УДК 004.2(045)

Гамаюн В. П., д-р техн. наук,
Яременко К. П.

АНАЛІЗ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ АДИТИВНОГО ПЕРЕТВОРЕННЯ З РОЗРЯДНО-ЛОГАРИФМІЧНИМИ КОДАМИ

Інститут комп'ютерних технологій Національного авіаційного університету

Проведений аналіз обчислювальних алгоритмів РЛ додавання, віднімання та множення. На основі статистичного дослідження запропонована структура швидкодіючого обчислювача.

Вступ

Підвищення продуктивності ЕОМ передбачає, насамперед, апаратну підтримку алгоритмів, що виконуються. Така структуризація виконується на основі аналізу загальних операційних властивостей алгоритмів обробки, а також способів обробки даних. Урахування таких факторів обумовлює рівень продуктивності комп'ютерних засобів.

Загальні положення РЛ кодування

Нехай A – двійкове число розрядністю n представлене у форматі з фіксованою комою:

$$A = \sum_i a_i p_i ,$$

де $a_i = \{0,1\}$, $p=2$ – основа системи числення.

Кожен розряд a_i, що не дорівнює нулю ($a_i p_i \neq 0$), можна представити у вигляді номеру позиції (розряду) N_i, в якій цей розряд знаходиться. Таким чином, двійкове число A представляється набором номерів ненульових розрядів:

$$A \to A^N = \{N_1, N_2, \ldots, N_i, \ldots N_n\},$$

де N_i – номер ненульового розряду ($a_i p_i \neq 0$).

Для представлення числа з плаваючою комою можна збільшити розрядність зображення кожного номера ненульового розряду на один біт і додати до кожного номера значення порядка, якщо порядок є показником степені двійки.

Узагальнену структуру РЛ даних можна представити у вигляді:

$SignA\ Q\ SignN_1\ N_1{+}p\ SignN_2\ N_2{+}p\ \ldots\ SignN_n\ N_n{+}p.$

Перехід від двійкової форми представлення до форми, де кожний ненульовий розряд подано своїм номером, визначається однозначно і реалізується без додаткових функціональних перетворень – виконується операція підстановки по кожному ненульовому розряду.

В таблиці 1 наведені кількості двійкових розрядів n_2 і розряди РЛ представлення $n_{РЛ}$, що отримані при кодуванні у відповідних n_2, а також діапазони зміни чисел D при розрядних сітках $n_{РЛ}$. При РЛ представленні діапазон оброблюваних чисел збільшується, що забезпечує коректне виконання операцій за рахунок виключення процедур округлення та нормалізації.

Таблиця 1

n_2	8	16
$n_{РЛ}$	510	131070
D	$2^{-255} \le A \le 2^{+255}$	$2^{-65536} \le A \le 2^{+65536}$

Внаслідок перерахованих переваг розрядно-логарифмічне кодування доцільно використовувати у комп'ютерних засобах при високоточних розрахунках чисел великих діапазонів.

Аналіз обчислювальних алгоритмів обробки з РЛ кодами. Для розробки структурно-апаратної частини ЕОМ або системи виконаємо аналіз алгоритмів базових адитивних перетворень з метою визначення принципів організації.

Аналіз операції додавання

Доданки представляються у вигляді файлів (наборів) розрядно-логарифмічних (РЛ) кодів. Операнди оброблюються, починаючи з молодших розрядів, з виконанням наступних процедур. Виконується поелементне порівняння чисел

$A \rightarrow \{sign,\ Q(A),\ N_1,\ N_2 ..\ N_i,\ ...N_m\}$ і $B \rightarrow \{sign,\ Q(B),\ N_1,\ N_2 ..\ N_j,\ ...N_k\}$: якщо $N_i \neq N_j$, то в результат S записуються обидва елементи в порядку зростання. Інакше $(N_i = = N_j)$ відбувається наступне: деякій проміжній змінній присвоюється значення N_i+1, після чого отримане число порівнюється з наступними елементами чисел A та B, тобто з N_{i+1} і N_{j+1}, якщо рівності не виявлено, то в результат записується N_i+1. У іншому випадку проводиться ще одна ітерація порівнянь. Слід зазначити випадок, коли доданки рівні між собою: $A=B$. У цьому випадку результат C дорівнює одному з доданків, зі зсувом на одиницю вправо, тобто

$S \rightarrow N_1+1,\ N_2+1,\ N_3+1, ..., N_i+1, ...N_m+1...$

Приклад. Нехай дані $A \rightarrow \{0,5,7.6.3.2.0\}$ і $B \rightarrow \{0,3,6.4.1\}$, необхідно знайти їхню суму. Порівнявши поелементно A та B, видно що в обох числах є елемент $\{6\}$, збільшуємо його на $1 \rightarrow 7$ і порівнюємо з наступними елементами A і B: в A елементів уже немає, а в B є елемент рівний 7, значить ще раз збільшуємо на $1 \rightarrow 8$. Ні A, ні B не містять більше елементів, отже остаточний результат буде наступним:

$$C \rightarrow \{0,6,8.4.3.2.1.0\}.$$

Для виконання операції додавання в з операндами в РЛ представленні варто використовувати схему порівняння на $]\log_2(n+1)[$ розряд, що відрізняє виконання додавання в РЛ представленні від традиційних способів підсумовування.

При підсумовуванні чисел A і B читаються коди, що відповідають номерам значущих одиниць, починаючи з молодших. Коди подаються на схему порівняння й у відповідності зі значеннями виходів, операційний пристрій формує значення порозрядних сум і переносів.

Необхідно відзначити, що при виконанні РЛ підсумовування можливе прискорення, яке відбувається завдяки властивості РЛ кодування, якщо коди менше мінімального значущого розряду одного з доданків, то вони не обробляються, а заносяться в суму без змін. Надалі такі кроки повторюються до зчитування всіх значущих одиниць у доданках.

Закон функціонування вузла послідовного суматора для РЛ даних наведений у таблиці 2.

Таблиця 2

$P^N=0$	$S^N_i(k)$	$P^N_{i+1}(k+1)$	P^N_{i+1}	$S^N_i(k)$	$P^N_{i+1}(k+1)$
$A^N_i>B^N_k$	A^N_i,B^N_k	---	$A^N_i>B^N_k$	B^N_k	$1(A^N_{i+1})$
$A^N_i=B^N_k$	---	$1(A^N_{i+1})$	$A^N_i=B^N_k$	A^N_i	$1(A^N_{i+1})$
$A^N_i<B^N_k$	B^N_i,A^N_k	---	$A^N_i<B^N_k$	A^N_i	$1(A^N_{i+1})$

Лістинг програми додавання з підрахунком порівнянь розрядів та зведень подібних наведений нижче. В якості доданків за допомогою генератора випадкових чисел були сформовані масиви заданої довжини Q з обмеженням діапазону до Q^2.

```
#include "StdAfx.h"
#include ".\rlnumber.h"
CRlNumber::CRlNumber(int digits, int max)
        :m_digits(digits),
        m_ComparisonCount(0),
        m_IncrementCount(0),
        m_max(max+1)
{       m_arr = new int[m_digits];
        memset(m_arr, -1,
sizeof(int)*m_digits);}
CRlNumber::~CRlNumber(void)
{       delete[] m_arr;}
void
CRlNumber::GenerateRandom(unsigned
seed)
```

```
{srand(seed);
for (int i=0; i<m_digits; i++)
{double r = double(rand())/RAND_MAX;
        int num = int(r*m_max);
        while (!Check(i, num))
        {r = double(rand())/RAND_MAX;
        num = int(r*m_max);}
        m_arr[i] = num;}
        Sort();}
int CRlNumber::GetDigit(int i)
{       return m_arr[i];}
void CRlNumber::Sort(void)
{       bool work = true;
        while(work)
        {work = false;
         for (int i=0; i<(m_digits-1); i++)
        {if (m_arr[i+1] < m_arr[i])
        {int t = m_arr[i+1];
        m_arr[i+1] = m_arr[i];
        m_arr[i] = t;
        work = true;    }}}}
bool CRlNumber::Check(int i, int num)
{       for (int j=0; j<=i; j++)
        {if (m_arr[j] == num)
         return false; // coincidence   }
        return true; // ok }
void CRlNumber::Sum(CRlNumber& n1,
CRlNumber& n2)
{       for (int i=0; i<m_digits/2; i++)
        {m_arr[i] = n1.GetDigit(i);}
        for (int i=0; i<m_digits/2; i++)
        {m_arr[m_digits/2 + i] =
n2.GetDigit(i);}
        Sort();
        int res_index = 0;
        int* res = new int[m_digits];
 memset(res, 0, sizeof(int)*m_digits);
 for (int pos=0; pos<m_digits; pos++)
 {if (pos == m_digits-1) // last digit
 {res[res_index++] = m_arr[pos]; // save result
break;          // exit}
m_ComparisonCount++;
if (m_arr[pos] != m_arr[pos+1])
{res[res_index++] = m_arr[pos]; // save result }
else
{m_IncrementCount++;
m_arr[pos+1]++;       Sort(); }}
memcpy(m_arr, res, sizeof(int)*m_digits);
delete res;}
int CRlNumber::GetComparisonCount(void)
{       return m_ComparisonCount;}
int CRlNumber::GetIncrementCount(void)
{       return m_IncrementCount;}// debug
void CRlNumber::_Set(int n1, int n2, int n3,
int n4)
{       memset(m_arr, 0,
sizeof(int)*m_digits);
        m_arr[0] = n1;
        m_arr[1] = n2;
        m_arr[2] = n3;
        m_arr[3] = n4;}
```

На графіку (рис. 1) наведені значення кількості зведень подібних m_1 та порівнянь m_2 для доданків довжиною Q, що дорівнює 8, 16, 32 та 64 розрядам. Процентні співвідношення кількості порівнянь зі зведенням подібних та кількості порівнянь наведені у таблиці 3.

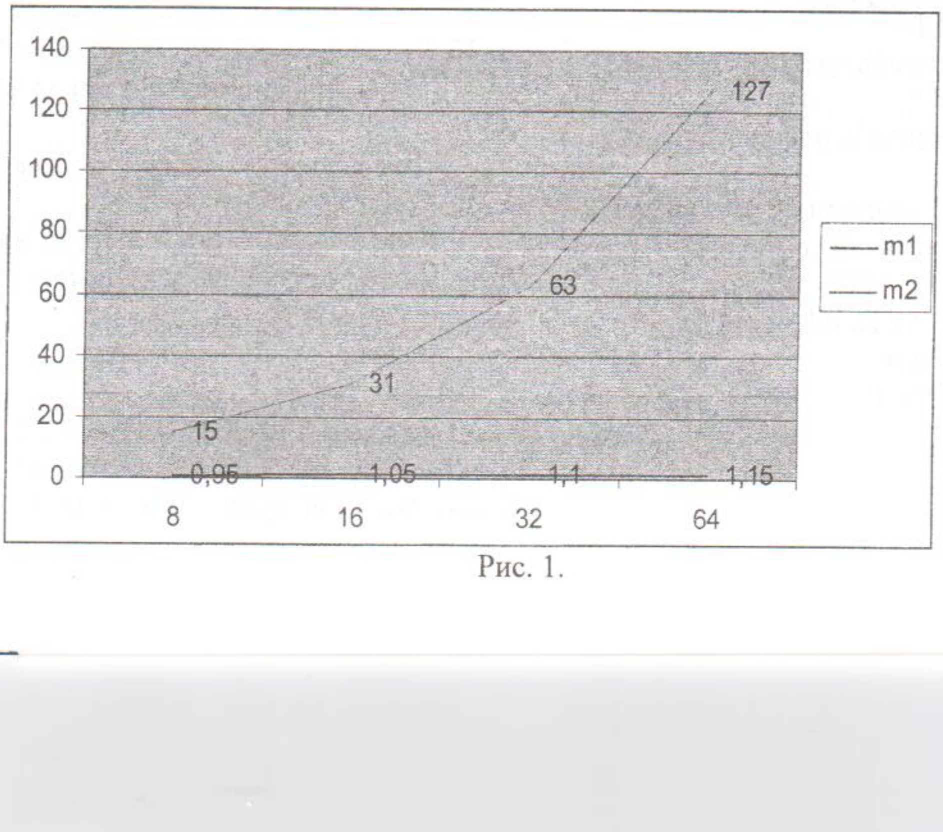

Рис. 1.

Таблиця 3.

	Q=8	Q=16	Q=32	Q=64
m_1	6	3,4	1,6	0,9
m_2	94	96,6	98,4	99,1

Аналіз операції віднімання

У загальному випадку операція віднімання зводиться до операції додавання зменшуваного без старшого розряду і від'ємника, представленого в оберненому коді. За аналогією з двійковою системою, на останньому етапі виконання операції необхідно додати молодший розряд числа заданого діапазону (для цілих чисел – 0.). Обернений код формується таким чином: береться старший розряд зменшуваного, зменшений на одиницю, і записуються всі розряди в порядку убування, крім тих, що є у від'ємнику.

Приклад. Нехай дані $A \rightarrow \{0,3,8.3.2\}$ і $B \rightarrow \{0,4,6.5.4.3\}$, необхідно знайти їхню різницю. A і B додатні числа й $A>B$. Записуємо від'ємник в оберненому коді: старший розряд зменшуваного 9, 9-1=8; $B_{зв} \rightarrow \{0,5,8.7.2.1.0\}$. Складаємо A без старшого розряду $\{0,2,3.2\}$ і $B_{зв}$, проміжний результат $C' \rightarrow \{0,4,7.2.1.0\}$, кінцевий результат $C=C'+0. \rightarrow \{0,3,7.4.2\}$.

У наведеному нижче фрагменті показано отримання оберненого кода зменшуваного та сам алгоритм віднімання.

```
if ((yRLREV=(RLType) malloc(sizeof(int)*
sz)) == NULL) {
   printf("Not enough memory to allocate
buffer Minus\n");
   exit(1);  /* terminate program if out of
memory */  }
iyr=sz-1;         // на старший эл-т yRLREV
 // Начальное значение r = Старший раз-
ряд xRL-1.
  for(r=(*(xRL+ix)-1); iy!=1; r--) {
   if(r!=*(yRL+iy)) {
     *(yRLREV+iyr)=r;
     iyr--;    }
   else
     iy--; }
  *(yRLREV+iyr)=*(yRL+2); // last elem of
yRL
  *(yRLREV)=0;
  *(yRLREV+1)=sz;
 if(ix==2) {
   if(rezRL==xRL) FreeRL(xRL);
   if(rezRL==yRL) FreeRL(yRL);
 return(yRLREV);  }
// xRLwithoutFirst
 if ((xRLwoFirst=(RLType)
malloc(sizeof(int)*(ix))) == NULL) {
   printf("Not enough memory to allocate
buffer Minus\n");
   exit(1);  /* terminate program if out of
memory */  }
  *(xRLwoFirst)=0;
  *(xRLwoFirst+1)=ix;
 for(;ix>2;ix--)
    *(xRLwoFirst+ix-1)= *(xRL+ix-1);
rezRL=Plus (rezRL, xRLwoFirst, yRLREV);
```

Дослідження щодо кількості порівнянь та зведень подібних не мають змісту, оскільки віднімання зводиться до додавання, тобто дані цілком ідентичні.

Аналіз операції множення

Операція множення виконується відповідно до правила порозрядної операції множення:

$N_i * N_k = N_i + N_k$

де N_i, N_k – ненульові розряди операндів.

При виконанні множення двох чисел A і B:

- кожен елемент числа A складається з усіма елементами числа B;
- серед отриманого набору номерів ненульових розрядів добутку виконується зведення подібних;
- здійснюється упорядкування елементів добутку;

Приклад. Нехай дані $A \rightarrow \{0,2,2.1\}$ і $B \rightarrow \{0,2,3.1\}$, необхідно знайти їхній добуток. Часткові суми: {3.2} і {5.4}. Результат *C* дорівнює сумі {3.2} і {5.4}, тобто {0,4,5.4.3.2}.

Нижче наведений повний лістинг функції множення.

```
RLType RLMult(RLType rezRL, RLType xRL, RLType yRL) {
 int SIGN;
 int ix, iy;
 RLType BUF;
 RLType rez=rezRL;
 if((GetRLSign(xRL)*GetRLSign(yRL))>0) SIGN=0;
 else SIGN=1;
 if ( (BUF=(RLType) malloc(sizeof(int)*GetRLQ(yRL))) == NULL) {
   printf("Not enough memory to allocate buffer RLMult\n");
   exit(1);  /* terminate program if out of memory */  }
 *(BUF)=0;
 *(BUF+1)=GetRLQ(yRL);
 rezRL=x10xRL(0); // начальное значение rezRL=0;
 for(ix=2; ix<GetRLQ(xRL); ix++) {
   for(iy=2; iy<GetRLQ(yRL); iy++) {
     *(BUF+iy)= *(xRL+ix) + *(yRL+iy);    }
   rezRL=Plus(rezRL, rezRL, BUF);  }
 FreeRL(BUF);
 if(rez==xRL)
   FreeRL(xRL);
 if(rez==yRL)
   FreeRL(yRL);
 *(rezRL)=SIGN;
return(rezRL);}
```

Розподілення кількості порівнянь та порівнянь зі зведенням подібних при виконанні операції множення представлене на рис. 2.

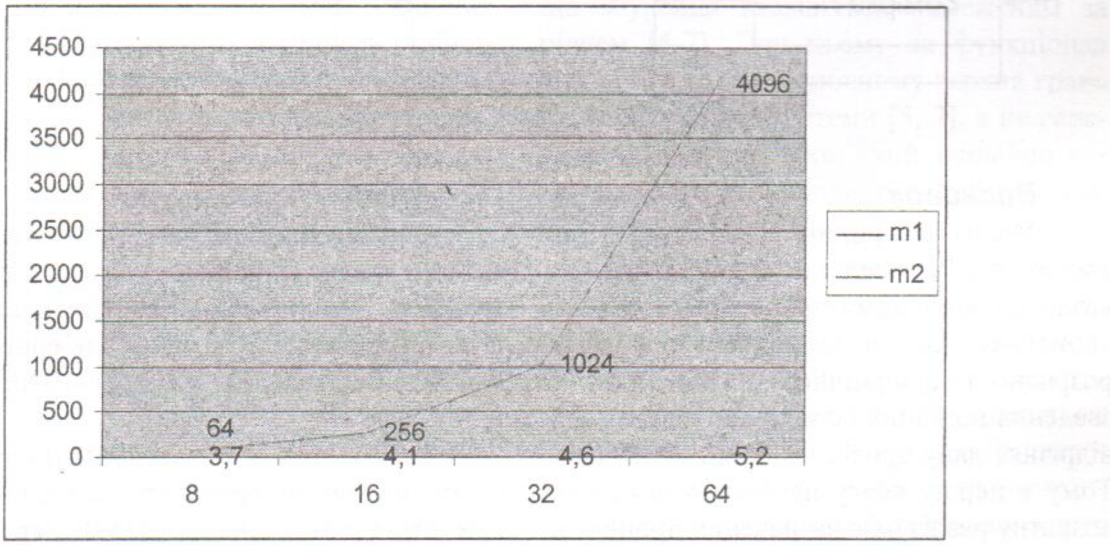

Рис. 2.

При структурній реалізації адитивного перетворення з використанням асоціативного способу обробки даних для виконання процедури сортування (порівняння розрядів) відбувається значне підвищення швидкодії – у кілька разів (у заданому діапазоні) порівняно з алгоритмом сортування, використаним у наведених вище програмах. Використання асоціативного способу обробки дозволяє подолати багато обмежень, що властиві адресному доступу до пам'яті, за рахунок завдання деякого критерію відбору й проведення необхідних перетворень, тільки над тими даними, які задовольняють цьому критерію. Критерієм відбору може бути збіг з будь-яким елементом даних, достатнім для виділення даних з усіх наявних. Пошук даних може відбуватися по фрагменту, що має більшу або меншу кореляцію із заданим елементом даних.

Асоціативні системи (рис. 3) відносяться до класу: один потік команд – багато потоків даних (*SIMD = Single Instruction Multiple Data*). Ці системи включають велику кількість операційних пристроїв, здатних одночасно за командами керуючого пристрою вести обробку декількох потоків даних. У асоціативних обчислювальних системах інформація на обробку надходить від асоціативних запам'ятовувальних пристроїв (АЗП), що характеризуються тим, що інформація в них обирається не за певною адресою, а за її змістом.

Рис. 3.

Висновок

Аналіз алгоритмів адитивного перетворення з РЛ представленням даних показав, що спільними процедурами, які використовуються в зазначених операціях розрядно-логарифмічної арифметики, є зведення подібних і сортування даних, що відрізняє дану арифметику від класичної. Тому в першу чергу необхідно виконати апаратну реалізацію зазначених процедур. Така архітектура може бути застосована для високоточних розрахунків складних виробів, моделювання на новому якісному рівні багатофакторних систем і розробки високопродуктивних паралельних комп'ютерних засобів.

Список літератури

1. *Гамаюн В. П.* Макрооператорные методы вычисления многоместных произведений // Микропроцессорные системы и их применение. – К.: Ин-т кибернетики им. В.М.Глушкова АН УССР , 1990. – С. 23-28.

2. *Гамаюн В. П.* Организация обработки в многооперандных вычислительных структурах. (Препр./ НАН Украины. Ин-т кибернетики; 96 - 3). – К., 1996. – 20 с.

3. *Гамаюн В. П.* Метод многооперандного умножения // УСиМ, 1994. – № 4-5. – С. 57-61.

4. *Карцев М. А., Брик В. А.* Вычислительные машины и синхронная арифметика. – М.: Радио и связь, 1981. – 360 с.