Звіт з лабораторної роботи №1

з архітектури обчислювальних систем

студента групи К-24

Романуса Теодора

Лабораторна робота виконана мовою C++ з використанням Visual Studio Community 2015. Повністю код лабораторної роботи можна знайти тут.

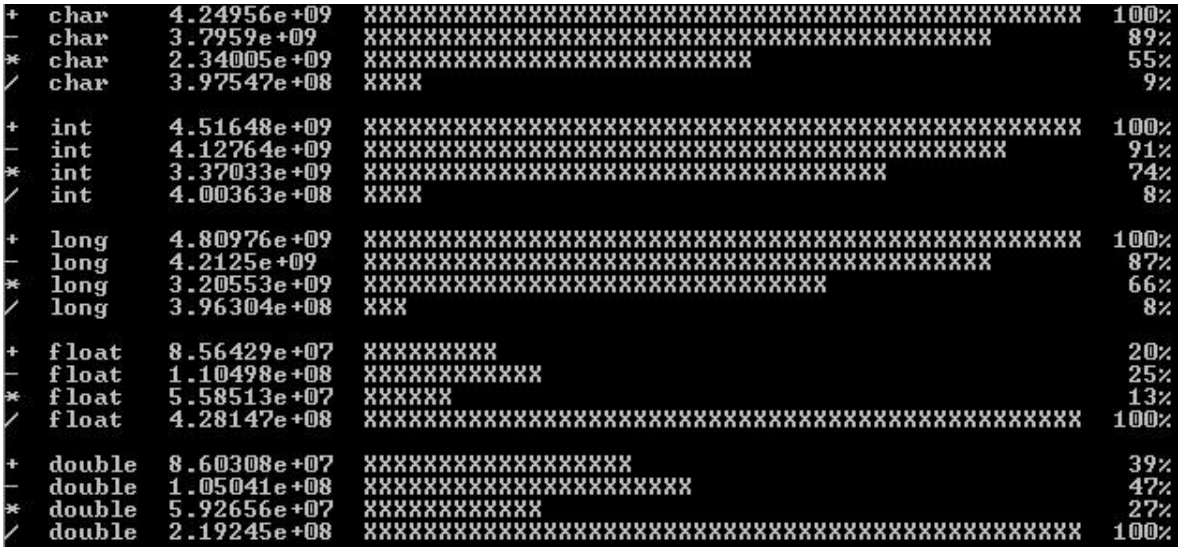
Тестування проводилось на комп'ютері з процесором Intel Pentium 2020M з двома ядрами і тактовою частотою 2,4 GHz з установленою Windows 10 Pro x64.

Кількості арифметичних операцій в циклах брались рівними тактовій частоті процесора ($2{,}4 * 10^9$ операцій). Для дійсних чисел кількість операцій в 100 разів менша.

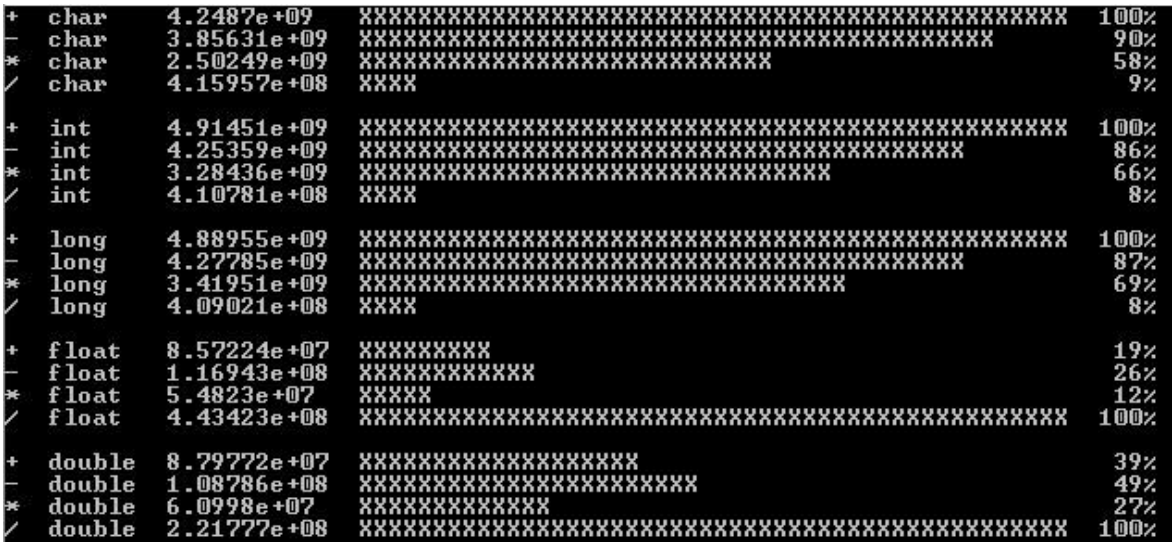
Для більшої точності обчислень і унеможливлення впливу на результат роботи фонових програм тестування проводилось перед запуском процесу explorer.exe відразу після ввімкнення комп'ютера. Також програма тестувалась в звичайному режимі.

Результати виявились не зовсім очікуваними.

В звичайному режимі:



Запуск через Shell:



Як видно, для цілих чисел все досить очікувано: додавання виконується найшвидше, трохи повільніше віднімання, наполовину повільніше множення, а ділення – взагалі на порядок повільніше.

Отримані ж дані для дійсних чисел (float і double) зовсім не відповідають очікуванням: найшвидше виконується ділення, потім віднімання, тоді додавання і множення.

Варто вказати, що автор тестував програму на дійсних числах більше 100 разів, в звичайному режимі і в Shell'і, і завжди результати були такими, як на зображеннях (відхилення від цих значень в більшості випадків не перевищувало 10%).

Аналізуючи результати для дійсних чисел, логічно виникають два питання:

1) Чому віднімання виконується швидше, ніж віднімання? (на 20% і більше)
2) Чому ділення виконується так швидко?

Відповіді на ці запитання так і не було отримано. Далі буде детальніше розглянуто код алгоритму для дійсних чисел, а також, чому отримані за його допомогою результати повинні бути достовірними.

1) Алгоритм обчислення кількості операцій за секунду для дійсних чисел суттєво відрізняється від алгоритму для цілих чисел тим, що в ньому необхідно додатково передбачити неможливість переповнення числа (тобто, якщо для чисел типу int, наприклад, можна написати a=b+c, то для дійсних так робити на можна, бо якщо числа b і c будуть достатньо великими, змінна a може прийняти значення inf (нескінченність)).
   Саме тому для дійсних чисел до кожної операції ще була додана функція остачі від ділення, тобто код a=b+c перетворився на a=(b+c)%m, де m – досить велике число, в цій лабораторній його значення становить INT_MAX. Очевидно, що в другому варіанті значення змінної ніколи не стане нескінченним. Аналогічно був модифікований пустий цикл:

```
t1 = std::chrono::high_resolution_clock::now();
for (long i = 0; i < ITER; ++i)
{
    a1 = fmod(e4, MULTIPLIER) + 2;
    a2 = fmod(b2, MULTIPLIER) + 3;
    a3 = fmod(a4, MULTIPLIER) + 4;
    a4 = fmod(f4, MULTIPLIER) + 5;
    a5 = fmod(b5, MULTIPLIER) + 6;
    b1 = fmod(b1, MULTIPLIER) + 7;
    b2 = fmod(c2, MULTIPLIER) + 8;
    b3 = fmod(h3, MULTIPLIER) + 9;
    b4 = fmod(f2, MULTIPLIER) + 2;
    b5 = fmod(b2, MULTIPLIER) + 3;
    c1 = fmod(d1, MULTIPLIER) + 4;
    c2 = fmod(h2, MULTIPLIER) + 5;
    c3 = fmod(d4, MULTIPLIER) + 6;
```

Пустий цикл для дійсних чисел

```
t1 = std::chrono::high_resolution_clock::now();
for (long i = 0; i < ITER; ++i)
{
    a1 = fmod(b1 - c2, MULTIPLIER) + 2;
    a2 = fmod(b2 - d2, MULTIPLIER) + 3;
    a3 = fmod(a4 - c3, MULTIPLIER) + 4;
    a4 = fmod(f4 - d5, MULTIPLIER) + 5;
    a5 = fmod(b5 - g3, MULTIPLIER) + 6;
    b1 = fmod(a1 - b3, MULTIPLIER) + 7;
    b2 = fmod(c2 - d5, MULTIPLIER) + 8;
    b3 = fmod(h3 - e3, MULTIPLIER) + 9;
    b4 = fmod(f2 - a4, MULTIPLIER) + 2;
    b5 = fmod(b2 - d5, MULTIPLIER) + 3;
    c1 = fmod(d1 - e5, MULTIPLIER) + 4;
    c2 = fmod(h2 - f2, MULTIPLIER) + 5;
```

Цикл з відніманням

```
t1 = std::chrono::high_resolution_clock::now();
for (long i = 0; i < ITER; ++i)
{
    a1 = fmod(b1 + c2, MULTIPLIER) + 2;
    a2 = fmod(b2 + d2, MULTIPLIER) + 3;
    a3 = fmod(a4 + c3, MULTIPLIER) + 4;
    a4 = fmod(f4 + d5, MULTIPLIER) + 5;
    a5 = fmod(b5 + g3, MULTIPLIER) + 6;
    b1 = fmod(a1 + b3, MULTIPLIER) + 7;
    b2 = fmod(c2 + d5, MULTIPLIER) + 8;
    b3 = fmod(h3 + e3, MULTIPLIER) + 9;
    b4 = fmod(f2 + a4, MULTIPLIER) + 2;
    b5 = fmod(b2 + d5, MULTIPLIER) + 3;
    c1 = fmod(d1 + e5, MULTIPLIER) + 4;
    c2 = fmod(h2 + f2, MULTIPLIER) + 5;
```

Цикл з додаванням

В кожному випадку до змінних додавались числа, щоб унеможливити множення на 0 і 1 (аналогічно зображеним виглядає і цикл з множенням, для оптимізації часу роботи програми для цих трьох операцій був створений лише один пустий цикл).

Виходячи з того, що код для циклу з додаванням і відніманням ідентичний, за винятком одного знаку, дуже дивно, що віднімання виконується швидше.

2) Ділення відрізняється від інших арифметичних операцій тим, що ділити на нуль не можна. Тому для ділення повинен бути свій алгоритм обчислення і свій пустий цикл:

```
for (long i = 0; i < ITER; ++i)
{
    a1 = b1 + 2;
    a2 = b2 + 3;
    a3 = a4 + 4;
    a4 = f4 + 5;
    a5 = b5 + 6;
    b1 = a1 + 7;
    b2 = c2 + 8;
    b3 = h3 + 9;
    b4 = f2 + 2;
    b5 = b2 + 3;
    c1 = d1 + 4;
    c2 = h2 + 5;
```

Пустий цикл

```
for (long i = 0; i < ITER; ++i)
{
    a1 = b1 / c2 + 2;
    a2 = b2 / d2 + 3;
    a3 = a4 / c3 + 4;
    a4 = f4 / d5 + 5;
    a5 = b5 / g3 + 6;
    b1 = a1 / b3 + 7;
    b2 = c2 / d5 + 8;
    b3 = h3 / e3 + 9;
    b4 = f2 / a4 + 2;
    b5 = b2 / d5 + 3;
    c1 = d1 / e5 + 4;
    c2 = h2 / f2 + 5;
    c3 = d4 / a3 + 6;
```

Цикл з діленням

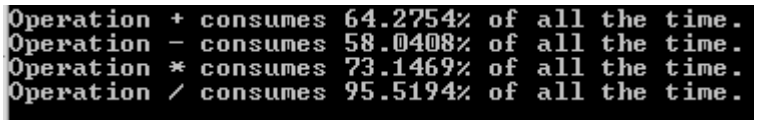
Знову ж таки, в циклі з діленням до змінних додавались ще додатково числа, щоб не було inf (дуже велике число ділиться на дуже мале).

Зважаючи на простоту алгоритму для ділення, здавалося б, він має показати достовірний результат. Хоча те, що відображається в консолі, важко раціонально описати.

Далі автор задався питанням: а, може, під час роботи його алгоритму все-таки деякі змінні приймають значення inf? Особливо для ділення такий результат був би виправданим. Але, тим не менше, після численних перевірок з розставлення точок

зупинки усі значення виявляються скінченними. Отже, якщо існує помилка, воно точно не в діленні на 0.

Під час аналізу результатів помилка може бути також в тому, що час виконання пустого циклу і робочого циклу (з арифметичною операцією) можуть не сильно відрізнятись, тоді похибка обчислення буде дуже великою. Хоча, проти цього твердження виступало те, що в процесі численних прогонів тестів усі відповіді були дуже стабільними (усі в межах 10% навколо якогось значення). І справді, якщо вивести на екран відношення часу, який в циклі йде на виконання конкретної арифметичної операції, до часу роботи цього циклу, отримаємо такі результати (float):



Очевидно, що коли час виконання арифметичних операцій відносно часу виконання пустого циклу того самого порядку, про похибку такого роду й мови не може йти.