Applied Symbolic Computation

(CS 567)

Assignment 3

Zexi Liu

12/08/2010

1

Timing GMP's Integer Multiplication Algorithm

- NxN limb[1] multiplications and squares are done using one of seven algorithms, as the size N increases

Algorithm	**Threshold**
Basecase	(none)
Karatsuba	MUL_TOOM22_THRESHOLD
Toom-3	MUL_TOOM33_THRESHOLD
Toom-4	MUL_TOOM44_THRESHOLD
Toom-6.5	MUL_TOOM6H_THRESHOLD
Toom-8.5	MUL_TOOM8H_THRESHOLD
FFT	MUL_FFT_THRESHOLD

1. A limb means the part of a multi-precision number that fits in a single machine word.

2

Timing GMP's Integer Multiplication Algorithm

- Environment Information
  - Ubuntu 10.10, Linux 2.6.35-23-generic, 64-bit
  - gcc version 4.4.5
  - CPU: Intel core i-7 2667 MHz (idle 1600 MHz)
  - 6 GB memory
- Performance Optimization
  - In GMP package, build and run the ***tuneup*** program in the ***tune*** subdirectory

```
cd tune
make tuneup
./tuneup
```

3

Timing GMP's Integer Multiplication Algorithm

- Results from running ***tuneup***

```
#define MUL_TOOM22_THRESHOLD    18
#define MUL_TOOM33_THRESHOLD    65
#define MUL_TOOM44_THRESHOLD    166
#define MUL_TOOM6H_THRESHOLD    226
#define MUL_TOOM8H_THRESHOLD    336
#define MUL_FFT_THRESHOLD       4100
```

- Occasional error when running ***tuneup***

  speed_measure() could not get 4 results within 0.5%

  tuneup will be aborted with this error.

  Have to re-run it several times to get the results.

1. A limb means the part of a multi-precision number that fits in a single machine word.

4

Timing GMP's Integer Multiplication Algorithm

- Verify the calculated thresholds

#define MUL_TOOM22_THRESHOLD	18
#define MUL_TOOM33_THRESHOLD	65
#define MUL_TOOM44_THRESHOLD	166
#define MUL_TOOM6H_THRESHOLD	226
#define MUL_TOOM8H_THRESHOLD	336
#define MUL_FFT_THRESHOLD	4100

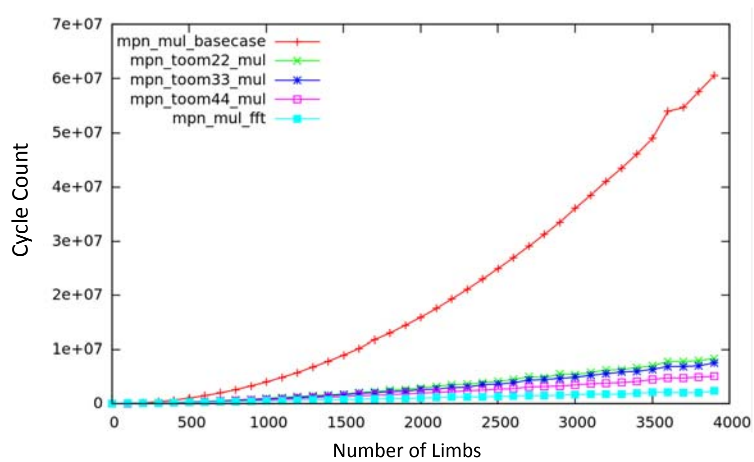
- In GMP package, build and run the ***speed*** program in the ***tune*** subdirectory

./speed -s 1-4000 -t 100 -c -P P1 mpn_basecase

-s 1-4000: multiply from 1x1 to 4000x4000 (limbs)
-t 100: step size 100, i.e., 1x1, 100x100, 200x200, …
-c: times in cpu cycles
-P: plot using ***gnuplot***

Timing GMP's Integer Multiplication Algorithm

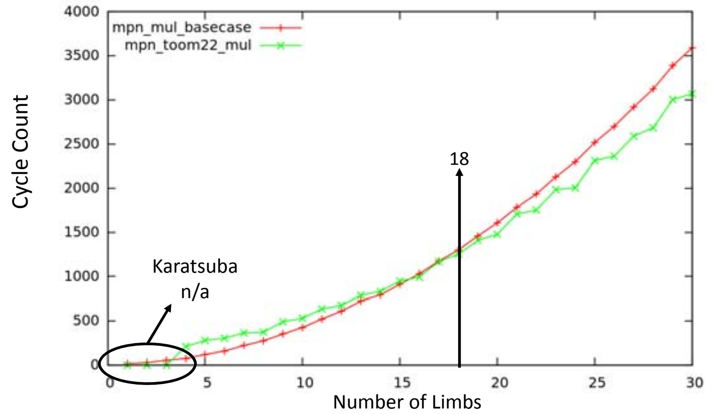
- An overview



Timing GMP's Integer Multiplication Algorithm

- MUL_TOOM22_THRESHOLD - 18

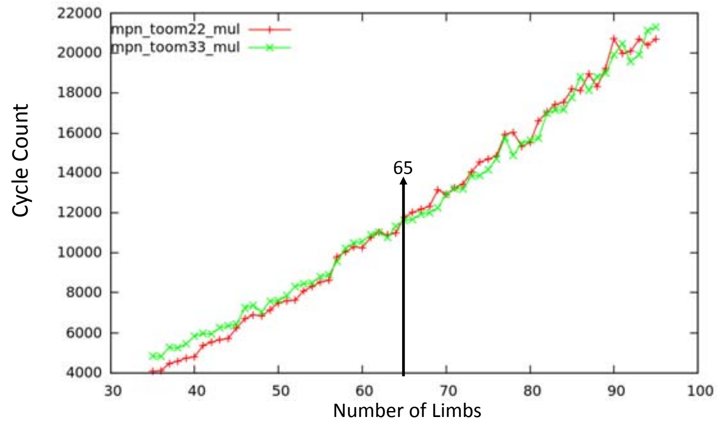
Threshold between basecase and karatsuba



Timing GMP's Integer Multiplication Algorithm

- MUL_TOOM33_THRESHOLD - 65

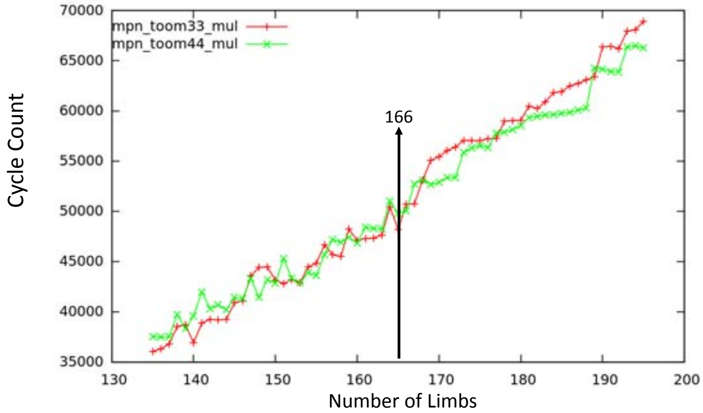
Threshold between karatsuba and Toom-3



Timing GMP's Integer Multiplication Algorithm

- MUL_TOOM33_THRESHOLD - 166

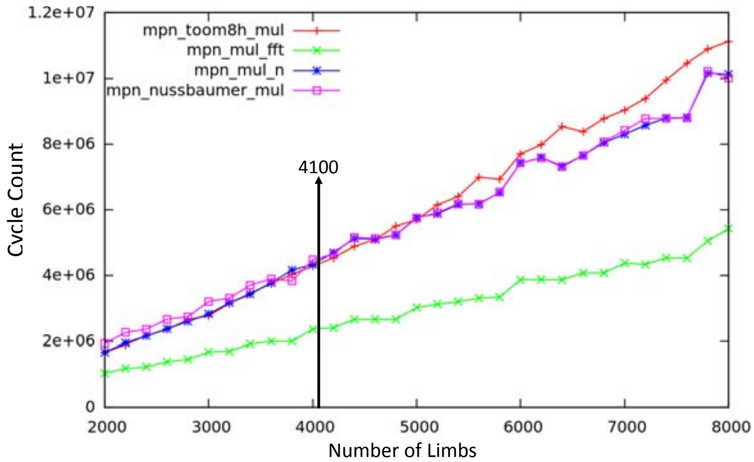
Threshold between Toom-3 and Toom-4



9

Timing GMP's Integer Multiplication Algorithm

- MUL_FFT_THRESHOLD - 4100



10

Timing GMP's Integer Multiplication Algorithm

- MUL_FFT_THRESHOLD – 4100

mpn_mul_fft – Schoenhage-Strassen's Algorithm

Schoenhage-Strassen's fast multiplication modulo 2^N+1

mpn_nussbaumer_mul – Nussbaumer Negacyclic Convolution

Multiply {ap, an} and {bp, bn} using Nussbaumer Negacyclic Convolution. It calls mpn_mul_fft.

11

Timing GMP's Integer Multiplication Algorithm

```
void mpn_mul_n (mp_ptr p, mp_srcptr a, mp_srcptr b, mp_size_t n)  {
    if (BELOW_THRESHOLD (n, MUL_TOOM22_THRESHOLD))  {
        mpn_mul_basecase (p, a, n, b, n);
    }
    else if (BELOW_THRESHOLD (n, MUL_TOOM33_THRESHOLD))   {
        mpn_toom22_mul (p, a, n, b, n, ws);
    }
    else if (BELOW_THRESHOLD (n, MUL_TOOM44_THRESHOLD))   {
        mpn_toom33_mul (p, a, n, b, n, ws);
    }
    else if (BELOW_THRESHOLD (n, MUL_TOOM6H_THRESHOLD))   {
        mpn_toom44_mul (p, a, n, b, n, ws);
    }
    else if (BELOW_THRESHOLD (n, MUL_TOOM8H_THRESHOLD))   {
        mpn_toom6h_mul (p, a, n, b, n, ws);
    }
    else if (BELOW_THRESHOLD (n, MUL_FFT_THRESHOLD))   {
        mpn_toom8h_mul (p, a, n, b, n, ws);
    }
    else   {
        mpn_fft_mul (p, a, n, b, n);            // mpn_nussbaumer_mul
    }
}
```

12

Implementation of SSA in GMP

- Improvement 1 – Arithmetic Modulo 2^n+1
  - The addition of two semi-normalized representations (residue mod 2^n+1)

```
/* r <- a+b mod 2^(n*GMP_NUMB_BITS)+1. Assumes a and b are semi-normalized. */
static inline void mpn_fft_add_modF (mp_ptr r, mp_srcptr a, mp_srcptr b, int n)
{
        mp_limb_t c, x;
        c = a[n] + b[n] + mpn_add_n (r, a, b, n);
        /* 0 <= c <= 3 */
        x = (c - 1) & -(c != 0);
        r[n] = c - x;
        MPN_DECR_U (r, n + 1, x);
}
```

13

Implementation of SSA in GMP

- Improvement 1 – Arithmetic Modulo 2^n+1
  - The multiplication by 2^e

```
/* r <- a*2^d mod 2^(n*GMP_NUMB_BITS)+1 with a = {a, n+1} Assumes a is semi-normalized, i.e.
a[n] <= 1. r and a must have n+1 limbs, and not overlap. */
static void mpn_fft_mul_2exp_modF (mp_ptr r, mp_srcptr a, unsigned int d, mp_size_t n)
{
        ..........................
        sh = d % GMP_NUMB_BITS;
        d /= GMP_NUMB_BITS;
        if (d >= n)                          /* negate */
        {
        /* r[0..d-1]  <-- lshift(a[n-d]..a[n-1], sh) r[d..n-1]  <-- -lshift(a[0]..a[n-d-1],  sh) */
                d -= n;
                if (sh != 0)
                {
                        /* no out shift below since a[n] <= 1 */
                        mpn_lshift (r, a + n - d, d + 1, sh);
                        rd = r[d];
                        cc = mpn_lshiftc (r + d, a, n - d, sh);
                }
        ..........................
```

14

Implementation of SSA in GMP

- Improvement 2

  Cache Locality During the Transforms

  – The Belgian Transform

    One solution is to perform the trees of butterflies following the BitReverse order.

  – Higher Radix Transforms

    The principle of higher radix transforms is to use an atomic operation which groups several butterflies.

  – Bailey's 4-step Algorithm

    A threshold is setup for activating Bailey's algorithm only for large sizes.

  – Mixing Serveral Phases

    Another way to improve locality is to mix different phases of the algorithm in order to do as much work as possible on the data while they are in the cache.

15

Implementation of SSA in GMP

- Improvement 3

  Fermat and Mersenne Transforms

  Power-of-two roots of unity are needed only at the "lower level", i.e., in Rn+. Therefore one can replace RN by RN- --- i.e., the ring of integers modulo 2^N - 1 --- in the original algorithm, and replace the weighted transform by a classical cyclic convolution, to compute a product mod $2^N - 1$.

- Improvement 4

  The $\sqrt{2}$ Trick

  We can use $\sqrt{2} = 2^{\frac{3n}{4}} - 2^{\frac{n}{4}}$ as a root of unity of order 2^{k+2} in the transform to double the possible transform length for a given n.

16

Implementation of SSA in GMP

- Improvement 5

  Harley's and Granlund's Tricks

  Assume 2M + k is just above an integer multiple of K, say λK. Then we have to use n = (λ + 1)K, which gives an efficiency of only about λ /(λ +1). Harley's idea is to use n = λ K instead, and recover λ+1 the missing information from a CRT-reconstruction with an additional computation modulo the machine word 2^w.
- Improvement 6 – Improved Tuning
  - Tuning the Fermat and Mersenne Transforms
  - Tuning the Plain Integer Multiplication
- TODO

  In fact, some of these improvements are still on their TODO list:
  - Implement some of the tricks published at ISSAC'2007 by Gaudry, Kruppa, and Zimmermann.
  - It might be possible to avoid a small number of MPN_COPYs by using arotating temporary or two.
  - Cleanup and simplify the code!

17