Conjugate Gradient in CUDA

Ludek Cizinsky (student ID: 377297)
ludek.cizinsky@epfl.ch

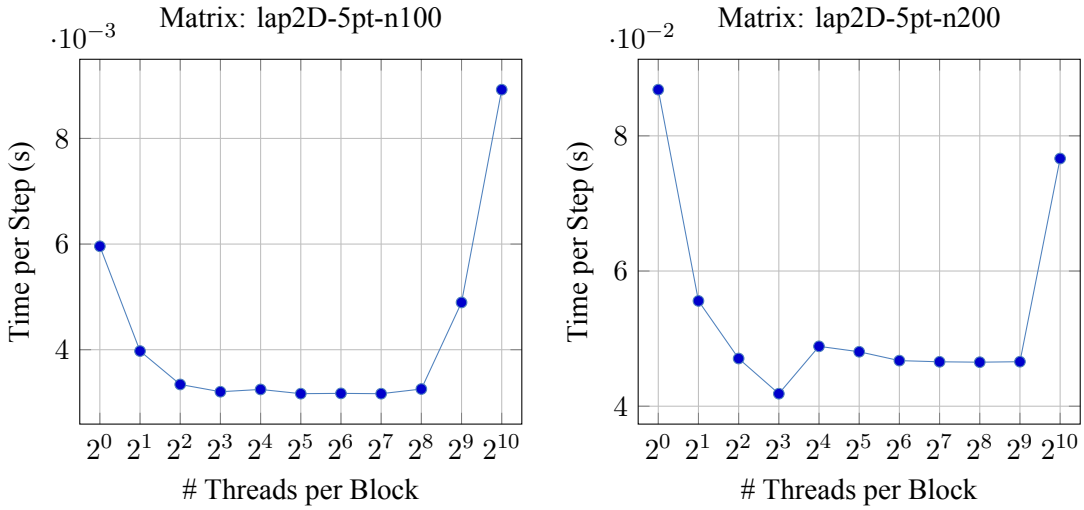

Figure 1. CI95 of time per CG iteration step for different number of threads.

Computational and Communication Costs. Each Conjugate Gradient (CG) iteration consists of a dense matrix-vector multiplication and a few vector operations (dot products and AXPY updates). In the mat-vec step on a dense $n \times n$ matrix, GPU needs to perform $2n^2$ FLOPs while moving roughly $16n^2$ bytes of global memory, giving an arithmetic intensity AI $= \frac{2n^2}{16n^2} = 0.125$ FLOP/byte. On a V100 GPU (7 TFLOP/s, 900 GB/s [1]), the break-even AI is ≈ 7.8 FLOP/byte, hence the CG solver is memory-bound. Over k iterations both compute work and memory load scale as $O(kn^2)$.

Experimental Setup. To account for problem size, I ran the CG solver on two Laplace-type problems with $n \approx 10{,}000$ and $40{,}000$ on an NVIDIA V100 (80 SMs, 128 KiB shared/L1, 65 536 registers/SM [1]). During experiments, I varied CUDA threads-per-block from 1 to 1024 in powers of two, each configuration repeated 5 times to compute mean time/step and 95% confidence intervals (Fig. 1).

Results and Interpretation. Figure 1 exhibits a pronounced U-shape for both problem sizes. With only 1-2 threads/block, although all 80 SMs can be active, each SM hosts at most 1-2 active warps. Once a warp issues a global-memory load it stalls for hundreds of cycles and there are no other ready warps to hide the latency, so the SM sits idle, memory bandwidth is under-utilized, and per-step time is high. In the range of 8 to 256 threads per block, the GPU achieves sufficient warp occupancy, which means that enough threads are active to effectively hide the delays caused by memory latency. Under these block sizes, the memory accesses by the threads are efficiently coalesced, meaning that the memory requests are combined into fewer transactions, optimizing the use of memory bandwidth. Therefore, as you increase the number of threads, the time taken per operation decreases proportionally, up to a point where other factors may start to limit performance. For $N = 10\,000$, performance begins to degrade already at 512 threads per block because only 5 SMs remain active, yielding just $5 \times 64 = 320$ warps to issue memory loads-far too few to saturate HBM2. In contrast, for $N = 40\,000$ at 512 threads per block there are still 20 SMs active, providing $20 \times 64 = 1280$ warps to keep the memory pipelines full. Therefore, the memory-starvation inflection point shifts to 1024 threads per block. See the Appendix for the detailed occupancy breakdowns.

Conclusion. The CG solver on the V100 is memory-bound, with optimal performance at 8-64 threads per block due to efficient warp occupancy and memory coalescing. Performance degrades at low thread counts due to underutilized SMs and at high counts due to memory starvation. Next step would be to consider using warp-level primitives to share data via registers, avoiding shared memory and sync barriers.

References

[1] Nvidia tesla v100 gpu architecture. Technical Report WP-08608-001_v1.1, NVIDIA Corporation, 2017. Accessed: 2025-05-07.

Appendix

Occupancy breakdown. To better understand the performance of the CG solver across different block sizes, I computed the occupancy breakdown for both problem sizes. When computing the values, I took into consideration the V100 architecture, specifically its total number of SMs (80), maximum number of blocks per SM (32), and maximum number of warps per SM (64).

Table 1. Occupancy breakdown for $N = 10\,000$ on NVIDIA V100

Threads/block	Blocks total	Blocks/SM	Warps/SM	Active SMs
1	10000	32	1	80
2	5000	32	2	80
4	2500	32	4	79
8	1250	32	8	40
16	625	32	16	20
32	313	32	32	10
64	157	32	64	5
128	79	16	64	5
256	40	8	64	5
512	20	4	64	5
1024	10	2	64	5

Table 2. Occupancy breakdown for $N = 40\,000$ on an NVIDIA V100

Threads/block	Blocks total	Blocks/SM	Warps/SM	Active SMs
1	40000	32	1	80
2	20000	32	2	80
4	10000	32	4	80
8	5000	32	8	80
16	2500	32	16	79
32	1250	32	32	40
64	625	32	64	20
128	313	16	64	20
256	157	8	64	20
512	79	4	64	20
1024	40	2	64	20