DOI: 10.15514/ISPRAS-2023-35(6)-14

Эффективная реализация быстрого метода мультиполей для взаимодействия частиц с ньютоновским потенциалом

В.М. Аушев, ORCID: 0009-0009-3571-5448 <*aushevvm@gmail.com*>

Московский государственный технический университет имени Н.Э. Баумана, 105005, Россия, г. Москва, ул. 2-я Бауманская, д. 5, к. 1

Аннотация. В работе рассматривается быстрый метод мультиполей с использованием матриц поворота для операторов трансляции для расчета взаимодействия частиц с нютоновским потенциалом, а также его приложение для случая, когда взаимодействие между частицами описывается законом Био—Савара. В работе приведены формулы, необходимые для реализации алгоритма, а также такие редко затрагиваемые моменты, как нормировка сферических гармоник и связанная с ней нормировка матриц Вигнера. Основное внимание в работе уделено описанию деталей программной реализации, которые позволяют существенно ускорить работу кода, как для CPU, так и для GPU (с использованием технологии CUDA). Приведено подробное изложение предлагаемых методик, а также иллюстрирующие их листинги кода. С их использованием был реализован программный комплекс на C++ и проведено сравнение с открытыми программными реализациями быстрого метода мультиполей. Данное сравнение подтверждает высокую эффективность предложенной реализации.

Ключевые слова: быстрый метод мультиполей; матрицы Вигнера; мортоновское дерево; SIMD; CUDA; открытые библиотеки

Для цитирования: Аушев В.М. Эффективная реализация быстрого метода мультиполей для взаимодействия частиц с ньютоновским потенциалом. Труды ИСП РАН, том 35, вып. 6, 2023 г., стр. 213–234. DOI: 10.15514/ISPRAS–2023–35(6)–14

Effective Implementation of the Fast Multipole Method for Particle Interaction with Newtonian Potential

V.M. Aushev, ORCID: 0009-0009-3571-5448 <*aushevvm@gmail.com*>

Bauman Moscow State Technical University,
105005, Moscow, 2-nd Baumanskaya st., 5.

Abstract. We consider rotation-based fast multipole method for the Laplace equation and its application in cases where particle interactions are governed by the Biot—Savart law. The paper presents the necessary formulas for algorithm implementation and addresses less frequently discussed topics, such as the normalization of spherical harmonics and the associated Wigner matrices normalization. The main focus of the paper is devoted to describing the details of the software implementation that significantly accelerate the performance of the code both on CPU and GPU (using CUDA technology). We provide a comprehensive explanation of proposed techniques and include code examples. We have implemented a C++ program using these methods and conducted a comparative analysis with open-source implementations of the fast multipole method, confirming the high efficiency of our approach.

Keywords: fast multipole method; Wigner matrices; Morton tree; SIMD; CUDA; open source libraries

For citation: Aushev V.M. Effective implementation of the fast multipole method for particle interaction with Newtonian potential. *Trudy ISP RAN/Proc. ISP RAS*, vol. 35, issue 6, 2023. pp. 213-234 (in Russian). DOI: 10.15514/ISPRAS-2023-35(6)-14

1. Введение

Данная работа посвящена практической реализации быстрого метода мультиполей (БММ) для N тел, взаимодействующих по закону Ньютона в трехмерном случае (сила обратно пропорциональна квадрату расстояния), а также между телами, сила взаимодействия для которых описывается законом Био—Савара. Быстрый метод мультиполей позволяет решать описанную задачу N тел с линейной сложностью, в то время как прямой метод решения имеет сложность $O(N^2)$ — таким образом, прямым методом крайне затруднительно решать задачи с большим числом тел. Быстрый метод мультиполей является развитием другого метода, который нельзя не упомянуть в контексте быстрых методов. Речь идет о методе Барнса—Хата [1], который имеет сложность $O(N \log N)$.

Задача N тел возникает во многих областях математики и физики; самый простой пример — расчет траектории небесных тел под действием гравитации. В математике задача N тел возникает как следствие дискретизации интегрального уравнения для решения задач физики, например, в электродинамике [2-3], в гидродинамике и теории упругости [4-5]. Теоретическому описанию быстрого метода мультиполей посвящено множество работ, например, [6-8]. Отдельно выделим работу [9], в которой авторы не только подробно описывают теорию, но и объясняют, за счет чего достигается быстродействие алгоритма на простых примерах.

Изначальная формулировка быстрого метода мультиполей для трехмерных задач, хоть и имеет линейную асимптотику, имеет сложность $O(p^4)$, где p — количество мультиполей. В связи с этим практическое использование метода для реального количества тел оказывается медленнее, чем уже упомянутый метод Барнса—Хата. Для исправления этой ситуации были созданы модификации метода, использующие вращения (сложность $O(p^3)$) и диагональные формы операторов трансляции (сложность $O(p^2)$) [7]. Существуют также и другие подходы, позволяющие понизить сложность алгоритма, основанные на сочетании с методом частиц [10-11]. Использование идей метода частиц позволяет также ускорить метод за счет использования быстрого преобразования Фурье для операторов трансляции [12]. Также

возможна реализация быстрого метода мультиполей “без мультиполей”, как описано в статье [13].

Отдельно упомянем существующие открытые программные реализации быстрого метода мультиполей [14-18]. В данной работе мы проведем сравнение с двумя реализациями [14-18].

Дальнейшее изложение устроено следующим образом. Во втором разделе мы приведем краткое теоретические описание используемых формул, а также подробно рассмотрим нормировку сферических гармоник и следующую из нее нормировку матриц Вигнера; кроме этого, мы представим выражения для вычисления сил с использованием мультипольных разложений. Во втором разделе мы перейдем к подробному описанию деталей программной реализации, включая рекуррентные формулы для присоединенных полиномов Лежандра и матриц Вигнера, а также различные техники, позволяющие в разы ускорить код на C++ в случае CPU версии кода и CUDA C++ для GPU версии. В третьем разделе приведены таблицы, иллюстрирующие скорость работы кода для разных версий, включая распараллеливание для систем с общей и распределенной памятью. В четвертом разделе приведено сравнение реализованного программного комплекса[1] с уже упомянутыми выше реализациями.

2. Теоретические основы быстрого метода мультиполей

2.1 Общая схема работы быстрого метода мультиполей

Пусть имеется N частиц с координатами $\mathbf{x} \in \mathrm{R}^n$, и для каждой частицы необходимо вычислить выражение вида

$$u(\mathbf{x}_i) = \sum_{i \neq j} f(\mathbf{x}_i, \mathbf{y}_j) = g(\mathbf{x}_i - \mathbf{y}_j) = g(\mathbf{r}_{ij}).$$

Задачи такого вида возникают во многих областях науки, например, при расчете движения тел под действием гравитационных сил, в этом случае $g(\mathbf{r}_{ij}) = Gm_i m_j \dfrac{\mathbf{x}_j - \mathbf{y}_i}{\mathrm{P}\mathbf{x}_i - \mathbf{y}_j \mathrm{P}}$. Очевидно, прямой метод подсчета сил для всех тех имеет сложность $O(N^2)$, поэтому представляют интерес методы, решающие данную задачу с меньшей сложностью; такие методы называются быстрыми. Основная идея быстрых методов состоит в том, что функцию $g(\mathbf{x}-\mathbf{y})$ при достаточно удаленных точках $\mathbf{x}$ и $\mathbf{y}$ можно приближенно заменить функцией $h(\mathbf{x},\mathbf{y})$, которая факторизуется, т.е.

$$h(\mathbf{x},\mathbf{y}) = \sum_{i=0}^{p} a_i(\mathbf{x}) b_i(\mathbf{y}).$$

Смысл этого приближения виден на простом примере, когда

$$g(\mathbf{x},\mathbf{y}) = A(\mathbf{x})B(\mathbf{y}),$$

в этом случае мы можем решить задачу N тел с линейной сложностью: на первом шаге находим величину $Sy = \sum_{i=1}^{N} B(\mathbf{y}_i)$, на втором шаге находим влияние всех тел на частицу как $u(\mathbf{x}_i) = A(\mathbf{x}_i)Sy, \ \ i = \overline{1, N}.$

Одним из представителей быстрых методов является быстрый метод мультиполей, имеющий сложность $O(N)$. Описание работы алгоритма приведено во многих статьях, например, в [9]

[1] https://github.com/ViktorAushev/laplace-fmm/

представлено очень подробное изложение идеи метода, а также приведены конкретные формулы. В этой связи мы ограничимся кратким описанием алгоритма.

1) На первом шаге для всех частиц строится квад- или октодерево в зависимости от размерности задачи (рис. 1, а-в). Данная процедура очень важна для эффективной численной реализации алгоритма, поэтому в дальнейшем мы подробно рассмотрим этот момент. Под деревом на примере двумерной области мы понимаем следующую структуру: на самом верхнем уровня дерева имеем корень — квадрат, содержащий все частицы, уровнем ниже имеем четыре квадратных ячейки, получающиеся дроблением исходного квадрата, то есть родительской ячейки; для корня дерева полученные четыре ячейки — дочерние (потомки). Рекурсивно разбивая ячейки, получаем структуру дерева, на самом нижнем уровне имеем ячейки без детей — то есть листья. Под обходом дерева “вверх” будем понимать обход уровней дерева от листьев к корню, под движением “вниз” — путь от корня к листьям. Отметим, что в общем случае дерево можно строить так, что листья находятся на разных уровнях, но мы этот случай не будем рассматривать. Также считаем, что в дереве нет пустых ячеек, то есть которые не содержат частиц — таким образом имеем адаптивную структуру дерева (при описании реализации дерева в дальнейшем повествовании приведен пример такого дерева).
2) Нахождение мультипольных разложений в листьях дерева (назовем этот шаг P2M, рис. 1, г). Данные разложения представляют собой аппроксимацию функции $g(\mathbf{x},\mathbf{y})$ вдали от ячейки, то есть находим коэффициенты $b_i(\mathbf{y})$ для функции $h(\mathbf{x},\mathbf{y})$.
3) Поднимаясь вверх по уровням дерева, для каждой ячейки находим ее мультипольные разложения, используя разложения дочерних ячеек. Для этого необходимо выполнять трансляцию мультипольных разложений из дочерних ячеек в родительские (операция M2M, рис. 1, г).
4) Спускаемся от корня дерева вниз, для каждой ячейки находим аппроксимацию влияния дальних соседей на частицы внутри ячейки, которое представляется в виде локального разложения. Для нахождения локальных разложений необходимо выполнять трансляцию мультипольных разложений в локальные (операция M2L, рис. 1, д).
5) Спускаемся от корня дерева вниз, для каждой ячейки находим локальное разложение, связанное с влиянием дальних соседей на родительскую ячейку — для этого выполняется трансляция локального разложения родительской ячейки в дочернюю (операция L2L, рис. 1, е).
6) Для всех частиц в листьях находим потенциал (силу): влияние от дальних ячеек учитываем, используя локальные разложения (операция L2P, рис. 1, е), а взаимодействие между частицами из ближайших соседей листа находим по прямой формуле (т.е. используя функцию $g(\mathbf{x},\mathbf{y})$ — операция P2P, рис. 1, ж).

В алгоритме мы упомянули ближних и дальних соседей. Ближние соседи ячейки — ячейки того же уровня дерева, соприкасающиеся с ячейкой. Для нахождения дальних соседей мы берем родительские ячейки ближних соседей, рассматриваем все их дочерние ячейки; среди них дальними соседями являются все, которые не являются ближними.

Таким образом, для реализации алгоритма быстрого метода мультиполей нам необходимо выполнять операции P2M, M2M, M2L, L2L и L2P. Далее приведено краткое теоретическое описание данных операций применительно к ньютоновскому потенциалу и силе.

Все этапы работы алгоритма приведены на единой схеме на рис. 1.

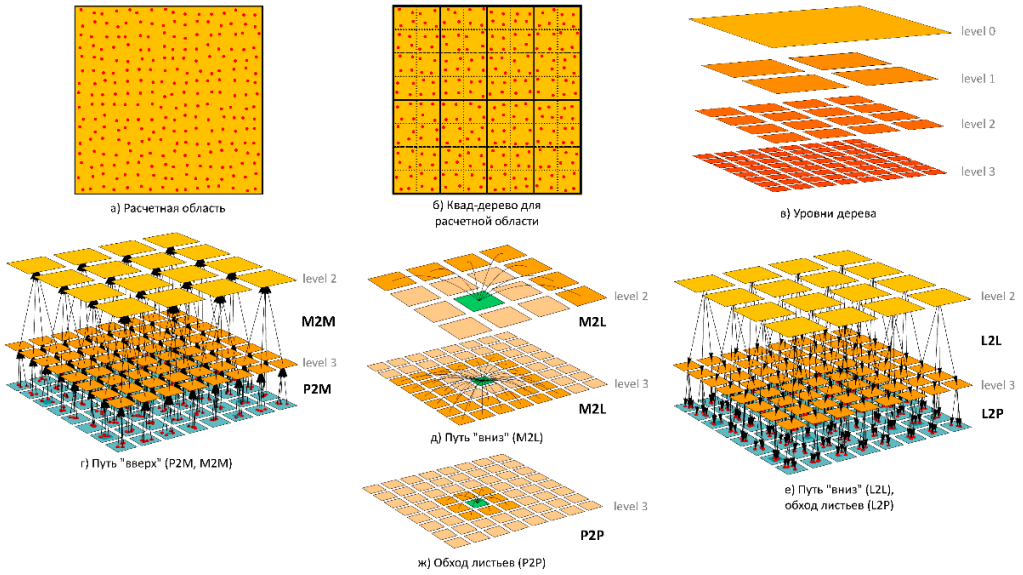


Рис. 1. Этапы алгоритма быстрого метода мультиполе
Fig. 1. Stages of the fast multipole method

2.2 Теория для ньютоновского потенциала и силы

Будем считать, что потенциал заряда q, расположенного в точке $\mathbf{y}$, имеет вид

$$\Phi(\mathbf{x},\mathbf{y}) = \frac{q}{\mathrm{P}\mathbf{x}-\mathbf{y}\mathrm{P}}, \qquad \mathbf{x},\mathbf{y} \in \mathrm{R}^3,$$

а напряженность поля —

$$\mathbf{E}(\mathbf{x},\mathbf{y}) = q\frac{\mathbf{x}-\mathbf{y}}{\mathrm{P}\mathbf{x}-\mathbf{y}\mathrm{P}^3}, \qquad \mathbf{x},\mathbf{y} \in \mathrm{R}^3.$$

Изложение теории начнем с введения сферических функций Бесселя [7]:

$$Y_n^m(\theta,\phi) = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} \cdot P_n^{|m|}(\cos\theta)e^{im\phi}, \tag{1}$$

где $P_n^m(x)$ — присоединенный полином Лежандра, определяемый формулой Родрига:

$$P_n^m(x) = (-1)^m(1-x^2)^{m/2}\frac{d^m}{dx^m}P_n(x),$$

а $P_n(x)$ — полином Лежандра степени n. Отметим тот момент, что введенные в (1) и используемые в быстром методе мультиполей для трехмерного уравнения Лапласа [7-8] сферические гармоники отличаются от тех, которые получаются в физике при выводе собственных функций оператора момента импульса в квантовой механике [19-20]:

$$\tilde{Y}_n^m(\theta,\phi) = (-1)^m\sqrt{\frac{2n+1}{4\pi}}\sqrt{\frac{(n-m)!}{(n+m)!}} \cdot \tilde{P}_n^m(\cos\theta)e^{im\phi}, \tag{2}$$

где

$$\tilde{P}_n^m(x) = (1-x^2)^{m/2}\frac{d^m}{dx^m}P_n(x),$$

то есть для присоединенных полиномов Лежандра отличие в множителе $(-1)^m$. Таким образом,

$$\tilde{Y}_n^m(\theta,\phi) = \sqrt{\frac{2n+1}{4\pi}}\sqrt{\frac{(n-m)!}{(n+m)!}}\cdot P_n^m(\cos\theta)e^{im\phi}.$$

В итоге, имеем следующее соответствие между сферическими функциями с разными нормировками:

$$\varepsilon_m\sqrt{\frac{4\pi}{2n+1}}Y_n^m(\theta,\phi) = \tilde{Y}_n^m(\theta,\phi), \tag{3}$$

где

$$\varepsilon_m = (-1)^{(|m|-m)/2} = \begin{cases}(-1)^m, & m<0,\\ 1, & m,, 0.\end{cases}$$

Соотношение (3) пригодится в дальнейшем изложении при определении матриц Вигнера. Подробно про различные варианты нормировок сферических гармоник и их обоснование написано в [19].

Перейдем к изложению основных формул. Подробную формулировку и доказательство используемых теорем можно найти в [7-8].

Теорема 1. (Мультипольное разложение, P2M)

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{M_n^m}{r^{n+1}}\cdot Y_n^m(\theta,\phi),$$

где

$$M_n^m = \sum_{i=1}^{N} q_i\cdot\rho_i^n\cdot Y_n^{-m}(\alpha_i,\beta_i).$$

Теорема 2. (Локальное разложение, L2P)

$$\Phi(\mathbf{x}) = \sum_{j=0}^{\infty}\sum_{k=-j}^{j}L_j^k\cdot Y_j^k(\theta,\phi)\cdot r^j, \tag{4}$$

где

$$L_j^k = \sum_{i=1}^{N} q_i\cdot\frac{Y_j^{-k}(\alpha_i,\beta_i)}{\rho_i^{j+1}}.$$

Если требуется найти не потенциал, а силу, поступим следующим образом. Найдем градиент разложения (4):

$$-\mathbf{E}(\mathbf{x}) = \nabla_{(r,\theta,\phi)}\Phi(\mathbf{x}) = \begin{pmatrix}\sum_{n=1}^{\infty}\sum_{m=-n}^{n}L_n^m\cdot Y_n^m(\theta,\phi)\cdot n\cdot r^{n-1}\\ \sum_{n=0}^{\infty}\sum_{m=-n}^{n}L_n^m\cdot\frac{\partial Y_n^m(\theta,\phi)}{\partial\theta}\cdot r^n\\ \sum_{n=0}^{\infty}\sum_{m=-n}^{n}L_n^m\cdot\frac{\partial Y_n^m(\theta,\phi)}{\partial\phi}\cdot r^n\end{pmatrix}.$$

Получили выражение в сферических координатах. Для перехода в декартовы координаты нам понадобится обратная матрица Якоби:

$$(J^{-1})^T = \begin{pmatrix} \sin\theta\cos\phi & \dfrac{\cos\theta\cos\phi}{r} & -\dfrac{\csc\theta\sin\phi}{r} \\ \sin\theta\sin\phi & \dfrac{\cos\theta\sin\phi}{r} & \dfrac{\csc\theta\cos\phi}{r} \\ \cos\theta & -\dfrac{\sin\theta}{r} & 0 \end{pmatrix}.$$

В итоге получаем

$$\nabla_{(x,y,z)}\Phi(\mathbf{x}) = (J^{-1})^T \nabla_{(r,\theta,\phi)}\Phi(\mathbf{x}) = \begin{pmatrix} A\sin\theta\cos\phi + B\cos\theta\cos\phi - C\csc\theta\sin\phi \\ A\sin\theta\sin\phi + B\cos\theta\sin\phi + C\csc\theta\cos\phi \\ A\cos\theta - B\sin\theta \end{pmatrix},$$

где

$$\begin{pmatrix} A \\ B \\ C \end{pmatrix} = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} L_n^m r^{n-1} \begin{pmatrix} n\cdot Y_n^m(\theta,\phi) \\ \dfrac{\partial Y_n^m(\theta,\phi)}{\partial\theta} \\ \dfrac{\partial Y_n^m(\theta,\phi)}{\partial\phi} \end{pmatrix}. \qquad (5)$$

Распишем производные [21]:

$$\frac{\partial Y_n^m(\theta,\phi)}{\partial\phi} = i\cdot m\cdot Y_n^m(\theta,\phi),$$

$$\frac{\partial Y_n^m(\theta,\phi)}{\partial\theta} = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}}\cdot\frac{(n+1-|m|)P_{n+1}^{|m|}(\theta) - (n+1)\cos(\theta)P_n^{|m|}(\theta)}{\sin(\theta)}\cdot e^{im\phi}.$$

Заметим, что при $n = m = 0$ производные обращаются в ноль, в итоге (5) примет вид

$$\begin{pmatrix} A \\ B \\ C \end{pmatrix} = \sum_{n=1}^{\infty}\sum_{m=-n}^{n} L_n^m r^{n-1} \begin{pmatrix} n\cdot Y_n^m(\theta,\phi) \\ \sqrt{\dfrac{(n-|m|)!}{(n+|m|)!}}\cdot\dfrac{(n+1-|m|)P_{n+1}^{|m|}(\theta) - (n+1)\cos(\theta)P_n^{|m|}(\theta)}{\sin(\theta)}\cdot e^{im\phi} \\ i\cdot m\cdot Y_n^m(\theta,\phi) \end{pmatrix}.$$

Замечание: при $\theta = 0$ имеем неопределенность в выражении для $\dfrac{\partial Y_n^m(\theta,\phi)}{\partial\theta}$; в этом случае выражение следует понимать в смысле предела при $\theta \to 0$. Однако при численной реализации из-за погрешности вычислений и случайного расположения частиц θ не равняется нулю, и все вычисления проходят корректно.

Теорема 3. (Трансляция мультипольного разложения, M2M)

$$M_j^k = \sum_{n=0}^{j}\sum_{m=-n}^{n} \frac{O_{j-n}^{k-m}\cdot i^{|k|-|m|-|k-m|}\cdot A_n^m\cdot A_{j-n}^{k-m}\cdot\rho^n\cdot Y_n^{-m}(\alpha,\beta)}{A_j^k}.$$

Теорема 4. (Трансляция мультипольного разложения в локальное, M2L)

$$L_j^k = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \frac{O_n^m i^{|k-m|-|k|-|m|}\cdot A_n^m\cdot A_j^k\cdot Y_{j+n}^{m-k}(\alpha,\beta)}{(-1)^n A_{j+n}^{m-k}\cdot\rho^{j+n+1}}.$$

Теорема 5. (Трансляция локального разложения, L2L)

$$L_j^k = \sum_{n=j}^{p}\sum_{m=-n}^{n} \frac{O_n^m \cdot i^{|m|-|m-k|-|k|} \cdot A_{n-j}^{m-k} \cdot A_j^k \cdot Y_{n-j}^{m-k}(\alpha,\beta)\cdot \rho^{n-j}}{(-1)^{n+j}\cdot A_n^m}.$$

Коэффициенты A_n^m имеют вид

$$A_n^m = \frac{(-1)^n}{\sqrt{(n-m)!\cdot(n+m)!}}. \tag{6}$$

Заметим, что выполнение операций M2M, M2L и L2L имеет сложность $O(p^4)$. Улучшить ситуацию можно двумя способами: используя вращения или диагональные формы операторов [7-8]. В первом случае сложность составляет $O(p^3)$, во втором — $O(p^2)$. В данной работе мы рассмотрим первый вариант. Для начала нам понадобятся две леммы [22].

Лемма 1. *Пусть гармоническая функция* $\Phi(\mathbf{x})$ *имеет вид*

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\left(L_n^m r^n + \frac{M_n^m}{r^{n+1}}\right)Y_n^m(\theta,\phi),$$

где (r,θ,ϕ) *— сферические координаты точки* $\mathbf{x}$. *При повороте системы координат на угол* β *вокруг оси* z *в положительном направлении функция* $\Phi(\mathbf{x})$ *примет вид*

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\left(\tilde{L}_n^m r^n + \frac{\tilde{M}_n^m}{r^{n+1}}\right)Y_n^m(\theta,\phi'),$$

где (r,θ,ϕ') *— новые координаты точки* $\mathbf{x}$, *при этом*

$$\tilde{L}_n^m = L_n^m e^{im\beta}, \quad \tilde{M}_n^m = M_n^m e^{im\beta}.$$

Лемма 2. *Пусть гармоническая функция* $\Phi(\mathbf{x})$ *имеет вид*

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\left(L_n^m r^n + \frac{M_n^m}{r^{n+1}}\right)Y_n^m(\theta,\phi),$$

где (r,θ,ϕ) *— сферические координаты точки* $\mathbf{x}$. *При повороте системы координат на угол* α *вокруг оси* y *в положительном направлении функция* $\Phi(\mathbf{x})$ *примет вид*

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\left(\tilde{L}_n^m r^n + \frac{\tilde{M}_n^m}{r^{n+1}}\right)Y_n^m(\theta',\phi),$$

где (r,θ',ϕ) *— новые координаты точки* $\mathbf{x}$, *при этом*

$$\tilde{L}_n^m = \sum_{k=-n}^{n}\tilde{d}_{mk}^n(\alpha)L_n^k, \quad \tilde{M}_n^m = \sum_{k=-n}^{n}\tilde{d}_{mk}^n(\alpha)M_n^k.$$

Здесь $\tilde{d}_{mk}^n$ — компоненты матрицы Вигнера [23, 19, 20]. Матрицы Вигнера используются для пересчета сферических гармоник при повороте системы координат [23, 19]:

$$\tilde{Y}_n^m(\theta',\phi') = \sum_{k=-n}^{n}\tilde{D}_{km}^n(\alpha,\beta,\gamma)\tilde{Y}_n^k(\theta,\phi), \tag{7}$$

где (α,β,γ) — эйлеровы углы, $\tilde{Y}_n^m$ — сферические гармоники, определенные в (2), $\tilde{D}_{km}^n$ — матрица Вигнера, определенная как [23]

$$\tilde{D}_{mk}^n(\alpha,\beta,\gamma) = e^{im\alpha}\tilde{d}_{mk}^n(\beta)e^{ik\gamma},$$

$$\tilde{d}^n_{mk}(\beta) = \sqrt{(n+k)!(n-k)!(n+m)!(n-m)!} \times \sum_{s=\max(0,k-m)}^{\min(n-m,n+k)} \frac{(-1)^s \left(\sin\left(\frac{\theta}{2}\right)\right)^{2s-k+m} \left(\cos\left(\frac{\theta}{2}\right)\right)^{k-m+2(n-s)}}{s!(s+m-k)!(k+n-s)!(n-m-s)!}.$$

Подставим в (7) связь сферических гармоник с разной нормировкой (3), получим

$$\varepsilon_m Y^m_n(\theta',\phi') = \sum_{k=-n}^{n} \tilde{D}^n_{km}(\alpha,\beta,\gamma) Y^k_n(\theta,\phi)\varepsilon_k.$$

Так как $1/\varepsilon_m = \varepsilon_m$, получаем

$$Y^m_n(\theta',\phi') = \sum_{k=-n}^{n} D^n_{km}(\alpha,\beta,\gamma) Y^k_n(\theta,\phi),$$

где

$$D^n_{mk}(\alpha,\beta,\gamma) = \varepsilon_m \varepsilon_k \tilde{D}^n_{mk}(\alpha,\beta,\gamma). \tag{8}$$

Соотношения (3) и (8) важно учитывать при написании кода, например, в Wolfram Mathematica, где под сферическими функциями Бесселя и матрицами Вигнера понимаются как раз $\tilde{Y}^m_n$ и $\tilde{D}^n_{mk}$.

Обозначим за T оператор трансляции (M2M, M2L или L2L), $R_y(\theta)$ — оператор поворота вокруг оси y, $R_z(\phi)$ — оператор поворота вокруг оси z. Тогда оператор T записывается в виде

$$T = R_z(-\phi) R_y(-\theta) T_z R_y(\theta) R_z(\phi),$$

где T_z — оператор трансляции вдоль оси z, при этом в T_z входят сферические гармоники, вычисленные для нулевых углов, что позволяет существенно упростить запись операторов трансляции, так как $Y^m_n(0,0) = \delta_{m0}$.

Теорема 6. (Операторы трансляции вдоль оси z). *При трансляции в положительном направлении вдоль оси z верны следующие формулы пересчета коэффициентов:*

$$\text{M2M:} \qquad M^k_j = \sum_{n=0}^{j-|k|} \frac{O^k_{j-n} \cdot A^0_n \cdot A^k_{j-n} \cdot \rho^n}{A^k_j},$$

$$\text{M2L:} \qquad L^k_j = \sum_{n=k}^{\infty} \frac{O^k_n \cdot A^k_n \cdot A^k_j}{(-1)^{n+k} A^0_{j+n} \cdot \rho^{j+n+1}},$$

$$\text{L2L:} \qquad L^k_j = \sum_{n=j}^{p} \frac{O^k_n \cdot A^0_{n-j} \cdot A^k_j \cdot \rho^{n-j}}{(-1)^{n+j} \cdot A^k_n}.$$

Обратим внимание на то, как поменялись индексы суммирования. Так как для коэффициента A^m_n из условия неотрицательности выражения под знаком факториала имеем $n > m$ и $n+m > 0$, а в M2M имеется A^k_{j-n}, то получаем ограничение на индекс n: $n < j-|k|$. Аналогично в M2L, благодаря коэффициенту A^k_n имеем $n > k$, из-за чего меняется нижний индекс суммирования, а в показателе (-1) добавляется k.

Как видно из приведенных выше формул, сложность оператора поворота $R_z(\phi)$ составляет $O(p^2)$, оператора $R_y(\theta)$ — $O(p^3)$ и оператора T_z также $O(p^3)$, в итоге с помощью вращений можно выполнять трансляции T со сложностью $O(p^3)$.

2.3 Вычисление взаимодействий по закону Био—Савара

Быстрый метод мультиполей для ядра трехмерного уравнения Лапласа можно применить к частицам, взаимодействующим по закону Био—Савара. Пусть частице с координатой $\mathbf{y}$ поставлен в соответствие ток $\mathbf{j}$, и нас интересует вычисление выражения

$$\mathbf{B}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{j}\times(\mathbf{x}-\mathbf{y})}{\mathrm{P}\mathbf{x}-\mathbf{y}\,\mathrm{P}^3}. \tag{9}$$

Распишем данное выражение покомпонентно, обозначив $\mathbf{r} = \mathbf{x}-\mathbf{y}$:

$$\begin{aligned} B_x &= \frac{j_y r_z - j_z r_y}{\mathrm{P}\mathbf{r}\,\mathrm{P}^3}, \\ B_y &= \frac{j_z r_x - j_x r_z}{\mathrm{P}\mathbf{r}\,\mathrm{P}^3}, \\ B_z &= \frac{j_x r_y - j_y r_x}{\mathrm{P}\mathbf{r}\,\mathrm{P}^3}. \end{aligned} \tag{10}$$

Обозначим

$$\mathbf{F}^\alpha = j_\alpha \frac{\mathbf{r}}{\mathrm{P}\mathbf{r}\,\mathrm{P}^3}, \quad F_\beta^\alpha = j_\alpha \frac{r_\beta}{\mathrm{P}\mathbf{r}\,\mathrm{P}^3}, \qquad \alpha,\beta = x,y,z,$$

тогда (10) можно переписать следующим образом:

$$\begin{aligned} B_x &= F_z^y - F_y^z, \\ B_y &= F_x^z - F_z^x, \\ B_z &= F_y^x - F_x^y. \end{aligned} \tag{11}$$

Для вычисления $\mathbf{F}^\alpha$ можно использовать уже описанный быстрый метод мультиполей для ядра трехмерного уравнения Лапласа, после чего можно найти все компоненты вектора $\mathbf{B}$, используя формулы (11).

3. Детали программной реализации

3.1 Вычисление специальных функций

Непосредственная программная реализация описанных формул быстрого метода мультиполей сопряжена с некоторыми трудностями. Во-первых, для трехмерного уравнения Лапласа необходимо вычислять сферические гармоники и компоненты матриц Вигнера. Для первых все сводится к вычислению присоединенных полиномов Лежандра; для этого мы воспользуемся рекуррентными соотношениями [2]:

$$(n-m)P_n^m(cos\theta) = (2n-1)\cos\theta\cdot P_{n-1}^m(\cos\theta) - (n+m-1)P_{n-2}^m(\cos\theta),\ 0,,\ m,,\ n-2,$$

$$P_m^m(\cos\theta) = \frac{(2m)!}{2^m m!}(-\sin\theta)^m,\ m,,\ 0$$

$$P_{m+1}^m(\cos\theta) = (2m+1)\cos\theta\cdot P_m^m(\cos\theta),\ m,,\ 0.$$

Для вычисления компонент матриц Вигнера мы также будем использовать рекуррентные соотношения [24]:

$$\tilde{d}^n_{mn} = (-1)^{n+m} g_{nm} (1+\cos\theta)^m (\sin\theta)^{n-m}, \quad n,, 0,0,, m,, n,$$

$$\tilde{d}^n_{n,k-1} = \frac{n+k}{\sqrt{n(n+1)-k(k-1)}} \frac{\sin\theta}{1+\cos\theta} \tilde{d}^n_{nk}, \quad n>0,-n<k,, n$$

$$\tilde{d}^n_{m,k-1} = \sqrt{\frac{n(n+1)-m(m+1)}{n(n+1)-k(k-1)}} \tilde{d}^n_{m+1,k} + \tag{12}$$

$$+\frac{m+k}{\sqrt{n(n+1)-k(k-1)}} \frac{\sin\theta}{1+\cos\theta} \tilde{d}^n_{mk}, \quad n>0,0,, m<n,-n<k,, n,$$

где

$$g_{nm} = \frac{1}{2^n}\sqrt{\frac{(2n)!}{(n-m)!(n+m)!}},$$

причем коэффициенты g_{nm} можно вычислить рекурсивно:

$$g_{0,0}=1, \quad g_{n,0}=\sqrt{\frac{2n-1}{2n}} g_{n-1,0}, \quad g_{n,m}=\sqrt{\frac{n-m+1}{n+m}} g_{n,m-1}.$$

Данные формулы подходят для случая, когда аргумент $\theta \in [0;\frac{\pi}{2}]$. Для остальных случаев можно использовать следующие соотношения:

$$\tilde{d}^n_{m,k}(\pi-\theta) = (-1)^{n+m} \tilde{d}^n_{m,-k}(\theta),$$

$$\tilde{d}^n_{m,k}(-\theta) = (-1)^{m-k} \tilde{d}^n_{m,k}(\theta).$$

Для нахождения матриц Вигнера применительно к быстрому методу мультиполей необходимо использовать уже упомянутое соотношение (8).

Формулы (12) дают приемлемую погрешность при небольшом числе мультиполей [24], например, при $N=20$ максимальная относительная ошибка составляет порядка 10^{-10}; впрочем, на практике такое число мультиполей зачастую даже избыточно, так как при $N=20$ ошибка L_2 подсчета силы и потенциала составляет не больше 10^{-7}. Если же есть необходимость использования большого числа мультиполей, то целесообразно использовать псевдоспектральный метод поворота мультипольных коэффициентов [25], при использовании которого не нужно знать матрицы Вигнера и который обеспечивает достаточную точность даже для $N=1000$, однако этот метод работает примерно в 2 раза дольше, чем при использовании матриц Вигнера.

3.2 Построение дерева

Перейдем к обсуждению деталей, позволяющих ускорить работу кода. Одной из ключевых вещей в алгоритме является построение дерева, и здесь существует два подхода:

- рекурсивное построение дерева;
- построение дерева на основе фрактальной кривой.

Рекурсивное построение дерева реализуется очень просто, однако имеет несколько существенных недостатков:

1) данные хранятся в разных участках памяти, из-за этого не используется кэш при обращении к данным;
2) из-за наличия рекурсий трудно эффективно распараллелить код на CPU;
3) так как дерево строится рекурсивно, реализовать его построение на CUDA становится практически невозможно, так как в kernel-функциях невозможно

динамически выделять память. Из-за этого приходится строить дерево на CPU, затем копировать его на GPU, что влечет за собой использование сложных структур данных, снижает производительность и читаемость кода.

В этой связи наиболее целесообразным представляется построение дерева с помощью фрактальной кривой; в нашей реализации используется мортоновская кривая. Реализовать такое построение несколько сложнее, чем рекурсивное, однако у него имеются следующие достоинства:

1) данные хранятся последовательно в виде одномерных массивов, что позволяет оптимизировать обращения к памяти, так как используется кэш;
2) последовательное хранение данных позволяет эффективно распараллелить не только построение дерева, но и все операции с мультипольными разложениями;
3) код для GPU получается аналогичным коду для CPU, новые структуры данных практически не нужно использовать, в итоге улучшается читаемость кода и сам код эффективно распараллеливается;
4) построение дерева с использованием фрактальной кривой работает быстрее, чем рекурсивное построение.

Суть построения дерева на основе мортоновской кривой состоит в следующем (рассмотрим для наглядности двумерную область, в трехмерной все аналогично).

1) Пусть все частицы расположены в квадрате $[0;1)\times[0;1)$. Если это не так, все координаты можно смаштабировать, чтобы это условие выполнялось.
2) Выбирается глубина дерева, например, $\text{depth} = 3$. В этом случае в дереве будет три уровня, нулевой — корень с одной ячейкой $[0;1)\times[0;1)$, на первом уровне — максимум 4 ячейки ($[0;0.5)\times[0;0.5)$, $[0.5;1)\times[0;0.5)$, $[0;0.5)\times[0.5;1)$, $[0.5;1)\times[0.5;1)$), на втором — максимумум 16 ячеек, получающиеся дроблением на 4 ячеек второго уровня.
3) Для каждой частицы на основе ее координат (x, y) присваивается мортоновский индекс следующим образом. Пусть координаты (x, y) в двоичной записи имеют вид $0, x_1x_2x_3...$ и $0, y_1y_2y_3...$ соответственно, тогда для уровня дерева на глубине d мортоновский индекс

$$\text{idx}_d(x, y) = x_1y_1x_2y_2...x_dy_d.$$

Например, если у нас имеется только нулевой уровень дерева, то есть корень, то считаем, что индекс равен нулю. Для первого уровня возможно четыре варианта: $00_2 = 0_{10}$, $01_2 = 1_{10}$, $10_2 = 2_{10}$, $11_2 = 3_{10}$.

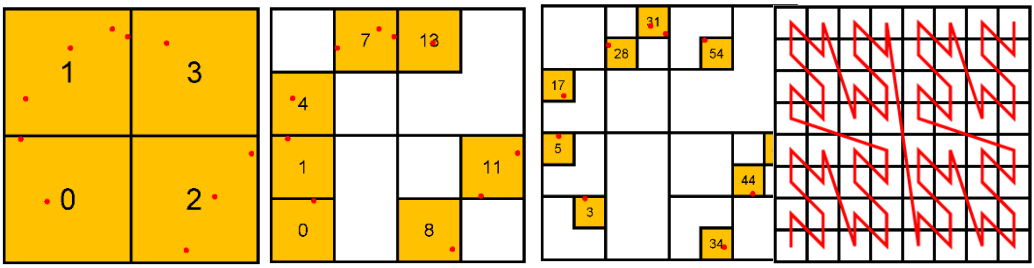
4) Находим мортоновские индексы частиц, считая, что они находятся на уровне $d = \text{depth} - 1$, то есть в листьях дерева. Координаты центров листьев при этом можно пересчитать из индексов по формуле

$$x = \frac{1}{2^{d+1}} + \sum_{i=1}^{d} \frac{1}{2^i} x_i, \qquad y = \frac{1}{2^{d+1}} + \sum_{i=1}^{d} \frac{1}{2^i} y_i.$$

5) Для определения индекса верхнего уровня достаточно выполнить битовый сдвиг индекса: $\text{idx}_{d-1} = \text{idx}_d >> 2$. При этом мы получим родительскую ячейку, а ее потомки — все ячейки нижнего уровня, у кого имеется одинаковый сдвинутый индекс. Данную процедуру повторяем для всех уровней, поднимаясь “вверх”, таким образом получая структуру дерева. Отметим, что для быстрого определения потомков ячеек целесообразно отсортировать частицы по их мортоновским

индексам, тогда частицы, принадлежащие одной ячейке, будут идти в массиве последовательно.

Пример получившихся уровней для некоторого распределения частиц для $\mathrm{depth} = 4$, а также визуализация обхода ячеек мортоновской кривой приведены на рис. 2.



а) *б)* *в)* *г)*

Рис. 2. а)—в): ячейки на первом, втором и третьем уровне дерева соответственно с подписанными мортоновскими индексами ячеек; г): обход всех ячеек дерева третьего уровня мортоновской кривой.
Fig. 2. а)—в): cells at the first, second, and third levels of the tree, respectively, with assigned Morton indices of cells; г): traversing all cells at the third level of the Morton curve.

Основное время построения дерева занимает сортировка частиц по их мортоновским кодам, поэтому важно сделать эту сортировку оптимальной. Для сортировки можно использовать различные библиотечные функции; также была реализована сортировка подсчетом. Сравнение всех вариантов представлено в табл. 1.

Табл. 1. Сортировка 50,000,000 частиц на CPU с помощью различных функций. В скобках указано ускорение относительно одного потока.
Table 1. Sorting 50,000,000 particles on a CPU using different functions, with the acceleration relative to a single thread shown in parentheses.

Число потоков	`std::sort`	`tbb::parallel_sort`	`boost::block_indirect_sort`	`counting sort`
1	7.82	5.79	5.68	4.52
2	4.32	3.18	3.04	3.05
4	2.82	2.11	1.78	1.93
8	1.58	1.26	1.25	1.08
16	1.31 (x5.97)	1.06 (x5.46)	0.78 (x7.28)	0.61 (x7.4)

В табл. 2 приведено аналогичное сравнение с доступными библиотечными функциями на GPU, в том числе с `cub::SortPairs`, которая реализует эффективную для сортировки целых чисел Radix Sort.

Табл. 2. Сортировка 50,000,000 частиц на GPU с помощью различных функций.
Table 2. Sorting 50,000,000 particles on a CPU using different functions.

`thrust::sort`	`cub::SortPairs`	`counting sort`
0.193	0.154	0.130

Исходя из приведенных таблиц, можно сделать вывод, что оптимальнее всего использовать сортировку подсчетом: она не только эффективно распараллеливается, но также легко реализуется.

3.3 Ускорение кода на CPU и GPU

При реализации формул мультипольных разложений необходимо постоянно возводить числа в целую степень, но делать это с помощью встроенной функции `pow`, которая учитывает

возможность возведения в нецелую степень, будет неэффективно. Вместо этого следует использовать “быстрое” возведение в целую неотрицательную степень, код для которого выглядит следующим образом:

```
template <typename T>
__device__ __host__ constexpr T Pow (T base, unsigned int exp) {
       T res = 1;
       while (exp) {
       if (exp & 1) {
       res *= base;
       }
       exp >>= 1;
       base *= base ;
       }
       return res ;
}
```

Листинг 1. Реализация функции возведения в целую неотрицательную степень
Listing 1. Implementation of the power function for non-negative integer exponents

В формулах для мультипольных разложений также присутствуют различные постоянные коэффициенты, как, например, биномиальные коэффициенты, которые можно просчитать заранее один раз и сохранить в массив. Аналогично можно один раз просчитать матрицы Вигнера: в силу того, что ячейки дерева всегда имеют форму квадрата, все возможные углы поворота заранее известны, поэтому можно в начале работы программы просчитать все матрицы. Все возможные углы можно найти, рассмотрев куб, состоящий из $7\times7\times7$ кубиков, и посчитав углы между центральной и всеми остальными ячейками, не включая ближних соседей.

Заметим, что сферические гармоники обладают следующим свойством:

$$Y_n^m(\theta,\phi) = \overline{Y_n^{-m}(\theta,\phi)}, \tag{13}$$

аналогично $M_n^m = \overline{M_n^{-m}}$, $L_n^m = \overline{L_n^{-m}}$, где черта означает комплексное сопряжение. Из этого следует, что можно хранить только половину коэффициентов и выполнять операции только над этой половиной, что в два раза сокращает количество вычислений.

После применения данных простых оптимизаций мы перейдем к обсуждению оптимизаций самых трудозатратных алгоритмов БММ, а именно M2L и P2P. В случае M2L можно соптимизировать вращения коэффициентов: при повороте вокруг оси z необходимо вычислять выражение вида

$$d_{m,k}^n M_n^k + d_{m,-k}^n M_n^{-k}.$$

Используя соотношение (13) для сферических функций, получаем

$$d_{m,k}^n M_n^k + d_{m,-k}^n M_n^{-k} = \mathrm{Re}\, M_n^k (d_{m,k}^n + d_{m,-k}^n) + i\,\mathrm{Im}\, M_n^k (d_{m,k}^n - d_{m,-k}^n),$$

то есть имеем два умножения вместо четырех.

Перейдем к оптимизации P2P. Заметим, что при расчете влияния двух частиц достаточно посчитать расстояние один раз, а затем можем получить потенциал или силу для обеих частиц; таким образом, мы переходим от кода

```
for (int i = 0; i < N; ++i)
       for (int j = 0; j < N; ++j) {
       // считаем расстояние между точками r(i,j)
       p[i] += q[j] / r(i,j);
       }
```

к коду

```
for (int i = 0; i < N; ++i)
       for (int j = i; j < N; ++j) {
       // считаем расстояние между точками r(i,j)
              p[i] += q[j] / r(i,j);
              p[j] += q[i] / r(i,j);
       }
```

В зависимости от типа взаимодействия (сила или потенциал), ускорение может составить от : 1.3 до : 1.9 раз.

Следующая оптимизация, позволяющая еще более существенно ускорить P2P — это векторизация, то есть использование SIMD инструкций. Для этого необходимо хранить в виде отдельных массивов все x, y и z координаты частиц, а также их заряды. В результате при обработке листьев можно с помощью одной инструкции процессора находить разность координат для 2, 4 или 8 частиц (SSE, AVX и AVX512 инструкции), а также их произведение, сумму, корень (для нахождения расстояния) и т.д. В случае использования AVX512 инструкций ускорение P2P может составлять около 3 раз.

С учетом всего вышесказанного, можно реализовать не только эффективный последовательный код, но и параллельный. Для CPU версии был реализован параллелизм для систем с общей и распределенной памятью с помощью библиотек TBB и MPI. Реализация параллельных расчетов на CPU не составляет большой сложности при готовом последовательном коде, в то время как реализация для GPU имеет ряд особенностей, которые необходимо учесть для получения максимального ускорения:

1) Использование константной памяти. Для реализации формул мультипольных разложений необходимо знать биномиальные коэффициенты, коэффициенты трансляции M2M, M2L, L2L и матрицы Вигнера. Если их передавать в качестве внешней переменной в kernel-функцию, то будет происходить обращение к глобальной памяти, которое занимает много времени. В то же время перечисленные коэффициенты (за исключением матриц Вигнера, они занимают слишком много места, а константная память ограничена 64 Кб) можно хранить в константной памяти, доступ к которой лишь немногим медленнее, чем к регистровой.
2) Использование регистровой и локальной памяти. Если массив приходит на вход kernel-функции и обращение к его элементам происходит чаще одного раза, то можно завести локальный статический массив и скопировать в него элементы внешнего массива, таким образом существенно экономя на обращениях к памяти. Данный статический массив будет находится в локальной памяти, доступ к которой несколько медленее, чем к регистровой, но быстрее, чем к глобальной. Если же нужно часто обращаться к одному элементу массива, то его можно скопировать в локальную переменную, в этом случае будет обращение к регистровой памяти.
3) Использование shared memory и исполнение нитями в блоке одинакового кода. Нагляднее всего это можно продемонстрировать при обходе листьев для подсчета потенциала (силы) из локальных разложений. Можно реализовать параллелизм по частицам, тогда у всех нитей в блоке в общем случае будут различные локальные разложения и будет исполняться различный код. В то же время, параллелизм можно осуществить по листьям, тогда у всех нитей в блоке будет одинаковое локальное разложение, при этом коэффициенты локального разложения можно скопировать в shared memory, оптимизировав обращения к памяти. В этом случае упомянутое выше копирование в локальный массив не даст выигрыша, так как для подсчета потенциала (силы) каждая нить только один раз обращается к коэффициентам разложения; при использовании же shared memory каждой нити достаточно скопировать какую-то часть массива локальных коэффициентов в общий массив, а затем в цикле обращаться уже к локальному массиву – таким образом оптимизируются обращения

к памяти.

4. Результаты расчетов

В данном разделе мы представим результаты расчетов для взаимодействия частиц по закону Ньютона и по закону Био—Савара. Для тестов бралось равномерное распределение частиц в кубе $[0;1]^3$. Погрешность алгоритма будем считать в пространстве L_2, т.е.

$$\varepsilon_{L2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{\mathrm{P}\mathbf{f}_i^{\text{approx}} - \mathbf{f}_i^{\text{exact}}\,\mathrm{P}^2}{\mathrm{P}\mathbf{f}_i^{\text{exact}}\,\mathrm{P}^2}}.$$

Так как код был распараллелен для систем с общей и распределенной памятью, а также с использованием CUDA, будут приведены таблицы, отражающие эффективность распараллеливания. Модель CPU — Intel Core i9-10980XE, GPU — NVIDIA Titan V.

4.1 Параллелизм для систем с общей памятью

В табл. 3-4 представлено время работы (в секундах) различных этапов алгоритма при разном числе потоков (то есть тестируется система с общей памятью).

Табл. 3. Время работы различных этапов БММ для ньютоновского потенциала. Число частиц — 5,000,000, погрешность $\varepsilon_{L2} \approx 10^{-7}$ (в скобках указано ускорение относительно одного потока)

Table 3. The time taken by different stages of the FMM for the Newtonian potential. The particle count is 5,000,000, with the error $\varepsilon_{L2} \approx 10^{-7}$ (the acceleration relative to a single thread is provided in parentheses)

Число потоков	Tree	P2M	M2M	M2L+L2L	P2P+L2P	Total	Direct
1	9.49	3.31	0.36	18.5	42.7	75.2	: 2 мес.
2	6.54	1.65	0.18	9.23	23.25	41.35	: 4 нед.
4	3.05	0.85	0.09	4.61	11.95	21.05	: 2 нед.
8	2.05	0.40	0.042	2.26	6.05	11.26	: 8 дн.
16	1.57 (x6.0)	0.22 (x15.0)	0.025 (x14.4)	1.17 (x15.8)	3.22 (x13.3)	6.66 (x11.3)	: 4 дн.
GPU	0.59 (x16)	0.089 (x37)	0.018 (x20)	0.44 (x42)	0.41 (x104)	1.55 (x49)	: 8 ч.

Табл. 4. Время работы различных этапов БММ для закона Био—Савара. Число частиц — 5,000,000, погрешность $\varepsilon_{L2} \approx 10^{-7}$ (в скобках указано ускорение относительно одного потока)

Table 4. The time taken by different stages of the FMM for the Biot—Savart law. The particle count is 5,000,000, with the error $\varepsilon_{L2} \approx 10^{-7}$ (ehe acceleration relative to a single thread is provided in parentheses)

Число потоков	Tree	P2M	M2M	M2L+L2L	P2P+L2P	Total	Direct
1	0.69	5.44	0.40	69.4	30.6	106.7	: 36 ч
2	0.46	2.92	0.22	36.7	16.2	56.8	: 18 ч
4	0.30	1.48	0.11	18.9	8.55	29.6	: 9 ч
8	0.20	0.89	0.07	11.4	5.35	18.2	: 4.5 ч
16	0.154 (x4.5)	0.46 (x11.8)	0.034 (x11.8)	5.77 (x12.0)	3.17 (x9.7)	9.85 (x10.8)	: 1.2 ч
GPU	0.082 (x8.1)	0.26 (x21)	0.071 (x5.6)	5.78 (x12)	0.40 (x77)	6.61 (x26)	: 250 с

В случае видеокарты видно, что эффективнее всего параллелится обработка листьев и M2L — это самые трудоемкие операции во всем алгоритме, поэтому суммарно получается хорошее ускорение, несмотря на незначительное ускорение остальных процедур.

Отметим, что алгоритм для закона Био—Савара можно реализовать, выполнив три раза БММ для ньютоновского потенциала, однако эффективнее будет реализовать его отдельно, считая заряды частиц векторами, в результате все мультипольные коэффициенты становятся трехмерными векторами из комплексных чисел вместо обычных комплексных чисел. Данный подход позволяет эффективно векторизовать код, и в результате для расчета сил по закону Био—Савара на CPU требуется всего на $20-30\%$ больше времени по сравнению с обычным БММ; наивная реализация потребовала бы, очевидно, в три раза больше времени. Для видеокарты разница между двумя алгоритмами составляет больше двух раз в связи с тем, что больше всего после векторизации зарядов замедляется операция M2L, а ее эффективность распараллеливания не такая высокая, как у обработки листьев, которая не сильно меняется по времени при изменении способа расчеты силы. Связано это прежде всего с тем, что среди операций P2P и L2P большую часть времени занимает прямой расчет P2P, а подсчеты ньютоновской силы и силы по закону Био—Савара по прямой формуле почти не отличаются по времени.

4.2 Параллелизм для систем с распределенной памятью

В табл. 5-6 представлено время работы (в секундах) БММ в зависимости от различного числа MPI процессов и некоторого числа потоков на каждом процессе (то есть тестируется система с общей и распределенной памятью).

В дальнейшем мы приведем сравнение с открытой реализацией БММ, где будет показано, что полученное ускорение как для систем с общей памятью, так и для систем с распределенной памятью является весьма хорошим.

Табл. 5. Время работы БММ для ньютоновского потенциала при разном числе MPI процессов (M) и потоков (N). Число частиц — 5,000,000, погрешность $\varepsilon_{L2} \approx 10^{-7}$ (в скобках указано ускорение относительно последовательной версии)

Table 5. Performance of the FMM for the Newtonian potential with different number of MPI processes (M) and threads (N). The system involves 5,000,000 particles, with an error $\varepsilon_{L2} \approx 10^{-7}$ (acceleration relative to the sequential version is provided in parentheses)

N нитей / *M* проц.	1	2	4	8	16	GPU
1 (без MPI)	78.9	41.1 (x1.92)	24.7 (x3.19)	12.8 (x6.16)	7.15 (x11.0)	2.70 (x29)
2	84.1 (x0.94)	48.8 (x1.62)	24.6 (x3.21)	13.0 (x6.07)	7.3 (x10.8)	—
3	48.7 (x1.62)	25.2 (x3.13)	13.1 (x6.02)	7.2 (x11.0)	4.6 (x17.2)	—
4	21.1 (x3.74)	12.3 (x6.41)	7.02 (x11.2)	4.29 (x18.4)	3.06 (x25.8)	—

Табл. 6. Время работы БММ для закона Био—Савара при разном числе MPI процессов (M) и потоков (N). Число частиц — 5,000,000, погрешность $\varepsilon_{L2} \approx 10^{-7}$ (в скобках указано ускорение относительно последовательной версии)

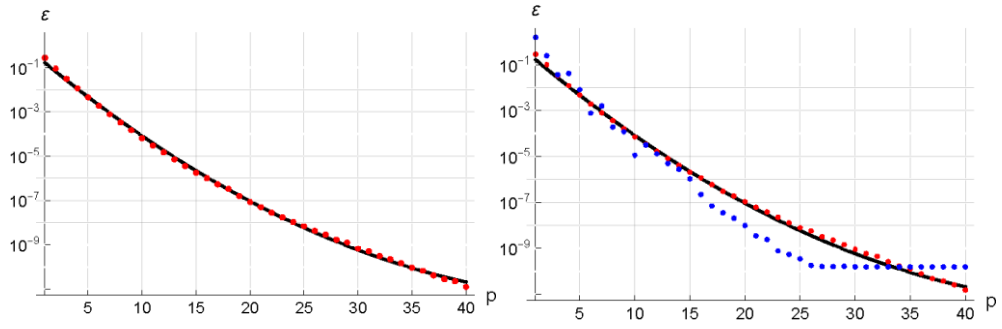
Table 6. Performance of the FMM for the Biot—Savart law with different number of MPI processes (M) and threads (N). The system involves 5,000,000 particles, with an error $\varepsilon_{L2} \approx 10^{-7}$ (acceleration relative to the sequential version is provided in parentheses)

N нитей / *M* проц.	1	2	4	8	16	GPU
1 (без MPI)	106.7	56.8 (x1.88)	29.6 (x3.60)	18.2 (x5.86)	9.85 (x10.8)	6.61 (x16)
2	107 (x0.99)	57.9 (x1.84)	36.0 (x2.96)	18.7 (x5.71)	10.2 (x10.5)	—

3	55.1 (x1.94)	30.5 (x3.50)	19.0 (x5.6)	10.0 (x10.7)	6.3 (x16.9)	—
4	28.8 (x3.70)	15.8 (x6.75)	8.84 (x12.1)	5.97 (x17.9)	4.0 (x26.7)	—

4.3 Определение количества мультиполей и глубины дерева

В быстром методе мультиполей имеются две величины, которые необходимо определить перед выполнением программы. Первая — это число мультиполей, которое влияет на точность расчета. Для определения зависимости погрешности от числа мультиполей мы рассмотрим задачу со $100\,000$ частиц (были рассмотрены варианты и с другим числом частиц, но результаты везде оказались схожие). Результаты изображены на рис. 3.



а) закон Ньютона *б) закон Био—Савара*

Рис. 3. Ошибка ε_{L2} для подсчета силы (красный цвет) и потенциала (синий цвет) в зависимости от числа мультиполей. Черным цветом изображена аппроксимация зависимости $\varepsilon(p)$ для силы.

Fig. 3. The error ε_{L2} in calculating force (red) and potential (blue) as a function of the number of multipoles. The approximation of the force error $\varepsilon(p)$ is represented in black.

В результате получилось, что погрешности для ньютоновской силы и для силы по закону Био—Савара имеют схожие величины. Интересно поведение ошибки вычисления потенциала — при небольшом числе мультиполей она убывает быстрее, чем для силы, а затем выходит на плато.

Погрешность для силы была аппроксимирована с помощью метода наименьших квадратов, в результате получилось

$$\varepsilon(p) \approx \exp(0.00868\,p^2 - 0.94\,p - 0.88), \quad p(\varepsilon) \approx 54.2 - 10.73\sqrt{\ln(2.4\varepsilon) + 25.5}.$$

Данные формулы удобно использовать в алгоритме для выбора числа мультиполей в зависимости от требуемой точности.

Другой важный параметр — глубина дерева. Глубина дерева влияет на время расчета, и для каждой конкретной задачи существует оптимальная глубина. Однако эта величина зависит от многих факторов: число мультиполей, оптимальность реализации различных этапов алгоритма (в первую очередь, M2L и P2P — они занимают наибольшее время), а также само расположение частиц: при равномерном распределении и при сильно неоднородном оптимальная глубина дерева может сильно отличаться. В этой связи мы не можем привести конкретные формулы для определения оптимальной глубины дерева.

4.4 Сравнение с другими реализациями

Будем проводить сравнение для задачи взаимодействия частиц по закону Ньютона. На рис. 4 представлено сравнение с реализацией [14], где есть однопоточный CPU вариант и GPU вариант.

В случае кода для CPU текущая реализация работает быстрее примерно в 3 раза; в случае же видеокарты ускорение куда более существенное и достигает 10 раз.

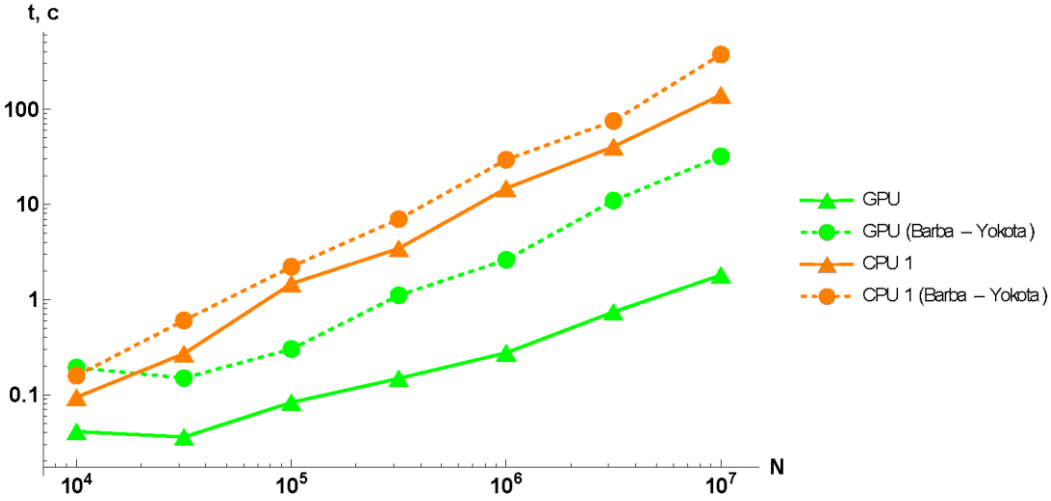


Рис. 4. Сравнение текущей реализации БММ с реализацией R. Yokota и L.A. Barba на видеокарте (GPU) и на одном потоке процессора (CPU 1).
Fig. 4. The comparison between the proposed FMM implementation and R. Yokota and L.A. Barba's GPU implementation and sequential CPU version (CPU 1).

Подробный анализ качества реализации CPU версии можно провести, сравнивая с реализацией [18], которая включает распараллеливание для систем с общей и распределенной памятью, а также, что очень важно, векторизацию P2P (рис. 5). Результаты на одном потоке получились практически идентичные; на 18 потоках текущая реализация несколько лучше параллелится, в то время как с добавлением MPI на очень большом числе частиц ($>10^7$) возможно небольшое отставание; в целом же, временные результаты получились схожими.

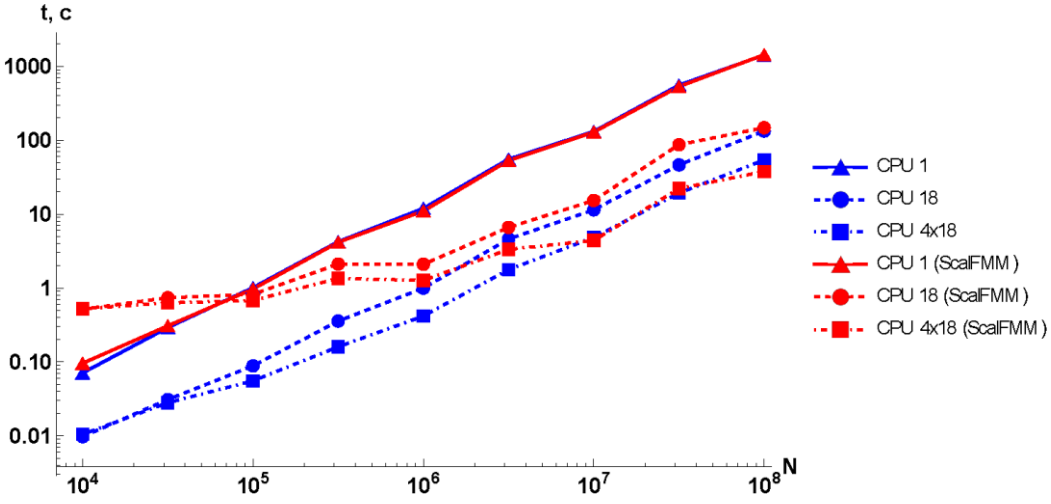


Рис. 5. Сравнение текущей реализации трехмерного БММ со ScalFMM на одном потоке (CPU 1), на 18 потоках (CPU 18) и на 4 узлах с 18 потоками (CPU 4x18).
Fig. 5. Comparison of the proposed implementation of the FMM with ScalFMM on a single thread (CPU 1), on 18 threads (CPU 18), and on 4 nodes with 18 threads each (CPU 4x18).

Данные сравнения подтверждают эффективность текущей реализации трехмерного быстрого метода мультиполей.

5. Заключение

В работе представлено описание быстрого метода мультиполей для частиц, взаимодействующих по закону Ньютона и по закону Био—Савара. Представлены все необходимые для расчета формулы, а также упомянута нормировка сферических гармоник и матриц Вигнера – тематика, редко освещаемая в других работах. Кроме теоретичеких формул, приведены рекуррентные соотношения, позволяющие запрограммировать алгоритм; кроме этого, подробно описаны техники, позволяющие существенно ускорить работу кода, в том числе подробно изложено построение дерева с помощью кривой Мортона, что позволяет эффективно распараллелить код на CPU и GPU. С использованием данных техник был реализован программный комплекс на языке C++, распараллеленный для систем с общей памятью (с помощью библиотеки TBB) и для систем с распределенной памятью (с использованием MPI), а также для видеокарт с использованием технологии CUDA. Для анализа эффективности написанной программы было проведено сравнение с двумя реализациями, которые также включают различные способы распараллеливания, и наглядно подтверждена высокая скорость работы кода.

Список литературы / References

[1]. J. Barnes, P. Hut. A hierarchical O(NlogN) force-calculation algorithm. // Nature. 1986. Vol. 324. P. 446–449. DOI: 10.1038/324446a0.

[2]. K. Nabors, J. White. FastCap: A multipole-accelerated 3-D capacitance extraction program // IEEE Transactions on Computer-Aided Design. 1991. Vol. 10. P. 1447–1459.

[3]. M. Kamon, M. J. Tsuk, J. K. White. FASTHENRY: a multipole-accelerated 3-D inductance extraction program. // IEEE Transactions on Microwave Theory and Techniques. 1994. Vol. 42, № 9. P. 1750–1758.

[4]. L. Greengard, M.C. Kropinski, A. Mayo. Integral equation methods for stokes flow and isotropic elasticity in the plane. // J. Comput. Physics. 1996. Vol. 125, № 2. P. 403–414. DOI: 10.1006/jcph.1996.0102

[5]. A.-K. Tornberg, L. Greengard. A fast multipole method for the three-dimensional Stokes equations. // Journal of Computational Physics. 2008. Vol 227, № 3. P. 1613–1619. DOI: 10.1016/j.jcp.2007.06.029

[6]. L. Greengard, V. Rokhlin. A fast algorithm for particle simulations. // J. Comput. Phys. 1987. Vol. 73, № 2. P. 325–348. DOI: 10.1016/0021-9991(87)90140-9

[7]. H. Cheng, L. Greengard, V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. // J. Comput. Phys. 1999. Vol. 155. P. 468–498. DOI: 10.1006/jcph.1999.6355

[8]. M. A. Epton, B. Dembart. Multipole translation theory for the three-dimensional Laplace and Helmholtz equations. // SIAM J. on Scientific Computing. 1995. Vol. 16, №4. P. 865–897. DOI: 10.1137/0916051

[9]. R. Beatson, L. Greengard. A short course on fast multipole methods. // Wavelets, Multilevel methods and Elliptic PDEs. 1997. P. 1–37.

[10]. J. Makino. Yet another fast multipole method without multipoles — pseudoparticle multipole method. // J. Comput. Phys. 1999. Vol. 151, № 2. P. 910–920.

[11]. A. Kawai, J. Makino. A simple formulation of the fast multipole method: pseudo-particle multipole method. // SIAM Conference on Parallel Processing for Scientific Computing. 1998. DOI: 10.48550/arXiv.astro-ph/9812431

[12]. N.Y. Gnedin. Hierarchical Particle Mesh: An FFT-accelerated fast multipole method. // The Astrophysical Journal Supplement Series. 2019. Vol. 243, № 2. 19 p. DOI: 10.3847/1538-4365/ab2d24

[13]. C.R. Anderson. An implementation of the fast multipole method without multipoles. // SIAM J. on Scientific and Statistical Computing. 1992. Vol. 13. P. 923–947. DOI: 10.1137/0913055

[14]. R. Yokota, L.A. Barba. Treecode and fast multipole method for N-Body simulation with CUDA. // GPU Computing Gems. Emerald Edition. Boston: Elsevier and Morgan Kaufmann, 2011. P. 113–132. DOI: 10.1016/B978-0-12-384988-5.00009-7

[15]. B. Bramas. (2020). TBFMM: A C++ generic and parallel fast multipole method library. // Journal of Open Source Software. 2020. Vol. 5, № 56. P. 2444–2453. DOI: 10.21105/joss.02444

[16]. Z. Gimbutas, L. Greengard. Computational Software: Simple FMM Libraries for Electrostatics, Slow Viscous Flow, and Frequency-Domain Wave Propagation. // Communications in Computational Physics. 2015. Vol. 18, № 2. 516–528. DOI: 10.4208/cicp.150215.260615sw

[17]. T. Wang, R. Yokota, L.A. Barba. ExaFMM: a high-performance fast multipole method library with C++ and Python interfaces. // Journal of Open Source Software. 2021. Vol. 6, № 61. P. 3145–3149. DOI: 10.21105/joss.03145
[18]. P. Blanchard, B. Bramas, O. Coulaud, et al. ScalFMM: A Generic Parallel Fast Multipole Library. // SIAM Conference on Computational Science and Engineering. 2015.
[19]. E.O. Steinborn, K. Ruedenberg. Rotation and translation of regular and irregular solid spherical harmonics. // Advances in Quantum Chemistry. 1973. Vol. 7. P. 1–81.
[20]. L. Biedenharn, J. Louck, P. Carruthers. Angular momentum in quantum physics: Theory and Application. Cambridge: Cambridge University Press, 1984. 716 p.
[21]. Abramowitz M., Stegun I.A. Handbook of mathematical functions. New York: Dover, 1965. 1046 p.
[22]. L. Greengard, V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. // Acta Numerica. 1997. Vol. 6. P. 229–269. DOI: 10.1017/S0962492900002725
[23]. E. Wigner. Group theory and its application to the quantum mechanics of atomic spectra. New York: Academic Press, 1959. 384 p. DOI: 10.1017/S0013091500025220
[24]. H. Dachsel. Fast and accurate determination of the Wigner rotation matrices in the fast multipole method. // J. Chem. Phys. 2006. Vol. 124. P. 144115–144121. DOI: 10.1063/1.2194548
[25]. Z. Gimbutas, L. Greengard. A fast and stable method for rotating spherical harmonic expansions. J. Comput. Phys. 2009. Vol. 228, № 16. P. 5621-5627. DOI: 10.1016/j.jcp.2009.05.014

Информация об авторах / Information about authors

Виктор Михайлович АУШЕВ – студент магистратуры кафедры “Прикладная математика”. Сфера научных интересов: быстрые методы, высокопроизводительные вычисления, численные методы электродинамики, метод конечных элементов, численные методы линейной алгебры.

Viktor Mikhailovich AUSHEV – master's student of Applied Mathematics department. Research interests: fast methods, high performance computing, numerical methods of electrodynamics, finite element method, numerical methods of linear algebra.