Modeling multi-chiplet architectures for TPU co-design

Kavya Sreedhar, Pritha D. Narayanappa, Hung-Ming Hsu, Khai Tran, Hardie Cate, Narges Shahidi, Zhijie Deng, Avinash Lingamneni, Thejasvi Vijayaraj, Aditya Yanamandra, Sameer Kumar*, Ming Liu, Lluis-Miquel Munguia
kavyasreedhar@google.com
Google

ABSTRACT

We present an experience report on modeling multi-chiplet hardware accelerators for TPU co-design, and argue that detailed chiplet modeling is necessary to identify bottlenecks and optimizations for future hardware design. Our chiplet modeling tool (CMT) shows that at lower target serving latencies, serving queries/second/chip (QPS/chip) can be up to 2.2× different with chiplet modeling compared to modeling with aggregated compute and memory resources. With this tool, we identify interdependencies between chiplet modeling and the rest of the system: we highlight an example where cost savings from prefetching weights from HBM to local SRAM need to be incorporated in the selection heuristic for chiplet sharding strategies. Using CMT, we explore performance sensitivity to chiplet properties in a design space exploration that highlights considerations for high-performance serving. We use an open-source MoE model (OSS-MoE) served on Ironwood-like TPU architectures as a case study. We conclude by discussing the need to establish best sharding practices to narrow this large design search space.

1 INTRODUCTION

State-of-the-art hardware for machine learning is adopting multi-chiplet hardware designs [1–3]. For example, Google's Ironwood TPU has the total compute and memory resources split across two homogeneous chiplets [1]. In theory, workloads can be parallelized in a transparent way across chiplets, given their high bandwidth and low latency compared to off-chip networks. However, in practice, chiplet collective latencies may still dominate, and there may not be much parallelization left in the workloads.

A natural way to start modeling homogeneous chiplet-based architectures is with "megacore" modeling, which specifies the combined compute and memory resources of all the chiplets as one chip. We can add fixed latency overheads to approximate chiplet communication. However, we find that this communication may or may not be exposed and is highly variable, depending on sharding strategies and other system bottlenecks. We argue that we need detailed chiplet modeling to identify when we face chiplet overheads and develop targeted optimizations to minimize exposed overheads.

2 CHIPLET MODELING TOOL (CMT)

We aim to decide how to partition the data per chip across chiplets to maximize performance. Thus, we distill chiplet modeling into two key questions: (1) are we searching the relevant sharding strategies? and (2) are we selecting the best strategy?

Sharding strategy	Which dimensions are sharded
Batch	Dims in all inputs and outputs
Input	Dims in input activations and output
Output	Dims in weights / KV cache and output
Contracting	Dims in both inputs but not the output
Expert, Token [4]	Expert, token dimension (FFN only)
Replicated	None – all tensors replicated in all chiplets

Table 1: The different sharding strategies that CMT considers.

2.1 Search Space

CMT allows a user to specify the number of chiplets, chiplet-to-chiplet latency and bandwidth, and chiplet topology for a multi-chiplet architecture. The topology describes how chiplets are connected. CMT then automatically searches for the set of optimal sharding strategies for every operator in a given workload. Each operator can be sharded along a different dimension, but we assume that all chiplets are used towards the same form of parallelism for a given operator[1]. We also assume that chiplet communication can be partially overlapped with compute and chip-level collectives.

For each operator, we currently consider the sharding strategies shown in Table 1[2]. We leverage memoization techniques to efficiently explore this large sharding search space with (number of sharding strategy)^(number of operators) options. By definition, each sharding strategy may have some tensors (input, weights/KV cache, output) replicated. For example, for the input sharding strategy, the weights/KV cache will need to be replicated on each chiplet.

We add an optimization to evaluate storing these replicated tensors in a distributed manner across HBM stacks. This distributed strategy trades the cost of a chiplet collective to rematerialize the tensor for the computation for the ability to (1) leverage the collective HBM bandwidth for reads/writes and (2) use a smaller distributed memory footprint to evaluate what tensor sizes fit within the HBM capacity. Thus, distributing these tensors can reduce exposed memory transfer time and enable using larger batch sizes.

However, we observed some scenarios bottlenecked by the chiplet collective to rematerialize the tensor. Then, it can be more performant to select a different strategy set with cheaper chiplet collectives or not distribute these replicated tensors. Thus, for every set of sharding strategies considered, CMT also automatically evaluates distributed or replicated storage for the replicated tensors.

CMT automatically inserts the chiplet collectives required based on the shardings of pairs of operators. Importantly, the user does not need to reason about or specify parallelization strategies and

* Work done while at Google.

LATTE '26, March 23, 2026, Pittsburgh, PA, USA

[1] CMT can be extended to support multiple forms of parallelism across chiplets. Given the larger search space, it is important to understand preferred sharding strategies.

[2] May not be exhaustive with model developments. Other sharding strategies can easily be added, e.g., sharding by experts when moving from dense models to MoEs.

resulting collectives, since CMT automatically explores these trade-offs for every possible combination of shardings for all operators.

We provide some intuition on CMT's decisions. If chiplet collectives are relatively cheap, the most performant sharding strategies can require a lot of on-chip communication. If chiplet collectives are relatively expensive, the optimal sharding strategies will prioritize data locality to avoid data transfers with chiplet collectives, e.g., full replication may be favored over any sharding strategy.

2.2 Selection Heuristic

In our setup, CMT plugs into an analytical simulator for TPU hardware. This simulator includes a prefetcher, which considers when weights can be prefetched from HBM or pinned in local SRAM, to minimize exposed memory transfer time. CMT is executed before the prefetcher is run, since evaluating all per-operator chiplet sharding strategy sets with the prefetcher would result in prohibitively long simulation times. Thus, CMT compares compute and communication costs for different strategy sets, assuming all weights are stored in HBM. Then, the one optimal strategy set returned by CMT is considered for prefetching/pinning opportunities.

We found that CMT can select a strategy set that appears to have higher performance with weights in HBM, but this set may be sub-optimal after the prefetcher is run. Consider an example with strategy sets A and B considered by CMT. A and B are bottlenecked by compute, and A's compute time, as considered in CMT, is slightly faster than B's compute time. However, the chiplet communication time is longer for A than B. CMT returns A as the optimal strategy set, but this small difference in compute times between A and B can be made up with prefetching weights. Then, the communication time in A dominates, and A becomes a sub-optimal choice compared to B after the prefetcher. With CMT, we identified up to 20% performance loss in these types of scenarios, and are actively working on incorporating SRAM costs in our selection heuristic.

3 DESIGN SPACE EXPLORATION

We use the analytical TPU simulator, with CMT, to sweep combinations of different batch sizes, numbers of chips, chip-level sharding strategies for serving OSS-MoE. The number of chips goes up to the 9216 pod size for Ironwood [1]. Chip-level sharding strategies are evaluated from a set of pre-defined recipes such as expert parallelism[3]. We then plot Pareto-optimal serving configurations for serving queries/second/chip vs target end-to-end serving latency[4].

The costs of chiplet collectives depends on the number of chiplets, chiplet-to-chiplet latency and bandwidth, and topology. CMT enables us to easily explore these sensitivities. In Figure 1, we partition the same aggregate resources, 4× the compute and 8× the memory of Ironwood (to provide more resources per chiplet with > 2 chiplets), across 2 to 16 chiplets. For this setup, we find that:

- Megacore and CMT report the same performance until > 4 chiplets, where there are exposed chiplet collectives and NUMA effects due to partitioned memory/compute resources.

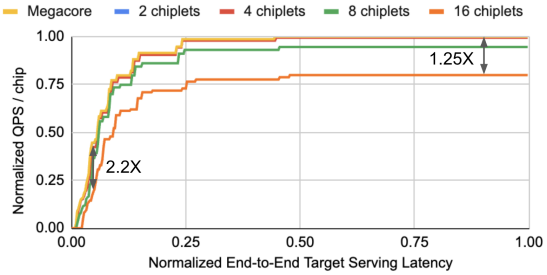

Figure 1: Normalized QPS/chip vs normalized latency with different numbers of chiplets but same aggregate resources.

- For 8 chiplets, CMT identifies a 1.06× (higher serving latencies) to 1.22× (lower serving latencies) gap compared to megacore, which we expected to be optimistic.
- This gap is exacerbated as the number of chiplets increases: QPS/chip is 1.25× to 2.2× lower than megacore for 16 chiplets.

At this point, it is valuable to explore on-chip latency optimizations, and consider increasing local SRAM storage to reduce data movement. With this takeaway, we consider a larger SRAM (1.5GiB) with CMT for an 8-chiplet architecture (4×2 2D mesh). We find that:

- QPS/chip is not very sensitive to chiplet bandwidth but is sensitive to chiplet latency, since the chiplet-collective payload sizes are small and thus latency-dominated.
- More connectivity (e.g., 2D mesh vs ring) leads to higher performance, even when more-connected topologies are penalized with smaller SRAM (emulates increased area costs).

In decode especially, there is often not substantial computation to hide the chiplet communication, resulting in exposed collectives.

4 BRIEF DISCUSSION

We can use CMT to analyze the frequency of chiplet sharding and replicated storage strategies chosen[5]. We argue that it is important to work towards establishing a set of best chiplet sharding practices to narrow this large search space. We would expect attention operators to shard by batch dimensions to avoid chiplet collectives and FFN operators to shard by experts, assuming that we can overlap the chiplet all-to-all (A2A) collective with the A2A across chips with expert parallelism. With CMT, we realized that it is not always this straightforward. For attention, small batch scenarios identified the need for weight replication with batch sharding. Otherwise, there may not be enough computation to hide the collective to gather the distributed weights. For FFN, small models may only require one chip, resulting in no chip-level expert parallelism. Then, the chiplet A2A can be exposed and expensive.

5 CONCLUSION

As chiplet counts grow, we argue that megacore modeling becomes a liability. We hope to inspire a discussion on how we can build high-fidelity chiplet modeling tools to provide more signal for hardware optimization opportunities and build a set of sharding recipes for compilers targeting multi-chiplet hardware accelerators.

Acknowledgements: We thank Ravi Iyer, Ananth Durbha, Eugene Ie, Ton Kalker, and the TPU co-design and architecture teams.

[3] CMT's per-operator exploration can be applied to chip-level shardings as well, since this problem fundamentally considers the same two questions we posed earlier.
[4] Axes are normalized to the maximum QPS/chip and latency in Figure 1.
[5] CMT currently considers SPMD but could be extended for MPMD in the future.

REFERENCES

[1] Norman P. Jouppi and Sridhar Lakshmanamurthy. 2025. Ironwood: Delivering Best in Class perf, perf/TCO and perf/Watt for Reasoning Model Training and Serving. In *2025 IEEE Hot Chips 37 Symposium (HCS)*.

[2] Don Soltis and Stephen Robinson. 2025. Clearwater Forest the Next Generation Intel Xeon Processor with Efficiency Cores. In *2025 IEEE Hot Chips 37 Symposium (HCS)*.

[3] Ajay Tirumala and Raymond Wong. 2024. NVIDIA Blackwell Platform: Advancing Generative AI and Accelerated Computing. In *2024 IEEE Hot Chips 36 Symposium (HCS)*.

[4] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. 2024. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664* (2024).