Corpus Size / Model	Small (1.1 MB)	Medium (3.5 MB)	Large (11.5MB)	Very Large (35.5 MB)	Huge (120 MB)
Serial CPU	0.83	2.47	8.64	26.05	86.53
GPU + OpenMP	0.59	1.90	6.22	19.48	62.22
GPU + stream + OpenMP	0.59	1.90	6.48	19.15	62.40
GPU + stream + OpenMP	0.59	1.90	6.48	19.15	62.40

*10 Execution

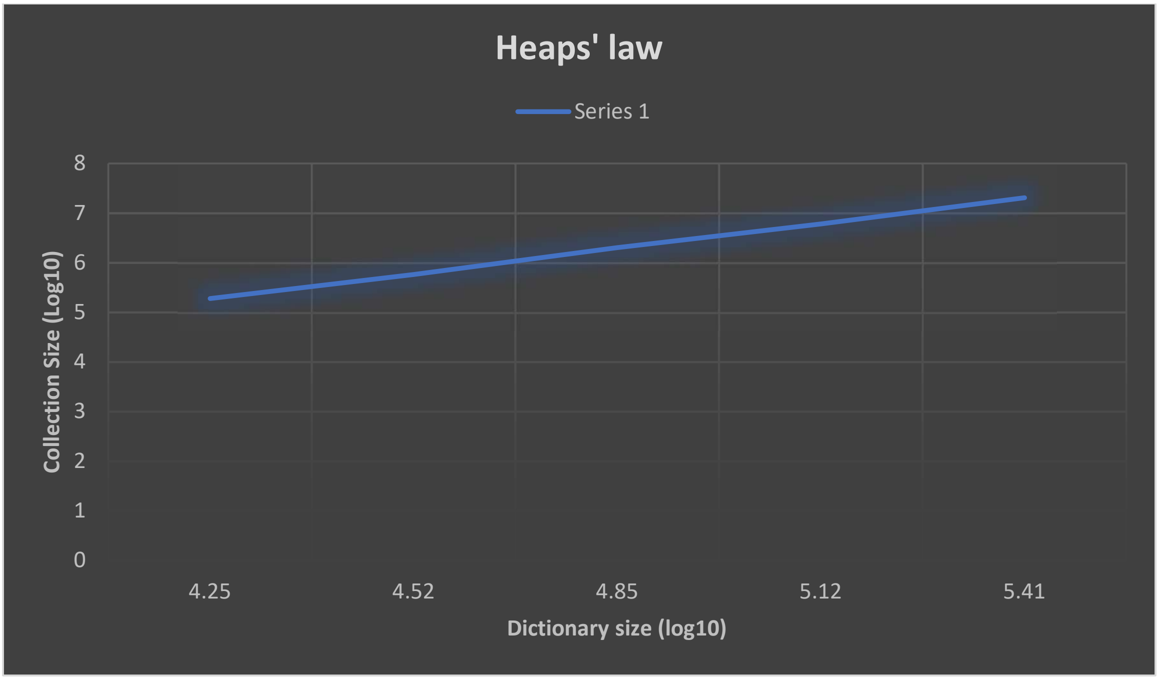
برای stemming از یک نوعالگوریتم snowball استفاده شده است و tokenization به کمک کتابخانه ی boost پیاده سازی شده است. بعلاوه قبل از پردازش بروی متن با استفاده های script ساده که داخل فایل zip هستند، حروف noise را حذف کرده. برای stopword ها از یک لیست معتبر که در repository sentencepiece قرار داشت استفاده کردم. امکان موازی سازی عملیات ها در gpu مخصوصا stemming و tokenization وجود ندارد زیرا divergence بسیار زیاد است و با یک نگاه سطحی میتوان فهمید که شدنی نیست. تنها حالت ممکن استفاده از مدل های subword tokenization است که تمرین دادن آنها زمان زیادی میبرد و در این مدت زمان ممکن نبود. اگرچه حتی خود این مدل ها هم کاملا مناسب نیستند (نمیتوان تمام بخش هارا موازی سازی کرد) برای بخش کار با openmp چالش هایی وجود دارد به طور مثال نمیتوان پیش پردازش را به تسک های ریز تقسیم کرد، چرا که از یک دیگر مستقل نیستند و باید به ترتیب اجرا شوند. تنها بخش موازی شده استفاده از نخ openmp برای اجرای kernel است به گونه ای که به یک نخ به عنوان وظیفه راه اندازی و اجرای کرنل cuda سپرده شده است. بیشتر پردازش در این قسمت برنامه بروی cpu است که از تصویر nsight systems هم معلوم است.

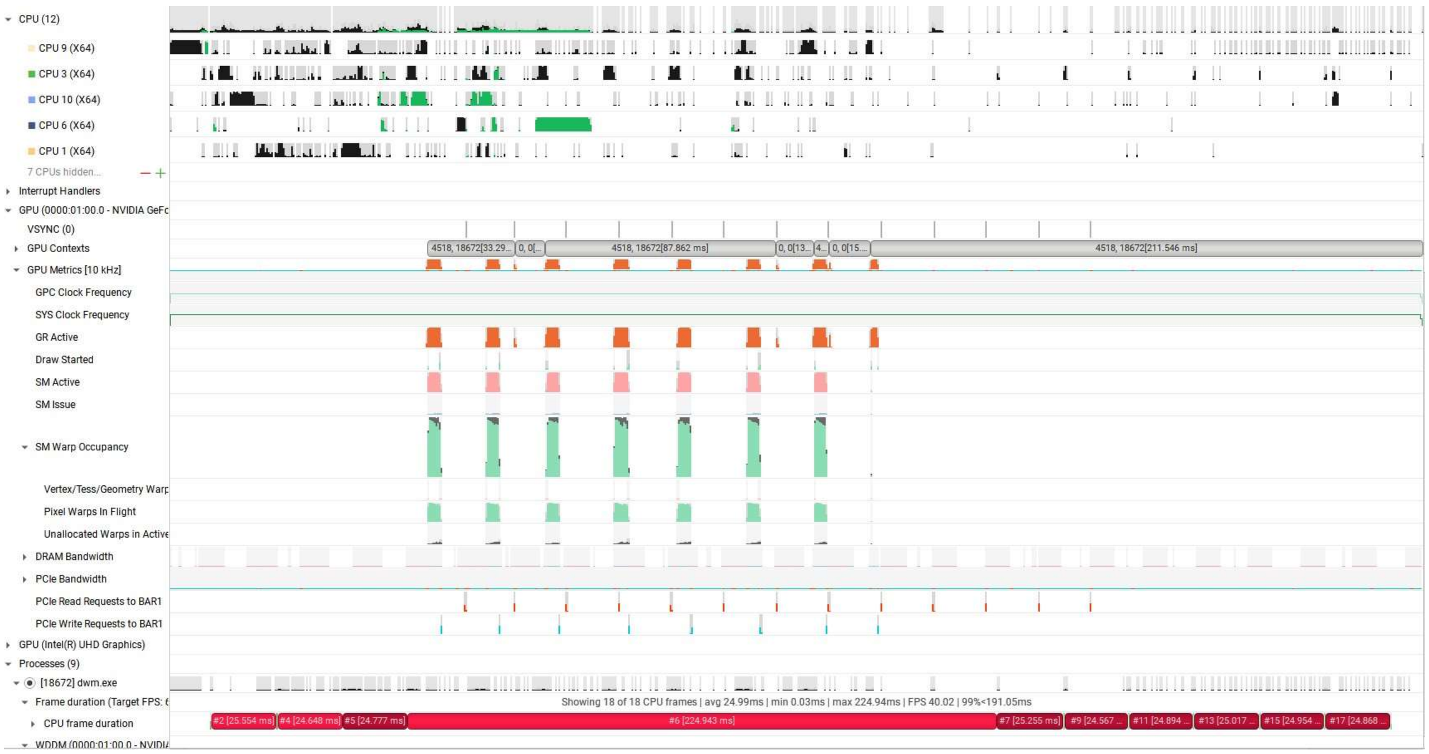
صلاحیت اسناد: یک معیار برای اندازه گیری کیفیت مجموعه اسناد اندازه گیری نسبت اندازه دیکشنری به تعداد کل کلمات است. با توجه به نمودار ترسیم شده که با قانون heaps همخوانی دارد میتوان گفت که اسناد به لحاظ تنوع کلمات مناسب هستند و حالت خاص نیستند.

چالش:

1. مسعله چه خواسته است
2. تحقیق اینکه چه چیز را میتوان موازی کرد و چه چیزی را نمیشه
3. انتظار چه speedup را باید داشته باشم
4. ترکیب تکنولوژی های مختلف مانند cuda, openmp چالش های زیادی داشت. Cmake را نمیشناختم. کامپایل و لینک کردن را بلد نبودم و داکیومنت خوب به اندازه ی کافی نبود.
5. انتظار اشتباه از مدل های مختلف داشتم به طور مثال فکر میکردم که sentencepiece یک tokenizer کامل است که اینطور نبود و زمان زیادی صرف راه اندازی آن هدر رفت.

SMALL: Most Frequent Word: (said:1590)	Dictionary Size: 17783	Total Counts: 192444
MED: Most Frequent Word: (said:4962)	Dictionary Size: 33401	Total Counts: 575106
LARGE: Most Frequent Word: (said:16521)	Dictionary Size: 70296	Total Counts: 2019437
VLAR: Most Frequent Word: (said:49456)	Dictionary Size: 132116	Total Counts: 6070588
HUGE: Most Frequent Word: (said:164635)	Dictionary Size: 260695	Total Counts: 20236831

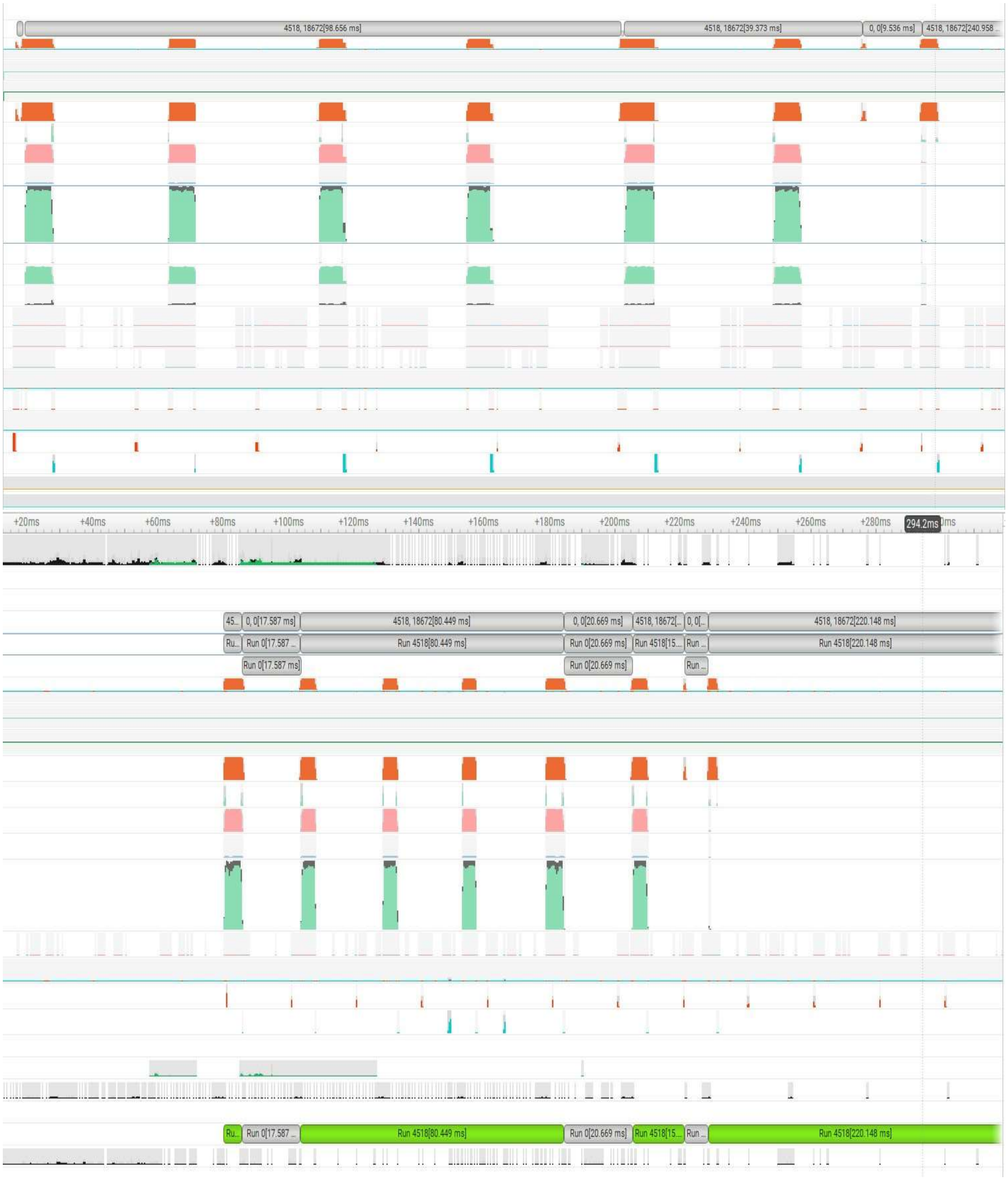


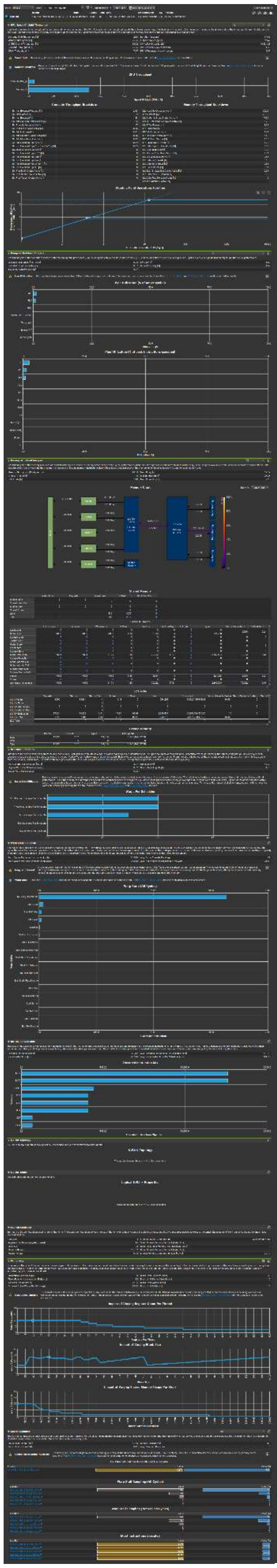


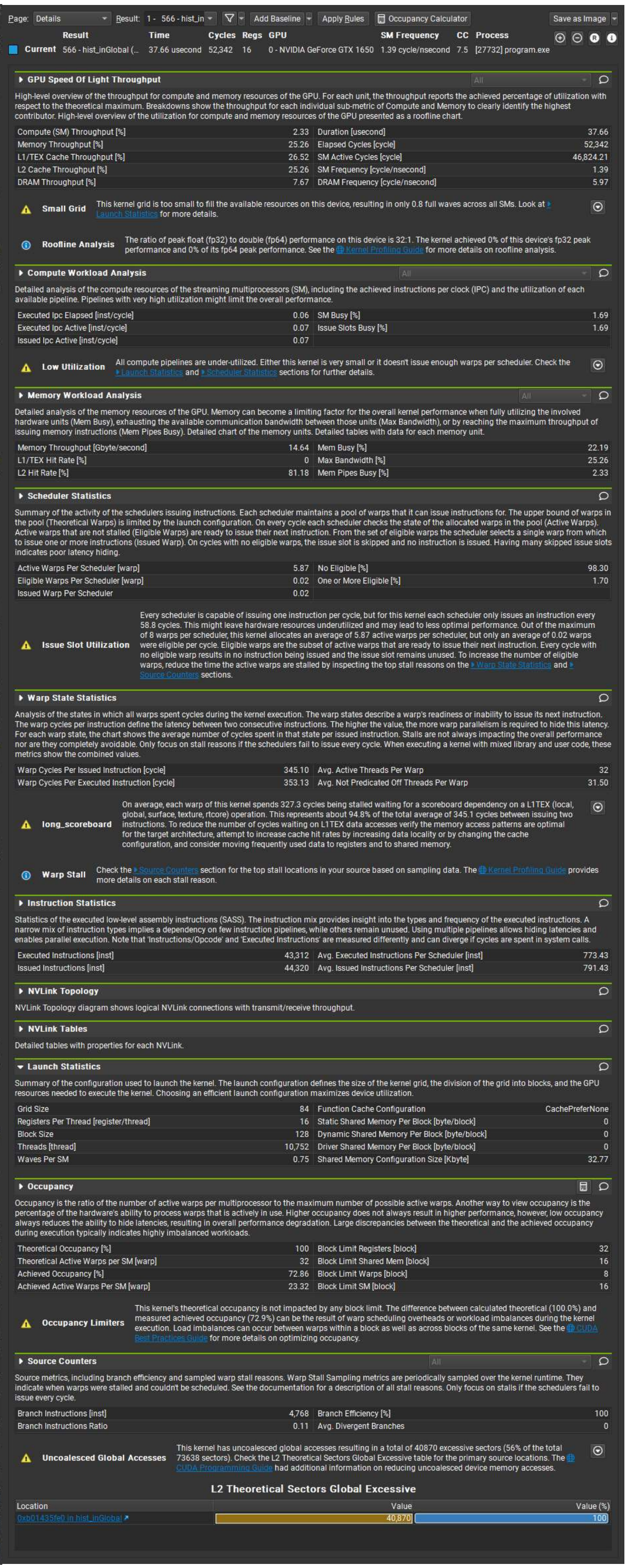
Showing 18 of 18 CPU frames | avg 24.99ms | min 0.03ms | max 224.94ms | FPS 40.02 | 99%<191.05ms

قسمت زیرین تنها مربوط به اجرای الگوریتم histogram است.Speedup ها به نسبت حالت سریال محاسبه شده اند. در کل به دلیل اینکه حجم پردازش بسیار نسبت به دسترسی و کار با حافظه کمتر است گرفتن speedup خیلی بالا در حالت استفاده از streamها زیاد نیست. در پایین هم در عکس تحلیل nsight systems قابل مشاهده است که حتی با اینکه کرنل ها با حافظه موازی شده اند همچنان مقدار زیادی از زمان gpu محاسبه انجام نمیدهد. مخصوصا اینکه خود استفاده از عملیات اتمی هزینه بالایی دارد و قابل پیاده سازی در shared memory برای تعداد bin های بالا (تعداد کلمات دیکشنری) را ندارد. البته میتوان از روش data output decomposition استفاده کرد که لازمه ی آن حرکت هر نخ بروی کل آرایه ها است که طبیعتا قرار نیست speedup بدهد.

تصویر اولی برای حالت بدون shared memory و دومی برای حالت با shared memory است. Record.txt شامل اجرا های مختلف الگوریتم histogram است.


4518, 18672[98.656 ms]
4518, 18672[39.373 ms]
0, 0[9.536 ms]
4518, 18672[240.958 ...
+20ms
+40ms
+60ms
+80ms
+100ms
+120ms
+140ms
+160ms
+180ms
+200ms
+220ms
+240ms
+260ms
+280ms
294.2ms
ms
45...
0, 0[17.587 ms]
4518, 18672[80.449 ms]
0, 0[20.669 ms]
4518, 18672[...
0, 0[...
4518, 18672[220.148 ms]
Ru...
Run 0[17.587 ...
Run 4518[80.449 ms]
Run 0[20.669 ms]
Run 4518[15...
Run ...
Run 4518[220.148 ms]
Run 0[17.587 ms]
Run 0[20.669 ms]
Run ...
Ru...
Run 0[17.587 ...
Run 4518[80.449 ms]
Run 0[20.669 ms]
Run 4518[15...
Run ...
Run 4518[220.148 ms]





Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current 566 - hist_inGlobal (...	37.66 usecond	52,342	16	0 - NVIDIA GeForce GTX 1650	1.39 cycle/nsecond	7.5	[27732] program.exe

GPU Speed Of Light Throughput

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 [%]	2.33	Duration [usecond]	37.66
Memory Throughput [%]	25.26	Elapsed Cycles [cycle]	52,342
L1/TEX Cache Throughput [%]	26.52	SM Active Cycles [cycle]	46,824.21
L2 Cache Throughput [%]	25.26	SM Frequency [cycle/nsecond]	1.39
DRAM Throughput [%]	7.67	DRAM Frequency [cycle/nsecond]	5.97

Small Grid This kernel grid is too small to fill the available resources on this device, resulting in only 0.8 full waves across all SMs. Look at Launch Statistics for more details.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 32:1. The kernel achieved 0% of this device's fp32 peak performance and 0% of its fp64 peak performance. See the Kernel Profiling Guide for more details on roofline analysis.

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]	0.06	SM Busy [%]	1.69
Executed Ipc Active [inst/cycle]	0.07	Issue Slots Busy [%]	1.69
Issued Ipc Active [inst/cycle]	0.07		

Low Utilization All compute pipelines are under-utilized. Either this kernel is very small or it doesn't issue enough warps per scheduler. Check the Launch Statistics and Scheduler Statistics sections for further details.

Memory Workload Analysis

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]	14.64	Mem Busy [%]	22.19
L1/TEX Hit Rate [%]	0	Max Bandwidth [%]	25.26
L2 Hit Rate [%]	81.18	Mem Pipes Busy [%]	2.33

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]	5.87	No Eligible [%]	98.30
Eligible Warps Per Scheduler [warp]	0.02	One or More Eligible [%]	1.70
Issued Warp Per Scheduler	0.02		

Issue Slot Utilization Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 58.8 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 8 warps per scheduler, this kernel allocates an average of 5.87 active warps per scheduler, but only an average of 0.02 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]	345.10	Avg. Active Threads Per Warp	32
Warp Cycles Per Executed Instruction [cycle]	353.13	Avg. Not Predicated Off Threads Per Warp	31.50

long_scoreboard On average, each warp of this kernel spends 327.3 cycles being stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, texture, rtcore) operation. This represents about 94.8% of the total average of 345.1 cycles between issuing two instructions. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture, attempt to increase cache hit rates by increasing data locality or by changing the cache configuration, and consider moving frequently used data to registers and to shared memory.

Warp Stall Check the Source Counters section for the top stall locations in your source based on sampling data. The Kernel Profiling Guide provides more details on each stall reason.

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]	43,312	Avg. Executed Instructions Per Scheduler [inst]	773.43
Issued Instructions [inst]	44,320	Avg. Issued Instructions Per Scheduler [inst]	791.43

NVLink Topology

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

NVLink Tables

Detailed tables with properties for each NVLink.

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	84	Function Cache Configuration	CachePreferNone
Registers Per Thread [register/thread]	16	Static Shared Memory Per Block [byte/block]	0
Block Size	128	Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	10,752	Driver Shared Memory Per Block [byte/block]	0
Waves Per SM	0.75	Shared Memory Configuration Size [Kbyte]	32.77

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 [%]	100	Block Limit Registers [block]	32
Theoretical Active Warps per SM [warp]	32	Block Limit Shared Mem [block]	16
Achieved Occupancy [%]	72.86	Block Limit Warps [block]	8
Achieved Active Warps Per SM [warp]	23.32	Block Limit SM [block]	16

Occupancy Limiters This kernel's theoretical occupancy is not impacted by any block limit. The difference between calculated theoretical (100.0%) and measured achieved occupancy (72.9%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

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]	4,768	Branch Efficiency [%]	100
Branch Instructions Ratio	0.11	Avg. Divergent Branches	0

Uncoalesced Global Accesses This kernel has uncoalesced global accesses resulting in a total of 40870 excessive sectors (56% of the total 73638 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide had additional information on reducing uncoalesced device memory accesses.

L2 Theoretical Sectors Global Excessive

Location	Value	Value (%)
0xb01435fe0 in hist_inGlobal	40,870	100