

# Analysis Report

evolvi(configurazione, curandStateXORWOW\*)

Duration	2.562 ms (2,561,783 ns)
Grid Size	[ 128,1,1 ]
Block Size	[ 1024,1,1 ]
Registers/Thread	32
Shared Memory/Block	0 B
Shared Memory Requested	64 KiB
Shared Memory Executed	64 KiB
Shared Memory Bank Size	4 B

## [0] GeForce 840M

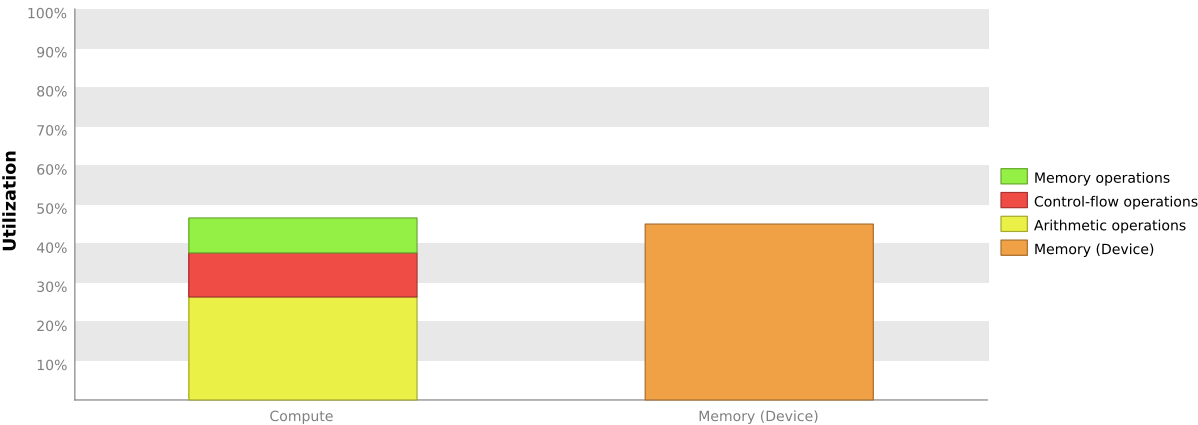
GPU UUID	GPU-4a39874c-8303-b7e3-9758-a265118f0297
Compute Capability	5.0
Max. Threads per Block	1024
Max. Shared Memory per Block	48 KiB
Max. Registers per Block	65536
Max. Grid Dimensions	[ 2147483647, 65535, 65535 ]
Max. Block Dimensions	[ 1024, 1024, 64 ]
Max. Warps per Multiprocessor	64
Max. Blocks per Multiprocessor	32
Single Precision FLOP/s	863.232 GigaFLOP/s
Double Precision FLOP/s	26.976 GigaFLOP/s
Number of Multiprocessors	3
Multiprocessor Clock Rate	1.124 GHz
Concurrent Kernel	true
Max IPC	6
Threads per Warp	32
Global Memory Bandwidth	14.4 GB/s
Global Memory Size	2 GiB
Constant Memory Size	64 KiB
L2 Cache Size	1 MiB
Memcpy Engines	1
PCIe Generation	2
PCIe Link Rate	5 Gbit/s
PCIe Link Width	4

# 1. Compute, Bandwidth, or Latency Bound

The first step in analyzing an individual kernel is to determine if the performance of the kernel is bounded by computation, memory bandwidth, or instruction/memory latency. The results below indicate that the performance of kernel "evolvi" is most likely limited by instruction and memory latency. You should first examine the information in the "Instruction And Memory Latency" section to determine how it is limiting performance.

## 1.1. Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "GeForce 840M". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.



## 2. Instruction and Memory Latency

Instruction and memory latency limit the performance of a kernel when the GPU does not have enough work to keep busy. The results below indicate that the GPU does not have enough work because instruction execution is stalling excessively.

### 2.1. Instruction Latencies May Be Limiting Performance

Instruction stall reasons indicate the condition that prevents warps from executing on any given cycle. The following chart shows the break-down of stalls reasons averaged over the entire execution of the kernel. The kernel has good theoretical and achieved occupancy indicating that there are likely sufficient warps executing on each SM. Since occupancy is not an issue it is likely that performance is limited by the instruction stall reasons described below.

**Memory Dependency** - A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns.

**Texture** - The texture sub-system is fully utilized or has too many outstanding requests.

**Instruction Fetch** - The next assembly instruction has not yet been fetched.

**Execution Dependency** - An input required by the instruction is not yet available. Execution dependency stalls can potentially be reduced by increasing instruction-level parallelism.

**Pipeline Busy** - The compute resource(s) required by the instruction is not yet available.

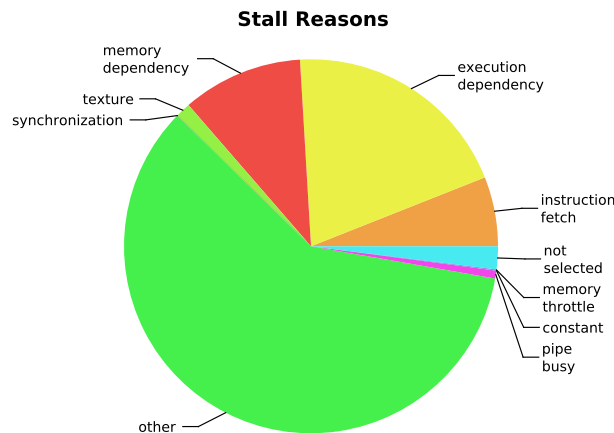
**Memory Throttle** - Large number of pending memory operations prevent further forward progress. These can be reduced by combining several memory transactions into one.

**Synchronization** - The warp is blocked at a `__syncthreads()` call.

**Not Selected** - Warp was ready to issue, but some other warp issued instead. You may be able to sacrifice occupancy without impacting latency hiding and doing so may help improve cache hit rates.

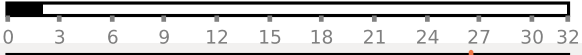


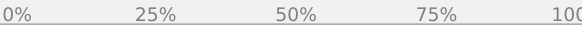


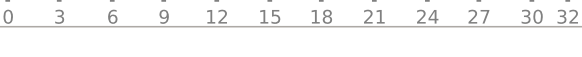


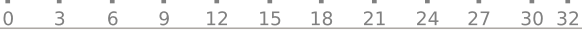


**Constant** - A constant load is blocked due to a miss in the constants cache.

*Optimization: Resolve the primary stall issue; other.*



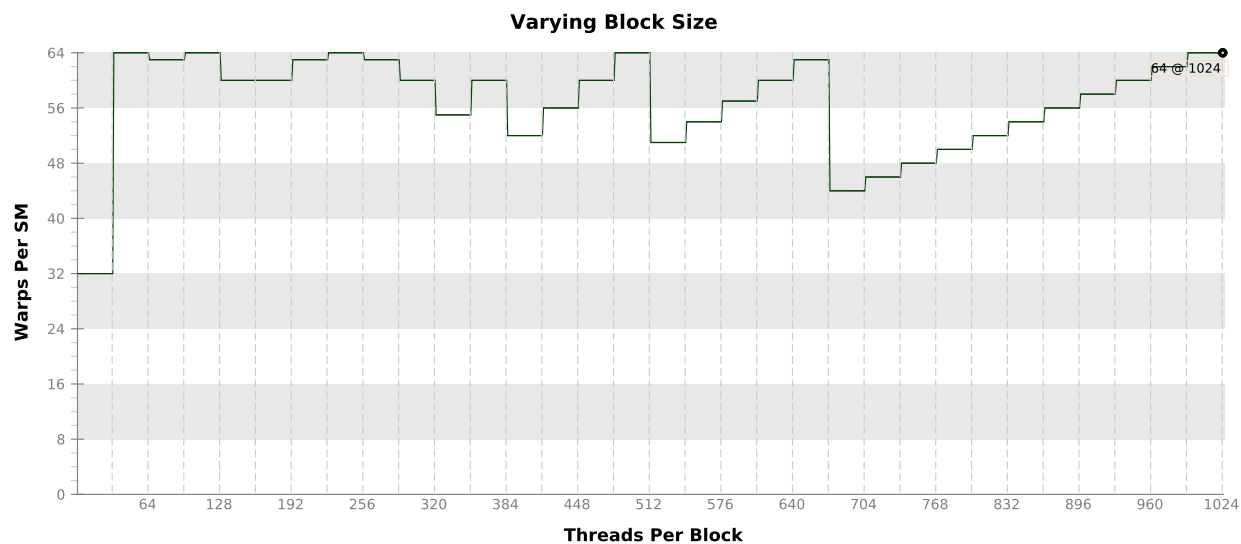
### 2.2. Occupancy Is Not Limiting Kernel Performance

The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU.

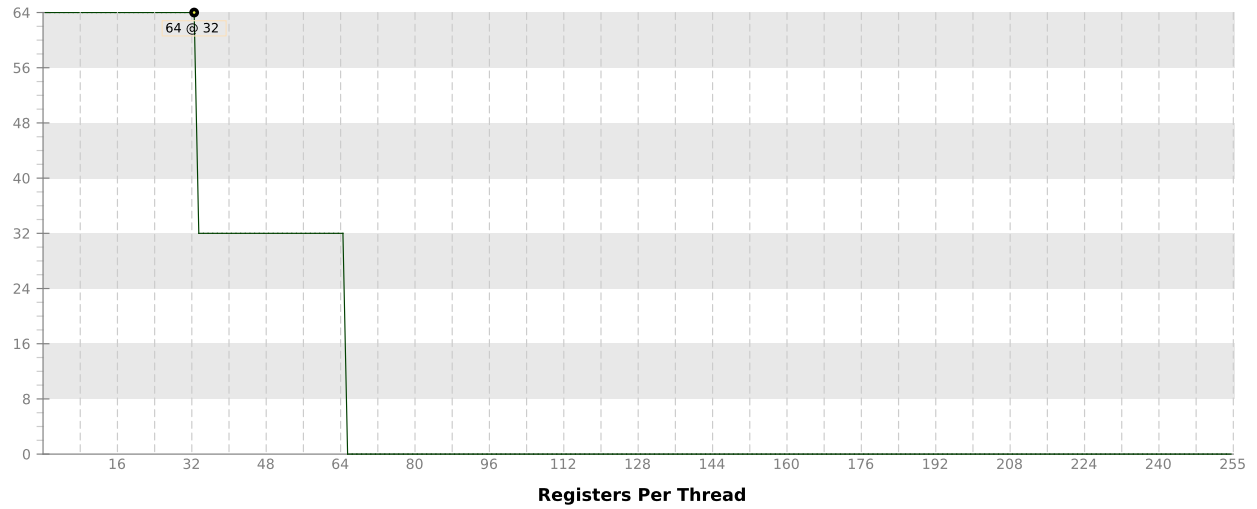
Variable	Achieved	Theoretical	Device Limit	Grid Size: [ 128,1,1 ] (128 blocks) Block Size: [ 1024,1,1 ] (1024 threads)
Occupancy Per SM				
Active Blocks		2	32	
Active Warps	52.63	64	64	
Active Threads		2048	2048	
Occupancy	82.2%	100%	100%	
Warps				
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	32	
Registers				
Registers/Thread		32	255	
Registers/Block		32768	65536	
Block Limit		2	32	
Shared Memory				
Shared Memory/Block		0	65536	
Block Limit			32	

## 2.3. Occupancy Charts

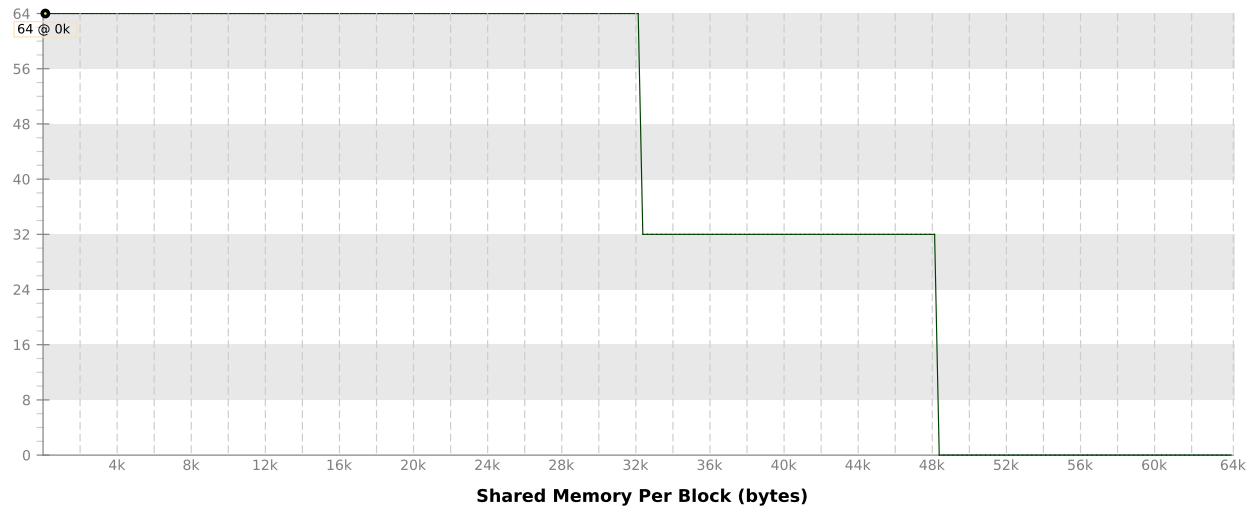
The following charts show how varying different components of the kernel will impact theoretical occupancy.



**Varying Register Count**



**Varying Shared Memory Usage**



### 3. Compute Resources

GPU compute resources limit the performance of a kernel when those resources are insufficient or poorly utilized. Compute resources are used most efficiently when all threads in a warp have the same branching and predication behavior. The results below indicate that a significant fraction of the available compute performance is being wasted because branch and predication behavior is differing for threads within a warp.

#### 3.1. Low Warp Execution Efficiency

Warp execution efficiency is the average percentage of active threads in each executed warp. Increasing warp execution efficiency will increase utilization of the GPU's compute resources. The kernel's warp execution efficiency of 29% is less than 100% due to divergent branches and predicated instructions. If predicated instructions are not taken into account the warp execution efficiency for these kernels is 31.3%.

*Optimization: Reduce the amount of intra-warp divergence and predication in the kernel.*

#### 3.2. Divergent Branches

Compute resource are used most efficiently when all threads in a warp have the same branching behavior. When this does not occur the branch is said to be divergent. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources.

*Optimization: Each entry below points to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence.*

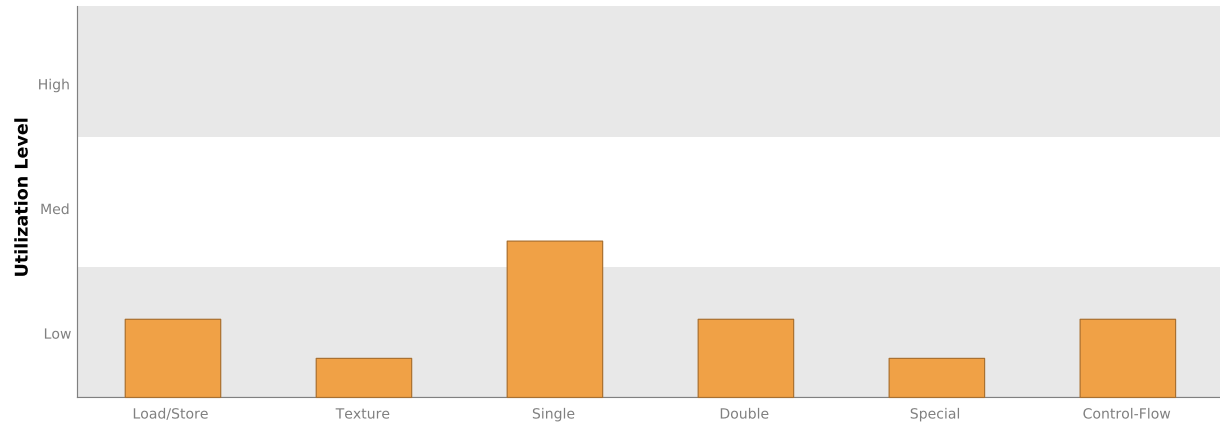
[/home/giuseppe/Documents/myCUDA/ostacoli-2D/poligoni/improved/cuda\\_impr/const\\_memory/inside.cu](/home/giuseppe/Documents/myCUDA/ostacoli-2D/poligoni/improved/cuda_impr/const_memory/inside.cu)

Line 104	Divergence = 80.8% [ 165404 divergent executions out of 204800 total executions ]
Line 104	Divergence = 80.8% [ 165403 divergent executions out of 204800 total executions ]
Line 158	Divergence = 98.9% [ 4049 divergent executions out of 4096 total executions ]

#### 3.3. Function Unit Utilization

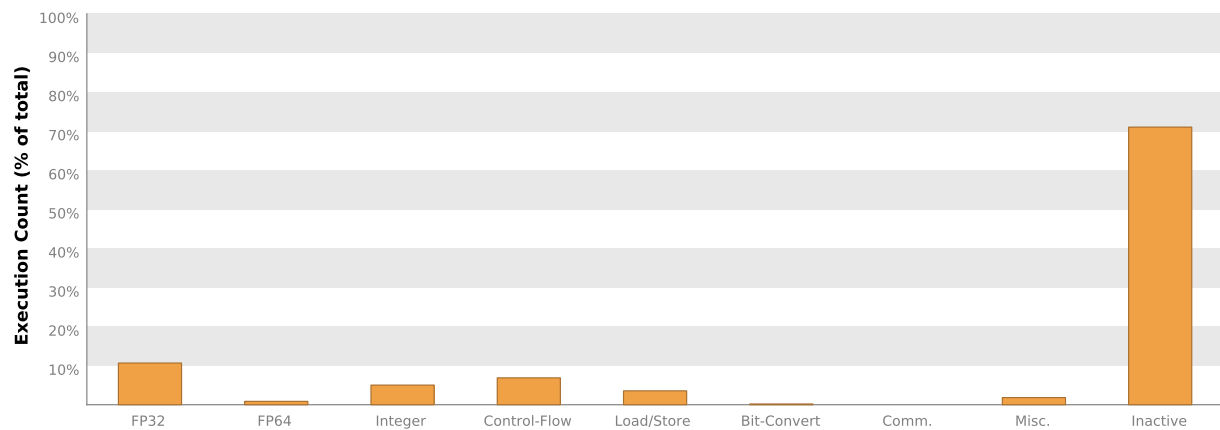
Different types of instructions are executed on different function units within each SM. Performance can be limited if a function unit is over-used by the instructions executed by the kernel. The following results show that the kernel's performance is not limited by overuse of any function unit.

Load/Store - Load and store instructions for shared and constant memory.  
Texture - Load and store instructions for local, global, and texture memory.  
Single - Single-precision integer and floating-point arithmetic instructions.  
Double - Double-precision floating-point arithmetic instructions.  
Special - Special arithmetic instructions such as sin, cos, popc, etc.  
Control-Flow - Direct and indirect branches, jumps, and calls.



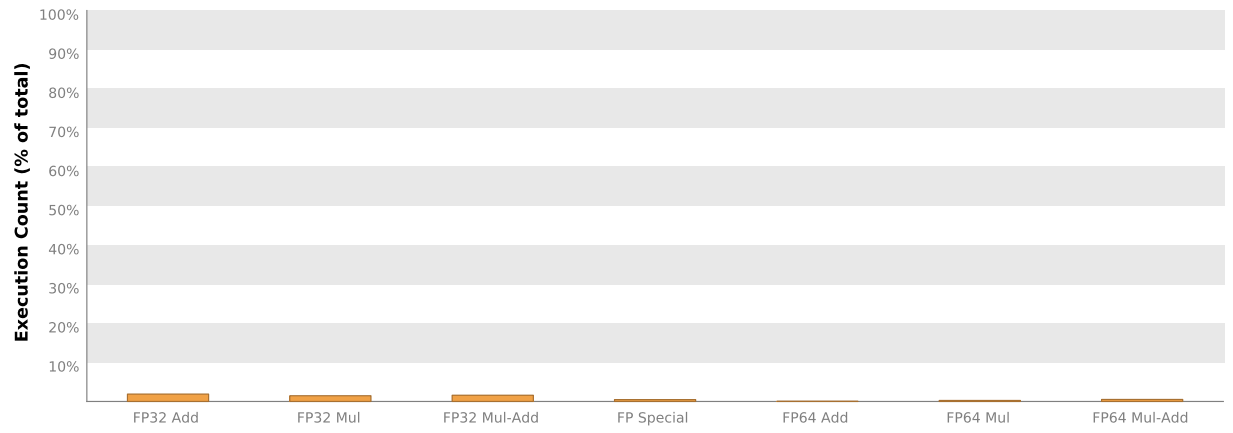
### 3.4. Instruction Execution Counts

The following chart shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing instructions in that class. The "Inactive" result shows the thread executions that did not execute any instruction because the thread was predicated or inactive due to divergence.



### 3.5. Floating-Point Operation Counts

The following chart shows the mix of floating-point operations executed by the kernel. The operations are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing operations in that class. The results do not sum to 100% because non-floating-point operations executed by the kernel are not shown in this chart.







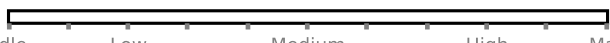


## 4. Memory Bandwidth

Memory bandwidth limits the performance of a kernel when one or more memories in the GPU cannot provide data at the rate requested by the kernel.

### 4.1. Memory Bandwidth And Utilization

The following table shows the memory bandwidth used by this kernel for the various types of memory on the device. The table also shows the utilization of each memory type relative to the maximum throughput supported by the memory.

Transactions	Bandwidth	Utilization	
Shared Memory			
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	
L2 Cache			
Reads	1086867	13.464 GB/s	
Writes	1888366	23.392 GB/s	
Total	2975233	36.856 GB/s	
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	1086803	13.463 GB/s	
Global Stores	1888360	23.392 GB/s	
Texture Reads	317854	3.937 GB/s	
Unified Total	3293017	40.793 GB/s	
Device Memory			
Reads	262346	3.25 GB/s	
Writes	296361	3.671 GB/s	
Total	558707	6.921 GB/s	
System Memory			
[ PCIe configuration: Gen2 x4, 5 Gbit/s ]			
Reads	0	0 B/s	
Writes	5	61.938 kB/s	