AutoTune-TCU: Hardware-Software Co-Design of a Learning Based Self-Optimizing TCU for HPC

Nitesh Narayana GS[1,2], Sonam[2], Xavier Martorell[1,2]

[1]Universitat Politècnica de Catalunya Barcelona, [2]Barcelona Supercomputing Center

***Abstract*—Domain-specific tensor and matrix compute units, such as Google TPUs, NVIDIA Tensor Cores, and Intel AMX, are increasingly integrated with general-purpose CPU cores to accelerate tensor- and vector-centric workloads. However, fixed settings for tiling, dataflow, and precision are often suboptimal across changing kernel mixes and operating conditions, motivating online adaptation under power, thermal, and memory constraints.**

This work presents AutoTune-TCU, a hardware–software co-design of a learning-based self-optimizing Tensor Compute Unit tightly coupled with a CPU core(In-order RISC-V CVA6) complex. AutoTune-TCU combines a multi-mode TCU microarchitecture with a hierarchical control framework: a fast loop for kernel-boundary reconfiguration and a slow loop for drift-aware policy refinement. A constraint-enforcing actuation path validates each candidate configuration against CSR-encoded power, thermal, and numerical limits before hardware commit.

To demonstrate versatility beyond GEMM-centric designs, the implemented substrate supports GEMM, GEMV, DOT, AXPY, SCAL, and REDUCE, and is realised in RTL as a core-attached coprocessor with memory-mapped control/status registers and performance counters. Simulation results confirm correctness across all modes and show adaptive behaviour through controller metrics such as constraint-clamp events, drift flags, and adaptation count. Overall, AutoTune-TCU demonstrates practical online self-optimisation for reconfigurable tensor hardware in High-performance Computing (HPC) systems.

I. Introduction

Tensor-centric computation has become fundamental to modern high-performance computing (HPC), spanning dense linear algebra, iterative solvers, data-driven simulation, and emerging scientific AI pipelines [8], [18]. To sustain performance within power and thermal envelopes, contemporary HPC nodes increasingly integrate domain-specific tensor engines—such as Tensor Compute Units (TCUs)—as on-die coprocessors next to general-purpose CPU cores [6], [7], [11]. In practice, static hardware settings for tiling, dataflow, precision, and buffering at the TCU interface are frequently suboptimal across kernel phases, and memory-system conditions, limiting sustained efficiency in production deployments.

Learning-based runtime optimization is promising, but direct online tuning of TCU knobs from the CPU runtime is difficult because of the large configuration space and strict deployment constraints. Unconstrained adaptation can violate power, thermal, bandwidth, or numerical-error limits, while offline autotuning and heuristic policies may fail to track non-stationary traces and cross-kernel shifts [19]. In AutoTune-TCU, runtime TCU configurations are chosen by a learned policy, using fast and slow control loops over observed metrics instead of fixed manually tuned settings, enabling online adaptation to better operating points under CSR-enforced constraints.

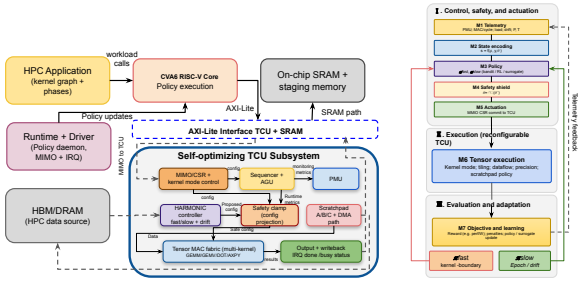
This paper presents **AutoTune-TCU**, a hardware/software co-design of learning-based self-optimizing tensor hardware. AutoTune-TCU contributions can be summarised as:

1) **Multi-mode reconfigurable TCU:** A parameterized TCU supporting GEMM, GEMV, DOT, AXPY, SCAL, and REDUCE with configurable precision (FP32/BF16/INT8), dataflow, tiling, and memory mapping.
2) **Learning-based auto-tuning:** An ML tuner (Bayesian optimization + surrogate modeling) that predicts high-quality TCU configurations from workload/runtime features extracted from hardware performance counters.
3) **Closed-loop adaptive control.** Integration of hardware counters (utilization, stalls, bandwidth, buffer pressure) for online feedback-driven reconfiguration.
4) **Constraint-enforcement layer.** Realise a CSR-driven constraint-enforcement plane that projects every candidate configuration into a hardware-admissible set before TCU commit, and exports clamp statistics for analysis.

II. Background and Related Work

Tensor-accelerator efficiency in HPC is governed by a coupled compute–memory–control trade-off: dataflow and storage hierarchy determine reuse, throughput, and energy, making runtime adaptation of tiling, memory orchestration, and precision an architectural necessity [5]. Static and offline tuning cannot track phase changes, contention, or thermal drift, while learning-based control can—but hardware deployment demands explicit constraint handling. In HPC, adaptation must also preserve numerical fidelity, as precision choices affect convergence and solution quality [1], [2], [10].

Roofline remains the standard compute–bandwidth lens [18]. Prior accelerators (TPU, Eyeriss) and mapping tools (Timeloop, MAESTRO) highlight the importance of dataflow-aware optimization, but do not provide self online actuation [4], [12]–[14], [17]. Learning-based methods (Bayesian optimization, safe RL), SARA, and TVM/AutoTVM demonstrate adaptive optimization in accelerator and software stacks [3], [9], [15], [16]. AutoTune-TCU targets this gap with RTL-level multi-mode TCU



(a) Framework (b) Control projection

Fig. 1: AutoTune-TCU overview and closed-loop control

hardware reconfiguration using hierarchical fast/slow online control and constraint-enforcing clamping before commit.

III. AutoTune-TCU

A. System Overview

AutoTune-TCU consists of three planes: (i) a reconfigurable TCU hardware substrate, (ii) a runtime-metrics plane, and (iii) a hierarchical learning-control plane, as shown in Figure 1. The implemented platform integrates a CVA6-style RISC-V host core with a TCU subsystem comprising a sequencer, scratchpad, MAC array, CSR/PMU block, and the AutoTune controller (Fig. 1a). Host software configures kernels via MMIO CSRs and reads telemetry and status for policy updates.

The hardware exposes runtime knobs including tile sizes, dataflow mode, memory partitioning, precision mode, and DVFS. The controller consumes runtime metrics and emits configuration actions, while the constraint-enforcement plane validates each proposed action against CSR-encoded limits before hardware actuation.

B. Self Learning-Based Auto-Tuning & Closed-Loop Control

At each kernel/job boundary, AutoTune-TCU updates its configuration using a fast rule and optional slow-loop refinement (Fig. 1b). Drift is detected from load proxy changes, and every candidate action is projected by the constraint-enforcement logic before hardware commit. Constraint-clamp events and adaptation counters are exported via status CSRs.

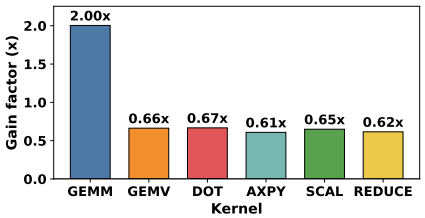
A closed-loop runtime tuner (surrogate model plus Upper Confidence Bound (UCB) policy) consumes runtime features and emits per-epoch reconfiguration plans, complementing hardware constraint clamping with software-level learning. Table I shows kernel-specific operating points over tile shape, precision, and memory mapping, confirming that the TCU is reconfigured online rather than statically fixed. *MAC increment* denotes per-profile multiply–accumulate work ($y \leftarrow y + a \times b$) used for load and performance accounting. Positive kernel-wise and epoch-wise Δ-values indicate consistent gains from adaptive reconfiguration over the prior policy.

IV. Evaluation Metrics and Results

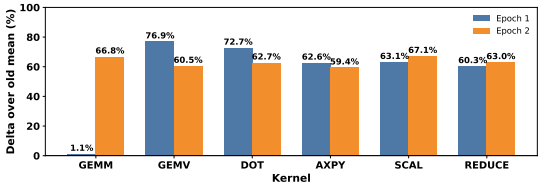
We evaluate AutoTune-TCU on six kernel classes (GEMM, GEMV, DOT, AXPY, SCAL, REDUCE) using epoch-wise closed-loop traces over the implemented reconfigurable space (precision, dataflow, memory mapping, and tile shape). We report: (i) per-kernel gain after each auto-tuner update and (ii) epoch-wise mean-score shift relative to the previous policy.

TABLE I: Runtime kernel-profile values used by the AutoTune reconfiguration flow.

Kernel	Tile$_m$	Tile$_n$	Tile$_k$	Precision/Mem-map	MAC increment
GEMM	8	8	4	FP32 / BAL	128
GEMV	8	1	8	BF16 / A_PRI	96
DOT	1	1	8	INT8 / BAL	64
AXPY	8	1	2	BF16 / A_PRI	72
SCAL	8	1	1	INT8 / B_PRI	56
REDUCE	1	1	8	BF16 / C_PRI	68



(a) Per-kernel gain factor after auto-tuner update



(b) Kernel-wise delta (%) across two epoch passes

Fig. 2: Post-update improvements: per-kernel gain and epoch-wise score shift.

The objective combines performance with runtime costs (safety, memory pressure, and power), so effectiveness is measured by the policy improvement (Δ) relative to the prior policy (Fig. 2). Thus, $\Delta > 0$ indicates a better tuner policy. Figure 2a shows $\Delta > 0$ for all kernels, including compute-dominated GEMM and memory-sensitive primitives, indicating cross-kernel generalization rather than single-kernel overfitting. Figure 2b confirms that $\Delta > 0$ for every epoch, demonstrating stable and consistent policy improvement over time.

V. Conclusion and Future Work

AutoTune-TCU demonstrates stable, constrained online adaptation on a reconfigurable TCU. Overall, AutoTune-TCU delivers a **+68.06%** mean objective gain, positive shifts for all six kernels, and positive deltas across all 12 epochs—demonstrating practical, repeatable online adaptation on reconfigurable tensor hardware.

Future Work. We will extend AutoTune-TCU by (i) integrating with the RISC-V Vortex GPGPU stack for dynamic SIMT–TCU partitioning, and (ii) deploying on chiplet-based heterogeneous nodes for mixed HPC+GenAI workloads. The long-term goal is a unified adaptive controller across control, SIMT/vector, tensor, and memory chiplets under cross-chiplet power–thermal safety for higher sustained throughput.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insights. We thank the ISCA 2026 Student Travel Grant initiative, which helped us present this work at CogArch 2026. This work has been supported by the Spanish Ministry of Science, Innovation and Universities (research project ST4HPC - PID2023-147979NB-C21 financiado por MICIU/AEI /10.13039/501100011033 y por FEDER, UE).

REFERENCES

[1] E. Altman, *Constrained Markov decision processes*. Routledge, 2021.

[2] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *arXiv preprint arXiv:1204.5721*, 2012.

[3] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, 2018, pp. 578–594.

[4] 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, 2016.

[5] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[6] J. Choquette, "Nvidia hopper h100 gpu: Scaling performance," *IEEE Micro*, vol. 43, no. 3, pp. 9–17, 2023.

[7] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "Nvidia a100 tensor core gpu: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, 2021.

[8] R. Chowdhury, A. C. Jacob, T. B. Jablin, M. P. Tripunitara, J. Henkel, and D. R. Kaeli, "A computational model for tensor core units," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 303–305. [Online]. Available: https://doi.org/10.1145/3350755.3400252

[9] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[10] N. J. Higham and T. Mary, "A new approach to probabilistic rounding error analysis," *SIAM journal on scientific computing*, vol. 41, no. 5, pp. A2815–A2835, 2019.

[11] Intel Corporation, "What is intel® advanced matrix extensions (intel® amx)?" https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-amx.html, 2024, accessed: 2026-04-16.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[13] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *IEEE micro*, vol. 40, no. 3, pp. 20–29, 2020.

[14] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.

[15] A. Samajdar, M. Pellauer, and T. Krishna, "Self-adaptive reconfigurable arrays (sara): Using ml to assist scaling gemm acceleration," *arXiv preprint arXiv:2101.04799*, 2021.

[16] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[17] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *Efficient processing of deep neural networks*. Springer, 2020, vol. 51.

[18] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[19] L. Zheng, C. Jia, M. Sun, Z. Wu, C. H. Yu, A. Haj-Ali, Y. Wang, J. Yang, D. Zhuo, K. Sen *et al.*, "Ansor: Generating {High-Performance} tensor programs for deep learning," in *14th USENIX symposium on operating systems design and implementation (OSDI 20)*, 2020, pp. 863–879.