Lichtman. Saturated Reconstruction of a Volume of Neocortex. *Cell*, Vol. 162, No. 3, pp. 648–661, 2015.

[71] Tomoki Kazawa, Hidetoshi Ikeno, and Ryohei Kanzaki. Development and application of a neuroinformatics environment for neuroscience and neuroethology. *Neural Networks*, Vol. 21, No. 8, pp. 1047–1055, 2008.

[72] Tomoki Kazawa, Hidetoshi Ikeno, and Ryohei Kanzaki. Development and application of a neuroinformatics environment for neuroscience and neuroethology. *Neural networks : the official journal of the International Neural Network Society*, Vol. 21, No. 8, pp. 1047–55, oct 2008.

[73] Naomi Keren, Noam Peled, and Alon Korngreen. Constraining Compartmental Models Using Multiple Voltage Recordings and Genetic Algorithms. *Journal of Neurophysiology*, Vol. 94, No. 6, pp. 3730–3742, dec 2005.

[74] Wouter Klijn, Ben Cumming, Alexander Peyser, Vasileios Karakasis, and Stuart Yates. NestMC A morphologically detailed neural network simulator for modern high performance computer architectures. p. 15.

[75] S. Kumar, P. Heidelberger, D. Chen, and M. Hines. Optimization of Applications with Non-blocking Neighborhood Collectives via Multisends on the Blue Gene/P Supercomputer. *IPDPS*, Vol. 2010, pp. 1–11, Apr 2010.

[76] K. S. Kundert. *Sparse matrix techniques and their applications to circuit simulation. In Circuit Analysis Simulation and Design.* North-Holland, New York, 1986.

[77] Susanne Kunkel, Tobias C Potjans, Jochen M Eppler, Hans Ekkehard Plesser, Abigail Morrison, and Markus Diesmann. Meeting the memory challenges of brain-scale network simulation. *Frontiers in neuroinformatics*, Vol. 5, p. 35, jan 2011.

[78] Susanne Kunkel, Maximilian Schmidt, Jochen M Eppler, Hans E Plesser, Gen Masumoto, Jun Igarashi, Shin Ishii, Tomoki Fukai, Abigail Morrison, Markus Diesmann, and Moritz Helias. Spiking network simulation code for petascale computers. *Frontiers in neuroinformatics*, Vol. 8, p. 78, jan 2014.

[79] Yi Hsuan Lee, Yen Nan Lin, Chao Chun Chuang, and Chung Chuan Lo. SPIN: A method of skeleton-based polarity identification for neurons. *Neuroinformatics*, Vol. 12, No. 3, pp. 487–507, 2014.

[80] Xin Liu, Yi Zeng, Tielin Zhang, and Bo Xu. Parallel Brain Simulator: A Multi-scale and Parallel Brain-Inspired Neural Network Modeling and Simulation Platform. *Cognitive Computation*, Vol. 8, No. 5, pp. 967–981, 2016.

[81] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J. Ligocki, Matthew J. Cordery, Nicholas J. Wright, Mary W. Hall, and Leonid Oliker. Roofline model toolkit: A practical tool for architectural and program analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 8966, pp. 129–148, 2015.

[82] William W. Lytton, Alexandra H. Seidenstein, Salvador Dura-Bernal, Robert A. McDougal, Felix Schürmann, and Michael L. Hines. Simulation Neurotechnologies for Advancing Brain Research: Parallelizing Large Networks in NEURON. *Neural Computation*, Vol. 28, No. 10, pp. 2063–2090, oct 2016.

[83] H. Markram. The blue brain project. *Nat. Rev. Neurosci.*, Vol. 7, No. 2, pp. 153–160, Feb 2006.

[84] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W. Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, Guy Antoine Atenekeng Kahou, Thomas K. Berger, Ahmet Bilgili, Nenad Buncic, Athanassia Chalimourda, Giuseppe Chindemi, Jean-Denis Courcol, Fabien Delalondre, Vincent Delattre, Shaul Druckmann, Raphael Dumusc, James Dynes, Stefan Eilemann, Eyal Gal, Michael Emiel Gevaert, Jean-Pierre Ghobril, Albert Gidon, Joe W. Graham,

Anirudh Gupta, Valentin Haenel, Etay Hay, Thomas Heinis, Juan B. Hernando, Michael Hines, Lida Kanari, Daniel Keller, John Kenyon, Georges Khazen, Yihwa Kim, James G. King, Zoltan Kisvarday, Pramod Kumbhar, Sébastien Lasserre, Jean-Vincent Le Bé, Bruno R.C. Magalhães, Angel Merchán-Pérez, Julie Meystre, Benjamin Roy Morrice, Jeffrey Muller, Alberto Muñoz-Céspedes, Shruti Muralidhar, Keerthan Muthurasa, Daniel Nachbaur, Taylor H. Newton, Max Nolte, Aleksandr Ovcharenko, Juan Palacios, Luis Pastor, Rodrigo Perin, Rajnish Ranjan, Imad Riachi, José-Rodrigo Rodríguez, Juan Luis Riquelme, Christian Rössert, Konstantinos Sfyrakis, Ying Shi, Julian C. Shillcock, Gilad Silberberg, Ricardo Silva, Farhan Tauheed, Martin Telefont, Maria Toledo-Rodriguez, Thomas Tränkler, Werner Van Geit, Jafet Villafranca Díaz, Richard Walker, Yun Wang, Stefano M. Zaninetta, Javier DeFelipe, Sean L. Hill, Idan Segev, and Felix Schürmann. Reconstruction and Simulation of Neocortical Microcircuitry. *Cell*, Vol. 163, No. 2, pp. 456–492, oct 2015.

[85] Stefano Masoli, Sergio Solinas, and Egidio D'Angelo. Action potential processing in a detailed Purkinje cell model reveals a critical role for axonal compartmentalization. *Frontiers in cellular neuroscience*, Vol. 9, p. 47, jan 2015.

[86] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, December 1995.

[87] Robert A. McDougal, Thomas M. Morse, Ted Carnevale, Luis Marenco, Rixin Wang, Michele Migliore, Perry L. Miller, Gordon M. Shepherd, and Michael L. Hines. Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience. *Journal of Computational Neuroscience*, Vol. 42, No. 1, pp. 1–10, feb 2017.

[88] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, Vol. 345, No. 6197, pp. 668–673, 2014.

[89] Meschach. `http://homepage.math.uiowa.edu/~dstewart/meschach/`.

[90] M. Migliore, C. Cannia, W. W. Lytton, H. Markram, and M. L. Hines. Parallel network simulations with NEURON. *J Comput Neurosci*, Vol. 21, No. 2, pp.

119–129, Oct 2006.

[91] M Migliore, C Cannia, W W Lytton, Henry Markram, and M L Hines. Parallel network simulations with NEURON. *Journal of computational neuroscience*, Vol. 21, No. 2, pp. 119–29, oct 2006.

[92] M. Migliore, F. Cavarretta, M. L. Hines, and G. M. Shepherd. Distributed organization of a brain microcircuit analyzed by three-dimensional modeling: the olfactory bulb. *Front Comput Neurosci*, Vol. 8, p. 50, 2014.

[93] Michele Migliore, Francesco Cavarretta, Michael L Hines, and Gordon M Shepherd. Functional neurology of a brain system: a 3D olfactory bulb model to process natural odorants. *Functional neurology*, Vol. 28, No. 3, pp. 241–3, jan 2013.

[94] Marcin Miłkowski. Explanatory completeness and idealization in large brain simulations: a mechanistic perspective. *Synthese*, Vol. 193, No. 5, pp. 1457–1478, 2016.

[95] Daisuke Miyamoto, Tomoki Kazawa, and Ryohei Kanzaki. Neural circuit simulation of hodgkin-huxley type neurons toward peta scale computers. In *SC Companion*, p. 1541. IEEE Computer Society, 2012.

[96] Gordon E Moore, et al. Cramming more components onto integrated circuits, 1965.

[97] Abigail Morrison, a D Aertsen, Markus Diesmann, A Morrison, and M Diesmann. Spike-Timing-Dependent Plasticity in Balanced Random Networks. *Neural Computation Massachusetts Institute of Technology*, Vol. 19, pp. 1437–1467, 2007.

[98] Abigail Morrison, a D Aertsen, Markus Diesmann, Abigail Morrison, Markus Diesmann, Tammo Ippen, Jochen Martin Eppler, Hans Ekkehard Plesser, Markus Diesmann, Abigail Morrison, Carsten Mehring, Theo Geisel, a D Aertsen, Markus Diesmann, N Brunel, N Brunel, Susanne Kunkel, Tobias C Potjans, Jochen Martin Eppler, Hans Ekkehard Plesser, Abigail Morrison, Markus Diesmann, Maximilian Schmidt, Jochen Martin Eppler, Hans Ekkehard Plesser, Gen Masumoto, Jun Igarashi, Shin Ishii, Tomoki Fukai, Abigail Morrison, Markus Diesmann, and Moritz Helias. Constructing neuronal network models in massively parallel environments. *Frontiers in neuroinformatics*, Vol. 8, No. 8, pp. 183–208, jan 2005.

[99] Abigail Morrison, Carsten Mehring, Theo Geisel, a D Aertsen, and Markus Diesmann. Advancing the boundaries of high-connectivity network simulation

with distributed computing. *Neural computation*, Vol. 17, No. 8, pp. 1776–1801, 2005.

[100] S. Namiki, S. S. Haupt, T. Kazawa, A. Takashima, H. Ikeno, and R. Kanzaki. Reconstruction of virtual neural circuits in an insect brain. *Front Neurosci*, Vol. 3, No. 2, pp. 206–213, Sep 2009.

[101] Shigehiro Namiki, Satoshi Iwabuchi, Poonsup Pansopha Kono, and Ryohei Kanzaki. Information flow through neural circuits for pheromone orientation. *Nature Communications*, Vol. 5, p. 5919, dec 2014.

[102] I. Nishikawa, Y. Igarashi, T. Kazawa, S. Namiki, H. Ikeno, and R. Kanzaki. Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on, 2011.

[103] Ikuko Nishikawa, Yuka Yamagishi, Hidetoshi Ikeno, Tomoki Kazawa, Shu Namiki, and Ryohei Kanzaki. Estimation of the information pathway in an insect brain based on the physiological data. In *Information Science and Service Science and Data Mining (ISSDM), 2012 6th International Conference on New Trends in*, pp. 463–465, 2012.

[104] Christian Nowke, Daniel Zielasko, Benjamin Weyers, Alexander Peyser, Bernd Hentschel, and Torsten W. Kuhlen. Integrating Visualizations into Modeling NEST Simulations. *Frontiers in Neuroinformatics*, Vol. 9, No. October, pp. 1–20, 2015.

[105] H. Ogawa, Y. Baba, and K. Oka. Dendritic calcium accumulation regulates wind sensitivity via short-term depression at cercal sensory-to-giant interneuron synapses in the cricket. *J. Neurobiol.*, Vol. 46, No. 4, pp. 301–313, Mar 2001.

[106] OpenMP Architecture Review Board. Openmp application programming interface version 4.5, 2015.

[107] Danilo Pani, Paolo Meloni, Giuseppe Tuveri, Francesca Palumbo, Paolo Massobrio, and Luigi Raffo. An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks. *Frontiers in Neuroscience*, Vol. 11, p. 90, feb 2017.

[108] Ruchi Parekh and Giorgio A Ascoli. Neuronal morphology goes digital: a research hub for cellular and system neuroscience. *Neuron*, Vol. 77, No. 6, pp. 1017–38, mar 2013.

[109] Hanchuan Peng, Alessandro Bria, Zhi Zhou, Giulio Iannello, and Fuhui Long. Extensible visualization and analysis for multidimensional images using Vaa3D. *Nature Protocols*, Vol. 9, No. 1, pp. 193–208, jan 2014.

[110] W. Rall. *Core conductor theory and cable properties of neurons.* American

Physiological Society, 1977.

[111] GNU Readline Library. http://tiswww.case.edu/php/chet/readline/rltop.html.

[112] K. Rein, M. Zockler, M. T. Mader, C. Grubel, and M. Heisenberg. The Drosophila standard brain. *Curr. Biol.*, Vol. 12, No. 3, pp. 227–231, Feb 2002.

[113] A. Rodriguez, D. B. Ehlenberger, P. R. Hof, and S. L. Wearne. Three-dimensional neuron tracing by voxel scooping. *J. Neurosci. Methods*, Vol. 184, No. 1, pp. 169–175, Oct 2009.

[114] A Roth and M Häusser. Compartmental models of rat cerebellar Purkinje cells based on simultaneous somatic and dendritic patch-clamp recordings. *The Journal of physiology*, Vol. 535, No. Pt 2, pp. 445–72, sep 2001.

[115] S Rotter and M Diesmann. Exact digital simulation of time-invariant linear systems with applications to neuronal modeling. *Biological cybernetics*, Vol. 81, No. 5-6, pp. 381–402, 1999.

[116] Kentaro Sano and Satoru Yamamoto. Performance Evaluation of FPGA-based Custom Accelerators for Iterative Linear-Equation Solvers. No. June, 2011.

[117] Toshio Sato, Shimpei and Sato, Yukinori and Endo. Investigating Potential Performance Benefits of Memory Layout Optimization Based on Roofline Model. *Proceedings of the 2Nd International Workshop on Software Engineering for Parallel Systems*, pp. 50–56, 2015.

[118] SPARC64VIIIfx Extensions. http://img.jp.fujitsu.com/downloads/jp/jhpc/sparc64viiifx-extensions.pdf.

[119] SPARSE 1.3. http://www.netlib.org/sparse/.

[120] SUNDIALS. http://computation.llnl.gov/casc/sundials/main.html.

[121] Herb Sutter. The free lunch is over: A fundamental turn toward concurrency in software. Dr. Dobb' s journal, Vol. 30, No. 3, pp. 202–210, 2005.

[122] Ataru Tanikawa, Kohji Yoshikawa, Keigo Nitadori, and Takashi Okamoto. Phantom-GRAPE: Numerical software library to accelerate collisionless N-body simulation with SIMD instruction set on x86 architecture. *New Astronomy*, Vol. 19, pp. 74–88, feb 2013.

[123] Fukuda Tetsuya, Kazawa Tomoki, and Kanzaki Ryohei. Establishment of the estimation method of the neural network using CMA-ES for elucidating the neural mechanism of a silkworm moth brain. *Frontiers in Neuroinformatics*, Vol. 10, , 2016.

[124] TOP500. http://www.top500.org/.

[125] Werner Van Geit, Michael Gevaert, Giuseppe Chindemi, Christian Rössert, Jean-Denis Courcol, Eilif B. Muller, Felix Schürmann, Idan Segev, and Henry Markram. BluePyOpt: Leveraging Open Source Software and Cloud Infrastructure to Optimise Model Parameters in Neuroscience. *Frontiers in Neuroinformatics*, Vol. 10, p. 17, jun 2016.

[126] Satoshi Wada and Ryohei Kanzaki. Neural control mechanisms of the pheromone-triggered programmed behavior in male silkmoths revealed by double-labeling of descending interneurons and a motor neuron. *Journal of Comparative Neurology*, Vol. 484, No. 2, pp. 168–182, 2005.

[127] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, No. 393, pp. 440–442, 1998.

[128] Samuel Williams, Andrew Waterman, and David Patterson. Roofline. *Communications of the ACM*, Vol. 52, No. 4, p. 65, 2009.

[129] Takayuki Yamasaki, Teijiro Isokawa, Nobuyuki Matsui, Hidetoshi Ikeno, and Ryohei Kanzaki. Reconstruction and simulation for three-dimensional morphological structure of insect neurons. *Neurocomputing*, Vol. 69, No. 10-12, pp. 1043–1047, jun 2006.

[130] Tadashi Yamazaki and Jun Igarashi. Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in realtime using a graphics processing unit. *Neural Networks*, Vol. 47, pp. 103–111, nov 2013.

[131] Tadashi Yamazaki, Jun Igarashi, Junichiro Makino, and Toshikazu Ebisuzaki. Real-time simulation of a cat-scale artificial cerebellum on PEZY-SC processors. *The International Journal of High Performance Computing Applications*, p. 109434201771070, jun 2017.

[132] Tadashi Yamazaki, Hidetoshi Ikeno, Yoshihiro Okumura, Shunji Satoh, Yoshimi Kamiyama, Yutaka Hirata, Keiichiro Inagaki, Akito Ishihara, Takayuki Kannon, and Shiro Usui. Simulation Platform: A cloud-based online simulation environment. *Neural Networks*, Vol. 24, No. 7, pp. 693–698, sep 2011.

[133] Esin Yavuz, James Turner, and Thomas Nowotny. GeNN: a code generation framework for accelerated brain simulations. *Scientific reports*, Vol. 6, p. 18854, jan 2016.

[134] Friedemann Zenke and Wulfram Gerstner. Limits to high-speed simulations of spiking neural networks using general-purpose computers. *Frontiers in neuroinformatics*, Vol. 8, No. September, 2014.

[135] 佐藤陽平. 細胞形態を考慮した神経活動シミュレーション環境の開発. 2011.

東京大学 大学院工学系研究科 先端学際工学専攻
学籍番号：37-157160

氏名：宮本 大輔

指導教員：神崎 亮平 教授

Build: 2018 年 2 月 8 日