LIMA: Hardware for FFT based Large Integer Multiplication

James Nguyen, Michael Cai, Ziyi Zuo, Larry Tang, Ken Mai, Franz Franchetti
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA
jnnguyen, lawrenct, franzf@andrew.cmu.edu

***Abstract*—Applications in theorem proving and cryptography heavily rely on the multiplication of large integer values. Utilizing properties of the Fourier Transform, one can multiply two values in $\mathcal{O}(n \log n)$ time as compared to the traditional $\mathcal{O}(n^2)$ grade school algorithm. In this work, we propose a FFT-based multiplication algorithm that can be computed $\mathcal{O}(n \log n)$ time, utilizing interval arithmetic to circumvent the real number uncertainty. This work will also present hardware for FFT based large integer multiplication, LIMA, which is a test chip implementing of a portion of this algorithm fabricated on the TSMC 28 nm process. We aim to demonstrate LIMA as a high throughput implementation of this portion of the algorithm, with the intent of being able to run at a 1 GHz with a area of 1.9 mm^2. LIMA utilizes custom double precision floating point and FIFO architectures optimized for area and high clock speed operation.**

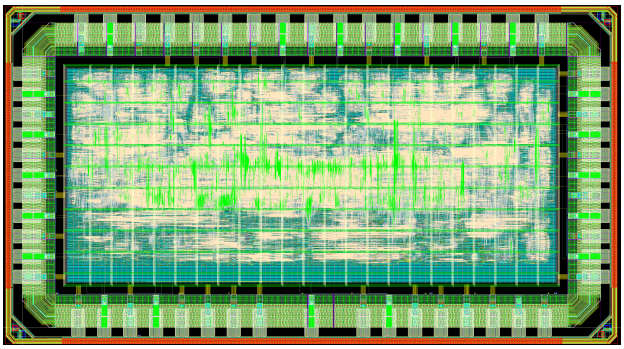

Fig. 1. LIMA Die Shot

I. Introduction

The traditional "grade school multiplication," with a $\mathcal{O}(n^2)$ asymptotic complexity for the multiplication of two n-bit numbers, is far too slow for large integer values. Although there exist alternative algorithms that are faster than the grade school algorithm, such as the Karatsuba or the Conba algorithm, these algorithms are nonetheless still in the realm of polynomial time [1] [2] [3].

We propose a FFT based multiplication algorithm that can be computed in $\mathcal{O}(n \log n)$ time, utilizing interval arithmetic to circumvent the real number uncertainty of traditional FFT algorithms, which inputs and outputs real values. If the range of uncertainty is small enough and includes a single integer value, then we can be confident that our result is that integer value.

We present LIMA, a hardware implementation of a portion of this algorithm fabricated on the TSMC 28 nm process. Our design implements the Radix-4 FFT and the twiddle factor multiplication portion of the algorithm. We aim to demonstrate LIMA as a high throughput implementation of this portion of the algorithm, utilizing FIFOs to deliver test vectors to the design. Utilizing custom floating point, we are able to achieve a area of 1.9 mm^2 with a simulated clock frequency of 1 GHz.

II. Large Integer Multiplication Algorithm

A. Fast Fourier Transform

The FFT is an algorithm that computes the DFT with a time complexity of $\mathcal{O}(n \log n)$. FFT based multiplication algorithms utilize the convolution theorem, which states that convolution in one domain is multiplication in the other. Thus a convolution of two integer values in the frequency domain is equivalent to multiplication of those two values in the time domain. To perform the FFT on these integer values, we can represent them as a sequence, with each element being a digit of the integer. Then utilizing the FFT, these integer sequences are converted from time domain to frequency domain, where they are convolved, which can be done by doing a point-wise multiplication and carry addition. Then a inverse FFT (IFFT) is taken over the result to get the final multiplied value back in the time domain. Thus the multiplication algorithm is as follows:

$$y = IFFT(FFT(x_1) * FFT(x_2)) \tag{1}$$

The overall algorithm's time complexity is dictated by the time complexity of the FFT, thus the multiplication algorithm is $\mathcal{O}(n \log n)$. LIMA implements the FFT portion of the algorithm, specifically the Radix-4 variation of the FFT. The radix indicates the size of the butterfly matrix that is used to compute the FFT. The butterfly matrix can be generalized as two stages of complex adds and subtracts, with the exception of a single multiply with j. However, this calculation can be done without doing a multiply by negating the complex output and swapping the real and complex portions of the output (so the complex part becomes the real part and vice versa). The outputs of the butterfly matrix are then multiplied with the

corresponding twiddle factor, which are constant values that are used to compute the FFT.

B. Interval Arithmetic

The main issue of simply using FFTs to do integer multiplication is that FFTs output real values instead of integer values. Real numbers present some amount of uncertainty to the true integer value of the multiplication. However, we can circumvent this by using interval arithmetic, which provides an upper and lower bound, or in other words, a interval, to the real value result of the FFT. If the upper and lower bounds of the interval vary by one machine epsilon to the true value and ranges over a integer value, then we can be certain that the true value of the interval is that integer value. The interval arthmetic operations for addition (2), subtraction (3) and multiplication (4) are shown as:

$$(x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2) \tag{2}$$

$$(x_1, x_2) - (y_1, y_2) = (x_1 - y_2, x_2 - y_1) \tag{3}$$

$$(x_1, x_2) * (y_1, y_2) = (min(x_1 * y_1, x_1 * y_2, x_2 * y_1, x_2 * y_2), max(x_1 * y_1, x_1 * y_2, x_2 * y_1, x_2 * y_2)) \tag{4}$$

Where (x_1, x_2) represents the lower and upper bound of the interval x and (y_1, y_2) represents the lower and upper bound of the interval y. Since LIMA operates on complex real values, intervals are computed for both the real and complex portions of the FFT output.

III. Accelerator Design

A. Floating Point Units

The floating point units employ two key area and timing saving optimizations. There are only two rounding modes implemented by the floating point units, round to positive infinity and round to negative infinity. These rounding modes ensure that the interval operations result in worst case intervals. Hence, for any operation to generate the lower interval, the floating point unit will round to negative infinity and vice versa. The floating point units also do not deal with denormalized numbers, assuming them to be infinity.

IV. LIMA Implementation

A. Computation Kernels

The Radix-4 FFT, shown in Fig. 2 is split into two blocks, the Radix-4 Interval Kernel and the Complex Diagonal Multiply Block. The entire computation kernel has a pipeline depth of 13 stages. The Radix-4 Interval Kernel is two stages of complex interval additions and subtractions, which are interval addition and subtraction on both the real and the complex parts of the complex interval inputs. Each interval addition/subtraction subblock uses four floating point adders. The Complex Diagonal Multiply Block multiplies the output of the Radix-4 Interval Kernel with the corresponding twiddle factors fed from the twiddle FIFO. The complex multiply operation employs the two floating point multiplier optimization, and uses four floating point multipliers and two floating point adders. Overall, the computation kernels consists of 48 floating point adders and 32 floating point multipliers. The total area of the computation kernel is 0.75 mm^2.

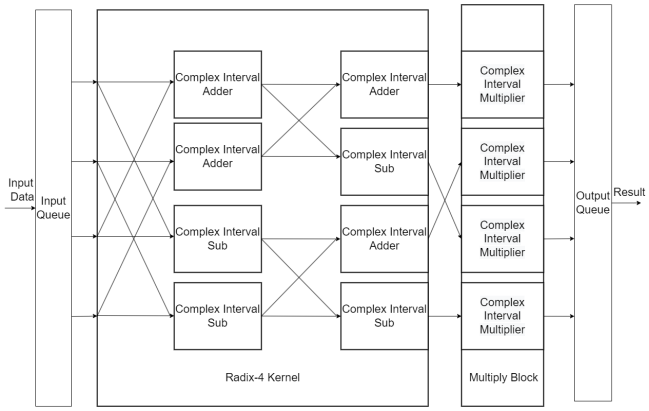

Fig. 2. LIMA Block Diagram

B. Top Level Design

The top level block diagram is shown in Fig. 2. Virtuoso floorplan diagrams is shown in Fig. 1. LIMA was synthesized to a 1 GHz clock speed, with a area of 1.9 mm^2. The dimensions for the chip are 1.9 mm by 1.0 mm. Power simulations of the chip predict that it operates on a average power draw of 922 mW during continuous operation. The latency for a single computation is 13 cycles.

Due to size constraints, LIMA instead utilizes FIFOs to send test vectors through the computation kernels. Because of this, LIMA is not able to properly compute a FFT of a meaningful size fully on chip, however, we are able to demonstrate that we are able to run the FFT computation at 1 GHz. Future work on LIMA would involve adding memory to properly compute multiplication of two integers.

V. Conclusion

This work presents a traditional FFT based large integer multiplication accelerator design fabricated on the TSMC 28nm process. Future work will involve implementing memory instead of FIFOs to be able to hold the intermediate values of the FFT to properly compute FFT on chip. The partial sums addition and inverse FFT will also be implemented, with the whole end to end algorithm being implemented to run the full integer multiplication algorithm.

References

[1] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," IEEE Transactions on Computers, vol. 66, no. 8, pp. 1369–1382, 2017.

[2] J.-H. Ye and M.-D. Shieh, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 9, pp. 1727–1736, Sep. 2018.

[3] W. Wang, X. Huang, Niall Emmart, and C. Weems, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 9, pp. 1879–1887, Sep. 2014.