Abstract—Though optical computing has been unable to compete with CMOS technology in mainstream computing, its natural capacity for computing Fourier transforms gives it an advantage in convolution-type problems. This may make optics a viable approach in post-Moore’s era computing. This paper presents an optical approach to one suitable yet nontrivial problem: modular multiplication. We first explore the mathematical details of performing a basic optical convolution using lenses and masks. Then we discuss a simulation tool developed by the authors to explore some design considerations for an optical convolution circuit. Finally, we lay out an algorithm for performing Montgomery modular multiplication in an optical system along with simulation results for an all-optical implementation. The proposed approach presents an extremely energy efficient solution to computing 16-bit modular multiplication without the need for analog-digital conversions in intermediate steps.

Index Terms—optical computing, convolution processor, Fourier optics, optical simulation, Montgomery multiplication

I. INTRODUCTION

Optical computing, performing calculations using photons instead of electrons, is an idea that has been around for decades [1], [2], [3], [4]. However, like many competing non-CMOS technologies, it was ultimately beat out by Moore’s Law scaling of transistors. Optics, being fundamentally limited by the wavelength of light, cannot attain the density of transistors which, in 2017, achieved a 10nm process technology in commercial electronics [5]. Furthermore, as an analog technology typically limited to 8-12 bits of resolution [6], optical computing does not compete with digital technologies in terms of computational accuracy. In spite of the hegemony of CMOS technology for computing, there continues to be active research in methods of leveraging optics for computation. With the recent slowing of Moore’s Law scaling accompanied by advances in nanophotonics [7] and some initial commercial adoption of optical computing technologies [8], the research community has demonstrated renewed interest in exploring the role of optics in post-Moore’s era computing [9].

Multiplication can be a very expensive procedure when done with large numbers. A key strength in optical computing is the use of Fourier transforms to turn multiplication from an $O(n^2)$ problem into an $O(n)$ one. The reason this has not been advantageous in digital computing is the high overhead cost of computing Fourier transforms [10], [11]. Light performs this operation natively using lenses and masks [12], [13], making it a superior medium for convolution-type problems.

Optical correlators designed to perform multiplication, particularly vector-matrix multiplication, are not new [14]. Neither are the optical simulations used to design such systems. However, the authors believe this to be the first time that such correlators have been demonstrated by simulation as a viable design for 16-bit modular multiplication in optics. Therefore, the contributions of this work are the end-to-end simulation of a novel application of Fourier optics to the implementation of high precision arithmetic and a new approach to computing modular multiplication based on discrete convolution.

This paper is organized as follows. First, we look at Fourier transforms and how convolution can be achieved optically using lenses and masks. Then we shift to describe the implementation of a Fourier optics simulation, enabling us to explore some physical design details through simulation experiments. Finally, we transition to consider a modular multiplication algorithm and present the design for an all-optical circuit solving 16-bit modular multiplication problems.

II. OPTICAL CONVOLUTION

A. Fourier Transforms

Function convolution is the continuous form of $O(n^2)$ multiplication. By the convolution theorem we can transform a convolution operation into a pointwise product of Fourier transforms, the continuous analog of an optical convolution circuit. This procedure is stated below. First we Fourier transform a function $f$, then perform a pointwise multiply, then inverse transform.

$$\mathcal{F}\{f\} = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} \, dx$$  \hspace{1cm} (1)

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

$$f(x) * g(x) = \int_{-\infty}^{\infty} \mathcal{F}\{f * g\} e^{i\omega y} \, dy$$  \hspace{1cm} (2)

To build some intuition for this, notice that a single point of the Fourier transformed function (Equation 1) is the integral...
of the original equation with each point attached to a different phase factor. The phase factor has a special property that allows us to perform an inverse transform, but the key idea is that a single point in \( F \{f \} \) contains the sum of all points in \( f(x) \) distinguished by phase. By the distributive property, the product of \( F \{f \}(a) \) and \( F \{g \}(a) \) contains the sum of all possible products of points in \( f(x) \) and \( g(x) \). In doing a pointwise multiply, we construct this sum in the domain of the resulting function, making it exactly the Fourier transform of a convolution. Finally, we perform an inverse transform (Equation 2) to “de-sum” each point and retrieve the convolution.

To see how this applies to multiplication of integers, consider each integer as a string of digits. Now choose some domain over which to define a function and break it into segments, one for each digit. Construct a piecewise function such that each segment takes on the value of its corresponding digit. Convolution of this function with another similarly constructed function corresponds with multiplication of the two numbers. This means that we can use the Fourier transform to turn this convolution into a pointwise multiply, the equivalent of turning an \( O(n^2) \) operation into an \( O(n) \) one.

**B. Lens/Mask Configuration**

Light naturally performs a Fourier transform when passed through the right configuration of lenses and masks. To understand this, consider the equation that governs the electromagnetic field \( U_1 \) in a plane behind a mask [15], [16]:

\[
U_1(x_1, y_1) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U_0(x_0, y_0) \frac{-z}{4\pi r^2} e^{-ikr} \, dx_0 \, dy_0 \tag{3}
\]

\( U_0 \) is the field in the plane of the mask, \( z \) is the distance from the mask to the plane where \( U_1 \) is being calculated, and \( r \) is the distance from the point \((x_0, y_0)\) to the point \((x_1, y_1)\). We need to analyze the \( x \) and \( y \) terms of the exponent in order to extract the Fourier transform properties, so we take the binomial expansion \((1 + z)^n = \sum_{k=0}^{n} \binom{n}{k} x^n \) of the exponent, keeping only the first two terms:

\[
r = \sqrt{\frac{2}{z} + z^2 + (\frac{x_0 - x_1}{z})^2 + (\frac{y_0 - y_1}{z})^2}
= z \left[ 1 + \frac{1}{2} \left( \frac{x_0 - x_1}{z} \right)^2 + \frac{1}{2} \left( \frac{y_0 - y_1}{z} \right)^2 \right]
\tag{4}
\]

Substituting this expression into the exponent, Equation 3 becomes:

\[
U_1(x_1, y_1) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U_0(x_0, y_0) \frac{-z}{4\pi r^2} e^{-ikr}
\]

\[
e^{-\frac{i\pi}{f_1}[(x_0-x_1)^2+(y_0-y_1)^2]} \, dx_0 \, dy_0 \tag{5}
\]

With the \( x \) and \( y \) terms isolated in this manner, we can now analyze the position dependence of the exponent.

Taking the exponential term of Equation 5 and multiplying out the exponent, we get:

\[
e^{-\frac{i\pi}{f_1}[(x_0-x_1)^2+(y_0-y_1)^2]} = e^{-\frac{i\pi}{f_1} [x_0^2 + y_0^2]} e^{-\frac{i\pi}{f_1} [x_1^2 + y_1^2]} e^{-\frac{i\pi}{f_1} [x_0 x_1 + y_0 y_1]}
\]

Notice how we have separated the exponent into three exponential terms. The third is the phase term in a 2-dimensional Fourier transform, so we want to isolate it inside the integral by removing the other two terms. The second term does not depend on the integration variables, so we can safely move it outside the integral. The first term does affect the integral, but we can cancel it with a phase shift in the input plane. A lens placed in front of the mask conveniently results in a phase shift given by:

\[
A(x_0, y_0) = e^{-i k \Delta} e^{\frac{i\pi}{f_1} [x_0^2 + y_0^2]}
\]

where \( f \) is the focal length of the lens. This exactly cancels the first of our exponential terms if \( f = z \). We also assume that \( \frac{z^2}{f^2} \approx \frac{1}{f^2} \), since \( z \) is much greater than the transverse components of \( r \). The resulting field is given by:

\[
U_1(x_1, y_1) = e^{-\frac{i\pi}{f_1} [x_1^2 + y_1^2]} e^{-i k \Delta} \frac{1}{f} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} U_0(x_0, y_0) e^{-\frac{i\pi}{f_1} [x_0 x_1 + y_0 y_1]} \, dx_0 \, dy_0 \tag{6}
\]

So, the focal plane of a lens behind a mask is an exact Fourier transform of the plane containing the mask.

The inverse transform can be performed in the same manner as the initial transform. This is due to the symmetry of the Fourier transform plane: \( U_0(x_0, y_0) = U_0(-x_0, -y_0) \), allowing us to reverse the sign in the exponent. However, the factor of \( e^{-\frac{i\pi}{f_1} [x_1^2 + y_1^2]} \) which we moved outside the integral in the initial transform contains the integration variables of the next transform. Our next lens must cancel both \( e^{-\frac{i\pi}{f_2} [x_1^2 + y_1^2]} \) and \( e^{-\frac{i\pi}{f_2} [x_1^2 + y_1^2]} \). This requires that \( f_2 \) satisfies:

\[
\frac{1}{z_1} + \frac{1}{z_2} = \frac{1}{f_2} \tag{7}
\]

A pointwise multiply after the first lens causes the term outside the integral to be squared: \( e^{-\frac{i\pi}{f_2} [x_1^2 + y_1^2]} \). To achieve an exact inverse Fourier transform, we must therefore satisfy a slightly modified equation:

\[
\frac{2}{z_1} + \frac{1}{z_2} = \frac{1}{f_2} \tag{8}
\]

In general, the \( i^{th} \) lens must be at a distance:

\[
z_i = \frac{1}{f_i} - \frac{n+z}{z_i+1} \tag{9}
\]

where \( n \) is the number of pointwise multiplies done between lenses \( i \) and \( i - 1 \).
C. Pointwise Multiplication

After the Fourier transform, we must perform a pointwise multiply in the Fourier domain. This remains a challenge, and a full treatment is outside the scope of this paper. However, we give a brief explanation of the possible approaches and why we believe this problem is solvable.

The first thing to note is that function values in the mathematical Fourier transform map to field magnitudes in the optical Fourier transform. So, just as we described forming a piecewise function to represent digits in a number (Section II-A), our optical digit encoding will be a function from the spatial plane to field magnitude. Similarly, the mathematical phase factor maps to the optical phase of the field. This means that the result of pointwise multiplication must also encode numerical information in the field magnitude and preserve the correct phase in order for the inverse transform to work.

A product in field magnitudes could be obtained by passing light from one transformed input through an amplitude mask representing the other transformed input. For example, if the amplitude at a specific point on one of the inputs was 30% of the peak amplitude, it would correspond to a point on the mask that lets 30% of the light through. If the corresponding point in the other input was 70% of the peak amplitude, after going through the mask it would be 21% of the peak, effectively representing $3 	imes 7$. The problem with this is that we would have to make a new mask for every pair of inputs. It is feasible to use some photochromic material, exposed to the diffraction pattern of a light source, to create such a mask during computation. However, this process is likely to be slow and would create a bottleneck in computation.

It is tempting to look for a phase mask that will prepare both beams so that they yield a pointwise multiply when superimposed. However, this involves a fair amount of computation that does not arise naturally in optical operations. The equation is as follows:

\begin{align*}
Ae^{-ikt} + Be^{-i(kt+\varphi)} &= AB e^{-ikt} \\
Be^{-i(kt+\varphi)} &= (AB - A) e^{-ikt} \\
e^{-i(kt+\varphi)} &= \frac{A}{B} (B - 1) e^{-ikt} \\
-i(kt + \varphi) &= \ln \left[ \frac{A}{B} (B - 1) e^{-ikt} \right] \\
\varphi &= i \ln \left[ \frac{A}{B} (B - 1) \right]
\end{align*}

If we need an auxiliary computer to calculate this phase shift and generate a corresponding phase mask, it would probably be easier to simply do the pointwise multiply with a detector and electronic computer.

There appears to be a promising avenue of further research in the direction of spatial light modulators or electro-optical modulators. Devices such as these could theoretically handle amplitude and phase modulations necessary for creating the correct output. For now, we make the reasonable assumption that pointwise multiplication is possible and proceed with experimenting via simulation.

III. Simulation

A. Simulation Code

Substantial work has been done in creating simulators for Fourier optics [17], [18], [19]. However, we opted to create our own simulator so as to better target it to our purposes. Since the project requires a modest range of functionality (lenses, masks, detectors, beam splitters, digital pointwise multiplication, beam sources), we implemented a lightweight tool consisting of approximately 1800 lines of C code compiled on a Linux workstation using gcc. It is parallelized using the OpenMP library to distribute the 2D convolution workload over threads. Profiling with gprof demonstrates that the code spends at least 95% of its time computing the Fresnel approximation for lenses. Runtimes vary with the pixel count of the image. Running on 20 threads of a dual socket Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz node, the simulation time ranges from around 25 seconds per lens at 250 × 250 pixels resolution to 80 minutes per lens at 1000 × 1000 pixels resolution.

The simulation is configured using an instruction file listing the devices to be modeled during each step of the execution. An example instruction file is shown in Figure 9. The code can process arbitrary numbers of optical paths which can be split and recombined over the course of the simulation. The instruction file also provides the ability to specify “tap” files to write out intermediate data values. Output consists of two files per data request: one containing a grid of field magnitudes and the other containing a gnuplot script for generating a heatmap-style image of amplitude. The simulation results were validated by comparison with rectangular aperture results documented by Abedin [20].

B. Encoding Digits

The simulation allowed us to experiment with some practical considerations of a physical implementation. One important design choice is how to encode numbers into light patterns. There are several options, the most obvious of which is to set equally sized bands to different amplitudes. This works reasonably well, and the results of several multiplication problems are shown below. Note that the output plane is the mirror image of the answer, so digits must be read from right to left. This is denoted by an asterisk next to the digit representation. It is possible to distinguish different digit values in the bands of the output, but it is rather difficult to tell where each digit...
C. Cropping

In a physical system, we do not have the luxury of operating across the entire (infinite) Fourier plane when pointwise multiply and inverse transform. We need to determine how much of the Fourier plane is necessary to keep and how much we can crop out without too much loss of information. An experiment done by simulation explores the interaction of different parameters (cropping size, beam wavelength, and lens focal length) and their effects on the quality of the recovered image. In this experiment, we transform and inverse transform a perfect checkerboard pattern. The first lens has a focal length \( f \). The second lens, with focal length \( \frac{f}{2} \), is a distance \( f \) away from the first lens. The plane of the second lens is the Fourier plane, so we crop it to some finite (square) area. Finally, the output plane is a distance \( f \) from the second lens. This is the setup that appears in Fig. 1. It satisfies Equation 8 and thus is an exact Fourier transform.

We ran the experiment a number of times with different wavelengths of light and values of \( f \) for the focal length of the first lens, and the results are summarized in Fig. 4 and 5. The grayscale of each image is scaled so that the max value is white, so shades are not necessarily comparable between images. However, peak intensities are typically not much greater than 1, the input light intensity. We have plotted wavelength and focal length against the cropped area \( A \) of the Fourier plane to demonstrate how they affect the “spreading out” of information in the Fourier plane.

Fig. 4 shows that halving the wavelength is equivalent to quadrupling the area of the Fourier plane. We get visually identical images if \( \lambda \sqrt{A} = \text{constant} \). Clearly, we can keep shrinking the size of the Fourier plane if we use higher and higher frequencies of light, but we are limited by what is practical to produce in a physical device. 200nm is a reasonable bound, which is why we stop there. Similarly, Fig. 5 shows that halving the focal length is equivalent to quadrupling \( A \). This suggests that output is visually identical for \( f \lambda \sqrt{A} = \text{constant} \). Notice that rows three and four of Fig. 5 are visually the same as rows one and two of Fig. 4, giving evidence for this hypothesis. Examining Equation 6 sheds light on why this is: \( z \), which we set at \( f \) in this experiment, always appears next to \( \lambda \) (keeping in mind that \( k = \frac{2\pi}{\lambda} \)). It is tempting to suggest that continuing to shrink the focal length arbitrarily will allow us to shrink the size of the Fourier plane, which would be a double win for the size of the device. The top row of Fig. 5 demonstrates that this is not the case. There is a breakdown point after which the image becomes garbled.

So far, all these experiments have been run on a 99 × 99 pixel grid. If we increase the resolution in our breakdown point experiment to 198 × 198 pixels, the image becomes clearer again. We have shown this in Fig. 6. This suggests that the resolution of our detector (assuming we need one for pointwise multiplication) governs how small we can make our device. Detector resolutions of about 1 micron are reasonable,
and future technology may improve that number. However, the existence of some breakdown point is required by the approximation we made back in Equation 6 that 
\[ \frac{1}{\sqrt{A}} \approx \frac{1}{\lambda^2}. \]
When \( z \) approaches the scale of the transverse directions in the Fourier plane, this approximation no longer works and we can no longer achieve an exact Fourier transform. Even if we could optimize our device to handle arbitrarily short focal lengths, we would still hit a barrier when \( f \approx \sqrt{A} \). There is much more work that can be done to completely characterize the optimal combination of parameters. However, for this paper it is enough to be familiar with how the parameters interact and the fact that image quality breaks down at some point.

**IV. MODULAR MULTIPLICATION**

**A. Montgomery multiplication**

Now we transition to consider a specific application of optical convolution. Modular multiplication involves solving problems of the form \( c = a \cdot b \mod m \) for integer values of \( a, b \) and \( m \). Typically, the value of each of these will be of similar magnitude. What makes solving this problem difficult on modern computers is the fact that the modulo, or remainder, operation requires a division which is an expensive operation. However, for cases in which the modulus \( m \) is odd, Montgomery has proposed an alternate formulation of the problem that replaces division by \( m \) with division by an alternate base \( r > m \) that is chosen to be a power of two. For binary arithmetic, this reduces the costly division problem to one of inexpensive masks and shifts [21], [22].

Choose a value \( r = 2^k \) to be the next power of two greater than \( m \) and compute \( M = -m^{-1} \mod r \). Montgomery observed that when the values of \( a \) and \( b \) are transformed to new values \( \tilde{a} = a \cdot r \mod m \) and \( \tilde{b} = b \cdot r \mod m \) then we can calculate the corresponding value of \( \tilde{c} \) as follows:

\[
\tilde{c} = \frac{\tilde{a} \tilde{b} + (\tilde{a} \tilde{b} M \mod r) m}{r} \tag{10}
\]

We observe that for an arbitrary integer \( x \), the value of \( x \mod r \) is the lower \( k \) bits of the binary representation of \( x \) while division by \( r \) is the upper \( |x| - r \) bits. Note that to recover the solution \( c \), one needs to convert the result back out of the Montgomery domain which is accomplished by setting \( c = \tilde{c} \cdot R \mod m \) where \( R = r^{-1} \mod m \).

**B. Montgomery example**

To illustrate Montgomery multiplication, we compute \( a \cdot b \mod m \) using \( a = 28510, b = 38672 \) and \( m = 36057 \). Seeing that \( a, b \) and \( m \) are all 16-bit values, choose \( r = 2^{17} = 131072 \) and pre-compute \( M \) and \( R \).

\[ M = -36057^{-1} \mod 131072 = 52375 \]
\[ R = 131072^{-1} \mod 36057 = 14408 \]
Next, convert \(a\) and \(b\) into the Montgomery basis.

\[
\bar{a} = 28510 \cdot 131072 \mod 36057 = 23411 \\
\bar{b} = 38672 \cdot 131072 \mod 36057 = 31495
\]

Now, the steps of the Montgomery multiplication proceed as shown below.

\[
k_1 = \bar{a}\bar{b} \\
    = 23411 \cdot 31495 = 737329445 \\
k_2 = k_1 \cdot M \\
    = 737329445 \cdot 52375 = 38617629681875 \\
k_3 = k_2 \mod r \\
    = \text{AND}(38617629681875, 131071) = 92371 \\
k_4 = k_3 \cdot m \\
    = 92371 \cdot 36057 = 3330621147 \\
k_5 = k_1 + k_4 \\
    = 737329445 + 3330621147 = 4067950592 \\
\check{c} = k_5 / r \\
    = \text{SHIFT}(4067950592, 17) = 31036
\]

Finally, convert the result back out of the Montgomery domain.

\[
c = \check{c} \cdot R \mod m = 31036 \cdot 14408 \mod 36057 = 23831
\]

We see that this is the correct solution to our initial problem: \(28510 \cdot 38672 \mod 36057 = 23831\). Looking back at the calculation above, there are a few important simplifications which will make the problem easier to implement using optics. They will be discussed in the next section.

C. Distinguishing high and low order bits

A closer inspection of Equation 10 reveals that the \(\mod r\) in the numerator will eliminate the high bits of \(\bar{a}\bar{b}M\), so they do not need to be calculated. In addition, as mentioned by Warren [22], computation of the lower \(k\) bits of the numerator is not required. Not only will they be truncated in the division by \(r\), but it can be shown that their sum is always equal to \(r\). We demonstrate this by calculating the sum of the low-order bits of each term in the numerator, which means taking them \(\mod r\).

\[
\bar{a}\bar{b} \mod r + ((\bar{a}\bar{b}M \mod r) \cdot m) \mod r \\
    = \bar{a}\bar{b} \mod r + (\bar{a}\bar{b}M) \mod r \\
    = \bar{a}\bar{b} \mod r + (\bar{a}\bar{b}(-m^{-1} \mod r) \cdot m) \mod r \\
    = \bar{a}\bar{b} \mod r + (\bar{a}\bar{b}(-m^{-1}) \cdot m) \mod r \\
    = \bar{a}\bar{b} \mod r + (-\bar{a}\bar{b}) \mod r \\
    = \bar{a}\bar{b} \mod r + r - (\bar{a}\bar{b} \mod r) \\
    = r
\]

This results in a carry bit, which we can confidently add on to the high bits without computing the low bits. The pathological case is, of course, when \(\bar{a}\bar{b} = 0\), in which case there will be no carry bit. This can occur, so we need to check whether \(\bar{a}\bar{b} \equiv 0 \mod r\). However, we proceed in taking these observations into account, and the Montgomery multiplication example above now proceeds as follows.

\[
k_1 = \bar{a}\bar{b} \\
    = 23411 \cdot 31495 = 737329445 \\
k_{1_{\text{hi}}} = \text{AND}(737329445, 131071) = 49455 \\
k_{1_{\text{lo}}} = \text{RSHIFT}(737329445, 17) = 5625 \\
k_2 = k_{1_{\text{lo}}} \cdot M \\
    = 49455 \cdot 52375 = 2589681875 \\
k_3 = k_2 \mod r \\
    = \text{AND}(2589681875, 131071) = 92371 \\
k_4 = k_3 \cdot m \\
    = 92371 \cdot 36057 = 3330621147 \\
k_{4_{\text{hi}}} = \text{SHIFT}(3330621147, 17) = 25410 \\
\check{c} = k_{1_{\text{lo}}} + k_{4_{\text{hi}}} + 1 \\
    = 5625 + 25410 + 1 = 31036
\]

D. Approximate implementation

The simplifications of the previous section will form the basis of an optical approach to computing the modular multiplication. This is because limiting the number of bits carried from one operation to the next reduces the optical resolution required to accurately compute a solution, thus enabling computation of larger numbers of digits. The intuition behind this is that the frequency of the Fourier transform will scale with the number of digits in the optically encoded representation of each number, and the resolution required to accurately evaluate the pointwise multiplication will be bounded by the Nyquist frequency and the size of the image in the transform plane.

Note that there are some important distinctions between Montgomery in its digital implementation and the hybrid optical approach we present below. The most obvious difference is the fact that “multiplications” in optics are not actually multiplications but convolutions, as illustrated in Figure 7. Convolutions perform no carries from one digit to the next, so the value of digits can grow beyond the magnitude of the base. In addition, the lack of a leading carry bit means that convolution requires one fewer digit in its representation than multiplication. The consequence is that we now need a number of overlap digits when we compute the high bits of the numerator, to take care of unresolved carries. Placeing a mathematical upper bound on this overlap is a complicated calculation. For the purposes of the example in this paper, we performed experiments to determine by trial-and-error that six digits is sufficient to prevent carry errors in our example problem.

In terms of the optical computation, this approach trades off resolution for increased dynamic range. It must also contend with the fact that shifting and masking does not translate
Fig. 7. Comparing multiplication on the left and convolution on the right in terms of the high and low \( k \)-digits of the result. When masking high order bits of the convolution operation, additional bits are necessary to capture the unpropagated carries from the low bits. Futher, the middle bit is counted as both a high and low bit, which will result in the shift of one additional digit when computing our approximate \( \bar{c} \) below.

directly to the results of convolution arithmetic. For clarity, values are written as a vector of their binary digits.

\[
\begin{align*}
\vec{a} &= (1,0,1,1,0,1,1,1,0,1,1,1,1,1,1,0,0,1,1) \\
\vec{b} &= (1,1,1,1,0,1,0,0,0,0,0,1,1,1,1) \\
m &= (1,0,0,0,1,1,0,0,1,1,0,1,1,0,0,1) \\
M &= (1,1,0,0,1,1,0,0,1,0,0,1,0,1,1,1)
\end{align*}
\]

Now we reproduce the steps from the preceding section as follows, using six overlap digits. The notation \( x \otimes y \) denotes the digit convolution of \( x \) and \( y \), and \( x \oplus y \) denotes the digit sum.

\[
\begin{align*}
k_1 &= \vec{a} \otimes \vec{b} \\
&= (1,1,2,3,2,4,4,3,5,4,4,5,4,4,6,5,3,3, \\
&\qquad 4,3,2,3,2,1,1,2,2,1) \\
&= 737329445 \\
k_{1_h} &= (1,1,2,3,2,4,4,3,5,4,4,5,4,4,6,5,6,5,3,3) \\
&= 720045 \\
k_{1_0} &= (4,4,6,6,5,3,3,4,3,2,3,2,1,1,2,2,1) \\
&= 57373 \\
k_2 &= k_{1_h} \odot M \\
&= (4,8,10,12,15,16,16,19,22,17,17,22, \\
&\quad 19,20,23,22,25,25,24,20,16,15,13, \\
&\quad 11,9,8,6,5,5,3,1) \\
&= 30049265875 \\
k_3 &= k_2 \mod r \\
&= (32,28,25,24,20,16,15,13,11,9,8,6,5, \\
&\quad 5,5,3,1) \\
&= 3762387 \\
k_4 &= k_3 \otimes m \\
&= (32,28,25,24,52,76,68,62,87,105,92, \\
&\quad 115,133,114,102,121,100,86,79,66,64, \\
&\quad 51,43,37,30,23,19,14,9,6,5,3,1) \\
&= 135660388059 \\
k_{4_h} &= (32,28,25,24,52,76,68,62,87,105,92,115, \\
&\quad 133,114,102,121,100,86,79,66,51,43)
\end{align*}
\]

\[
k_{5_4} = k_{4_h} \oplus k_{4_0} \oplus 1 \\
= (32,28,25,25,53,78,71,64,91,109,95,120, \\
&\quad 137,118,107,126,104,92,85,71,54,46) \\
&= 133200926 \\
\bar{c} &= \left\lfloor k_{5_4} / 2^7 \right\rfloor \\
&= \left[ \begin{array}{cccccccccccc}
1 & 7 & 25 & 25 & 53 & 39 & 71 & 1 \\
4 & 32 & 128 & 128 & 128 & 64 & 128 & 2 \\
91 & 109 & 95 & 15 & 137 & 59 & 107 & 63 & \\
128 & 128 & 128 & 16 & 128 & 64 & 128 & 64 & \\
13 & 23 & 85 & 71 & 27 & 23 & 16 & 32 & 128 & 128 & 64 & 64 \\
& & & & & & & & & & & & &
\end{array} \right]
\]

\[
\bar{c} = 1040632
\]

This is a different value of \( \bar{c} \) than obtained in the previous example. However, converting the result back out of the Montgomery domain with

\[
c = \bar{c} \cdot R \mod m = 1040632 \cdot 14408 \mod 36057 = 23831
\]

demonstrates that we have recovered the correct value of \( c \). The convolutional approach is an approximate modular multiplication in the sense that it produces a result that will typically be an arbitrary multiple of the modulus. Notice that in the last step of computing \( \bar{c} \) the last seven digits have been removed, despite the fact that the overlap was only six digits. That is because of the one fewer digits resulting from convolution compared to multiplication. The most significant \( k \) digits of the result include one digits from the \( k \) least significant digits which must also be removed.

\[E. \text{ An optical implementation}\]

We now present simulation results for an all-optical computation of \( 28510 \cdot 38672 \mod 36057 = 23831 \) using the algorithmic approach described above. Shifts and masks are accomplished through optical masking and alignment. Addition is performed using a beam-splitter in reverse to combine values. We incorporate multiplications by one to make certain that each value being added has passed through the same number of multiplier steps, a necessary step to keep the phase information consistent across the summands. For purposes of simplification, we are assuming that \( \vec{a}, \vec{b}, R \) and \( M \) are pre-computed and provided as inputs to the simulated optical system. Further, we assume that \( \bar{c} \) will be converted back out of the Montgomery domain as part of a separate calculation.

In general, the most significant bit (MSB) of a number is will be in the upper right hand corner of the image, and the least significant bit (LSB) is in the lower left hand corner. However, after each multiplication the resulting image is reversed. As before, the reversal is denoted by an asterisk next to a value in the image. The optical configuration to implement the modular multiplication is illustrated in Figure 8 above, and the instruction file to run the simulation is shown in Figure 9. In general, the pointwise multiplies are assumed
to be performed using a spatial light modulator (SLM) which creates an image mask of the Fourier transform of one of the values to be multiplied. As discussed above, this step is generally considered to be slow, but it is important to point out that only one of the SLMs, corresponding to the Fourier transform of $\hat{b}$ actually needs to be updated in real time. The remaining transforms can be precomputed, and thus the setting of the SLM once initialized will remain fixed.

The results of the computation are illustrated in Figure 10 that follows. Digit values printed below each image are the result of integrating the light intensity over the area of each digit and normalizing. The results after step #13 are then rounded to the nearest integer. The maximum absolute error before rounding is 0.3419 and the RMS error is 0.1442, meaning that the analog result matches the correct integer solution. The main sources of error were found to be resolution and dynamic range. Specifically, the resolution in the Fourier transform plane needs to be sufficient for the pixel density to be at least 1442, meaning

$$3419 \times 1442 = 38672 \mod 36057 = 23831$$

and the RMS error is 0.1442, meaning

$$\sum (x_i - \hat{x}_i)^2 / n$$

which must be performed in real time with $\mu m$. The dynamic range was assumed a minimum pixel pitch of 1 mm imaging area at each pointwise multiply and thus the setting of the SLM once initialized will remain fixed.

The results of the computation are illustrated in Figure 10 that follows. Digit values printed below each image are the result of integrating the light intensity over the area of each digit and normalizing. The results after step #13 are then rounded to the nearest integer. The maximum absolute error before rounding is 0.3419 and the RMS error is 0.1442, meaning that the analog result matches the correct integer solution. The main sources of error were found to be resolution and dynamic range. Specifically, the resolution in the Fourier transform plane needs to be sufficient for the pixel density to be at least twice the Nyquist frequency of the transform. This simulation used a 1 mm imaging area at each pointwise multiply and assumed a minimum pixel pitch of 1 mm × 1 mm. The dynamic range was assumed to be at least 12 stops (i.e., bits) or approximately 36 dB. The values $\mathcal{F}(M^*)$, $\mathcal{F}(m)$, $\mathcal{F}(1^*)$ and $\mathcal{F}(1)$ are precomputed inputs to the pointwise multiply unit. The performance bottleneck for this device is the pointwise multiply by $\mathcal{F}(\hat{b})$ which must be performed in real time with each modular multiplication.
F. Comparison with digital technology

Here we will make estimates of the efficiency of a 16-bit optical modular multiplier based on commercially available free-space optical technology and compare those numbers to some published results both in electronics and nano-photonics. Based on our simulation results, approximately 250,000 pixels are required to resolve the convolution with sufficient accuracy in the Fourier transform plane. For the device as simulated, this would correspond to a pixel pitch of around 2 μm, which appears reasonable using commercial technology. We estimate that as few as 1 photon per detector site may be necessary for room temperature detection based on results published by [23]. Photon energy is approximately $10^{-18}$ Joules for 200 nm wavelength light. The estimated photon loss end-to-end across the optical multiplier is about a factor of 3000 based on simulation. Thus, we estimate the overall efficiency of the device as simulated would be $250000 \times 10 \times 10^{-18} \times 3000 = 7.5 \times 10^{-10} \approx 1\text{nJ}$ per operation.

This number is high when compared to CMOS results where our three 16-bit multiplies along with associated adds, shifts and masks can probably be accomplished for as little as 10 pJ [24]. However, recent work in energy efficient nano-photonics suggest that it may be possible to operate an all-optical design at as low as 10 fJ/bit [25]. Currently, performance is limited by the spatial light modulators (SLMs) required to perform the multiply. A brief survey of several commercially available liquid crystal SLMs suggests that they are typically limited to kilohertz switching speeds. However, the introduction of active metamaterials and an electro-optical modulator (EOM) in place of the SLM may make it possible, in theory at least, to shrink the size of the components and operate them at gigahertz or even terahertz speeds at room temperature [26]. That said, it remains beyond the scope of this paper to assess the cost and technical feasibility of nano-photonics-based solutions.

V. Conclusions

Optical techniques appear to be very promising in the future of high-performance computing. The most significant area of future work will be to explore ways of performing a pointwise multiply optically. If that can be done efficiently, this method may provide huge speed-ups over current technology. Even if the pointwise multiply must be done with an auxiliary computer, the computation will be $O(n)$ instead of $O(n^2)$ thanks to the nearly instantaneous Fourier transform achieved optically. If this path proves to be optimal, more work would need to be done to decrease the load for the electronic computer. A lookup table approach is possible, as are methods of keeping field values at whole numbers. Other future work might include finding the limits of how much information can be packed into a given area, how small the Fourier transform plane can be cropped, and how many operations can be done in series before the noise threshold exceeds the required computational accuracy.

Based on our simulation results it also appears feasible for an architecture similar to the one analyzed to accurately perform all-optical modular multiplication with greater precision than has been traditionally associated with analog computation [2]. Given our estimates of parameters such as size, resolution and dynamic range, such a device could be manufacturable from commercially available components. If these simulation results bear out in the lab then the ability to accurately perform 16-bit arithmetic could be a significant step forward for optical computation. A further contribution of this work is the demonstration of an approximate algorithm, implementable in optics, that is capable of correctly computing discrete modular multiplication. Future work should include further refinement to the simulation, optimization of the device configurations, particularly with regards to the lenses, and then fabrication and characterization of an actual device.

REFERENCES

[1] Alexander A. Sawchuk and Timothy C. Strand, “Digital Optical Computing,” Proceedings of the IEEE, 72, 7 (1984).
[2] Demetri Psaltis and Ravindra A. Athale, “High accuracy computation with linear analog optical systems: a critical study,” Applied Optics, 25, 18 (1986).
[3] Dror G. Feitelson, Optical Computing: a Survey for Computer Scientists, MIT Press, Cambridge, MA, USA. (1988).
[4] Joe Touch and Abdel-Hameed Badawy and Volker J. Sorger, “Optical computing,” Nanophotonics, 6, 3 (2017), Retrieved 2 Nov. 2017, from doi:10.1515/nanoph-2016-0185.
[5] “Samsung Starts Industry’s First Mass Production of System-on-Chip with 10-Nanometer FinFET Technology.” N.p, n.d. Web, (2017), https://news.samsung.com/global/samsung-starts-industries-first-mass-production-of-system-on-chip-with-10-nanometer-finset-technology.
[6] J. Hasler and S. Shah, “Reconfigurable Analog PDE computation for Baseband and RF Computation,” GOMAC, March, 2017.
[7] A. Femius Koenderink and Andrea Alù and Albert Polman, “Nanophotonics: Shrinking light-based technology,” Science, 348, 6234 (2015).
[8] “Optalysys White Paper,” N.p, n.d. Web, 23 February 2015, https://www.scribd.com/document/308816109/Optalysys-White-Paper.
[9] IEEE Rebooting Computing, http://rebootingcomputing.ieee.org/.
[10] A. Schönhage and V. Strassen, “ Schnelle Multiplication großer Zahlen,” Computing, 7, 3 (1971).
[11] M. Furer, “Faster Integer Multiplication,” Proceedings of the 39th Annual ACM Symposium on Theory of Computing, June, 2007.
[12] Vinod Chandran and Thomas F. Krile and John F. Walkup, “Optical techniques for real-time binary multiplication,” Applied Optics, 25, 24 (1986).
[13] Peter Bryanston-Cross, “The lens as a Fourier Transform System,” N.p, n.d. Web. Optical Engineering Laboratory, University of Warwick, 26 November 2002, http://www.eng.warwick.ac.uk/espbc/courses/undergrad/lec8/fourier_lens.htm.
[14] Nathan T. Shaked, Tal Tabib, Gil Simon and Joseph Rosen, “Optical binary-matrix synthesis for solving bounded NP-complete combinatorial problems,” Optical Engineering 46, 10 (2007).
[15] Joseph W. Goodman, Introduction to Fourier Optics, The McGraw-Hill Companies, Inc, New York, NY, USA, (1996).
[16] M.K. Kim, Digital Holographic Microscopy: Principles, Techniques, and Applications, http://www.springer.com/978-1-4419-7792-2, XVI (2011).
[17] Ahmed Louri and Jongwoha Na, “Modeling and simulation methodology for digital optical computing systems,” Applied Optics, 33, 5 (1994).
[18] Seymour Trester, “Computer-Simulated Fresnel Diffraction Using the Fourier Transform,” IEEE Computing in Science and Engineering, 1, 5 (1999).
[19] D. Rafferty and U.H. Wagner and C.Rau and P.Chang and S.Alcock and R. Dockree and I.R. Robinson, N.p, n.d. Web, “Development of a computer model to simulate wavefront propagation,” (2010), http://www.ucl.ac.uk/ucapikr/project.htm.

[20] K.M. Abedin, M.R. Islam and A.F.M.Y Haider. “Computer simulation of Fresnel diffraction from rectangular apertures and obstacles using the Fresnel integrals approach,” Optics and Laser Technology, 39 (2007).

[21] Peter L. Montgomery, “Modular multiplication without trial division,” Mathematics of Computation, 44 (1985).

[22] Henry Warren, “Montgomery Multiplication,” N.p, n.d. Web, http://www.hackersdelight.org/MontgomeryMultiplication.pdf.

[23] Jiaju Ma, Saleh Masoodian, Dakota Starkey, and Eric Fossum, “Photon-number-resolving megapixel image sensor at room temperature without avalanche gain,” Optica, 4 (12), 2017.

[24] S. P. Pohokar, R. S. Sisal, K. M. Gaikwad, M. M. Patil, and R. Borse, “Design and implementation of 16x16 multiplier using Vedic mathematics,” International Conference on Industrial Instrumentation and Control (ICIC), 28-30 May 2015 pp. 1174-1177

[25] Zheng Wang, Zhoufeng Ying, Shounak Dhar, Zheng Zhao, David Pan and Ray T. Chen, “Nanophotonic Devices for Power-efficient Computing and Optical Interconnects,” IEEE Photonics Society Summer Topical Meeting Series (SUM) San Juan, 2017 pp. 7-8.

[26] Saroj Rout and Sameer Sonkusale, “A low-voltage high-speed terahertz spatial light modulator using active metamaterial”, APL Photonics 1, 086102 (2016).
Fig. 10. Images depict the digit encoding pattern after enumerated steps of the optical modular multiplication illustrated in Figure 8. Vector values are the integrated optical intensity for each digit. Those values are normalized and rounded to the nearest integer. An asterisk after a vector denotes values that will appear in reverse order in its corresponding image.