

Physically Reconfigurable High-Performance Computing with Volume Mount Devices

Dimitar Dimitrov

The Center for Bits and Atoms
Massachusetts Institute of Technology
Cambridge, MA
dimitar.dimitrov@cba.mit.edu

Erik Strand

The Center for Bits and Atoms
Massachusetts Institute of Technology
Cambridge, MA
erik.strand@cba.mit.edu

Neil Gershenfeld

The Center for Bits and Atoms
Massachusetts Institute of Technology
Cambridge, MA
neil.gershenfeld@cba.mit.edu

Abstract—We show how Volume Mount Devices can be used to create reconfigurable computing accelerators. These accelerators continually embody the algorithms they run, reducing data movement and enabling seamless software-hardware codesign. We demonstrate such an accelerator using prototype devices based on a commercial off-the-shelf processor. We then show how this technology can scale to realize cost-effective and energy-efficient high-performance computing.

Index Terms—Volume Mount Devices, Accelerators

I. INTRODUCTION

Since the dawn of computing, the same paradigm has held: machines are designed with a fixed, immutable set of capabilities, and workloads are mapped to those capabilities. It is primarily driven by an impedance mismatch between the fast, low-cost, low-risk nature of iteration in software, and the slow, expensive, and risk-laden nature of iteration in hardware. Volume Mount Devices (VMDs) promise to correct this mismatch, by providing the ability to compile high-performance hardware in a parametric, scalable way, similar to the way in which software is developed. VMDs, introduced by the project team [1], [2], are a generalization of Surface Mount Devices (SMDs) to three dimensions. The reconfigurable nature of VMD systems allows for a continuous topological equivalency between hardware and software, removing the need for difficult-to-optimize workload-to-machine mappings and dramatically reducing power-hungry data movement.

In this work, we demonstrate how accelerators can be constructed out of VMDs. To aid development and experimentation, we use commercial off-the-shelf (COTS) processors as building blocks. Section II briefly covers some background concepts. Section III discusses why we design VMD accelerators. Section IV discusses how we design VMD accelerators. Section V describes one such implementation. Section VI presents results from that implementation. Section VII discusses how VMD accelerators scale. Section VIII concludes this work.

II. BACKGROUND

A. Prior Work

a) *Three-Dimensional Electronics:*

For most of the history of electronics, devices have been assembled using flat, two-dimensional components on flat, two-dimensional boards. There has been an increased interest

recently in developing three-dimensional integration, partly because of the physical scaling limits of lithography, and partly because of an increased demand for communication density [3], [4], [5]. Examples include 3D Heterogeneous Integration [6] and device-specific solutions such as High Bandwidth Memory [7]. All of these technologies, however, are limited to a few layers in the third dimension. Thus, they do not enable new forms of computation fundamentally. Also, all of these technologies suffer from heat dissipation concerns [8].

b) *Modular Electronics:*

There has also been a long history [9] of modular electronics. Most of these attempts, however, have not been aimed at high-performance computing (HPC), and suffer from scaling limitations. Chiplets are one modular solution for HPC [10], but they do not enable the rapid development cycle modular systems aim to enable.

c) *Reconfigurable Electronics:*

Chiplets are also not reconfigurable after assembly. The most prevalent reconfigurable solution is the field-programmable gate array (FPGA). FPGAs, however, suffer from substantial overheads leading to high power dissipation, large area, and – as a consequence – considerable cost [11].

d) *Compute-In-Memory (CIM):*

Numerous CIM approaches have been explored as methods for avoiding the so-called von Neumann bottleneck [12], [13], [14]. All of these methods, however, require algorithms to be mapped to the hardware. This mapping is not only difficult, but often significantly less efficient than the optimal scenarios the hardware was designed for.

e) *Accelerators:*

Similarly, various accelerators have been designed for specific algorithms [15], [16]. These bespoke solutions are usually based on custom silicon, leading to high costs and long development cycles.

General-purpose accelerators, on the other hand, suffer from a lack of heterogeneity. Most are based on systolic arrays of fixed sizes [17], [18], and require frequent movement of constant values into these arrays.

B. VMDs

VMDs seek to address all of these challenges [1]. Instead of being mounted to two-dimensional carriers, VMDs are assembled natively in three dimensions using specialized assemblers that are a cross between a pick-and-place and a

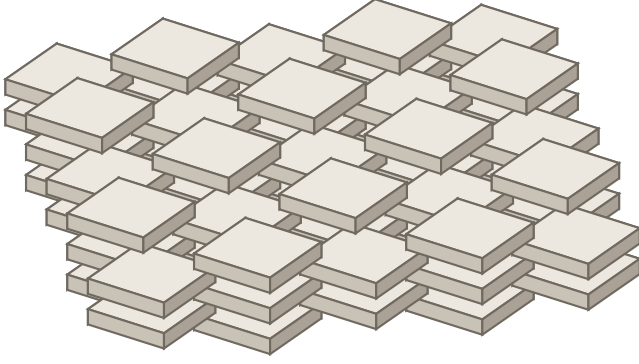


Fig. 1. An assembly of planar square VMDs.

3D printer [2]. They are interdigitated so that air or coolant can naturally flow through the structure, reducing the power-dissipation concerns typical for three-dimensional structures.

VMDs are being developed at multiple physical scales, from the centimeter-scale devices discussed in this work down to micron-scale devices. They can take various shapes, but a convenient shape is a planar square that can be assembled by tiling in three dimensions, as shown in Fig. 1. For certain VMDs, the assembly process is reversible, so that VMDs can be reused as a design changes.

VMDs are also being developed at multiple functional scales. Transistor-level VMDs can be used to fill gaps in scaling roadmaps, reduce supply chain risk by democratizing hardware development, or realize heterogeneous systems that satisfy high current or voltage requirements. Functional-level VMDs leverage traditional integration, while keeping system architecture modular and flexible.

Accelerators can benefit particularly well from the reconfigurability of VMD systems. Firstly, it reduces the cost and shortens the timeline for accelerator development, obviating the need for custom silicon solutions. Secondly, it removes arbitrary constraints from software development, allowing for truly iterative software-hardware codesign. Most importantly, reconfigurability allows for machines to physically embody the workloads they run. This naturally realizes CIM, without the need for per-workload mapping. Mirroring hardware and software substantially reduces data movement, which can drastically reduce overall energy per operation [19].

As discussed in Section VII, care is taken to ensure functional-level VMDs remain scalable, with cost- and energy-efficiency maintained at the VMD level. This allows VMD accelerators to be used for HPC, and lets their cost- and power-saving benefits be realized at scale.

III. VMD ACCELERATORS

The ability of VMD accelerators to physically mirror the algorithms they run can decrease data movement and increase efficiency in a variety of workloads. This is due not only to the reconfigurability and programmability of VMD systems, but also to their three-dimensional nature.

A. Machine Learning

The first class of workloads we consider is machine learning algorithms, which are typically run on Graphics Processing Units (GPUs). While GPUs are highly performant and computationally efficient, they do not mirror the algorithms they run, and therefore require constant data movement, which leads to excessive power consumption [19]. Solutions to this problem have generally taken the form of accelerators in custom silicon [15], [16]. The permanent nature of a traditional silicon chip, however, means that mirroring the algorithm is at odds with reconfigurability. So, custom silicon solutions quickly become obsolete as the algorithms develop, leaving the less heterogeneous GPUs as a more attractive long-term solution. VMD accelerators, on the other hand, allow for both reconfigurability and for machine-workload mirroring simultaneously.

For certain parts of the workload, GPUs work fairly well. Take, for example, general-element matrix-vector (GEMV) multiplication. A biased $m \times n$ GEMV multiplication is given by:

$$\vec{O}_m = \mathbf{W}_{m \times n} \times \vec{I}_n + \vec{B}_m \quad (1)$$

Topologically, this is a flat, two-dimensional operation, which looks similar to the flat, two-dimensional silicon die in a GPU. It is not a perfect fit, though, because each compute node in the GPU is too large, complex, and power-hungry for the simple task of multiply-accumulate. This makes the die much larger than it has to be for the operation. To reduce the area demand, the same silicon is reused for different parts of the operation. So the projection values (which are unchanging on the order of months) have to continually be loaded and erased to make use of the hardware, degrading performance and increasing energy consumption dramatically. For the $2 \times m \times n$ operations involved in a single GEMV multiplication (counting multiplication and accumulation as separate operations), the accelerator loads n elements for the input vector and $m \times n$ elements for the matrix and stores m elements for the output vector. The arithmetic intensity is thus $\frac{2 \times m \times n}{m \times n + m + n}$, which is always less than 2. Because of the high energetic cost of accessing memory, a low arithmetic intensity leads to a low energy efficiency.

Using a VMD accelerator for GEMV multiplication allows for smaller functional units, decreasing two-dimensional area. Furthermore, by folding the operation into a third dimension, the total volume of the accelerator shrinks, allowing for projection values to remain where they are used for a long time. This means that the $m \times n$ elements for the matrix do not need to be loaded, so the arithmetic intensity is increased to $\frac{2 \times m \times n}{m + n}$ – which can be very high for large m, n . As such, GEMV multiplication is a natural first demonstration of a VMD accelerator.

Other operations are an even worse fit for GPUs than GEMV multiplication. Take, for example, the ubiquitous multi-head attention (MHA) operation, reproduced in Fig. 2a [20]. While each head in MHA uses a unique linear map for the query, key, and value, the inputs to these linear maps are the same data, making MHA a three-dimensional workload. Mapping this workload to a two-dimensional accelerator

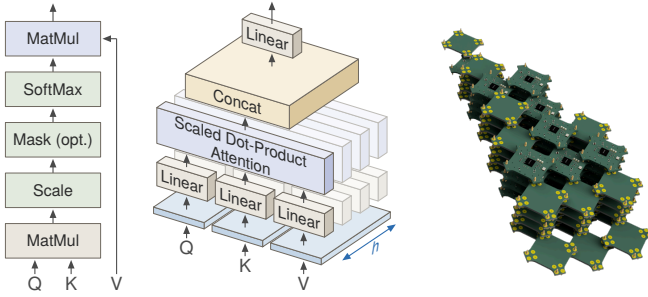


Fig. 2a. “(left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.” [20]

Fig. 2b. A VMD accelerator for MHA using processor VMDs.

Fig. 2. Multi-Head Attention.

inherently presents data movement challenges, since one of the three axes has to be flattened. Meanwhile, it is trivial to construct a three-dimensional VMD accelerator for such a workload, as illustrated in Fig. 2b.

Both GEMV multiplication and MHA are used as “layers” in machine learning algorithms. While each such layer can be optimized in a VMD accelerator, the machine learning algorithms as a whole can be optimized as well. Namely, the logical layers of the algorithm become physical layers of the VMD device. This allows for removing all movement of constant data.

B. Physics Simulation

Many physics simulations are governed primarily by local interactions, making spatial decomposition a natural approach for both parallelization and mapping the workload to a VMD accelerator. In molecular dynamics, for example, many interaction potentials are truncated at a cutoff distance, so a particle interacts only with others within a bounded radius. This locality permits the domain to be partitioned into cells no smaller than the cutoff, such that each cell need only exchange particle data with its immediate neighbors [21]. Anton, the molecular dynamics supercomputer built by D. E. Shaw Research, is built around this decomposition [22]. Similarly, in a VMD accelerator, each cell can be assigned to a VMD, and the VMDs can be arranged in a three-dimensional grid that mirrors the spatial decomposition. In this configuration all communication is between physically adjacent VMDs and interconnect length is independent of system size, as depicted in Fig. 3.

Furthermore, the work performed within each cell for each time step is typically proportional to the volume of the cell (e.g. if particles are roughly evenly distributed in space), while the amount of communication is proportional to the surface area of the cell. Since the compute-to-communication ratio grows linearly with cell edge length, the cell size can be tuned to balance the two for any given VMD processor’s compute throughput and link bandwidth.

C. Map-Reduce

Not all workloads carry their own spatial structure. Map-reduce computations are a case in point [23]: a large data set is processed element-wise in parallel, and the results are

combined by an associative operator into a single result. Here the data has no intrinsic geometry, but the reduction is a tree, and the shape of that tree is ours to choose. In a VMD accelerator, the data can be distributed across a three-dimensional grid of VMDs, and the reduction performed dimension-wise as depicted in Fig. 3: first combining along X so that each row collapses to a single value, then along Y so that the remaining plane collapses to a line, and finally along Z to yield the final result. Every step in this schedule exchanges data only between physically adjacent VMDs, so as in Section III.B, interconnect length remains independent of problem size.

A grid of n^3 VMDs processing n^3 elements completes this reduction in $O(n)$ steps, against the $O(\log n)$ operations of a binary tree. But a binary tree over n^3 leaves cannot be embedded in three dimensions with bounded wire length. Signal propagation therefore imposes an $\Omega(n)$ latency regardless of logical depth. So the dimension-wise reduction meets this bound while using only nearest-neighbor links. We note that operations requiring a data-dependent shuffle, such as a group-by or a join, are not local in this way and fall outside the decomposition considered here.

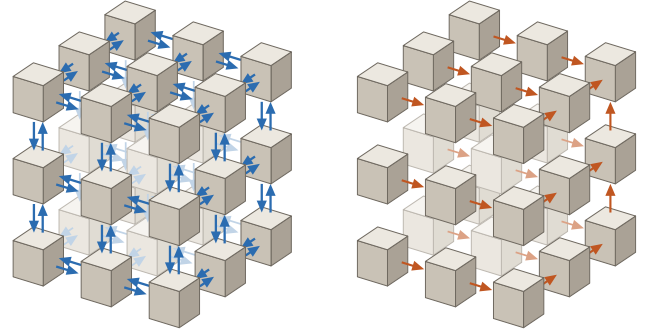


Fig. 3. Communication schemes on a VMD grid. Left: spatially decomposed physics simulation, where every pair of adjacent VMDs exchanges data in both directions each step. Right: dimension-wise reduction with a binary operation, where partial results flow along +X, then +Y, then +Z to produce a result in a single root VMD.

IV. DESIGN

A. Processor VMDs

Ultimately, VMD accelerators will be assembled from a modest set of specialized VMDs that implement various functional building blocks efficiently. For this work, we instead created VMDs using COTS processors, which can implement each of these functions using the same hardware but different firmware. Naturally, this approach introduces significant overheads in both cost and energy. However, it lets us prototype architectures and protocols and develop compilers. We use the results from this experiment to derive scaling laws and forecast the performance of specialized VMDs.

We based our processor VMDs on the Raspberry Pi RP2350 microcontroller [24]. The chip includes two 32-bit Arm Cortex-M33 cores, 520 KiB of SRAM, and an Arm single-precision floating point unit (FPU). The RP2354 variant also includes 2 MiB of QSPI Flash on a separate die in the

package. The programmable input/output (PIO) subsystems on the RP2350 can move a full 32-bit word onto or off of the device every clock cycle, meaning that any individual VMD based on this microcontroller need not be memory-bound for virtually any workload. The microcontroller runs at 150 MHz by default, but we found that overclocking to 300 MHz produced no adverse effects over weeks of testing at room temperature, with no additional heat-sinking. Raspberry Pi provides an extensive and well-documented software development kit for the device [25], which we used to develop our firmware. For our processor VMDs, we used the 60-pin RP2354A, which is commercially available for US\$ 1.00.

B. Routing for Processor VMDs

Before designing these generic processor VMDs, we consider how they will interact. For arbitrary data movement in three dimensions, each processor VMD needs at least six bi-directional data ports. Ultimately, power may be delivered on these same ports, but we currently assume that at least two dedicated ports for power delivery are required as well.

Using these assumptions, we present two possible routing strategies. The first assumes small, planar, square processor VMDs that interface at corners. They naturally provide eight contact points, which can map to the eight required ports. However, to distribute power through a VMD system arbitrar-

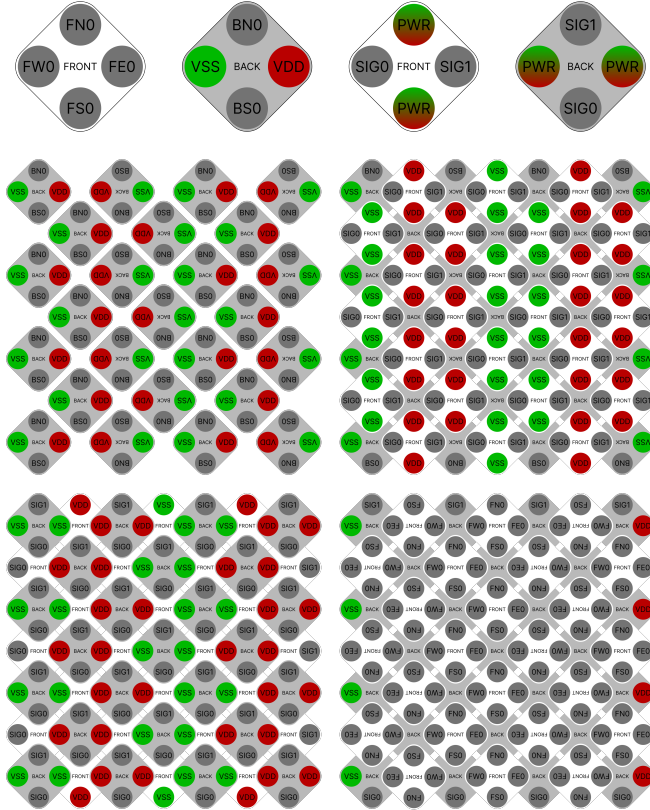


Fig. 4. Routing of small VMDs. First row: front and back of processor VMD and front and back of power routing VMD. Second row: layer $4n$ (compute) and layer $4n + 1$ (power routing). Third row: layer $4n + 2$ (power routing) and layer $4n + 3$ (compute). Power ports on power routing VMDs colorized for clarity.

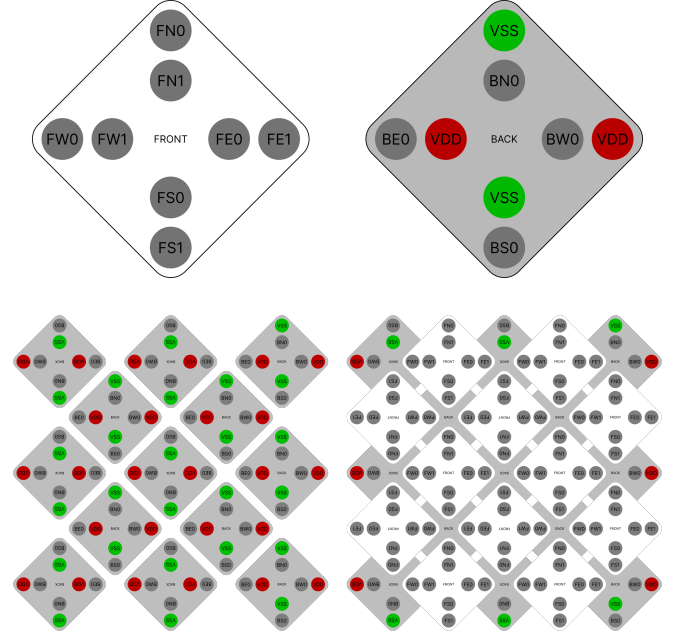


Fig. 5. Routing of larger VMDs. First row: front and back of processor VMD. Second row: layer $2n$ (compute) and layer $2n + 1$ (compute).

ily, each VMD needs to have (on average) two of each power port, not one. We address this by introducing “power routing” layers, consisting of passive VMDs which act as T-junctions for power. A diagram of this approach is shown in Fig. 4.

The second strategy assumes larger, planar, square processor VMDs, with two contacts on each side of each corner, for sixteen contact points in total. By dedicating four of these to power, we eliminate the need for power routing layers. Next, we dedicate two contacts to two additional communication ports, so that each VMD can communicate with the four VMDs in the layer below it as well as the four VMDs in the layer above it. We dedicated the last four additional contacts, one each, to four of the communication ports, providing double-bandwidth or full-duplex communication on one side of the VMD. A diagram of this approach is shown in Fig. 5.

For VMDs that are larger still, more contacts may be added for higher bandwidth. In such cases, many more strategies for routing exist.

C. Communication Protocol

Depending on the routing approach, some or all of the communication ports consist of a single contact. For arbitrary data movement in three dimensions, a half-duplex protocol is needed. We developed a half-duplex open-drain protocol for the RP2350 using the chip’s PIO subsystem, illustrated in Fig. 6. The protocol transmits 32-bit words, with 1 bit of turnaround and 1 bit of synchronization, for a rate of $\frac{16}{17}$.

D. VMD GEMV Multiplication

For GEMV multiplications, we structure the VMD accelerator as a systolic array. Consider the biased $m \times n$ GEMV multiplication defined in (1). To build a VMD accelerator

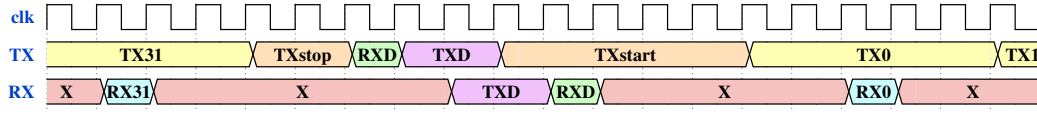


Fig. 6. Proposed half-duplex open-drain communication protocol. TXstart pulls the line(s) low. TX# pulls the line(s) low or leaves them floating depending on data. All other states, including TXend, leave the line(s) floating. Detection is performed during RXD and RX#. TXD detects data in the transmit FIFO and takes 2 cycles due to a peculiarity in the RP2350 PIO subsystem. In our processor VMDs, the clock as shown here runs at 150 MHz.

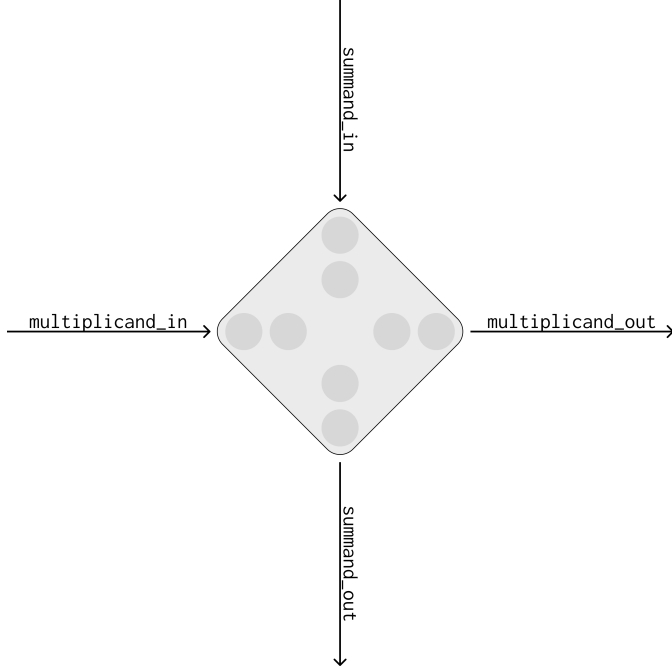


Fig. 7. A processor VMD for matrix-vector multiplication.

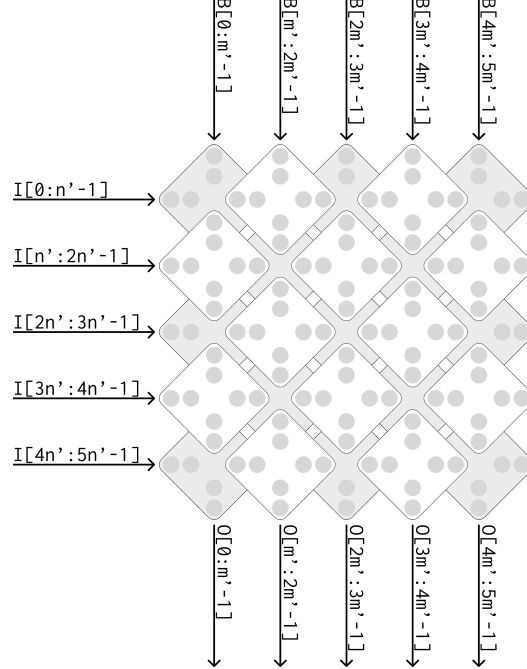


Fig. 8. An array of processor VMDs for matrix-vector multiplication.

for this operation, each VMD is programmed to perform the following operation:

```
summand_out = summand_in + weights @
multiplicand_in
multiplicand_out = multiplicand_in
```

where multiplicand_in and multiplicand_out are vectors of length n' , summand_in and summand_out are vectors of length m' , and weights is a pre-programmed sub-matrix of W of dimensions $m' \times n'$. The VMD's inputs and outputs are mapped as shown in Fig. 7.

Then, a two-dimensional $\begin{bmatrix} m' \\ m' \end{bmatrix}$ by $\begin{bmatrix} n' \\ n' \end{bmatrix}$ array of VMDs is constructed. This is shown in Fig. 8 for some $4m' < m \leq 5m'$ and $4n' < n \leq 5n'$. Note that the multiplicand_out in the eastern-most column is ignored, and summand_in in the northern-most row can be ignored if no bias is needed. Note also that each VMD is configured so that its inputs and outputs are correct regardless of its physical orientation.

Operation is inherently pipelined at the VMD level. So, this same VMD accelerator may also be used for general-element matrix-matrix multiplication (GEMM) operations by writing the input matrix in one column at a time and reading the output matrix out one column at a time. If the system design calls for processing an entire matrix at once, a different VMD accelerator may be compiled. One of the advantages of VMD accelerators is that volume and cost can be traded off with respect to speed as needed.

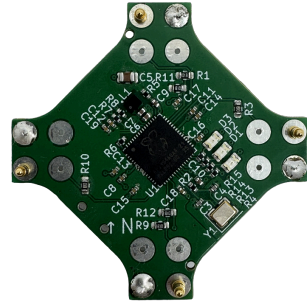


Fig. 9a. Front.

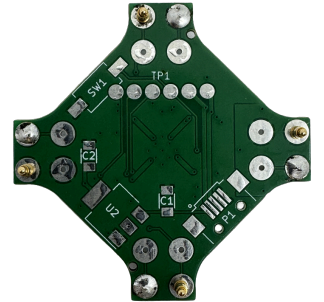


Fig. 9b. Back.

Fig. 9. A COTS processor VMD based on the RP2350.

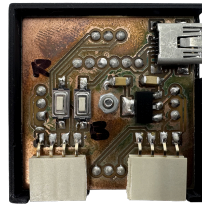


Fig. 10a. Bottom.



Fig. 10b. Top.

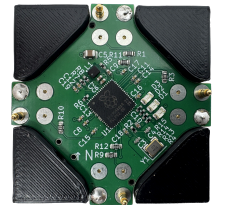


Fig. 10c. Loaded.

Fig. 10. A programmer for an RP2350-based processor VMD.

V. IMPLEMENTATION

A. Hardware

We designed COTS processor VMDs based on the RP2354A (Figure 9). We used pogo pins on the outer contacts, with landing targets on the inner contacts (note that the contacts are offset counter-clockwise to allow for through-hole assembly). We designed variants with both styles of routing discussed in Section IV.B. Only two layers were used, allowing for a thin (1 mm) printed circuit board (PCB). The contacts on the back are used for programming and testing (the programmer we designed is shown in Figure 10). Unpopulated footprints for a USB connector and voltage regulator are provided on the back side of the VMD for debug purposes only.

B. Firmware and Software

We wrote the firmware in an asynchronous manner, with two independent ring buffers for the summands and multiplicands. One core of the processor manages DMA and communications. The second core computes matrix-vector products as soon as both inputs are available. Outputs are written in place into the summand buffer. All events are logged in a compact event buffer.

We implemented the protocol described in Section IV.C using the RP2350's PIO subsystem, operating at 30 Mbps per contact. Without the use of any forward error correction, a 1,000,000-word test yielded a bit error rate of 0.

Additionally, we implemented an asynchronous serial interface, allowing for arbitrary reads and writes from and to memory. We also implemented a Python library that uses this interface to provide a full abstraction of the device, including telemetry, event logging, and explicit state machine control.

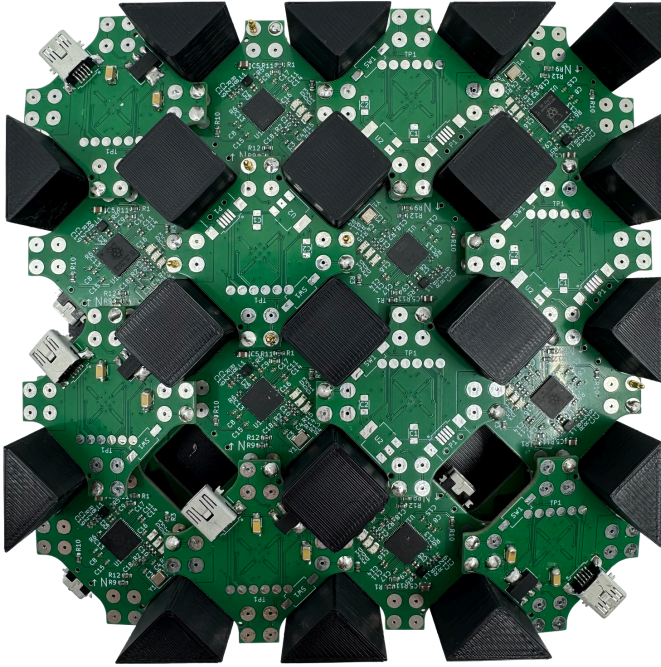


Fig. 11a. Top-down view.

Fig. 11. A four-by-four assembly of RP2350-based processor VMDs.

VI. RESULTS

A. Operation

We constructed a four-by-four array of RP2350-based processor VMDs to evaluate GEMV multiplication. We programmed sixteen processor VMDs as described in Section IV.D, with $m' = 256$ and $n' = 256$. An overall 1024×1024 weight matrix $\mathbf{W}$ was generated using NumPy (`np.random.rand(1024, 1024)`) and partitioned into sixteen 256×256 sub-matrices. Each was converted to the `binary32` format and programmed onto one of the sixteen VMDs. An overall 1024-element bias vector $\vec{B}$ was generated using NumPy (`np.random.rand(1024)`) and partitioned into four 256-element sub-vectors. Each was converted to the `binary32` format and programmed onto one of the four northern-most VMDs.

The assembled array of processor VMDs is shown in Figure 11. It was clamped down and connected to a host machine via USB. An overall 1024-element input vector $\vec{I}$ was generated using NumPy (`np.random.rand(1024)`). It was partitioned into four 256-element sub-vectors. Each was converted to the `binary32` format and uploaded to one of the western-most VMDs (using the Python library). From each of the southern-most VMDs, the 256-element output vector was downloaded (using the Python library). These four vectors were combined into the overall 1024-element output vector $\vec{O}$. The expected value of the output vector $\vec{O}$ was computed using NumPy (`O_expected = W @ I + B`) and compared to the value calculated by the VMD accelerator using NumPy (`np.isclose(O, O_expected)`). The values matched.

Then, 1,000,000 new random $\vec{I}$ vectors were generated, and the test was repeated for each without any failures.

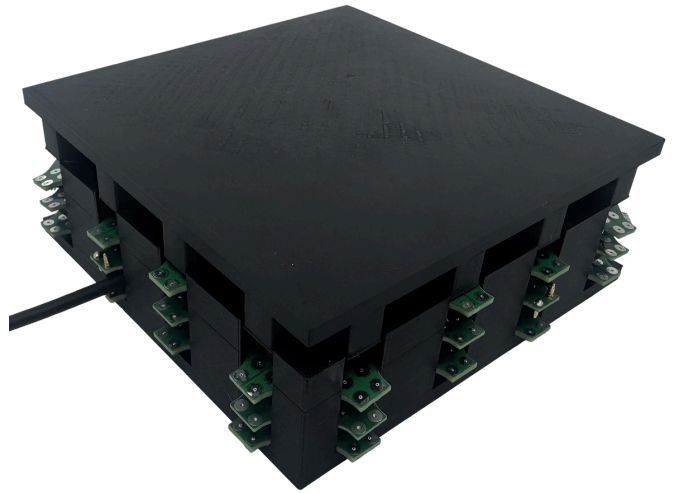


Fig. 11b. Fully assembled with cable.

TABLE I
POWER DISSIPATION OF RP2350 PROCESSOR VMD AT 150 MHz

Scenario	Power Dissipation
1 idle	74.9 mW
2 maximum compute	78.0 mW
3 maximum compute & proportional transmit	81.1 mW
4 maximum transmit	86.0 mW
5 maximum compute & maximum transmit	89.2 mW

B. Performance

Because input vectors are uploaded to the accelerator and output vectors are downloaded from the accelerator over the serial interface, performance is bottlenecked by the 12-Mbps USB Full-Speed peripheral provided on the RP2350. To validate raw performance of a single VMD, the firmware was modified so that all buffers were marked as containing valid input data at all times. Using this firmware at 150 MHz, the average processing time for a matrix-vector multiplication was found to be 2.17 ms. With $m' = 256$ and $n' = 256$, 131,072 operations are required for each such matrix-vector multiplication, yielding an overall per-VMD performance of 60.4 Mop/s. Running at 300 MHz, overall per-VMD performance was almost exactly double at 121.0 Mop/s. As discussed in Section V.B, this performance comes from a single Arm core. We also tested both cores simultaneously, using different firmware, and found performance nearly doubled. The performance of a full VMD accelerator scales with the number of VMDs. Furthermore, specialized VMDs are projected to have a much higher performance, as discussed in Section VII.

Note that while the contact-limited open-drain protocol described in Section IV.C does not take full advantage of the processor's input/output bandwidth, it is sufficient to saturate the above performance. Even with a single contact, transmitting at 30 Mbps with a data rate of $\frac{16}{17}$, each port is capable of moving 882,353 32-bit elements per second. With $m' = n' = 256$, this means a matrix-vector multiplication is possible every 0.290 ms, which is significantly less than the time it takes to complete a multiplication. To prove this, we modified the original firmware in a different way – instead of marking buffers as valid inputs at all times, we short-circuited the receive section of the state machine, exercising both the compute section and the transmit section. We found that computational performance remained at 60.4 Mop/s (with communication throughput for both multiplicands and summands at 3.8 Mbps, as expected).

C. Power

In the two performance tests, current from the 3.3 V supply was measured, and used to derive power dissipation. Three additional scenarios were also tested: the original firmware with no input (providing power when idle), modified firmware where all buffers were marked as containing valid output data at all times (providing power under maximum transmit) and modified firmware where some buffers were marked as containing valid input data at all times, some were marked as containing valid output data at all times, and the ring buffer

was prevented from advancing (providing power under both maximum compute and maximum transmit). All tests were performed at 150 MHz, and computational performance and communication throughput were observed to be nominal. The results are shown in Table I.

It is clear that much of the power dissipation is not a direct result of compute, but many of the sources of idle power dissipation may not be present during maximum compute. A lower bound on the energy efficiency of computation of 775 Mop/J can be derived from scenario 2. An upper bound of 19.4 Gop/J can be derived from the difference between scenarios 2 and 1.

Using the difference between scenarios 3 and 2, and noting that there are two active transmit channels, the energy efficiency of communication is 418 pJ/b. (Using scenarios 4 and 1 or scenarios 5 and 2 yields similar results.)

Finally, to derive the overall energy efficiency of the system, we use scenario 3:

$$\frac{60.4 \text{ Mop/s}}{81.1 \text{ mW}} = 745 \text{ Mop/J} \quad (2)$$

These numbers are explained by the overheads associated with COTS processors, and are expected to improve significantly for specialized VMDs.

VII. SCALING

While the cost and energy efficiency of the COTS RP2350-based processor VMDs is not competitive with modern high-performance computing, their development has allowed us to research and develop VMD accelerators, and to make projections about the cost and energy efficiency of accelerators based on specialized VMDs. In this section, we identify some of the major overheads associated with this prototype, including the high per-contact capacitance and outdated 40-nm fabrication process used for the RP2350 die. Using the relevant scaling laws, we show that accelerators based on specialized VMDs promise to be highly competitive.

A. Cost Efficiency

For a full VMD accelerator, assuming proper pipelining, the performance derived in Section VI.B scales with the number of VMDs. For the COTS processor VMDs, using one core per VMD at 300 MHz, the four-by-four array described above can perform at 1.94 Gop/s. A ten-by-forty-by-forty accelerator using these same COTS processor VMDs (costing approximately US\$ 32,000 including the PCB and supporting components) would perform at 1.94 Top/s.

At a similar price point, the NVIDIA H200 SXM performs at up to 52.67 Top/s based on in-house testing. However, a very high arithmetic intensity is required to achieve this performance. At lower arithmetic intensities, the performance of the H200 quickly degrades. For instance, at an arithmetic intensity of 2 operations per element, which is the upper limit for traditional GEMV multiplication, the H200 performs at 28.1 Gop/s based on in-house testing – 1/69 the projected performance of the large prototype VMD accelerator, and significantly less than the projected performance of an equivalent VMD accelerator based on specialized VMDs. Fig. 12 plots

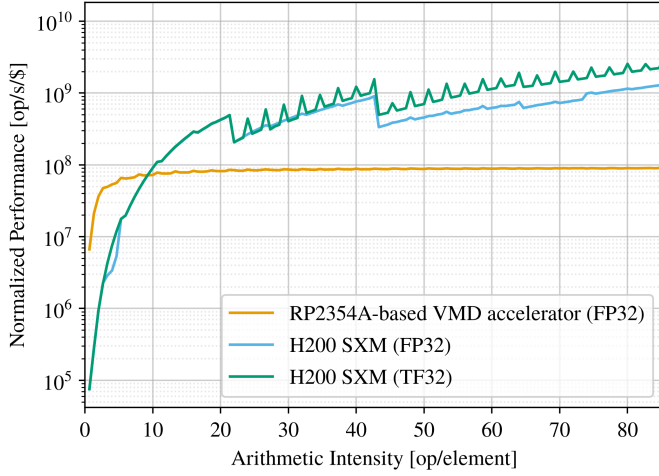


Fig. 12. Cost-Normalized Single-Precision Performance of an RP2350-based VMD accelerator and an NVIDIA H200 SXM, based on in-house testing.

this relationship, assuming US\$ 2 per VMD and US\$ 30,000 for an H200. (Note also the inconsistent performance of the H200; this is an effect of fixed systolic-array dimensions, and another class of inefficiencies which VMD accelerators avoid.)

The prototype VMDs are only competitive in the low-intensity regime partly because the RP2350 is fabricated using a 40 nm process, while the H200 is fabricated using a 4 nm process. Specialized VMDs, fabricated using a modern process, will show improved performance due to feature size scaling and innovations in device technology.

VMD accelerators based on specialized VMDs are also projected to be significantly more cost-effective not only compared to accelerators based on COTS parts, but also compared to traditional accelerators. Traditional accelerators require a substantial investment for each new variant or architecture. VMD accelerators only require investment for the development of the VMDs themselves. Once designed, variants and architectures simply require re-compilation (enabling machine-workload parity).

B. Energy Efficiency

Because of the various inherent overheads of the COTS processor VMDs, they are less energy efficient than specialized VMDs will be.

The first major overhead is again the fabrication process. When compared to the leader in the Green500 list from June 2026 [26], KAIROS, which achieves 73,282 Mop/J using 4 nm NVIDIA GH200 Grace Hopper Superchips, the RP2350-based VMD accelerator is approximately a hundred times less efficient. However, when compared to the leader in the Green500 list from June 2013 [27], Eurora, which achieves 3,208.8 Mop/J using 28 nm NVIDIA K20 GPUs (which is still a more advanced node than that of the RP2350), the RP2350-based VMD accelerator is only about one-fourth as efficient. This suggests that VMDs fabricated using a modern process will have an intra-VMD efficiency comparable to that of modern supercomputers.

In addition to being built using an outdated fabrication process, the RP2350 consumes significant power for func-



Fig. 13. Seven 64 mm-VMDs (top), seven 32 mm-VMDs (middle), and seven 16 mm-VMDs (bottom). VMDs with launches were de-embedded.

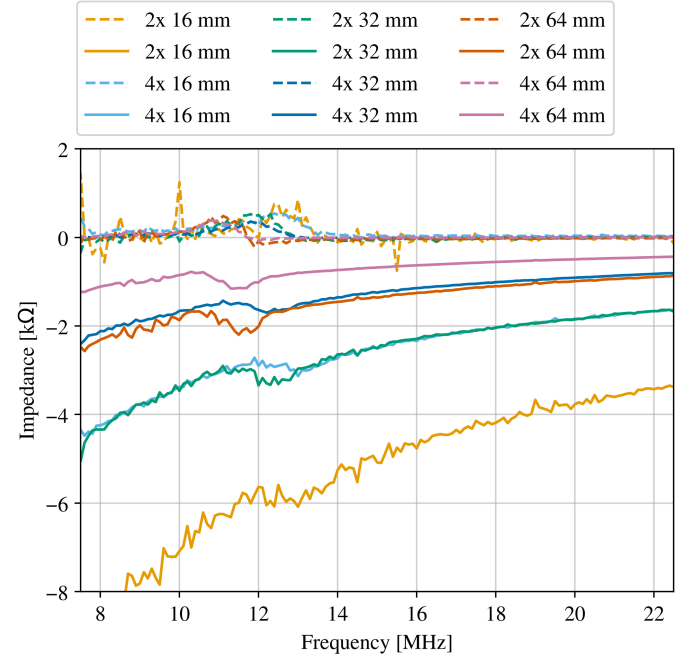


Fig. 14. Impedance of a series of VMDs of various sizes. Dashed lines are resistance ($\text{Re}(Z_{11})$). Solid lines are reactance ($\text{Im}(Z_{11})$).

tions other than computation and communication, as made evident by Table I. (Note that a completely idle RP2350 with no firmware running still consumes 52.7 mW.) Specialized VMDs will not require most of the supporting functional blocks present in microcontrollers, which will eliminate this large power overhead.

VMDs do inherently suffer from higher power draw for inter-VMD data movement, due to unavoidable capacitance on connectors. This is largely mitigated by the decreased data movement in VMD accelerators. Still, we characterized this capacitance for various counts of variously-sized planar VMDs by measuring the scattering parameters around the operating frequency. The test setup is shown in Fig. 13. The results were consistent with a linear shunt capacitance. Fig. 14 shows the equivalent Z_{11} impedances. At the lattice constant of the RP2350-based processor VMDs (32 mm), the capacitance per VMD per contact is 2.18 pF. At half that lattice constant (16 mm), the capacitance per VMD per contact is 1.08 pF. At double that lattice constant (64 mm), the capacitance per VMD per contact is 3.99 pF. This confirms that capacitance scales approximately linearly with lattice constant. So, for VMDs in a 1 mm-pitch lattice, the expected

capacitance per contact is 100 fF, comparable to the per-I/O capacitance of flip-chip interconnects [28].

Note that this inherent capacitance is not the primary driver for low communication energy efficiency in the prototype VMD accelerator. Since the pull-up voltage used in testing was 3.3 V and the data was random, the capacitance of the connectors explains 6.5 pJ/b of the power dissipation. The remaining 411 pJ/b is likely due to the substantial overhead stemming from the general-purpose nature of the microcontroller, such as routing and DMA. Like the idle power overhead, this type of overhead will not be present on specialized VMDs.

Another vestige of the on-board microcontroller is the high voltage used for inter-VMD communication. For small, specialized VMDs, core voltage closer to 1 V can be used. With a parasitic capacitance of 100 fF and a voltage of 1 V, the energy efficiency for communication would be 25 fJ/b.

Finally, VMD accelerators continue to benefit from the decreased data movement requirements inherent to all VMD systems. By mirroring the algorithms they execute, VMD accelerators have to move fewer bits, so the energy per bit is further amortized. Combining that with a modern fabrication process, the removal of unnecessary functional blocks, a smaller lattice pitch, and a lower communication voltage, VMD accelerators promise highly competitive energy efficiency.

VIII. CONCLUSION

Leveraging functional-level VMD technology for accelerators breaks the boundary between software and hardware, enabling rapid iteration on algorithms and ensuring they always run efficiently. In this work, we demonstrated an accelerator for a simple algorithm using VMDs based on COTS processors. We also showed how specialized VMDs, using a modern fabrication process, small physical size, and streamlined functionality can realize cost- and power-competitive accelerators for HPC. One such specialized VMD will be the systolic array building block used in our GEMV demonstration. Choosing the other members of the set of specialized VMDs is the subject of future work. Much as the Basic Linear Algebra Subprograms [29] enable a large number of workloads on general-purpose computers, the set of specialized VMDs will enable a large number of architectures in VMD systems, so care needs to be taken in choosing the scope and number of building blocks. Once the functions are chosen and implemented, VMD systems promise to unlock not only more cost-effective and energy-efficient computing, but a new paradigm for hardware-software codesign.

ACKNOWLEDGMENTS

This work was supported by ARIA's Nature Computes Better program, by the Schmidt Futures Bricks vs. Bonds project, and by the Center for Bits and Atoms consortium.

Anthropic's Claude Opus 4.7 was used to generate Figure 1, Figure 2a and Figure 3. Opus 4.7 was also used for a final proof-read for spelling, grammar, and style. Generative

artificial intelligence was not used in any other way for this work.

REFERENCES

- [1] Z. Fredin *et al.*, "Discrete Integrated Circuit Electronics (DICE)," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2020, pp. 1–8. doi: 10.1109/HPEC43674.2020.9286236.
- [2] E. Strand *et al.*, "From Surface to Volume Mount Devices (VMDs)," Manuscript submitted for publication, 2026.
- [3] K. Banerjee, S. J. Souri, P. Kapur, and K. C. Saraswat, "3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 602–633, 2001, doi: 10.1109/5.929647.
- [4] A. W. Topol *et al.*, "Three-dimensional integrated circuits," *IBM Journal of Research and Development*, vol. 50, no. 4.5, pp. 491–506, 2006, doi: 10.1147/rd.504.0491.
- [5] E. Beyne, "The 3-D Interconnect Technology Landscape," *IEEE Design & Test*, vol. 33, no. 3, pp. 8–20, 2016, doi: 10.1109/MDAT.2016.2544837.
- [6] T. Kazior, S. Woodruff, G. Jones, and I. Abdomerovic, "3D Heterogeneous Integration (3DHI): An Enabler For Next Generation RF Systems," in *2023 IEEE BiCMOS and Compound Semiconductor Integrated Circuits and Technology Symposium (BCICTS)*, Monterey, CA, USA: IEEE, Oct. 2023, pp. 1–4. doi: 10.1109/BCICTS4660.2023.10310934.
- [7] D. U. Lee, K. W. Kim, K. W. Kim, and others, "A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 432–433. doi: 10.1109/ISSCC.2014.6757501.
- [8] M. K. Sharma and B. Ramos-Alvarado, "Thermal management of 3-D heterogeneously integrated microelectronics: challenges and future research directions," *Communications Engineering*, vol. 5, no. 1, p. 28, Feb. 2026, doi: 10.1038/s44172-026-00590-y.
- [9] R. K. Jorgen, "Whatever happened to project tinkertoy?," *IEEE Spectrum*, vol. 24, no. 5, pp. 20–21, May 1987, doi: 10.1109/MSPEC.1987.6447916.
- [10] S. Naffziger *et al.*, "Pioneering Chiplet Technology and Design for the AMD EPYC and Ryzen Processor Families: Industrial Product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, 2021, pp. 57–70. doi: 10.1109/ISCA52012.2021.00014.
- [11] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007, doi: 10.1109/TCAD.2006.884574.
- [12] J. Backus, "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978, doi: 10.1145/359576.359579.
- [13] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory Devices and Applications for In-Memory Computing," *Nature Nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020, doi: 10.1038/s41565-020-0655-z.
- [14] G. Singh *et al.*, "Near-Memory Computing: Past, Present, and Future." Accessed: May 01, 2026. [Online]. Available: <http://arxiv.org/abs/1908.02640>
- [15] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017, doi: 10.1109/JSSC.2016.2616357.
- [16] N. Jouppi *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, Orlando FL USA: ACM, Jun. 2023, pp. 1–14. doi: 10.1145/3579371.3589350.
- [17] H. T. Kung, "Why Systolic Architectures?," *Computer*, vol. 15, no. 1, pp. 37–46, 1982, doi: 10.1109/MC.1982.1653825.
- [18] N. P. Jouppi, C. Young, N. Patil, D. Patterson, and others, "In-Datcenter Performance Analysis of a Tensor Processing Unit," in *Proceedings*

of the 44th Annual International Symposium on Computer Architecture (ISCA), ACM, 2017, pp. 1–12. doi: 10.1145/3079856.3080246.

- [19] M. Horowitz, “1.1 Computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, San Francisco, CA, USA: IEEE, Feb. 2014, pp. 10–14. doi: 10.1109/ISSCC.2014.6757323.
- [20] A. Vaswani *et al.*, “Attention Is All You Need.” Accessed: Mar. 10, 2026. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [21] S. Plimpton, “Fast Parallel Algorithms for Short-Range Molecular Dynamics,” *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995, doi: <https://doi.org/10.1006/jcph.1995.1039>.
- [22] D. E. Shaw *et al.*, “Anton, a special-purpose machine for molecular dynamics simulation,” *Commun. ACM*, vol. 51, no. 7, pp. 91–97, Jul. 2008, doi: 10.1145/1364782.1364802.
- [23] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008, doi: 10.1145/1327452.1327492.
- [24] Raspberry Pi Ltd, “RP2350 Datasheet.” 2025.
- [25] Raspberry Pi Ltd, “Raspberry Pi Pico SDK.” 2026.
- [26] TOP500.org, “Green500 List – June 2026.” Accessed: Jul. 14, 2026. [Online]. Available: <https://www.top500.org/lists/green500/green500-june-2026/>
- [27] TOP500.org, “Green500 List – June 2013.” Accessed: Jul. 14, 2026. [Online]. Available: <https://www.top500.org/lists/green500/2013/06/>
- [28] B. Dehlaghi, R. Beerkens, D. Tonietto, and A. Chan Carusone, “Interconnect Technologies for Terabit-per-Second Die-to-Die Interfaces,” in *2016 IEEE Compound Semiconductor Integrated Circuit Symposium (CSICS)*, IEEE, 2016, pp. 1–4. doi: 10.1109/CSICS.2016.7751049.
- [29] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, “Basic Linear Algebra Subprograms for Fortran Usage,” *ACM Transactions on Mathematical Software*, vol. 5, no. 3, pp. 308–323, 1979, doi: 10.1145/355841.355847.