自動チューニング化手法

井上裕太

平成 29 年 12 月 30 日

目 次

図 目 次

表 目 次

1 序論

1.1 神経科学においてシミュレーションを行う意義

・ボトムアップアプローチ

・また, 当研究室ではそうしたモデル構築のために実験などを行い, そうしたデータを元に様々なシミュレーション系を構築してきた
脳機能の理解を目的として, スーパコンピュータを用いた神経回路のシミュレーションが行われている. また, 消費電力やシミュレーションの割り当て時間といったリソースの問題やリアルタイムデータ同化への需要からシミュレーションの高速化・最適化が求められている.

また, 現代の計算機にも多様な種類が存在し, それぞれに対する最適化も個別に行われてきた. 本研究の目的はそれぞれの細胞モデルのシミュレーションコードを個々

のアーキテクチャに合わせて, 自動又は半自動的に最適化を行う手法を確立することである.

1.2 神経回路シミュレーションの高速化・最適化への需要

・神経回路シミュレーションには非常に大きな計算力が必要である一方で, こうした神経回路シミュレーションには非常に大きな計算能力が必要とされてきた.
本研究はスーパーコンピュータ京に関連するポスト京プロジェクトの一環として行われているが,
スーパーコンピュータを用いてもなお計算には多くの時間がかかっている.
そうした状態を踏まえ, 系の構築だけでなくシミュレーション自体の高速化・最適化が求められている.
・神経回路シミュレーションの最適化の難しさしかし, 神経細胞には様々な種類のものが存在するため, 個々の神経細胞のイオンチャンネルのモデルを最適化された形で実装するために, これまでそれぞれのモデルに対して多大な努力が行われてきた.
・本研究の意義そこで, 本研究では個々のイオンチャンネルモデルを自動で最適化するソフトウェアを作成することで, これまで人の手で逐次行われてきた最適化の汎用化を目指す.

1.3 先行研究

1.3.1 宮本さんの

すごい

1.3.2 片桐先生の

すごい
メモリが大事

1.4 研究の目的と手法

高速化・最適化への需要への項で述べたように, 本研究は個々の神経細胞のイオンチャンネルモデルに対し汎用的な最適化手法を開発することである.
神経回路シミュレーションを行うソフトウェアは多数存在するが, 本研究では先行研究で用いられていた NEURON というソフトウェアを利用する.
NEURON では, 神経細胞のモデルとそれぞれの細胞の関係を微分方程式の形でモデルファイル（MOD ファイル）として記述することができ, nmodl というトランスパイラが MOD ファイルを C ファイルに変換することで実行している.

先行研究では, この生成されたCファイルに着目し手動での最適化を行っていたが, 本研究ではCファイルの生成と実行, 結果の集約を自動で行うことで複数のパラメータを試し, シミュレーションを実行する上で最適なパラメータを選択することを目指す.

1.5 本論文の構成

本論文は全6章から構成されている.

本章では本研究の背景と目的を示した.

第2章では, 本研究が対象とする神経回路シミュレーションの系, そしてシミュレーションを行う環境について述べる.

第3章では, 本研究で作成したプログラムについての詳細を述べる.

第4章では, シミュレーションの結果を示す.

第5章では, シミュレーション結果の考察を述べる.

第6章では, 本研究のまとめ, 成果を示した上で将来の課題について述べる.

2　シミュレーションモデル

2.1　Hodgkin Huxley モデル

2.2　Purkinje Cell モデル

2.3　ベンチマークモデル

3　シミュレーション環境

3.1　京コンピュータ

3.1.1　CPU アーキテクチャ

3.1.2　キャッシュ・メモリ

3.2　研究室クラスタ

3.2.1　CPU アーキテクチャ

3.2.2　キャッシュ・メモリ

4　最適化の手法

本研究では, モデルに依存するパラメータと実行マシンに依存するパラメータ, そしてプログラムのコンパイル時に関わるパラメータ（コンパイルオプション）を調節することでシミュレーション系の最適化を目指した.
以下にそれぞれのパラメータの詳細を示す.

4.1　モデルに依存するパラメータ

以下に Hodgkin-Huxley 方程式のモデルを例としてそれぞれのパラメータを示す. モデルに依存するパラメータに関しては先行研究 (TODO: add reference) において SIMD 化, 配列構造の最適化により計算速度が大きく向上することが示されているため, その二つに加え配列構造の順序を入れ替えることによってキャッシュヒット率の向上が見込まれるかに対して取り組んだ.
Hodgkin-Huxley 方程式は,NEURON 内において MOD 形式で次のように記述されている.

Listing 1: aaa

```
1  TITLE hh_k.mod squid sodium, potassium, and leak channels
2
3  COMMENT
4   This is the original Hodgkin−Huxley treatment for the set of sodium,
5    potassium, and leakage channels found in the squid giant axon membrane.
6    ("A quantitative description of membrane current and its
         application
7    conduction and excitation in nerve" J.Physiol. (Lond.) 117:500−544
         (1952).)
8   Membrane voltage is in absolute mV and has been reversed in polarity
9    from the original HH convention and shifted to reflect a resting potential
10   of −65 mV.
11  Remember to set celsius=6.3 (or whatever) in your HOC file.
12  See squid.hoc for an example of a simulation using this model.
13  SW Jaslove 6 March, 1992
14 ENDCOMMENT
15
16 UNITS {
17         (mA) = (milliamp)
18         (mV) = (millivolt)
19         (S) = (siemens)
20 }
21
22 ? interface
23 NEURON {
24         SUFFIX hh_k
25         USEION na READ ena WRITE ina
26         USEION k READ ek WRITE ik
27         NONSPECIFIC_CURRENT il
28         RANGE gnabar, gkbar, gl, el, gna, gk
29         GLOBAL minf, hinf, ninf, mtau, htau, ntau
30         THREADSAFE : assigned GLOBALs will be per thread
31 }
32
33 PARAMETER {
34         gnabar = .12 (S/cm2) <0,1e9>
35         gkbar = .036 (S/cm2) <0,1e9>
36         gl = .0003 (S/cm2) <0,1e9>
37         el = −54.3 (mV)
38 }
39
40 STATE {
41         m h n
42 }
43
44 ASSIGNED {
45         v (mV)
46         celsius (degC)
47         ena (mV)
48         ek (mV)
49
50         gna (S/cm2)
51         gk (S/cm2)
52         ina (mA/cm2)
53         ik (mA/cm2)
```

```
54          il (mA/cm2)
55          minf hinf ninf
56          mtau (ms) htau (ms) ntau (ms)
57  }
58
59  ? currents
60  BREAKPOINT {
61          SOLVE states METHOD cnexp
62          gna = gnabar*m*m*m*h
63          ina = gna*(v - ena)
64          gk = gkbar*n*n*n*n
65          ik = gk*(v - ek)
66          il = gl*(v - el)
67  }
68
69
70  INITIAL {
71          rates(v)
72          m = minf
73          h = hinf
74          n = ninf
75  }
76
77  ? states
78  DERIVATIVE states {
79          rates(v)
80          m' = (minf-m)/mtau
81          h' = (hinf-h)/htau
82          n' = (ninf-n)/ntau
83  }
84
85  :LOCAL q10
86
87
88  ? rates
89  PROCEDURE rates(v(mV)) { :Computes rate and other constants at
        current v.
90                      :Call once from HOC to initialize inf at resting
                            v.
91          LOCAL alpha, beta, sum, q10
92          TABLE minf, mtau, hinf, htau, ninf, ntau DEPEND celsius FROM
               -100 TO 100 WITH 200
93
94  UNITSOFF
95          q10 = 3^((celsius - 6.3)/10)
96                 :"m" sodium activation system
97          alpha = .1 * vtrap(-(v+40),10)
98          beta = 4 * exp(-(v+65)/18)
99          sum = alpha + beta
100         mtau = 1/(q10*sum)
101         minf = alpha/sum
102                :"h" sodium inactivation system
103         alpha = .07 * exp(-(v+65)/20)
104         beta = 1 / (exp(-(v+35)/10) + 1)
105         sum = alpha + beta
106         htau = 1/(q10*sum)
107         hinf = alpha/sum
```

```
108                 :"n" potassium activation system
109         alpha = .01*vtrap(-(v+55),10)
110         beta = .125*exp(-(v+65)/80)
111         sum = alpha + beta
112         ntau = 1/(q10*sum)
113         ninf = alpha/sum
114 }
115
116 FUNCTION vtrap(x,y) { :Traps for 0 in denominator of rate eqns.
117         if (fabs(x/y) < 1e-6) {
118                 vtrap = y*(1 - x/y/2)
119         }else{
120                 vtrap = x/(exp(x/y) - 1)
121         }
122 }
123
124 UNITSON
```

先行研究の中でも示されている通り, この中でプロファイル結果から多くの計算時間を必要とするのは DERIVATIVE (TODO: reference) であり以下のパラメータの多くはこの計算を行う上でキャッシュヒット率をあげることを目的としている.

4.1.1 SIMD 化

・変数の配列化によるメモリアクセスの連続化

4.1.2 配列構造

・配列の構造変形（時間があれば）なければここを消す

4.1.3 配列順序

・変数がどう利用されるのかはその変数がどう呼び出されるかに依存する.
そのため，MOD ファイル内に記述された方程式で関連する変数を連続して定義した方が幾分効率化されると予測できる.
MOD ファイルから方程式部分を解析し, 関連する変数のペアを Union-find 木で作り関連する変数の組の中での順序をパラメータとして配列の順序を入れ替える.

4.2 実行マシンに依存するパラメータ

近年の CPU はシングルコアではなく, マルチコアによって計算を並列化することで全体としての計算能力を向上させている.
一方で, この並列化を行う上でのパラメータは実行するマシンごとに依存するもので

ある.
ここで主に対象としたパラメータは OpenMP のスレッド数と MPI のプロセス数である.

4.2.1 スレッド数

OpenMP のスレッドに関与するパラメータに関する説明 (TODO : わあああ)

4.2.2 プロセス数

MPI プロセスに関与するパラメータに関する説明 (TODO : わあああい)

4.3 コンパイルに関わるパラメータ

TODO: 時間があれば記述する

5 自動チューニングスクリプトと MOD トランスパイラの構築

本研究では環境・イオンチャンネルモデルに関わらない自動最適化を目的としている.
そのため, スーパーコンピュータ京・研究室クラスタ以外のマシンを用いる場合においても環境構築, プログラムの修正・実行にかかるコストは最小限になるべきである.
上記の要件を満たすため, 以下にあげる 3 種類のプログラムを作成した.
また, それぞれのプログラムは主として Python, Shell Script(TODO: reference) を使用している.

5.1 環境設定スクリプト

作成したシミュレータ・トランスパイラは Python(TODO: reference) のモジュールとして作成したが, pip(TODO: reference) のようなモジュール管理ツールが存在しない環境（スーパーコンピュータ京）においては, モジュールとして公開するだけでは不十分である.
特にスーパーコンピュータ京では, デフォルトの Python(TODO: reference) のバージョンが 2.6.6，sudo 権限を有しないため外部プログラムのインストールが難しいという環境であったため, Pyenv を利用して汎用的な環境を作成することにした.

以下作成したスクリプトの概要とその用途を示す.
・Makefile
このプロジェクトの Makefile（ TODO: reference）.
主に利用するのは以下の 3 つのコマンド
・make install
このコマンドでは以下に示す scripts/setup_env_and_install_libraries.sh を実行する.
・make pull
このコマンドでは以下に示す scripts/pull_required_projects.sh を実行する.
・make setup
上記の二つのコマンドを install,pull の順で実行する.
・scripts/
・setup_env_and_install_libraries.sh
このスクリプトは大きくわけ 2 つのことをする.
・Pyenv のインストール
・genie で必要となる Python のライブラリをインストール
・pull_required_projects.sh
このスクリプトはシミュレーションソフトである NEURON 環境を構築することを目的としている.

5.2 シミュレータ

最適化の方法として, 複数のパラメータからモデル, 実行環境に即したパラメータを選択するという手法を選択したが, そのためには複数のパラメータでシミュレーションを行いその結果を集約するプログラムが必要となる.
本研究ではこのパラメータ選択を容易かつ高速に行うため以下に示すプログラムを作成した.
・MOD ファイルからパラメータとなりうる変数を自動で抽出し, それぞれの関係性を元に配列とその順序の候補を生成する.
・ジョブキューのシステムを持っているマシンにおいて, 複数のジョブを並行して投げ結果を非同期的に集約できる.
・実行結果を最適化前のデフォルトの結果と比較し, 実行結果に対して影響がないかを確認する.
・json 形式で実行するファイル, 各パラメータの範囲（プロセス数は 1 から 10 など）を指定することができる.

5.2.1 全体構成

パラメータ探索の詳細を述べる前に作成したシミュレータプログラムの概略を疑似コードにて示す.

Listing 2: シミュレータ疑似コード

```
1  # Global variables
2  MAX_NUM_JOB = 4
3  running_job = []
4
5  main():
6    watch_job()
7
8    while model_param_candidates.has_next()
9      model_param = model_param_candidates.next()
10     while compile_param_candidates.has_next()
11       compile_param = compile_param_candidates.next()
12
13       build(model_param, compile_param)
14
15       while machine_param_candidates.has_next()
16         machine_param = machine_param_candidates.next()
17
18         job_id = run(machine_param)
19         running_job.add(job_id)
20
21       machine_param.reset()
22     compile_param.reset()
23
24 run(machine_param):
25   while running_job's size >= MAX_NUM_JOB:
26     sleep 10
27   job = make_job(machine_param)
28   job_id = deploy(job)
29   return job_id
30
31 watch_job():
32   while running_job's size == 0:
33     sleep 10
34   for job_id in running_job:
35     if is_finished(job_id):
36       summarize(job_id)
37       running_job.remove(job_id)
```

この疑似コードは大きく 3 つの機能から成立しているため, 次にその詳細を示す.

5.2.1.1 全体ループ

疑似コード内の 8-22 行目までが全体のループを構成している.

Listing 3: シミュレータ 全体ループ

```
1  while model_param_candidates.has_next()
2    model_param = model_param_candidates.next()
3    while compile_param_candidates.has_next()
4      compile_param = compile_param_candidates.next()
5
6      build(model_param, compile_param)
7
8      while machine_param_candidates.has_next()
9        machine_param = machine_param_candidates.next()
10
11       job_id = run(machine_param)
12       running_job.add(job_id)
13
14     machine_param.reset()
15   compile_param.reset()
```

ループ内部では, プログラムのビルド, ジョブスクリプトの生成, そして生成した実行形式とジョブスクリプトをジョブキューに deploy するという 3 つのことを行っている.
プログラムのビルド, ジョブの実行に比べジョブスクリプトの生成にかかる時間は無視できる程度のものであるため, ジョブスクリプトの生成をもっとも内側のループに持ってきた.
これにより, ジョブキューでジョブが実行されるのを待っているうちにビルドを行うことができ, 全体としてシミュレーションの時間を減らすことが期待できる.

5.2.1.2 ジョブ実行

Listing 4: シミュレータ ジョブ実行

```
1  run(machine_param):
2    while running_job's size >= MAX_NUM_JOB:
3      sleep 10
4    job = make_job(machine_param)
5    job_id = deploy(job)
6    return job_id
```

スーパーコンピュータ京のような複数のユーザーが用いるシステムにおいて, ジョブを一度に大量に投げるのは好ましくない.
そのため, ジョブをジョブキューに投げる前に事前に設定した最大同時ジョブ実行数と現在の実行中のジョブの数を比較し, 最大数と同数なのであれば待機する処理が必要である.
本研究では, グローバル変数として現在実行中のジョブの ID を保持するリストを定義し, そのリストの数と比較することで実現している.
また, 後述するジョブ結果の集約においてこのジョブ ID を保持するリストは別スレッドから参照されており, リスト内のジョブが完了した段階で mutex によってロックされた上で更新される.

5.2.1.3　ジョブ結果の集約

Listing 5: シミュレータ ジョブ結果の集約

```
watch_job():
  while running_job's size == 0:
    sleep 10
  for job_id in running_job:
    if is_finished(job_id):
      summarize(job_id)
      running_job.remove(job_id)
```

様々なパラメータの組の中から最適な組み合わせを選びたいため, それぞれのジョブの結果とパラメータの組を結びつける必要がある.
ジョブが完了したか否かは, ジョブの状況を取得するコマンドを利用することで得ることができる.
次にスーパーコンピュータ京と研究室クラスタで上記のコマンドを実行した結果を示す.

京

>> pjstat研究室クラスタ

>> qstat

5.2.2　モデルに依存するパラメータ

5.3　トランスパイラ

・MOD から C のトランスパイラの説明

6　シミュレーション結果

・結果を書きます

7　考察

・考察を書きます

8　結論

　頑張りました

Appendix