Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра системного программирования

# Фреймворк для распараллеливания модели масс-с-пружинками под различные архитектуры

Курсовая работа студента 344 группы

*Гудиева Артура Владимировича*

Научный руководитель …….....  С.Н. Николаев

аспирант  /подпись/

# Оглавление

# Введение

Биомоделирование является в настоящее время одним из самых актуальных направлений в научных исследованиях. Моделирование хирургических операций позволяет заранее рассчитать необходимые для врачей параметры, наглядно представить, как будет выглядеть объект после хирургического воздействия. Ресурсоемкость моделирования влечет необходимость распараллеливания вычислений для экономии времени. Часто при моделировании используют модель масс-с-пружинками.
Имеется уже реализованный модуль масс-с-пружинками, в котором ведутся лишь последовательные вычисления, но есть возможность реализовать параллельные.

# Актуальность

Проект кафедры системного программирования “Биомоделирование” использует для расчетов хирургических операций модель масс-с-пружинками, при этом основные вычисления при моделировании ведутся последовательно.

Большинство датацентров предлагают вычисления лишь на центральных процессорах. Необходимо понять, какое будет ускорение при вычислениях на графических картах.

# Постановка задачи

При моделировании хирургических операций используется уже реализованный модуль модели масс-с-пружинками с последовательными вычислениями. Необходимо:

- Провести профилирование имеющегося модуля масс-с-пружинками
- Реализовать фреймворк для распараллеливания модели масс-с-пружинками
- Измерить время вычислений на OpenMP, CUDA и OpenCL
- Сравнить полученные результаты

# Обзор существующих решений

На данный момент существуют следующие методы использования параллельных вычислений при биомоделировании с использованием модели масс-с-пружинками:

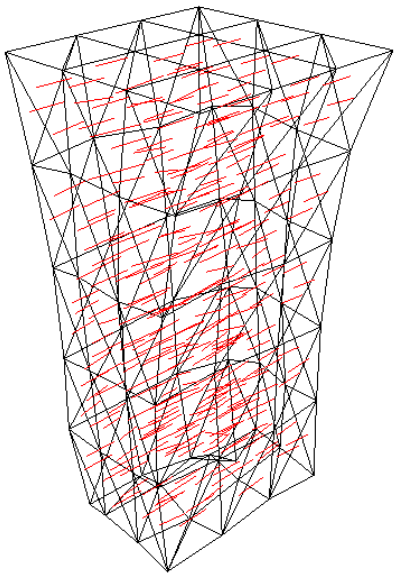
- Биомоделирование с использованием графического процессора общего назначения (GPGPU), ориентированное на отдельные биологические органы (сердце, печень....). При использовании данного метода возникают задержки по времени при последовательной обработке информации.[1]
- Биомоделирование с использованием центрального процессора (CPU), также напраленное лишь на отдельные биологические органы. Данный подход проигрывает по производительности при массивно параллельных вычислениях.[2]

В данной работе будет описан метод моделирования, позволяющий воспользоваться преимуществами двух вышеперечисленных подходов, уменьшая при этом их недостатки. Будут рассмотрены варианты моделирования под различные архитектуры (OpenMP, OpenCL и CUDA) с использованием графических вычислений, вычислений на центральном процессоре, а также гетерогенных вычислений. Будет произведено сравнение всех вариантов, также будет составлен оптимальный способ моделирования с использованием модели масс-с-пружинками.

# Теоретическая основа курсовой работы

## Модель масс-с-пружинками

При использовании данной модели объект разделяется на вершины-массы, которые соединяется между собой пружинками. Каждая вершина имеет свои координаты, силу, которая действует на нее, ускорение, также она может быть зафиксирована. Каждая пружинка в свою очередь имеет собственный набор параметров: вершины, которые она соединяет, начальную длину, упругость и вязкость. [3]



## Обзор архитектур

В курсовой работе будут рассмотрены наиболее распространенные архитектуры, такие как OpenMP, OpenCL и CUDA.

OpenMP — открытый стандарт для распараллеливания программ, написанных на языках C, C++ и Fortran. Ключевыми элементами OpenMP являются конструкции для создания потоков, распределения работы между потоками, для управления работой с данными, для синхронизации потоков, процедуры библиотеки поддержки времени выполнения и переменные окружения. Преимуществами OpenMP являются заложенная идея «инкрементального распараллеливания» и достаточно гибкий механизм, предоставляющий разработчику большие возможности контроля над поведением параллельного приложения.[4]

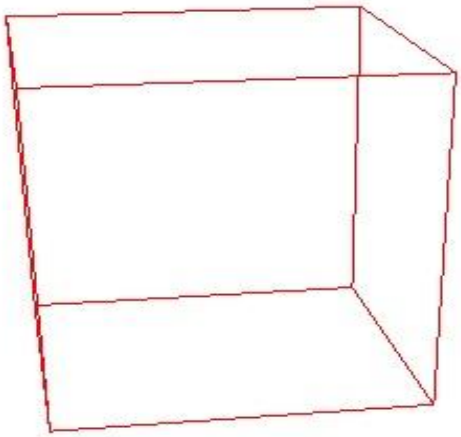
OpenCL — фреймворк для написания программ, связанных с параллельными вычислениями на различных графических и центральных процессорах. К особенностям OpenCL можно отнести отсутствие поддержки указателей на функции, рекурсии, битовых полей, массивов переменной длины, стандартных заголовочных файлов, а также расширения языка для параллелизма: векторные типы, синхронизация. OpenCL содержит в себе основные преимущества гетерогенных вычислений, такие как возможность увеличить скорости вычислений в десятки раз, сократить время выполнения расчетов и уменьшить количество необходимых для расчетов узлов.[5]

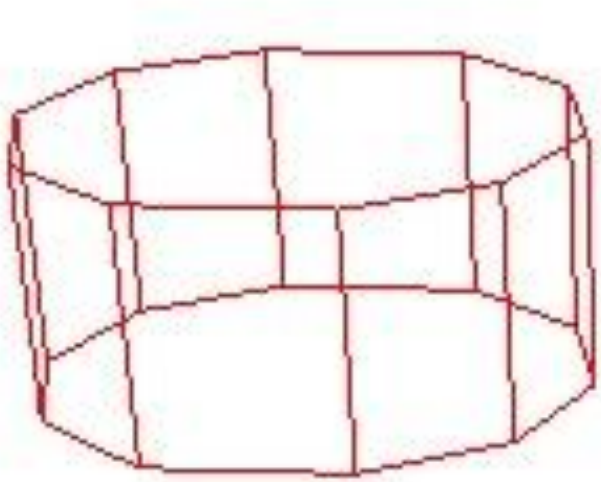
CUDA — программно-аппаратная архитектура параллельных вычислений, с использованием графических процессоров. Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью. К преимуществам можно отнести эффективные транзакции между памятью центрального процессора и видеопамятью, полную аппаратную поддержку целочисленных и побитовых операций, поддержка компиляции GPU кода средствами открытого LLVM.[6]

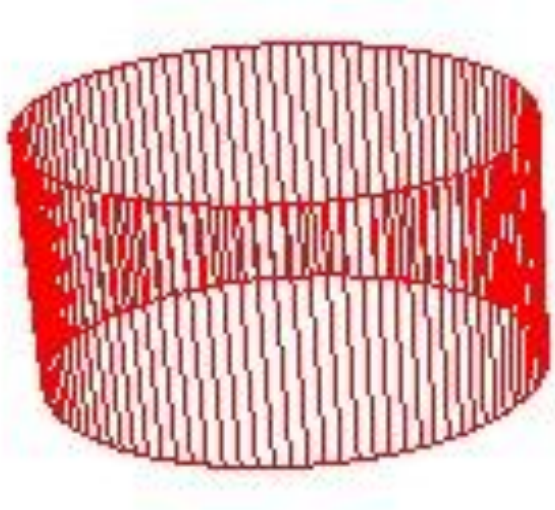
# Реализация

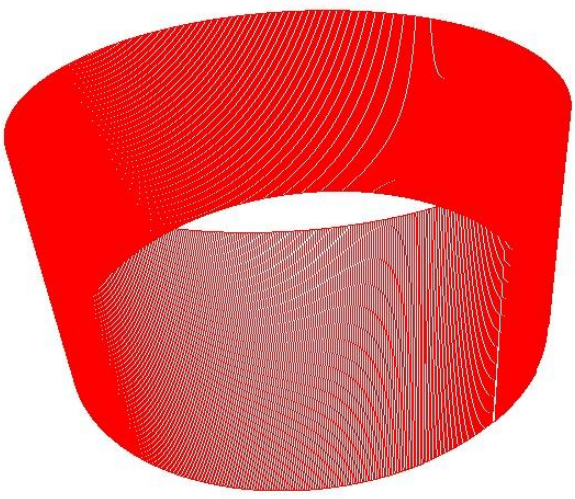
## Рассмотренные объекты и операции

При реализации фреймворка были рассмотрены простейшие геометрические фигуры, такие как куб и различные призмы с 20, 200 и 2000 вершинами, половина из которых была зафиксирована.

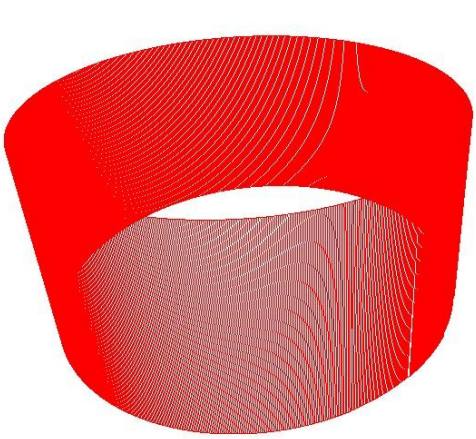


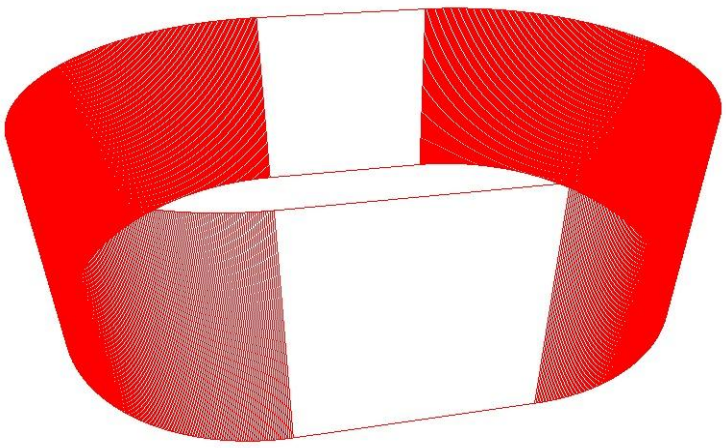






Ко всем вершинам применялась заранее заданная постоянная сила , поэтому получились простейшие операции: растяжение и сжатие.





# Структура модуля модели масс-с-пружинками

В модуле все основные вычисления происходят итеративно. Реализованный модуль масс-с-пружинками состоит из следующих основных компонентов.

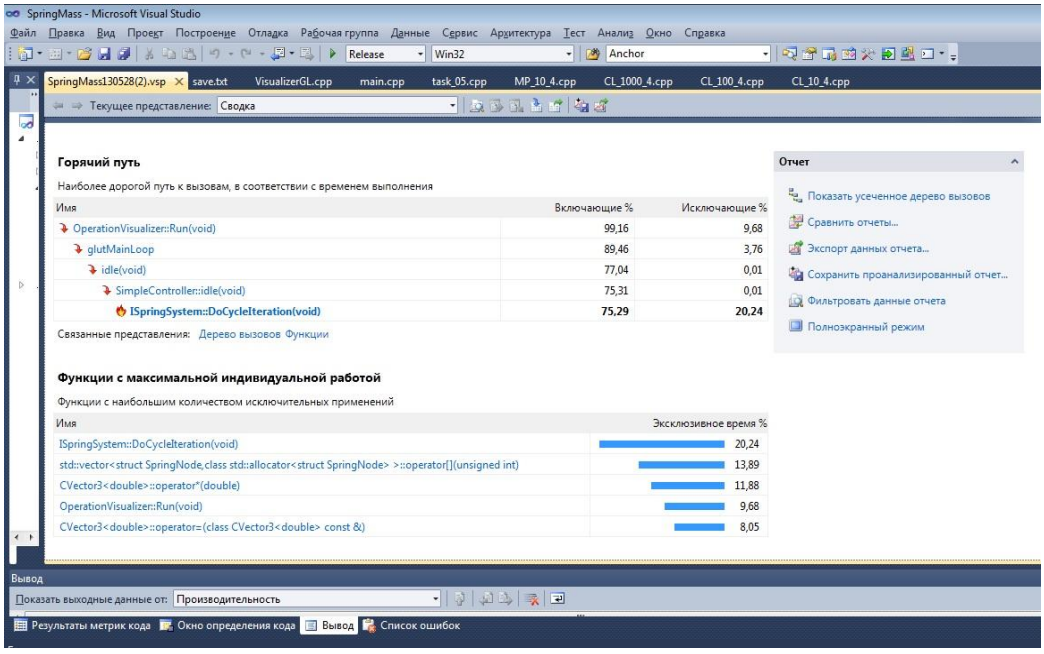
Процедура DoCycleIteration(). Она предназначается для систематического пересчета параметров вершин и соединяющих их пружин в течении заданного периода времени.

Процедура ConnectSprings(). При вызове этой процедуры задаются начальные данные для вершин и пружинок.

Процедура DrawSystem(). С её помощью объект изображается на экране.

# Профилирование

С помощью анализа мастера производительности Visual Studio было осуществлено профилирование данного модуля способом инструментирования, то есть измерялось количество и время вызовов каждой функции. Профилирование применялось при различном количестве вершин объекта: 8, 20, 200, 2000…




В результате оказалось, что при возрастании количества вершин с какого-то момента (примерно при 250-300) наиболее затратной оказывается процедура DoCycleIteration(), следовательно её и нужно распараллелить.

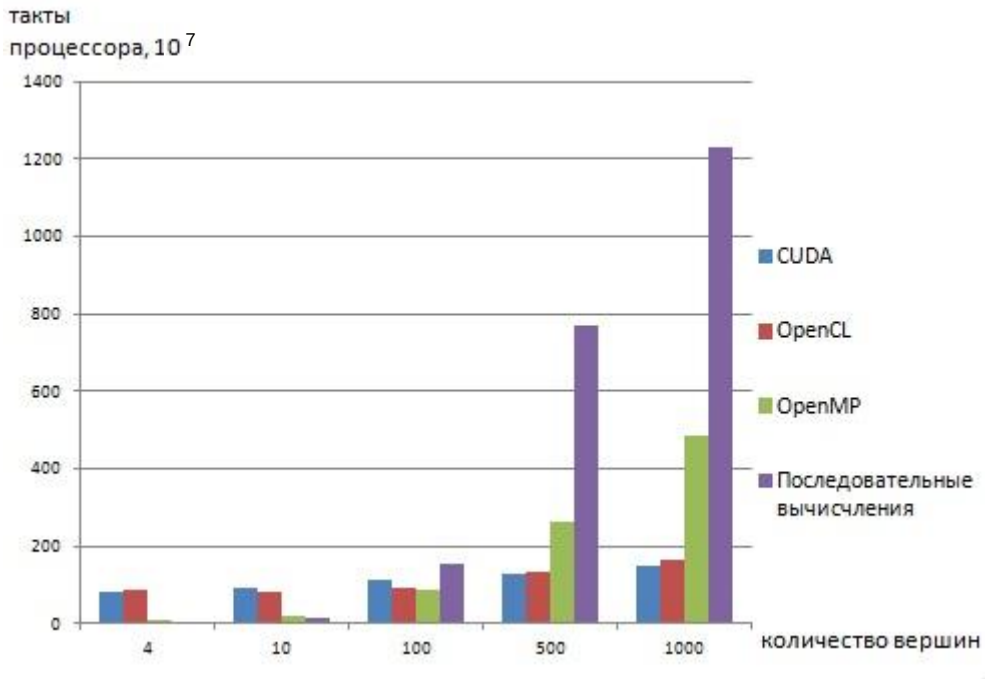
Псевдокод процедуры DoCycleIteration[7]:

Algorithm 1 DoCycleIteration(size)

```
1: time ← cureTime
2: for k ← 1 to Nodes.size() do
3:     if Nodes[k].isFixed = false then
4:         Nodes[i].position ← newPosition
5:         Nodes[i].velocity ← newVelocity
6:     end if
7: end for
8: sum ← sum + (cureTime − time)
```

# Полученные результаты

Полученные временные результаты можно представить в виде гистограммы:




По данной гистограмме можно сделать следующий вывод : при количестве вершин у объекта, меньшем 80, параллельные вычисления не нужны. От 80 до 250 следуют использовать OpenMP. От 250 до 1000-1500 целесообразно использовать OpenCL. При количестве вершин объекта, большем 1500, лучше всего подходит CUDA.

**Итог**:

- Осуществлено профилирование модуля масс-с-пружинками
- Реализован фреймворк для распараллеливания модели масс-с-пружинками
- Сделано сравнение последовательных вычислений, вычислений на OpenMP, OpenCL и CUDA
- Составлен оптимальный способ моделирования

# Ссылки

[1] A GPU Accelerated Spring Mass System for Surgical Simulation
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.2423

[2] GPU Accelerated Surgical Simulators for Complex Morphology

http://hgpu.org/?p=1758

[3] Spring Mass, formulaes, definitions

https://en.wikipedia.org/wiki/Effective_mass_(spring%E2%80%93mass_system)

[4] OpenMP, SPECIFICATIONS FOR PARALLEL PROGRAMMING

http:// http://openmp.org/wp/

[5] OpenCL, The open standard for parallel programming of heterogeneous systems

http://www.khronos.org/opencl/

[6] CUDA, Parallel Programming and Computing Platform

http://www.nvidia.com/object/cuda_home_new.html

[7] Source code

https://github.com/ArturGudiev/SpringMass