RESEARCH ARTICLE OPEN ACCESS

Comparative Study between Karatsuba Algorithm and FFT Algorithm

Siham Zoubir[1], Abderrahim Tragha[2]
[1,2]*(Department of Modeling and information technology (TIM) / University Hassan II Mohammedia, Faculty of sciences Ben M'sik, Casablanca)*

ABSTRACT
The product of polynomials and integers is an elementary operation, which intervenes in an impressive number of algorithms of the formal calculation. The efficiency of these algorithms is bases on product. To multiply two polynomials of n degree that has coefficients in a ring A, the classic method requires O(n²) operations in A. Also, the school algorithm of multiplication of two integers with n number required a number of binary operations there O(n²).We present in this article several algorithms of fast multiplication, which is Karatsuba algorithm with complexity O(n1,59), and Fast Fourier transform, with the complexity is essentially linear to n. The problems land on this article concern the arithmetical complexity of the multiplication of polynomials with a variable and the binary complexity of the Multiplication of integers.

Keywords: multiplication, algorithm, Karatsuba, Fast Fourier Transform (FFT)

I. INTRODUCTION

The hidden constants in O (·) are determining for the practical efficiency of such algorithms. Let us speak first about the polynomial case, for example when A is a finished body by reasonable size (typically, elements of which are represent on some machine words).

In the best current setting-up (magma, NTL):

- The algorithm of Karatsuba beats the naïve algorithm for about degrees 20;
- The methods have base of FFT in O (n log n) win for degrees of the order of 100, but cannot be utilizes for arbitrarily big degrees (comes a moment or we are lacking roots of the unity);FFT's algorithm in O (n log n log log n) is used for degrees of the order of some tens or hundreds thousands. Certain problems, in cryptology or in theory numbers, require to treat polynomials of degree of the order of 100 000, Sizes that need obligatory the fast algorithms.
- The setting-up of the fast algorithms for integers is delicate because of the restraints.

In the best current setting-up (magma, GMP):

- The algorithm of Karatsuba beats the naïve algorithm for numbers of the order of 100 binary numbers;
- The methods have base of FFT (Schonhage-Strassen) win for numbers 10 000 binary numbers.

  Again, problems come from cryptology or from number theory that need to treat numbers of colossal size (of the order of 100 000 000 numbers; needs 10 Mb to store such a number). This justifies amply the efforts of setting-up the fast algorithms.

II. ALGORITHM OF KARATSUBA

1. Presentation of the algorithm

A first refinement of the naïve algorithm bases on the following remark: it is possible to win a multiplication for the product of the polynomials of degree 1.

Are indeed has to multiply polynomials

$$F = f_0 + f_1X \text{ et } G = g_0 + g_1X$$

The product H = FG is written

$$H = f_0g_0 + (f_0g_1 + f_1g_0)X + f_1g_1X^2$$

Make all 4 products f_0g_0, f_0g_1, f_1g_0, f_1g_1 Corresponds to the quadratic algorithm but we can do better by noticing that the coefficient of X is written

$$f_0g_1 + f_1g_0 = (f_0 + f_1)(g_0 + g_1) - f_0g_0 - f_1g_1$$

This writing is leads to an algorithm which makes in total 3 multiplications and 4 additions. We lost some additions compared to the naïve algorithm, but the earnings of a multiplication are going to be transformed into earnings in the exhibitor of the algorithm, by recursive application.

Indeed let us cross in the general case any degrees. Inspire by the previous observation, we are going to split F and G into two.

We thus suppose that F and G are of degree in most n - 1, and that the integer n is even, n = 2k. We pose then:

$$F = F^{(0)} + F^{(1)}X^k,\ G = G^{(0)} + G^{(1)}X^k$$

F(0), F(1),G(0),G(1) having degrees in most k − 1. the product H = FG is writing
$H = F^{(0)}G^{(0)} + (F^{(0)}G^{(1)} + F^{(1)}G^{(0)})\, X^k + F^{(1)}G^{(1)}X^{2k}$

To write algorithm, we suppose that n is a power of 2 to be able to make all the appeals recursive which we wish. we obtain then:

Algorithm of KARATSUBA

```
Input: F,G of degree in most n - 1, n being a power of 2.Output : H = FG.
1. If  n = 1, return FG.
2. Calculate $A_1 = F^{(0)}G^{(0)}$ et $A_2 = F^{(1)}G^{(1)}$ Recursively.
3. Calculate $A_3 = F^{(0)} + F^{(1)}$ et $A4 = G^{(0)} + G^{(1)}$.
4. Calculate $A_5 = A_3A_4$ Recursively.
5. Calculate $A_6 = A_5 - A_1$ et $A_7 = A_6 - A_2$.
6. Return  $A_1 + A_7X^{n/2} + A_2X^n$.
```

2. Analysis of the algorithm KARATSUBA

By induction on n = max ([log2 a]; [log2 b]) + 1:

- If n = 1 then, we make only one multiplication, thus the algorithm is end.
- Let us suppose that for all n < n0, the algorithm KARATSUBA (a; b) ends, then for all a and b with n0 = max ([log2 a]; [log2 b]) + 1, KARATSUBA (a; b) also ends because he calls recursively three copies of KARATSUBA on smaller authorities (which by induction hypothese end).

By induction on k = max ($[\log_2 a]$; $[\log_2 b]$) + 1:

- If k = 1 then, KARATSUBA (a; b) return ab.
- Let us suppose that for all k < n0, the algorithm KARATSUBA (a; b) return ab, then for all a and b with n_0 = max ($[\log_2 a]$; $[\log_2 b]$) + 1:

KARATSUBA (a; b) = y x 2^{2k} + (x +y - sgn (a_1- A_0) sgn (b_1- B_0) z) x 2^k + x;

Who costs ab because by recurrence x = a_0b_0, = a_1b_1 and z = (|a_1- A_0) (|b_1- B_0)).

Analysis of the Complexity in number of elementary multiplications :

Let us note T (n) the number of necessary elementary multiplications to multiply two numbers of n numbers

The principle of Divide to rule allows to give the formula of recurrence for T :

T(1) = 1 et T(n) = 3T(n=2):

We deduce T(n) = (-) ($n^{\log 2\ 3}$) : $\log_2 3 = \ln(3)/\ln(2)$ ≈ 1:585; witch is more better than n² of naive multiplication.

III. ALGORITHM OF FAST FOURIER TRANSFORM

1. Presentation

The methods with Fast Fourier transform are what we know to do best to multiply polynomials.

To simplify the presentation, we suppose here that we try to multiply polynomials F and G in A [X], strictly lower degrees to n/2 (or more generally such as deg FG < n).

We make (temporarily) the hypothesis that the ring A is of cardinal at least n. We give ourselves different points a_0,......a_{n-1} in A.

The principle of the multiplication by processing of Fast Fourier is as following:

1. Evaluation. We calculate the values:

$Ev(F) = (F(a_0), . . . , F(a_{n-1}))$; $Ev(G) = (G(a_0), . . . ,G(a_{n-1}))$

2. Produce point to point

$Ev(F), Ev(G) \rightarrow Ev(FG) = (FG(a_0), . . . , FG(a_{n-1}))$

3. Interpolation

Ev(FG) → FG.

This operation returns to find the coefficients of FG from his values in a_0.......a_{n-1} (has to suppose that this operation of interpolation is possible, see below).

In matrix terms, the operation F → Ev (F) is linear and his matrix (for polynomials F of degree in most n-1, in the monomial base{1, X.... X^{n-1}}) is the matrix of Vandermonde.

2. Programming

We give ourselves the prime number P as well as the number N which is a power of 2 and which divides N - 1. We also determine a generator g of the group Cyclic U (p), what allows to find a primitive root N^{ème} of the unity in U (p), or W. Roots N^{ème} of the unity are the successive powers of W, returned modulo P. They are obtained by the following function, which records them in the board W [N]:

```
void rootnemesof1(int ww) /* in the principal program, we take ww = W */
{ int i;
w[0]=1; /* the board w[] should be declared in global */
for(i=1;i<N;i++) { w[i]=ww*w[i-1]; while (w[i]>=P) w[i]-=P; }
}
```

the gives coefficients *a*[] of polynomial *P are* put in the order of the bits inverted and placed in Board *t*[], as we do previously, this board *t*[] is declared in global like :

```
void bitreversedeaverst(int aa[]) /* this function is applied in initial polynomial P */
{ int i,j;
for(i=0;i<N;i++) { j=rev(i); t[j]=aa[i];} /* Take back the function rev () treated previously */
}
```

These two functions are going to be integrated into the function fft () which takes as variables the board of Coefficients a[] of the polynomial the transformed of Fast Fourier which we want to have, as well as Primitive root W used for it. It is about a program similar to that of the transformed of Fourier with complex numbers, except that in this case, it is about integers Much simpler, than we content with returning modulo P during the calculations. The transformed of Fourier of the polynomial is placed in the end in the board t [] the contents of which were transformed during execution of the function fft ().

```
void fft(int aa[],int ww)
{ int j,k,s,n,u,v;
rrootnemesof1(ww);
S=(log(N+0.0001)/log(2.)); /* S is declqred in global */
bitreversedeaverst(aa);
n=2;
for(s=1;s<=S;s++)
{ for(j=0;j<n/2;j++) for(k=j;k<N; k+=n)
{ u=t[k];
v=w[N/n*j]*t[k+n/2]; while(v>=P) v-=P;
t[k]=u+v; if (t[k]>=P) t[k]-=P;
t[k+n/2]=u-v; if (t[k+n/2]>=P) t[k+n/2]-=P; if (t[k+n/2]<0) t[k+n/2]+=P;
}
n=2*n;
}
}
```

the principal program include an input of polynomial P with their coefficients *a*[]. than the *fft*() function is called, and the result is posted in the board *t*[], or if we want in board *ffta*[].

```
for(i=0;i<N;i++) a[i]=rand()%(N-1)+1; /* example of polynomial */
printf("\n Polynomial a avec ses %d coefficients :\n",N); for(i=0;i<N;i++) printf(" %d ",a[i]);
fft(a,W);
printf("\n Transformed of Fourier de a\n"); for(i=0;i<N;i++) { ffta[i]=t[i]; printf(" %d ", ffta[i]);}
```

IV. CONCLUSION

The Russian mathematician A.N. Kolmogorov had guess at the beginning of 1960s that it would be impossible to multiply two integers of n numbers in a binary cost under-quadratic . In 1962 this guess was set aside by Karatsuba and Ofman. A generalization of the algorithm was proposed a few years later by Toom and CoOK. The algorithm of Toom-Cook has a binary complexity of O ($n\ 32^{\sqrt{\log n}}$) to multiply two integers of binary size n; This result can be matter in the polynomial world.

He founding article of Cooley and Tukey is doubtless one furthermore quote in computing. The article constitutes a good reference for the reader avid to get acquainted with the myriad of techniques of type FFT. An important overhang was in the discovery of Schonhage and Strassen of the equivalent result for the multiplication of integers. For a long time, we believed that this algorithm could not present that a purely theoretical interest.To date, most of the non-specialized software of formal calculation have a fast multiplication of integers: Maple and Mathematical uses the library GMP, Magma has its own setting-up.

ACKNOWLEDGEMENTS

FFT: Transformed of Fast Fourier

REFERENCES

[1]. Karatsuba A. A. The complexity of computations // Proc. Steklov Inst. Math., 211, 169–183 (1995); translation from Trudy Mat. Inst. Steklova, 211, 186–202 (1995)

[2]. Richard Brent, Paul Zimmerman, Modern Computer Algebra, Cambridge University Press, 2010.

[3]. Nussbaumer (Henri J.). – Fast polynomial transform algorithms for digital convolution. Institute of Electrical and Electronics Engineers. Transactions on Acoustics, Speech, and Signal Processing, vol. 28, n°2, 1980, pp. 205–215.

[4]. Cantor (D. G.) and Kaltofen (E.). – On fast multiplication of polynomials over arbitrary algebras. Acta Informatica, vol. 28, n°7, 1991, pp. 693–701.

[5]. van der Hoeven (Joris). – The truncated Fourier transform and applications. In ISSAC 2004, pp. 290–296. – ACM, New York, 2004.

[6]. Bernstein (D. J.). – Multidigit multiplication for mathematicians. – Preprint, available from http://cr.yp.to/papers.html.

[7]. G. BAUDOIN et J.-F. BERCHER - TRANSFORMÉE DE FOURIER DISCRÈTE - École Supérieure d'Ingénieurs en Électrotechnique et Électronique. Novembre 2001 – version 0.1.

[8]. Joseph COHEN - Laboratoire LSD, Institut IMAG, Université J. Fourier, B.P . n° 53X, 38401 GRENOBLE CEDEX, France .

[9]. Jacques MAX - CENG-LETI, BP n° 85X, 38041 GRENOBLE CEDEX, France.

[10]. Joseph COHEN - Laboratoire LSD, Institut IMAG, Université J. Fourier, B.P . n° 53X, 38401 GRENOBLE CEDEX, France . Jacques MAX CENG-LETI, BP n° 85X, 38041 GRENOBLE CEDEX, France .