CUDA getRestricted Kernel Execution Time Prediction Report

This report applies the CUDA kernel performance prediction template to the kernel:
getRestricted, launched with Grid = (5,5) and Block = (32,32).

We estimate the expected per-launch execution time on five NVIDIA GPUs:
- GeForce GTX TITAN Black
- GeForce GTX TITAN X
- NVIDIA TITAN V
- GeForce RTX 2080 Ti
- GeForce RTX 4070

```
__global__ void getRestricted(int countx, int county, int rows, int cols,
                              float * mX, int mXdim, float * vY, int vYdim,
                              float * mQ, int mQdim, float * mR, int mRdim,
                              float * vectB, int vectBdim) {

    int m = blockIdx.x * THREADSPERDIM + threadIdx.x, n, i, j, k;
    float sum, invnorm, *X, *Y, *Q, *R, *B, *coli, *colj, *colQ, *colX;

    if(m >= county) return;
    if(m == 1) n = 0;
    else n = 1;

    X = mX + (m * mXdim);
    ...
}
```

1. Effective Parallelism

Launch configuration:
- Grid = (5,5) → 25 blocks
- Block = (32,32) → 1024 threads per block
- THREADSPERDIM = 16
- countx = 5, county = 5
- rows = 100, cols = 10

The kernel uses:
m = blockIdx.x * THREADSPERDIM + threadIdx.x;

With gridDim.x = 5 and THREADSPERDIM = 16, m ranges from 0 to 95.
Only m < county (=5) do any work, so only blockIdx.x = 0 and threadIdx.x = 0..4 are active. For each of those 5 threads, all 32 threadIdx.y rows duplicate the same computation.

Total active threads performing work:
5 (m values) × 32 (threadIdx.y rows) = 160 threads.

Each active thread performs a full Gram–Schmidt QR factorization and backsubstitution over a rows×cols = 100×10 matrix, along with vector operations.

2. FLOP Count (Per Active Thread)

Per logical thread (fixed m):
• Copy X → Q: no FLOPs (just loads/stores).
• Gram–Schmidt orthogonalization:
- Inner products and projections over all i - Column norm and normalization over 10 columns → ~3,100 FLOPs.
→ Gram–Schmidt ≈ 21,100 FLOPs.

• R = Q■X and B = Q■Y, plus back-substitution:
- Matrix multiply Q■X: ≈ 20,000 FLOPs.
- Vector multiply Q■Y and back-substitution: ≈ 2,100 FLOPs.
→ R/B stage ≈ 22,100 FLOPs.

Total FLOPs per active thread:
F_thread ≈ 21,100 + 22,100 ≈ 43,200 FLOPs.

With 160 active threads doing identical work:
F_total = 43,200 × 160 ≈ 6.91 × 10■ FLOPs.

3. Memory Traffic (Approximate)

Per active thread we have:
• X initialization and X→Q copy (~1,000 elements): ≈ 8.4 KB
• Gram–Schmidt loads/stores of Q columns: ≈ 102 KB
• R/B computation involving Q, X, Y, R, B: ≈ 88 KB

Total per thread: B_thread ≈ 1.99 × 10■ bytes.

With 160 active threads:
B_total ≈ 1.99 × 10■ × 160 ≈ 3.18 × 10■ bytes ≈ 31.8 MB per kernel launch.

This kernel is therefore clearly **memory bound**.

4. GPU Specifications Used

Approximate FP32 peak performance and memory bandwidth:

GPU	Peak FP32 (FLOPs/s)	Bandwidth (bytes/s)
GTX TITAN Black	5.12e12	3.36e11
GTX TITAN X	6.14e12	3.365e11
TITAN V	1.49e13	6.528e11
RTX 2080 Ti	1.345e13	6.16e11
RTX 4070	2.9e13	5.04e11

5. Time Estimates

Using:
• F_total ≈ 6.91 × 10■ FLOPs
• B_total ≈ 3.18 × 10■ bytes

We compute for each GPU:
t_compute = F_total / Peak_FP32
t_mem = B_total / Bandwidth
t_body = max(t_compute, t_mem)
t_total ≈ t_body + 5 µs (kernel launch overhead).

GPU	t_compute (µs)	t_mem (µs)	t_body (µs)	t_total (µs)
GTX TITAN Black	1.35	94.71	94.71	≈ 99.71
GTX TITAN X	1.13	94.57	94.57	≈ 99.57
TITAN V	0.46	48.75	48.75	≈ 53.75
RTX 2080 Ti	0.51	51.66	51.66	≈ 56.66
RTX 4070	0.24	63.14	63.14	≈ 68.14

6. Conclusion

The getRestricted kernel performs roughly 6.9M FLOPs but moves about 32MB of data per launch, with only 160 active threads doing all the work while many others return immediately.

The heavy global-memory traffic makes the kernel strongly memory bound. On all five GPUs, t_mem dominates t_compute, and the additional 5 µs kernel launch overhead is relatively small compared to the 50–100 µs body time.

Predicted per-launch times:

- GTX TITAN Black / TITAN X: ≈ 100 µs
- TITAN V: ≈ 54 µs
- RTX 2080 Ti: ≈ 57 µs
- RTX 4070: ≈ 68 µs

These values are approximate but consistent with expectations for a memory-heavy QR-based solver with duplicated work per thread.