

Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Параллельные численные методы

Особенности реализаций

При поддержке компании Intel

Бастраков С.И., Донченко Р.В.
Кафедра математического обеспечения ЭВМ

Содержание

- Введение
- Архитектура x86-64
- Язык C
- Компиляторы Microsoft и Intel
- Литература



Введение

- Поддержку вычислительной системой стандарта IEEE-754 можно разложить на три компоненты:
  - поддержка на аппаратном уровне
  - поддержка языком программирования
  - поддержка транслятором языка
- Мы рассмотрим пример реализации каждой из этих компонент.



Архитектура x86-64

- x86-64 (также AMD64, Intel 64, EM64T, x64) — 64-битная архитектура системы команд процессоров, выпускаемых в основном компаниями Intel и AMD.
- В настоящее время используется на значительной части серверов и большинстве домашних компьютеров.
- Является расширением старой 32-битной архитектуры x86 (IA-32), и обратно совместима с ней.
- Поддерживает по меньшей мере два способа работы с плавающей запятой.



x87…

- x87 — название старой архитектуры сопроцессоров для работы с числами с плавающей запятой; сейчас обозначает функциональность x86-процессоров, повторяющую эту архитектуру.
- Предоставляет 8 регистров (fpr0-fpr7) особого формата:
  - Общая длина представления k = 80 бит.
  - Длина поля мантиссы t = 63 бита.
  - Длина поля порядка w = 15 бит.
  - Оставшийся бит (Integer bit) содержит цифру слева от запятой, и должен быть согласован с остальными полями.



x87

- Не считая лишнего поля, формат x87 устроен согласно стандарту.
- sNaN отличается от qNaN рекомендуемым образом — у qNaN установлен старший бит мантиссы.
- Инструкции x87 позволяют аппаратно выполнять большинство обязательных операций и часть рекомендуемых.
- Кроме того, x87 предоставляет средства для изменения нескольких аспектов своей работы…



x87 – точность

- Формат по умолчанию имеет точность больше, чем у стандартных форматов binary32 и binary64.
- С одной стороны, это означает, что вычисления производятся точнее, чем с этими форматами.
- С другой стороны, программа, которая работала с x87 может перестать возвращать корректный результат на архитектурах, использующих стандартные форматы.
- Для повышения предсказуемости x87 позволяет уменьшить используемую длину поля мантиссы до 23 или 52 бит, что соответствует binary32 и binary64.
- Впрочем, длина поля порядка остаётся неизменной, а значит, совместимость достигается не полная.



x87 – округление и исключения

- x87 позволяет выбрать один из четырёх стандартных способов округления — к ближайшему (чётному в случае ничьи), вниз, вверх и к нулю.
- x87 поддерживает 5 стандартных исключений, а также шестое, которое генерируется, если на вход операции поступает субнормальное число.
- Каждое исключение может быть маскировано (выполняется стандартная обработка по умолчанию) и немаскировано (в этом случае генерируется процессорное исключение). Обработка по умолчанию устанавливает соответствующий исключению флаг в статусном регистре FSW.



3DNow!

- 3DNow! — хронологически второй способ осуществлять операции с плавающей запятой.
- Работает с 64-битными регистрами mmx0-mmx7, которые вмещают в себя два числа в формате binary32.
- Эти регистры перекрывают x87-регистры fpr0-fpr7, поэтому нельзя одновременно использовать x87 и 3DNow!
- 3DNow! был реализован только на процессорах AMD и мало популярен; AMD в настоящий момент советует использовать вместо него SSE.



SSE

- SSE (Streaming SIMD Extensions), SSE2, SSE3, SSSE3, SSE4 — семейство расширений x86/x86-64 для поддержки векторных операций.
- Предоставляет 16 128-битных регистров XMM0-XMM15.
- Каждый может содержать 2 значения формата binary64 или 4 — формата binary32 (а также целые числа в ряде форматов).
- Инструкции позволяют работать как со всеми значениями регистра, так и с одним.
- В отличие от x87, поддерживаются только обязательные операции.



SSE и x87

- SSE можно использовать как замену x87.
- Компиляторы под x86-64 часто так и делают, поскольку SSE и SSE2 поддерживаются во всех процессорах x86-64.
- В отличие от x87, SSE работает со стандартными форматами, повышая предсказуемость.
- SSE поддерживает те же четыре режима округления и те же шесть исключений, что и x87.
- Кроме того, поддерживаются опции, позволяющие отключить поддержку субнормальных чисел. Как правило, операции с субнормальными числами выполняются медленнее, чем над нормальными и нулями, и иногда без них можно обойтись.



x86-64 – будущее

- На момент написания этих строк Intel и AMD внедряют в свои процессоры дополнительные расширения для SSE:
  - AVX (Advanced Vector Extensions):
    - новые 256-битные регистры YMM0-YMM15, перекрывающие XMM-регистры;
    - инструкции для работы с ними;
  - F16C — конвертирование значений в/из формата binary16. Операции для этого формата всё ещё не поддерживаются.
  - FMA3, FMA4 — операция fusedMultiplyAdd. Раньше эта операция не была реализована аппаратно, так как появилась только во втором издании IEEE 754. Реализации Intel и AMD похожи, но несовместимы.



C

- C — один из самых популярных языков для написания вычислительных программ.
- Определяется стандартом ISO/IEC:
  - ISO/IEC 9899:1990 (он же C90 или C89)
  - ISO/IEC 9899:1999 (он же C99)
- C90 реализован повсеместно, C99 не очень.
- Многие из нововведений C99 имеют отношение к арифметике с плавающей запятой.



C и IEEE 754

- Стандарт C не требует, чтобы действительные типы и операции с ними удовлетворяли IEEE 754. Для реализаций, где это так (а обычно это так), стандарт определяет набор дополнительных требований.
- На момент публикации C99 существовало только первое издание IEEE 754, поэтому часть операций из второго издания в C отсутствует.
- Конкретные реализации C часто добавляют свои расширения, которые могут иметь отношение к арифметике с плавающей запятой.



C – типы

- C предоставляет три так называемых вещественных типа с плавающей запятой:
  - float — binary32;
  - double — binary64;
  - long double — может быть эквивалентен binary64 или быть шире;
    - Например, 80-битный тип x87.



C – константы

- Способы задания вещественных констант:
  - В десятичном виде:
    - 1.5
    - 1.5e2 $=1.5\times10^{2}$
  - В шестнадцатеричном виде[C99]:
    - 0x1.ab3p-3 $=1.ab3_{16}\times2^{-3}$
    - Можно точно и компактно записать любое представимое число.
- Тип константы по умолчанию — double. Для float используется суффикс f/F; для long double — l/L.
- Для специальных значений есть макросы в <math.h>: HUGE_VAL (+∞), INFINITY[C99] (+∞) и NAN[C99] (qNaN). Стандарт не требует поддерживать sNaN.



C – функции

- Обязательные операции IEEE 754 реализуются операторами языка, а также функциями и макросами из стандартной библиотеки:
  - <math.h> — математические операции;
  - <stdio.h>, <stdlib.h> — преобразования в/из строк;
  - <fenv.h> — манипуляции окружением;
- В C99 почти все такие функции продублированы для всех трёх вещественных типов, например fabsf (float) – fabs (double) – fabsl (long double).
- Часть рекомендуемых операций доступна в виде функций из <math.h>. C, в отличие от IEEE 754, не требует считать эти функции с максимально возможной точностью.



C – окружение

- Текущий режим округления можно установить/определить с помощью функций <fenv.h> fesetround[C99] и fegetround[C99] (кроме режима округления к нулю в случае ничьи, который не поддерживается).
- Для исключений стандартом поддерживается только обработка по умолчанию (с флагами исключений можно работать с помощью функций из <fenv.h>), но предполагается, что реализации могут добавлять свои способы обработки.
- Программа, которая обращается к функциям из <fenv.h> должна включать директиву #pragma STDC FENV_ACCESS ON [C99], в противном случае поведение программы не определено.



C – промежуточные значения…

- C позволяет производить вычисления с плавающей точкой в формате большей точности, чем тип операндов. Результат должен приводиться к конкретному типу только при записи в переменную этого типа.
- Цель этого разрешения в том, чтобы реализации могли использовать самый эффективный способ выполнения операций, дающий заданную или лучшую точность.
- Например, компилятору можно использовать для промежуточных результатов расширенный тип x87, не выгружая каждый промежуточный результат в память для приведения типа.



C – промежуточные значения…

- Макрос <float.h> FLT_EVAL_METHOD^{C99} указывает, в каком формате представляются промежуточные значения:
  - 0 — в том же, что и аргументы;
  - 1 — double для аргументов float и double; long double для long double;
  - 2 — long double;
  - другое значение — какой-то другой вариант.



C – промежуточные значения

- Таким образом, значения двух выражений могут не совпадать, даже если они эквивалентны:
- float f = 1.0f / 3.0f;
  - Результат деления записывается в переменную, значит, округляется до типа binary32.
- bool b = f == 1.0f / 3.0f;
  - Результат деления используется только как промежуточный результат и может находиться в формате, зависящем от реализации.
  - b может быть как true, так и false.



C – сжатие выражений

- C разрешает реализациям посчитать сложное выражение как одну операцию, то есть вычислить его математически точное значение и один раз округлить его для представления в целевом формате.
- Например, заменить последовательность из умножения и сложения на операцию fusedMultiplyAdd.
- Можно запретить сжимать выражения с помощью директивы #pragma STDC FP_CONTRACT OFF[C99].
- В отсутствие этой директивы компилятор не обязан сжимать все выражения, которые способен. Если корректность результата зависит от использования сложной операции, её нужно использовать явно (в примере — функцией fma[C99]).



C – оптимизации

- Результат работы программы могут изменить возможные оптимизации.
- Или нет… C разрешает только преобразования программы, не влияющие на её поведение.
  - x * 0.5 ≡ x / 2.0
  - x + 0 ≢ x (если x равен -0, x + 0 равно +0)
  - x == x ≢ true (NaN не равен самому себе)
  - (a + b) + c ≢ a + (b + c)
- Но компиляторы для обеспечения скорости работы могут предоставлять режимы агрессивной оптимизации, игнорирующие эти ограничения. Иногда даже по умолчанию.



Компиляторы

- Мы рассмотрим поддержку арифметики с плавающей запятой в двух популярных компиляторах:
  - Компилятор Microsoft (MC), который входит в состав Microsoft Visual Studio 2008.
    - Создаёт программы под Windows для архитектур x86 и x86-64 (а также IA-64).
    - Поддерживает C90, но некоторые расширения повторяют часть функциональности C99.
  - Компилятор Intel (IC), который входит в состав Intel Composer XE 2011.
    - Создаёт программы под Windows, Linux, Mac OS X для тех же архитектур (мы рассмотрим Windows-версию, которая совместима с MC).
    - Поддерживает большую часть C99.



Компиляторы – наборы инструкций

- MC поддерживает x87, SSE и SSE2.
  - При компиляции под x86 используется только x87. SSE/SSE2 можно включить с помощью опции /arch.
  - При компиляции под x86-64 SSE и SSE2 включены сразу и не выключаются.
- IC также поддерживает SSE3, SSSE3, SSE4.1, SSE4.2 и AVX.
  - По умолчанию включены SSE и SSE2. Остальные наборы включаются опциями /arch и /Qx.
- Опции разрешают использовать конкретный набор команд, а не заставляют.



Компиляторы – векторизация…

- В отличие от MC, IC умеет автоматически применять векторные инструкции SSE, чем зачастую обеспечивает скомпилированным программам большую производительность, чем MC.
- Если вы используете MC, или вам кажется, что вы можете использовать векторные инструкции лучше, чем это делает IC, но при этом вы не хотите писать код на ассемблере, то вы можете векторизовать свой код вручную с помощью интринсик (intrinsics).
- Интринсики — это типы данных и функции, предоставляемые рассматриваемыми компиляторами, которые напрямую, или почти напрямую, соответствуют регистрам и инструкциям из семейства SSE.



Компиляторы – векторизация

- Например, следующая функция возвращает сумму квадратов элементов массива:

```
#include <xmmintrin.h>
float sumsquare_intrinsics(const float * a, int n)
{
    __m128 sum, element;
    int i;
    float result[4];

    sum = _mm_setzero_ps();

    for (i = 0; i < n; i += 4)
    {
        element = _mm_load_ps(a + i);
        sum = _mm_add_ps(sum,
            _mm_mul_ps(element, element));
    }

    _mm_store_ps(result, sum);

    return result[0] + result[1] + result[2] + result[3];
}
```



Компиляторы – режимы

- В обоих компиляторах есть опция /fp, которая позволяет менять семантику работы с плавающей запятой. Значения этой опции делятся на три группы, внутри каждой из которых можно выбрать ровно один из вариантов:
  - precise, fast, strict;
  - source, double, extended (только в IC);
  - except, except-;
- Последнюю группу мы рассмотрим в разделе об исключениях.



Компиляторы – управление оптимизацией…

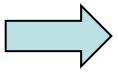
- Группа precise, fast, strict управляет тем, какие оптимизации разрешено выполнять компилятору.
- /fp:precise означает, что компилятору разрешается выполнять только оптимизации, не изменяющие результат вычислений. Это означает, что выражения будут выполняться «как написано», и значит, программист может пользоваться свойствами арифметики с плавающей запятой.



Компиляторы – управление оптимизацией…

- /fp:fast даёт компилятору разрешение оптимизировать код с использованием математических свойств операций.
- С одной стороны, компилятору даётся большая свобода при оптимизации, и итоговый код может получиться быстрее.
- С другой стороны, компилятор может вносить ошибки, связанные с несоблюдением семантики операций с плавающей запятой.

```
float error(float a, float b)
{
    return (a + b) - b - a;
}
```



```
float error(float a, float b)
{
    return 0;
}
```

Компиляторы – управление оптимизацией

- Автоматическая векторизация может производиться только с /fp:fast, так как она меняет порядок операций, а значит, и результат.
- MC по умолчанию использует /fp:precise, IC — /fp:fast.
- Третий вариант из этой группы, /fp:strict, эквивалентен /fp:precise, но с отключенной директивой fp_contract и включенной — fenv_access. Кроме того, /fp:strict также устанавливает /fp:except.
- fp_contract — аналог STDC FP_CONTRACT из C99, разрешает/запрещает сжатие выражений.
- fenv_access — аналог STDC FENV_ACCESS, разрешает/запрещает доступ к окружению.



Компиляторы — промежуточные значения…

- Группа /fp:source, double и extended определяет точность, с которой считаются промежуточные результаты.
  - source — формат результатов операций совпадает с форматом аргументов операций.
  - double — операции вычисляются в формате binary64.
  - extended — операции вычисляются в расширенном 80-битном формате x87.
- По умолчанию для x86 используется /fp:double, для x86-64 /fp:source.
- Заметим, что тип long double в обоих компиляторах синонимичен double.
- Эта опция имеет смысл только в режиме /fp:strict.



Компиляторы — промежуточные значения…

- Последствия выбора того или иного варианта зависят от используемого набора команд.
- /fp:extended форсирует использование x87 (в SSE нет такого формата).
- При использовании /fp:double или /fp:source с типом double и набором инструкций x87 вычисления проводятся не со стандартным форматом binary64, а с 80-битным форматом x87, мантисса которого урезана до 53 бит. Это означает, что используется более широкий диапазон порядков, чем в binary64.



Компиляторы — промежуточные значения

- Самая неудачная комбинация — это /fp:source при использовании x87 и типа float. Регистры при этом настроены на 53-битную мантиссу (на случай выражений типа double). для обеспечения точности float результат каждой операции должен быть выгружен из регистра в память и загружен обратно. Теряется как точность, так и производительность.
- При использовании /fp:double и extended формат промежуточных значений может не совпадать с форматом конечных. Однако, компилятор обязан привести промежуточное значение к конечному типу. Этих преобразований может быть достаточно много (и они могут замедлять работу программы), если выражений вычисляется много, а сами они короткие.



Компиляторы – окружение…

- Окружение арифметики с плавающей запятой включает в себя следующие настройки:
  - включена ли замена субнормальных чисел нулями;
  - включена ли обработка исключений по умолчанию;
  - текущий режим округления;
  - текущая используемая ширина поля мантиссы;
- Это те настройки, которые позволяют менять x87 и SSE. Кроме того, в окружение входят флаги исключений.
- Окружением можно манипулировать функцией <float.h> _controlfp_s. Она меняет параметры одновременно x87 и SSE, но, естественно, она не может задать те параметры, которых кто-то из них не поддерживает (т.е. ширину мантиссы для SSE).



Компиляторы – окружение

- Если в программе читается или изменяется окружение, она должна быть скомпилирована с директивой #pragma fenv_access (on).

```
void something();
float square, cube;

square = x * x;
something();
cube = x * x * x;

printf("%f %f\n", square, cube);
```

- Если не указать директиву, компилятор может использовать square при вычислении cube, хотя вызов something() мог изменить режим округления.



Компиляторы – исключения…

- Изначально все исключения обрабатываются по умолчанию — при их срабатывании устанавливается соответствующий флаг. Эти флаги можно прочитать с помощью функций <float.h> _statusfp и _clearfp (последняя очищает все флаги после получения их значения).

```
float a[] = { 1e30f };
float result;

_clearfp();
result = sumsquare(a, 1);
printf("Result: %f\n", result);
if (_statusfp() & _SW_OVERFLOW)
    puts("Overflow!");
```



Компиляторы – исключения…

- Чтобы обработать исключение прямо в момент возникновения, нужно сделать три вещи:
  - Указать ключ компиляции /fp:except. Этот ключ гарантирует, что исключение генерируется сразу после того, как возникла исключительная ситуация. /fp:except подразумевается в случае /fp:strict, и не совместим с /fp:fast.
- Отключить обработку по умолчанию для нужного исключения с помощью _controlfp_s. Предварительно стоит очистить соответствующий флаг, чтобы исключение не возникло сразу же.
- Обработать исключение самому с помощью предложения __try.



Компиляторы – исключения…

- Пример:

```
float a[] = { 1e30f };
float result;
unsigned old_control;

_clearfp();
_controlfp_s(&old_control, ~_EM_OVERFLOW, _MCW_EM);

__try
{
    result = sumsquare(a, 1);
    printf("Result: %f\n", result);
}
__except(1)
{
    puts("Overflow!");
}
```



Компиляторы – исключения

- Предложение __try … __except является расширением, реализованным в MC и IC.
- Это не try … catch из C++; __try работает как в C, так и в C++.
- __try работает независимо от try, и отличается синтаксисом: в части __except указывается не тип, а выражение, от результата которого зависит поведение программы. См. документацию Microsoft.
- Программисты на C++, которые уже используют исключения C++, могут использовать трансляцию исключений с плавающей запятой в исключения C++ с помощью пользовательской функции-транслятора. См. функцию _set_se_translator.



Компиляторы – элементарные функции

- Оба компилятора реализуют полный набор элементарных функций из заголовка <math.h> C90; в IC также входит альтернативный заголовок <mathimf.h>, который позволяет использовать и функции из C99.
- Если опции разрешают векторизацию, IC может заменить вызовы элементарных функций в цикле на реализации, оперирующие непосредственно с SSE-регистрами, из встроенной библиотеки SVML (Short Vector Math Library). Если программисту не нравится качество векторизации, он может использовать функции SVML вручную — они используются аналогично простым SSE-интринсикам.



VML

- Функции из SVML работают с SSE-регистрами, которые вмещают небольшое число значений.
- Если программа производит операции над большим числом элементов, то ускорить её производительность можно с помощью библиотеки VML (Vector Mathematical Functions Library), которая входит в Intel Composer.
- VML позволяет эффективно производить как простые арифметические операции, так и расчёты элементарных функций над массивами большой длины.
- Кроме того, VML предлагает несколько режимов вычислений, которые позволяют программисту выбрать между точностью вычислений и их скоростью.



Заключение

- Рассмотренные вопросы
  - Наборы инструкций x87, SSE
  - Типы данных, операции в С
  - Факторы, влияющие на результат выражений
  - Режимы работы компиляторов
  - Окружение арифметики с плавающей запятой
  - Обработка исключений



Литература

1. IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008.
2. International Organization for Standardization, International Electrotechnical Commission. ISO/IEC 9899:1999. Programming languages — C.
3. Intel® 64 and IA-32 Architectures Software Developer Manuals. [http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html].
4. Microsoft Visual C++ Floating-Point Optimization. [http://msdn.microsoft.com/en-us/library/aa289157(v=vs.71).aspx].



Авторский коллектив

- Бастраков Сергей Иванович,
  лаборант НИЛ кафедры Математического обеспечения ЭВМ факультета ВМК ННГУ.
  sergey.bastrakov@gmail.com
- Донченко Роман Владимирович,
  лаборант НИЛ кафедры Математического обеспечения ЭВМ факультета ВМК ННГУ.
  ss.donchenko@itlab.unn.ru

