# Impact of GPU Multistreaming on Task-Based DAG Execution in Dagger.jl

Víctor Câmara [1] Felipe Tomé [2†] Julian Samaroo [2†]

[1]University of Brasilia [2]Massachusetts Institute of Technology [†]Advisor

## The Challenge

GPU task graphs expose many independent kernels, but a single in-order stream runs them one after another. At 1 stream, device busy fraction stays at 0.99 on both GPUs, so kernels never overlap. We add N concurrent CUDA/HIP streams per device to Dagger.jl's datadeps scheduler, with no change to user code, and ask which stream count and placement policy actually speed up a DAG.

## Workloads: DAG Topologies

- How can we get the most out of a DAG based runtime on a GPU?

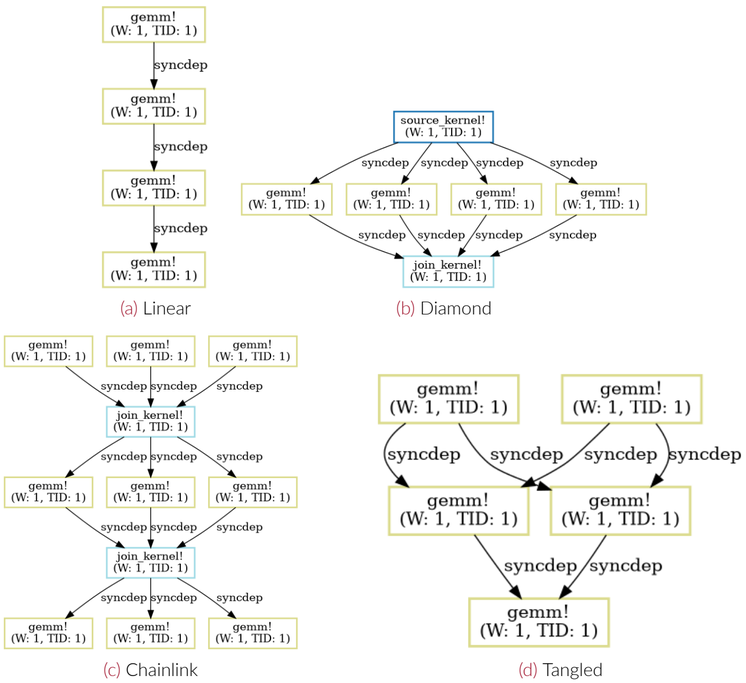
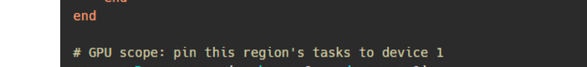



Figure 1. DAG Topologies

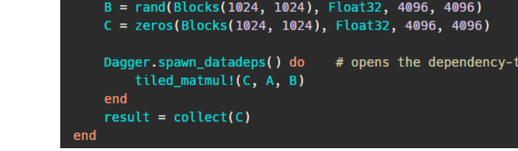
Task-graph runtimes expose independent work explicitly, letting a scheduler exploit all available parallelism instead of relying on manual thread/stream orchestration.

## Queues (CUDA/HIP Streams)

**Single queue** - Most DAG based runtimes run on a single execution queue for gpus.




**Multi Queues** - With multiples queues you can run concurrent tasks and have better use of the GPU




## Easy to Use

You can easily launch kernels, and Dagger.jl takes care of everything for you.

```julia
using Dagger, LinearAlgebra

function tiled_matmul!(C, A, B)
    Ac, Bc, Cc = A.chunks, B.chunks, C.chunks
    for n in axes(Cc, 2), m in axes(Cc, 1), k in axes(Ac, 2)
        beta = k == 1 ? 0f0 : 1f0
        Dagger.@spawn BLAS.gemm!('N', 'N', 1f0,
            Dagger.In(Ac[m, k]), Dagger.In(Bc[k, n]), beta, Dagger.InOut(Cc[m, n]))
    end
end

# GPU scope: pin this region's tasks to device 1
scope = Dagger.scope(worker = 1, cuda_gpu = 1)

Dagger.with_options(; scope) do
    A = rand(Blocks(1024, 1024), Float32, 4096, 4096)
    B = rand(Blocks(1024, 1024), Float32, 4096, 4096)
    C = zeros(Blocks(1024, 1024), Float32, 4096, 4096)

    Dagger.spawn_datadeps() do    # opens the dependency-tracked region
        tiled_matmul!(C, A, B)
    end
    result = collect(C)
end
```

Figure 2. Kernel wrap using Dagger.jl

## Scheduling Policies

- **Round Robin** Assigns tasks in a cyclic order until all processes are completed.
- **Random:** Assigns tasks to a random queue.
- **SDQ (Shortest queue)** Assigns tasks to the shortest queue avaliable.
- **locality:** Assigns each task to the stream that already holds its input data

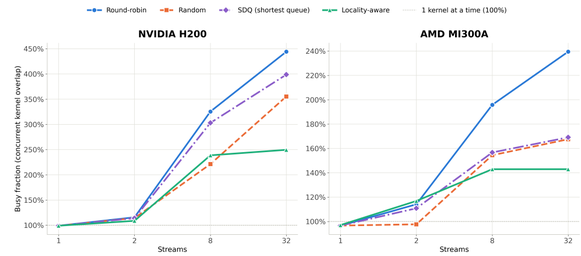



Figure 3. Total saturation

Busy fraction higher than 100% means that multiple kernels are executing at the same time

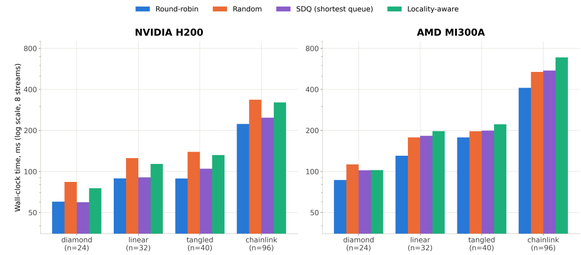



Figure 4. wall-clock x policy

## Takeaways

- More streams pay off: up to 5.3× (H200) and 2.7× (MI300A) at 32 streams.
- Locality cuts sync events by up to 88% but concentrates work on few streams, so it stops scaling beyond 8 streams.
- **Fewer syncs do not mean faster.** Locality removes up to 89% of cross-stream sync events but concentrates work on few streams, so it stops scaling after 8 streams



victor.camara@aluno.unb.br