Решение фундаментальных вопросов организации вычислений повышенной и произвольной точности в гетерогенных вычислительных системах

Валентин Александрович Голодов

«Южно-Уральский государственный университет (национальный исследовательский университет)», Челябинск

golodovva@susu.ru

IEEE 754/754-2008 Standard for Floating-Point Arithmetic

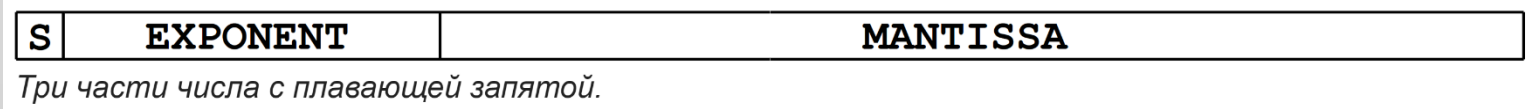
Нормализованный формат

- Множество конечных действительных чисел вида $(-1)^s \cdot (1 + M) \cdot b^E$
  Каждое описывается тремя целыми:
  $s \equiv$ *знак* (0 или 1),
  $M \equiv$ *остаток нормализованной мантиссы*,
  $E \equiv$ *смещенная экспонента*
- b основание (2 или 10).
- M целое в диапазоне от 0 до $b^{(p-1)}-1$
- E целое в диапазоне $1 - E_{max} \leq E \leq E_{max}$
- «Тихий» (qNaN), «сигнальный» (sNaN)
- ±∞ (Infinity)

Пример

- $155{,}625 = 1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3}$
- $155{,}625_{10} = 10011011{,}101_2$
- $1{,}55625 \cdot \exp_{10}{}^{+2} = 1{,}0011011101 \cdot \exp_2{}^{+111}$

1 бит	8 бит	23 бит	IEEE 754
0	1000 0110	001 1011 1010 0000 0000 0000	431BA000 (hex)
0(dec)	134(dec)	1810432(dec)	
знак числа	смещенная экспонента	остаток от мантиссы	число 155,625 в формате IEEE754



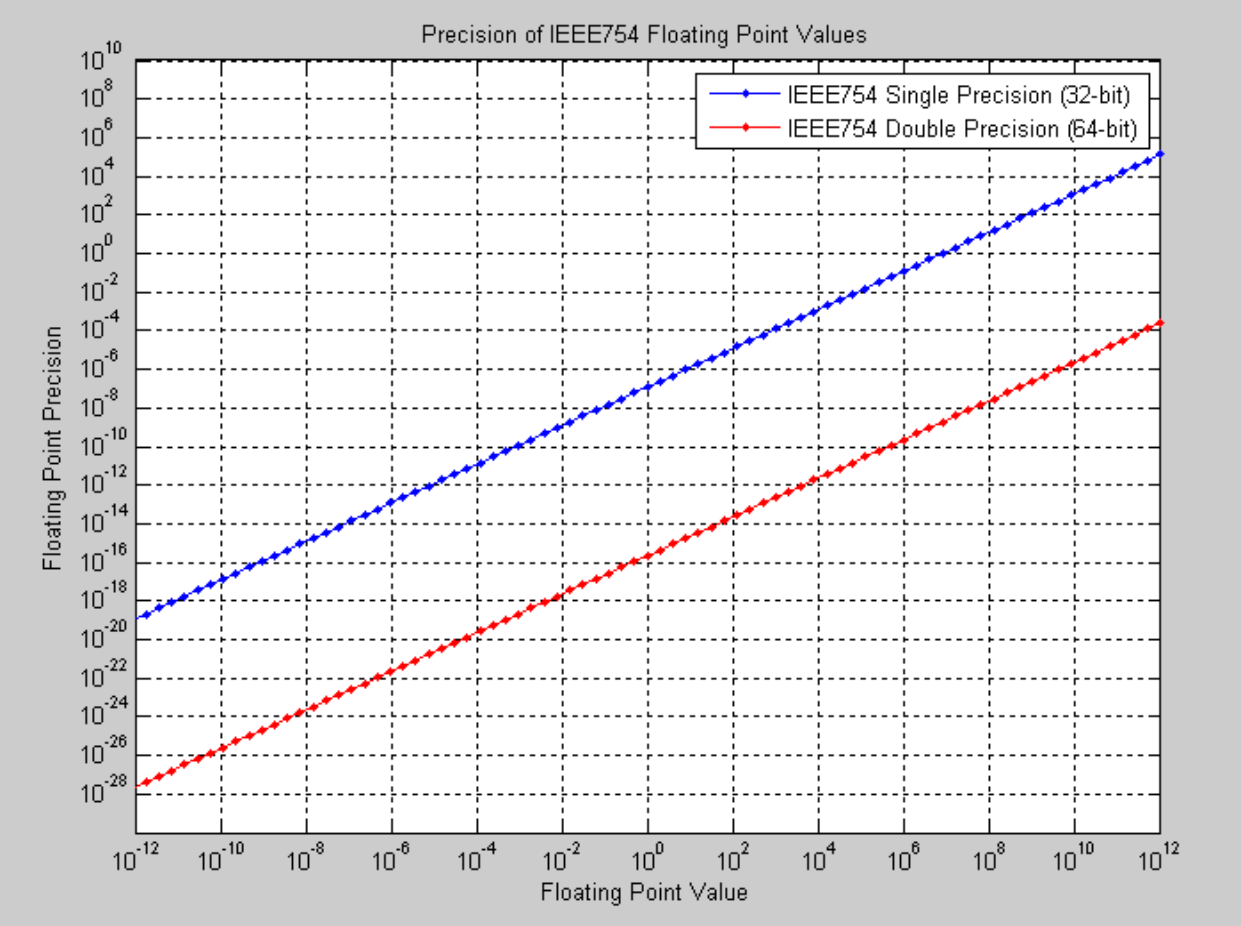
Три части числа с плавающей запятой.

Основные стандартные форматы

Название	Полное название	Основание	Кол-во двоичных разрядов мантиссы	Число десятичных разрядов	Экспонента (бит)	Десятичный E_{max}	Смещение экспоненты	E_{min}	E_{max}
binary32	Одинарная точность	2	24	7.22	8	38.23	$2^7-1 = 127$	−126	+127
binary64	Двойная точность	2	53	15.95	11	307.95	$2^{10}-1 = 1023$	−1022	+1023
binary128	Четырёхкратная точность	2	113	34.02	15	4931.77	$2^{14}-1 = 16383$	−16382	+16383
decimal64		10	16	16	9.58	384	398	−383	+384

Стандарт IEEE 754/754-2008

Точность вычислений в зависимости от величины числа



Математические проблемы стандарта

- Бинарные форматы binary32/binary64 НЕ способны точно представить большинство десятичных чисел, например, число 0.1_{10}
- Нарушение базовых математических законов (коммутативность, ассоциативность, дистрибутивность)
- Правила округления различны для различных базовых форматов binary32 $\frac{1}{3} \approx 3eaa\ aaab_{16}$, но $binary64\ \frac{1}{3} \approx 3fd5\ 5555\ 5555\ 5555_{16}$

Машинные проблемы стандарта

- Один из выводов в работе *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, by David Goldberg, published in the March, 1991 issue of Computing Surveys. Copyright 1991, Association for Computing Machinery, Inc.
- «… The idea that IEEE 754 prescribes precisely the result a given program must deliver is nonetheless appealing. Many programmers like to believe that they can understand the behavior of a program and prove that it will work correctly without reference to the compiler that compiles it or the computer that runs it. In many ways, supporting this belief is a worthwhile goal for the designers of computer systems and programming languages. Unfortunately, when it comes to floating-point arithmetic, the goal is virtually impossible to achieve. The authors of the IEEE standards knew that, and they didn't attempt to achieve it. As a result, despite nearly universal conformance to (most of) the IEEE 754 standard throughout the computer industry, programmers of portable software must continue to cope with unpredictable floating-point arithmetic.…»

Почему необходимо новое решение?

- Стандарт разработан в эпоху последовательных компьютеров, программа сохраняет повторяемость результата для различных запусков на одной и той же последовательной машине.
- В эпоху параллельных программ для верификации могут использоваться результаты, полученные реализациями различных алгоритмов на различных машинах, например, для подсчета ускорения $s=t_1/t_p$. Что снимает практически все гарантии.

Параллельные алгоритмы

- Для широко класса, так называемого линейного класса алгоритмов, известна процедура преобразования к параллельному виду, сохраняющая результат вычислений с точностью до ошибок округления [Воеводин В.В. Вычислительная математика и структура алгоритмов. – М.: Изд-во МГУ, 2006. 112 с.] стр. 79.

Пути решения намеченных проблем

- Unum ?
- Binary128, binary256 ?
- Математические модели числа лишенные указанных проблем!

Unum

- **The End of Error: Unum Computing**
- John L. Gustafson, CRC Press

- A Critique of John L. Gustafson's

THE END of ERROR — Unum Computation

and his

A Radical Approach to Computation with Real Numbers

by W. Kahan
www.eecs.berkeley.edu/~wkahan/UnumSORN.pdf

Unum

- Unum , Type I, Type II
- Unum Type III адаптирован для реализации «в железе»
- Более подробно https://www.posithub.org/

Математические свойства компьютерной модели числа

1. Обеспечивать математические законы (коммутативность, ассоциативность, дистрибутивность)
2. Предоставлять диапазон чисел достаточный для решения конкретной задачи, в том числе для промежуточных результатов
3. Обеспечивать переносимость программ между различными архитектурами вычислительных машин
4. Обеспечивать побитовую повторяемость результата при запуске на различных архитектурах и в различном программном окружении
5. Обеспечивать побитовую повторяемость результата при запуске на различном количестве вычислительных узлов
6. Обеспечивать достаточную эффективность (количество памяти, энергоэффективность, и т.д.)

Рациональные числа

- Всюду плотное множество в R. Т.е. Рациональным числом можно приблизить любую точку на действительной оси
- Выполняются ВСЕ базовые законы. Коммутативность, ассоциативность, дистрибутивность
- Основная проблема – техническая, рост размера числа

Проблема роста количества разрядов числа

- Рост количества разрядов представляет основную проблему с точки зрения эффективности, а не памяти
- Параллельные вычисления на уровне базовых операций

Математические перспективы

- Сложение/вычитание – параллельно
- Умножение – параллельно
- Деление – параллельно
- Сравнение – параллельно

Реализация требований

Для реализации описанных требований необходимы три составляющих:

1. Работа с представлением числа на уровне обращений к памяти (работа с битами)
2. Базовые арифметические операции непосредственно работающие с памятью (алгоритмы)
3. Операции ввода-вывода и конвертации в десятичную систему

Уровень работы с памятью

- 1) Доступ к битам представления числа
- 2)Выделение памяти для результатов арифметической операции

Реализация зависит устройства, на котором хранится число, НЕ зависит от выбранной модели числа.

Уровень базовых арифметических операций

- Для разных устройств различные реализации, в зависимости от архитектуры
- Алгоритмы не зависят от реализации нижнего слоя работающего с памятью (используют его интерфейс)

Реализация зависит от архитектуры устройства, выбранного алгоритма, от модели числа.

Операции ввода/вывода и конвертации

- Операции ввода/вывода и конвертации суть арифметические операции

Реализация зависит только реализации арифметических операций.

Реализация

- Уровень работы с памятью
  - malloc() / free() for C language and CPU
  - cudaMalloc() / cudaFree() for Cuda C language and GPU
- Арифметические операции
  - В зависимости от архитектуры устройства
- Ввод / вывод
  - Опирается на арифметические операции

Реализация рационального числа

Рациональное число:

```
class rational {private: overlong nmr; // Numerator
private:         overlong dmr; // Denumerator
…
};
class overlong { private: static ArifRealization realization; //Link to arithmetics realization
private:         MemHandle mhandle; //Memory handle member(very smart pointer)
private:         int32  leng; //Length of the number(amount of digits)
private:         int32 sgn; //Sign of the number(-1 or +1 only!)
…
};
```

- Длинное целое реализовано как число в позиционной системе счисления с основанием 2^{32}. При этом для класса рациональное число неважно как именно работает overlong, можно использовать систему остаточных классов, любое другое представление
- Основанием 2^{32} позволяет эффективно использовать память и быстрые битовые операции

Пример сложения двух длинных чисел

Сложение на CPU

```
addCPU(const d_t *buffA, int32 LA,const d_t *buffB, int32 LB,d_t *buffC,int32 &newleng, d_t &carry){
        int32 i;
        int64 tmp=0;
        for(i=0;i<LB;i++) {
                tmp+=(int64)buffA[i]+(int64)buffB[i];
                buffC[i]=tmp&MAX_DIGIT;
                tmp>>=BIT_IN_DIGIT;
        }
        for(;i<LA;i++) {
                tmp+=buffA[i];
                buffC[i]=tmp&MAX_DIGIT;
                tmp>>=BIT_IN_DIGIT;
        }
        carry=tmp&MAX_DIGIT;
        newleng= carry ? LA+1: LA;
}
```

Фрагмент сложения на GPU

```
void ArifRealization::add(const d_t *d_buffA,int32 LA,const d_t *d_buffB,int32 LB,d_t *d_buffC,int32 &newleng,d_t &carry){
        int threadsPerBlock = 512;  int blocksPerGrid = (LA + threadsPerBlock - 1) / threadsPerBlock;
        d_t *bGCarrydeviceptr=NULL,*ansCarryhostptr=new d_t[1];
        int32 *carry_flagdeviceptr=NULL,*carry_flaghostptr=new int32[1];
        checkCudaErrors( cudaMalloc((void**)&bGCarrydeviceptr, sizeof(d_t)*(blocksPerGrid+1)) );
        checkCudaErrors( cudaMemset((void*)bGCarrydeviceptr, 0, sizeof(d_t)*(blocksPerGrid+1))) ;
        checkCudaErrors( cudaMalloc((void**)&carry_flagdeviceptr, sizeof(int32)) );
        checkCudaErrors( cudaMemset((void*)carry_flagdeviceptr, 0, sizeof(int32)) );
        NumberAdd_part_1 <<< blocksPerGrid, threadsPerBlock >>>
(const_cast<d_t*>(d_buffA), LA, const_cast<d_t*>(d_buffB), LB, d_buffC, bGCarrydeviceptr,carry_flagdeviceptr);
checkCudaErrors( cudaMemcpy(carry_flaghostptr,carry_flagdeviceptr,sizeof(int32), cudaMemcpyDeviceToHost) );
        if(*carry_flaghostptr) /*есть перенос между группами*/{
                int group_size=threadsPerBlock,LCarry=blocksPerGrid;
                blocksPerGrid = (blocksPerGrid + threadsPerBlock - 1) / threadsPerBlock;
NumberAdd_part_2 <<< blocksPerGrid, threadsPerBlock>>> (d_buffC, bGCarrydeviceptr, group_size, LCarry, LA);
        }
        checkCudaErrors( cudaMemcpy(ansCarryhostptr,&bGCarrydeviceptr[LCarry], sizeof(d_t),
cudaMemcpyDeviceToHost) );
        carry = *ansCarryhostptr;
        newleng= carry ? LA+1: LA;
        delete[] ansCarryhostptr;
        checkCudaErrors( cudaFree(bGCarrydeviceptr) );
        checkCudaErrors( cudaFree(carry_flagdeviceptr) );
}
```

Тестирование параллельных алгоритмов (Сравнение двух чисел)

TABLE I. COMPARISON OF THE INTEGERS ON DIFFERENT ARCHITECTURES

L	Time in milliseconds					Speedup	
	CPU(S)	Fer(S)	Kep(S)	Fer(P)	Kep(P)	Fer(P)/CPU	Kep(P)/CPU
10^1	0.0001	0.130	0.120	0.146	0.148	0.0068	0.0067
10^2	0.0001	0.135	0.130	0.192	0.138	0.0052	0.0073
10^3	0.0001	0.320	0.410	0.155	0.149	0.0065	0.0067
10^4	0.0200	2.260	3.040	0.200	0.210	0.1000	0.0952
10^5	0.1900	21.62	29.91	0.230	0.220	0.8370	0.8636
10^6	1.9000	218.0	301.1	0.521	0.360	3.6538	5.2777
10^7	19.000	—	—	3.170	1.581	6.2776	12.5950
10^8	198.10	—	—	—	—	—	—

Тестирование параллельных алгоритмов (Сложение двух чисел)

TABLE II. ADDITION OF THE INTEGERS ON DIFFERENT ARCHITECTURES

L	Time in milliseconds					Speedup	
	CPU(S)	Fer(S)	Kep(S)	Fer(P)	Kep(P)	Fer(P)/CPU	Kep(P)/CPU
10^1	0.0001	0.300	0.180	0.240	0.240	0.0041	0.0041
10^2	0.0001	0.300	0.200	0.240	0.240	0.0041	0.0041
10^3	0.0001	0.610	0.620	0.240	0.240	0.0041	0.0041
10^4	0.0200	3.910	3.040	0.240	0.240	0.0041	0.0041
10^5	0.5000	37.60	4.010	0.540	0.700	0.9259	0.7142
10^6	6.0100	369.5	42.20	0.880	0.860	6.8181	6.9767
10^7	60.000	—	409.2	4.300	2.76	14.000	21.814
10^8	602.10	—	—	—	—	—	—

Рациональное число удовлетворяет требованиям

- Предоставлять диапазон числе достаточный для решения задачи
- Портируется на любую архитектуру
- Обеспечивает побитовую повторяемость результата
- Пригодно для параллельных вычислений
- При реализации на GPU может быть энергоэффективно? (нужен глубокий анализ)!

Диапазон

- Зависит от максимального размера участка памяти, который может быть выделен на физическом устройстве. Например, для числа в позиционной системе счисления по основанию 2^{32} минимальное ненулевое число представимое на современном GPU $1/(2^{(8\,(2^{31}-1))+1}-1) \sim 10^{-5.1*10^7}$.

Переносимость

- Реализация требует наличия на устройстве операций работы с памятью и ассемблерных `add`, `mul` and `div` с целыми числами.

Повторяемость результатов на различных архитектурах

- Целочисленные операции всегда и везде дают одинаковые результаты (в пределах размера регистров).

Повторяемость результатов в параллельных вычислениях

- Целочисленная арифметика дает верифицируемые и повторимые результаты. Масштабирование не влияет на результат (в пределах объема памяти устройств).

Эффективность

- Эффективность, как количество операций в секунду, над рациональными будет безусловно ниже, чем при использовании IEEE754. Однако рациональные числа могут обеспечить сходимость численных методов для плохо обусловленных задач.

Примеры использования рациональных чисел

- Решение неустойчивых задач, например, решение систем линейных уравнений с матрицами Гильберта, Вандермонда, Годунова и т.д.
- Переход к интервальным методам решения задач, например, использование *интервального погружения**.

Операции выводящие из поля рациональных чисел

- Sin, Cos, Sqrt
- Решение проблемы – разработка численных методов с минимальным контролируемым использованием «неточных» операций. Арифметика рациональных чисел и компьютерное исследование интегральных уравнений Максимов В.П. Соросовский образовательный журнал 1999 г.

Дальнейшая работа

1. Реализация удобного интерфейса для воплощения в компьютерных моделях ”качественных” представлений числа.
2. Реализация высокоэффективной низкоуровневой рациональной арифметики для уровня операционной системы (криптография) с сохранением переносимости на любую архитектуру.

Выводы

- Чтобы реализовать корректное компьютерное число нужно
  - Реализовать работу с памятью устройства
  - Реализовать алгоритмы работы с представлением
  - Обеспечить операции ввода вывода
- Реализация математически корректного числа продемонстрирована на примере рационального числа.

Публикации

- V. Golodov, “Properties of mathematical number model provided exact computing,” in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2017, pp. 225–228.
- V. Golodov, “Interval regularization approach to the firordt method of the spectrophotometric analysis of the non-separated mixtures,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9553, pp. 201–208, 2016.
- A. V. Panyukov and V. A. Golodov, “Parallel Algorithms of Integer Arithmetic in Radix Notations for Heterogeneous Computation Systems with Massive Parallelism,” *Bull. South Ural State Univ. Ser. “Mathematical Model. Program. Comput. Software,”* vol. 8, no. 2, pp. 117–126, 2015.