Федеральное государственное автономное образовательное учреждение высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Программа повышение конкурентоспособности ННГУ им. Н.И. Лобачевского
Стратегическая инициатива 7 «Достижение лидирующих позиций в области суперкомпьютерных технологий и высокопроизводительных вычислений»

Введение в динамику сигнальных процессов в нейронных сетях мозга (Блок лекций по аспектам высокопроизводительных вычислений)

Лекция № 1. Подходы к моделированию спайковых нейронных сетей.

В ходе предыдущих лекций были рассмотрены основные модели единичных нейронов и синапсов. Были изучены основные биофизические и динамические механизмы таких явлений в отдельных нейронах как формирование потенциала покоя, формирование порога, тормозное и возбуждающее воздействие, генерация спайков, распространение сигнала по отросткам типы возбудимости и динамические механизмы, лежащие в основе формирования таких типов, а также многое другое. Рассмотрены основные компоненты и принципы работы отдельных синапсов, изучены биофизические основы процессов синаптической нейропередачи, а также принципы и подходы к моделированию этих процессов.

В данном блоке лекций рассматриваются вопросы применения вычислительных алгоритмов и технологий для задач расчёта динамики нейронных сетей.

Для понимания способов и подходов к моделированию больших сетей с применением высокопроизводительных вычислений и суперкомпьютерных технологий необходимо дать представление о простых методах расчёта динамических систем.

Для решения ряда задач в компьютерном моделировании широкое применение находят численные методы. Они хорошо подходят, в том числе и для решения обыкновенных дифференциальных уравнений. Рассмотрим для начала задачу Коши для одного обыкновенного дифференциального уравнения. Задача Коши является одной из основных задач теории дифференциальных уравнений (обыкновенных и с частными производными) и состоит в нахождении решения дифференциального уравнения, удовлетворяющего так называемым начальным условиям. Начальные данные задаются при $t = 0$, а решение находится при $t > 0$. Задачу Коши можно рассматривать как одну из краевых задач, хотя область, в которой задается поиск решения, заранее не указывается.

Задача Коши имеет единственное решение, если она имеет решение $y = f(x)$ и никакое другое решение не отвечает интегральной кривой, которая в сколь угодно малой выколотой окрестности точки (x_0, y_0) имеет поле направлений, совпадающих с полем $y=f(x)$. Точка (x_0, y_0) задает начальные условия.

Метод Эйлера

Метод Эйлера играет важную роль в теории численных методов решения обыкновенных дифференциальных уравнений. Он является явным, одношаговым методом первого порядка точности, основанным на аппроксимации интегральной кривой кусочно-заданной функцией (ломанной Эйлера).

Пусть дана задача Коши для уравнения первого порядка

$$\frac{dy}{dx} = f(x, y), \tag{1}$$

$$y_{|x=x_0}=y_0, \tag{2}$$

где функция f определена на некоторой области $D \subset R^2$. Решение ищется на интервале $(x_0, b]$. На этом интервале введем узлы

$$x_0 < x_1 < \ldots < x_n \leq b$$

Тогда приближенное значение (y_i) в узлах x_i будет определяться по формуле

$$y_i=y_{i-1} + (x_i - x_{i-1})f(x_{i-1}, y_{i-1}) \quad i = 1,2,3, \ldots, n \tag{3}$$

Эти формулы обобщаются на случай обыкновенных дифференциальных уравнений.

Кроме метода Эйлера для решения обыкновенных дифференциальных уравнений применяется множество других методов. Среди них стоит упомянуть семейство методов Рунге-Кутты, неявный метод Эйлера и др.

Рассмотрим пример интегрирования модели нейрона при помощи программы, написанной на языке C++

Как и при изучении механизмов генерации потенциала действия (спайка) в качестве отправной точки возьмём модель Ходжкина-Хаксли.

В качестве единичного нейрона использовалась модификация модели Ходжкина-Хаксли [1], состоящая из 4-х обыкновенных дифференциальных уравнений следующего вида:

$$C\frac{dV}{dt} = -g_{Na}m^3h(V-E_{Na}) - g_K n^4(V-E_K) - g_{leak}(V-E_{leak}) - \sum(I_{exc}+I_{inh}) + I_e,$$

$$\frac{dm}{dt} = \frac{(m_\infty(V)-m)}{\tau_m(V)};$$

$$\frac{dh}{dt} = \frac{(h_\infty(V)-h)}{\tau_h(V)}; \qquad (4)$$

$$\frac{dn}{dt} = \frac{(n_\infty(V)-n)}{\tau_n(V)};$$

где $C = 1$ пкФ/см^2 – удельная мембранная ёмкость, V – мембранный потенциал, измеряемый в милливольтах, t – время в миллисекундах. Собственная динамика мембранного потенциала нейрона определяется тремя токами: натриевым, калиевым, и омическим током утечки с соответствующими максимальными проводимостями (в нСм/см^2) $g_{Na} = 120$, $g_K = 36$, $g_{leak} = 0.3$ и реверсивными потенциалами (в мВ) $E_{Na} = 55$, $E_K = -77$, $E_{leak} = -54.4$. Мгновенные значения активных ионных токов зависят от состояния воротных переменных m, h и n. $m_\infty(V)$, $h_\infty(V)$, $n_\infty(V)$, $\tau_m(V)$, $\tau_h(V)$ и $\tau_n(V)$ – их соответствующие равновесные функции активации и характерные потенциал зависимые времена релаксации, имеющие следующий вид:

$$m_\infty(V) = \frac{\alpha_m(V)}{\alpha_m(V)+\beta_m(V)};$$
$$\tau_m(V) = \frac{1}{\alpha_m(V)+\beta_m(V)};$$

$$h_\infty(V) = \frac{\alpha_h(V)}{\alpha_h(V)+\beta_h(V)};$$
$$\tau_h(V) = \frac{1}{\alpha_h(V)+\beta_h(V)};$$

$$n_\infty(V) = \frac{\alpha_n(V)}{\alpha_n(V)+\beta_n(V)};$$
$$\tau_n(V) = \frac{1}{\alpha_n(V)+\beta_n(V)};$$

где

$$\alpha_m = \frac{0.32(13-U)}{\frac{e^{13-U}}{4}-1},$$

$$\alpha_h = 0.128\frac{e^{17-U}}{18},$$

$$\alpha_n = \frac{0.032(15-U)}{\frac{e^{15-U}}{5}-1},$$

$$\beta_m = \frac{0.28(40-U)}{\frac{e^{40-U}}{5}-1},$$

$$\beta_h = \frac{4}{1+\frac{e^{40-U}}{5}},$$

$$\beta_n = \frac{1}{2}\frac{e^{10-U}}{40}.$$

Возбуждающие и тормозные синаптические токи вычисляются как

$I_{exc} = wy_{exc}(V - E_{exc})$

и

$I_{inh} = wy_{inh}(V - E_{inh}),$

где y – синаптическая переменная. Значения реверсивных синаптических потенциалов в милливольтах равны $E_{exc} = 0$ и $E_{inh} = -80$. Ток I_e моделирует внешнее воздействие.

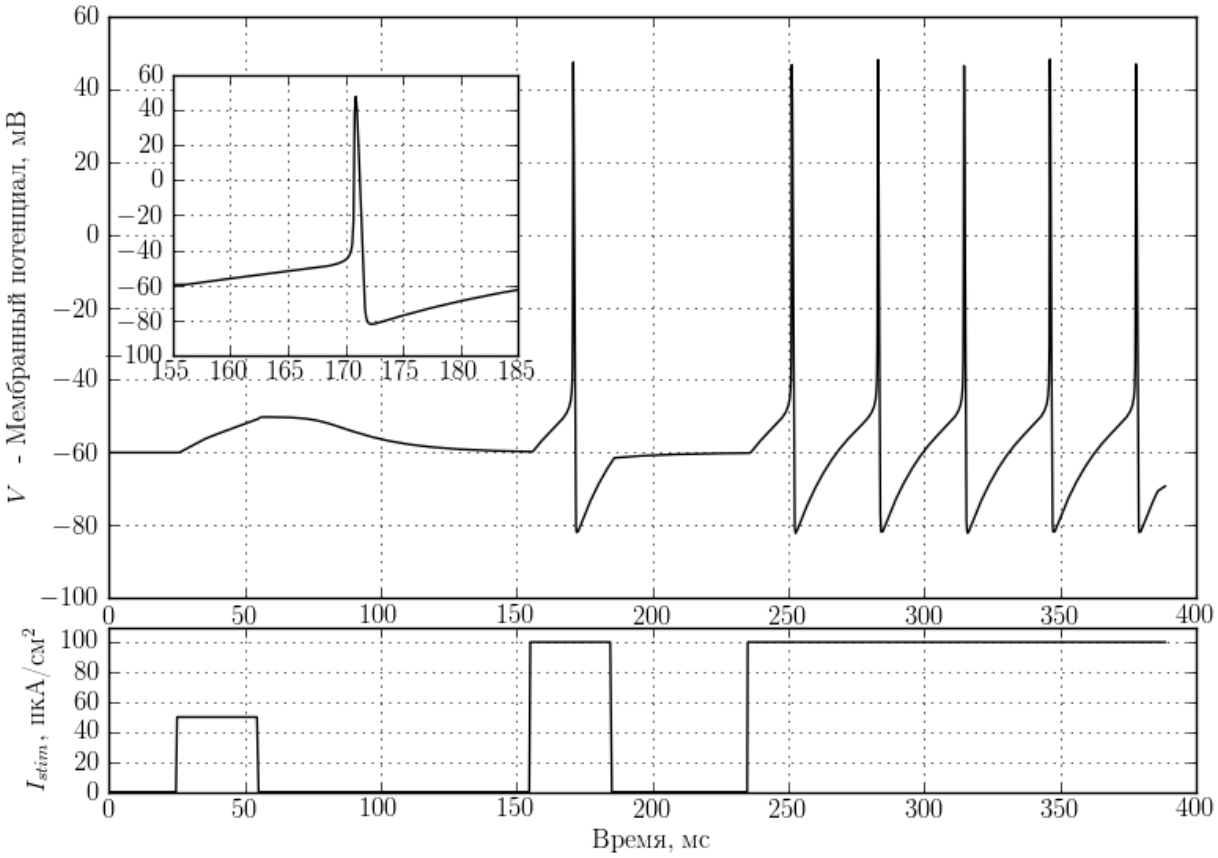

Рисунок 1. Эволюция мембранного потенциала (V из (1)) нейрона в ответ на предъявление подпорогового, сверхпорогового, а затем длительного сверхпорогового возбуждения. Нижний график – соответствующие значения стимулирующего тока I_e из (1). На вставке представлена форма спайка в увеличенном временном масштабе.

Рассмотрим пример реализации данной модели на языке программирования C++. Приведённый ниже код реализует решение системы (4) методом Эйлера. В код включены комментарии, поясняющие назначение всех строчек кода.

```
1  #include <iostream> // нужны для работы
2  #include <fstream>  // с файлами
3  #include <stdio.h>  // библиотека ввода-вывода
4  #include <math.h>   // для математических функций и констант
5
6  using namespace std; // чтобы не писать каждый раз "std::",
7  обращаясь к библиотеке
8
9  // объявим необходимые константы, задающие начальные условия
```

```
10  const double _V0 = -64.9537851420616;
11  const double _h0 = 0.594503593017066;
12  const double _n0 = 0.318385362336352;
13  const double _m0 = 0.0532216293734336;
14  //const double _s0 = 9.43291699944673E-13;
15  // длину шага интегрирования и время симуляции
16  const double _step = 0.05;
17  const double _Tsim = 10;
18
19   int main(){
20      // объявим все переменные, участвующие в интегрировании
21      double ALPHAn, BETTAn, ALPHAh, BETTAh, ALPHAm, BETTAm,
22             Minfinity, Ikal, Inatr, Ileak, t;
23      int i;
24      double* V_;
25      double V, n, m, h;
26      double Cm=1;
27      double gleak=0.3;
28      double Eleak=-54.4;
29      double gnatr=120;
30      double Enatr=55;
31      double phi=1;
32      double gkal=36;
33      double Ekal=-77;
34      double Iapp=0;
35      double THETAsyn=0;
36      double ALPHA=12;
37      double BETTA=0.1;
38      ofstream V_file;
39
40      // присвоим переменным уравнений динамики нейрона их начальные
            условия
41      V = _V0;
42      h = _h0;
43      n = _n0;
44      m = _m0;
45
46      // откроем файл для зяписи результатов симуляции
47      V_file.open("V.csv");
48
49      // начинаем цикл симуляции
50      for (t = 0; t < _Tsim; t += _step) {
```

```
51        // вычислим значения функций активации
52        ALPHAn = -0.01*(V + 55)/(exp(-0.1*(V+ 55))-1);
53        BETTAn = 0.125*exp(-(V + 65)/80);
54        ALPHAh = 0.07*exp(-(V+65)/20);
55        BETTAh = 1.0/(exp(-0.1*(35 + V))+1);
56        ALPHAm = -0.1*(V + 40)/(exp(-0.1*(V + 40))-1);
57        BETTAm = 4*exp(-(V+65)/18);
58        Minfinity = ALPHAm/(BETTAm + ALPHAm);
59        // вычислим значения токов
60        Ikal = gkal*pow(n, 4)*(V-Ekal);
61        Inatr = gnatr*pow(h, 3)*m*(V - Enatr);
62        Ileak = gleak*(V - Eleak);
63        // интегрируем методом Эйлера
64        V += _step*(1/Cm*(- Ikal - Inatr - Ileak + Iapp));
65        m += _step*(ALPHAm*(1-m) - BETTAm*m);
66        h += _step*(ALPHAh*(1-h) - BETTAh*h);
67        n += _step*(ALPHAn*(1-n) - BETTAn*n);
68        // записываем в файл результаты счёта построчно: текущее
              время, пробел, текущее значение мембранного
              потенциала, символ конца строки
69        V_file << t << " " << V << endl;
70      }
71
72      V_file.close(); // завершаем работу с файлом
73      return 0; // главная функция программы должна возвращать
              какое-то значение
74  }
```

Данный код решает систему уравнений, описывающих динамику модели нейрона Ходжкина-Хаксли (4), численно прямым методом Эйлера. Интегрирование дифференциальных уравнений происходит с шагом равным значению константы `_step = 0.05`, на интервале времени от 0 мс до значения константы `_Tsim`.

Программа на языке Си – один или несколько текстовых файлов, которые называются исходными. Для выполнения программы эти файлы должны быть скомпилированы, то есть должен быть создан исполняемый файл, который запускается на компьютере и имеет инструкции для процессора.

Компиляция – есть процесс преобразования исходных файлов в исполняемый. Если, например, программа состоит из одного исходного файла `hello.c`,то для его компиляции компилятором GNU C необходимо выполнить команду

```
bash$ gcc hello.c -o hello
```

Либо можно создать папку поближе к корневой папке пользователя `/home/username/folder/hello.c` и запустить

```
[username@localhost ~]$ gcc ~/development/hello.c -o hello
```

В результате получается файл `hello`, имя которого указано в опции `-o`. Этот файл исполняемый и запускается командой

```
bash$ ./hello
```

Символы `./` перед `hello` означают что исполняемый файл следует искать в текущей директории.

Строка

```
bash$ gcc xxx.c yyy.c -o zzz -I./common -I.. -lm
```

расшифровывается следующим образом: «скомпилировать файлы `xxx.c` `yyy.c` в программу `zzz`; заголовочные файлы находятся в директориях `./common` и `..` (родительский каталог); подключить библиотеку `libm`». Библиотека `libm` (которая подключается при помощи опции `-lm`), содержит откомпилированные математические функции, которые объявляются в заголовочном файле `math.h`. Если используются функции из этой библиотеки (такие как `log, sin, cos, exp`), то они должны быть подключены при компиляции.

Подробную информацию об опциях компилятора gcc можно узнать, если набрать

```
bash$ man gcc
```

или

```
bash$ info gcc
```

Для того чтобы скомпилировать программу для решения уравнений Ходжкина-Хаксли нужно в директории с файлом исходного кода запустить команду компиляции.

Данный пример демонстрирует базовые возможности подхода компьютерного моделирования. Современные задачи из области вычислительной нейронауки

предполагают решение куда более сложных задач, требующих построения крупномасштабных моделей нейронных сетей. Например, человеческий мозг содержит около 10^{11} нейронов, каждый из которых может формировать с другими нейронами до нескольких десятков тысяч контактов. Поэтому современные работы по моделированию нередко задействуют распределённые вычисления.

В течение последних десятилетий технологии компьютерного моделирования существенно продвинулись. Суперкомпьютерные технологии стали более доступными, существенно выросла вычислительная мощность современных кластеров и суперкомпьютеров, разработаны более новые и эффективные подходы к параллельным вычислениям. Методы и алгоритмы компьютерной симуляции спайковых нейронных сетей также претерпели существенные изменения, позволяющие, например, проведение вычислительных экспериментов на масштабе локальных кортикальных структур мозга (сети, состоящие из 10^5 нейронов и 10^9 синапсов). При этом такие модели учитывают такие принципиально важные нейрофизиологические явления, как синаптическая пластичность.

Распределённые вычисления являются способом одновременного решения различных частей одной вычислительной задачи с использованием нескольких процессоров одного или двух компьютеров, объединенных в сеть, и представляют собой, таким образом, частный случай параллельных вычислений. Под параллельными вычислительными системами подразумевают физические, компьютерные или программные системы, осуществляющие синхронную обработку данных на многих вычислительных узлах. Для обеспечения связи между отдельными процессами параллельной задачи разработан программный интерфейс, описанный в стандарте MPI. MPI реализует единую систему взаимодействия процессов независимо от машинной архитектуры, взаимного расположения процессов и API операционной системы. MPI относится к классу МКМД ЭВМ (системы с множественным потоком команд и множественным потоком данных) с индивидуальной памятью, то есть к многопроцессорным системам с обменом сообщениями.

Рассмотрим характерные особенности стандарта MPI. В состав MPI входит библиотека программирования, которая определяет имена процедур, вызов и результаты их работы. Таким образом, программы, написанные на языках FORTRAN, C и C++, откомпилированные обычными компиляторами и связанные с MPI-библиотекой, являются параллельными программами, то есть программами, исходный код которых содержит функции и инструменты из библиотеки MPI. MPI является моделью многопроцессорной ЭВМ с обменом данных. MPI имеет много качественных реализаций, что обеспечивает

большое разнообразие функциональных возможностей, эффективное управление буфером сообщений и асинхронную коммуникацию. MPI имеет богатый выбор различных способов коммуникации и виртуальных топологий. MPI и различные его реализации (MPICH, OpenMPI) успешно применяются для осуществления распределенных вычислений в локальных сетях.

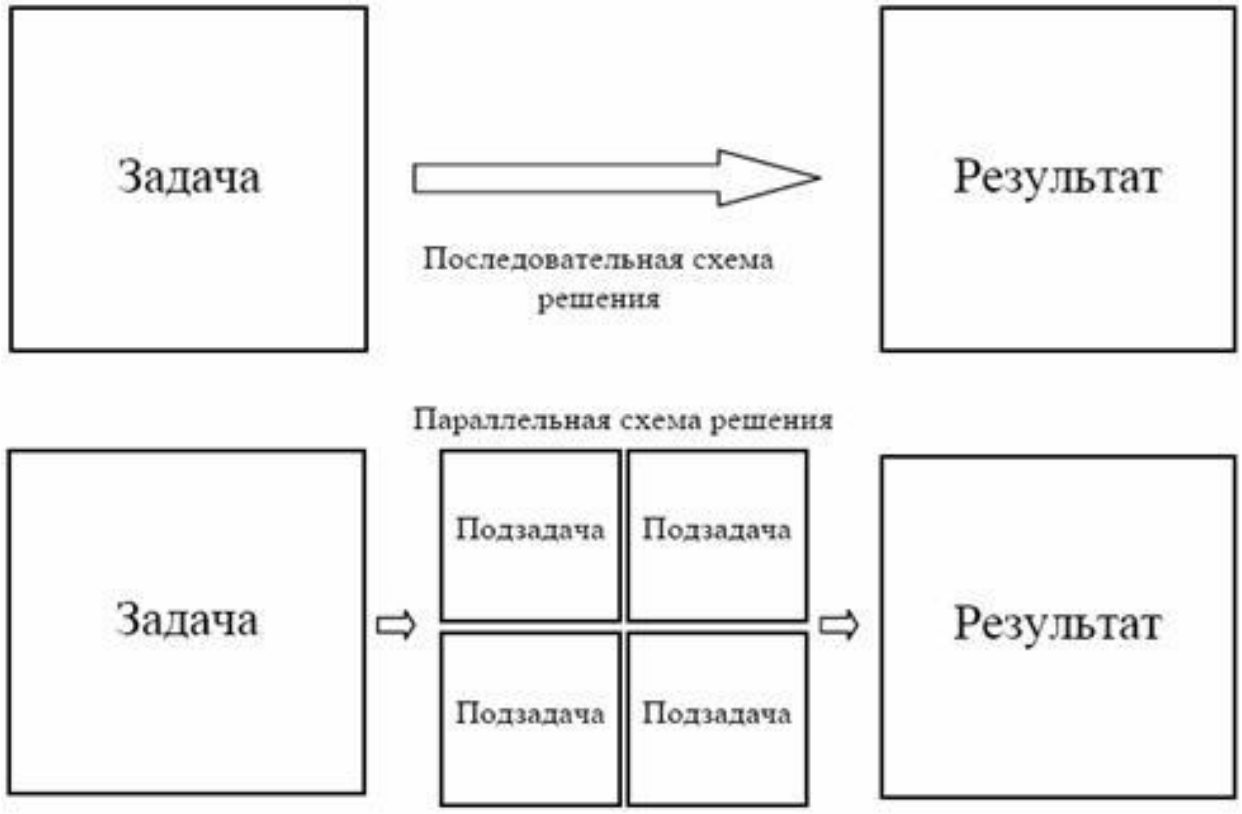

Рисунок 2. Пример способа распределённых вычислений

Параллельные вычисления заключают в себе определенные трудности, связанные со сложностью управления и координирования слаженной работы тысяч процессоров. Грамотное и адекватное распределение вычислений между процессорами системы должно учитывать тот факт, что операции коммуникации происходят гораздо медленнее, чем сами вычисления. Соответственно, необходимо минимизировать объем пересылаемых данных таким образом, чтобы избежать конфликта между степенью распараллеливания и объёмом коммуникаций (чем больше процессоров задействовано в решении параллельной задачи, тем больший объем данных необходимо пересылать между ними).

Среда параллельного программирования должна обеспечивать адекватное управление распределением и коммуникациями данных.

Из-за сложности параллельных компьютеров и их существенного отличия от традиционных однопроцессорных компьютеров нельзя просто воспользоваться

традиционными языками программирования и ожидать получения хорошей производительности.

В качестве примера применения параллельных вычислений возьмем расчет экспоненты через ряд Тейлора.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad -\infty < x < \infty \tag{5}$$

где суммирование происходит от *n=0* до бесконечности.

Данная формула легко поддается распараллеливанию, поскольку искомое число является суммой отдельных слагаемых, каждое из которых может быть вычислено отдельным процессором.

Количество вычисляемых слагаемых будет зависеть как от предела суммирования n, так и от количества задействованных процессоров k. Так, например, если длина интервала n=4, а количество процессоров k=5, то на каждый из первых четырех процессоров придется по одному слагаемому, а пятый останется свободным. В случае, когда n=10, k=5, на каждый процессор придется по два слагаемых для вычисления.

В начале главный процессор (первый процессор с рангом 0) отправляет остальным значение заданной пользователем переменной n с помощью функции широковещательной рассылки `MPI_Bcast`, которая в общем случае имеет следующий формат:

```
   int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm),
```

аргументы функции приведены ниже:

`buffer` – это адрес буфера с элементом,

`count` — количество элементов,

`datatype` — соответствующий тип данных в MPI,

`root` — ранг главного процессора,

`comm` — имя коммуникатора.

После того, как значение переменной n успешно отправлено, начинается вычисление слагаемых каждым процессором. Для этого в каждом шаге цикла к числу i, которое изначально равно рангу процессора, прибавляется число, равное количеству процессоров,

участвующих в вычислениях. Как только число i превысит заданное пользователем число n, выполнение цикла остановится.

В ходе выполнения цикла слагаемые прибавляются в отдельную переменную, которая после завершения цикла отправляется в главный процессор. Для этого используется функция приведения `MPI_Reduce`, в общем виде выглядящая так:

```
int MPI_Reduce(void *buf, void *result, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm).
```

С помощью этой функции объединяются элементы входного буфера каждого процесса в группе (операция `op`), и объединенное значение возвращается в выходной буфер процесса с именем `root`.

После выполнения программы на всех процессорах, главный процессор получит сумму всех слагаемых, которая и является искомым значением экспоненты.

Следует отметить, что при нахождении факториала как в случае параллельного, так и в случае последовательного метода вычисления экспоненты, используется рекурсивная функция. В данном случае вариант распараллеливания задачи нахождения факториала на разных процессорах является нерациональным ввиду следующих соображений. Так как коммуникационная среда является зачастую наиболее слабым звеном в системах параллельных вычислений, нахождение факториала каждого слагаемого раздельным образом существенно увеличит объем пересылаемых данных, что отрицательным образом скажется на общей скорости вычислений и падении производительности системы.

Данный пример хорошо иллюстрирует важность адекватного и всестороннего анализа при решении задач параллельного программирования.

Рассмотрим алгоритм выполнения кода:

1. Из визуальной оболочки в программу передается значения числа n, которое с помощью функции широковещательной рассылки передается всем процессорам.
2. Инициализация главного процессора запускает таймер
3. Выполняется цикл вычисления на каждом процессоре, где значением приращения является количество процессоров в системе. Результатом каждой итерации цикла является слагаемое, которое сохраняется в переменную `drobSum`, и суммируется в такими же слагаемыми от других итераций.
4. После завершения цикла каждый процессор прибавляет свое значение переменной `drobSum` к переменной `Result` посредством функции приведения `MPI_Reduce`.
5. После того, как вычисления на всех процессорах будут завершены, главный

процессор останавливает таймер, и получившееся значение переменной `Result` отправляет в поток вывода (вместе со значением времени вычисления, отмеренное таймером, в миллисекундах).

Лекция № 2. Обзор стандартных симуляторов спайковых нейронных сетей.

Использование модельного подхода действительно выглядит многообещающим, так как в современной нейронауке существует круг вопросов, решение которых возможно только с использованием моделирования. Однако существует ряд проблем, связанных с тем, что результаты подобных исследований трудно проверить и воспроизвести. Для проведения компьютерных симуляций разработанной модели исследователю необходимо разрабатывать собственные средства численного интегрирования. Уровень навыков программиста у разных исследователей различен, исследователи могут применять разные численные методы для одних и тех же моделей, причём делая это не всегда корректно [2]. Эти и другие причины зачастую приводят к неправильным результатам и/или невозможности воспроизвести чужие результаты. С целью преодолеть эти проблемы научным сообществом разрабатываются тщательно проверенные специализированные программные продукты для симуляции нейронных сетей. Они являются хорошо документированными и адресованы широкому кругу пользователей. Данные программные средства позволяют стандартизировать код, что в свою очередь существенно упрощает взаимодействие исследовательских групп [3]. Многие из таких нейросимуляторов также содержат встроенные средства для параллельного программирования, обеспечивающие использование арсенала современной информационно-вычислительной техники для задач моделирования активности нейронных сетей мозга [4–8]. Многие из этих симуляторов нашли широкое применение в построении крупномасштабных моделей нейронных сетей. Существует несколько типов симуляторов:

1. Симуляторы с простой моделью нейрона: PCSIM [6], NEST [9], Brain [10] и NCS [11].
2. Симуляторы с моделью нейрона, состоящей из нескольких компартментов: NEURON [12], GENESIS [13], SPLIT и MOOSE [14].
3. Событийно-управляемые симуляторы: MVASPIKE
4. Системы анализа динамических систем: XPP

К наиболее распространённым можно отнести NEST (Neuron Simulation Tool), NEURON.

Наряду с программными средствами моделирования широко применяются симуляторы с аппаратно реализованными нейронными сетями [15–18]. В качестве примера, можно привести симулятор FACETS, который представляет собой платформу, которая моделирует работу примерно 10^6 нейронов, расположенных на нескольких

взаимосвязанных платах, на каждой из которых располагаются аналоговые сетевые ядра (ANCs), являющиеся главным элементом архитектуры FACETS, состоящим из нейронов и синаптических связей (в среднем, каждый нейрон этой системы имеет около 1 тысячи контактов с другими).

NEURON

NEURON представляет собой симулятор, служащий для компьютерного моделирования отдельных нейронов и их сетей с высоким уровнем детализации, включая различные типы ионных каналов мембраны, разбиение клетки на составные части – компартменты – каждая из которых имеет свою собственную динамику. В связи с этим симулятор особенно удобен для исследований, основанных на экспериментальных данных, требующих учёта сложных анатомических и биофизических свойств клетки. NEURON имеет "естественный синтаксис", позволяющий пользователям определять свойства модели знакомыми им средствами, задавая биофизические параметры моделей компартментов, вместо программирования решения дифференциальных уравнений численными методами. В NEURON реализована возможность переключения между несколькими встроенными методами интегрирования, не переписывая при этом модель.

NEURON имеет графический интерфейс, доступный для пользователей с минимальными навыками в области программирования. В составе графического интерфейса есть билдер для однокомпартментных и многокомпартментных клеток, сетей, сетевых ячеек, каналов и линейных электрических цепей. Однокомпартментные и многокомпартментные клетки отличаются друг от друга тем, что многокомпартментные клетки состоят из нескольких отдельных секций, каждая из которых имеет свой набор параметров и свою собственную кинетику. На официальном вебсайте http://www.neuron.yale.edu/ можно найти множество руководств пользователя, включая выбор и настройку базовых моделей в билдере клеток, каналов и сетей. Используя эти билдеры, пользователь может формировать основу для проведения всех симуляций и моделирования.

NEST

NEST – программный симулятор с открытым исходным кодом, используемый для моделирования сетей, биологически реалистичных и феноменологических элементов и

связей. Нейросимулятор оптимизирован для расчёта больших нейрональных сетей, состоящих из сравнительно простых однокомпартментных нейронов. В настоящее время NEST способен симулировать модель, состоящую из 1.73 x 10^9 элементов (нейронов) и приблизительно 10^{13} связей (синапсов) между ними [19].

Схема модели в NEST представлена на рисунке 4 (b). Ее построение осуществляется с использованием узлов и связей. Узлами могут быть нейроны, устройства и подсети, которые могут обмениваться различного вида событиями (токами, спайками). NEST включает в себя встроенные модели нейронов, синапсов, устройств, а модульный принцип организации приложения дает пользователю возможность использования собственных моделей. Большинство реализованных моделей нейронов имеет небольшое количество компартментов, что напрямую отражается на степени детализации описываемых процессов и морфологии клетки. Нейроны объединяются в сеть с помощью синаптических контактов с собственной динамикой. Простейшая модель связи между узлами имеет два параметра — вес (сила взаимодействия между узлами) и время задержки (время перехода сигнала от одного узла к другому). В симуляторе реализованы встроенные механизмы нейронной пластичности, синаптической депрессии и восстановления, а также существует возможность создания различных топологических схем связей и структурирования больших сетей. Общая схема организации приложения представлена на рисунке 3.

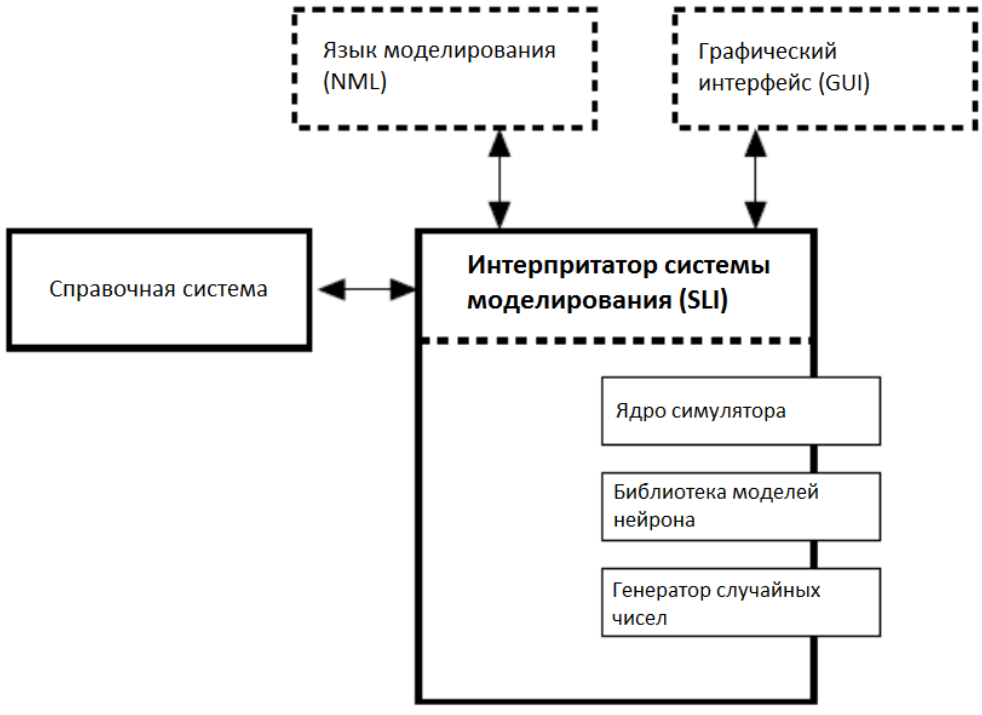

Рисунок 3. Структура системы моделирования.

В нейросимуляторе NEST реализован нисходящий (сверху-вниз) подход к описанию модели нейронной сети. В соответствии с принципом иерархичности, моделируемые нейронные сети рассматриваются как структуры с многоуровневой организацией, которые могут быть удобно представлены в виде графов. На рисунке 4 (a) проиллюстрирован пример структурированной модели нейронной сети, подразделенной на структуры, описывающие различные уровни организации системы: сетчатку и две модельные области мозга, V1 и V2.

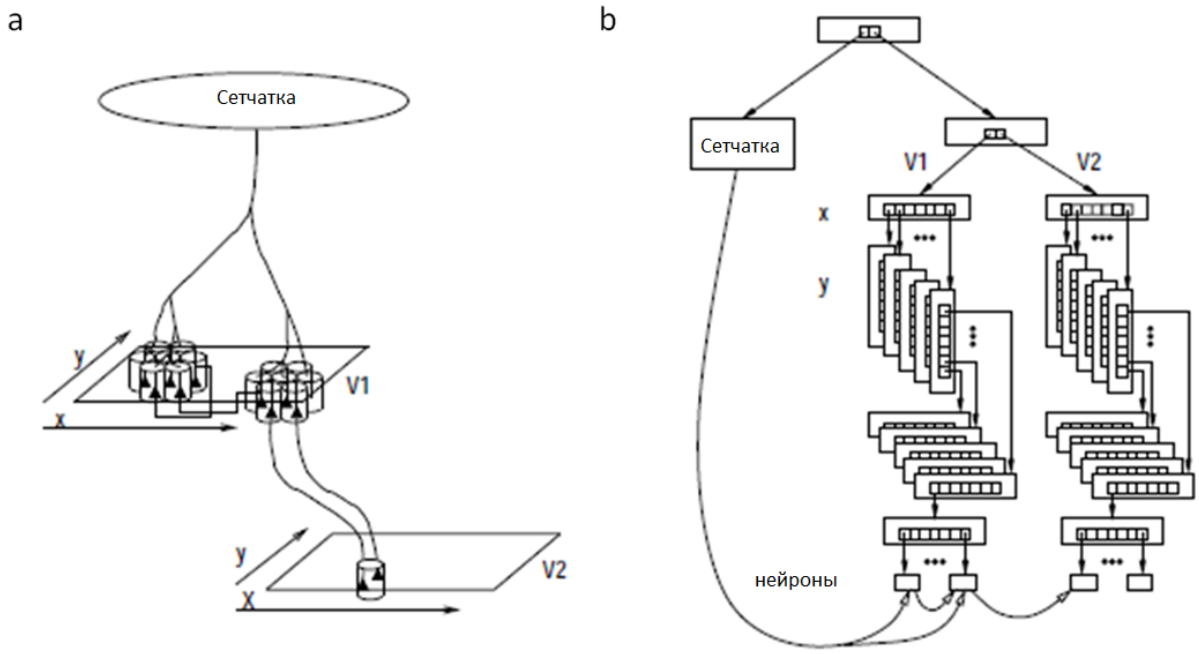

Рисунок 4 Пример структурированной модели нейронной сети (a) и ее представление (b).

В системе можно выделить ядро и интерпретатор системы моделирования, через который осуществляется взаимодействие графического интерфейса и языка моделирования с ядром системы.

NEST может быть использован как самостоятельное приложение, запускаемое нативно из операционной системы семейства Linux. При этом команды нейросимулятора вводятся либо в режиме командной строки, либо в виде последовательности команд, записанных в файл. При этом нейросимулятор использует свой собственный скриптовый язык, который носи название SLI. Другой возможностью использования NEST является вызов команд нейросимулятора из-под интерпретатора языка программирования python.

Для этих целей разработчиками был добавлен специальный модуль PyNEST [20,21]. Приведём здесь некоторые команды из основного набора синтаксиса PyNEST.

MUSIC

Кроме обычных нейросимуляторов, созданных для специфических задач, и симуляторов широкого профиля существуют мультисимуляторные стандарты и технологии, позволяющие использование нескольких симуляторов в рамках одной модели или обеспечивающие независимость реализации модели и её расчёта от конкретного симулятора. Среди таких систем стоит отметить MUSIC (https://www.incf.org/documents/program-documents/Music-UsersManual.pdf) и PyNN. MUSIC - специализированная программная система, обеспечивающая взаимодействие нескольких сред симуляции, необходимых для построения компьютерных моделей с высокой степенью детализации.

MUSIC (MUlti-SImulation Coordinator) представляет собой программный интерфейс (набор API), основанный на средствах мгновенных сообщений (MPI), с помощью которых симуляторы, параллельно запущенные на вычислительном кластере, могут обмениваться информацией во время выполнения моделирования. Среда разрабатывалась для осуществления связи симуляторов крупномасштабных нейронных сетей, или друг с другом, что позволяет моделям, построенных с использованием различных средств моделирования, рассчитывать эволюцию единой большой системы, или с другими инструментами. Особый акцент был сделан на проблеме совместимости с существующими симуляторами и их взаимодействию. С помощью MUSIC может быть реализована скоординированная работа приложений с разными временными шагами и принципами распределения данных с помощью механизмов обмена информацией о событиях и наборе параметров состояния моделируемой системы. Пример такой работы представлен на рисунке 5, где три приложения, выполняемые параллельно, обмениваются данными с использованием MUSIC.

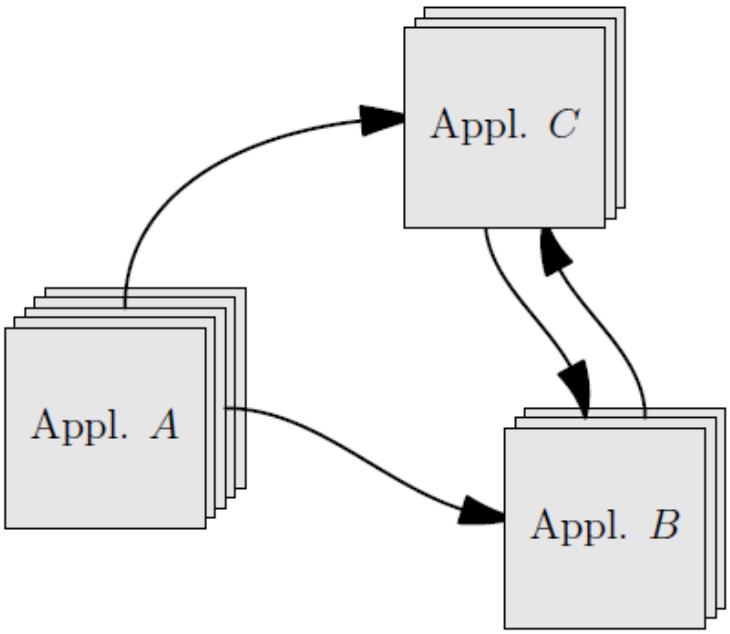

Рисунок 5. Схема использования системы MUSIC

Здесь приложение A генерирует данные, которые впоследствии используются приложениями B и C. Кроме того, B и C обмениваются информацией друг с другом. Данные, передаваемые этими приложениями, могут быть либо событием, таким как спайк, либо переменной состояния, например мембранный потенциал.

PyNN

Другим примером специализированного средства симуляции, позволяющим работать со свободно переносимыми компьютерными моделями, является PyNN [22,23] (произносится как «пайн»). В нем реализована возможность, написав скрипт модели один раз, используя язык программирования python, запускать его без модификаций на многих симуляторах (включая NEURON, NEST, PCSIM, Brian).

В качестве основных свойств PyNN можно указать следующие:

- Поддержка функций, реализованных в симуляторах нейронных сетей.
- Возможность написания программного кода модели, способного исполняться на любых поддерживаемых симуляторах без модификаций.
- Поддержка высокоуровневой абстракции.
- Портируемость моделей между симуляторами.
- Приоритет совместимости над оптимизацией.
- Параллельность (использование MPI).

Примеры использования PyNN:

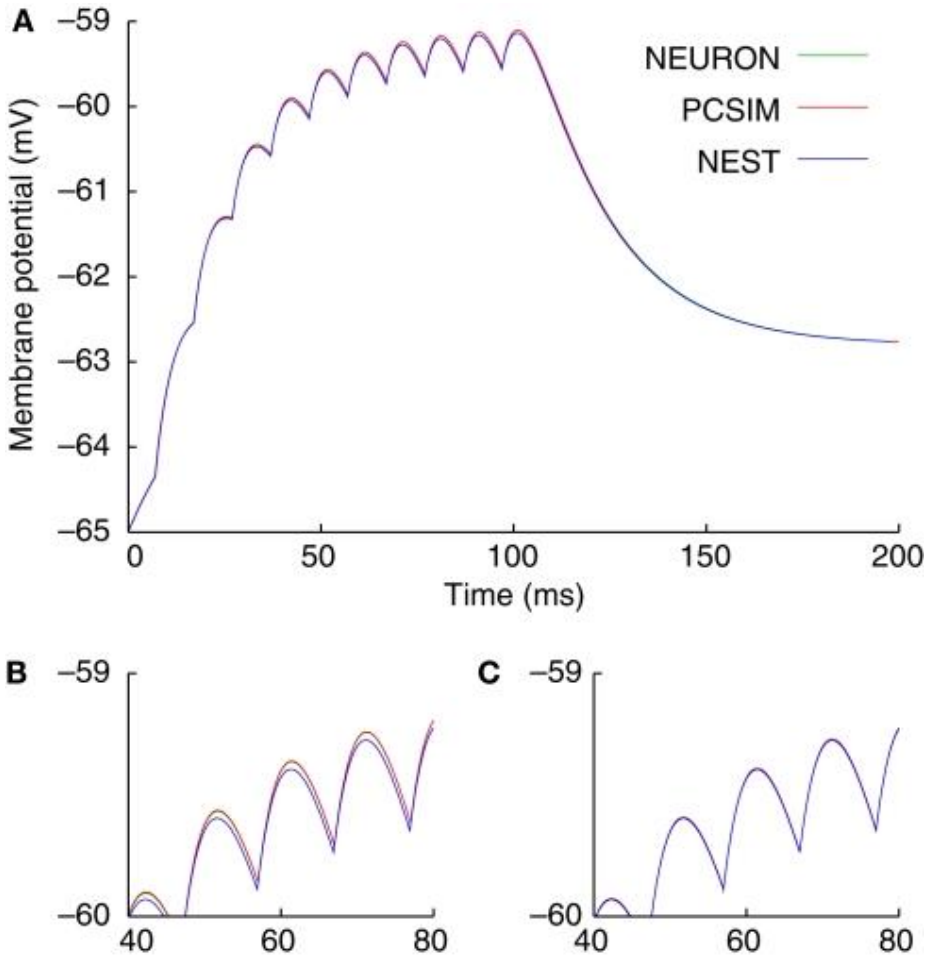

Рисунок 6. Пример использования NEURON, NEST и PCSIM. (A) Мембранный потенциал с временным шагом 0.1 мс. (B) Увеличена небольшая область, показывающая небольшие цифровые различия между результатами разных симуляторов. (C) Результаты симуляторов с интеграционным временным шагом 0.01 мс, показывающие исчезновения цифровых различий.

Для сравнения результатов симуляции с применением независимого языка для симуляции PyNN были использованы три различных симулятора, функции которых вызывались через один и тот же скрипт, содержащий реализацию модели. Рисунок 6 иллюстрирует воспроизводимость результатов численных расчётов на нескольких различных нейросимуляторах - NEURON, NEST и PCSIM. Показаны фрагменты осциллограмм мембранного потенциала нейрона. Видно, что результаты симуляции могут незначительно отличаться друг от друга при использовании шага интегрирования 0.1. Это связано с отличиями в используемых методах решения дифференциальных уравнений или в разных реализациях одного и того же метода разными симуляторами. Различия исчезают при уменьшении шага интегрирования, что подтверждает корректность использования всех нейросимуляторов и их взаимозаменяемость.

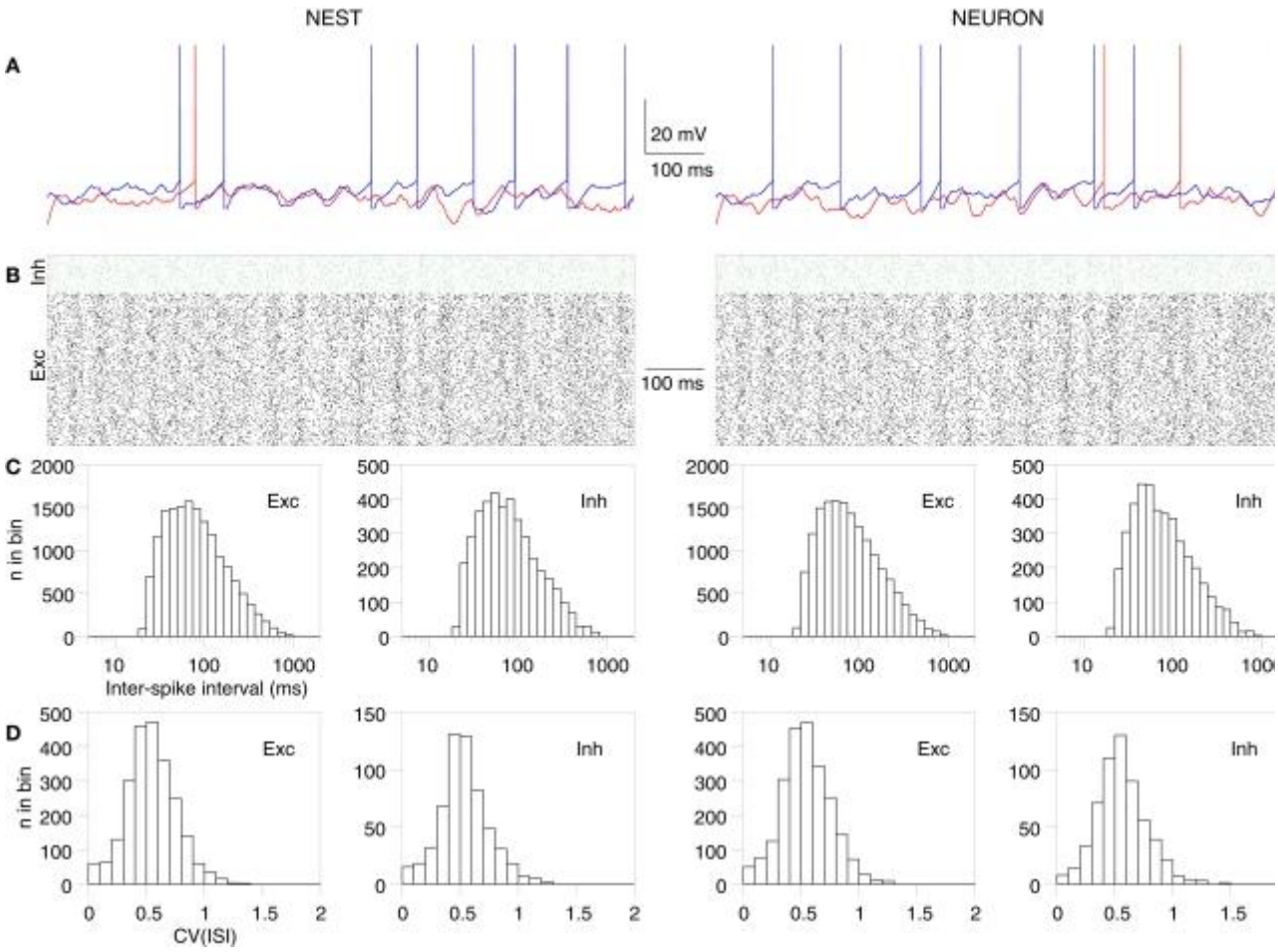

Рисунок 7. Пример использования NEURON и NEST. (A) Мембранные потенциал для двух возбуждающих нейронов. Результаты NEST и NEURON схожи первые 50 мс, но после отличны друг от друга из-за различия сетевой активности. (B) Спайковая активность возбуждающих (черный) и тормозных (зеленый) нейронов. (C) Распределение интерспайковых интервалов (ISIs) возбуждающих и тормозных нейронов. (D) Распределение нейронов по коэффициенту вариации ISI.

Другой пример использования языка, независимого от симулятора, демонстрирует более существенные различия в динамике мембранных потенциалов, как показано на рисунке 7. Результаты расчётов, сделанных при помощи NEST и NEURON схожи первые 50 мс, но после отличны друг от друга из-за различия сетевой активности. Данные различия проявляются из-за того, что при моделировании сложных высоко размерных систем, какими являются нейронные сети, результат сильно зависит от способа решения уравнений, начальных условий и многих других факторов. Это связано с расхождением фазовых траекторий и хаотической динамикой. Однако, как видно из рисунка 7 (C) и (D)7, даже при сложной сетевой динамике крупномасштабных моделей разные симуляторы демонстрируют статистически схожие результаты расчётов.

Рассмотренные нейросимуляторы успешно применялись для написания и реализации математических моделей и проведения вычислительных экспериментов с использованием параллельных вычислений для симуляции крупномасштабных моделей. Некоторые примеры подобных задач будут рассмотрены в следующей лекции.

Наиболее распространёнными и широко применяемыми нейросимуляторами являются NEST и NEURON.

Лекция № 3. Примеры задач, решаемых с использованием параллельных вычислений на стандартных симуляторах

Рассмотрим примеры решения задач крупномасштабного моделирования с использованием стандартных нейросимуляторов с технологией распараллеливания вычислений.

Параллельные вычисления с применением нейросимулятора NEST.

Один из наиболее широко используемых нейросимуляторов, NEST, обладает встроенной возможностью распараллеливать вычисления и расчёты динамики моделей спайковых нейронных сетей.

При моделировании на вычислительном кластере или многопроцессорных компьютерах, каждый компьютер или процессор воссоздает часть сети и хранит информацию о синаптических контактах только своих нейронов. Для распределения задач на все компьютеры (процессоры) применяется интерфейс мгновенных сообщений (MPI) и потоков POSIX (pthreads). Использование потоков позволяет воспользоваться преимуществом многоядерных и многопроцессорных компьютеров без привлечения дополнительного программного обеспечения. Использование распределенных вычислений имеет дополнительную выгоду в том, что позволяет проводить более масштабное моделирование, чем могло бы позволить использование памяти одного компьютера. Кроме того, общее время соединения и время выполнения моделирования могут быть уменьшены, поскольку сеть основана на множестве компьютеров, соединенных параллельно.

Чтобы упростить обработку распределения узлов потоков и процесса, основанного на распараллеливании, используется концепция локальных и удаленных потоков, называемых виртуальными процессами. Виртуальный процесс (VP) – это поток, находящийся в одном из процессов MPI. Виртуальные процессы распределены согласно модульно-ориентированному алгоритму в MPI процессах и непрерывно подсчитываются в течение всех процессов. Общее представление представлено на рисунке 8.

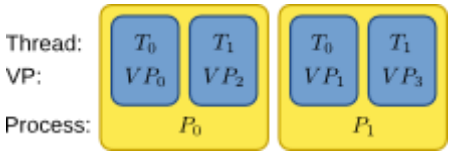

Рисунок 8. Общая схема подсчета потоков (T), виртуальных процессов (VP) и процессов MPI (P) в NEST

Алгоритм организации параллельных вычислений в NEST иллюстрирует следующий пример на псевдокоде:

```
t <- 0
  WHILE t < T_stop
    PARALLEL on all virtual processes
      deliver all events due
      call U(S_t) forall nodes
    END
    exchange events between all virtual processes
    increment time t <- t+Delta
  END
```

Время симуляции разбивается на дискретную сетку с определённым шагом $t_i := i \cdot \Delta$. Запускается цикл, пробегающий все шаги по этой временной сетке. Внутри цикла происходит распараллеливание вычислений между всеми виртуальными процессами.

Статус справочника каждого узла содержит три входа, которые связаны с параллельным вычислением:

- `local`: логический, который показывает, существует ли узел на локальном процессе или нет

- `thread`: идентификатор локального потока, к которому привязан узел

- `vp`: идентификатор виртуального процесса, к которому привязан узел.

Распределение узлов зависит от их типа. Нейроны привязываются автоматически к одному из виртуальных процессов также по модульно-заданному алгоритму. На всех других виртуальных процессах выделяется прокси узел. Присвоение происходит через $id_{VP} = id_{Node} \bmod N_{VP}$, где id_{VP} идентификатор виртуального процесса, к которому привязан нейрон, id_{Node} глобальный идентификатор узла, а N_{VP} общее число виртуальных процессов.

Устройства для моделирования и наблюдения за сетью повторяются в каждом потоке таким образом, что позволяет уравновесить нагрузку на разные потоки и минимизировать взаимодействие между различными потоками.

Привязывание узлов к виртуальным процессам по умолчанию – удобный выбор в плане соблюдения баланса загрузки процессов, если это соответствует случайному распределению для сетей со случайными связями. Однако, для сетей с известной архитектурой и определённым направлением потока передачи сигналов, всегда существуют лучшие распределения. Для осуществления более мелкомодульного распределения, модель subnet имеет логический параметр `children_on_same_vp`, который вызывает/активирует/принуждает все узлы в подсети быть привязанными к тому же виртуальному процессу.

Узловое распределение для малой сети состоит из `spike_generator`, четырех `iaf_neurons`, и `spike_detector` в примере с двумя процессами с двумя потоками, каждый показан в следующем рисунке 9.

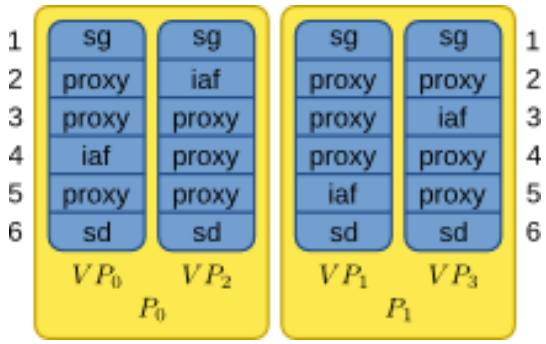

Рисунок 9. Иллюстрация узлового распределения, где sg - spike_generator, iaf – iaf_neuron, sd – spike_detector. Числа слева и справа демонстрируют глобальные идентификаторы

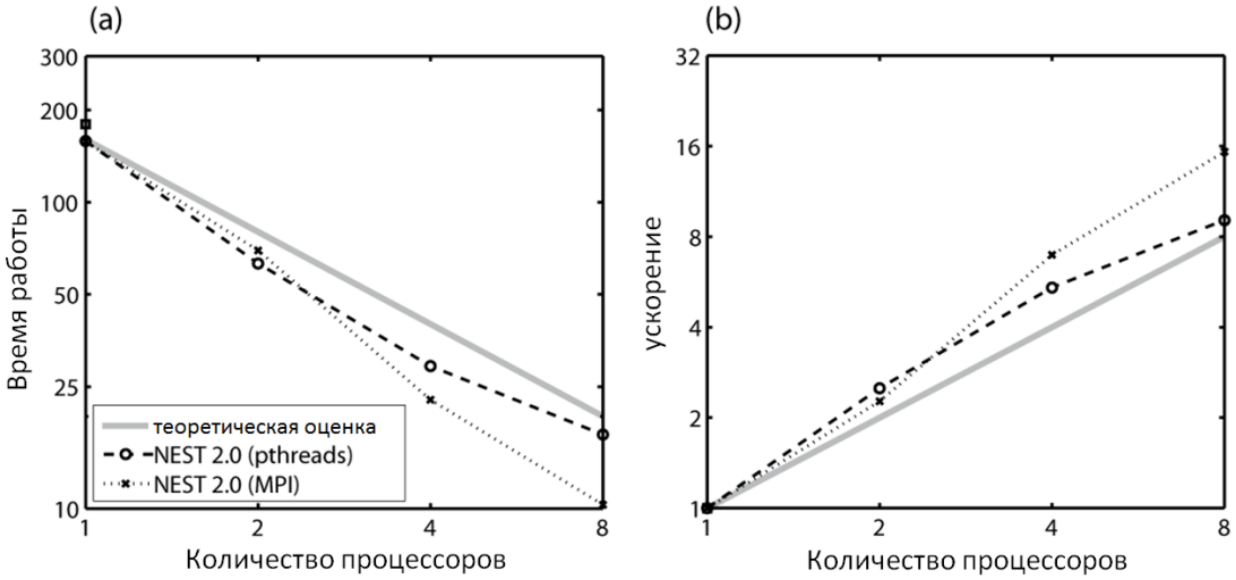

Рисунок 10. Моделирование нейронной сети в NEST. (a) абсолютное время работы и (b) ускорение как функция от количества задействованных процессоров.

На рисунке 10 представлены графики, показывающие зависимость производительности NEST от количества задействованных процессоров. Рассчитывается сеть из 12500 элементов, описываемая моделью integrate-and-fire (80 % возбуждающих, 20 % тормозных нейронов), на вход каждого из которых поступает сигнал от 10% всех нейронов. Общее количество синапсов в модели равняется $1{,}56\times10^{7}$. Нейроны

инициализируются случайными мембранными потенциалами и в дальнейшем стимулируются нерегулярным воздействием.

Можно привести следующие примеры использования NEST:

Модели обработки сенсорной информации (зрительной, слуховой и т.д.)

Модели динамики нейросетевой активности

Обучение и пластичность в моделях обработки сенсорной информации.

Модели спайковой синхронизации в сетях Synfire Chains.

На рисунке 11 представлены результаты исследования спайковой синхронизации в сетях Synfire Chains, с использованием симулятора NEST.

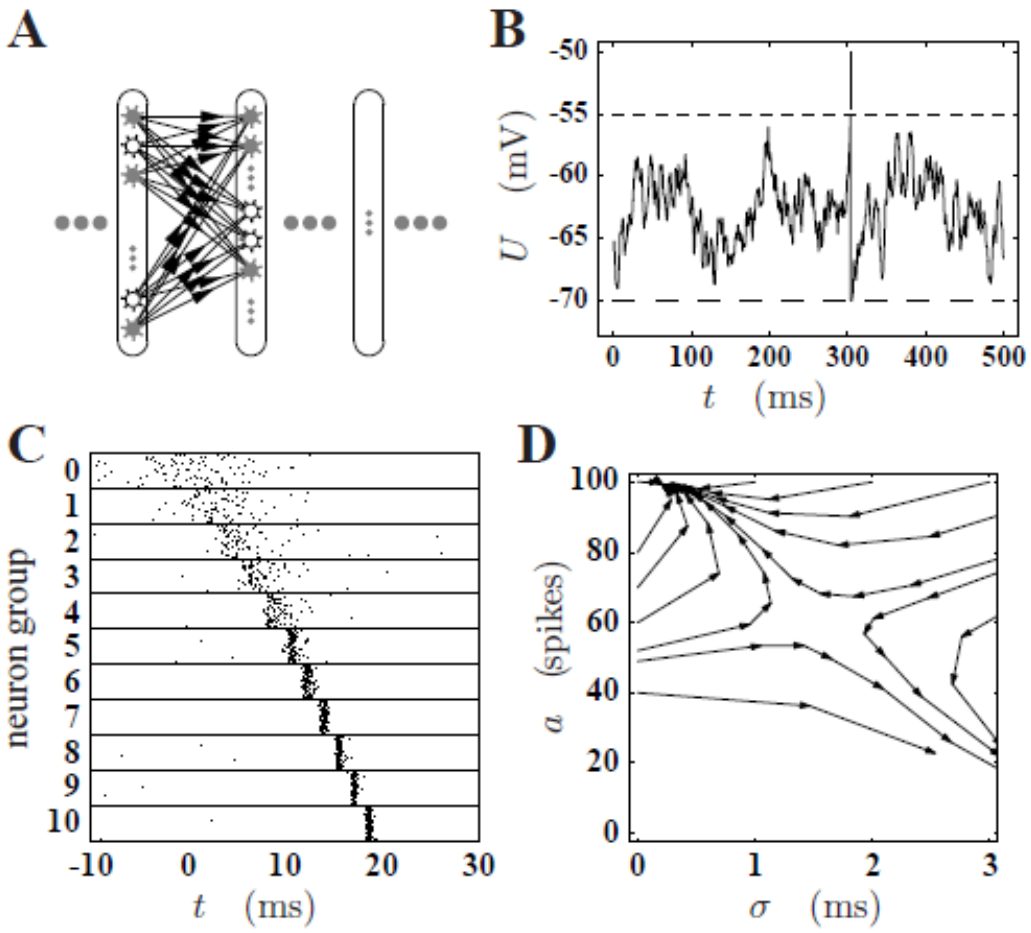

Рисунок 11 (A) Показана схема нейронной сети, известной как *synfire chain.* Группы нейронов соединены в цепочечную структуру. Каждый нейрон слоя образует контакты (большинство из них возбуждающие) со всеми нейронами следующего слоя, и принимает сигнал от всех нейронов предыдущего слоя. (B) Колебания мембранного потенциала отдельного нейрона, полученные с помощью симуляции. (C) Показан результат нейросетевого моделирования, где первая группа цепи стимулирована широкой пачкой

входящих спайков. В последующем наблюдается явление синхронизации спайков. (D) Фазовый портрет синхронизации в synfire chain.

Параллельные вычисления с применением нейросимулятора NEURON

Рассмотрим примеры параллельных вычислений с применением нейросимулятора NEURON [5]. В NEURON предусмотрена поддержка следующих типов параллельных вычислений:

1. Несколько симуляций распределяются между всеми процессорами, каждый процессор выполняет свою собственную симуляцию.
2. Симуляции распределённых нейронных сетей.
3. Симуляции распределённых одиночных клеток (каждый процессор обрабатывает часть клетки).

Было проведено исследование [4] возможностей параллельных вычислений в нейросимуляторе NEURON с использованием следующей вычислительной техники:

2 процессора 2 GHz PowerMac G5, with 512 KB L2 cache for each processor

12 64-bit процессор Beowulf cluster, 3.2 GHz Intel Xeon with 1024 KB cache

25 32-bit процессор Beowulf cluster, 2.4 GHz Intel Xeon with 512 KB cache

1024 процессор CINECA IBM Linux cluster, 2.8 GHz Xeon with 512 KB cache

8196 процессор EPFL IBM Blue Gene.

Полученные результаты представлены на рисунке 12:

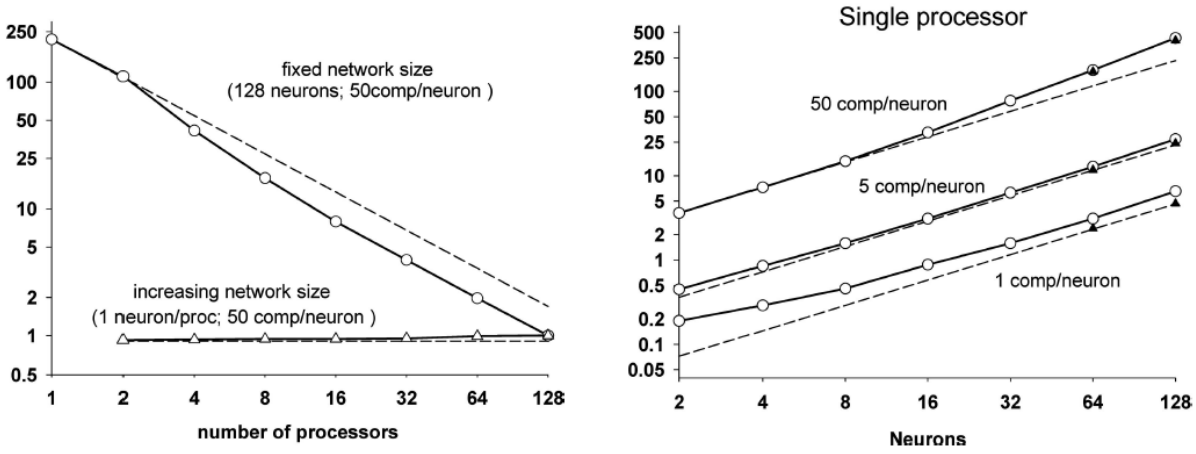

Рисунок 12. Производительность параллельного вычисления. (**A**) Время выполнения вычислений как функция количества процессоров. Прерывистая линия отображает идеальную ситуацию для каждого случая. Круги – время расчета сети из 128 нейронов, 50 компартментов/нейрон (comp/neuron), на различном количестве процессоров. Треугольники – время расчета сети с увеличивающимся количеством нейронов на

увеличивающемся количестве процессоров (1 нейрон/процессор). (**B**) Время расчета на одном процессоре в зависимости от количества нейронов в сети (круги).

Как видно из результатов симуляции, NEURON обладает хорошей масштабируемостью. Для различных способов распараллеливания результаты прироста производительности близки к идеальному случаю полной независимости данных между потоками. При этом увеличение размера сети также приводит к линейному росту времени расчёта.

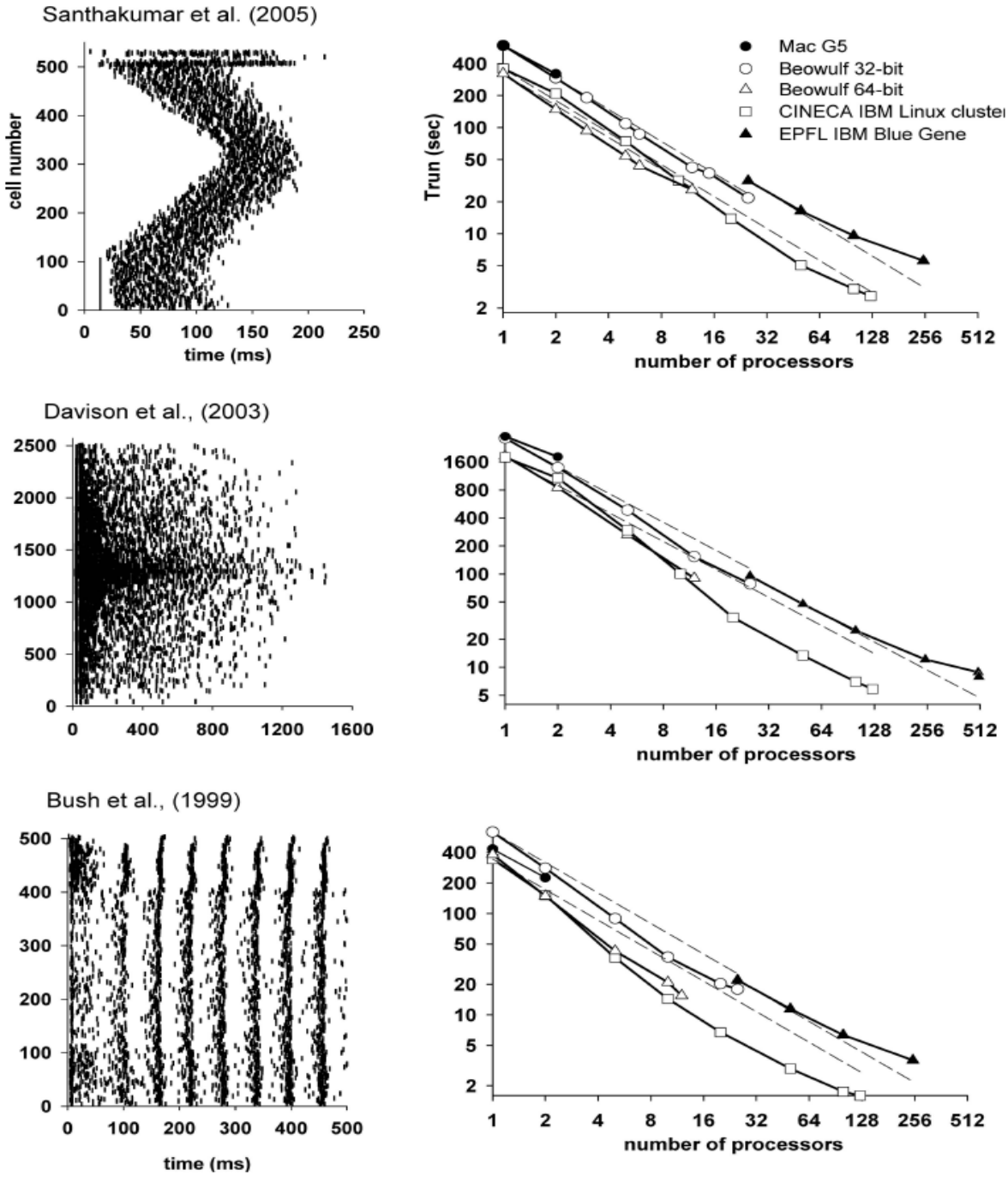

Рисунок 13. Параллельные вычисления сетевых моделей. Слева - растровый портрет каждой модели; справа – время расчета для каждой модели на различных параллельных системах. Штриховые линии во всех случаях соответствуют идеальному случаю.

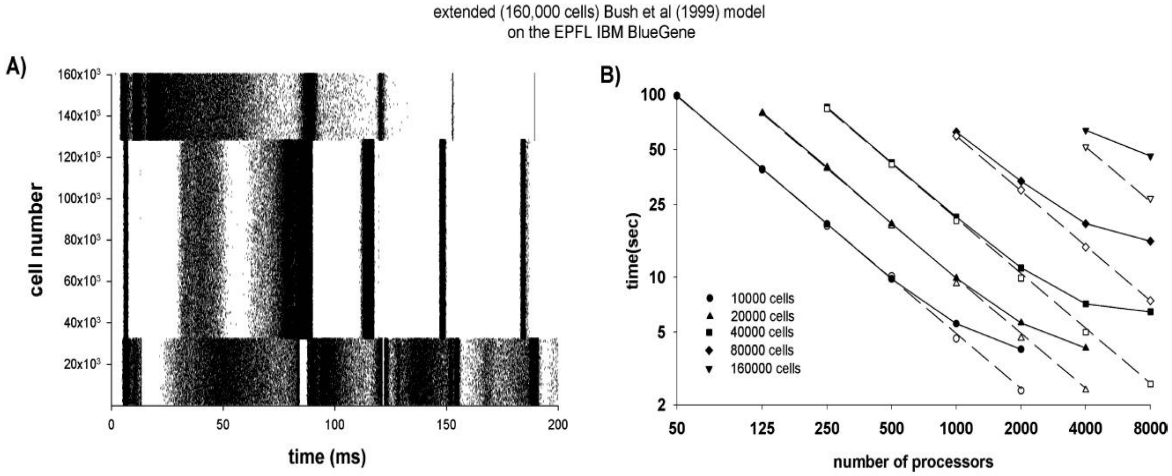

Рисунок 14. Симуляция большой сети (160000 клеток) на EPFL IBM BlueGene. (A) Растровый портрет параллельного расчета расширенной сетевой модели Bush et al (1999). (B) Время работы (закрашенные символы) и время процессорных вычислений (пустые символы) как функция количества процессоров для различного количества элементов в модели сети. Во всех случаях время симуляции 200 мс. Прерывистыми линиями отмечен идеальный случай.

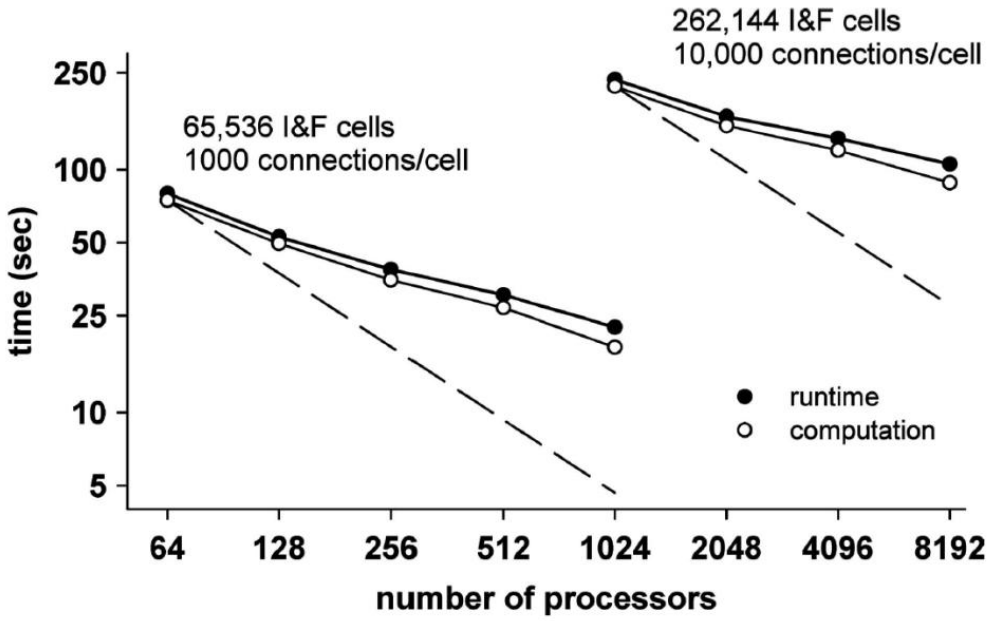

Рисунок 15. Время работы (закрашенные символы) в зависимости от количества процессоров для 65 536 и 262 144 Integrate-and-Fire клеток (I&F) с 1000 и 10000

связей/клетка соответственно.

Другие примеры удачного распараллеливания вычислений на нейросимуляторе NEURON приведены на рисунках 13 – 15. Во всех случаях применение параллельных вычислений давало близкий к линейному прирост в производительности. В основе этого результата лежит подход к организации высокопроизводительных вычислений для задач моделирования активности нейронных сетей мозга. Дело в том, что нейронная сеть состоит из отдельных нейронов, каждый из которых может быть присвоен отдельному процессу. Из общих соображений следует, что прирост в производительности можно наблюдать в случае, если обмен информацией между отдельными потоками занимает существенно меньше расчётного времени по сравнению с независимыми вычислениями, осуществляющимися отдельно в каждом потоке. Данное утверждение справедливо и для некоторого класса моделей нейронных сетей. При этом учитывается тот факт, что хотя и активность отдельных нейронов зависит от активности остальной сети, эта зависимость может быть легко замоделирована обменом мелкими событиями между потоками, не вовлекая при этом затратных пересылок и длительных обращений к памяти.

Лекция № 4. Примеры задач вычислительной нейронауки, решаемых с применением параллельных вычислений.

На данном этапе развития ни одну сферу человеческой деятельности невозможно представить без широкого использования передовых компьютерных технологий. Благодаря непрерывно совершенствующимся информационно-вычислительным технологиям, на сегодняшний день появляется возможность решать такие сложные задачи, как компьютерное моделирование больших систем, что еще некоторое время назад не представлялось возможным. С помощью моделирования строятся и анализируются модели реальных систем, исследуются протекающие в них процессы, объясняются присущие им закономерности, а также предсказываются интересующие исследователя феномены. Модельный подход не имеет тех ограничений и трудностей, что зачастую присущи лабораторному эксперименту, и поэтому имеет широкое распространение в естественнонаучных исследованиях. Разумеется, и нейронаука, главной задачей которой является исследование функционирования головного мозга и нервной системы в целом, широко пользуется методом компьютерного моделирования. Следует отметить, что в настоящее время исследовательские центры вычислительной нейронауки привлекают существенные инвестиции со стороны правительства и частных фондов, особенно в США (Центр Слоан-Шварц вычислительной нейронауки в Гарварде, Принстоне, Колумбии, Универитете Нью-Йорка и других ведущих университетов) и Германии (Центр Бернштейна вычислительной нейронауки, располагающийся в Берлине, Мюнхене, Геттингеме и Гейдельберге). Следует также указать ведущие мировые проекты по крупномасштабному моделированию структур и функций нейронных систем мозга, такие как The Human Brain Project, The Brain Corporation, SyNAPSE, The BRAIN Initiative. В силу сложнейшей внутренней архитектуры, большого количества элементов и связей между ними, моделирование системы, подобной головному мозгу, представляется крайне затруднительным ввиду большой вычислительной емкости получаемых данных. Однако применение суперкомпьютерных технологий с успехом решает эту проблему. Одним из примеров может служить крупномасштабное моделирование (от англ. *large-scale modelling*). Крупномасштабное моделирование – одно из направлений суперкомпьютерного моделирования, возникшее сравнительно недавно благодаря прогрессу в развитии области изготовления микросхем, параллельных вычислений и возросшей вычислительной мощности суперкомпьютерных систем. Целью крупномасштабного моделирования подхода является разработка и проведение численных экспериментов с глобальными компьютерными моделями больших систем, в которых

интегрированные микро и макро модели воспроизводят взаимосвязанное функционирование многоуровневых систем. В основе данного метода лежит принцип иерархической редукции, основанного на том, что любая сложная система представляет собой совокупность иерархически подчиненных подсистем с более низким уровнем организации. Согласно этому принципу мозг можно представить в виде нескольких взаимодействующих независимо описываемых подсистем. За счет увеличения или уменьшения числа уровней организации достигается требуемый в рамках исследования уровень детализации. Однако с другой стороны с увеличением числа уровней организации возрастает и количество параметров, необходимых для описания системы, что усложняет создание реалистичной модели. Реалистичность отражает способность модели описывать процессы, наблюдаемые в лабораторном эксперименте. Возрастание числа параметров модели ведет к увеличению объема входных данных, которые зачастую не обладают достаточным уровнем точности. Для решения этой проблемы используется метод абстрагирования, который позволяет маловажные параметры без потери удобства манипулирования.

Метод компьютерного моделирования в области исследований мозга млекопитающих является на сегодняшний день одним из ведущих и наиболее перспективных.

Можно выделить следующие наиболее значимые модели:

4. Модель визуального внимания [25]. Она базируется на модулях, которые связаны между собой различными областями дорсальных и вентральных путей визуальной коры.
5. Модель II/III слоев неокортекса [26]. Она соответствует коре головного мозга мелкого млекопитающего, состоит из 22 миллионов нейронов, 11 миллиардов синапсов (была реализована на Blue Gene/L суперкомпьютере).
6. Самоподдерживающаяся нерегулярная активность в крупномасштабной модели гиппокампальной области [27]. Данная модель соответствует анатомии мозга крысы, насчитывает 16 типов нейронов в количестве 200000 единиц. Исследуется активность сети в зависимости от изменения размеров или связей модели.
7. Проекты Blue Brain и The Human Brain Project [28].
8. Модель таламокортикальной системы млекопитающего [29].

Последние два проекта рассмотрим более подробно.

Blue Brain и The Human Brain Project

Проект Blue Brain [28,30] является результатом совместной работы Швейцарского Федерального Технического Института Лозанны (École Polytechnique Fédérale de Lausanne - EPFL) и корпорации IBM. Цель проекта - детальное моделирование отдельных нейронов и образуемых ими неокортикальных колонок. Неокортекс находится в верхнем слое полушарий мозга и отвечает за такие высшие нервные функции, как пространственная ориентация, сенсорное восприятие, моторные реакции, осознанное мышление и речь (у людей). Структурно он состоит из шести горизонтальных слоев нейронов, различных по типу и характеру связей. Вертикально нейроны образуют так называемые колонки кортекса. Именно их функционирование лежит в основе когнитивной и сенсорной обработки информации в мозге. Каждая колонка насчитывает порядка 10 тысяч нейронов, строго упорядоченных как внутри колонки, так и по отношению к внешним группам нейронов.

За несколько десятилетий исследования нервных клеток было накоплено достаточно данных для построения биологически релевантной модели. По этим данным на суперкомпьютере моделировалась пространственная ориентация разичных типов клеток в пространстве, их детальная морфология и архитектура межклеточных взаимодействий. По большим наборам экспериментальных данных с помощью метода автоматической корректировки параметров компартментных моделей нейронов рассчитывалась эволюция модели с использованием симулятора NEURON. В модели нейрона брались в расчет различия между типами нейронов, пространственная геометрия нейронов, распределение ионных каналов по поверхности мембраны клетки, что в совокупности определяет упорядоченное распространение сетевой нейронной активности. Модель реализована на суперкомпьютере Blue Gene/L (рисунок 16), позволяющем рассчитывать распространение электрической активности внутри неокортикальной колонки в режиме реального времени.

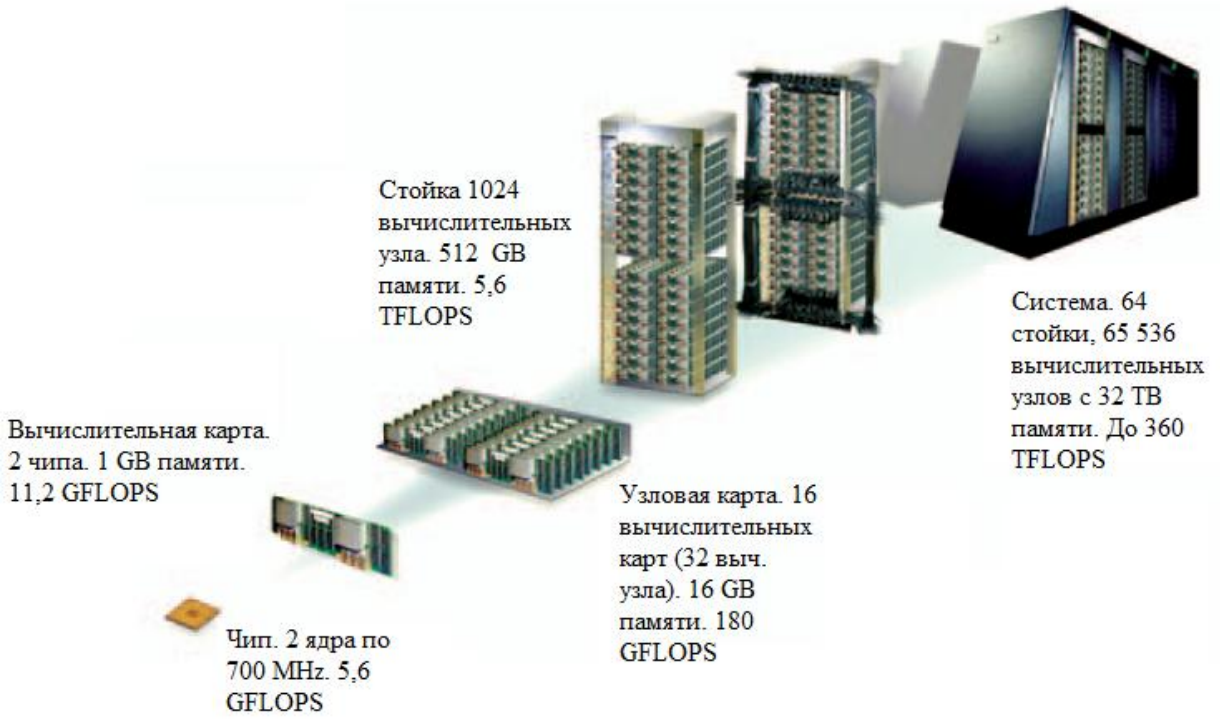

Рисунок 16. Схематическая архитектура суперкомпьютера Blue Gene/L

Особенностью суперкомпьютера Blue Gene/L является использование технологии целой системы на чипе, в которой все функции вычислительного узла (за исключением главной памяти) сосредоточены в одной интегральной схеме специализированного применения (англ. ASIC — application-specific integrated circuit). Каждый чип Blue Gene/L состоит из двух процессорных ядер PowerPC 400 с тактовой частотой 700 МГц. 64-битовый сопроцессор, совмещённый с каждым ядром, способен работать в режиме «одна инструкция, много данных» (англ. SIMD — single instruction, multiple data). Каждый сопроцессор может выполнять до двух операций умножения-сложения за раз. Таким образом, пиковая производительность одного чипа может достигать 8 операций с плавающей точкой за один цикл или 5.6 GFLOPS. Максимальная теоретическая производительность всего суперкомпьютера составляет 360 TFLOPS.

В рамках исследования предусматривается несколько этапов. В ходе первого из них была получена модель одной колонки неокортекса молодой крысы (конец 2006г). Для моделирования 10000 нейронов и 3×10^7 синапсов было задействовано 8192 процессора. По завершении первого этапа проекта (26 ноября 2007 года) были достигнуты следующие результаты: автоматическая генерация нейронной сети по предоставленным биологическим данным, автоматическая систематическая проверка и калибровка модели перед каждым релизом для более точного соответствия биологической природе, построена первая модель колонки неокортекса клеточного уровня исключительно по биологическим

данным. На данный момент проект дал начало одному из двух флагманских проектов Европейской комиссии, The Human Brain Project (HBP), удачное завершение которого, по мнению разработчиков, даст начало новой парадигме вычислительной архитектуры с многочисленными практическими приложениями во всех областях человеческой деятельности.

The Human Brain Project

The Human Brain Project включает в себя двенадцать подпроектов, кратко описанные ниже:

1. **SP 1** – Strategic Mouse Brain Data -основная задача – получение обширной информации для дополнения существующих данных о структуре мозга мышей и облегчения сопоставления их с человеческим мозгом.
2. **SP 2** – Strategic Human Brain Data – в этом подпроекте задачей является сбор данных о функционировании человеческого мозга в рамках корреляции их с подобными данными по мозгу мышей, а также обработка данных подпроекта SP1 для предсказания результатов исследуемых процессов. Таким образом, этот проект является своеобразным буфером между микроскопическим уровнем исследований проекта SP1 и макроскопическими исследованиями (например, МРТ).
3. **SP 3** – Cognitive Architectures **–** целью этого подпроекта является отбор строго очерченных когнитивных задач, уже частично исследованных в рамках когнитивной нейронауки (перцептивное восприятие, мотивация, принятие решений, обучение и память), для применения стантизированных протоколов стимуляции и для анализа связанных паттернов мозговой активности и динамики отклика.
4. **SP 4 –** Mathematical and Theoretical Foundations of Brain Research **–** задача подпроекта – разработка единой математической базы, которая смогла бы консолидировать разрозненные данные о структурах мозга на разных уровнях.
5. **SP 5 –** Neuroinformatics **–**один из основных подпроектов HBP, целью которого является разработка новых инструментов для конструирования многоуровневых атласов мозга, а также для анализа и интерпретации больших объемов структурных и функциональных данных.
6. **SP6 –** Brain Simulation **–** создание и разработка симуляторов, которые, основываясь на экспериментальных биологических данных, моделируют поведение отдельных участков головного мозга и, в будущем, целого мозга.
7. **SP7 -** High Performance Computing **-** обеспечение широкого доступа к

суперкомпьютерам, базам данных и облачным технологиям для сообщества нейроисследователей, а также поддержка в компьютерном обеспечении, интерактивном управлении вычислительными процессами и визуализации в рамках создания и симулирования многоуровневых многоуровневых моделей мозга.

8. **SP8** – Medical Informatics **–** имеет три объекта исследований. Первый – создать инструменты для интеграции клинических данных, включая генетические данные и имаджинг. , второй – внедрить проект в больницы и клиники для широкого использования, третий – разработать инструменты, которые сделают возможным извлечение уникальных биологических особенностей болезней из многоуровневых данных для пациентов, страдающих от нескольких заболеваний, а такжетакие инструменты, которые сделают возможным создание новой, всеобъемлющей классиффикации болезней мозга, основанной на настроенных комбинациях биологических признаков.
9. **SP9** – Neuromorphic Computing **–** разработка, реализация и управление нейроморфной компьютерной платформой, которая позволит нейробиологам и инженерам ставить эксперименты с помощью нейросимуляторов, разрабатываемых в подпроекте SP6.
10. **SP10** – Neurorobotics **-** этот подпроект ставит своей задачей предоставить учёным и разработчикам программное и аппаратное обеспечение, которое позволит им соединить апробированные модели мозга с детализированными симуляторами роботов, и использовать конечные нейророботические системы в экспериментах *in silico.*
11. **SP11 –** Applications **-** этот подпроект представляет собой первый шаг к достижению амбициозных целей HBP на этапе эксплуатации, а именно разработка концепций в трех основных областях –нейронауки будущего, нейромедицины будущего и вычислительной техники будущего.
12. **SP12** – Ethics and Society **–** целью этой программы станет исследование социальных, этических и философских аспектов проекта, содействие в организации встреч с руководящими лицами проекта и освещение его деятельности в средствах массовой информации, воспитывание доверия у широкой публики к исследованиям и инновациям путем повышения социальной и этической осведомленности среди участников проекта.

Компьютерная модель мозга Ижикевича-Эдельмана.

Модель воспроизводит поведенческие режимы, характерные для обычной мозговой активности, возникающие самопроизвольно как результат взаимодействий между анатомическими и динамическими процессами. Описывает спонтанную активность, чувствительность к изменениям в отдельных нейронах, появлении волн и ритмов, и функциональной возможности соединения в различных весах.

Проект стартовал в 2007 году. Авторы проекта для обсчета высоко детализированной модели коры мозга млекопитающих [29] использовали вычислительный кластер Beowulf.

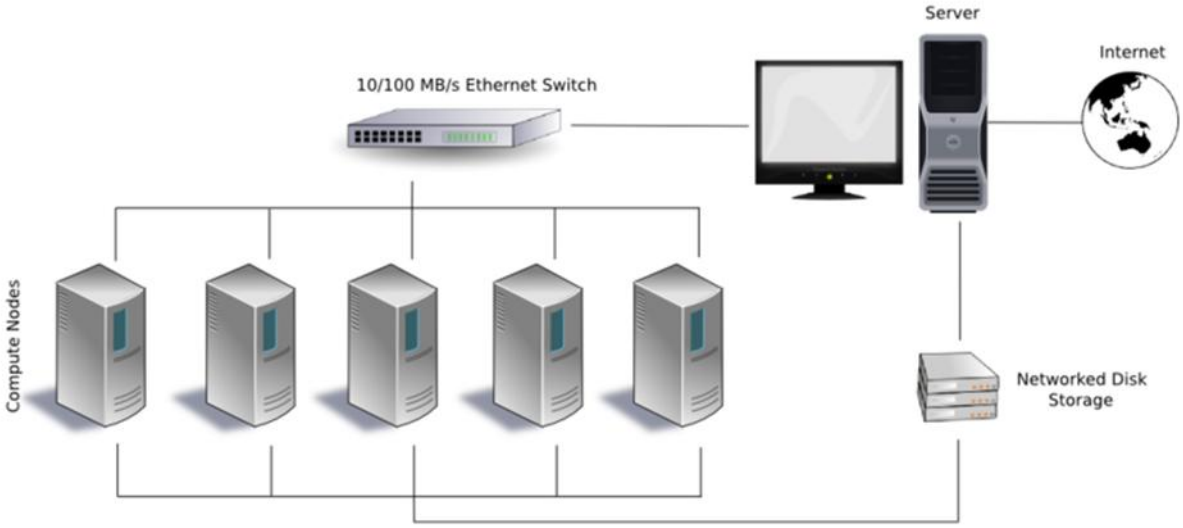

Рисунок 17. Схема организации Beowulf кластера

Особенностью такого кластера является масштабируемость, то есть возможность увеличения количества узлов системы с целью наращивания производительности вычислительной машины. Кластер Beowulf состоял из 60 узлов с 3 гигагерцовым процессором и 1.5 Гб оперативной памяти на каждом (рисунок 17). Для создания компьютерной модели использовался язык программирования C и технология MPI, позволяющая распределять обработку данных между вычислительными узлами. В ходе проекта была разработана высоко детализированная модель, которая воспроизводит работу миллиона спайковых нейронов, которые способны повторять поведение известных типов нейронов наблюдаемых *in vitro* в головном мозге крысы. В качестве модели нейрона выбрана феноменологическая модель Ижикевича. Для определения положения нейронов в модели были использованы эксперименталные данные координат нейронов в человеческом мозге. В ходе работы было использовано 22 типа нейронов. В зависимости от морфологии и целевого уровня среди модельных нейронов было выделено

8 типов возбуждающих нейронов [p2/3, ss4(L4), ss4(L2/3), p4, p5(L2/3), p5(L5/6), p6(L4), p6(L5/6)] и 9 типов тормозных [nb1, nb2/3, b2/3, nb4, b4, nb5, b5, nb6, b6].

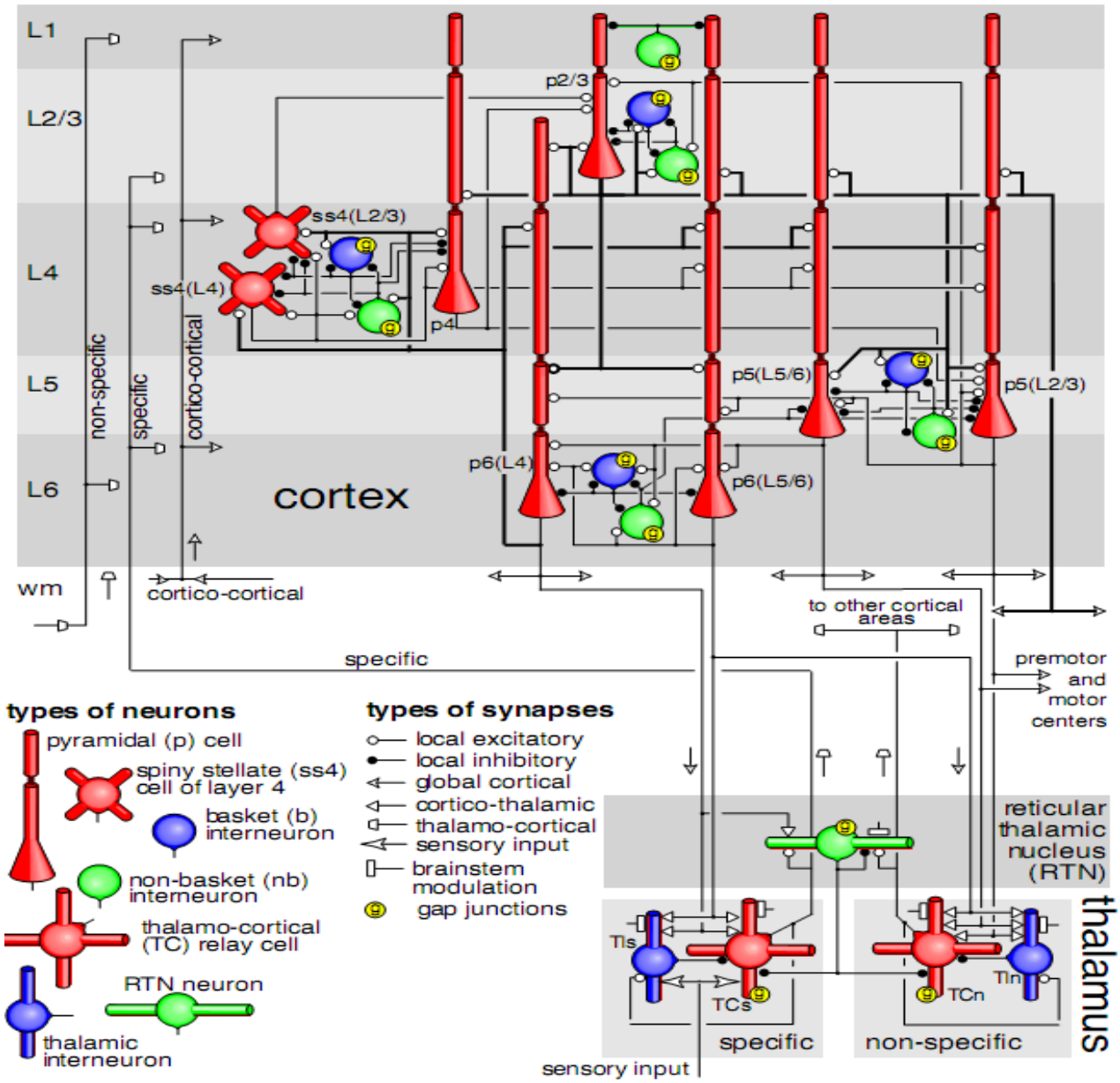

Рисунок 18. Упрощенная диаграмма микросхемы структуры ламинарной коры (вверху) и ядер таламуса (внизу). Схема представляет собой шестислойную структуру и содержит ядра таламуса. Диаграмма основана на данных исследований in vitro нейронов зрительной коры кошки.

Для соединения нейронов было использовано почти полмиллиарда синапсов с соответствующими рецепторами, кратковременной синаптической пластичностью и долговременной STDP-пластичностью. По соображениям производительности вычислений, плотность модельных нейронов и синапсов на квадратный миллиметр

поверхности была уменьшена. Поэтому модельные нейроны имели меньше синапсов и менее детальные дендритные деревья по сравнению с реальными нейронами. При этом для инициализации модели требовалось время около 10 минут и по 1 минуте на обсчет 1 секунды в модели (использовался субмиллисекундный шаг по времени).

Контрольные вопросы и задания

1. Какие подходы применяются к моделированию динамики нейронных сетей мозга?
2. Какие стандартные симуляторы для расчёта динамики нейронных сетей вам известны?
3. Какие задачи решаются с использованием параллельных вычислений на стандартных симуляторах динамики нейронов и нейронных сетей?
4. В чём суть численных методов решения обыкновенных дифференциальных уравнений?
5. Какие методы численного решения обыкновенных дифференциальных уравнений вам известны?
6. Напишите интегратор и решите уравнения Ходжкина-Хаксли методом Рунге-Кутты на любом языке программирования по вашему усмотрению.
7. Какая особенность динамики спайковых нейронных сетей используется для возможности распараллеливания вычислений при компьютерном моделировании?
8. Какие современные проекты, связанные с крупномасштабным моделированием мозга Вам известны? Расскажите подробнее про некоторые из них.

Литература

1. Hodgkin A.L., Huxley A.F. A quantitative description of membrane current and its application to conduction and excitation in nerve // J. Physiol. Blackwell Publishing, 1952. Vol. 117, № 4. P. 500–544.

2. Nordlie E., Gewaltig M.-O., Plesser H.E. Towards reproducible descriptions of neuronal network models. // PLoS Comput. Biol. 2009. Vol. 5, № 8. P. e1000456.

3. Brette R. et al. Simulation of networks of spiking neurons: A review of tools and strategies. № Bat 33. P. 1–66.

4. Migliore M. et al. Parallel network simulations with NEURON. // J. Comput. Neurosci. Springer, 2006. Vol. 21, № 2. P. 119–129.

5. King J.G. et al. A Component-Based Extension Framework for Large-Scale Parallel Simulations in NEURON // Front. Neuroinform. Frontiers Research Foundation, 2009. Vol. 3, № April. P. 11.

6. Pecevski D., Natschläger T., Schuch K. PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python // Front. Neuroinform. Frontiers Research Foundation, 2009. Vol. 3, № May. P. 15.

7. Vassanelli S. et al. On the Way to Large-Scale and High-Resolution Brain-Chip Interfacing // Cognit. Comput. 2012. Vol. 4, № 1. P. 71–81.

8. Van Albada S.J. et al. International Workshop, BrainComp 2013, Cetraro, Italy, July 8-11, 2013, Revised Selected Papers / ed. Grandinetti L., Lippert T., Petkov N. Cham: Springer International Publishing, 2014. Vol. 8603. P. pp 22–32.

9. Gewaltig M.-O., Diesmann M. NEST (NEural Simulation Tool) // Scholarpedia / ed. Izhikevich E. Eugene Izhikevich, 2007. Vol. 2, № 4. P. 1430.

10. Goodman D., Brette R. Brian: A Simulator for Spiking Neural Networks in Python // Front. Neuroinform. Frontiers Research Foundation, 2008. Vol. 2, № November. P. 10.

11. Drewes R. NCS, the NeoCortical Simulator Introduction to Brainlab More complex examples Prospects // Prospects. 2005.

12. Carnevale N.T., Hines M.L. The NEURON Book // Neuron. Cambridge University Press, 2006. Vol. 30, № 2. P. 457.

13. Bower J.M., Beeman D. Constructing realistic neural simulations with GENESIS. // Methods Mol. Biol. Clift. Nj. Humana Press Inc., 2007. Vol. 401. P. 103–125.

14. Ray S. et al. A general biological simulator: the multiscale object oriented simulation environment, MOOSE // BMC Neurosci. 2008. Vol. 9, № Suppl 1. P. P93.

15. Hammarlund P., Ekeberg O. Large neural network simulations on multiple hardware platforms. // J. Comput. Neurosci. 1998. Vol. 5, № 4. P. 443–459.

16. Mokhtar M. et al. Evolvable Systems: From Biology to Hardware / ed. Hornby G.S., Sekanina L., Haddow P.C. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Vol. 5216. P. 362–371.

17. Indiveri G. et al. Neuromorphic silicon neuron circuits. // Front. Neurosci. 2011. Vol. 5. P. 73.

18. Mokhtar M., Halliday D. Hippocampus-inspired spiking neural network on FPGA // Evolvable Syst. From Biol. to. 2008. P. 362–371.

19. Kunkel S. et al. Spiking network simulation code for petascale computers // Front. Neuroinform. 2014. Vol. 8. P. 78.

20. Eppler J.M. et al. PyNEST: A Convenient Interface to the NEST Simulator // Front. Neuroinform. Frontiers Research Foundation, 2009. Vol. 2, № January. P. 12.

21. Zaytsev Y. V, Morrison A. CyNEST: a maintainable Cython-based interface for the NEST simulator. // Front. Neuroinform. 2014. Vol. 8. P. 23.

22. Davison A. et al. PyNN: towards a universal neural simulator API in Python // BMC Neurosci. / ed. Neuroscience B.M.C. 2007. Vol. 8(Suppl 2), № Suppl 2. P. P2.

23. Davison A.P. et al. PyNN: A Common Interface for Neuronal Network Simulators. // Front. Neuroinform. Frontiers Media SA, 2008. Vol. 2. P. 11.

24. Santhakumar V., Aradi I., Soltesz I. Role of mossy fiber sprouting and mossy cell loss in hyperexcitability: a network model of the dentate gyrus incorporating cell types and axonal topography. // J. Neurophysiol. 2005. Vol. 93, № 1. P. 437–453.

25. Corchs S., Deco G. A neurodynamical model for selective visual attention using oscillators. // Neural Netw. 2001. Vol. 14, № 8. P. 981–990.

26. Djurfeldt M. et al. Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer // IBM J. Res. Dev. IBM, 2008. Vol. 52, № 1.2. P. 31–41.

27. Scorcioni R., Hamilton D.J., Ascoli G.A. Self-sustaining non-repetitive activity in a large scale neuronal-level model of the hippocampal circuit // BMC Neurosci. 2008. Vol. 9, № Suppl 1. P. P62.

28. Markram H. The blue brain project. // Nat. Rev. Neurosci. 2006. Vol. 7, № 2. P. 153–160.

29. Izhikevich E.M., Edelman G.M. Large-scale model of mammalian thalamocortical systems. // Proc. Natl. Acad. Sci. U. S. A. 2008. Vol. 105, № 9. P. 3593–3598.

30. Tsodyks M., Pawelzik K., Markram H. Neural networks with dynamic synapses. // Neural Comput. 1998. Vol. 10, № 4. P. 821–835.