N. N. Popova, N. G. Nikishin, Performance study of graphic processors usage in computational nanotechnology problems,
Comp. nanotechnol., 2014, Issue 2, 5–12

1. МОДЕЛИРОВАНИЕ НАНОСИСТЕМ И НАНОЭЛЕКТРОНИКА

1.1. ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ИСПОЛЬЗОВАНИЯ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ ДЛЯ РЕШЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ ЗАДАЧ НАНОТЕХНОЛОГИЙ

Попова Нина Николаевна, доц., доц., канд. физ.-мат. наук, факультет вычислительной математики и кибернетики, Московский Государственный Университет им. М.В. Ломоносова, popova@cs.msu.su

Никишин Николай Глебович, аспирант, факультет вычислительной математики и кибернетики, Московский Государственный Университет им. М.В. Ломоносова, mail@nikishinng.ru

Аннотация: Вычислительные нанотехнологии неотъемлемо связаны с использованием современных высокопроизводительных систем. С быстрым развитием аппаратного и программного обеспечения графических процессоров (GPU) их использование набирает популярность для задач, требовательных к вычислительной мощности. К числу таких задач относятся исследования в области наноматериалов, изучение и создание которых проводятся с использованием масс-спектрометров. Работа посвящена моделированию поведения ионов в ловушках масс-спектрометров на основе ионного циклотронного резонанса и преобразования Фурье. Решение задачи проводится на основе модели частиц в ячейке, которая применяется для прямого моделирования поведения ионов. Вычисления на графическом процессоре проводятся с использованием библиотек для GPU cuFFT и CULA.

В работе показано, что использование GPU-ориентированных библиотек существенно упрощает разработку параллельных алгоритмов для графических процессоров и позволяет достичь хорошую производительность параллельных приложений. Решение задачи выполнено на суперкомпьютере «Ломоносов», установленном в МГУ имени М.В. Ломоносова. В работе показано, что различные стратегии распределения параллельных процессов по узлам многопроцессорной вычислительной системы могут существенно влиять на производительность всей программы из-за одновременного доступа нескольких процессов к графическим ускорителям.

Полученные в работе результаты могут быть полезными для моделирования больших молекулярных структур, решения задач в области вычислительных нанотехнологий на современных высокопроизводительных параллельных вычислительных системах с использованием графических ускорителей

Ключевые слова: высокопроизводительные вычисления, неоднородные вычислительные системы, нанотехнологии, масс-спектроскопия, CUDA, Быстрое преобразование Фурье.

1.1. PERFORMANCE STUDY OF GRAPHIC PROCESSORS USAGE IN COMPUTATIONAL NANOTECHNOLOGY PROBLEMS

Popova Nina N., PhD., associate professor, PhD in Physics and Mathematics, Lomonosov Moscow State University, Department of Computational Mathematics and Cybernetics. popova@cs.msu.su

Nikishin Nikolai G., postgraduate, Computational Mathematics and Cybernetics Department, Lomonosov Moscow State University, mail@nikishinng.ru

Abstract: Computational nanotechnology is strongly connected with modern high-performance computing. With fast evolution of graphical processing units (GPU), general-purpose computing (GPGPU) became a popular choice for computationally demanding tasks. Tasks in nanomaterial researches with mass-spectrometers usage for analysis and material creation are good examples of such kind of tasks. Paper is devoted to ions behavior modeling in traps of mass-spectrometers based on Fourier transform. We use heterogeneous computational systems with GPU inside for calculation. Particle-in-cell model is used for direct modeling of ions behavior. We also use two GPU libraries: CULA and cuFFT.

Paper shows, that some GPU-oriented libraries could significantly ease the development of parallel algorithms for GPU and allow to get good performance of parallel applications. Calculations were performed on several systems including "Lomonosov" supercomputer in Moscow State University. Paper shows, that different strategies of mapping parallel processes to nodes could significantly effects on performance because of parallel access to an every single GPU from multiple processes.

Results obtained in this work could be useful for big molecular structure modeling, for solving computational nanotechnology problems on modern high-performance parallel computational systems with GPU

Index terms: HPC, heterogeneous computing systems, nanotechnology, mass-spectrometry, CUDA, Fast Fourier transform

1. Введение.

Вычислительные нанотехнологии неотъемлемо связаны с использованием современных высокопроизводительных систем. Использование ускорителей, интегрируемых в состав процессоров, является основной тенденцией развития таких систем. С быстрым развитием аппаратного и программного обеспечения графических процессоров (GPU) их использование набирает популярность для задач, требовательных к вычислительной мощности [1]. Системы, основанные на использовании GPU, привлекают особое внимание исследователей [2]. Это ставит вопрос об эффективности применения таких систем для моделирования больших молекулярных структур, в особенности, для разработок в сфере нанотехнологий [3]. Исследования, связанные с изучением и созданием новых наноматериалов, активно проводятся с использованием масс-спектрометров. В работе исследуется эффективность высокопроизводительных вычислений при проведении программного моделирования поведения ионов в масс-спектрометрах, построенных на основе ионного циклотронного резонанса и преобразования Фурье. В основу решения рассматриваемой задачи принята широко распространенная модель частиц в ячейках [4]. Для прямого моделирования изучаемых устройств, а также для исследования физических эффектов, влияющих на точность работы этих устройств, предложены методы и реализован ряд программ [5], [6]. В них расчет парного кулоновского взаимодействия частиц осуществляется через вычисление на трехмерной разностной сети коллективного самосогласованного поля на каждом шаге по времени на основе решения задачи для уравнения Пуассона с использованием быстрого преобразования Фурье [5]. Данная работа продолжает исследования в этом направлении. В работе предлагается параллельный алгоритм, основанный на использовании графических ускорителей. Основное внимание в работе уделяется тому, насколько GPU позволяют ускорить вычисления в модели частиц в ячейке. Для решения краевой задачи для уравнения Пуассона предлагается использование библиотека cuFFT для быстрого преобразования Фурье. В работе исследуется эффективность нахождения решения задачи Пуассона для различных параметров и проводится анализ асимптотической сложности решения. Расчет внешних удерживающих полей на разностной сетке от системы электродов производится с помощью параллельных алгоритмов решения пограничных интегральных уравнений [6]. Задача сводится к решению алгебраической системы уравнений с плотной матрицей. Решение системы проводится с использованием библиотеки CULA, реализующей основные операции линейной алгебры на основе технологии NVIDIA CUDA. Предложенный параллельный алгоритм решения трехмерной задачи реализуется на основе технологии параллельного программирования MPI. В работе проводится сравнение различных стратегий распараллеливания вычислений с возможностью использования нескольких параллельных потоков на GPU. Практическая реализация задачи выполнена на суперкомпьютере «Ломоносов» НИВЦ МГУ и на ряде вычислительных систем с различными графическими ускорителями.

Полученные в работе результаты и приведенные на их основе рекомендации могут быть полезны при построении численных кодов для решения различных задач, связанных с применением вычислительных нанотехнологий.

2. Задача моделирования эволюции частиц в ячейке.

Рассмотрим модель частиц в ячейке при моделировании поведения многих ионов в ловушках различных установок, в частности, масс-спектрометров [5]. Чтобы определить траектории частиц, необходимо решить N уравнений Ньютона для каждой частицы под действием силы Лоренца.

$$m_i \frac{dv_i}{dt} = q_i E^{\text{trap}}(r_i,t) + q_i[v_i \cdot B] + \frac{1}{4\pi\varepsilon_0}\sum_{j=1}^{N_p}\frac{q_i q_j (r_i - r_j)}{|r_i - r_j|^3} + E^{\text{wall}}(r_i) + F_{\text{coll}}(v_i) + E^{\text{exct}}(r_i)\cos\gamma(t),$$

где электрическое поле E определяется по вычисленным потенциалам: $u^{\text{trap}}(r_i)$— потенциалу удерживающего поля ловушки, $u^{\text{rf}}(r_i,t)$ – потенциалу поля возбуждения (заданного по времени) и $u^{\text{c}}(r_i,t)$ – кулоновскому потенциалу парного взаимодействия частиц, который необходимо рассчитывать на каждом шаге по времени.

Через u^{wall} обозначим значение потенциала на стенках электродов. Тогда:

$$E = -\nabla u, \quad u = u^{\text{trap}}(r_i) + u^{\text{rf}}(r_i,t) + u^{\text{c}}(r_i,t) + u^{\text{wall}} \qquad (1)$$

Наиболее трудозатратными в методе частица-сетка являются процедуры расчета сил кулоновского взаимодействия частиц и вычисления поля ловушки, удерживающего ионы, которое создаётся системой электродов произвольной формы. При этом в задаче моделирования масс-спектрометра приходится моделировать эволюцию в течение нескольких миллионов шагов по времени порядка миллиона ионов. Использование графических ускорителей для выполнения данных процедур представляет основной интерес.

3. Решение краевой задачи для уравнения Пуассона с целью определения кулоновского взаимодействия ионов.

Во избежание попарного вычисления взаимодействий между частицами, вводится пространственная трехмерная разностная сетка (x_k, y_l, z_m). Потенциал кулоновского поля определяется плотностью заряда $\rho(r_i,t)$, который вычисляется на сетке при помощи интерполяции зарядов частиц q_i внутри ячейки, в узлы данной разностной сети:

$$\rho(k,l,m) = \frac{1}{\mathrm{Vol}} \sum_{i}^{N_p} q_i W_{k,l,m}(x_i, y_i, z_i). \qquad (2)$$

где Vol есть объем ячейки, а $W_{k,l,m}$ – веса интерполяции, величина которых зависит от положения частицы относительно узла (k, l, m). Тогда коллективный потенциал кулоновского поля u^c находится из решения первой краевой задачи для уравнения Пуассона на пространственной сетке

$$\Delta u^c = -\frac{1}{\varepsilon_0}\rho(r_i,t), \quad u^c\,|_{\mathrm{bound}} = u^{\mathrm{wall}}(r,t), \qquad (3)$$

и по найденному потенциалу вычисляется сила, действующая на все ионы внутри ячейки:

$$F(x_i, y_i, z_i) = \sum_{\mathrm{grid}} q_i W_{\mathrm{grid}}(x_i, y_i, z_i) E_{\mathrm{grid}}. \qquad (4)$$

В результате количество действий для учета парного взаимодействия меняется с $O(N_p \cdot N_p)$ на $O(N_p \cdot M)$, где M – число ячеек сети.

Чтобы избежать флуктуаций силы при переходе через границы ячеек, число рассматриваемых частиц должно быть велико по сравнению с размерностью пространственной сетки. Для этого, как правило, берут порядка ста частиц на одну ячейку.

Переформулируем задачу (3) для трехмерной разностной сетки размера $N_1 \times N_1 \times N_2 \times N_3$, равномерно покрывающей прямоугольную область $[0, L_1] \times [0, L_2] \times [0, L_3]$. Тогда координаты узлов сетки задаются следующими выражениями:

$$x_i = kh_i, \quad i = 1,2,3, \qquad (5)$$

где $h_i = L_i / N_i$— шаги разностной сетки по каждому из направлений. Постановка задачи при этом примет вид:

$$\Delta_h u_h^c = \rho_h, \quad u_h^c\,|_{bound} = 0,$$

где $u^c{}_h$ и ρ_h – сеточные функции, заданные значениями функций u^c и ρ в узлах сетки, а Δ_h – разностная аппроксимация трехмерного оператора Лапласа.

Для удовлетворения нулевым граничным условиям в качестве функций базиса разложения следует использовать ортогональную систему произведений синусов, обращающихся в нуль на границе области. Дискретное синус-преобразование Фурье задаётся следующей формулой:

$$F_s[f](k,l,m) = \sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1}\sum_{j_3=0}^{N_3-1}\left[f(j_1,j_2,j_3)\sin\left(\frac{\pi j_1 k}{N_1}\right)\sin\left(\frac{\pi j_2 l}{N_2}\right)\sin\left(\frac{\pi j_3 m}{N_3}\right)\right]$$

Выполнив синус-преобразование Фурье левой и правой частей, перейдём к уравнению в пространстве частот:

$$\lambda_h \hat{u}_h^c = F_s[\rho_h],$$

где $\lambda_h(k,l,m)$ – собственные числа разностной задачи Дирихле. Они равны [7]

$$\lambda_h(k,l,m) = \sum_{\alpha=1}^{3} \frac{4}{h_\alpha^2}\sin^2\left(\frac{k_\alpha \pi h_\alpha}{2l_\alpha}\right), k_1 = k, k_2 = l, k_3 = m.$$

Поделив обе части уравнения на собственные числа, получим образ решения задачи:

$$\hat{u}_h^c = \frac{F_s[\rho_h]}{\lambda_h}.$$

Для получения потенциала на сетке остаётся только сделать обратное преобразование.

$$u_h^c = F_s^{-1}\left[\frac{F_s[\rho_h]}{\lambda_h}\right].$$

4. Программная реализация решения краевой задачи для уравнения Пуассона с использованием библиотеки cuFFT.

Программная реализация коллективного потенциала кулоновского поля действий с использованием GPU представлена в виде модуля, состоящего из трёх процедур.

Первая процедура включает в себя инициализацию работы с графическим ускорителем, выделение необходимой памяти на GPU, вычисление собственных значений для дальнейшего использования в программе.

Вторая процедура предназначена для решения краевой задачи для уравнения Пуассона в памяти GPU с использованием дискретного преобразования Фурье.

Третья процедура освобождает выделенную память. Чтобы избежать повторных вычислений, инициализация и завершение работы с ускорителем вынесены за пределы временного цикла.

Взаимодействие с основной программой осуществляется через интерфейсный модуль, обеспечивающий возможность вызова процедур, написанных на языке NVIDIA CUDA, из основной программы на языке Fortran. Для гарантии соответствия типов при его описании использовались именованные константы, предоставляемые модулем ISO_C_BINDING, который входит в стандарт языка Fortran 2003. В главной процедуре, реализующей решения уравнения, для выполнения преобразования Фурье используются функции библиотеки cuFFT версии 5.5 из состава программного пакета CUDA Toolkit. Библиотека cuFFT не содержит процедур для синус-преобразований. В ней содержатся только процедуры дискретного преобразования Фурье (ДПФ). Для получения синус-преобразования на интересующей нас области расширим нашу сеточную функцию в два раза по каждому из направлений, продолжив её нечётно относи-

тельно бывших границ. Это преобразование задаётся следующими равенствами:

$$f(j_1,j_2,j_3)=f(j_1,j_2,j_3),$$
$$f(2N_1-j_1,j_2,j_3)=f(j_1,2N_2-j_2,j_3)=f(j_1,j_2,2N_3-j_3)=-f(j_1,j_2,j_3),$$
$$f(2N_1-j_1,2N_2-j_2,j_3)=f(2N_1-j_1,j_2,2N_3-j_3)=f(j_1,2N_2-j_2,2N_3-j_3)=f(j_1,j_2,j_3),$$
$$f(2N_1-j_1,2N_2-j_2,2N_3-j_3)=-f(j_1,j_2,j_3),\ j_i=1,\ldots,N_i-1,$$
$$f(j_1,j_2,j_3)=0,\ j_i=0,N_i,2N_i,\ i=1,2,3.$$

В этих предположениях, для любого i = 1,2,3 верно, что

$$\sum_{j_i=N_i}^{2N_i-1} f(\ldots,j_i,\ldots)e^{2\pi\mathrm{i}j_i m/(2N_i)}=\{j_i'=2N_i-j_i\}=$$
$$=\sum_{j_i'=1}^{N_i} f(\ldots,2N_i-j_i',\ldots)e^{2\pi(2N_i-j_i')m/(2N_i)}=-\sum_{j_i'=0}^{N_i-1} f(\ldots,j_i',\ldots)e^{-2\pi\mathrm{i}j_i' m/(2N_i)}.$$

Используя эту формулу, покажем, что ДПФ от расширения нашей функции даст нужный результат на интересующей нас части сетки.

$$F[f](k,l,m)=\sum_{j_1=0}^{2N_1-1}\sum_{j_2=0}^{2N_2-1}\sum_{j_3=0}^{2N_3-1}\left[f(j_1,j_2,j_3)e^{2\pi\mathrm{i}j_1 k/(2N_1)}e^{2\pi\mathrm{i}j_1 l/(2N_2)}e^{2\pi\mathrm{i}j_3 m/(2N_3)}\right]=$$
$$=\sum_{j_1=0}^{2N_1-1}\sum_{j_2=0}^{2N_2-1}\left\{e^{2\pi\mathrm{i}j_1 k/(2N_1)}e^{2\pi\mathrm{i}j_1 l/(2N_2)}\sum_{j_3=0}^{N_3-1}\left[f(j_1,j_2,j_3)\left(e^{2\pi\mathrm{i}j_3 m/(2N_3)}-e^{-2\pi\mathrm{i}j_3 m/(2N_3)}\right)\right]\right\}=$$
$$=\sum_{j_1=0}^{2N_1-1}\sum_{j_2=0}^{2N_2-1}\left\{e^{2\pi\mathrm{i}j_1 k/(2N_1)}e^{2\pi\mathrm{i}j_1 l/(2N_2)}2i\sum_{j_3=0}^{N_3-1}\left[f(j_1,j_2,j_3)\sin\left(\frac{\pi j_3 m}{N_3}\right)\right]\right\}=$$
$$=-8i\sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1}\sum_{j_3=0}^{N_3-1}\left[f(j_1,j_2,j_3)\sin\left(\frac{\pi j_1 k}{N_1}\right)\sin\left(\frac{\pi j_2 l}{N_2}\right)\sin\left(\frac{\pi j_3 m}{N_3}\right)\right]=-8iF_s[f](k,l,m).$$

После проведения прямого и обратного преобразований необходимо провести нормировку на общее количество элементов сетки, так как процедуры cuFFT не делают это автоматически.

Решение находится по следующей цепочке действий. Значения сеточной функции ρ_h пересылаются из основной оперативной памяти в память GPU. Затем эти значения переупорядочиваются из массива с индексацией «по столбцам», принятого в языке Fortran, в массив с индексацией «по строкам», естественного для языков семейства C. Сеточная функция ρ_h нечетно продолжается в два раза по каждому из направлений. Далее производится дискретное преобразование Фурье продолженной сеточной функции. Все мнимые части результата (только они не равны нулю) делятся на собственные значения разностной задачи. Затем производится обратное дискретное преобразование Фурье образа решения. После обратного переупорядочивания «по столбцам» массив значений сеточной функции u^c пересылается обратно в ОЗУ.

В предположении, что исходная трехмерная сетка содержит N_g узлов по каждому из направлений, количество операций с плавающей точкой для осуществления прямого и обратного преобразования Фурье, а также деления на собственные числа оценивается по следующим формулам

$$N_{op}=2N_{FFT}+(2N_g)^3,\quad N_{FFT}=5(2N_g)^3\log_2\left((2N_g)^3\right), \qquad (6)$$

где N_{op} – общее количество операций, а N_{FFT} – количество операций, необходимое для реализации алгоритма быстрого преобразования Фурье над массивом комплексных чисел. Общая асимптотическая сложность составляет $O(N_g^3\log(N_g))$.

5. Оценки эффективности однопроцессорной программной реализации с использованием различных графических процессоров.

Для оценки технических характеристик графических процессоров рассчитываются следующие величины:

- максимально возможная пропускная способность памяти,

- пиковая производительность на операциях с одинарной точностью,

- пиковая производительность на операциях с двойной точностью.

Данные величины могут быть получены с помощью вызовов CUDA runtime API и зависят только от характеристик графических карт.

$$\mathrm{BW}_{\mathrm{theor}}\,(\mathrm{GB/s})=\frac{\mathrm{Memory\,Clock\,rate\,(MHz)}\cdot 10^6\cdot\mathrm{MemoryBusWidth\,(Bits)}/8}{10^9}, \qquad (7)$$
$$\mathrm{Perf}_{\mathrm{peak}}^{\mathrm{sp}}\,(\mathrm{GFLOPS})=2\cdot\mathrm{Total\,CUDA\,Cores}\cdot\mathrm{GPU\,Clock\,rate\,(GHz)},$$
$$\mathrm{Perf}_{\mathrm{peak}}^{\mathrm{dp}}\,(\mathrm{GFLOPS})=2\cdot\mathrm{Total\,CUDA\,DPU}\cdot\mathrm{GPU\,Clock\,rate\,(GHz)}.$$

Время решения задачи для уравнения Пуассона замерялось на каждом шаге по времени и затем вычислялось среднее время по 100 шагам. Решение считалось удовлетворительным, если абсолютная ошибка по сравнению с точным аналитическим решением составляла величину ~10^{-5}.

Производительность программ с использованием GPU оценивается с помощью двух метрик [8]: $\mathrm{BW}_{\mathrm{eff}}$ – достигнутой пропускной способности памяти и $\mathrm{Perf}_{\mathrm{eff}}$ – достигнутой производительности. Значение $\mathrm{BW}_{\mathrm{eff}}$ вычисляется по формуле:

$$\mathrm{BW}_{\mathrm{eff}}\,(\mathrm{GB/s})=\frac{R_B+W_B}{t\cdot 10^9}, \qquad (8)$$

где R_B – количество прочитанных из памяти GPU (в байтах), а W_B – количество записанных в память данных. В нашем случае $R_B = W_B = N^3$, то есть нам требуется чтение сеточной функции ρ_h и запись ответа u_h.

$$\mathrm{Perf}_{\mathrm{eff}}\,(\mathrm{GFLOPS})=\frac{N_{op}}{t\cdot 10^9}. \qquad (9)$$

Мы не будем рассматривать сетки с $N < 32$ с целью достаточной загрузки мультипроцессоров GPU. Время решения задачи в зависимости от числа точек (одинаковое число точек по каждому из направлений) на устройстве GeForce GTX TITAN для одинарной и двойной точности показано в таблице 1.

Таблица 1

Время решения задачи Дирихле на устройстве GeForce GTX TITAN. Усреднение по 100 шагам по времени

N_g	Одинарная точность	Двойная точность
32	0.001152	0.001475
64	0.004569	0.007384
128	0.026510	0.053721
256	0.259035	0.520362

Аппроксимация в смысле наименьших квадратов кривыми $Cx^3\log_2(x)$, соответствующими асимптотической оценке сложности, представлена на рис. 1. Накладные расходы начинают играть все меньшую

роль с увеличением размера задачи, уступая вычислительной составляющей. Время вычислений для нахождения решения составляет от 50% до 70% от общего времени выполнения процедуры и растет с увеличением размера задачи. Отношение времени, затраченного на вычисление решения, к общему времени выполнения процедуры, включающему пересылку и подготовку данных, для различных размеров задачи показано в таблице 2.

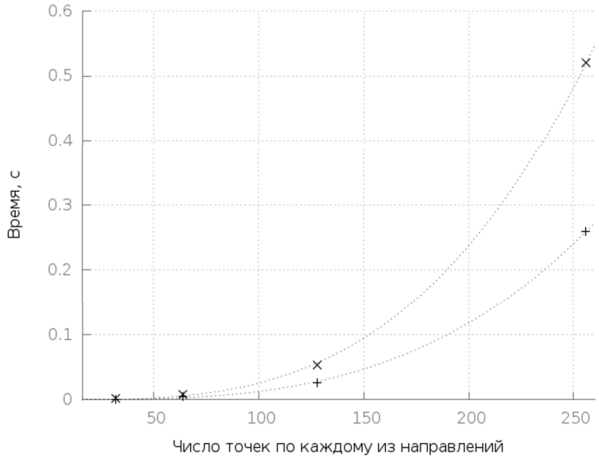


Рисунок 1. Время решения первой краевой задачи для уравнения Пуассона для одинарной (+) и двойной (×) точности. Пунктиром показаны кривые вида $Cx^3\log_2(x)$ наилучшим образом аппроксимирующие числовые данные в смысле наименьших квадратов.

Таблица 2

Доля вычислений от общего времени решения задачи одинарной точности на GeForce GTX TITAN

N_g	Решение (с)	Общее время (с)	Доля решения в % от общего
64	0.002461	0.004569	53.862990
128	0.016318	0.026510	61.554131
256	0.188552	0.259035	72.790163

Сравнение эффективности различных ускорителей было проведено для характерного для задачи числа точек по каждому из направлений N_g = 128. Соответствующие результаты приведены в таблице 3.

Таблица 3

Сравнение времени решения задачи Дирихле, пиковой и достигнутой пропускной способности памяти, пиковой и достигнутой производительности на GeForce GT 640, Tesla C2075 и GeForce GTX TITAN. Усреднение по 100 итерациям для решения задачи с одинарной точностью N_g = 128.

GPU	Время, с	BW_{theor}, ГБ/с	BW_{eff}, ГБ/с	$Perf^{sp}_{peak}$, ТФЛОПС	$Perf^{sp}_{eff}$, ТФЛОПС
GeForce GT 640	0.090254	40.08	0.05	0.81	0.0448
Tesla C2075	0.037220	150.34	0.11	1.03	0.1086
GeForce GTX TITAN	0.026510	208.00	0.16	3.54	0.1525

Из таблицы видно сильное превосходство ускорителей GeForce GTX TITAN и Tesla C2075 над GeForce GT 640. Это объясняется большим количеством потоковых мультипроцессоров (14 против 2) и повышенной пропускной способностью памяти. GeForce GTX TITAN оказывается быстрее Tesla C2075 за счет большего количества ядер на мультипроцессоре.

В исследуемой задаче количество точек сетки выбиралось разумным для серийных расчетов по модели частиц в ячейках и составило порядка $1{,}7 \cdot 10^7$ точек. Такое количество данных допускается широтой пропускания памяти, но после чтения исходных данных мы каждый раз вынуждены выполнять транспонирование и расширение массива в 8 раз, что вызывает заметные задержки, так как память GPU обладает высокой латентностью. В зависимости от ускорителя достигнутая производительность составляет от 44 до 150 ГФЛОПС, что соответствует 4–10% от пиковой производительности. Описанные выше особенности алгоритма не позволяют в полной мере задействовать все ресурсы GPU, однако решение задачи осуществляется за хорошее время и может использоваться в приложениях.

6. Вычисление потенциалов удерживающего поля от электродов произвольной формы.

Для вычисления потенциала поля от незамкнутых электродов произвольной формы требуется решить уравнение Лапласа

$$\Delta V = 0 \qquad (10)$$

с граничными условиями первого рода на электродах, окружающих ловушку:

$$V|_{\Sigma_k} = U_k, \qquad (11)$$

где k = 1,2,3,…,K – номер поверхности электрода, и U_k – заданный постоянный потенциал на каждой поверхности.

Мы ищем решение как сумму потенциалов двойного слоя W_k, создаваемого каждой поверхностью:

$$V(M) = \sum_k W_k(M).$$

Неизвестные дипольные моменты $\nu_k(P)$ на каждой поверхности дают нам решение задачи – потенциал $V(M)$. После определения $\nu_k(P)$ мы подставляем дипольные моменты в выражение для W_k:

$$W_k(M) = -\iint_{\Sigma_k} \frac{\partial}{\partial n_p}\left(\frac{1}{R_{MP}}\right)\nu_k(P)d\sigma_p.$$

Здесь P есть точка на поверхности Σ_k, M – точка наблюдения (точка трехмерной разностной сети, покрывающей объем ловушки),

$R_{MP} = \sqrt{(x_m - x_p)^2 + (y_m - y_p)^2 + (z_m - z_p)^2}$ представляет собой расстояние между точкой P и точкой M, n_p – нормаль к поверхности в точке P, а $\nu_k(P)$ является неизвестной поверхностной плотностью дипольного момента, которая различна на каждой поверхности.

Дипольные моменты $\nu_k(P)$ получаются из граничного условия для потенциала двойного слоя, когда мы помещаем точку M последовательно на все поверхности, $M = P_0^{(k)}$. Так мы получаем систему уравнений для

дипольных моментов $v_k(P)$ на каждой поверхности. $v_k(P)$ определяются граничными условиями, а именно значениями U_k на каждой поверхности и потенциалами, наведенными другими поверхностями. В каждом уравнении один из интегралов в сумме при $i = k$ и $P_0^i = P$ имеет сингулярность, что приводит к тому, что в численной схеме такие члены должны иметь специальную аппроксимацию.

Внешняя часть поверхности является отталкивающей, а внутренняя – притягивающей.

$$2\pi v_k\left(P_0^{(k)}\right) - \iint\limits_{\Sigma_k} \frac{\partial}{\partial n_p}\left(\frac{1}{R_{P_0^k,P}}\right) v_k(P)d\sigma_p = U_k + \sum_{\substack{i=1,\\ i\neq k}}^{K} \iint\limits_{\Sigma_i} \frac{\partial}{\partial n_p}\left(\frac{1}{R_{P_0^k,P^i}}\right) v_i(P^i)d\sigma_p^i.$$

В левой части мы оставляем интегрирование только по поверхности k (точка P), при этом точка наблюдения потенциала $P_0^{(k)}$ тоже находится на этой поверхности. В правой части уравнения находятся приложенный потенциал на поверхности k и сумма потенциалов от всех поверхностей Σ_i, которые индуцируют электрическое поле на данной поверхности. Идея итерационной процедуры состоит в фиксации дипольных моментов для всех поверхностей, кроме k. Правая часть уравнения известна на текущем итерационном шаге и может быть вычислена. По решениям для всех поверхностей на этом шаге можно вычислить новую правую часть. Система уравнений переписывается в форме

$$2\pi v_k\left(P_0^{(k)}\right) - \iint\limits_{\Sigma_k} \frac{\partial}{\partial n_p}\left(\frac{1}{R_{P_0^k,P}}\right) v_k(P)d\sigma_p = \Phi_k \qquad (12)$$

со следующей правой частью,

$$\Phi_k = U_k + \sum_{\substack{i=1,\\ i\neq k}}^{K} \iint\limits_{\Sigma_i} \frac{\partial}{\partial n_p}\left(\frac{1}{R_{P_0^k,P^i}}\right) v_i(P^i)d\sigma_p^i. \qquad (13)$$

7. Решение системы интегральных уравнений на гибридных вычислительных системах с CPU и GPU.

После написания квадратур для интегралов, полная система для набора поверхностей (12) может быть переписана в форме алгебраических уравнений.

Обозначим интегралы для поверхности n после дискретизации как

$$\sum_{i=0}^{N_1}\sum_{j=0}^{N_2} F_{kmij}^{(n)} u_{ij}^{(n)}$$

где F учитывает квадратурную формулу, N_1, N_2 – число сеточных точек по двум направлениям, $u_{ij}^{(n)}$ есть решение для n-ой поверхности.

Пусть s есть номер итерации. Тогда получаем следующую итерационную процедуру:

$$2\pi u_{km}^{n_0,s+1} - \left\{\sum_{i=0}^{N_1}\sum_{j=0}^{N_2} F_{kmij}^{(n_0,s+1)} u_{ij}^{(n_0,s+1)}\right\} = f_{km}^{n_0} + \sum_{\substack{n=1,\\ n\neq n_0}}^{N_n}\left\{\sum_{i=0}^{N_1}\sum_{j=0}^{N_2} F_{kmij}^{(n,s)} u_{ij}^{(n,s)}\right\}.$$

Первый член правой части представляет собой заданный поверхностный потенциал, а второй найден на предыдущем шаге и представляет собой влияние других поверхностей на распределение дипольного момента с номером n_0.

Каждое уравнение переписывается в форме алгебраической системы

$$\hat{A}Y = \Phi, \qquad (14)$$

где Y – линейный вектор решения, Φ – линейный вектор известной правой части, и Â есть матрица, включающая геометрические характеристики электродов. Для решения можно использовать соответствующие стандартные процедуры решения систем линейных алгебраических уравнений с плотной матрицей из библиотеки LAPACK.

Электрический потенциал, индуцированный на каждой поверхности электрода, может быть вычислен параллельно для всех поверхностей с номерами n_s = 1,2,…,m. На каждой итерации определяются поля $V_{\text{ind}}(n_s,n_d)$, индуцированные собственной поверхностью процесса на всех других поверхностях с номерами n_d. Редукционной операцией суммирования, реализуемой функцией MPI_Allreduce, во всех процессах определяется полное поле, индуцированное всеми электродами, и новая правая часть в уравнениях.

На новой итерации используется модифицированное граничное условие для каждой поверхности. В каждом MPI-процессе на графических процессорах решается алгебраическая система уравнений (14) со своей матрицей и правой частью. Схема параллельного алгоритма показана на рис. 2.

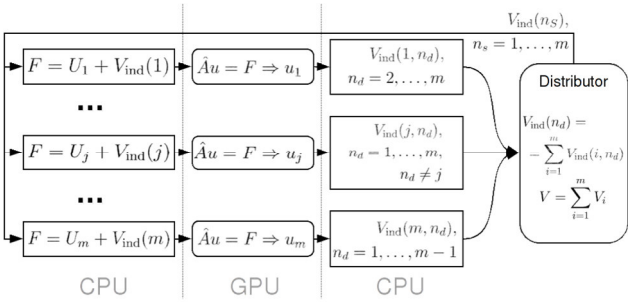


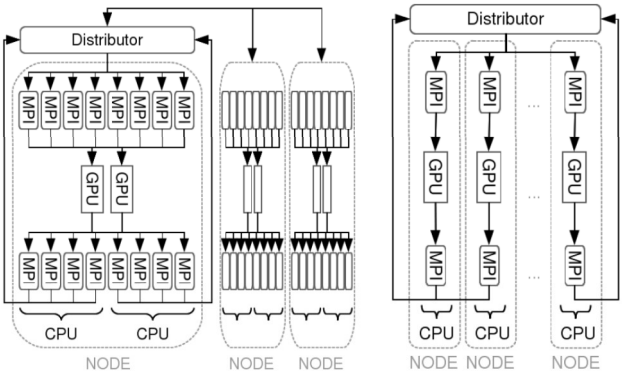
Рисунок 2. Схема параллельного алгоритма расчета удерживающего поля ловушки. В каждом параллельном процессе решение алгебраической системы уравнений проводится на GPU.

8. Решение алгебраических систем уравнений на GPU для каждой поверхности с помощью библиотеки CULA.

Решение алгебраической системы для каждой из поверхностей электрода выполняется на GPU с использованием библиотеки CULA v.R17. В рассматриваемой задаче параметры выбраны так, что алгебраическая система уравнений (14) имеет матрицу размера $N_1 \times N_2$, $N_1 = N_2$ = 35, 45, 55, 65, 75. Расчеты по решению задачи проводились на суперкомпьютере «Ломоносов». Для решения выбирались процессорные узлы, оборудованные двумя 4-ядерными CPU и двумя GPU Tesla X2070, по характеристикам аналогичными Tesla C2070. Исследовались различные стратегии распределения параллельных процессов по

узлам. Первая стратегия состояла в распределении 20 MPI-процессов по 1 процессу на вычислительное ядро процессора. Доступ к каждому GPU осуществлялся одновременно несколькими процессами. Вторая стратегия подразумевала, что каждый из 20 MPI-процессов запускается на своем вычислительном узле, при этом в вычислениях задействовалось только одно из четырех ядер процессора.

На рис. 3 показаны схемы обеих стратегий.



а) По четыре MPI-процесса работают с одним GPU

б) Каждый MPI-процесс запускается на своем узле

Рисунок 3. Схемы распределения задачи расчета удерживающего поля ловушки по узлам суперкомпьютера «Ломоносов».

Первая стратегия характеризуется тем, что несколько процессов конкурируют за ресурсы GPU. Каждый из ускорителей единовременно может выполнять лишь один CUDA-контекст, соответствующий одному MPI-процессу. Процедура решения СЛАУ в CULA не монолитна и разбивается на некоторое количество подзадач, что позволяет переключать контексты нескольких процессов по мере необходимости. Накладные расходы на такие переключения сокращаются при увеличении размеров обрабатываемых данных.

При использовании второй стратегии каждый процесс получает доступ к GPU сразу же, без ожидания, и выполняет подзадачи одного процесса до получения окончательного ответа.

Средние времена решения СЛАУ при использовании первого и второго вариантов и их отношение показаны в таблице 4. При увеличении числа параллельно обращающихся к одному GPU процессов разброс времени возвращения из процедуры решения также увеличивается. На рис. 4 показано время выполнения процедуры решения СЛАУ в зависимости от условного номера параллельного процесса при характерном числе точек в матрице N = 65. На рисунке также показано время решения при использовании лишь одной видеокарты GeForce GT 740M.

Таблица 4.

Сравнение среднего времени решения СЛАУ одним процессом при использовании первого и второго вариантов.

N	Вариант а) (с)	Вариант б) (с)	Отношение а) к б)
35	0.3082	0.0491	6.27
45	0.5082	0.0778	6.53
55	0.5855	0.1487	3.94
65	1.2202	0.3128	3.90
75	1.7293	0.5168	3.35

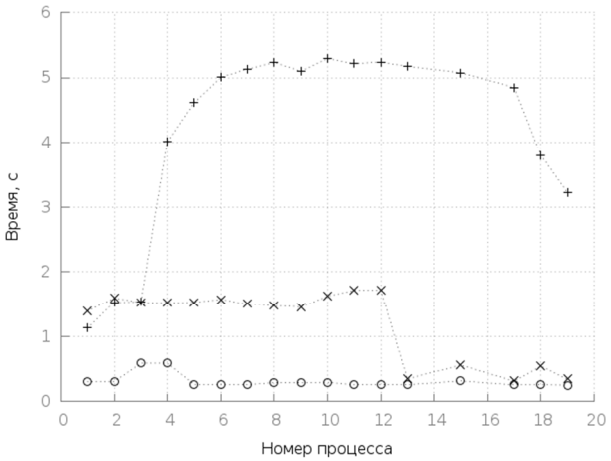


Рисунок 4. Время решения линейной системы уравнений при параллельном выполнении вычислений при использовании схем а) (○) и б) (×). Время вычислений показано в зависимости от условного номера параллельного процесса при числе точек в матрице системы уравнений N = 65. В каждом параллельном процессе на CPU используется GPU для решения алгебраической системы уравнений. Для сравнения показано также время решения при использовании лишь одной видеокарты GeForce GT 740M (+).

9. Заключение.

В работе представлены результаты реализации модели частиц в ячейке для моделирования работы масс-спектрометров на основе ионного циклотронного резонанса и преобразования Фурье на гибридных вычислительных системах с использованием GPU.

Использование графических ускорителей для выполнения дискретного преобразования Фурье и решения систем линейных алгебраических уравнений позволяет достигать хорошего времени решения возникающих подзадач. К ним относятся определение кулоновского взаимодействия ионов с помощью решения первой краевой задачи для уравнения Пуассона и параллельное вычисление полей от каждой поверхности электрода через решение алгебраических систем. Учёт особенностей архитектуры вычислительной системы при организации параллельных вычислений с использованием GPU позволяет избежать дополнительных накладных расходов на перегрузку контекстов, связанных с переключением потоков.

Представленные в работе результаты могут быть полезны при разработке параллельных программ для моделирования молекулярных структур в сфере разработок в области нанотехнологий в силу аналогич-

ных подходов к использованию исследованных процедур.

Работа выполнена при поддержке грантов РФФИ №13-01-12078 офи-м, №14-07-00654 и №14-07-00628.

This work was supported by the Russian Foundation for Basic Research, projects no. 13-01-12078 ofi_m, no. 14-07-00654 and no. 14-07-00628.

Список литературы:

1. Wilt N. The CUDA Handbook: A Comprehensive Guide to GPU Programming. Addison-Wesle, 2013. P. 528

2. Parallel Computing: From Multicores and GPU's to Petascale / Chapman B., Desprez F., Gerhard R., et al. IOS Press, 2010. Vol. 19. P. 739

3. Попов А.М. Вычислительные нанотехнологии: учебное пособие. М.: КНОРУС, 2014. 312 с.

4. Хокни Р., Иствуд Дж. Численное моделирование методом частиц: пер.с англ. М.: Мир, 1987. 640 с.

5. Nikolaev E. N. Heeren R. M. A. Popov A. M. Realistic modeling of ion cloud motion in a Fourier transform ion cyclotron resonance cell by use of a particle-in-cell approach // Rapid Communication in Mass Spectrometry. 2007. Vol. 21(22). P. 3527–3546.

6. Misharin A., Popov A. Parallel numerical code «parTfield» to simulate ion optical elements for any electrode geometry // Proc. 60th ASMS Conference on Mass Spectrometry and Allied Topics, Vancouver, Canada, May 20–24. 2012.

7. Самарский А. А., Николаев Е.С. Методы решения сеточных уравнений. М.: Наука, 1978. 590 с.

8. Malony A.D., et al. Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs // International Conference on Parallel Processing. ICPP 2011, Taipei, Taiwan, September 13–16. 2011. P. 176–185.