# Tachyon: A Programmable Optoelectronic Hardware Accelerator for Ultrafast Tensor Arithmetic

**Sathvik Redrouthu, Jagadeepram Maddipatla, Pranav Vadde, and Anant Khandelwal**

**Abstract** We discuss the design and implementation of Tachyon, a high-speed, scalable optical hardware accelerator. In addition, we go over programming the accelerator with an instruction set architecture (ISA) capable of efficiently communicating with on-chip optical components to perform ultrafast $\mathcal{O}(1)$ matrix-vector multiplication (MVM). We first introduce the optoelectronic accelerator's computing scheme. We then go over our pipeline for programming the accelerator and introduce our optical ISA. Our results show that the accelerator's optical core is scalable and can perform an MVM in a single pass with a performance per watt (PPW) clocked 3 orders of magnitude higher than Google's Tensor Processing Unit. The core is rated at 7 TFLOPS and 245 TFLOPS/W for a $4 \times 4$ matrix tile, but scales to 478 TFLOPS and 261 TFLOPS/W for a $256 \times 256$.

**Keywords** Data analytics · High-speed computing · Machine learning · Matrix-vector multiplication · Neural networks · Optical computing

## 1 Introduction

Over the past decade, there has been a dramatic increase in the parameter count of neural networks, driven by advances in machine learning algorithms, hardware, and data availability. This increase has enabled significant improvements in performance on a wide range of tasks, from image classification to natural language processing. In

S. Redrouthu (✉) · J. Maddipatla (✉) · P. Vadde · A. Khandelwal
Procyon Corp, Clearwater, FL, USA
e-mail: sathvikr@procyoncorp.io

J. Maddipatla
e-mail: jag.maddipatla@procyoncorp.io

P. Vadde
e-mail: pranav.vadde@procyoncorp.io

A. Khandelwal
e-mail: pranav.vadde@procyoncorp.io

2012, the state-of-the-art image classification model, AlexNet, had only 61 million parameters . By 2015, the VGG-19 model had 143 million parameters, and by 2016, the ResNet-152 model had 60 million parameters. In 2018, the DenseNet-264 model had 36.4 million parameters, while the EfficientNet-B7 model, released in 2019, had 66 million parameters [1]. The parameter counts of natural language models have also increased significantly. In 2015, the state-of-the-art language model had only 5 million parameters. By 2018, the OpenAI GPT-2 model had 1.5 billion parameters, and by 2020, the GPT-3 model had 175 billion parameters [2].

The increase in parameter count has enabled significant improvements in performance on a range of benchmarks. For example, in the ImageNet Large Scale Visual Recognition Challenge, the top-1 error rate of the winning models decreased from 26.2% in 2012 to 3.5% in 2021 . Similarly, in the GLUE benchmark for natural language understanding, the state-of-the-art performance has increased from 81.6% in 2018 to 90.4% in 2021 [2].

However, the increase in parameter count has also raised concerns about the computational cost and environmental impact of training and running these models. For example, training the GPT-3 model on a single accelerator can consume up to 1.2 GWh of electricity [3]. To address these concerns, in addition to exploring various techniques for reducing the neural network parameter counts while maintaining performance, researchers are building more efficient hardware. The market demand for high performance per watt chips is incredibly high, resulting in large companies to join in developing powerful hardware applicable to even future model iterations, e.g., Cerebras AI lining up to tackle GPT-4's estimated trillion-parameter workload.

Such AI-based applications primarily base their functionality in tensor arithmetic. NVIDIA, a pioneer in high performance computing (HPC), takes advantage of this by building efficient electronic processors labeled Graphics Processing Units (GPUs). These GPUs are able to accelerate artificial intelligence processes by enabling high-speed matrix multiplication to take place through tensor cores or unique applications of the Single Instruction-Stream, Multiple Data-Stream (SIMD) architecture. Google developed a digital Application-Specific Integrated Circuit (ASIC) specialized in AI processing, labeled a Tensor Processing Unit (TPU). The TPU specializes in computing important activation functions and matrix multiplication for NN inference [4]. The Edge TPU is able to perform 4 trillion operations per second using 2 watts of power (2 TOPS/watt) [5]. It is apparent that many significant hardware advancements have been made over the decades.

However, while these advancements have proven to be advantageous with regard to computational efficiency, they are fundamentally limited by the decline of trends such as Moore's Law and Dennard Scaling. Moore's Law asserts that the number of transistors on a digital electronic microchip will approximately double every two years, directly correlating to the increase in computing power [6]. However, it has been predicted by experts that this trend will no longer be viable by 2036 due to an inability to further reduce the size of transistors. For this reason, coupled with the end of Dennard Scaling, their computational ability will stagnate unless matrix processing systems are reinvented to function without the usage of digital electronics.

To account for this problem, the use of analog computing for AI has been investigated; such a platform eliminates the reliance on transistors. Years of research and production have confirmed that while having lower accuracy and higher loss, the analog approach is generally more efficient for operations such as matrix multiplication [7, 8]. Mythic AI's M1076 analog matrix processor achieves the AI compute performance of a digital electronic desktop GPU while consuming 10% of the energy, the typical power consumption for running complex neural networks being approximately 3–4 W [9]. Digital-analog electronic hybrid processors have also been investigated in an attempt to maximize speed and accuracy together [8].

However, most electronics-based systems are subject to interference and temperature problems. Even recently created analog architectures, such as Mythic's AMPs, possess limitations from electromagnetic crosstalk and Joule heating. In addition, the computing power required for functional operation scales dramatically with NN feature size. Such problems are sidestepped through silicon photonics, with matrix-vector multiplication (MVM) able to be done cleanly and passively. While such circuits are themselves inherently analog, they are significantly faster than their analog electronic equivalents. In this paper, we explore a hybrid silicon photonic-digital electronic hardware accelerator. We first go over the accelerator design and implementation. We then discuss the programming interface used to run tensor arithmetic on the accelerator. We finish by discussing results and implications.

## 2 Accelerator Design

### *2.1 Overview*

The Tachyon accelerator contains an optical matrix processing unit (MPU), an engine where MVM sequences are executed at high speed to power important linear algebra-based tasks such as NN inference. When an operation is run within an application, relevant aspects of it are sent to the MPU (Fig. 1).

Tachyon's instruction set architecture (ISA) is given in Table 1. When instructed to a) write a matrix or vector to the MPU, or b) compute an MVM, the host platform sends instructions to a master microcontroller within the Tachyon accelerator via serial communications. The master relays corresponding digital signals to digital-to-analog converters (DACs); corresponding analog signals are then used to drive a grid of laser beams that send analog optical signals within the MPU. These signals are then manipulated to perform an MVM as outlined in Sect. 2.2. Output optical signals are read through photodetectors and converted into corresponding analog current streams, which are then converted to digital signals through analog-to-digital converters (ADCs). The bitstreams are sent back to the master and results are reported back to the host through serial.

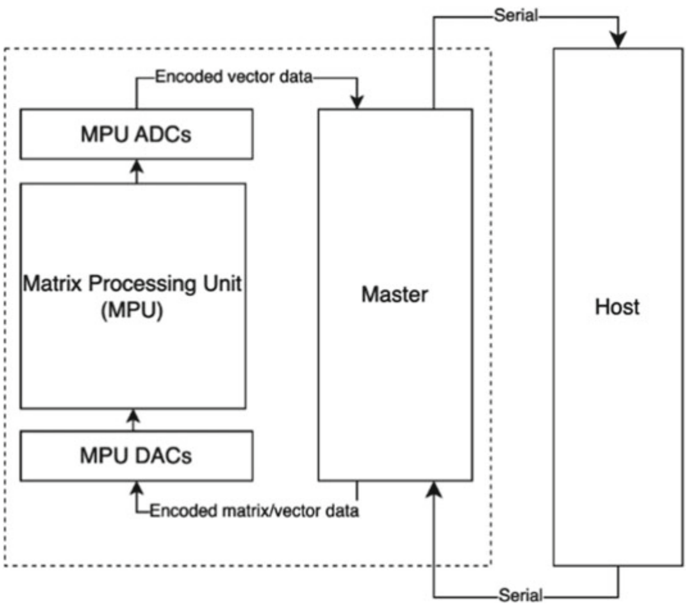


**Fig. 1** System on-chip diagram of the Tachyon hardware accelerator

**Table 1** Instruction set

| Name | Location |
|---|---|
| **mwrite** m | MPU |
| **vsend** v | MPU |
| **mvmul** a b | MPU |

## *2.2 Matrix Processing Unit*

The MPU, shown in Fig. 3, performs a matrix-vector multiplication (MVM) using analog optical signals. A single laser toggles on and the output beam is split into $n$ channels through an even beam splitter network. A modulator is integrated into each splitter arm to alter incoming light discretely, inherently allowing unique elements in the input vector $\vec{\mathbf{x}}$ to the MPU. The grid of laser light is then altered by components within the MPU that correspond to a matrix transformation $\vec{\mathbf{x}} \mapsto \mathbf{M}\vec{\mathbf{x}}$, where $\mathbf{M}$ is determined by the specific component settings. The results are sent as optical analog signals to a grid of photodetectors, each of which reports back an analog current stream. This concept is shown in Fig. 2.

To configure the components within the MPU to represent any $\mathbf{M} \in \mathbb{R}^{m\times n}$, we take advantage of the fact that any such matrix $\mathbf{M}$ can be represented as $\mathbf{U}\Sigma\mathbf{V}^*$ through the singular value decomposition (SVD), where $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V}^* \in \mathbb{R}^{n\times n}$ are unitary, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix. The MPU performs $\vec{\mathbf{x}} \mapsto \mathbf{M}\vec{\mathbf{x}}$ by splitting $\mathbf{M}$ into these 3 smaller matrix transformations, as shown in Fig. 4. Light passes through each input port; the corresponding Stokes vector is transformed by $\mathbf{V}^*$, $\Sigma$, and $\mathbf{U}$,

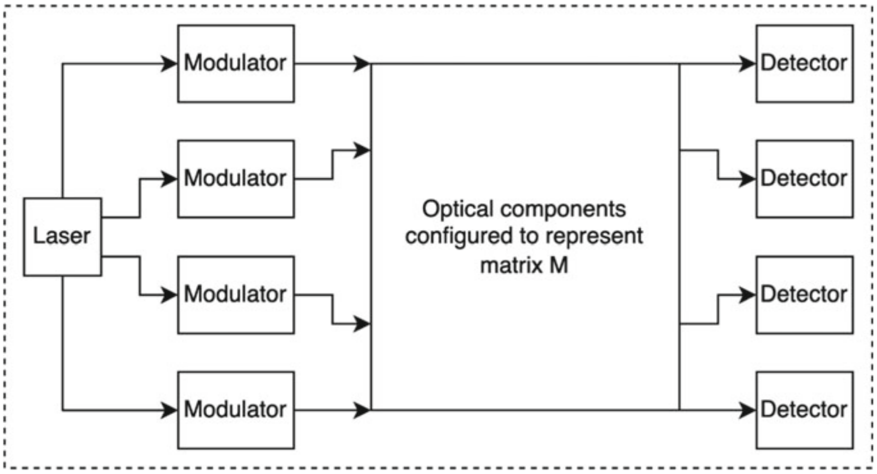


**Fig. 2** $\vec{\mathbf{x}} \mapsto \mathbf{M}\vec{\mathbf{x}}$ transformation pipeline. On-chip driving laser split across beam splitter network, tuned according to input vector $\vec{\mathbf{v}}$, sent to optical components implementing **M**, and read out via on-chip photodetectors

respectively. This essentially means that rather than explicitly manipulating each entry in $\mathbf{U}$, $\Sigma$, and $\mathbf{V}^*$, the MPU manipulates a set of phase angles parameterizing each matrix. Doing so allows the MVM computation to be done on hardware in $\mathcal{O}(1)$, but extra work is required for setting up calculations (writing matrices and vectors).

### 2.2.1 Optical Unitary Matrix Transform

The MPU uses silicon photonic sub-circuits to implement the unitary matrix transforms given by $\mathbf{U}$ and $\mathbf{V}^*$. A lattice of Mach-Zehnder Modulators (MZMs) are laid out in the pattern proposed by Clements et al. [10]. Each MZM is 50:50 directional-coupler based, with the following $\mathbb{R}^{2\times 2}$ transfer matrix:

$$\mathbf{U}^{(2)} = \frac{1}{2}\begin{bmatrix} e^{j\alpha^{(2)}}(e^{j\theta^{(2)}} - 1) & je^{j\alpha^{(2)}}(1 + e^{j\theta^{(2)}}) \\ je^{j\beta^{(2)}}(1 + e^{j\theta^{(2)}}) & e^{j\beta^{(2)}}(1 - e^{j\theta^{(2)}}) \end{bmatrix} \in \mathbb{R}^{2\times 2}. \tag{1}$$

The phase angles $\theta, \alpha, \beta$ are shifts induced by modulators incorporated in rib waveguides within the MZM. The full MZM lattice then represents the unitary matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{n\times n}$ as

$$\mathbf{U}^{(n)} = \prod_{i=1}^{n}\prod_{j=1}^{n-i} \mathbf{T}_{n-j,n-j}(\theta_{i,j}^{(n)}, \alpha_{i,j}^{(n)}, \beta_{i,j}^{(n)}) \in \mathbb{R}^{n\times n}, \tag{2}$$

where $\mathbf{T}_{i,j} \in \mathbb{R}^{n\times n}$ contains the top left of a unitary matrix $\mathbf{U}^{(2)} \in \mathbb{R}^{2\times 2}$ at location $(i, j)$, a diagonal of 1s (excluding $\mathbf{U}^{(2)}$), and the remaining entries as 0s [10].

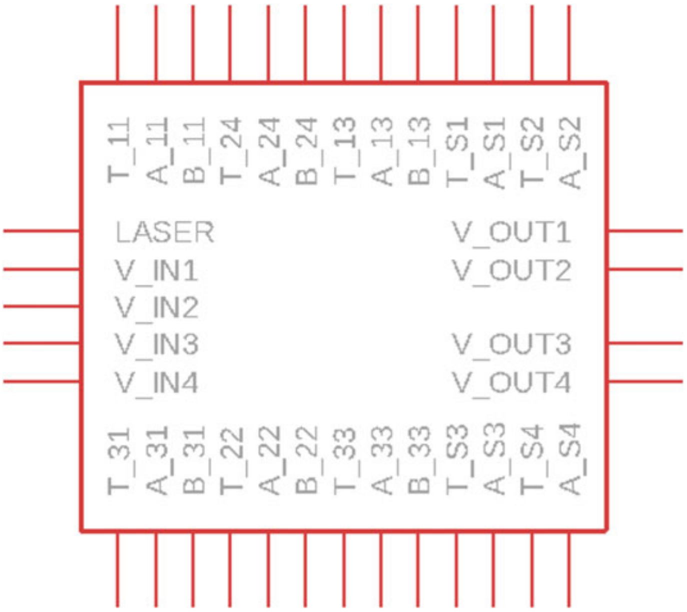


**Fig. 3** Example MPU accepting an $\mathbb{R}^4$ vector as input. One optical input from the LASER port is tuned by analog voltage signals from 30 other ports

**Fig. 4** Optical $\mathbf{V}^*$, $\Sigma$, and $\mathbf{U}$ cores making up the MPU, respectively

### 2.2.2 Optical Diagonal Matrix Transform

Left-multiplying a vector $\vec{\mathbf{v}} \in \mathbb{R}^n$ by a diagonal matrix $\Sigma$ effectively scales each entry in $\vec{\mathbf{v}}$ by a corresponding entry in $\Sigma$. We therefore use $n$ single ports of dual-modulator MZMs arranged in a column, where each MZM scales the input electric field:

$$E_{\text{out}} = \frac{E_{\text{in}}}{2} \mathrm{e}^{j\omega}(\mathrm{e}^{j\phi} - 1). \tag{3}$$

## 2.3 Supporting Electronics

Voltaire is the interconnect enabling the optical Tachyon chip to communicate with external systems. The MPU operates using analog voltage signals, which are used to produce and manipulate laser light. We use a series of off-the-shelf microcontrollers and other components such as digital-to-analog converters (DACs) in a master-slave system to organize, distribute, and convert digital signals from the host hardware. The host interfaces with these components through serial communications protocols. After the MPU completes its operation, it outputs an analog differential current signal which is converted to a digital signal using analog-to-digital converters (ADCs). These digital signals are sent back to the host hardware, and the next operation is ready to be carried out.

### 2.3.1 Microcontroller Master-Slave System

Information from the host is communicated to a Teensy 4.1 development board through a hardware serial communications protocol [11]. The Teensy is connected to an additional pair of microcontrollers: Texas Instrument's Arm Cortex based RM57L843 [12]. The devices are set up in a master-slave system using the Serial Peripheral Interface (SPI) protocol with the Teensy as the master and the 2 RM57L843 microcontrollers as slaves. The vector and matrix data from the host is distributed from the Teensy to each RM57L843. The slaves then distribute the digital signals representing element data to each DAC. The output analog signals from the MPU are routed directly to the slaves which then convert them back to digital signals using onboard ADCs. The resultant element data is communicated back to the host via the Teensy through SPI and hardware serial.

### 2.3.2 Signal Converter System

The digital signals from the host need to be converted to analog signals to be compatible with the optical components in the MPU. The digital signals from the RM57L843 slave microcontrollers are converted to analog signals using Analog Devices' AD9748 DAC [13]. The converter accepts 8 digital signals as input to represent an 8 bit digital representation of the signal. The DAC then outputs a differential current as the analog representation of the 8 bit digital signal. The AD9748 includes an option for scaling the output to fit the MPU's input parameters. This differential current is then converted to an analog voltage signal using Analog Device's AD8047 operational amplifier (op amp) [14]. Figure 5 shows this DAC assembly. This analog voltage is passed to MPU for calculation. There are a total of 36 DACS for every input the MPU requires, including the laser, the initial 4 dimensional vector, and the initial $4 \times 4$ matrix. Figure 6 shows a schematic of the DACs connected to the MPU.

The MPU's photodetectors output a differential current to represent each output vector element. This is converted to a single analog voltage signal and routed to the ADCs on each slave microcontroller. Each slave has 2 ADC modules with up to 12 bits of resolution, with the option of reducing resolution for higher conversion speeds. The slave ADCs convert the 4 output analog signals into digital form to be sent back to the host.

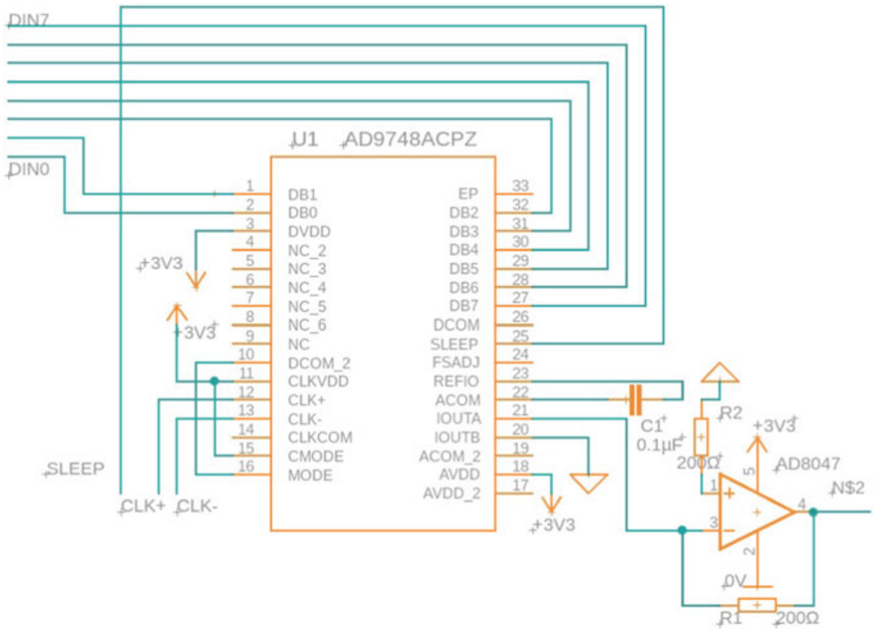


**Fig. 5** AD9748 current-driven DAC extended to output single-ended voltage with an AD8047 operational amplifier. Both resistors have a 200 resistance, whereas the capacitor is rated at 0.1. Connections for digital signal inputs, analog signal output, power, timing signals, and the power down function are noted

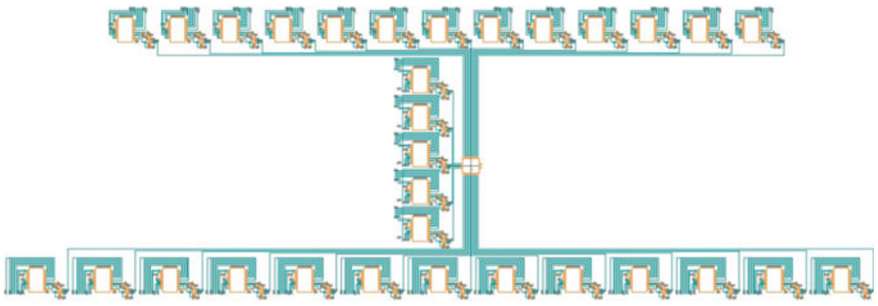


**Fig. 6** Schematic of DACs connected to the MPU

## 3 Accelerator Implementation

Chip fabrication took place at the Maryland NanoCenter. We used the Raith eLINE electron-beam lithography (EBL) machine operated at 100 keV with an 8 nA beam current on a 20 nm process node. The Matrix Processing Unit (MPU) itself is on a diced 605 μm × 410 μm × 0.22 μm silicon wafer.

## 3.1 Matrix Processing Unit

### 3.1.1 Passive Circuitry

We use strip waveguides for passive structures throughout the Matrix Processing Unit, each with a height of 220 nm and a width of 20 nm. The waveguide cladding is composed of Silicon Dioxide ($SiO_2$). The waveguide bend radius is 5, the smallest bend radius practical for transverse electric (TE) polarization. The waveguide mode profile, along with a corresponding wavelength sweep, is shown in Fig. 7. Through the obtained data, the compact model was calculated:

$$n_{\text{eff(undoped)}}(\lambda) = 2.56862 - 0.729403(\lambda - 1.55) + 0.0995556(\lambda - 1.55)^2 + jn_{\text{eff(imag)}}. \quad (4)$$

The last term is included to convert the loss to the imaginary part of the effective index. Here, $n_{\text{eff(imag)}}$ is equal to the standard $\frac{\lambda[cm]}{4\pi}\ln(10^{loss[\frac{dB}{cm}]/10})$.

### 3.1.2 Phase Shifters

We use rib waveguides within our phase shifters. To achieve the phase shifts $\theta$, $\alpha$, $\beta$, we modulate the light using electro-optic modulators. Electro-optic modulators induce a phase shift by altering the effective index of the rib waveguide composing it. Forward-biased PIN diodes are able to achieve a larger effective index change $\Delta n_{\text{eff}}$. However, their speed is limited by the carrier recombination rate. In addition, they consume a significant amount of power and have a large absorption $\alpha$. Reverse-biased PN diodes, on the other hand, consume less power and are able to achieve higher speeds, but have a smaller effect on $\Delta n_{\text{eff}}$ and a greater loss.

Taking these factors into consideration, we proceed to use reverse-biased PN diodes in our electro-optic modulators, a Computer Aided Design (CAD) model of which is shown in Fig. 8. The cathode and anode are made from aluminum, and we use the standard carrier recombination models. An increase in applied voltage to one of these diodes corresponds to a decrease in the width of the depletion layer, directly modulating the light. The rib waveguide after carrier injection is shown in Fig. 9.

### 3.1.3 Mach-Zehnder Modulator

To implement a Mach-Zehnder Modulator (MZM), we place an electro-optic modulator on each arm of a Mach-Zehnder Inteferometer. Our undoped MZM's compact model is given by

$$n_{\text{eff(undoped)}}(\lambda) = 2.54606 - 0.872858(\lambda - 1.55) + 0.196977(\lambda - 1.55)^2 + jn_{\text{eff(imag)}}. \quad (5)$$

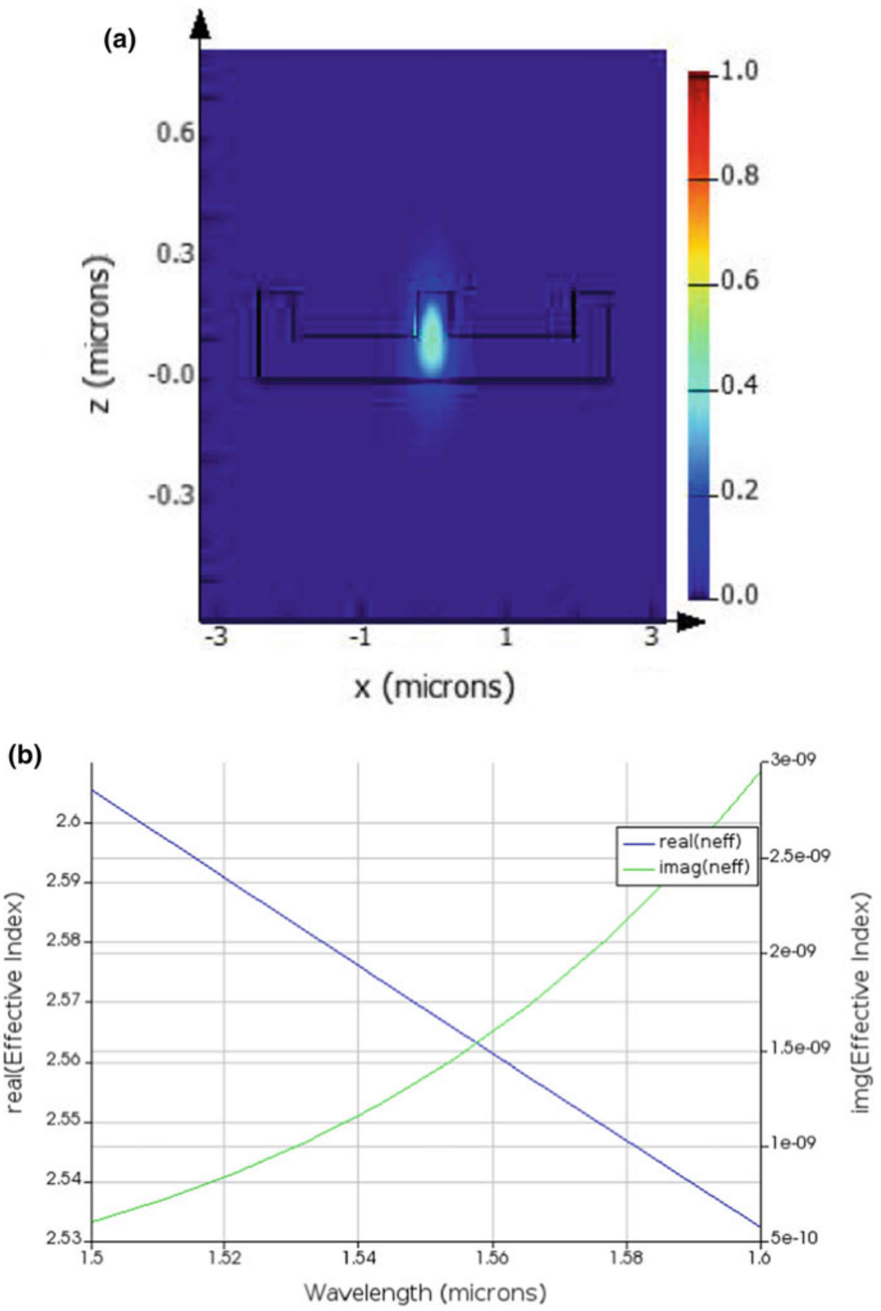


**Fig. 7** **a** Mode profile and **b** wavelength versus effective index in our undoped rib waveguides

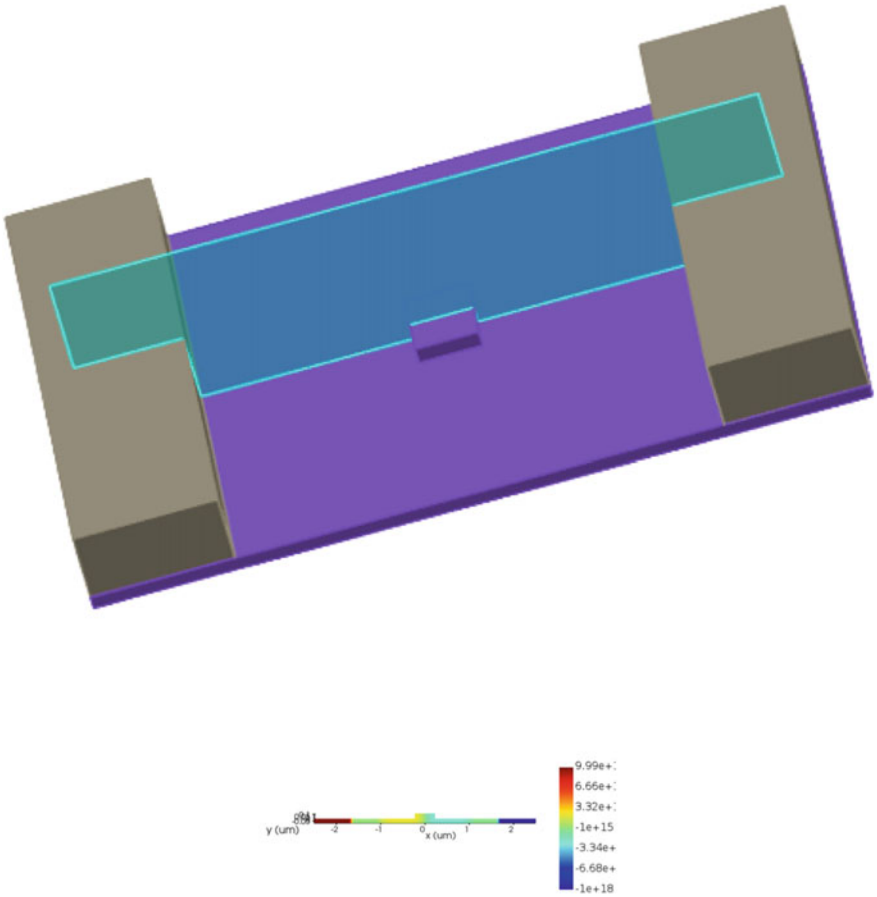


**Fig. 8** **a** We integrate a PN junction into the center of a rib waveguide. **b** N and P doped regions are placed to the left and right, respectively, increasing in doping concentration as shown

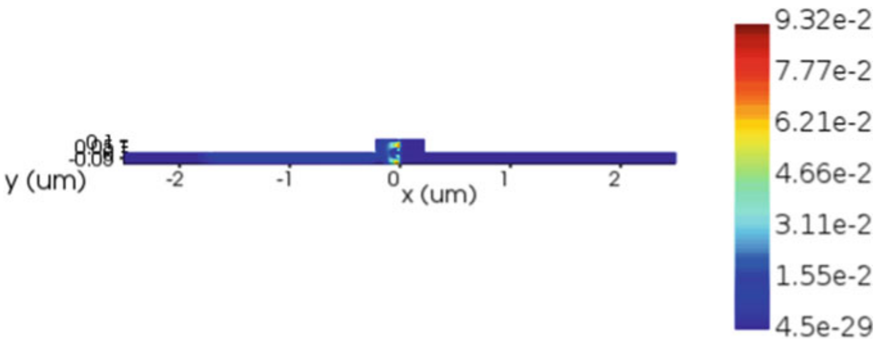


**Fig. 9** Waveguide charge profile

We now let the phase angles $\theta, \alpha, \beta = \frac{2\pi}{\lambda} L(n_{\text{eff(undoped)}_{1,2}}(\lambda) + \Delta n_{\text{eff}_{1,2}}(V))$, noting that we can halve the driving voltage needed by using the push-pull operation, where we apply opposite voltages to each electro-optic modulator on the MZM arm.

We integrate broadband directional couplers and extra modulators to our MZMs when needed. Note that our directional couplers each have a transmission coefficient $\tau = \frac{1}{\sqrt{2}}$ and a coupling coefficient $\kappa = \frac{1}{\sqrt{2}}$.

#### 3.1.4 Optical I/O

Our design makes use of on-chip quantum lasers and photodetectors for optical I/O.

During our fabricated chip tests, we used an off-chip HP81680A continuous wave (CW) diode laser and HP81635A InGaAs photodetectors.

### *3.2 Supporting Electronics*

All electronic and optical components are to be placed on a custom designed Printed Circuit Board (PCB). Most off-the-shelf and all optical components are surface mounted. The Teensy 4.1 development board is inserted into surface mounted female headers and connected to the host using a USB connection. The MPU is packaged into a Ceramic Pin Grid Array (CPGA) format which is inserted into a socket on the PCB. An example of the CPGA package is shown in Fig. 10. The RM57L843 slave microcontrollers are manufactured into 337-pin Ball Grid Array (BGA) packages, the AD9748 DACs are 32-lead Lead Frame Chip Scale Packages (LFCSP), and the AD8047 op amps are 8-lead Small Outline Integrated Circuit (SOIC) packages.

**Listing 1** Initiating a tensor algebra request on the host platform through Apollo.

```
// initialize vector v and rank–3 tensor T
let tensor v = {2, 9, 5};
let tensor T =
{
    {
        {3, 4, 2},
        {0, 8, 9},
        {1, 9, 4}
    },
    {
        {31, 3, 5},
        {0, 0, 3},
        {43, 0, 16}
    }
};
// compute tensor dot product and store resultant tensor R
let tensor R = T * v;
```

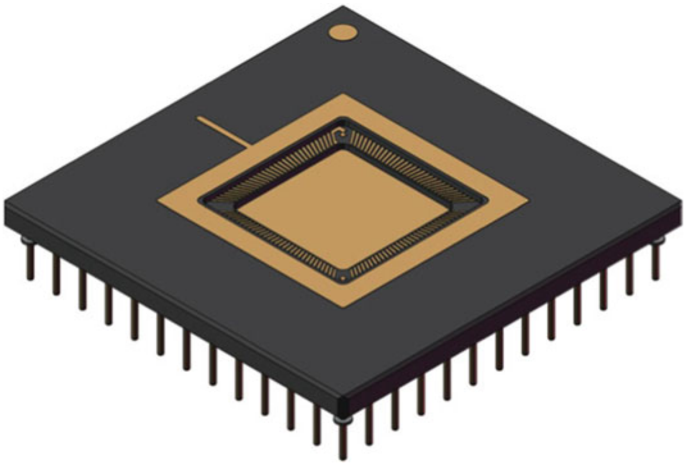


**Fig. 10** CPGA packaging used for the MPU

## 4 Programming the Accelerator

### *4.1 Overview*

We intend for the Tachyon accelerator to be programmable from an external host platform. Therefore, we developed a custom Domain-Specific Language (DSL) oriented toward tensor algebra computations, coined Apollo [15]. Apollo supports the scalar-tensor product, rank-$n$ Kronecker product, tensor inner product, tensor dot product, Khatri-Rao product, face-splitting product, and the $\mathbb{R}^3$ vector cross product. The language also supports writing tensors and vectors directly to the MPU for ease of future calculations [15].

As indicated in Fig. 11, the accelerator programming pipeline starts when user creates a program on a host platform. The Apollo compiler front end generates an optimized program tree given any input code sequence.[1] The compiler back end then takes the optimized tree and traverses it to generate appropriate Virtual Machine (VM) instructions. The VM then determines if the selected instruction involves MVM. If so, it accepts each leftover matrix or vector from the tensor computation, (1) mapping into corresponding phase angles/voltages and (2) scheduling groups of instructions to be sent to the on-accelerator microcontroller. If not, it keeps it within the host platform and executes it traditionally.

[1] The specifics of the compiler front end are discussed in [15].

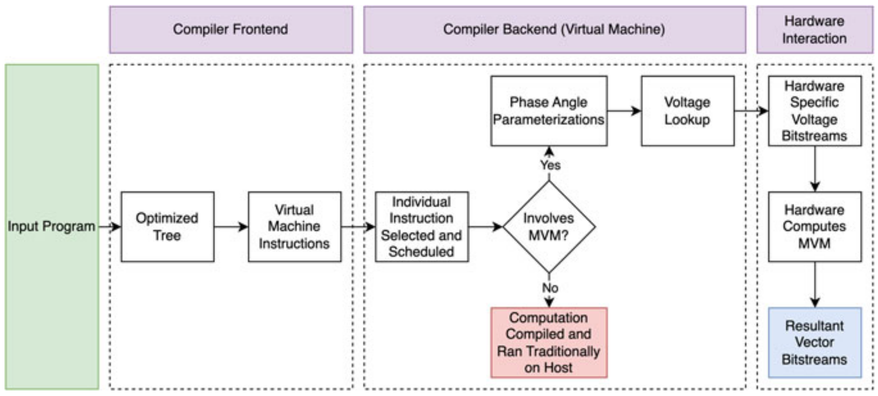

Compiler Frontend
Compiler Backend (Virtual Machine)
Hardware Interaction
Input Program
Optimized Tree
Virtual Machine Instructions
Individual Instruction Selected and Scheduled
Involves MVM?
Yes
No
Phase Angle Parameterizations
Voltage Lookup
Computation Compiled and Ran Traditionally on Host
Hardware Specific Voltage Bitstreams
Hardware Computes MVM
Resultant Vector Bitstreams

**Fig. 11** General program flow within the Apollo DSL compiler

## *4.2 Compiler Back End*

The compiler back end takes the generated Virtual Machine (VM) instructions as input. For any selected instruction, it schedules it to be run and determines whether or not it is MVM-based. If it is MVM-based, the VM runs a process unique to the Tachyon setup. In the case where a rank-$n$ tensor-related request (where $n > 2$) is initialized as in Listing 1, the compiler implicitly allocates memory in the Binary Sparse Tensor Tree format [15]. The tensor is recursively unrolled into matrices as shown in Fig. 12. Each matrix $\mathbf{M}$ is then converted into a set of voltages which, when sent to the modulators, would induce phase shifts collectively representing $\mathbf{M}$. This is done through the phase angle parameterizations discussed in Sect. 4.3. The assembly instruction in Table 1 corresponding to the request is then executed. If $n < 2$, the unrolling step is skipped, and the remaining steps stay the same. Results are sent back to the host computer and ripple up the compiler back to the user.

## *4.3 Phase Angle Parameterization*

It is important to calculate the required phase angles as parameterizations of input matrices and vectors to allow the compiler to send the appropriate signals. Since the Matrix Processing Unit (MPU) represents a matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ through the SVD, parameterizing $\mathbf{M}$ is done by parameterizing each individual SVD term.

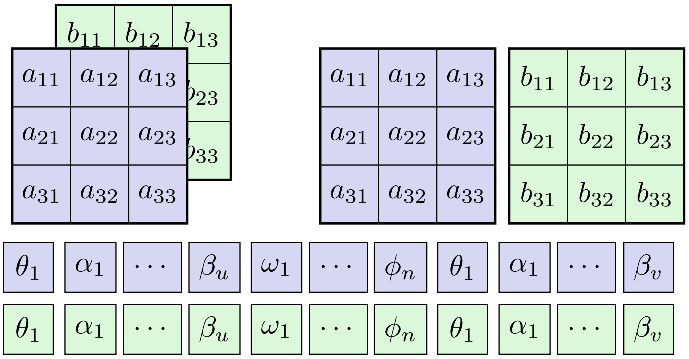

$a_{11}$ $a_{12}$ $a_{13}$
$a_{21}$ $a_{22}$ $a_{23}$
$a_{31}$ $a_{32}$ $a_{33}$
$b_{11}$ $b_{12}$ $b_{13}$
$b_{21}$ $b_{22}$ $b_{23}$
$b_{31}$ $b_{32}$ $b_{33}$
$\theta_1$ $\alpha_1$ $\cdots$ $\beta_u$ $\omega_1$ $\cdots$ $\phi_n$ $\theta_1$ $\alpha_1$ $\cdots$ $\beta_v$

**Fig. 12** Writing a tensor to the MPU. **a** An tensor is initialized and **b** unrolled into matrices by the Apollo compiler; **c** each matrix is then translated into a set of angles parameterizing it. The angles are mapped into the voltages needed to achieve them and are then sent to the MPU

### 4.3.1 Unitary Matrix Phase Angles

We first go over phase angle parameterizations of unitary matrices $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V}^T \in \mathbb{R}^{n\times n}$. Recall that the MPU uses a Mach-Zehnder Modulator (MZM) to implement (1). The values of the phase angles $\theta^{(2)}$, $\alpha^{(2)}$, and $\beta^{(2)}$ in terms of the desired matrix constants can be calculated through a system of equations. For any given unitary matrix $\mathbf{U}^{(2)} \in \mathbb{R}^{2\times 2}$, with values $u_{11}^{(2)}, u_{12}^{(2)}, u_{21}^{(2)}, u_{22}^{(2)}$, it is true that

$$u_{11}^{(2)}(\alpha^{(2)}, \beta^{(2)}, \theta^{(2)}) = \frac{1}{2}\mathrm{e}^{-j\alpha^{(2)}}(\mathrm{e}^{-j\theta^{(2)}} - 1) \tag{6a}$$

$$u_{12}^{(2)}(\alpha^{(2)}, \beta^{(2)}, \theta^{(2)}) = \frac{j}{2}\mathrm{e}^{-j\alpha^{(2)}}(1 + \mathrm{e}^{-j\theta^{(2)}}) \tag{6b}$$

$$u_{21}^{(2)}(\alpha^{(2)}, \beta^{(2)}, \theta^{(2)}) = \frac{j}{2}\mathrm{e}^{-j\beta^{(2)}}(1 + \mathrm{e}^{-j\theta^{(2)}}) \tag{6c}$$

$$u_{22}^{(2)}(\alpha^{(2)}, \beta^{(2)}, \theta^{(2)}) = -\frac{1}{2}\mathrm{e}^{-j\beta^{(2)}}(\mathrm{e}^{-j\theta^{(2)}} - 1). \tag{6d}$$

We vary the phase angle values and store $\vec{\mathbf{f}}_{u^{(2)}}(\theta^{(2)}, \alpha^{(2)}, \beta^{(2)}) = u_{11}^{(2)}\hat{\mathbf{e}}_1 + u_{12}^{(2)}\hat{\mathbf{e}}_2 + u_{21}^{(2)}\hat{\mathbf{e}}_3 + u_{22}^{(2)}\hat{\mathbf{e}}_4$ in a lookup table, where $\vec{\mathbf{f}}_{u^{(2)}} : \{0 \leq \alpha^{(2)}, \beta^{(2)}, \theta^{(2)} \leq 2\pi\} \to \mathbb{C}^4$. We read phase angle values backwards from the stored $\mathbf{U}^{(2)}$ constants; if they are not available during lookup, we compute them in real time. Solving (6) yields the following phase angle equations under constraints $\{u_{11}^{(2)} = u_{22}^{(2)} = 0, u_{12}^{(2)} \neq 0, u_{21}^{(2)}u_{12}^{(2)} = -1\}$:

$$\theta^{(2)} = 2\pi C_1 \tag{7a}$$

$$\alpha^{(2)} = 2\pi j C_2 + \ln\left[-j u_{22}^{(2)}\right] \tag{7b}$$

$$\beta^{(2)} = 2\pi C_3 - j \ln \frac{j}{u_{12}^{(2)}}. \tag{7c}$$

or, under constraints $\{u_{12}^{(2)^2} - u_{11}^{(2)^2} \neq 0, u_{11}^{(2)} \neq 0, u_{11}^{(2)} - j u_{12}^{(2)} \neq 0, u_{21}^{(2)} = -\frac{u_{12}^{(2)}}{u_{12}^{(2)^2} - u_{11}^{(2)^2}}, u_{22}^{(2)} = -\frac{u_{11}^{(2)}}{u_{11}^{(2)^2} - u_{12}^{(2)^2}}\}$:

$$\theta^{(2)} = 2\pi C_1 - j \ln \frac{j u_{12}^{(2)} - u_{11}^{(2)}}{u_{11}^{(2)} + u_{12}^{(2)}} \tag{8a}$$

$$\alpha^{(2)} = 2\pi C_2 - j \ln\left[-u_{11}^{(2)} - j u_{12}^{(2)}\right] \tag{8b}$$

$$\beta^{(2)} = 2\pi C_3 - j \ln \frac{u_{11}^{(2)} + j u_{12}^{(2)}}{u_{12}^{(2)^2} - u_{11}^{(2)^2}}. \tag{8c}$$

where $C_1, C_2, C_3$ are constants. These give the required phase angles directly. We set them to zero for simplicity.

Recall that on the MPU, a unitary matrix $\mathbf{U}^{(n)} \in \mathbb{R}^{n \times n}$ is represented as (2). The angles $\theta$, $\alpha$, and $\beta$ then build up to $\frac{3}{2}n(n-1)$ angles used to parameterize $\mathbf{U}^{(n)}$:

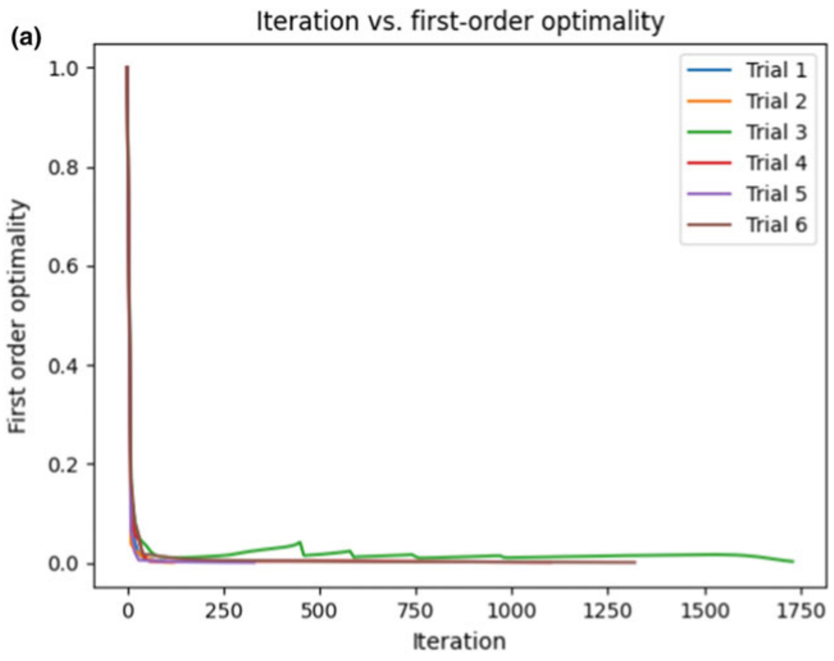
$$\theta_1^{(n)}, \alpha_1^{(n)}, \beta_1^{(n)}, \theta_2^{(n)}, \ldots, \alpha_{\frac{1}{2}n(n-1)}^{(n)}, \beta_{\frac{1}{2}n(n-1)}^{(n)}. \tag{9}$$

We solve for these angles by equating each entry of $\mathbf{U}^{(n)}$ to a series of $u_{m_{ij}}^{(2)}$ products and substituting each vector field angle stored. In the $4 \times 4$ case,

$$\begin{aligned}\mathbf{U}^{(4)} = &\begin{bmatrix} 1 & & \\ & 1 & \\ & & \mathbf{U}_1^{(2)} \end{bmatrix} \begin{bmatrix} 1 & & \\ & \mathbf{U}_2^{(2)} & \\ & & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_3^{(2)} & & \\ & 1 & \\ & & 1 \end{bmatrix} \\ &\begin{bmatrix} 1 & & \\ & 1 & \\ & & \mathbf{U}_4^{(2)} \end{bmatrix} \begin{bmatrix} 1 & & \\ & \mathbf{U}_5^{(2)} & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & \mathbf{U}_6^{(2)} \end{bmatrix} \in \mathbb{R}^{4 \times 4}.\end{aligned} \tag{10}$$

Note that the constraint $\det(\mathbf{U}^{(4)}) = \det(\mathbf{U}_{1,2,\ldots,6}^{(2)}) = 1$ must still hold. Since there are 16 $u_{ij}^{(4)}$ values and 24 $u_{m_{ij}}^{(2)}$ values, many possible solutions exist for this system. The Jacobian matrix $\mathbf{J}$ of a function $\vec{\mathbf{f}}(u_{1_{11}}^{(2)}, u_{1_{12}}^{(2)}, \ldots, u_{6_{22}}^{(2)}) : \mathbb{R}^{24} \to \mathbb{R}^{16}$ mapping vectors of $\mathbf{U}^{(2)}$ values to $\mathbf{U}^{(4)}$ values is highly sparse and rank-deficient, so we solve using a tensor-Krylov method rather than a multivariate form of Newton's method [16]. Convergence plots are shown for various trials in Fig. 13.

We store $\vec{\mathbf{f}}_{u^{(n)}} : \mathbb{C}^{\frac{3}{2}n(n-1)} \to \mathbb{C}^{n^2}$ in a secondary lookup table for efficient access.



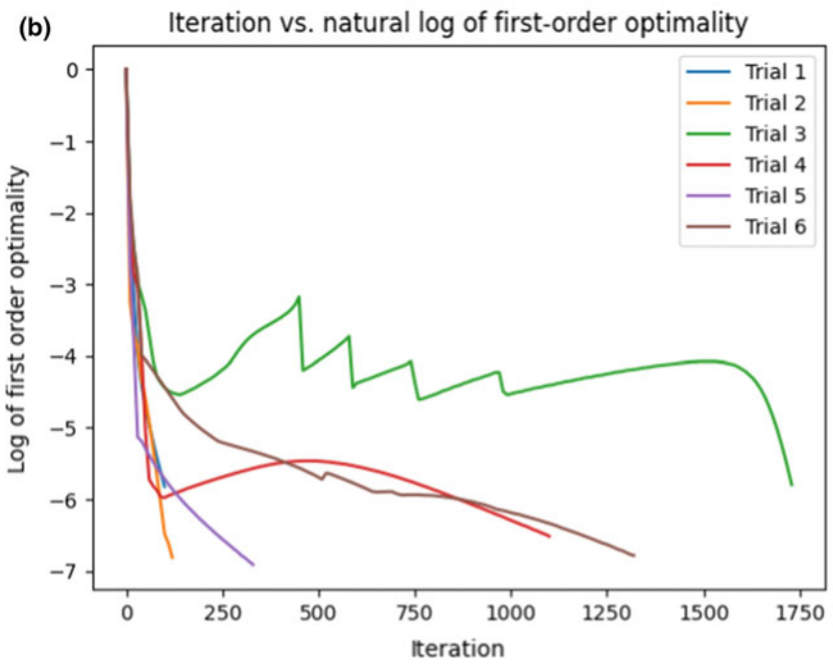


**Fig. 13** Convergence plots for various tensor-Krylov runs. **a** Iteration versus first-order optimality. **b** Same algorithm iteration versus ln(first-order optimality)

### 4.3.2 Diagonal Matrix Phase Angles

The diagonal matrix in the SVD for a matrix $\mathbf{M} \in \mathbb{R}^{m\times n}$ is given by

$$\Sigma = \left[\begin{array}{cccc|c} \sigma_{11} & & & & 0 \cdots 0 \\ & \sigma_{22} & & & 0 \cdots 0 \\ & & \ddots & & \vdots \ddots \vdots \\ & & & \sigma_{nn} & 0 \cdots 0 \end{array}\right] \in \mathbb{R}^{m\times n}. \tag{11}$$

Augmenting the $(m-n)\times n$ zero matrix to the original $n \times n$ diagonal matrix allows the SVD product to retain the initial horizontal dimension of $\mathbf{M}$. On Tachyon, each entry $\sigma_{ii}$ is represented as a function of phase angles $\omega_{ii}^{(n)}$ and $\phi_{ii}^{(n)}$:

$$\sigma_{ii}(\omega_{ii}, \phi_{ii}) = \frac{1}{2}\mathrm{e}^{j\omega_{ii}}(\mathrm{e}^{j\phi_{ii}} - 1). \tag{12}$$

Thus, a parameterization of $\Sigma$ requires $2n$ phase angles $\omega_{11}, \phi_{11}, \omega_{22}, \phi_{22}, \ldots, \omega_{nn}, \phi_{nn}$. We solve for these angles by letting $\omega_{ii} = \phi_{ii}$, since we want to distribute the workload evenly between modulators. Ideally, we find that

$$\omega_{ii} = \phi_{ii} = 2\pi n - j \ln \frac{1 \pm \sqrt{8\sigma_{ii}+1}}{2}, \tag{13}$$

where $\sqrt{8\sigma_{ii}+1} \neq -1, 1$ for the positive and negative cases respectively, and $n \in \mathbb{Z}$; setting $n = 0$ minimizes $|\omega_{ii}, \phi_{ii}|$, simplifies (13), and is usable. We vary $\omega_{ii}$ and $\phi_{ii}$ and store $\sigma_{ii}$ in a lookup table for fast access.

### 4.3.3 Vector Phase Angles

Unlike the matrix case, a length $n$ vector $\vec{\mathbf{v}}$ has exactly $n$ parameters $\gamma_1, \gamma_2, \ldots, \gamma_n$, which are all phase angles. From Fig. 2, it is apparent that

$$\vec{\mathbf{v}} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \frac{E_{in}}{2}\begin{bmatrix} \mathrm{e}^{j\gamma_1} \\ \mathrm{e}^{j\gamma_2} \\ \vdots \\ \mathrm{e}^{j\gamma_n} \\ , \end{bmatrix} \tag{14}$$

where $E_{in}$ is the amplitude of the input electric field of the laser light into the chip.[2] We solve for the $k$-th phase angle $\gamma_k$; since $v_k = \frac{E_{in}}{2}\mathrm{e}^{j\gamma_k}$,

[2] Note that $E_{in}$ is a global variable kept constant throughout each individual MVM.

$$\gamma_k = -j \ln \frac{2v_k}{E_{in}}. \tag{15}$$

$E_{in}$ is calculated by averaging the input vector's electric field amplitudes.

These phase angles and amplitude values are converted to distinct voltages to sent to the Tachyon accelerator through a lookup table.

## 5 Results

We demonstrate the efficacy of our core on matrix multiplication with matrices of various sizes, and we measure the efficacy for each size. The MPU stats are given in Table 2. Unlike most traditional digital electronic processors, the MPU performs an MVM in a single pass. Raw clock speed is 3 orders of magnitude higher than Google's Tensor Processing Unit (TPU). The performance per watt is 245 TFLOPS/watt for the MPU's optical core. Power draw does not scale dramatically with matrix size, leading to an estimated 261 TFLOPs/watt rating for a 256 × 256 optical core (the dimensions currently used within Google's TPU v4).

The MPU core comes with several benefits. It can perform an entire ANN forward pass in a single clock cycle. In addition, it expels negligible heat due to its silicon photonic architecture. At scale, it is faster and more energy-efficient than specialized ML hardware: Google's TPU v4 and GPUs across the board (NVIDIA's A100, all GTX/RTX chips, AMD's Radeon chips, etc.).

However, the overall Tachyon accelerator's performance limiting factor is the surrounding electronic interconnect. We estimate a performance drop to approximately 13.4 KFLOPS and 1.6 KFLOPS/W with the cheap, off-the-shelf components used in research; the drop would be much lower with high quality, custom analog integrated circuits (ICs). In addition, due to possessing a very specialized circuit, the accelerator is not applicable (advantageously) to training neural networks without further component optimization.

**Table 2** Optical MPU core stats

| Specification | Value (4 × 4) | Value (256 × 256) |
|---|---|---|
| Instructions | 1 MVM/cycle | 1 MVM/cycle |
| Clock | 220 GHz | 3.7 GHz |
| PPW | 245 TFLOPS/watt | 261 TFLOPS/watt |
| Performance | 7 TFLOPS | 478 TFLOPS |

## 6 Conclusion

In this paper, we introduce Tachyon, a highly scalable optical hardware accelerator targeted toward matrix-vector multiplication (MVM). We go over the accelerator design, cost-efficient implementation, and programming interface. We then go over accelerator statistics, benefits, and scalability. We show that the accelerator is able to

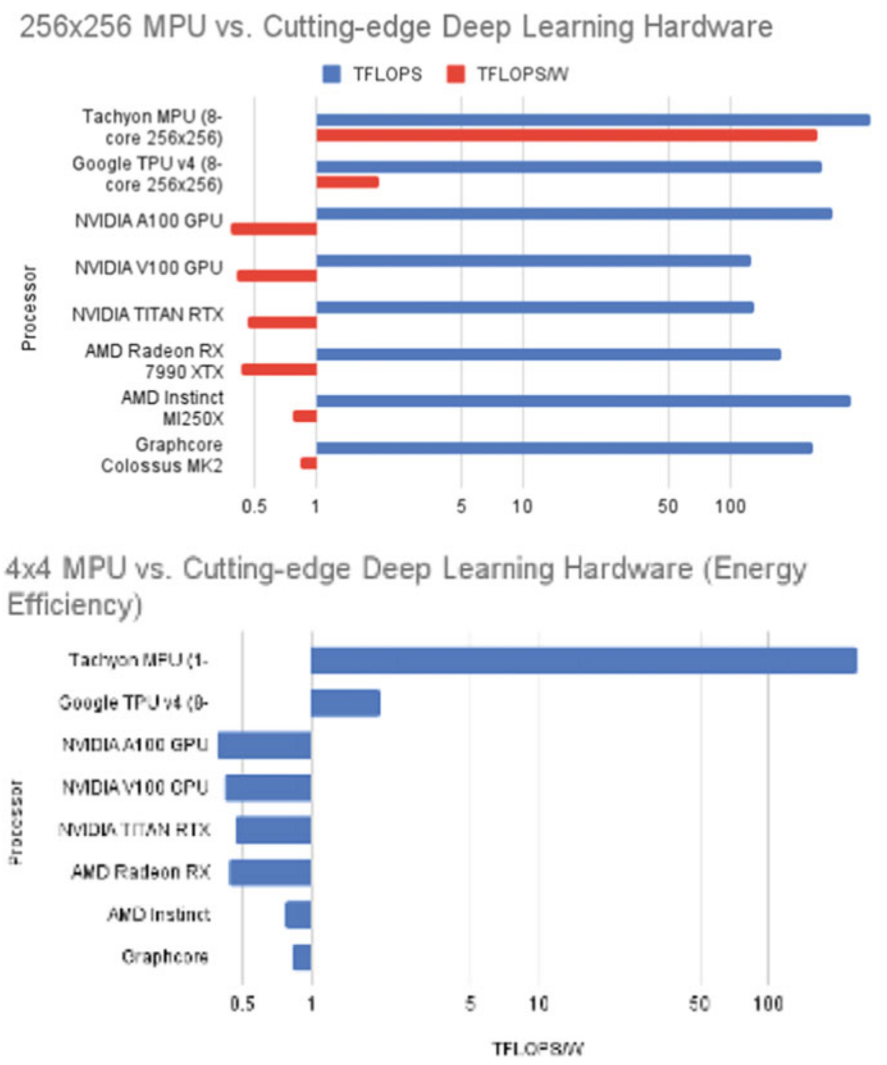


**Fig. 14** **a** Scaled 256 × 256 MPU core outperforms TPU v4 and GPUs in FLOPS and FLOPS/W [17, 18]. **b** Tested 4 × 4 MPU core has an energy-efficiency similar to scaled version. However, the FLOPS rating is significantly below most deep learning hardware (7.0 TFLOPS) and is not shown

perform an MVM in a single cycle with a clock speed 3 orders of magnitude higher than Google's TPU v4 and a PPW 2 orders of magnitude higher, while still scaling very efficiently to higher matrix dimensions (Fig. 14).

The Tachyon accelerator, in using configured phase shifter settings to carry out computations, is best suited for MVM situations where the matrix is held constant and vectors are multiplied sequentially. This is the case in neural network inference, making the Matrix Processing Unit (MPU) highly tuned toward forward passes. Future research from a software perspective will focus on creating efficient compilers to map large models onto Tachyon to take advantage of this capability. Integration with packages in popular languages will also be investigated to allow an end user to easily access the accelerator's capabilities.

From a hardware perspective, future research will focus on developing integrated circuits along with optimizing the bottleneck between optics and electronics. In addition, further optical enhancements will be investigated, such as parallelization boosts through wavelength division multiplexing. Such enhancements are stepping stones toward our end goal of having the Tachyon accelerator as a product that consumers can use to advantageously accelerate their machine learning workloads.

**Acknowledgements** We thank the following individuals and organizations for their contributions to this project: Mr. Jonathan Hummel from the Maryland NanoCenter for printing initial designs with the Raith e-LiNE; Dr. Lukas Chrostowski and Dr. Iman Taghavi from the University of British Columbia for printing later designs with the JEOL JBX-6300F and performing our requested tests; Thorlabs Quantum Electronics for advice regarding design optimization; Dr. Yanne Chembo from the Institute for Research in Electronics and Applied Physics for advice on future development methodologies; Sherry Yu and Tarini Basireddy for fabrication advice; Mr. Mark Hannum for providing mathematical assistance and helpful resources; Vishal Nandakumar for recommending Newton-class approximation methods; Shihao Cao for support and useful discussion about the project's future; Mr. Hari Krishnan from NVIDIA for insightful discussion; and finally, our families for extended support and patience.

## References

1. Garcia-Gasulla D, Peres F, Vilalta A. On the behavior of convolutional nets for feature extraction
2. Li C (2022) OpenAI's GPT-3 language model: a technical overview. Lambda, Inc. https://lambdalabs.com/blog/demystifying-gpt-3
3. Saul J, Bass D (2023) How much energy do AI and CHATGPT use? no one knows for sure. Bloomberg. https://www.bloomberg.com/news/articles/2023-03-09/how-much-energy-do-ai-and-chatgpt-use-no-one-knows-for-sure
4. Jouppi NP, Young C, Patil N, Patterson DA, Agrawal G, Bajwa R, Bates S, Bhatia S, Boden N, Borchers A Boyle R, Cantin P, Chao C, Clark C, Coriell J, Daley M, Dau M, Dean J, Gelb B, Ghaemmaghami TV, Gottipati R, Gulland W, Hagmann R, Ho CR, Hogberg D, Hu J, Hundt R, Hurt D, Ibarz J, Jaffey A, Jaworski A, Kaplan A, Khaitan H, Koch A, Kumar N, Lacy S, Laudon J, Law J, Le D, Leary C, Liu Z, Lucke K, Lundin A, MacKean G, Maggiore A, Mahony M, Miller K, Nagarajan R, Narayanaswami, R, Ni, R, Nix, K, Norrie, T, Omernick, M, Penukonda, N, Phelps, A, Ross, J, Salek, A.,Samadiani, E, Severn, C, Sizikov, G, Snelham, M, Souter J, Steinberg D, Swing A, Tan M, Thorson G, Tian B, Toma H, Tuttle E, Vasudevan V, Walter R,

Wang W, Wilcox E, Yoon DH (2017) In-datacenter performance analysis of a tensor processing unit. CoRR arXIV:abs/1704.04760 1704.04760 (2017)
5. Cloud tensor processing units (TPUs) nbsp;|nbsp; google cloud. Google. https://cloud.google.com/tpu/docs/tpus
6. Moore's Law or how overall processing power for computers will double every two years. http://www.mooreslaw.org/
7. Cowan GER, Melville RC, Tsividis YP (2006) A VLSI analog computer/digital computer accelerator. IEEE J Solid-State Circ 41(1):42–53. https://doi.org/10.1109/JSSC.2005.858618
8. Guo N, Huang Y, Mai T, Patil S, Cao C, Seok M, Sethumadhavan S, Tsividis Y (2016) Energy-efficient hybrid analog/digital approximate computation in continuous time. IEEE J Solid-State Circ 51(7):1514–1524. https://doi.org/10.1109/JSSC.2016.2543729
9. M1076 analog matrix processor (2022). https://mythic.ai/products/m1076-analog-matrix-processor/
10. Clements WR, Humphreys PC, Metcalf BJ, Kolthammer WS, Walmsley IA (2017) An optimal design for universal multiport interferometers (2017)
11. PJRC: Teensy®4.1 Development Board. PJRC. https://www.pjrc.com/store/teensy41.html
12. Texas Instruments (2016) 16/32 Bit arm cortex-R5F flash MCU, RISC, EMAC. Texas Instruments. Rev. C. https://www.ti.com/lit/ds/symlink/rm57l843.pdf?
13. Analog Devices (2005) 8-Bit, 210 MSPS TxDAC®D/A converter. Analog Devices. Rev. B. https://www.analog.com/media/en/technical-documentation/data-sheets/AD9748.pdf
14. Analog Devices (1995) 250 MHz, General purpose voltage feedback Op amps gain 1 stable. Analog Devices. Rev. A. https://www.analog.com/media/en/technical-documentation/data-sheets/AD8047_8048.pdf
15. Redrouthu S, Athavale R (2022) Tensor algebra on an optoelectronic microchip. https://arxiv.org/abs/2208.06749
16. Bader BW, Pawlowski RP, Kolda TG (2005) Robust large-scale parallel nonlinear solvers for simulations. https://www.osti.gov/servlets/purl/876345
17. Jouppi NP, Kurian G, Li S, Ma P, Nagarajan R, Nai L, Patil N, Subramanian S, Swing A, Towles B, Young C, Zhou X, Zhou Z, Patterson D (2023) TPU v4: an optically reconfigurable supercomputer for machine learning with hardware support for embeddings (2023)
18. Index. https://gadgetversus.com/