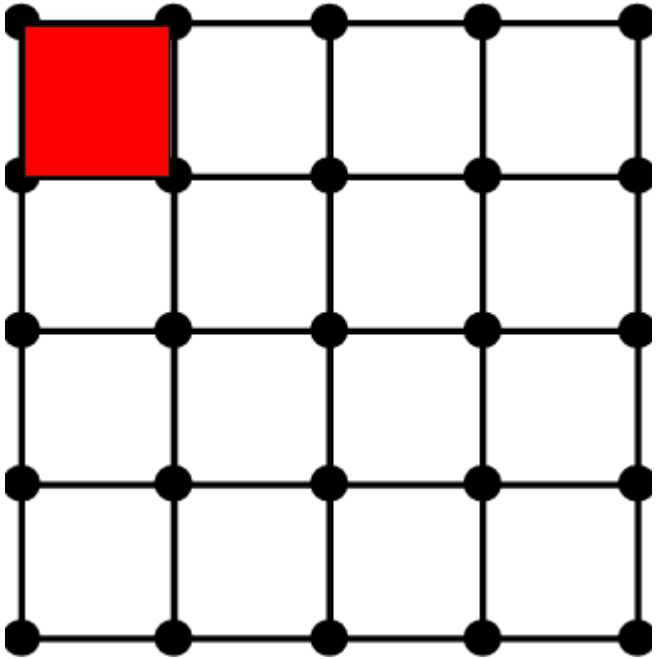
Spatial GPU Proposal

Jacob Levinson
Dylan Kupsh

Motivation

- In modern GPUs, a threadblock is mapped to one SM/core.
- Often, this thread block will access a predetermined section of GPU main memory, such as in a hotspot kernel.

Example: Threadblock 1 will need access to the highlighted portion of the input matrix to calculate it's values.

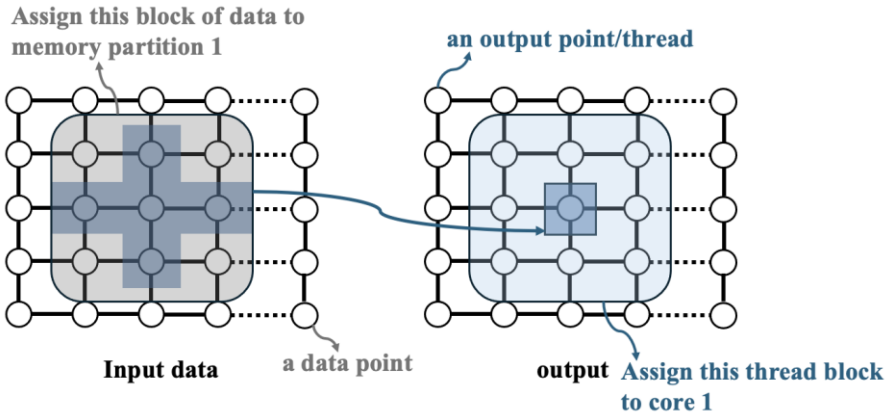


Input Data

Motivation

- It would be nice if we could control which core each thread block was assigned to, and then partition the GPU memory such that each core had access to one partition more quickly, as in a mesh network

- We could then tell the GPU to allocate the part of memory that thread block will be working on to the nearby memory partition/bank

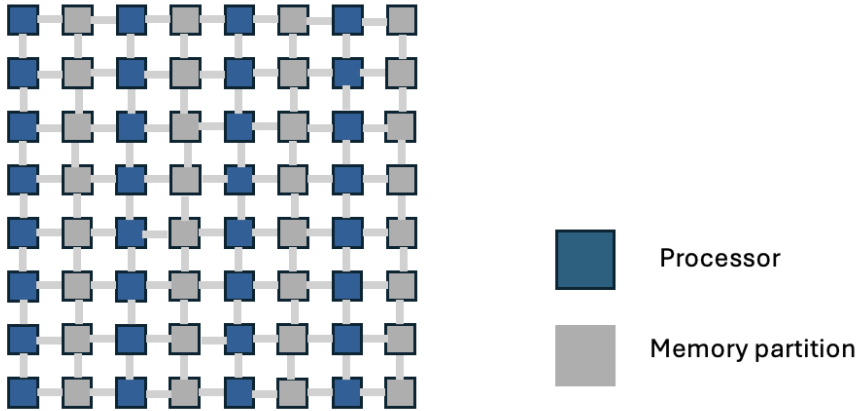


Hotspot kernel example

Motivation

- In an ordinary crossbar memory system, this could reduce bank conflicts in the main GPU memory, since each core will be sending most of its memory requests to one partition.

- In a mesh system, this could significantly reduce latency and energy consumption by spreading memory partitions across the chip, each next to (and primarily serving) one or more cores

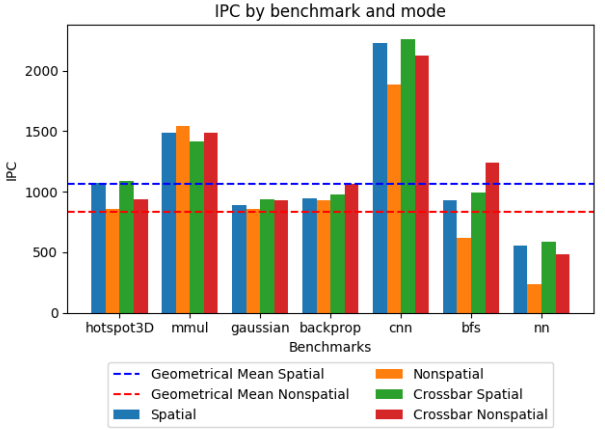


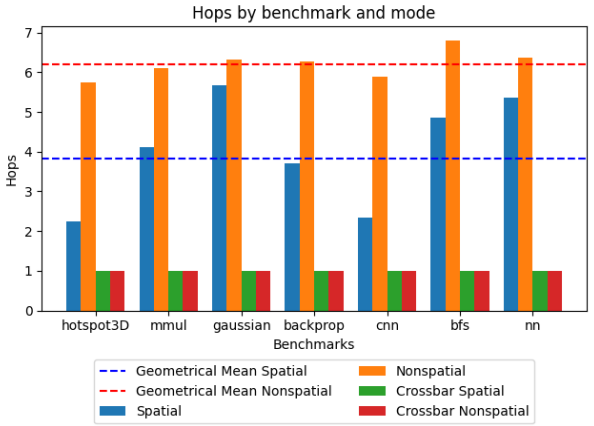
Example mesh network

Goal

- This work has already been investigated by professor Tony Nowatzki's PolyArch research group in GPGPU Sim

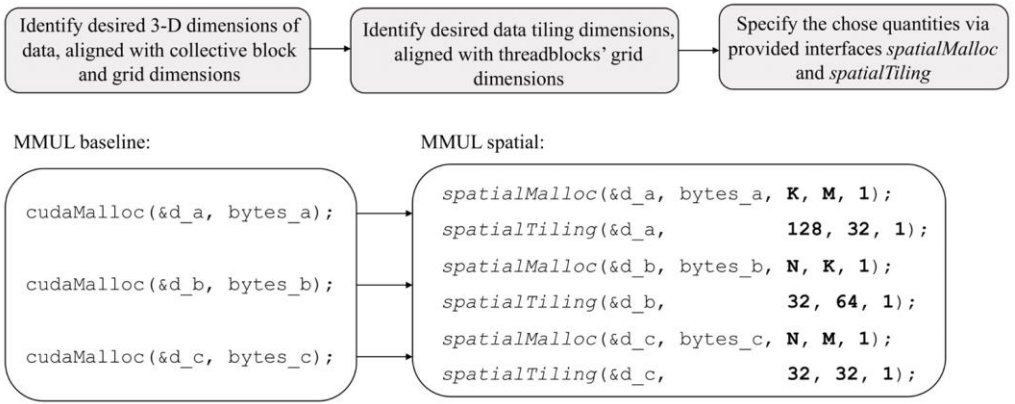
- The goal of this project is to implement a similar system within Vortex Open GPU





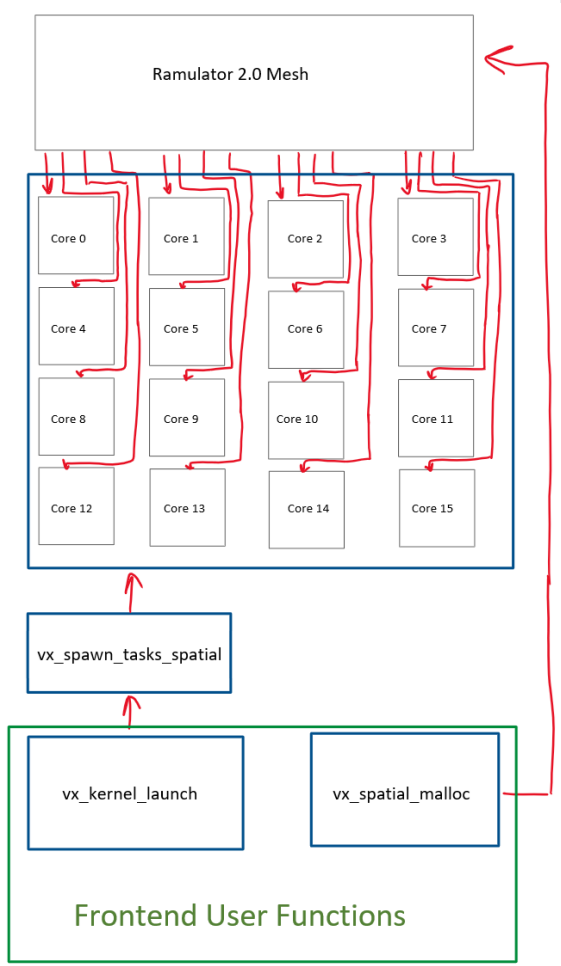
Goal

- The main goal is to implement a spatial GPU interface similar to the one already implemented.
- This will need to be supported by implementing a mesh network for the main memory as well.



High-level Architecture

- The ramulator ideally has number of partitions equal to the number of cores
- We integrate the spatial tiling and spatial malloc calls into one function , which will require the tiling as parameters
- We implement a custom spawn tasks to map the logical thread blocks to specific cores based on their thread block index.



Ramulator 2.0 Mesh

- We will have to investigate implementing the mesh either inside or outside ramulator
  - Multiple ramulator banks could be used to create a mesh

- We will also have to investigate the timing of the mesh vs crossbar to add it to our simulations
  - Also maybe investigate energy model of mesh vs crossbar

vx_spatial_malloc

- Will take in the following parameters:
  - devPtr - pointer to allocated memory
  - size_x - size of tile in x dimension
  - size_y - size of tile in y dimension
  - size_z - size of tile in z dimension
  - num_tiles_x - how many tiles in x dimension
  - num_tiles_y - how many tiles in y dimension
  - num_tiles_z - how many tiles in z dimension
- Ideally, we want the number of total tiles to match with the total number of thread blocks(or a multiple), to perfectly match one core to one memory tile/partition
- Total memory size allocated is all the parameters multiplied

vx_kernel_launch

- This function will be very similar to the cuda thread launch, taking in block dimensions and grid dimensions.

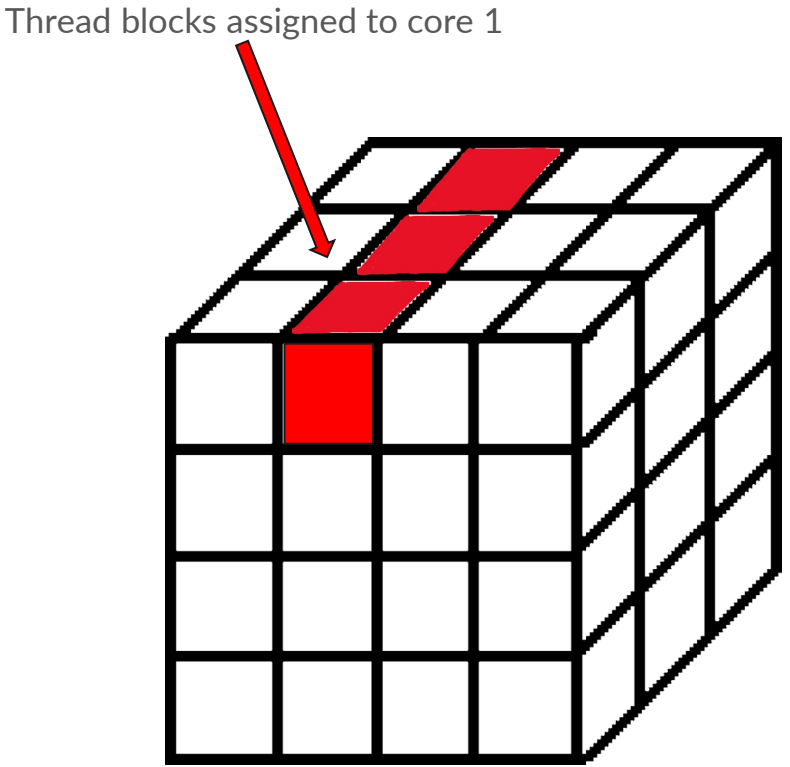
- This allows us to assume the spatial locality of the input data, which in turn is what enables the spatial optimizations.

vx_spawn_tasks_spatial

- This function is similar to the spawn_tasks_ex in the updated codebase.
- We use the gid dimensions to map specific thread blocks to specific cores, matching the dimensions with the index of out cores in a grid.
- Each thread block becomes one “group”

vx_spawn_tasks_spatial

- To map the thread block to each core, we will use a similar system to the GPGPU spatial gpu
- Ideally, there will be an equal number of x * y thread blocks as number of cores in
  - If not, we can “wrap back around”
- Then, we assign the thread blocks to cores based on their x and y grid dimensions
  - Blocks with the same x and y but different z are still assigned to the same core, thus creating a “cube” of logical threads on each core



Evaluation

- Evaluating method against the following benchmarks:

- Using baseline GPU architecture to measure performance against

Matrix Multiplication	Matrix multiplication using shared memory
Hotspot	Thermal simulation tool, similar workload computation to output value
Backpropagation	Forward/Backwards phase, similar to matrix multiplication
Convolutional Neural Network	2D Convolutional layer, storing inputs and filters in shared memory
Breadth-first Search	Spatial allocation of nodes/edges
Gaussian	Gaussian elimination worklaod
K-Nearest Neighbors	Testing of point-storage organization

Success Definition

- We expect SPGPU to have a performance increase, compared to GPGPU, across several benchmarked workloads. The performance increase arises from better memory organization (fewer bank conflicts).

- Recognizing limited implementation time, we expect future improvements to significantly improve performance, especially the creation of a mesh-based network. Even small improvements would be a success.