**Задание самостоятельной работы студента**

В рамках выполнения данной самостоятельной работы Вам будет необходимо доработать и отладить программное обеспечение выполняемое на ПК с МП Intel, оценить точность решения задачи, проанализировать ее производительность, оптимизировать программное обеспечение по рекомендациям от инструментария oneAPI, спрогнозировать увеличение производительности при использовании гетерогенной системы, определить узкие места реализации ПО, дать оценку составляющих производительности приложения.

**Постановка задачи**

Необходимо модернизировать и оптимизировать программу, выполняющую симуляцию гравитационного взаимодействия n-тел для трехмерного пространства (n-body).

Задана система из тел, каждое из которых обладает массой , и эти тела, пребывая в вакууме, попарно взаимодействуют друг с другом согласно закону тяготения Ньютона в течение времени :

- сила гравитационного взаимодействия, оказываемая на -е тело со стороны системы. Где (6,67430(15)⋅10−11 Н·м²·кг−2),  *—* масса тела, применительно к которому действует рассматриваемая сила, массы остальных тел системы, радиус-вектор, описывающий положение тела в пространстве и скорость i-го тела соответственно.

Необходимо обновить программу, моделирующую эволюцию динамической системы (изменение положения тел в пространстве), которую можно описать с помощью системы дифференциальных уравнений:

Поскольку нам нужно определить, как изменится положение тел в пространстве с течением времени, и согласно второму закону Ньютона , то получаем систему следующего вида:

На данный момент, в общем виде, для , решена данная задача может быть только численно. В простейшем случае приведем пример решения описанной выше системы методом [Эйлера](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0):

Тогда для нашей системы получим следующую схему решения:

Где i, j – обозначение i-го и j-го тела в системе тел, а k – обозначение шага итерации численного метода.

То есть, если сравнивать исходную схему Эйлера с полученной схемой решения поставленной задачи получается, что

Вычисление заменяется на , то есть, на определение новых значений ускорения для всех тел системы. А последующее вычисление заменяется на вычисление значений и .

Для выполнения данных действий в программе уже реализованы две функции:

- **compute\_f(y, f)**

Вычисление временного объекта f, который в дальнейшем будет использоваться в update\_system для обновления состояния системы тел (тут под состоянием понимаются положения тел в пространстве, их координаты и значения скорости в k момент времени).

- **update\_system(src, dst, f, dt)**

Вычисление нового состояния системы – dst, относительно ее исходного состояния – src (в соответствии с методом Эйлера, вычисление ), с учетом вычисленных ускорений и скоростей на предыдущем шаге (это вычисленный объект f).

В начальный момент времени тела системы должны быть равномерно распределены в пределах единичного куба. Начальные скорости так же для простоты должны быть распределены равномерно в пределах единичного куба, центр которого совпадает с центром системы координат (по каждой из осей Евклидовой системы координат получается в диапазоне [-0.5;0.5]). Начальные ускорения тел равны 0.

**Метод решения**

Далее необходимо согласно варианту решить поставленную задачу с применением одного из следующих численных методов:

**Вариант 1.** Модифицированный метод Эйлера (МЭ);

**Вариант 2.** Метод Хойна 2 порядка (Х2);

**Вариант 3.** Метод Хойна 3 порядка (Х3);

**Вариант 4.** Метод Рунге-Кутты 4 порядка (РК4).

К примеру, рассмотрим метод Рунге-Кутты 4 порядка.

где h – dвеличина шага по x, что в контексте данной задачи соответствует шагу во времени.

Для уже представленного в программе метода Эйлера необходимо воспользоваться уже написанными функциями compute\_f (…) и update\_system (…).

Тогда для вычисления необходимо взять исходное состояние системы, массив particles, и вычислить f от них - **compute\_f (particles, k1)**.

Чтобы получить значение необходимо соответственно снова воспользоваться функцией compute\_f. Однако, что тут будет выступать в качестве исходного массива particles? Это уже не просто , а ). А как было описано ранее, для этого действия, вычисления нового состояния системы, используется функция - **update\_system**, где в качестве src массива будет выступать particles (), а в качестве dst будет выступать временный массив tmp:

**update\_system (particles, tmp, k1, dt/2), обратите внимание, используется не полный шаг h, а половинный.**

Тогда для вычисления можно выполнить **compute\_f (tmp, k2).**

После вычисления всех коэффициентов – их можно последовательно просуммировать с particles через update\_system, как это называется, in place, передавая particles как src и dst, а в качестве f по очереди все эти k с нужным значением dt (dt/6, dt/3, dt/3, dt/6).

Вспомогательные функции:

* ***compute\_k\_energy(…)***
* ***compute\_p\_energy(…)***
* ***compute\_impulse(…)***

Функции для вычисления кинетической и потенциальной энергии системы тел для проверки точности симуляции на каждом шаге. Исходя из закона сохранения энергии на каждом шаге нужно смотреть на изменения относительной погрешности при вычислении величины полной энергии системы. Потенциальная энергия рассчитывается как сумма потенциальных энергий каждой частицы в гравитационном поле других частиц. Вместе с этим нужно проверить сохранение величины импульса системы.

Моделирование взаимодействия тел выполнять в течение 1 секунды. Установление величины шага – на усмотрение студента. Естественно, с большим шагом мы получим большую точность, однако это приведет к увеличению времени выполнения алгоритма. В качестве примера рассмотрим разделение временного интервала на 10 частей, тогда каждый раз время моделируемой системы будет инкрементироваться на 1/10 секунды. H = 0,1 секунды. Кол-во тел, для которых производится симуляция гравитационного взаимодействия ≥ 10000.

Для решения задачи необходимо реализовать следующую систему классов и структур:

1. ***Body –*** структура, описывающая состояние одного тела, содержит поля:

* float pos\_x, pos\_y, pos\_z;
* float vel\_x, vel\_y, vel\_z;
* float mass;

1. ***Simulation*** – класс, описывающий состояние симулируемой системы, содержит поля: particles – массив тел. Методы класса: start – начать симуляцию, init\_system – проинициализировать систему начальными значениями. Соответственно, в методе start выполняются описанные выше базовые функции на протяжении n шагов, итераций численного метода.

Используемые константы:

* const double G = 6.67259e-11; // Гравитационная постоянная
* const double softeningSquared = 1e-6; // Используется для случая, когда расстояния между телами слишком малы, чтобы не получить 0 в знаменателе.

На каждом шаге для отладки необходимо выводить следующую информацию:

* номер шага;
* текущее моделируемое время (= номер шага \* размер шага);
* суммарная энергия системы тел и суммарный импульс системы тел;
* время, затраченное на итерацию (произведение вычислений).

**Выполнение по этапам**

Работа выполняется и сдается в 4 этапа, каждый из которых оценивается отдельно:

1. Модифицировать под свой вариант и отладить ПО, реализующее один из методов решения задачи, измерить эффективность его работы с помощью Intel Advisor, изменить точность данного решения (сравнить свой вариант с исходным методом Эйлера, показать, что ваш сходится лучше при том же кол-ве шагов. При необходимости увеличить кол-во тел системы). Модификацию выполнить путем комбинации блоков **compute\_f *()*** и **update\_system *()***.
2. Проанализировать код с помощью Intel Advisor, применить предлагаемые шаги по оптимизации, проверить эффективность обращения к памяти (memory level roofline with cache simulation), оценить возможности по распараллеливанию (выделить параллельные секции). На данном шаге необходимо выполнить оптимизацию программы и получить прирост производительности без использования параллелизма.
3. Реализовать параллельную версию метода с использованием средств OpenMP.
4. Предоставить отчет об эффективности выполнения параллельной версии алгоритма, проанализировать на наличие проблем в параллельной версии (Intel VTune + Intel Inspector)

**Вспомогательные материалы**

1. <https://en.wikipedia.org/wiki/N-body_problem>
2. <https://github.com/SandalovKY/n-body-example>
3. <https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods>
4. <https://en.wikipedia.org/wiki/Heun%27s_method>
5. <https://www.cfm.brown.edu/people/dobrush/am33/Mathematica/ch3/modify.html>
6. <https://en.wikipedia.org/wiki/List_of_Runge%E2%80%93Kutta_methods>
7. <https://mipt.ru/upload/medialibrary/87d/rk.pdf>