FlatAttention: Dataflow and Fabric Collectives Co-Optimization for Efficient Multi-Head Attention on Tile-Based Many-PE Accelerators

Chi Zhang*, Luca Colagrande*, Renzo Andri‡, Thomas Benz*, Gamze Islamoglu*, Alessandro Nadalini†, Francesco Conti†, Yawei Li*, Luca Benini*†
*Integrated Systems Laboratory (IIS), ETH Zurich, Zurich, Switzerland
†Department of Electrical, Electronic, and Information Engineering (DEI), University of Bologna, Bologna, Italy
‡Computing Systems Lab, Huawei Zurich Research Center, Zurich, Switzerland
{chizhang, colluca, tbenz, gislamoglu, yawli, lbenini}@iis.ee.ethz.ch,
{alessandro.nadalini3, f.conti}@unibo.it, renzo.andri@huawei.com

***Abstract*—Multi-Head Attention (MHA) is a critical computational kernel in transformer-based AI models. Emerging scalable tile-based accelerator architectures integrate increasing numbers of tightly-packed processing elements (PEs) with tensor units. MHA dataflow mapping is crucial for achieving high utilization of the available units. We propose FlatAttention, a new dataflow for MHA on tile-based many-PE accelerators, minimizing costly main memory (HBM) accesses by leveraging collective primitives integrated into the on-chip network fabric. FlatAttention achieves up to 89.3% utilization, and 4.1× performance speedup over FlashAttention-3 dataflow on tile-based accelerators whilst reducing HBM traffic by 16×. Through algorithm-architecture co-exploration, we identify an optimal configuration for a large scaled-out tile-based accelerator featuring a 32×32 tile mesh with 1024 TFLOPS @ FP16 peak performance, comparable to the state-of-the-art Nvidia H100 GPU. FlatAttention in this configuration achieves up to 1.3× higher utilization over FlashAttention-3 on the H100 GPU. Meanwhile, this tile-based accelerator configuration requires 40% less HBM bandwidth compared to the H100 GPU, enabling a 1.8× reduction in die size, estimated on the same technology node.**

***Index Terms*—Multi-Head Attention, Tile-Base Architecture, Network on Chip, Collective Primitives.**

I. Introduction

Transformer-based artificial intelligence (AI) models, such as GPT-4, LLaMA, and DeepSeek-V3 are dominating Large Language Models (LLMs). Among the computational kernels in transformer-based models, Multi-Head Attention (MHA) exhibits quadratic complexity over sequence length [1], making it a critical factor in performance, especially for long sequences. Previous LLM studies [2], [3] have shown that most operations in MHA are bottlenecked by memory accesses.

The "attention bottleneck" has driven extensive research focused on optimizing MHA dataflows on the dominant AI hardware platform, namely Nvidia GPUs. One of the most widely adopted solutions is FlashAttention [4], which efficiently fuses MHA microkernels. Over two generations of improvements, FlashAttention-2 [5] introduced algorithmic optimizations, whereas FlashAttention-3 [6] additionally leverages asynchronous execution for improved performance. However, still no more than 75%[1] utilization was achieved on the H100 GPUs [6].

Moreover, Nvidia's state-of-the-art (SoA) H100 GPU comes with significant cost and power requirements, featuring an 814 mm^2 die on TSMC's 5nm process node, coupled with six High Bandwidth Memory (HBM) stacks accounting for over 50% of the total cost, and a Thermal Design Power (TDP) of 700 W. Given the suboptimal utilization of the MHA layer and the high cost and power requirements, many competitors are working on hardware and software to improve system cost and energy efficiency with highly optimized accelerators designed for strong LLM performance. The goal is to offer competitive performance while boosting the efficiency of the system by minimizing energy-hungry HBM accesses.

As recent application trends, linked to the "reasoning" use of LLMs, emphasize inference as a value-added workload, a scalable design pattern for inference accelerators targeting scaled-up transformer-based AI models is emerging [7]–[10]: these accelerators are constructed as large, full-reticle, or even multi-die integrated systems [8] structured as meshes of compute tiles containing extremely dense, large matrix units, coupled with vector and scalar engines to accelerate all key kernels in LLMs, and local, explicitly managed memories for main memory data buffering and latency hiding. Several HBMs are typically employed as main memory, positioned at die boundaries and supported by multiple memory controllers.

Such a tile-based many-Processing Element (PE) accelerator architectural template favors silicon efficiency and scalability, featuring a dense compute tile placement and a software-controlled partitioned memory hierarchy. However, mapping LLM inference workloads onto these scalable tile-based accelerators presents a significant challenge. The inter-tile and tile-to-HBM dataflow must be carefully designed to achieve high utilization of the tiles' matrix engines and minimize energy-hungry off-chip access. In this work, we show that leveraging collective communication primitives in the Network on Chip

[1]FlashAttention-3 baseline. Numbers use arXiv v1 (11 Jul 2024) [FA3-arXiv], the newest version when experiments were run; a later NeurIPS 2024 release reports ∼10 % higher throughput [FA3-NeurIPS].

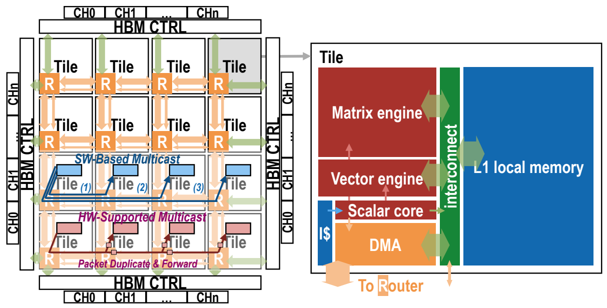

Fig. 1: Tile-Based Many-PE Architecture Template

(NoC), such as reduction operations and multicast, can be highly beneficial for both of these objectives.

Existing works investigating mapping and architecture co-exploration for LLM workloads either fail to fully consider key optimizations, such as operator fusion and overlap within the MHA layer, falling short of FlashAttention's highly optimized SoA dataflow [11], or do not explore the use of inter-tile collective primitives in NoCs [12], [13], or both [14]. Although some AI accelerator vendors claim to achieve high LLM inference efficiency through optimized dataflows [7], [8], the details of their dataflow implementations—particularly for the MHA layer—and on-chip fabric architectures remain confidential. Additionally, accelerator architecture design must be co-explored alongside dataflow optimizations to determine optimal design parameters and features.

This work takes the MHA in the prefill stage of LLM inference as a case study, which aligns with FlashAttention [4] and could be generalized to training. We propose a new dataflow for scalable tile-based accelerators and present a co-design approach for algorithm-architecture co-exploration. The contributions of this paper are:

- A modeling and simulation framework that enables estimating the performance of a large set of tile-based accelerators and the co-design of network primitives.
- An efficient workload allocation and scheduling strategy called FlatAttention. FlatAttention leverages collective primitives on the on-chip network fabric to achieve up to 89.3% utilization for MHA layer on tile-based many-PE accelerators, and 4.1× performance speedup over FlashAttention-3 dataflow on the same tile-based accelerator, whilst reducing HBM traffic by 16×.
- Co-exploration of accelerator architecture and FlatAttention parameters, highlighting key trends and trade-offs to guide the selection of optimal FlatAttention parameters.
- An optimal tile-based accelerator configuration that matches the peak performance of the current SoA Nvidia H100 GPU. FlatAttention in our configuration achieves up to 1.3× higher utilization over FlashAttention-3 on H100. Meanwhile, this tile-based accelerator configuration requires 40% less HBM bandwidth than the H100 while achieving a 1.8× reduction in die size, estimated on the same technology node.

II. Reference Tile-Based Many-PE Architecture

This section introduces the architecture of the scalable tile-based many-PE design pattern, prominently featured in SoA commercial Machine Learning (ML) accelerators, e.g., Tesla's Dojo system [15] and Tenstorrent's Blackhole chip [9].

As illustrated in Fig. 1, the fundamental building block of the many-PE architecture is the "tile". Each tile comprises PEs, local memory (L1), a Direct Memory Access (DMA) engine, and local interconnects. There are three main types of PEs: scalar cores, vector engines, and matrix engines. Scalar cores mainly handle dataflow control tasks, whereas heavy computational tasks are offloaded to the vector and matrix engines based on the computation type. All PEs within a tile can directly access the local L1 memory via the local interconnect. The DMA engine in each tile is responsible for data movement in and out of the local L1 memory. The tile-based many-PE system uses an on-chip 2D-mesh NoC to connect tiles. Off-chip memory, such as HBM, is located at the boundary of the mesh NoC, interfaced through the respective memory controllers.

Collective communication operations [16], such as multicast and reduction, are involved in L1 data exchange among tiles. Traditional software-based collective primitives rely on successive point-to-point inter-tile transfers, leading to high communication latency. In contrast, NoCs with hardware-supported collective communication primitives establish direct, optimized communication paths, significantly reducing communication overhead [17]. For example, consider multicasting a message of size α to a chain of N clusters, demonstrated in Fig. 1. Given an L1-to-NoC router latency of L_d, router-to-router latency of L_r, and a router link bandwidth of β, the communication latency for software-based collective primitives is $N\left(\frac{\alpha}{\beta}+2L_d+\frac{N+1}{2}L_r\right)$. In contrast, NoCs with hardware-supported collective communication primitives employ a path-based forwarding strategy. Each packet injected from the source node is duplicated and forwarded in-flight along the multicast-aware routing path, as illustrated in Fig. 1. This optimization reduces the communication latency to $\frac{\alpha}{\beta}+2L_d+NL_r$. For example, when $\alpha = 16\,KB,\ \beta = 128\,B/cycle,\ L_d = 10\,cycles,\ L_r = 4\,cycles,\ N = 7$, the multicast latency is reduced by 6.1×.

III. FlatAttention Dataflow

A. Motivation

As both FlashAttention-2 and FlashAttention-3 share the dataflow introduced in FlashAttention-2, we refer to it in the following. Algorithm 1 outlines the FlashAttention-2 algorithm[2] for each head, designed to operate directly on tile-based many-PE architectures. The MHA workload is partitioned over the batch, number of heads, and output sequence length dimensions, and these blocks are distributed to the tiles where they can be processed in parallel. With this mapping, every

[2]To simplify the dataflow for tile-based many-PE accelerators, we assume the K matrix is pre-transposed in HBM while still accounting for the pre-transposition time when comparing to FlashAttention on H100 for a fair comparison [6].

Algorithm 1 FlashAttention on tile-based many-PE architecture

```
Require: Input Matrices $\mathbf{Q} \in \mathbb{R}^{S\times D}, \mathbf{K}^\top \in \mathbb{R}^{D\times S}, \mathbf{V} \in \mathbb{R}^{S\times D}$, and output matrx $\mathbf{O} \in \mathbb{R}^{S\times D}$
    in HBM, block sizes $B_c, B_r$.
 1: Divide $\mathbf{Q}, \mathbf{O}$ into $T_r = \lceil \frac{S}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r} \in \mathbb{R}^{B_r\times D}$ and $\mathbf{O}_1, \dots, \mathbf{O}_{T_r} \in \mathbb{R}^{B_r\times D}$.
    Divide $\mathbf{K}^\top, \mathbf{V}$ into $T_c = \lceil \frac{S}{B_c} \rceil$ blocks $\mathbf{K}^\top{}_1, \dots, \mathbf{K}^\top{}_{T_c} \in \mathbb{R}^{D\times B_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c} \in \mathbb{R}^{B_c\times D}$.
 2: for $1 \le i \le T_r$ do
 3:     On chip, initialize $\mathbf{O}_i^{(0)} = \mathbf{0}$.
 4:     Load $\mathbf{Q}_i$ from HBM to tile on-chip L1.
 5:     for $1 \le j \le T_c$ do
 6:         Load $\mathbf{K}^\top{}_j, \mathbf{V}_j$ from HBM to tile on-chip L1.
 7:         tile on-chip compute $\mathbf{S}_i^{(j)} = \mathbf{Q}_i \mathbf{K}_j^\top$.
 8:         tile on-chip compute $\mathbf{S}_i^{(j)} = \frac{\mathbf{S}_i^{(j)}}{\sqrt{D}}$, $m_i^{(j)} = \mathrm{rowmax}(\mathbf{S}_i^{(j)})$
 9:         if $j > 1$ then
10:             Tile on-chip compute $m_i^{(j)} = \max(m_i^{(j-1)}, m_i^{(j)})$
11:         end if
12:         Tile on-chip compute $\tilde{\mathbf{P}}_i^{(j)} = \exp(\mathbf{S}_i^{(j)} - m_i^{(j)})$.
13:         Tile on-chip compute $\ell_i^{(j)} = \mathrm{rowsum}(\tilde{\mathbf{P}}_i^{(j)})$.
14:         if $j > 1$ then
15:             Tile on-chip compute $\ell_i^{(j)} = e^{m_i^{(j-1)} - m_i^{(j)}} \ell_i^{(j-1)} + \ell_i^{(j)}$.
16:             Tile on-chip compute $\mathbf{O}_i^{(j)} = \mathrm{diag}(e^{m_i^{(j-1)} - m_i^{(j)}}) \mathbf{O}_i^{(j-1)}$.
17:         end if
18:         Tile on-chip compute $\mathbf{O}_i^{(j)} = \mathbf{O}_i^{(j)} + \tilde{\mathbf{P}}_i^{(j)} \mathbf{V}_j$.
19:         Tile on-chip update $m_i^{(j-1)} \leftarrow m_i^{(j)}, \ell_i^{(j-1)} \leftarrow \ell_i^{(j)}$.
20:     end for
21:     On chip, compute $\mathbf{O}_i = \mathrm{diag}(\ell_i^{(T_c)})^{-1} \mathbf{O}_i^{(T_c)}$.
22:     Write $\mathbf{O}_i$ to HBM as the $i$-th block of $\mathbf{O}$.
23: end for
```

tile processes distinct data, which it can independently access in HBM. No communication between tiles is required, but at the same time no reuse of data across tiles is exploited.

With sequnce length S, head dimension D, number of heads H, batch size B and block size $M := B_r = B_c$, this dataflow results in an HBM I/O complexity of:

$$\mathrm{IO} = 2 \cdot H \cdot B \cdot D \cdot S \cdot \left(1 + \frac{S}{M}\right).$$

While all other parameters are fixed by the computation, the block size parameter M can be increased to lower the I/O complexity. Intuitively, larger blocks favor the reuse of data in the L1 memory of a tile. However, the block size is constrained by the L1 memory of a *single* tile, which must be able to simultaneously host tensors Q_i, K_j^T, V_j, O_i, at any given time.

With the goal of further reducing off-chip I/O accesses, we propose the FlatAttention dataflow, which fundamentally redefines how MHA is parallelized on tile-based architectures. FlatAttention leverages multiple tiles as a unified entity to process an MHA block, as defined above, of a significantly larger size, given that the aggregate L1 memory of a group of tiles can now be used to collectively store the block. When N tiles are grouped together, the resulting I/O complexity becomes:

$$\mathrm{IO} = 2 \cdot H \cdot B \cdot D \cdot S \cdot \left(1 + \frac{S}{\sqrt{N} \cdot M}\right).$$

For example, when $S = 4096$, $M = 128$, and $N = 64$, this results in a $6.6\times$ theoretical reduction in HBM accesses.

B. Detailed FlatAttention Dataflow

The FlatAttention dataflow is depicted in Fig. 2b. We refer to a set of tiles collectively processing a block, as previously introduced, as a *group*, demonstrated in Fig. 2a. We define the shape of the group as $G_x \times G_y$. FlatAttention applies the same tiling and mapping scheme to groups as FlashAttention applies to tiles but introduces a secondary level of blocking within each group. This secondary blocking divides the $\{B_c, B_r\}$ block dimensions into smaller *slices* based on the group shape $\{G_x, G_y\}$, resulting in $\{\frac{B_c}{G_x}, \frac{B_r}{G_y}\}$ slice sizes for every tile.

Algorithm 2 outlines the FlatAttention dataflow. At a high level, the algorithm is conceptually similar to FlashAttention (Algorithm 1): different groups process distinct data, so no communication between groups is required. However, distributing the computation of an MHA block to tiles in a group introduces distinct data movement patterns within the group:

- **Loading and Multicasting**: Only tiles on the west edge of the group load Q slices from HBM (line ⑤), followed by multicasting Q slices row-wise ⑥ to the other tiles in the group. When entering the inner loop, the tiles on the south edge load K^T and V slices from HBM ⑧ followed by multicasting them column-wise ⑨.
- **Computing Attention ($Q \cdot K^T$) and Rowmax**: Each tile computes a segment of the attention score matrix ⑩. During the computation of row-wise maxima for Softmax, tiles compute partial row maxima locally⑪ updated with the tracking maxima⑬, followed by a row-wise reduction within the group to calculate the global row maxima⑮. The results are then multicasted row-wise to ensure that each tile holds the global row maxima ⑯.
- **Softmax Denominator**: After computing the partial Softmax denominator locally with global row maxima⑰⑱, the same reduction ⑲ and multicast ⑳ procedure applies to computing the global denominator, which is then updated with the tracking maxima and denominator ㉒.
- **Output Matrix (O)**: Each tile updates local O slices and tracking statistics in the inner loop, and computes partial results for O slices on exit ㉓-㉘. FlatAttention then performs a row-wise reduction of O slices ㉙ followed by storing O slices in HBM ㉚ only from west-edge tiles.

These communication requirements are a direct result of FlatAttention's parallelization scheme, which enables minimizing costly *global* off-chip I/O by exploiting on-chip data reuse across tiles through *local* on-chip communication. This trade-off of global for local requirements enables FlatAttention to achieve better scalability and performance compared to FlashAttention methods for tile-based many-PE architectures, as long as local on-chip communication is efficiently handled, as will be discussed in Section V-A.

C. Asynchronous FlatAttention

In the naïve version of FlatAttention (Algorithm 2), data movement and SoftMax-related computations still account for a significant portion of the runtime, as illustrated in Fig. 2c. This reduces the overall utilization, as the system's peak performance is primarily determined by the matrix engine, which has much higher computational power compared to the vector engine. To further improve utilization, we propose leveraging the asynchronous nature of DMA, vector and matrix engine invocations to overlap the runtime of data movement and SoftMax operations with matrix multiplications.

The optimized dataflow schedules the computation of two heads concurrently on each group. While the matrix engine processes matrix multiplications for one head, the DMA and vector engine perform data movement and SoftMax operations

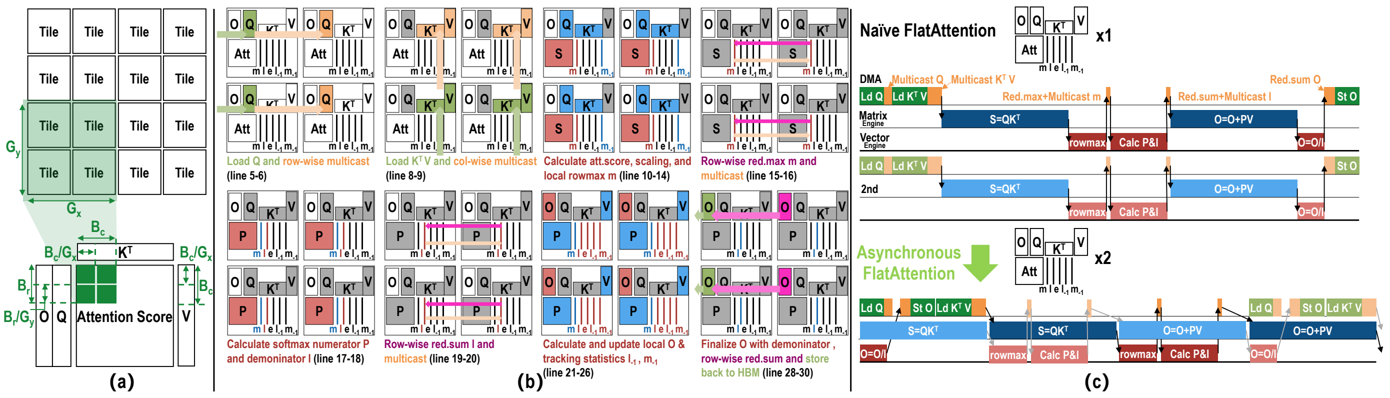
Tile
G_y
G_x
B_c
B_c/G_x
K^T
B_r
B_r/G_y
O Q
Attention Score
V
(a)
Att
m l e$_1$ m$_1$
Load Q and row-wise multicast (line 5-6)
Load K^T V and col-wise multicast (line 8-9)
Calculate att.score, scaling, and local rowmax m (line 10-14)
Row-wise red.max m and multicast (line 15-16)
Calculate softmax numerator P and demoninator l (line 17-18)
Row-wise red.sum l and multicast (line 19-20)
Calculate and update local O & tracking statistics l$_{-1}$, m$_{-1}$ (line 21-26)
Finalize O with denominator, row-wise red.sum and store back to HBM (line 28-30)
(b)
Naïve FlatAttention
x1
DMA
Matrix Engine
Vector Engine
Ld Q
Ld K^T V
Multicast Q
Multicast K^T V
Red.max+Multicast m
Red.sum+Multicast l
Red.sum O
St O
S=QK^T
O=O+PV
rowmax
Calc P&l
O=O/l
2nd
Asynchronous FlatAttention
x2
(c)

Fig. 2: (a) Parametric definition of FlatAttention. (b) Detailed FlatAttention dataflow, with each step corresponding to the line numbers in Algorithm 2. (c) FlatAttention dataflow optimization.

Algorithm 2 FlatAttention on tile-based many-PE architecture

```
Require: Input Matrices $\mathbf{Q} \in \mathbb{R}^{S\times D}, \mathbf{K}^\top \in \mathbb{R}^{D\times S}, \mathbf{V} \in \mathbb{R}^{S\times D}$, and output matrx $\mathbf{O} \in \mathbb{R}^{S\times D}$
    in HBM, block sizes $B_c, B_r$, tile group sizes $G_x, G_y$. In the following algorithm, $x, y$ denote
    tile coordinates relative to the group.
 1: Divide $\mathbf{Q}, \mathbf{O}$ into $T_r = \lceil \frac{S}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r} \in \mathbb{R}^{B_r\times D}$ and $\mathbf{O}_1, \dots, \mathbf{O}_{T_r} \in \mathbb{R}^{B_r\times D}$.
    Divide $\mathbf{K}^\top, \mathbf{V}$ into $T_c = \lceil \frac{S}{B_c} \rceil$ blocks $\mathbf{K}^\top{}_1, \dots, \mathbf{K}^\top{}_{T_c} \in \mathbb{R}^{D\times B_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c} \in \mathbb{R}^{B_c\times D}$.
 2: Further divide each $\mathbf{Q}_i, \mathbf{O}_i$ into $G_y$ slices $\mathbf{Q}_{i1}, \dots, \mathbf{Q}_{iG_y} \in \mathbb{R}^{\frac{B_r}{G_y}\times D}$ and $\mathbf{O}_{i1}, \dots, \mathbf{O}_{iG_y} \in$
    $\mathbb{R}^{\frac{B_r}{G_y}\times D}$, and divide each $\mathbf{K}^\top{}_j, \mathbf{V}_j$ into $G_x$ slices $\mathbf{K}^\top{}_{j1}, \dots, \mathbf{K}^\top{}_{jG_x} \in \mathbb{R}^{D\times \frac{B_c}{G_x}}$ and
    $\mathbf{V}_{j1}, \dots, \mathbf{V}_{jG_x} \in \mathbb{R}^{\frac{B_c}{G_x}\times D}$. Slices are accessed according to tile coordinates $x, y$
 3: for $1 \le i \le T_r$ do
 4:     On chip, initialize $\mathbf{O}_{iy}^{(0x)} = \mathbf{0}$.
 5:     ($x = 0$ edge tiles in group) Load $\mathbf{Q}_{iy}$ from HBM to tile on-chip L1.
 6:     ($x = 0$ edge tiles in group) Row-wise multicast $\mathbf{Q}_{iy}$ in tile group.
 7:     for $1 \le j \le T_c$ do
 8:         ($y = 0$ edge tiles in group) Load $\mathbf{K}^\top{}_{jx}, \mathbf{V}_{jx}$ from HBM to tile on-chip L1.
 9:         ($y = 0$ edge tiles in group) Column-wise multicast $\mathbf{K}^\top{}_{jx}, \mathbf{V}_{jx}$ in tile group.
10:         Tile on-chip compute $\mathbf{S}_{iy}^{(jx)} = \mathbf{Q}_{iy}\mathbf{K}_{jx}^\top$.
11:         Tile on-chip compute $\mathbf{S}_{iy}^{(jx)} = \frac{\mathbf{S}_{iy}^{(jx)}}{\sqrt{D}}$ , $m_{iy}^{(jx)} = \mathrm{rowmax}(\mathbf{S}_{iy}^{(jx)})$
12:         if $j > 1$ then
13:             Tile on-chip compute $m_{iy}^{(jx)} = \max(m_{iy}^{(j-1)}, m_{iy}^{(jx)})$
14:         end if
15:         ($x = 0$ edge tiles in group) Row-wise reduce $m_{iy}^{(j)} = \max(m_{iy}^{(jx)})$ in tile group.
16:         ($x = 0$ edge tiles in group) Row-wise multicast $m_{iy}^{(j)}$ in tile group.
17:         Tile on-chip compute $\tilde{\mathbf{P}}_{iy}^{(jx)} = \exp(\mathbf{S}_{iy}^{(jx)} - m_{iy}^{(j)})$.
18:         Tile on-chip compute $\ell_{iy}^{(jx)} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{iy}^{(jx)})$.
19:         ($x = 0$ edge tiles in group) Row-wise reduce $\ell_{iy}^{(j)} = \sum(\ell_{iy}^{(jx)})$ in tile group.
20:         ($x = 0$ edge tiles in group) Row-wise multicast $\ell_{iy}^{(j)}$ in tile group.
21:         if $j > 1$ then
22:             Tile on-chip compute $\ell_{iy}^{(j)} = e^{m_{iy}^{(j-1)} - m_{iy}^{(j)}} \ell_{iy}^{(j-1)} + \ell_{iy}^{(j)}$.
23:             Tile on-chip compute $\mathbf{O}_{iy}^{(jx)} = \mathrm{diag}(e^{m_{iy}^{(j-1)} - m_{iy}^{(j)}})\mathbf{O}_{iy}^{(j-1)x}$.
24:         end if
25:         Tile on-chip compute $\mathbf{O}_{iy}^{(jx)} = \mathbf{O}_{iy}^{(jx)} + \tilde{\mathbf{P}}_{iy}^{(jx)}\mathbf{V}_{jx}$.
26:         Tile on-chip update $m_{iy}^{(j-1)} \leftarrow m_{iy}^{(j)}, \ell_{iy}^{(j-1)} \leftarrow \ell_{iy}^{(j)}$.
27:     end for
28:     Tile on-chip compute $\mathbf{O}_{iy}^{(T_c)x} = \mathrm{diag}(\ell_{iy}^{(T_c)})^{-1}\mathbf{O}_{iy}^{(T_c)x}$.
29:     ($x = 0$ edge tiles in group) Row-wise reduce $O_{iy}^{(T_c)} = \sum(O_{iy}^{(T_c)x})$ in tile group.
30:     ($x = 0$ edge tiles in group) Write $\mathbf{O}_{iy}$ to HBM as the $y$-th slice in $i$-th block of $\mathbf{O}$.
31: end for
```

for the other[3]. Fig. 2c demonstrates this optimization, showcasing how it can ensure the matrix engine remains nearly fully active, provided that the matrix multiplications runtime overlaps completely with data movement and SoftMax operations. Notably, FlashAttention-3 employs the same technique to improve over FlashAttention-2.

IV. Modeling and Analysis Methodology

We developed a modeling and simulation framework for tile-based many-PE accelerators on the GVSoC event-based simulator [18] for functional and performance simulation. GVSoC is open source and released with models for the Snitch [19] single-issue RISC-V core, the Spatz [20] vector engine supporting the RISC-V Vector (RVV) extension, the iDMA [21] engine, and tile-local L1 memory and interconnect. To extend these capabilities, we developed and calibrated new models for the RedMulE [22] matrix engine and the FlooNoC [23] fabric according to their open-source RTL implementations. The NoC model incorporates both software and hardware-supported collective communication primitives, as presented in Section II, for design space exploration. Specifically, we can simulate hardware support for row-wise and column-wise multicast, sum-reduction, and max-reduction operations. We also extended Spatz with a custom RVV instruction for exponential operations and a dedicated exponential unit within the FPU. Furthermore, we integrated the DRAMSys [24] simulator into GVSoC for HBM modeling. i.e., we used the HBM2e specification with 64 GB/s bandwidth per channel.

Using these building blocks, we constructed the **SoftHier** model and analysis framework: a flexible, parameterizable tile-based many-PE accelerator simulator with functionality and performance models calibrated on cycle-accurate simulations of open-source RTL. SoftHier is configurable using architecture configuration files, enabling the instantiation of specific accelerator designs, e.g. to explore different numbers of Compute Elements (CEs) in the RedMulE units.

In the SoftHier framework, we implemented the FlashAttention and FlatAttention dataflows in C, incorporating APIs for matrix engine offload and DMA engine inter-tile communication, along with NoC collective primitives. The software was compiled using the `GNU RISC-V GCC` compiler with `-O3` optimization. We assume a 1 GHz clock frequency, and preheat every tile's instruction cache at the start of the simulation. For FlashAttention, we parallelize across the batch, number of heads and output sequence length dimensions to ensure that all tiles are utilized. For both implementations, we select the slice size per tile to maximize local L1 memory occupancy while maintaining a square configuration, i.e., $\frac{B_r}{G_y} = \frac{B_c}{G_x}$.

[3]The same optimization can be applied with two output row blocks O_i instead of two heads, reducing memory requirements as the K_j^T and V_j blocks are shared. To simplify the evaluation, where sufficient row blocks are not available in all configurations, we adopt the presented implementation.

TABLE I: System Specifications

System	32×32 Tiles, 1024-bit NoC link width
HBM	16x2 Channels, equally divided over west and south edges
Tile	**RedMulE Matrix Engine**: 32×16 CE array, 1 TFLOPS@FP16
	Spatz Vector Engine: 16 FPU, 128 GFLOPS@FP16
	Local Memory: 384 KB, 512 GB/s
Summary	1024 TFLOPS Peak Performance , 2 TB/s Peak HBM Bandwidth

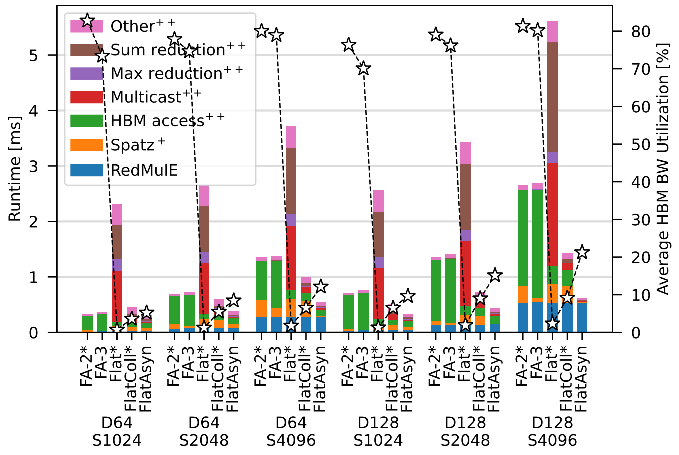

Fig. 3: Runtime breakdown (bars) and average HBM BW utilization (star markers) for different MHA implementations and layer sizes. $^{+}$Runtime not overlapped with RedMulE. $^{++}$Runtime not overlapped with either Spatz or RedMulE. *Implementations without double buffering.

V. Experimental Results

A. *FlashAttention vs. FlatAttention*

We first compare different MHA implementations and layer sizes on a given tile-based many-PE accelerator configuration, as specified in Table I. We evaluate both FlashAttention-2 (*FA-2*) and FlashAttention-3 (*FA-3*) implementations on tile-based many-PE accelerators, where *FA-3* introduces a similar scheduling mechanism as presented in Section III-C. For FlatAttention, we set the group size to include all tiles in the system, i.e. $G_x = G_y = 32$. We evaluate a naïve implementation without (*Flat*) and with (*FlatColl*) hardware support for efficient collective primitives on the NoC, as well as the optimized FlatAttention dataflow (*FlatAsyn*) described in Section III-C with NoC collective primitives. We evaluate multiple MHA layers, varying the sequence length $S \in \{1024, 2048, 4096\}$ and head dimension $D \in \{64, 128\}$, while fixing batch size $B = 2$ and heads $H = 32$.

Fig. 3 presents the runtime breakdown and average HBM bandwidth utilization. We observe that FlashAttention exhibits a highly memory-bound behavior on the target tile-based many-PE system, with HBM bandwidth utilizations reaching up to 80% on average. HBM access is the dominant runtime component, limiting overall compute utilization. Even with *FA-3*'s optimized dataflow for overlapping matrix multiplication and Softmax operations, the saturated HBM bandwidth prevents further speedup. Additionally, *FA-3* introduces an overhead for more complex scheduling.

Flat significantly reduces HBM access time compared to *FA-3* due to the decreased I/O complexity. However, software-based collective primitives used in *Flat* rely on successive

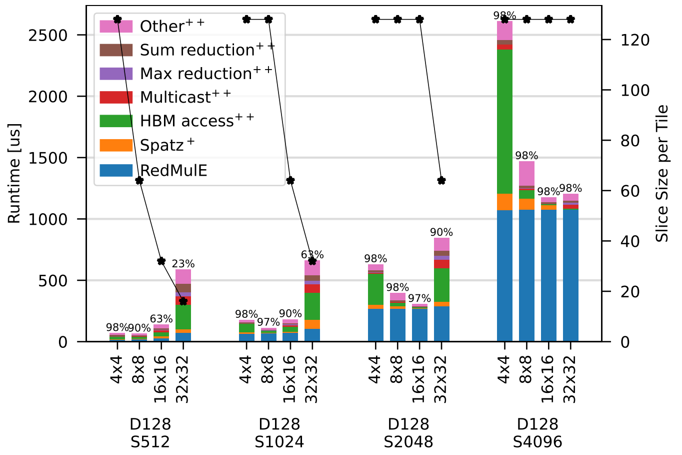

Fig. 4: Runtime breakdown for different (square) flattening scales and layer sizes. Percentage labels above the bars indicate the average utilization of the RedMulE units when active. $^{+}$Runtime not overlapped with RedMulE. $^{++}$Runtime not overlapped with either Spatz or RedMulE.

point-to-point inter-tile transfers. For instance, a row-wise multicast in this configuration requires 31 sequential unicast transmissions, incurring substantial on-chip communication overhead and resulting in worse performance than FlashAttention. Instead, with efficient collective primitives enabled on the NoC (*FlatColl*), on-chip inter-tile communication is significantly accelerated, leading to better performance than FlashAttention across most MHA layers. The *FlatAsyn* implementation shows that we can further improve performance by overlapping SoftMax, data movement, and matrix multiplication operations. Overall, our optimizations result in up to 4.1× speedup and 16× HBM traffic reduction over *FA-3* (D128, S4096).

B. *Tile Group Scale Trade-offs for FlatAttention*

Although *FlatAsyn* achieves the best performance across all MHA layers in Fig. 3, it does not fully optimize utilization for shorter sequence lengths such as 2048 and 1024, where RedMulE runtime cannot completely overlap with other operations. To determine the optimal performance configuration for FlatAttention, we analyze the impact of different (square) group sizes $G_x, G_y \in \{4, 8, 16, 32\}$, on the specific tile-based many-PE accelerator configuration defined in Table I.

We evaluate multiple MHA layers, varying the sequence length $S \in \{512, 1024, 2048, 4096\}$, while fixing $D = 128$, $H = 32$ and $B = 4$.

Fig. 4 presents the runtime breakdown and MHA workload slice size per tile, i.e. $\frac{B_r}{G_y} = \frac{B_c}{G_x}$, across different group scales. For long sequence lengths such as 4096, the slice size per tile remains constant due to L1 memory capacity. In this case, increasing the group size reduces overall HBM I/O complexity, as demonstrated in Section III-B, leading to reduced HBM access runtimes and improved overlap with matrix multiplication on RedMulE. The 16×16 and 32×32 group scales achieve 88% and 87% utilization, respectively, for a sequence length of 4096.

However, for shorter sequence lengths such as 512, the situation is different. As the group scale increases, the slice size per tile decreases due to the fixed sequence length, introducing two performance overheads:

- **Reduced RedMulE utilization**: Smaller slices per tile lead to lower RedMulE utilizations. For example, in a 32×32 group with a sequence length of 512, every tile's RedMulE achieves only 23% utilization when active.
- **Increased synchronization overhead**: Smaller slices per tile result in shorter RedMulE runtimes. As a result, constant overheads associated with synchronization and data movement, such as HBM access latency ($\sim$200 cycles), constitute a larger fraction of the overall runtime.

We refer to this effect as *over-flattening*. For moderate sequence lengths, both effects occur simultaneously: larger group scales effectively reduce I/O complexity but also introduce the risk of over-flattening. For every sequence length, there exists an optimal group scale balancing the two effects.

C. Co-exploration of Architecture and Algorithm Parameters

Lastly, we demonstrate how the SoftHier framework can be used to identify an optimal tile-based many-PE architecture configuration. Our goal is to design a tile-based accelerator with comparable peak performance to Nvidia's H100 (989 TFLOPS FP16/BF16 without sparsity) while improving utilization and reducing overall HBM bandwidth requirements on MHA workloads. We evaluated a set of candidate architecture configurations with varying NoC fabric granularity and HBM channel connectivity. Table II presents the tile specifications as a function of fabric granularity, ensuring constant peak system performance (1024 TFLOPS) and on-chip memory capacity. For every accelerator candidate, we evaluate multiple MHA layers, searching for optimal performance across different dataflow implementations, including FlashAttention-3 and FlatAttention with varying square-shaped group sizes.

Based on the results shown in Fig. 5a, we can select a configuration (*BestArch*) that optimizes for performance over cost, featuring a 32×32 fabric granularity and 16×2 HBM channels. We then compare its performance directly against FlashAttention-3, based on the H100 performance numbers in Shah et al. [6], using the same MHA layers while also accounting for the K matrix pre-transposition time in FlatAttention for fair comparison. In Fig. 5b, the *BestArch* configuration with FlatAttention achieves up to 1.3× higher utilization while requiring 40% less HBM bandwidth compared to the H100 GPU. Beyond MHA, common GEMM kernels utilizing the collective-based SUMMA dataflow [25] on *BestArch* also achieve up to 1.2× higher utilization over H100 [26] in Fig. 5c. Using the Gate Equivalent (GE) reported for the individual components [19]–[23], we estimated the die size of *BestArch* in TSMC 5nm technology, the same used by the H100. Considering 4 transistors per GE, a transistor density of 138.2 MTr/mm^2, an SRAM bit-cell size of 0.021 µm^2, and assuming 66% area utilization, *BestArch* features a die size of 457 mm^2, enabling a 1.8× reduction to H100 GPU.

VI. Conclusion

We propose FlatAttention, an optimized dataflow for MHA on tile-based many-PE accelerators, co-designed with NoC collective primitives, achieving up to 89.3% utilization while reducing HBM traffic by 16× compared to FlashAttention-3 dataflow, on tile-based accelerators. Through algorithm-architecture co-exploration, an optimal accelerator configuration matching the peak performance of Nvidia's H100 GPU is selected, which requires 40% less HBM bandwidth than H100 and 1.8× reduction in die size, with FlatAttention achieving up to 1.3× higher utilization compared to FlashAttention-3 on H100, and its GEMM reaching up to 1.2× higher utilization over H100. Future work includes end-to-end LLM inference on multi-chiplet systems with 3D-stacked memory.

TABLE II: Fabric Granularity and Tile Specifications

Fabric Granularity	**32×32**	**16×16**	**8×8**
RedMulE CE Array	32×16	64×32	128×64
Spatz FU Count	16	64	256
Local Memory Size (KB)	386	1526	6144
Local Memory Bandwidth (GB/s)	512	2048	8192

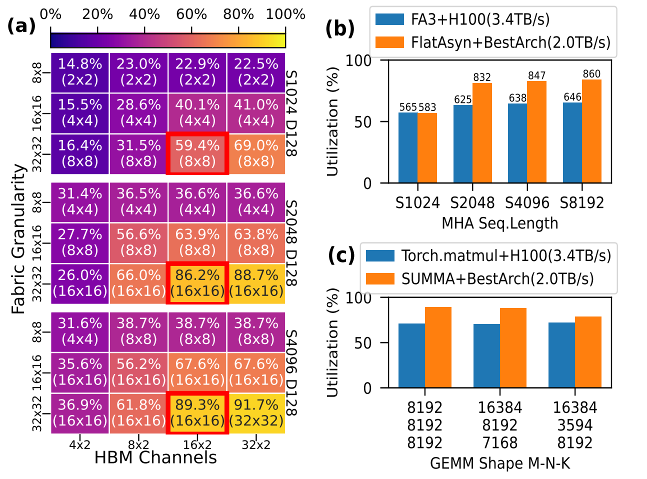

Fig. 5: (a) Heatmap of utilization with best group size. (b) Comparison with FlashAttention-3 on H100 solutions, with absolute performance in TFLOPS labeled above the bar. (c) Comparison of GEMMs, including FFN layer in Meta's LLaMA 70B [26], between *BestArch* and H100.

Acknowledgment

This work is supported by the ETH Future Computing Laboratory (EFCL) and Huawei ZRC.

References

[1] F. D. Keles *et al.*, "On the computational complexity of self-attention," in *ALT*, 2023.
[2] Z. Yuan *et al.*, "LLM inference unveiled: Survey and roofline model insights," *arXiv preprint arXiv:2402.16363*, 2024.
[3] A. Ivanov *et al.*, "Data movement is all you need: A case study on optimizing transformers," in *MLSys*, 2021.
[4] T. Dao *et al.*, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *NeurIPS*, 2022.
[5] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in *ICLR*, 2024.
[6] J. Shah *et al.*, "Flashattention-3: Fast and accurate attention with asynchrony and low-precision," *arXiv preprint arXiv:2407.08608*, 2024.
[7] R. Prabhakar *et al.*, " SambaNova SN40L: Scaling the AI memory wall with dataflow and composition of experts," in *MICRO*, 2024.
[8] S. Lie, "Wafer-scale AI: GPU impossible performance," in *HCS*, 2024.
[9] J. Vasiljevic *et al.*, "Blackhole & TT-Metalium: The standalone AI computer and its programming model," in *HCS*, 2024.
[10] S. M. Lee *et al.*, "16.2 rngd: A 5nm tensor-contraction processor for power-efficient inference on large language models," in *ISSCC*, 2025.

[11] M. Thüning, "Attention in SRAM on tenstorrent grayskull," *arXiv preprint arXiv:2407.13885*, 2024.
[12] S.-C. Kao *et al.*, "FLAT: An optimized dataflow for mitigating attention bottlenecks," in *ASPLOS*, 2023.
[13] N. Nayak *et al.*, "Fusemax: Leveraging extended einsums to optimize attention accelerator design," in *MICRO*, 2024.
[14] J. Cai *et al.*, "Gemini: Mapping and architecture co-exploration for large-scale DNN chiplet accelerators," in *HPCA*, 2024.
[15] E. Talpes *et al.*, "DOJO: The microarchitecture of Tesla's exa-scale computer," in *HCS*, 2022.
[16] R. van de Geijn *et al.*, "Collective communication," in *Encyclopedia of Parallel Computing*. Springer US, 2011, pp. 318–327.
[17] T. Krishna *et al.*, "Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication," in *MICRO*, 2011.
[18] N. Bruschi *et al.*, "GVSoC: a highly configurable, fast and accurate full-platform simulator for RISC-V based IoT processors," in *ICCD*, 2021.
[19] F. Zaruba *et al.*, "Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads," *IEEE TCOMP*, 2020.
[20] M. Perotti *et al.*, "Spatz: Clustering compact RISC-V-based vector units to maximize computing efficiency," *IEEE TCAD*, 2025.
[21] T. Benz *et al.*, "A high-performance, energy-efficient modular DMA engine architecture," *IEEE TCOMP*, 2023.
[22] Y. Tortorella *et al.*, "RedMule: A mixed-precision matrix–matrix operation engine for flexible and energy-efficient on-chip linear algebra and TinyML training acceleration," *FGCS*, 2023.
[23] T. Fischer *et al.*, "FlooNoC: A 645-Gb/s/link 0.15-pJ/B/hop open-source NoC with wide physical links and end-to-end AXI4 parallel multistream support," *IEEE TVLSI*, 2025.
[24] M. Jung *et al.*, "DRAMSys: a flexible DRAM subsystem design space exploration framework," *IPSJ T-SLDM*, 2015.
[25] V. D. Geijn *et al.*, "SUMMA: Scalable universal matrix multiplication algorithm," in *Concurrency: Practice and Experience*. Wiley Online Library, 1997, pp. 255–274.
[26] D. Schor *et al.*, "MI300X vs H100 vs H200 Benchmark Part 1: Training," 2024. [Online]. Available: https://semianalysis.com/2024/12/22/mi300x-vs-h100-vs-h200-benchmark-part-1-training/