УДК 004.021

ПРИМЕНЕНИЕ ГРАФИЧЕСКОГО ПРОЦЕССОРА В ЗАДАЧЕ МНОГОКРАТНОГО ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ НА ОСНОВЕ ОБОБЩЕННОЙ ФОРМУЛЫ СИМПСОНА

С. А. Запрягаев, А. А. Карпушин

Воронежский государственный университет

Поступила в редакцию 29.03.2011 г.

Аннотация. В статье рассматривается возможность использования графического процессора в задаче численного интегрирования пространственно заданной функции на основе обобщения квадратурной формулы Симпсона. Проводится сравнительный анализ вариантов реализаций, их достоинства и недостатки.

Ключевые слова. Параллельные вычисления, численное интегрирование на графическом процессоре, OpenCL.

Annotation. The article describes advantages and disadvantages of graphics processing unit usage in numerical integration of some three-dimensional function, based on Simpson's rule. Several different implementations are investigated in terms of their limitations and drawbacks

Keywords: Numerical integration on GPU, Simpson's rule, GPGPU, OpenCL.

ВВЕДЕНИЕ

Задача численного интегрирования функции $f(x, y, z)$ заданной в трехмерном пространстве является важной практической проблемой для многих известных приложений. Одним из классов таких прикладных задач, где проблема вычисления кратных интегралов возникает достаточно регулярно, являются задачи проведения различных квантово-механических расчетов. Так, например, для определения полного сечения рассеяния электронов на сложном молекулярном кластере, требуется выполнять процедуру кратного численного интегрирования на большом множестве данных десятки тысяч раз [1]. Исходными данными для задачи вычисления кратного интеграла являются значения функции, заданные в дискретных точках пространства. Если шаг между соседними точками, в которых задана функция, является постоянным по каждой из координатных осей, то подходящим методом численного интегрирования является метод Симпсона, обобщенный на трехмерный случай. При этом, если число значений функции в исходном множестве данных является достаточно большим, время выполнения процедуры численного интегрирования на центральном процессоре может оказаться существенно большим для решения практических задач.

Между тем, широкая потребность реализации высококачественной, интерактивной трехмерной графики привела в последние годы к существенному технологическому развитию графических процессоров (GPU), являющихся неотъемлемой частью любого персонального компьютера. Графический процессор приобрел качество высокопроизводительного устройства, основанного на применении параллельных технологий. При этом современный графический процессор предоставляет возможность осуществлять программирование обработки исходных данных на уровне прямых команд графического процессора.

Возможность программировать GPU привела к использованию его для решения широкого класса задач, несвязанных с графикой непосредственно. Такое использование графического процессора получило название GPGPU (General Purpose computations on Graphics Processing Unit). Типичными областями применения GPGPU стали: видеообработка [2], вычислительная химия [3], визуализация в медицине [4], обработка сигналов [5] и др.

Результатом развития GPU и его использования в задачах, несвязанных с компьютерной графикой, стала разработка единого стандарта описания гетерогенных вычислений на высокопараллельных системах – OpenCL (Open Computational Language).

OpenCL позволяет описывать вычисления, абстрагируясь от конкретного устройства, на котором эти вычисления могут быть исполнены. Так, алгоритмы, написанные с использованием OpenCL, могут исполняться на нескольких ядрах центрального процессора, или на графическом процессоре, или на процессорах IBM Cell/B.E.

В настоящей работе предлагаются ряд алгоритмов численного интегрирования по методу Симпсона для графических процессоров с использованием библиотеки OpenCL. Алгоритмы реализованы на языке Python с применением дополнительных программных пакетов NumPy и PyOpenCL. Проводится сравнительный анализ алгоритмов, определяются возможности применения каждого из алгоритмов, их достоинства и недостатки.

1. ОБОБЩЕНИЕ МЕТОДА СИМПСОНА НА ТРЕХМЕРНЫЙ СЛУЧАЙ

Метод Симпсона использует приближение подынтегральной функции интерполяционным многочленом второй степени [13]:

$$\int_a^b f(x)dx \approx \approx \frac{h}{3}\left[f(x_0)+4f\left(x_1\right)+f(x_2)\right], \tag{1}$$

где $h=\frac{b-a}{2}$, $x_0=a$, $x_i=x_0+h\cdot i, \quad i=0,1,2.$

Для обобщения метода Симпсона на трехмерный случай используется последовательное применение формулы (1) для каждой из переменных интегрирования:

$$\int_{a_z}^{b_z}\int_{a_y}^{b_y}\int_{a_x}^{b_x} f(x,y,z)dxdydz \approx \frac{h_x}{3}\times \times\int_{a_z}^{b_z}\int_{a_y}^{b_y}(f(x_0,y,z)+4f(x_1,y,z)+ +f(x_2,y,z))dydz, \tag{2}$$

В результате

$$\int_{a_z}^{b_z}\int_{a_y}^{b_y}\int_{a_x}^{b_x} f(x,y,z)dxdydz \approx \approx \frac{h_x h_y h_z}{27}\sum_{i=0}^{2}\sum_{j=0}^{2}\sum_{k=0}^{2} A_{j,k}^{(i)}\times \times f\left(x_i,y_j,z_k\right), \tag{3}$$

где $A^{(0)}=\begin{pmatrix}1&4&1\\4&16&4\\1&4&1\end{pmatrix}$, $A^{(1)}=\begin{pmatrix}4&16&4\\16&64&16\\4&16&4\end{pmatrix}$, $A^{(2)}=\begin{pmatrix}1&4&1\\4&16&4\\1&4&1\end{pmatrix}$.

Геометрический смысл матриц $A^{(i)}$ представлен на рис. 1.

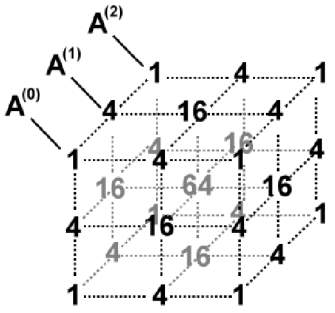

Рис. 1. Геометрическая интерпретация матриц $A^{(i)}$.

В случае, если исходный интервал данных разбит на четное количество отрезков равной длины, метод Симпсона применяется отдельно для каждой пары отрезков. Результат суммирования для конечного числа интервалов имеет вид:

$$\int_{a_z}^{b_z}\int_{a_y}^{b_y}\int_{a_x}^{b_x} f(x,y,z)dxdydz \approx \approx \frac{h_x h_y h_z}{27}\sum_{i=0}^{2I}\sum_{j=0}^{2J}\sum_{k=0}^{2K} A_{j,k}^{(i)}\cdot f\left(x_i,y_j,z_k\right), \tag{4}$$

где I,J,K – количество интервалов интегрирования по каждой из координатных осей,

$$A^{(0)}=A^{(2I-1)}=\begin{pmatrix}1&4&2&4&\cdots&1\\4&16&8&16&\cdots&4\\2&8&4&8&\cdots&2\\4&16&8&16&\cdots&4\\\vdots&\vdots&\vdots&\vdots&\ddots&\vdots\\1&4&2&4&\cdots&1\end{pmatrix},$$

$$A^{(2n+1)}=\begin{pmatrix}4&16&8&16&\cdots&4\\16&64&32&64&\cdots&16\\8&32&16&32&\cdots&8\\16&64&32&64&\cdots&16\\\vdots&\vdots&\vdots&\vdots&\ddots&\vdots\\4&16&8&16&\cdots&4\end{pmatrix},$$

$$A^{(2n)} = \begin{pmatrix} 2 & 8 & 4 & 8 & \cdots & 2 \\ 8 & 32 & 16 & 32 & \cdots & 8 \\ 4 & 16 & 8 & 16 & \cdots & 4 \\ 8 & 32 & 16 & 32 & \cdots & 8 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 8 & 4 & 8 & \cdots & 2 \end{pmatrix}.$$

Коэффициенты $A_{j,k}^{(i)}$ могут быть вычислены по следующей формуле:

$$A_{j,k}^{(i)} = a_i a_j a_k, \qquad (5)$$

где $a_i = 1$ для граничных значений индекса i, $a_i = 2$ для нечетных значений индекса i, $a_i = 4$ для четных значений индекса i.

2. АЛГОРИТМ ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ ПО МЕТОДУ СИМПСОНА НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

Как следует из формулы (4), вычисление приближенного значения искомого интеграла по методу Симпсона может быть разбито на два этапа.

Первый этап заключается в вычислении произведений вида $A_{j,k}^{(i)} \cdot f\left(x_i, y_j, z_k\right)$ для всех исходных точек $\left(x_i, y_j, z_k\right)$. Этот этап не использует зависимости между отдельными элементами данных и может выполняться параллельно на всем множестве вычислительных элементов. Исходными данными для первого этапа являются множество значений функции f в каждой из исходных точек пространства, а также массив коэффициентов $A_{j,k}^{(i)}$. При этом следует учесть, что с увеличением числа исходных точек, увеличивается и число коэффициентов, что в свою очередь увеличит время, необходимое на пересылку данных на устройство выполнения OpenCL, и может снизить производительность [6]. В качестве оптимизации первого этапа и уменьшении времени его выполнения в настоящей работе предлагается алгоритм вычисления коэффициентов $A_{j,k}^{(i)}$ по индексам i, j и k.

Второй этап работы алгоритма выполняет редукцию полученных на первом этапе данных, суммируя вычисленные произведения. Суммирование данных является операцией над зависимыми данными и требует использования синхронизирующих примитивов (барьеров) при ее распараллеливании. Не смотря на то, что в некоторых работах [7], [8] предлагаются довольно эффективные методы для выполнения параллельной редукции данных на графическом процессоре, нельзя однозначно точно сказать будет ли параллельная редукция данных на GPU выполнятся быстрее, чем последовательная редукция на CPU на определенной конфигурации компьютера [9].

Таким образом, для составления наиболее эффективного алгоритма численного интегрирования по методу Симпсона на графическом процессоре, необходимо прежде всего проанализировать, насколько большое влияние на производительность оказывает каждый из перечисленных выше факторов: использование заранее подготовленного массива коэффициентов $A_{j,k}^{(i)}$ или их вычисление, применение алгоритма параллельной редукции на графическом процессоре или последовательной редукции на центральном процессоре.

Для анализа этих факторов в настоящей работе реализованы три алгоритма численного интегрирования по методу Симпсона:

- первый алгоритм использует дополнительный массив коэффициентов $A_{j,k}^{(i)}$ и выполняет последовательную редукцию на центральном процессоре;
- второй алгоритм не использует дополнительный массив коэффициентов $A_{j,k}^{(i)}$, а рассчитывает их в момент вычисления и также выполняет последовательную редукцию на CPU;
- третий алгоритм рассчитывает коэффициенты $A_{j,k}^{(i)}$ и реализует алгоритм параллельной редукции [3].

Для вычисления коэффициентов $A_{j,k}^{(i)}$ во втором и третьем варианте используется формула (5).

Аналитические формулы для оценки времени выполнения алгоритма в зависимости от количества элементов исходных данных можно вывести, используя следующие допущения. Будем считать, что все арифметические операции выполняются за одинаковое время t_a, операции чтения и записи в глобальную память выполняются за время t_g, операции чтения и записи в локальную память выполняются за время t_l. С учетом введенных допущений, формулы для оценки времени выполнения каждого из алгоритмов можно записать следующим образом:

$$T_1 = \frac{N}{N_p} \cdot \left(3t_g + t_a\right) + Nt_a + C, \qquad (6)$$

$$T_2 = \frac{N}{N_p} \cdot \left(2t_g + 2t_a\right) + Nt_a + C, \qquad (7)$$

$$T_3 = \frac{N}{N_p} \cdot \left(2t_g + 2t_a\right) + \frac{N}{L_{size}} \cdot \left(t_a + t_l\right) + \\ + \sum_{i=1}^{\log_2 L_{size}} L_{size} \cdot 2^{-i} \cdot t_l + C, \quad (8)$$

где N – общее количество элементов данных; N_p – количество доступных ядер для параллельного использования; C – неучтенное время, затрачиваемое на пересылку данных средствами OpenCL и другие вспомогательные операции, это время не зависит от исходных данных; L_{size} – количество элементов в используемой локальной группе при реализации алгоритма [8].

Каждый из алгоритмов в данной работе реализован на языке программирования Python [10]. Работа с библиотекой OpenCL обеспечивалась с помощью программной библиотеки PyOpenCL [11]. Для реализации последовательной редукции на центральном процессоре использовался пакет NumPy [12].

OpenCL программа для первого алгоритма имеет вид:

```
__kernel void simpson1(__global
const float * data,
__global const float * a, __global
float * out)
{
   int gid = get_global_id(0);
     out[gid] = data[gid] *
a[gid];
}
```

Где параметр `data` определяет массив со значениями функции f в исходных точках. Параметр `a` определяет массив коэффициентов $A_{j,k}^{(i)}$ для каждой из исходных точек. Параметр `out` является массивом выходных значений.

OpenCL программа второго алгоритма:

```
__kernel void simpson2(__global
const float * data,
unsigned nx, unsigned ny, unsigned
nz,
__global float * out)
{
  int gid = get_global_id(0);
  int i = gid % nz;
  int j = ( gid / nz ) % ny;
  int k = gid / ( nz * ny );
  int ai = ( i & 1 ) != 0 ? 4 : 2;
  if( i == 0 || i == nz - 1 )
    ai--;
  int aj = ( j & 1 ) != 0 ? 4 :
2;
  if( j == 0 || j == ny - 1 )
    aj--;
  int ak = ( k & 1 ) != 0 ? 4 : 2;
  if( k == 0 || k == nx - 1 )
    ak--;
  int a = ai * aj * ak;
  out[gid] = a * data[gid];
}
```

В приведенном выше листинге программы сначала определяются индексы i, j и k по глобальному индексу очередной точки в пространстве, затем вычисляются коэффициенты a_i, a_j, a_k и вычисляется произведение коэффициентов и значения функции в текущей точке пространства.

OpenCL программа третьего алгоритма:

```
__kernel void simpson3(__global
const float *data,
__local float * partial_sum,
unsigned nx, unsigned ny, unsigned
nz,
unsigned total_count,
__global float *out)
{
  int lid = get_local_id( 0 );
  float sum = 0.0f;
   for (uint l = lid; l < total_
count; l += get_local_size(0))
  {
   int i = l % nz;
   int j = ( l / nz ) % ny;
   int k = l / ( nz * ny );
   int ai = ( i & 1 ) != 0 ? 4 : 2;
    if( i == 0 || i == nz - 1 )
      ai--;
   int aj = ( j & 1 ) != 0 ? 4 : 2;
    if( j == 0 || j == ny - 1 )
      aj--;
   int ak = ( k & 1 ) != 0 ? 4 : 2;
    if( k == 0 || k == nx - 1 )
      ak--;
    int a = ai * aj * ak;
    sum += a * out[l];
  }
  partial_sum[lid] = sum;
   for (uint stride = get_local_
size(0)/2; stride > 0; stride /= 2)
  {
      barrier(CLK_LOCAL_MEM_
FENCE);
    if (lid < stride)
       partial_sum[lid] += par-
tial_sum[lid + stride];
```

```
    }
    if( get_global_id( 0 ) == 0 )
      c[ 0 ] = partial_sum[ 0 ];
}
```

Листинг программы третьего алгоритма является комбинированным подходом второго алгоритма и алгоритма параллельной редукции, предложенного в [9].

Заключение

Работа каждого алгоритма анализировалась на компьютерах двух различных конфигураций. На первой конфигурации компьютера в качестве вычислительного устройства OpenCL использовался центральный процессор, на второй конфигурации – графический процессор.

Результат работы каждого из алгоритмов анализировался по двум основным критериям: общее время выполнения и время выполнения программы OpenCL. Отличие критериев заключается в том, что для первого и второго алгоритма общее время выполнения определяется как сумма времени параллельного вычисления произведений $A_{j,k}^{(i)} \cdot f\left(x_i, y_j, z_k\right)$ на графическом процессоре и времени вычисления суммы полученных произведений на центральном процессоре. Третий алгоритм выполняет и вычисление произведений $A_{j,k}^{(i)} \cdot f\left(x_i, y_j, z_k\right)$ и их суммирование полностью на графическом процессоре, в этом случае общее время выполнения программы отличается от времени выполнения OpenCL незначительно.

Для точного подсчета времени выполнения использовались специальные аппаратные таймеры, имеющие точность измерения до нескольких микросекунд. Итоговая оценка времени являлась арифметическим средним ста итераций для каждого из алгоритмов, выполняемых на каждой из конфигураций компьютеров. Время выполнения каждого алгоритма на каждой из конфигураций вычислялось при различных размерах массивов исходных данных: от N = 1 элемента до N = 10^6 элементов.

Время выполнения алгоритмов представлено в таблице 1.

Отдельно заслуживает рассмотрения график зависимости времени выполнения алгоритма 1 на графическом процессоре от количества элементов исходных данных (рис. 2). График показывает, что время работы алгоритма можно существенно сократить, используя определенное количество элементов исходных данных.

Исходя из полученных результатов можно сделать следующие выводы:

1. Вариант реализации алгоритма с вычислением коэффициентов $A_{j,k}^{(i)}$ «на лету» выполняется на графическом процессоре быстрее (5.892мс для 10^6 элементов), чем вариант с заранее посчитанными коэффициентами (6.085мс для 10^6 элементов). При выполнении на центральном процессоре наблюдается обратная зависимость: вариант алгоритма с вычислением коэффициентов $A_{j,k}^{(i)}$ (58.001мс для 10^6 элементов) выполняется медленнее, чем вариант с посчитанными заранее коэффициентами (40.549мс для 10^6 элементов). Такая зависимость объясняется прежде всего тем, что доступ к памяти при выполнении программы на центральном процессоре осуществляется значительно быстрее, чем при ее выполнении на графическом процессоре. Поэтому, для оптимизации программы для выполнения на графическом процессоре необходимо учитывать этот факт и

Таблица 1

Время выполнения алгоритмов.

Алгоритм		$N = 1000$	$N = 100000$	$N = 500000$	$N = 1000000$
Алгоритм1-CPU	Общее время, мс	0.375	6.264	23.611	40.549
	Время OpenCL, мс	0.156	3.833	15.227	25.434
Алгоритм2-CPU	Общее время, мс	0.425	7.707	30.361	58.001
	Время OpenCL, мс	0.173	5.415	22.126	42.778
Алгоритм3-CPU	Общее время, мс	0.248	5.569	26.446	50.546
	Время OpenCL, мс	0.099	5.353	25.174	50.248
Алгоритм1-GPU	Общее время, мс	0.331	1.009	3.421	6.085
	Время OpenCL, мс	0.154	0.229	1.058	1.653
Алгоритм2-GPU	Общее время, мс	0.331	0.957	3.183	5.892
	Время OpenCL, мс	0.02	0.165	0.727	1.168
Алгоритм3-GPU	Общее время, мс	0.432	1.699	6.56	13.815
	Время OpenCL, мс	0.325	1.288	6.027	12.202

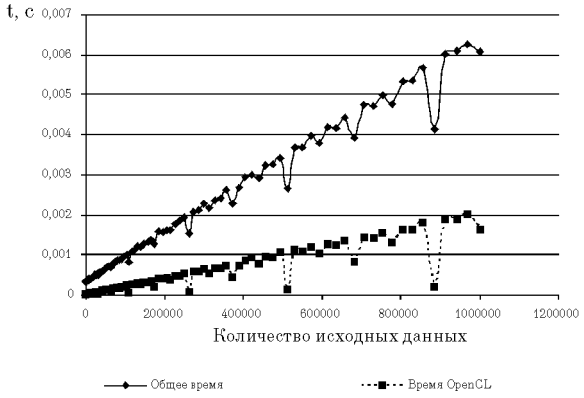

Рис. 2. Результат работы алгоритма 1 на графическом процессоре

использовать прямое вычисление некоторых данных в ходе выполнения программы, вместо передачи их отдельным массивом.

2. В зависимости от количества исходных данных время выполнения программы на графическом процессоре может быть уменьшено на порядок (рис. 2, локальные минимумы). На рис. 2 локальные минимумы, в которых наблюдается значительное уменьшение времени выполнения программы соответствуют количеству исходных данных в 48^3, 64^3, 96^3 элементов. Это объяснятся особенностями работы графического процессора с памятью.

3. Наиболее быстрой реализацией для большого количество элементов исходных данных является реализация алгоритма 2 на графическом процессоре: вычисление коэффициентов $A_{j,k}^{(i)}$ «на лету» и выполнение последовательной редукции на центральном процессоре. Между тем, эта реализация полностью использует ресурсы центрального процессора, что может негативно сказаться на фоновой работе других приложений и выполнении других задач, не связанных с ресурсоемкими вычислениями.

4. Реализация алгоритма 3 на графическом процессоре выполняется примерно в два раза медленнее других алгоритмов, но при этом параметр утилизации работы графического процессора, рассчитываемый как отношение времени выполнения вычислений на графическом процессоре к времени выполнения всей программы, максимальный. Реализация алгоритма 3 практически не использует ресурсы центрального процессора и может использоваться в качестве замены алгоритма 2 в тех случаях, когда помимо выполнения ресурсоемких вычислений требуется выполнять и другие задачи, например, обрабатывать ввод пользователя, обращаться к базе данных и т.п.

СПИСОК ЛИТЕРАТУРЫ

1. *Запрягаев С. А.* Сечение упругого рассеяния электронов на фуллеренах и углеродных нанотрубках / С. А. Запрягаев, А. А. Карпушин // Математика. Компьютер. Образование : 17 Междунар. конф. : тез., Дубна, 25–30 янв. 2010 г. – М. – Ижевск, 2010. – Вып. 17. – С. 117–118.

2. *Bart Pieters et al.* Motion estimation for H.264/AVC on multiple GPUs using NVIDIA CUDA. Proc. SPIE, Vol. 7443, 74430X (2009).

3. *John E. Stone et al.* High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs. ACM International Conference Proceeding Series; Vol. 383 (2009).

4. *Hongsheng Li, et al.* Actin Filament Tracking Based on Particle Filters and Stretching Open Active Contour Models, Int'l Conf. Medical Image Computing and Computer Assisted Intervention (MICCAI), 2009.

5. *Carmine Clemente, et al.* PROCESSING OF SYNTHETIC APERTURE RADAR DATA WITH GPGPU.

6. *Sunpyo Hong and Hyesoon Kim.* An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09). ACM, New York, NY, USA, 152–163. DOI=10.1145/1555754.1555775 http://doi.acm.org/10.1145/1555754.1555775.

7. *Vignesh T. Ravi, Wenjing Ma, David Chiu, and Gagan Agrawal.* Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. In Proceedings of the 24th ACM International Conference on Supercomputing(ICS '10). ACM, New York, NY, USA, 137–146. DOI=10.1145/1810085.1810106 http://doi.acm.org/10.1145/1810085.1810106.

8. http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingOverview.pdf.

9. http://developer.amd.com/documentation/articles/Pages/OpenCL-Optimization-Case-Study-Simple-Reductions.aspx.

10. http://www.python.org.

11. http://mathema.tician.de/software/pyopencl.

12. http://numpy.scipy.org/.

13. *Самарский А. А.* Численные методы: учеб. пособие для вузов. / А. А. Самарский, А. В. Гулин – М.: Наука. Гл. ред. физ-мат. лит., 1989. – 432 с.

Запрягаев Сергей Александрович – доктор физико-математических наук, профессор кафедры цифровых технологий Воронежского государственного университета. Тел. (4732) 208-257. Email: zsa@main.vsu.ru

Карпушин Андрей Александрович – аспирант, кафедра цифровых технологий, Воронежский государственный университет. Тел. (4732) 208-257. Email: reven86@gmail.com

Zapryagaev S. A. – Doctor of Physics-math. Sciences, Professor of the dept. of digital technologies Voronezh State University. Tel. (4732) 208-257. Email: zsa@main.vsu.ru

Karpushin A. A. – Post-graduate student of the dept. of digital technologies Voronezh State University. Tel. (4732) 208-257. Email: reven86@gmail.com