41862 - Seminar Advanced Algorithms

Schönhage-Strassen algorithm for fast integer multiplication

Michael Senn michael.senn@students.unibe.ch — 16-126-880

2022-01-14

This report serves as an introduction to the fast integer multiplication algorithm described by Schönhage and Strassen.

1 Introduction

This report aims to provide the reader an introduction into the fast integer multiplication algorithm proposed by Schönhage and Strassen in 1971. [SS71]

It will start with a discussion of the problem to be solved and historical solutions in chapter 1. Chapter 2 will introduce the required mathematical background. Chapter 3 will show how to model the problem of integer multiplication as a convolution problem, along with a basic algorithm to solve it. Chapter **??** will then introduce the algorithm, and explain its workings. Finally we will conclude in chapter **??**.

1.1 Problem statement

The problem the Schönhage-Strassen algorithm intends to solve is to, given two integers $x, y \in \mathbb{Z}$, calculate their product $z \coloneqq x \cdot y$.

1.2 Historical algorithms

1.2.1 Long multiplication

One of the most common algorithm for multiplication taught in school is known as 'long multiplication'. Given a representation of the multiplicand and multiplier in some base — commonly base 10 — each digit of the multiplicand is multiplied by each digit of the multiplier. All intermediary products encoding for the same digit of the chosen base will then be summed up. Finally a potential carry propagation takes place, yielding a representation of the product in the chosen base.

It is trivial to see that this algorithm requires $O(n^2)$ multiplication and additions.

2 Mathematical background

This chapter will provide a brief introduction into the discrete Fourier transform and discrete weighted transform, along with a way to efficiently calculate either by means of the fast Fourier transform. It will then provide an introduction into convolution theory, and show how convolutions and the discrete Fourier transform are linked.

2.1 Notation

The following defines notation used throughout the report.

2.2 Discrete Fourier Transform

The Discrete Fourier Transform ("DFT") allows performing Fourier analysis of a discrete n-length signal $x = (x_0, x_1, \ldots, x_{n-1})$ over some algebraic field. Its result is the the n-length signal X, whose components are as follows:[CP05]

$$\mathrm{DFT}(x) : X_k = \sum_{j=0}^{n-1} x_j \cdot g^{-jk}$$

Where g is a primitive n-th root of unity of the algebraic field, which is to say that for some $n \in \mathbb{N}$:

$$g^n = 1 \wedge g^m \neq 1 \ \forall m < n$$

The inverse DFT is defined equivalently as the n-length signal x whose components are as follows:

$$\mathrm{DFT}^{-1}(X) : x_j = \frac{1}{n} \sum_{k=0}^{n-1} X_k \cdot g^{jk}$$

Such that $\mathrm{DFT}^{-1}(\mathrm{DFT}(x)) = x$ for all discrete signals x.

2.2.1 Calculation via fast Fourier Transform

The naive approach to calculate the DFT would require $O(k^2)$ multiplications and additions. However it can be shown that the DFT can be efficiently calculated by means of the Fast Fourier Transform ("FFT") algorithm, which has a complexity of $O(n \log(n))$:[CP05]

$$\mathrm{DFT}(x) = \mathrm{FFT}(x)$$

2.3 Discrete Weighted Transform

We further introduce the forward and inverse Discrete Weighted Transform ("DWT") of a discrete n-length signal x and n-length weight vector a, defined to be the n-length signal X with components as follows:

$$\mathrm{DWT}(x, a) : X_k = \sum_{j=0}^{n-1} (a_j x_j) \cdot g^{-jk}$$

$$\mathrm{DWT}^{-1}(X, a) : x_j = \frac{1}{n a_j} \sum_{k=0}^{n-1} X_k \cdot g^{jk}$$

Again with g a primitive n-th root of unity of the field.

Clearly from the definition it follows that $\mathrm{DWT}(x, a) = \mathrm{DFT}(a \cdot x)$, where $\cdot$ denotes component-wise multiplication in the algebraic field. This also implies that $\mathrm{DWT}(x, 1) = \mathrm{DFT}(x)$.

2.4 Convolution theory

We now introduce the concept of a convolution — an operation $*$ on two functions f and g producing a third function $f * g$:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau$$

As we work with discrete rather than continuous signals however, we will introduce a set of different discrete convolution operations. For the following paragraphs, assume x and y to be two n-length discrete signals.

2.4.1 Cyclic convolution

The cyclic convolution $z = x \times y$ is defined as the n-length signal with components as follows. It is visualized in figure 2.1a:

$$z_k \coloneqq \sum_{i+j \equiv k \pmod n} x_i \cdot y_i$$

2.4.2 Weighted cyclic convolution

The weighted cyclic convolution $z = x \times_a y$, with an n-length weight vector a, is defined as the n-length signal with components as follows. It is visualized in figure **??**.

$$z_k = \frac{1}{a_k} \sum_{i+j \equiv k \pmod n} (x_i a_i) \cdot (y_j a_j)$$

2.4.3 Acyclic convolution

The acyclic convolution $u = x \times_A y$ is defined as the $2n$-length signal with components as follows. It is visualized in figure 2.1b:

$$u_k \coloneqq \sum_{i+j=k} x_i \cdot y_i$$
$$u_{2n-1} \coloneqq 0$$

2.4.4 Halfcyclic convolution

The halfcylic convolution $w = x \times_H y$ is defined as the n-length signal consisting of the first n components of the acylic convolution $x \times_A y$. It is visualized in figure 2.1c:

$$w_k \coloneqq (x \times_A y)(k)$$

2.4.5 Negayclic convolution

The negacylic convolution $v = x \times_- y$ is defined as the n-length signal with components as follows. It is visualized in figure 2.1d:

$$v_k \coloneqq \sum_{i+j=n} x_i \cdot y_i - \sum_{i+j=n+k} x_i \cdot y_i$$

It can be seen that the negacylic convolution splits the summands of each signal component of the cyclic convolution into those where the modular addition operation of the indices 'wrapped around', and those where it did not. Those where it wrapped around are then subtracted from the sum of those where it did not.

2.4.6 Relations between convolutions

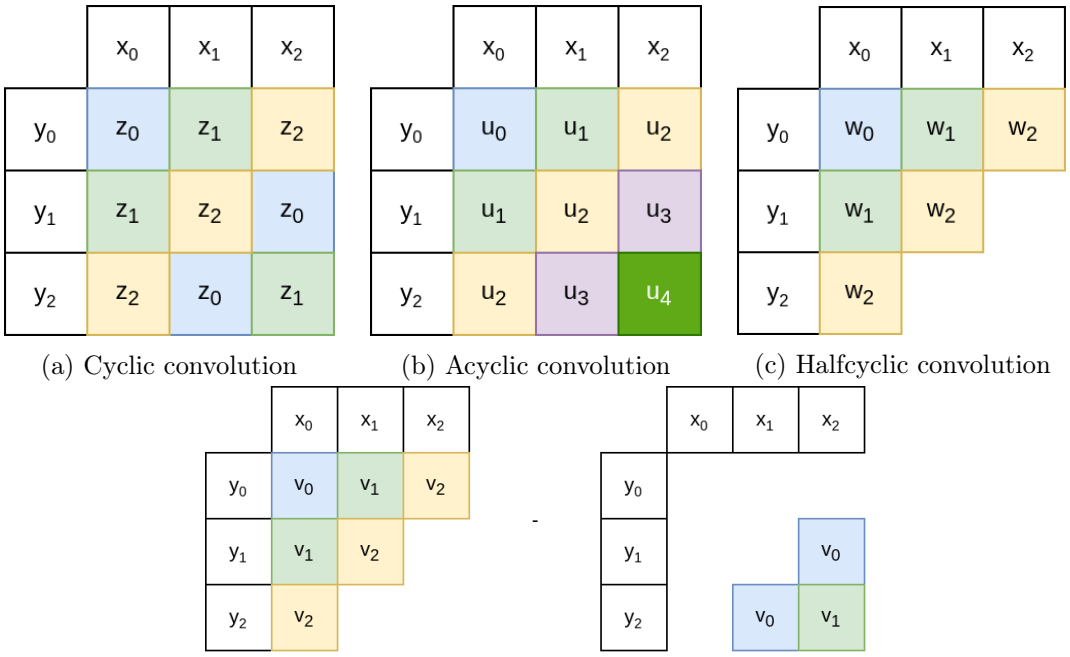
Various relations between the different types of convolutions exist. One we will use is the following. Let $L(x)$ be the left-padding of the n-length signal x to length $2n$ with signal components of value 0. Then it holds that: [CP05]

$$L(x) \times_A L(y) = x \times y = x \times_- y$$

That is the acyclic convolution of the left padded signals is equal to the cyclic and negacyclic convolution of the unpadded signals.

2.4.7 Convolution theorem

It can further be shown that the cyclic convolution and weighted cyclic convolution can be calculated by means of the DFT and DWT: [CP05]



(a) Cyclic convolution (b) Acyclic convolution (c) Halfcyclic convolution

(d) Negacyclic convolution

Figure 2.1: Different types of discrete convolutions. The input signals $x = (x_0, x_1, x_2)$ and $y = (y_0, y_1, y_2)$ are shown along the axes. The 'cross product' of the input signals represents components of the convolution's output, with labels and colours indicating towards which component of the output the respective product contributes. Unless specified otherwise, each output signal component is the sum of all products with the corresponding label and colour.

$$x \times y = \mathrm{DFT}^{-1}(\mathrm{DFT}(x) \cdot \mathrm{DFT}(y))$$
$$x \times_a y = \mathrm{DWT}^{-1}(\mathrm{DWT}(x, a) \cdot \mathrm{DWT}(y, a), a)$$

Where $\cdot$ denotes component-wise multiplication. This allows calculation of convolutions with complexity $O(n \log(n))$.

By choosing an appropriate weight vector a, this also allows the calculation of other convolutions. Consider choosing the weight vector $a = (a_j) = A^j$ for a scalar A. As described by Crandall and Fagin [CF94], the weighted cyclic convolution of two length-n signals will then take on the following form:

$$x \times_a y = (x \times_H y) + A^n(x \times y - x \times_H y)$$

Assume then that A is a primitive $2n$-th root of unity. That is $a^{2n} = 1 \Rightarrow a^n = -1$. Then clearly the result of the weighted cyclic convolution will be the negacyclic convolution:

$$x \times_a y = (x \times_H y) - (x \times y - x \times_H y) = 2 \cdot (x \times_H y) - (x \times y) = x \times_- y$$

3 Modelling integer multiplication as a convolution problem

We now aim to show how the problem of integer multiplication can be phrased as a convolution problem, and then present a basic algorithm to perform integer multiplications using the FFT.

Consider x, y two n-digit numbers in some base B, where x_0, y_0 is the least significant digit.

3.1 Integer multiplication as a convolution operation

Let $\otimes$ be an integer multiplication with delayed carry operation. That is $u = x \otimes y$ is defined as follows:

$$u_i = \sum_{j+k=i} x_j y_k$$

Observe how by this definition the operation $u = x \otimes y$ is exactly the acyclic convolution $u = x \times_A y$. This is further illustrated in figure 3.1.

3.1.1 Applying the delayed carry operation

Given $u = x \otimes y$, one can calculate the result of a regular multiplication $v = x \cdot y$ by applying the carry operation. Let div denote integer division:

$$\begin{aligned} \overline{v}_i &= u_i + (\overline{v}_{i-1} \operatorname{div} B) \\ v_i &= \overline{v}_i \bmod B \end{aligned}$$

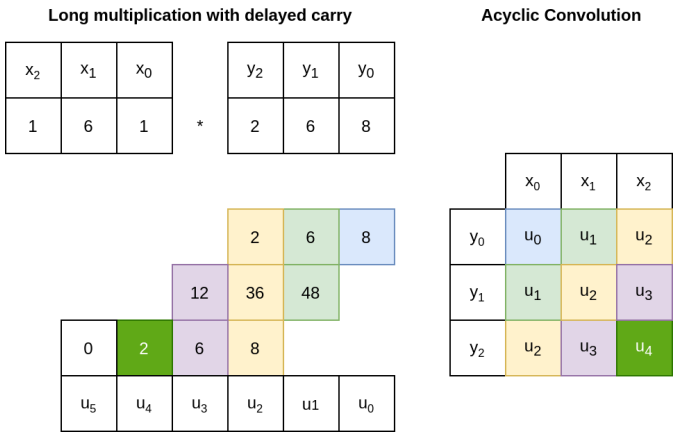


Figure 3.1: Integer multiplication with delayed carry operation is equivalent to acyclic convolution

3.2 Integer multiplication with the FFT

This now allows specification of a basic algorithm for fast integer multiplication using the FFT, as done by e.g. Crandall and Pomerance, shown in algorithm 1.

The algorithm starts by zero-padding the two input signals to length $2n$, and then calculates $\mathrm{DFT}^{-1}(\mathrm{DFT}(x_{\text{padded}}) \cdot \mathrm{DFT}(y_{\text{padded}}))$. Recall that as per sections 2.4.6 and 2.4.7 this will thus — after the rounding required as this generic FFT operates over the field of complex numbers — yield the acyclic convolution $x \times_A y$.

The algorithm will then perform the carry operation outlined above, returning the result $z = x \cdot y$.

Algorithm 1 Fast integer multiplication with FFT

function FFTMULT(x, y)
    $x \leftarrow$ PAD(x, 2n) ▷ Zero-pad to length $2n$ on high-order side
    $y \leftarrow$ PAD(y, 2n)

    $X \leftarrow$ DFT(x)
    $Y \leftarrow$ DFT(y)
    $Z \leftarrow X \cdot Y$ ▷ Dyadic product
    $z \leftarrow \mathrm{DFT}(Z)^{-1}$
    $z \leftarrow$ ROUND(z)

    $carry \leftarrow 0$
    for $i \leftarrow 0$ to $2n$ **do**
        $v \leftarrow z_i + carry$
        $z_n \leftarrow v \bmod B$
        $carry \leftarrow v \operatorname{div} B$
    end for
    Return $z_0||z_1||\ldots||z_n||carry$

end function

Bibliography

[SS71] A. Schönhage and V. Strassen. "Schnelle Multiplikation großer Zahlen". In: *Computing* 7.3-4 (Sept. 1971), pp. 281–292. ISSN: 0010-485X, 1436-5057. DOI: 10.1007/BF02242355. URL: http://link.springer.com/10.1007/BF02242355 (visited on 12/13/2021).

[CF94] Richard Crandall and Barry Fagin. "Discrete Weighted Transforms and Large-Integer Arithmetic". In: *Mathematics of Computation* 62.205 (1994), pp. 305–324. ISSN: 0025-5718, 1088-6842. DOI: 10.1090/S0025-5718-1994-1185244-1. URL: https://www.ams.org/mcom/1994-62-205/S0025-5718-1994-1185244-1/ (visited on 01/14/2022).

[CP05] Richard E. Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. 2nd ed. New York, NY: Springer, 2005. 597 pp. ISBN: 978-0-387-25282-7.