Быстрое распараллеливание на GPU программ для расчетов моделей кардиологии

Карпаев А.А.[1], Алиев Р.Р.[1,2]

[1]*Московский физико-технический институт (государственный университет)*

[2]*Институт теоретической и экспериментальной биофизики РАН*

Абстракт

Произведено распараллеливание программ для моделирования 2D-ткани миокарда на видеокарты (GPU) с помощью высокоуровневой технологии OpenACC. В качестве моделей клеток были выбраны модели AP, BR и YNI, формально покрывающие весь спектр моделей кардиоцитов. Начальные условия задавались таким образом, чтобы привести к возникновению реентри. Для каждой модели на различных пространственных сетках получены значения ускорений параллельной программы в сравнении с последовательной. При вычислениях использовались GPU любительского (entry level ©) (серия GeForce) и профессионального (серия Tesla) уровней. Обсуждены применение продвинутых опций оптимизации OpenACC, а также выгода использования последней по сравнению с низкоуровневой технологией CUDA.

Ключевые слова: электрофизиология, кардиология, параллельные вычисления, GPU, OpenACC, концептуальные модели, детальные модели.

Введение

Сердечно-сосудистые заболевания занимают лидирующее место в мире в списке причин смерти пациентов.

Компьютерное моделирование играет значительную роль в кардиологической практике: позволяет избежать экспериментов на сердцах животных и человека, неэтичных и дорогостоящих, проводимых для оценки эффективности воздействия на сердце лекарственных препаратов, объяснения причин возникновения аритмий, оптимизации воздействия имплантов, дефибрилляторов и др. В настоящее время уровень развития детальных моделей клеток миокарда (кардиоцитов) достаточно высок, что позволяет получать результаты, применимые на практике, в ближайшем будущем (5-10 лет [ссылка]). Как и в других предметных областях, в кардиологии ускорение расчетов играет значительную роль.

Ранее для этой цели использовали кластеры из CPU; в последнее десятилетие трендом в области высокопроизводительных вычислений является использование GPU (graphic processing unit). Строго говоря, с появлением соответствующих технологий программирования графические процессоры стали называть GP-GPU (general purpose graphic processing unit).

Для программирования GPU первой была создана технология CUDA (подходит только для GPU фирмы NVidia), затем появилась технология OpenCL (для GPU фирмы AMD и многоядерных CPU различных производителей). Обе технологии являются низкоуровневыми и, следовательно, довольно громоздки в программной реализации: от разработчика требуется явное управление копированием данных между CPU и GPU, явная синхронизация потоков исполнения и др.; это серьезно замедляет труд ученых-биофизиков, не всегда досконально разбирающихся технологиях программирования.

Решением этой проблемы стала технология OpenACC, впервые увидевшая свет в ~~ноябре~~ 2011 г.: была выпущена спецификация OpenACC – технологии, позволяющей переносить код на GPU посредством кратких директив (#pragma) для компилятора, что не требует погружения в детали архитектуры GPU и значительно сокращает время, затрачиваемое на распараллеливание исходной программы. Синтаксически OpenACC очень похожа на технологию OpenMP, используемой для распараллеливания программ на многоядерных CPU уже более 20 лет.

В данной работе приводится демонстрация применения технологии OpenACC для быстрого распараллеливания программ численного моделирования электрической

активности клеток миокарда. Используются типичные представители концептуальных и детальных моделей клеток как из «ждущей», так и автоколебательной [ссылки на термины] сред. Материал статьи будет полезен в первую очередь ученым-биофизикам, использующих компьютерное моделирование в своих исследованиях; программистам данная работа также сможет представить некоторый интерес.

Материалы и методы

Введение в архитектуру GPU

В первом приближении GPU можно представить как многоядерный CPU. Главные отличия следующие: на порядок большее число вычислительных ядер, существенно меньшая тактовая частота их работы, меньший размер кэша, коллективный доступ к памяти ядрами, [что-то еще]. Из-за данных особенностей архитектуры GPU более энергоэффективны и относительно дешевы в производстве.

Архитектура GPU принадлежит к типу SIMT (Single Instruction Multiple Thread), являющейся вариацией типа SIMD из классификации параллельных архитектур по М. Флинну [ссылка]. Все потоки (в дальнейшем будем называть их *нитями*) исполнения упорядочены в сетку, разделенную на блоки. [Написать что-нибудь про блоки] Фактически параллельно способны исполняться только нити, объединенные внутри блока в так называемые *warp'ы* (наборы по 32 нити), при этом процессом их запуска заведует управляющая логика GPU.

Соответственно, параллельное программирование для GPU немного отличается от такового для многоядерного CPU.

Технологии программирования GPU

Существует несколько технологий подобного сорта (приведены в порядке появления на свет):

1. [шейдеры или preCUDA]
2. CUDA (2007)
3. OpenCL (2008)
4. OpenACC (2011)

Технологии CUDA и OpenCL представляют из себя расширения для языка Си, привнесшие каждая в него новые типы данных и компилятор, соответственно; технологии

являются *низкоуровневыми* – распараллеливание программ требует значительных трудовремязатрат от программиста.

Технология OpenACC

CUDA является широкораспространенной технологией для распараллеливания программ на GPU. OpenACC была создана с целью упрощения данного процесса. Управление инициализацией начальных условий, обменом данными между CPU и GPU, запуск фунцкии-ядра GPU производится *неявно*, путем передачи управления специальным директивам (#pragma), вставляемых в код программы в требуемых местах. Директивы остаются прозрачными для «обычного» компилятора (к примеру, gcc), что позволяет использовать одну и ту же версию программы как для последовательных, так и параллельных вычислений. Стоит отметить, что технология сильно схожа по синтаксису с технологией OpenMP, широко используемой для распараллеливания программ на многоядерные CPU (системы с общей памятью) уже более 20 лет. С учетом этого заметим, что в ближайшем будущем [год?] OpenACC планируется совместить с OpenMP [http://ccoe.msu.ru/ru/articles/openacc-intro; ссылка - мб найти другую?].

Перечислим использованный в работе набор директив и опций OpenACC, который в принципе можно назвать «базовым»:

- #pragma acc data copy [deviceptr] ~~/copyin/copyout~~
- #pragma acc parallel [present, num_workers, vector_length, async]
- #pragma acc loop [gang, vector]
- #pragma acc routine
- #pragma acc wait
- #pragma acc update [host]

Компилятор технологии OpenACC (pgcc) автоматически распознает, какие части кода должны исполняться на CPU или GPU, какие участки отвечают за управление памятью и перенос данных между CPU и GPU, и автоматически сгенерирует соответствующие функции (перенос данных, функции-ядра), беря на себя эту часть работы разработчика.

Математические модели

В данной работе мы рассматриваем модели Алиева-Панфилова (AP), Билера-Рейтера (BR) и Янагихары-Номы-Ирисавы (YNI), *формально* представляющие весь спектр моделей кардиоцитов: концептуальная и детальная клеток ждущей среды и детальная клеток автоколебательной среды, соответственно.

[Статья для физиологов – используем принятую у них форму записи: без знаков сумм, произведений и пр.]

Модель AP

Из концептуальных моделей электрофизиологии была выбрана описывающая активность кардиоцитов. Модель можно использовать для эффективного 3D-моделирования миокарда. Уравнения имеют следующий вид:

$$\begin{cases} \dfrac{\partial u}{\partial t} = D\Delta V - (ku(u-a)(u-1) + uv) + i_{stim}, \\ \dfrac{\partial v}{\partial t} = \varepsilon(u,v)\big(-v - ku(u-a-1)\big). \end{cases}$$

Здесь t – безразмерное время, k – коэффициент, определяющий время деполяризации, a – пороговая величина. Мембранный потенциал можно вычислить по формуле $\mathrm{V[мВ]} = 100u - 80$, время – $t\mathrm{[мс]} = 12.9t$. В данной работе использовались значения параметров, указанные в оригинальной модели [Aliev, Panfilov, 1996].

Модель BR

Модель описывает активность кардиоцитов желудочка (у собаки). Из множества детальных моделей кардиоцитов «ждущей» среды она обладает относительно средней сложностью и при этом учитывает в себе основные электрофизиологические процессы (некоторые – на качественном уровне), происходящие в кардиоците.

Данный выбор, с одной стороны, позволил авторам провести незагроможденную демонстрацию применения технологии распараллеливания на небольшой последовательной программе, а с другой, позволит читателям без труда обобщить применение технологии на программы для моделирования, использующие современные детальные модели. Дополнительно отметим, что модель BR не является сильно устаревшей (1977 г.) (по сравнению, например, с моделью Noble (1962 г.)) и до сих пор используется для 3D-моделирования миокарда в силу средней требовательности к вычислительным ресурсам [A Tveito].

Уравнения модели записываются в следующем виде:

$$\begin{cases} C_m \dfrac{\partial V}{\partial t} = D\Delta V - I_{ion} + I_{stim}, \\ I_{ion} \equiv I_{Na} + I_K + I_x + I_s, \\ \dfrac{dg}{dt} = \alpha_g(V)(1-g) - \beta_g(V)g, \\ g = m, h, j, d, f, x, \\ \dfrac{dc}{dt} = -10^{-7} I_s + 0.07(10^{-7} - c), \qquad c = [Ca^{2+}]. \end{cases}$$

Модель YNI

Модель описывает автоколебания в клетках-водителях ритма синусового узла (у кролика). Мотивация выбора данной модели – та же самая, что и для модели BR. Отметим, что изучение моделей кардиоцитов автоколебательной среды является профессиональным интересом авторов в последние годы.

Уравнения детальной модели в общем виде [не пишу систему в общем виде, т.к. целевая аудитория статьи – биофизики, которым не требуется разъяснений]:

$$\begin{cases} C_m \dfrac{\partial V}{\partial t} = D\Delta V - I_{ion} + I_{stim}, \\ I_{ion} \equiv I_{Na} + I_K + I_s + I_h + I_l, \\ \dfrac{dg}{dt} = \alpha_g(V)(1-g) - \beta_g(V)g, \\ g = m, h, p, d, f, q. \end{cases}$$

[Это copypaste из статьи по «Сравнению численных методов»] Здесь C_m – емкость клеточной мембраны, V – мембранный потенциал, I_{ion} – полный ионный ток, $g_j(\boldsymbol{m})$ – проводимость для j-го тока (нелинейная функция от вектора воротных переменных $\boldsymbol{m}$), E_j – равновесный потенциал для j-го тока; α_g и β_g являются нелинейными функциями от мембранного потенциала V. Пейсмейкерным током в модели является I_s. Значения всех параметров модели можно найти в оригинальной работе [статья YNI(1980)].

Начальные условия

Для всех моделей начальное условие в точках сетки задавалось с помощью введения понятия *фазы* $\varphi(t) \equiv \omega t + \varphi_0$, для которой должны выполняться следующие неявные соотношения:

$$V(t_0) = V_{threshold}, \qquad {dV}/{dt}\,(t_0) > 0, \qquad \text{где } t_0 = t(\varphi = 0),$$
$$V(t_1) = V_{threshold}, \qquad {dV}/{dt}\,(t_1) > 0, \qquad \text{где } t_1 = t(\varphi = 2\pi).$$

Для того, чтобы получить реентри, требуется в каждой точке (x, y) расчётной области задать фазу, равную углу точки в полярных координатах (с началом координат в центре области).

Численные методы

Модель AP

Для дискретизации по пространству в уравнении на приведенный мембранный потенциал u использовалась стандартная формула на шаблоне «крест»; для дискретизации по времени в обоих уравнениях использовался явный метод Эйлера.

Модели BR и YNI

Для дискретизации по пространству в уравнении на мембранный потенциал V использовалась формула на шаблоне «крест». Для дискретизации по времени в данном уравнении и в уравнении на ионную концентрацию – явный метод Эйлера.

Для дискретизации по времени в уравнениях на воротные переменные m_i использовался метод Раша-Ларсена [ссылка] – широко распространенная практика при решении задач кардиологии.

Вообще, не смотря на свою простоту, метод Эйлера является одной из самых эффективных схем для численного решения уравнений данной области [ссылка1, ссылка2, …].

Программная реализация на GPU

Часть программы (выделение памяти, задание начальных условий, перенос данных на GPU) выполняется на CPU, основные ресурсоемкие вычисления – на GPU. При распараллеливании программ используются термины host (для управляющего CPU) и device (для GPU). В GPU-программе можно выделить основные этапы:

1. Выделение памяти на host и device
2. Инициализация начальных условий на host
3. Перенос начальных условий с host на device
4. Запуск функции-ядра на device, проводящую расчеты
5. Выгрузка результатов обратно на host
6. Пост-обработка результатов на host.

Подробное описание GPU-алгоритма

В программе каждому узлу расчетной сетки ставится в соответствие своя нить, проводящая вычисления всех ассоциированных с узлом сетки (элементарным объемом) переменных. Алгоритм:

1. Расчет начальных (стационарных) значений ионных концентраций.
2. Инициализация начальных условий
3. Цикл по времени. На каждом шаге по времени:
    a. Расчет вектора g^{n+1}, реакционной части уравнения на V^{n+1}, $[c]^{n+1}$
    b. Расчет «диффузионной» части уравнения на V^{n+1}
    c. Вычисление граничных условий для V^{n+1} (нулевой ток): ghost-ячейки
    d. Вывод V^{n+1} на host и печать в файл
4. Очистка памяти на host и device
5. Пост-обработка результатов.

Algorithm 1 Generalized code block that is evaluated at each time step to compute transmembrane potential (Vm).

```
for Xstep = 1 → Nx do
  for Ystep = 1 → Ny do
    brgates(); //update ionic gating equations
    brcurrents(); //update ionic currents
  end for
end for
bcs(); // set ghost nodes' potentials
for Xstep = 1 → Nx do
  for Ystep = 1 → Ny do
    Vmdiff(); //update diffusion terms for next time step
  end for
end for
```

Рис. 1 Псевдокод последовательной программы [СТОРОННИЙ РИСУНОК].

Algorithm 2 The sequential program with OpenACC pragmas.

```
#pragma acc data copyin(constarr,D,Dp,Afield) copy(datarr)
for T = 0 → final do
  #pragma acc loop independent vector(32) worker(2) gang(256)
  for Xstep = 1 → Nx do
    #pragma acc loop independent vector(32) worker(2) gang(256)
    for Ystep = 1 → Ny do
      brgates(); //update ionic gating equations
      brcurrents(); //update ionic currents
    end for
  end for
  #pragma acc parallel vector_length(1) num_workers(1) num_gangs(1)
  bcs(); // set ghost nodes' potentials
  #pragma acc loop independent vector(32) worker(2) gang(256)
  for Xstep = 1 → Nx do
    #pragma acc loop independent vector(32) worker(2) gang(256)
    for Ystep = 1 → Ny do
      Vmdiff(); //update diffusion terms for next time step
    end for
  end for
end for
```

Figure 4. The OpenACC version of the algorithm.
doi:10.1371/journal.pone.0086484.g004

Рис. 2 Псевдокод параллельной программы [СТОРОННИЙ РИСУНОК]

Оптимизация GPU-программы

Программа была написана на языке Си (C99) и организована следующим образом:

1. для каждой из переменных V, $\{g\}$, I_j, $[c]$ выделялся отдельный 1D-массив, представляющий собой «разложенный» по-горизонтали 2D-массив.
2. Размерности расчетной сетки были выбраны каждая кратная размеру warp – 32, чтобы при расчетах отсутствовали простаивающие ядра/нити.
3.

Результаты

[вступительный текст] Для первичного представления результатов были выбраны размерности сеток $\{256x256, 512x512, 1024x1024, 2056x2056\}$, взятые из работы [ссылка]; даже наименьшая из них является достаточно большой и подходящей для практических расчетов [ссылки]. Проведены замеры соответствующих времен выполнения.

Затем проведена серия коротких расчетов с большим числом размерностей, также сделаны замеры времен выполнения (выступающие в роли экспериментальных данных), теоретически получена формула (с параметрами) зависимости ускорения от размерности сетки, сделан fitting параметров для каждой из моделей (с помощью метода наименьших квадратов), и на основе последнего сделан вывод о степени верности теоретической формулы.

[Ремарки по представлению цифр:
в стате MKay2014:
--- ускорения указываются *целыми* числами, и фразами типа «more than 70x speedup, 100x speedup» – мне не нужно заботиться о серьезной точности цифр и подробном описании каждого значения ускорения в тексте
--- также указаны (совсем немного) абсолютные величины времен вычислений на CPU и GPU]

Модель AP

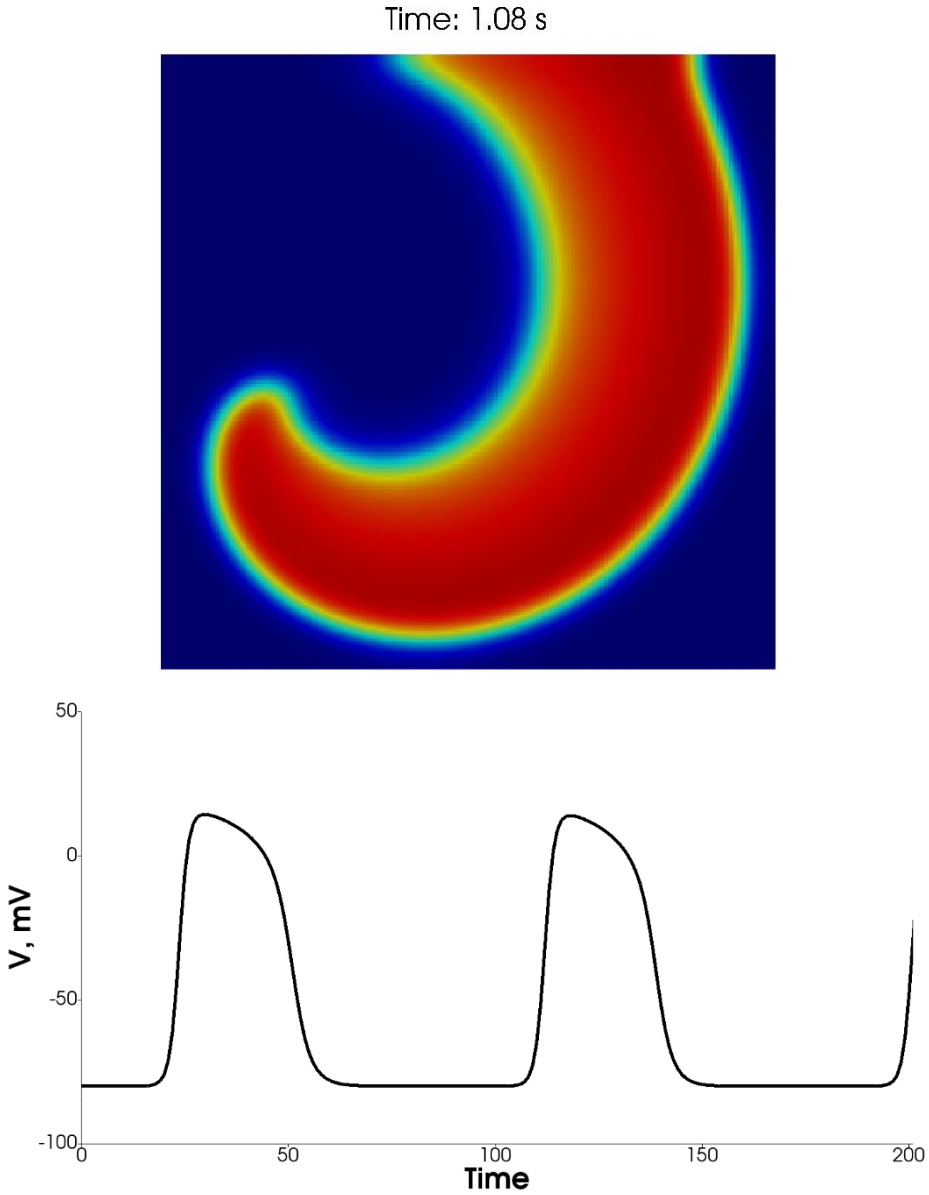


Рис. 3 Модель AP. Вверху: стоп-кадр мембранного потенциала V, на котором видно реентри. Внизу: динамика $V(t)$ в точке X [указать точку на рисунке].

[Текст с описанием графиков] С помощью программы визуализации результатов расчетов получен snapshot, на котором видна спираль реентри, практически идентичная по форме таковой в статье [статья AP модели], что подтверждает корректность расчетов на GPU. Также, рядом, изображена динамика мембранного потенциала в узле сетки с координатами (YY, ZZ), демонстрирующая последовательность из нескольких ПД, также практически идентичных таковой в статье [статья AP модели].

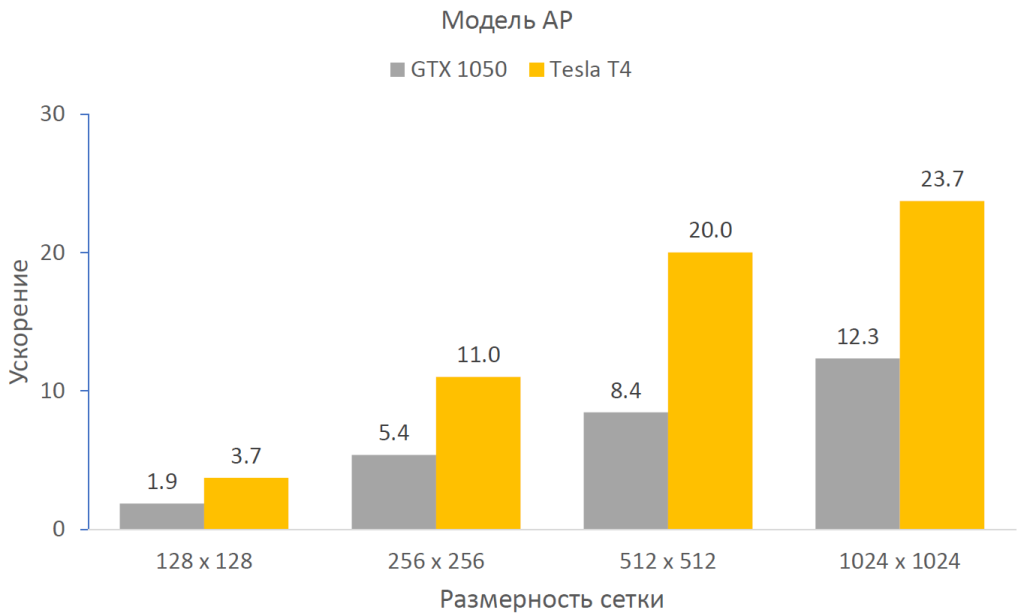


Рис. 4 Модель AP: зависимость ускорения от размерности сетки по пространству: 256 x 256, 512 x 512, 1024 x 1024, 2048 x 2048 (клеток), для 2-х моделей GPU.

[Текст с описанием графиков] На диаграмме можно видеть, что величина ускорения растет с увеличением размерности сетки, при этом значения ускорений для профессионального GPU выше таковых для любительского, что ожидаемо. Значения ускорений при данных размерностях сетки не превышают YY и ZZ для любительского и профессионального GPU, соответственно.

Модель BR

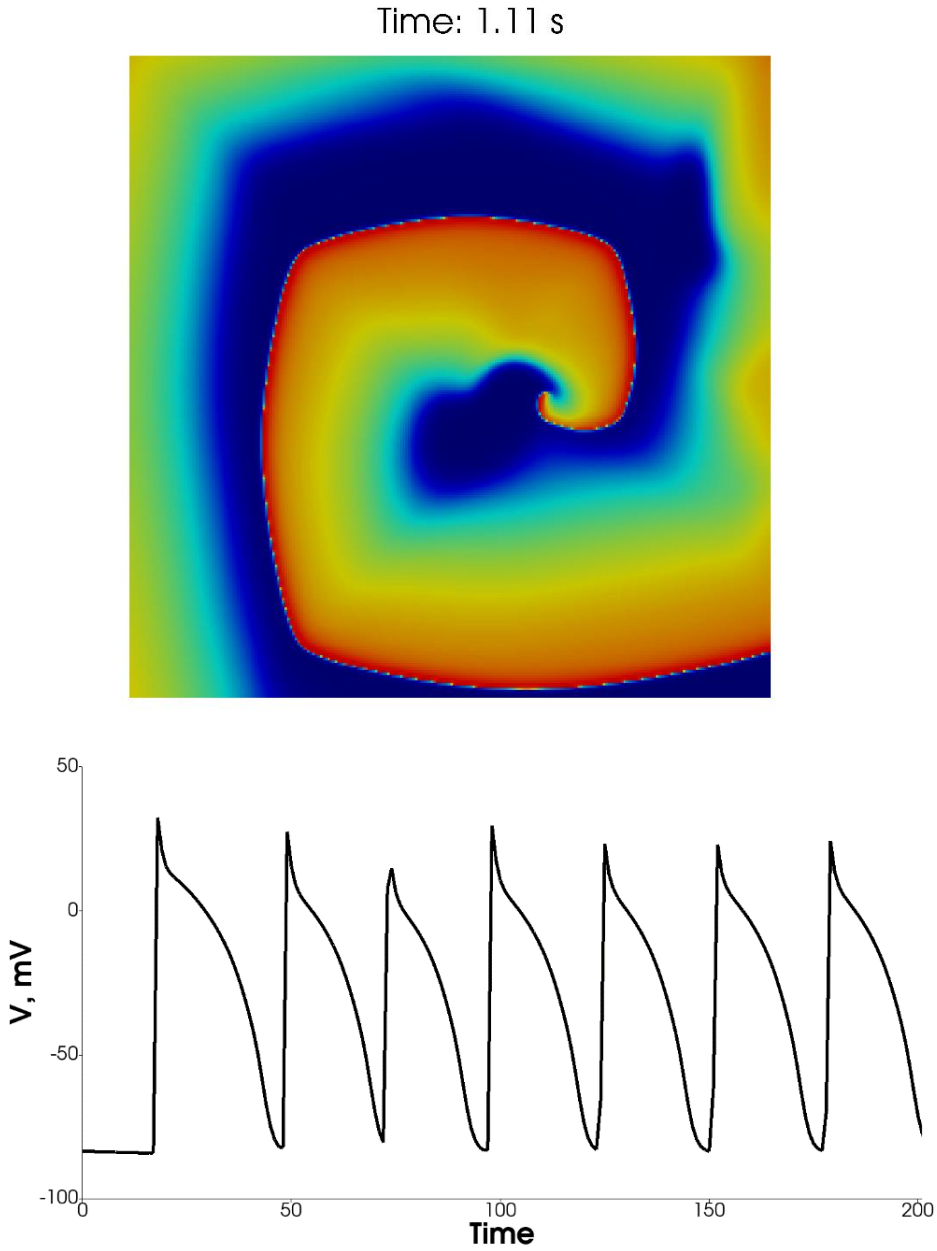


Рис. 5 Модель BR. Вверху: стоп-кадр мембранного потенциала V, на котором видно реентри. Внизу: динамика $V(t)$ в точке X [указать точку на рисунке].

[Текст с описанием графиков] С помощью программы визуализации результатов расчетов получен snapshot, на котором видна спираль реентри, похожая по форме на таковую в статье [статья cardiac_gpu_#2], что подтверждает корректность расчетов на GPU. Также, рядом, изображена динамика мембранного потенциала в узле сетки с координатами (YY, ZZ), демонстрирующая последовательность из многих ПД: они следуют практически без пауз друг за другом. Данная картина также похожа на таковую в статье [статья cardiac_gpu_#2].

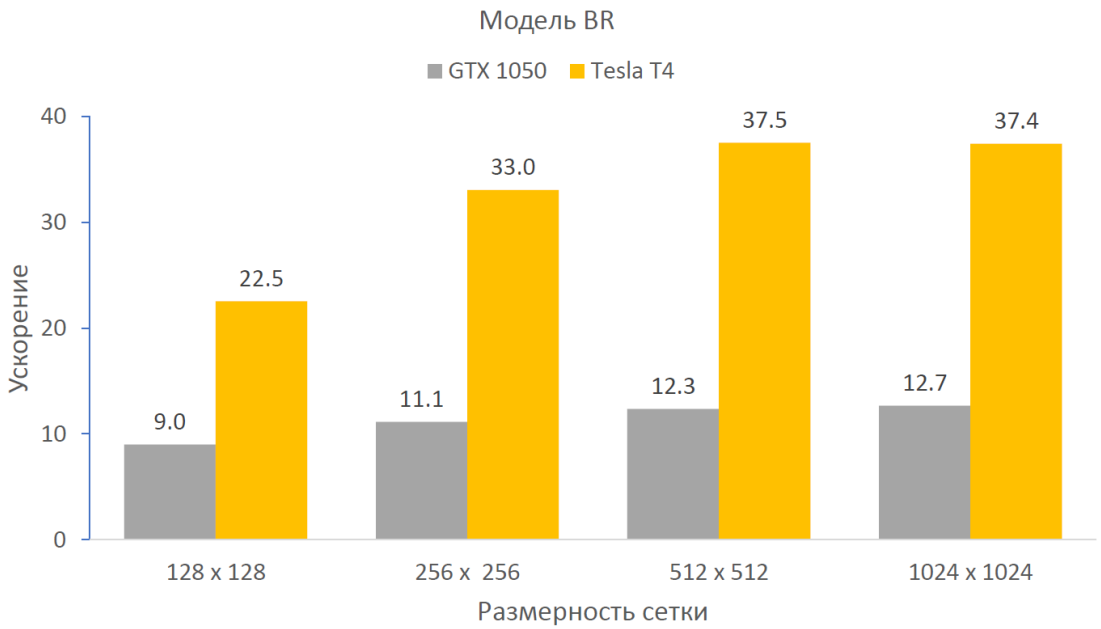


Рис. 6 Модель BR: зависимость ускорения от размерности сетки по пространству: 256 x 256, 512 x 512, 1024 x 1024, 2048 x 2048 (клеток), для 2-х моделей GPU.

[Текст с описанием графиков] На диаграмме можно видеть, что величина ускорения растет с увеличением размерности сетки, при этом значения ускорений для профессионального GPU выше таковых для любительского, что ожидаемо. Значения ускорений при данных размерностях сетки не превышают YY и ZZ для любительского и профессионального GPU, соответственно. Имеет место большее отношение ускорений между любительским и профессональным GPU [посчитать и вставить сюда, после пересчета в Google Colab], чем для модели AP.

Модель YNI

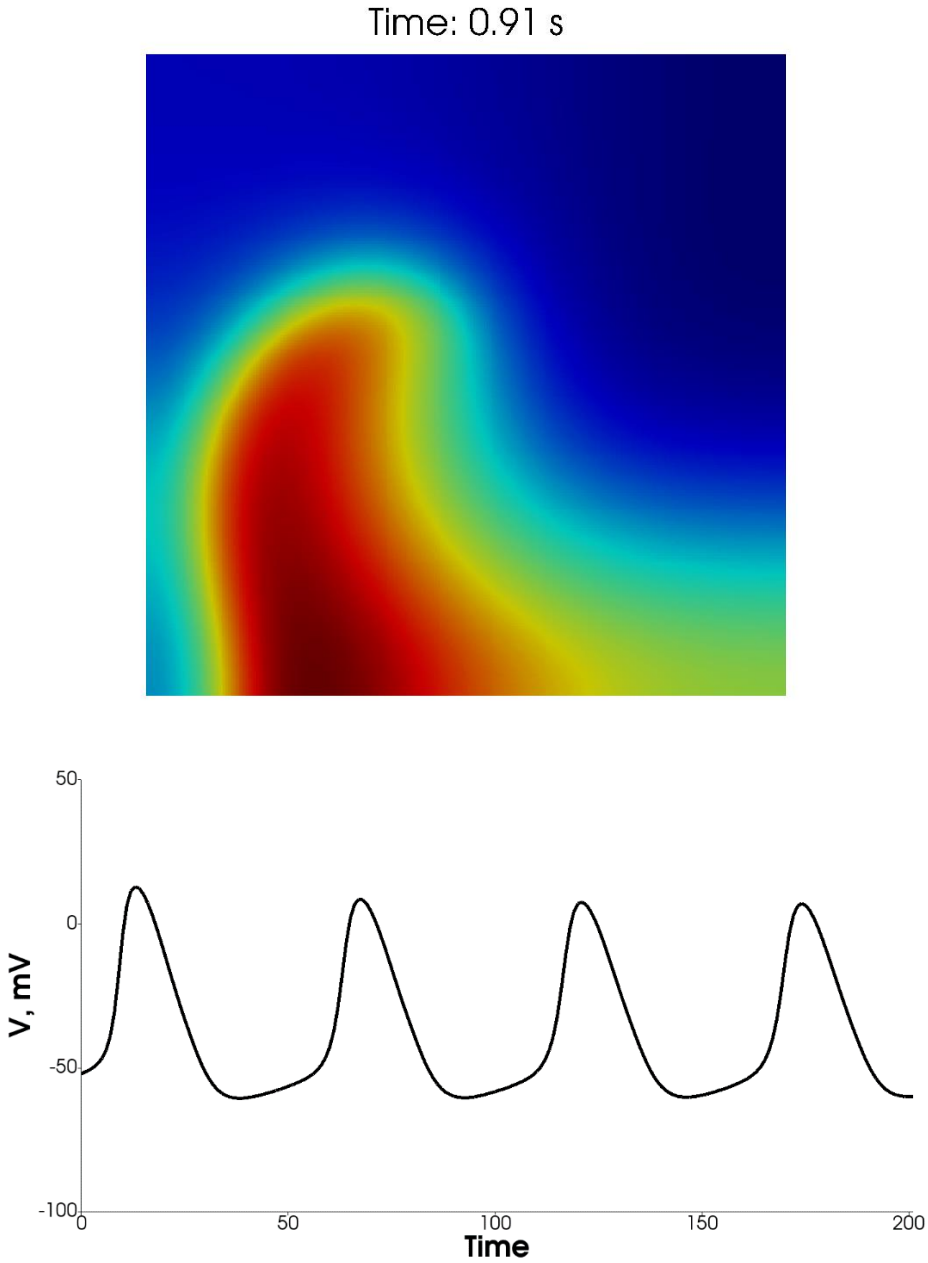


Рис. 7 Модель YNI. Вверху: стоп-кадр мембранного потенциала V, на котором видно реентри. Внизу: динамика $V(t)$ в точке X [указать точку на рисунке].

[Текст с описанием графиков] С помощью программы визуализации результатов расчетов получен snapshot, на котором виден рукав (вместо спирали, см. «Обсуждение») реентри, похожий по форме на таковой в статье [статья Романа 2016], что подтверждает корректность расчетов на GPU.

Также, рядом, изображена динамика мембранного потенциала в узле сетки с координатами (YY, ZZ), демонстрирующая последовательность из нескольких ПД: в каждом из них присутствуют фазы медленной деполяризации (кроме первого ПД),

деполяризации, реполяризации, гиперполяризации. Несмотря на связанность клеток, данная картина похожа на картину активности изолированной клетки СУ [cardiac_pacemaking_review].

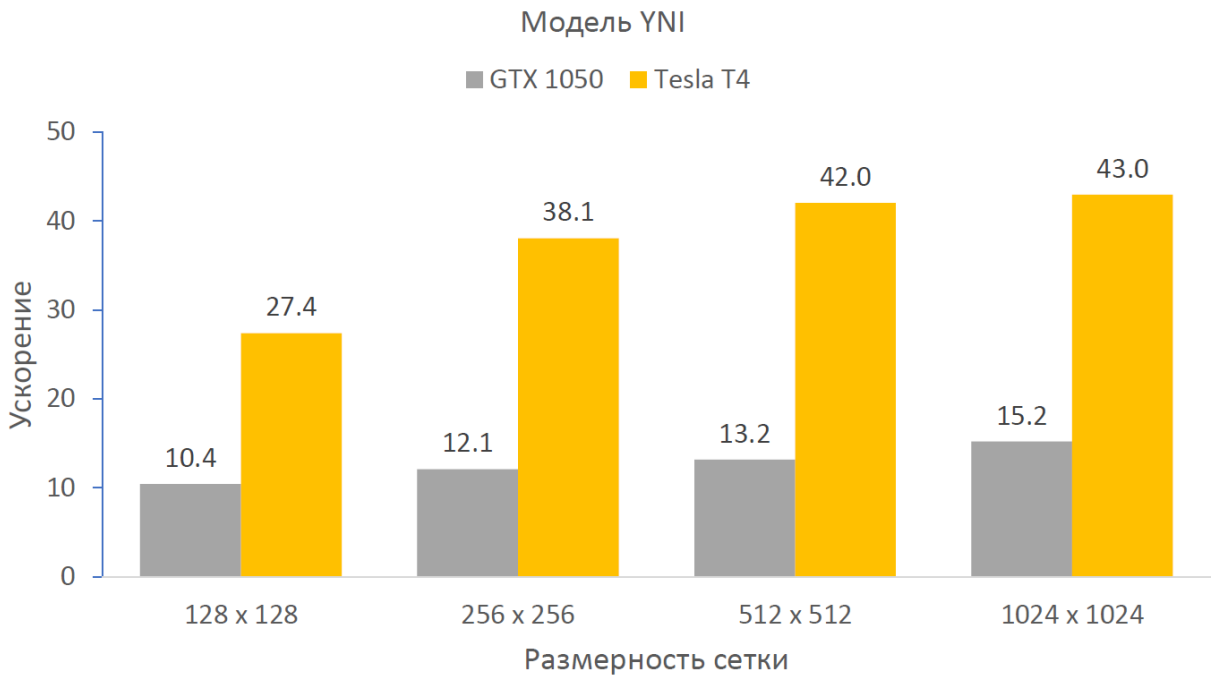


Рис. 8 Зависимость ускорения от размерности сетки по пространству: 256 x 256, 512 x 512, 1024 x 1024, 2048 x 2048 (клеток), для 2-х моделей GPU.

[Текст с описанием графиков] На диаграмме можно видеть, что величина ускорения растет с увеличением размерности сетки, при этом значения ускорений для профессионального GPU выше таковых для любительского, что ожидаемо. Значения ускорений при данных размерностях сетки не превышают YY и ZZ для любительского и профессионального GPU, соответственно. Имеет место большее отношение ускорений между любительским и профессиональным GPU [посчитать и вставить сюда, после пересчета в Google Colab], чем для модели AP. Имеются небольшие отличия между аналогичными отношениями для модели BR [посчитать и вставить].

Анализ графиков ускорений

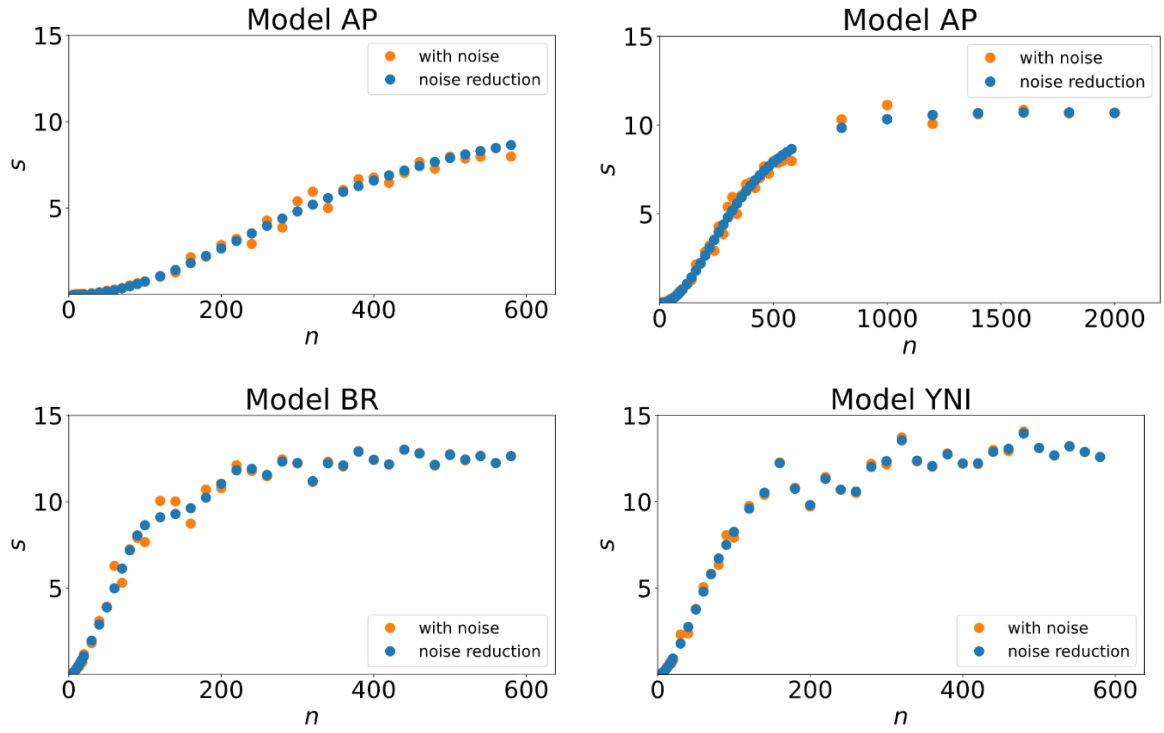
~~Мотивация: более глубокое исследование зависимости ускорения от размерности сетки и сравнение с теоретическими оценками.~~

Для получения значений ускорения при большом числе различных размерностей для каждой модели была проведена серия коротких расчетов и замерены времена исполнения последних (их мы будем называть «экспериментальными данными»). При замере времени исполнения программы возможно interfering процессов ОС компьютера, поэтому в данных будет присутствовать неотрицательный шум. Поэтому авторы решили уменьшить шум с помощью МНК.

Экспериментальные данные: “with noise” – исходные и “noise reduction” – обработанные. По определению:

$$s^{(k)}(n) = \frac{t^k_{CPU}(n)}{t^k_{GPU}(n)}, \text{где } k = \text{with noise, noise reduction.}$$

Значения $\{t^{noise\ reduction}_{CPU/GPU}(n_i) = sp(n_i)\}$ – вычислялись как значения аппроксиманта в виде кубического сплайна $sp(n)$, коэффициенты которого находились методом сглаживания (схож с методом наименьших квадратов). [для сплайна не нашел нахождение коэффициентов методом наименьших квадратов] [нужно ли «устранение» шума вообще? Это может вызвать лишние вопросы у рецензента] [подумать, как вычислять коэфф сплайна с помощь МНК, а не непонятного метода сглаживания]



По данным графикам можно отметить следующее:

- Для всех моделей имеется выход практически на константу при больших n. При этом имеются колебания в процессе выхода на константу, что особенно сильно видно для моделей BR и YNI
- [?] При малых n [вставить соответствующий увелич фрагмент графика?] точки стремятся к нулю.

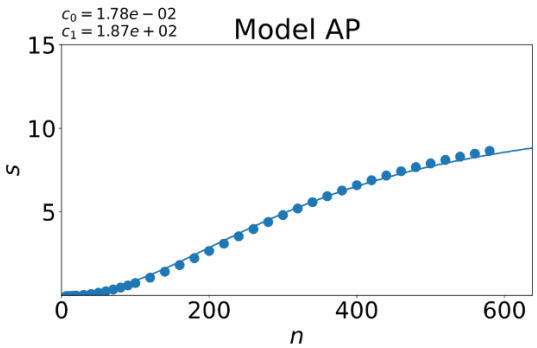
Далее везде использовались обработанные значения.

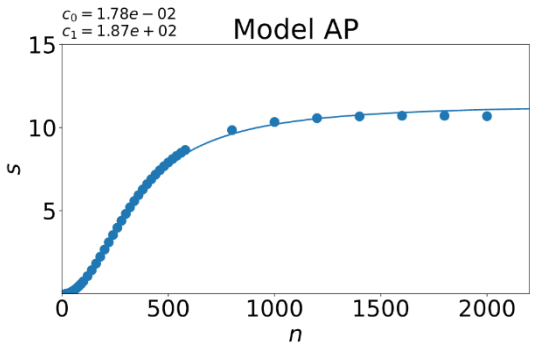
Теперь получим теоретическую формулу для ускорения в зависимости от числа узлов по оси x. При этом будем учитывать фактор копирования данных с CPU на GPU и обратно, при этом приняв, что время копирования не зависит от размера данных [ссылка на статью про закон Амдала для GPU]. Теоретическая формула:
[вставить вывод формулы] [вставить другой вид формулы, с единицей в числителе]

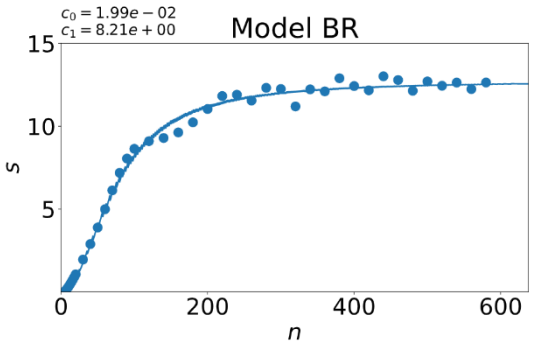
$$s(n; c_0, c_1) = \frac{c_0 n^2}{CEILING(n^2/M) + c_1} \text{ .[вставить вывод формулы]}$$

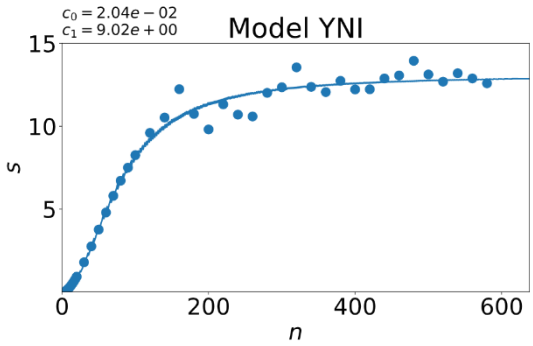
$$\text{где } c_0 = c_0(CPU, GPU) \text{ – не зависит от } model \text{ ,}$$
$$c_1 = c_1(model, CPU, K, J, hardware), \qquad (*)$$

Можно отметить, что при $n \to 0$ $s \to 0$ и $n \to \infty$ $s \to const$ (при любых значениях параметров c_0, c_1), что отражает экспериментальную зависимость.
Для вычисления параметров c_i можно воспользоваться методом наименьших квадратов (МНК). Графики получившегося аппроксиманта приведены на Рис. YY.









[значения коэффициентов --> в таблицу]

Можно заметить, что полученные кривые «хорошо» (хотя бы визуально) ложатся на экспериментальные точки:

- Модель AP: видно практически идеальное наложение при любых n
- Модель BR и YNI: заметно практически идеальное наложение при $n < 100$; затем, при увеличении n, имеет место колебание точек относительно кривой (что можно списать на большой шум в данных или некоторую неточность теоретической формулы).

Также оценим визуально асимптотику кривых: при $n \to 0$ $s \to 0$ и при $n \to \infty$ $s \to const$, как и должно быть согласно теоретическим оценкам.

Значения коэффициента c_0, который согласно теории должен быть одинаковым для всех моделей, немного ([вставить соответств. кусок текста из другого файла]) различаются. Скорее всего, здесь играет свою роль *фундаментальная неточность МНК* и учет не всех факторов в теоретической формуле.

Приведенные анализ позволяет говорить о большой степени удовлетворительности использования теоретической формулы для ~~априорной оценки ускорения программы~~ аппроксимации зависимости ускорения от размерности сетки. ~~При этом, значения для коэффициентов c_t нужно вычислять явно по формулам (*).~~ Для каждой модели и вычислительной системы значения коэффициентов будут различаться. [этот абзац – заключения; подумать, какие заключения сделать вместо написанных].

Обсуждение

С точки зрения числа арифметических операций, численное решение моделей BR и YNI практически идентично. Технически, в модели YNI пейсмейкерный ток можно рассматривать как ток стимуляции, зависящий не от времени, а от вектора переменных системы; также, в модели YNI на 1 больше ионных токов. Мы хотели бы повторить, что выбор моделей был *формальным*, чтобы покрыть весь спектр моделей кардиоцитов.

В численной схеме мы не использовали популярный метод *расщепления по физическим процессам*: это приводит к значительному увеличению громоздкости программной реализации при неочевидном приросте производительности. Плюс к этому, классический метод Эйлера также используется во многих современных работах [ссылки]. TODO

Мы не включали в замеры времени процесс вычисления начальных условий для инициирования реентри: для этого могут использоваться различные протоколы [ссылка] и технические способы (чтение из файла, вычисление в расчетной программе «на лету» и пр.); кроме этого, начальные условия могут быть произвольными, т.е. задаваться согласно произвольному протоколу. Поэтому требовалось исключить данную вариабельность из замеров времени и оставить только замер времени выполнения timestepping-цикла, одинакового для любых начальных условий.

В данной работе мы рассматривали миокард как *сплошную* среду, что является широко распространенной аппроксимацией, используемой при моделировании в кардиологии. При моделировании относительно больших участков ткани желудочка (модели AP и BR) данная аппроксимация является хорошей; однако, размер синусового узла мал (порядка 20x5 мм [Mathematical models of cardiac pacemaking function]) по сравнению с размером жедудочков и его ткань состоит из клеток различного типа, поэтому требуется переход к модели *дискретной* среды [ссылки]. При этом алгоритм расчетного цикла не поменяется, а следовательно, полученные значения ускорений можно считать универсальными, также соответствующими программам, реализующих алгоритм расчета дискретной среды. [привести вывод формулы для коэффициента диффузии]

Оптимизации GPU

Для большей оптимизации в технологии OpenACC имеются и другие директивы (например cache, atomic), а также опции тонкой настройки исполнения циклов (gang, worker, vector) [ссылки]; рассмотрение требует более глубокого знания архитектуры GPU и выходит за рамки данной статьи. [упомянуть больше директив для солидности текста?]. Применение данных оптимизаций поможет приблизиться к эффективности CUDA-кода.

Технологии программирования

Программа была написана с использованием/помощью [именно «с помощью» верно] процедурного подхода на языке C(99): для написания расчетных программ учеными данный язык используется чаще [примеры] языка C++.

Полученные показатели эффективности параллельных программ (ускорения ~2X раз) получились «средними», но ожидаемыми [ссылка на учебную презентацию с разбором распараллеливания метода Якоби]. При задействовании технологий CUDA/OpenCL возможно получить большую эффективность, но ценой значительно более долгого процесса программирования и низкой степени читабельности получаемого программного кода.

Технология OpenACC развивается: в последнее время ([указать год]) появляются все новые опции (например, unified memory), позволяющие еще больше упростить процесс распараллеливания, приближая сложность использования к таковой для технологии OpenMP. В будущем [дата] планируется объединить обе упомянутые технологии в одну [ссылка с proofom].

Дополнительное ускорение проще всего получить экстенсивным путем, незначительно модифицировав программу для выполнения расчетов на *гибридных* архитектурах (узел, состоящий из нескольких GPU под управлением многоядерного CPU; подобные узлы, объединенные в MPI-кластер). [ссылки на статьи MPI + OpenMP + OpenACC].

Заключение

Степень сложности детальных моделей кардиоцитов постоянно возрастает, в них учитываются все новые физиологические процессы. Для эффективного моделирования жизненно необходимы развитие параллельных алгоритмов и наращивание вычислительных мощностей.

В работе продемонстрировано использование технологии OpenACC. Авторы показали, что технология позволяет значительно ускорить процесс распараллеливания программ на GPU по сравнению с технологией CUDA. В итоге, это уменьшит полное время до получения результатов, жизненно важных в медицинской практике. [звучит как реклама, отчасти]. При успехах в развитии моделей в ближайшем будущем, появится возможность применять компьютерное моделирование в терапевтической практике (персонализированная медицина): для сохранения здоровья, а иногда и жизни пациента, в клинике потребуется проводить моделирование всего сердца практически в режиме реального времени.

TODO

Список литературы

1. CUDA Full Manual.
2. OpenACC Full Manual.
3. OpenACC best practices guide.
4. OpenACC – обзорная статья.
5. Р.Р. Алиев. Концептуальные и детальные модели электрической активности миокарда.
6. GPU Accelerating 2D-heat equation solution.
7. https://devblogs.nvidia.com/parallelforall/openacc-example-part-1/.
8. https://devblogs.nvidia.com/parallelforall/getting-started-openacc/.
9. Fast Acceleration of 2D Wave Propagation Simulations Using Modern Computational Accelerators.
10. GPU Accelerating 2D-heat equation solution.
11. http://www.cs.otago.ac.nz/pmam2014/slides/Xu.pdf.
12. Exploring Programming Multi-GPUs using OpenMP & OpenACC-based Hybrid Model.
13. …