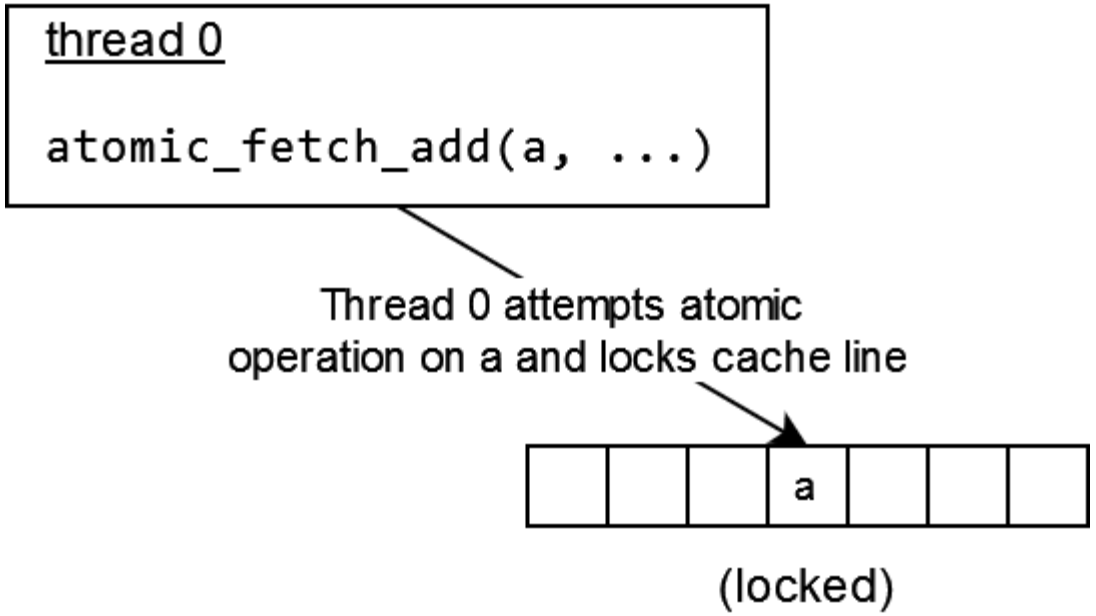
Vulkanised 2024
The 6th Vulkan Developer Conference
Sunnyvale, California | February 5-7, 2024

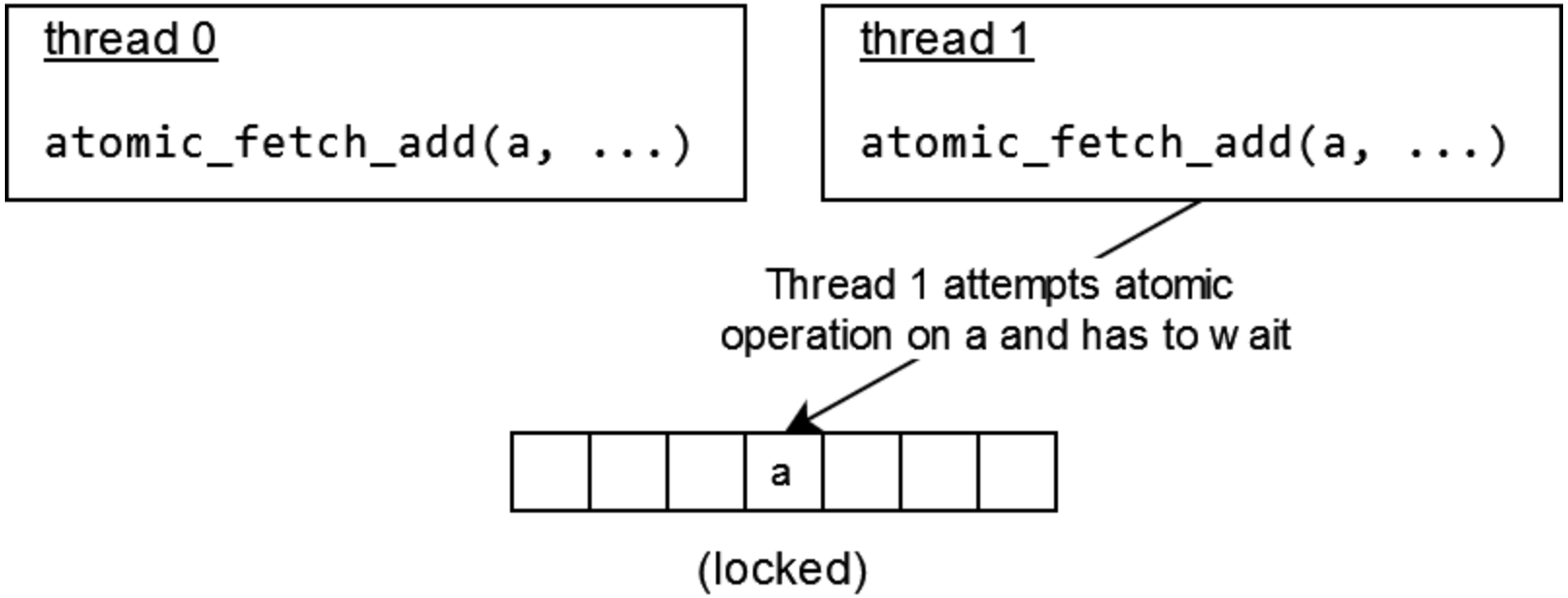
GPU Atomic Performance Modeling with Microbenchmarks

Devon McKee, UC Santa Cruz Concurrency and Heterogeneous Programming Lab

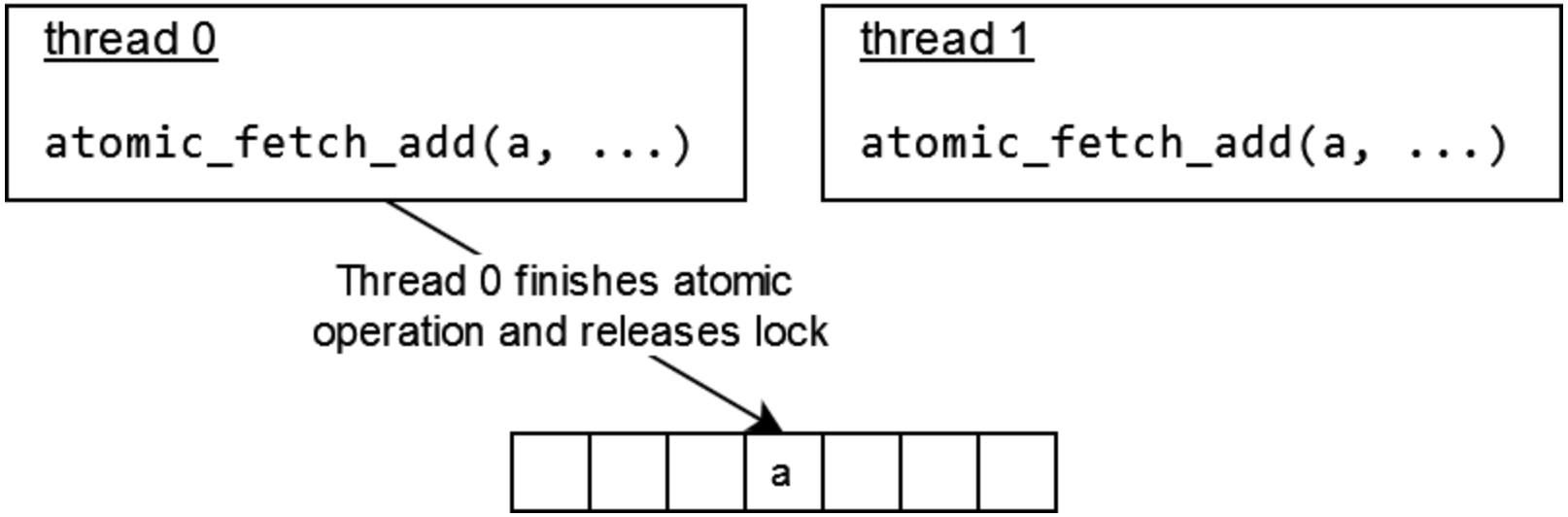
CPU ATOMIC RMWS



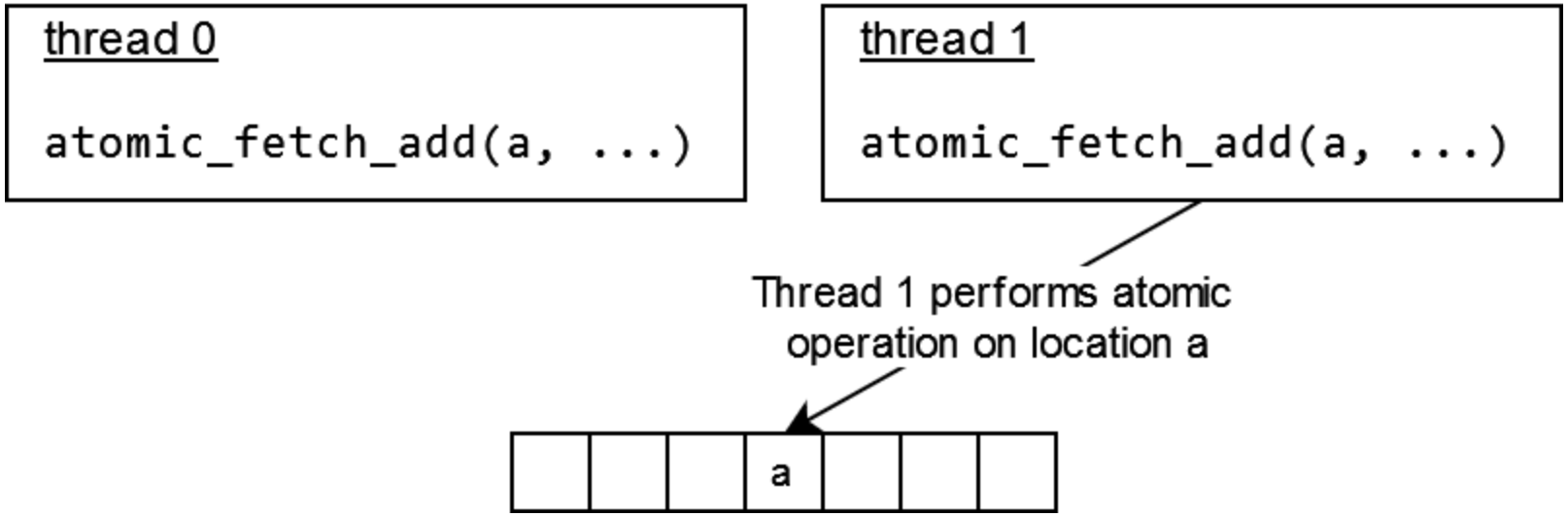
CPU ATOMIC RMWS



CPU ATOMIC RMWS



CPU ATOMIC RMWS



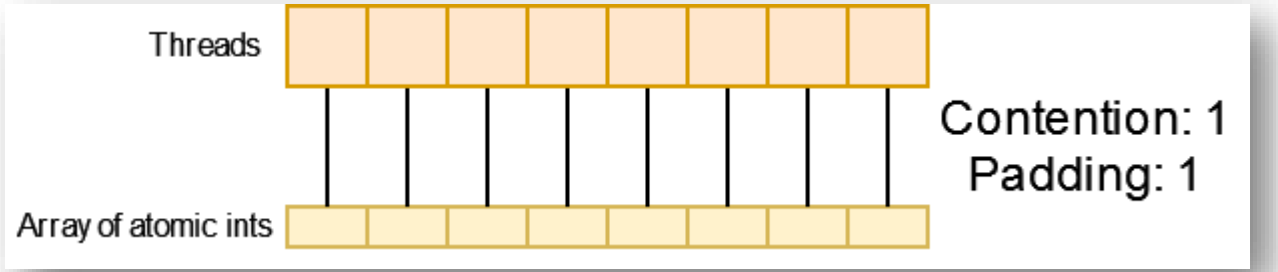
CPU ATOMIC RMWS

- CPU atomic read-modify-write instructions (RMWs) have a well-defined performance profile

CPU ATOMIC RMWS

- CPU atomic read-modify-write instructions (RMWs) have a well-defined performance profile
- Threads needing access prefer to lock cache lines

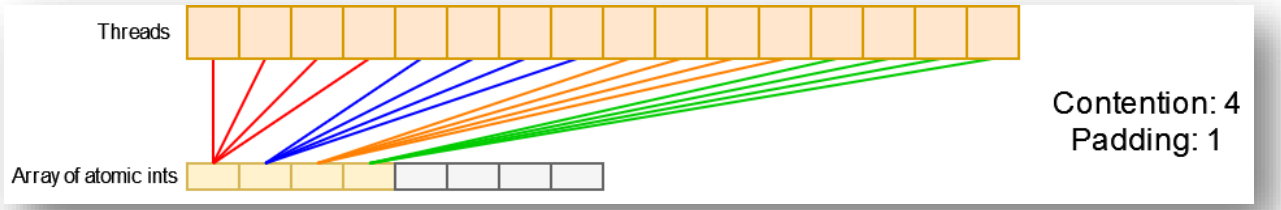
Each thread accessing one atomic location, no padding:
73.14 atomic ops/microsecond



CPU ATOMIC RMWS

- CPU atomic read-modify-write instructions (RMWs) have a well-defined performance profile
- Threads needing access prefer to lock cache lines
- RMW throughput is hurt by threads contending for same location

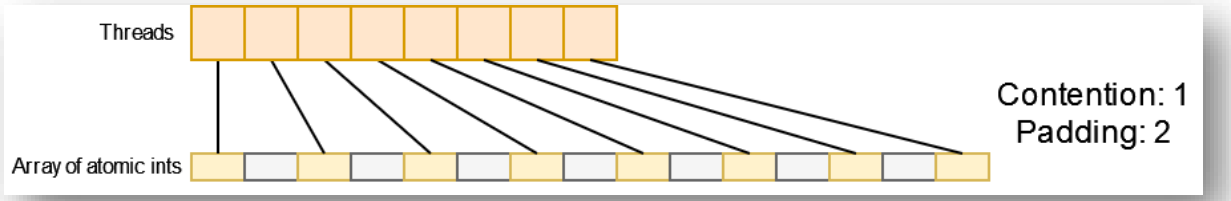
Every 4 threads accessing each location, no padding:
40.89 atomic ops/microsecond



CPU ATOMIC RMWS

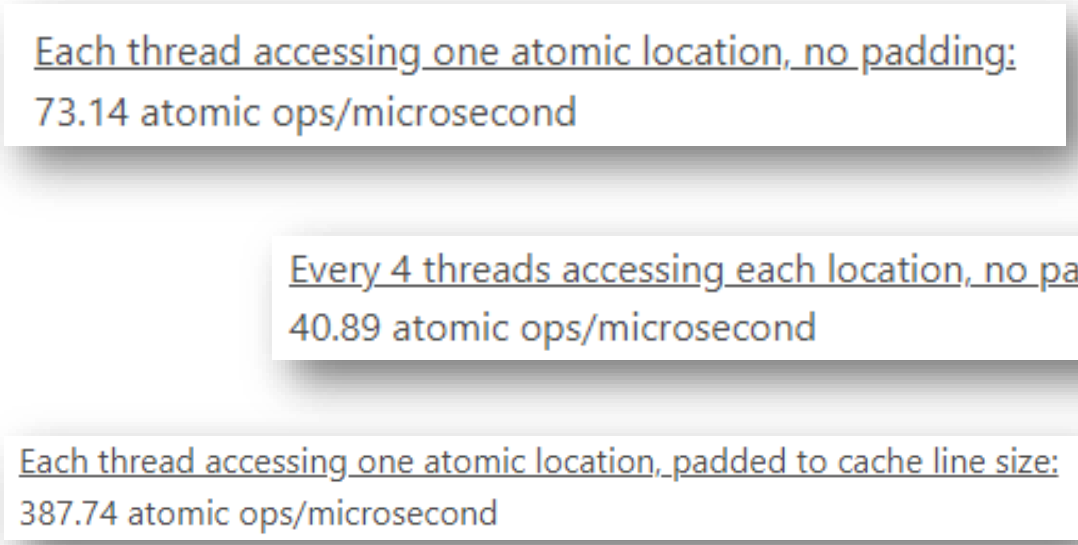
- CPU atomic read-modify-write instructions (RMWs) have a well-defined performance profile
- Threads needing access prefer to lock cache lines
- RMW throughput is hurt by threads contending for same location
- RMW throughput is helped by locations padded out to cache line size

Each thread accessing one atomic location, padded to cache line size:
387.74 atomic ops/microsecond



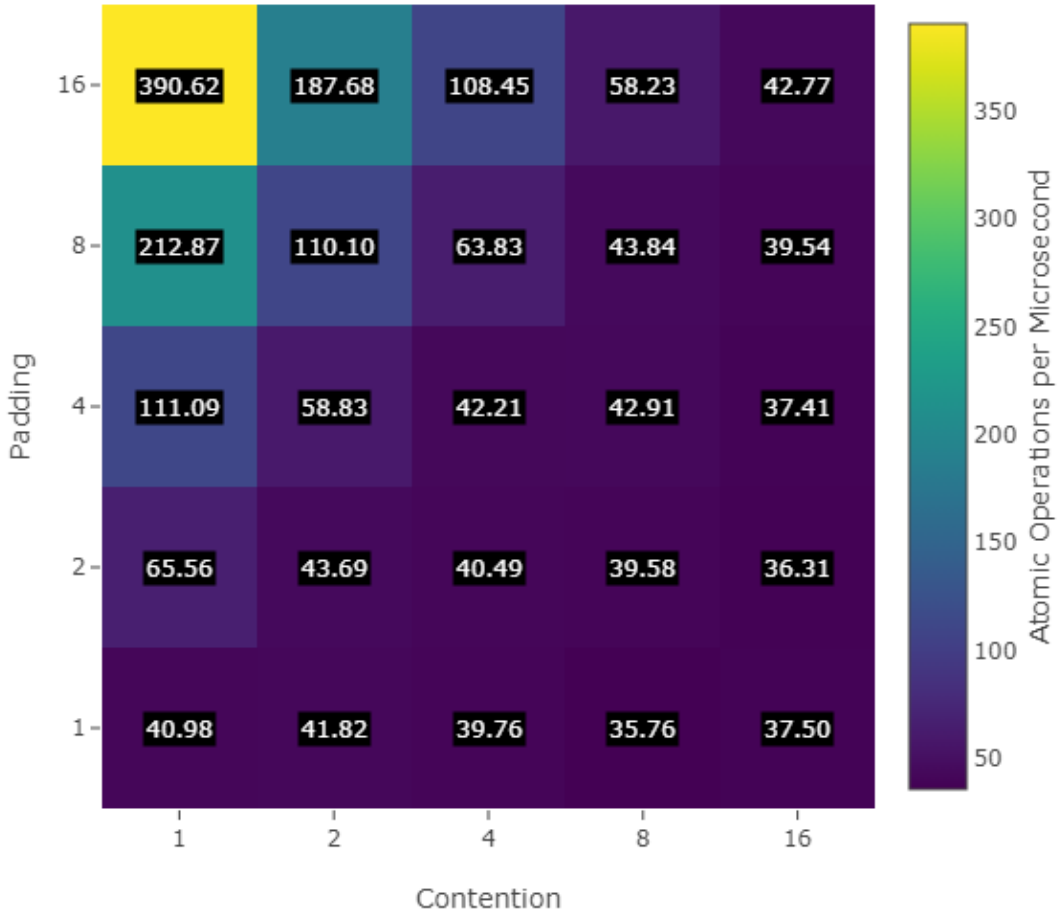
CPU ATOMIC RMWS

- CPU atomic read-modify-write instructions (RMWs) have a well-defined performance profile
- Threads needing access prefer to lock cache lines
- RMW throughput is hurt by threads contending for same location
- RMW throughput is helped by locations padded out to cache line size



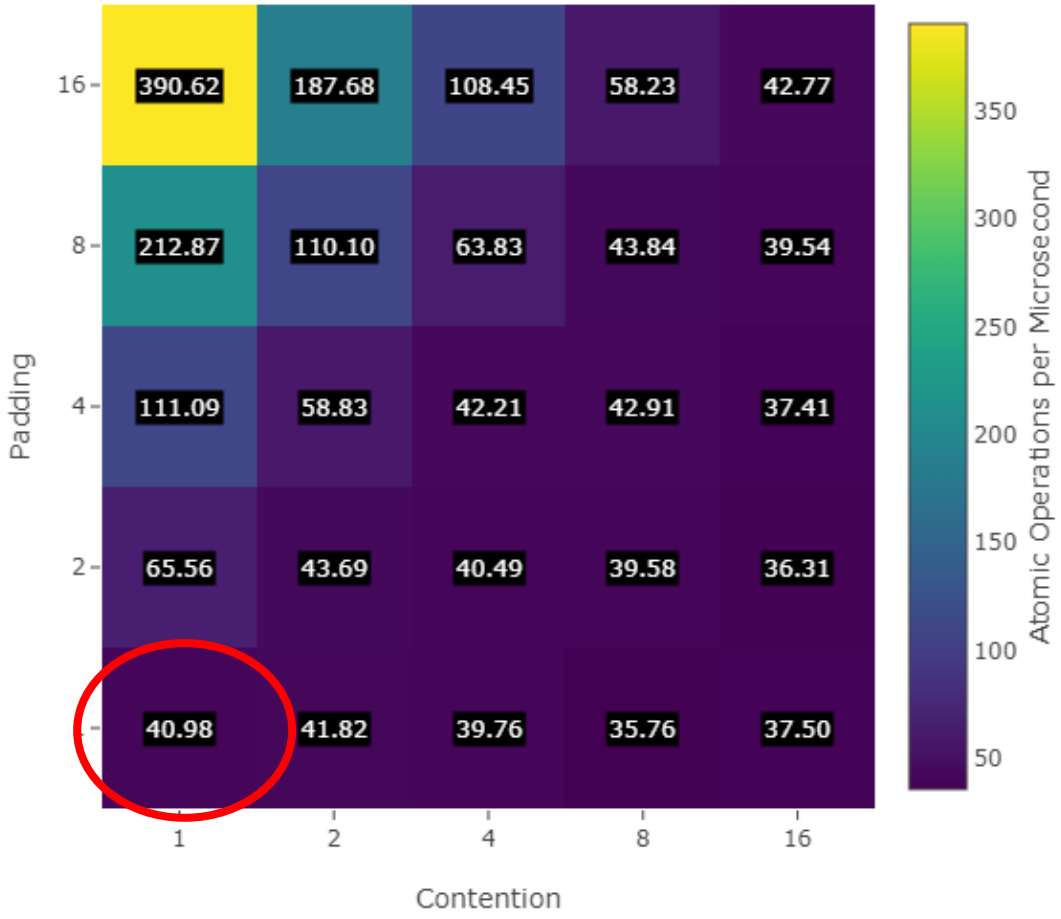
Live demo of CPU atomics:
devon.engineering/epiphron-web

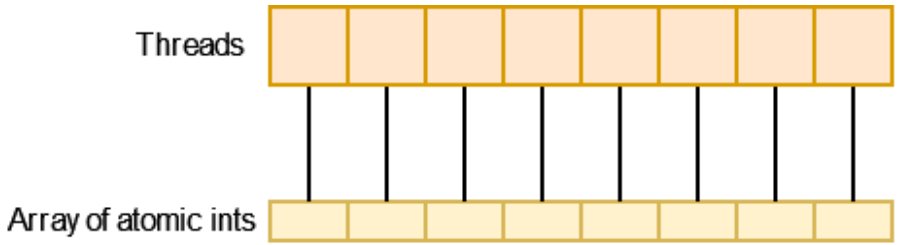
CPU ATOMIC RMWS



Live demo of CPU atomics:
devon.engineering/epiphron-web

CPU ATOMIC RMWS

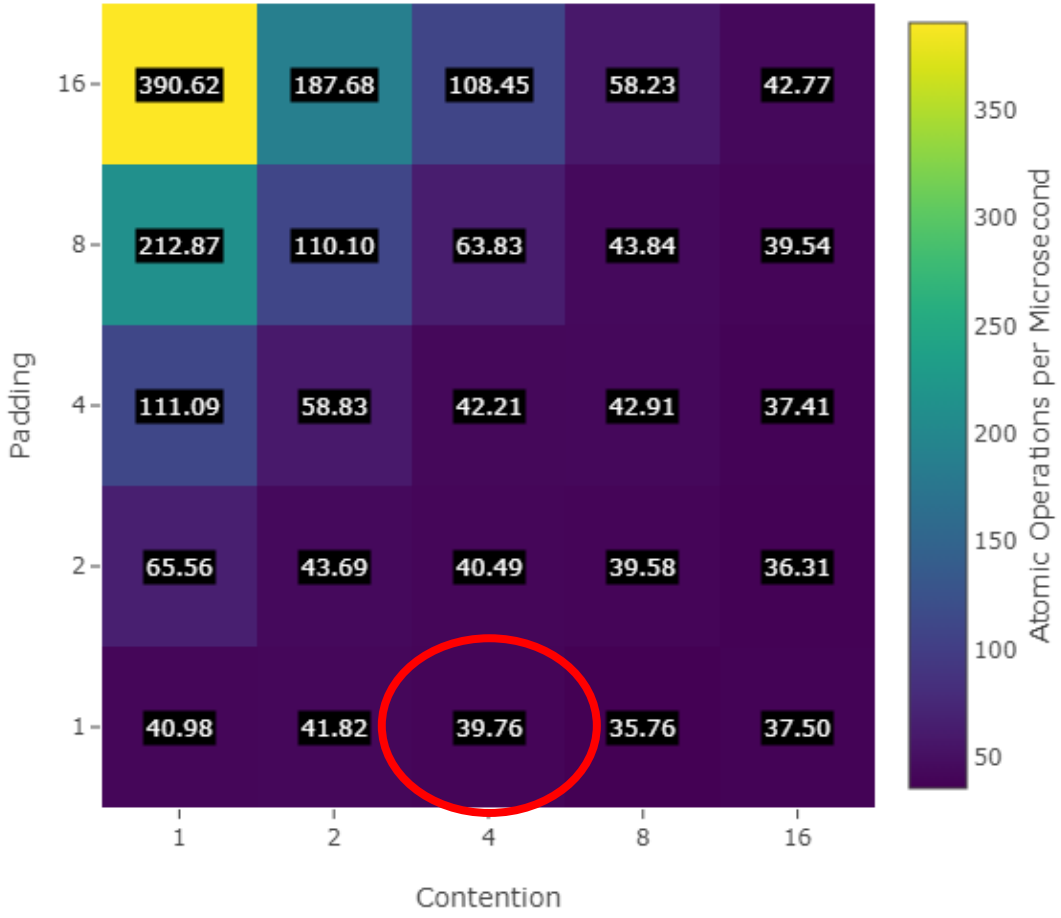


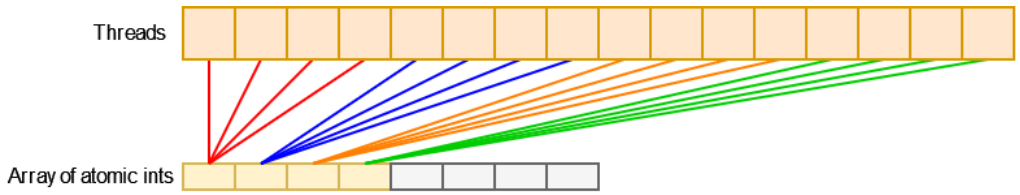


Contention: 1
Padding: 1

Live demo of CPU atomics:
devon.engineering/epiphron-web

CPU ATOMIC RMWS

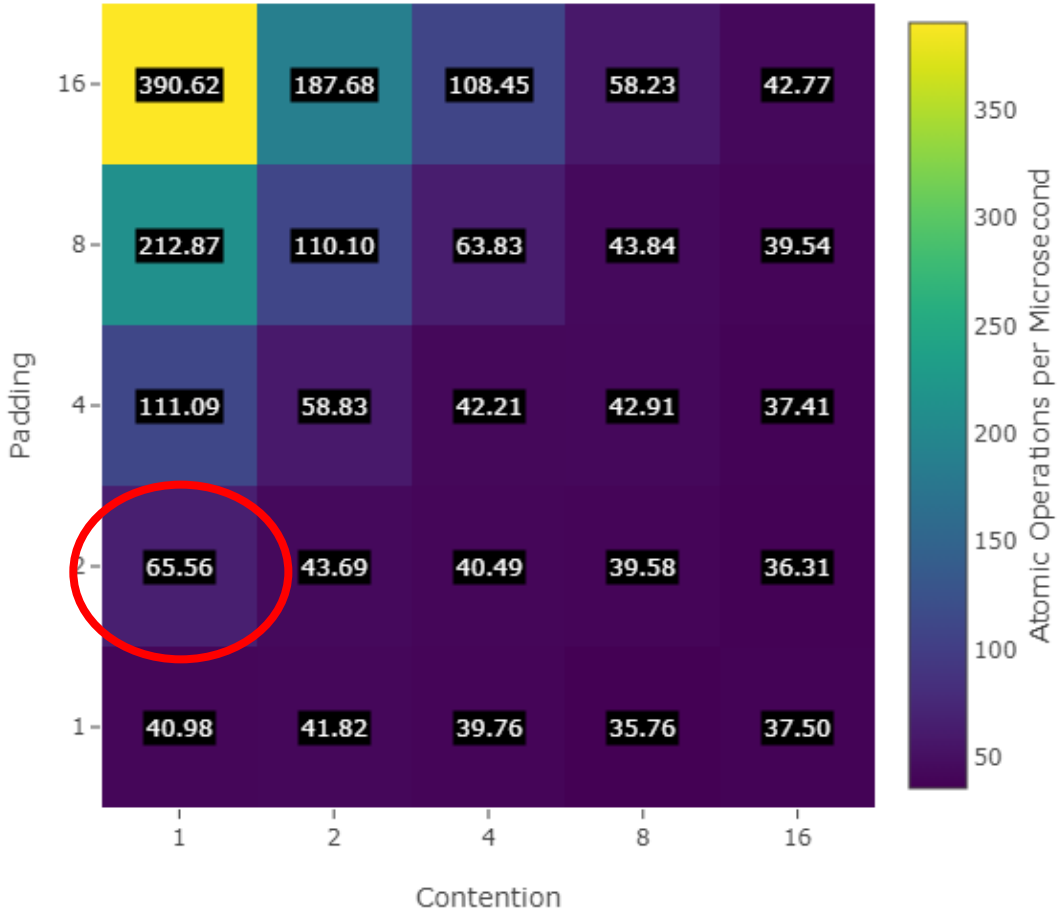


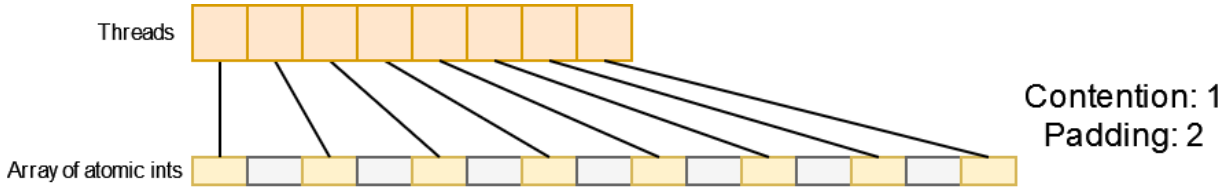


Contention: 4
Padding: 1

Live demo of CPU atomics:
devon.engineering/epiphron-web

CPU ATOMIC RMWS





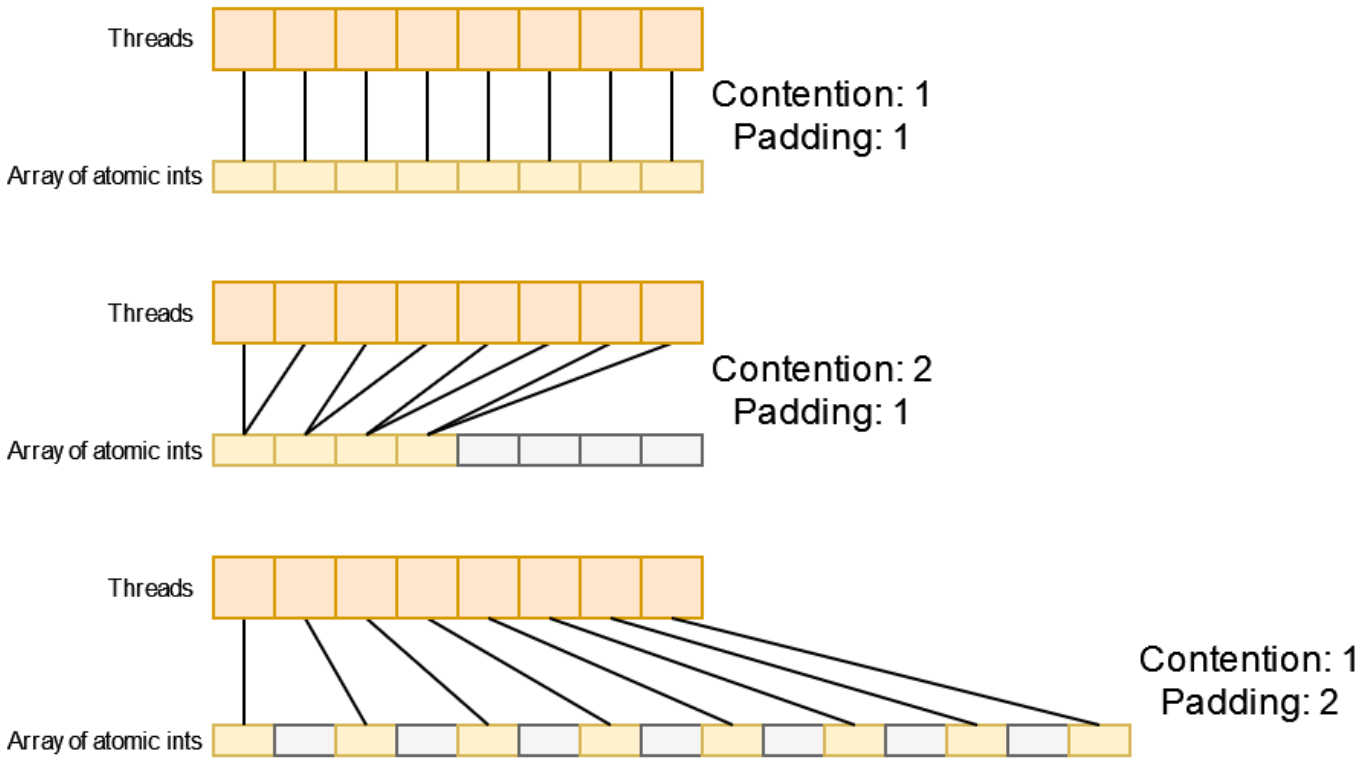
Live demo of CPU atomics:
devon.engineering/epiphron-web

GPU SYNCHRONIZATION

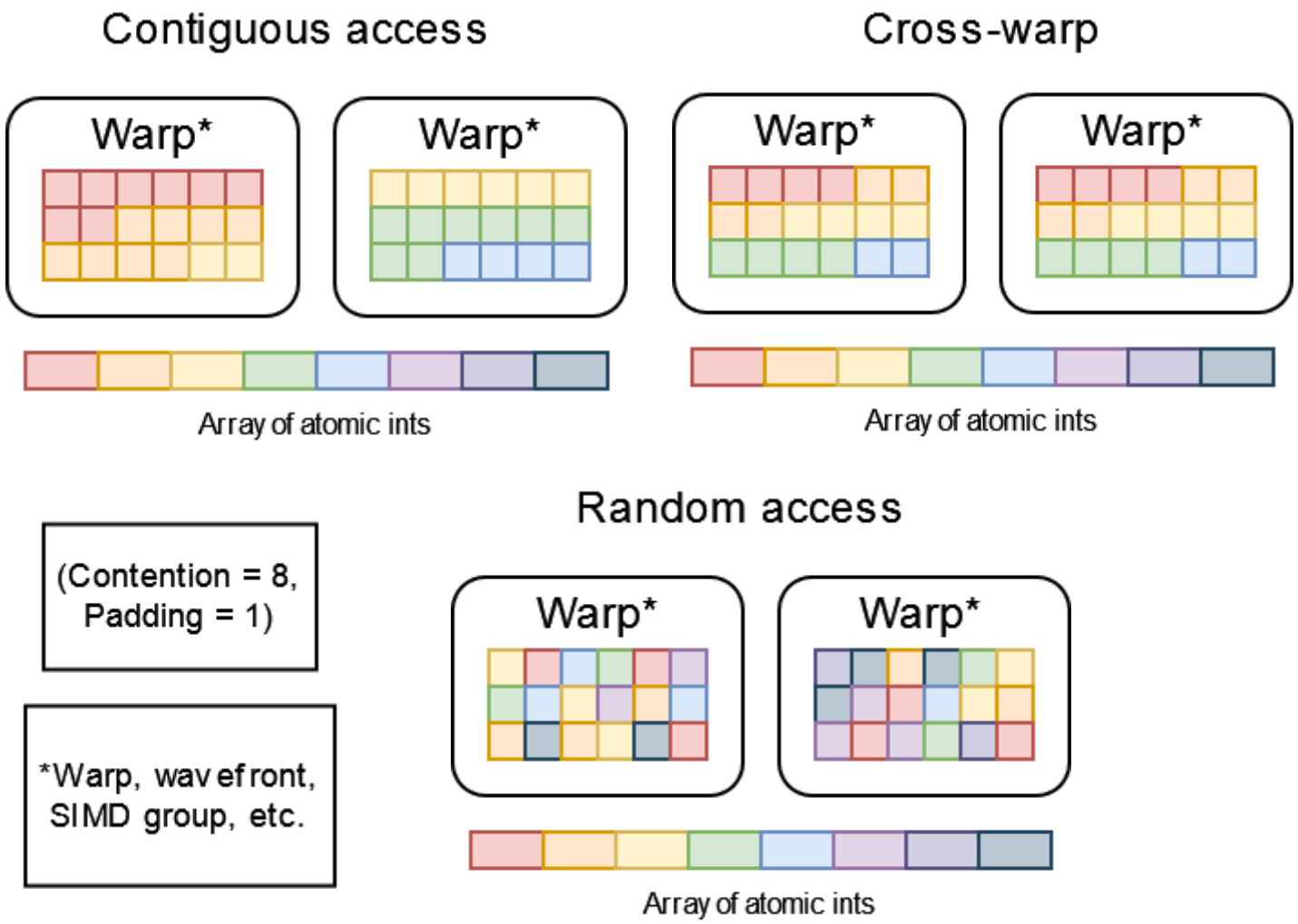
- GPU compute becoming increasingly critical
- GPUs provide a high degree of concurrency
- Heavily concurrent algorithms require synchronization
- GPU synchronization comes in the form of two major primitives:
  - Atomic instructions
  - Barriers

GPU ATOMIC PERFORMANCE CHARACTERISTICS

- GPU atomic performance varies greatly across devices
- GPU atomics sensitive to several performance characteristics
  - Contention
  - Padding
  - Thread access patterns



VULKAN MICROBENCHMARKING SUITE



- Contiguous access: chunking threads within warps
- Cross-warp: striding threads across warps
- Random access: distributing threads to atomic locations randomly*

VULKAN MICROBENCHMARKING SUITE

```
// Atomic Fetch Add Relaxed
__kernel void rmw_test( __global atomic_uint* res, global uint* iters,
                        global uint* indexes) {

    uint index = indexes[get_global_id(0)]; // chunking
    for (uint i = 0; i < *iters; i++) {
        atomic_fetch_add_explicit(&res[index], 1, memory_order_relaxed);
    }
}
```

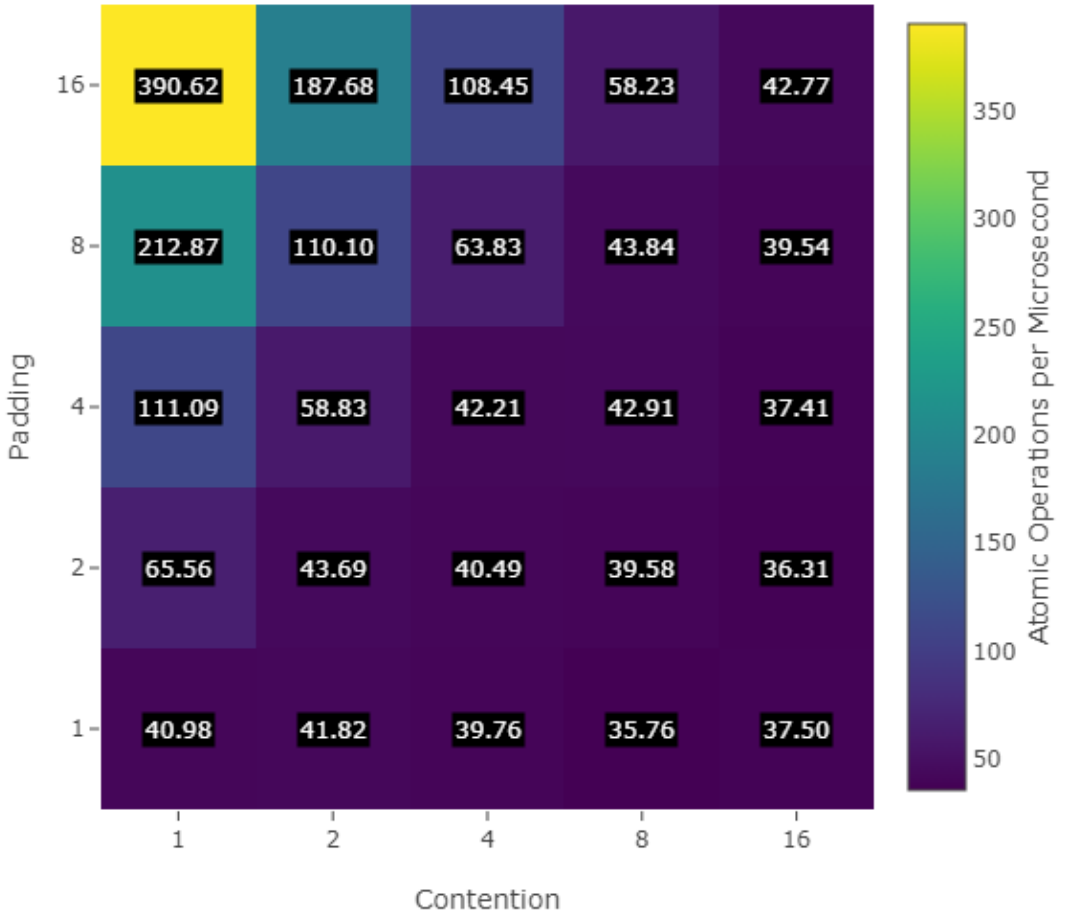
```
// Atomic Fetch Add Relaxed
__kernel void rmw_test( __global atomic_uint* res, global uint* iters,
                        global uint* indexes) {

    uint index = indexes[get_global_id(0)]; // striding
    for (uint i = 0; i < *iters; i++) {
        atomic_fetch_add_explicit(&res[index], 1, memory_order_relaxed);
    }
}
```

```
__kernel void rmw_test( __global atomic_uint* res, global uint* iters, global uint* indexes, global uint* buf_size) {
    uint prev = indexes[get_global_id(0)];
    uint index;
    for (uint i = 0; i < *iters; i++) {
        index = ((prev * 1664525) + 1013904223) % (*buf_size);
        atomic_fetch_add_explicit(&res[index], 1, memory_order_relaxed);
        prev = index;
    }
}
```

- Microbenchmarks currently targeting atomic read-modify-write (RMW) instructions
- Written in Vulkan for cross-platform testing
- C++ application, currently being rewritten for Android testing
- Uses OpenCL kernels, transpiled to SPIR-V using clspv

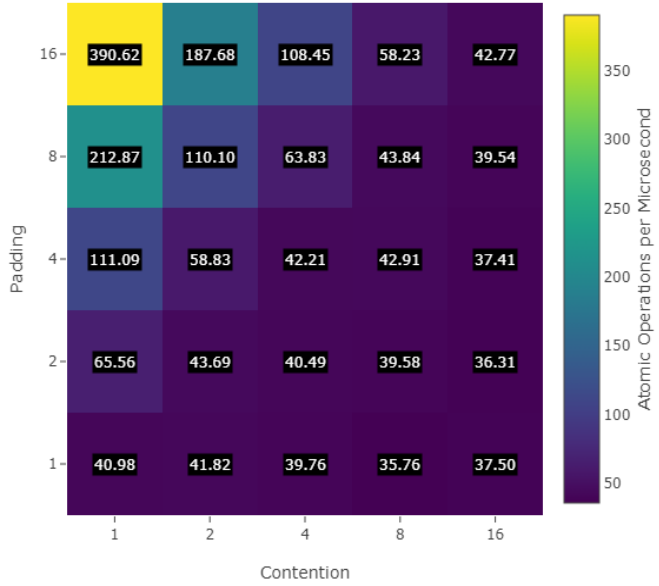
VULKAN MICROBENCHMARKING SUITE

- Benchmarks sweep values for contention and padding
- Separate benchmarks for each access pattern
- Heatmaps are produced to show trends in throughput

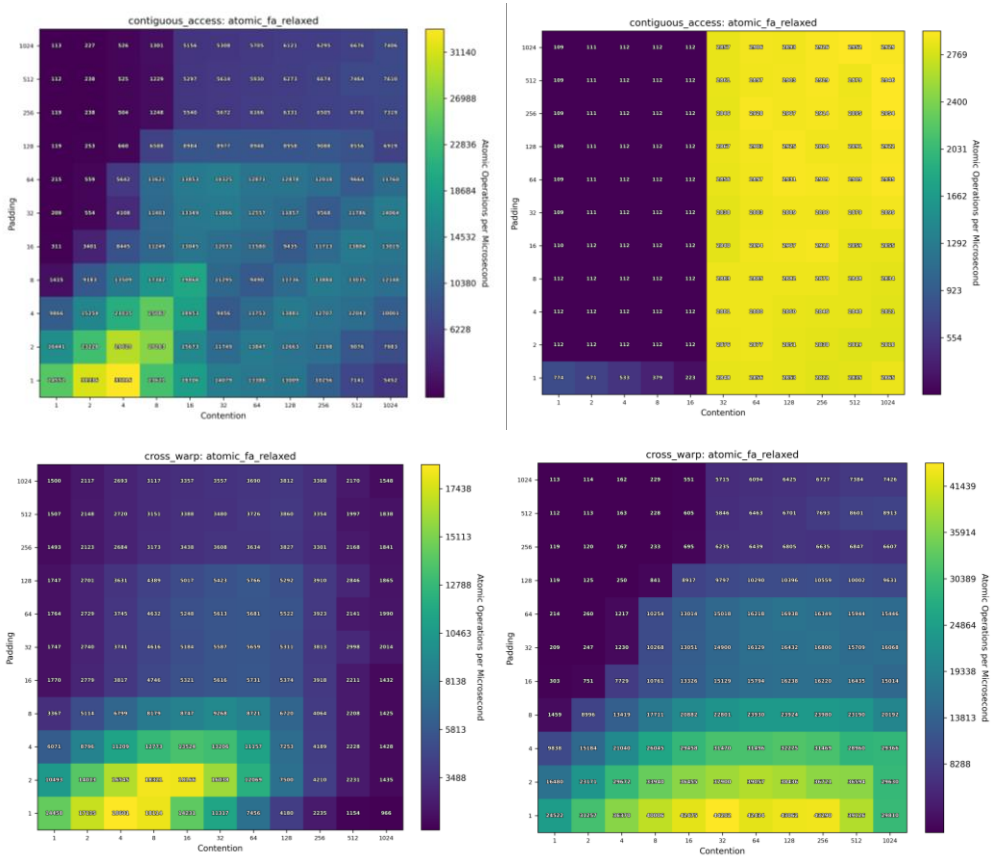


VULKAN MICROBENCHMARKING SUITE

CPU



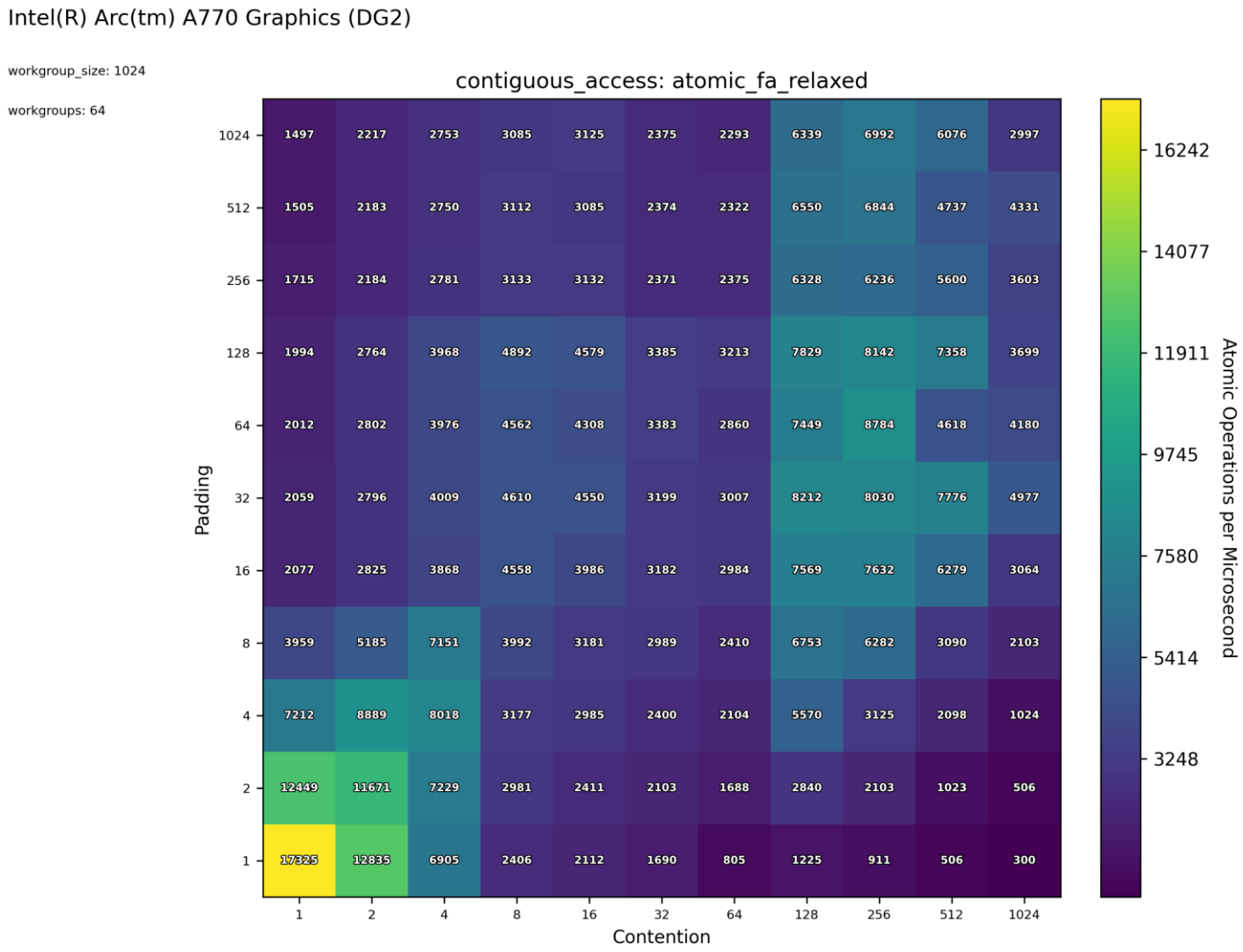
GPU



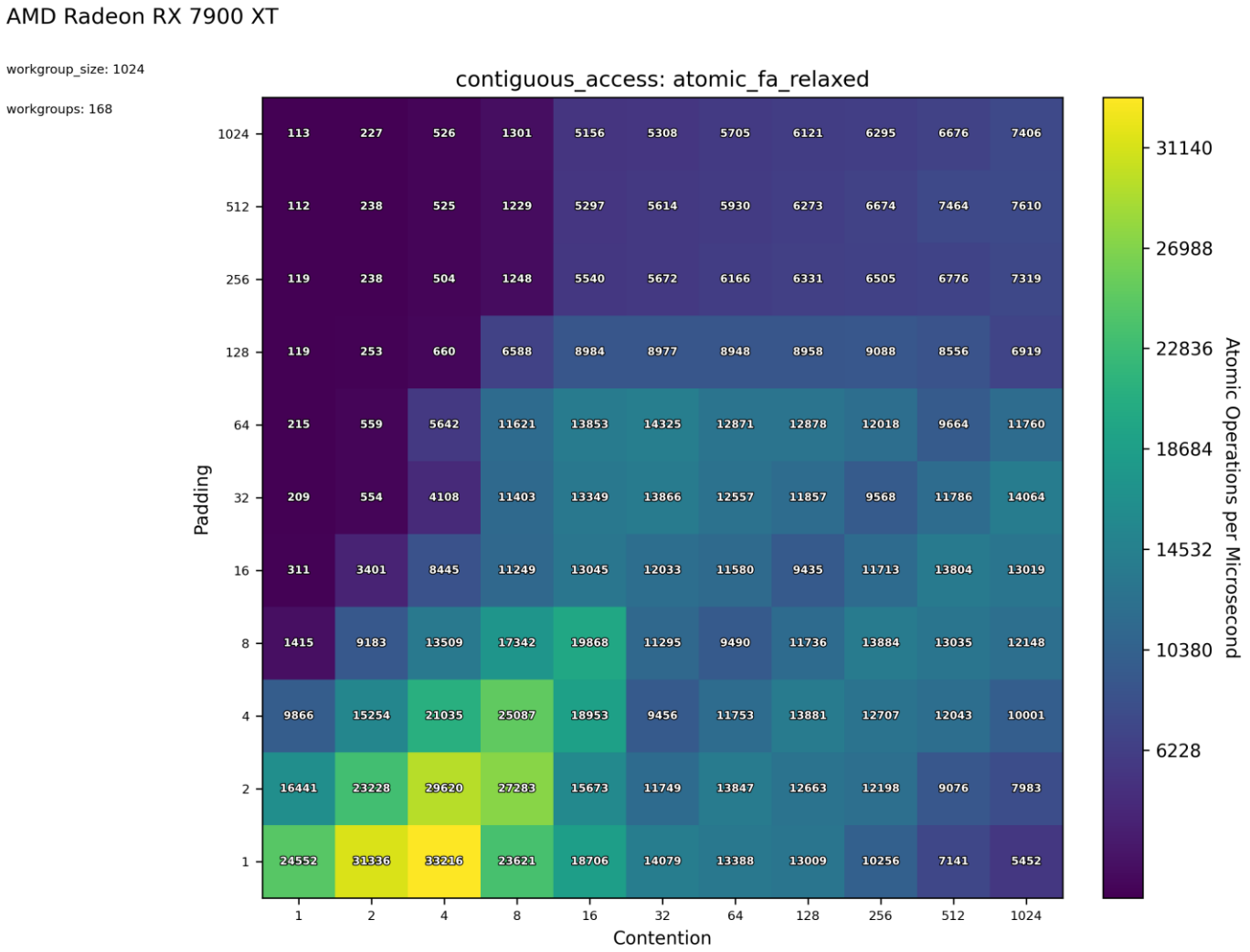
MICROBENCHMARK RESULTS

Intel discrete card results (contiguous access):

- Peak throughput at contention = 1, padding = 1
- Throughput degrades down to less than 5% of peak throughput in some scenarios



MICROBENCHMARK RESULTS



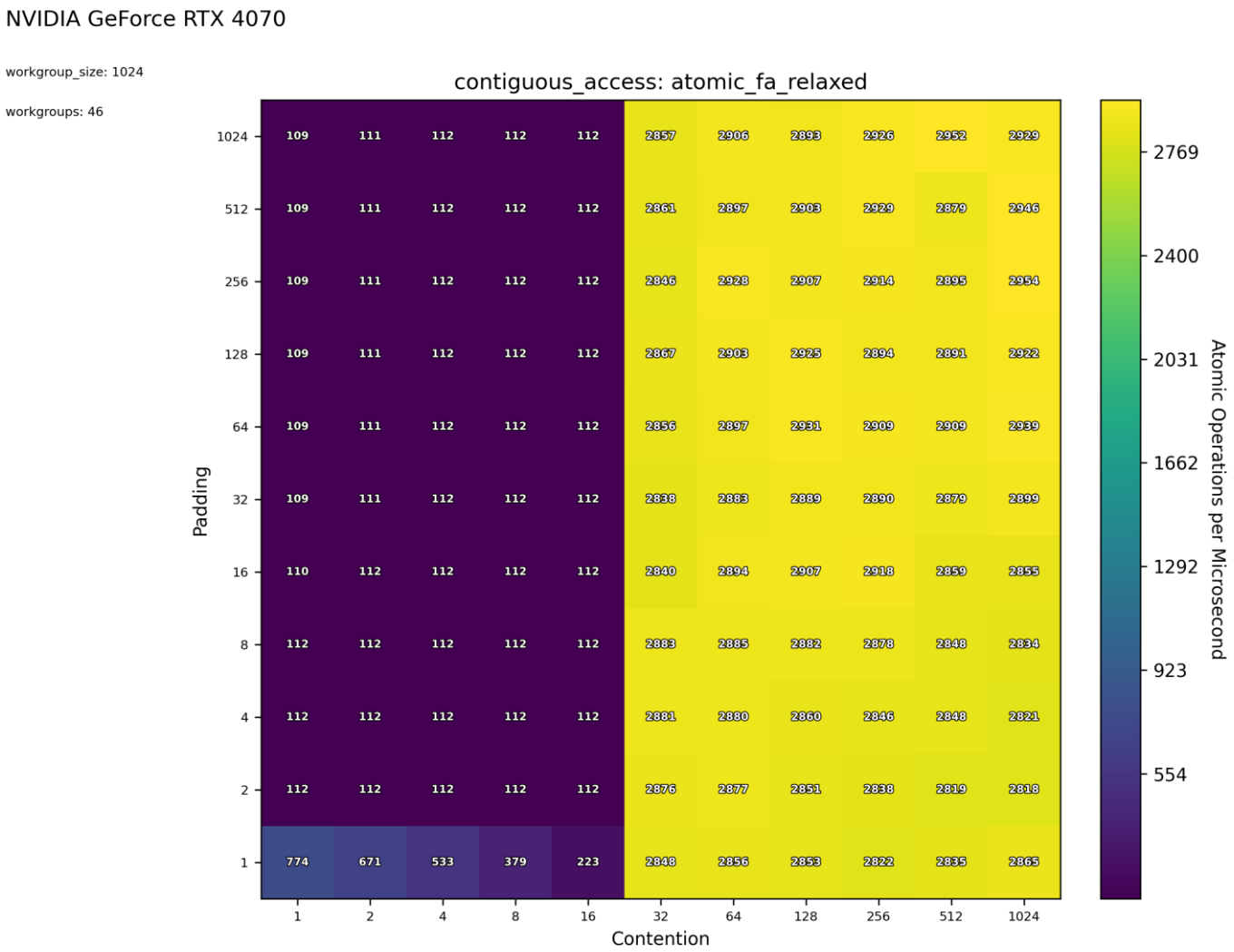
AMD discrete card results (contiguous access):

- Peak throughput at contention = 4, padding = 1
- Throughput degrades away all the way down to ~1% of peak

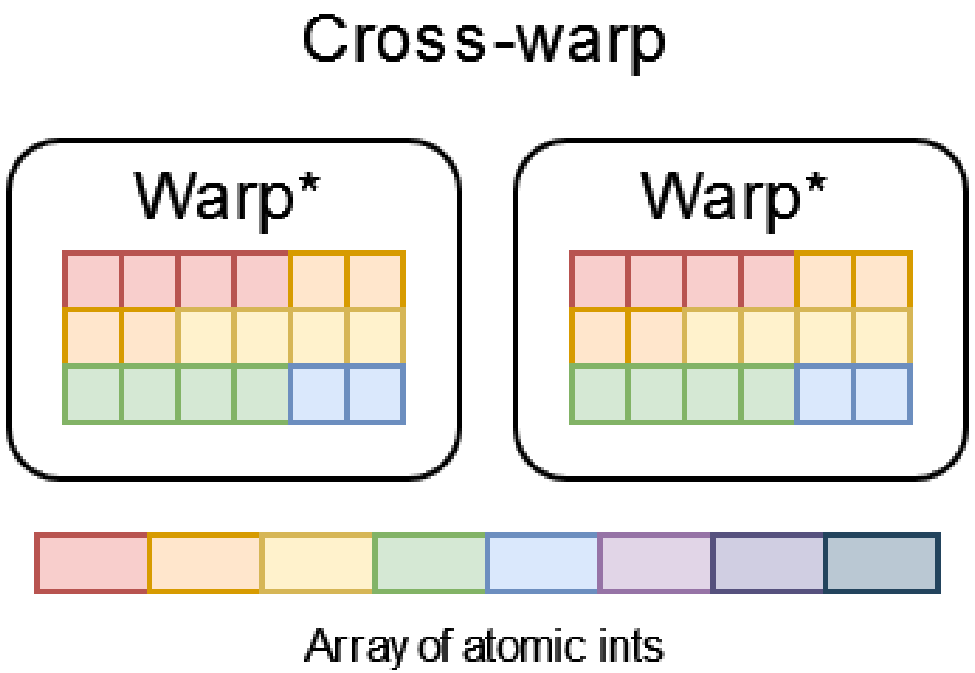
MICROBENCHMARK RESULTS

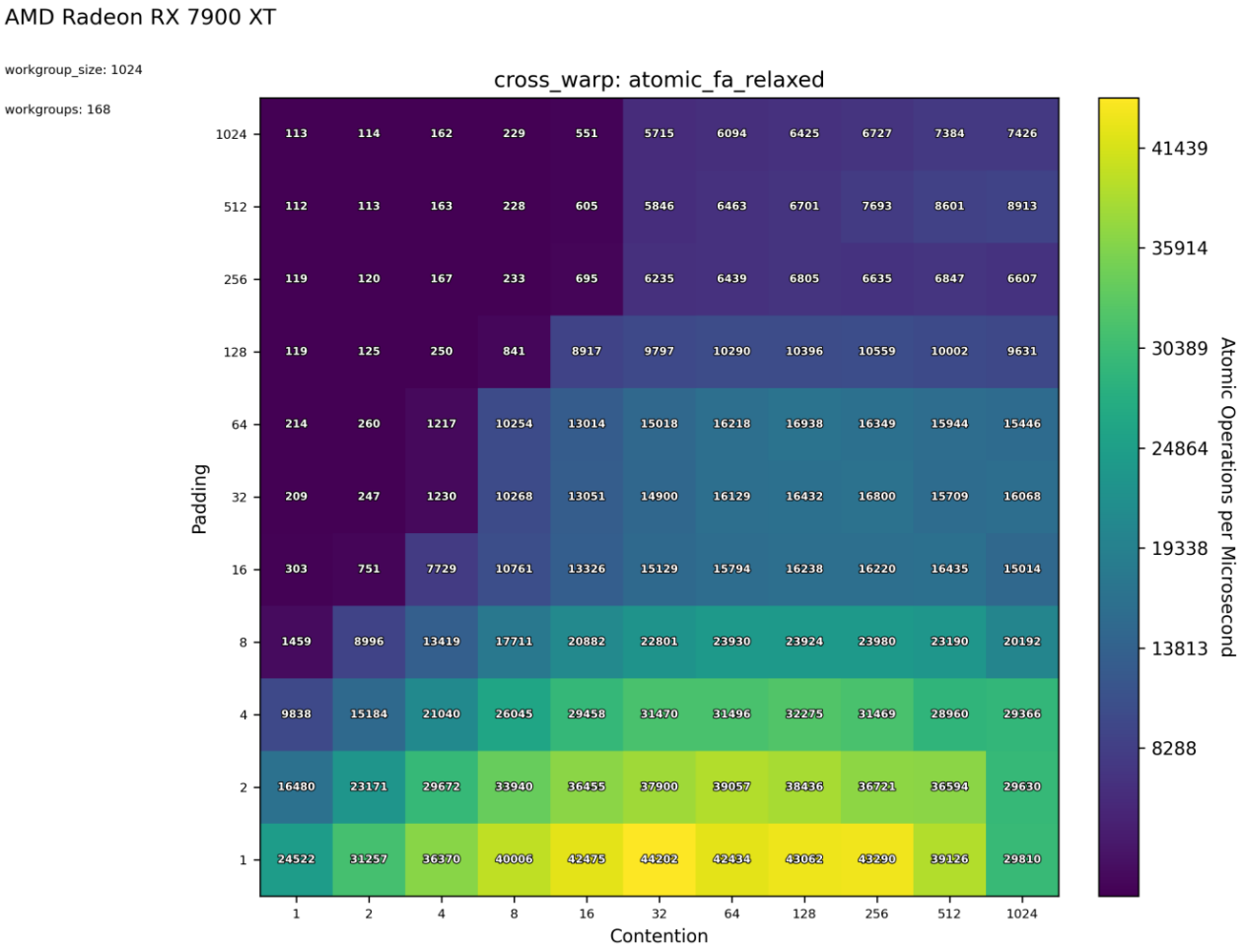
NVIDIA discrete card results (contiguous access):

- Peak throughput past contention = 32
- Slower region throughput is ~4% of peak region



MICROBENCHMARK RESULTS

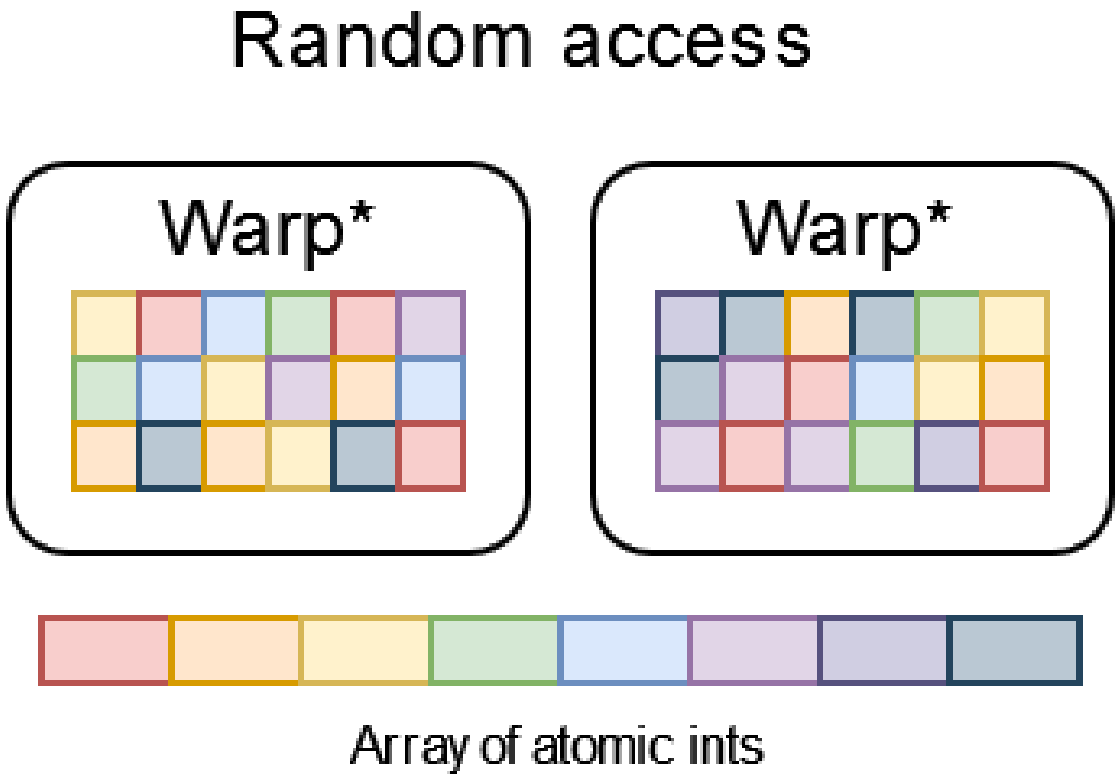




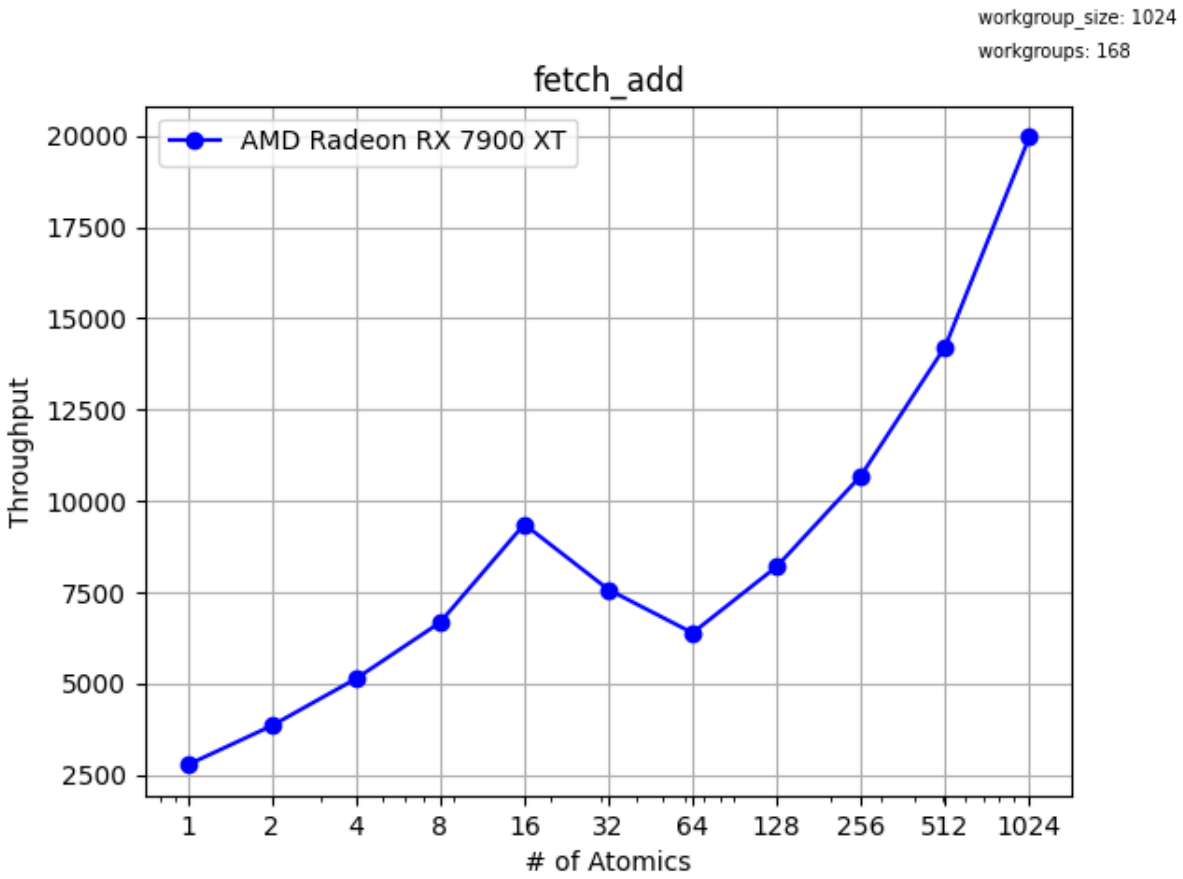
MICROBENCHMARK RESULTS

Random access microbenchmark:

- Distributes GPU threads randomly across array of atomic integers
- Varies size of array to distribute threads more



MICROBENCHMARK RESULTS



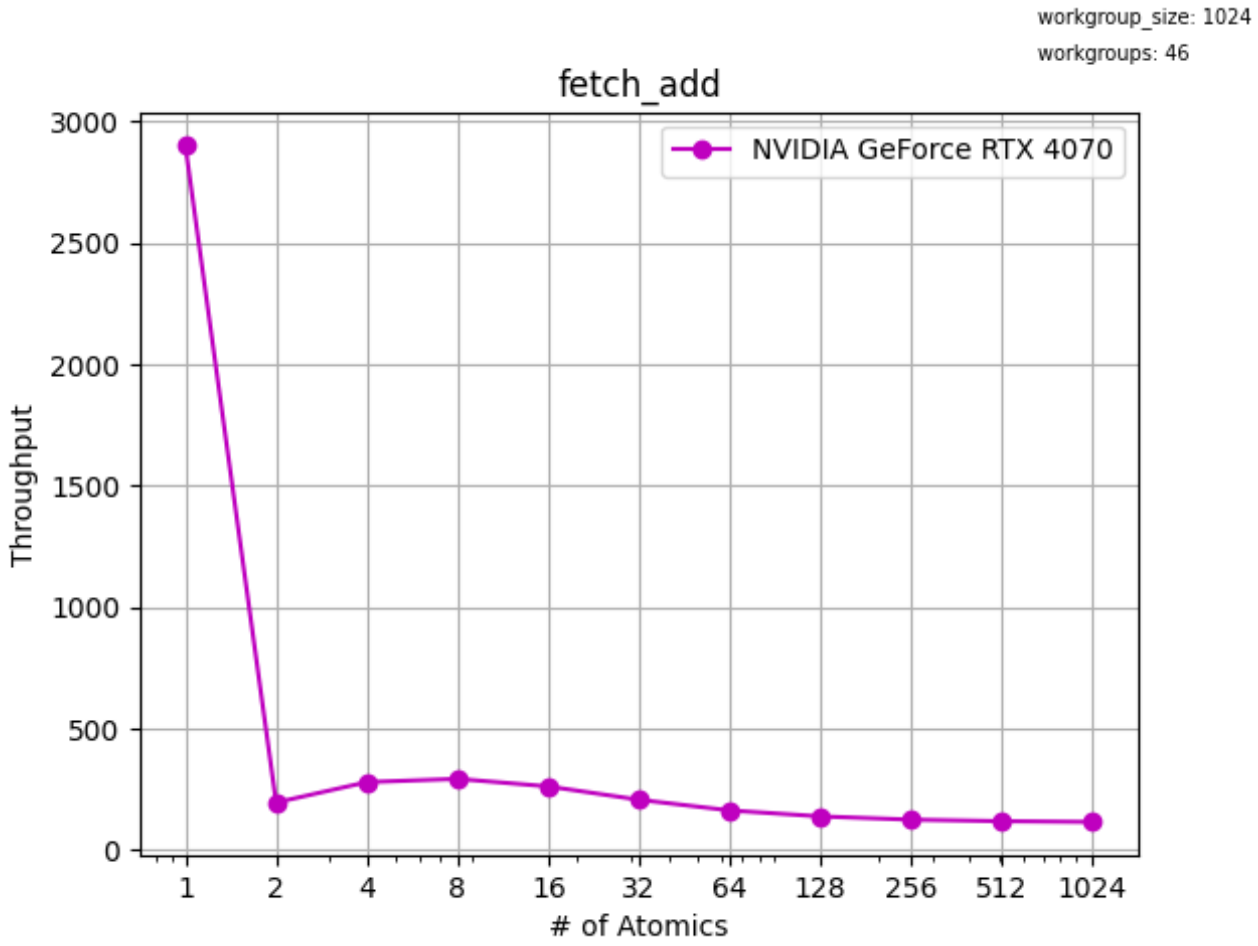
AMD discrete card results (random access):

- Throughput reaches local peak at 16 atomics, increases past 64 atomics
- With lower number of atomics needed, bit packing until 16 atomics used greatly improves throughput

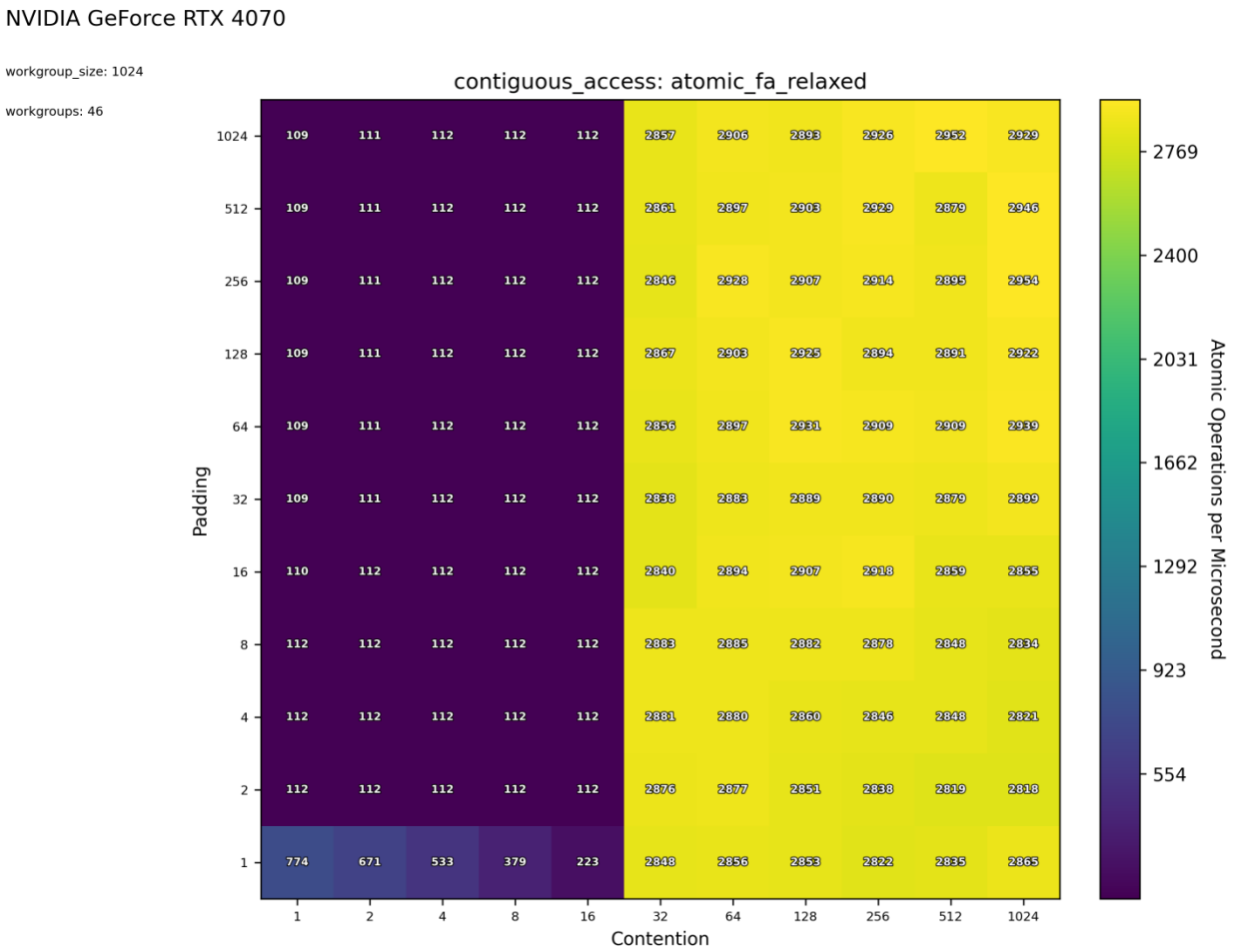
MICROBENCHMARK RESULTS

NVIDIA discrete card results (random access):

- Throughput falls immediately upon number of atomics increasing
- If application can reduce number of atomic locations used to 1, throughput is greatly improved



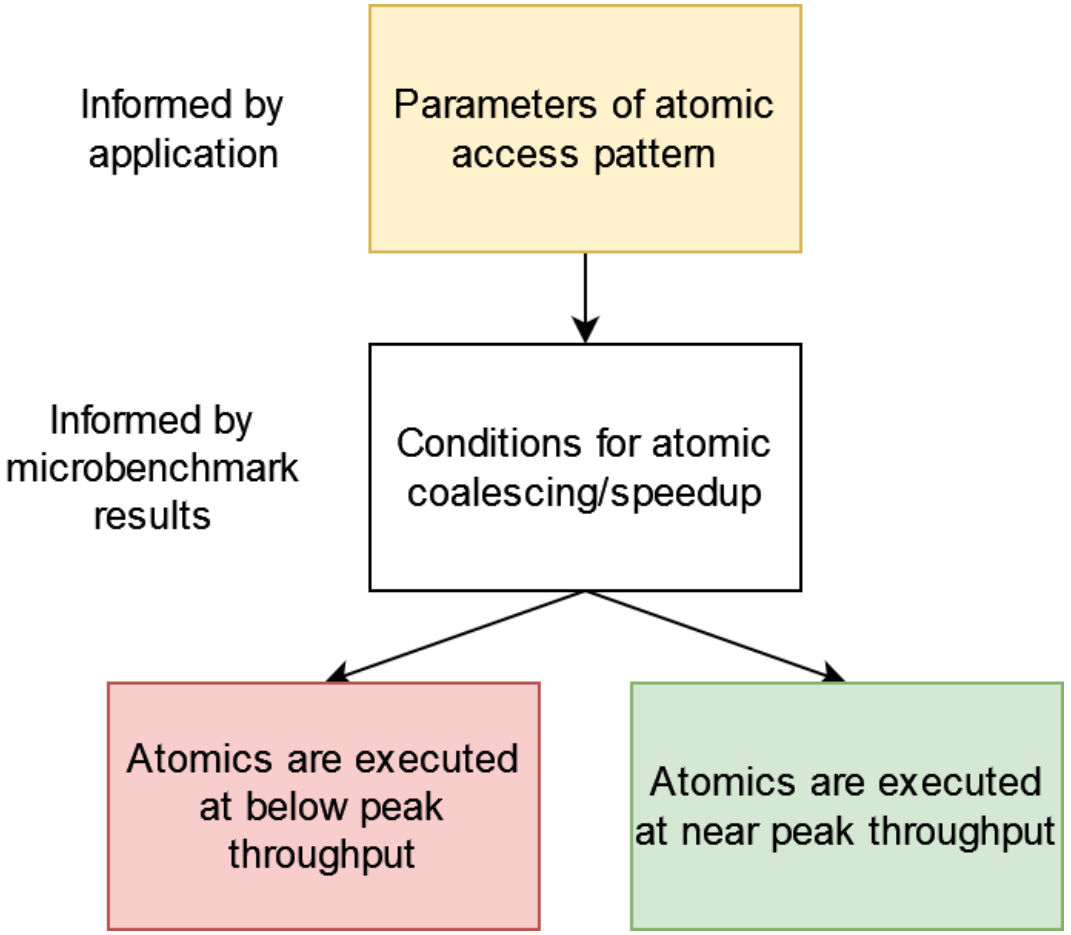
ATOMIC PERFORMANCE MODEL



- How do we characterize these results?
- Can atomic performance profiles be reduced to conditions to give greater throughput?

ATOMIC PERFORMANCE MODEL

- Working on creating performance model to describe atomic behavior
- Basic model involves some access pattern and conditions for coalescing
- Complexities of some results make it hard to identify conditions



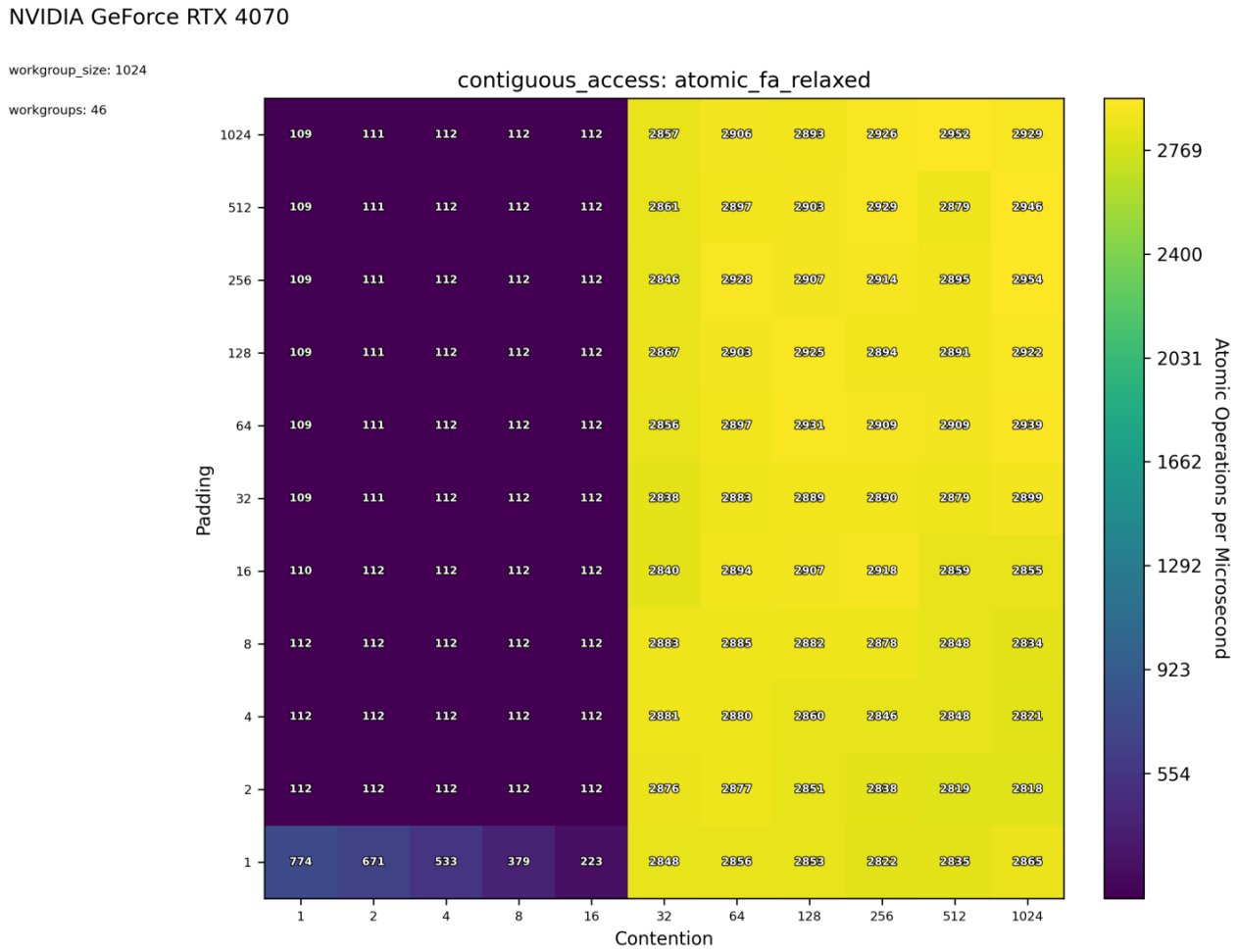
ATOMIC PERFORMANCE MODEL

Contiguous distribution of threads with chunks of 64 threads accessing same atomic location

At least 32 threads contending on the same location

Atomics are executed at below peak throughput

Atomics are executed at near peak throughput



ATOMIC PERFORMANCE MODEL

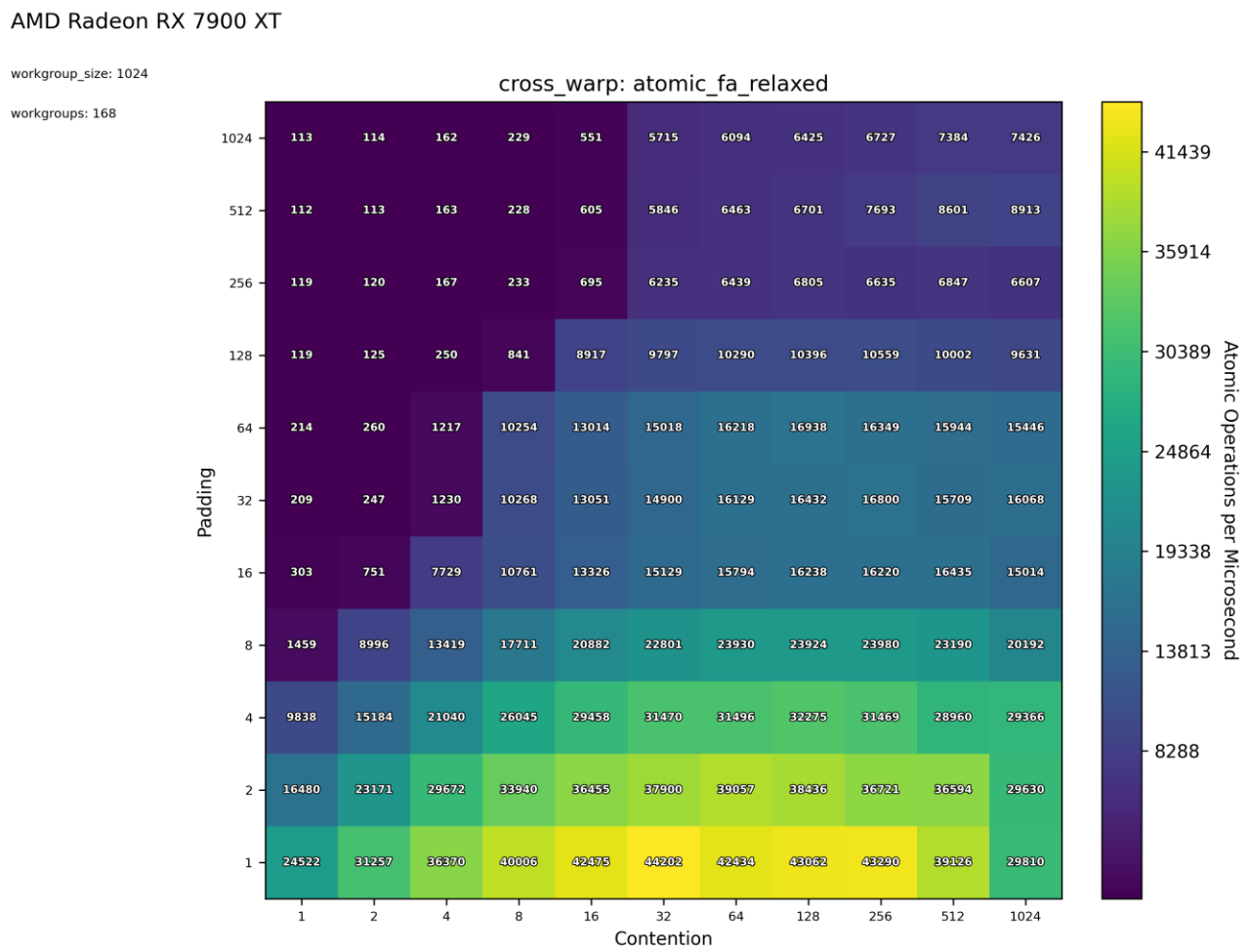
Warp strided distribution of threads across a number of atomic locations

↓

Conditions for atomic coalescing/speedup

↓

Some function of peak throughput

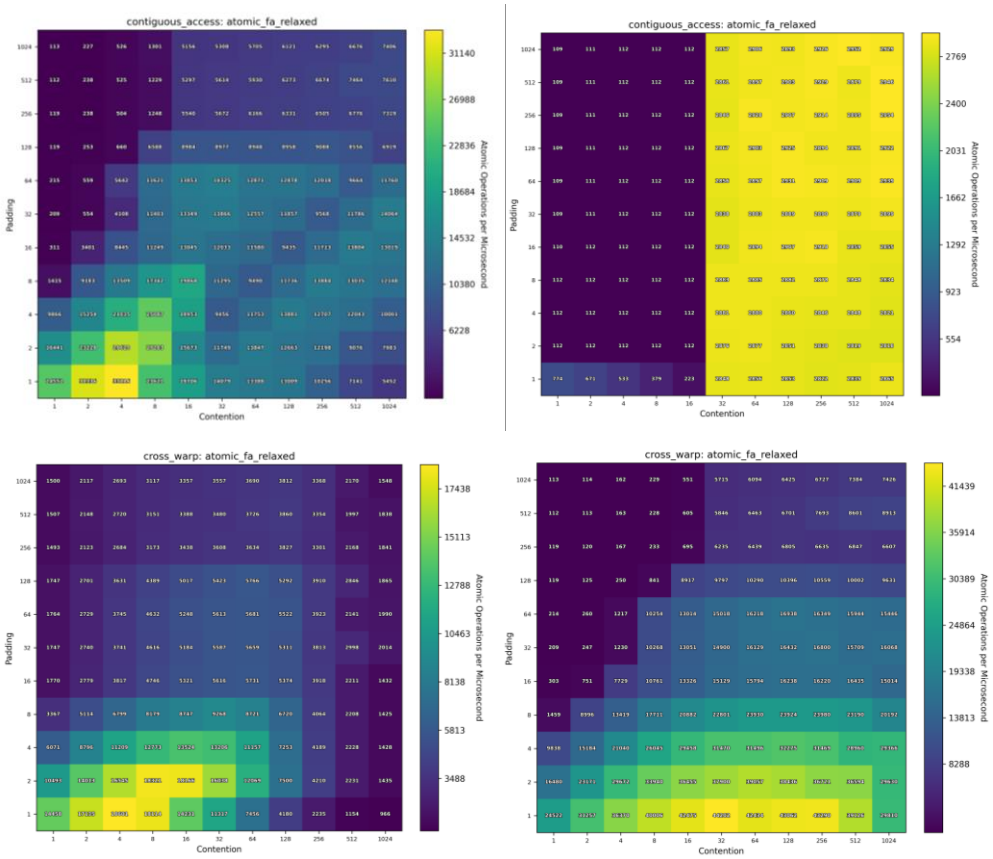


SUMMARY OF CONTRIBUTIONS

- Study of relative atomic performance across GPUs
- Microbenchmark suite to be executed on any device supporting Vulkan
- General performance model to capture complexities of GPU atomic performance

CONCLUSION

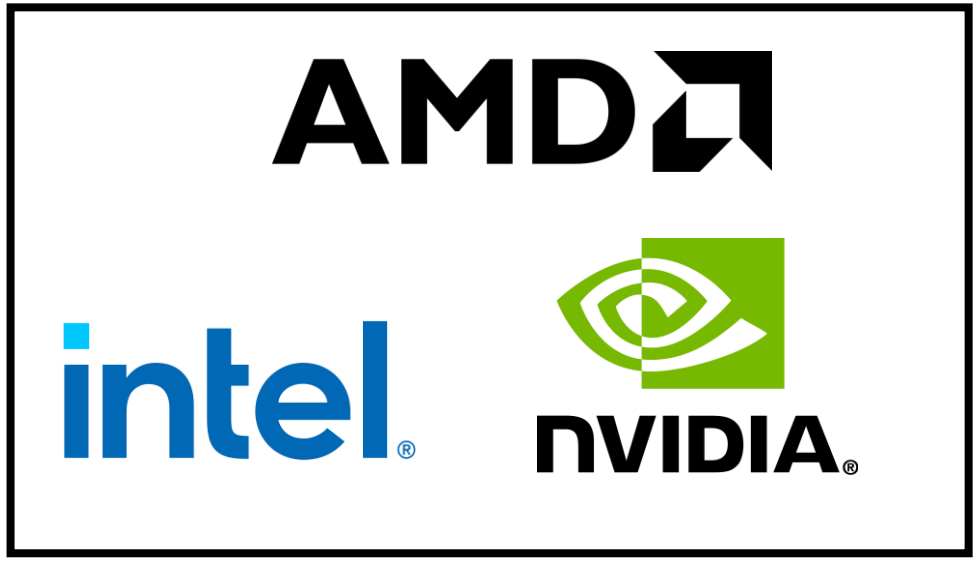
GPU



- How do these results help developers optimize applications?
- Performance model allows developers to structure atomic RMWs as a device-specific optimization
- Looking for opportunities to include these results as device-specific information in Vulkan specification

FUTURE DIRECTIONS

- Modification of atomic implementation in GPGPU simulator to examine effect on applications
- Expanded testing on new devices, including mobile platforms and Apple devices (through MoltenVK)
- Application of methodology to other synchronization primitives (namely, barriers)



THANK YOU

Devon McKee

dlmckee@ucsc.edu

Tyler Sorensen, Primary Investigator

Ishita Chaturvedi, Project Collaborator

Gurpreet Dhillon, Project Collaborator

Sean Siddens, Project Collaborator