УДК 004.272.2 519.245

# РЕАЛИЗАЦИЯ ПОСЛЕДОВАТЕЛЬНОГО МЕТОДА МОНТЕ-КАРЛО В СИСТЕМАХ С МАССОВЫМ ПАРАЛЛЕЛИЗМОМ ДЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ В ОПТИЧЕСКОЙ КОГЕРЕНТНОЙ ТОМОГРАФИИ

© 2015 г. **П. С. Скаков**, канд. техн. наук; **И. П. Гуров**, доктор техн. наук

Университет ИТМО, Санкт-Петербург

E-mail: pavelsx@gmail.com

Рассмотрены особенности использования систем с массовым параллелизмом на примере графических процессоров для эффективной реализации последовательного метода Монте-Карло (ПММК) в случае обработки данных в оптической когерентной томографии. Проведено исследование зависимости быстродействия реализаций ПММК в различных программных и аппаратных конфигурациях. Показана целесообразность применения платформы OpenCL для реализации ПММК, в том числе при использовании только процессора общего назначения. Показана эффективность вычислений на графических процессорах, что позволило провести обработку за 30 с трехмерного томографического изображения размером 1280×1022×376 отсчетов последовательным методом Монте-Карло.

***Ключевые слова:*** *последовательный метод Монте-Карло, GPGPU, анализ интерферометрических сигналов.*

Коды OCIS: 100.2000 100.6950.

*Поступила в редакцию 20.03.2015.*

## Введение

Обработка данных в корреляционной оптической когерентной томографии (ОКТ) требует оценивания огибающей сигнала на основе зарегистрированных значений интенсивности интерференционных полос малой когерентности. Одним из вариантов получения этих оценок при регистрации набора видеокадров является последовательный метод Монте-Карло (ПММК) [1, 2]. Данный метод обладает рядом достоинств по сравнению с другими подходами, например, основанными на использовании вариантов фильтра Калмана [3], а именно: устойчивость к существенным локальным нелинейностям и невысокие требования к точности задания модели. Тем не менее метод не лишен недостатков. Самым заметным из них при практическом применении обычно оказывается существенно бóльшее время обработки данных, вызванное большим объемом необходимых вычислений, что приводит к времени обработки томографического изображения среднего размера (1280×1022×376) порядка нескольких часов обычными реализациями ПММК. В данной статье рассматривается возможность существенного уменьшения времени обработки ПММК за счет эффективного использования систем с массовым параллелизмом.

## Последовательный метод Монте-Карло

В системах ОКТ значения интерферометрического сигнала, как известно [4], могут быть определены выражением

$$s_k = A_k \cos\Phi_k + B_k + \upsilon_k, \qquad (1)$$

где $k = 0,\ldots, K-1$ – номер дискретного отсчета, $s_k$ – наблюдаемое значение интенсивности, $A_k$ – амплитуда, $\Phi_k$ – фаза, $B_k$ – фоновая составляющая, $\upsilon_k$ – шум наблюдения.

Эволюцию вектора параметров $\mathbf{w}_k = (A_k, \Phi_k, B_k)^{\mathrm{T}}$ можно записать в виде

$$\mathbf{w}_{k+1} = \mathbf{w}_k + (0, \Delta\Phi_{k+1}, 0)^{\mathrm{T}} + \mathbf{u}_{k+1}, \qquad (2)$$

где $\Delta\Phi_k$ – приращение фазы на шаге $k$, $\mathbf{u}_k$ – шум системы.

В общем случае динамическая система задается уравнениями системы и наблюдения

$$\mathbf{w}_{k+1} = \mathbf{g}_k(\mathbf{w}_k, \mathbf{u}_{k+1}), \quad (3)$$

$$\mathbf{s}_k = \mathbf{h}_k(\mathbf{w}_k, \mathbf{v}_k), \quad (4)$$

где $\mathbf{g}_k$ и $\mathbf{h}_k$ – известные нелинейные дифференцируемые векторные функции. Предполагается, что плотность вероятности распределения параметров в начальный момент времени $p(\mathbf{w}_0)$ известна.

Последовательность действий алгоритма ПММК с мультиоблачным этапом повторной выборки [5] представлена на рисунке. Входными данными являются набор из $N$ апостериорных оценок значений параметров, полученный на предыдущем шаге ($\hat{\mathbf{w}}_{k-1,i}$, $i = 0,..., N-1$) и наблюдаемое значение сигнала $\mathbf{s}_k$. Результатом является набор апостериорных оценок значений параметров для текущего шага $\hat{\mathbf{w}}_{k,i}$.

На первом этапе генерируется $M$ априорных оценок $\tilde{\mathbf{w}}_{k,j}$. В случае $M$, кратного $N$, это может быть просто реализовано генерацией $M/N$ априорных оценок $\tilde{\mathbf{w}}_{k,j}$ для каждой апостериорной оценки $\hat{\mathbf{w}}_{k-1,i}$.

На втором этапе по априорным оценкам $\tilde{\mathbf{w}}_{k,j}$ строятся предсказания значений сигнала $\tilde{\mathbf{s}}_{k,j}$.

На третьем этапе для каждой априорной оценки $\tilde{\mathbf{w}}_{k,j}$ вычисляется расстояние $\Delta_{k,j}$ между предсказанным $\tilde{\mathbf{s}}_{k,j}$ и зарегистрированным $\mathbf{s}_k$ значениями сигнала. Для интерферометрических сигналов обычно используется евклидово расстояние, но возможны и более сложные метрики.

На четвертом этапе из $M$ априорных оценок $\tilde{\mathbf{w}}_{k,j}$ выбирается $N$ $\hat{\mathbf{w}}_{k,i}$, соответствующие результату апостериорной оценки на шаге $k$, на основе критерия минимальности вычисленного расстояния $\Delta_{k,j}$.

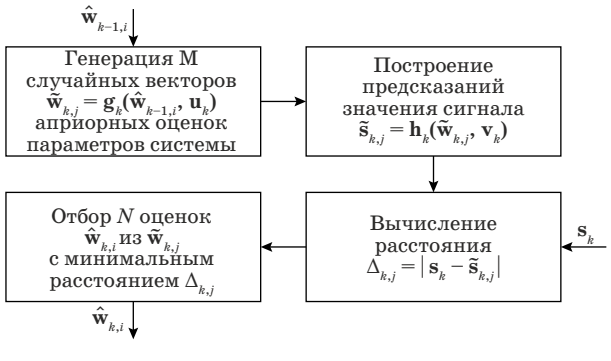


Схема алгоритма динамического оценивания параметров последовательным методом Монте-Карло с мультиоблачным этапом повторной выборки.

Уравнения (1)–(4) записаны для одномерного сигнала. В ОКТ обычно требуется обработка трехмерных данных с применением как описанной одномерной модели независимо по отдельным линиям данных, так и путем построения аналогичных двумерных и трехмерных моделей фильтрации [6].

## Последовательный метод Монте-Карло в системах с массовым параллелизмом

Анализ алгоритма ПММК позволяет заключить, что обработка данных ОКТ таким методом хорошо поддается распараллеливанию: количество независимых потоков вычислений имеет порядок сотен и более. При этом возможна параллельность двух видов

– параллельность внутри шага ПММК, связанная с построением множества априорных оценок $M$ (порядка сотни);

– параллельность применения метода ПММК с одномерной или двумерной моделями к трехмерным данным ОКТ (порядка нескольких сотен по каждому дополнительному измерению).

Большое количество возможных независимых потоков вычислений позволяет поставить вопрос о возможности более эффективного вычисления ПММК в системах с массовым параллелизмом по сравнению с традиционными системами с максимизацией скорости выполнения небольшого числа потоков.

Наиболее распространенным и доступным видом систем с массовым параллелизмом являются графические процессоры (GPU, видеокарты). Рассмотрим особенности таких систем и соответствующие модификации алгоритма ПММК, позволяющие их эффективно использовать.

Первым требованием эффективного выполнения на GPU является наличие достаточного количества независимых потоков вычислений (порядка нескольких тысяч, а лучше – нескольких десятков тысяч). Данное требование не может быть обеспечено только распараллеливанием вычислений внутри одного шага ПММК, что неприемлемо для трехмерной фильтрации, или использованием только параллельности обработки по одной координате, что обуславливает целесообразность двумерной фильтрации в прямой зависимости от эффективности распараллеливания одного шага ПММК. Одномерная фильтрация легко удовлетворяет данному требованию.

Вторым требованием эффективности является малое относительно общего объема вычислений количество обращений к глобальным данным (таким, которые доступны всем потокам). ПММК удовлетворяет данному требованию. При этом обработка после получения всех данных в системе ОКТ будет более эффективной, чем обработка параллельно со сканированием, так как в этом случае в процессе вычислений не будет необходимости сохранения довольно большого состояния каждого потока (набора векторов $\hat{\mathbf{w}}_{k,i}$) в глобальной памяти.

Здесь также возможна дополнительная оптимизация реализации ПММК в случае отсутствия распараллеливания вычислений внутри шага. Так как в результате шага из $M$ генерируемых априорных оценок $\tilde{\mathbf{w}}_{k,j}$ необходимо отобрать лишь $N$ апостериорных оценок $\hat{\mathbf{w}}_{k,i}$ ($M$ гораздо больше $N$) с минимальными значениями $\Delta_{k,j} = |\mathbf{s}_k - \tilde{\mathbf{s}}_{k,j}|$, то можно не сохранять значения $\tilde{\mathbf{w}}_{k,j}$, а обновлять множество $\hat{\mathbf{w}}_{k,i}$ непосредственно в процессе генерации априорных оценок. Данный подход позволяет настолько снизить объем рабочих переменных каждого потока, что они все могут разместиться в регистрах GPU, что влечет за собой резкое увеличение эффективности использования вычислительных устройств.

Существенным ограничением вычислений на GPU выступает объем доступной глобальной памяти, составляющий обычно несколько гигабайт. В сочетании с большим количеством отсчетов, образующих типичные данные ОКТ (порядка миллиарда), это приводит к необходимости максимально компактного представления входных и выходных данных, для чего целесообразно использовать 8–16-битные целые числа в соответствии с требуемой точностью регистрируемых данных.

Характерной чертой ПММК, как одного из представителей методов группы Монте-Карло, является необходимость генерирования большого объема случайных чисел. При обработке данных в ОКТ адекватной моделью можно считать случайные числа, имеющие нормальное распределение. Важной особенностью вычислений на GPU в данном случае является высокая скорость выполнения многих операций, которые на процессорах общего назначения требуют существенных временны́х затрат, таких как тригонометрические операции, вычисление квадратного корня и логарифмов. В сочетании с выгодой минимизации обращений к глобальной памяти это может привести к существенному изменению скорости выполнения различных алгоритмов генерации случайных чисел относительно друг друга и необходимости выбора алгоритма в зависимости от среды выполнения.

## Экспериментальные результаты

В качестве тестовой системы использовался процессор Intel i7-930 (4 ядра, 8 потоков), работающий на фиксированной частоте 4 ГГц, предельная теоретическая вычислительная мощность которого составляет 128 Gflops. Использовался 64-битный код. Выполнялась обработка интерферометрических трехмерных данных размером 1280×1022×376 отсчетов методом ПММК с параметрами $N = 10$, $M = 200$. Распараллеливание вычислений производилось в результате одномерной обработки многомерных данных. В таблицах приведено среднее время обработки одного B-скана (размер 1280×376), вычисленное как время обработки всех данных, разделенное на количество B-сканов (1022).

При сравнении быстродействия реализаций алгоритмов на процессорах общего назначения (CPU) и системах с массовым параллелизмом (GPU), к сожалению, сравнение нередко проводится не вполне корректно. Сравниваются примитивная реализация для CPU и оптимизированная для GPU, в результате чего прирост скорости может быть существенно завышен. Поэтому вначале необходимо оценить быстродействие различных реализаций ПММК на CPU. Соответствующие оценки приведены в табл. 1.

ПММК на CPU исследовался в следующих реализациях:

- MS Base – базовая реализация ПММК на языке C без использования многопоточности, компилятор Visual Studio 2013 update 4.
- MS OpenMP – код MS Base модифицирован для параллельного вычисления по дополнительным размерностям обрабатываемых данных на основе OpenMP.

**Таблица 1.** Время обработки одного B-скана различными реализациями ПММК на CPU

| Реализация | MS Base | MS OpenMP | ICC OpenMP | MS Zig | ICC Zig |
|---|---|---|---|---|---|
| Время, мс | 12056 | 2805 | 2109 | 1692 | 697 |

• ICC OpenMP – аналогично MS OpenMP, но компилятор Intel C++ Compiler XE 15.0 (ICC).

• MS Zig – код ПММК аналогичен MS OpenMP, но для генерации случайных чисел с нормальным распределением вместо стандартного алгоритма Marsaglia polar method [7] используется Ziggurat algorithm [8].

• ICC Zig – аналогично MS Zig, но компилятор ICC.

Таким образом, хорошо видно, что на скорость выполнения ПММК существенно влияет не только распараллеливание вычислений, но и метод генерации случайных чисел и даже используемый компилятор. При этом самый лучший вариант превосходит по скорости базовую реализацию более чем в 17 раз.

Для выполнения на системах с массовым параллелизмом алгоритм ПММК был реализован на OpenCL 1.1.

Интересной особенностью OpenCL является возможность выполнения такого кода на обычном процессоре, а не только специальными системами с массовым параллелизмом, результаты чего приведены в табл. 2.

Исследовалось выполнение ПММК на следующих конфигурациях OpenCL:

• CL AMD – выполнение на платформе AMD Accelerated Parallel Processing, OpenCL 1.2 AMD-APP.

• CL AMD Zig – аналогично CL AMD, но используется Ziggurat algorithm.

• CL Intel – выполнение на платформе Intel(R) OpenCL, OpenCL 1.2.

• CL Intel Zig – аналогично CL Intel, но используется Ziggurat algorithm.

OpenCL вариант ПММК при выполнении на CPU показывает примерно то же быстродействие, что и обычный код на языке C с использованием OpenMP, и даже немного выигрывает. Интересна зависимость быстродействия различных методов генерации случайных чисел от выбранной OpenCL платформы. Ziggurat algorithm оказался быстрее на платформе AMD, но на Intel он существенно медленнее, чем Marsaglia polar method.

**Таблица 2.** Время обработки одного B-скана различными OpenCL реализациями ПММК при выполнении на CPU

| Реализация | CL AMD | CL AMD Zig | CL Intel | CL Intel Zig |
|---|---|---|---|---|
| Время, мс | 1106 | 776 | 687 | 995 |

Для выполнения OpenCL версии ПММК на GPU использовались следующие видеокарты:

1. Radeon HD6950. 1536 вычислительных ядер на частоте 850 МГц, 2611 Gflops.

2. Radeon HD6990. 1536×2 вычислительных ядер на частоте 880 МГц, 5407 Gflops.

3. Quadro 600. 96 вычислительных ядер на частоте 1280 МГц, 246 Gflops.

4. GeForce GTX 590 (half). 512 вычислительных ядер на частоте 1215 МГц, 1244 Gflops.

5. GeForce GTX 590. 512×2 вычислительных ядер на частоте 1215 МГц, 2488 Gflops.

Выполнение ПММК исследовалось на следующих конфигурациях OpenCL:

• R1 – выполнение на Radeon HD6950.

• R1 Zig – выполнение на Radeon HD6950, Ziggurat algorithm.

• R2 – выполнение совместно на Radeon HD6950 и Radeon HD6990.

• NV1 – выполнение на Quadro 600.

• NV2 – выполнение на GeForce GTX 590 (half).

• NV3 – выполнение на GeForce GTX 590.

• NV3 Zig – выполнение на GeForce GTX 590, Ziggurat algorithm.

Результаты по выполнению на различных конфигурациях GPU представлены в табл. 3. Здесь, помимо времени обработки одного B-скана, приведены также значения “эффективности” использования аппаратных ресурсов – величины, обратной произведению теоретической вычислительной мощности использованной конфигурации на время обработки. Сравнивая эффективность, можно сделать следующие выводы:

1. При одновременном выполнении на нескольких GPU (конфигурации R2 и NV3) эффективность изменяется незначительно относительно аналогичных конфигураций с одним GPU (конфигурации R1 и NV2 соответственно), что означает линейную масштабируемость по дополнительным вычислительным устройствам.

**Таблица 3.** Время обработки одного B-скана различными OpenCL реализациями ПММК при выполнении на GPU

| Реализация | R1 | R1 Zig | R2 | NV1 | NV2 | NV3 | NV3 Zig |
|---|---|---|---|---|---|---|---|
| Время, мс | 121 | 130 | 40 | 362 | 59 | 30 | 44 |
| Эффективность, $Tflops^{-1}c^{-1}$ | 3,2 | 2,9 | 3,1 | 11,2 | 13,6 | 13,5 | 9,1 |

**Таблица 4.** Сводная таблица времен обработки одного B-скана ПММК конфигураций с лучшим быстродействием

| Реализация | ICC Zig | CL Intel | R2 | NV3 |
|---|---|---|---|---|
| Время, мс | 697 | 687 | 40 | 30 |
| Эффективность, $Tflops^{-1}c^{-1}$ | 11,2 | 11,4 | 3,1 | 13,5 |

2. Данная реализация ПММК использует вычислительные возможности рассмотренных видеокарт NVIDIA архитектуры Fermi существенно полнее, чем рассмотренных видеокарт AMD архитектуры VLIW4, что при примерно равных пиковых вычислительных мощностях (GeForce GTX 590 и Radeon HD6950) дает первым значительное превосходство по времени выполнения.

3. Ziggurat algorithm во всех случаях проигрывает стандартному Marsaglia polar method (приведены только два результата для принципиально разных конфигураций).

Сводные результаты различных реализаций и конфигураций выполнения ПММК с лучшим быстродействием приведены в табл. 4.

## Заключение

Реализация ПММК на OpenCL для систем с массовым параллелизмом не только позволяет достичь существенного ускорения обработки при использовании видеокарт (в 17–23 раза), но и равна по скорости лучшей стандартной реализации на C при выполнении на процессоре общего назначения, что позволяет рекомендовать ее к использованию в любом случае, даже на системах без возможности вычисления на видеокартах.

OpenCL реализация линейно масштабируется по скорости при использовании нескольких OpenCL устройств, что позволяет легко повысить быстродействие системы установкой дополнительных видеокарт при возникновении такой потребности.

В заключение важно отметить, что время обработки полного тестового массива данных ОКТ при использовании лучшей рассмотренной конфигурации NV3 составляет около 30 с, что позволяет говорить о преодолении недостатка ПММК, заключающегося в длительной обработке данных традиционными реализациями метода.

Работа выполнена при поддержке Министерства образования и науки Российской Федерации.

* * * * *

*ЛИТЕРАТУРА*

1. *Волынский М.А.*, *Гуров И.П.*, *Ермолаев П.А.*, *Скаков П.С.* Динамическое оценивание параметров интерферометрических сигналов на основе последовательного метода Монте-Карло // Научно-технический вестник информационных технологий, механики и оптики. 2014. № 3 (91). С. 18–24.

2. *Волынский М.А.*, *Гуров И.П.*, *Ермолаев П.А.*, *Скаков П.С.* Исследование биологических объектов в оптической когерентной томографии с обработкой данных последовательным методом Монте-Карло // Научно-технический вестник информационных технологий, механики и оптики. 2014. № 4 (92). С. 23–28.

3. *Simon D.* Optimal state estimation. N.Y.: John Wiley & Sons, Inc., 2006. 526 p.

4. *Гуров И.П.* Оптическая когерентная томография: принципы, проблемы и перспективы // Проблемы когерентной и нелинейной оптики / Под ред. Гурова И.П., Козлова С.А. СПб.: СПбГУ ИТМО, 2004. С. 6–30.

5. *Волынский М.А.*, *Гуров И.П.*, *Скаков П.С.* Рекуррентный алгоритм обработки интерферометрических сигналов на основе мультиоблачной модели предсказания // Научно-технический вестник информационных технологий, механики и оптики. 2014. № 4 (92). С. 18–22.

6. *Скаков П.С.* Динамическое оценивание параметров интерферометрических систем и сигналов на основе последовательного метода Монте-Карло // Автореф. канд. дис. СПб.: ИТМО, 2014. 19 с.

7. *Marsaglia G.*, *Bray T.A.* A convenient method for generating normal variables // SIAM Review. 1964. V. 6. Is. 3. P. 260–264.

8. *Marsaglia G.*, *Tsang W.W.* The ziggurat method for generating random variables // Journal of Statistical Software. 2000. V. 5. Is. 8. P. 1–7.