Aritmetik på stora heltal – algebra, algoritmer och assembly

Torbjörn Granlund

Avancerade algoritmer 2011

DEL 1: Optimeringsverktyg (för aritmetik på stora heltal)
DEL 2: Multiplikation i GMP

Verktyg 1: Algebra. Exempel: RSA-signering

Vi ska beräkna RSA-n (i tid $O(n^3)$)

$$s = m^d \bmod pq$$

där p och q är primtal, och $n = \log pq \approx \log m \approx \log d$.

Sätt $d_p = d \bmod (p-1)$ och $d_q = d \bmod (q-1)$.

Utför sedan de två exponentieringarna:

$$\begin{aligned} s_p &= (m \bmod p)^{d_p} \bmod p \\ s_q &= (m \bmod q)^{d_q} \bmod q \end{aligned}$$

Sedan ges s med CRT från s_p och s_q (i tid $O(n^2)$).

Verktyg 1: Algebra. Exempel: RSA-signering

Vi ska beräkna RSA-n (i tid $O(n^3)$)

$$s = m^d \bmod pq$$

där p och q är primtal, och $n = \log pq \approx \log m \approx \log d$.

Sätt $d_p = d \bmod (p-1)$ och $d_q = d \bmod (q-1)$.

Utför sedan de två exponentieringarna:

$$\begin{aligned} s_p &= (m \bmod p)^{d_p} \bmod p \\ s_q &= (m \bmod q)^{d_q} \bmod q \end{aligned}$$

Sedan ges s med CRT från s_p och s_q (i tid $O(n^2)$).

Verktyg 1: Algebra. Exempel: RSA-signering

Vi ska beräkna RSA-n (i tid $O(n^3)$)

$$s = m^d \bmod pq$$

där p och q är primtal, och $n = \log pq \approx \log m \approx \log d$.

Sätt $d_p = d \bmod (p-1)$ och $d_q = d \bmod (q-1)$.

Utför sedan de två exponentieringarna:

$$\begin{aligned} s_p &= (m \bmod p)^{d_p} \bmod p \\ s_q &= (m \bmod q)^{d_q} \bmod q \end{aligned}$$

Sedan ges s med CRT från s_p och s_q (i tid $O(n^2)$).

Verktyg 2: Effektiva algoritmer

Exempel:
Karatsubas dekompositionsalgoritm för multiplikation.

$$U = 2^{n/2} U_1 + U_0, \qquad V = 2^{n/2} V_1 + V_0$$

$$UV = (2^n + 2^{n/2}) U_1 V_1 - 2^{n/2} (U_1 - U_0)(V_1 - V_0) + (2^{n/2} + 1) U_0 V_0$$

Verktyg 3: Algoritmval efter operandstorlek (1)

Naiv Karatsuba-implementation:

```
mul (word *w, word *u, word *v, size_t n)
{
  if (n == 1)
    w[0] = u[0] * v[0];
  else    /* Karatsuba code */
    U1 = u + n/2; U0 = u; V1 = v + n/2; V0 = v;
    mul (P0, U1, V1, n/2);
    mul (P1, U0, V0, n/2);
    sub (Ud, U1, U0, n/2); sub (Vd, V1, V0, n/2);
    mul (Pd, Ud, Vd, n/2);
    copy (w, P0, n);       copy (w + n, P1, n);
    add (w + n/2, w + n/2, P0, n);
    add (w + n/2, w + n/2, P1, n);
    sub (w + n/2, w + n/2, Pd, n);
}
```

Verktyg 3: Algoritmval efter operandstorlek (2)

Listigare Karatsuba-implementation:

```
mul (word *w, word *u, word *v, size_t n)
{
  if (n < 17)
    mul_skolbok (w, u, v, n);
  else    /* Karatsuba code */
    U1 = u + n/2; U0 = u; V1 = v + n/2; V0 = v;
    mul (P0, U1, V1, n/2);
    mul (P1, U0, V0, n/2);
    sub (Ud, U1, U0, n/2); sub (Vd, V1, V0, n/2);
    mul (Pd, Ud, Vd, n/2);
    copy (w, P0, n);       copy (w + n, P1, n);
    add (w + n/2, w + n/2, P0, n);
    add (w + n/2, w + n/2, P1, n);
    sub (w + n/2, w + n/2, Pd, n);
}
```

Verktyg 3: Algoritmval efter operandstorlek (3)

Resultat:

Naiva Karatsuba-koden är enligt mina tester snabbare än skolboksmultiplikation från 8000 bitar (ca 2400 decimaler).

Listiga Karatsuba-koden är snabbare redan vid ca 830 bitar (250 decimaler).

(Tester utförda på Athlon.)

Slutsats 1:

En oavancerad implementation
av en avancerad algoritm kan vara till skada.

Verktyg 4: Minnes- och cache-lokalitet

- Temporal lokalitet
- Spatial lokalitet
- Data layout, padding

Algoritmegenskap: Dekompositionsalgoritmer har god lokalitet.

Verktyg 5: Utrullning av loopar

Istället för:

```
for (i = 0; i < n; i++)
  jobbenhet
```

Skriver vi:

```
for (i = 0; i < n mod 4; i++)
  jobbenhet
for (i = 0; i < n; i += 4)
  jobbenhet
  jobbenhet
  jobbenhet
  jobbenhet
```

Verktyg 6: Software pipelining (1)

Mål: Hantera fördröjningar (latency) för operationer.

Metod: Vi gör om loopen...

```
for (...)
  {
    a0 = *ap++;
    b0 = *bp++;
    r0 = a0 * b0;
    *rp++ = r0;
  }
```

Verktyg 6: Software pipelining (2)

... till:

```
for (...)
  {
    *rp++ = r0;
    r0 = a0 * b0;
    a0 = *ap++;
    b0 = *bp++;
  }
```

Verktyg 6: Software pipelining (3)

Med feed-in och wind-down:

```
a0 = *ap++;
b0 = *bp++;
r0 = a0 * b0;
a0 = *ap++;
b0 = *bp++;
for (...)
  {
    *rp++ = r0;
    r0 = a0 * b0;
    a0 = *ap++;
    b0 = *bp++;
  }
*rp++ = r0;
r0 = a0 * b0;
*rp++ = r0;
```

Verktyg 5 + 6: Kombinera utrullning och software pipelining

```
   feed-in       |      pipelined loop          |    wind-down
_________________|______________________________|____________________
a0 = *ap++;      |      for (...)               |
b0 = *bp++;      |        {                     |
a1 = *ap++;      |          *rp++ = r0;         |     *rp++ = r0;
b1 = *bp++;      |          r0 = a0 * b0;       |     r0 = a0 * b0;
                 |          a0 = *ap++;         |     *rp++ = r1;
r0 = a0 * b0;    |          b0 = *bp++;         |     r1 = a1 * b1;
a0 = *ap++;      |          *rp++ = r1;         |
b0 = *bp++;      |          r1 = a1 * b1;       |     *rp++ = r0;
r1 = a1 * b1;    |          a1 = *ap++;         |     *rp++ = r1;
a1 = *ap++;      |          b1 = *bp++;         |
b1 = *bp++;      |        }                     |
```

Verktyg 7: "Grundifiering" av rekurrenser (1)

Definition: Rekurrens = beroende mellan loopsteg.

I bignum-kod: Olika varianter av propagering av minnessiffra ("carry")

Påstående: Har vi k operationer för att generera utdata för en rekurrens från det att vi konsumerar indata för rekurrensen, kan ingen processor i världen utföra en iteration på $< k$ steg.

Verktyg 7: "Grundifiering" av rekurrenser (2)

Ganska djup rekurrens:

```
add (word *r, word *u, word *v, size_t n)
{
  cy = 0;
  for (i = 0; i < n; i++)
    {
      uword = u[i];
      vword = v[i];
      sum0 = uword + cy;
      cy0 = sum0 < uword;
      sum1 = sum0 + vword;
      cy1 = sum1 < sum0;
      cy = cy0 + cy1;
      r[i] = sum1;
    }
}
```

Verktyg 7: "Grundifiering" av rekurrenser (2b)

Ganska djup rekurrens:

```
add (word *r, word *u, word *v, size_t n)
{
  cy = 0;
  for (i = 0; i < n; i++)
    {
      uword = u[i];
      vword = v[i];
      sum0 = uword + cy;             0       4       8
      cy0 = sum0 < uword;            1       5      ...
      sum1 = sum0 + vword;           1       5
      cy1 = sum1 < sum0;             2       6
      cy = cy0 + cy1;                3       7
      r[i] = sum1;
    }
}
```

Verktyg 7: "Grundifiering" av rekurrenser (3)

Mindre djup rekurrens:

```
add (word *r, word *u, word *v, size_t n)
{
  cy = 0;
  for (i = 0; i < n; i++)
    {
      uword = u[i];
      vword = v[i];
      sum0 = uword + vword;
      cy0 = sum0 < uword;
      sum1 = sum0 + cy;
      cy1 = sum1 < sum0;
      cy = cy0 + cy1;
      r[i] = sum1;
    }
}
```

Verktyg 7: "Grundifiering" av rekurrenser (3b)

Mindre djup rekurrens:

```
add (word *r, word *u, word *v, size_t n)
{
  cy = 0;
  for (i = 0; i < n; i++)
    {
      uword = u[i];
      vword = v[i];
      sum0 = uword + vword;
      cy0 = sum0 < uword;
      sum1 = sum0 + cy;               0      3      6
      cy1 = sum1 < sum0;              1      4     ...
      cy = cy0 + cy1;                 2      5
      r[i] = sum1;
    }
}
```

Verktyg 7: "Grundifiering" av rekurrenser (4)

Riktigt grund rekurrens:

```
add (word *r, word *u, word *v, size_t n)
{
  cy = 0;
  for (i = 0; i < n; i++)
    {
      uword = u[i];
      vword = v[i];
      sum0 = uword + vword;
      cy0 = sum0 < uword;
      sum1 = sum0 + cy;
      cy1 = cy & (sum0 == ~0);
      cy = cy0 + cy1;
      r[i] = sum1;
    }
}
```

Verktyg 7: "Grundifiering" av rekurrenser (4b)

Riktigt grund rekurrens:

```
add (word *r, word *u, word *v, size_t n)
{
  cy = 0;
  for (i = 0; i < n; i++)
    {
      uword = u[i];
      vword = v[i];
      sum0 = uword + vword;
      cy0 = sum0 < uword;
      sum1 = sum0 + cy;                   0       2       4
      cy1 = cy & (sum0 == ~0);            0       2      ...
      cy = cy0 + cy1;                     1       3
      r[i] = sum1;
    }
}
```

Verktyg 8: Assembly

Implementera i assembly!

- Leta användbara instruktioner
- Designa mikro-algoritmer efter tillgängliga instruktioner
- Ta hänsyn till fördröjning för valda instruktioner
- Vilka instruktioner kan utföras parallellt?
- Alignment
- Trial-and-measure
- Trial-and-measure
- ...

Verktyg 9: Av-hoppning

Villkorliga hopp av två sorter:

1. Prediktabla
2. Slumpmässiga (eller av andra skäl icke prediktabla)

Ett icke prediktabelt hopp kostar som kanske 30 andra instruktioner.

Optimeringsverktyg för aritmetik på stora heltal

1. Algebra
2. Effektiva algoritmer
3. Algoritmval efter operandstorlek
4. Minnes- och cache-lokalitet
5. Utrullning av loopar
6. Software pipelining
7. "Grundifiering" av rekurrenser
8. Assembly
9. Hopp-prediktabilitet

DEL 2: Stora heltal i GMP

Heltalsmultiplikation

Problem: Beräkna $W \leftarrow U \times V, \quad U, V \in \mathbb{Z}$

I GMP $\log_2(U), \log_2(V) < 2^{50}$

Mål: Maximal praktiskt prestanda $+$ lägsta komplexitet

Algoritm-1: "Skolboks"-multiplikation (1)

$$W = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \beta^{i+j} u_i v_j = \sum_{i=0}^{n-1} \left(\beta^i u_i \sum_{j=0}^{n-1} \beta^j v_j \right)$$

Tidskomplexitet: $O(n^2)$

Vår mul_skolbok kan bli väldigt enkel:

```
mul_skolbok (word *w, word *u, size_t un, word *v, size_t vn)
{
  zero (w, un + vn);
  for (i = 0; i < vn; i++)
    w[un + i] = mul1add (w + i, u, un, v[i]);
}
```

Hur ser `mul1add` ut? Kanske så här:

```
mul1add (word *w, word *u, size_t un, word vword)
{
  cy_word = 0;
  for (j = 0; j < un; j++)
    {
      uword = u[i];
      lo = LO (uword * vword);
      hi = HI (uword * vword);
      wword = w[i];
      w[i] = LO (wword + lo + cy_word);
      cy_word = hi + HI (wword + lo + cy_word);
    }
  return cy_word;
}
```

Eller så här (PowerPC64):

```
mul1add:
    mtctr   r5
    li      r9, 0        # cy_word = 0
    addic   r0, r0, 0    # hw cy flag = 0
    addi     r3, r3, -8
    addi     r4, r4, -8
    nop                  # alignment
    nop                  # alignment
    nop                  # alignment

L1: ldu     r0, 8(r4)    # r0 = (*++u)
    ld      r10, 8(r3)   # r10 = (*w)
    mulld   r7, r0, r6   # low 64 product bits
    mulhdu  r8, r0, r6   # high 64 product bits
    adde    r7, r7, r9   # add old cy_word [in]
    addze   r9, r8       # new cy_word     [out]
    addc    r7, r7, r10  # add loaded (*w)
    stdu    r7, 8(r3)    # *++w = result
    bdnz    L1

    addze   r3, r9
    blr
```

Eller lite mer komplicerat på Alpha. . .

```
        ldq     r3,     0(r17)
        and     r18,    7,      r20
        lda     r18,    -9(r18)
        cmpeq   r20,    1,      r21
        beq     r21,    $L1
$1mod8: ldq     r5,     0(r16)
        mulq    r19,    r3,     r7
        umulh   r19,    r3,     r8
        addq    r5,     r7,     r23
        cmpult  r23,    r7,     r20
        addq    r8,     r20,    r0
        stq     r23,    0(r16)
        bge     r18,    $ent1
        ret     r31,    (r26),  1
$L1:    lda     r8,     0(r31)
        lda     r24,    0(r31)
        cmpeq   r20,    2,      r21
        bne     r21,    $2mod8
        cmpeq   r20,    3,      r21
        bne     r21,    $3mod8
        cmpeq   r20,    4,      r21
        bne     r21,    $4mod8
        cmpeq   r20,    5,      r21
        bne     r21,    $5mod8
        cmpeq   r20,    6,      r21
        bne     r21,    $6mod8
        cmpeq   r20,    7,      r21
        beq     r21,    $0mod8
$7mod8: ldq     r5,     0(r16)
        lda     r17,    8(r17)
        mulq    r19,    r3,     r7
        umulh   r19,    r3,     r24
        addq    r5,     r7,     r23
        cmpult  r23,    r7,     r20
        addq    r24,    r20,    r24
        stq     r23,    0(r16)
        lda     r16,    8(r16)
        ldq     r3,     0(r17)
$6mod8: ldq     r1,     8(r17)
        mulq    r19,    r3,     r25
        umulh   r19,    r3,     r3
        mulq    r19,    r1,     r28
        ldq     r0,     16(r17)
        ldq     r4,     0(r16)
        umulh   r19,    r1,     r8
        ldq     r1,     24(r17)
        lda     r17,    48(r17)
        mulq    r19,    r0,     r2
        ldq     r5,     8(r16)
        lda     r16,    -32(r16)
        umulh   r19,    r0,     r6
        addq    r4,     r25,    r4
        mulq    r19,    r1,     r7
        br      r31,    $ent6
$ent1:  lda     r17,    8(r17)
        lda     r16,    8(r16)
        lda     r8,     0(r0)
        ldq     r3,     0(r17)
$0mod8: ldq     r1,     8(r17)
        mulq    r19,    r3,     r2
        umulh   r19,    r3,     r6
        mulq    r19,    r1,     r7
        ldq     r0,     16(r17)
        ldq     r4,     0(r16)
        umulh   r19,    r1,     r24
        ldq     r1,     24(r17)
        mulq    r19,    r0,     r25
        ldq     r5,     8(r16)
        umulh   r19,    r0,     r3
        addq    r4,     r2,     r4
        mulq    r19,    r1,     r28
        lda     r16,    -16(r16)
        br      r31,    $ent0
$3mod8: ldq     r5,     0(r16)
        lda     r17,    8(r17)
        mulq    r19,    r3,     r7
        umulh   r19,    r3,     r8
        addq    r5,     r7,     r23
        cmpult  r23,    r7,     r20
        addq    r8,     r20,    r24
        stq     r23,    0(r16)
        lda     r16,    8(r16)
        ldq     r3,     0(r17)
$2mod8: ldq     r1,     8(r17)
        mulq    r19,    r3,     r25
        umulh   r19,    r3,     r3
        mulq    r19,    r1,     r28
        ble     r18,    $n23
        ldq     r0,     16(r17)
        ldq     r4,     0(r16)
        umulh   r19,    r1,     r8
        ldq     r1,     24(r17)
        lda     r17,    16(r17)
        mulq    r19,    r0,     r2
        ldq     r5,     8(r16)
        lda     r16,    0(r16)
        umulh   r19,    r0,     r6
        addq    r4,     r25,    r4
        mulq    r19,    r1,     r7
        br      r31,    $ent2
$5mod8: ldq     r5,     0(r16)
        lda     r17,    8(r17)
        mulq    r19,    r3,     r7
        umulh   r19,    r3,     r24
        addq    r5,     r7,     r23
        cmpult  r23,    r7,     r20
        addq    r24,    r20,    r8
        stq     r23,    0(r16)
        lda     r16,    8(r16)
        ldq     r3,     0(r17)
$4mod8: ldq     r1,     8(r17)
        mulq    r19,    r3,     r2
        umulh   r19,    r3,     r6
        mulq    r19,    r1,     r7
        ldq     r0,     16(r17)
        ldq     r4,     0(r16)
        umulh   r19,    r1,     r24
        ldq     r1,     24(r17)
        lda     r17,    32(r17)
        mulq    r19,    r0,     r25
        ldq     r5,     8(r16)
        lda     r16,    16(r16)
        umulh   r19,    r0,     r3
        addq    r4,     r2,     r4
        mulq    r19,    r1,     r28
        cmpult  r4,     r2,     r20
        addq    r4,     r8,     r22
        ble     r18,    $Lend
ALIGN(16)
$Loop:
        bis     r31,    r31,    r31
        cmpult  r22,    r8,     r21
        addq    r6,     r20,    r6
        ldq     r0,     0(r17)
        bis     r31,    r31,    r31
        addq    r5,     r7,     r23
        addq    r6,     r21,    r6
        ldq     r4,     0(r16)
        umulh   r19,    r1,     r8
        cmpult  r23,    r7,     r20
        addq    r23,    r6,     r23
        ldq     r1,     8(r17)
        mulq    r19,    r0,     r2
        cmpult  r23,    r6,     r21
        addq    r24,    r20,    r24
        ldq     r5,     8(r16)
        umulh   r19,    r0,     r6
        addq    r4,     r25,    r4
        stq     r22,    -16(r16)
        stq     r23,    -8(r16)
        bis     r31,    r31,    r31
        mulq    r19,    r1,     r7
        bis     r31,    r31,    r31
        addq    r24,    r21,    r24
$ent2:
        cmpult  r4,     r25,    r20
        bis     r31,    r31,    r31
        lda     r18,    -8(r18)
        addq    r4,     r24,    r22
        bis     r31,    r31,    r31
        cmpult  r22,    r24,    r21
        addq    r3,     r20,    r3
        ldq     r0,     16(r17)
        bis     r31,    r31,    r31
        addq    r5,     r28,    r23
        addq    r3,     r21,    r3
        ldq     r4,     16(r16)
        umulh   r19,    r1,     r24
        cmpult  r23,    r28,    r20
        addq    r23,    r3,     r23
        ldq     r1,     24(r17)
        mulq    r19,    r0,     r25
        cmpult  r23,    r3,     r21
        addq    r8,     r20,    r8
        ldq     r5,     24(r16)
        umulh   r19,    r0,     r3
        addq    r4,     r2,     r4
        stq     r22,    0(r16)
        stq     r23,    8(r16)
        bis     r31,    r31,    r31
        mulq    r19,    r1,     r28
        bis     r31,    r31,    r31
        addq    r8,     r21,    r8
$ent0:
        cmpult  r4,     r2,     r20
        bis     r31,    r31,    r31
        lda     r17,    64(r17)
        addq    r4,     r8,     r22
        bis     r31,    r31,    r31
        cmpult  r22,    r8,     r21
        addq    r6,     r20,    r6
        ldq     r0,     -32(r17)
        bis     r31,    r31,    r31
        addq    r5,     r7,     r23
        addq    r6,     r21,    r6
        ldq     r4,     32(r16)
        umulh   r19,    r1,     r8
        cmpult  r23,    r7,     r20
        addq    r23,    r6,     r23
        ldq     r1,     -24(r17)
        mulq    r19,    r0,     r2
        cmpult  r23,    r6,     r21
        addq    r24,    r20,    r24
        ldq     r5,     40(r16)
        umulh   r19,    r0,     r6
        addq    r4,     r25,    r4
        stq     r22,    16(r16)
        stq     r23,    24(r16)
        bis     r31,    r31,    r31
        mulq    r19,    r1,     r7
        bis     r31,    r31,    r31
        addq    r24,    r21,    r24
$ent6:
        cmpult  r4,     r25,    r20
        bis     r31,    r31,    r31
        lda     r16,    64(r16)
        addq    r4,     r24,    r22
        bis     r31,    r31,    r31
        cmpult  r22,    r24,    r21
        addq    r3,     r20,    r3
        ldq     r0,     -16(r17)
        bis     r31,    r31,    r31
        addq    r5,     r28,    r23
        addq    r3,     r21,    r3
        ldq     r4,     -16(r16)
        umulh   r19,    r1,     r24
        cmpult  r23,    r28,    r20
        addq    r23,    r3,     r23
        ldq     r1,     -8(r17)
        mulq    r19,    r0,     r25
        cmpult  r23,    r3,     r21
        addq    r8,     r20,    r8
        ldq     r5,     -8(r16)
        umulh   r19,    r0,     r3
        addq    r4,     r2,     r4
        stq     r22,    -32(r16)
        stq     r23,    -24(r16)
        bis     r31,    r31,    r31
        mulq    r19,    r1,     r28
        bis     r31,    r31,    r31
        addq    r8,     r21,    r8
        cmpult  r4,     r2,     r20
        addq    r4,     r8,     r22
        ldl     r31,    256(r17)
        bgt     r18,    $Loop
$Lend:  cmpult  r22,    r8,     r21
        addq    r6,     r20,    r6
        addq    r5,     r7,     r23
        addq    r6,     r21,    r6
        ldq     r4,     0(r16)
        umulh   r19,    r1,     r8
        cmpult  r23,    r7,     r20
        addq    r23,    r6,     r23
        cmpult  r23,    r6,     r21
        addq    r24,    r20,    r24
        ldq     r5,     8(r16)
        addq    r4,     r25,    r4
        stq     r22,    -16(r16)
        stq     r23,    -8(r16)
        addq    r24,    r21,    r24
        cmpult  r4,     r25,    r20
        addq    r4,     r24,    r22
        cmpult  r22,    r24,    r21
        addq    r3,     r20,    r3
        addq    r5,     r28,    r23
        addq    r3,     r21,    r3
        cmpult  r23,    r28,    r20
        addq    r23,    r3,     r23
        cmpult  r23,    r3,     r21
        addq    r8,     r20,    r8
        stq     r22,    0(r16)
        stq     r23,    8(r16)
        addq    r8,     r21,    r0
        ret     r31,    (r26),  1
$n23:   ldq     r4,     0(r16)
        ldq     r5,     8(r16)
        umulh   r19,    r1,     r8
        addq    r4,     r25,    r4
        cmpult  r4,     r25,    r20
        addq    r4,     r24,    r22
        cmpult  r22,    r24,    r21
        addq    r3,     r20,    r3
        addq    r5,     r28,    r23
        addq    r3,     r21,    r3
        cmpult  r23,    r28,    r20
        addq    r23,    r3,     r23
        cmpult  r23,    r3,     r21
        addq    r8,     r20,    r8
        stq     r22,    0(r16)
        stq     r23,    8(r16)
        addq    r8,     r21,    r0
        ret     r31,    (r26),  1
```

Prestanda nu

n	**Skolb**
10^0	8 ns
10^1	123 ns
10^2	10 μs
10^3	1 ms
10^4	165 ms
10^5	32 s
10^6	$>$ 1 h
10^7	$>$ 4 d
10^8	$>$ 1 y
10^9	$>$ 100 y

(Tider på en 2.5 GHz Opteron PC, GMP 4.3.)

Bas-β heltal vs polynom (1)

Integer:

$$U = \sum_{i=0}^{n-1} u_i \beta^i, \quad u_i < \beta$$

Motsvarande polynom:

$$u(x) = \sum_{i=0}^{n-1} u_i x^i$$

Bas-β heltal vs polynom (1)

Integer:

$$U = \sum_{i=0}^{n-1} u_i \beta^i, \quad u_i < \beta$$

Motsvarande polynom:

$$u(x) = \sum_{i=0}^{n-1} u_i x^i$$

Bas-β heltal vs polynom (2)

Om vi har ett tal i bas-10

$$18\ 5054\ 0856\ 8445\ 1320\ 8201$$

ock låter $\beta = 10^4$, så motsvarar detta polynomet

$$18x^5 + 5054x^4 + 0856x^3 + 8445x^2 + 1320x + 8201$$

Algoritm-2: Karatsubas “magiska formel” (1)

Givet heltal $U, V < 2^{2n}$. Låt $\beta = 2^n$.

$$U = \beta U_1 + U_0, \qquad V = \beta V_1 + V_0$$

$$\begin{aligned} UV = {} & (\beta^2 + \beta) U_1 \times V_1 + \\ & - \beta (U_1 - U_0) \times (V_1 - V_0) + \\ & + (\beta + 1) U_0 \times V_0 \end{aligned}$$

Tidskomplexitet:

$$T(n) = 3T(n/2) + O(n)$$

$$T(n) = O(n^{1.59})$$

Prestanda nu

n	Skolb	Kara
10^0	8 ns	n/a
10^1	120 ns	180 ns (bad!)
10^2	10 μs	7.0 μs
10^3	1 ms	280 μs
10^4	170 ms	14 ms
10^5	32 s	479 ms
10^6	$>$ 1 h	17 s
10^7	$>$ 4 d	680 s
10^8	$>$ 1 y	7 h
10^9	$>$ 100 y	11 d

Algoritm-3: Tooms generalisering av Karatsuba (1)

Låt heltalet U representeras av polynomet $u(x)$, dvs $u(\beta) = U$ för ett β^n vi väljer. Analogt för V, $v(x)$.

Tooms observation: Vi kan evaluera $u(x)$ and $v(x)$ i några punkter $x_0, x_1 \ldots x_k$, sedan multiplicera $u(x_0)$ med $v(x_0)$, $u(x_1)$ med $v(x_1)$ etc. Produkten $w(x)$ får man fram med interpolation.

Om $u(x)$ och $v(x)$ har grad k, $w(x)$ kommer ha graden $2k$, och vi behöver $2k + 1$ punkter för att bestämma koefficienterna hos $w(x)$.

I Toom-språk, så har Karatsubas algoritm $k = 1$ och evaluerar i punkterna $0, -1$ och ∞.

Algoritm-3: Tooms generalisering av Karatsuba (1)

Låt heltalet U representeras av polynomet $u(x)$, dvs $u(\beta) = U$ för ett β^n vi väljer. Analogt för V, $v(x)$.

Tooms observation: Vi kan evaluera $u(x)$ and $v(x)$ i några punkter $x_0, x_1 \ldots x_k$, sedan multiplicera $u(x_0)$ med $v(x_0)$, $u(x_1)$ med $v(x_1)$ etc. Produkten $w(x)$ får man fram med interpolation.

Om $u(x)$ och $v(x)$ har grad k, $w(x)$ kommer ha graden $2k$, och vi behöver $2k + 1$ punkter för att bestämma koefficienterna hos $w(x)$.

I Toom-språk, så har Karatsubas algoritm $k = 1$ och evaluerar i punkterna $0, -1$ och ∞.

Algoritm-3: Tooms generalisering av Karatsuba (2)

Exempel:

Kapa operanderna U and V in 3 delar vardera för att forma två grad-2 polynom $u(x)$ and $v(x)$. Vi behöver evaluera i $k+1=5$ punkter, t ex $-1, 0, +1, +2, \infty$.

Tidskomplexitet:

$$T(n) = 5T(n/3) + O(n)$$

$$T(n) = O(n^{1.46})$$

Kapa operanderna i 4 delar och evaluera i $k+1=7$ punkter, t ex $-1, -1/2, 0, +1/2, +1, +2, \infty$.

Tidskomplexitet:

$$T(n) = 7T(n/4) + O(n)$$

$$T(n) = O(n^{1.40})$$

Prestanda nu

n	Skolb	Kara	Toom 3,4
10^0	8 ns	n/a	n/a
10^1	120 ns	180 ns	n/a
10^2	10 μs	7.0 μs	6.9 μs
10^3	1 ms	280 μs	220 μs
10^4	170 ms	14 ms	6.3 ms
10^5	32 s	470 ms	180 ms
10^6	$>$ 1 h	17 s	5.0 s
10^7	$>$ 4 d	680 s	130 s
10^8	$>$ 1 y	7 h	1 h
10^9	$>$ 100 y	11 d	0.9 d

Algoritm-4: FFT-familjen

FFT är en algoritm som beräknar vissa DFTs effektivt. Indata är ett grad-2^k-polynom, utdata är ett annat grad-2^k-polynom.

FFT behöver koefficienter i en ring R *principalenhetsrötter* av ordning 2^k
En FFT-baserad heltalsmultiplikation ser ut så här:

1. $u(x) \leftarrow \textsc{split}(U)$, $v(x) \leftarrow \textsc{split}(V)$,
2. $u'(x) \leftarrow \textsc{fft}(u(x))$, $v'(x) \leftarrow \textsc{fft}(v(x))$
3. $p'(x) \leftarrow u'(x) \cdot v'(x)$ punktvis multiplikation
4. $p(x) \leftarrow \textsc{fft}^{-1}(p'(x))$
5. $P = p(\beta)$

Prestanda nu

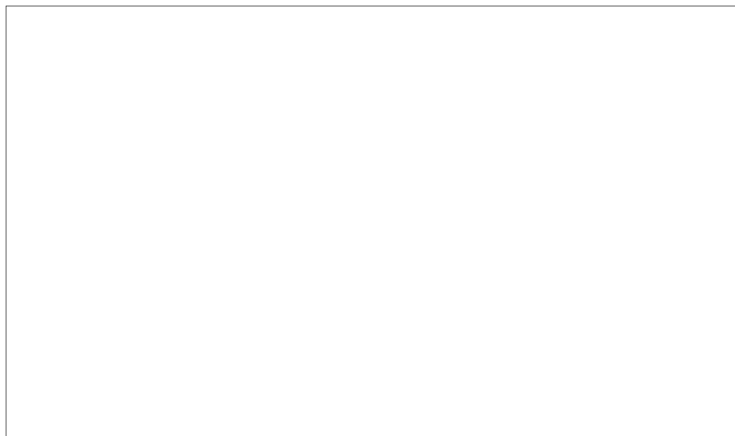
n	**Skolb**	**Kara**	**Toom 3,4**	**SS FFT**
10^0	8 ns	n/a	n/a	n/a
10^1	120 ns	180 ns (bad!)	n/a	n/a
10^2	10 μs	7.0 μs	6.9 μs	40 μs
10^3	1 ms	280 μs	220 μs	416 μs
10^4	170 ms	14 ms	6.3 ms	5.4 ms
10^5	32 s	470 ms	180 ms	83 ms
10^6	$>$ 1 h	17 s	5.0 s	1.3 s
10^7	$>$ 4 d	680 s	130 s	15 s
10^8	$>$ 1 y	7 h	1 h	210 s
10^9	$>$ 100 y	11 d	0.9 d	$\approx$ 1 h

Likstora vs olikstora operands (1)

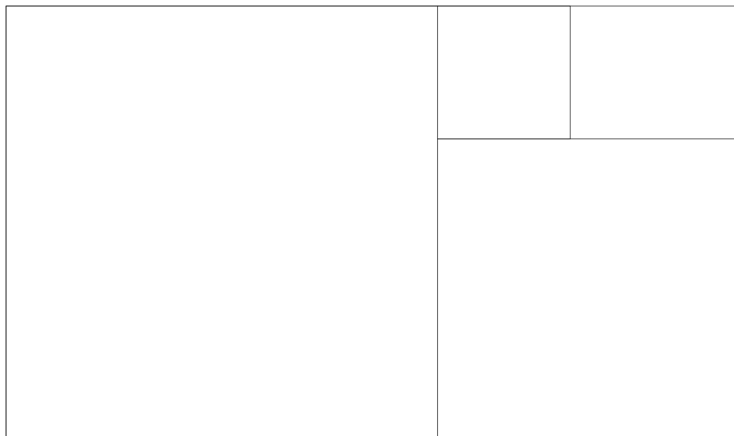
Vi har hittills bara berört likstora operander, och mappat dessa på två polynom av samma gradtal.

Hur kan vi hantera olikstora operander?

Likstora vs olikstora operander (2)



Likstora vs olikstora operander (3)



Anpassning av algoritmer till olikstora operander

- Skolboksalgoritmen fungerar
- Karatsuba-Toom inte så uppenbar
- Karatsuba-Toom-Bodrato-Zanoni hjälper
- FFT är ”enkel” (men blir söligare)

Bodrato-Zanonis generalisering av Toom

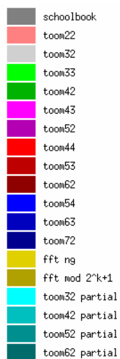
År 2006 generaliserade M.Bodrato and A.Zanoni Tooms algoritm att använda polynom $u(x)$, $v(x)$ av **olika** gradtal.

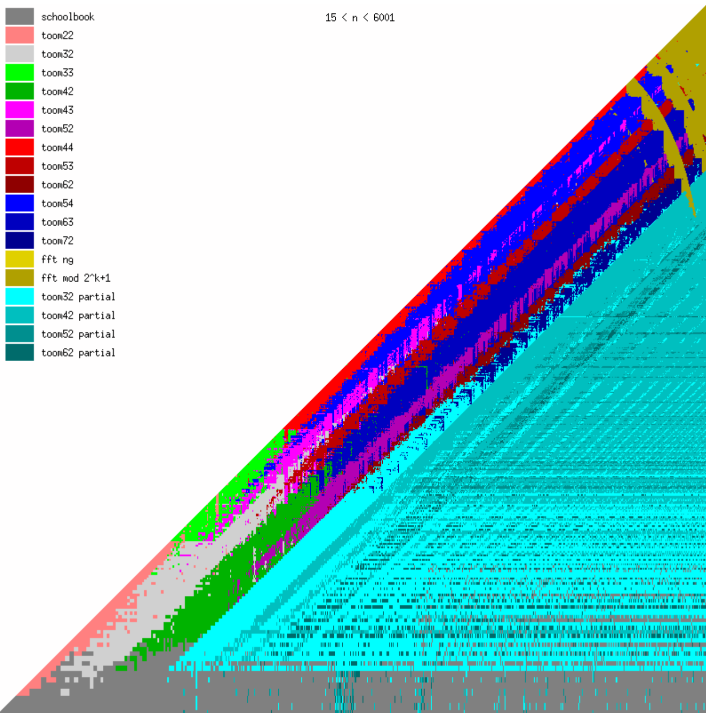
Detta är användbart för multiplikation av *olikstora* heltalsoperander.

Exempel: Om storleken av U och V är i relationen 3:2, kan vi mappa U på ett grad-2-polynom $u(x)$ och V på ett grad-1-polynom $v(x)$. Vi behöver evaluera i 4 punkter.

Toom-Bodrato-Zanoni primitiv i GMP

deg(u)	**deg**(v)	**k**	**punkter**	**namn**
1	1	3	$-1, 0, \infty$	toom22_mul
2	1	4	$-1, 0, +1, \infty$	toom32_mul
2	2	5	$-1, 0, +1, +2, \infty$	toom33_mul
3	1	5	$-1, 0, +1, +2, \infty$	toom42_mul
3	2	6	$-2, -1, 0, +1, +2, \infty$	toom43_mul
4	1	6	$-2, -1, 0, +1, +2, \infty$	toom52_mul
3	3	7	$-2, -1, 0, +1/2, +1, +2, \infty$	toom44_mul
4	2	7	$-2, -1, 0, +1/2, +1, +2, \infty$	toom53_mul
5	1	7	$-2, -1, 0, +1/2, +1, +2, \infty$	toom62_mul


schoolbook
toom22
toom32
toom33
toom42
toom43
toom52
toom44
toom53
toom62
toom54
toom63
toom72
fft ng
fft mod 2^k+1
toom32 partial
toom42 partial
toom52 partial
toom62 partial
0 < n < 6001



schoolbook
toom22
toom32
toom33
toom42
toom43
toom52
toom44
toom53
toom62
toom54
toom63
toom72
fft ng
fft mod 2^k+1
toom32 partial
toom42 partial
toom52 partial
toom62 partial
15 < n < 6001