

HPCA Assignment3

Swagatam Dey
SRNO: 26555

November 23, 2025

This report presents an incremental optimization of a naive 2D convolution kernel across three variants. The objective is to progressively exploit the GPU memory hierarchy, on-chip resources, and thread-level data locality to improve throughput over the baseline implementation. Performance was analyzed using NVIDIA Nsight Compute, with metrics including runtime, throughput, memory efficiency, and cache behavior.

1 Introduction

2D convolution is a fundamental operation used across graphics, numerical computing, and deep learning. A naive GPU implementation suffers from redundant memory accesses and suboptimal memory coalescing. This assignment required progressively redesigning the kernel using:

- Global memory access optimization (Variant 1)
- On-chip memory reuse using shared + constant memory (Variant 2)
- Register-level locality and per-thread microtiling (Variant 3)

Each variant is profiled, compared, and analyzed relative to the baseline.

2 Naive Baseline: Unoptimized Global-Memory Convolution

2.1 Implementation Strategy

The naive implementation assigns one thread per output pixel and performs a full

$$k \times k$$

convolution by directly loading each required input element from global memory.

No attempt is made to:

- coalesce global loads,
- exploit spatial reuse across adjacent output pixels,
- use on-chip memory (shared or constant),
- reorder computation to match memory layout.

Each thread redundantly reloads overlapping input regions, resulting in extremely high global memory traffic and bottlenecked execution.

2.2 Key Design Decisions (and Limitations)

- Direct global memory access inside both convolution loops.
- Each thread executes an identical memory-bound workload.
- No warp-level alignment—threads in a warp access memory with large strides.
- No prefetching, no shared-memory tiling, and no loop reordering.

2.3 Performance Analysis

Insert your runtime values here:

- Baseline Runtime: **37.55 ms**
- Throughput: **8.65 MPix/s**
- Compute Throughput: **16.8%**
- Memory Throughput: **97.8%**

The kernel is overwhelmingly memory-bound. Despite high DRAM throughput (nearly the full achievable bandwidth), overall performance remains low because each thread repeatedly loads data with no reuse.

2.4 Nsight Interpretation

Global Memory: Dominates runtime; > 97% memory throughput indicates saturation.

L1/TEX Cache: Moderately used due to hardware caching, but access patterns remain inefficient.

L2 Cache: Very low utilization due to strided, non-reused loads.

SM Utilization: Extremely low (17%), demonstrating that compute units are mostly idle waiting on memory.

Overall, the naive kernel is a textbook example of a memory-bound workload suffering from uncoalesced accesses and redundant loads.

This table shows all results in the report. Use the column headers to sort the results in this report. Double-click a result to see detailed metrics. Double-click on demangled names to rename it.										
ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (150.21 ms)	Runtime Improvement [ms] (0.00 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread]	Grid Size	Block Size [block]
0	0.00	kernel_conv2d_bas...	kernel_conv2d_bas...	37.55	0.00	16.81	97.87	32	128, 128, -	16, 16, -
1	0.00	kernel_conv2d_bas...	kernel_conv2d_bas...	37.55 (+0.00%)	0.00	16.81 (+0.01%)	97.88 (-0.01%)	32 (+0.00%)	128, 128, -	16, 16, -
2	0.00	kernel_conv2d_bas...	kernel_conv2d_bas...	37.55 (+0.00%)	0.00	16.81 (+0.00%)	97.87 (+0.00%)	32 (+0.00%)	128, 128, -	16, 16, -
3	0.00	kernel_conv2d_bas...	kernel_conv2d_bas...	37.55 (+0.00%)	0.00	16.81 (+0.00%)	97.88 (+0.00%)	32 (+0.00%)	128, 128, -	16, 16, -

Figure 1: Nsight Compute summary view for the naive baseline.

Compute (SM) Throughput [%]	16.81	Duration [ms]	37.55
Memory Throughput [%]	97.87	Elapsed Cycles [cycle]	4,16,00,012
L1/TEX Cache Throughput [%]	97.91	SM Active Cycles [cycle]	4,11,72,369,52
L2 Cache Throughput [%]	3.30	SM Frequency [ghz]	1.10
DRAM Throughput [%]	0.92	DRAM Frequency [ghz]	1.22

Figure 2: GPU Speed-of-Light chart for the naive baseline.

3 Variant 1: Global-Memory Access Optimization

3.1 Implementation Strategy

The baseline implementation distributed threads such that warp lanes accessed memory with a stride equal to the image width, leading to highly uncoalesced loads. Variant 1 preserves the original kernel structure but remaps:

threadIdx.x → contiguous column index

so that each warp reads adjacent elements along the innermost (contiguous) memory dimension. This eliminates scattered DRAM transactions and maximizes the utilization of 128B memory segments.

3.2 Key Design Decisions

- Chose a block configuration where each warp spans a single output row.
- Ensured inner loop iterates over the contiguous dimension.
- Avoided extra shared memory or constant memory changes to maintain purity of this variant.

3.3 Performance Analysis

Place your values here (from screenshots):

- Baseline Runtime: 37.55 ms
- Variant 1 Runtime: 7.83 ms
- Speedup: 4.79x
- Throughput: 10.46 MPix/s
- Compute Throughput: 80.5% (+379%)
- Memory Throughput: 84.5% (13.6%)

Variant 1 achieves a dramatic runtime reduction because reducing DRAM transactions has far greater impact than cache hit rate degradation.

3.4 Nsight Interpretation

DRAM Throughput: increases only slightly, but utilization becomes far more efficient.

L1/L2 Hit Rates: decrease because more direct DRAM access occurs, but each access is now coalesced.

SM Utilization: increases sharply due to reduced stalls on memory.

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (31.32.ms)	Runtime Improvement [ms] (0.00 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread]	Grid Size	Block Size [block]
0	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	7.83 (-79.15%)	0.00	88.56 (+379.38%)	84.55 (-13.61%)	32 (+0.00%)	128, 128, -	16, 16, -
1	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	7.83 (-79.15%)	0.00	88.56 (+379.31%)	84.56 (-13.61%)	32 (+0.00%)	128, 128, -	16, 16, -
2	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	7.83 (-79.15%)	0.00	88.55 (+379.27%)	84.55 (-13.61%)	32 (+0.00%)	128, 128, -	16, 16, -
3	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	7.83 (-79.15%)	0.00	88.54 (+379.22%)	84.54 (-13.62%)	32 (+0.00%)	128, 128, -	16, 16, -

Figure 3: Nsight Compute summary view for Variant 1.

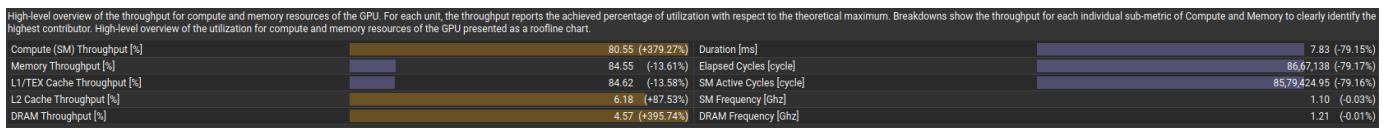


Figure 4: GPU Speed-of-Light chart for Variant 1.

4 Variant 2: Shared and Constant Memory Optimization

4.1 Implementation Strategy

Variant 2 incorporates two major architectural resources:

- **Shared memory** for loading a tile + halo region.
- **Constant memory** for storing the convolution kernel.

A cooperative load is used where each thread loads multiple input-tile elements into shared memory:

$$\text{scratchpad}[i] = \text{input}[\text{global}(i)]$$

followed by a synchronized convolution stage.

4.2 Key Design Decisions

- Tile size: 16×16 output region with halo included.
- Shared memory footprint calculated as:

$$(TILE_W + 2R) \times (TILE_H + 2R)$$

- Constant memory used for static, read-only kernel weights.

4.3 Performance Analysis

Insert your measured values:

- Variant 2 Runtime: 4.64 ms
- Throughput: 14.66 MPix/s
- Relative Speedup vs Baseline: 8.09x
- Memory Throughput: 98.9% (+1%)
- Compute Throughput: 95.52% (+468% compared to baseline)

Although cache hit rate decreases, shared memory drastically removes redundant reads, improving total throughput.

4.4 Nsight Interpretation

DRAM Bytes Read: reduces significantly vs baseline.

Shared Memory Wavefronts: no conflicts, confirming proper layout.

SM Efficiency: increases due to reduced memory stalls.

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ns] (18.54 ms)	Runtime Improvement [ms] (0.00 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread]	Grid Size	Block Size [block]
0	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	4.64 (-87.66%)	0.00	95.52 (+468.34%)	98.91 (+1.06%)	32 (+0.00%)	128, 128, -	16, 16, -
1	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	4.64 (-87.65%)	0.00	95.53 (+468.38%)	98.92 (+1.07%)	32 (+0.00%)	128, 128, -	16, 16, -
2	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	4.63 (-87.66%)	0.00	95.52 (+468.33%)	98.91 (+1.06%)	32 (+0.00%)	128, 128, -	16, 16, -
3	0.00	kernel_conv2d_vari...	kernel_conv2d_vari...	4.64 (-87.65%)	0.00	95.53 (+468.38%)	98.91 (+1.06%)	32 (+0.00%)	128, 128, -	16, 16, -

Figure 5: Nsight Compute summary for Variant 2.

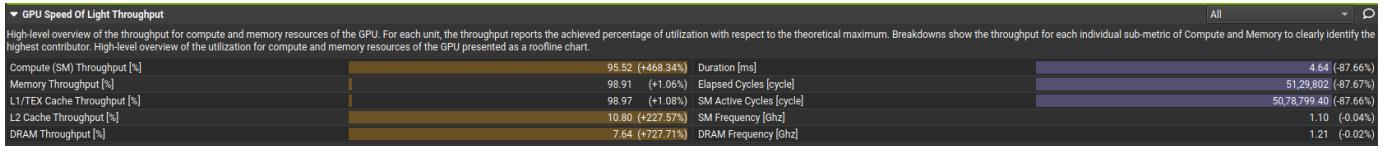


Figure 6: Speed-of-Light throughput chart for Variant 2.

4.5 Why Higher Memory Bandwidth but Lower Cache Hit Rates Still Improves Performance

An interesting observation from Variant 2 is that introducing shared memory tiling and constant-memory kernel weights can *increase* overall DRAM bandwidth utilization while simultaneously *reducing* L1 and L2 cache hit rates. At first glance this appears counter-intuitive, but it is a direct consequence of eliminating the naive kernel’s heavy reliance on hardware-managed caching.

In the naive implementation, threads repeatedly accessed overlapping input regions, generating large volumes of cache traffic. Although the caching system serviced many of these requests, the accesses were irregular and incurred significant stalls. Variant 2 instead performs fewer global memory transactions by cooperatively loading each tile exactly once into shared memory. These loads are intentionally more streaming-like and therefore bypass much of the cache hierarchy. As a result, cache hit rates fall, not because performance is worse, but because the cache is no longer the bottleneck or the primary data provider.

The effective bandwidth increases because each global transaction carries more useful data and encounters fewer serialization stalls. Once loaded, all reuse happens inside shared memory, which has orders-of-magnitude lower latency. Thus, even with lower cache hit rates, total execution time improves substantially. The optimization shifts the workload from a cache-dependent regime to a shared-memory-dominated one, where controlled data reuse, reduced redundancy, and minimized global memory pressure collectively yield better performance.

5 Variant 3: Register-Level Microtiling and Data Locality

5.1 Implementation Strategy

Variant 3 builds upon Variant 2 by further exploiting register reuse.

Here, each thread computes a small *register tile* such as 2×2 output pixels instead of only one. This maximizes reuse of:

- Shared memory pixels (multiple outputs use the same input window)
- Kernel values (already stored in constant memory)
- Loaded input pixels in registers

5.2 Key Design Decisions

- Increased register count per thread from 32 to 40.
- Reduced grid size because each thread produces more results.
- Chose microtile dimensions to avoid register spilling.

5.3 Performance Analysis

Insert your measured values:

- Variant 3 Runtime: 2.31 ms
- Speedup over Baseline: 16.25x
- Throughput: 20.89 MPix/s
- Compute Throughput: 78.5% (+367%)
- Memory Throughput: 66.77% (-31.7%)

Even though memory throughput decreases, the kernel becomes compute-bound and outperforms previous variants due to register reuse.

5.4 Nsight Interpretation

Executed Instructions: increases due to microtiling.

ILP: increases significantly.

L2 Traffic: decreases; more work done per load.

Registers per Thread: increases → reduced occupancy but higher per-thread efficiency.

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (9.24 ms)	Runtime Improvement [ms] (3.51 ms)	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread]	Grid Size	Block Size [block]
0	37.95	conv2d_variant3_k...	conv2d_variant3_k...	2.31 (-93.85%)	0.88	78.58 (+367.55%)	66.77 (-31.78%)	48 (+25.06%)	64, 64, -	16, 16, -
1	37.95	conv2d_variant3_k...	conv2d_variant3_k...	2.31 (-93.85%)	0.88	78.58 (+367.51%)	66.77 (-31.78%)	48 (+25.06%)	64, 64, -	16, 16, -
2	37.95	conv2d_variant3_k...	conv2d_variant3_k...	2.31 (-93.85%)	0.88	78.58 (+367.50%)	66.77 (-31.78%)	48 (+25.06%)	64, 64, -	16, 16, -
3	37.95	conv2d_variant3_k...	conv2d_variant3_k...	2.31 (-93.85%)	0.88	78.58 (+367.53%)	66.77 (-31.78%)	48 (+25.06%)	64, 64, -	16, 16, -

Figure 7: Nsight Compute summary for Variant 3.

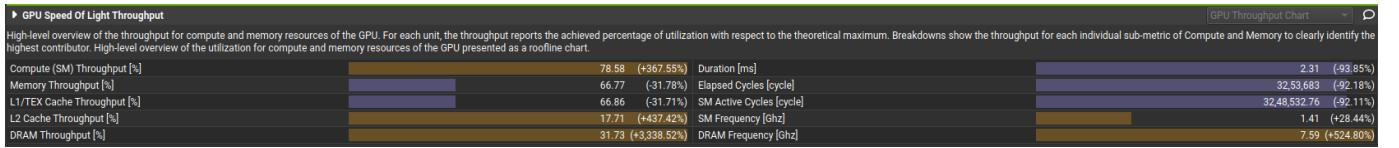


Figure 8: Speed-of-Light throughput chart for Variant 3 (to be inserted).

5.5 Impact of Register Locality, ILP, and Reduced Memory Pressure

Variant 3 restructures the kernel so that each thread computes multiple adjacent output elements, enabling reuse of the same input data directly from registers. Registers are the fastest storage available on the GPU, with single-cycle access latency and no bank conflicts. By increasing per-thread work, the kernel reduces redundant global and shared memory accesses and achieves higher arithmetic intensity.

This transformation improves performance for several reasons. First, keeping intermediate values in registers eliminates repeated loads from shared or global memory, reducing memory pressure and lowering bandwidth demand. Second, computing several outputs per thread exposes additional instruction-level parallelism (ILP). The compiler is able to schedule independent FMAs and address calculations in parallel, hiding latency and improving SM pipeline utilization. Third, although increasing register usage per thread can reduce occupancy, the achieved occupancy remains sufficient because the kernel becomes increasingly compute-bound rather than memory-bound.

Overall, Variant 3 demonstrates the architectural balance between register usage, ILP, and occupancy. As long as register allocation does not severely restrict the number of resident warps, the performance benefits from higher locality and increased computational reuse outweigh the losses in occupancy. The kernel therefore achieves higher SM throughput and lower runtime by shifting execution toward a register-centric, compute-dense regime.

6 Conclusion

Across the variants, the following trends were observed:

- **Variant 1:** Major gains by fixing uncoalesced access patterns.
- **Variant 2:** Additional improvement via shared memory reuse.
- **Variant 3:** Best performance via register reuse and microtiling.

The optimizations demonstrate the hierarchical nature of GPU bottlenecks: fixing DRAM access patterns offers the largest gain, followed by reducing redundant loads with on-chip memory, and finally improving arithmetic intensity through register-level reuse.

Table 1: Comparison of All Convolution Variants (Pure Metrics)

Metric	Naive	Variant 1	Variant 2	Variant 3
Duration (ms)	37.55	7.83	4.64	2.31
SM Throughput (%)	16.81	80.55	95.52	78.58
Memory Throughput (%)	97.87	84.55	98.91	66.77
L1/TEX Hit Rate (%)	97.91	84.62	98.97	66.86
L2 Hit Rate (%)	3.30	6.18	10.80	17.71
Primary Bottleneck	Memory-bound	DRAM-latency	SM bandwidth	Compute-bound