Моделирование электрических процессов в системе соединённых нейронов

Кознов И. В.

2014

РЕФЕРАТ

Пояснительная записка к выпускной квалификационной работе магистра, 41 страница, 6 рисунков, 2 таблицы, 14 источников.

УРАВНЕНИЕ ХОДЖСКИНА – ХАКСЛИ, OPENCL, ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ, АРХИТЕКТУРА SIMD, НЕЙРОННАЯ СЕТЬ, ГЕТЕРОГЕННЫЕ ВЫЧИСЛЕНИЯ, МОДЕЛЬ КОГНИТИВНЫХ ПРОЦЕССОВ

Объектом исследования является использование общедоступной цифрой вычислительной техники для моделирования электрических процессов в системе нейронов.

Целью работы является создание эффективной реализации модели, рассчитывающей электрический ток биологической нейронной сети.

В процессе работы была рассмотрено уравнение Ходжкина – Хаксли для описания электрического тока сквозь мембрану нервной клетки. Дано основание для применения параллельных вычислений для расчёта системы из нескольких нервных клеток. Создана реализация использующая технологию *OpenCL* для одновременного расчёта нейронов. Проведено тестирование реализации на множестве различных программных и аппаратных платформах.

Результатом работы, является демонстрация того, что модели биологических нейронных сетей могут быть рассчитаны на недорогостоящем общедоступном оборудовании в масштабе сравнимом с реальным временем. Дана зависимость времени вычислений от параметров нейронной сети и вычислительного устройства, используя которую можно предполагать ожидаемое время расчёта модели.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной квалификационной работе магистра проводится моделирование электрических токов нервного волокна. Используются общедоступные современные вычислительные системы. За основу взято уравнение Ходжкина – Хаксли. При проведении моделирования уделяется внимание эффективности расчётов, исходя из доступности технической базы и сущности самих вычислений.

Для построения данной модели когнитивных процессов используется цифровая вычислительная техника, проводится численное решение дифференциальных уравнении, описывающих токи на мембране нейрона. Для ускорения расчёта системы из нескольких нейронов применяются параллельные вычисления. В качестве метода параллельных вычислений вычислений была выбрана технология *OpenCL*.

Репозиторий с материалами работы расположен по адресу `https://bitbucket.org/iKoznov/koznov-masters-thesis`. Он содержит текст данного документа, исполняемые файлы и исходные тексты написанных для проведения моделирования программ.

1 Предметная область и постановка задачи

Работа представляет собой отчёт о эксперименте по симуляции системы нейронов. Используется физическая модель нейрона Ходжкина – Хаксли. В настоящем эксперименте не ставится задача максимально точного, на сколько это возможно, вычисления. Имеет место компромисс между скоростью и точность, а именно важна эффективность расчётов. Исходя из этого подбирается наиболее подходящая и доступная техническая база.

1.1 Физическая модель токов мембраны нервной клетки

Модель Ходжкина и Хаксли является наиболее точным описанием поведения нейрона [1]. В серии статей [2—5] описаны электрические механизмы, обуславливающие генерацию и передачу нервного сигнала в гигантском аксоне кальмара. За это авторы модели получили Нобелевскую премию [6] в области физиологии и медицины за 1963 год.

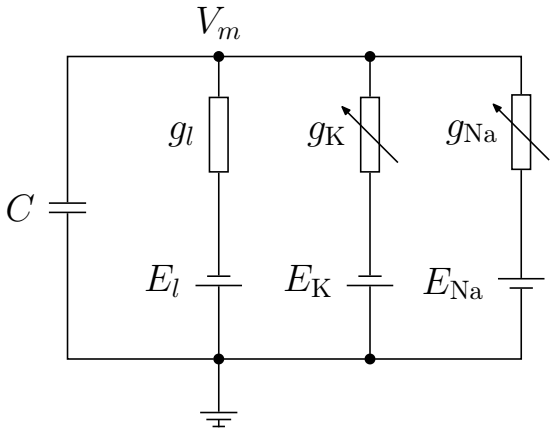

Рисунок 1.1 — Эквивалентная схема мембраны аксона.

Согласно этой модели электрические процессы в мембране могут быть описаны схемой показаной на рисунке 1.1. Обозначения величин сохранены такими же как и в оригинальных работах Ходжкина и Хаксли. Заряд может переносится сквозь мебрану как заряжанием мембранной ёмкости, так и движением ионов через сопротивления впараллель ёмкости. Ионный ток разделён на компоненты переносимые ионами натрия и калия (I_Na и I_K), и небольшой «ток утечки» (I_l) вызванный хлоридом и другими ионами. Каждая компонента ионного тока определяется движущей силой, которая

легко может быть измерена как разность электрических потенциалов и коэффициент проницаемости имеющий величину проводимости. Таким образом ток натрия (I_{Na}) равняется проводимости натрия (g_{Na}) помноженной на разность между мембранным потенциалом (E) и равновесным потенциалом для иона натрия (E_{Na}). Аналогичные равенства применимы к I_{K} и I_l, и собраны на странице 8.

Эксперименты Ходжкина и Хаксли предполагают, что g_{Na} и g_{K} являются функциями от времени мембранного потенциала, но E_{Na}, E_{K}, E_l, C_M и $\bar{g}_l$ могут быть взяты константами. Влияние потенциала мембраны на проницаемость может быть обобщено высказыванием: во-первых, что деполяризация вызывает кратковременное повышение проводимости натрия и медленное, но сохраняющееся увеличение проводимости калия; во-вторых, что эти изменения оцениваются, и что они могут быть отменены реполяризацией мембраны. Для того чтобы решить, являются ли эти эффекты достаточными для объяснения сложных явлений таких как потенциал действия и рефрактерный период, необходимо получить соотношения, связывающие проводимости натрия и калия со временем и мембранным потенциалом. Перед этим кратко рассмотрим, какие типы физических систем, вероятно, будут соответствовать наблюдаемым изменениям проницаемости.

Характер изменений проницаемости

Первое заключение, которое следует в том, что изменения в проницаемости, по-видимому, зависят от мембранного потенциала, а не от мембранного тока. При фиксированной деполяризации ток натрия проходит со временем сквозь мембрану в форме независимой от напряжения. Если концентрация натрия такая, что $E_{\mathrm{Na}} < E$, ток натрия является внутренним; если он снижен пока $E_{\mathrm{Na}} > E$, ток меняется в знаке, но всё ещё по-видимому следует тому же курсу во времени. Дальнейшая поддержка мнения, о том что мембранный потенциал является переменной контролирующей проницаемость обеспечивается тем фактом, что восстановление нормального мембранного потенциала вызывает снижение проводимости натрия или калия до наименьшего значения на любом этапе реакции.

Зависимость g_{Na} и g_{K} от потенциала мембраны предполагает, что изменения проницаемости возникают от воздействия электрического поля на распределение или ориентацию молекул с зарядом или дипольный момент. Под этим не подразумевается исключение химических реакций, для скорости, с которой они происходят, возможна зависимость от положения заряженного субстрата или катализатора. Всё это означает, что небольшие изменения мембранного потенциала весьма маловероятно вызывали бы большие изменения в состоянии мембраны, которая полностью состоит из электрически нейтральных молекул.

1.2 Математическое описание

Суммарный ток мембраны

Первым шагом является разделение общего тока мембраны на ток конденсатора и на ионный ток. Таким образом

$$I = C_M \frac{\mathrm{d}V}{\mathrm{d}t} + I_i, \tag{1.1}$$

где

I — суммарная плотность тока мембраны (входящий ток положительный);

I_i — плотность ионного тока (входящий ток положительный);

V — смещение мембранного потенциала от его значения покоя (деполяризация отрицательная);

C_M — ёмкость мембраны на единицу площади (предполагается постоянной);

t — время.

Основанием для этого уравнения является то, что оно наиболее простое, которое может быть использовано и что оно даёт величины для ёмкости мембраны, которые независимы от величины или знака V и мало зависит от изменения V со временем. Доказательство того, что ёмкостной ток и ионный ток параллельны (как предложено уравнением (1.1)) обеспечивается сходством ионных токов измеренными с $\frac{\mathrm{d}V}{\mathrm{d}t} = 0$ и посчитанными из $-C_M \frac{\mathrm{d}V}{\mathrm{d}t}$ с $I = 0$.

Единственное значительное замечание, которое должно быть сделано к уравнению (1.1), это то, что оно не учитывает диэлектрические потери

в мембране. Нет простого способа оценки погрешности внесённой этим приближением, но она не считается большой пока изменение во времени ёмкостного всплеска было разумно близко к рассчитанному для идеального конденсатора.

Ионный ток

Дальнейшее подразделение тока мембраны может быть сделано разбиением ионного тока на компоненты переносимые ионами натрия (I_{Na}), ионами калия (I_{K}) и другими ионами (I_l):

$$I_i = I_{\mathrm{Na}} + I_{\mathrm{K}} + I_l \tag{1.2}$$

Отдельные ионные токи

В третьей публикации серии работ Ходскина и Хаксли [4] показано, что ионная проницаемость мембраны может быть удовлетворительно выражена через ионные проводимости (g_{Na}, g_{K} и $\bar{g}_l$). Отдельные ионные токи получены из этих соотношений

$$\begin{aligned} I_{\mathrm{Na}} &= g_{\mathrm{Na}}(E - E_{\mathrm{Na}}), \\ I_{\mathrm{K}} &= g_{\mathrm{K}}(E - E_{\mathrm{K}}), \\ I_l &= \bar{g}_l(E - E_l), \end{aligned}$$

где E_{Na} и E_{K} равновесные потенциалы для ионов натрия и калия. E_l потенциал при котором «ток утечки» из-за хлорида и других ионов равняется нулю. Для практического применения удобно записывать эти уравнения в виде

$$I_{\mathrm{Na}} = g_{\mathrm{Na}}(V - V_{\mathrm{Na}}), \tag{1.3}$$

$$I_{\mathrm{K}} = g_{\mathrm{K}}(V - V_{\mathrm{K}}), \tag{1.4}$$

$$I_l = \bar{g}_l(V - V_l), \tag{1.5}$$

где

$$V = E - E_r,$$
$$V_{\mathrm{Na}} = E_{\mathrm{Na}} - E_r,$$
$$V_{\mathrm{K}} = E_{\mathrm{K}} - E_r,$$
$$V_l = E_l - E_r,$$

и E_r абсолютная величина потенциала покоя. V, V_{Na}, V_{K} и V_l могут быть измерены непосредственно как смещения от потенциала покоя.

Проводимость калия

Формальные допущения, используемые для описания проводимости калия:

$$g_{\mathrm{K}} = \bar{g}_{\mathrm{K}} n^4, \tag{1.6}$$

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n(1-n) - \beta_n n, \tag{1.7}$$

где $\bar{g}_{\mathrm{K}}$ константа с размерностью [проводимость/площадь], α_n и β_n константы скорости, которые изменяются с напряжением, но не со временем и имеют размерность [время$^{-1}$], n безразмерная переменная, которая может изменяться от 0 до 1.

Этим уравнениям может быть дан физический смысл, если предполагается, что ионы калия пересекают мембрану только тогда, когда четыре подобные частицы занимают определённую область мембраны. n представляет собой долю частиц в определённом положении (например, на внутренней стороне мембраны) и $1-n$ доля частиц где-то ещё (например, на внешней стороне мембраны). α_n определяет скорость передачи снаружи внутрь, в то время как β_n определяет передачу в обратном направлении. Если частица имеет отрицательный заряд, α_n должна возрасти и β_n должна уменьшиться, когда мембрана деполяризована.

В состоянии покоя, определяемым $V = 0$, n имеет величину покоя равную

$$n_0 = \frac{\alpha_{n0}}{\alpha_{n0} + \beta_{n0}}.$$

Если V внезапно изменено, α_n и β_n мгновенно принимают значения соответствующие новому напряжению. Решение уравнения (1.7) с начальным условием $n = n_0$ при $t = 0$

$$n = n_\infty - (n_\infty - n_0) \exp(-t/\tau_n) \tag{1.8}$$

где

$$n_\infty = \alpha_n / (\alpha_n + \beta_n) \tag{1.9}$$

и

$$\tau_n = 1 / (\alpha_n + \beta_n) \tag{1.10}$$

Проводимость натрия

Существуют по меньшей мере два обобщённых способа описания переходных изменений в проводимости натрия. Во-первых, можно предположить, что проводимость натрия определяется переменной, которая подчиняется дифференциальному уравнению второго порядка. Во-вторых, можно предположить, что она определяется двумя переменными, каждая из которых подчиняется уравнению первого порядка. Эти две альтернативы примерно соответствуют двум основным типам механизмов, упомянутых в связи с характером инактивации.

Сделанные формальные предположения:

$$g_{\mathrm{Na}} = m^3 h \bar{g}_{\mathrm{Na}}, \tag{1.11}$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m(1 - m) - \beta_m m, \tag{1.12}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h(1 - h) - \beta_h h, \tag{1.13}$$

где $\bar{g}_{\mathrm{Na}}$ константа и α-ы и β-ы функции от V, но не от t.

Для данных уравнений может быть дано физическое основание, если проводимость натрия предполагается пропорциональной числу мест на внутренней стороне мембраны, которые заняты одновременно тремя активирующими молекулами, но не блокируются деактивирующей молекулой. m тогда представляет долю активирующих молекул внутри и $1 - m$

долю активирующих молекул снаружи; h доля деактивирующих молекул снаружи и $1-h$ доля внутри. α_m или β_h и β_m или α_h представляют постоянные скорости передачи в обоих направлениях.

Уравнение Ходжкина – Хаксли

Соберём вместе все уравнения описывающие изменение тока на мембране:

$$I = C_M \frac{\mathrm{d}V}{\mathrm{d}t} + \bar{g}_{\mathrm{K}} n^4 (V - V_{\mathrm{K}}) + \bar{g}_{\mathrm{Na}} m^3 h (V - V_{\mathrm{Na}}) + \bar{g}_l (V - V_l), \tag{1.14}$$

где

$$\frac{\mathrm{d}n}{\mathrm{d}t} = \alpha_n (1 - n) - \beta_n n, \tag{1.7}$$

$$\frac{\mathrm{d}m}{\mathrm{d}t} = \alpha_m (1 - m) - \beta_m m, \tag{1.12}$$

$$\frac{\mathrm{d}h}{\mathrm{d}t} = \alpha_h (1 - h) - \beta_h h, \tag{1.13}$$

и

$$\alpha_n = \frac{0.01(V + 10)}{\exp \frac{V+10}{10} - 1}, \tag{1.15}$$

$$\beta_n = 0.125 \exp(V/80), \tag{1.16}$$

$$\alpha_m = \frac{0.1(V + 25)}{\exp \frac{V+25}{10} - 1}, \tag{1.17}$$

$$\beta_m = 4 \exp(V/18), \tag{1.18}$$

$$\alpha_h = 0.07 \exp(V/20), \tag{1.19}$$

$$\beta_h = \frac{1}{\exp \frac{V+30}{10} + 1} \tag{1.20}$$

Уравнение (1.14) выводится несложно из уравнений (1.1)–(1.6) и (1.11). Четыре слагаемых в правой части дают соответственно ток на ёмкости, ток переносимый ионами K, ток переносимый ионами Na и ток утечки на 1 см^2

Таблица 1.1 — Значения констант в модели Ходжкина – Хаксли.

константа	выбранная величина
C_M	1.0 мкФ/см^2
V_{Na}	-115 мВ
V_{K}	$+12$ мВ
V_l	-10.613 мВ
$\bar{g}_{\text{Na}}$	120 мСм/см^2
$\bar{g}_{\text{K}}$	34 мСм/см^2
$\bar{g}_l$	0.3 мСм/см^2

мембраны. Эти четыре компоненты включены параллельно и составляют общую плотность тока через мембрану I. Проводимости для K и Na заданы константами $\bar{g}_{\text{K}}$ и $\bar{g}_{\text{Na}}$ вместе с безразмерными величинами n, m и h, изменение которых со временем после изменения потенциала мембраны определяется тремя вспомогательными уравнениями (1.7), (1.12) и (1.13). α-ы и β-ы в этих уравнениях зависят только от мгновенного значения мембранного потенциала, и даются оставшихся шести уравнениях. Вывод уравнений (1.15)–(1.20) опущен, они были получены аппроксимацией экспериментальных данных при измерении проводимости натриевого и калиевого каналов. Экспериментальные данные и получение уравнений были опубликованы Ходжкином и Хаксли в статье [5].

Потенциалы даны в мВ, плотность тока в мкА/см^2, проводимости в мСм/см^2, ёмкость в мкФ/см^2, и время в мс. Выражения для α и β соответствуют температуре 6.3° C; для других температур они должны быть масштабированы с температурным коэффициентом Q_{10} равным 3.

Константы в уравнении (1.14) взяты как независимые от температуры. Выбранные величины даны в таблице 1.1

1.3 Похожие исследования

В связи с ростом доступности вычислительных ресурсов, исследования с применением компьютерного моделирования становятся все более

популярными. Остановимся на часто упоминаемых в литературе проектах *NEST*, *NEURON*, *Blue Brain* и *GENESIS*.

NEST — программный симулятор, используемый для моделирования сетей, биологически реалистичных элементов и связей [7]. Программа оптимизирована для симуляции больших нейронных сетей и способна обработать модель, состоящую из 100000 элементов (нейронов) и приблизительно одного миллиарда связей (синапсов) между ними. Главные составляющие элементы системы: ядро и интерпретатор системы моделирования. Интерпретатор осуществляет взаимодействие графического интерфейса и языка моделирования с ядром системы. Принципы организации приложения *NEST* способствуют эффективному использованию вычислительных ресурсов компьютеров с многоядерным процессором, многопроцессорных компьютеров и кластеров. При моделировании на вычислительном кластере или многопроцессорных компьютерах, каждый компьютер или процессор воссоздаёт часть сети и хранит информацию о синаптических контактах только своих нейронов. Для распределения задач на все компьютеры (процессоры) NEST использует интерфейс мгновенных сообщений (MPI) и потоков POSIX (pthreads).

NEURON — симулятор, служащий для компьютерного моделирования отдельных нейронов и их сетей. Среда предоставляет эффективные инструменты для создания, управления, и использования моделей. Программный пакет *NEURON* является особенно подходящим для выполнения симуляций, основанных на экспериментальных данных, тех, что требуют учёта сложных анатомических и биофизических свойств клетки. Это связано с возможностью моделирования нейронов с высоким уровнем детализации, включая различные типы ионных каналов мембраны, разбиение клетки на составные части — компартменты — каждая из которых имеет свою собственную динамику. Главная особенность данного симулятора состоит в «естественном синтаксисе», который позволяет исследователям определять свойства модели знакомыми им средствами, задавая биофизические параметры моделей компартментов, вместо программирования решения дифференциальных уравнений численными методами. *NEURON* пред-

лагает несколько встроенных методов интегрирования с возможностью переключения между ними, не переписывая модель. Вычислительное ядро симулятора использует специальным образом адаптированные алгоритмы, достигающие наивысшей производительности, при работе с уравнениями, структура которых схожа с уравнениями, описывающими нейроны. В *NEURON* предусмотрена поддержка нескольких 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).

Проект Blue Brain стартовал 1 июля 2005 года как результат сотрудничества Швейцарского Федерального Технического Института Лозанны и корпорации IBM. Целью проекта является детальное моделирование отдельных нейронов и образуемых ими типовых колонок неокортекса мозга — неокортикальных колонок. Неокортекс располагается в верхнем слое полушарий мозга, имеет толщину 2–4 миллиметра и отвечает за высшие нервные функции — сенсорное восприятие, выполнение моторных команд, пространственную ориентацию, осознанное мышление и, у людей, речь. Для расчёта модели использовался симулятор *NEURON*, содержащий реализации моделей, параметры которых подгонялись для соответствия экспериментальным данным. Исследования по проекту *Blue Brain* предполагают разбиение на несколько фаз. Одним из важнейших результатов первого этапа стала модель одной колонки неокортекса молодой крысы, полученная в конце 2006 года.

В работе Эрика Шаттера [8] было проведено моделирование клетки Пуркинье (крупные нервные клетки имеющиеся в можжечке). Была использована уточнённая модель Ходжкина – Хаксли. Симуляция проводилась в системе *GENESIS*. В результате была получена наглядная

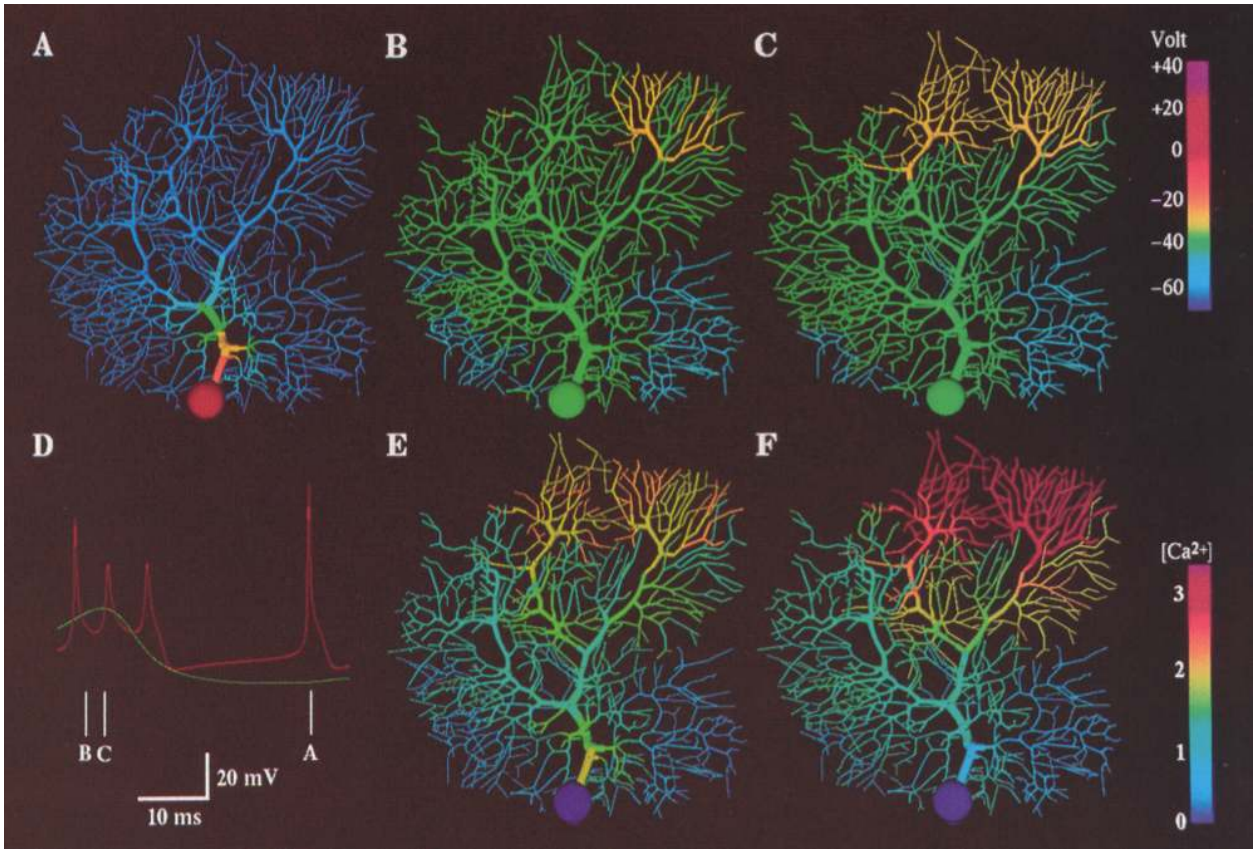

Рисунок 1.2 — Демонстрация работы модели Эрика Шаттера. Показано изменение электрического напряжения в различных частях клетки Пуркинье.

демонстрация процессов в клетке (рисунок 1.2), при воздействии на неё электрическим током. Модель зарегистрирована на видео [9]. Исходные файлы модели находятся в открытом доступе [10]. Необходимые расчёты для симуляции доли секунды жизни клетки заняли несколько часов.

2 Методика моделирования

Реализуемая модель является дискретным представление непрерывной модели Ходжкина – Хаксли. Для расчётов используется цифровая вычислительная техника. Данный подход позволяет легко менять структуру и параметры нейронной сети, однако вносится погрешность вычисления. Для получения характеристики электрического тока на мембранах нейронов производится численное решение дифференциального уравнения (1.14).

2.1 Процедура решения уравнения Ходжкина – Хаксли

Численное решение уравнения Ходжкина – Хаксли осуществляется методом Рунге – Кутты четвёртого порядка [11]. На каждой итерации выполняется вычисление новых значений для V, n, m и h (объяснение смысла переменных дано в главе 1):

$$
\begin{aligned}
m_{i+1} &= m_i + \frac{\delta}{6}(a_1 + 2a_2 + 2a_3 + a_4),\\
n_{i+1} &= n_i + \frac{\delta}{6}(b_1 + 2b_2 + 2b_3 + b_4),\\
h_{i+1} &= h_i + \frac{\delta}{6}(c_1 + 2c_2 + 2c_3 + c_4),\\
V_{i+1} &= V_i + \frac{\delta}{6}(d_1 + 2d_2 + 2d_3 + d_4),
\end{aligned}
$$

где δ шаг интегрирования, и

$$
\begin{aligned}
a_1 &= m'(m, V),\\
b_1 &= n'(n, V),\\
c_1 &= h'(h, V),\\
d_1 &= V'(m, n, h, V),
\end{aligned}
$$

$$
\begin{aligned}
a_2 &= m'(m + \frac{\delta}{2}a_1, V + \frac{\delta}{2}d_1),\\
b_2 &= n'(n + \frac{\delta}{2}b_1, V + \frac{\delta}{2}d_1),\\
c_2 &= h'(h + \frac{\delta}{2}c_1, V + \frac{\delta}{2}d_1),\\
d_2 &= V'(m + \frac{\delta}{2}a_1, n + \frac{\delta}{2}b_1, h + \frac{\delta}{2}c_1, V + \frac{\delta}{2}d_1),
\end{aligned}
$$

$$
\begin{aligned}
a_3 &= m'(m + \frac{\delta}{2}a_2, V + \frac{\delta}{2}d_2),\\
b_3 &= n'(n + \frac{\delta}{2}b_2, V + \frac{\delta}{2}d_2),\\
c_3 &= h'(h + \frac{\delta}{2}c_2, V + \frac{\delta}{2}d_2),\\
d_3 &= V'(m + \frac{\delta}{2}a_2, n + \frac{\delta}{2}b_2, h + \frac{\delta}{2}c_2, V + \frac{\delta}{2}d_2),
\end{aligned}
$$

$$
\begin{aligned}
a_4 &= m'(m + \delta a_3, V + \delta d_3),\\
b_4 &= n'(n + \delta b_3, V + \delta d_3),\\
c_4 &= h'(h + \delta c_3, V + \delta d_3),\\
d_4 &= V'(m + \delta a_3, n + \delta b_3, h + \delta c_3, V + \delta d_3).
\end{aligned}
$$

2.2 Оценка времени вычислений

В данном разделе приведены зависимость времени расчёта модели (T) от количества нейронов N и связей между ними, размера шага интегрирования δ, длины временного отрезка Δt, на котором рассчитывается модель, и числа процессорных элементов P, которые можно задействовать.

Временной отрезок Δt разбивается на равные интервалы δ, время выполнения одной итерации не зависит от Δt и δ. Общее время расчёта модели прямо пропорционально Δt и обратно пропорционально δ.

Из процедуры расчёта изменения напряжения за одну итерацию численного решения уравнения Ходжкина – Хаксли (разделы 1.2 и 2.1) следует, что каждое следующее состояние отдельного нейрона зависит от его состояния и состояний других нейронов на предыдущем шаге, но не зависит от состояний других нейронов на текущем рассчитываемом шаге. Поэтому состояния нейронов на одной итерации могут вычисляться независимо друг от друга, а итерации численного решения должны проходить последовательно, одна за другой.

Согласно закону Амдала $T(P) = T_s + T_p/P$, где T_s — не распараллеливаемая часть программы, а T_p — распараллеливаемая, программа не может выполниться быстрее не распараллеливаемой части. В нашем случае доля не распараллеливаемой части равняется нулю, а максимальное число

процессоров, которое можно задействовать равняется числу нейронов N. Следовательно T прямо пропорционально $min(N, P)$.

Каждый нейрон может иметь число связей не большее, чем некоторое число. В расчёте напряжения на нейроне участвует сумма токов на аксонах, по этому время расчёта связей каждого нейрона также не превосходит некоторое значение, обозначим его S.

Из изложенного выше следует, что время расчёта модели T прямо пропорционально величине $\frac{\Delta t}{\delta}(const + S)\min(N, P)$.

2.3 Использованные технологии

Основной этап моделирования проводился с использованием нескольких ЭВМ, под управлением различных операционных систем. На предварительном этапе моделирования использовалась среда *PTC Mathcad Prime* 3.0.

При написании программного обеспечения для эксперимента преследовалась цель собрания большего количества опытных данных с различных вычислительных машин с использованием различного программного. Исходя из этого был выбран набор, позволяющий максимально достичь независимости от аппаратного, так и программного обеспечения.

Наиболее ресурсоёмкой частью вычислений является, решение уравнения Ходжкина – Хаксли. В разделе 2.2 дано обоснование возможности и выгоды параллельного вычисления решения уравнения Ходжкина – Хаксли. Для каждого нейрона применятся одна и та же процедура расчёта, которая имеет малое число условных переходов, либо не имеет их совсем, в зависимости от реализации. Поэтому возможно эффективное применение процессорной архитектуры SIMD (single instruction, multiple data — одна команда, много данных). Данная архитектура широко применятся в графических процессорах (GPU, graphics processing unit). Были рассмотрены несколько способов реализации расчётов на GPU: *CUDA* [12], *OpenCL* [13; 14] и шейдеры. В соответствии с поставленными целями была выбрана технология *OpenCL*. Технологии *CUDA* не было отдано предпочтение, поскольку она использует только аппаратное обеспечение фирмы *nVidia*. Однако, с посредством написания шейдеров, можно добиться большого,

возможно даже большего множества поддерживаемого оборудования, чем с использованием OpenCL, но их основное применение это графика, поэтому для выполнения вычислений потребовались бы дополнительные затраты. Выбор *OpenCL* во многом обусловлен открытостью стандарта *OpenCL* и наличием реализаций для большого числа оборудования. Помимо использования графического процессора *OpenCL* способен задействовать и центральный процессор для исполнения той же самой программы, что даёт дополнительные возможности для эксперимента. В главе 2.4 освящены ключевые особенности *OpenCL*.

Для обеспечения конфигурации и сборки написанной для эксперимента программы была использована сборочная система *CMake*. Это позволило с небольшими усилиями обеспечить конфигурацию и сборку для различных операционных систем и компиляторов. Программа написанная для эксперимента запускалась на операционных системах: *Windows* 7 *SP* 1, *Ubuntu* 14.04 и *OS X* 10.10. В главе 3 приведены измерения времени вычисления различных сборок программы на различном оборудовании.

2.4 Технология OpenCL

OpenCL (Open Computing Language) является открытым стандартом для кросс-платформенного, параллельного программирования процессоров таких как многоядерные CPU и программируемые GPU, разработан в 2008 году в результате взаимодействия компании *Khronos Group* со фирмами-разработчиками программного обеспечения и производителями процессоров. Основная идея OpenCL — предоставить программисту универсальный инструмент для использования всех вычислительных мощностей современных вычислительных систем. Например, при использовании *OpenCL* возможно написать программу для GPU, не заботясь об адаптации алгоритмов для различных API (аппаратно-программный интерфейс) таких как

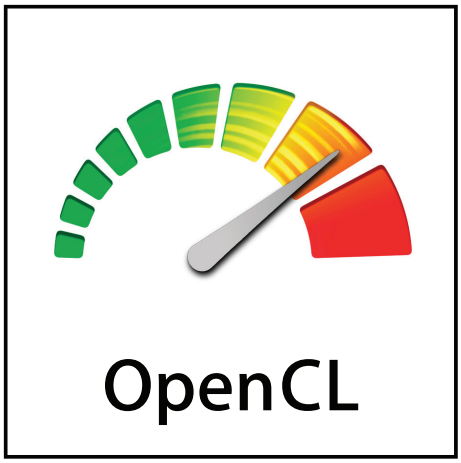

Рисунок 2.1 — Логотип OpenCL.

OpenGL или *DirectX*.

OpenCL состоит из трёх частей: языка основанного на стандарте C99, API для разработки и среды выполнения, распределяющей нагрузку между ядрами на CPU или GPU. Для описания основных идей разработчики стандарта используют четыре модели [13]:

- модель платформы (platform model)
- модель памяти (memory model)
- модель вычислений (execution model)
- модель программирования (programming model)

Модель платформы

Модель платформы изображена на рисунке 2.2. Модель состоит из хоста подключённого к одному или множеству OpenCL устройств. OpenCL устройство разделено на одно или несколько вычислительных единиц (compute unit, CU), которые в свою очередь делятся на один или несколько процессорных элементов (processing element, PE).

OpenCL приложение выполняется на хосте в зависимости от модели нативной для хост-платформы. Приложение отправляет команды от хоста для выполнения вычислений на процессорных элементах устройства. Процессорные элементы одной вычислительной единицы исполняют поток инструкций как SIMD команды (исполнение только одного потока инструкций) или как SPMD (у каждого PE есть свой счётчик команд).

Модель вычислений

Модель вычислений описывает абстрактное представление того, как потоки инструкций выполняются в гетерогенной системе. Исполнение программы *OpenCL* происходит в два этапа: исполнение ядер на одном или нескольких *OpenCL* устройствах и исполнение хост-программы. Хост-программа определяет контекст для ядер и управляет их работой. Когда ядро назначено для исполнения хостом, определяется индексное пространство (*NDRange*). Экземпляр ядра называется рабочим элементом (*work-item*) и идентифицируется по точке в индексном пространстве, которое

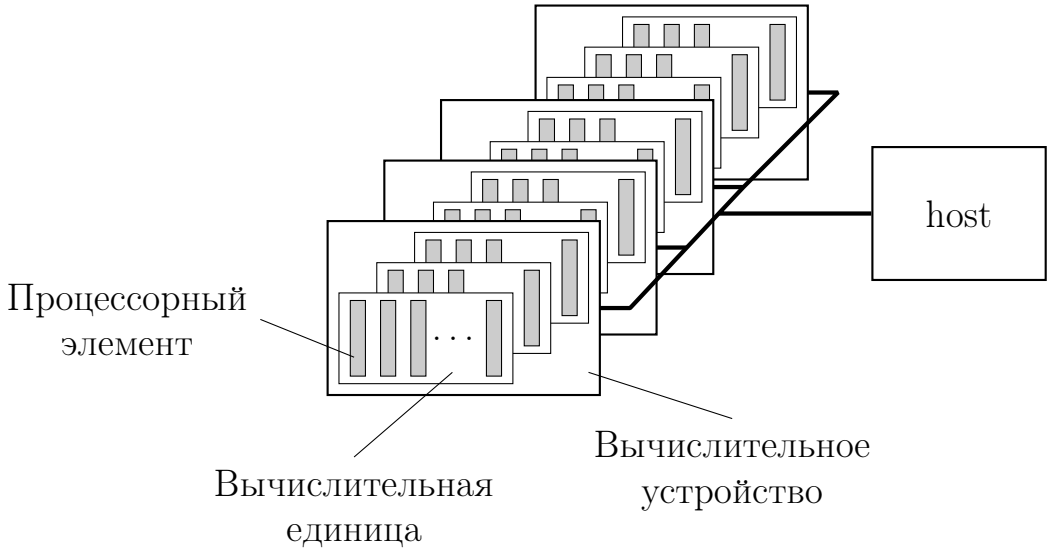

Рисунок 2.2 — Общее устройство *OpenCL*.

предоставляет глобальный идентификатор (*global ID*) для рабочего элемента.

Множество всех рабочих элементов разбивается на рабочие группы (*work-group*). С каждой рабочей группой сопоставляется свой уникальный идентификатор (*work-group ID*). Все рабочие элементы в одной группе идентифицируются уникальным в пределах своей группы номером: *local ID*. Таким образом каждый рабочий элемент определяется как по уникальному *global ID* так и по комбинации *work-group ID* и *local ID* внутри своей группы.

Все рабочие элементы в пределах одной группы выполняются параллельно на процессорных элементах одной вычислительной единицы *OpenCL*-устройства. Это гарантируется стандартом, в то время как совершенно не гарантируется, что несколько рабочих элементов из разных групп будут выполнены параллельно. Об этом важном свойстве параллелизма необходимо всегда помнить при разработке *OpenCL*-программ.

Другим важным понятием модели вычислений является контекст, определение которого необходимо в начале работы *OpenCL*-приложения. Контекст определяет среду выполнения ядер, в которую входят устройства, сами ядра, программы (*program objects*, текст программ и исполняемый код ядер), объекты памяти.

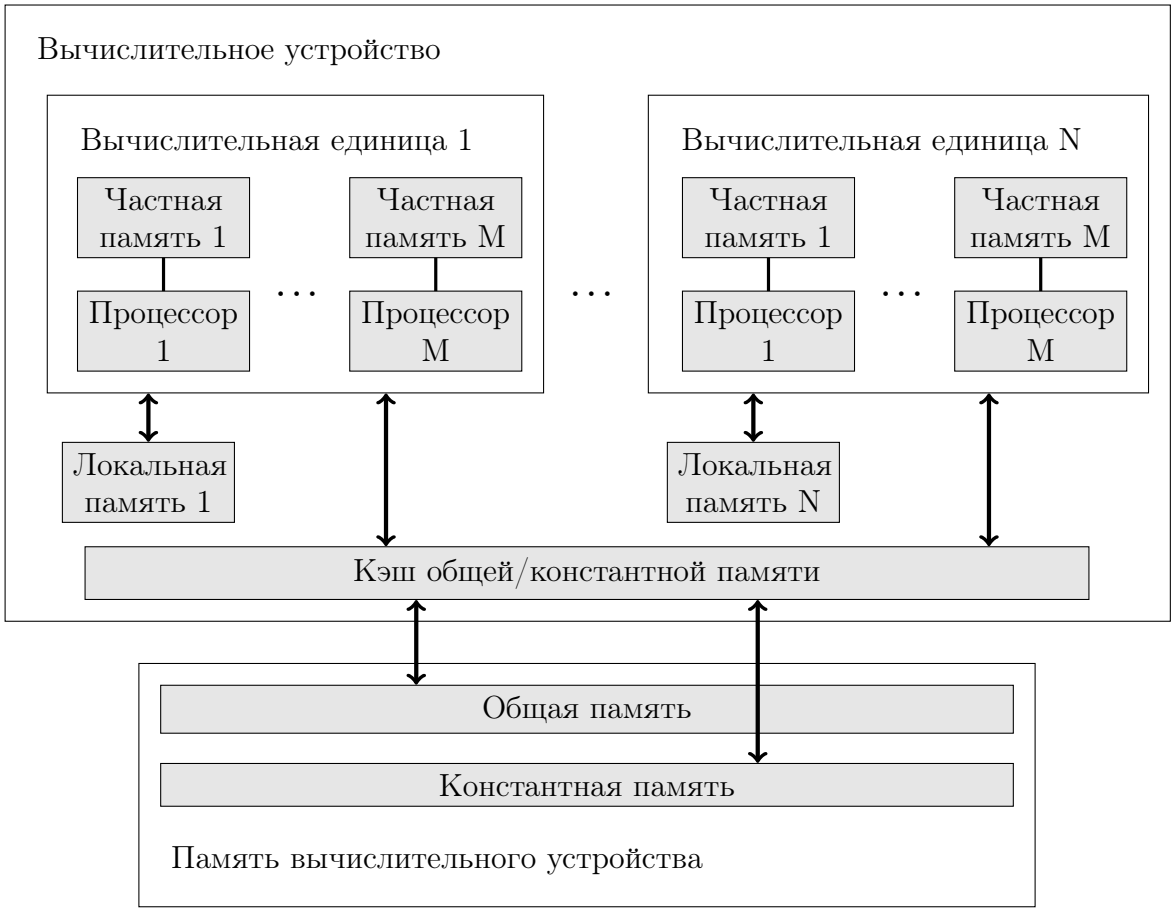

Рисунок 2.3 — Виды памяти *OpenCL*.

Взаимодействие между хостом и *OpenCL*-устройстром происходит посредством команд, помещённых в командную очередь (command-queue). Данные команды ожидают в командной очереди своего выполнения на *OpenCL*-устройстве. Командная очередь создаётся хостом и сопоставляется одному *OpenCL*-устройству после того, как будет определён контекст. Команды делятся на те, что отвечают за: выполнение ядер, управление памятью и синхронизацию выполнения команд в очереди. Команды могут выполняться последовательно или внеочерёдно. Второй вариант организации очередей поддерживается не всеми платформами.

Модель памяти

Рабочие элементы, исполняющие ядро, имеют доступ к четырём различным областям памяти:

– Глобальная память (*Global Memory*). Эта часть памяти предоставляет

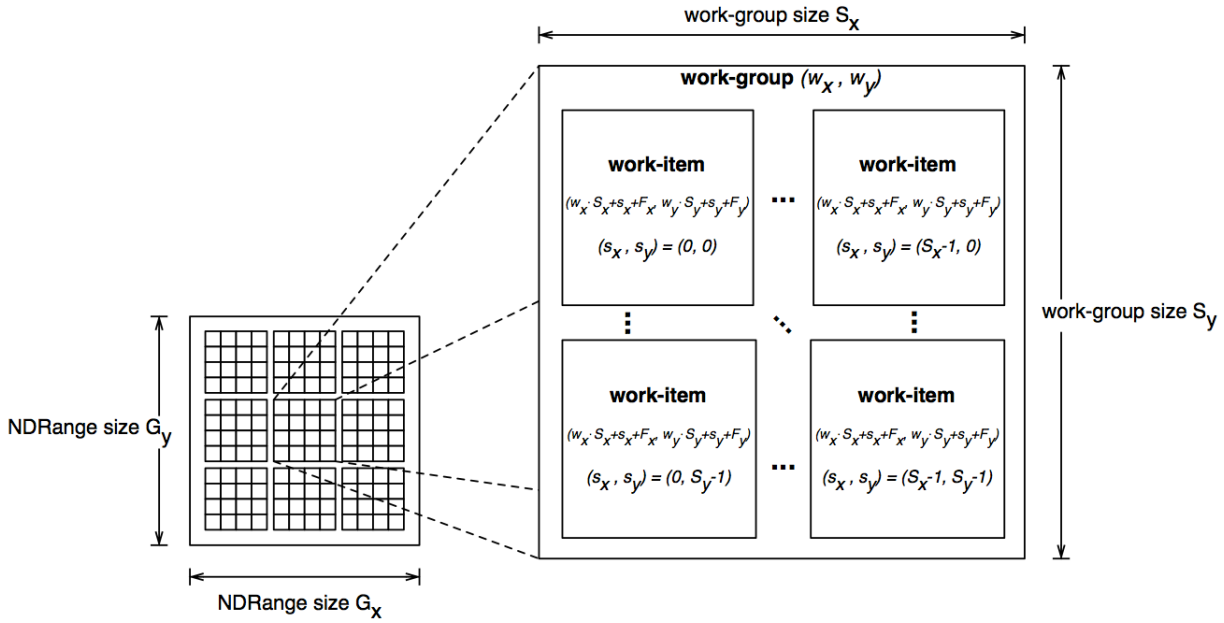

Рисунок 2.4 — Пример индексного пространства NDRange, показывающий рабочие элементы, их глобальные идентификаторы и их соответствие паре идентификатора группы и локального идентификатора.

права на чтение и запись всем рабочим элементам всех групп. Глобальная память может быть кэширована, в зависимости от устройства.

– Константная память (*Constant Memory*): часть глобальной памяти остающейся неизменной во время вычислений ядра. Выделение памяти для объектов в константной памяти и их инициализация выполняются хостом.
– Локальная память (*Local Memory*): участок памяти, относящийся к рабочей группе. Может быть использован для переменных, используемых во всех элементах рабочей группы. Может быть реализован, как выделенные участки памяти на *OpenCL* устройстве.
– Частная память (*Private Memory*): область памяти, выделенная на один рабочий элемент. Переменные определённые в памяти одного элемента не видны другим элементам.

Области памяти, и то, как они относятся к модели платформы показано на рисунке 2.3.

Модель программирования

Модель программирования описывает варианты переноса абстрактного алгоритма на гетерогенные вычислительные ресурсы. *OpenCL* определяет два типа модели программирования: параллелизм по данным (*data parallelism*) и параллелизм по задачам (*task parallelism*).

Параллелизм по данным

Этот тип модели программирования организован вокруг структур данных: каждый элемент структуры данных, определенной хостом, обновляется одновременно (параллельно) копиями одного и того же *OpenCL*-ядра. Дизайн такой структуры должен поддерживать возможность одновременного изменения различных её частей.

Модель параллелизма по данным наиболее естественная модель программирования для *OpenCL*, так как индесное пространство *NDRange* создается непосредственно перед запуском ядра на устройстве. Задача программиста описать решаемую задачу в терминах структуры данных, отобразив её на *NDRange* и инкапсулировав её в объекте памяти.

В случае, если между несколькими рабочими элементами необходимо взаимодействие, то требуется проектировать алгоритм с учетом разбиения множества всех рабочих элементов на некоторое количество групп, в рамках которых элементы могут осуществлять такое взаимодействие. Взаимодействие может осуществляться либо через чтение и запись локальных областей памяти, либо через синхронизацию посредством группового барьера (*work-group barrier*).

Если групповой барьер определён в тексте программы ядра, то все рабочие элементы в пределах одной рабочей группы должны дойти до этого барьера прежде чем все рабочие элементы в этой группе перешагнут барьер и продолжат своё выполнение. Примером задачи, где может понадобится такая синхронизация служит суммирование элементов массива. Ввиду ориентированности технологии *OpenCL* на широкий круг устройств программисту необходимо учитывать, что *OpenCL* не поддерживает механизмы синхронизации между несколькими рабочими элементами из разных рабочих групп.

Стандарт *OpenCL* описывает иерархическую модель параллелизма по данным: внутри каждой группы и между различными группами. Все рабочие элементы в пределах группы выполняются одновременно, и группы в свою очередь групп могут выполняться в одно и тоже время. При проектировании алгоритма имеется две возможности относительно определения групп. Первое (*explicit model*) — определить размер групп заранее. Второе (*implicit model*) — предоставить возможность разбиения на группы самой системе.

Параллелизм по задачам

В стандарте *OpenCL* под заданием понимается ядро, выполняемое как единый элемент. При этом вместе с таким заданием в устройстве могут одновременно выполняться и другие элементы. Необходимость параллелизма по заданиям может возникнуть, если параллелизм уже заключён в самом задании. Например, если в ядре задания производятся векторные операции над векторным типом данных.

Также данный тип модели программирования применим в случае, когда несколько команд запуска ядер помещаются в очередь, в которой запуск происходит сразу же после попадания в неё команды. В ряде случаев это позволяет увеличить степень использования *OpenCL*-устройств, позволяя системе планировать запуск множества заданий. Параллелизм по заданиям с внеочередным запуском может не работать на некоторых вычислительных платформах и данный вариант модели программирования является скорее опциональной возможностью *OpenCL*.

Третий способ параллелизма по заданиям возникает, когда множество заданий зависимо между собой и они объединяются в граф с использованием событий *OpenCL*. Одни команды, помещённые в очередь могут генерировать события, другие могут ожидать этих событий, чтобы начать своё выполнение.

2.5 Реализация ядра вычислений

При реализации программ использующих *OpenCL* для получения выгоды от параллельных вычислений очень важно правильно выбрать тип

аппаратного устройства, тип параллелизма (по данным или задачам) и выявить независимые участки программы. Следует минимизировать долю нераспараллеливаемой части. Также следует уменьшать количество копирований данных между хостом и устройством *OpenCL*, использовать помять наиболее подходящего адресного пространства `__global`, `__constant`, `__local` или `__private` [14].

В ходе исследований учитывались эти критериями. Были предприняты две попытки реализации параллелизма для расчёта нейронной сети. Далее изложены ключевые аспекты каждой реализации.

Первая попытка

Первая попытка реализации ядра была сделана для одного из упрощений модели Ходжкина – Хаксли, модели Ижкевича [15]:

$$\begin{cases} C_m \dfrac{\mathrm{d}V_m}{\mathrm{d}t} = k(V_m - V_r)(V_m - V_t) - U_m + I_b + I_{syn} \\ \dfrac{\mathrm{d}U_m}{\mathrm{d}t} = a(b(V_m - V_r) - U_m) \end{cases} \tag{2.1}$$

если $V_m \geq V_{peak}$, то

$$\begin{cases} V_m = c \\ U_m = U_m + d \end{cases}$$

где a, b, c, d, k безразмерные параметры нейрона. C_m — ёмкость мембраны, V_m — разность потенциалов на внутренней и внешней стороне мембраны, V_{peak} минимальная разность потенциалов во время спайка. U_m — вспомогательная переменная измеряется в вольтах. Постоянный ток I приложен к мембране. t — время.

Решение осуществлялось методом Эйлера. Каждая итерация решения системы дифференциальных, описывающей нейронную сеть разбита на два последовательных этапа:

1. вычисление изменения разности потенциалов V_m внешней и внутренней стороны мембраны, за счёт приложенного тока I и изменения времени $\mathrm{d}t$;
2. вычисление влияния связей между нейронами на разность потенциалов.

Текст программы исполняемой на устройстве *OpenCL* содержит два ядра вычислений `izh_neuron` и `izh_connection`. Они используются для расчёта изменения разности потенциалов и влияния связей соответственно.

```
float izhik_Vm ( int neuron, int time,
                 __global float* Vms, __global float* Ums,
                 __global float* Iex, __global float* Isyn )
{
  return
    ( k * ( Vms[neuron + time*Nneur] - Vr ) *( Vms[neuron + time*Nneur] - Vt )
    - Ums[neuron + time*Nneur] + Iex[neuron] + Isyn[neuron] ) / Cm;
}

float izhik_Um ( int neuron, int time,
                 __global float* Vms, __global float* Ums )
{
  return a * ( b * ( Vms[neuron + time*Nneur] - Vr )
    - Ums[neuron + time*Nneur] );
}

__kernel void izh_neuron ( __global int* t_,
                           __global float* Vms, __global float* Ums,
                           __global float* Iex, __global float* Isyn )
{
  const int t = *t_;
  const int neur = get_global_id ( 0 );

  Vms[neur + t*Nneur] = Vms[neur + ( t-1 ) *Nneur]
    + h*izhik_Vm ( neur, t-1, Vms, Ums, Iex, Isyn );
  Ums[neur + t*Nneur] = Ums[neur + ( t-1 ) *Nneur]
    + h*izhik_Um ( neur, t-1, Vms, Ums );
  Isyn[neur] = 0.0f;

  if ( Vms[neur + ( t-1 ) *Nneur] > Vpeak ) {
    Vms[neur + t*Nneur] = c;
    Ums[neur + t*Nneur] = Ums[neur + ( t-1 ) *Nneur] + d;
  }
}
```

```
__kernel void izh_connection (
  __global int* t_, __global float* y,
  __constant float* Vms, __global float* Isyn,
  __global int* pre_conns, __global int* post_conns,
  __global float* weights )
{
  const int t = *t_;
  const int con = get_global_id ( 0 );
  const float expire_coeff = exp ( -h/psc_excxpire_time );

  y[con + t*Nneur] = y[ con + (t-1)*Nneur ] * expire_coeff;

  if ( Vms[pre_conns[con] + (t-1)*Nneur] > Vpeak ) {
    y[con + t*Nneur] = 1.0f;
  }
  Isyn[post_conns[con]] += y[con + t*Nneur] * weights[con];
}
```

Текст программы хоста, осуществляющий вызовы ядер вычислений:

```
for ( int t = 1; t < Tsim; ++t ) {
  clEnqueueWriteBuffer ( command_queue, mem_t, CL_TRUE, 0,
                         sizeof ( Tsim ), &t, 0, NULL, NULL );
  clSetKernelArg ( kernel_izh_neuron, 0,
                   sizeof ( cl_mem ), &mem_t );
  clSetKernelArg ( kernel_izh_connection, 0,
                   sizeof ( cl_mem ), &mem_t );
  clEnqueueNDRangeKernel ( command_queue, kernel_izh_neuron, 1, NULL,
                           global_neuron_work_size, local_work_size,
                           0, NULL, NULL );
  clEnqueueNDRangeKernel ( command_queue, kernel_izh_connection, 1, NULL,
                           global_connection_work_size, local_work_size,
                           0, NULL, NULL );
}
```

Все операции выполняются в синхронной очереди команд `command_queue`. Это означает, что все команды в очереди выполняются одна за другой, поэтому не требуется синхронизация. Вызовы `clEnqueueNDRangeKernel` соответствуют первому и второму этапу вычислений. Они означают, что

процедуры расчёта изменения разности потенциалов и влияния связей будут исполнены последовательно одна за другой, но каждая процедура исполняется одновременно в нескольких экземплярах, соответствующим каждому отдельному нейрону и каждому отдельному аксону.

При профилировании данного участка программы оказалось, что передача в ядро вычислений номера шага на каждой итерации пагубно сказывается на времени исполнения (рисунок 2.5). Вызовы `clEnqueueWriteBuffer` и `clSetKernelArg` в приведённом выше тексте программы отвечают за это. Падение производительности объясняется тем, что копирование памяти хоста в *OpenCL* устройство дорогая операция: в случае когда в роли устройства используется дискретный графический процессор происходит передача данных про шине PCI Express из оперативной памяти в память GPU. А поскольку все этапы расчётов выполняются последовательно, графический процессор простаивает некоторое время пока происходит копирование. Далее было решено избавиться от данной задержки.

Вторая попытка

Во второй попытке использовалась модель Ходжкина – Хаксли, решение дифференциальных уравнений осуществлялось методом Рунге – Кутта. Был учёт опыт первой попытки, и счётчик итераций перенесён в ядро вычислений *OpenCL*. И число вычислительных ядер сократилось до одного. Каждый экземпляр ярда соответствует одному нейрону. Несмотря на то, ядра вычисляются параллельно, рассинхронизации вычислений не происходит ввиду устройства *OpenCL*. Все процессоры управляемые *OpenCL* в момент времени исполняют одну и ту же команду, но применительно к разным данным.

Теперь в время вычислений не происходит передачи информации между хостом и устройством *OpenCL*, и соответственно отсутствует задержка, вызванная копированием данных. Текст программы запуска вычислений на устройстве сократился до одного вызова:

```
clEnqueueNDRangeKernel ( command_queue, kernel_hodgkin_huxley, 1, NULL,
                         global_work_size, local_work_size, 0, NULL, NULL );
```

а текст программы счётчика итераций перенесён в ядро:

	Name	Domain	Start Time (µs)	Duration (µs)
426	clEnqueueWriteBuffer	OpenCL	194,869.886	117.629
427	clSetKernelArg	OpenCL	194,987.980	0.897
428	clSetKernelArg	OpenCL	194,989.159	0.489
429	clEnqueueNDRangeKernel	OpenCL	194,989.939	33.685
430	clEnqueueNDRangeKernel	OpenCL	195,023.980	30.579
431	clEnqueueWriteBuffer	OpenCL	195,054.932	118.659
432	clSetKernelArg	OpenCL	195,174.226	0.934
433	clSetKernelArg	OpenCL	195,175.442	0.491
434	clEnqueueNDRangeKernel	OpenCL	195,176.298	36.505
435	clEnqueueNDRangeKernel	OpenCL	195,213.179	30.450
436	clEnqueueWriteBuffer	OpenCL	195,244.004	136.286
437	clSetKernelArg	OpenCL	195,380.801	0.889
438	clSetKernelArg	OpenCL	195,381.927	0.478
439	clEnqueueNDRangeKernel	OpenCL	195,382.747	33.979
440	clEnqueueNDRangeKernel	OpenCL	195,417.133	30.719
441	clEnqueueWriteBuffer	OpenCL	195,448.247	115.976
442	clSetKernelArg	OpenCL	195,564.698	0.973
443	clSetKernelArg	OpenCL	195,565.909	0.487
444	clEnqueueNDRangeKernel	OpenCL	195,566.693	40.020
445	clEnqueueNDRangeKernel	OpenCL	195,607.218	31.028

Рисунок 2.5 — Фрагмент последовательности вызовов *OpenCL* во время исполнения первой реализации параллельного расчёта нейронов. Видно, что вызовы `clEnqueueWriteBuffer`, осуществляющие копирование номера итерации из памяти хоста на устройство, занимают более половины времени. Время исполнения команды в милисекундах показано в последнем столбце. Измерения снимались с устройства GeForce GTX 480. Для профилирования был использован инструментарий *Nsight* 3.0.

```
__kernel void hodgkin_huxley_neuron ( __global struct Neuron* neurons )
{
  // получение идентификатора нейрона
  const int i = get_global_id ( 0 );
  // счётчик итераций
  for ( int t = 0; t < STEPS - 1; ++t ) {
    const double dt = DT;
    const double n = neurons[i].n, m = neurons[i].m, h = neurons[i].h;
    const double V = neurons[i].V[t];

    // расчёт влияния тока на аксонах
    double I = 0;
    for ( int k = 0; k < NUMBER_OF_AXONS; ++k )
      I += neurons[i].w * neurons[neurons[i].axon[k]].V[t];
      I /= NUMBER_OF_AXONS;
      I += neurons[i].I;

      // решение задачи Коши методом Рунге-Кутты
      const double k1m = dt * dm ( m, V );
      const double k1n = dt * dn ( n, V );
      const double k1h = dt * dh ( h, V );
      const double k1V = dt * dv ( m, n, h, V, I );

      const double k2m = dt * dm ( m+k1m/2, V+k1V/2 );
      const double k2n = dt * dn ( n+k1n/2, V+k1V/2 );
      const double k2h = dt * dh ( h+k1h/2, V+k1V/2 );
      const double k2V = dt * dv ( m+k1m/2, n+k1n/2, h+k1h/2, V+k1V/2, I );

      const double k3m = dt * dm ( m+k2m/2, V+k2V/2 );
      const double k3n = dt * dn ( n+k2n/2, V+k2V/2 );
      const double k3h = dt * dh ( h+k2h/2, V+k2V/2 );
      const double k3V = dt * dv ( m+k2m/2, n+k2n/2, h+k2h/2, V+k2V/2, I );

      const double k4m = dt * dm ( m+k3m, V+k3V );
      const double k4n = dt * dn ( n+k3n, V+k3V );
      const double k4h = dt * dh ( h+k3h, V+k3V );
      const double k4V = dt * dv ( m+k3m, n+k3n, h+k3h, V+k3V, I );

      neurons[i].m += ( k1m + 2*k2m + 2*k3m + k4m ) / 6;
```

```
      neurons[i].n += ( k1n + 2*k2n + 2*k3n + k4n ) / 6;
      neurons[i].h += ( k1h + 2*k2h + 2*k3h + k4h ) / 6;
      neurons[i].V[t+1] = V + ( k1V + 2*k2V + 2*k3V + k4V ) / 6;
  }
}
```

Текст программы ядра, содержащий описание уравнения Ходжкина–Хаксли (1.14):

```
__constant double g_n = 120.0;
__constant double g_k = 36.0;
__constant double g_l = 0.3;
__constant double v_n = 115.0;
__constant double v_k = -12.0;
__constant double v_l = 10.613;
__constant double c = 1.0;

double alpha_n ( double v ) {
  return ( v!=10 ) ? 0.01* ( -v+10 )
    / ( exp ( ( -v+10 ) /10 ) - 1 ) : 0.1;
}
double beta_n ( double v ) {
  return 0.125*exp ( -v/80 );
}
double n_inf ( double v ) {
  return alpha_n ( -v )
    / ( alpha_n ( -v ) +beta_n ( -v ) );
}

double alpha_m ( double v ) {
  return ( v!=25 ) ? 0.1* ( -v+25 )
    / ( exp ( ( -v+25 ) /10 ) - 1 ) : 1;
}
double beta_m ( double v ) {
  return 4*exp ( -v/18 );
}
double m_inf ( double v ) {
  return alpha_m ( -v )
    / ( alpha_m ( -v ) +beta_m ( -v ) );
```

```
}

double alpha_h ( double v ) {
  return 0.07*exp ( -v/20 );
}
double beta_h ( double v ) {
  return 1/ ( exp ( ( -v+30 ) /10 ) +1 );
}
double h_inf ( double v ) {
  return alpha_h ( -v )
    / ( alpha_h ( -v ) +beta_h ( -v ) );
}
```

По завершению работы ядра возвращается массив структур со значениями напряжений на всех нейронах на каждом шаге моделирования:

```
struct Neuron {
  double V[STEPS]; // значения напряжения
  double n, m, h;
  double I, w;
  int axon[NUMBER_OF_AXONS];
};
```

В отличие от предыдущей попытки, эта дала выигрыш во времени расчёта нейронной сети, по сравнению с реализацией использующей центральный процессор и один поток. Далее в главе 3 даны замеры времени расчёта модели, все результаты даны для этой, второй, попытки реализации.

3 Результаты моделирования

В рамках данной работы было создано реализации модели Ходжкина – Хаксли для расчёта тока сквозь мембраны нейронов. Обе реализации способны вычислять токи в нейронные сетях различных размеров. Первая предназначена для исполнения в одном потоке посредством центрального процессора. Её цель дать сравнительную характеристику второй реализации, которая использует технологию *OpenCL* и вычисляется параллельно, используя ресурсы либо центрального процессора, либо графического. В обоих реализация проводятся аналогичные расчёты описанные в разделе 2.1. Обе реализации кроссплатформенные, что позволило не ограничивать эксперимент какой-либо одной программной или аппаратной платформой.

Для начала рассмотрим результат работы модели трёх последовательно соединённых нейронов. Нейроны соединены следующим образом: дендрит первого соединён с аксоном второго, через соединение ток передаётся полностью бе потерь; дендрит второго нейрона соединён с аксоном третьего нейрона, ток передаётся с некоторыми потерями. На мембрану первого нейрона совершалось электрическое воздействие постоянным током различной величины. На рисунке 3.1 показаны графики изменения напряжения на нейронах. При малом токе нейрон не активируется, а напряжение на его мембране после небольшого скачка стабилизируется на некотором уровне. После прекращения воздействия током нейрон возвращается в состояние покоя, разность потенциалов на внутренней и внешней стороне мембраны равняется нулю. В случае если ток больше некоторого значения, то напряжения на нейроне достигает порога, происходит активация нейрона. После активации напряжение может стабилизироваться, или нейрон начнёт переодически активироваться с частотой зависимой от величины приложенного тока. Амплитуда изменения напряжения при активации не зависит от величины приложенного тока. Графики напряжения на мембранах второго и третьего нейронов демонстрируют эффект распространения потенциала действия. В зависимости от характера связи скачок напряжения, спайк,

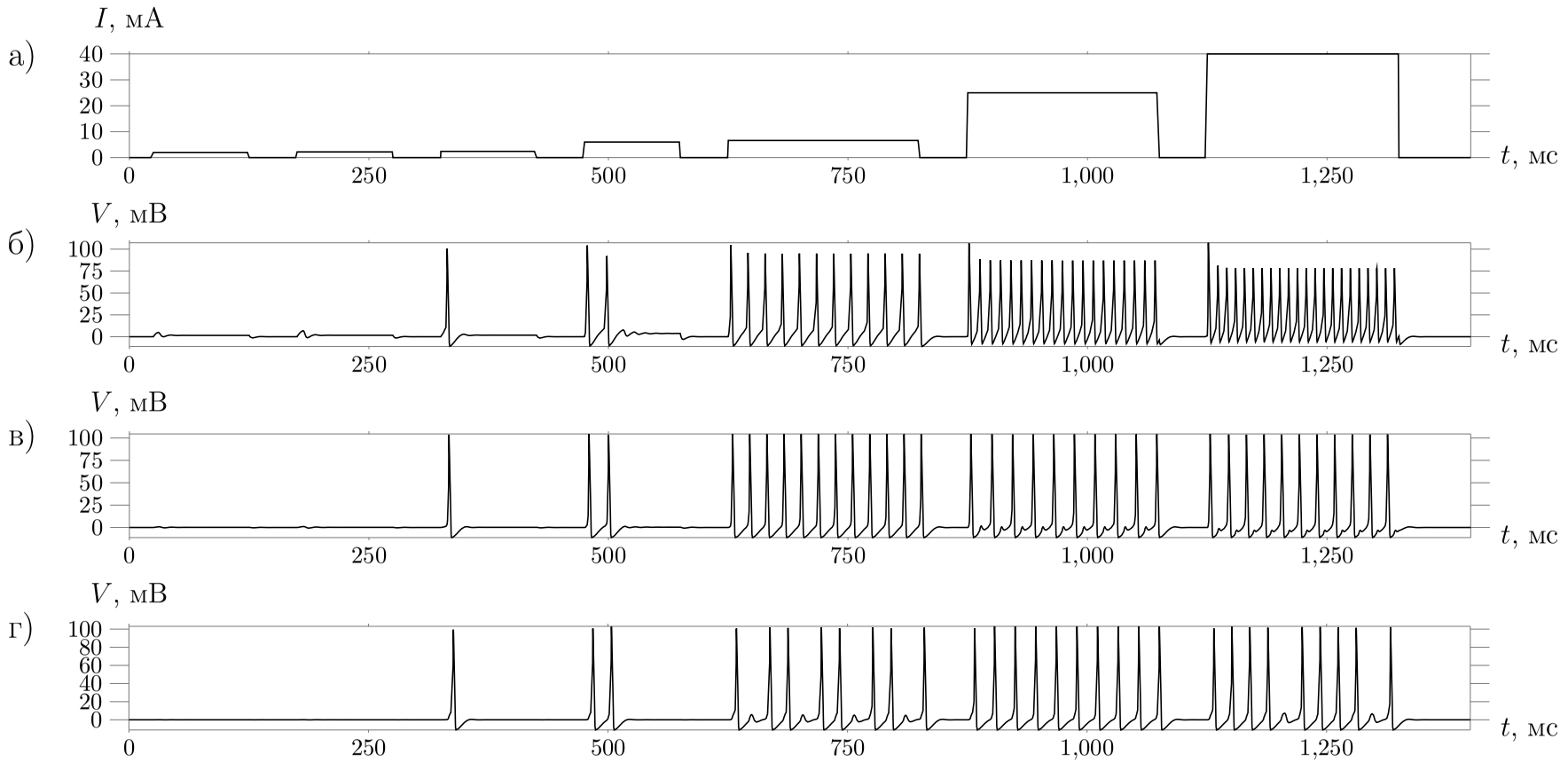

Рисунок 3.1 — Реакция трёх последовательно соединённых нейронов на воздействие постоянным током на один из нейронов. а) Импульсы постоянного тока. б) Первый нейрон в цепочке, к нему приложено внешнее воздействие током. в) Второй нейрон в цепочке соединён с аксоном первого, ток передаётся без потерь. г) Третий нейрон соединён с аксоном второго, ток передаётся с потерями.

может передаться или нет соединённому нейрону.

Следующий пример работы модели, нейронная сеть состоящая из двадцати нейронов соединённых случайными связями. На графике 3.2 энцефалограмма этой сети, полученная как среднее значение напряжения на каждом нейроне. Видно как напряжение переодически достигает своего пика.

В таблице 3.1 приведены замеры времени расчёта модели сети из 1024 нейронов. Данные даны для различного оборудования. Из таблицы видно кратное уменьшение времени вычисления модели с использованием реализации *OpenCL* по сравнению с реализацией вычислений в один поток. Это справедливо для всего оборудования, за исключением процессора для мобильных устройств Intel Core i5-2557M. Используя OpenCL графические и центральные процессоры одинаковых ценовых категорий показали примерно одинаковое время вычислений.

Выражаю свою благодарность Андрею Громову за предоставленную информацию о тестовых запусках модели на процессорах Intel Core i5-2500 и GeForce GTX 770.

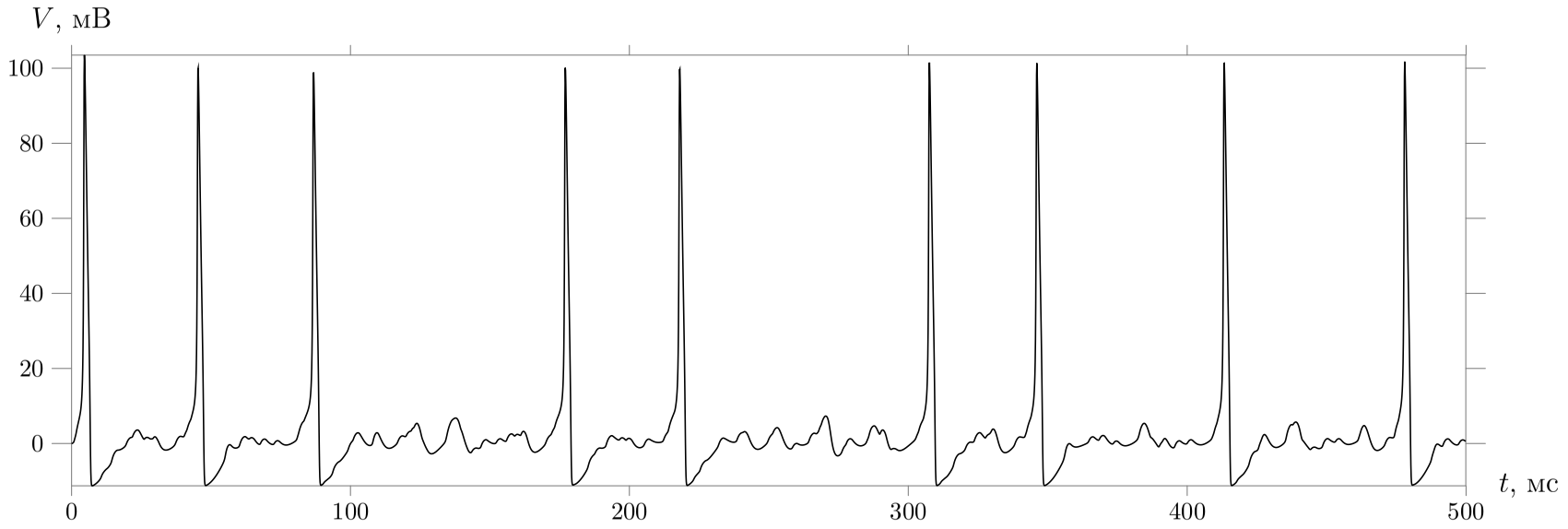

Рисунок 3.2 — Энцефалограмма нейронной сети состоящей из 20-ти нейронов со случайными связями за 500 мс. К 60% нейронов был приложен случайный ток в размере от 0 до 50 мА.

Таблица 3.1 — Время расчёта тока в сети из 1024-х нейронов с 256-ю случайными связями у каждого на временном отрезке 500 мс с шагом интегрирования 0.025 мс. К 60% нейронов был приложен случайный ток в размере от 0 до 50 мА. Тестировались реализации с использованием *OpenCL* и без. Если в четвёртом столбце стоит единица, то *OpenCL* не использовался, и вычисления проводились в один поток посредством центрального процессора. В остальных случаях использовался OpenCL. Для компиляторов *gcc*, *clang* и *Visual C++* был выставлен флаг оптимизации `O2`.

устройство	операционная система	компилятор	реализация	среднее время выполнения, с
GeForce GTX 770	Windows 7	встроен в драйвер	OpenCL	5.0
GeForce GTX 480	Windows 7	встроен в драйвер	OpenCL	13.2
GeForce GTX 480	Ubuntu 14.04	встроен в драйвер	OpenCL	13.2
Intel Core i5-2500	Windows 7	Visual C++ 2013	C++, один поток	48.9
Intel Core i7-860	Ubuntu 14.04	gcc 4.8.2	C++, один поток	33.7
Intel Core i7-860	Windows 7	Visual C++ 2013	C++, один поток	56.1
Intel Core i7-860	Windows 7	встроен в драйвер	OpenCL	7.7
Intel Core i5-2557M	OS X 10.10	clang-503.0.38	C++, один поток	59.1
Intel Core i5-2557M	OS X 10.10	встроен в драйвер	OpenCL	73.5

ЗАКЛЮЧЕНИЕ

В работе было проведено моделирование нейронной сети с использованием параллельных вычислений посредством технологии *OpenCL*. За основу была взята непрерывная модель Ходжкина – Хаксли, которая адаптирована для расчёта на цифровых вычислительных устройствах путём дискретизации. В результате получена компьютерная программа для симуляции электрических токов в биологической нейронной сети. Были проведены сравнительные запуски полученной модели на различном оборудовании. Проведено сравнение с однопоточной реализацией, которое показало эффективность использования параллельных вычислений.

Важным результатом, является демонстрация того, что модели биологических нейронных сетей могут быть рассчитаны на недорогостоящем общедоступном оборудовании в масштабе сравнимом с реальным временем. Дана зависимость времени вычислений от параметров нейронной сети и вычислительного устройства, используя которую можно предполагать ожидаемое время расчёта модели.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Koch C.*, *Bernander Ö.* Axonal Modeling // The Handbook of Brain Theory and Neural Networks / ed. by M. A. Arbib. — 2nd ed. — London, England : A Bradford book, 2004.
2. *Hodgkin A. L.*, *Huxley A. F.* Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo // The Journal of Physiology. — 1951. — No. 116. — Pp. 449–472.
3. *Hodgkin A. L.*, *Huxley A. F.* The components of membrane conductance in the giant axon of Loligo // The Journal of Physiology. — 1951. — No. 116. — Pp. 473–496.
4. *Hodgkin A. L.*, *Huxley A. F.* The dual effect of membrane potential on sodium conductance in the giant axon of Loligo // The Journal of Physiology. — 1951. — No. 116. — Pp. 497–506.
5. *Hodgkin A. L.*, *Huxley A. F.* A quantitative description of membrane current and its application to conduction and excitation in nerve // The Journal of Physiology. — 1952. — No. 117. — Pp. 500–544.
6. The Nobel Prize in Physiology or Medicine 1963. — URL: `http://www.nobelprize.org/nobel_prizes/medicine/laureates/1963/` (дата обр. 21.05.2014).
7. *Gewaltig M.-O.*, *Diesmann M.* NEST (NEural Simulation Tool) // Scholarpedia. — 2007. — Vol. 2, no. 4. — P. 1430.
8. *Schutter E.*, *Bower J. M.* An Active Membrane Model of the Cerebellar Purkinje Cell // The Journal of Physiology. — USA, 1994. — Jan. — Vol. 71, no. 1. — Pp. 375–419.
9. Purkinje Cell Model [Видеозапись]. — Дата обновления: 25.12.2007. — URL: `https://www.youtube.com/watch?v=eiGgengz2qM` (дата обр. 28.02.2014).

10. *Cornelis H.*, *Airong D.* Cerebellar purkinje cell (De Schutter and Bower 1994) [Электронный ресурс]. — URL: `http://senselab.med.yale.edu/modeldb/showmodel.asp?model=7176` (дата обр. 28.02.2014).

11. *Бахвалов Н. С.*, *Жидков Н. П.*, *М. К. Г.* — Лаборатрия Базовых Знаний, 2002.

12. CUDA C programming guide / nVidia. — Feb. 2014.

13. The OpenCL specification, version 1.2 / Khronos OpenCL Working Group ; ed. by A. Munshi. — Nov. 14, 2012. — URL: `https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf`.

14. The OpenCL Programming Book / R. Tsuchiyama [et al.] ; trans. by S. Tagawa. — Fixstars, 2012. — P. 324.

15. *Izhikevich E. M.* Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting. — The MIT Press, 2007. — P. 210.