Эффективная реализация алгоритма расчета ближних невалентных взаимодействий на процессоре PowerXCell 8i*

Н. А. Алемасов, Э. С. Фомин
Институт цитологии и генетики СО РАН, Новосибирск, Россия,
e-mail: alemasov@bionet.nsc.ru, fomin@bionet.nsc.ru

Показано, что реализация алгоритма расчета ближних невалентных взаимодействий на процессоре IBM PowerXCell 8i при изучении движения молекул методом молекулярной динамики позволяет достичь ускорения его работы в 43 раза против AMD Athlon X2 5000+.

Ключевые слова: молекулярная динамика, ближние невалентные взаимодействия, параллельные вычисления, многопоточное программирование, Cell, IBM PowerXCell 8i, SPE-центричная модель, PPE-центричная модель, SIMD, NUMA.

Введение

Важнейшим средством теоретического исследования структуры, динамики и термодинамических свойств комплексов биомакромолекул является метод молекулярной динамики (МД), который представляет собой метод компьютерного моделирования, позволяющий в течение заданного периода времени проследить эволюцию системы взаимодействующих атомов с помощью численного интегрирования уравнений движения [1, 2]. Метод МД широко используется для решения задач исследования термостабильности белков, конформационных переходов, транспорта молекул, белкового фолдинга и пр. К сожалению, разброс масштабов различных физических явлений огромен — от 10^{-8} с (время движения доменов) до 1–10 с (время самоорганизации белков и нуклеиновых кислот) и находится вне возможностей современной МД. По этой причине в настоящее время изучены конформационные переходы макромолекул только для систем малых белков и пептидов [3, 4], а для систем размером более 1 млн атомов исследования проводятся в ограниченном временном интервале, не превышающем 50 нс [5]. Возможности современных программ МД не позволяют также проводить анализ сродства низкомолекулярных лигандов к активным центрам заданных ферментов, что актуально в современных технологиях разработки лекарств, и вынуждает использовать существенно менее точные эмпирические методы, основанные на функциях оценки [6], результаты которых подтверждаются экспериментом не более чем в 10 % случаев. Подходы, использующие МД для компьютерной фармакологии, активно разрабатываются [7], поскольку они включают существенно больше элементов “реальной физики” и, следовательно, потенциально более точны. Существенное, на порядки, увеличение скорости работы

*Работа выполнена при поддержке Междисциплинарных интеграционных проектов фундаментальных исследований СО РАН № 26, 113 и 119, проектов Программы фундаментальных исследований Президиума РАН А.II.6, Б.26, Госконтракта П857, Научной Школы НШ-2447.2008.4.

МД-программ не только кардинально ускоряет решение многих вышеупомянутых задач, но и открывает возможности решения задач нового класса, например, прямого расчета константы ассоциации белково-лигандных комплексов [8].

Программы МД включают множество алгоритмов, однако наиболее общим для всех расчетов и при этом представляющим собой "узкое место" с точки зрения затрат времени является расчет ближних невалентных (электростатических и ван-дер-ваальсовых) взаимодействий. Характерный пример — тесты производительности популярной программы GROMACS [9], в которых при моделировании небольшого, из 35 аминокислотных остатков, белка villin headpiece в ячейке, содержащей 3000 молекул воды, время выполнения данного алгоритма составляло 83 % общего времени выполнения программы, причем при условии, что подавляющая часть пар в расчетах была связана с молекулами воды, для которых программа GROMACS использует эффективные алгоритмы оптимизации [10].

Применение процессоров традиционной архитектуры x86 в вычислительных кластерах с производительностью, ограниченной 50 Гфлопс, не позволяет заметно ускорить МД-расчеты даже при использовании параллельных алгоритмов и всех ядер таких процессоров. В этой ситуации является естественным выполнение вычислений на высокопроизводительных кластерах, построенных на специализированных процессорах, таких как IBM Cell [10 – 15], и NVIDIA GP GPU [16, 17], и обладающих производительностью от 150 Гфлопс до 1 Тфлопс. Такие кластеры имеют наибольшую производительность. Например, в списке TOP500 суперкомпьютеров мира за ноябрь 2009 г. [18] на втором месте находится суперкомпьютер RoadRunner, в котором для интенсивных вычислений используются 12 960 процессоров PowerXCell 8i и 6480 процессоров AMD Opteron (дополнительно 432 процессора AMD Opteron выполняют системные функции).

1. Инструменты и методы

1.1. Процессор PowerXCell 8i

Архитектура Cell Broadband Engine (CBEA) была разработана при сотрудничестве компаний Sony, Toshiba и IBM. Процессор на ее основе вначале применялся для обработки мультимедиа и векторных данных, но впоследствии стал использоваться и для научных расчетов [10 – 15].

PowerXCell 8i [19] (далее Cell) представляет собой многоядерный процессор, содержащий девять ядер. Одно ядро — PPE (PowerPC Processor Element) — процессор архитектуры PowerPC, предназначенный для взаимодействия с операционной системой. Остальные восемь ядер SPE (Synergistic Processor Element) имеют отличную от PPE архитектуру и используются в качестве основных вычислительных устройств. В свою очередь, PPE содержит в себе блоки PPU (PowerPC Processor Unit) и PPSS (PowerPC Processor Storage Subsystem) — систему доступа к оперативной памяти. Ядро SPE состоит из блока SPU (Synergistic Processor Unit), локальной памяти LS (Local Storage) и контроллера потоков памяти MFC (Memory Flow Controller). SPE не имеют прямого доступа в основную память и могут напрямую работать только с локальной памятью LS. Память LS имеет размер 256 Кб и предназначена для хранения как кода, так и данных. Передача данных происходит по кольцевой шине EIB (Element Interconnect Bus), соединяющей все вычислительные элементы процессора и имеющей теоретическую пропускную способность более 200 Гб/с (около 25 Гб/с на каждый процессорный элемент).

Шина EIB состоит из четырех колец, два из которых работают по часовой, другие два — против часовой стрелки. На каждом кольце параллельно может осуществляться до трех передач. Для обеспечения большей пропускной способности шины в процессоре Cell имеется DMA-контроллер, позволяющий без участия PPE запрашивать данные из основной памяти и помещать их в LS, а также производить операции в обратном направлении, от SPE к PPE, и операции передачи данных между SPE. Контроллер позволяет делать DMA-передачи параллельно с вычислениями.

Каждый SPE имеет 128 128-битных регистров, что дает возможность компилятору перегруппировывать команды [20], а также широкий набор SIMD-инструкций, позволяющий эффективно работать с этими регистрами. В качестве типов данных с плавающей точкой могут быть использованы типы одинарной и двойной точности. Помимо возможности векторной обработки присутствует возможность выполнения до двух команд за один такт: один слот выдачи команды поддерживает операции с плавающей и фиксированной запятой, а другой обеспечивает загрузку (сохранение), операции перестановки байтов и перехода. По отдельности операции с фиксированной запятой занимают два такта, а команды одинарной точности с плавающей запятой и команды загрузки требуют шести тактов. SIMD-команды над двумя числами двойной точности с плавающей запятой выполняются минимум за семь тактов. Так как SPU не обладают механизмами предсказания ветвлений, разработчику необходимо "подсказывать" переходы. Команда "подсказки" перехода уведомляет SPU об адресе предстоящей команды перехода и его целевом адресе. После этого SPU (предполагая, что есть свободное место в локальной памяти) заранее выбирает по меньшей мере 17 команд по целевому адресу перехода. Для уменьшения числа ветвлений в коде можно также использовать поразрядную команду выбора с тремя источниками (оператор ? в C++).

Тестирование производительности параллельной реализации рассматриваемого в данной работе алгоритма происходило на сервере PeakCell S [21]. Сервер содержит два процессора PowerXCell 8i с частотой 3.2 ГГц, имеет 16 Гб оперативной памяти и работает под управлением ОС Fedora 7. Процессор PowerXCell 8i отличается от Cell/B.E. тем, что выполнен по более "тонкому" технологическому процессу и может работать с памятью типа DDR2. Оба процессора системы соединены шиной FlexIO, имеющей пропускную способность в 20 Гб/с. Память вычислительных узлов организована по схеме NUMA (Non-Uniform Memory Architecture), которая отличается от архитектуры SMP с общей системной шиной тем, что каждый узел имеет свою собственную память и шину для доступа к ней. Преимущество NUMA проявляется, когда процессоры многопроцессорной системы имеют настолько высокую производительность, что общая шина перестает справляться с запросами в общую память от каждого из узлов и становится "узким" местом.

1.2. Программные инструменты

В соответствии с аппаратной архитектурой программа для процессора Cell должна иметь два исходных файла: для PPE и SPE. При этом, чтобы создать программу, работающую только на PPE, достаточно воспользоваться компилятором, поддерживающим архитектуру PowerPC, например, соответствующую версию g++. Однако для создания эффективной программы, которая помимо PPE могла бы выполнять расчеты на SPE, необходимо использовать специальный компилятор spu-g++, который входит в IBM SDK for Multicore acceleration for Linux [22] и способен сгенерировать код для SPE.

Кроме того, поскольку ядра SPE не имеют прямого доступа в основную память, приходится явно вызывать функции чтения/записи данных. Одной из библиотек, содержащих функции передачи данных, является runtime-библиотека libspe2 из SDK. Помимо передач, требуется создать SPE-потоки, например, с помощью библиотеки Pthreads, загрузить специально подготовленный образ SPE-программы в память SPE и запустить ее на исполнение.

Важно отметить, что библиотека libspe2 предоставляет интерфейсы функций довольно низкого уровня, сравнимые с интерфейсом MPI [23], который широко используется для передачи данных между вычислительными узлами в системах с распределенной памятью. Однако с помощью libspe2 разработчик может организовать максимально оптимизированные вычисления, опираясь на архитектуру процессора и особенности алгоритма, который он реализует.

К явным ограничениям функций libspe2 относится необходимость выравнивания данных источника и приемника на границу в 16 Б при осуществлении передач данных. Согласно рекомендациям документации SDK, следует заметить, что для более быстрых передач необходимо выравнивать данные на границу 128 Б. Кроме того, размер одной передачи должен быть равен 1, 2, 4, 8, 16 Б или кратен 16 Б, но не должен превышать 16 Кб. Это ограничение в большинстве случаев требует уделять дополнительное внимание программированию коммуникаций в программе. Например, в данном случае можно использовать специальные атрибуты выравнивания для буферов передач, модифицировать структуры данных так, чтобы их размер был кратен 16 Б, а также применять несколько буферов по 16 Кб или особенные функции передач для обмена большими массивами данных.

Самая важная возможность, которую может предоставить процессор Cell, — это способность каждого SPE выполнять SIMD-инструкции. Существует пять разновидностей векторных библиотек: стандартная библиотека SIMDmath, библиотека MASS, inline-варианты этих двух библиотек и также библиотека MASSV. Согласно данным по исследованию производительности [24], наиболее быстрой является библиотека MASSV, оптимизированная для векторов большой длины.

1.3. Подходы к созданию программ для Cell

Для распределения задач и данных между ядрами процессора Cell обычно используются следующие модели [25] (рассмотрим классификацию относительно иерархии ядер Cell):

— PPE-центричная модель — основная модель, в которой главное приложение исполняется на PPE, а отдельные задачи исполнялись на SPE. При этом PPE подготавливает данные, координирует исполнение задач и ожидает получения результатов от SPE. Данная модель применяется при наличии последовательных данных и параллельных вычислений, при этом SPE можно использовать тремя основными способами: в качестве многостадийного конвейера (в этом случае каждый SPE выступает в роли определенной стадии конвейера), в качестве параллельных обработчиков, которые обрабатывают разные порции данных (в отличие от конвейера, данные поступают “сразу” на все SPE и обрабатываются параллельно), в качестве исполнителей сервисов (SPE выполняют различные виды вычислений, например, анализ данных, шифрование и кодирование мультимедиа-данных);

— SPE-центричная модель программирования предполагает, что бóльшая часть кода находится на SPE, а PPE в этом случае выступает только в роли диспетчера ресурсов для SPE. Когда активная задача завершается, SPE самостоятельно запрашивает новую задачу из основной памяти PPE.

Помимо классификации моделей по распределению задач и данных, возможна их классификация по способам использования ресурсов процессора:

— удаленный вызов процедур (Remote Procedure Call, RPC) подразумевает "ручное" разделение кода на PPE- и SPE-коды. Удаленный вызов реализуется с помощью заглушек на стороне PPE-кода, и соответствующая заглушка осуществляет передачу данных для исполнения функции на SPE и прием полученных результатов;

— ускорение расчетов — SPE-центричная модель, в которой PPE исполняет роль менеджера ресурсов. Задачи для SPE разделяются вручную или автоматически (компилятором) и обрабатываются параллельно;

— модель многоядерного процессора с разделяемой памятью. В ней операции работы с разделяемой памятью заменяются DMA-операциями между основной памятью и локальной памятью определенного SPE. DMA-операции используют эффективный адрес, общий для PPE и SPE. Такие замены могут производиться компилятором;

— наконец, модель исполнения асимметричных потоков, согласно которой "нити" могут создаваться как на PPE, так и на SPE. Эта модель эквивалентна обычной модели многопоточного программирования.

1.4. Исходный алгоритм

Данная работа была выполнена в рамках программного комплекса MOLKERN [26], предназначенного для решения задач, связанных с моделированием структуры и динамики комплексов биомакромолекул, состоящих из десятков и сотен тысяч атомов. В реализации комплекса MOLKERN используются оптимизированные алгоритмы с вычислительной сложностью не выше $O(N \log N)$. Библиотека MOLKERN написана на языке C++ с использованием библиотек STL, BOOST, CBLAS, FFTW, MPI и интерфейса OpenMP для процессоров архитектуры x86. Ядро программного комплекса реализует электростатические взаимодействия двумя способами: через стандартный кулоновский потенциал $1/r$ со сглаживаем разрывов потенциала на радиусе обрезания r_{cutoff} и пренебрежением взаимодействиями на дальнем расстоянии и через разделение кулоновского потенциала на ближнюю $erfc(\alpha r)/r$ и дальнюю $erf(\alpha r)/r$ части с суммированием дальних кулоновских взаимодействий с помощью метода P3M [27]. Ван-дер-ваальсовы взаимодействия представляются потенциалом 6–12 Леннарда-Джонса. Подход, основанный на разделении взаимодействия на ближнюю и дальнюю части, уменьшает вычислительную сложность с $O(N^2)$ до $O(N \log N)$. Все пары атомов, для которых расстояния между атомами меньше радиуса обрезания $r < r_{cutoff}$, помещаются в так называемые списки соседей.

Для реализации функции расчета парных взаимодействий характерно наличие цикла по множеству атомных пар, в котором для каждой пары вычисляются энергия взаимодействия и силы, действующие на оба атома пары. Поскольку для расчета взаимодействий между атомами в зависимости от типа атомов необходимо загружать совокупность тех или иных параметров и сохранять его результаты, то элементарный расчет пары атомов характеризуется множеством операций "чтение—модификация—запись" по ячейкам памяти в широком диапазоне адресов. Размер этого диапазона при

расчетах больших систем значителен и не зависит от способа хранения, а суммарный объем требуемых данных существенно превышает объем кэш-памяти компьютера. Поэтому при неоднократных обращениях к памяти последовательность хранения данных существенно влияет на скорость работы программы.

Алгоритмы расчета парных взаимодействий работают с хаотично распределенными данными, так как номера атомов, попадающих в пары, могут весьма различаться. В силу этого загрузка в кэш-память даже большого блока данных по адресам, близким к запрашиваемым, при расчете взаимодействий для некоторой пары атомов не гарантирует, что для следующей пары атомов уже загруженный блок данных будет содержать нужную информацию и перезагрузка кэша не потребуется. Последнее является причиной многочисленных промахов при работе с кэшем и соответствующего падения производительности приложения, а циклический характер расчета, при котором параметры загружаются для каждой пары (т. е. операции запросов к памяти постоянно чередуются с вычислительными операциями), может еще более увеличить количество промахов кэша.

В программном комплексе MOLKERN для уменьшения числа промахов кэша в последовательном алгоритме имеется механизм частичного упорядочения данных таким образом, чтобы максимизировать вероятность наличия близких номеров у атомов соседних пар. Такая частичная упорядоченность обусловлена процессом построения комплекса молекул, в ходе которого близким в пространстве атомам назначаются близкие значения идентификаторов, сохраняется при оптимизации геометрии, поскольку в этом процессе дальние смещения атомов маловероятны, и исчезает только в длительной молекулярной динамике.

В качестве дополнительного способа минимизации промахов кэша использован стандартный метод уменьшения размеров структур данных, необходимых для того или иного расчета. Удаление из структур данных лишней информации увеличивает диапазон загружаемых в блок элементов массивов, т. е. повышает вероятность нахождения нужных данных в кэше. Кроме указанных выше алгоритмических оптимизаций, в MOLKERN выполнено разделение процессов загрузки данных с вычислениями. Функция расчета парных взаимодействий $dU\_\_dX()$ разбита на три функции, которые для любой заданной совокупности пар выполняют следующие задачи (рис. 1):

— $put()$ — считывание всех данных из памяти в один блок;

— $calculate()$ — расчет энергий и сил для пар и сохранение их в этом же блоке;

— $get()$ — отправка рассчитанных данных из блока в память.

Каждый блок содержит данные, необходимые для расчета энергий и сил функцией $calculate()$. Эти данные включают информацию об атоме, для которого будет производиться расчет, а также параметры всех атомов, присутствующих в матрице соседей данного выделенного атома. Заполнение блока происходит в функции $put()$. После нее вызывается функция $calculate()$, которая, используя информацию, содержащуюся в переданном ей блоке, производит вычисления сил и энергии для пар, записанных в том же блоке. Функция $get()$ накапливает силы, действующие на тот или иной атом, и сохраняет результаты в памяти.

Таким образом, циклический расчет каждой пары в отдельности заменен блочным расчетом. Алгоритм, построенный на блочном расчете, естественно распараллеливается и может быть использован на любой системе с общей памятью. Кроме того, на стандартных процессорах блочная стратегия расчета по сравнению со стратегией расчета по отдельным парам атомов дает ускорение до 30 %. Блочный алгоритм оптимизирует

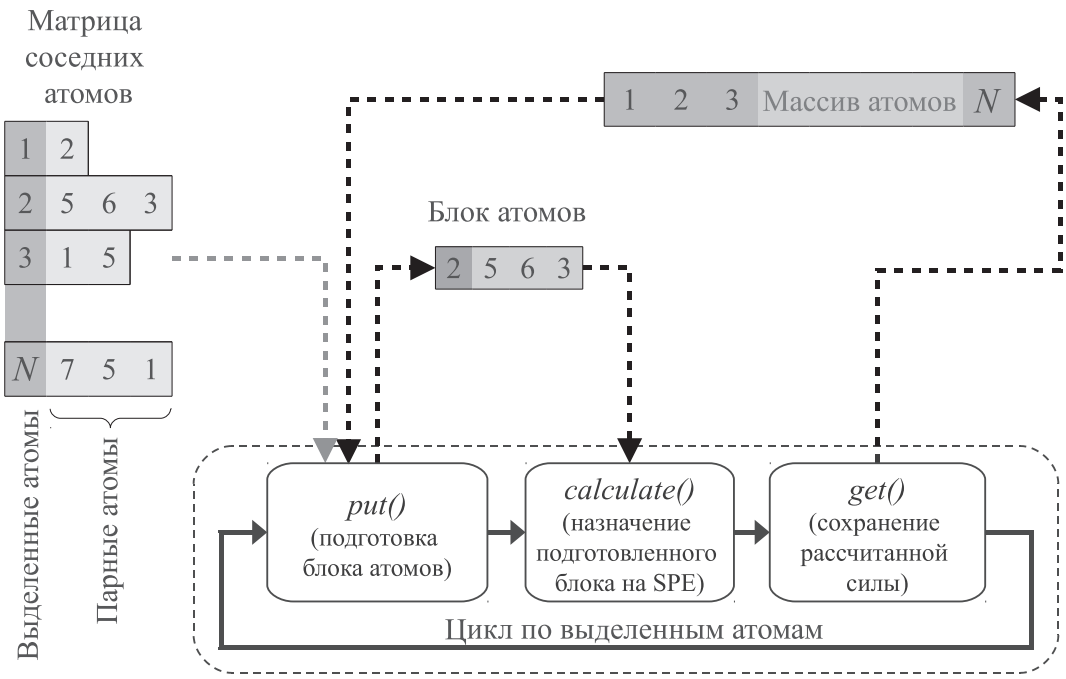

Рис. 1. Схема последовательного алгоритма расчета энергии и сил

обращения в память косвенным образом, т. е. максимизирует вероятность нахождения необходимых данных в кэше.

2. Параллельная реализация алгоритма

2.1. Архитектурные особенности параллельной реализации

В качестве инструментов для параллельной реализации были выбраны runtime-библиотека libspe2 и библиотека Pthreads для создания SPE-потоков. Все SPE-потоки создаются при запуске основной программы и уничтожаются при ее завершении. После запуска SPE-поток в цикле ожидает команду от PPE, выполняет ее и затем отправляет статус выполнения на PPE.

При переносе расчетов $dU\_\_dX()$ на SPE весь вычислительный код был переписан с учетом поддержки SIMD-инструкций. Использовались инструкции с элементами векторов одинарной точности, поскольку для молекулярной динамики потеря в точности не является существенной. В силу того что для рассматриваемой авторами реализации характерно большое количество векторов ограниченной длины, было решено остановиться на inline-варианте библиотеки векторных операций MASS.

Следует также отметить, что к организации векторной обработки координат атомов, хранящихся в структурах вида $\{x, y, z\}$, возможны два подхода: первый — преобразование структуры, содержащей координаты, т. е. $\{x, y, z\}$, в вектор вида $\{x, y, z, 0\}$ (последний элемент вектора не используется) и оперирование полученными векторами как векторами в геометрическом смысле слова, второй — преобразование набора из четырех структур вида $\{x, y, z\}$ в набор векторов вида $\{x1, x2, x3, x4\}$, $\{y1, y2, y3, y4\}$, $\{z1, z2, z3, z4\}$, каждый элемент которых содержит одну и ту же компоненту координаты всех четырех исходных структур. В первом случае можно выиграть в скорости

преобразования структуры в вектор, так как данные в памяти, содержащей $\{x, y, z\}$, будут идти в том же порядке (с поправкой на возможности компилятора, который может изменить этот порядок). Во втором случае выигрываем в вычислениях, так как получается не трехкратное, а четырехкратное ускорение. В настоящей работе был использован второй подход.

В процессе работы над реализацией для Cell был применен подход с пошаговым улучшением кода, минимизирующий количество необходимых изменений в исходной программе между этапами оптимизации. В первую очередь исходная программа была перенесена на Cell без изменений, т. е. фактически программа выполнялась только на PPE и не использовала SPE-ядра. В дальнейшем эта версия программы использовалась в качестве базы для оценки ускорения финальной программы. Следующим этапом было написание кода для SPE и перенос на них вычислений ближних невалентных взаимодействий. После этого в программу добавлялась возможность выполнения вычислений параллельно с передачей данных, а затем — векторизация расчетов с помощью SIMD-инструкций. Первая работающая версия программы была выполнена в рамках PPE-центричной модели. Дальнейшие улучшения связаны с переходом на SPE-центричную модель распределения задач между PPE и SPE.

2.2. PPE-центричная модель

Для сохранения внешнего интерфейса доступа к функциям библиотеки MOLKERN при реализации функции $dU\_\_dX()$ под Cell была использована модель удаленного вызова процедур, согласно которой функция должна иметь заглушку на PPE, а реальные расчеты производить на SPE. В этом случае структура функции $dU\_\_dX()$ вида $put() - calculate() - get()$ сохраняется (рис. 2) однако назначение функций $put()$, $calculate()$ и $get()$ становится другим. Функция $calculate()$ перестает производить непо-

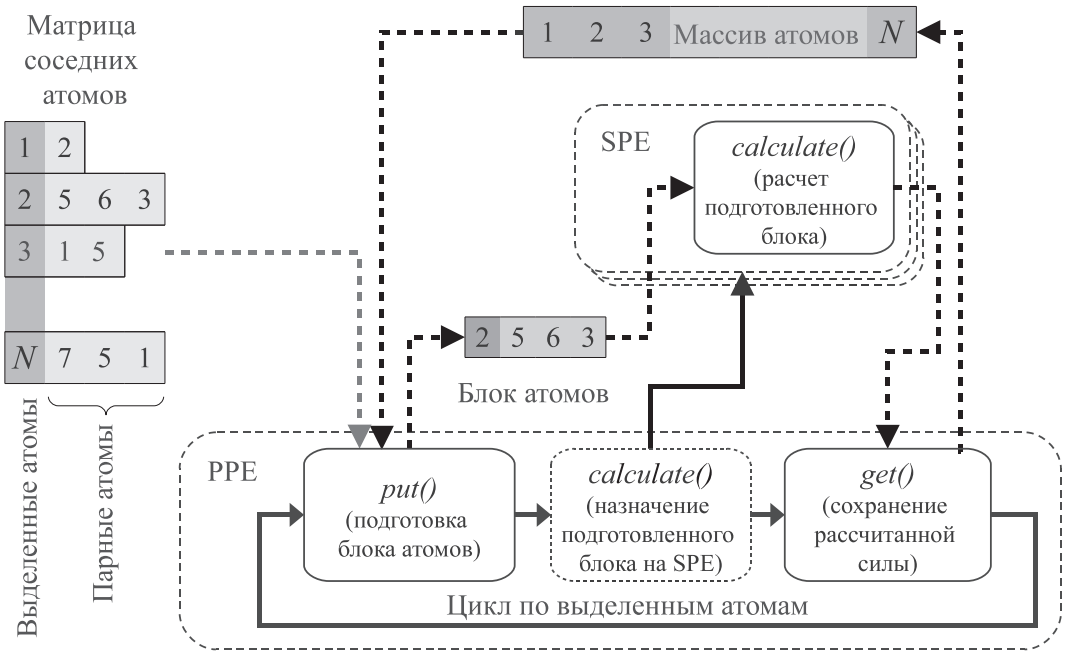

Рис. 2. Схема алгоритма расчета энергии и сил, выполненного в рамках PPE-центричной модели

средственные вычисления, вместо этого она занимается диспетчеризацией SPE-потоков. В зависимости от наличия не занятых расчетами SPE *calculate*() посылает свободному SPE команду загрузки пакета блоков данных, подготовленных функцией *put*().

Структура программы SPE сходна со структурой $dU\_\_dX()$ на PPE: функция SPE *put*() при получении команды от PPE производит загрузку пакета; *calculate*() рассчитывает силы для блоков в пакете; *get*() производит передачу рассчитанных сил и энергии в основную память, доступную PPE.

На PPE после *calculate*() вызывается функция *get*(), которая ожидает завершения расчетов какого-либо SPE. Как только один из SPE заканчивает свою работу и посылает PPE команду о ее завершении, *get*() осуществляет суммирование полученных от SPE частичных сил и энергии.

Таким образом, две из подфункций $dU\_\_dX()$ — *put*() и *get*() — по-прежнему выполняются на PPE, производя соответственно подготовку данных и сохранение результата, третья же — *calculate*() — производит на PPE диспетчеризацию SPE-потоков, раздавая SPE новые задачи; а на SPE рассчитывает энергии и силы для всех пар в блоке данных. Так как работу по подготовке данных для расчетов выполняет PPE, а SPE играют роль вычислителей, такую модель организации вычислений можно назвать PPE-центричной. Результаты первичных реализаций данной модели и более детальное представление особенностей ее реализации приведены в [28, 29].

2.3. SPE-центричная модель

SPE-центричная модель распределения задач в отличие от PPE-центричной модели не предполагает подготовки данных на стороне PPE. В этом случае SPE самостоятельно обращаются в основную память за данными, необходимыми для расчетов. Таким образом, скорость расчетов на SPE перестает зависеть от количества и производительности внешних потоков и может масштабироваться на любое число ядер SPE. Этот факт играет существенную роль в ситуации, когда скорость отработки данных превышает скорость их загрузки.

В SPE-центричной модели функция *put*() на PPE занимается подготовкой блоков, содержащих не атомы, а только указатели на них. Реальная загрузка атомов по полученным указателям выполняется на стороне SPE.

К сожалению, функцию *get*() по сохранению результатов в основной памяти нельзя перенести на SPE, поскольку в этом случае возможен конфликт между несколькими SPE при записи данных в одну и ту же ячейку памяти в ситуации, когда разные SPE рассчитывают различные частичные силы для одного и того же выделенного атома. Поэтому, как и в предыдущей модели, силы, полученные от SPE, суммирует PPE (рис. 3).

Кроме того, была изменена структура SPE-программы. Эффективность схемы *put*()—*calculate*()—*get*() на обычных процессорах связана с разделением по времени операций работы с памятью и вычислительных операций, так как эти процессоры имеют кэш-память. Ядра SPE кэш-памятью не обладают, однако имеют не менее эффективные средства по избавлению от задержек, связанных с доступом в память. С помощью DMA-контроллера SPE могут инициировать асинхронные запросы, что позволяет принимать или передавать данные в локальную или основную память параллельно с выполнением вычислений.

Таким образом, имея два буфера для приема параметров атомов, мы инициируем запросы на получение данных сразу в оба буфера. Дождавшись прихода данных в первый

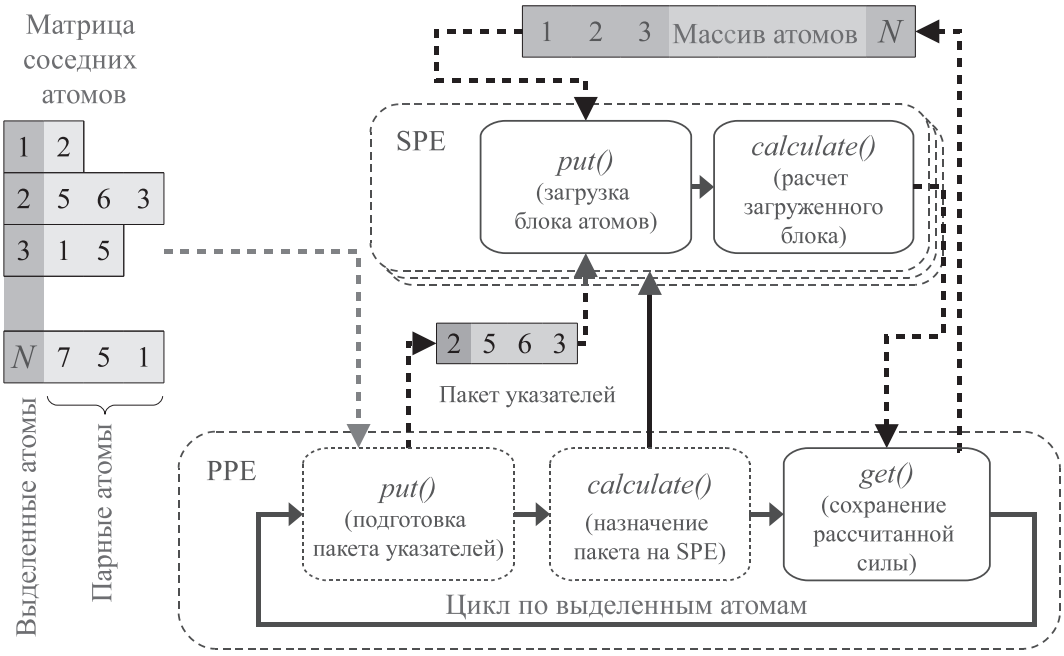

Рис. 3. Схема алгоритма SPE-центричного расчета энергии и сил

буфер, начинаем расчет данных, которые в нем находятся. В это время очередная порция данных, запрошенная во второй буфер, постепенно поступает в локальную память SPE, и когда расчеты с первым буфером закончатся, вся информация, запрошенная во второй буфер, уже будет загружена. Это позволит SPE без всякой задержки приступить к расчету данных, содержащихся во втором буфере. Но перед этим, поскольку первый буфер уже освободился, снова можно инициировать запрос получения в него уже третьей порции данных. Такой способ работы с основной памятью позволяет свести задержки при передаче данных к задержке выполнения только первого запроса — остальные запросы выполняются параллельно с вычислениями.

SPE в данной модели вынуждены запрашивать данные из основной памяти самостоятельно, имея только их адреса. Так как эти адреса указывают на непоследовательные участки памяти, нет надобности запрашивать за один раз большое количество атомов. Мы организовали получение данных порциями по четыре атома для того, чтобы разместить их параметры в элементах векторов — аргументах SIMD-инструкций. Было организовано также выполнение загрузки очередной порции из четырех атомов параллельно расчетам предыдущей порции.

3. Результаты и их обсуждение

На рис. 4 в логарифмическом масштабе представлена диаграмма ускорения исходной функции в зависимости от количества применяемых SPE. Для тестов были использованы два комплекса молекул: 1AIE и 1GC1 в водном окружении с числом атомов 6024 и 124 723; число пар атомов составляло более 1.15 и 25.0 млн соответственно.

Для PPE-центричной модели максимальное ускорение достигается при использовании восьми SPE и составляет 26 раз и 5 раз относительно одного PPE и процессора AMD Athlon X2, 2.6 ГГц в однопоточном режиме. Для увеличения скорости загрузки данных в блоки мы воспользовались тем, что PPE процессора PowerXCell 8i способен

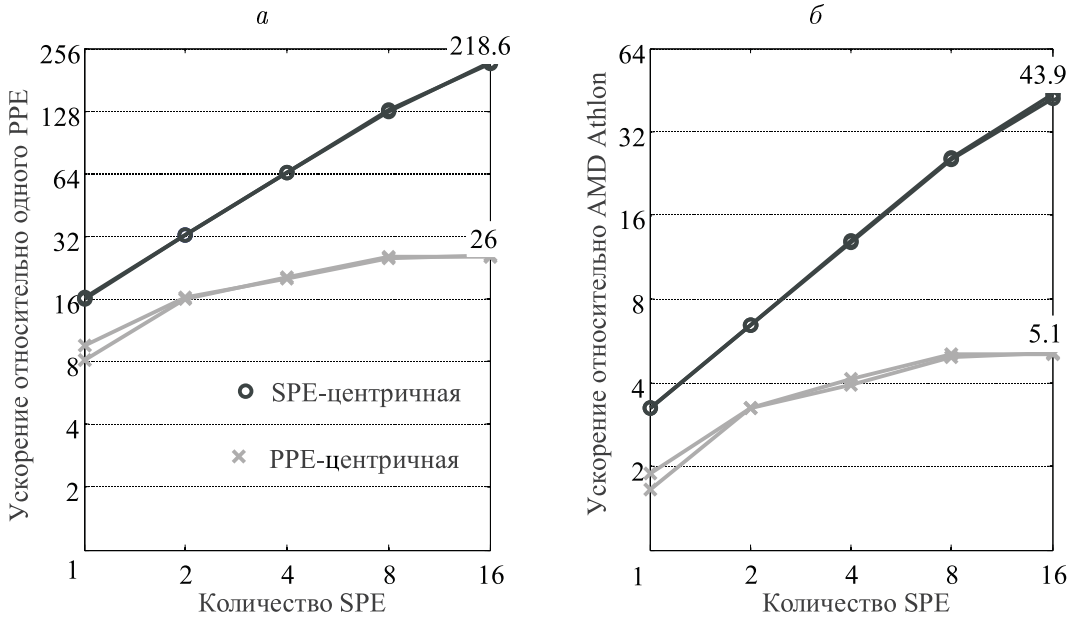

Рис. 4. Диаграмма зависимости ускорения функции $dU\_\_dX()$ от числа используемых SPE: *а* — относительно одного PPE, *б* — относительно однопоточной программы на процессоре AMD Athlon X2, 2.6 ГГц

обрабатывать два потока одновременно, т. е. обладает аналогом Hyper Threading. Был применен интерфейс OpenMP для исполнения *put*() и *get*() в двух потоках на каждом из двух процессоров сервера. Таким образом, для подготовки данных использовались четыре потока PPE. Но и такой прием при активации дополнительных восьми SPE не дал никакого выигрыша, позволяя SPE второго процессора Cell простаивать. Иными словами, при использовании PPE-центричной модели проблема низкой масштабируемости заключается в малой скорости подготовки данных для SPE на стороне PPE.

SPE-центричная модель дает существенно лучшие результаты. Максимальное ускорение достигается при использовании 16 SPE и составляет 218 раз и 43 раза относительно одного PPE и процессора AMD соответственно. Кроме того, в случае применения модели с "активными" SPE-потоками наблюдается практически линейное ускорение с увеличением числа SPE, занятых в расчетах, и только при переходе от загрузки одного процессора Cell к загрузке двух процессоров происходит заметное ухудшение масштабируемости. Из диаграммы производительности каждого SPE в зависимости от его номера и количества активных SPE (рис. 5) видно, что эффективность работы SPE при увеличении числа одновременно работающих SPE заметно снижается — в пределах 3 % для четырех SPE относительно двух SPE и для шести SPE относительно четырех SPE и далее на 5–12 % для каждого последующего набора из 8–16 SPE, что в целом уменьшает производительность каждого SPE при работе всех 16 SPE до 60 % относительно производительности одного SPE при использовании только двух SPE. Вероятнее всего, это связано с тем, что каждый SPE в представленной программе инициирует по четыре запроса в основную память (происходит прием информации о четырех атомах). Согласно характеристикам процессорной шины EIB, если все доступные SPE инициируют свои запросы одновременно, то в таком случае параллельно может быть принято до шести наборов по четыре атома (при использовании двух процессоров каждый процессор имеет четыре кольца EIB, каждое из которых, в свою очередь, дает возможность

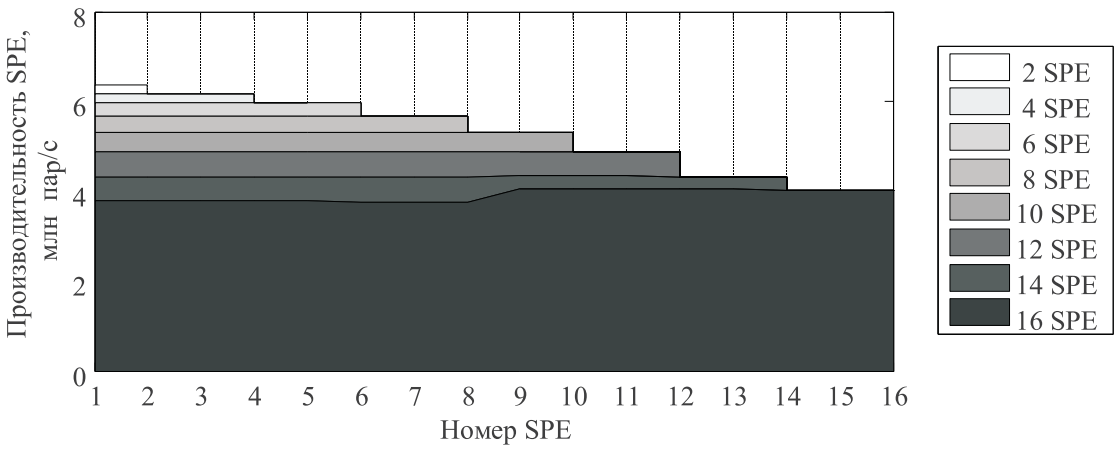

Рис. 5. Диаграмма зависимости количества пар, обработанных за одну секунду, от номера SPE, на котором происходили вычисления при запуске программы, с использованием от 2 до 16 SPE

работать параллельно трем передачам). Иными словами, при такой схеме организации данных уже шесть SPE заполняют шину своими запросами, а при активации большего количества SPE снижается эффективность работы каждого SPE и программы в целом.

Кроме того, из диаграммы на рис. 5 следует, что при использовании обоих вычислительных узлов (активируя 16 SPE) разница в производительности между SPE разных процессоров составляет до 6 % ("ступенька" на графике 16 SPE). Это, вероятно, связано с более высокими задержками при обращении к памяти через межпроцессорную шину тестового сервера. Так как мы не обращали внимание на равномерность распределения исходных данных по узлам сервера, то эти данные, очевидно, попали только на один из них и, следовательно, второму узлу неизбежно приходилось обращаться к данным, расположенным не в локальной относительно него памяти.

В таблице приведена информация для сравнения результатов реализаций разных программ молекулярной динамики для Cell. Первая программа представляет собой простое приложение, выполняющее 10 итераций молекулярной динамики над 2048 атомами [11]. Алгоритм не использовал оптимизаций, связанных с введением отсекающего дальние взаимодействия $cutoff$-радиуса, а также списки ближайших соседей, которые позволяют получить кэшированный доступ к памяти. Вычисления происходили с одинарной точностью. На SPE SIMD-векторы содержали соответствующие компоненты реальных векторов ($\{x, y, z, 0\}$, использовались только три компоненты SIMD-вектора). Данная программа при задействовании восьми SPE позволила получить 26-кратное ускорение относительно одного PPE и пятикратное относительно процессора AMD Opteron.

Программа GROMACS выполняла расчеты молекулярной динамики белка villin headpiece в водном окружении в течение 5000 временных шагов [9]; размер комплекса составил примерно 10 000 атомов. Программа была оптимизирована в части, ответственной за расчет невалентных взаимодействий между молекулами воды. Вычислительное ядро GROMACS для Cell использовало и $cutoff$-радиус, и список соседних атомов, и третий закон Ньютона, позволяющий суммировать частичные силы сразу для двух парных атомов, вдвое уменьшая время выполнения расчета. Как и в первой программе, SIMD-векторы составлялись из компонент реальных векторов ($\{x, y, z, 0\}$, последняя

Сравнение реализаций программ молекулярной динамики для процессора Cell

Программа	Ускорение (16 SPE) относительно PPE	Ускорение (16 SPE) относительно традиционного процессора		Характеристики традиционного процессора
		Однопоточная версия	Многопоточная версия	
Простое МД-ядро	28×(8 SPE)	5×(8 SPE)	—	AMD Opteron, 2.2 ГГц
GROMACS	—	15.5×(8 SPE)	—	Intel Xeon, 3.4 ГГц
CHARMM27	35×(8 SPE)	19×(8 SPE)	—	AMD Opteron, 2.2 ГГц
NAMD	61×	25.8	—	Intel Xeon, 3.0 ГГц
SPaSM	—	6.2×	—	Двухъядерный AMD Opteron, 1.8 ГГц
eHiNS	214×	—	70×	Двухъядерный Intel, 2.4 ГГц
MOLKERN	218×	43×	21×	AMD Athlon X2, 2.6 ГГц

компонента не использовалась). С помощью всех возможных оптимизаций программа на Cell, задействуя восемь SPE, дала ускорение в 15.5 раз относительно Intel Xeon.

Третья программа реализовывала силовое поле CHARMM27 для Cell [12]. Для теста производительности выполнялись 50 итераций молекулярной динамики над комплексом из трансмембранного белка Gramicidin A, встроенного в DMPC-липидный двойной слой с водой. Весь комплекс состоял из 29 000 атомов. Программа выполняла расчеты потенциала Леннарда-Джонса и электростатических взаимодействий на SPE. В вычислениях на SPE использовалась одинарная точность, на PPE — двойная точность. Алгоритм применял оптимизацию с $cutoff$-радиусом. Максимально достигнутое ускорение расчетов при использовании восьми SPE относительно одного PPE составило 35 раз, а относительно последовательной версии программы на AMD Opteron — 19 раз.

Программа NAMD предназначена для моделирования больших биомолекулярных систем. Версия NAMD для Cell отличается от исходной рядом упрощений [13], в числе которых — отсутствие списка соседних атомов, а также расчеты только невалентных взаимодействий (потенциала Леннарда-Джонса и электростатических взаимодействий методом PME). Использовалась как одинарная, так и двойная точность вычислений. Ускорение программы достигло 61 раз относительно одного PPE и 25.8 раз относительно последовательной версии на Intel Xeon.

Программа SPaSM выполняла расчеты потенциала Леннарда-Джонса для исследования нестабильности жидкостей и была реализована для суперкомпьютера Road-Runner [14]. В каждом вычислительном узле кластера межатомные взаимодействия рассчитывались на четырех процессорах PowerXCell 8i, в то время как два двухъядерных AMD Opteron применялись для межузловых коммуникаций. Вычисления производились над числами с двойной точностью. Алгоритм использовал $cutoff$-радиус, списки соседних атомов (неявные), третий закон Ньютона. SPaSM на Cell получила ускорение в 6.2 раза относительно последовательного кода на AMD Opteron.

Однако наиболее близкой по производительности к результатам настоящей работы является программа eHiTS. Небольшая ее часть, реализованная для Cell [15], выполняла расчеты 6–12 потенциала Леннарда-Джонса. В этой реализации не использовался радиус отсечения дальних взаимодействий. Финальное ускорение составило 214 раз относительно одного PPE и около 70 раз относительно двухъядерного процессора Intel Xeon. Данная программа реализовывала более простой вид расчетов, чем в MOLKERN, что, вероятно, и позволило получить представленные результаты.

Таким образом, расчет ближних взаимодействий был перенесен на процессор Cell. Основная часть программы выполнялась на PPE, управляя потоками SPE и сохраняя полученные от них результаты, в то время как SPE выполняли все главные вычисления, связанные с расчетом сил и энергии комплекса молекул. Были реализованы две модели организации вычислений: PPE-центричная и SPE-центричная. Наиболее эффективной оказалась вторая, так как в ней SPE-потоки не зависели от скорости работы PPE-потоков, которые в PPE-центричной модели, кроме сохранения результатов, также были заняты подготовкой исходных данных. SPE-центричная модель показала практически линейную масштабируемость в зависимости от числа задействованных SPE, что указывает на высокую оптимизацию параллельной реализации. При использовании двух процессоров Cell масштабируемость ухудшается, что, видимо, связано с особенностями алгоритма доступа к памяти, а также с NUMA-организацией оперативной памяти тестового сервера. В результате переноса вычислений функции расчета ближних невалентных взаимодействий на процессор Cell было достигнуто 43-кратное ускорение вычислений относительно системы на базе процессора AMD Athlon X2 и 218-кратное — относительно программы, запущенной на одном ядре PPE.

Авторы благодарят компанию T-platforms (http://www.t-platforms.ru) за поддержку данной работы и предоставление доступа к серверу PeakCell S.

Список литературы

[1] Alder B.J., Wainwright T.E. Phase transition for a hard sphere system // J. Chem. Phys. 1957. Vol. 27. P. 1208–1209.

[2] Gibson J.B., Goland A.N., Milgram M., Vineyard G.H. Dynamics of radiation damage // Phys. Rev. 1960. Vol. 120. P. 1229–1253.

[3] Zhou R., Berne B.J. Can a continuum solvent model reproduce the free energy landscape of α β-hairpin folding in water? // Proc. Natl. Acad. Sci. 2002. Vol. 99, No. 20. P. 12777–12782.

[4] Nymeyer H., Garcia A.E. Simulation of the folding equilibrium of α-helical peptides: A comparison of the generalized Born approximation with explicit solvent // Ibid. 2003. Vol. 100, No. 24. P. 13934–13939.

[5] Freddolino P.L., Arkhipov A.S., Larson S.B. et al. Molecular dynamics simulations of the complete satellite tobacco mosaic virus // Structure. 2006. Vol. 14. P. 437–449.

[6] Rester U. From virtuality to reality — virtual screening in lead discovery and lead optimization: A medicinal chemistry perspective // Curr. Opin. Drug Discov. Devel. 2008. Vol. 11, No. 4. P. 559-–568.

[7] Taufer M., Crowley M., Price D. et al. Study of a highly accurate and fast protein-ligand docking algorithm based upon molecular dynamics // Proc. of the 3rd IEEE Intern.

Workshop on High Performance Comput. Biology (HiCOMB '04). Santa Fe. NM. USA, 2004. P. 1627–1641.

[8] Теплухин А.В. Мультипроцессорный расчет констант ассоциации низкомолекулярных лигандов в водном растворе методом Монте-Карло // Параллельные вычислительные технологии (ПаВТ'2009): Тр. междунар. науч. конф. Челябинск: Изд-во ЮУрГУ, 2009. С. 724–731.

[9] Berendsen H.J.C., Postma J.P.M., van Gunsteren W.F., Hermans J. Interaction models for water in relation to protein hydration // Intermolecular Forces / Ed. B.D. Pullman. Holland, Dordrecht: Reidel Publ. Comp., 1981. P. 331–342.

[10] Olivier S., Prins J., Derby J. Porting the GROMACS molecular dynamics code to the cell processor // Proc. of the 21st Intern. Parallel and Distributed Proc. Symp. (IPDPS 2007). Long Beach, California, USA, 2007. P. 1–8.

[11] Meredith J.S., Alam S.R., Vetter J.S. Analysis of a computational biology simulation technique on emerging processing architectures // Proc. of the 21st Intern. Parallel and Distributed Proc. Symp. (IPDPS 2007). Long Beach, California, USA, 2007.

[12] Fabritiis G.D. Performance of the Cell processor for biomolecular simulations // Comput. Phys. Communicat. 2007. Vol. 176, No. 11-12. P. 660–664.

[13] Shi G., Kindratenko V. Implementation of NAMD molecular dynamics non-bonded force-field on the Cell Broadband Engine processor // Proc. of the 9th IEEE Intern. Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2008). Miami, Florida, USA, 2008. P. 1–8.

[14] Swaminarayan S., Kadau K., Germann T.C., Fossum G.C. 369 Tflop/s molecular dynamics simulations on the Roadrunner general-purpose heterogeneous supercomputer // Proc. of the 2008 ACM/IEEE Conf. on Supercomputing. Austin, Texas, USA, 2008. P. 1–10.

[15] Using Cell Broadband Engine Technology to Improve Molecular Modeling Applications / SimBioSys: White Papers. 2009. URL: http://www.simbiosys.ca/

[16] Anderson J.A., Lorenz C.D., Travesset A. General purpose molecular dynamics simulations fully implemented on graphics processing units // J. Comput. Sci. 2008. Vol. 227. P. 5342.

[17] Liua W., Schmidt B., Vossa G., Muller-Wittig W. Accelerating molecular dynamics simulations using graphics processing units with CUDA // Comput. Phys. Communicat. 2008. Vol. 179, No. 9. P. 634–641.

[18] TOP500 List — November 2009 (1-100) / TOP500 Supercomputing Sites. 2009. URL: http://www.top500.org/

[19] IBM PowerXCell 8i processor datasheet / IBM: Resources. 2009. URL: http://www-03.ibm.com/

[20] Kahle J.A., Day M.N., Hofstee H.P. et al. Introduction to the Cell multiprocessor // IBM J. Res. and Development. 2005. Vol. 49, No. 4-5. P. 589–604.

[21] Сервер PeakCell S / Т-Платформы. 2009. URL: http://www.t-platforms.ru/

[22] Programmer's Guide to the IBM SDK for Multicore Acceleration v3.0 / IBM. 2009. URL: http://www.ibm.com/

[23] MPI: The Message Passing Interface / Official site. URL: http://www.mpi-forum.org/

[24] Performance Information for the MASS Libraries for Cell/B.E. SPU / IBM. 2009. URL: http://www.ibm.com/

[25] Cell/B.E. Programming Tutorial v3.0 / IBM. 2009. URL: http://www.ibm.com/

[26] Фомин Э.С., Алемасов Н.А., Чирцов А.С., Фомин А.Э. Библиотека программных компонент MOLKERN для построения программ молекулярного моделирования // Биофизика. 2006. Т. 51, № 7. С. 110–113.

[27] Hockney R., Eastwood J. Computer simulation using particles. N.Y.: McGraw-Hill, 1981. 540 p.

[28] Фомин Э.С., Алемасов Н.А. Оптимизация вычислительного ядра библиотеки молекулярного моделирования MOLKERN под архитектуру Cell // Параллельные вычислительные технологии (ПаВТ'2009): Тр. междунар. науч. конф. Челябинск: Изд-во ЮУрГУ, 2009. С. 772–777.

[29] Fomin E., Alemasov N. Implementation of non-bonded interaction algorithm for the Cell architecture // LNCS. Parallel Computing Technologies 2009 / Ed. V. Malyshkin. Berlin, Heidelberg: Springer-Verlag, 2009. Vol. 5698. P. 399–405.

Поступила в редакцию 4 декабря 2009 г.,
с доработки — 16 марта 2010 г.