**Съдържание**

**1 Въведение 3**

**2 Анализ 3**

2.1 Анализ на проблема . . . . . . . . . . . . . . . . . . . . . . . 3

2.2 Анализ на подобни трудове . . . . . . . . . . . . . . . . 4

**3 Потребителски интерфейс 4**

**4 Проектиране, реализация и тестване 5**

4.1 Директен алгоритъм . . . . . . . . . . . . . . . . . . . . . . 5

4.2 fast nBody . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

4.2.1 Реализация . . . . . . . . . . . . . . . . . . . . . . 6

4.2.2 Анализ . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4.2.3 Тестване . . . . . . . . . . . . . . . . . . . . . . . . . 8

4.2.3.1 Тестване с измамни резултати. . . . 9

4.2.3.2 Тестване без измамни резултати. . 11

4.2.3.3 Тестване на отклонението . . . . . . . . 13

**5 Заключение 13**

**6 Източници 14**

1. **Въведение**

Формулировката на nBody проблемът е следната: Имаме n на брой космически тела разположени в пространството. В текущият момент от времето знаем техните позиции, скорост вектори (накъде се движат и с каква скорост), и маси. Изчислете в произволен момент от времето (след началният) какви са позициите на всички тела. Началните стойности и параметри, които знаем са нужни и достатъчни, за да можем да пресметнем очакваните в задачата стойности.

Някои от приложенията на способността за решаване на nBody проблемътса следните:

* Предсказване на сблъсъци между космически тела.
* Предсказването на стабилност на слънчеви системи – колко време една планета би останала в една система, така че да има нужната светлинна енергия, за да е годна за обитаване.
* Процеса на създаване на слънчеви системи – предвиждане на перспективни групи от тела, които може да образуват системи годни за живот.
* Предсказване на траектори за движение при космически мисии.
* Предсказване на движението на частици на молекулно ниво.

Поради голямата си значимост, проблемът е силно изследван. Съществуват множество трудове, които изполват различни апроксимации и техники, за да намерят максимално ефикасно решение на проблема, макар и неточно. В този документ е изследвано едно такова решение на проблемът, което ни дава изключително висока производителност и ефикасност за сметка на точността.

1. **Анализ**

**2.1 Анализ на проблема**

Аналитично решение на проблема съществува за 2 тела. За 3 и повече тела няма общо аналитчно решение (съществуват решения за n=3 при определени условия изпълнени за началната конфигурация на телата).

По тази причина проблемът се решава чрез числени методи. Решаването чрез числени методи води до някои предизвикателства свързани с естеството на проблема – на всяка стъпка трябва да се сметне как всяко едно тялко влияе на всяко друго. Това изключително ограничава възможността за паралелизация на алгоритъма. Поради тази причина много от утвърдените алгоритми, които решават този проблем, намалят тези зависимости като жертват прецизност в резултата, за да постигнат разумно време.

* 1. **Анализ на подобни трудове**

Съществуват два основни алгоритъма за решаване на nBody проблема:

* The Barnes-Hut Algorithm [1] [2] – иползва следната апроксимация, за да намали броят на изчисленията и да увеличи възможността за паралелизация: „Ако едно тяло е достатъчно далече от група от тела, то гравитационната сила, с коята те му влияят може да бъде апроксимирана с техният център на тежестта “. Използвайки quadtrees за разделяне на пространството на квадранти рекурсивно този алгоритъм успява да постигне O() за изчисление на на гравитационните сили в даден момент (вместо O() от директният алгоротъм). Quadtree поддържа дистанциите и центровете на тежестта в корена на всяко поддърво, за да може всяко тяло да трябва да обходи само нужната част от дървото, за да изчисли силите, които му действат.
* Fast multipole method [3] – Използва същата апроксимация като горният алгоритъм. Разликата е че използва multipole expansions вместо център на тежестта, за да изчислява гравитационните сили. Пак се използва quadtree, но в него се изчисляват multipole expansions на групите от тела, които се намират близко едно до друго.

1. **Потребителски интерфейс**

Имплементацията на алгоритъма описан в този документ е достъпна в Github[4]. Състои се от:

* Сорс код на алгоритъма, който намира решение на nBody проблема във nBody.cpp. Програмата може да бъде компилирана със следната команда - g++ -pthread -O3 -std=c++11 nBody.cpp. Използвани са само стандартни библиотеки – std::thread за многонишковото програмиране и синхронизационни примититиви предоставени от стандартната библиотека. Програмата приема аргументи от комадният ред подредени в реда, в който са описани:
  + Брой на нишките – по подразбиране 1
  + Брой на телата – по подразбиране 3
  + Име на файлът, в който да се генерира решението – по подразбиране nBody.txt
  + Начално време – по подразбиране 0
  + Крайно време – по подразбиране 20
  + Времева стъпка – по подразбиране 0.1
  + Омекотяване – константа, коята да омекотява сблъсъка между две тела, за да не „изхвърчат“ в различни посоки с висока скорост при сблъсък – по подразбиране 11.6
  + Гравитационна константа – по подразбиране 9.8
* Сорс код на Python tool (tools/animator.py), който може да прочете генерираното решение на проблема и да създаде анимация, с която да го визуализира. За да се стартира скрипта трябва да се инсталират matplotlib и pandas. Скрипа приема команден аргумент име на файла, от който да прочете данните за анимиране.

1. **Проектиране, реализация и тестване**

4.1 **Директен алгоритъм**

1. За всяко тяло изчисляваме гравитационната сила, с коят му влияят останалите тела по следната формула

A black text on a white background

Description automatically generated

* G – гравитационна константа
* mi​ и mj​ са масите на тела i и j
* ri​ и rj​ са позициите на тела i и j
* ri​j е векторът на посоката между i и j

1. За всяко тяло изчиляваме скоростта и посоката му във следващият момент на база гравитационните сили, които му действат по следната формула:

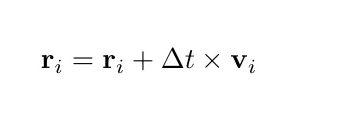
A mathematical equation with numbers and symbols

Description automatically generated

1. Всяко тяло се премества на база ускорението и посоката изчислена в предишните стъпки. Ако текущото време е по - малко от крайното се връщаме обратно на стъпка 1. Ако не, алгоритъмът приключва.

A math symbols and symbols

Description automatically generated with medium confidence

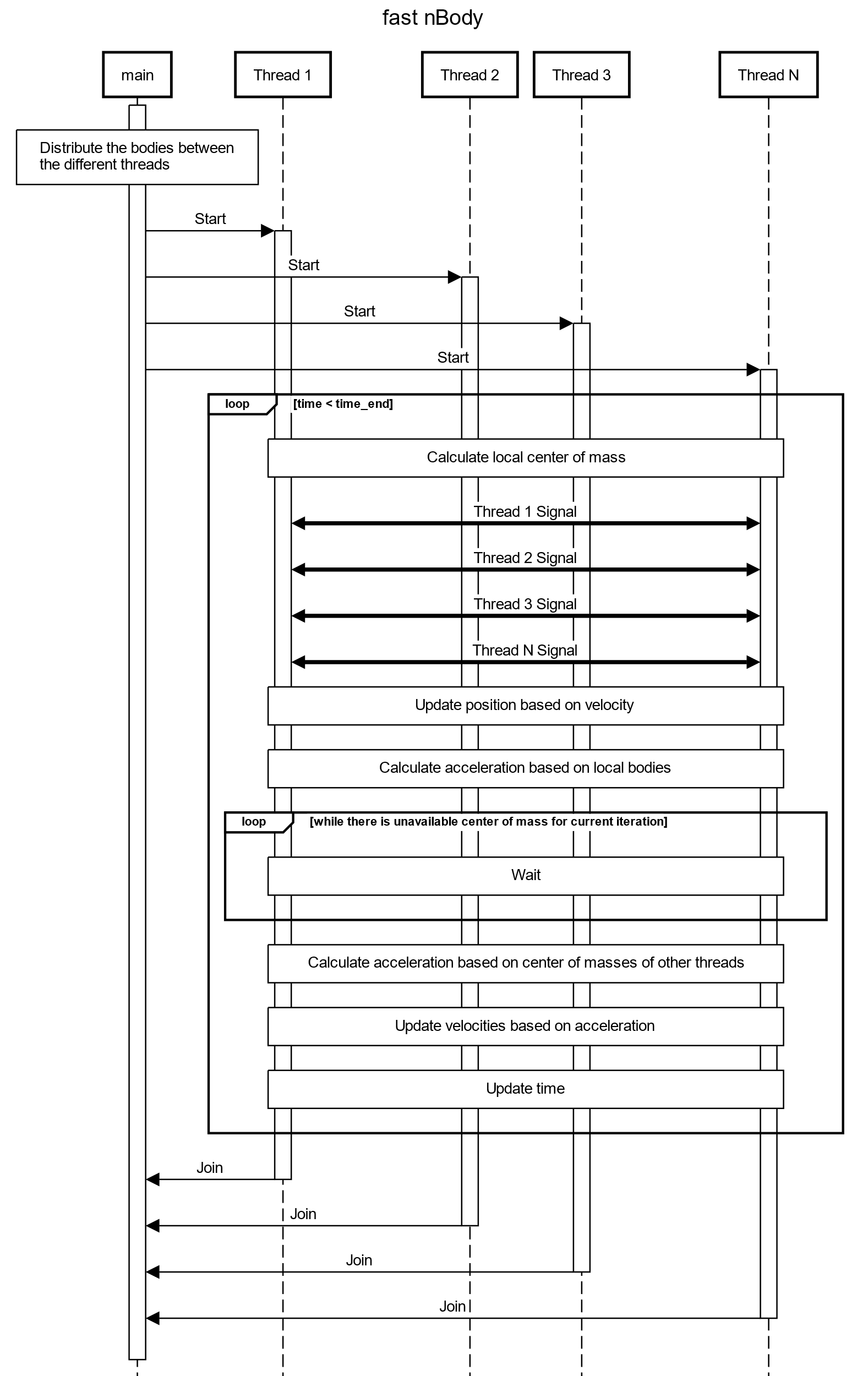


* 1. **fast nBody**
     1. **Реализация**

Предложената в този документ имплементация се основава на Barnes-Hut алгоритъма. Разликата е, че игнорираме строенето на quadtree и следене и обновяване на телата, близки едно до друго. Хипотезата, която изследваме е, че телата, които в началото са близки едно до друго ще се движат заедно и ще останат близки едно друго спрямо останалите тела, които са били далече в началното състояние. Ще се фокусираме върху това да изследваме за кои начални конфигурации този алгоритъм дава достатъчно малка грешка, за да бъде приложим. Алгоритъмът е следният:

1. Разделяме равномерно телата в групи. Броят на групите е равен на броят на нишките, на които ще се изпълнява програмата. Групите се създават, така че в една група да са телата, които са близко едно до друго в пространството. През цялото изпълнение на програмата, всяка нишка ще прави изчисления само върху разпределените и в тази стъпка тела.
2. Всяка нишка изчислява центърът на тежестта на телата в своята група.
3. Всяка нишка изчислява директният алгоритъм за своето подмножество от тела. Изчислява гравитационната сила и векторът на скоростта за всяко тяло, оценявайки въздействието на останалите тела в своята група.
4. За всяко тяло изчислява директният алгоритъм за него и центровете на тежестта на другите групи. Изчислява гравитационната сила и векторът на скоростта за всяко тяло, оценявайки въздействието на другите групи върху телата от своята група.
5. Всички нишки се изчакват да приключат текущата итерация. Ако тази стъпка не е последната всички нишки преминават на следващата итерация и се връщаме на стъпка 2 от алгоритъма. Нишките трябва да се изчакват, защото когато вземат предвид влиянието на другите групи върху своята, трябва да се вземе предвид положението на всяко тяло от другата група във същият момент от време. Все пак описаното тук е опростено за леснота на разбирането. Реално нишките се приспиват, когато поискат да прочетат центъра на тежестта, който още не е изчислен, което става в стъпка 4. Когато центърът на тежестта се изчисли, нишката, която е поискала да го прочете се събужда. Тъй като смятането на центровете става в стъпка 2, а стъпка 3 отнема най дълго време, очакването е че нишките рядко ще се налага да се приспиват при достатъчно голям брой тела.

Диаграма 1. Последователна диаграма на fast nBody алгоритъма.



* + 1. **Анализ**
* **Цена на комуникацията:** Тъй като комуникацията ни е нужда единствено, когато трябва да изчислим влиянието на центъра на тежестта на другите групи върху локалните ни тела, цената е относително ниска. На практика синхронизацията не би трябвало да оказва никакъв ефект върху скоростта на алгоритъм, тъй като ако някоя нишка бъде приспана, защото чака друга да сметне своят център на тежестта, то тогава изчакваната нишка има да извърши още доста изчисления след като изчакващата нишка бъде събудена.
* **Баланс на товара**: Изчисленията на всяка стъпка са едни и същи и имат равен на брой операции (няма условности, които да променят потокът на изпълнение на програмата). Нишките получават равен брой тела, което води до изключително равномерна балансираност на товара.
* **Грануларност:** Едра, поради добрият баланс на товара.
* **Сложност по време:** При този алгоритъм броят на нишките има силно значение за сложността по време. При директният алгоритъм, изпълнен на 1 нишка, сложността му е O(), тъй като за всяко тяло трябва да бъде изчислена гравитационната сила, която му действа на база на всички останали тела. Ако програмата се изпълнява на р нишки, за всяко тяло трябва да бъде изчислена гравитационната сила на база на телата в локалната група, които са на брой, плюс броят на центровете на масите на останалите групи, които са p на брой. Всяка нишка ще изпълни О( + ) = О () операции (тъй като p << n). Всички нишки в такъв случай ще изпълнят операции. Интересното е, че с броят на нишките, намаляват броят на операциите, което заедно с факта, че нишките се изпълняват паралелно ни дава очакване за квадратично ускорение. Разбира се това ускорение е измамно, тъй като с увеличение на броят на нишките ние намаляме изчисленията.
  + 1. **Тестване**

Машината на която беше проведено тестването на алгоритъма има следните характеристики:

* 2 x Intel Xeon @ 2.20 GHz
* 2 x 8 ярда
* L1 cache: 32KB
* Архитектура x86\_64

Тестването беше проведено при 15000 тела във времеви интервал от 0 до 40, при времева стъпка 1.

* + - 1. **Тестване с измамни резултати, както описани в 4.2.2 – Сложност по време**

Таблица 1. Измерване на времето за изпълнение, ускорението и ефективността (измамно)

A table of numbers with numbers

Description automatically generated

Фигура 1. Време в секунди за изпълнение спрямо броят ядра (измамно)

A graph with orange lines

Description automatically generated

Фигура 2. Ускорение спрямо броят ядра (измамно)

A graph with orange lines

Description automatically generated

* + - 1. **Тестване без измамни резултати, както описани в 4.2.2 – Сложност по време**

Таблица 2. Измерване на времето за изпълнение, ускорението и ефективността

A table of numbers with green text

Description automatically generated

Фигура 3. Време в секунди за изпълнение спрямо броят ядра

A graph with orange lines

Description automatically generated

Фигура 4. Ускорение спрямо броят ядра

A graph with orange lines

Description automatically generated

* + - 1. **Тестване на отклонението спрямо реалните позиции на телата**

Отклонение при 2000 тела за време от 0 до 100, 0.1 времева стъпка. Отклоненията са изчислени спрямо алгоритъма, изпълнен на 4 нишки. Грешката нараства квадратично спрямо броят на телата. Координатите основно са между -1000 и 1000

* При случайно разпределение на телата :

Average difference in X: 274.48674189710005

Average difference in Y: 263.35324659015004

Average difference in Z: 307.8635179212501

* При равномерно разпределение на телата в отделни групи. Тук всички тела в групите са близко едно до друго и са далече от другите групи:

Average difference in X, Y, Z: ~30

1. **Заключение**

Хипотезата за това, че телата близки едно до друго в началното състояние остават близки се оказа невярна за произволно разпределение на телата. Хипотезата е вярна за специфичният случай, когато телата са фокусирани в групи, коита са далече една от друга.

При този алгоритъм, паралелизма трябва да бъде съобразен с началните условия на задачата. Броят на описаните групи ни дава паралелизма, който трябва да бъде използван, за оптимално по време и точност решение на задачата.

Алгоритъма е доста по бърз от установените за решаване на този проблем. За сметка на това прецизността му в общият слуачай е изключително ниска.

В случаят, когато телата са фокусирани в групи, които са далече една от друга, прецизността е по – висока и скоростта на алгоритъма надминава конкуретните алгоритми.

Във архива на този документ могат да бъдат намерени видеа с анимации на разумен брой за визуализация тела. Анимациите, които са изчислени на 1 нишка са абсолютно прецизни. С увеличаване на броят на нишките прецизността на модела намалява:

* 3Body.mov – симулация на 3 тела на 1 нишка.
* 100Body\_1thread.mov – симулация на 100 тела на 1 нишка.
* 100Body\_2thread.mov – симулация на 100 тела на 2 нишки.
* 100Body\_4thread.mov – симулация на 100 тела на 4 нишки.

**6. Източници**

[1] Barnes, J., Hut, P. A hierarchical O(N log N) force-calculation algorithm. Nature 324, 446–449 (1986). https://doi.org/10.1038/324446a0

[2] TOM VENTIMIGLIA & KEVIN WAYNE - The Barnes-Hut Algorithm - http://arborjs.org/docs/barnes-hut

[3] By P Koumouts - Fast multipole methods for threedimensional Nbody prob - <https://web.stanford.edu/group/ctr/ResBriefs95/pemp>

[4] Source code of this document’s algorithm - https://github.com/dimitarkyurtov/nBody