# Maximizing the Utilization of GPUs Used by Cloud Gaming through Adaptive Co-location with Combo

Binghao Chen, Han Zhao, Weihao Cui, Yifu He, Shulai Zhang, Quan Chen, Zijun Li, Minyi Guo

SoCC'23

# Introduction

- Adaptive co-location in cloud game
  - **RT core**
  - CUDA core
  - Tensor core

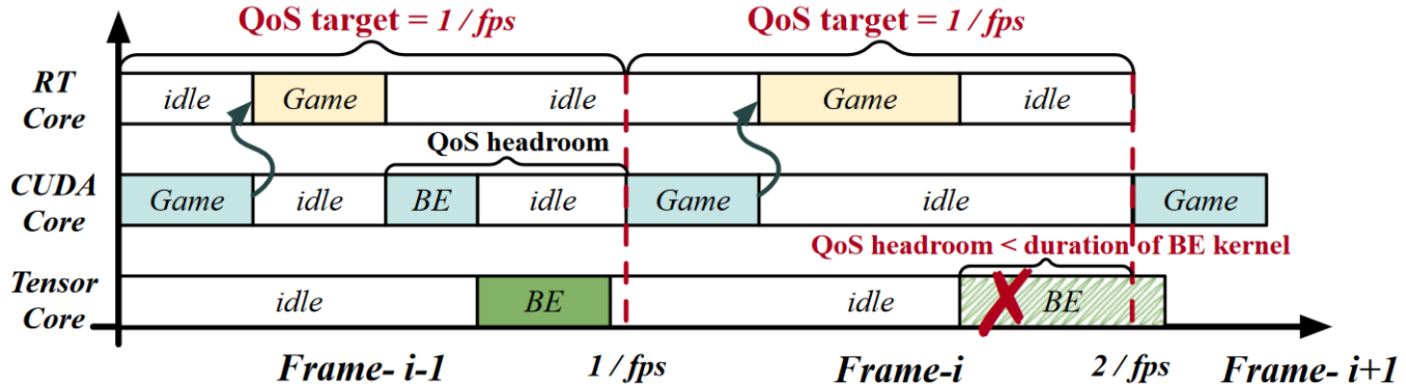



**Figure 1: Active timeline of RT/CUDA/Tensor Cores when PilotFish co-locates a game and BE task.**

# Introduction

- kernel's duration in workflow

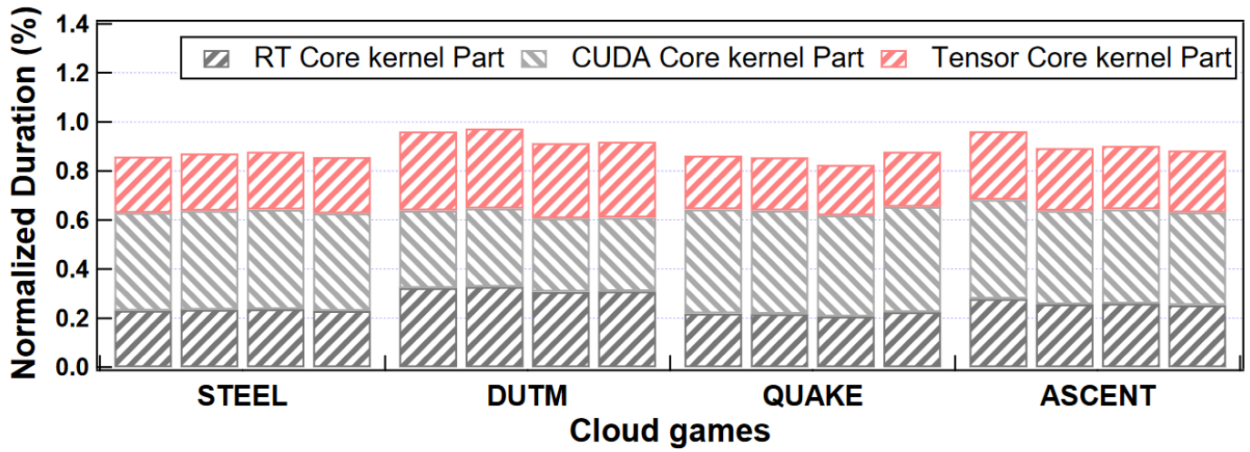



**Figure 2: The active time of the kernels with Pilotfish.**

**Table 1: Specifications of an Nvidia RTX 3090 GPU.**

| Resource | Value | Resource | Value |
|---|---|---|---|
| Number of SMs | 82 | Max Threads per SM | 1024 |
| RT Cores per SM | 1 | Tensor Cores per SM | 4 |
| CUDA Cores per SM | 128 | Shared Memory per SM | 64 KB |

**Table 2: Configurations of Cloud games.**

| Cloud games | Configurations |
|---|---|
| Served Steel (STEEL) | High quality: 2560*1440; FPS: 120 |
| Deliver Us The Moon (DUTM) | High quality: 2560*1440; FPS: 120 |
| Quake2(QUAKE) | High quality: 2560*1440; FPS: 120 |
| The Ascent(ASCENT) | High quality: 2560*1440; FPS: 120 |

# Motivation

- Intra-SM sharing: Unexploited intra-SM parallelism.

$$Makespan\ Reduction = \frac{T_1 + T_2 - T_{colo}}{T_1 + T_2}$$

**Table 3: The *Makespan Reduction* of six possible co-located kernel pairs.**

| Co-located kernel pair | Makespan Reduction |
|---|---|
| $Kernel_{RT}$ + $Kernel_{CD}$ | 39.87% |
| $Kernel_{RT}$ + $Kernel_{TC}$ | 34.62% |
| $Kernel_{CD}$ + $Kernel_{TC}$ | 44.12% |
| $Kernel_{RT}$ + $Kernel_{RT}$ | 0% |
| $Kernel_{CD}$ + $Kernel_{CD}$ | 0% |
| $Kernel_{TC}$ + $Kernel_{TC}$ | 0% |

# Motivation

- Inter-SM sharing: Under-utilized free GPU cycles.

**Table 4: The SM ratio required by cloud games on RTX 3090. The fps target is set to 120fps.**

| Game | High Quality | Medium Quality | Low Quality |
|---|---|---|---|
| Served Steel | 100% | 68.97% | 23.44% |
| DUTM | 100% | 73.17% | 47.62% |
| Quake 2 | 100% | 72.29% | 44.12% |
| Ascent | 100% | 39.73% | 25.64% |

# Challenge

- **Lack of implementations** for fine-grained management of RT kernels.
- There is a large decision space involved in achieving the theoretical improvement of **two-level SM sharing**.
- Hard to ensure each frame's QoS under varying rendering loads with spatial sharing enabled.

# System Overview

- PTB adaptor
- duration predictor
- Co-location scheme selector

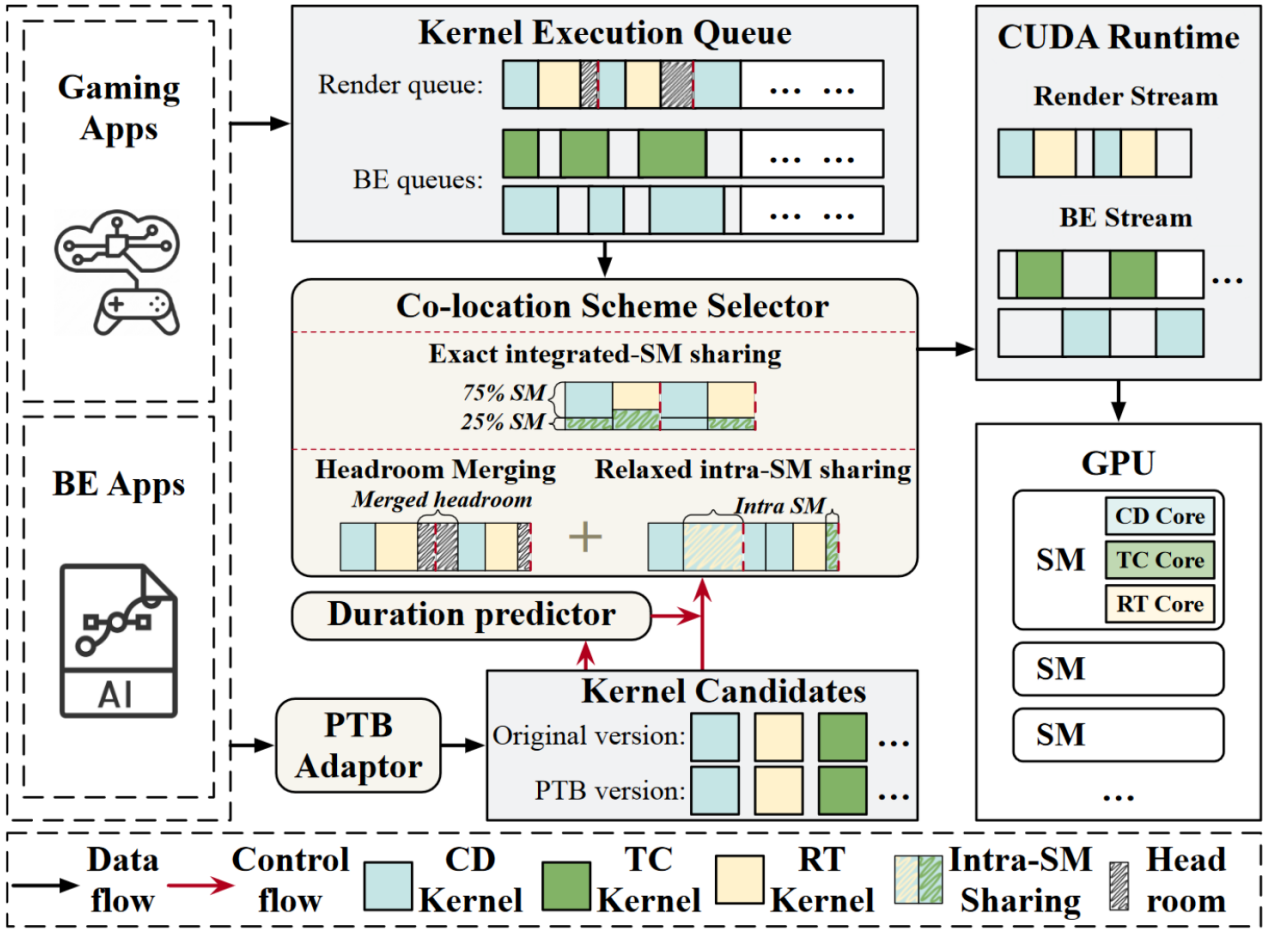



Figure 3: The overview of COMBO.

# PTB Adaptor

- Persistent-Thread-Block method (PTB)

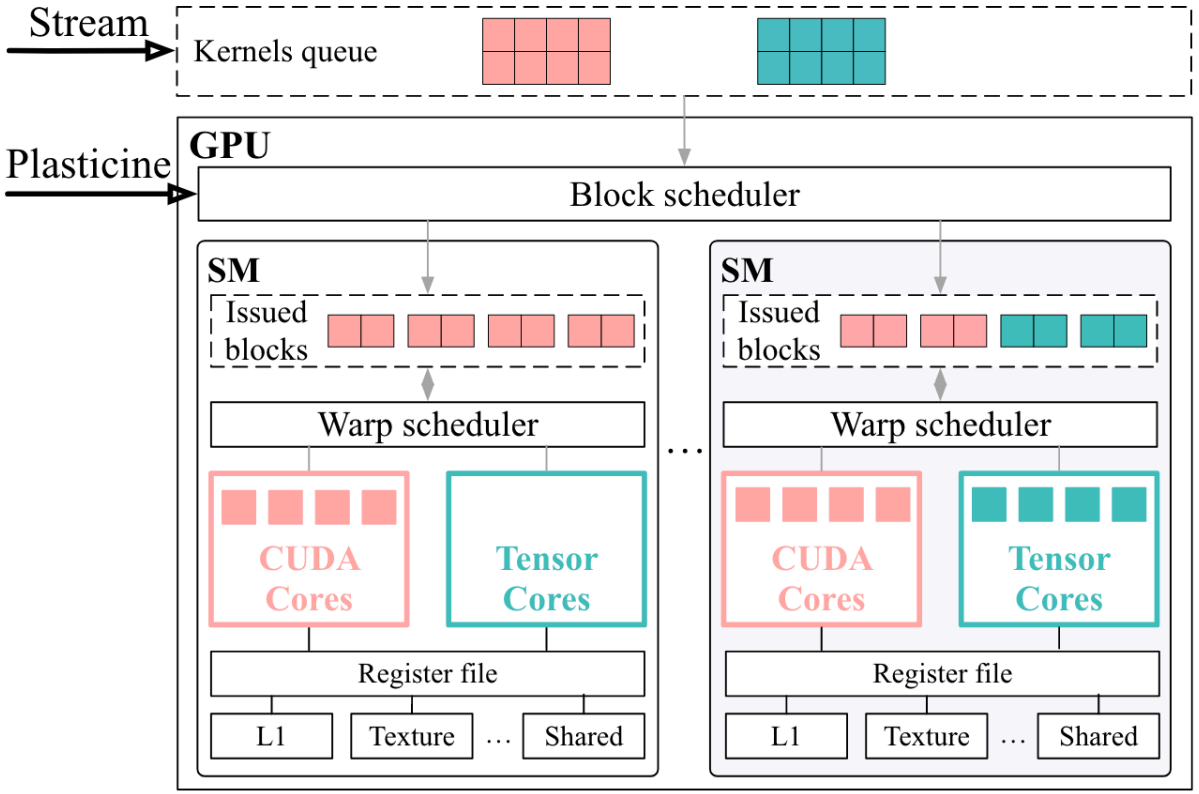



Fig. 1: Difference between Plasticine and prior work.

# PTB Adaptor

- Transfer Render kernel to PTB render kernel

```
__global__ void render_ori(params){
    int idx_x = optixGetLaunchIndex().x;
    int idx_y = optixGetLaunchIndex().y;
    render_pixel(params, idx_x, idx_y)
}

#define blocksize 128
__global__ void render_ptb(params){
    int threadid = blockIdx.x * blocksize +
↪   threadIdx.x;
    for (int j = threadid; j < params.width *
↪   params.height; j += SM_NUM * blocksize){
        int idx_x = threadid % params.width;
        int idx_y = threadid / params.height;
        render_pixel(params, idx_x, idx_y)
    }
}
```

Figure 5: An example to construct PTB render kernel.

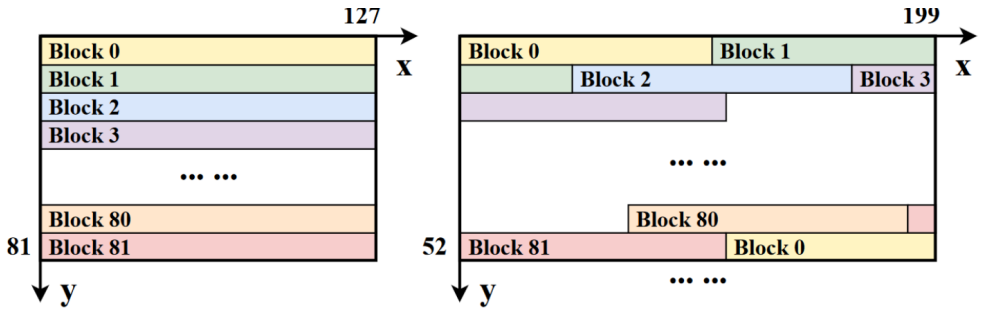
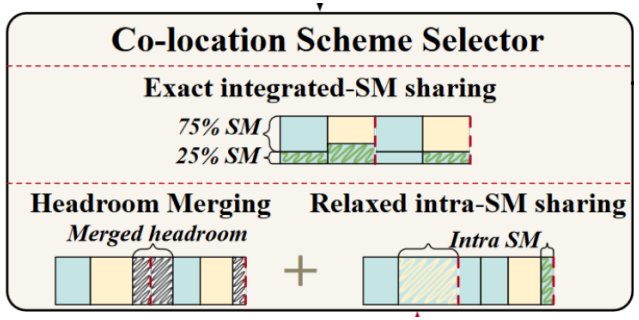



Figure 6: Examples of rendering pixels with persistent thread blocks. The number of SMs is 82 here and each SM only possesses one persistent thread block. The x-axis represents the x coordinates of pixels and the y-axis represents the y coordinates of pixels.

# Co-location scheme selector

- Exact Integrated SM Sharing.



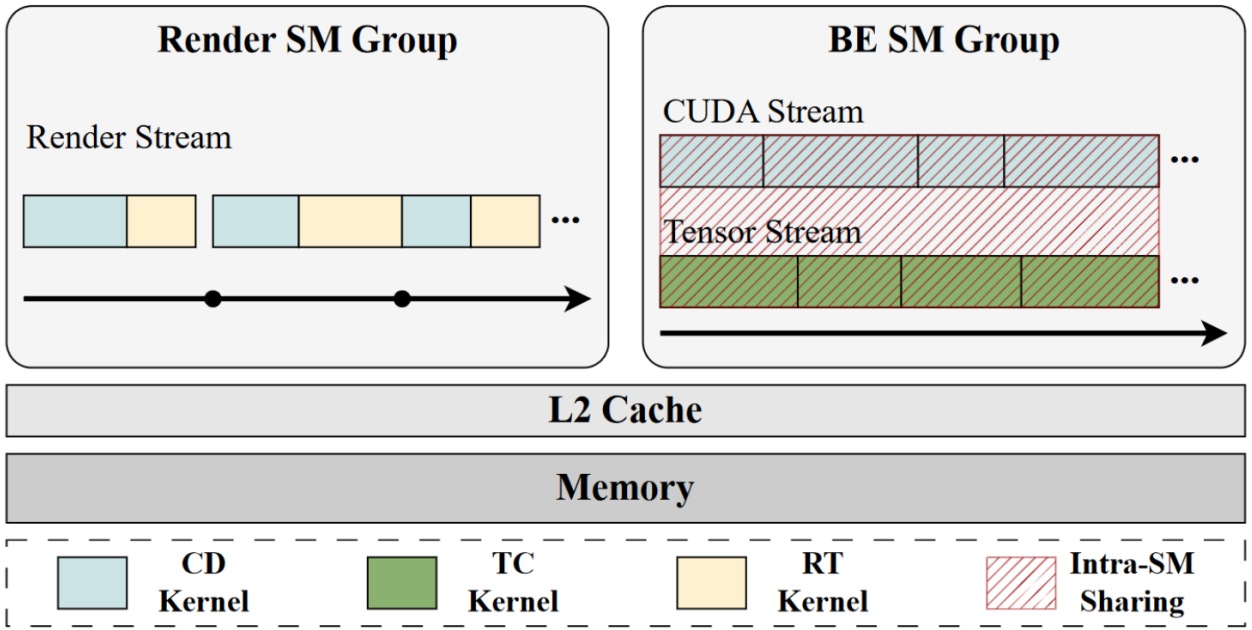




Figure 9: Timeline of exact integrated SM sharing.

# Duration predictor

- Partitioning SM Group.

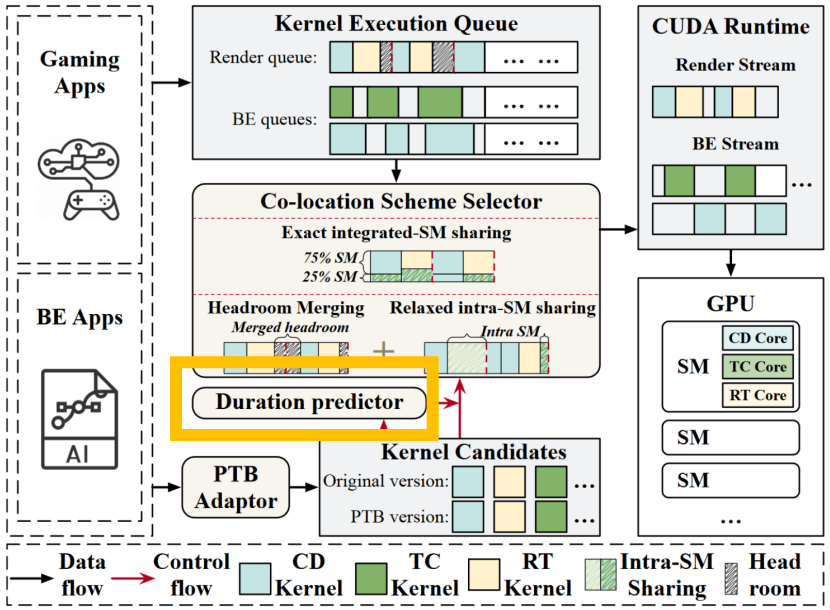



Figure 3: The overview of COMBO.

$$rendertime_{estimate} = time_{rt} + time_{cuda}$$

$$time_{rt} = a \times height \times width \times sample$$

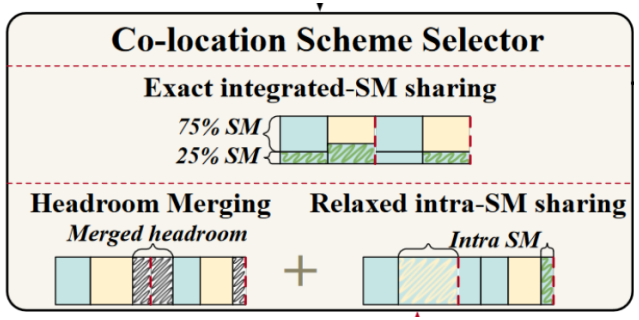
$$time_{cuda} = b \times height \times width \times num_{triangle}$$

$$SM_{render} = SM_{total} \times \frac{QoS\ target}{rendertime_{estimate}} = SM_{total} \times \frac{1}{fps \times rendertime_{estimate}}$$

$$SM_{compute} = SM_{total} - SM_{render}$$

# Co-location scheme selector

- Exact Integrated SM Sharing.
  - CUDA stream



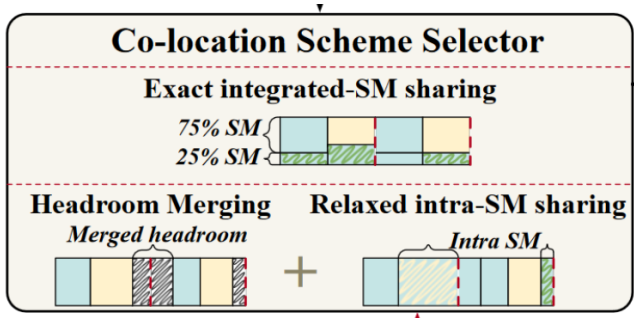

```
#define blocksize 128
__global__ void render_sm_bounding(params,
↪  sm_split_num){
    if (SM_id >= sm_split_num){
        return;
    }
    int threadid = blockIdx.x * blocksize +
↪  threadIdx.x;
    for (int j = threadid; j < params.width *
↪  params.height; j += sm_split_num * blocksize){
        int idx_x = threadid % params.width;
        int idx_y = threadid / params.height;
        render_pixel(params, idx_x, idx_y)
    }
}

__global__ void compute_sm_bounding(params,
↪  sm_split_num){
    if (SM_id < sm_split_num){
        return;
    }
    int block_id = SM_id - sm_split_num;
    int thread_id = threadIdx.x;
    for (;block_id < grid_dim; block_id +=
↪  (SM_NUM-sm_split_num)){
       compute_kernel(params, block_id, thread_id);
    }
}
```

**Figure 8: An example to restrict SM usage of render kernel and compute kernel for inter-SM sharing.**

# Co-location scheme selector

- Relaxed Intra-SM Sharing.
  - RT & CD kernel
  - TC & CD kernel



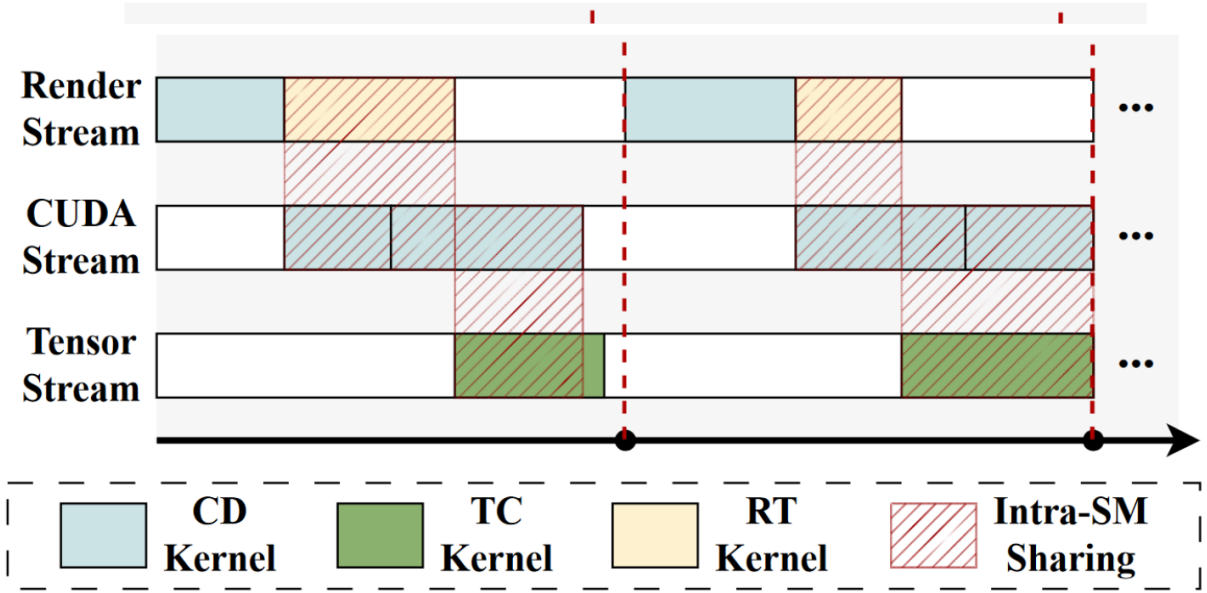




Figure 10: Timeline of relaxed intra-SM sharing.

# Co-location scheme selector

- Relaxed Intra-SM Sharing.
  - Merged headroom

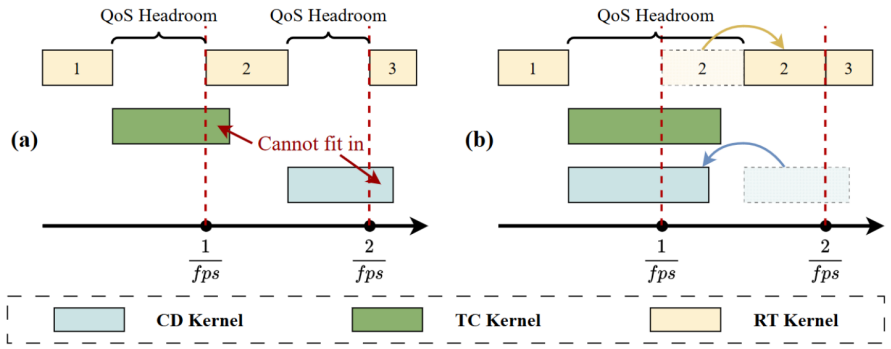



Figure 12: The timeline of 3 kernels before and after headroom merge.

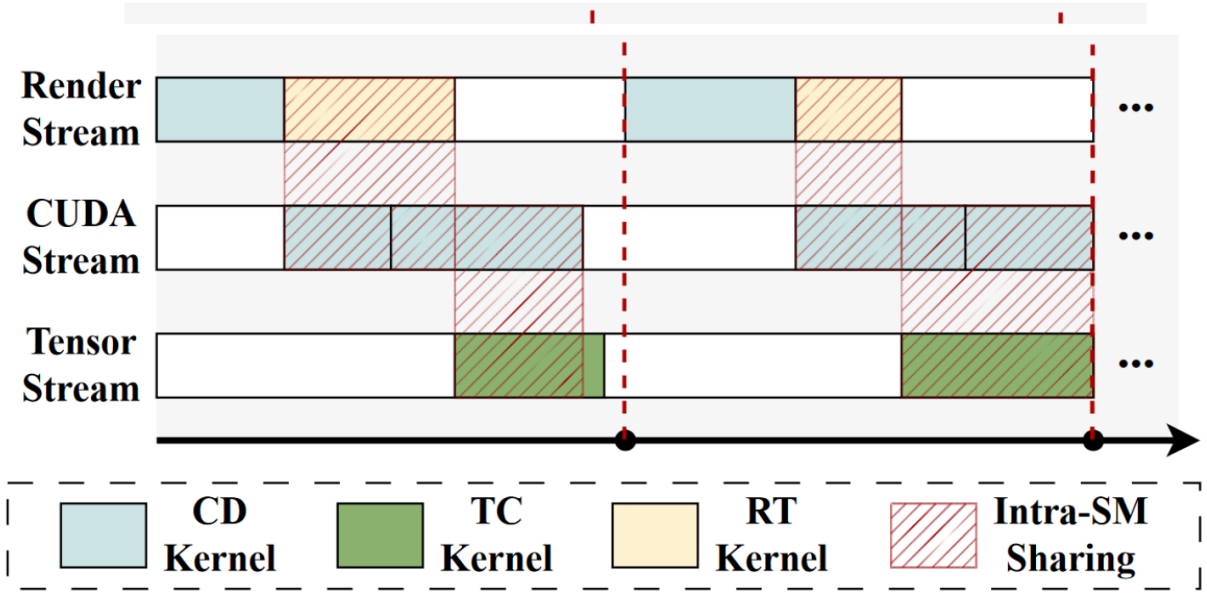



Figure 10: Timeline of relaxed intra-SM sharing.

# Co-location scheme selector

- Scheme Switching Strategy.
  - Dynamic workload
  - Static workload

$$state = \frac{max(rendertime) - min(rendertime)}{avg(rendertime)}$$

$$wokload\ state\ is \begin{cases} static, & state < 10\% \\ dynamic, & state \geq 10\% \end{cases}$$

# Evaluation

- Compare with PilotFish

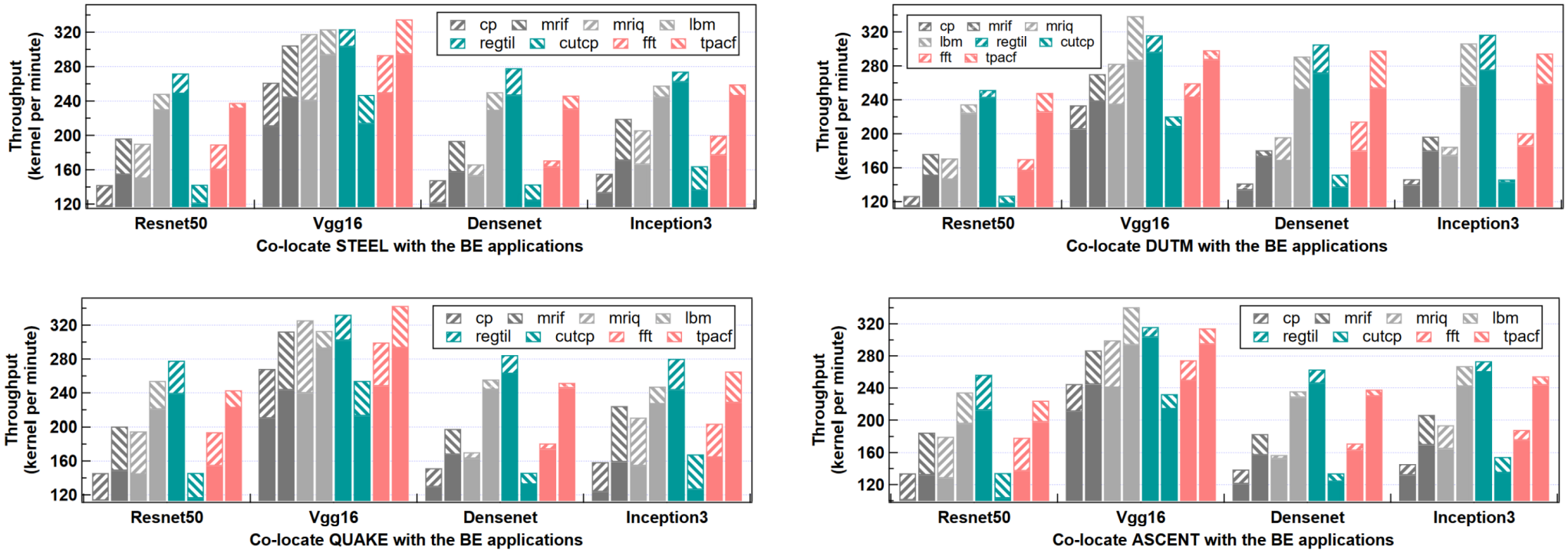



**Figure 13: The throughput comparison of BE applications between COMBO and PilotFish. Solid columns stand for PilotFish, dashed columns stand for COMBO.**

# Evaluation

- Switch mode

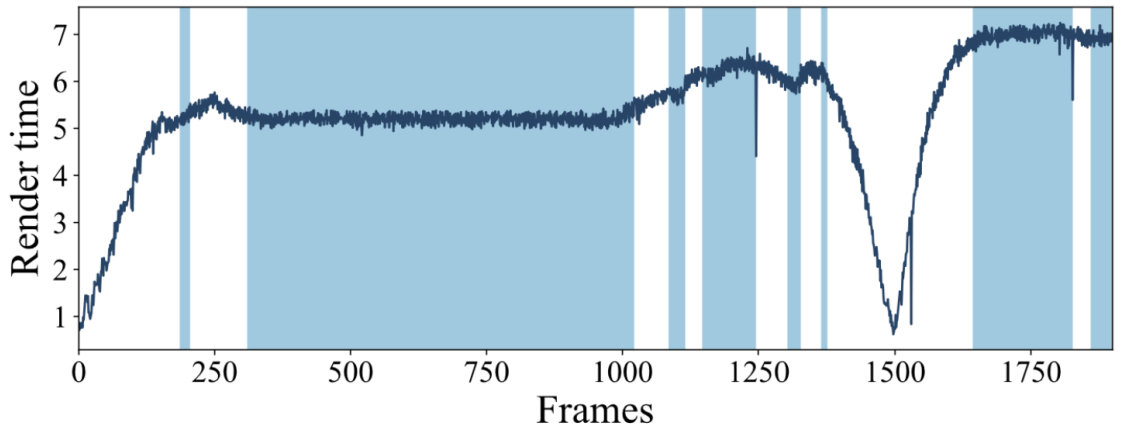



**Figure 18: The trace of the rendering time of frames in THI. In the white region, the rendering time of the frame fluctuates and thus relaxed intra-SM scheme is applied. In the blue region, the rendering time of the frame is stable and thus the scheduling scheme is switched to the exact integrated SM sharing scheme.**