Параллельное вычисление нормированных полиномов Лежандра с использованием графических ускорителей*

К.С. Исупов, В.С. Князьков, А.С. Куваев, М.В. Попов

Вятский государственный университет, г. Киров

В ряде физических расчетов, например, при определении геопотенциала или момента импульса в квантовой механике, возникает необходимость вычисления нормированных полиномов Лежандра. Мы рассмотрим решение этой задачи на современных графических процессорах, массивно-параллельные архитектуры которых позволяют выполнять вычисления сразу для множества аргументов, степеней и порядков полиномов. При большой степени полинома процесс вычисления характеризуется существенным разбросом численных значений и приводит к переполнению и/или потере значимости. Для исключения этих ситуаций реализована поддержка арифметики расширенного динамического диапазона.

Ключевые слова: нормированные полиномы Лежандра, CUDA, арифметика расширенного диапазона.

1. Введение

Присоединенные полиномы Лежандра являются решением дифференциального уравнения [1, глава. 12.5]

$$(1-x^2)\frac{d^2}{dx^2}P_n^m(x) - 2x\frac{d}{dx}P_n^m(x) + \left[n(n+1) - \frac{m^2}{1-x^2}\right]P_n^m(x) = 0,$$

где степень n и порядок m удовлетворяют условиям $0 \le n$ и $0 \le m \le n$, а x — вещественная переменная в интервале $[-1,1]$, обычно выражаемая через $\cos\theta$ (здесь θ — коширота).

Эти полиномы имеют важное значение при расчетах геопотенциала поверхности Земли [2,3], сферических функций в молекулярной динамике [4], момента импульса в квантовой механике [5], а также в ряде других физических приложений. От максимальной степени полинома, который может быть корректно вычислен, напрямую зависит точность и масштаб численного моделирования. Современные приложения оперируют, как правило, полиномами первого рода ($m = 0$) со степенями 10^3 и выше.

Функции $P_n^m(x)$ растут комбинаторно с ростом n, что приводит к переполнению при $n \approx 150$. Поэтому при больших n вместо $P_n^m(x)$ вычисляют *нормированные* присоединенные полиномы Лежандра. Среди различных способов нормировки выделяют вычисление полностью нормированных (fully normalized) полиномов

$$\bar{P}_n^m(x) = \sqrt{\frac{2n+1}{2}\frac{(n-m)!}{(n+m)!}}(1-x^2)^{m/2}\frac{\partial^m}{\partial x^m}P_n(x), \tag{1}$$

которые удовлетворяют уравнению

$$\int_{-1}^{1}\{P_n^m(x)\}^2 dx = 1.$$

Для вычисления $\bar{P}_n^m(x)$ предложен ряд рекуррентных алгоритмов [7–9]. Один из наиболее распространенных основан на следующем соотношении [7]:

$$\bar{P}_n^{m-1}(x) = \frac{2mx}{\sqrt{(1-x^2)(n+m)(n-m+1)}}\bar{P}_n^m(x) - \sqrt{\frac{(n-m)(n+m+1)}{(n+m)(n-m+1)}}\bar{P}_n^{m+1}(x). \tag{2}$$

*Исследование выполнено при финансовой поддержке РФФИ (проект № 16-37-60003 мол_а_дк).

Отправной точкой для рекурсии (2) являются значения $\bar{P}_n^{n+1}(x)$ и $\bar{P}_n^n(x)$, определяемые следующим образом:

$$\bar{P}_n^{n+1}(x) = 0,$$
$$\bar{P}_n^n(x) = \sqrt{\frac{1}{2}\frac{3 \cdot 5 \cdots (2n+1)}{2 \cdot 4 \cdots 2n}}(1-x^2)^{n/2}. \quad (3)$$

Соотношения (2) и (3) асимптотически устойчивы при любых допустимых параметрах x, m и n, поэтому, если рассматривать их с точки зрения чистой математики, пригодны для вычисления полиномов произвольно высокой степени. Вместе с тем, при больших n вычисление этих соотношений затруднительно, что обусловлено следующими причинами:

- вычисления занимают неприемлемо много времени;
- при вычислениях возникает переполнение (overflow) или потеря значимости (underflow).

Первая проблема связана с тем, что при проведении численного моделирования требуется вычисление сразу группы полиномов различных степеней при фиксированном угле, либо группы полиномов фиксированной степени для множества углов. Эффективное решение этой проблемы стало возможным благодаря интенсивному развитию массивно-параллельных вычислительных архитектур нового поколения, в частности графических ускорителей (GPU), и соответствующих им программных моделей, таких как CUDA.

Вторая проблема связана с ограниченностью динамического диапазона вещественных чисел, представимых в компьютерах. В результате этого, если x находится вблизи ± 1, то вычисление (3) приводит к потере значимости, даже когда результат вычисления соотношения (2) лежит в пределах допустимого диапазона. Например, если $x = 0.98481$, что соответствует углу $\theta = 10°$, то $\bar{P}_{5000}^{5000}(x) \approx 9.82 \times 10^{-3802}$, а $\bar{P}_{5000}^{0}(x) \approx 0.98$. Минимальное нормализованное значение, представимое в формате IEEE-754 двойной точности, составляет примерно 10^{-308}. Таким образом, для вычисления $\bar{P}_{5000}^{5000}(x)$ требуется увеличение динамического диапазона более чем на порядок. И хотя $\bar{P}_{5000}^{5000}(x)$ не представляет самостоятельного практического интереса, без его корректного вычисления невозможно начать рекурсию для вычисления $\bar{P}_{5000}^{0}(x)$, поскольку если из-за потери значимости $\bar{P}_{5000}^{5000}(x) = 0$, то все последующие значения $\bar{P}_{5000}^{4999}(x), \bar{P}_{5000}^{4998}(x)$, и т.д., также будут обращаться в нуль. Некоторая информация о диапазоне углов и ограничении степеней полиномов, при которых вычисления в арифметике IEEE-754 не приводят к исключениям, представлена в [2].

Для исключения потери значимости предложены методы, использующие глобальные коэффициенты масштабирования [6]. Однако, как отмечено в [3], это позволяет решить проблему лишь для ограниченных диапазонов аргументов и степеней. Общее решение проблемы переполнения и / или потери значимости при вычислении нормированных полиномов Лежандра предложено в [7] и заключается в использовании арифметики расширенного динамического диапазона.

В данной работе рассматривается параллельное вычисление нормированных полиномов $\bar{P}_n^m(x)$ высоких степеней в арифметике расширенного диапазона на CUDA-совместимых GPU. Благодаря высокому уровню параллелизма задачи, перенос вычислений на GPU позволил добиться значительного прироста производительности, по сравнению с CPU-реализацией.

2. Арифметика расширенного диапазона

Стандарт IEEE-754 [10] является на сегодняшний день основным стандартом двоичной арифметики с плавающей точкой. Он определяет два наиболее распространенных формата: одинарной точности (binary32) и двойной точности (binary64). Эти форматы в той или иной степени поддерживаются как на уровне аппаратной части, так и на уровне языков программирования. В 2008 году вышла ревизия стандарта [11], которая дополнительно описывает

Таблица 1: Форматы IEEE-754 одинарной и двойной точности. Количество p цифр мантиссы определяет точность (precision) формата, а числа $e_{\min}$ и $e_{\max}$, ограничивающие e, задают динамический диапазон; $n_{\max}$ — наибольшее положительное конечное число, $n_{\min}$ — наименьшее положительное нормализованное число, $s_{\min}$ — наименьшее положительное денормализованное число. Отрезок $[n_{\min}, n_{\max}]$ задает диапазон представления нормализованных положительных чисел, а отрезок $[s_{\min}, n_{\max}]$ — полный диапазон представления конечных положительных чисел.

	p	$e_{\min}$	$e_{\max}$	$n_{\min}$	$n_{\max}$	$s_{\min}$
binary32	24	-126	$+127$	2^{-126}	$(2-2^{-23})\times 2^{127}$	2^{-149}
binary64	53	-1022	$+1023$	2^{-1022}	$(2-2^{-52})\times 2^{1023}$	2^{-1074}

двоичный формат binary128 и два десятичных формата (decimal64 и decimal128). Однако поддержка эти новых форматов в настоящее время реализуется лишь в редких случаях. Основные параметры форматов binary32 и binary64 представлены в таблице 1.

Ситуация, когда округленный результат арифметической операции или функции превысил по абсолютной величине наибольшее конечное число с плавающей точкой $n_{\max} = (b - b^{1-p}) \times b^{e_{\max}}$, классифицируется как переполнение (overflow). Ситуация, когда результат арифметической операции расположен слишком близко к нулю, т.е. является не нулевым, но по абсолютному значению строго меньше наименьшего положительного нормализованного числа $n_{\min} = b^{e_{\min}}$, классифицируется как потеря значимости (underflow) [11, 12]. При переполнении результат, в зависимости от используемого режима округления, заменяется бесконечностью ($\pm\infty$) либо максимальным конечным числом ($\pm n_{\max}$). При потере значимости результат заменяется нулем, денормализованным числом или $\pm n_{\min}$. Во всех случаях знак округленного результата совпадает со знаком промежуточного результата. Рассмотренные исключения показаны на Рис. 1.

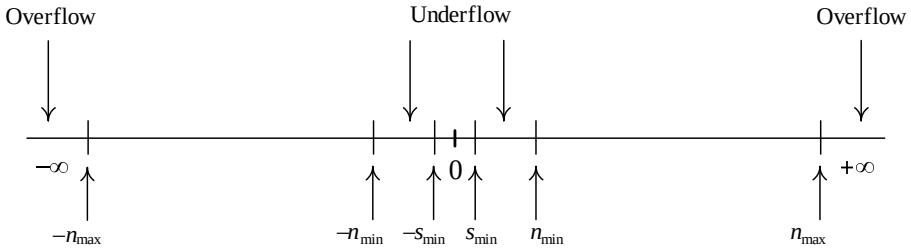

Рис. 1: Переполнение и потеря значимости в арифметике с плавающей точкой

Одним из способов исключения переполнения или потери значимости является масштабирование. Этот способ предполагает оценку величины исходных операндов и их умножение на коэффициент K выбранный таким образом, что все промежуточные результаты будут находиться в пределах нормализованного диапазона. После получения конечного результата осуществляется корректировка его масштаба путем деления на K [12]. С точки зрения скорости вычислений такой прием, очевидно, является наилучшим. Однако он требует детального анализа всего вычислительного процесса и не применим во многих случаях. Более общим вариантом является эмуляция арифметики расширенного диапазона (extended-range floating-point). Для этого выполняется объединение целого e с обычным машинным числом

с плавающей точкой f, и эта пара рассматривается как число [7,13]

$$f \times B^e, \tag{4}$$

где B – предопределенная константа, обозначающая основание системы счисления (база экспоненты), в качестве которой целесообразно выбирать некоторую натуральную степень двойки. Например, если f — стандартное машинное число двойной точности (binary64), e — 32-битное целое число со знаком ($-2147483648 \leq e \leq 2147483647$) и $B = 2^{256}$, то диапазон представимых чисел будет превышать $10^{\pm 165492990270}$. Мантисса f может принимать значения в интервале $(1/B, B)$. Исходя из этого, B должно быть таким, чтобы при любой операции над f не произошло переполнения или потери значимости.

Алгоритмы выполнения основных операций в расширенном диапазоне представлены в [7,13], поэтому остановимся лишь на некоторых из них, которым, на наш взгляд, в литературе уделено недостаточно внимания, либо на тех, которые претерпели существенные изменения в процессе разработки.

Будем считать, что основание экспоненты B однозначно определено, и будем записывать число с расширенной экспонентой в виде пары (f, e). Алгоритм 1 выполняет «регулировку» числа. С его помощью обеспечивается контроль диапазона изменения мантиссы f, а также исправление в случае, если на вход поступает некорректная кодировка нуля. Он является одним из основных алгоритмов арифметики расширенного диапазона.

Алгоритм 1 Регулировка представления с расширенной экспонентой

```
1:  procedure ADJUST(f, e)
2:      if f = 0 then return (0, 0)
3:      else if |f| ≥ B then
4:          f ← f/2^(log2 B)          ▷ Вычитание log2 B из экспоненты f
5:          e ← e + 1
6:      else if |f| ≤ 1/B then
7:          f ← f × 2^(log2 B)        ▷ Прибавление log2 B к экспоненте f
8:          e ← e − 1
9:      end if
10:     return (f, e)
11: end procedure
```

Важно отметить, что иногда однократного выполнения процедуры Adjust оказывается недостаточно. Такая ситуация может иметь место, по крайней мере, в следующих двух случаях: (1) когда выполняется преобразование числа из машинного формата или формата с отличной от текущей базой экспоненты; (2) когда выполняется вычитание близких по величине чисел. В любой из этих ситуаций возможно, что после выполнения Adjust мантисса f окажется меньше $1/B$. Если игнорировать это и продолжить вычисления, то вполне вероятно постепенное “зануление” результата. Для исключения этой ситуации, необходимо использовать процедуру циклической регулировки, которая представляет собой многократное повторение Adjust до тех пор, пока f не окажется в требуемом диапазоне. Эта процедура реализуется Алгоритмом 2.

Алгоритм 3 реализует сложение чисел. Алгоритмы вычитания и сравнения очень схожи с алгоритмом сложения, и их рассмотрение не вызывает интереса.

3. Вычисление нормированных полиномов Лежандра на GPU

Для поддержки вычислений в расширенном диапазоне на центральных процессорах и графических процессорах видеокарты объявлены типы данных, представленные на Рис. 2.

Список реализованных CPU- и CUDA (device)-функций включает:

Алгоритм 2 Циклическая регулировка представления с расширенной экспонентой. Должна быть использована в алгоритмах преобразования типов, сложения и вычитания

procedure BADJUST(f, e)
  (f_1, e_1) ← ADJUST(f, e)
  while $e \neq e_1$ **or** $f \neq f_1$ **do**
    (f, e) ← (f_1, e_1)
    (f_1, e_1) ← ADJUST(f, e)
  end while
  return (f_1, e_1)
end procedure

Алгоритм 3 Сложение чисел с расширенной экспонентой, $(f_z, e_z) \leftarrow (f_x, e_x) + (f_y, e_y)$

procedure ADD(f_x, e_x, f_y, e_y)
  if $f_x = 0$ **and** $e_x = 0$ **then return** (f_y, e_y)
  else if $f_y = 0$ **and** $e_y = 0$ **then return** (f_x, e_x)
  end if
  $\Delta e = |e_x - e_y|$
  if $e_x > e_y$ **then**
    $f_z \leftarrow f_x + f_y \times 2^{-\Delta e \times \log_2 B}$
    $e_z \leftarrow e_x$
  else if $e_y > e_x$ **then**
    $f_z \leftarrow f_y + f_x \times 2^{-\Delta e \times \log_2 B}$
    $e_z \leftarrow e_y$
  else if $e_y = e_x$ **then**
    $f_z \leftarrow f_x + f_y$
    $e_z \leftarrow e_x$
  end if
  return BADJUST(f_z, e_z)
end procedure

```
typedef struct {
    er_frac_t frac;    //significand
    er_exp_t exp;      //exponent
} __extended_range_struct;

typedef __extended_range_struct *er_t;        //for single number
typedef __extended_range_struct *er_arr_t;    //for arrays
typedef __extended_range_struct er_static_t; //for device side code
```

Рис. 2: Типы данных с расширенной экспонентой: `er_frac_t` — стандартное число с плавающей точкой (по умолчанию `double`), `er_exp_t` — машинное целое (по умолчанию `int64_t`)

1. функции управления памятью и предвычислительной инициализации констант;
2. функции сложения, вычитания, умножения и деления, поддерживающие все режимы округления стандарта IEEE-754[1], а также функции сравнения;
3. целочисленные функции floor (наибольшее целое, которое меньше аргумента или равно ему) и ceil (наименьшее целое, которое больше аргумента, или равно ему) и функции получения дробной части числа;
4. функции преобразования чисел из типа данных двойной точности IEEE-754 в типы данных с расширенной экспонентой, и обратно;
5. вычисление факториала, экспоненты, степени, квадратного корня и ряд других математических функций.

Основание экспоненты B задается в параметрах[2]. Объявления CPU- и GPU-функций идентичны (для GPU-функций используется пространство имен `cuda`). Для эффективной передачи параметров используются указатели. Все функции являются потокобезопасными. На основе этих функций разработаны подпрограммы, позволяющие вычислять $\bar{P}_n^m(x)$ для больших $n > 0$ и при любых $m, 0 < m < n$. Их описание представлено в таблице 2.

Листинг CUDA-ядра `legendre_lst` представлен на Рис. 3. В ходе выполнения этой функции i-й поток вычисляет полином степени `n[i]` порядка `m[i]` для аргумента `x[i]`. Результат записывается с соответствующим смещением в массив `res` для дальнейшего копирования в память CPU. Число требуемых CUDA-блоков определяется соотношением

$$N = \left\lceil \frac{size}{max\_threads\_per\_block} \right\rceil,$$

где $size$ — размер векторов входных данных, $max\_threads\_per\_block$ — максимальное число потоков в блоке. Если N не превышает максимально возможного числа блоков для данного устройства, то возможно полностью параллельное вычисление всех полиномов. В противном случае часть потоков вычисляет более одного полинома.

4. Экспериментальные результаты

Оценка корректности и эффективности разработанных подпрограмм выполнялась на стенде HP SL390+NVIDIA Tesla M 2090 платформы UniHUB.ru ИСП РАН [14]. Рассмотрены три программные реализации рекуррентного алгоритма (2): CPU- и GPU-реализации на основе арифметики расширенного диапазона и вычисления с использованием пакета многократной точности The GNU MPFR Library.

В ходе первого эксперимента исследовалась зависимость времени вычислений полинома первого рода ($m = 0$) от n. В качестве аргумента выступало значение $\cos(179°) \approx -0.9998$.

[1] Для эффективной реализации направленных округлений на GPU используется CUDA Math API.

[2] По умолчанию $B = 2$. Этого вполне достаточно для проводимых расчетов.

Таблица 2: Подпрограммы для вычисления нормированных полиномов Лежандра.

Подпрограмма	Параметры	Описание	Примечание

`legendre_eqls`	`er_t res`
	`er_t x`
	`uint32_t const n`
		Вычисляет $\bar{P}_n^n(x)$ в соответствии с (3). Результат — число в памяти с указателем `res`.	В GPU-реализации имеет спецификатор `__device__`.

`legendre_recur`	`er_t res`
	`er_t x`
	`er_t p1`
	`er_t p2`
	`uint32_t const n`
	`uint32_t const m`
		Выполнение одной итерации рекурсии (2). Для заданных $\bar{P}_n^m(x)$ (параметр `p1`) и $\bar{P}_n^{m+1}(x)$ (параметр `p2`) вычисляет $\bar{P}_n^{m-1}(x)$. Результат — число в памяти с указателем `res`.	В GPU-реализации имеет спецификатор `__device__`.

`legendre`	`er_t res`
	`double const x`
	`uint32_t n`
	`uint32_t m`
		Вычисляет нормированный полином Лежандра $\bar{P}_n^m(x)$ степени `n` порядка `m`. Результат — число в памяти с указателем `res`.	Является глобальной функцией в GPU-реализации (имеет спецификатор `__global__`).

`legendre_lst`	`er_arr_t res`
	`double const *x`
	`uint32_t const *n`
	`uint32_t const *m`
	`uint32_t size`
		Для заданных вектора аргументов `x`, вектора степеней `n` и вектора порядков `m` размера `size` вычисляет вектор нормированных полиномов Лежандра. Результат — массив в памяти с адресом `res`.	Является глобальной функцией в GPU-реализации (имеет спецификатор `__global__`).

Степень n изменялась в диапазоне от 100 до 53200 с удвоением на каждом этапе тестирования. Результаты представлены на Рис. 4.

Во втором эксперименте при фиксированных $m = 0$ и $n = 20000$ вычислялся вектор полиномов, размер которого варьировался в диапазоне от 32 до 8192. Аргументы определялись по формуле: $x_i = \cos\left(i \times \frac{180}{vector\_size}\right)$, что позволило для каждого вектора выполнять вычисления в диапазоне углов $[0°, 180°]$ с равномерным шагом, определяемым размером вектора ($vector\_size$). Результаты эксперимента представлены на Рис. 5. Некоторые вычисленные значения полиномов представлены в таблице 3.

В ряде известных программных пакетов, таких как The GNU Scientific Library, Boost, ALGLIB, реализованы функции вычисления ненормированных и нормированных присоединенных полиномов Лежандра. Однако они позволяют производить вычисления лишь для сравнительно небольших степеней (до нескольких тысяч). Поэтому эти реализации не рассматривались в экспериментах.

```
__global__ void legendre_lst(er_arr_t res,
                             double const *x,
                             uint32_t const *n,
                             uint32_t const *m,
                             uint32_t size)
{
    const uint32_t id = threadIdx.x + blockIdx.x * blockDim.x;
    if (id < size) {
        uint32_t thread_n = n[id];
        uint32_t thread_m = m[id];
        er_static_t thread_x;
        cuda::er_set_d(&thread_x, x[id]);
        legendre_eqls(&res[id], &thread_x, thread_n);
        if (thread_n > thread_m) {
            er_static_t p0, p1, p2;
            cuda::er_set(&p1, &res[id]);
            cuda::er_set_d(&p2, 0.0);
            uint32_t current_m = thread_n;
            legendre_recur(&p0,&thread_x,&p1,&p2,thread_n,current_m);
            uint32_t iter_n = thread_n - thread_m - 1;
            for (uint32_t i = 0; i < iter_n; i++) {
                current_m = current_m - 1;
                cuda::er_set(&p2, &p1);
                cuda::er_set(&p1, &p0);
                legendre_recur(&p0,&thread_x,&p1,&p2,thread_n,current_m);
            }
            cuda::er_set(&res[id], &p0);
        }
    }
}
```

Рис. 3: CUDA-ядро для вычисления нормированных полиномов Лежандра

5. Заключение

В работе рассмотрена задача вычисления нормированных полиномов Лежандра с использованием GPU. При высоких степенях n и порядках m эти полиномы характеризуются большим разбросом численных значений, что значительно ограничивает возможности их корректного вычисления в стандартной арифметике IEEE-754. В частности, при $n = m = 20000$, получение корректных результатов в формате двойной точности binary64 возможно лишь для углов от 75° до 105°. Вычисления для углов, выходящих за эти границы, приводят к потере значимости. Для преодоления этого ограничения реализована арифметика расширенного динамического диапазона на GPU. Полученные экспериментальные оценки эффективности показывают, что, благодаря естественному параллелизму задачи и простой вычислительной схеме, использование GPU целесообразно даже при небольшой длине векторов (64 и более). С ростом размерности входных данных выигрыш во времени становится более значительным. Возрастание времени вычислений, наблюдаемое при размере вектора 4096, связано с ограниченными ресурсами использованной видеокарты.

При параллельном расчете полиномов одной степени для набора различных аргументов схема вычислений является сбалансированной, поскольку время выполнения операций расширенной арифметики не зависит существенным образом от порядка величин. Если выполняется расчет вектора полиномов различных (высоких) степеней, то для увеличения производительности GPU-реализации может быть применена более сложная вычислительная схема с балансировкой нагрузки.

Литература

1. Arfken G.B., Weber H.J. Mathematical Methods for Physicists, Sixth Edition. Elsevier Academic Press, USA, 2005. 1182 p.

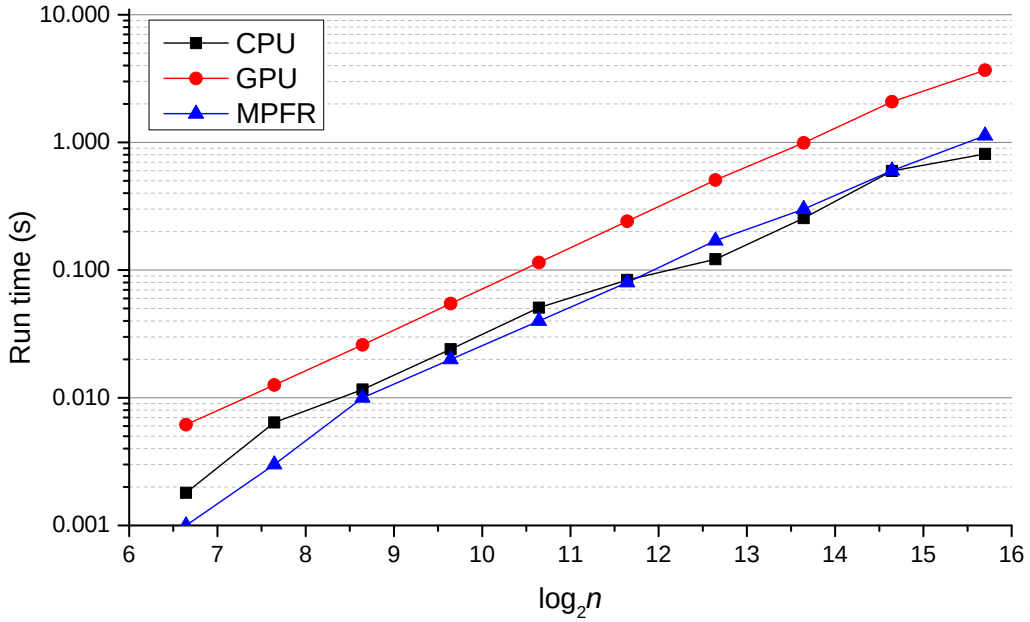

Рис. 4: Зависимость времени вычисления $\bar{P}_n^m(x)$ от степени n при $m = 0$ и $x = \cos(179°)$

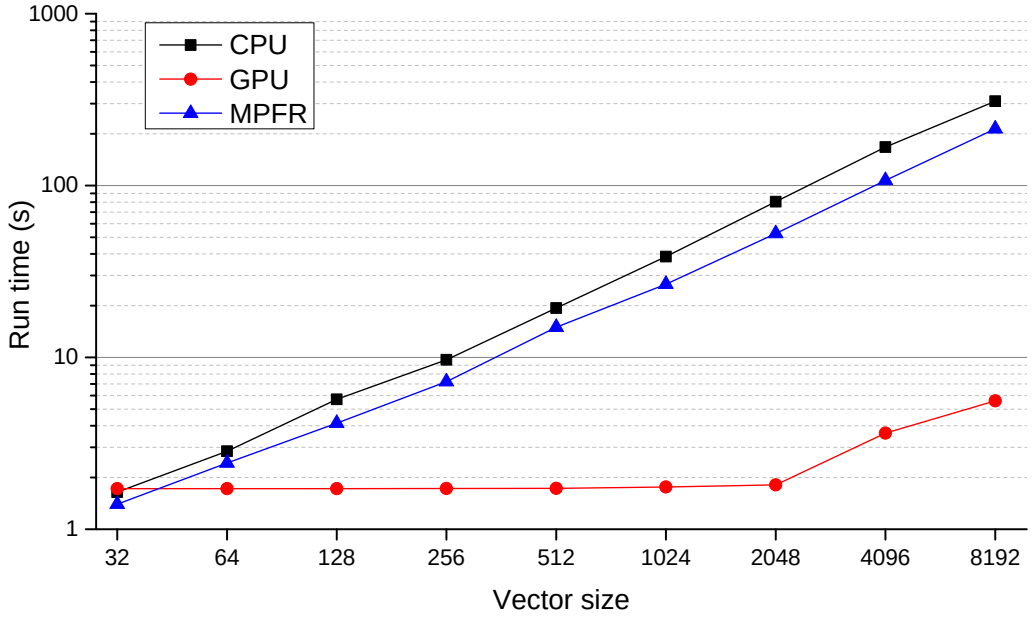

Рис. 5: Зависимость времени вычисления вектора полиномов $\bar{P}_n^m(x)$ от длины при фиксированных $m = 0$ и $n = 20000$. Вычисления на CPU выполняются без распараллеливания.

2. Wittwer T., Roland K., Kurt S., Bernhard H. Ultra-high degree spherical harmonic analysis and synthesis using extended-range arithmetic. Journal of Geodesy. 2008. Vol. 82, No. 4, 223–229.

3. Fukushima T. Numerical computation of spherical harmonics of arbitrary degree and order by extending exponent of floating point numbers. Journal of Geodesy. 2012. Vol. 86, No. 4, 271–285.

4. Morris R. J., Najmanovich R. J., Kahraman A., Thornton J. M. Real spherical harmonic expansion coefficients as 3D shape descriptors for protein binding pocket and ligand comparisons. Bioinformatics. 2005. Vol. 21, No. 10. P. 2347–2355.

Таблица 3: Значения нормированных полиномов Лежандра, вычисленные при $m = 0$ / $m = 20000$ и $n = 20000$. Верификация результатов выполнена с использованием пакета The GNU MPFR Library.

θ	$x = \cos\theta$	$\bar{P}_{20000}^{20000}(x)$	$\bar{P}_{20000}^{0}(x)$
5°	0.9962	$7.4326 \times 10^{-21194}$	-1.4523
15°	0.9659	$7.5091 \times 10^{-11740}$	2.0475×10^{-1}
30°	0.8660	2.2442×10^{-6020}	-9.7721×10^{-1}
45°	0.7071	4.4773×10^{-3010}	8.7662×10^{-1}
60°	0.5000	3.6611×10^{-1249}	-2.2190×10^{-1}
75°	0.2588	6.7071×10^{-301}	-4.9421×10^{-1}
89°	0.0175	4.2459×10^{-1}	-7.4741×10^{-1}
105°	-0.2588	6.7071×10^{-301}	-4.9421×10^{-1}
120°	-0.5000	3.6611×10^{-1249}	-2.2190×10^{-1}
135°	-0.7071	4.4773×10^{-3010}	8.7662×10^{-1}
150°	-0.8660	2.2442×10^{-6020}	-9.7721×10^{-1}
175°	-0.9962	$7.4326 \times 10^{-21194}$	-1.4523

5. Rose M.E. Elementary Theory of Angular Momentum. John Wiley & Sons Inc., New York, 1957. 272 p.

6. Wenzel H.G. Ultra high degree geopotential models GPM98A and GPM98B to degree 1,800. // Proceedings of joint meeting international gravity commission and international geoid commission, Budapest (10–14 March. Rep 98:4, Finnish Geodetic Institute, Helsinki), pp. 71-80.

7. Smith J.M., Olver F.W.J., Lozier D.W. Extended-range arithmetic and normalized Legendre polynomials. ACM Trans. Math. Softw. 1981. Vol. 7, No. 1, 93–105.

8. Paul M. K. Recurrence relations for integrals of Associated Legendre functions. Bulletin Géodésique. 1978. Vol. 52, No. 3, 177–190.

9. Holmes S.A., Featherstone W.E. A unified approach to the Clenshaw summation and the recursive computation of very high degree and order normalised associated Legendre functions. Journal of Geodesy. 2002. Vol. 76, No. 5, 279–299.

10. IEEE Standard for Binary Floating-Point Arithmetic. Introduced 1985-03-21. New York : Institute of Electrical and Electronics Engineers, 1985. 20 p.

11. IEEE Standard for Floating-Point Arithmetic. Introduced 2008-08-29. New York : Institute of Electrical and Electronics Engineers, 2008. 70 p.

12. Muller J.-M. Handbook of Floating-Point Arithmetic // J.-M. Muller [et al.] — Birkhäuser Boston, 2010.

13. Hauser J.R. Handling Floating-Point Exceptions in Numeric Programs // ACM Transactions on Programming Languages and Systems. 1996. Vol. 18, No. 2. P. 139–174.

14. Технологическая платформа программы “Университетский кластер”. URL: https://unihub.ru/resources/sl390m2090 (дата обращения: 17.06.2016).

Parallel Computation of Normalized Legendre Polynomials using Graphics Processors

K.S. Isupov, V.S. Knyazkov, A.S. Kuvaev, M.V. Popov

Vyatka State University, Kirov

The computation of normalized Legendre polynomials is needed in many physical calculations, for example, in the calculation of geopotential and in the calculation of angular momentum in quantum mechanics. We consider the solution of this problem on modern graphics processing units, whose massively parallel architectures allow to perform calculations for many arguments, orders and degrees of polynomials simultaneously. When the degree of polynomial is large computation process is characterized by a significant spread of numerical values and lead to overflow and/or underflow exceptions. Extended-range arithmetic is implemented to eliminate these situations.

Keywords: normalized Legendre polynomials, CUDA, extended-range arithmetic.

References

1. Arfken G.B., Weber H.J. Mathematical Methods for Physicists, Sixth Edition. Elsevier Academic Press, USA, 2005. 1182 p.

2. Wittwer T., Roland K., Kurt S., Bernhard H. Ultra-high degree spherical harmonic analysis and synthesis using extended-range arithmetic. Journal of Geodesy. 2008. Vol. 82, No. 4, 223–229.

3. Fukushima T. Numerical computation of spherical harmonics of arbitrary degree and order by extending exponent of floating point numbers. Journal of Geodesy. 2012. Vol. 86, No. 4, 271–285.

4. Morris R. J., Najmanovich R. J., Kahraman A., Thornton J. M. Real spherical harmonic expansion coefficients as 3D shape descriptors for protein binding pocket and ligand comparisons. Bioinformatics. 2005. Vol. 21, No. 10. P. 2347–2355.

5. Rose M.E. Elementary Theory of Angular Momentum. John Wiley & Sons Inc., New York, 1957. 272 p.

6. Wenzel H.G. Ultra high degree geopotential models GPM98A and GPM98B to degree 1,800. // Proceedings of joint meeting international gravity commission and international geoid commission, Budapest (10–14 March. Rep 98:4, Finnish Geodetic Institute, Helsinki), pp. 71-80.

7. Smith J.M., Olver F.W.J., Lozier D.W. Extended-range arithmetic and normalized Legendre polynomials. ACM Trans. Math. Softw. 1981. Vol. 7, No. 1, 93–105.

8. Paul M. K. Recurrence relations for integrals of Associated Legendre functions. Bulletin Géodésique. 1978. Vol. 52, No. 3, 177–190.

9. Holmes S.A., Featherstone W.E. A unified approach to the Clenshaw summation and the recursive computation of very high degree and order normalised associated Legendre functions. Journal of Geodesy. 2002. Vol. 76, No. 5, 279–299.

10. IEEE Standard for Binary Floating-Point Arithmetic. Introduced 1985-03-21. New York : Institute of Electrical and Electronics Engineers, 1985. 20 p.

11. IEEE Standard for Floating-Point Arithmetic. Introduced 2008-08-29. New York : Institute of Electrical and Electronics Engineers, 2008. 70 p.

12. Muller J.-M. Handbook of Floating-Point Arithmetic // J.-M. Muller [et al.] — Birkhäuser Boston, 2010.

13. Hauser J.R. Handling Floating-Point Exceptions in Numeric Programs // ACM Transactions on Programming Languages and Systems. 1996. Vol. 18, No. 2. P. 139–174.

14. Tehnologicheskaja platforma programmy "Universitetskij klaster" [The "University cluster" program's technological platform]. URL: https://unihub.ru/resources/sl390m2090 (accessed: 17.06.2016).