	Result	Time	Cycles	GPU	SM Frequency	Process	Attributes
Current	604 - get_hamiltonian_between_atoms_kernel (1404, 939, 3)x(32, 1, 1)	1.09 second	1,680,095,674	0 - NVIDIA GeForce RTX 4060 Laptop GPU	1.54 cycle/nsecond	[109798] tester	
Baseline 2	1169 - get_hamiltonian_between_atoms_kernel (1404, 939, 3)x(36, 1, 1)	1.53 second	2,365,633,101	0 - NVIDIA GeForce RTX 4060 Laptop GPU	1.54 cycle/nsecond	[103857] tester	

GPU Speed Of Light Throughput

GPU Throughput Chart

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	21.55 (+23.59%)	Duration [second]	1.09 (-28.96%)
Memory Throughput [%]	13.84 (+15.47%)	Elapsed Cycles [cycle]	1,680,095,674 (-28.98%)
L1/TEX Cache Throughput [%]	15.30 (+14.86%)	SM Active Cycles [cycle]	1,675,960,623.25 (-29.05%)
L2 Cache Throughput [%]	13.84 (+15.47%)	SM Frequency [cycle/nsecond]	1.54 (-0.04%)
DRAM Throughput [%]	2.63 (+66.91%)	DRAM Frequency [cycle/nsecond]	7.67 (-4.07%)

Latency Issue This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of this device. Achieved compute throughput and/or memory bandwidth below 60.0% of peak typically indicate latency issues. Look at Scheduler Statistics and Warp State Statistics for potential reasons.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 64:1. The kernel achieved close to 0% of this device's fp32 peak performance and 2% of its fp64 peak performance. See the Kernel Profiling Guide for more details on roofline analysis.

PM Sampling

Timeline view of PM metrics sampled periodically over the workload duration. Data is collected across multiple passes. Use this section to understand how workload behavior changes over its runtime.

Maximum Sampling Interval [msecond]	1.02	# Pass Groups	2
Maximum Buffer Size [Mbytes]	3.31	Dropped Samples [sample]	0

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	0.51	SM Busy [%]	21.57
Executed Ipc Active [inst/cycle]	0.51	Issue Slots Busy [%]	12.68
Issued Ipc Active [inst/cycle]	0.51		

Balanced FP64 is the highest-utilized pipeline (21.6%) based on active cycles, taking into account the rates of its different instructions. It executes 64-bit floating point operations. It is well-utilized, but should not be a bottleneck.

Memory Workload Analysis

Memory Chart

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	6.45 (+60.11%)	Mem Busy [%]	9.73
L1/TEX Hit Rate [%]	34.41	Max Bandwidth [%]	13.84
L2 Hit Rate [%]	99.62	Mem Pipes Busy [%]	13.77
L2 Compression Success Rate [%]	0	L2 Compression Ratio	0

L1TEX Local Load Access Pattern
Est. Speedup: 13.91%
The memory access pattern for local loads from L1TEX might not be optimal. On average, only 2.9 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced local loads.

L1TEX Local Store Access Pattern
Est. Speedup: 13.75%
The memory access pattern for local stores to L1TEX might not be optimal. On average, only 3.2 of the 32 bytes transmitted per sector are utilized by each thread. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced local stores.

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	3.13	No Eligible [%]	87.12
Eligible Warps Per Scheduler [warp]	0.13	One or More Eligible [%]	12.88
Issued Warp Per Scheduler	0.13		

Issue Slot Utilization
Est. Local Speedup: 78.45%
Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 7.8 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 12 warps per scheduler, this kernel allocates an average of 3.13 active warps per scheduler, but only an average of 0.13 warps were eligible per cycle. Eligible warps are the subset of active warps that are ready to issue their next instruction. Every cycle with no eligible warp results in no instruction being issued and the issue slot remains unused. To increase the number of eligible warps, reduce the time the active warps are stalled by inspecting the top stall reasons on the Warp State Statistics and Source Counters sections.

Warp State Statistics

Analysis of the states in which all warps spent cycles during the kernel execution. The warp states describe a warp's readiness or inability to issue its next instruction. The warp cycles per instruction define the latency between two consecutive instructions. The higher the value, the more warp parallelism is required to hide this latency. For each warp state, the chart shows the average number of cycles spent in that state per issued instruction. Stalls are not always impacting the overall performance nor are they completely avoidable. Only focus on stall reasons if the schedulers fail to issue every cycle. When executing a kernel with mixed library and user code, these metrics show the combined values.

Warp Cycles Per Issued Instruction [cycle]	24.33	Avg. Active Threads Per Warp	11.85
Warp Cycles Per Executed Instruction [cycle]	24.37	Avg. Not Predicated Off Threads Per Warp	11.78

Wait Stalls
Est. Speedup: 56.31%
On average, each warp of this kernel spends 13.7 cycles being stalled waiting on a fixed latency execution dependency. Typically, this stall reason should be very low and only shows up as a top contributor in already highly optimized kernels. Try to hide the corresponding instruction latencies by increasing the number of active warps, restructuring the code or unrolling loops. Furthermore, consider switching to lower-latency instructions, e.g. by making use of fast math compiler options. This stall type represents about 56.3% of the total average of 24.3 cycles between issuing two instructions.

Warp Stall Check the Warp Stall Sampling (All Samples) table for the top stall locations in your source based on sampling data. The Kernel Profiling Guide provides more details on each stall reason.

Thread Divergence
Est. Speedup: 13.62%
Instructions are executed in warps, which are groups of 32 threads. Optimal instruction throughput is achieved if all 32 threads of a warp execute the same instruction. The chosen launch configuration, early thread completion, and divergent flow control can significantly lower the number of active threads in a warp per cycle. This kernel achieves an average of 11.8 threads being active per cycle. This is further reduced to 11.8 threads per warp due to predication. The compiler may use predication to avoid an actual branch. Instead, all instructions are scheduled, but a per-thread condition code or predicate controls which threads execute the instructions. Try to avoid different execution paths within a warp when possible.

Instruction Statistics

Statistics of the executed low-level assembly instructions (SASS). The instruction mix provides insight into the types and frequency of the executed instructions. A narrow mix of instruction types implies a dependency on few instruction pipelines, while others remain unused. Using multiple pipelines allows hiding latencies and enables parallel execution. Note that 'Instructions/Opcode' and 'Executed Instructions' are measured differently and can diverge if cycles are spent in system calls.

Executed Instructions [inst]	20,377,641,382	Avg. Executed Instructions Per Scheduler [inst]	212,267,097.73
Issued Instructions [inst]	20,408,570,104	Avg. Issued Instructions Per Scheduler [inst]	212,589,271.92

FP32 Non-Fused Instructions
Est. Speedup: 0.16%
This kernel executes 1629564 fused and 44196240 non-fused FP32 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP32 performance could be increased by up to 48% (relative to its current performance). Check the Source page to identify where this kernel executes FP32 instructions.

FP64 Non-Fused Instructions
Est. Speedup: 8.73%
This kernel executes 87514736 fused and 371183657 non-fused FP64 instructions. By converting pairs of non-fused instructions to their fused, higher-throughput equivalent, the achieved FP64 performance could be increased by up to 40% (relative to its current performance). Check the Source page to identify where this kernel executes FP64 instructions.

NVLink Topology

NVLink Topology diagram shows logical NVLink connections with transmit/receive throughput.

NVLink Tables

Detailed tables with properties for each NVLink.

NUMA Affinity

Non-uniform memory access (NUMA) affinities based on compute and memory distances for all GPUs.

Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	3,955,068 (+0.00%)	Function Cache Configuration	CachePreferNone (CachePreferNone)
Registers Per Thread [register/thread]	112 (+0.00%)	Static Shared Memory Per Block [Kbyte/block]	5.30 (+0.00%)
Block Size	32 (-11.11%)	Dynamic Shared Memory Per Block [byte/block]	0 (+0.00%)
Threads [thread]	126,562,176 (-11.11%)	Driver Shared Memory Per Block [Kbyte/block]	1.02 (+0.00%)
Waves Per SM	10,299.66 (-50.00%)	Shared Memory Configuration Size [Kbyte]	102.40 (+56.25%)
Uses Green Context	0 (+0.00%)	# SMs [SM]	24 (+0.00%)

Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	33.33 (+0.00%)	Block Limit Registers [block]	16 (+100.00%)
Theoretical Active Warps per SM [warp]	16 (+0.00%)	Block Limit Shared Mem [block]	16 (+60.00%)
Achieved Occupancy [%]	25.72	Block Limit Warps [block]	48 (+100.00%)
Achieved Active Warps Per SM [warp]	12.35	Block Limit SM [block]	24 (+0.00%)

Theoretical Occupancy
Est. Speedup: 66.67%
The 4.00 theoretical warps per scheduler this kernel can issue according to its occupancy are below the hardware maximum of 12. This kernel's theoretical occupancy (33.3%) is limited by the number of required registers. This kernel's theoretical occupancy (33.3%) is limited by the required amount of shared memory.

GPU and Memory Workload Distribution

Analysis of workload distribution in active cycles of SM, SMP, SMSP, L1 & L2 caches, and DRAM

Average SM Active Cycles [cycle]	1,675,960,623.25 (-29.05%)	Average L1 Active Cycles [cycle]	1,675,960,623.25 (-29.05%)
Average L2 Active Cycles [cycle]	1,276,671,395.56 (-22.94%)	Average SMSP Active Cycles [cycle]	1,651,123,291.64 (-29.55%)
Average DRAM Active Cycles [cycle]	219,183,976 (+13.74%)	Total SM Elapsed Cycles [cycle]	40,258,170,544 (-29.01%)
Total L1 Elapsed Cycles [cycle]	40,258,170,544 (-29.01%)	Total L2 Elapsed Cycles [cycle]	25,119,132,576 (-29.30%)
Total SMSP Elapsed Cycles [cycle]	161,032,682,176 (-29.01%)	Total DRAM Elapsed Cycles [cycle]	33,360,699,392 (-31.85%)

Source Counters

Source metrics, including branch efficiency and sampled warp stall reasons. Warp Stall Sampling metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the documentation for a description of all stall reasons. Only focus on stalls if the schedulers fail to issue every cycle.

Branch Instructions [inst]	1,387,039,660	Branch Efficiency [%]	91.45
Branch Instructions Ratio [%]	0.07	Avg. Divergent Branches	1,000,505.94

Uncoalesced Global Accesses
Est. Speedup: 27.19%
This kernel has uncoalesced global accesses resulting in a total of 56497222 excessive sectors (33% of the total 168981154 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide has additional information on reducing uncoalesced device memory accesses.

L2 Theoretical Sectors Global Excessive

Location	Value	Value (%)
main.cu:1365 (0x709c7925efa0 in get_hamiltonian_between_atoms_kernel)	8,554,992	15
main.cu:1365 (0x709c7925ef40 in get_hamiltonian_between_atoms_kernel)	8,554,992	15
main.cu:1341 (0x709c7925e160 in get_hamiltonian_between_atoms_kernel)	8,554,992	15
main.cu:1341 (0x709c7925e100 in get_hamiltonian_between_atoms_kernel)	8,554,992	15
main.cu:1334 (0x709c7925dd20 in get_hamiltonian_between_atoms_kernel)	3,636,844	6

Uncoalesced Shared Accesses
Est. Speedup: 25.75%
This kernel has uncoalesced shared accesses resulting in a total of 150407967 excessive wavefronts (26% of the total 583572814 wavefronts). Check the L1 Wavefronts Shared Excessive table for the primary source locations. The CUDA Best Practices Guide has an example on optimizing shared memory accesses.

L1 Wavefronts Shared Excessive

Location	Value	Value (%)
0x709c7929f9b0 in __dAtomicAdd	84,261,660	56
0x709c7929fa70 in __dAtomicAdd	60,600,556	40
main.cu:1341 (0x709c7925dfa0 in get_hamiltonian_between_atoms_kernel)	851,199	1
main.cu:383 (0x709c79286a20 in transform1)	560,248	0
main.cu:383 (0x709c79277b20 in transform1)	420,186	0