
USENIX ATC '25

# WIC: Hiding Producer-Consumer Synchronization Delays with Warp-Level Interrupt-based GPU Communications

Jiajian Zhang, Fangyu Wu*, Hai Jiang, Qiufeng Wang, Genlang Chen, Chaoyi Pang

Presenter: Jiajian Zhang
Xi'an Jiaotong-Liverpool University
USENIX ATC 2025 | July 8, 2025




# Inter-device Communication

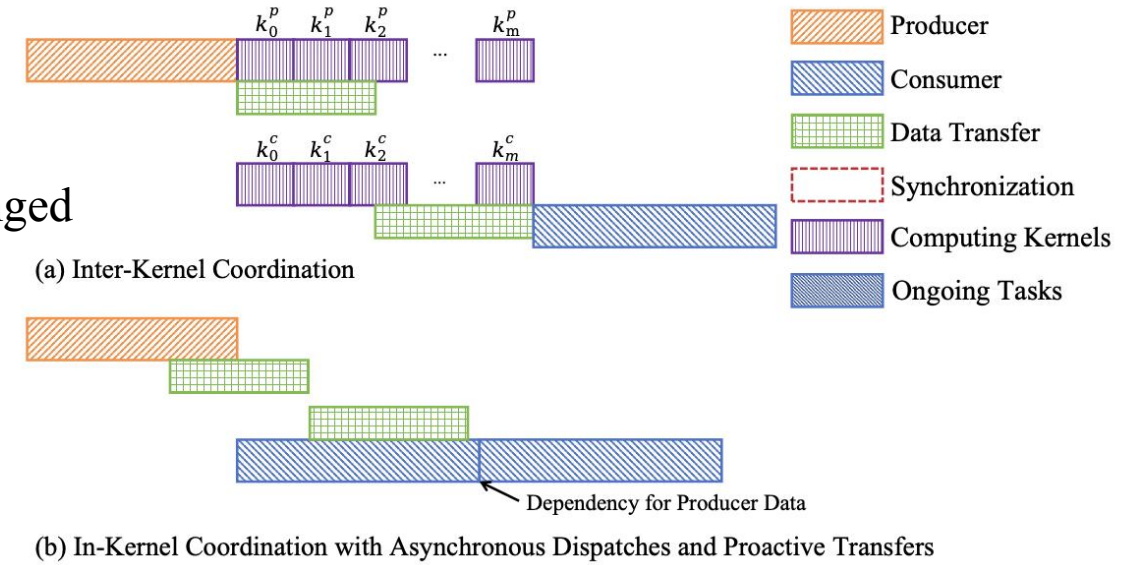


- Definition: Inter-device communication includes both **GPU–GPU** and **GPU–Host** data exchange
- Crucial for large-scale collaborative computation across devices
- Advancements in fabrics and architectures
  - GPUDirect P2P, RDMA, NVLinks
  - improve raw data movement
- Major remaining issue: Producer–Consumer synchronization
- Frequent polling adds overhead and increases system complexity

# Literature Reviews



- Inter-kernel[1][2][3]
  - Overlap Sync.
  - Via computing kernels
  - Throughput improvement
  - Computation time unchanged
- In-Kernel
  - Producer-side[4][5]
    - Async Sender
    - Transfer Threads
    - Page placement for dispatch
  - Consumer-side[6]
    - Proactive transfers -- Consumers needn't wait




[1] Augonnet et al. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Euro-Par Parallel Processing, 2009
[2] Bak et al. Task-graph scheduling extensions for efficient synchronization and communication. ICS 2021
[3] Gerzhoy et al. Pipelined cpu-gpu scheduling to reduce main memory accesses. In Proceedings of the International Symposium on Memory Systems, pages 1–10, 2021.
[4] Muthukrishnan et al. Finepack: Transparently improving the efficiency of fine-grained transfers in multi-gpu systems. HPCA 2023
[5] Muthukrishnan et al. Efficient multi-gpu shared memory via automatic optimization of fine-grained transfers. ISCA 2021.
[6] Muthukrishnan et al. Gps: A global publish-subscribe model for multi-gpu memory management. MICRO 2021

# Analysis Methodology



- A novel classification for benchmark selection
  - Problem with traditional method
    - Macroscopic view
    - Memory access patterns ≠ actual communication behavior
    - Example: Bitonic Sort - random Vs. gather-scatter
- A New Angle on Communication Patterns
  - Dual-dimensional
    - Communication access patterns
      - streaming
      - adjacent
      - scatter-gather
      - random
    - Communication models
      - Defined by data transfer per iteration
      - Synchronization methods

Table 1: Communication Models

| | |
|---|---|
| **Unidirectional Communication** $(D_0 \rightarrow D_1)^N$ This model represents one-way communication between two devices ($D_{0,1}$). $D_0$ consistently acts as the producer, while $D_1$ always functions as the consumer. $N$ represents the number of communication iterations. | **Alternating Communication** $(D_0 \rightarrow D_1, \quad D_1 \rightarrow D_0)^N$ Within a single communication iteration, $D_0$ starts as the producer, sending messages to $D_1$. After processing these messages, $D_1$ then acts as the producer, sending the resultant data back to $D_0$. Both devices alternate roles between producer and consumer throughout a communication iteration. |
| **Multiple Communication** $((D_0 \rightarrow D_1)^{n_0}, \quad (D_1 \rightarrow D_0)^{n_1})^N$ This model involves multiple transmissions between devices within a single communication iteration. $D_0$, acting as the producer, transfers messages to $D_1$ multiple times ($n_0$), and after processing, $D_1$ transfers messages back to $D_0$ $n_1$ times. | **Probabilistic Communication** $((D_0 \rightarrow D_1)^{\varepsilon_i^0 \cdot n_0}, \quad D_1 \rightarrow D_0)^{\varepsilon_i^1 \cdot n_1})^N$ This model characterizes the communication between devices where the mode and frequency of transfers within each communication cycle are probabilistic. |

# Analysis Methodology



- Benchmark Selection
  - 10 apps from 6 benchmark suites
  - based on different communication patterns
  - Avoid benchmarking crimes
- System Specifications
  - CPU-GPU Setup
    - 1 NVIDIA RTX 4090 GPU
    - 1 14900KF CPU
  - Inter-GPU Setup
    - 4 NVIDIA A800 GPUs
    - 2 Xeon 6138 CPUs

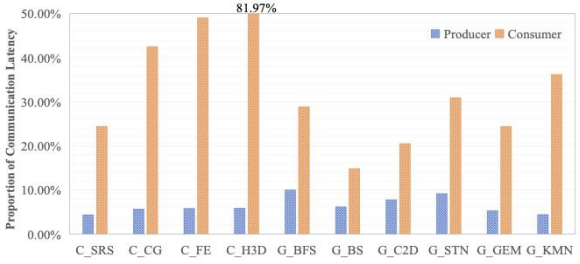
Table 2: List of Applications

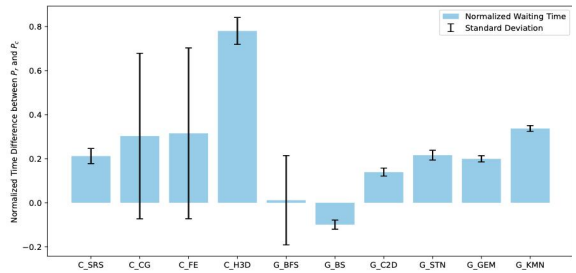
| Abbr. | Applications | Benchmark Suite | Comm. Model | Access Pattern |
|---|---|---|---|---|
| C_SRS | Satellite Remote Sensing Image Processing and Distribution | FAIR1M | streaming | Unidirectional |
| C_CG | High Performance Computing Conju-gate Gradients | Mantevo | Random | Alternating |
| C_FE | Finite Element Mini-Application | Mantevo | Random | Alternating |
| C_H3D | Halo 3D Cube | Comb | Adjacent | Multiple |
| G_BFS | Breadth-first Search | Rodinia | Random | Probabilistic |
| G_BS | Bitonic Sort | Savina | Scatter-Gather | Multiple |
| G_C2D | Convolution 2D | Polybench | Adjacent | Alternating |
| G_STN | Stencil | Polybench | Adjacent | Multiple |
| G_GEM | General Matrix Multiplication | Polybench | streaming | Multiple |
| G_KMN | Kmeans | Rodinia | streaming | Unidirectional |

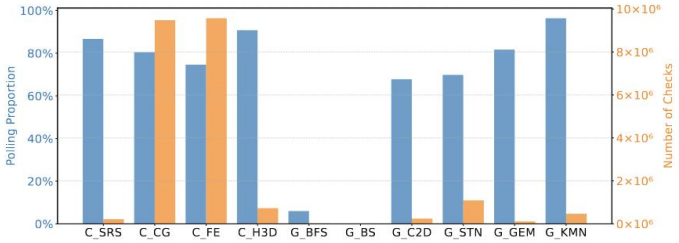
# Motivation



- Observation 1: The **consumer side** exhibits significantly higher latencies compared to the producer side.
- Observation 2: In most applications, consumers face **substantial waiting times** for producer data, with significant variability across and within applications.
- Observation 3: Consumer-side polling, where consumers **repeatedly check** the availability of the producer's data, constitutes a significant portion of the consumer overhead.



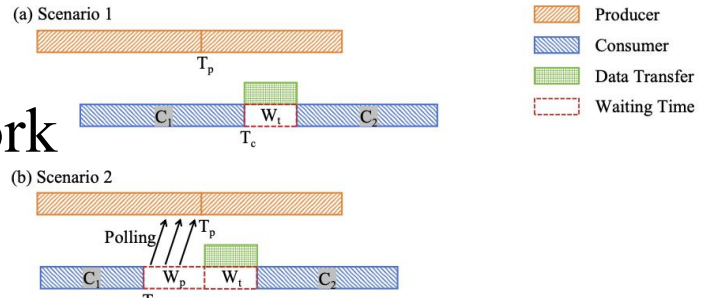



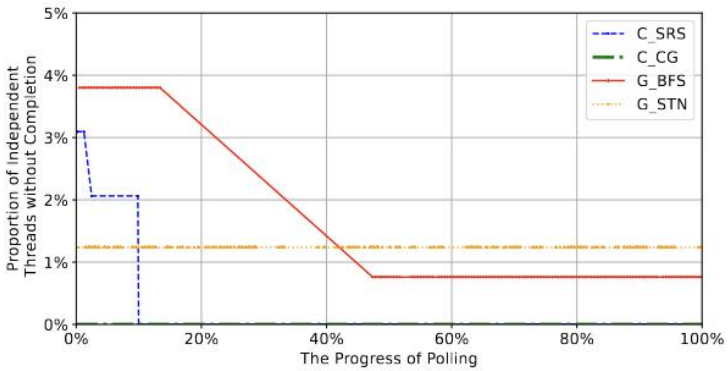



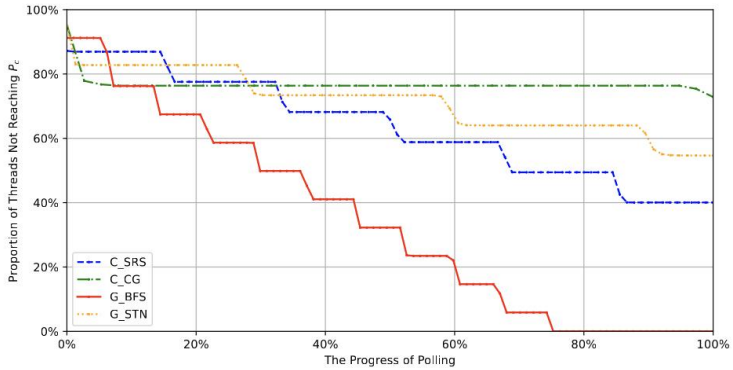
# Motivation



- Observation 4: Consumer's polling significantly impacts the completion of computational tasks before reaching the $T_c$.
- Observation 5: Consumer's polling significantly disrupts the computational tasks of independent threads.
- Polling often starts too early
- Threads sit idle Vs. doing useful work



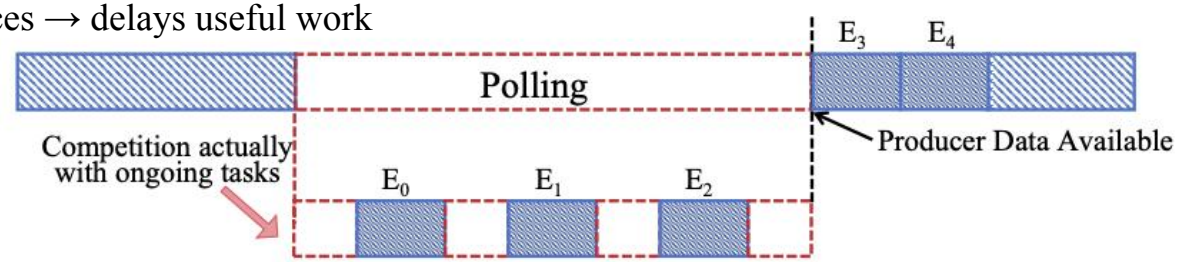



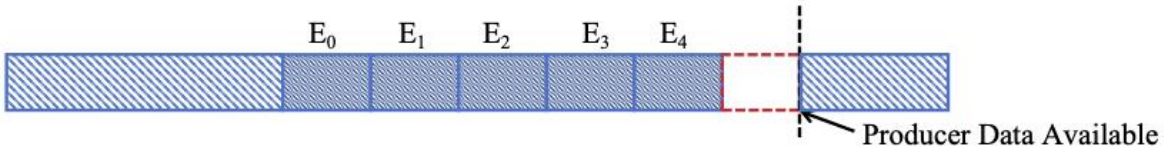
# Motivation (Takeaway)



- In CPU producer-consumer models
  - Polling starts after all work finishes
  - No opportunity to overlap — passive waiting
  - Interrupts vs polling → similar impact
- GPU Reality: Thousands of Threads
  - Threads finish at different times
  - Polling warps start too early
    - Compete with unfinished warps
    - Steal resources → delays useful work




(c) Detailed Analysis of GPU Producer-Consumer Model

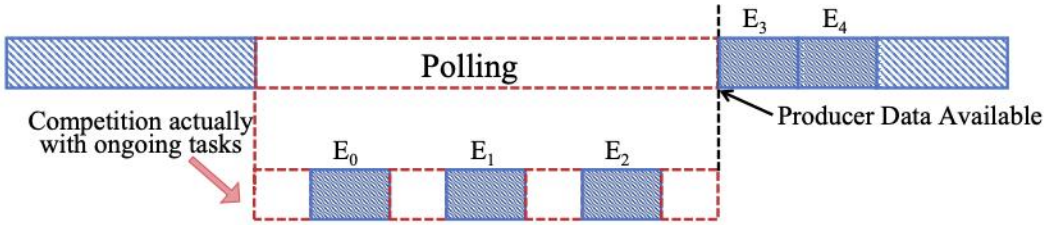



(d) Synchronization Hidden behind Ongoing Computations ($E_3$ and $E_4$) with WIC

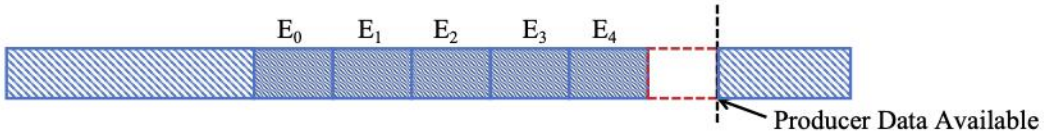
# Main Idea



- Detect early polling warps
- Preempt & suspend them
- Allow unfinished warps to continue
- Turns sync from blocking → background




(c) Detailed Analysis of GPU Producer-Consumer Model

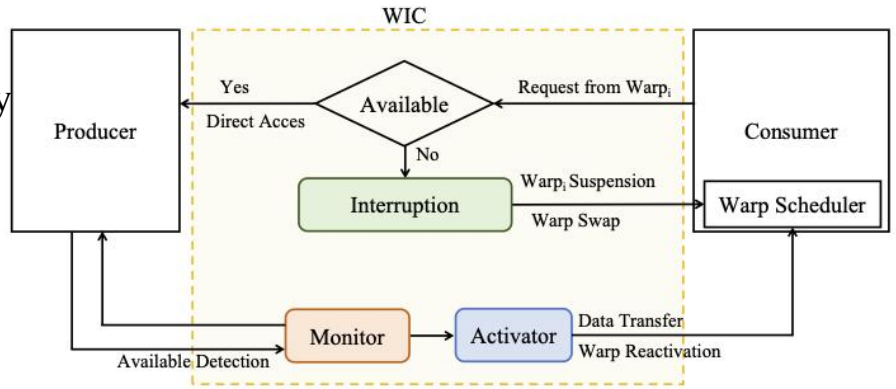



(d) Synchronization Hidden behind Ongoing Computations ($E_3$ and $E_4$) with WIC

# WIC's Overview



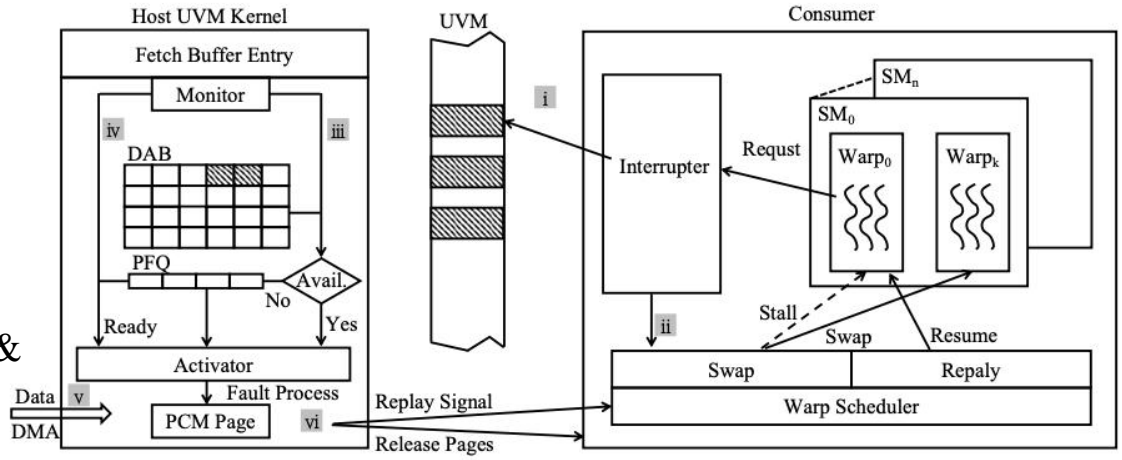
- Three Core Modules
  - Interrupter
    - Detects warps polling for producer data
    - Preempts and swaps in other ready warps
  - Monitor
    - Tracks when producer data becomes available
  - Activator
    - Resumes stalled warps when data is ready
- Integration with UVM
  - Uses UVM page placements
    - Transfer messages
    - Track data readiness
- Implemented via UVM host kernel modifications
  - No manual polling or sync logic required
  - WIC handles sync like a system call



# How WIC Works



- Consumer Sides
  - 1. Warp Sends Data Request
    - Interrupter receives request
    - Allocates PCM page
  - 2. Warp Accesses PCM Page
    - Triggers UVM page fault → warp is stalled
- Host-Side Processing
  - 3. Monitor Module
    - Captures PCM fault
    - Checks producer data availability
  - 4.
    - If Not Ready: Suspend fault, queue into PFQ
    - If Ready: Update DAB, scan PFQ & Send to Activator
- Data Transfer and Replay
  - 5. Activator Module
    - Write producer data to PCM page
    - Send replay signal to resume warp
  - 6. Post-processing
    - PCM page recycled



# Evaluation

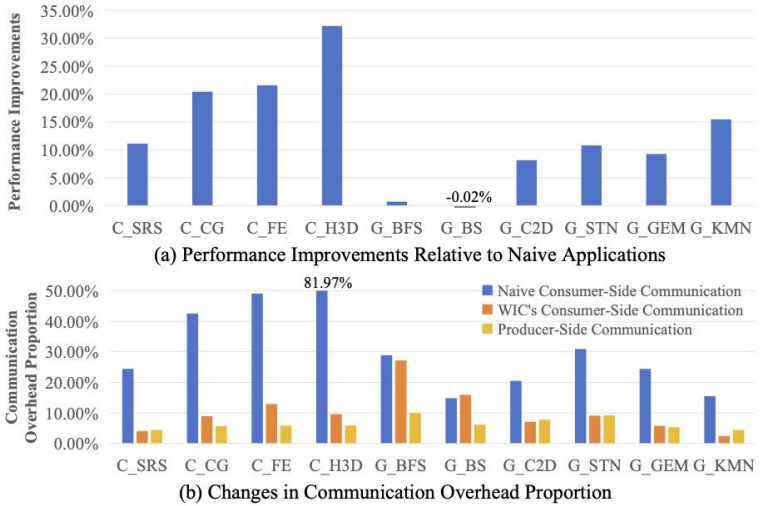


- Overall Performance vs. Traditional Communication
- Task Overlap Effectiveness (from Motivation)
- WIC Overhead Analysis
- Scalability with Thread Count
- Comparison with State-of-the-Art (SOTA) Systems

# Overall Performance



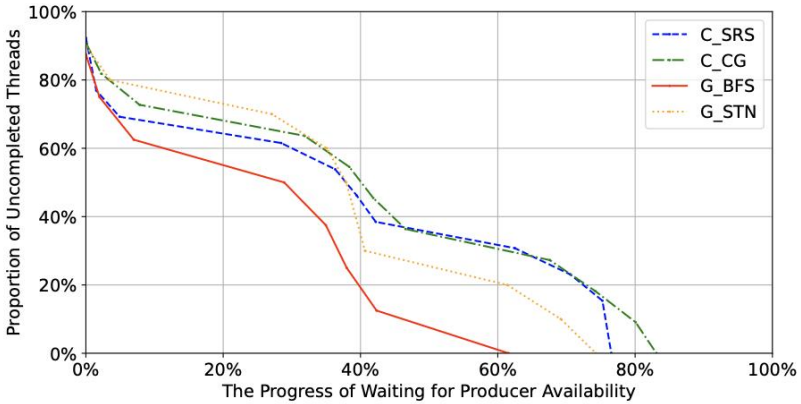
- Evaluated
  - Communication performance
  - Overhead breakdown
- Key Observations:
  - Significant performance improvement
  - Broad scenario compatibility
  - Reduced consumer-side overhead → better runtime

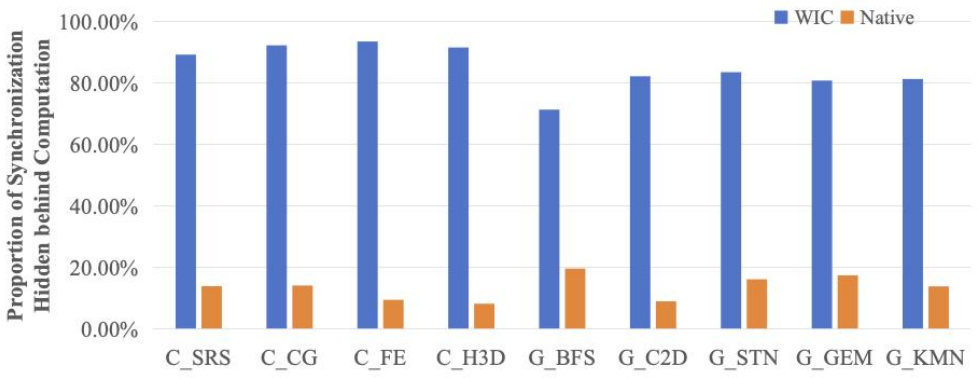


# Task Overlap Effectiveness



- Goal: Overlap more computation with waiting time
- Observation 1
  - Majority of tasks complete before waiting begins
- Observation 2
  - Synchronization effectively hidden behind computation

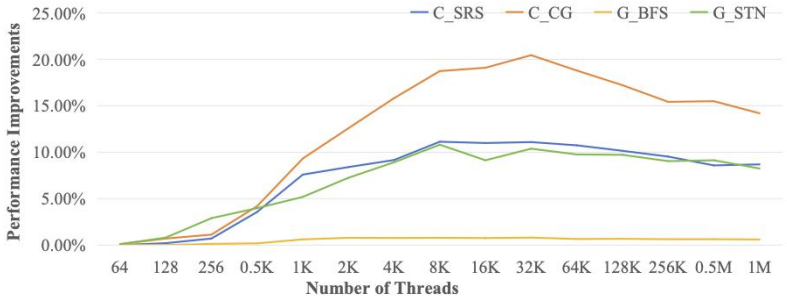


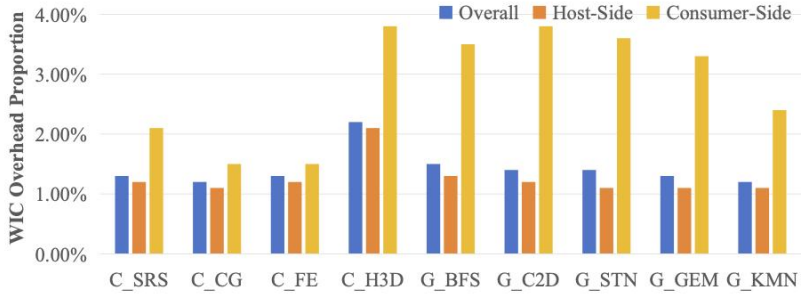



# Scaling & Overheads



- Scaling Evaluation
  - Tested from 64 to 1 million threads
  - WIC consistently improves performance at all scales
- Overhead Analysis
  - Overhead split: Consumer side & Host side
  - Total overhead is minor and stable across applications

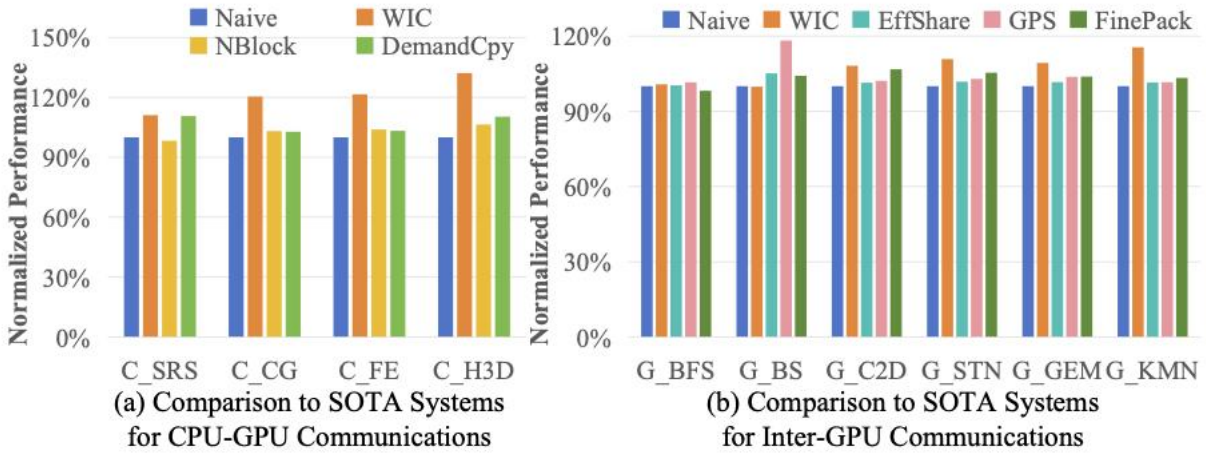


# Comparison with SOTA



- Baselines
  - CPU–GPU: NBlocking[HPC–Aisa2024] , DemandCpy[IEEE Access 2022]
  - Inter–GPU: EffShare[ISCA 2021], GPS[MICRO 2021], FinePack[HPCA 2023]
- Results
  - WIC outperforms most SOTA systems
  - Effective across diverse communication patterns
  - Superior at hiding sync overhead



# Thanks for Listening!

We sincerely thank our shepherd, Aurojit Panda, and the anonymous reviewers for their insightful and constructive comments.

Jiajian Zhang
jiajianz94@gmail.com
Xi'an Jiaotong-Liverpool University
USENIX ATC 2025 | July 8, 2025