KOÇ ÜNİVERSİTESİ

PAR CORE LAB

An Autonomous Execution Model for GPUs: When CPUs Take a Back Seat

Assoc. Prof. Didem Unat

dunat@ku.edu.tr

Koç University, Istanbul, Turkiye

European Research Council
Established by the European Commission

SC23@ESMP2 Workshop • 13 Nov 2023

Traditional Multi-GPU Model

- Data path has moved to the GPU
- **Control path still on the CPU**
- **CPU involvement adds latencies**

```
// ❶ Time loop on CPU
while (iter++ < num_iterations) {
    // ❷ Launch compute kernel
    stencil_kernel<<<..., comp_stream>>>(...)
    // ❸ Sync with neighboring GPUs
    wait_neighbors(comm_stream)
    // ❹ Compute and communicate boundaries
    compute_boundaries<<<..., comm_stream>>>(...)
    write_neighbors(comm_stream)
    // ❺ Sync comm and comp streams
    sync(comm_stream, comp_str
}
```

CPU-controlled

What if we move everything to the GPU?

KOÇ ÜNİVERSİTESİ

How do we free the GPU?

PAR CORE LAB

- Move the time loop to the GPU
- Initiate communication from the GPU
- Explicitly overlap on the GPU
- Synchronize within and across devices directly from the GPU

```
__global__ void CPU_Free_Jacobi(...) {
  // Time loop on GPU
  while (iter++ < num_iterations) {
    //ⓐ Compute boundary using top neighbor's values
    if (TB_index == 0) {
      //❶ Wait for top neighbor to signal
      wait_top_neighbor(...)
      //❷ Compute top boundary using halos
      top_boundary = compute(top_halo, south, ...)
      //❸ Write to top neighbor's bottom halo
      write_top_neighbor(top_boundary)
      //❹ Signal top neighbor that iteration is done
      signal_top_neighbor(...)
    }
    //ⓑ Compute boundary with bottom neighbor's values
    if (TB_index == 1) { ... }
    //ⓒ Remaining TBs compute the inner
    if (TB_index == <rest>) {
    //❺ Synchronize all
    grid.sync()
}}
```

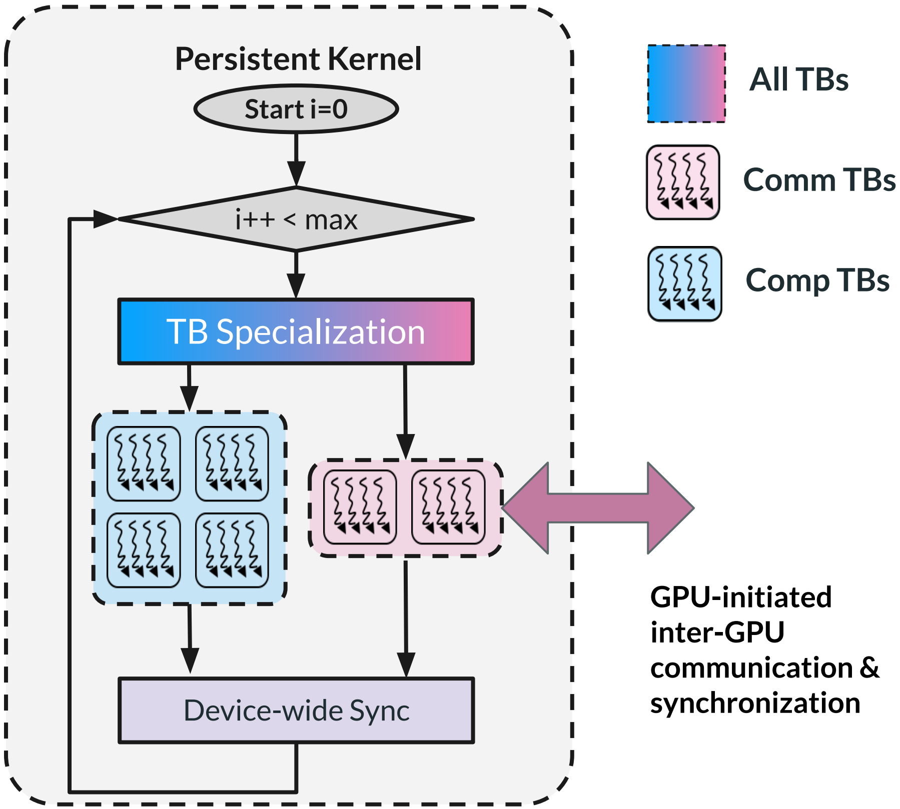
CPU-Free

Allow the GPU to take the reins of the data and *control* paths!

KOÇ ÜNİVERSİTESİ

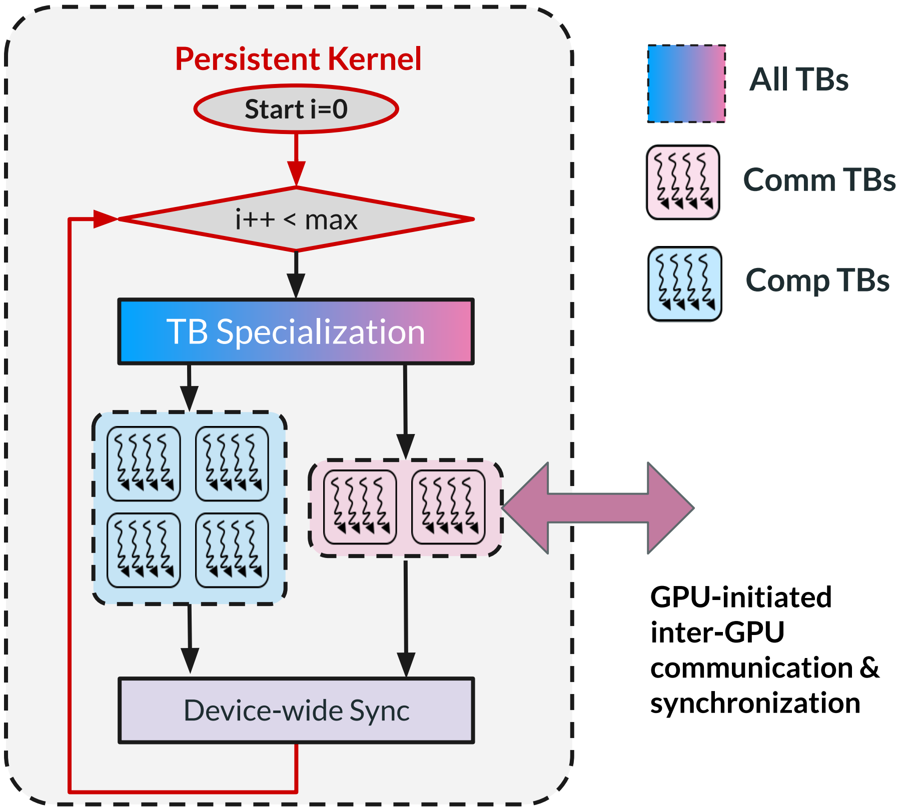
CPU-Free Execution Model

- CPU-Free model grants complete autonomy to the GPU
- Both data and control paths move to device-side
- **Components**
  - **Persistent kernels**
  - **GPU-initiated data movement**
  - **TB specialization**
  - **Device-side synchronization**



Persistent Kernels

- **Move time loop to GPU using a persistent kernel**
- *Discrete* kernel - Launched repeatedly
- *Persistent* kernel - Launched once
- Long running persistent kernel grants GPU more autonomy



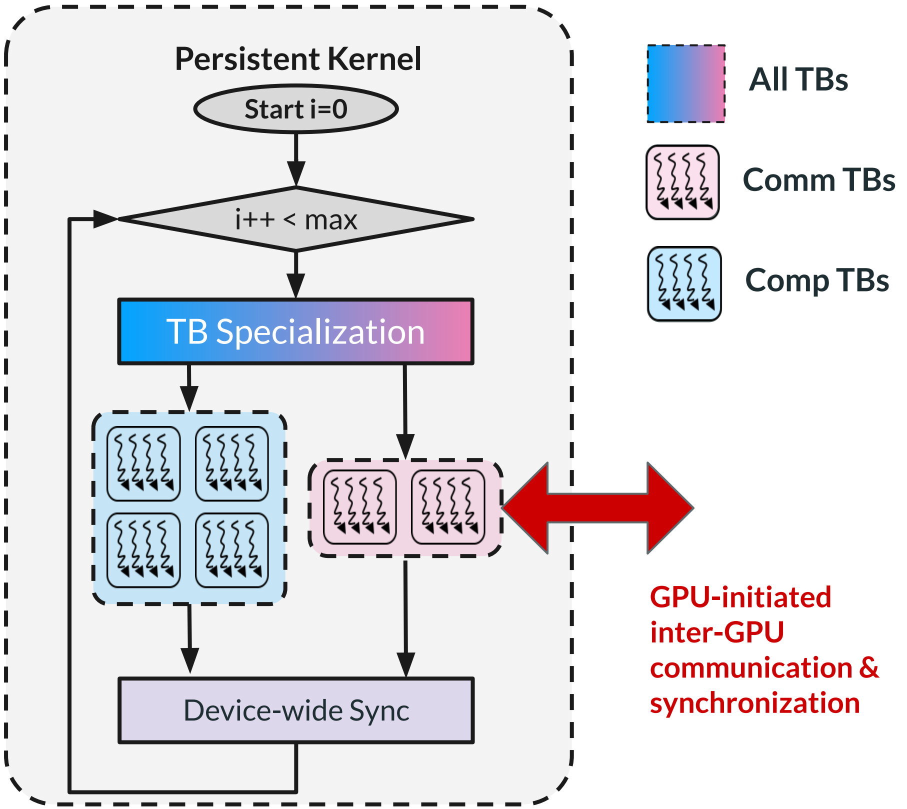
GPU-initiated Data Movement

- **Initiate communication directly from GPU**

- Communication:
  - Data transmission
  - Multi-GPU synchronization

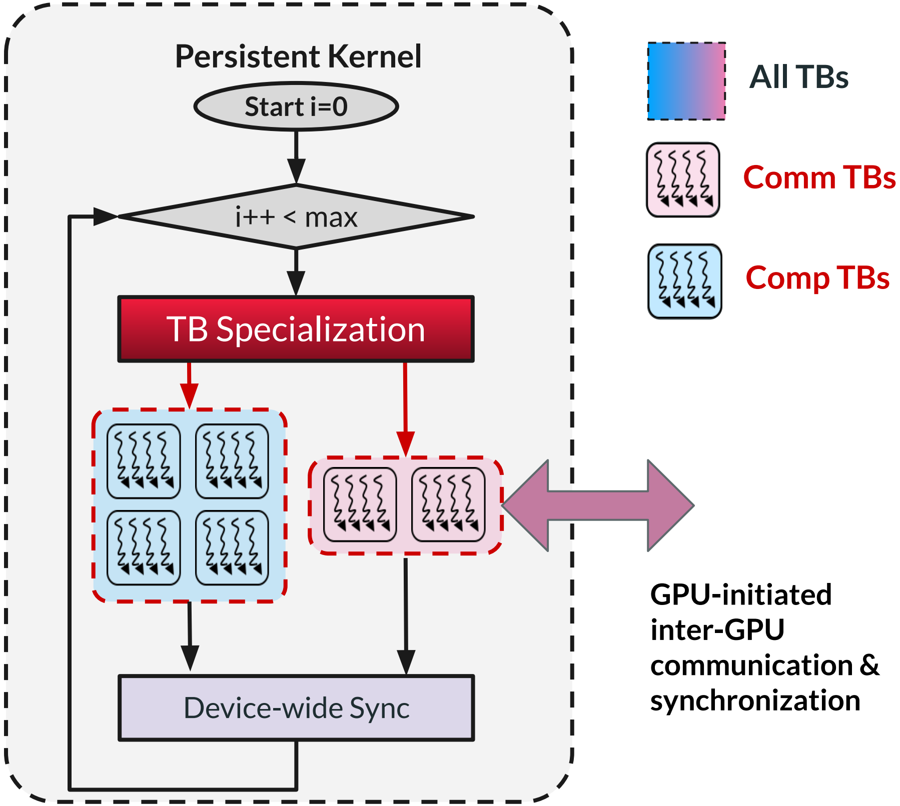
- NVSHMEM used as communication mechanism

- NVSHMEM provides efficient GPU-side communication primitives



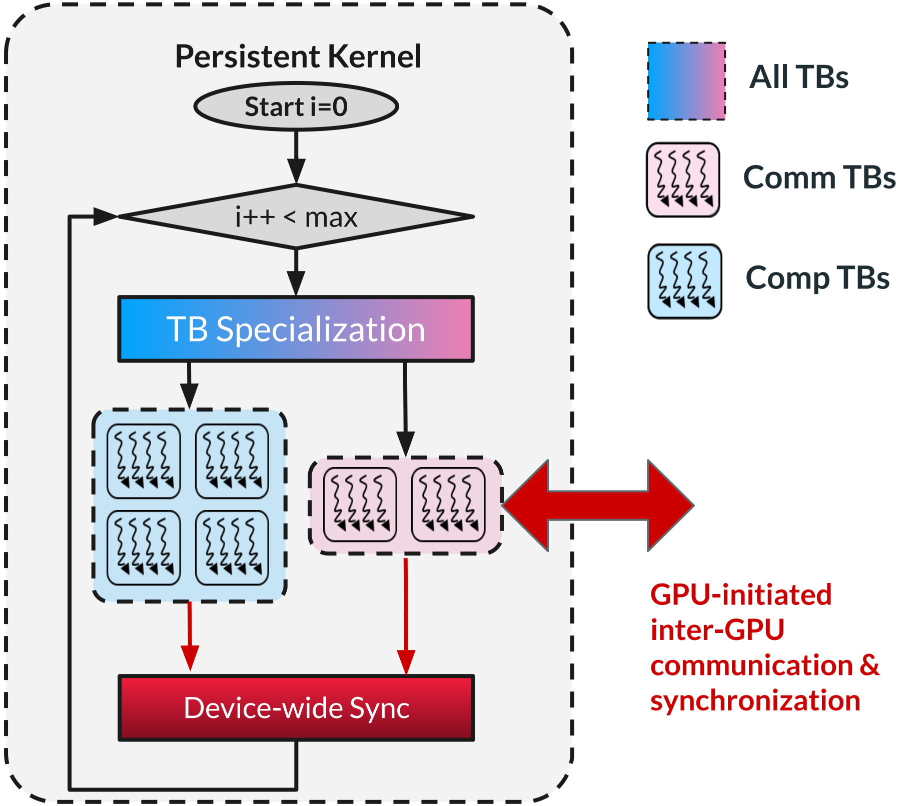
TB Specialization for Overlap

- **Specialize few thread blocks to handle communication**
- **Remaining thread blocks compute**
- Traditional overlap uses concurrent GPU streams
- Streams are synced through GPU events



Device-side Synchronization

- *Within* GPU - Device-side barriers
- *Across* GPUs - Device-initiated signal / flag mechanisms
- CUDA Cooperative Groups API
- NVSHMEM signal operations / barriers for multi-GPU sync



Reduced Kernel Launch / CPU Synchronization Overheads

CPU-controlled execution needs multiple API calls

- Kernel launches
- Communication calls
- Multiple streams
- GPU events
- Global barriers (OpenMP, MPI barriers)

One fused kernel eliminates these overheads

Reduces Communication Overheads and Overlaps Better

PAR CORE LAB

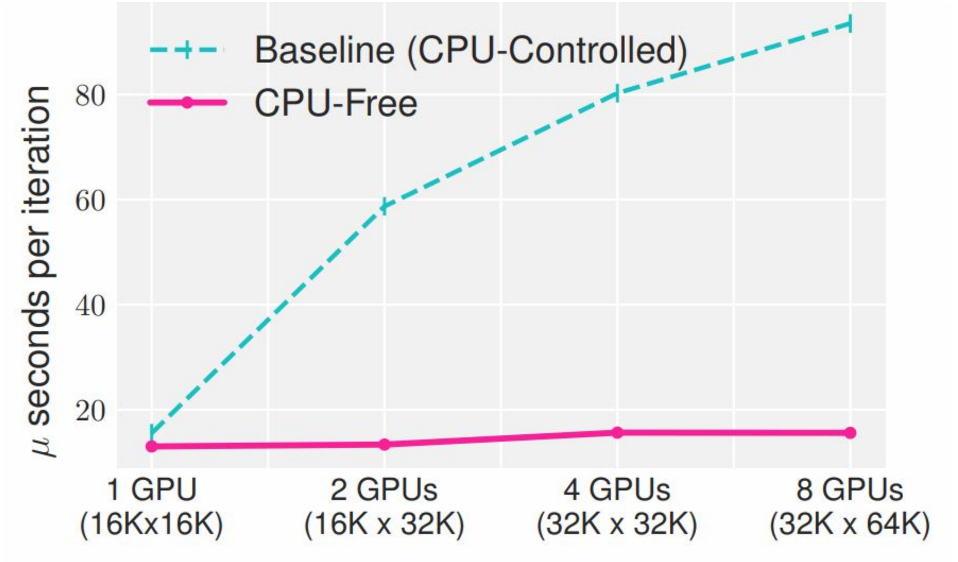
- Can initiate communication as soon as the data is ready

- Can inline communication with computation
- When device not saturated, API call latencies dominate

- CPU-Free can overlap in small domains

- Especially relevant in ***strong scaling***

- More asynchrony



(b) Communication overhead with no computation

Shared Memory Caching

- Shared memory has lifetime of kernel
- CPU-Free execution can reuse shared memory across iterations
- We integrate PERKS kernel into our model

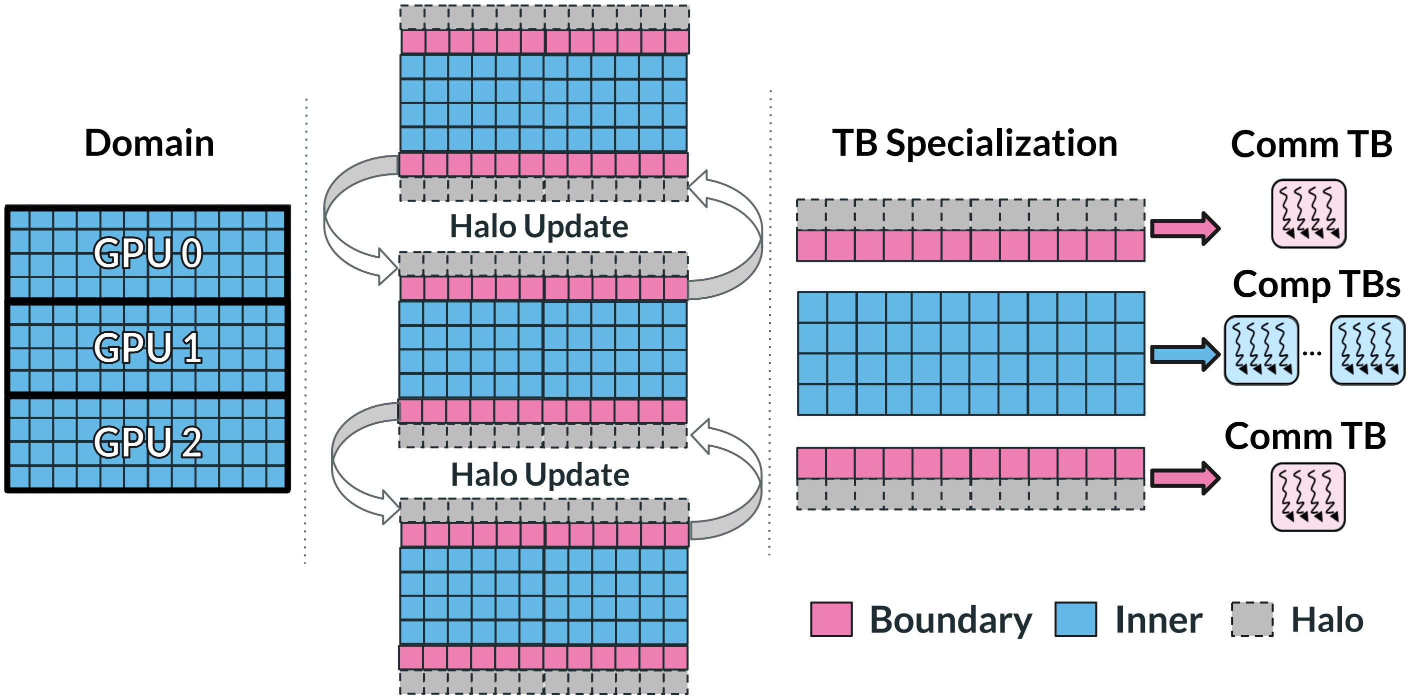
[1]: Zhang et al., PERKS: a Locality-Optimized Execution Model for Iterative Memory-bound GPU Applications

Use-Cases

- Jacobi 2D / 3D
- Conjugate Gradient

Published at Multi-GPU Communication Schemes for Iterative Solvers: When CPUs are Not in Charge. In Proceedings of the 37th International Conference on Supercomputing (ICS '23).

Use-Case 1 - Jacobi Stencil



Use-Case 2 - Conjugate Gradient

Saxpy (purple) | Dot (green) | SpMV (blue)

PAR CORE LAB

$$
\begin{aligned}
&\textbf{procedure } \textsc{Standard}(\mathbf{A}, \mathbf{b}, \mathbf{x}_0)\\
&\quad \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0\\
&\quad \mathbf{p}_0 = \mathbf{r}_0\\
&\quad \gamma_0 = \mathbf{r}_0 \odot \mathbf{r}_0\\
&\quad \textbf{for } k = 0, 1, \ldots \textbf{ do}\\
&\qquad \mathbf{s} = \mathbf{A}\mathbf{p}_k \quad \text{(SpMV)}\\
&\qquad \delta_k = \mathbf{p}_k \odot \mathbf{s} \textbf{ and } \alpha_k = \gamma_k / \delta_k \quad \text{(Dot)}\\
&\qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \quad \text{(Saxpy)}\\
&\qquad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s} \quad \text{(Saxpy)}\\
&\qquad \gamma_{k+1} = \mathbf{r}_{k+1} \odot \mathbf{r}_{k+1} \textbf{ and } \beta_k = \gamma_{k+1} / \gamma_k \quad \text{(Dot)}\\
&\qquad \mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \quad \text{(Saxpy)}\\
&\quad \textbf{end for}\\
&\textbf{end procedure}
\end{aligned}
$$

Saxpy - Local computations; no communication

SpMV - Need entries on other GPUs; need communication

Dot - **Global sync point because of global reduction**

Each step depends on previous

No possible overlap

[2]: M.R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems

Use-Case 2 - Conjugate Gradient

Saxpy | Dot | SpMV

PAR CORE LAB

procedure $\text{PIPELINED}(\mathbf{A}, \mathbf{b}, \mathbf{x}_0)$
  $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
  $\mathbf{w}_0 = \mathbf{A}\mathbf{r}_0$
  for $k = 0, 1, \ldots$ **do**
    $\gamma_k = \mathbf{r}_k \odot \mathbf{r}_k$
    $\delta_k = \mathbf{w}_k \odot \mathbf{r}_k$
    $\mathbf{q}_k = \mathbf{A}\mathbf{w}_k$
    if $k > 0$ **then**
      $\beta_k = \gamma_k / \gamma_{k-1}$
      $\alpha_k = \gamma_k / (\delta_k - (\beta_k \gamma_k)/\alpha_{k-1})$
    else
      $\beta_k = 0$
      $\alpha_k = \frac{\gamma_k}{\delta_k}$
    end if
    $\mathbf{z}_k = \mathbf{q}_k + \beta_k \mathbf{z}_{k-1}$
    $\mathbf{s}_k = \mathbf{w}_k + \beta_k \mathbf{s}_{k-1}$
    $\mathbf{p}_k = \mathbf{r}_k + \beta_k \mathbf{p}_{k-1}$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{s}_k$
    $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{z}_k$
  end for
end procedure

Introduce auxiliary vectors to allow overlap

Can implement dot product global reductions with *one* reduction

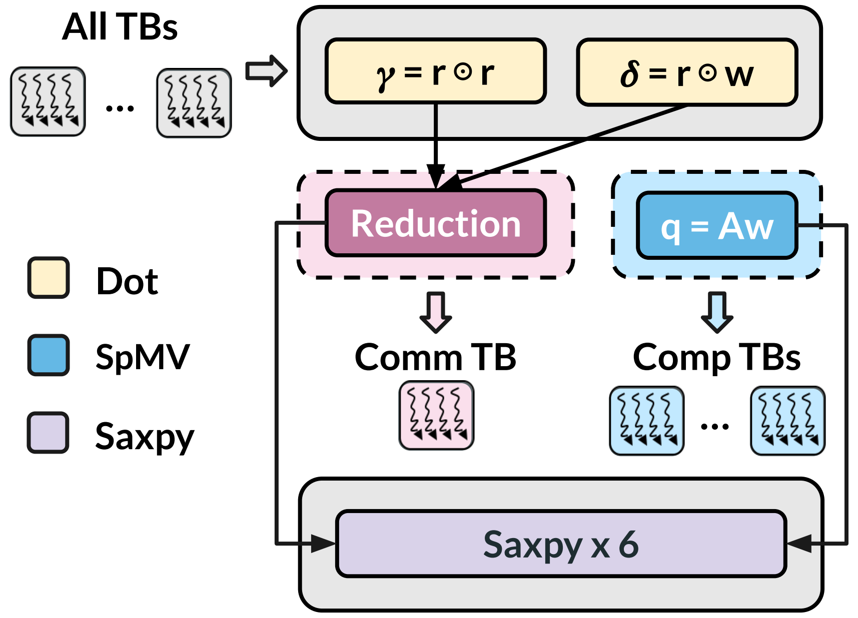
Can overlap dot product global reductions with SpMV

- Pipelined CG can showcase overlap
- We implement CPU-Free for both Standard and Pipelined CG

KOÇ ÜNİVERSİTESİ

[3]: P. Ghysels and W. Vanroose, Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm

Use-Case 2 - Conjugate Gradient




```
__global__ void CPU_Free_PipelinedCG(...) {
  // ❶ Time loop on GPU
  while (iter++ < num_iterations) {
    local_dot(r, r, gamma)
    local_dot(r, w, delta)
    ...
    // ❷ Specialize one TB for communication
    if (TB_index == 0) {
      // ❸ Multi-GPU reduction
      sum_reduce(gamma, delta)
    } else {
      SpMV(A, w, q)
    }
    // ❹ Sync within device
    grid.sync()
    ...
    saxpy(...) //x6
    // ❺ Sync across devices
    multi_gpu.sync()
}}
```

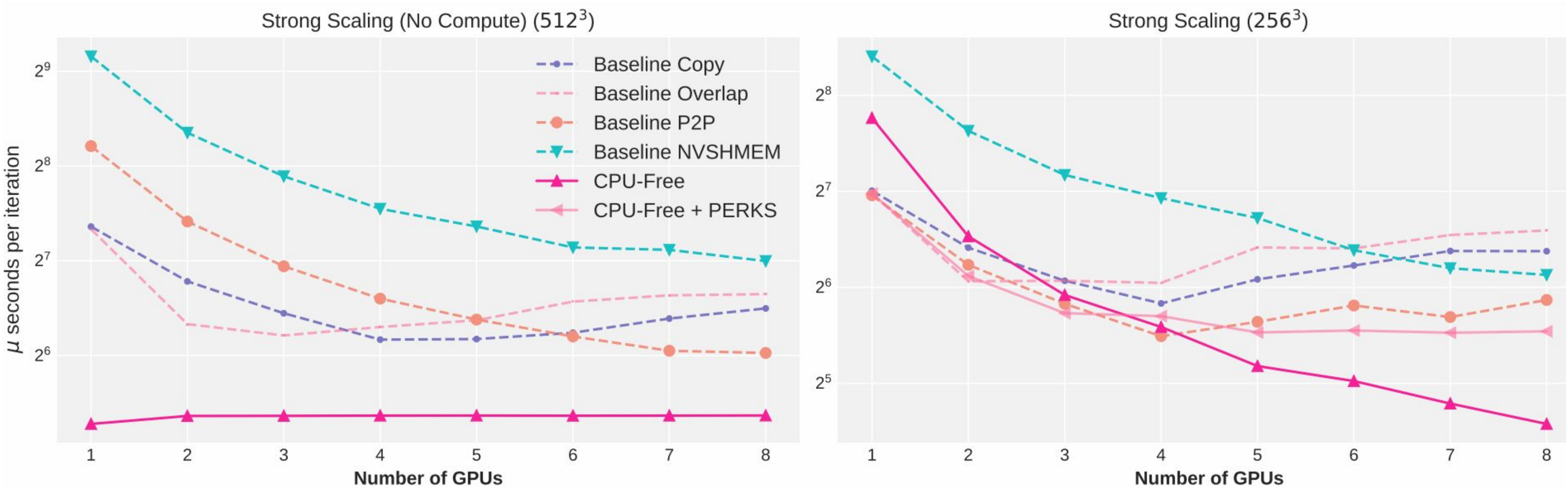
Evaluation

- Jacobi 2D/3D
- Conjugate Gradient

8 NVLink all-to-all connected NVIDIA A100 GPUs

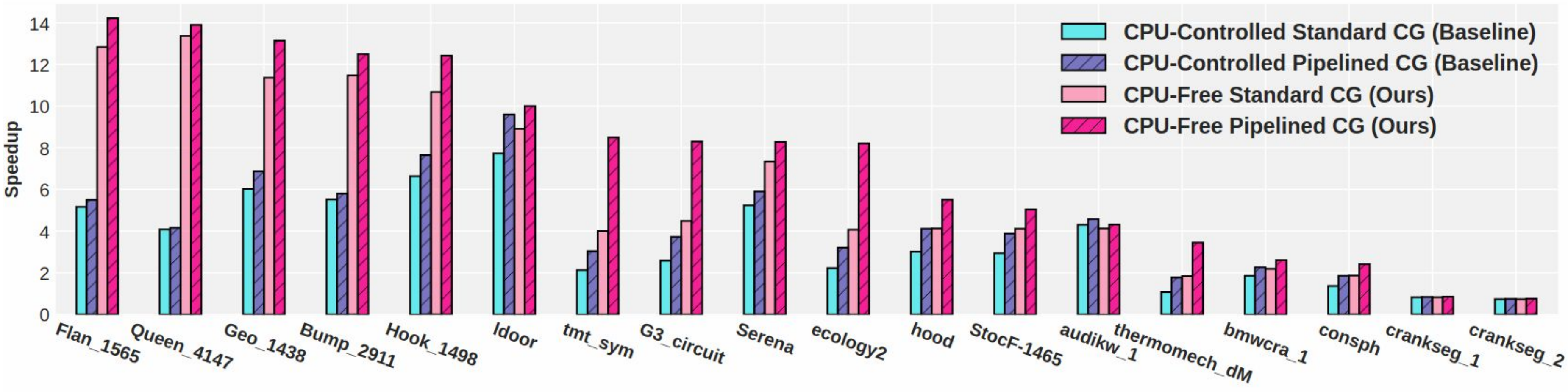
Published at Multi-GPU Communication Schemes for Iterative Solvers: When CPUs are Not in Charge. In Proceedings of the 37th International Conference on Supercomputing (ICS '23).

Jacobi 3D - Strong Scaling



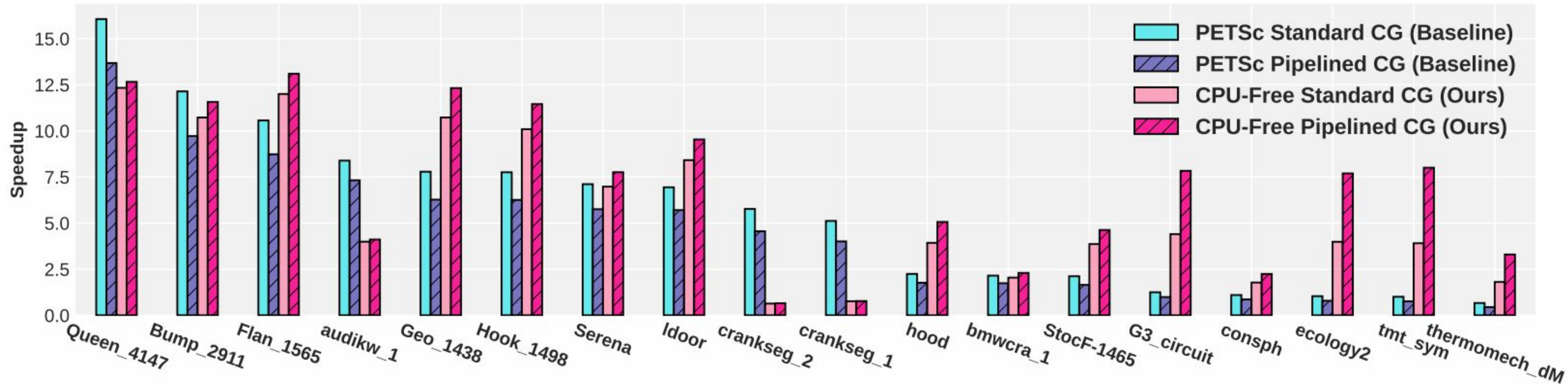
- Consistently lower communication overheads
- Excels in strong scaling scenarios
    - Underperforms at small numbers of GPUs (compute-bound)
    - As GPU count increases, overheads start to dominate
    - CPU-Free pulls ahead at larger GPU counts

Conjugate Gradient



CPU-Free achieves **1.63x** and **1.54x** geo mean speedup over CPU-Controlled for Pipelined and Standard CG, respectively

Comparison with Petsc



Outperforms PETSc for 13 out 18 sparse matrices

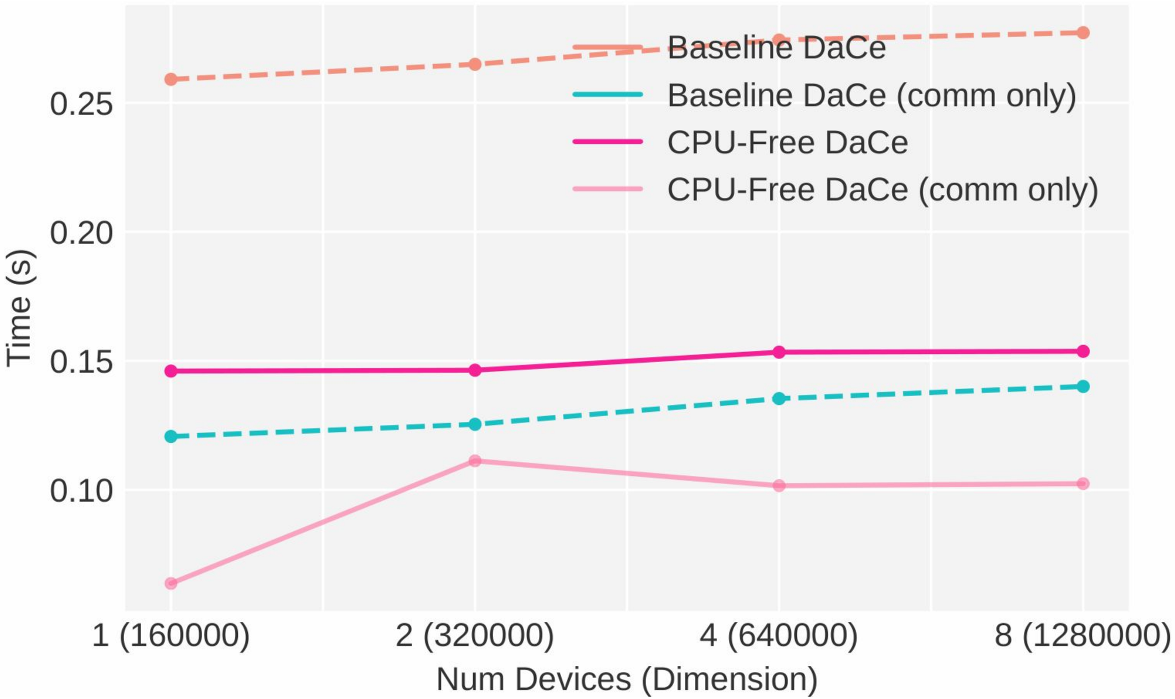
Work in Progress

PAR CORE LAB

- Compiling autonomous CPU-free code from Python
- Developing profiling and monitoring tools for inter-GPU communication
- Building a runtime system that uses CudaGraphs for CPU-free execution - particularly useful for irregular applications
- Extending the CPU-free model to AMD GPUs

Extending the DACE compiler

- **44.5%** performance improvement at 8 GPUs
- **26.8%** improvement in communication latency
- Little overall communication, improvements attributed to synchronization overheads



Outlook

PAR CORE LAB

- **Hard-to-scale** management overheads can be moved to GPU
- GPUs can be more autonomous
  - Managing their own communication, synchronization etc
- Less reliance on CPU
- Need to automate this process and support it with tools
- Do we need fat expensive CPUs to be attached to GPUs?
  - Modular supercomputers
    - Use light CPUs for GPUs nodes
    - Reduce cost and energy consumption

KOÇ ÜNİVERSİTESİ

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 949587).

erc European Research Council Established by the European Commission

CPU-Free Model

- Outperforms CPU-controlled baselines for both Jacobi 2D/3D and CG
- Especially suited for **communication latency-bounded and strong scaling scenarios**
  - Any application suggestions?
- Code available at https://github.com/ParCoreLab/CPU-Free-model

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 949587).