УДК 519.6

ОПТИМИЗАЦИЯ АЛГОРИТМОВ В ПРИКЛАДНОМ МЕТОДИЧЕСКОМ ТЕСТЕ MD ДЛЯ ЭФФЕКТИВНОГО ИСПОЛЬЗОВАНИЯ GPU

А. М. Ерофеев, М. В. Ветчинников
(ФГУП "РФЯЦ-ВНИИЭФ", г. Саров Нижегородской области)

Описываются алгоритмы тестовой программы молекулярной динамики (MD), позволившие осуществить полный перенос всех вычислений на GPU и тем самым избавить от постоянной необходимости обмена между устройствами информацией о частицах. В результате взаимодействие между GPU и CPU необходимо только для организации пересылок граничной информации между отдельными GPU с помощью MPI на CPU, а это значительно меньше, чем пересылки в первоначальном коде, о чем свидетельствуют тестовые замеры эффективности. На разных по размеру задачах на одном GPU получено ускорение алгоритма относительно первоначального от 8,7 до 12,5 раза. При задействовании двух GPU такое ускорение составило от 6,6 до 12,5 раза. Эффективность распараллеливания на двух GPU V100 составила 76,3—79,6 % на задачах с числом частиц от 4 млн, на двух GPU A100 — 77,3—81,8 % на задачах с числом частиц от 13,5 млн.

Ключевые слова: молекулярная динамика, эффективность, GPU, CUDA, алгоритмы.

Введение

Впервые метод молекулярной динамики был применен в середине XX века [1]. Сразу стало очевидно, что без мощных вычислительных машин моделировать ансамбли микрочастиц (атомов и молекул) для получения полезной информации с помощью метода невозможно. Поэтому развитие молекулярной динамики неразрывно связано с развитием вычислительных средств: увеличение вычислительных мощностей открывает новые возможности для моделирования разных классов задач, что, в свою очередь, значительно расширяет сферу применения молекулярной динамики.

Существует несколько известных пакетов программ по классическому молекулярно-динамическому моделированию, таких как DL-POLY [2], HOOMD [3], LAMMPS [4]. Основными целями разработчиков каждого из этих пакетов являлись не только написание алгоритмов, реализующих методы молекулярной динамики, но и их высокая эффективность для различного вида устройств. Из приведенных в [5] данных о вычислительной производительности разных пакетов программ на современных CPU и GPU видно, что, в частности, в пакете LAMMPS на GPU-устройствах достигаются значительные ускорения для различных потенциалов (табл. 1).

В РФЯЦ-ВНИИЭФ создаются вычислительные системы, которые позволяют решать поставленные задачи в более короткие сроки. Для того чтобы оценить эффективность работы вычислительных систем на реальных задачах, в

Таблица 1

Ускорения LAMMPS на GPU V100 для различных потенциалов [5]

Potential	Dual Cascade Lake 6240 (CPU-Only)	1× V100 SXM2 32 GB	2 × V100 SXM2 32 GB
LJ 2.5	1x	3x	5x
EAM	1x	2x	5x
Tersoff	1x	5x	9x

РФЯЦ-ВНИИЭФ существует набор прикладных методических тестов.

Одним из таких тестов является тест MD [6], реализующий метод молекулярной динамики, который применяется для решения широкого класса задач, связанных:

- с моделированием свойств твердых тел и жидкостей;
- развитием и поведением дефектов (в том числе трещин) внутри твердого тела, на его поверхности, в тонких пленках и микроэлектронике;
- изучением свойств поверхностей и межфазных границ;
- изучением свойств атомных и молекулярных кластеров и их взаимодействием с поверхностями;
- моделированием процессов в живых клетках и т. д.

Благодаря тому, что тест MD обладает хорошей масштабируемостью, большим количеством независимых арифметических вычислений и относительно небольшими количеством обменов и объемом передаваемых при этом данных, он позволяет высокоэффективно использовать графические ускорители.

Основы теста MD

Тест MD создан на базе программы молекулярной динамики [7] для исследования производительности и эффективности распараллеливания многопроцессорных вычислительных систем. Он представляет собой численное интегрирование уравнений движения Ньютона для систем частиц с заданным законом межчастичного взаимодействия:

$$\begin{aligned} &\frac{d\vec{r}_i}{dt} = \vec{v}_i; \\ &m_i \frac{d\vec{v}_i}{dt} = -\sum_{j \neq i} \nabla_r u\left(r_{ij}\right); \\ &\vec{F}_{ij} = -\nabla_r u\left(r_{ij}\right), \end{aligned} \tag{1}$$

где $\vec{r}_i$, $\vec{v}_i$, m_i — соответственно координаты, скорость и масса i-й частицы; $\nabla_r u\left(r_{ij}\right)$ — градиент межчастичного потенциала взаимодействия, зависящего от расстояния между частицами r_{ij}; $\vec{F}_{ij}$ — сила, действующая на частицу с номером i со стороны частицы с номером j; dt — временной интервал.

Законы взаимодействия пары частиц или многочастичные взаимодействия определены в виде потенциалов взаимодействия, по которым находятся компоненты сил для каждой конкретной частицы. По результирующей силе рассчитываются скорость частицы и ее смещение в пространстве за небольшой временной промежуток — шаг по времени dt.

Наиболее трудоемкая часть в организации молекулярно-динамических вычислений содержится в блоке вычисления сил взаимодействия между частицами, а именно при нахождении потенциалов взаимодействия. При этом критичны по времени как перебор частиц с многократным доступом к памяти, так и сами расчеты потенциалов по довольно сложным формулам. Если в системе из N взаимодействующих друг с другом частиц каждая частица испытывает на себе действие любой другой частицы, то необходимо рассчитать N^2 таких взаимодействий. Но, как правило, начиная с какого-то расстояния между частицами, потенциалы взаимодействия имеют значение, близкое к нулю. Поэтому в большинстве программ молекулярной динамики вводится эффективный радиус взаимодействия (его еще называют радиусом обрезания потенциала), дальше которого потенциал принимает нулевое значение.

Вычисление потенциала не производится, если частицы находятся на расстоянии, большем радиуса обрезания. Это первое допущение, позволяющее уменьшить количество расчетов потенциалов. Но при этом расстояние между частицами все же рассчитывается, а для этого необходимо извлечь из памяти данные о соседней частице.

Для уменьшения такого перебора существует несколько алгоритмов. Один из них — это создание так называемых списков Верле [8]: для каждой частицы создается список частиц, взаимодействующих с ней в течение определенного количества шагов. Данные списки требуют дополнительной памяти, и, кроме того, их необходимо периодически обновлять.

Другой алгоритм, который как раз реализован в тесте MD, основан на сеточном подходе. Область моделируемого пространства разбивается на параллелепипеды (ячейки) со стороной, равной или чуть большей радиуса обрезания. Поэтому потенциал взаимодействия ищется между частицами, находящимися в своей ячейке и в двадцати шести соседних ячейках. В этом случае количество операций поиска взаимодействующих частиц пропорционально n^2, где n — коли-

чество частиц в ячейке, что значительно меньше, чем общее число частиц N.

В памяти информация о частицах располагается последовательно в виде списка, когда есть адрес начала списка для ячейки и у каждой частицы (соответствующего ей элемента списка) есть указатель на следующую частицу в списке. Для ячейки также хранится количество частиц, которые она содержит, и адрес последней частицы для удобства изменения списка. Последняя частица в списке для ячейки ссылается на пустой адрес *null*.

При перелете частицы часто возникает ситуация, когда она покидает одну ячейку и переходит в другую. Здесь необходимо скорректировать информацию о принадлежности частиц ячейкам, изменив соответствующие элементы массивов указателей.

Правильно выбранная стратегия развития программы молекулярной динамики позволила создать комплекс программ MoDyS [9] для проведения численных экспериментов по получению различных характеристик исследуемых материалов [10] с высокой эффективностью распараллеливания.

Перенос программ на GPU

Первоначальная адаптация. Адаптация алгоритмов молекулярной динамики для графических ускорителей впервые была выполнена в 2008—2009 гг. [11]. В тот момент только началось использование данного типа архитектуры для моделирования физических процессов и задач. После анализа возможности портирования кода комплекса программ MoDyS на GPU-устройство первой для этой цели была выбрана самая трудоемкая часть кода — модуль расчета сил. Расчет сил занимает в среднем от 80 до 95 % общего времени выполнения счетного шага и зависит от типа потенциала взаимодействия, атомной структуры и радиуса обрезания в расчете потенциала взаимодействия.

Схема счетного шага с использованием GPU стала выглядеть, как показано на рис. 1. В начале шага с CPU, или хоста (host), на устройство (device) GPU копируется информация с координатами и типами частиц. На GPU осуществляется расчет потенциалов и по ним — сил взаимодействия. Расчет сил взаимодействия между частицами может выполняться независимо, поэтому данный процесс на GPU хорошо распа-

Рис. 1. Схема счетного шага с использованием GPU

раллеливается. Для расчета потенциалов было написано счетное ядро нахождения потенциала одной частицы. При этом число запускаемых параллельных процессов, или нитей, на GPU равняется количеству частиц, рассчитываемых на MPI-процессе. После расчета потенциалов компоненты сил отсылаются обратно на CPU. Следующее действие на шаге — это расчет на CPU скоростей частиц и по ним — новых координат. Применение сеточного подхода при реализации алгоритмов молекулярной динамики вызвало также необходимость расчета перехода частиц из одной ячейки в другую и между процессами.

Оказалось, что первоначальная адаптация к использованию вычислительных мощностей GPU имеет право на существование и показывает неплохие ускорения по отношению к CPU. При этом не потребовалось дорабатывать сервисную часть комплекса. Кроме того, у данной адаптации остался еще резерв ускорения.

Дело в том, что устройство GPU имеет смешанную архитектуру параллелизма: в нем не все нити соответствуют принципу SIMD (Single Instruction Multiple Data), физически одновременно выполняются лишь только нити в пределах одного так называемого *warp* (группа из 32 нитей). Нити различных *warp* могут быть на разных этапах выполнения кода программы. Такой метод обработки данных производитель GPU Nvidia называет SIMT (Single Instruction Multiple Threads).

Таким образом, очередная цель — получить ускорение кода, в том числе по возможности за счет SIMD-составляющей.

Для достижения данной цели прежде всего нужно иметь в виду, что у первоначальной адаптации к GPU есть большой недостаток — необходимость обмена информацией, находящейся на всем устройстве, т. е. пересылки координат частиц и сил (см. рис. 1). Так как по вычислительной мощности GPU превосходит CPU и на него передается много данных, то это становится фактором, уменьшающим эффективность использования GPU.

Одним из путей совершенствования кода молекулярной динамики становится полный перенос всех вычислений на GPU, который избавляет от постоянной необходимости обмена информацией о частицах между счетными устройствами. Тогда обмен между устройством и хостом необходим только для организации пересылок граничной информации между отдельными GPU с помощью MPI на CPU, а это значительно меньше, чем пересылки в первоначальном варианте кода с адаптацией к GPU (*старом* алгоритме).

Реализация *нового* алгоритма. Основная задача, которая решалась авторами, — избавиться от необходимости пересылки большого объема информации. Для этого все счетные модули, выполняемые на CPU, были последовательно адаптированы к устройству GPU.

В качестве ускорительных устройств используются графические ускорители фирмы Nvidia. Инструментом портирования кода на GPU является программно-аппаратная архитектура параллельных вычислений CUDA.

CUDA-программа может запускаться как на CPU, так и на GPU. Параллельный код можно перенести на GPU, где он будет выполняться множеством нитей (threads), менее мощных по сравнению с ядрами CPU, но число которых больше. Достигаемое на GPU ускорение обусловлено возможностью одновременного запуска на обработку большого количества информации: выигрыш относительно CPU получается как за счет большего количества нитей, так и за счет большей ширины векторного регистра.

Будем считать, что область решения задачи содержит *npart* частиц и разбита на *ncell* ячеек.

Проводится адаптация решения уравнений (1) для GPU, которое ранее выполнялось на CPU. Кроме решения самих уравнений, на GPU переводятся все вспомогательные модули, отвечающие за отслеживание информации о частицах в памяти, формирование списков частиц и т. п.

После того как силы рассчитаны, решение уравнений можно выполнить независимо, параллельно запустив *npart* нитей (по числу частиц). При этом для каждой частицы по ее новому местоположению можно узнать, осталась ли она в той же ячейке или мигрировала в другую. Если частица покинула ячейку, то возможны два варианта: она осталась на своем MPI-процессе или перешла на другой MPI-процесс.

Сначала разберем вариант, когда частица остается на своем MPI-процессе, но переходит из одной ячейки в другую. В таком случае программы, обрабатывающие ячейки, будут запускаться на GPU с числом нитей, равным числу ячеек — *ncell*. Каждая нить будет обрабатывать цепочку частиц, содержащихся в "ее" ячейке.

Вариант с параллельным запуском нитей по числу частиц проблематичен, так как возникают конфликты по указателям для ячеек и ча-

стиц. Однако таких конфликтов можно избежать с помощью атомарных операций `atomic`, которые блокируют нити при одновременном обращении к памяти, образуя при этом очередь.

Было реализовано два варианта кода: с числом нитей, равным числу частиц (с применением атомарных операций), и с числом нитей, равным числу ячеек (без атомарных операций). Вариант с обработкой по ячейкам оказался несколько быстрее варианта с применением `atomic`. На одном GPU прием с запуском нитей по числу ячеек отработан на модуле расчета переходов в другую ячейку. Дальше такой способ будет применяться и для других целей.

Теперь рассмотрим случай, когда частица переходит на другой MPI-процесс. Выполняемый в этом случае алгоритм схематично показан на рис. 2 и заключается в следующем.

На GPU формируется буфер частиц, покидающих связанный с устройством MPI-процесс. Процесс формирования буфера разделен на две части. Первая часть — это отбор частиц, покидающих MPI-процесс. Такой отбор запускается с числом нитей, равным числу ячеек, но фактически полезная информация поступает из ячеек, находящихся на границе с другими MPI-процессами. Вторая часть запускается с тем же числом нитей, равным числу ячеек, — она и формирует окончательно буфер, записывая информацию о пересылаемых частицах. На каждую частицу приходится восемь элементов: номер ячейки, куда частица переходит, номер материала частицы и по три координаты и компоненты скорости.

Рис. 2. Алгоритм пересылки частиц на другой MPI-процесс

Далее сформированный буфер копируется с устройства на хост, и на нем осуществляются обмены средствами MPI.

После того как осуществлена пересылка между MPI-процессами, хост копирует на устройство принятый буфер, который разбирается на GPU в два приема. Сначала выполняется копирование информации из буфера в массивы частиц, которое запускается с числом нитей, равным числу пришедших частиц. Затем запускаются нити на тех ячейках, в которых идет добавление прилетевших частиц, и формируются новые цепочки частиц для этих ячеек.

Следующий модуль, который нужно было адаптировать, — это модуль работы с *несчетными* ячейками (содержащими частицы, которые рассчитываются на соседнем процессоре). Здесь также возможны два варианта: 1) когда есть граница с периодическими условиями, но нет распараллеливания в этом направлении; 2) многопроцессорный случай. Во втором случае наличие периодических условий на границе приводит лишь к дополнительным пересылкам, так как границы, на которых заданы связанные условия периодичности, рассчитываются разными MPI-процессами.

В варианте 1 происходит дублирование информации из счетных ячеек на границе в несчетные ячейки на смежной по условиям периодичности границе (координаты частицы при этом изменяются на размер смещения по условиям периодичности). Это осуществляется запуском на GPU числа нитей, равного числу ячеек на границе.

В варианте 2 (многопроцессорный случай) сначала на GPU запускается программа формирования буфера с числом нитей, равным числу ячеек для пересылки. Так как количество частиц в ячейках известно, то буфер сразу форматируется для плотного заполнения. Для передачи необходимо всего пять элементов: номер ячейки, куда частица переходит, номер материала частицы и три координаты. Сформированный буфер копируется устройством на хост. Там осуществляется обмен информацией между процессами, и полученный в результате буфер хост возвращает на устройство. После этого копирование информации о частицах и ячейках за-

пускается на нитях, число которых равно числу несчетных ячеек.

В результате адаптации всех счетных и вспомогательных программ получен счетный код, полностью работающий на GPU и использующий CPU только для обмена информацией между процессами. Тем самым пропала необходимость копирования большого объема информации с GPU на CPU и обратно.

Модификация работы с памятью. При проведении тестирования нового алгоритма было отмечено, что первый счетный шаг проходит быстрее, чем последующие. Было проведено исследование данного явления и определено, что списочная структура хранения частиц в ячейках в процессе моделирования приводит к запутанности списков. После перехода частиц из ячейки в ячейку может оказаться, что частицы, расположенные рядом в списке, в физической памяти устройства находятся далеко друг от друга. На рис. 3 показано, как запутаны списки и как в процессе расчета сил взаимодействия постоянно идет обращение к памяти с далеко отстоящими друг от друга адресами, что сильно замедляет счет.

Для решения этой проблемы новый алгоритм был модифицирован: в конце каждого шага была введена сортировка частиц в ячейках на устройстве, учитывающая все переходы как между ячейками внутри GPU, так и между ячейками на разных GPU. Тем самым на следующем шаге для каждой ячейки достигнуто последовательное размещение всех частиц в физической памяти устройства (рис. 4).

Результаты тестирования и анализ эффективности новых алгоритмов

Для тестирования эффективности реализованных алгоритмов были выбраны следующие начальные данные: образец меди с гранецентрированной кубической решеткой, характерный шаг атомной решетки $a_0 = 3{,}615\,\text{Å}$; радиус обрезания 7,23 Å; атомная масса 63,546 а.е.м.; $T_0 = 300\,\text{K}$. Граничные условия — периодические. Рассматривались образцы разных размеров с количеством частиц от 442 тыс. до 210 млн. Используемый потенциал взаимодействия — парный потенциал Морзе. Количество счетных шагов в каждом расчете равно 100.

При верификации реализованных алгоритмов полученные по ним результаты расчетов полностью совпали с результатами расчетов на CPU.

Тестирование проводилось на сервере с двумя графическими ускорителями Nvidia Tesla V100 и двумя CPU Intel Skylake 6132.

Ускорение счета алгоритмов рассчитывалось по формуле

$$k = \frac{T_{old}}{T_{new}},$$

где T_{old}, T_{new} — время счета на GPU со старым и новым алгоритмом соответственно.

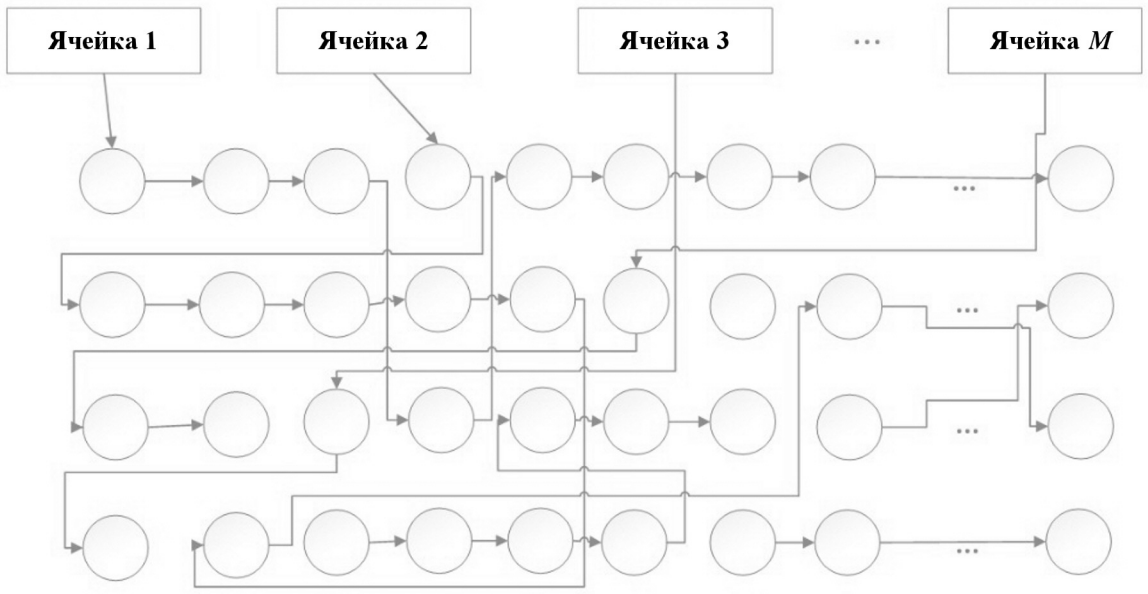

Рис. 3. Запутанность списков частиц в ячейках

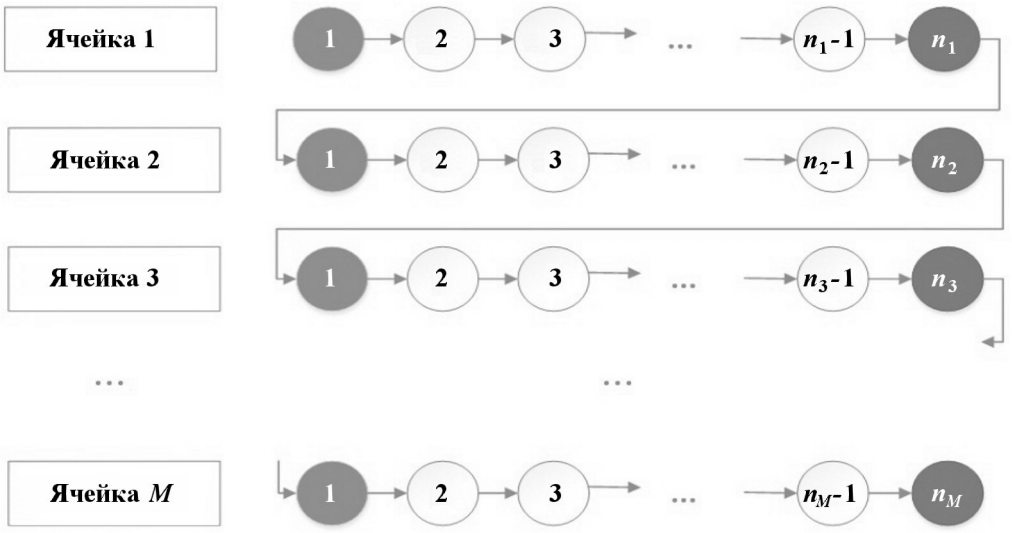

Рис. 4. Непрерывный список ячеек

Тест 1. Выполнена профилировка реализованного нового алгоритма на GPU и сравнение ее с данными замеров этапов выполнения старого алгоритма (табл. 2).

Как видно из таблицы, время копирования информации в старом алгоритме занимало около 50 % времени счетного шага. Причина — в передаче большого объема данных о всех рассчитываемых частицах.

Алгоритмы расчета уравнений движения и обновления связных списков в старом алгоритме выполнялись на одном ядре CPU, а в новом — на GPU. Из-за наличия большого количества логических операций, выполняющихся в определенной последовательности, получено небольшое ускорение относительно CPU (в 5 раз). Тем не менее перенос всех вычислений на GPU позволил полностью избавиться от копирования информации между устройствами и обеспечил значительный прирост общей производительности тестовой программы.

Тесты 2, 3. Было также проведено тестирование нового алгоритма на разных по объему задачах. Результаты сравнительных расчетов приведены в табл. 3. Видно, что на одном GPU получено ускорение нового алгоритма относительно старого от 4,7 до 7,5 раза.

В табл. 3 также приведены результаты исследования производительности нового алгоритма после модификации работы с памятью, которые свидетельствуют о том, что удалось ускорить код программы в 1,41—1,85 раза. Из табл. 3 также видно, что ускорение модифицированного кода относительно старого составляет 8,7—12,5 раза.

Тест 4. Проведено сравнительное тестирование нового и старого алгоритмов в режиме multi-GPU на двух устройствах. Результаты показаны в табл. 4. Видно, что в случае использования двух GPU ускорение нового алгоритма относительно старого составляет от 4,7 до 9,1 раза.

Таблица 2

Профилировка счетного шага при расчете на одном GPU (время — в секундах)

Алгоритм на GPU	Копирование информации на GPU	Расчет сил	Копирование информации на CPU	Расчет уравнений движения и обновление связных списков	Время счетного шага
Старый	$1{,}19 \cdot 10^{-1}$	$4{,}48 \cdot 10^{-2}$	$7{,}09 \cdot 10^{-2}$	$2{,}25 \cdot 10^{-1}$	0,47
Новый	Только на первом шаге	$4{,}51 \cdot 10^{-2}$	–	$4{,}4 \cdot 10^{-2}$	0,09

Таблица 3

Время (в секундах) и ускорение выполнения 100 шагов при расчете на одном GPU

Алгоритм GPU	Размер образца, a_0 (число частиц)						
	48 (442,368 тыс.)	86 (2,544 млн)	100 (4 млн)	150 (13,5 млн)	172 (20,353 млн)	200 (32 млн)	210 (37,044 млн)
Старый	4,281	28,87	33,87	110,69	162,40	252,02	289,9
Новый	0,635	3,87	6,76	20,75	34,14	47,84	61,49
Модифицированный	0,449	2,313	3,711	12,21	18,38	28,23	33,23
Ускорение:							
новый/старый	6,8	7,5	5,0	5,3	4,8	4,8	4,7
модифицированный/новый	1,41	1,67	1,82	1,7	1,86	1,7	1,85
общее	9,53	12,48	9,13	9,07	8,84	8,93	8,72

Таблица 4

Время (в секундах) и ускорение выполнения 100 шагов при расчете на двух GPU

Алгоритм GPU	Размер образца, a_0 (число частиц)						
	48 (442,368 тыс.)	86 (2,544 млн)	100 (4 млн)	150 (13,5 млн)	172 (20,353 млн)	200 (32 млн)	210 (37,044 млн)
Старый	2,62	13,09	29,55	61,29	90,37	138,19	159,97
Новый	0,51	1,99	3,25	11,4	18,34	29,47	32,78
Ускорение	5,13	6,58	9,1	5,37	4,93	4,7	4,88

Тест 5. Исследование эффективности реализованных алгоритмов проводилось на серверах с двумя GPU Nvidia Tesla V100 и четырьмя GPU Nvidia A100 40 GB SXM из состава программно-аппаратного полигона Национального центра физики и математики (НЦФМ) г. Сарова. Результаты приведены в табл. 5, 6, где $t1$, $t2$ — время 100 шагов на одном и двух GPU соответственно. Эффективность распараллеливания вычисляется по формуле

$$E2 = \frac{R2}{2 \cdot R1} \cdot 100\,\%,$$

где $R1$ — метрика производительности, равная числу обработанных частиц за секунду на одном GPU, $R2$ — на двух GPU.

Из табл. 5, 6 видно, что GPU V100 полностью загружается для задач с числом частиц от 4 млн, а полная загрузка GPU A100 начинается для задач с числом частиц от 13,5 млн. Эффективность распараллеливания на двух GPU V100 для задач с числом частиц от 4 млн составляет 76,3—79,6 %, на двух GPU A100 для задач с числом частиц от 13,5 млн — 77,3—81,8 %.

Таблица 5

Результаты тестирования на V100

Характеристика	Размер образца, a_0 (число частиц)						
	48 (442,368 тыс.)	86 (2,544 млн)	100 (4 млн)	150 (13,5 млн)	172 (20,353 млн)	200 (32 млн)	210 (37,044 млн)
$t1$, с	0,449	2,313	3,711	12,21	18,38	28,23	33,23
$t2$, с	0,398	1,586	2,366	7,67	11,67	18,49	21,31
$R1$, частиц/с	$9{,}85 \cdot 10^7$	$1{,}10 \cdot 10^8$	$1{,}08 \cdot 10^8$	$1{,}11 \cdot 10^8$	$1{,}11 \cdot 10^8$	$1{,}13 \cdot 10^8$	$1{,}11 \cdot 10^8$
$R2$, частиц/с	$1{,}11 \cdot 10^8$	$1{,}60 \cdot 10^8$	$1{,}69 \cdot 10^8$	$1{,}76 \cdot 10^8$	$1{,}74 \cdot 10^8$	$1{,}73 \cdot 10^8$	$1{,}74 \cdot 10^8$
$E2$, %	56,4	72,9	78,4	79,6	78,7	76,3	78,0

Таблица 6

Результаты тестирования на A100

Характеристика	Размер образца, a_0 (число частиц)						
	48 (442,368 тыс.)	86 (2,544 млн)	100 (4 млн)	150 (13,5 млн)	172 (20,353 млн)	200 (32 млн)	210 (37,044 млн)
$t1$, с	0,434	1,928	3,081	10,56	16,54	25,45	30,04
$t2$, с	0,435	1,533	2,179	6,6	10,11	16,46	18,72
$R1$, частиц/с	$1{,}02 \cdot 10^8$	$1{,}32 \cdot 10^8$	$1{,}3 \cdot 10^8$	$1{,}28 \cdot 10^8$	$1{,}23 \cdot 10^8$	$1{,}26 \cdot 10^8$	$1{,}23 \cdot 10^8$
$R2$, частиц/с	$1{,}02 \cdot 10^8$	$1{,}66 \cdot 10^8$	$1{,}84 \cdot 10^8$	$2{,}05 \cdot 10^8$	$2{,}01 \cdot 10^8$	$1{,}94 \cdot 10^8$	$1{,}98 \cdot 10^8$
$E2$, %	49,9	62,9	70,7	80,0	81,8	77,3	80,2

Заключение

В результате переноса всех счетных и вспомогательных модулей на GPU и оптимизации работы с памятью получена новая версия теста MD. Таким образом, при избавлении от копирования большого объема информации с GPU на CPU и обратно на одном GPU получено ускорение нового алгоритма относительно старого от 8,7 до 12,5 раза в зависимости от размеров задач. При использовании двух GPU получено ускорение нового алгоритма относительно старого от 6,6 до 12,5 раза. Эффективность распараллеливания на двух GPU V100 составила 76,3—79,6 % для задач с числом частиц от 4 млн, на двух GPU A100 — 77,3—81,8 % для задач с числом частиц от 13,5 млн. Версии программ сравнивались между собой, результаты тестовых расчетов по разным кодам совпадают.

Для повышения эффективности MPI-распараллеливания теста в режиме multi-GPU планируется реализовать MPI-обмены данными, расположенными в памяти GPU, т. е. не использовать CPU для обмена информацией.

Все наработки по ускорению кода, сделанные в тесте MD, будут перенесены в комплекс программ MoDyS, для которого предполагается дальнейшее развитие с целью моделирования задач не только классической молекулярной динамики, но и механики сплошной среды на основе разрабатываемого метода гамильтоновой динамики [12].

В данной статье описываются подходы, которые были применены для GPU, но их также можно использовать для оптимизации вычислений на CPU. Исследование таких подходов будет проведено в следующей работе.

Настоящее исследование выполнено в рамках научной программы НЦФМ г. Сарова (проект "Национальный центр исследования архитектур суперкомпьютеров").

Список литературы

1. *Alder B. J., Wainwright T. E.* Studies in Molecular Dynamics. I. General Method // J. Chem. Phys. 1957. Vol. 27. P. 1208.
2. Сайт DL_POLY. https://www.scd.stfc.ac.uk/Pages/DL_POLY.aspx.
   Sayt DL_POLY. https://www.scd.stfc.ac.uk/Pages/DL_POLY.aspx.
3. Сайт HOOMD. http://glotzerlab.engin.umich.edu/hoomd-blue.
   Sayt HOOMD. http://glotzerlab.engin.umich.edu/hoomd-blue.
4. Сайт LAMMPS. https://www.lammps.org.
   Sayt LAMMPS. https://www.lammps.org.
5. Официальный сайт Nvidia. https://developer.nvidia.com/hpc-application-performance.
   Ofitsialny sayt Nvidia. https://developer.nvidia.com/hpc-application-performance.
6. *Алексеев А. В., Беляев С. П., Бочков А. И., Быков А. Н., Ветчинников М. В., Залялов А. Н., Нуждин А. А., Огнев С. П., Самсонова Н. С., Сапронов И. Д., Чистякова И. Н., Шемякина Т. В., Шагалиев Р. М., Янилкин Ю. В.* Методические прикладные тесты РФЯЦ-ВНИИЭФ для численного исследования параметров высокопроизводительных вычислительных систем // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 2020. Вып. 2. С. 86—100.
   Alekseev A. V., Belyaev S. P., Bochkov A. I., Bykov A. N., Vetchinnikov M. V., Zalyalov A. N., Nuzhdin A. A., Ognev S. P., Samsonova N. S., Sapronov I. D., Chistyakova I. N., Shemyakina T. V., Shagaliev R. M., Yanil-

kin Yu. V. Metodicheskie prikladnye testy RFYaTs-VNIIEF dlya chislennogo issledovaniya parametrov vysokoproizvoditelnykh vychislitelnykh sistem // Voprosy atomnoy nauki i tekhniki. Ser. Matematicheskoe modelirovanie fizicheskikh protsessov. 2020. Vyp. 2. S. 86—100.

7. *Алейников А. Ю., Барабанов Р. А., Бутнев О. И., Быков А. Н., Веселов Р. А., Воронин Б. Л., Ганчук Н. С., Делов В. И., Ерофеев А. М., Пронин В. А., Рудько Н. М., Селезнев А. А., Скрыпник С. И.* Программа МДП-СОВЦ решения задач молекулярной динамики на параллельных ЭВМ с распределенной памятью // Там же. 2001. Вып. 1. С. 3—13.
Aleynikov A. Yu., Barabanov R. A., Butnev O. I., Bykov A. N., Veselov R. A., Voronin B. L., Ganchuk N. S., Delov V. I., Erofeev A. M., Pronin V. A., Rudko N. M., Seleznev A. A., Skrypnik S. I. Programma MDP-SOVTs resheniya zadach molekulyarnoy dinamiki na parallelnykh EVM s raspredelennoy pamyatyu // Tam zhe. 2001. Vyp. 1. S. 3—13.
8. *Verlet L.* Computer "experiments" on classical fluids. Thermodynamical properties of Lennard-Jones molecules // Phys. Rev. 1967. Vol. 159. P. 98—103.
9. *Анисимов А. Н., Грушин С. А., Воронин Б. Л., Копкин С. В., Ерофеев А. М., Дёмин Д. А., Дёмина М. А., Здорова М. В., Ветчинников М. В., Еричева Н. С., Коваленко Н. О., Крючков И. А., Кечин А. Г., Дегтярёв В. А., Урм В. Я.* Комплекс программ молекулярно-динамического моделирования (MoDyS). Свидетельство о государственной регистрации программы для ЭВМ № 2010614974 // Электронный бюллетень. 2010. http://www.fips.ru/wps/wcm/connect/content_ru/ru.
Anisimov A. N., Grushin S. A., Voronin B. L., Kopkin S. V., Erofeev A. M., Dyemin D. A., Dyemina M. A., Zdorova M. V., Vetchinnikov M. V., Ericheva N. S., Kovalenko N. O., Kryuchkov I. A., Kechin A. G., Degtyarev V. A., Urm V. Ya. Kompleks programm molekulyarno-dinamicheskogo modelirovaniya (MoDyS). Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2010614974 // Elektronny byulleten. 2010. http://www.fips.ru/wps/wcm/connect/content_ru/ru.
10. *Ветчинников М. В., Дёмина М. А., Анисимов А. Н., Грушин С. А., Кечин А. Г., Фомин В. П., Дегтярёв В. А.* Молекулярно-динамическое моделирование процесса самозатачивания ударника из W при проникании в мишень из Fe // Вопросы атомной науки и техники. Сер. Теоретическая и прикладная физика. 2017. Вып. 3. С. 23—34.
Vetchinnikov M. V., Dyemina M. A., Anisimov A. N., Grushin S. A., Kechin A. G., Fomin V. P., Degtyaryev V. A. Molekulyarno-dinamicheskoe modelirovanie protsessa samozatachivaniya udarnika iz W pri pronikanii v mishen iz Fe // Voprosy atomnoy nauki i tekhniki. Ser. Teoreticheskaya i prikladnaya fizika. 2017. Vyp. 3. S. 23—34.
11. *Воронин Б. Л., Ерофеев А. М., Копкин С. В., Крючков И. А., Рыбкин А. С., Степаненко С. А., Южаков В. В.* Применение графических арифметических ускорителей для расчета задач молекулярной динамики по программному комплексу МД // Там же. Сер. Математическое моделирование физических процессов. 2009. Вып. 2. С. 55—61.
Voronin B. L., Erofeev A. M., Kopkin S. V., Kryuchkov I. A., Rybkin A. S., Stepanenko S. A., Yuzhakov V. V. Primenenie graficheskikh arifmeticheskikh uskoriteley dlya rascheta zadach molekulyarnoy dinamiki po programmnomu kompleksu MD // Tam zhe. Ser. Matematicheskoe modelirovanie fizicheskikh protsessov. 2009. Vyp. 2. S. 55—61.
12. *Ветчинников М. В., Софронов В. Н., Дёмина М. А.* Использование методов гамильтоновой динамики в численных расчетах задач механики сплошной среды // Там же. 2020. Вып. 4. С. 5—21.
Vetchinnikov M. V., Sofronov V. N., Dyemina M. A. Ispolzovanie metodov gamiltonovoy dinamiki v chislennykh raschetakh zadach mekhaniki sploshnoy sredy // Tam zhe. 2020. Vyp. 4. S. 5—21.

Статья поступила в редакцию 01.02.22.