УДК 004.021

Стиренко С.Г., к.т.н.,
Грубый П.В.

MULTI-GPU АЛГОРИТМ ОЦЕНКИ ВЗАИМНОЙ ИНФОРМАЦИИ НА ОСНОВЕ B-СПЛАЙН ФУНКЦИИ

Національний технічний університет України "КПІ"

Представлен новый подход к ускорению метода оценки взаимной информации, основанном на B-сплайн функции, при помощи графических ускорителей. Для получения эффективного отображе-ния на этот тип архитектуры, была использована модель программирования Compute Unified Device Architecture (CUDA) для разработки и реализации нового распределенного алгоритма, основанного на CUDA-MI. Предложенная реализация показала ускорения до 224 раз при использовании двойной точности на 4 GPU по сравнению с многопоточной реализацией на четырехядерном процессоре для больших наборов данных. Полученные результаты использовались для генерации корреляционных матриц траекторий движения молекул сложных белков. Сравнение с существующими методами, включая g_corellation показало повышение качества получения матриц корреляций за меньшее время

Введение

Взаимная информация (*Mutual Information*, далее *MI*) используется для измерения взаимной зависимости между двумя случайными величинами в теории информации. Как теоретико-информационный подход, она была использована в различных областях, включая физику[1], обработку изображений [2], распознавание речи [4], и биоинформатику [5, 6]. Преимуществом MI по сравнению со многими другими мерами сходства (например, коэффициент корреляции Пирсона), является ее способность обнаружить нелинейные корреляции между двумя переменными [7]. В работе [1], представлен рекурсивный метод для расчета MI и протестирован на динамических системах и хаотических данных. Обзор *MI*, используемой в медицинских целях для визуализации процессов, представлен в [2]. Приложение, основанное на *MI* для нахождения решений аудиовизуальных задач распознавания речи, предложено в [4]. Чжоу и соавторы [5] использовали *MI* для кластеризации молекул, чтобы определить их регуляционную функцию. В работе [6], *MI* используется для измерения нелинейной связи между выражениями из двух молекул.

Из-за присущей алгоритмической сложности произведения корреляционных матриц с профилей молекулярных выражений (микромассивов), такие задачи остаются большой проблемой в вычислительной биологии [7-9]. Для их решения были использованы такие подходы, как актуальность сетей [10], теоретико-информационные методы [11], и байесовские сети [12, 13]. Однако, в связи с их высокой вычислительной сложностью, эти методы являются очень трудоемкими и не могут быть использованы для обработки больших наборов данных. Дауб и соавторы [7] предложили метод оценки взаимной информации, основанный на *B*-сплайн функции, который может достичь точности сравнимой с методом ядра. Так как B-сплайн подход имеет меньшую вычислительную сложность, он широко используется на практике. Однако, этот подход все еще слишком медлителен для больших наборов данных. Так как наличие и размер таких данных быстро растет, очень важно найти быстрое решение данной задачи.

В этой статье представлен современный подход к ускорению оценки *MI* при помощи *B*-сплайн функции на основе *CUDA*-MI [26]. Для ускорения операций ввода/вывода используется общая память. Вычисления выполнены с двойной точностью и с использованием эффективных схем разбиения данных для преодоления ограничений памяти *GPU* при больших наборах данных. Представленная реализация тестируется при помощи микромассивов данных. Приложение достигает увеличения скорости до 224 на 4-х *NVIDIA Tesla* C2050 *GPU* по сравнению с *g-correlation* [27], который работает на процессоре *Intel Quad-Core* i7-920 2,66 ГГц. Результаты оценки *MI* используются для создания корреляционных матриц. Результаты показывают, что время, необходимое для создания корреляционных матриц гораздо меньше, а качество получаемой матрицы лучше по сравнению с первоначальным *g-correlation* [28].

Документ, представленный Вильсоном и соавторами [15] похож на подход, представленный в этой статье, поскольку он также использует *CUDA* для ускорения оценки *MI* на основе *B*-сплайнов. Тем не менее, он всего лишь реа

лизует вычисления матрицы весов на *GPU*. Расчет взвешенной матрицы является лишь одним шагом в методе оценки взаимной информации, основанном на *B*-сплайн функции. Остальные шаги выполняются на *CPU*, что ограничивает возможное ускорение. Предложенное решение преодолевает это ограничение за счет ускорения всех этапов оценки *MI* на основе B-сплайнов при помощи нескольких ядер *CUDA*. За счет этого достигается большее ускорение, чем в работе [15].

Определение взаимной информации

Взаимная информация (MI) используется как количественная оценка для измерения взаимозависимости двух дискретных случайных величин. Для случайной величины *X* со значениями из конечного множества *M* возможных значений (или состояний) $\{x_1, x_2, ..., x_M\}$, энтропия Шеннона *H (X)* определяется следующим
образом:

$$H(X) = -\sum_{i=1}^{M} p(x_i) \log p(x_i) \quad (1)$$

где $p(x_i)$ – это вероятность значения x_i.Энтропия Шеннона – это мера неопределенности *X*. Если результат измерения *X* полностью определен, то $p(x_i) = 1$ и энтропия будет равна 0. Взаимная энтропия *H (X, Y)* двух дискретных переменных *(X* и *Y),* состоящих из состояний $\{x_1, ..., x_M\}$ и $\{x_1, ... y_M\}$ определяется следующим образом:

$$H(X,Y) = -\sum_{i=1}^{M}\sum_{j=1}^{M} p(x_i, y_j) \log p(x_i, y_j) \quad (2)$$

где $p(x_i, y_j)$ является взаимной вероятностью между двумя состояниями x_i и y_j. Совместная энтропия - это количество энтропии, содержащейся в совместной системе двух переменных. *MI* для *X* и *Y* может быть как:

$$MI(X,Y) = H(X) + H(Y) - H(X,Y) \quad (3)$$

Из уравнения (3) видно, что *MI* равна нулю, когда *X* и *Y* являются независимыми.

Оценка взаимной информации непрерывных данных

Выполнить оценку MI можно легко и для непрерывных данных. Для того чтобы оценить MI для непрерывных данных (или измерений), которые поставляются многими приложениями, обычно непрерывные

величины разделяют на *R* дискретных контейнеров $\{a_1, ..., a_R\}$. Каждый контейнер может содержать несколько точек. Предположим, что непрерывный набор данных состоит из *M из измерений* $\{x_1, ..., x_M\}$. Характеристическая функция θ_j используется для подсчета числа измерений в пределах каждого контейнера a_j (j = 1, ..., *R*) [7]. Вероятность для каждого контейнера равна числу измерений в контейнере, разделенному на общее число измерений и определена в (4)

$$p'(a_j) = \frac{1}{M}\sum_{i=1}^{M} \theta_j(x_i) \quad (4)$$

Характеристическая функция θ_j является бинарной и определяется как:

$$\theta_j(x_i) = \begin{cases} 1 \text{ if } x_i \in a_j \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

Взаимная вероятность $p'(a_j, b_k)$ для двух случайных величин с измерениями $\{x_1, ..., x_M\}$ и $\{y_1, ..., y_M\}$ и двух контейнеров A_J и B_K, рассчитывается следующим образом:

$$p'(a_j, b_k) = \frac{1}{M}\sum_{i=1}^{M} (\theta_j(x_i) \times \theta_k(y_i)) \quad (6)$$

После оценки вероятностей с помощью уравнений (4) и (6), вычисляется *MI* с помощью уравнений (1), (3). В простом методе разбиения на контейнеры, который упоминался выше, каждому непрерывному значению присваивается только один контейнер. Значения рядом с границами контейнера могут быть сдвинуты в соседние контейнеры небольшими колебаниями. Таким образом, результат *MI* сильно зависит от шума. Для того чтобы преодолеть этот недостаток, *Daub* и соавт. [7] ввели способ *B*-сплайна. В способе *Daub'a*, каждое измерение может быть назначено на несколько контейнеров с весом, который определяется *B*-сплайн функцией.

B-сплайн-функция

Используется тот же узловой вектор в B-сплайн функции, как и в [7].Узловой вектор t_i с R контейнерами и порядком сплайна *k* определено в (7). Порядок сплайна k должен удовлетворять $k \in \{1, ..., R - 1\}$

$$t_i = \begin{cases} 0 \text{ if } i < k \\ i - k + 1 \text{ if } k \le i \le R - 1 \\ R - k + 1 \text{ if } i > R - 1 \end{cases} \quad (7)$$

Степень полинома определяется порядком сплайна k. *B*-сплайн функция определяется следующим образом.

$$B_{i,1}(z) = \begin{cases} 1 \text{ if } t_i \le z \le t_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

$$B_{i,k}(z) = B_{i,k-1}(z)\frac{z - t_i}{t_{i+k-1} - t_i} + B_{i+1,k-1}(z)\frac{t_{i+k} - z}{t_{i+k-1} - t_{i+1}} \quad (8)$$

где $i \in [1, R]$ и $z \in [0, R - k + 1]$ является интервалом *B*-сплайн функции. Следует обратить внимание, что непрерывные значения должны быть нормализованы, чтобы вписаться в анте

рвал. Предположим, у есть M непрерывных точек в наборе данных, $x_{1,}$ ..., $x_{M,}$ процедуру нормализации можно выполнить следующим образом.

1. Найти минимальное и максимальное значение x_{min} и x_{max} среди всех точек M.

2. Вычислить нормализованное значение z_i, (i = 1, ..., M) для непрерывного значения с помощью

$$z_i = \frac{(x_i - x_{min}) \times (R-k+1)}{x_{max} - x_{min}} \quad (9)$$

где R - количество контейнеров, используемых в B-сплайн алгоритме, а k - порядок сплайна.

Последовательная оценка взаимной информации на основе B-сплайна и технологии CUDA

В предложеном CUDA-алгоритме используется понятие матрицы весов для каждой данной случайной величины, которая определяется следующим образом.

Определение. Матрица весов (WM): Рассмотрим случайную величину $X = \{x_1, x_2, ..., x_M\}$, R контейнеров $\{a_1, ..., a_R\}$ и B-сплайн функцию порядка k. Матрицей весов для X является матрица $M \times R$, которая обозначаются как $W(X)$, где $W(X)_{i,j}$ содержит весовой коэффициент значения x_i в контейнере a_j т. е. $W(X)_{i,j} = B_{j,k}(z_i)$ где z_i является B-сплайн интервалом нормализованных значений x_i для каждого $1 \le i \le M$ и $1 \le j \le R$.

Из уравнений (1), (8), получаем последовательный алгоритм оценки MI на основе B-сплайнов. Он состоит из двух частей: *WM (X, R, K)* и *Single_MI* (*X, Y, R, k*), которые описаны в алгоритме 1 и алгоритме 2.

Алгоритм 1: *WM (X, R, K)*

Ввод: Случайная величина $X = \{x_1, ..., x_M\}$, число контейнеров R, B-сплайн порядка k

Вывод: *W (X)*

foreach i, $1 \le i \le M$ **do**

Рассчитать нормированную переменную z_i, (i = 1, ..., M) с помощью уравнения 9.

foreach j, $1 \le j \le R$ **do**

Расчет весовых коэффициентов $B_{j,k}(z_i)$ используя уравнения (7) и (8) с нормированным значением z_i ;

end

end

Вывод $W(X)_{i,j} = B_{j,k}(z_i)$

Алгоритм 2: *Single_M I (X, Y, R, K)*

Ввод: Случайные переменные $X = \{x_1, ..., x_M\}$ и $Y = \{y_1, ..., y_M\}$, число контейнеров R, B-сплайн порядка k

Вывод: *MI (X, Y)*

Вызов *WM (X, R, K)*, чтобы получить *W (X);*

Вызов *WM (Y, R, K)*, чтобы получить *W (Y);*

foreach j, $1 \le j \le R$ **do**

Рассчитать вероятность каждого контейнера для X и Y;

$$p_x\ a_j = \frac{1}{M} \sum_{i=1}^{M} W\ X_{i,j}$$

$$p_y\ b_j = \frac{1}{M} \sum_{i=1}^{M} W\ Y_{i,j}$$

end

Вычисление собственной энтропии $H(X)$ и $H(Y)$;

$$H\ X = -\sum_{j=1}^{R} p_x\ a_j \log P_x\ a_j$$

$$H\ Y = -\sum_{j=1}^{R} p_y\ b_j \log P_y\ b_j$$

foreach k, $1 \le k \le R$ **do**

foreach l, $1 \le l \le R$ **do**

Рассчитать взаимные вероятности;

$$p\ a_k, b_i = \frac{1}{M} \sum_{i=1}^{M} (W\ X_{i,k} \times W\ Y_{i,k}) \quad (6)$$

end

end

Рассчитать взаимную энтропию;

$$H\ X,Y = -\sum_{k}^{R} \sum_{i=1}^{R} p\ a_k, b_i \log p\ a_k, b_i$$

Вычислить взаимную информацию с помощью уравнения 3.

Алгоритм *Single_MI* показывает, как рассчитывается *MI* для одной пары случайных величин. На практике, как правило, входными данными выступает большее количество случайных величин, где взаимная информация оценивается для каждой пары переменных [27].

Параллельная оценка взаимной информации на основе B-сплайна и технологии CUDA

До сих пор, параллельная оценка взаимной информации, основанная на B-сплайн функции, была ограничена многопоточностью на многоядерных процессорах [7] и кластерах [8,

23]. Результаты, представленные в работе, показывают, что использование технологии *CUDA* обеспечивает более высокую производительность при разумной стоимости оборудования.

Алгоритм 3: *Multi-GPU* Алгоритм оценки MI на основе *CUDA*

Ввод: *N* молекул, каждая с *M* экспериментами.

Выход: Пары значений MI .

/ * Хост программа выполняется на *CPU* * /

Инициализация параметров, определяющих оценку MI;

Загрузка молекулярных выражений в память каждого устройства GPU и запуск ядра;

/ * Ядро программы выполняется на *GPU* * /;

Вычислить *WM* для каждой молекулы (Kernel 1);

Проверить целостность данных в исходных данных (*Kernel* 2);

Вычислить собственную энтропию для каждой молекулы (*Kernel* 3);

Вычислить совместную энтропию и *MI* значение для каждой молекулярной пары (*Kernel* 4);

/ * Хост программ выполняется на *CPU* * /

Считать результаты на процессор и вывести данные;

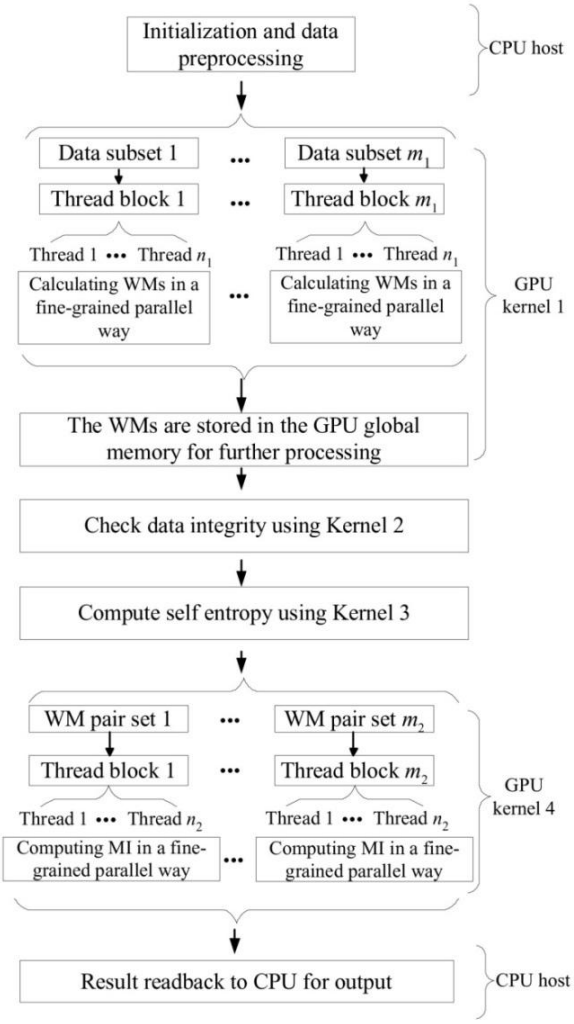

Рис. 1. Алгоритм 1

В Алгоритме 3 приведен псевдокод *MULTI-GPU* алгоритма оценки *MI*. Алгоритм загружает четыре *CUDA* ядра. В *Kernel* 1 молекулярные выражения делятся поровну на подмножества в соответствии с общим числом потоковых блоков. Все потоковые блоки затем выполняются параллельно для вычисления вероятностей локального подмножества молекул с помощью алгоритма 1. Вероятность вхождения в контейнер для каждой молекулы хранятся в матрице весов (*WM*). В *Kernel* 1, потоки, находящиеся в одном и том же блоке, вычисляют матрицу весов по принципу мелкозернистого параллелизма. Перед вычислением собственной энтропии для каждой молекулы с помощью *Kernel* 3, сначала используется *Kernel* 2 для проверки целостности данных. Выполнение *Kernel* 2 необходимо, если есть пустые значения (вызванные отсутствием эксперимента) в молекулярных выражениях. В *Kernel* 4, веса распределяется равномерно на все блоки. Потоки, находящиеся в одном и том же блоке затем вычисляют значение *MI* для каждой молекулярной пары по принципу мелкозернистого параллелизма. Алгоритм изображен на рис. 1.

Ядро 1 используется для расчета матриц весов (*WM*) из молекулярных выражений, которые являются исходными данными. Мы воспользовались естественным параллелизмом матричных операций и спроектировали алгоритм по принципу мелкозернистого параллелизма. В нашем алгоритме, молекулярные выражения сначала делятся поровну на подмножества в соответствии с общим числом потоковых блоков. Каждый потоковый блок затем вычисляет *WM* для выделенного ему подмножества исходных данных. Так как все элементы в одной строке *WM* можно вычислить независимо, *WM* вычисляется параллельно по строкам. В целях ускорения операций ввода / вывода, строки хранятся в высокопроизводительной общей памяти. Размер каждого массива в общей памяти равняется *R*, что равно числу контейнеров. В ядре программы есть *R* потоков, работающих по принципу мелкозернистого параллелизма одновременно для подсчета каждой строки WM. Предполагая, что есть *M* строк в WM, разбиваем WM на мелкие отрезки таким образом, чтобы каждый отрезок мог быть отображен в общей памяти (Рис. 2). На рис 2, размер каждого отрезка $Q \times R$. WM может быть вычислена последовательно отрезок за отрезком.

Рис. 3 показывает, что общее количество $Q \times R$ общей памяти, необходимое для расчета каждого отрезка.

На практике существует $R + k - 1$ потоков, работающих в *Kernel1* для вычисления каждой строки *WM* по принципу мелкозернистого параллелизма. Таким образом, всего выполняется $Q \times (R + k - 1)$ потоков для вычисления *WM* для каждой молекулы. Следует отметить, что вычисление *WM* для каждой молекулы может выполнятся в различных потоковых блоках, потому что максимум потоков для каждого блока в *GPU* ограничен, а размерность *WM* варьируется с количеством экспериментов *M, контейнеров R* и *K* порядка сплайна k.

Причина использования $R + k - 1$ потоков, вместо *R* для вычисления каждой строки *WM* в том, что B-сплайн функция в уравнении (8) определяется рекурсивно. Причина использования $R + k - 1$ потоков, вместо *R* для вычисления каждой строки *WM* в том, что *B*-сплайн функция в уравнении (8) определяется рекурсивно. Для эффективного вычисления каждой строки *WM*, мы развернули рекурсию в уравнении (8) на *k* зависимых шагов.

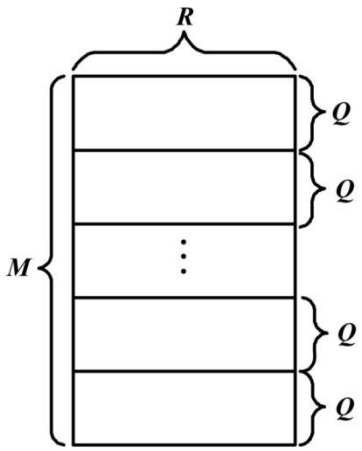

Рис. 2. Пускай есть R контейнеров и M строк в WM. Разобьем WM на мелкие отрезки, так что размер каждого отрезка есть $Q \times R$

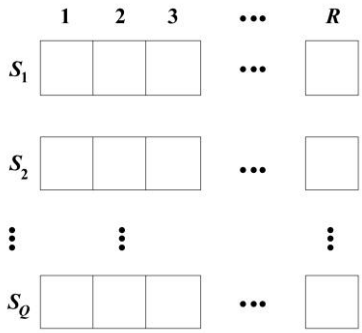

Рис. 3. Есть *R* контейнеров и *Q* строк в отрезке WM. S_1, S_2, ..., S_Q – это массивы в общей памяти, которые хранят строки отрезка. Размерность каждого массива разделяемой памяти массив равен *R* (количество контейнеров).

На каждом шаге $R + k - 1$ потоков работают по принципу мелкозернистого параллелизма, вычисленные значения записываются в массивы в общей памяти. На первом этапе, массивы в общей памяти инициализируются с помощью уравнения (8). В следующем шаге за вычислением каждой ячейки общей памяти следует зависимое отношение на рисунке 4. [27] После того, как последний шаг сделан, первые *R* значений массива в общей памяти - это окончательный результат вычисления текущей строки. Рис. 4 иллюстрирует этот метод для параметров $k = 3$ и $R = 10$.

После вычисления всех *WM*, для проверки целостности данных будет вызван *Kernel* 2. *Kernel* 3 затем выполняет вычисление собственной энтропии для каждого гена. *WM* и собственная энтропия, которые были вычислены в *Kernel* 1 и 3 будут переданы в *Kernel* 4 для вычисления значений *MI* всех пар молекул.

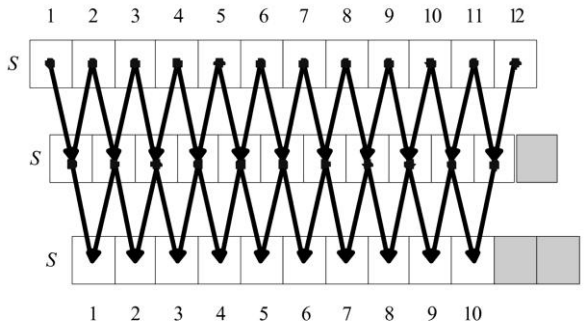

Рис. 4. Иллюстрация развертывания рекурсивной формулы (8) на *k* зависимых шагов, используя $k = 3$ и $R = 10$ для вычисления одной строки WM. *S* является массивом в общей памяти. В каждом шаге, 12 потоков работать параллельно, чтобы вычислить значения S. После 3 шагов, первые 10 значений в *S* являются конечным результатом

Результаты тестирования

Разработанное приложение *MULTI-CUDA-MI* с использованием технологии *CUDA Toolkit* 3.0 с двойной точностью вычислений тестировалось на следующей аппаратной конфигурации: *NVIDIA Tesla C2050*: 1,15 ГГц, 14 мультипроцессоров, 3 Гб *GDDR5* памяти устройства, 48 КБ общей памяти/мультипроцессор. Тестирование было проведено с этой картой, установленной в ПК с процессором *Intel Quad-Core* i7-920 2,66 ГГц с активированными режимами *Turbo Boost* и *Hyper Threading*, 12 Гб ОЗУ под управлением *Linux Fedora* 10 x86_64. Мы использовали *MULTI-CUDA-MI*, чтобы оценить значения *MI* для различных наборов молекулярных выражений. Для моделирования использовалось два вида наборов данных. Первый вид состоит из реальных данных; другой содержит случайным образом сгенерированные наборы при помощи моделирования движений молекул. Это было сделано с целью тестирования масштабируемости алгоритма. В экспериментах использовались параметры по умолчанию $R = 10$ и $K = 3$ для g_corellation и *MULTI-CUDA-MI*. Оба приложения используют двойную точность вычислений.

В табл. 1 показана производительность во время выполнения g_corellation и *MULTI-CUDA-MI* при обработки различных наборов молекулярных выражений.

Таблица 1. Сравнение времени выполнения (в секундах) между *g_corellation* и *MULTI-CUDA-MI* для разных наборов данных

Число молекул	***g-correlation (4 threads)***	***MULTI-CUDA-MI (4 GPUs)***	**Ускорение**
2048	220,87	5,09	43,34
4096	867,35	14,57	59,53
8192	5960,87	41,13	144,91
10240	34315,69	152,97	224,32

Рис. 4. Сравнение времени выполнения (в секундах) между g_corellation и *MULTI-CUDA-MI* для разных наборов данных

Таблица 2. Сравнение времени работы (в секундах) для обработки данных размерностью 4096 с различными параметрами.

Количество контейнеров	**Порядок сплайна**	**g-*correlation* (*4 threads*)**	***MULTI-CUDA-MI (4 GPUs*)**	**Ускорение**
10	3	867,35	14,57	59,53
10	4	867,24	15,82	54,84
10	5	868,3	15,94	54,48
15	3	1926,67	20,55	93,73
15	4	1947,45	20,83	93,48
15	5	1949,65	21,15	92,16
20	3	3468,07	35,77	96,95
20	4	3657,64	36,10	101,33
20	5	3658,83	36,17	101,16

Из табл. 1 видно, что *MULTI-CUDA-MI* выигрывает в скорости в 224 раза по сравнению с g-*correlation*. Из-за большого потребления памяти, в наборах 8192 и 10240 время вычислений перекрывает время передачи данных, поэтому было достигнуто большее ускорение. Из таблицы 1 видно, что ускорение *MULTI-CUDA-MI* возрастает с увеличением числа молекул. Причины тому следующие: во-первых, для большого числа молекул выше арифметическая интенсивность; во-вторых, накладные расходы на инициализацию ядра уменьшаются на больших наборах данных. В табл. 2 представлено время выполнения g_*corellation* и *MULTI-CUDA-MI* для размерности 4096 и различных параметров. Величина порядка сплайна варьируется от 3 до 5, а число контейнеров колеблется от 10 до 20. Проанализировав результаты таблицы 2 можно сделать следующие заключения:

1. Ускорение существенно не меняется при увеличении порядка сплайна.

2. Ускорение значительно возрастает с увеличением количества контейнеров.

Таким образом порядок сплайна не имеет большого влияния на общее время выполнения алгоритма оценки MI. Поскольку в MULTI-CUDA-MI используется $R \times R$ потоков, то с большим количеством контейнеров выполняется большее количество потоков. Следовательно, MULTI-CUDA-MI работает эффективнее с большим количеством контейнеров R.

Выводы

В работе предложен Multi-GPU алгоритм MULTI-CUDA-MI на основе технологии CUDA для ускорения оценки MI с использованием B-сплайн функции. Для того, чтобы использовать возможности GPU для ускорения оценки MI, была использована быстрая общая память, мелкозернистый параллелизм, и эффективное разбиение данных для реализации алгоритма. Представленная реализация обеспечивает увеличение скорости до 244 по сравнению с многопоточной g_*corellation* версией на современных *Intel Quad-Core* процессорах. Полученный результат показывает, что архитектура *CUDA* является достаточно эффективной аппаратной платформой для данного типа вычислений. Помимо прочего был использован вывод *MULTI-CUDA-MI* для создания корреляционных матриц траекторий движения молекул. Эксперименты показали, что по сравнению с g_*corellation*, *MULTI-CUDA-MI* имеет лучшие характеристики производительности.

Список литературы

1. Fraser AM, Swinney HL: Independent coordinates for strange attractors from mutual information. Physical Review A 1986, 33:2318-2321.

2. Pluim JPW, Maintz JBA, Viergever MA: Mutual-information-based registration of medical images: a survey. IEEE Transactions on Medical Imaging 2003, 22:986-1004. PubMed

3. Arsic I, Thiran JP: Mutual information eigenlips for audio-visual speech recognition. Proc 14th Eur Signal Processing Conf (EUSIPCO) 2006.

4. Zhou X, Wang X, Dougherty ER: Construction of genomic networks using mutual-information clustering and reversible-jump Markov-chain-Monte-Carlo predictor design. Signal Processing 2003, 83:745-761.

5. Zhou X, Wang X, Dougherty ER, Russ D, Suh E: Gene Clustering Based on Clusterwide Mutual Information. Journal of Computational Biology 2004, 11:147-161.

6. Daub CO, Steuer R, Selbig J, Kloska S: Estimating mutual information using B-spline functions-an improved similarity measure for analysing gene expression data. 2004., 5:

7. Zola J, Aluru M, Aluru S: Parallel information theory based construction of gene regulatory networks. Hipc 2008, 336-349.

8. Butte AJ, Kohane IS: Mutual information relevance networks: functional genomic clustering using pairwise entropy measurements. Pacific Symposium on Biocomputing 2000, 415-426.

9. Schafer J, Strimmer K: An empirical Bayes approach to inferring large-scale gene association networks. Bioinformatics 2005, 21(6):754-764.

10. D'Haeseleer P, Wen X, Fuhrman S, Somogyi R: Mining the gene expression matrix: Inferring gene relationshops from large scale gene expression data. Second International Workshop on Information Processing in Cells and Tissues 1998, 203-212.

11. Friedman N, Linial M, Nachman I, Pe'er D: Using Bayesian networks to analyze expression data. Journal of Computational Biology 2000, 7:601-620.

12. Chen X, Chen M, Ning K: BNArray: an R package for constructing gene regulatory networks from microarray data by using Bayesian network. Bioinformatics Application Note 2006, 22:2952-2954.

13. Margolin AA, Nemenman I, Basso K, Wiggins C, Stolovitzky G, Favera RD, Califano A: ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context. BMC Bioinformatics 2006., 7(S7):

14. Wilson J, Dai M, Jakupovic E, Watson S, Meng F: Supercomputing with toys: harnessing the power of NVIDIA 8800GTX and playstation 3 for bioinformatics problems. Comput Syst Bioinformatics Conf 2007, 387-390.

15. Lindholm E, Nickolls J, Oberman S, Montrym J: NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 2008, 28:40-52.

16. NVIDIA: NVIDIAFermiArchitecture. [http://www.NVIDIA.com/object/fermi_architecture.html] webcite

17. Manavski SA, Valle G: CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics 2008., - 98 c.

18. Schatz MC, Trapnell C, Delcher AL, Varshney A: High-throughput sequence alignment using graphics processing units. BMC Bioinformatics 2007, - P. 143-155.

19. Liu Y, Maskell DL, Schmidt B: CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. 2(73) BMC Research Notes 2009. - P. 139-141.

20. Liu Y, Schmidt B, Maskell DL: CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Research Notes 2010., 3(93), - P. 98-107.

21. Liu W, Schmidt B, Voss G, Muller-Wittig W: Accelerating Moleculer Dynamics simulations using Graphics Processing Units with CUDA. Computer Physics Communications 2008, - 367 p.

22. Zola J, Aluru M, Sarje A, Aluru S: Parallel Information Theory Based Construction of Genome-wide Gene Regulatory Networks. IEEE ansactions on Parallel and Distributed Systems 2010, - P. 1721-1733.

23. CUDA N: NVIDIA CUDA C Programming Guide Version 3.1.1. 2010.

24. Den Bulcke TV, Leemput KV, Naudts B, van Remortel P, Ma H, Verschoren A, Moor BD, Marchal K: SynTReN: a generator of synthetic gene expression data for design and analysis of structure learning algorithms. BMC Research Notes 2011, - 253 p.

25. Haixiang Shi, Bertil Schmidt, Weiguo Liu and Wolfgang Muller-Wittig: Parallel mutual information estimation for inferring gene regulatory networks on GPUs. BMC Research Notes 2011, - P. 243-253.

26. Alexander Kraskov, Harald Stögbauer, and Peter Grassberger: Estimating mutual information. Phys. Rev. E 69, 066138 (2004), - P. 134-149.

27. П.Грубый, С. Стиренко: Multi-GPU алгоритм оценки взаимной информации для моделирования молекулярной динамики. High performance computing HPC-UA’2012, - P. 158-163.

O.F. Lange, H. Grubmuller, “Generalized Correlation for Biomolecular Dynamics,” PROTEINS: Structure, Function, and Bioinformatics, 62, 2006, - P. 1053-1061.