УДК 004.021

Новокшонов А. К., аспірант

A. K. Novokshonov, graduate student

Аналіз ефективності «машин, що складають»

Performance analysis of “addition machines”

Київський національний університет імені Тараса Шевченка, факультет кібернетики, 03680, м. Київ, пр. Академіка Глушкова, 4д,
e-mail: andrey.novokshonov@ukr.net

Taras Shevchenko National University of Kyiv, Faculty of Cybernetics, 03680, Kyiv, 4d Academician Glushkov avenue,
e-mail: andrey.novokshonov@ukr.net

У даній роботі представлені результати обчислювального експерименту, метою якого є уточнення реальної ефективності арифметичних алгоритмів цілочисельної «машини, що складає». «Машина, що складає» - це математична абстракція, введена Р. Флойдом та Д. Кнутом, суть якої полягає у тому, що лише за допомогою операцій додавання, віднімання, порівняння, присвоювання та обмеженої кількості регістрів можна з прийнятною обчислювальною ефективністю виразити більш складні операції, такі як знаходження лишку за модулем, множення, знаходження найбільшого спільного дільника, піднесення до степеня за модулем. Особливістю даного експерименту є використання арифметики довільної точності, що може бути корисним для використання у криптографічних алгоритмах. В результаті дослідження були отримані практичні оцінки складності основних алгоритмів цілочисельної «машини, що складає».

Ключові слова: додавання, віднімання, найбільший спільний дільник, піднесення до степеня, числа Фібоначчі, C++, Python, GMP.

This paper presents the results of the numerical experiment, which aims to clarify the real performance of integer "addition machine" arithmetic algorithms. "Addition machine" is a mathematical abstraction, introduced by R. Floyd and D. Knuth. The essence of "addition machine" is the following: using only operations of addition, subtraction, comparison, assignment and a limited number of registers it is possible to calculate more complex operations such as finding the residue modulo, multiplication, finding the greatest common divisor, exponentiation modulo with reasonable computational efficiency. One of the features of this implementation is the use of arbitrary precision arithmetic, which may be useful in cryptographic algorithms. Practical complexity estimates of main integer "addition machines" algorithms have been computed and compared with original theoretical complexity estimates given by R. Floyd and D. Knuth. Graphical performance comparison results of “addition machines” algorithms implemented both in C++ with GMP library and Python programming languages have been presented too.

Key words: addition, subtraction, greatest common divisor, exponentiation, Fibonacci numbers, C++, Python, GMP.

Статтю представив чл.-кор. НАНУ, д. ф.-м. н., проф. Анісімов А. В.

Вступ

Розвиток криптографії та поява великої кількості сучасних малопотужних обчислювальних пристроїв спричиняють необхідність більш детального дослідження і розробки арифметичних алгоритмів, які могли б бути ефективно реалізовані програмно і володіли б необхідними властивостями. Однією з таких цікавих математичних моделей є «машина, що складає», яка має наступні властивості.

Розглянемо обмежений набір операцій: присвоєння, додавання, віднімання та порівняння двох цілих чисел. Виникає задача ефективної реалізації інших, більш складних операцій, використовуючи тільки наведені вище прості операції.

У роботі [1] Р. Флойдом та Д. Кнутом була запропонована така математична абстракція, як *«машини, що складають»* (англ. *addition machines*), і було доведено, що з їх допомогою можлива реалізація складних операцій з лінійним уповільненням. До таких складних операцій відносять знаходження лишку за модулем, множення, ділення, знаходження найбільшого спільного дільника та піднесення до степеня за

модулем. Варто зазначити, що за допомогою цих складних операцій виражається значна частина арифметичних алгоритмів, які використовуються у сучасних криптографічних протоколах.

Таким чином, «машини, що складають» - це математична модель, яка являє собою обчислювальний пристрій з обмеженою кількістю регістрів, над якими можна здійснювати лише такі операції: 1) введення: *read x*; 2) виведення: *write x*; 3) присвоювання: $x \leftarrow y$; 4) додавання: $x \leftarrow x + y$; 5) віднімання: $x \leftarrow x - y$; 6) порівняння: $x \geq y$.

«Машини, що складають» можуть працювати як з цілими числами, так і з дійсними. У даній роботі усі операції вважаємо такими, що здійснюються над цілими числами, тому описаний пристрій називається цілочисельною «машиною, що складає».

Метою даної роботи є дослідження ефективності цілочисельної «машини, що складає» при її реалізації за допомогою мов програмування C++ і Python та використанні арифметики довільної точності. Використання арифметики довільної точності були вибрані у зв'язку із можливим подальшим застосуванням результатів даного дослідження у криптографії.

Методика дослідження

У даній роботі обмежимося розглядом і порівнянням таких операцій над цілими числами: 1) множення двох чисел; 2) піднесення до степеня за модулем; 3) знаходження лишку за модулем; 4) знаходження найбільшого спільного дільника (НСД).

У роботі [1] були наведені наступні теоретичні оцінки складності операцій, які можуть бути реалізовані за допомогою «машини, що складає» (табл. 1). Також дані оцінки можна знайти у роботі [2].

Таблиця 1

Число команд регістрової машини при виконанні арифметичних операцій

Операція	**Час виконання**
лишок $x \bmod y$	$O(\log(x/y))$
множення xy	$O(\log(\min(\|x\|,\|y\|))$
найбільший спільний дільник $НСД(x,y)$	$O(\log(\max(x,y)/НСД(x,y))$
експонента $x^y \bmod z$	$O((\log y)(\log z)+\log(x/z))$

Основною ідеєю алгоритму знаходження лишку від цілочисельного ділення, який був запропонований Р. Флойдом та Д. Кнутом у [1], є використання представлення Фібоначчі замість традиційного бінарного представлення. Відомо, що будь-яке невід'ємне ціле число може бути представлене сумою чисел Фібоначчі. Далі ключовими моментами є те, що, по-перше, за допомогою «машини, що складає» можна легко переходити від пари чисел Фібоначчі $<F_t, F_{t+1}>$ до наступної пари $<F_{t+1}, F_{t+2}>$ використанням лише однієї операції додавання, або до попередньої пари $<F_{t-1}, F_t>$ лише за допомогою однієї операції віднімання. Числа Фібоначчі зростають експоненційно, а саме тому можуть бути використані як аналоги степенів двійки.

У командах регістрової «машини, що складає» алгоритм знаходження лишку від цілочисельного ділення **x mod y** (*P*1) має наступний вигляд [1]:

```
P1: read x; read y; {вважається, що x ≥ 0, y > 0}
if x >= y then
  begin z ← y;
  repeat <y, z> ← <z, y + x> until not x >= z;
  repeat if x >= y then x ← x - y;
    <y, z> ← <z - y, y>;
  until y >= z;
  end;
write x.
```

Операція $<y, z> \leftarrow <z, y + x>$ позначає одночасне присвоювання $y \leftarrow z$ та $z \leftarrow y + x$.

Аналогічно алгоритм обчислення **x[y/z]** (*P*2) у командах регістрової «машини, що складає» є наступним [1]:

```
P2: read x; read y; read z;  { вважається, що y ≥ 0,
z > 0 }
w ← w - w;
if y >= z then
  begin u ← x; v ← z;
  repeat <u, x> ← <x, u + x>; <v, z> ← <z, v + z>;
  until not y >= z;
  repeat if y >= v then <w, y> ← <w + u, y - v>;
    <u, x> ← <x - u, u>; <v, z> ← <z - v, v>;
  until  v >= z;
  end;
write w.
```

Алгоритм обчислення найбільшого спільного дільника **НСД(x,y)** (*P*3) у командах регістрової «машини, що складає» виглядає наступним чином [1]:

*P*3: *read x; read y;* { вважається, що $x > 0, y \geq 0$ }
$z \leftarrow y$; $z \leftarrow z + z$;
while not $y >= z$ *do*
begin while $x >= z$ *do* $\langle y, z\rangle \leftarrow \langle z, y + z\rangle$
repeat if $x >= y$ *then* $x \leftarrow x - y$;
$\langle y, z\rangle \leftarrow \langle z - y, y\rangle$
until $y >= z$;
$\langle x, y\rangle \leftarrow \langle y, x\rangle$; $z \leftarrow y$; $z \leftarrow z + z$;
end;
write x.

І, нарешті, останній важливий алгоритм обчислення $\mathbf{x^y}$ **mod z** (*P*5) за допомогою «машини, що складає» виконується наступним чином, враховуючи допоміжний алгоритм *P*4, який використовується для обчислення представлення Фібоначчі [1]:

*P*4: $u \leftarrow 1$; $v \leftarrow 1$; $w \leftarrow y$;{ $u = F_l, v = F_{l+1}, l = 1$ }
repeat $\langle u, v\rangle \leftarrow \langle v, u + v\rangle$
until not $w >= v$; { $u = F_l, v = F_{l+1}, y \geq u$ }
$r \leftarrow 1$; $s \leftarrow 1$; $t \leftarrow t - t$; { $u = F_l, v = F_{l+1}, l = \lambda y$ }
repeat if $w >= u$ *then*
begin $w \leftarrow w - u$; $t \leftarrow t + s$;
end;
$\langle u, v\rangle \leftarrow \langle v - u, u\rangle$; $\langle r, s\rangle \leftarrow \langle s, r + s\rangle$
{ $l \leftarrow l - 1$ }
until $u >= v$.

*P*5: *read x; read y; read z;*
$\langle r, s, t\rangle \leftarrow \langle F_{\lambda y}, F_{\lambda y+1}, y^R\rangle$
$x \leftarrow x \bmod z$; $w \leftarrow x$; $u \leftarrow 1$; { $x = x_l, w = x_{l+1}, l = 1$ }
repeat if $t >= r$ *then*
begin $t \leftarrow t - r$; $u \leftarrow (uw) \bmod z$;
end;
$\langle r, s\rangle \leftarrow \langle s - r, r\rangle$; $\langle x, w\rangle \leftarrow \langle w, (xw) \bmod z\rangle$;
{ $l = l + 1$ }
until $r >= s$;
write u.

Результати дослідження

Метою експериментального дослідження є уточнення наведених вище теоретичних оцінок при практичній реалізації алгоритмів.

Для реалізації та порівняння швидкодії алгоритмів були використані наступні мови програмування, що входять до першої п'ятірки світового рейтингу мов програмування [3] – це мова C++ з бібліотекою арифметики довільної точності GMP [4] та мова Python, яка набрала найбільшу популярність за останні 5 років [5], з використанням вбудованої арифметики довільної точності.

Таким чином, було виконане у тому числі порівняння швидкодії цих двох мов

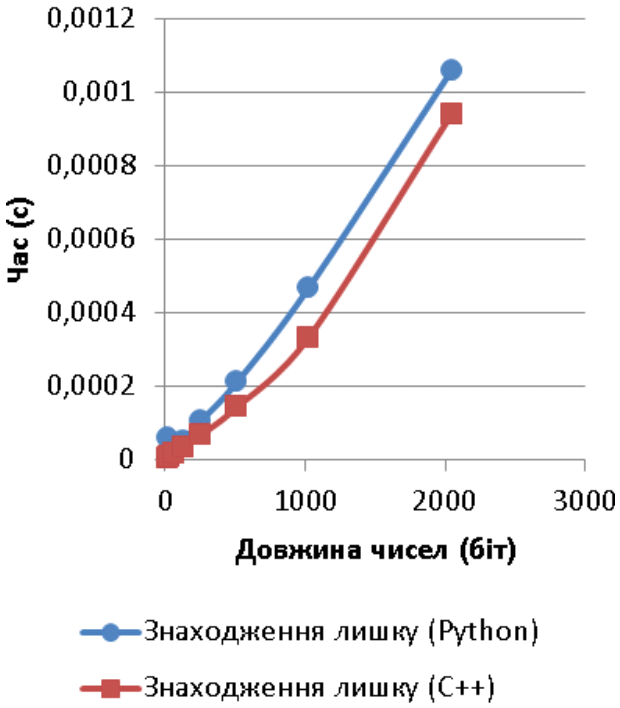

Рис. 1 Порівняння часу виконання операції знаходження лишку від ділення при реалізації на мовах програмування C++ та Python

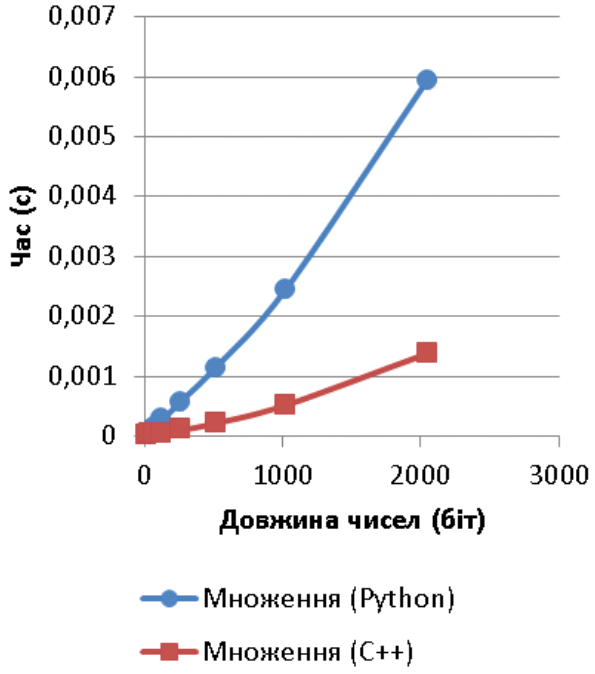

Рис. 2 Порівняння часу виконання операції множення при реалізації на мовах програмування C++ та Python

програмування (звичайно, у межах даних арифметичних алгоритмів). Деякі більш глибокі результати порівняння швидкодії мов

програмування C++ та Python можна знайти у роботі [6].

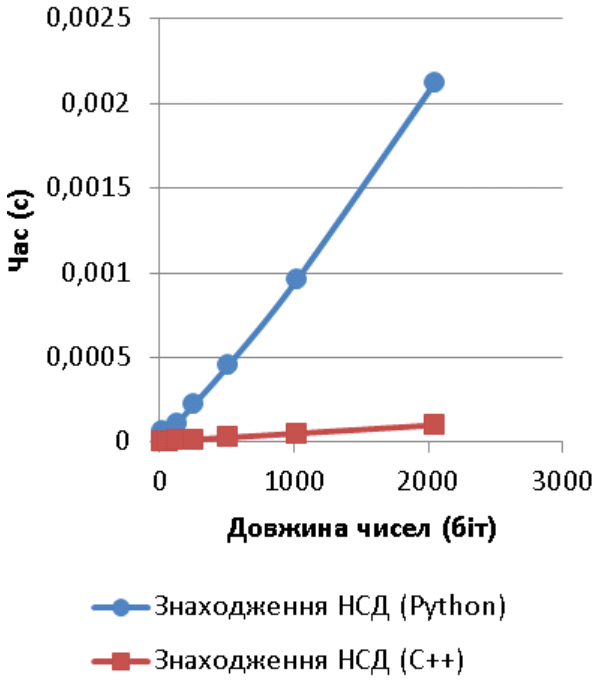

Рис. 3 Порівняння часу виконання операції знаходження найбільшого спільного дільника при реалізації на мовах програмування C++ та Python

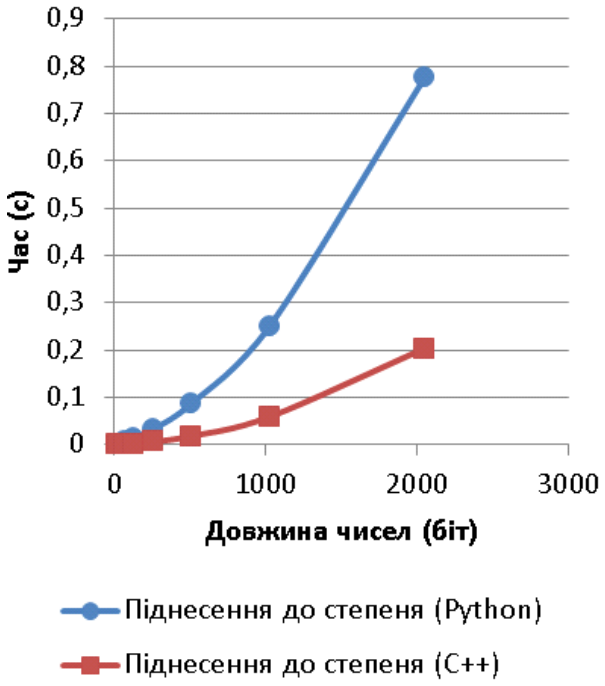

Рис. 4 Порівняння часу виконання операції піднесення до степеня при реалізації на мовах програмування C++ та Python

Характеристики персонального комп'ютера, на якому було проведене тестування реалізованих алгоритмів, такі: процесор Intel Pentium Duo 2.8 ГГц, 4 Гб оперативної пам'яті, ОС Ubuntu 14.04 LTS x64. Версії програмного забезпечення: GCC 4.8.4, GMP 5.1.3, Python 2.7.6.

На рис. 1 наведені результати порівняння швидкодії алгоритму *P*1 (x mod y) на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python.

На рис. 2 представлені результати порівняння швидкодії алгоритму *P*2 (x[y/z]) на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python.

На рис. 3 наведені результати порівняння швидкодії алгоритму *P*3 (НСД(x,y)) на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python.

На рис. 4 представлені результати порівняння швидкодії алгоритму *P*5 ($x^y \bmod z$) на персональному комп'ютері при реалізації його за допомогою мов програмування C++ та Python. Для зручності тестування в якості степеня y була обрана константа 16.

Висновки

В результаті дослідження були отримані чисельні характеристики ефективності алгоритмів «машини, що складає», побудовані відповідні графіки та практичні оцінки складності алгоритмів, що дозволило експериментально підтвердити та уточнити теоретичні оцінки складності даних алгоритмів, виведені Р. Флойдом та Д. Кнутом.

Одним із практичних застосувань розглянутих алгоритмів може бути їх ефективне використання на спеціально побудованому апаратному забезпеченні, наприклад, для схеми RSA [1, 7], основою якої є операція піднесення до степеня за модулем ($x^y \bmod z$), а також застосування у криптографічних протоколах для випадків, коли набір доступних операцій є обмеженим.

Цікавим фактом є те, що швидкодія алгоритму НСД(x,y) на C++ є повністю однаковою як для вбудованої операції mpz_gcd() з бібліотеки GMP [4], так і для його реалізації за допомогою «машини, що складає». Для Python-реалізації алгоритму НСД(x,y) ситуація схожа: стандартна функція fractions.gcd() лише у 2 рази швидша за реалізацію за допомогою «машині, що складає».

Також експериментальним шляхом було визначено, що:

- для C++/GMP середня швидкодія вбудованих складних операцій є у 47 разів

більшою, ніж їх аналогів, реалізованих за допомогою «машини, що складає»;
- для Python середня швидкодія вбудованих складних операцій є у 228 разів більшою, ніж їх аналогів, реалізованих за допомогою «машини, що складає»;
- реалізація на C++/GMP арифметичних алгоритмів «машини, що складає» є у 3,8 разу швидшою за аналогічну реалізацію на Python.

Отже, результати експериментального порівняння надають можливість зробити висновок про те, що реалізація арифметичних алгоритмів над цілими числами довільної точності на C++ є в середньому у 3,8 разу швидшою за Python-реалізацію, але у той же час процес програмування на C++ є складнішим.

Втім для кожного класу задач необхідно використовувати правильні інструменти та керуватися визначеними пріоритетами (швидкість розробки, розмір виконуваного файлу, швидкодія програми, доступні платформи та бібліотеки), тоді процес розробки та отримані результати будуть найоптимальнішими.

Загалом, інтерпретатор Python показав досить високу швидкодію, що разом з вбудованою арифметикою довільної точності, зручністю програмування та різноманіттям програмних модулів, у тому числі математичних, робить мову Python одним з досить перспективних інструментів для дослідників в області інформатики та математики.

Список використаних джерел

1. Floyd R. Addition Machines / R. Floyd, D. Knuth. // SIAM J. Comput.–1990.– №19(2).– P. 329–340.
2. Анісімов А. В. Алгоритмічна теорія великих чисел. Модулярна арифметика великих чисел / А. В. Анісімов // К.: Академперіодика, 2001.
3. TIOBE Index | Tiobe - The Software Quality Company [Електронний ресурс] – Режим доступу до ресурсу: http://www.tiobe.com/tiobe_index.
4. The GNU MP Bignum Library [Електронний ресурс] – Режим доступу до ресурсу: https://gmplib.org/.
5. PYPL PopularitY of Programming Language index [Електронний ресурс] – Режим доступу до ресурсу: https://pypl.github.io/PYPL.html.
6. Prechelt L. An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl / L. Prechelt. // IEEE Computer. – 2000. – №33(10). – P. 23–29.
7. Rivest R. A method for obtaining digital signatures and public-key cryptosystems / R. Rivest, A. Shamir, L. Adleman. // Communications of the ACM. – 1978. – №21(2). – P. 120–126.

References

1. FLOYD, R. and KNUTH, D. (1990). *Addition Machines*. SIAM J. Comput., 19(2), pp.329-340.
2. ANISIMOV, A. V. (2001). Algorithmic Theory of Large Numbers. Modular Arithmetic of Large Numbers, Akademperiodika, Kyiv.
3. TIOBE.COM, (2015). *TIOBE Index | Tiobe - The Software Quality Company*. [online] Available at: http://www.tiobe.com/tiobe_index [Accessed 10 Dec. 2015].
4. GMPLIB.ORG, (2015). *The GNU MP Bignum Library*. [online] Available at: https://gmplib.org/ [Accessed 10 Dec. 2015].
5. PYPL.GITHUB.IO, (2015). *PYPL PopularitY of Programming Language index*. [online] Available at: https://pypl.github.io/PYPL.html [Accessed 10 Dec. 2015].
6. PRECHELT, L. (2000). *An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl*. IEEE Computer, 33(10), pp.23-29.
7. RIVEST, R., SHAMIR, A. and ADLEMAN, L. (1978). *A method for obtaining digital signatures and public-key cryptosystems*. Communications of the ACM, 21(2), pp.120-126.

Надійшла до редколегії 13.12.15