Применение ПЛИС для расчета деполимеризации микротрубочки методом броуновской динамики

[1,3] *Ю.А.Румянцев <yarumyantsev@gmail.com>*
[2] *П.Н. Захаров <pavel.n.zaharov@gmail.com>*
[1] *Н.А. Абрашитова <natascha.abraschitowa@gmail.com >*
[1] *А.В. Шматок <papercompute@gmail.com >*
[3] *В.О. Рыжих <vo.ryzhikh@mail.ru>*
[2,3,4] *Н.Б.Гудимчук <gudimchuk@phys.msu.ru>*
[2,3,4] *Ф.И.Атауллаханов <ataullakhanov.fazly@gmail.com>*
[1] *НПО РОСТА,*
123103, Россия, Москва, ул. Живописная, д. 3 к. 1
[2] *Центр теоретических проблем физико-химической фармакологии РАН,*
119991, Россия, Москва, ул. Косыгина 4
[3] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1.
[4] *Федеральный научно-клинический центр детской гематологии, онкологии и иммунологии имени Дмитрия Рогачева,*
117997, Россия, Москва ГСП-7, ул. Саморы Машела, д. 1

Аннотация. В данной работе рассмотрена аппаратная реализация расчета деполимеризации белковой микротрубочки методом броуновской динамики на кристалле программируемой логической интегральной схемы (ПЛИС) Xilinx Virtex-7 с использованием высокоуровневого транслятора с языка Си Vivado HLS. Реализация на ПЛИС сравнивается с параллельными реализациями этого же алгоритма на многоядерном процессоре Intel Xeon и графическом процессоре Nvidia K40 по критериям производительности и энергоэффективности. Алгоритм работает на броуновских временах и поэтому требует большого количества нормально распределенных случайных чисел. Оригинальный последовательный код был оптимизирован под многоядерную архитектуру с помощью OpenMP, для графического процессора - с помощью OpenCL, а реализация на ПЛИС была получена посредством высокоуровневого транслятора Vivado HLS. В работе показано, что реализация на ПЛИС быстрее CPU в 17 раз и быстрее GPU в 11 раз. Что касается энергоэффективности (производительности на ватт), ПЛИС была лучше CPU в 227 раз и лучше GPU в 75 раз. Ускоренное на ПЛИС приложение было разработано с помощью SDK, включающего готовый проект ПЛИС, имеющий PCI Express интерфейс для связи

с хост-компьютером, и софтверные библиотеки для общения хост-приложения с ПЛИС ускорителем. От конечного разработчика было необходимо только разработать вычислительное ядро алгоритма на языке Си в среде Vivado HLS, и не требовалось специальных навыков ПЛИС разработки.

Ключевые слова: Высокопроизводительные вычисления; ПЛИС; микротрубочки; высокоуровневый синтез; броуновская динамика

DOI: 10.15514/ISPRAS-2016-28(3)-15

Для цитирования: Румянцев Ю.А., Захаров П.Н., Абрашитова Н.А., Шматок А.В., Рыжих В.О., Гудимчук Н.Б., Атауллаханов Ф.И. Применение ПЛИС для расчета деполимеризации микротрубочки методом броуновской динамики. Труды ИСП РАН, том 28, вып. 3, 2016 г., стр. 241-266. DOI: 10.15514/ISPRAS-2016-28(3)-15.

1. Введение

Высокопроизводительные вычисления проводят на процессорах (CPU), объединенных в кластеры и/или имеющих аппаратные ускорители – графические процессоры на видеокартах (GPU) или программируемые логические интегральные схемы (ПЛИС) [1]. Современный процессор сам по себе является отличной платформой для высокопроизводительных вычислений. К достоинствам CPU можно отнести многоядерную архитектуру с общей когерентной кэш-памятью, поддержку векторных инструкций, высокую частоту, а также огромный набор программных средств, компиляторов и библиотек, обеспечивающий высокую гибкость программирования. Высокая производительность платформы GPU основывается на возможности запустить тысячи параллельных вычислительных потоков на независимых аппаратных ядрах. Для GPU доступны хорошо зарекомендовавшие себя средства разработки (CUDA, OpenCL), снижающие порог использования GPU платформы для прикладных вычислительных задач. Несмотря на это, в последнее десятилетие ПЛИС все чаще стали использоваться в качестве платформы для ускорения задач, в том числе использующих вещественные вычисления [2]. ПЛИС обладают уникальным свойством, резко отличающим их от CPU и GPU, а именно возможностью построить конвейерную аппаратную схему под конкретный вычислительный алгоритм. Поэтому, несмотря на значительно меньшую тактовую частоту, на которой работают ПЛИС (по сравнению с CPU и GPU), на некоторых алгоритмах на ПЛИС удается добиться большей производительности [3]–[5]. С другой стороны, меньшая частота работы означает меньшее энергопотребление, и ПЛИС практически всегда более эффективны, чем CPU и GPU, если использовать метрику «производительность на ватт» [5].

Одним из классических приложений, требующих высокопроизводительных вычислений является метод молекулярной динамики, использующийся для

расчета движения систем атомов или молекул. В рамках этого метода взаимодействия между атомами и молекулами описываются в рамках законов Ньютоновской механики с помощью потенциалов взаимодействия. Расчет сил взаимодействия проводится итеративно и представляет существенную вычислительную сложность, учитывая большое количество атомов/молекул в системе и большое количество расчетных итераций. Ускорению расчетов молекулярной динамики было уделено много внимания в литературе в различных системах: суперкомпьютерах [6], кластерах [7], специализированных под молекулярно-динамические расчеты машинах [8]–[10], машинах с ускорителями на основе GPU [11] и ПЛИС [12]–[17]. Было продемонстрировано, что ПЛИС может являться конкурентной альтернативой в качестве аппаратного ускорителя для молекулярно-динамических вычислений во многих случаях, однако на сегодняшний день не существует консенсуса о том, для каких именно задач и алгоритмов выгоднее применять платформу ПЛИС.

В данной работе мы рассматриваем важный частный случай молекулярной динамики – броуновскую динамику. Основная особенность метода броуновской динамики по сравнению с молекулярной динамикой заключается в том, что, молекулярная система моделируется более грубо, т.е. в качестве элементарных объектов моделирования выступают не отдельные атомы, а более крупные частицы, такие как отдельные домены макромолекул или целые макромолекулы. Молекулы растворителя и другие малые молекулы в явном виде не моделируются, а их эффекты учитываются в виде случайной силы. Таким образом удается значительно снизить размерность системы, что позволяет увеличить интервал времени, покрываемый модельными расчетами на порядки.

Нам неизвестны описанные в литературе попытки исследовать эффективность ПЛИС по сравнению с альтернативными платформами для ускорения задач броуновской динамики. Поэтому мы предприняли исследование данного вопроса на примере задачи моделирования деполимеризации микротрубочки методом броуновской динамики.

Микротрубочки – это трубки диаметром около 25 нм и длиной от нескольких десятков нанометров до десятков микрон, состоящие белка тубулина и входящие в состав внутреннего скелета живых клеток. Ключевой особенностью микротрубочек является их динамическая нестабильность, т.е. возможность спонтанно переключаться между фазами полимеризации и деполимеризации [18]. Это поведение важно прежде всего для захвата и перемещения хромосом микротрубочками во время клеточного деления. Кроме того, микротрубочки играют важную роль во внутриклеточном транспорте, движении ресничек и жгутиков и поддержании формы клетки [19]. Механизмы, лежащие в основе работы микротрубочек, исследуются уже несколько десятков лет, но лишь недавно развитие вычислительных

технологий позволило описывать поведение микротрубочек на молекулярном уровне. Наиболее подробная молекулярная модель динамики микротрубочек, созданная недавно нашей группой на основе метода броуновской динамики, была реализована базе CPU и позволяла рассчитывать времена полимеризации/деполимеризации микротрубочек порядка нескольких секунд [20]. Это пролило свет на ряд важных аспектов динамики микротрубочек, однако, тем не менее, многие ключевые экспериментально наблюдаемые явления остались за рамками теоретического описания, т.к. они происходят в микротрубочках на временах десятков и даже сотен секунд [21]. Таким образом, для прямого сравнения теории и эксперимента критически важно достигнуть ускорения расчетов динамики микротрубочек хотя бы на порядок величины.

В данной работе мы исследуем возможность ускорения расчетов броуновской динамики микротрубочки на ПЛИС и сравниваем результаты, полученные при реализации одного и того же алгоритма динамики микротрубочек на трех разных платформах, по критериям производительности и энергоэффективности.

2. Математическая модель

2.1 Общие сведения о структуре микротрубочки

Структурно микротрубочка представляет собой цилиндр, состоящий из 13 цепочек – протофиламентов (Рис.1).

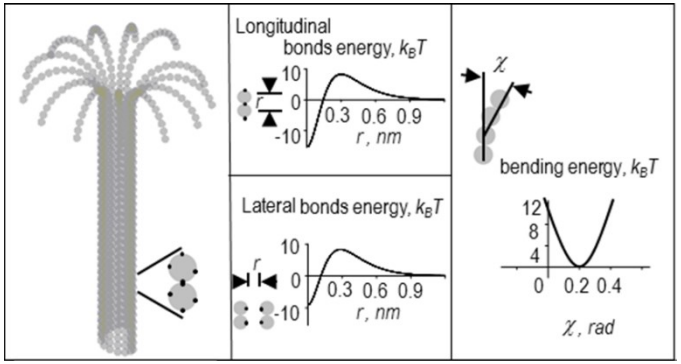

Рис. 1. Слева – схема модели микротрубочки. Серым показаны субъединицы тубулина, черными точками – центры взаимодействия между ними. Справа - вид энергетических потенциалов взаимодействия между тубулинами.

Fig. 1. On the left – the scheme of a microtubule model. Grey shows tubulin subunits, black spots – centers of interaction between them. Right – the kind of energy potential interaction between tubulins.

Каждый протофиламент построен из димеров белка тубулина. Соседние протофиламенты связаны друг с другом боковыми связями и сдвинуты относительно друг друга на расстояние 3/13 длины одного мономера, так что микротрубочка имеет спиральность. При полимеризации димеры тубулина присоединяются к концам протофиламентов, причем протофиламенты микротрубочки стремятся принимать прямую конформацию. При деполимеризации боковые связи между протофиламентами на конце микротрубочки разрываются, и протофиламенты закручиваются наружу. При этом от них случайным образом отрываются олигомеры тубулина.

2.2 Моделирование деполимеризации микротрубочки методом броуновской динамики

Используемая здесь молекулярная модель микротрубочки была впервые представлена в статье [20]. Поскольку задачей настоящего исследования являлось сравнение производительности различных вычислительных платформ, мы ограничились моделированием только деполимеризации микротрубочки.

Вкратце, микротрубочка моделировалась как набор сферических частиц, представляющих собой мономеры тубулина. Мономеры могли двигаться только в соответствующей им радиальной плоскости, т.е. в плоскости, проходящей через ось микротрубочки и соответствующий протофиламент. Таким образом, положение и ориентация каждого мономера полностью определялись тремя координатами: двумя декартовыми координатами центра мономера и углом ориентации. Каждый мономер имел четыре центра взаимодействия на своей поверхности: два центра бокового взаимодействия и два центра продольного взаимодействия. Энергия тубулин-тубулинового взаимодействия зависела от расстояния r между сайтами взаимодействия на поверхности соседних субъединиц и от угла наклона между соседними мономерами тубулина в протофиламенте.

Боковые и продольные взаимодействия между димерами тубулина определялись потенциалом, имеющим следующий вид:

$$v\ (r\ ) = A\ \cdot\left(\frac{r}{r_o}\right)^2 \cdot \exp\left(\frac{-r}{r_o}\right) - b\ \cdot \exp\left(\frac{-(r)^2}{d \cdot r_o}\right) \qquad (1)$$

где A и b определяли глубину потенциальной ямы и высоту энергетического барьера, r_0 и d – параметры, задающие ширину потенциальной ямы и форму потенциала в целом. Параметр A принимал различные значения для боковых и продольных связей, так что боковые взаимодействия были слабее продольных, все остальные параметры совпадали для обоих типов связей (полный список параметров и их значений представлен в Table 1 в [20]). Продольные взаимодействия внутри димера моделировались как неразрывные пружины с квадратичным энергетическим потенциалом $u(r)$:

$$u(r\ ) = \frac{1}{2} k \cdot r^2 \qquad (2)$$

где k - жесткость связи тубулин-тубулинового взаимодействия.

Энергия изгиба $g(\chi)$ связана с поворотом мономеров друг относительно друга и также описывалась квадратичной неразрывной функцией:

$$g(\chi) = \frac{B}{2}(\chi\ - \chi_o)^2 \qquad (3)$$

где χ - угол между соседними мономерами тубулина в протофиламенте, χ_0 - равновесный угол между двумя мономерами, B - изгибная жесткость.

Полная энергия микротрубочки записывалась следующим образом:

$$U_{total} = \sum_{n=1}^{13}\sum_{i=1}^{K_n}\left(v_{k,n}^{lateral} + v_{k,n}^{longitudinal} + u_{k,n} + g_{k,n}\right) \qquad (4)$$

где n - номер протофиламента, i - номер мономера в n-ом протофиламенте, K_n - число субъединиц тубулина в n-ом протофиламенте, $v_{k,n}^{lateral}$ - энергия бокового взаимодействия между мономерами, $v_{k,n}^{longitudinal}$ - энергия продольного взаимодействия между димерами.

Эволюция системы рассчитывалось с помощью метода Броуновской динамики [22]. Изначальной конфигурацией микротрубочки была короткая «затравка», содержащая 12 мономеров тубулина в каждом протофиламенте. Мы рассматривали только деполимеризацию МТ и моделировали все тубулины с равновесным углом $\chi_0 = 0.2$ рад. Координаты всех мономеров системы на i-ой итерации выражались следующим образом:

$$\begin{cases} q_{k,n}^{i} = q_{k,n}^{i-1} - \dfrac{dt}{\gamma_q} \cdot \dfrac{\partial U_{total}}{\partial q_{k,n}^{i}} + \sqrt{2k_B T \dfrac{dt}{\gamma_q}} \cdot N(0,1) \\ \tau_{k,n}^{i} = \tau_{k,n}^{i-1} - \dfrac{dt}{\gamma_\tau} \cdot \dfrac{\partial U_{total}}{\partial \tau_{k,n}^{i}} + \sqrt{2k_B T \dfrac{dt}{\gamma_\tau}} \cdot N(0,1) \end{cases} \qquad (5)$$

$$\gamma_q = 6\pi r \eta$$

$$\gamma_\tau = 8\pi r^3 \eta$$

где dt - шаг по времени, U_{total} выражается через (4), k_B - постоянная Больцмана, T- температура, $N(0,1)$ – случайное число из нормального распределения, сгенерированное с помощью алгоритма вихрь Мерсенна [23]. γ_q и γ_τ - вязкостные коэффициенты сопротивления для сдвига и поворота соответственно, рассчитанные для сфер радиуса $r = 2$ нм.

Производная полной энергии по независимым координатам $q^i_{k,n}$ выражалась через боковую, продольную составляющие энергии взаимодействия между соседними димерами и внутри димера, а также энергию изгиба:

$$\frac{\partial U_{total}}{\partial q^i_{k,n}} = \frac{\partial v^{lateral}_{k,n}}{\partial q^i_{k,n}} + \frac{\partial v^{longitudinal}_{k,n}}{\partial q^i_{k,n}} + \frac{\partial u_{k,n}}{\partial q^i_{k,n}} + \frac{\partial g_{k,n}}{\partial q^i_{k,n}} \qquad (6)$$

Для ускорения расчетов были использованы аналитические выражения для всех градиентов энергии:

$$\frac{\partial v^{lateral}_{k,n}}{\partial q^i_{k,n}} = \left(A_{lateral} \cdot \frac{r}{r_o^{\,2}} \cdot \exp\left(\frac{-r}{r_o}\right) \cdot \left(2 - \frac{r}{r_o}\right) + \frac{2 \cdot b_{lateral} \cdot r}{d \cdot r_o} \cdot \exp\left(\frac{-(r)^2}{d \cdot r_o}\right) \right) \cdot \frac{\partial r}{\partial q^i_{k,n}} \quad (7)$$

$$\frac{\partial v^{longitudinal}_{k,n}}{\partial q^i_{k,n}} = \left(A_{longitudinal} \cdot \frac{r}{r_o^{\,2}} \cdot \exp\left(\frac{-r}{r_o}\right) \cdot \left(2 - \frac{r}{r_o}\right) + \frac{2 \cdot b_{longitudinal} \cdot r}{d \cdot r_o} \cdot \exp\left(\frac{-(r)^2}{d \cdot r_o}\right) \right) \cdot \frac{\partial r}{\partial q^i_{k,n}} \quad (8)$$

$$\frac{\partial u_{k,n}}{\partial q^i_{k,n}} = k \cdot r \cdot \frac{\partial r}{\partial q^i_{k,n}} \quad (9)$$

$$\frac{\partial g_{k,n}}{\partial q^i_{k,n}} = \frac{\partial g_{k,n}}{\partial \chi_{k,n}} \cdot \frac{\partial \chi_{k,n}}{\partial q^i_{k,n}} = B \cdot (\chi - \chi_o) \cdot \frac{\partial \chi_{k,n}}{\partial q^i_{k,n}} \quad (10)$$

Следует отметить, что размер данной задачи сравнительно мал. Мы рассматривали только 12 слоев мономеров, что дает полное число частиц равное 156. Однако, это нисколько не уменьшает значимость вычислений, т.к. в реальных расчетах достаточно вычислять положение крайних нескольких (порядка 10) слов мономеров, т.к. при росте микротрубочки дальние от конца микротрубочки молекулы тубулина образуют устойчивую цилиндрическую конфигурацию, и брать их в расчет нет смысла.

2.3 Псевдокод алгоритма расчета

Алгоритм является итеративным по времени с шагом 0.2 нс. Существуют массив трехмерных координат молекул, а также массивы сил поперечного (латерального) и продольного (лонгитудального) взаимодействий. На каждой итерации по времени последовательно выполняются два вложенных цикла по молекулам, в первом производится вычисление сил взаимодействия по известным координатам, во втором – обновляются сами координаты. В цикле вычисления сил взаимодействия необходимо прочитать координаты трех молекул, одна центральная и две соседние («левая» и «верхняя», см. Рис.2), а результатом вычисления будет сила поперечного взаимодействия между центральной и левой молекулами и сила продольного взаимодействия между центральной и верхней.

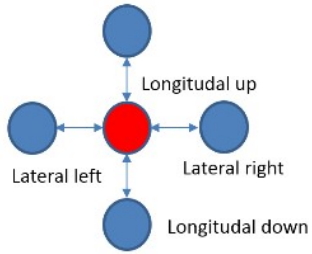

Рис. 2. Схема расположения взаимодействующих субъединиц в модели микротрубочки

Fig. 2. The scheme of arrangement of interacting subunits in the microtubule model

В итоге после этого цикла оказываются вычисленными все силы взаимодействия между всеми молекулами. В цикле обновления координат по известным силам вычисляются изменения координат, а также берутся в расчет случайные добавки для учета Броуновского движения. Таким образом, псевдокод алгоритма можно записать следующим образом.

Вход: массив координат молекул M = {x, y, teta}. Граничные условия на силы взаимодействия.

Выход: массив координат M после K шагов по времени

```
for t in {0.. K-1} do
    for i in {0.. 13} // количество протофиламентов
    for j in {0.. 12} do // количество слоев молекул
        Mc <- M[i,j]
        Ml <- M[i+1,j]
        Mu <- M[i,j+1]
        // по формулам (7, 8, 9, 10)
        F_lat[i,j]  <- calc_calteral(Mc, Ml)
        F_long[i,j] <- calc_long(Mc, Mu)
    end for

    for i in {0.. 13}
    for j in {0.. 12} do
        // по формулам (5)
        M[i,j] <- update_coords(F_lat[i,j], F_long[i,j])
    end for
end for
```

3. Программная реализация на CPU и GPU

3.1 Реализация на CPU

Была предпринята попытка максимально распараллелить код на CPU Intel Xeon E5-2660 2.20GHz под управлением ОС Ubuntu 12.04 с помощью библиотеки OpenMP. Параллельная секция начиналась до цикла по времени. Циклы расчета сил взаимодействия и обновления координат были распараллелены с помощью директивы *omp for schedule(static)*, между циклами была вставлена барьерная синхронизация. Массивы, содержащие силы взаимодействия и координаты молекул, были объявлены как *private* для каждого потока.

При реализации расчетов на CPU было обнаружено, что размер задачи не позволял ее эффективно распараллелить. Зависимость времени выполнения одной итерации от числа параллельных потоков была немонотонна. Минимальное время расчета одной итерации по времени было получено при использовании всего 2 потоков (ядер CPU). Объясняется это тем что, с увеличением количества потоков растет время на копирование данных между потоками и на их синхронизацию. При этом размер задачи очень мал, чтобы выигрыш от увеличения количества ядер превысил эти накладные расходы. При этом эксперименты показали, что задача слабо масштабируется при увеличении размера (weak scaling), т.е. при одновременном увеличении размера задачи и числа параллельных потоков время вычисления оставалось примерно одинаковым. В итоге лучшим результатом на данном CPU было 22 мкс на одну итерацию по времени при использовании двух ядер CPU. Код не был векторизован из-за сложности вычислений сил взаимодействия.

3.2 Реализация на GPU

Мы запускали OpenCL реализацию на граф процессоре Nvidia Tesla K40. Циклы, вычисляющие силы взаимодействия и обновления координат были распараллелены, главный цикл по времени был итеративным. Были реализованы два варианта – с одной и несколькими рабочим группами (work groups). В первом случае было выделено по одному рабочему потоку (work item) на каждую молекулу. В каждом потоке был цикл по времени, в котором вычислялись силы и координаты молекулы потока. При этом применялась барьерная синхронизация после вычисления сил и после обновления координат. В этом случае участие хоста не требовалось для вычислений, он только занимался управлением и запуском ядер.

Во втором случае были два типа потоков, в одном просто вычислялись силы для одной молекулы, во втором – обновлялись координаты. Главный цикл по времени был на хосте, который управлял запуском и синхронизацией ядер на каждой итерации цикла по времени.

Наибольшая производительность была получена в расчетах с одной группой потоков и барьерной синхронизацией между ними. Без использования генераторов псевдослучайных чисел одна итерация вычислялась в течение 5 мкс, если использовать один генератор чисел на все потоки, то время работы возрастало до 9 мкс, а при максимальном заполнении общей памяти (shared memory) удавалось включить 7 независимых генераторов, при этом время вычисления одной итерации по времени составило 14 мкс, что было в 1.57 раза быстрее реализации на CPU.

Загруженность ядер GPU составила 7% от одного мультипроцессора (SM), при этом общая память, где размещались массивы сил, координат и буферы данных генераторов псевдослучайных чисел, была заполнена на 100%. Т.е. с одной стороны размер задачи был явно мал для полной загрузки GPU, с другой стороны при увеличении размера задачи пришлось бы использовать глобальную DDR память, что могло бы привести к ограничению роста производительности.

4. Реализация на ПЛИС

4.1 Описание платформы

Вычисления на ПЛИС производились на платформе ПЛИС RB-8V7 производства фирмы НПО “Роста”. Она представляет собой 1U блок для установки в стойку. Блок состоит из 8 кристаллов ПЛИС Xilinx Virtex-7 2000T. Каждая ПЛИС имеет 1 GB внешней DDR3 памяти и PCI Express x4 2.0 интерфейс к внутреннему PCIe коммутатору. Блок имеет два интерфейса PCIe x4 3.0 к хост-компьютеру через оптические кабели, которые должны быть соединены со специальным адаптером, установленным в хост-компьютер.

В качестве хост-компьютера был использован сервер с CPU Intel Xeon E5-2660 2.20 GHz, работающий под управлением ОС Ubuntu 12.04 LTS – такой же как и для вычислений просто на CPU с помощью OpenMP. Программное обеспечение, работающее на CPU хост-компьютера «видит» блок RB-8V7 как 8 независимых ПЛИС устройств, подключенных по шине PCI Express. Далее будет описываться взаимодействие CPU только с одной ПЛИС XC7V72000T, при этом система позволяет использовать ПЛИС независимо и параллельно.

Ускоренное с помощью ПЛИС приложение было разработано с помощью SDK со следующей моделью. На CPU хост-компьютера (далее просто CPU) работает основная программа, которая использует ускоритель ПЛИС для наиболее вычислительно емких процедур. CPU передает данные в ускоритель и обратно через внешнюю DDR память, подключенную к ПЛИС, а также управляет работой вычислительного ядра в ПЛИС. Вычислительное ядро создается заранее на языке C/C++, верифицируется и транслируется в RTL код с помощью средства Vivado HLS. RTL код вычислительного ядра вставляется в основной ПЛИС проект, в котором уже реализована необходимая логика

управления и передачи данных, включающая PCI Express ядро, DDR контроллер и шину на кристалле (Рис. 3). Основной ПЛИС проект иногда называют Board Support Package (BSP), он разрабатывается производителем оборудования, и от пользователя не требуется его модификации. Вычислительное ядро HLS после запуска само обращается в DDR память, считывает оттуда входной буфер данных для обработки и записывает туда же результат вычислений. На уровне языка C++ обращение в память происходит через аргумент функции верхнего уровня вычислительного ядра типа указатель.

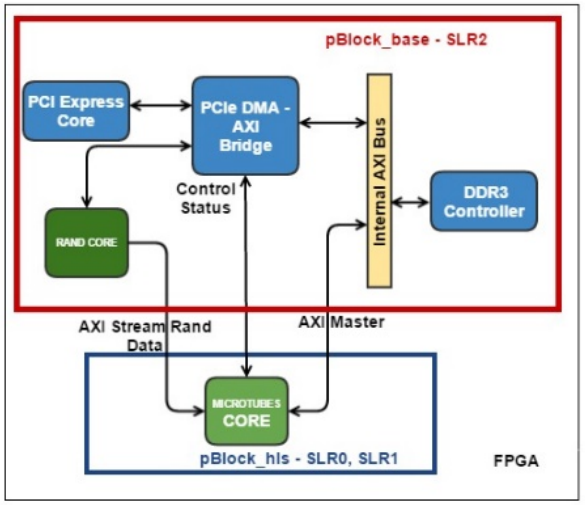

Рис. 3. Блок-схема проекта ПЛИС. Синим и желтым цветами отмечены блоки, входящие в BSP. Зеленым обозначены вычислительные HLS ядра. Также обозначено разбиение ядер проекта на блоки (pBlocks) для наложения пространсвенных ограничений при трассировке.

Fig. 3. A block diagram of the FPGA design. Blue and yellow colors mark blocks included in the BSP. Green color mark HLS computing cores. Splitting the cores of the project on the blocks (pBlocks) is also marked to impose space restrictions on tracking.

Для создания ускоренного приложения была разработана методология, состоящая из нескольких шагов. Во-первых, оригинальный последовательный код компилировался в среде Vivado HLS, и проверялось, что скомпилированный таких образом код не изменяет выходных данных опорного последовательного кода. Во-вторых, из этого кода выделялась основная вычислительная и подходящая для ускорения часть; эта часть отделялась от основного кода с помощью функции-обертки. После чего создавалось две копии такой функции и логика проверки на соответствие результатов обеих частей. Первая копия была опорной реализацией алгоритма

в Vivado HLS, а вторая была оптимизирована для трансляции в RTL код. Оптимизации включали в себе переписывание кода, такие как использование статических массивов вместо динамических, использование специальных функций для ввода/вывода в HLS ядро, методы экономии памяти и переиспользования результатов вычислений. После каждого изменения результат функции сравнивался с результатом опорной реализации. Другим методом оптимизации было использование специальных директив Vivado HLS, не меняющих логическое поведение, но влияющих на конечную производительность RTL кода. На данной стадии следует оставаться до тех пор, пока не будут получены удовлетворительные предварительные результаты трансляции C в RTL, такие как производительность схемы и занимаемые ресурсы.

Следующая стадия – это имплементация разработанного вычислительного ядра в системе Vivado вне контекста основного проекта. Здесь задача добиться отсутствия временных ошибок уже разведенного дизайна внутри разработанного вычислительного ядра. Если на этом этапе наблюдаются временные ошибки, то можно применять другие параметры имплементации, либо возвращаться на предыдущую стадию и пытаться изменить C++ код или использовать другие директивы.

На следующей стадии необходимо имплементировать вычислительное ядро уже вместе с основным проектом и его временными и пространственными ограничениями. На данной стадии также необходимо добиться отсутствия временных ошибок. Если они наблюдаются, то можно либо изменить частоту работы вычислительной схемы, наложить другие пространственные ограничения на размещение схемы на кристалле, либо опять заняться изменением C++ кода и/или использовать другие директивы.

Последняя стадия разработки – это проверка на соответствие результатов, полученных на реальном запуске в железе и с помощью опорной модели на CPU. Проходит она на небольшом промежутке времени, при этом считается, что на более длительных запусках (когда сравнить с CPU уже проблематично) ПЛИС решение выдает правильные результаты.

4.2 Работа в среде Vivado HLS

В работе использовалось два Vivado HLS ядра (Рис. 3): основное ядро, реализующее алгоритм молекулярной динамики микротрубочек (MT ядро), и ядро для генерации псевдослучайных чисел (RAND ядро). Нам пришлось разделить алгоритм на два вычислительных ядра по следующей причине. Кристалл ПЛИС Virtex-7 2000T – это самый большой кристалл ПЛИС семейства Virtex-7 на рынке. Он на самом деле состоит из четырех кристаллов кремния, соединенных на подложке множеством соединений и объединенных в один корпус микросхемы. По терминологии Xilinx каждый такой кристалл называется SLR (Super Logic Region). При использовании таких больших

ПЛИС всегда возникают проблемы с цепями, пересекающими границы SLR. Xilinx рекомендует вставлять регистры на такие цепи с обеих сторон границы SLR.

Полное HLS ядро, включающее и MT и RAND ядра, требовали аппаратных ресурсов больше, чем было доступно в одном SLR, поэтому были цепи, которые пересекали границу независимых кристаллов кремния. На стадии трансляции с языка C++ в RTL Vivado HLS ничего «не знает» о том, какие цепи будут впоследствии пересекать границу, и поэтому не может заранее вставить дополнительные регистры синхронизации. Поэтому мы приняли решение разделить ядра на два, пространственно ограничить их в разные SLR и вставить регистры синхронизации на интерфейсные цепи между ядрами на уровне RTL.

4.2.1 Ядро MT

Данный алгоритм очень хорошо подходит для реализации на ПЛИС, потому для сравнительно небольшого количества данных из памяти (координаты двух молекул) необходимо вычислить сложную функцию сил взаимодействия и удается построить длинный вычислительный конвейер.

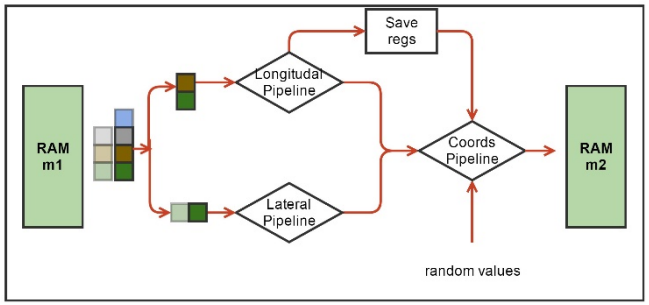

Рис. 4. Блок-схема аппаратной вычислительной процедуры ядра MT. Зеленым обозначены аппаратные блоки памяти для хранения координат молекул. Обозначены вычислительные конвейеры сил и обновления координат, а также блок Save Regs для хранения промежуточных результатов вычислений. Псевдослучайные числа поступают в конвейер обновления координат из другого HLS ядра.

Fig. 4 Block diagram of hardware procedure of MT core. Green color designates hardware memory blocks to store coordinates of molecules. Instruction pipelines to calculate forces and to update coordinates are marked, as well as block Save Regs for storing intermediate results of calculations. The pseudo-random numbers receive in the pipeline for coordinate updates from other HLS core.

Каждая молекула, т.е. мономер тубулина, взаимодействует только с четырьмя своими соседями (Рис. 2). На каждой итерации по времени надо сначала вычислить силы взаимодействия, а затем обновить координаты молекул.

Функции вычисления сил взаимодействия включают в себя множество арифметических, экспоненциальных и тригонометрических операторов. Нашей первой задачей было синтезировать конвейер для этих функций. Рабочим типом данных был вещественный тип float. Vivado HLS синтезировала такие функции в виде конвейеров, работающих на частоте 200 МГц, с латентностью порядка 130 тактов. При этом конвейеры были однотактовые (или, как говорят, с интервалом инициализации равной 1), что означает, что на вход они могли принимать координаты новых молекул каждый такт, а затем после начальной задержки (латентности) – выдавать обновленные значения сил также каждый такт. Выходные силы взаимодействия использовались для обновления координат, что тоже было конвейеризовано. Для обновления каждой координаты каждой молекулы были необходимо независимые псевдослучайные нормально распределенные числа, получаемые из другого HLS ядра. Если взять три молекулы («текущую», «левую» и «верхнюю») то получилось возможным объединить конвейеры вычисления сил и обновления координат в один конвейер, реализующий все вычисления для одной молекулы. Такой конвейер имел латентность равную 191 такт (Рис. 4).

Алгоритм проходит по всем молекулам в цикле. На каждой итерации цикла необходимо иметь координаты трех молекул: одна молекула рассматривается как «текущая», также есть «левая» и «правая» молекулы. Соответственно рассчитываются силы взаимодействия между этими тремя молекулами. Далее при обновлении координат текущей молекулы левая и верхняя компоненты сил взаимодействия брались из расчета на текущей итерации, а нижняя и правая компоненты брались либо из граничных условий, либо с предыдущих итераций из локального регистрового файла Save Regs (Рис. 4).

Количество молекул N в системе было небольшим (13 протофиламентов x 12 молекул = 156 молекул). На каждую молекулу требуется 12 байт. Схема использовала два массива координат m1 и m2, общим объемом меньше 4 КБ, соответственно эти данные легко помещались во внутреннюю память ПЛИС – BRAM, реализованную внутри HLS ядра. Схема была устроена таким образом, что на четных итерациях по времени координаты считывались из массива m1 (и записывались в m2), а на нечетных – наоборот. С точки зрения алгоритма можно было читать и писать в один массив координат, но Vivado HLS не могла создать схему, способную на одном и том же такте читать и писать один и тот же аппаратный массив, что требуется для работы однотактового конвейера. Поэтому было принято решение удвоить количество независимых блоков памяти.

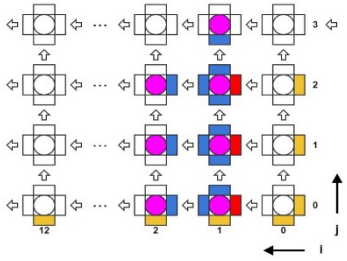

Рис. 5. Схема конвейерного расчета взаимодействий тубулинов в микротрубочке.

Fig. 5. The scheme of pipeline calculating of tubulin interactions in the microtubule.

Оказалось возможным реализовать три полных параллельных конвейера, способных обновлять координаты трех молекул каждый такт (Рис. 5). Тогда во избежание простаивания конвейеров необходимо было увеличить пропускную способность к локальной памяти и читать координаты семи молекул каждый такт. Это проблема легко решилась, практически не меняя исходный C++ код, а лишь за счет использования специальной директивы, физически разбивающей исходный массив данных по четырем независимым аппаратным блокам памяти. Т.к. память BRAM в ПЛИС является двупортовой, то из четырех блоков памяти можно прочитать 8 значений за такт. Но, так как три конвейера требуют координаты 7ми молекул за такт (см рис 5), это решило проблему.

```
#pragma HLS DATA_PACK variable=m1, m2
#pragma HLS ARRAY_PARTITION variable=m1, m2 cyclic factor=4 dim=2
```

Табл. 1: Производительность и утилизация схемы HLS с тремя полными конвейерами

Table. 1: Performance and utilization of HLS scheme with three complete pipelines

Период	L	II	BRAM	DSP	FF	LUT	Утилизация
5 нс	191 такт	1 такт	52	498	282550	331027	Абсолютная
			2 %	23 %	11 %	27 %	Относительная

В табл. 1 приводится утилизация схемы HLS (т.е. количество потребляемых ей аппаратных ресурсов ПЛИС, в абсолютных и относительных единицах для кристалла Virtex-7 2000T) и ее производительность. L – это задержка или латентность схемы, т.е. количество тактов между подачей в конвейер первых входных данных и получением первых выходных данных, II – это интервал инициализации (или пропускная способность) конвейера, означающее через сколько тактов на вход конвейера можно подавать следующие данные.

Утилизация приводится как в абсолютных величинах (сколько требуется триггеров FF или таблиц LUT) для реализации схемы, так и в относительных к полному количеству данного ресурса в кристалле.

Как видно из табл. 1 латентность L полного конвейера была равна 191 такту, при этом каждый конвейер должен был обработать третью часть все молекул, что дает теоретическую оценку времени вычисления одной итерации равную

$$\tau_{ПЛИС} = (L+N/3)*5нс=1.2\ мкс$$

Из табл. 1 также видно, что в кристалле осталось еще много неиспользованной логики, но дальше увеличивать количество параллельных конвейеров непрактично. Будет уменьшаться только второе слагаемое, а начальная задержка все равно будет давать значительный вклад во время работы. При этом увеличение количество логики усложнит размещение и трассировку схемы на следующих стадиях разработки проекта в Vivado.

4.2.2 Ядро RAND

Как было указано, алгоритм учитывает Броуновское движение молекул, одним из методов расчета которого является прибавление нормальной случайной добавки к изменению координат на каждой итерации по времени. Необходимо очень много нормально распределенных случайных чисел, на каждую итерацию – по N*3 чисел, что дает поток $420^{*}10^{6}$ чисел/с. Такой поток не может быть загружен с хоста, поэтому его необходимо генерировать внутри ПЛИС «на лету». Для этого, как и в опорном коде для CPU, был выбран генератор вихрь Мерсенна, дающий равномерно распределенные псевдослучаные числа. Далее к ним применялось преобразование Бокса-Мюллера и на выходе получались нормально распределенные последовательности. Исходный открытый код вихря Мерсенна был модифицирован для получения аппаратного конвейера с интервалом инициализации в 1 такт. Алгоритм требует 9 нормальных чисел каждый такт, поэтому ядро RAND включало в себя 10 независимых генераторов вихря Мерсенна, т.к. преобразование Бокса-Мюллера требует два равномерно распределенных числа для получения 2х нормально распределенных. В табл. 2 приводится утилизация ядра RAND.

Табл. 2: Утилизация ядра RAND

Table. 2: Utilization of the RAND core

BRAM	DSP	FF	LUT	Утилизация
30	41	48395	64880	Абсолютная
1.2 %	9 %	0.1 %	5.3 %	Относительная

Видно, что такое ядро требует значительную часть DSP ресурсов кристалла, и это ядро было бы сложно разместить в одном SLR с ядром MT, т.к. сумма

утилизаций двух ядер хотя бы по DSP ресурсу 31% больше чем может вместить один SLR (25 %).

4.3 Создание битстрима

После интеграции вычислительных ядер в Vivado на проект были наложены пространственные ограничения на размещение IP блоков. Используемая ПЛИС Virtex-7 2000T имеет 4 независимых кристалла кремния (SLR0, SLR1, SLR2, SLR3). Было показано, что ядро MT не умещалось в один SLR, поэтому было решено создать два региона размещение (pBlock): pBloch_hls для размещения только MT ядра и pBlock_base для размещения остальных ядер проекта (Рис. 3). Регион размещения pBlock_hls включал в себя SLR0 и SLR1, pBock_base – SLR2. Такой подход позволил разместить логику оптимальным образом, вставить регистры синхронизации на интерфейсы, пересекающие регионы размещения (а значит и SLR) и добиться положительных времянных результатов после трассировки проекта.

5. Результаты

5.1 Производительность

Результаты работы всех трех реализаций (CPU, GPU и ПЛИС) были логически верифицированы относительного оригинального кода и признаны состоятельными. Сравнение производительностей производилось замером времени работы программ на 10^7 итераций алгоритма и вычислением времени, требующегося для расчета одной итерации. При этом производительность GPU и ПЛИС платформ брали в расчет время передачи данных между хост-компьютером и ускорителем.

Для оценки производительности обычно используется метрика операций в секунду. Для данного алгоритма нам оказалось сложным вычислить точное значений вещественных операций, поэтому мы просто сравниваем времена работы алгоритма для вычисления одной итерации, определяя производительность CPU платформы равной 1. Результаты сравнения приводятся в табл. 3, во втором столбце которой приводятся времена вычисления одной итерации алгоритма в микросекундах, а в третьем – относительная производительность платформ.

Табл. 3: Сравнение производительности трех платформ

Table 3: Comparison of the performance of the three platforms

Платформа	Время, мкс	Производительность
CPU	22	1
GPU	14	1,6
ПЛИС	1.3	17

Из таблицы видно, что реализация на GPU быстрее CPU всего в 1.6 раза, в то время как ПЛИС быстрее CPU в 17 раз. Это означает, что ПЛИС быстрее GPU в 11 раз. Полученное экспериментально время работы ПЛИС равно 1.3 мкс на итерацию больше расчетного времени в 1.2 мкс из-за учета накладных расходов на передачу данных по шине PCI Express.

5.2 Энергоэффективность

Для измерения энергопотребления мы использовали следующие средства. Для CPU платформы – утилиту Intel Power Gadget. Для GPU платформы - утилиту Nvidia-smi. Для ПЛИС – специальные программно-аппаратные средства, включенные в состав блока RB-8V7. Во всех случаях замерялась разница в потреблении всего чипа до запуска задачи и во время вычислений. Результаты приведены в таблице 4.

Табл. 4: Сравнение энергопотребления вычислительных платформ

Table 4: Comparison of power consumption of computing platforms

Платформа	Мощность, Вт	E_x, W^{-1}	E_X^{rel}
CPU	89.6	0.011	1
GPU	67	0.033	3
ПЛИС	9.6	2.5	227

Во втором столбце таблицы приводится мощность, выделяющаяся при расчете на разных платформах. В третьем столбце приводятся значения абсолютной энергоэффективности (производительности на Вт) для данной задачи, определяемой по формуле

$$E_x = {P_x}/{Pow_x}$$

В четвертом столбце приводятся значения относительной энергоэффективности разных платформ для данной задачи, вычисляемой по формуле

$$E_X^{rel} = {E_x}/{E_{CPU}}$$

Для обеих формул, x = {CPU, GPU, ПЛИС}.

Видно, что у ПЛИС есть большое преимущество в энергоэффективности перед другими платформами, что может сыграть роль в средне и долгосрочной перспективе использования ПЛИС ускорителей в датацентрах при оплате счетов за электроэнергию. Достигается это в первую очередь за счет того, что ПЛИС работают на порядок меньшей частоте.

6. Обсуждение

В ранее опубликованных работах технология ПЛИС неоднократно применялась к решению задач молекулярной динамики [12]–[17]. Исследователям из лаборатории CAAD Бостонского Университета удалось разработать эффективное ядро для расчета короткодействующих межмолекулярных сил, которое было реализовано на плате ProcStar-III (производство фирмы Gidel), с установленным кристаллом ПЛИС Altera Stratix-III SE260. Плата имела PCI Express интерфейс к хост-компьютеру. Было показано, что разработанное ускоренное решение было в 26 раз быстрее чистой реализации на CPU на бенчмарке Apoal. В работе [24] авторы перенесли часть пакета для расчета молекулярной динамики LAMMPS на ПЛИС. Ускоренная часть включала в себя вычисления дальнодействующих взаимодействий. Разработанное аппаратное ядро состояло из четырех одинаковых независимых конвейеров, работающих параллельно. Задача была выполнена на суперкомпьютере Maxwell, каждый узел которого состоит из одного процессора Intel Xeon и двух кристаллов ПЛИС Xilinx Virtex-4 [25]. Авторы заявили, что разработанное ускоренное решение легко масштабировалось на множество узлов суперкомпьютера Maxwell. Из анализа производительности только ускорителя следовало, что на двух узлах компьютера можно было получить ускорение в 13 раз по сравнению с чисто программным решением. Однако полное время работы гибридного решения было хуже чисто программного из-за того, что время на пересылку данных между CPU и внешней памятью SDRAM, подключенной к ПЛИС занимало 96% времени работы всего алгоритма. Но в работе утверждается, что если улучшить интерфейс передачи данных, то можно получить полный выигрыш в скорости в 8-9 раз.

В настоящей работе мы применили ПЛИС к расчету движения ансамбля белковых молекул методом броуновской динамики. Наша программно-аппаратная реализация алгоритма деполимеризации микротрубочки показала, что производительность ПЛИС при расчете одной траектории микротрубочки в 17 раз превосходила производительность CPU и в 11 раз производительность GPU. Полученное ускорение при расчете деполимеризации микротрубочки методом броуновской динамики позволяет осуществить расчет на временах порядка нескольких десятков и даже сотен секунд. Это позволит предсказывать поведение реальных микротрубочек на экспериментально доступных временах и проанализировать механизмы динамической нестабильности микротрубочек, что будет предметом будущей работы в данном направлении.

Полученный выигрыш на задаче броуновской динамики позволяет говорить о перспективности применения ПЛИС для решения данного типа задач. Насколько нам известно, это первая попытка сравнить производительность и

энергоэффективность различных типов аппаратных ускорителей на данном алгоритме.

Долгое время главной проблемой использования ПЛИС являлось отсутствие высокоуровневых средств программирования. Традиционные языки описания аппаратуры всегда требуют значительного времени для реализации алгоритма, в то время как первые высокоуровневые трансляторы [26] генерировали RTL код низкого качества. Однако, несколько лет назад компании Altera и Xilinx стали уделять значительные ресурсы этой проблеме и выпустили на рынок свои высокоуровневые средства программирования (Altera SDK for OpenCL и Xilinx Vivado HLS). Данные трансляторы генерируют намного более эффективный код и позволяют прикладному программисту использовать языки C/C++ (Xilinx) и OpenCL (Altera) для создания качественных аппаратных вычислительных схем. В последнее время появилось множество работ, в которых использовались средства высокоуровневого синтеза для разработки ПЛИС ускорителей [27]–[29]. Например, в работе [28] с помощью средства Vivado HLS реализован алгоритм оптического потока на платформе Xilinx Zynq-7000. Разработанная система имела производительность сравнимую с реализацией на CPU, при этом потребление энергии было в 7 раз меньше. Авторы особенно подчеркивали, что использование средств HLS по сравнению с традиционными RTL языками значительно сократило срок разработки. Использование средства Vivado HLS в ходе выполнения настоящей работы также позволило значительно сократить время и трудоемкость разработки и привлекать к программированию разработчиков, не владеющих специальными навыками работы с ПЛИС. Все это позволяет говорить об ПЛИС как о состоявшейся платформе для высокопроизводительных вычислений в области молекулярной и броуновской динамики.

Признательности

Работа была поддержана грантом РФФИ, проект № 16-04-01862 А и проект № 16-34-60113 мол_а_дк. Авторы благодарят НПО РОСТА за предоставленное оборудование. Программы, использованные для расчетов в этом исследовании, могут быть найдены по ссылке: https://github.com/urock/FpgaMicrotubule.

Список литературы

[1]. B. Liu, D. Zydek, H. Selvaraj, and L. Gewali. «Accelerating High Performance Computing Applications: Using CPUs, GPUs, Hybrid CPU/GPU, and FPGAs». *In 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2012, pp. 337–342.

[2]. Wim Vanderbauwhede и K. Benkrid. *High-Performance Computing Using FPGAs*. Springer, 2013.

[3]. J. Fowers, G. Brown, P. Cooke, and G. Stitt. «A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-window Applications». *In Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, 2012, pp. 47–56.

[4]. K. Sano, Y. Hatsuda, and S. Yamamoto. «Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth». *IEEE Trans. Parallel Distrib. Syst.* 2014, vol. 25, no. 3, pp. 695–705.

[5]. K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian. «High Performance Biological Pairwise Sequence Alignment: FPGA Versus GPU Versus Cell BE Versus GPP». *Int. J. Reconfig. Comput.*, vol. 2012, 2012.

[6]. B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, M. C. Pitman, and R. S. Germain. «Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics». *In Proceedings of the ACM/IEEE SC 2006 Conference*, 2006, pp. 44–44.

[7]. K. J. Bowers, D. E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw, «Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters». *In Proceedings of the ACM/IEEE SC 2006 Conference*, 2006, pp. 43–43.

[8]. Y. Komeiji, M. Uebayasi, R. Takata, A. Shimizu, K. Itsukashi, and M. Taiji. «Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer». *J. Comput. Chem.*, vol. 18, no. 12, pp. 1546–1563, 1997.

[9]. D. E. Shaw, J. P. Grossman, J. A. Bank, B. Batson, J. A. Butts, J. C. Chao, M. M. Deneroff, R. O. Dror, A. Even, C. H. Fenton, A. Forte, J. Gagliardo, G. Gill, B. Greskamp, C. R. Ho, D. J. Ierardi, L. Iserovich, J. S. Kuskin, R. H. Larson, T. Layman, L.-S. Lee, A. K. Lerer, C. Li, D. Killebrew, K. M. Mackenzie, S. Y.-H. Mok, M. A. Moraes, R. Mueller, L. J. Nociolo, J. L. Peticolas, T. Quan, D. Ramot, J. K. Salmon, D. P. Scarpazza, U. Ben Schafer, N. Siddique, C. W. Snyder, J. Spengler, P. T. P. Tang, M. Theobald, H. Toma, B. Towles, B. Vitale, S. C. Wang, and C. Young. «Anton 2: Raising the Bar for Performance and Programmability in a Special-purpose Molecular Dynamics Supercomputer». *In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Piscataway, NJ, USA, 2014, pp. 41–53.

[10]. M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, and A. Konagaya. «Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations». *In Supercomputing, 2003 ACM/IEEE Conference*, 2003, pp. 15–15.

[11]. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, and W.-Mei Hwu. «GPU acceleration of cutoff pair potentials for molecular modeling applications». *Proceedings of the 5th conference on Computing frontiers*, 2008, pp. 273–282.

[12]. S. R. Alam, P. K. Agarwal, M. C. Smith, J. S. Vetter, and D. Caliga. «Using FPGA Devices to Accelerate Biomolecular Simulations». *Computer*, vol. 40, no. 3, 2007, pp. 66–73.

[13]. N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow. «Reconfigurable molecular dynamics simulator». *In 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004. FCCM 2004*, pp. 197–206.

[14]. Y. Gu, T. VanCourt, and M. C. Herbordt. «Improved Interpolation and System Integration for FPGA-Based Molecular Dynamics Simulations». *In 2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.

[15]. V. Kindratenko and D. Pointer. «A case study in porting a production scientific supercomputing application to a reconfigurable computer». 2006, pp. 13–22.

[16]. R. Scrofano, M. B. Gokhale, F. Trouw, and V. K. Prasanna. «Accelerating Molecular Dynamics Simulations with Reconfigurable Computers». *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 6, 2008, pp. 764–778.

[17]. M. Chiu and M. C. Herbordt. «Molecular Dynamics Simulations on High-Performance Reconfigurable Computing Systems». *ACM Trans Reconfigurable Technol Syst*, vol. 3, no. 4, 2010, pp. 23:1–23:37.

[18]. T. Mitchison and M. Kirschner. «Dynamic instability of microtubule growth». *Nature*, vol. 312, no. 5991, 1984, pp. 237–242.

[19]. A. Desai and and T. J. Mitchison. «Microtubule Polymerization Dynamics». *Annu. Rev. Cell Dev. Biol.*, vol. 13, no. 1, 1997, pp. 83–117.

[20]. P. Zakharov, N. Gudimchuk, V. Voevodin, A. Tikhonravov, F. I. Ataullakhanov, and E. L. Grishchuk. «Molecular and Mechanical Causes of Microtubule Catastrophe and Aging». *Biophys. J.*, vol. 109, no. 12, , 2015, pp. 2574–2591.

[21]. M. K. Gardner, M. Zanic, C. Gell, V. Bormuth, and J. Howard, «Depolymerizing Kinesins Kip3 and MCAK Shape Cellular Microtubule Architecture by Differential Control of Catastrophe». *Cell*, vol. 147, no. 5, , 2011, pp. 1092–1103.

[22]. D. L. Ermak and J. A. McCammon, «Brownian dynamics with hydrodynamic interactions». *J. Chem. Phys.*, v. 69, issue 4, , 1978, pp. 1352–1360.

[23]. M. Matsumoto and T. Nishimura, «Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator». ACM Trans. Model. Comput. Simul., vol. 8, 1998, pp. 3–30.

[24]. S. Kasap and K. Benkrid, «Parallel processor design and implementation for molecular dynamics simulations on a FPGA-Based supercomputer». *J. Comput.*, vol. 7, no. 6, 2012, pp. 1312–1328.

[25]. R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, «Maxwell - a 64 FPGA Supercomputer». *In Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, 2007, pp. 287–294.

[26]. J. M. P. Cardoso, P. C. Diniz, and M. Weinhardt, «Compiling for reconfigurable computing: A survey». *ACM Comput. Surv. CSUR*, vol. 42, no. 4, p. 13, 2010.

[27]. Y. Liang, K. Rupnow, Y. Li, D. Min, M. N. Do, and D. Chen, «High-level synthesis: productivity, performance, and software constraints». *J. Electr. Comput. Eng.*, vol. 2012, 2012, p. 1.

[28]. J. Monson, M. Wirthlin, and B. L. Hutchings, «Implementing high-performance, low-power FPGA-based optical flow accelerators in C». IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors, 2013, pp. 363–369.

[29]. T. Hussain, M. Pericàs, N. Navarro, and E. Ayguadé, «Implementation of a Reverse Time Migration kernel using the HCE High Level Synthesis tool». Field-Programmable Technology (FPT), 2011 International Conference, 2011, pp. 1–8.

PGA HPC Implementation of Microtubule Brownian Dynamics Simulations

[1, 3] *Rumyanstev Y.A. <yarumyantsev@gmail.com>*
[2] *Zakharov P.N. <pavel.n.zaharov@gmail.com>*
[1] *Abrashitova N. A. <natascha.abraschitowa@gmail.com >*
[1] *Shmatok A.V. <papercompute@gmail.com >*
[3] *Ryzhikh V.O. <vo.ryzhikh@mail.ru>*
[2,3,4] *Gudimchuk N.B. <gudimchuk@phys.msu.ru>*
[2,3,4] *Ataullakhanov F.I. <ataullakhanov.fazly@gmail.com>*
[1] *ROSTA LTD,*
Jivopisnaya str., 3/1, Moscow, 123103, Russia
[2] *Center for Theoretical Problems of Physico-chemical Pharmacology, Russian Academy of Sciences,*
Kosigina str, 4, Moscow, 119991, Russia
[3] *Lomonosov Moscow State University,*
Leninskie gori, 1, Moscow, 119991, Russia
[4] *Federal Research Center of Pediatric Hematology, Oncology and Immunology named after Dmitriy Rogachev*
Samory Mashela, 1, Moscow, 117997, Russia (FRC-PHOI)

Abstract. This paper presents high performance simulation of microtubule molecular dynamics implemented on Xilinx Virtex-7 FPGA using high level synthesis tool Vivado HLS. FPGA implementation is compared to multicore Intel Xeon CPU and Nvidia K40 GPU implementations in terms of performance and energy efficiency. Algorithm takes into account Brownian motion thus heavily uses normally distributed random numbers. Original sequential code was optimized for different platforms using OpenMP for CPU, OpenCL for GPU and Vivado HLS for FPGA. We show that in terms of performance FPGA achieved 17x speed up against CPU and 11x speedup against GPU for our best optimized CPU and GPU versions. As to power efficiency, FPGA outperformed CPU 227 times and GPU 75 times. FPGA application is developed using SDK, which has Board Support Package including FPGA project framework where accelerator kernel (designed in Vivado HLS) IP core is to be integrated, and host-side libraries used to communicate with FPGA via PCI Express. Developed flow does not require expert FPGA skills and can be used by programmer with little knowledge of hardware design methodology that could use C\C++ language for complete development of FPGA accelerated solution.

Keywords: HPC; FPGA; Microtubule; HLS; Brownian Dynamics

DOI: 10.15514/ISPRAS-2016-28(3)-15

For citation: Rumyanstev Y.A., Zakharov P.N., Abrashitova N. A., Shmatok A.V., Ryzhikh V.O., Gudimchuk N.B., Ataullakhanov F.I. PGA HPC Implementation of Microtubule Brownian Dynamics Simulations. *Trudy ISP RAN / Proc. ISP RAS*, vol. 28, issue 3, 2016, pp. 241-266 (in Russian). DOI: 10.15514/ISPRAS-2016-28(3)-15

References

[1]. B. Liu, D. Zydek, H. Selvaraj, and L. Gewali. «Accelerating High Performance Computing Applications: Using CPUs, GPUs, Hybrid CPU/GPU, and FPGAs». *In 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2012, pp. 337–342.

[2]. Wim Vanderbauwhede and K. Benkrid. *High-Performance Computing Using FPGAs*. Springer, 2013.

[3]. J. Fowers, G. Brown, P. Cooke, and G. Stitt. «A Performance and Energy Comparison of FPGAs, GPUs, and Multicores for Sliding-window Applications». *In Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, 2012, pp. 47–56.

[4]. K. Sano, Y. Hatsuda, and S. Yamamoto. «Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth». *IEEE Trans. Parallel Distrib. Syst.* 2014, vol. 25, no. 3, pp. 695–705.

[5]. K. Benkrid, A. Akoglu, C. Ling, Y. Song, Y. Liu, and X. Tian. «High Performance Biological Pairwise Sequence Alignment: FPGA Versus GPU Versus Cell BE Versus GPP». *Int. J. Reconfig. Comput.*, vol. 2012, 2012.

[6]. B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. Giampapa, M. C. Pitman, and R. S. Germain. «Blue Matter: Approaching the Limits of Concurrency for Classical Molecular Dynamics». *In Proceedings of the ACM/IEEE SC 2006 Conference*, 2006, pp. 44–44.

[7]. K. J. Bowers, D. E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw, «Scalable Algorithms for Molecular Dynamics Simulations on Commodity Clusters». *In Proceedings of the ACM/IEEE SC 2006 Conference*, 2006, pp. 43–43.

[8]. Y. Komeiji, M. Uebayasi, R. Takata, A. Shimizu, K. Itsukashi, and M. Taiji. «Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer». *J. Comput. Chem.*, vol. 18, no. 12, pp. 1546–1563, 1997.

[9]. D. E. Shaw, J. P. Grossman, J. A. Bank, B. Batson, J. A. Butts, J. C. Chao, M. M. Deneroff, R. O. Dror, A. Even, C. H. Fenton, A. Forte, J. Gagliardo, G. Gill, B. Greskamp, C. R. Ho, D. J. Ierardi, L. Iserovich, J. S. Kuskin, R. H. Larson, T. Layman, L.-S. Lee, A. K. Lerer, C. Li, D. Killebrew, K. M. Mackenzie, S. Y.-H. Mok, M. A. Moraes, R. Mueller, L. J. Nociolo, J. L. Peticolas, T. Quan, D. Ramot, J. K. Salmon, D. P. Scarpazza, U. Ben Schafer, N. Siddique, C. W. Snyder, J. Spengler, P. T. P. Tang, M. Theobald, H. Toma, B. Towles, B. Vitale, S. C. Wang, and C. Young. «Anton 2: Raising the Bar for Performance and Programmability in a Special-purpose Molecular Dynamics Supercomputer». *In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Piscataway, NJ, USA, 2014, pp. 41–53.

[10]. M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, and A. Konagaya. «Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations». *In Supercomputing, 2003 ACM/IEEE Conference*, 2003, pp. 15–15.

[11]. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, and W.-Mei Hwu. «GPU acceleration of cutoff pair potentials for molecular modeling applications». *Proceedings of the 5th conference on Computing frontiers*, 2008, pp. 273–282.

[12]. S. R. Alam, P. K. Agarwal, M. C. Smith, J. S. Vetter, and D. Caliga. «Using FPGA Devices to Accelerate Biomolecular Simulations». *Computer*, vol. 40, no. 3, 2007, pp. 66–73.

[13]. N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow. «Reconfigurable molecular dynamics simulator». *In 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004. FCCM 2004*, pp. 197–206.

[14]. Y. Gu, T. VanCourt, and M. C. Herbordt. «Improved Interpolation and System Integration for FPGA-Based Molecular Dynamics Simulations». *In 2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.

[15]. V. Kindratenko and D. Pointer. «A case study in porting a production scientific supercomputing application to a reconfigurable computer». 2006, pp. 13–22.

[16]. R. Scrofano, M. B. Gokhale, F. Trouw, and V. K. Prasanna. «Accelerating Molecular Dynamics Simulations with Reconfigurable Computers». *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 6, 2008, pp. 764–778.

[17]. M. Chiu and M. C. Herbordt. «Molecular Dynamics Simulations on High-Performance Reconfigurable Computing Systems». *ACM Trans Reconfigurable Technol Syst*, vol. 3, no. 4, 2010, pp. 23:1–23:37.

[18]. T. Mitchison and M. Kirschner. «Dynamic instability of microtubule growth». *Nature*, vol. 312, no. 5991, 1984, pp. 237–242.

[19]. A. Desai and T. J. Mitchison. «Microtubule Polymerization Dynamics». *Annu. Rev. Cell Dev. Biol.*, vol. 13, no. 1, 1997, pp. 83–117.

[20]. P. Zakharov, N. Gudimchuk, V. Voevodin, A. Tikhonravov, F. I. Ataullakhanov, and E. L. Grishchuk. «Molecular and Mechanical Causes of Microtubule Catastrophe and Aging». *Biophys. J.*, vol. 109, no. 12, , 2015, pp. 2574–2591.

[21]. M. K. Gardner, M. Zanic, C. Gell, V. Bormuth, and J. Howard, «Depolymerizing Kinesins Kip3 and MCAK Shape Cellular Microtubule Architecture by Differential Control of Catastrophe». *Cell*, vol. 147, no. 5, , 2011, pp. 1092–1103.

[22]. D. L. Ermak and J. A. McCammon, «Brownian dynamics with hydrodynamic interactions». *J. Chem. Phys.*, v. 69, issue 4, , 1978, pp. 1352–1360.

[23]. M. Matsumoto and T. Nishimura, «Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator». ACM Trans. Model. Comput. Simul., vol. 8, 1998, pp. 3–30.

[24]. S. Kasap and K. Benkrid, «Parallel processor design and implementation for molecular dynamics simulations on a FPGA-Based supercomputer». *J. Comput.*, vol. 7, no. 6, 2012, pp. 1312–1328.

[25]. R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, «Maxwell - a 64 FPGA Supercomputer». *In Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, 2007, pp. 287–294.

[26]. J. M. P. Cardoso, P. C. Diniz, and M. Weinhardt, «Compiling for reconfigurable computing: A survey». *ACM Comput. Surv. CSUR*, vol. 42, no. 4, p. 13, 2010.

[27]. Y. Liang, K. Rupnow, Y. Li, D. Min, M. N. Do, and D. Chen, «High-level synthesis: productivity, performance, and software constraints». *J. Electr. Comput. Eng.*, vol. 2012, 2012, p. 1.

[28]. J. Monson, M. Wirthlin, and B. L. Hutchings, «Implementing high-performance, low-power FPGA-based optical flow accelerators in C». IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors, 2013, pp. 363–369.

[29]. T. Hussain, M. Pericàs, N. Navarro, and E. Ayguadé, «Implementation of a Reverse Time Migration kernel using the HCE High Level Synthesis tool». Field-Programmable Technology (FPT), 2011 International Conference, 2011, pp. 1–8.