Equicore: Accelerating Clebsch-Gordan Tensor Product of Equivariant Neural Networks on FPGA

Shidi Tang[1], Chuanzhao Zhang[1], Ruiqi Chen[2], Yuxuan Lv[1], Bruno da Silva[2], Ming Ling[1,*]

[1]School of Integrated Circuits, Southeast University, [2]ETRO, Vrije Universiteit Brussel

[*]Corresponding author: trio@seu.edu.cn

***Abstract*—Equivariant neural networks (ENNs) are a powerful framework for modeling 3D geometric data in physical and biological systems. The Clebsch–Gordan tensor product (CGTP)—a core operation for preserving equivariance—remains the primary computational bottleneck in ENNs. Although Clebsch–Gordan (CG) coefficients exhibit pronounced structural sparsity, prior work has neither fully leveraged this property nor adopted hardware-friendly quantization, leading to limited efficiency. We present *Equicore*, a software–hardware co-design framework to accelerate CGTP in ENNs. *Equicore* introduces three key innovations: (1) a sparse-bypass strategy that exploits the CG structural sparsity together with a novel CG data format to pack the overlapping non-zeros, bypassing redundant data accesses and computations comparing to previous sparse solutions; (2) a merged-shift quantization strategy that enables full Int8 representation of irreps, weights, and CG coefficients using shift-only operations; and (3) a cascaded processing unit that tightly couples the FPGA hardware resources to achieve high operating frequency while supporting efficient sparse and quantized computation. Deployed on a AMD Virtex VCU128 platform, *Equicore* delivers up to 10.5× speedup and 17.4× energy-efficiency improvement over state-of-the-art GPU libraries and FPGA designs across diverse CGTP types in a benchmark of eleven ENN models.**

I. Introduction

Recently, equivariant neural networks (ENNs) have demonstrated significant success in modeling 3D geometric data across multiple applications including molecular docking [1], [2], force field modeling [3]–[5], physical dynamics simulation [6]. While conventional neural networks primarily handle 1D sequences (e.g., natural language processing) and 2D matrices (e.g., image processing), 3D geometric structures (e.g., molecular systems, point clouds) present fundamental challenges due to the curse of dimensionality [7]. ENNs address this through equivariance - a core geometric deep learning principle [8] that guarantees predictable output transformations under input space transformations [9].

The key operation in ENNs is the Clebsch-Gordan tensor product (CGTP) [9], which transforms two input feature vectors (irreducible representations or irreps) into an output feature vector. Unlike conventional multi-dimensional tensor computations (e.g. general matrix multiplication, GEMM), CGTP is grounded in representation theory and utilizes Clebsch-Gordan (CG) coefficients [10] to preserve equivariance. However, CGTP exhibits $\mathcal{O}(l^6)$ computational complexity for irreps of order l, making it significantly computational intensive [9], [11]–[13].

Previous works, such as CuEquivariance [14], provides an efficient CUDA library for accelerating CGTP operations on GPUs. OpenEquivariance [15] further exploits CG sparsity to reduce redundant computations using customized JIT-compiled CUDA kernels. However, the achieved acceleration is far below the theoretical limit, reaching only $2.9\times$ even at sparsity ratio of 85.7%, when compared to non-sparse operations. This limitation arises because CGTP is a specialized operation, distinct from common workloads like GEMM in modern AI models, which hinders the use of efficient computing units, such as Tensor Cores, on contemporary GPUs. Additionally, Modeling [16] uses SystemC for modeling the CGTP FPGA acceleration, but it it limited to high-level abstraction, lacking detailed description of underlying hardware. DiffDock-FPGA [17] applies model quantization to compress CGTP by converting irreps and weights to lower-precision data types. However, their approach merely supports six types of CGTP and neglects the quantization of CG coefficients, which are essential for CGTP. Moreover, their quantization method is not optimized for hardware, as the quantization and dequantization operations introduce additional hardware overhead, complicating the design of the computing unit and leading to timing issues at high computational frequencies. We compare afore-mentioned existing approaches, plus the baseline CGTP library e3nn [18], with our approach in Table I. Our *Equicore* fully exploits sparsity and quantization while maintaining high flexibility to support various CGTP types.

We thoroughly analyze the CGTP computation flow and identify the following challenges:

Challenge 1: **Discrepancy between theoretical and actual acceleration by leveraging the structural sparsity of CG coefficients.** Existing solutions for exploiting CG structural sparsity on GPUs fail to deliver theoretical speedups due to the inability to customize the underlying hardware.

Challenge 2: **Full quantization while maintaining hardware compatibility and model accuracy.** Existing quantization solutions are inefficient due to the full precision representations of the CG coefficients and the scale factor, limiting the hardware efficiency.

Challenge 3: **Efficient hardware with high computational frequency.** The overhead to handle the logic data paths for sparse computation, quantization, and intermediate storage introduces significant timing issues, which in turn constrains the system's maximum clock frequency.

To address these challenges, we propose *Equicore*, a software-hardware co-design solution to systematically accelerate CGTP operations in ENNs. Our contributions include:

- We propose the sparse bypass strategy (SBS) to efficiently exploit the structural sparsity of CG coefficients. SBS reduces redundant computation by utilizing the non-zero elements of CG coefficients to directly fetch correspond-

This work is supported in part by the National Natural Science Foundation of China under Grants 92464301. This research work is also supported by the Big Data Computing Center of Southeast University.

978-3-9826741-1-7/DATE26/© 2026 EDAA

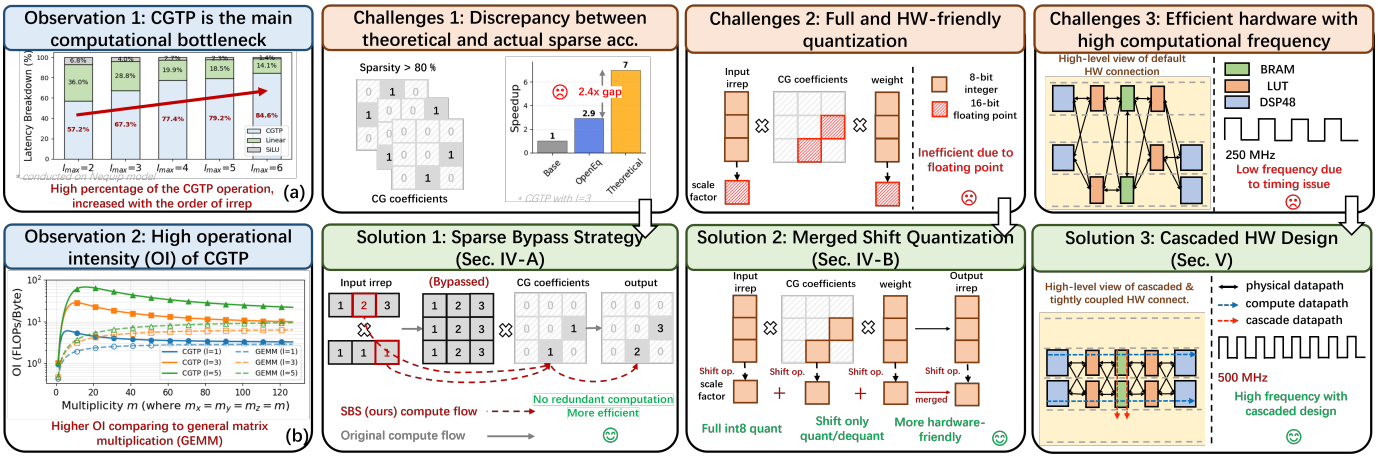

Fig. 1: Challenges and contributions of accelerating the tensor product operations in equivariant neural network.

TABLE I: A taxonomy for comparing existing tensor product accelerations

Works	Platform	Sparsity	Quantization	Flexibility
e3nn [18]	GPU	✗	✗	High
CuEquivariance [14]	GPU	✓	✗	High
OpenEquivariance [15]	GPU	✓	✗	High
DiffDock-FPGA [17]	FPGA	✗	✓	Low
Equicore (Ours)	FPGA	✓	✓	High

ing elements from the input irreps. Additionally, a reorganized CG format is introduced to eliminate redundant computations when overlapping non-zero elements exist along the third dimension of CG coefficients.

- We introduce the merged shift quantization (MSQ) method, which fully supports Int8 quantization for all data (input/output irreps, weights, CG coefficients). This method preserves model accuracy with fine-grained element-wise CG quantization while being hardware-friendly due to its simple shift operations compared to previous approaches.
- We present the tensor product processing unit (*Equicore*), which efficiently supports sparse computation with the reorganized CG format and the hardware friendly quantization. It also integrates a cascaded design that tightly couple the logic with RAM and DSP resources, simplifying the logic data paths, boosting the core computation frequency to 500 MHz.
- Benchmarks on various ENNs with different CGTP types demonstrate that *Equicore* achieves up to 10.5× speedup and 17.4× energy-efficiency comparing to SOTA GPU libraries and FPGA designs.

II. Background

A. Equivariant Neural Networks

ENN [12], [15], [19] generally refers to a class of neural networks whose output is predictable when the input undergoes transformations under a specific group G. Formally, for a given model $f : X \to Y$ that maps the input vector space X to the output vector space Y, f is defined as equivariant under group G if, for any transformation $g \in G$, the following condition holds: $f(D_X(g)x) = D_Y(g)f(x), \forall g \in G, x \in X$, where $D_X(g)$ and $D_Y(g)$ are the group representations of g acting on the vector spaces X and Y, respectively. Prevailing ENNs typically use the $O(3)$ group (i.e., translation, rotation, reflection), which allows the group representation $D(g)$ to be block-diagonalized into a canonical form [9]. For simplicity, $D(g)$ is generally represented as:

$$D(g) \cong m_1 \times l_1 p_1 + m_2 \times l_2 p_2 + \dots \quad (1)$$

where $m, l \in \{0, 1, 2, \dots\}$ represent the multiplicity and the order. $p \in \{e, o\}$ represents the parity [9] used to enforce reflection equivariance. **Each term (e.g., $m_1 \times l_1 p_1$) is called the order-l irreducible representation (*irrep*).**

B. Clebsch-Gordan Tensor Product

Prevailing ENNs [1], [4], [5], [11] employ the Clebsch-Gordan tensor product (CGTP) to enforce the equivariant property of the model. The CGTP operation takes the order-l_x irrep I_x and order-l_y irrep I_y, along with the Clebsch-Gordan coefficient [10] $CG \in \mathbb{R}^{(2l_x+1)\times(2l_y+1)\times(2l_z+1)}$ and the learnable weight W, to compute the output order-l_z irrep I_z:

$$I_z = CGTP(I_x, I_y, CG, W) \quad (2)$$

Note that the term `tensor product` in this work is distinct from the typical multiplications (e.g., GEMM) of multi-dimensional array. The detailed mathematical formulation of CGTP can be referred to previous works [9], [16]. In this work, we focus on the computation dataflow of CGTP, illustrated in Figure 2. For instance, the input irreps $I_x = 16 \times 1o$ and $I_y = 16 \times 1e$ both have matrix shapes of (16×3), and the output irrep $I_z = 32 \times 1o$ has a tensor shape of (32×3). Step ❶: The outer product is performed across all rows of two tensors, producing 256 tensors of shape (3×3). Step ❷: The Hadamard product is applied to each tensor with the CG coefficients, which is a 3-dimensional constant tensor with shape $(3 \times 3 \times 3)$, determined by l_x, l_y, l_z. Step ❸: The results are summed across the first two CG dimensions to produce a (3×1) vector, which is then multiplied by the learned weight. The results from all outer product tensors are summed to produce the partial (3×1) output vector. The output vectors multiplied by different weights are gathered to produce the final output irrep with a shape of (3×32).

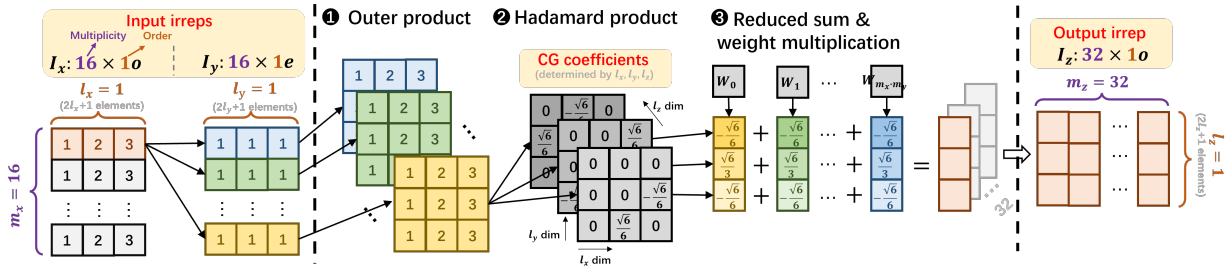

Fig. 2: Clebsch-Gordan tensor product (CGTP) computation flow, as exampled by input irreps $I_x = 16 \times 1o, I_y = 16 \times 1e$, output irrep $I_z = 32 \times 1o$.

C. Quantization

Model quantization transforms the high-bit number into its low-bit (typically integer) representation through the following affine transformation $x_{int} = \text{clip}(\lceil \frac{x_{fp}}{s} - z \rfloor, c_{max}, c_{min})$, where s and z are the scale factor and zero point. $\text{clip}(\cdot, c_{max}, c_{min})$ is the clip function that truncates the value to the range $[c_{max}, c_{min}]$ where c_{max} (c_{min}) indicates the maximum (minimum) integer. $\lceil \cdot \rfloor$ is the rounding function that rounds the value to the nearest integer. And the quantized integer can be recovered (dequantized) with $x_{fp} = s \times (x_{int} + z)$. The symmetric quantization [20] calculates the scaling factor $s = \frac{2 \times max(|x_{max}|, |x_{min}|)}{c_{max} - c_{min}}$ and $z = 0$ for signed integer. x_{max} and x_{min} represents the maximum and the minimum values of the floating number. Typically, symmetric quantization is more hardware efficient than asymmetric quantization since zero point is eliminated, and the scale factors are represented in the power-of-two form $\hat{s}$:

$$\hat{s} = 2^n, n = ceil(log_2 s) \tag{3}$$

so that the multiplication (division) can be implemented with bit-shift operation, which is more hardware-friendly.

III. Observations and Motivations

Bottleneck and Operational Intensity. First, we analyze the computation bottleneck of prevailing ENNs in Figure 1 (a). For example, in the Nequip [4] model, the CGTP operations account for most of the latency, ranging from 57.2% to 84.6% as the maximum CGTP order l_{max} increases. This observation highlights that CGTP is the main bottleneck in accelerating ENNs, especially for higher-order cases. Next, we analyze the operational intensity (OI, FLOPs/Byte) of CGTP, which is defined as the number of floating point operations per Byte of data loaded from memory, as shown in Figure 1 (b). We also compare CGTP with general matrix multiplication (GEMM) [21]. For a fair comparison, the two input matrices of GEMM are set to the same shape as the two CGTP irreps. The results indicate that the OI of CGTP increases with the order l and converges as the multiplicity m increases. Moreover, the OI of CGTP is consistently higher than that of GEMM, which motivates us to improve the computational capability of our acceleration solution.

Structural Sparsity of CG Coefficients. The CG coefficients exhibit structural sparsity [22]–[24] due to the underlying symmetries and selection rules [25]. Since CG coefficients are determined by the orders of the input and output (l_x, l_y, l_z), we analyze this sparsity under different irrep orders in Figure 3 (a). The CG coefficients show high sparsity ($> 80\%$) across most irrep orders. These observations highlight the significant potential for accelerating CGTP operations by exploiting CG sparsity.

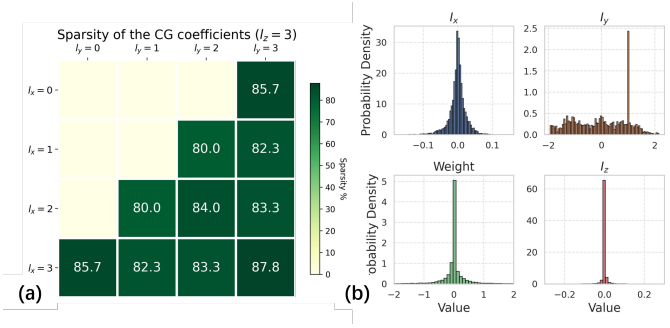

Fig. 3: (a) Sparsity ratio of the CG coefficients under different irrep orders. CG is valid only when $|l_x - l_y| \leq l_z \leq |l_x + l_y|$ [9]. (b) Data distribution of the CGTP input irrep I_x, I_y, output irrep I_z and the weight.

Symmetry Distribution of Irreps and Weights. To balance quantization accuracy and computational efficiency, we analyze the data distribution of the input/output irreps and the weight of the CGTP operations. The results of Nequip [4] model in Figure 3 (b) show that both input/output irrep and weights are evenly distributed around zero. Therefore, we can apply symmetric quantization to both, simplifying quantization/dequantization to a simple multiplication with the scale factor.

IV. Methods

A. Sparse Bypass Strategy

Figure 4 (a) illustrates the proposed sparse bypass strategy. Instead of performing the full Hadamard product between the outer product results and the CG coefficients, we only consider the non-zero elements for computation. For instance, only the elements within the colored boxes are included, while the zero elements in the CG coefficients and the corresponding elements in the outer product results are excluded. Furthermore, we observe that the elements of the outer product originate from the two input irreps I_x and I_y, which indicates that the outer product can be bypassed, allowing the input irreps to directly interact with the non-zero CG coefficients, as shown by the red dashed lines.

This strategy requires careful dataflow design, as the elements from I_x and I_y depend on the positions of the non-zero elements in the CG coefficients. The previous method, OpenEquivariant [15], exploits the parallelism of non-zero CG elements. However, this strategy introduces redundant operations when two (or more) values overlap along the l_z

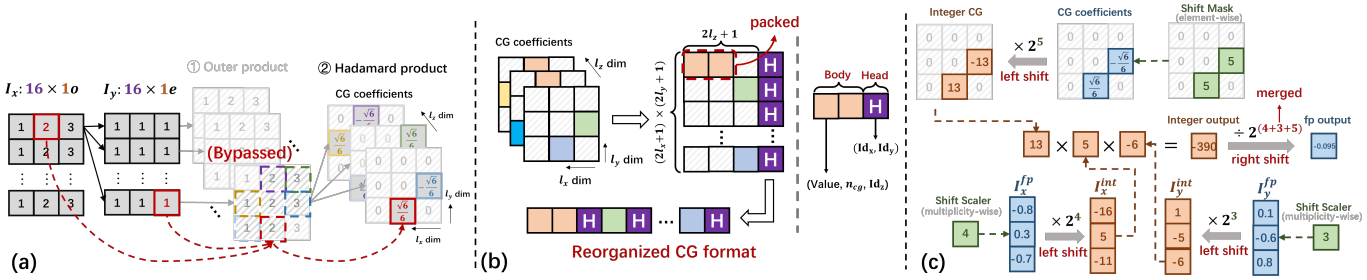

Fig. 4: Illustration of the proposed (a) sparse bypass strategy, (b) the reorganized CG coefficients format, and (c) merged shift quantization.

dimension of the CG coefficients. For example, in Figure 4 (a), the red-boxed value 2 from the second column of I_x and the value 1 from the third column of I_y are determined by the position (i.e., the third row and second column) of the red-boxed value $\frac{\sqrt{6}}{6}$ in the first l_z dimension of the CG coefficients. When another CG non-zero element exists in the second l_z dimension at the same position as the red-boxed one, we would need to fetch and compute the same values 2 and 1 twice, causing redundant data movement and computation.

To address this issue in OpenEquivariant [15], we introduce a reorganized CG coefficients format that packs the overlapped values to allow bypassing the multiplication of I_x and I_y. Instead, we directly use the previous result and multiply it with the new CG value. The proposed format is illustrated in Figure 4 (b), where the original 3-dimensional CG coefficients are reorganized along the l_z dimension to pack the overlapped non-zero elements together. We extract the indexes along the l_x dimension (Id_x) and the l_y dimension (Id_y) as the *Head*, since the overlapped elements share the same Id_x and Id_y. The non-zero CG data (Value), the number of shifts (n_{cg} for quantization, details in Section IV-B), and the index along the l_z dimension (Id_z) are extracted as the *Body* for each non-zero element. Our hardware design leverages this CG format to efficiently avoid redundant computation, with further details provided in Section V.

B. Merged Shift Quantization

The proposed merged shift quantization (MSQ) is illustrated in Figure 4 and formalized by:

$$\begin{aligned} & I_x^{fp} \times I_y^{fp} \times CG^{fp} \\ &= (I_x^{int} \times s_x) \times (I_y^{int} \times s_y) \times (CG^{int} \times s_{cg}) \\ &= (I_x^{int} << n_x) \times (I_y^{int} << n_y) \times (CG^{int} << n_{cg}) \\ &= \underbrace{(I_x^{int} \times I_y^{int} \times CG^{int})}_{\text{integer mul}} << \underbrace{(n_x + n_y + n_{cg})}_{\text{merged}} \end{aligned} \tag{4}$$

where s_x, s_y, and s_{cg} are the scale factors for the input irreps I_x, I_y, and the CG coefficients CG, respectively. n_x, n_y, and n_{cg} represent the number of shifts derived from the power-of-two scale factors, as given by Equation 3. For example, in Figure 4 (c), the floating-point irreps I_x^{fp} and I_y^{fp} are shifted according to the shift scaler n_x and n_y, and then rounded into signed integers I_x^{int} and I_y^{int}. We use multiplicity-wise shift scaler to ensure that the shifts are independent across the multiplicity dimension, thereby achieving lower quantization error.

However, traditional symmetric quantization introduces large quantization errors when applied to the CG coefficients. This is due to the extremely wide value range of the CG matrix, which causes greater rounding errors for smaller values and leads to accuracy degradation. To address this, we propose a simple yet effective method to find the optimal number of shifts that minimizes the quantization error by solving the following minimization problem:

$$\underset{n_{cg} \in [n_{\min}, n_{\max}]}{\arg\min} \left| \mathrm{clip}\left(\lceil cg \times 2^{n_{cg}} \rceil \right) \times 2^{-n_{cg}} - cg \right| \tag{5}$$

where cg is the non-zero element of the CG matrix, and $n_{\min}$ and $n_{\max}$ define the minimum and maximum range of n_{cg}, respectively. The shift mask in Figure 4 (c) contains the optimal n_{cg} for each non-zero element of the CG matrix.

Once all the irreps and CG coefficients are converted to their integer representations, we can perform the computation using efficient integer arithmetic (e.g., MUL). The advantage of using a power-of-two scale factor is that these scale factors can be seamlessly merged, requiring only a single shift operation to dequantize the integer result back to its full precision type.

V. Hardware Architecture

A. System Overview

As shown in Figure 5 (a), the system consists the high-bandwidth memory (HBM), the BRAM buffer for storing the on-chip input/output irrep, weight, the reorganized CG coefficients, and several *Equicore* units connected by the memory controller. On the runtime, the input irrep and weight data is transferred from HBM to on-chip buffer in 16-bit full precision datatype together with its 3-bit number of shift. Before computation in the *Equicore* unit, these data is quantized into its Int8 representation by shift operations and truncation. The CG coefficients are quantized into Int8 datatype offline since they are constant numbers, so they are loaded in Int8 datatype together with its 3-bit number of shift. After computation in the *Equicore* units, the output irrep data is transferred back to the HBM.

B. Equicore Design

As illustrated in Figure 5 (b), the *Equicore* module is responsible for performing the CGTP operations. *Equicore* loads the reorganized CG format and decodes the *Head* and *Body* into three 8-bit indexes (id_x, id_y, id_z), one 8-bit integer CG value ($Value$) and one 3-bit number of shift. id_x, id_y indicate the addresses to fetch data from input irreps I_x and

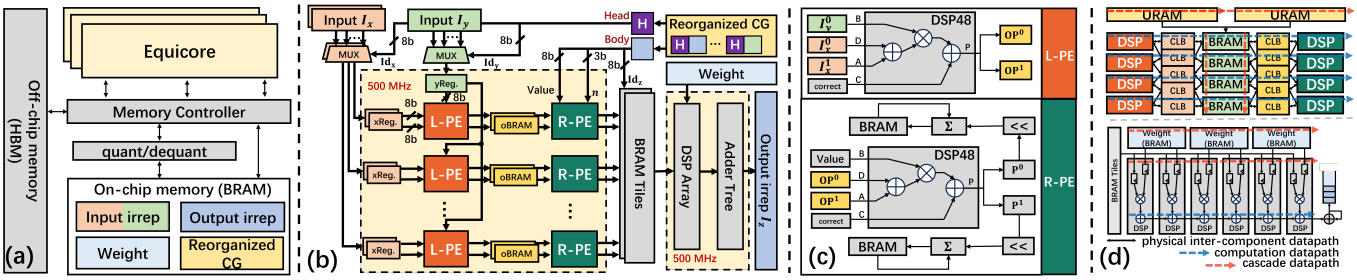

Fig. 5: (a) System architecture. Micro-architecture of (b) the *Equicore* unit to perform CGTP operations, (c) the L-PE and R-PE modules, and (d) tightly coupled cascaded design of the L-PE&R-PE modules (top) and the DSP array (bottom) to boost the computation frequency.

I_y. Since the reduced sum results in Step ❸ are independent along the l_z dimension, we use id_z to address BRAM tiles for storing these results. The input irrep I_x is loaded from HBM to on-chip BRAM with multiple (e.g. 4) rows in the multiplicity dimension, while input irrep I_y is loaded with single row. This design choice is based on our observation that most of the ENNs only have one multiplicity for the irrep I_y. The indexes id_x and id_y select the corresponding elements that are involved into the computation, these elements are computed in the L-PE module, with the results stored in the output BRAM (oBRAM) to be computed in the R-PE module with the CG coefficients. If encountered an overlapped CG coefficient, the temporary results in the oBRAM will be reused, bypassing the computation in the L-PE. The results from R-PE will be multiplied with weights in a DSP array followed by an adder tree to sum up all the results, producing the final output irrep I_z.

C. PE design

The L-PE and R-PE modules are illustrated in Figure 5 (c). These modules leverage the DSP48 resources to efficiently support Int8 multiplications with DSP packing technic [26]. The L-PE is responsible for computing the product of two input irreps. We feed two I_x elements (I_x^0, I_x^1) from different multiplicity dimensions at port D and port A of DSP48, and we feed one I_y^0 element at the port B. The calculated results at port P can be divided into two results representing $op^0 = I_x^0 \times I_y^0$ and $op^1 = I_x^1 \times I_y^0$. Similarly, the R-PE module feeds the calculated op^0 and op^1 into port D and port A of DSP48, and feeds the CG value at port B. The computing results p^0 and p^1 are shifted (dequantized) into the full precision according to n_s and cumulatively summed into the BRAM, which is addressed by id_z from the BRAM tiles. The correction logic for recovering corrupted bits in P port when packing two signed 8-bit integers is added to C port, similar to previous work [27].

D. Cascaded Design

Figure 5 (d) illustrates our approach, which exploits UltraScale FPGA device characteristics to reduce logic routing overhead and realize a frequency-doubled compute array with error-corrected DSP packing. While prior cascade designs supported DSP arrays over 500 MHz [28], [29], the growing complexity of external computation and the adoption of DSP-packing techniques make cascades of DSP, BRAM, and URAM impractical [30]. To overcome this limitation, we incorporate the closest multiple CLB columns between DSP and BRAM as part of the L-PE and R-PE modules in a tightly coupled fashion. This design connects each FF directly to DSP inputs/outputs and reduces the processing footprint to fewer than 16 LUTs, which are control logic for sparse and quantized computation.

TABLE II: Resource utilization on VCU128 FPGA Board

Resource	LUT	FF	BRAM	URAM	DSP
Used	183,306	358,121	1,796.0	384.0	3,840
Available	1,303,680	2,607,360	2,016	960	9,024
Utilization	14.1%	13.7%	89.1%	40.0%	42.55%

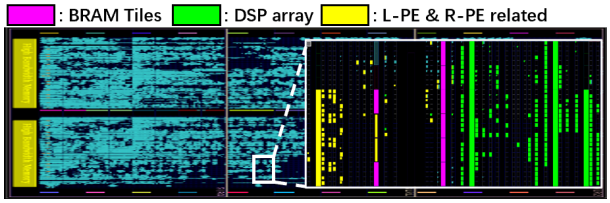

Fig. 6: Implementation layout on VCU128.

VI. Experiments

A. Experiment Setup

Models and Datasets. For algorithm evaluation of our merged shift quantization, we choose Nequip [4] models with typical setting $l_{max} = 3$. We use three public datasets (Aspirin, Benzene, Toluene) from sGDML [31]. For hardware evaluation of *Equicore*, we follow the OpenEquivariance [15] benchmark, which includes CGTP types with varying irrep multiplicities and orders from several ENN models (Nequip, MACE [11], DiffDock [1], tetris-poly [18]), as described in their original papers. We use the batch size 3000 to accommodate the inference scenario of typical ENNs.

Algorithm and Hardware Implementations. For algorithm, we follow Nequip [4] and use the energy MAE (meV), and force MAE (meV/Å) of molecular systems to measure the accuracy on Nequip Pytorch model. For each dataset, we randomly select 20% of the test samples as a calibration set to compute the scale factors for irreps and weights. We set $n_{max} = 7$ and $n_{min} = 0$ of Equation 5 to balance storage overhead and quantization accuracy. For hardware, each *Equicore* contains 4 L-PE, 4 R-PE and one DSP array with 32 DSPs according to our DSE results to balance the latency of pipeline stages. The whole design is implemented on an AMD Virtex Ultrascal+ VCU128 FPGA using Verilog and synthesized with core at 500 MHz and peripheral logic at 250 MHz. Power consumption is estimated with the Xilinx Power Estimator (XPE) tool, and hardware resource utilization (Table

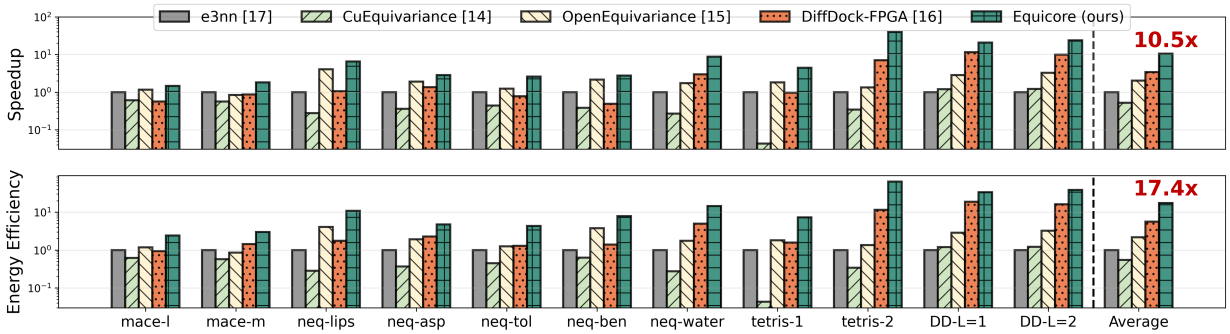

Fig. 7: Speedup and energy efficiency comparison of our *Equicore* with GPU-based and FPGA-based works. Model notations: `mace`: MACE [11], `neq`: Nequip [4], `DD`: DiffDock [1], `tetris`: tris-poly [18].

TABLE III: Quantization accuracy comparison

Method	Dataset	Irrep	Weights	CG	HW friendly	Force MAE (meV/Å) ↓	Energy MAE (meV) ↓
Full Precision		fp32	fp32	fp32	low	0.038	0.274
Vanilla*	aspirin	int8	int8	fp32	medium	0.338	0.8826
Ours		**int8**	**int8**	**int8**	**high**	**0.309**	**0.8180**
Full Precision		fp32	fp32	fp32	low	0.007	0.014
Vanilla*	benzene	int8	int8	fp32	medium	0.069	0.4363
Ours		**int8**	**int8**	**int8**	**high**	**0.064**	**0.2822**
Full Precision		fp32	fp32	fp32	low	0.012	0.058
Vanilla*	toluene	int8	int8	fp32	medium	0.144	**0.1696**
Ours		**int8**	**int8**	**int8**	**high**	**0.096**	0.2653

* Vanilla quantization adapted from DiffDock-FPGA [17]

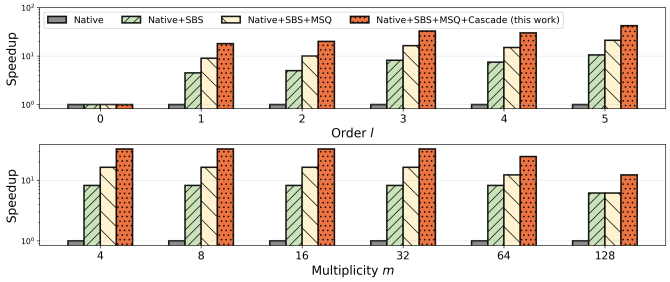

Fig. 8: Ablation study with different orders l (m fixed to 8) and multiplicities m (l fixed to 3).

II) and implementation (Fig. 6) are obtained after synthesis and implementation in Vivado 2024.1. Hardware performance is assessed in terms of speedup and energy efficiency relative to other hardware designs.

Baselines. We compare *Equicore* with three GPU-based (NVIDIA V100 32GB) libraries and one FPGA-based (re-implemented in AMD VCU128) design: ❶ **e3nn** [18]. The basic Pytorch implementation on GPU. ❷ **CuEquivariance** [14]. The NVIDIA official library for accelerating tensor product through efficient CUDA kernels. ❸ **OpenEquivariance** [15]. The state-of-the-art library for efficient sparse tensor product with JIT-enabled kernels on GPU. ❹ **DiffDock-FPGA** [17]. The FPGA-based design for dedicated CGTP operations in DiffDock [1]. Since DiffDock-FPGA only support fixed six types of CGTP, we re-implement and extend its design to accommodate the cases in our experiments. The SAS strategy of DiffDock-FPGA is excluded because it's highly depended on the ReLU function, which is irrelevant to the CGTP studied in this work. DSP resource usage is normalized for DiffDock-FPGA and *Equicore* for a fair comparison.

B. Model Accuracy

The comparison of model accuracy is shown in Table III. Our MSQ method successfully quantizes all components (irreps, weights, and CG coefficients) into 8-bit integers while maintaining comparable accuracy across all metrics. Although MSQ performs slightly worse in some cases (e.g., energy MAE on the toluene datasets), our MSQ is more hardware-friendly because: 1) all CGTP computations can be executed using integer arithmetic, and 2) dequantization can be efficiently implemented with simple shift operations.

C. Speedup and Energy Efficiency

We compare the performance of our *Equicore* and other GPU and FPGA based works in Figure 7. *Equicore* achieves $10.5\times$ and $5.3\times$ speedup comparing to the GPU baseline library e3nn and the SOTA GPU library OpenEquivariance, respectively. *Equicore* also achieves $3.1\times$ speedup comparing to the FPGA-based DiffDock-FPGA. For energy efficiency, *Equicore* achieves $17.4\times$ and $7.9\times$ improvement against e3nn and OpenEquivariance, $3.1\times$ improvement against DiffDock-FPGA. The performance gain is due to both the software optimization of efficient quantization and sparsification and efficient hardware design to support these optimizations. Besides, our design also shows high flexibility to support various types of CGTP with different orders and multiplicities across many ENN cases.

D. Ablation Study

In figure 8, all three methods demonstrate high effectiveness across different CGTP order and multiplicity. Exceptions occur in a few extreme cases, for example, under the setting of $l = 0$, SBS method shows no performance gain because CG coefficient is dense matrix under this setting. And our MSQ and cascade methods show no speedup gain because the CGTP operation is memory-bounded by the limited HBM bandwidth. Future explorations like adjusting the number of DSP resource within each *Equicore* or switching to a higher HBM bandwidth FPGA board would help to solve these limitations.

VII. Conclusion

In this work, we present *Equicore*, a software-hardware co-design to accelerate the CGTP operations in ENNs on FPGA. Implemented on VCU128 FPGA board, *Equicore* obtains up to $10.5\times$ and $17.4\times$ speedup and energy efficiency improvement comparing to various GPU libraries and FPGA-based designs.

REFERENCES

[1] G. Corso, H. StÃ, B. Jing, R. Barzilay, T. Jaakkola *et al.*, "Diffdock: Diffusion steps, twists, and turns for molecular docking," in *International Conference on Learning Representations (ICLR 2023)*, 2023.

[2] M. A. Ketata, C. Laue, R. Mammadov, H. Stark, M. Wu, G. Corso, C. Marquet, R. Barzilay, and T. S. Jaakkola, "Diffdock-pp: Rigid protein-protein docking with diffusion models," in *ICLR 2023-Machine Learning for Drug Discovery workshop*.

[3] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller, "Machine learning of accurate energy-conserving molecular force fields," *Science advances*, vol. 3, no. 5, p. e1603015, 2017.

[4] S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky, "E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials," *Nature communications*, vol. 13, no. 1, p. 2453, 2022.

[5] A. Musaelian, S. Batzner, A. Johansson, L. Sun, C. J. Owen, M. Kornbluth, and B. Kozinsky, "Learning local equivariant representations for large-scale atomistic dynamics," *Nature Communications*, vol. 14, no. 1, p. 579, 2023.

[6] J. Brandstetter, R. Hesselink, E. van der Pol, E. J. Bekkers, and M. Welling, "Geometric and physical quantities improve e (3) equivariant message passing," in *International Conference on Learning Representations*.

[7] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604–613.

[8] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv preprint arXiv:2104.13478*, 2021.

[9] S. Luo, T. Chen, and A. S. Krishnapriyan, "Enabling efficient equivariant operations in the fourier basis via gaunt tensor products," in *The Twelfth International Conference on Learning Representations*.

[10] E. Wigner, *Group theory: and its application to the quantum mechanics of atomic spectra*. Elsevier, 2012, vol. 5.

[11] I. Batatia, D. P. Kovacs, G. Simm, C. Ortner, and G. Csányi, "Mace: Higher order equivariant message passing neural networks for fast and accurate force fields," *Advances in neural information processing systems*, vol. 35, pp. 11 423–11 436, 2022.

[12] Y.-L. Liao, B. M. Wood, A. Das, and T. Smidt, "Equiformerv2: Improved equivariant transformer for scaling to higher-degree representations," in *The Twelfth International Conference on Learning Representations*.

[13] G. Simeon and G. De Fabritiis, "Tensornet: Cartesian tensor representations for efficient learning of molecular potentials," *Advances in Neural Information Processing Systems*, vol. 36, pp. 37 334–37 353, 2023.

[14] Mario Geiger, Emine Kucukbenli, Becca Zandstein, and Kyle Tretina, "Accelerate drug and material discovery with new math library nvidia cuequivariance," 2024, https://developer.nvidia.com/blog/accelerate-drug-and-material-discovery-with-new-math-library-nvidia-cuequivariance/ Accessed: 2025-06-16.

[15] V. Bharadwaj, A. Glover, A. Buluç, and J. Demmel, "An efficient sparse kernel generator for o (3)-equivariant deep networks," in *2025 Proceedings of the Conference on Applied and Computational Discrete Algorithms (ACDA)*. SIAM, 2025, pp. 32–46.

[16] S. Tang, X. Zhou, and M. Ling, "Modeling equivariant neural networks for hardware acceleration, a case study on the molecular docking tool diffdock," in *2024 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, 2024, pp. 1–6.

[17] C. Zhang, R. Liu, S. Tang, S. Li, and M. Ling, "Diffdock-fpga: A hardware accelerator for molecular docking with customized tensor product framework and sparse-aware access strategy," in *Proceedings of the Great Lakes Symposium on VLSI 2025*, 2025, pp. 435–441.

[18] M. Geiger and T. Smidt, "e3nn: Euclidean neural networks," *arXiv preprint arXiv:2207.09453*, 2022.

[19] J. E. Gerken, J. Aronsson, O. Carlsson, H. Linander, F. Ohlsson, C. Petersson, and D. Persson, "Geometric deep learning and equivariant neural networks," *Artificial Intelligence Review*, vol. 56, no. 12, pp. 14 605–14 662, 2023.

[20] X. Zhao, Y. Wang, X. Cai, C. Liu, and L. Zhang, "Linear symmetric quantization of neural networks for low-precision integer hardware," in *International Conference on Learning Representations*, 2020.

[21] K. Fatahalian, J. Sugerman, and P. Hanrahan, "Understanding the efficiency of gpu algorithms for matrix-matrix multiplication," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, 2004, pp. 133–137.

[22] S. Wouters, W. Poelmans, P. W. Ayers, and D. Van Neck, "Chemps2: A free open-source spin-adapted implementation of the density matrix renormalization group for ab initio quantum chemistry," *Computer Physics Communications*, vol. 185, no. 6, pp. 1501–1514, 2014.

[23] S. Passaro and C. L. Zitnick, "Reducing so (3) convolutions to so (2) for efficient equivariant gnns," in *International conference on machine learning*. PMLR, 2023, pp. 27 420–27 438.

[24] I. Batatia, M. Geiger, J. Munoz, T. Smidt, L. Silberman, and C. Ortner, "A general framework for equivariant neural networks on reductive lie groups," *Advances in Neural Information Processing Systems*, vol. 36, pp. 55 260–55 284, 2023.

[25] P. McNamee, F. Chilton *et al.*, "Tables of clebsch-gordan coefficients of s u 3," *Reviews of Modern Physics*, vol. 36, no. 4, p. 1005, 1964.

[26] J. Sommer, M. A. Özkan, O. Keszocze, and J. Teich, "Dsp-packing: Squeezing low-precision arithmetic into fpga dsp blocks," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2022, pp. 160–166.

[27] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with int8 optimization on xilinx devices," *White Paper*, 2016.

[28] A. Samajdar, T. Garg, T. Krishna, and N. Kapre, "Scaling the cascades: Interconnect-aware fpga implementation of machine learning problems," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 342–349.

[29] R. Chen, J. Liu, S. Tang, Y. Liu, Y. Zhu, M. Ling, and B. Da Silva, "Ate-gcn: An fpga-based graph convolutional network accelerator with asymmetrical ternary quantization," in *2025 Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–6.

[30] S. Lu, T. Zhao, T.-J. Lin, R. Zhang, C. Wu, and L. He, "Mcoreopu: An fpga-based multi-core overlay processor for transformer-based models," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 18, no. 3, 2025.

[31] S. Chmiela, H. E. Sauceda, I. Poltavsky, K.-R. Müller, and A. Tkatchenko, "sgdml: Constructing accurate and data efficient molecular force fields using machine learning," *Computer Physics Communications*, vol. 240, pp. 38–45, 2019.